Model Counting meets $F_0$ Estimation*

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

Arnab Bhattacharyya ₋
arrib@nus.edu.sg
National University of Singapore

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

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

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 $\varphi$ over a set of variables in a finite domain $D$, the problem of model counting is to estimate the number of solutions of $\varphi$. We are often interested when $\varphi$ 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 $a = a_1, a_2, \ldots, a_m$ wherein each item $a_i \subseteq [N]$. The zeroth frequency moment, denoted as $F_0$, of $a$ is the number of distinct elements appearing in $a$, i.e., $|\cup_i a_i|$ (traditionally, $a_i$ are singletons; we will also be interested in the case when $a_i$ 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
design new algorithms to estimate \( \hat{F}_0 \) 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 \( \hat{F}_0 \) 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 \((\epsilon, \delta)\)-approximation schemes. The complexity of approximate model counting depends on its representation. When the model \( \varphi \) is represented as a CNF formula \( \varphi \), designing an efficient \((\epsilon, \delta)\)-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 \( \varphi \) is a CNF formula while \#DNF to refer to the case when \( \varphi \) is a DNF formula.

For \#CNF, Stockmeyer [55] provided a hashing-based randomized procedure that can compute \((\epsilon, \delta)\)-approximation within time polynomial in \( |\varphi|, \epsilon, \delta \), 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 \((\epsilon, \delta)\)-approximation of the \(k\)th frequency moments \( \hat{F}_k \) is a central problem in the data streaming model [3]. In particular, considerable work has been done in designing algorithms for estimating the \(0^*\) frequency moment \( \hat{F}_0 \), 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 \( \text{poly}(\log N, 1/\epsilon) \) where \( N \) is the size of the universe \(^1\).

The first algorithm for computing \( \hat{F}_0 \) 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 \( \hat{F}_0 \) 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 \((\epsilon, \delta)\)-approximation algorithms with space and time complexity \( \log N \cdot \text{poly}(1/\delta) \). 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 \( \hat{F}_0 \). This line of work resulted in the development of an algorithm with optimal space complexity \( O(\log N + 1/m) \) 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 \( \hat{F}_0 \) in rich scenarios with particular focus to handle the cases \( a_i \subseteq \mathbb{N} \) 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 \( \hat{F}_0 \) 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 \( \hat{F}_0 \) 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 \( \hat{F}_0 \) 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 ceiling \( h^{-1}(0^m) \) 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 ceiling \( h^{-1}(0^m) \) 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 \( \hat{F}_0 \) estimation which we call the bucketing algorithm. We will assume the universe \( \mathbb{N} = \{0, 1\}^n \). The algorithm maintains a bucket of size

\(^1\)We ignore \( O(\log 1/\epsilon) \) factor in this discussion
Thresh and starts by picking a hash function \( h : \{0, 1\}^n \rightarrow \{0, 1\}^n \).
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 \( \epsilon \)-estimate, the Thresh value is chosen as \( O\left(\frac{1}{\epsilon^2}\right) \) 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 \((\epsilon, \delta)\)-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 (sufficient representation of a) subset of the universe. Thus, the stream is \( S_1, S_2, \ldots \) where each \( S_i \subseteq \{0, 1\}^n \) 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 \( \Pr[Z : \Omega] \) to denote the probability of outcome \( Z \) when sampling from a probability space \( \Omega \). For brevity, we omit \( \Omega \) when it is clear from the context.

**F0 Estimation.** A data stream over domain \( |N| \) can be represented as \( a = a_1, a_2, \ldots, a_m \) wherein each item \( a_i \in |N| \). Let \( a_u = \cup_i (a_i) \). \( F_0 \) of the stream \( a \) is \( |a_u| \). We are often interested in a a **probably approximately correct** scheme that returns an \((\epsilon, \delta)\)-estimate \( c \), i.e.,

\[
\Pr\left[ \frac{|a_u|}{1 + \epsilon} \leq c \leq (1 + \epsilon)|a_u| \right] \geq 1 - \delta
\]

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

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

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

\[\text{Please refer to Remark 1 in Section 5 for a discussion on the earlier work on multidimensional ranges [58].}\]
PAC guarantees are also sometimes referred to as \((\epsilon, \delta)\)-guarantees. We use \#CNF (resp. \#DNF) to refer to the model counting problem when \(\phi\) is represented as CNF (resp. DNF).

**k-wise Independent hash functions.** Let \(n, m \in \mathbb{N}\) and \(H(n, m) \triangleq \{h : \{0, 1\}^n \rightarrow \{0, 1\}^m\}\) be a family of hash functions mapping \([0, 1]^n\) to \([0, 1]^m\). We use \(h \sim \mathcal{H}(n, m)\) to denote the probability space obtained by choosing a hash 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 for all \(x_1, x_2, \ldots, x_k \in \{0, 1\}^n\), distinct \(x_1, x_2, \ldots, x_k \in \{0, 1\}^n\), \(h \sim \mathcal{H}(n, m)\),

\[
\Pr[(h(x_1) = a_1) \land (h(x_2) = a_2) \ldots (h(x_k) = a_k)] = \frac{1}{2^m} \tag{1}
\]

We will use \(\mathcal{H}_{k\text{-wise}}(n, m)\) 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 \(\mathcal{H}\text{\textnormal{toeplitz}}(n, m)\), which is known to be 2-wise independent \([10]\). The family is defined as follows: \(\mathcal{H}\text{\textnormal{toeplitz}}(n, m) \triangleq \{h : \{0, 1\}^n \rightarrow \{0, 1\}^m\}\) 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 \(\mathcal{H}\text{\textnormal{haar}}(n, m)\), where \(H(x)\) 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 \(\mathcal{H}\text{\textnormal{toeplitz}}\) and \(\mathcal{H}\text{\textnormal{haar}}\) are 2-wise independent but it is worth noticing that \(\mathcal{H}\text{\textnormal{toeplitz}}\) can be represented with \(O(n)\)-bits while \(\mathcal{H}\text{\textnormal{haar}}\) requires \(O(n^2)\) 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 \(O(1/\epsilon^2)\)-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 \(O(\log(1/\epsilon))\)-independent hash family. For each hash function \(h_j\), we say that \(h_j\) is not lonely if there exists \(a_j \in \{0, 1\}\) such that \(h_j(a_j) = 0\). One can then estimate \(F_0\) of 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 \(S\). 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, \(H[i]\) denote the \(i\)th 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 \(\Theta\) is set to \(96/\epsilon^2\).

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

**Minimum.** For a new item \(x\), if \(H[i][x] = 0^m\) is smaller than the max \(S[i]\), then we replace max \(S[i]\) with \(H[i][x]\).

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

ComputeEst. Finally, for each of the algorithms, we estimate \(F_0\) based on the sketch \(S\) 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 \(S\) 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\((n, \epsilon, \delta)\)**

1. \(\text{Thresh} \leftarrow 96/\epsilon^2\)
2. \(t \leftarrow 35 \log(1/\delta)\)
3. \(H \leftarrow \text{ChooseHashFunctions}(n, \text{Thresh}, t)\)
4. \(S \leftarrow \{}\)
5. while true do
6. if EndStream then exit;
7. \(x \leftarrow \text{input()}\)
8. ProcessUpdate\((S, H, x, \text{Thresh})\)
9. \(Est \leftarrow \text{ComputeEst}(S, \text{Thresh})\)
10. return \(Est\)
Algorithm 2 ChooseHashFunctions($n$, $Thresh$, $t$)
1: switch AlgorithmType do
2: case AlgorithmType==Bucketing then
3: $H \leftarrow$ PickHashFunctions($\mathcal{H}_{\text{toeplitz}}(n, n), t)$
4: case AlgorithmType==Minimum then
5: $H \leftarrow$ PickHashFunctions($\mathcal{H}_{\text{toeplitz}}(n, 3n), t)$
6: case AlgorithmType==Estimation then
7: $s \leftarrow 10 \log(1/\epsilon)$
8: $H \leftarrow$ PickHashFunctions($\mathcal{H}_{\text{wise}}(n, n), t \times Thresh$)
return $H$

Algorithm 3 ProcessUpdate($S$, $H$, $x$, $Thresh$)
1: for $i \in [1, |H|]$ do
2: switch AlgorithmType do
3: case Bucketing then
4: $m_i = |S[i][0]|$
5: if $|H[i][m_i](x)| = 0$ then
6: $S[i][0] \leftarrow S[i][0] \cup \{x\}$
7: if size($S[i][0]$) > Thresh then
8: $S[i][1] \leftarrow S[i][1] + 1$
9: for $y \in S$ do
10: if $|H[i][m_i+1](y)| = 0$ then
11: Remove($S[i][0], y$)
12: case Minimum then
13: if size($S[i]$) < Thresh then
14: $S[i].$Append($H[i](x)$)
15: else
16: $i \leftarrow \arg \max S[i]$
17: if $|S[i][j]| > |H[i][j]|$ then
18: $S[i][j] \leftarrow |H[i][j]|$
19: case Estimation then
20: for $j \in [1, Thresh]$ do
21: $S[i][j] \leftarrow \max(S[i][j], \text{TrailZero}(H[i][j](x)))$
22: return $S$

Algorithm 4 ComputeEst($S$, $Thresh$)
1: switch AlgorithmType do
2: case Bucketing then
3: return Median($\{\text{size}(S[i][0]) \times 2S[i][1] \}_{i}$)
4: case Minimum then
5: return Median($\{\text{Threshold}^{|m_i|} \times \text{Max}(S[i][1]) \}_{i}$)
6: case Estimation($r$) then
7: return Median($\{\ln(1 - \frac{1}{r \ln n - \text{Threshold}^{|m_i|}}) \}_{i}$)

3.1 A Recipe For Transformation

Observe that for each of the algorithms, the final computation of $F_0$ estimation depends on the sketch $S$. Therefore, as long as for two streams $a$ and $\bar{a}$, if their corresponding sketches, say $S$ and $\bar{S}$ 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 $\mathcal{P}(S, H, a_u)$ between the sketch $S$, set of hash functions $H$, and set $a_u$ at the end of stream. Recall that $a_u$ is the set of all distinct elements of the stream $a$.
2. The formula $\phi$ is viewed as symbolic representation of the unique set $a_u$ represented by the stream $a$ such that $\text{Sol}(\phi) = a_u$.
3. Given a formula $\phi$ and set of hash functions $H$, design an algorithm to construct sketch $S$ such that $\mathcal{P}(S, H, \text{Sol}(\phi))$ holds. And now, we can estimate $|\text{Sol}(\phi)|$ from $S$.

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 PFRAS’s when $\phi$ 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 $S$ that we will describe. Here we use $\mathcal{H}_{\text{toeplitz}}$ as our choice of pairwise independent hash functions. The sketch $S$ is an array where, each $S[i]$ of the form $\langle c_i, m_i \rangle$. We say that the relation $\mathcal{P}_1(S, H, a_u)$ holds if

1. $|a_u \cap \{ x \mid H[i][m_i-1](x) = 0 \}| \geq \frac{\epsilon}{\delta}$
2. $c_i = |a_u \cap \{ x \mid H[i][m_i](x) = 0 \}| \leq \frac{\epsilon}{2\delta}$

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

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

$$\Pr \left[ \frac{|a_u|}{(1 + \epsilon)} \leq c \leq (1 + \epsilon)|a_u| \right] \geq 1 - \delta.$$  

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 $\phi$ over $n$ variables, a hash function $h \in \mathcal{H}_{\text{toeplitz}}(n, m)$, and a number $p$ as inputs, returns $\min(p, |\text{Sol}(\phi \land h(x) = 0^m)|)$. If $\phi$ is a CNF formula, then BoundedSAT makes $O(p)$ calls to a NP oracle. If $\phi$ is a DNF formula with $k$ terms, then BoundedSAT takes $O(n^3 \cdot k \cdot p)$ time.


\begin{algorithm}
\caption{ApproxMC($\varphi, \varepsilon, \delta$)}
\begin{algorithmic}[1]
\STATE $t \leftarrow 35 \log(\frac{1}{\delta})$
\STATE $H \leftarrow \text{PickHashFunctions}(\mathcal{H}_{\text{Toeplitz}}(n, n), t)$
\STATE \text{S} \leftarrow \{\}$
\STATE \text{Thresh} \leftarrow \frac{96}{\delta^2}$
\FOR{$i \in [1, t]$}
\STATE $m_i \leftarrow 0$
\STATE $c_i \leftarrow \text{BoundedSAT}(\varphi, H[i]|_{m_i}, \text{Thresh})$
\STATE while $c_i \geq \text{Thresh}$ do
\STATE $m_i \leftarrow m_i + 1$
\STATE $c_i \leftarrow \text{BoundedSAT}(\varphi, H[i]|_{m_i}(x), \text{Thresh})$
\STATE $S[i] \leftarrow (c_i, m_i)$
\STATE $\text{Est} \leftarrow \text{Median}((S[i](0) \times 2^S[i](1)))$
\ENDFOR
\RETURN $\text{Est}$
\end{algorithmic}
\end{algorithm}

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 $(\varphi \land H[i]|_{m_i}(x) = 0^{m_i})$ is less than Thresh. Interestingly, an approximate model counting algorithm, called ApproxMC, based on bucketing strategy was discovered independently by Chakraborty et. al. \cite{12} 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 $S$ 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 $P_2(S, H, \text{Sol}([\varphi]))$ 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 $\mathcal{H}_{\text{Toeplitz}}$ and $\mathcal{H}_{\text{Kor}}$ lead to same time complexity. Furthermore, Chakraborty et. al. \cite{12} observed no difference in empirical runtime behavior due to $\mathcal{H}_{\text{Toeplitz}}$ and $\mathcal{H}_{\text{Kor}}$

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

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

Further Optimizations. We now discuss how the setting of model counting allows for further optimizations. Observe that for all $i$, $\text{Sol}([\varphi \land (H[i]|_{m_i-1}(x) = 0^{m_i-1})] \subseteq \text{Sol}([\varphi \land (H[i]|_{m_i}(x) = 0^{m_i})]$.

Note that we are interested in finding the value of $m_i$ such that $|\text{Sol}([\varphi \land (H[i]|_{m_i-1}(x) = 0^{m_i-1})]| \geq \frac{96}{\delta^2}$ and $|\text{Sol}([\varphi \land (H[i]|_{m_i}(x) = 0^{m_i})]| < \frac{96}{\delta^2}$; 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 follow-up work \cite{14}, which proposed the ApproxMC2, thereby reducing the number of calls to NP oracle from $O(n \cdot \frac{1}{\varepsilon} \log(1/\delta))$ to $O(\log n \cdot \frac{1}{\varepsilon} \log(1/\delta))$.

Furthermore, the reduction in NP oracle calls led to significant runtime improvement in practice. It is worth commenting that the usage of ApproxMC2 as PFRAS for DNF is shown to achieve runtime efficiency over the alternatives based on Monte Carlo methods \cite{44–46}.

### 3.3 Minimum-based Algorithm

For a given multiset $a$ (eg: a data stream or solutions to a model), we now specify the property $P_2(S, H, a_u)$. The sketch $S$ is an array of sets indexed by members of $H$ that holds lexicographically $p$ minimum elements of $H[i](a_u)$ where $p$ is $\min(\frac{96}{\delta^2}, |a_u|)$. $P_2$ is the property that specifies this relationship. More formally, the relationship $P_2$ holds, if the following conditions are met.

1. $\forall i, |S[i]| = \min(\frac{96}{\delta^2}, |a_u|)$
2. $\forall i, \forall y \notin S[i], \forall y' \in S[i]$ it holds that $H[i](y) \geq H[i](y')$

The following lemma due to Bar-Yossef et al. \cite{7} establishes the relationship between the property $P_2$ and the number of distinct elements of a multiset. Let $\text{max}(S_i)$ denote the largest element of the set $S_i$.

**Lemma 2.** \cite{7} Let $a \subseteq \{0, 1\}^n$ be a multiset and $H \subseteq \mathcal{H}_{\text{Toeplitz}}(n, m)$ where $m = 3n$ and each $H[i]$ are independently drawn and $|H| = O(\log(1/\delta))$ and let $S$ be such that the $P_2(S, H, a_u)$ holds. Let $c = \text{Median}(\frac{n \cdot \max(S[i])}{m^2})$.

Then

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

Therefore, we can transform the Minimum algorithm for $\mathcal{F}_0$ estimation to that of model counting given access to a subroutine that can compute $S$ such that $P_2(S, H, \text{Sol}([\varphi]))$ 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 $\varphi$ over $n$ variables, $h \in \mathcal{H}_{\text{Toeplitz}}(n, m)$, and $p$ as input, returns a set, $\mathcal{B} \subseteq h(\text{Sol}([\varphi]))$ so that if $|h(\text{Sol}([\varphi]))| \leq p$, then $\mathcal{B} = h(\text{Sol}([\varphi]))$, otherwise $\mathcal{B}$ is the $p$ lexicographically minimum elements of $h(\text{Sol}([\varphi]))$. Moreover, if $\varphi$ is a CNF formula, then FindMin makes $O(p \cdot m)$ calls to an NP oracle, and if $\varphi$ is a DNF formula with $k$ terms, then FindMin takes $O(m^2 \cdot n \cdot k \cdot p)$ 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 $\varphi$ is in DNF, we have ApproxModelCountMin as a PFRAS for DNF formulas.

**Theorem 3.** Given $\varphi, \varepsilon, \delta$, ApproxModelCountMin returns $c$ such that

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

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

**Implementing the Min-based Algorithm.** We now give a proof of Proposition 2 by giving an implementation of FindMin subroutine.
Algorithm 6: ApproxModelCountMin($\varphi, r, \delta$)

1. $t \leftarrow 35 \log(1/\delta)$
2. $H \leftarrow \text{PickHashFunctions}(\mathcal{H}_{\text{triplext}}(n, 3n), t)$
3. $S \leftarrow \{\}$
4. $\text{Threshold} \leftarrow \frac{2s}{t}$
5. for $i \in [1, t]$ do
6. $\text{FindMin}(\varphi, H[i], \text{Threshold})$
7. $\text{Est} \leftarrow \text{Median}(\text{Threshold}2^\mathbb{Z})$
8. return $\text{Est}$

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

Let $\varphi = T_1 \lor T_2 \lor \cdots \lor T_k$ be a DNF formula over $n$ variables where $T_i$ is a term. Let $h : \{0,1\}^{n} \rightarrow \{0,1\}^{m}$ be a linear hash function in $\mathcal{H}_{\text{arr}}(n,m)$ defined by a $m \times n$ binary matrix $A$. Let $C$ be the set of hashed values of the satisfying assignments for $\varphi$: $C = \{h(x) \mid x \models \varphi\} \subseteq \{0,1\}^{m}$. 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 $O(m^3np)$ 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 $O(m^3nk^p)$ 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 $r_l$ 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 $y_1 \ldots y_{r_l-1}1$ 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 $O(m^3n)$ time algorithm. Invoking the above procedure $p$ times results in an algorithm to compute $C_p$ in $O(m^3np)$ time.

If $\varphi$ is a CNF formula, we can employ the same prefix searching strategy. Consider the following NP oracle: $O = \{\langle \varphi, h, y', y'' \rangle \mid \exists x, \exists y', y'' \text{ such that } x \models \varphi, y' y'' > y, h(x) = y' y''\}$. 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 $a$ and chosen hash functions $H$, the sketch $S$ corresponding to the estimation-based algorithm satisfies the following relation $P_3(S, H, a_u) = \max_{x \in a_u} \text{TrailZero}(H[i,j](x))$ (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 $P_3$ and $F_0$.

Lemma 3. [7] Let $a \subseteq \{0,1\}^{n}$ be a multiset. For $i \in [T]$ and $j \in [M]$, suppose $H[i,j]$ is drawn independently from $\mathcal{H}_{\text{wise}}(n, w)$ where $s = O(\log(1/\epsilon))$, $T = O(\log(1/\delta))$, and $M = O(1/\epsilon^2)$. Let $H$ denote the collection of these hash functions. Suppose $S$ satisfies $P_3(S, H, a_u)$. For any integer $r$, define:

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

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

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

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 $\varphi$ 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 $\varphi$ over $n$ variables and hash function $h \in \mathcal{H}_{\text{wise}}(n, n)$, returns $t$ such that

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

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

Proof. Consider an NP oracle $O = \{\langle \varphi, h, t \rangle \mid \exists x, \exists y, x \models \varphi, h(x) = y^t\}$. Note that $h$ can be implemented as a degree-$s$
polynomial \( h : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n} \), 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 \( r \) for which \((\varphi, h, t)\) belongs to \( O \). □

We note that unlike Propositions 1 and 2, we do not know whether \FindMaxRange\ can be implemented efficiently when \varphi\ is a DNF formula. For a degree-\( s \) polynomial \( h : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n} \), 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 \varphi\ and a suitable value of \( r \) and returns \(|\Sol(\varphi)|\). 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 \( S \) 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 \( \varphi \), parameters \( \epsilon \) and \( \delta \), and \( r \) such that \( 2F_{\varphi} \leq 2^r \leq 50F_{\varphi} \), the algorithm \ApproxModelCountEst\ returns \( c \) satisfying

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

\ApproxModelCountEst\ makes \( O\left(\frac{1}{\epsilon^2} \log n \log(\frac{1}{\delta})\right) \) calls to an NP oracle.

In order to obtain \( r \), we run in parallel another counting algorithm based on the simple \( F_{\varphi} \)-estimation algorithm [3, 29] which we call FlajoletMartin. Given a stream \( a \), the FlajoletMartin algorithm chooses a random pairwise-independent hash function \( h \in \mathcal{H}_{\text{wise}}(n,n) \), computes the largest \( r \) so that for some \( x \in a \), 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 \( F_{\varphi} \) with probability 3/5. Using our recipe, we can convert FlajoletMartin into an algorithm that approximates the number of solutions to a CNF formula \( \varphi \) 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 \( \varphi \) 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 \( \varphi \) is represented as DNF. The algorithms based on Estimation have been shown to achieve optimal space efficiency in the context of \( F_{\varphi} \) 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 \( \varphi \) is partitioned into \( k \) DNF subformulas \( \varphi_1, \ldots, \varphi_k \), where each \( \varphi_i \) is a subset of the terms of the original \( \varphi \), with the \( j \)th site receiving only \( \varphi_j \). The goal is for the coordinator to obtain an \((\epsilon, \delta)\)-approximation of the number of solutions to \( \varphi \), 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 **distributing functional monitoring** framework formalized by Cormode et al. [20]. Here, the input is a stream which is partitioned arbitrarily into sub-streams \( a_1, \ldots, a_k \) 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...
variables, a hash function 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 \( a_i \) corresponds to the set of satisfying assignments to each subformula \( \varphi_i \), 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 \( O(1/\epsilon^2) \). Correctness follows from Bar-Yossef et al. [7] and the earlier discussion.

We consider an adaptation of BoundedSAT that takes in \( \varphi \) over \( n \) variables, a hash function \( h \in \mathcal{H}_{Treepl1}(n, m) \), and a threshold \( t \) as inputs, returns a set \( U \) of solutions such that \( |U| = \min(t, |\text{Sol}(\varphi \land h(x) = 0^m)|) \).

**Bucketing.** Setting \( m = O(\log(k/\delta^2)) \), the coordinator chooses \( H[1], \ldots, H[T] \) from \( \mathcal{H}_{Treepl1}(n, m) \) and \( G \) from \( \mathcal{H}_{Tree}(n, m) \). Then it sends them to the \( k \) sites. Let \( m_j \) be the smallest \( m \) such that the size of the set BoundedSAT\((\varphi_j, H[1]_m, \text{thresh}) \) is smaller than Thresh. The \( j \)th site sends the coordinator the following tuples: \( (G(x), \text{TrailZero}(H[1]|x)) \) for each \( i \in [T] \) and for each \( x \) in BoundedSAT\((\varphi_j, H[1]_m, \text{thresh}) \). Note that each site only sends tuples for at most \( O(1/\delta^2) \) 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 \( O(k(n + 1/\delta^2) \cdot \log(1/\delta)) \), and the time complexity for each site is polynomial in \( n, \epsilon^{-1} \), and \( \log(\delta^{-1}) \).

**Minimum.** The coordinator chooses hash functions \( H[1], \ldots, H[T] \) from \( \mathcal{H}_{Treepl1}(n, 3n) \) 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 \( \varphi_j \). 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 \( O(kn/\epsilon^2 \cdot \log(1/\delta)) \) to account for the \( k \) sets sending the outputs of their FindMin invocations. The time complexity for each site is polynomial in \( n, \epsilon^{-1} \), and \( \log(\delta^{-1}) \).

**Estimation.** For each \( i \in [T] \), the coordinator chooses Thresh hash functions \( H[i, 1], \ldots, H[i, \text{thresh}] \), drawn pairwise independently from \( \mathcal{H}_{Tree}(n, n) \) (for \( s = O(\log(1/\epsilon)) \)) 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 \( S[i, j, \ell] \subseteq [n] \) for each \( i \in [T], j \in \text{Thresh} \) 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 \( O(k(n + 1/\epsilon^2) \log(1/\delta)) \). 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 \( \epsilon \). Woodru

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

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 \( \bigcup_i S_i \cdot \text{ number of distinct elements in the union of all the sets in the stream.} \) 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 \( \varphi \) be a DNF formula over \( n \) variables. Then the \( DNF \text{ Set corresponding to } \varphi \) is the set of satisfying assignments of \( \varphi \). The size of this representation is the number of terms in the formula \( \varphi \).

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 \( \varphi_i \). This quantity is same as the number of satisfying assignments of the formula \( \bigvee_i \varphi_i \). We show that the algorithms described in the previous section carry over to obtain \((\epsilon, \delta)\) estimation algorithms for this problem with space and per-item time \( poly(1/\epsilon, n, k, \log(1/\delta)) \) 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 \( \varphi(x) \) 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 \((\epsilon, \delta)\) approximation of \( F_0 \) over DNF sets. This algorithm takes space \( O(\frac{n^4}{\epsilon^2} \cdot \log(\frac{1}{\delta})) \) and processing time \( O(n^4 \cdot k \cdot \frac{1}{\epsilon^2} \cdot \log(\frac{1}{\delta})) \) per item where \( k \) is the number (size 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 \( h \in \mathcal{H}_{Treepl1}(n, 3n) \) maintains the set \( B \) consisting of \( t \) lexicographically minimum elements of the set \( \{h(Sol(\varphi_1), \ldots, Sol(\varphi_{t-1}))\} \) after processing \( i - 1 \) items. When \( \varphi_i \) arrives, it computes the set \( B' \) consisting of the \( t \) lexicographically minimum values of the set \( \{h(Sol(\varphi_i))\} \) and subsequently updates \( B \) by computing the \( t \) lexicographically smallest elements from \( B \cup B' \). By Proposition 2,
computation of $B'$ can be done in time $O(n^d \cdot k \cdot t)$ where $k$ is the number of update in $\varphi_i$. Updating $B$ can be done in $O(t \cdot n)$ time.

Thus update time for the item $\varphi_i$ is $O(n^d \cdot k \cdot t)$. For obtaining an $(\varepsilon, \delta)$ approximations we set $t = O\left(\frac{1}{\varepsilon^2}\right)$ and repeat the procedure $O(\log \frac{1}{\delta})$ times and take the median value. Thus the update time for item $\varphi$ is $O(n^d \cdot k \cdot \frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta})$. For analyzing sapce, each hash function uses $O(n)$ bits and to store $O\left(\frac{1}{\varepsilon^2}\right)$ minimums, we require $O\left(\frac{n}{\varepsilon^2}\right)$ space resulting in overall space usage of $O\left(\frac{n}{\varepsilon^2} \cdot \log \frac{1}{\delta}\right)$. 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 streamings 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 $(a_1, b_1, \ldots, a_d, b_d)$. 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 efficiently. 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 $\varphi_R$ 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 $\varphi_R$ 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 $U$. We will first describe the formula to represent the multi-dimensional range as a conjunction of $d$ DNF formulae $\varphi_1, \ldots, \varphi_d$ each with at most $2n$ terms, where $\varphi_i$ represents $[a_i, b_i]$, the range in the $i^{th}$ dimension. Converting this into a DNF formula will result in the formula $\varphi_R$ with $(2n)^d$ terms.

For any $\ell$ bit number $c$, $1 \leq c \leq 2^n$, it is straightforward to write a DNF formula $\varphi_{\leq c}$, of size at most $\ell$, 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 $\ell$ for the range $[c, 2^\ell - 1]$. Now we construct a formula to represent the range $[a, b]$ over $U$ as follows.

Let $a_1a_2\cdots a_n$ and $b_1b_2\cdots b_n$ be the binary representations of $a$ and $b$ respectively. Let $\ell$ be the largest integer such that $a_1a_2\cdots a_\ell = b_1b_2\cdots b_\ell$. Hence $a_{\ell+1} = 0$ and $b_{\ell+1} = 1$. Let $a'$ and $b'$ denote the integers represented by $a_{\ell+2}\cdots a_n$ and $b_{\ell+2}\cdots b_n$. Also, let $\psi$ denote the formula (a single term) that represents the string $a_1\cdots a_{\ell}$. Then the formula representing $[a, b]$ is $\psi$ $\land$ $(a' \lor b' \lor \forall x_{\ell+1} \varphi_{\geq c'})$. This can be written as a DNF formula by distributing $\psi$ and the number of terms in the resulting formula is at most $2n$, and has $n$ variables. Note that each $\varphi_i$ can be constructed using $O(n)$ space. To obtain the final DNF representing the range $R$, we need to convert $\varphi_1 \land \cdots \land \varphi_d$ into a DNF formula. It is easy to see that for any $i$, then $i^{th}$ 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 algorithm for estimating $F_0$ over multidimensional ranges in a range-efficient manner.

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

**Remark 1.** Thrishapura and Woodruff [38] 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 $\text{poly}(n, d, 1/\varepsilon, \log 1/\delta)$. However, they have retracted their claim [60]. Their method only yields $\text{poly}(n^d, 1/\varepsilon, \log 1/\delta)$ 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 $S = (S_1, S_2, \ldots, S_M)$ wherein each $S_i \subseteq \{0, 1\}^n$ belongs to Delphic family, computes an $(\varepsilon, \delta)$-approximation of $\sum_{i=1}^M |S_i|$ with worst case space complexity $O(n \cdot \log (M/\delta) \cdot \varepsilon^{-2})$ and per-item time is $O(n \cdot \log (M/\delta) \cdot \varepsilon^{-2})$. 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 $\varphi$ for this range has size (number of terms) $\geq n^d$. For any $1 \leq j \leq d$, we use the set of variables
$X^j = \{x^j_1, x^j_2, \ldots, x^j_{k_j}\}$ for representing the $j^{th}$ coordinate of $R$. Then $R$ can be represented as the formula $\varphi_R = \lor_{(i_1, i_2, \ldots, i_d) \in X^1 \times X^2 \times \cdots \times X^d} x_{i_1}^1 x_{i_2}^2 \cdots x_{i_d}^d$, where $1 \leq i_j \leq n$. This formula has $n^d$ terms. Let $\varphi$ be any other DNF formula representing $R$. The main observation is that any term $T$ of $\varphi$ is completely contained (in terms of the set of solutions) in one of the terms of $\varphi_R$. This implies that $\varphi$ should have $n^d$ terms. Now we argue that $T$ is contained in one of the terms of $\varphi_R$. $T$ should have at least one variable as a positive literal from each of $X_i$. Now let $x^j_i$ be a variable from $X^j_i$ that is in $T$. Then clearly $T$ is in the term $x^1_{i_1} x^2_{i_2} \cdots x^d_{i_d}$ 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 $P \neq NP$. 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 $\phi_1$ arrives, we need to compute the $t$ lexicographically smallest elements of $h(Sol(\phi_1))$ where $h \in \mathcal{H}_{\text{Toepplitz}}(n, 3n)$. By Proposition 2, this can be done in polynomial-time by making $O(nd)$ calls to an NP oracle since $\phi_1$ is a CNF formula over $nd$ variables. Thus if $P$ equals NP, then the time taken per range is polynomial in $n$, $d$, and $1/\epsilon^2$. Thus a super-polynomial time lower bound on time per item implies that $P$ differs from NP.

**From Weighted $d$-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 $\varphi$ 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 $\sigma$ can be defined as follows:

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

Furthermore, we define the weight of a formula $\varphi$ as

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

Given $\varphi$ and $\rho$, the problem of weighted counting is to compute $W(\varphi)$. We consider the case where for each $x_j, \rho(x_j)$ is represented using $m_j$ bits in binary representation, i.e., $\rho(x_j) = \frac{2^{m_j}}{2^m}$. 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 $F_0$ on $n$-dimensional ranges. The reduction is as follows: we transform every term of $\varphi$ into a product of multi-dimension ranges where every variable $x_j$ is replaced with $[k_j + 1, 2m_j]$ and every $\land$ is replaced with $\times$. For example, a term $(x_1 \land x_2 \land \neg x_3)$ is replaced with $[1, k_1] \times [k_2 + 1, 2m_2] \times [k_3 + 1, 2m_3]$. Given $F_0$ of the resulting stream, we can compute the weight of $\varphi$ simply as $W(\varphi) = \frac{2^{m_j}}{2^m}$. 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 $[a, b, \epsilon]$ 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 $[a, b, 2^d]$ can be expressed as a DNF formula as follows: Let $\psi$ be the DNF formula representing the range $[a, b]$ and let $a_1, \ldots, a_d$ be the least significant bits of $a$. Let $\psi'$ be the term that represents the bit sequence $a_1 \cdots a_d$. Now the formula to represent the arithmetic progression $[a, b, 2^d]$ is $\psi \land \psi'$ 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 $(\epsilon, \delta)$ approximation of $F_0$ over $d$-dimensional arithmetic progressions, whose common differences are powers of two, that takes space $O(nd^2 \cdot \log 1/\delta)$ and processing time $O(nd^3 \cdot n^d \cdot 4^d(1/\delta))$ per item.

**Affine Spaces.**

Another example of a 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 $(A_1, B_1), (A_2, B_2), \ldots$, where each $(A_i, B_i)$ is succinctly represented as a set $\{x \in \{0, 1\}^n \mid Ax = B_i\}$.

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

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

**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 $(\varepsilon, \delta)$ approximation of $F_0$ over affine spaces. This algorithm takes space $O(\frac{n}{\varepsilon^2} \cdot \log(1/\delta))$ and processing time of $O(n^k \frac{1}{\varepsilon^2} \log(1/\delta))$ 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 estimating 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 $H_{\text{topolitz}}$ and $H_{\text{topolitz}}$ 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-NRFFA11-2019-0002] and an Amazon Research Award. Meel was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme[NRF-NRFFA11-2019-0004 ] 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 Ilkcan 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] Chiriz Arackaparambil, Joshua Brody, and Amit Chakrabarti. 2009. Functional monitoring without monotonicity. In International Colloquium on Automata, Languages, and Programming. Springer, 95–106.
[5] Megasthenis Asterias and Alexandros G Dimakis. 2016. LDLR 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 LucaTrevisan. [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 Ralf 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, Drrt Fried, Kuldeep S Mee, 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 Condilise, Feifei Li, George Kolios, and John W. Byers. 2004. Approximate Aggregation Techniques for Sensor Databases. In Proc. of ICDE, 2. Meral Oszolyogia and Stanley B. Zdonik (Eds.). IEEE Computer Society, 449–460.
[18] Graham Cormode, Minos Garofalakis, Shanmugavelyathum Muthukrishnan, and Rajev 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, Shanmugavelyathum Muthukrishnan, and Ke Yi. 2011. Algorithms for distributed functional monitoring. ACM Transactions on Algorithms (TALG) 7, 2 (2011), 1–20.
[21] Graham Cormode, Shanmugavelyathum 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 ICMLE. 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 IJCAI. 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 Proc. of TACAS. Springer, 189–198.
[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] Philipp B. Gibbons and Srikanth Tirthapura. 2001. Estimating simple functions on the union of data streams. In Proc. of SPAA, Arnold L. Rosenberg (Ed.). ACM, 281–291.
