Model Counting Meets Distinct Elements in a Data Stream

[Extended Abstract]

A. Pavan ⓡ
Iowa State University

N. V. Vinodchandran ⓡ
University of Nebraska, Lincoln

Arnab Bhattacharyya ⓡ
National University of Singapore

Kuldeep S. Meel
National University of Singapore

ABSTRACT

Constraint satisfaction problems (CSPs) 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 CSPs 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.

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 in topics including probabilistic reasoning, networks, databases, and verification. *Model counting* and computation of *zeroth frequency moment* (F_0) are fundamental problems for CSPs 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 $\mathbf{a} = \langle a_1, a_2, \cdots, a_m \rangle$ where each item a_i is a subset of $[N]$. The *zeroth frequency moment*, denoted as F_0, of $\mathbf{a}$ is the number of distinct domain 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 design new algorithms to estimate F_0 for streaming sets that are succinctly represented by constraints. It is worth noting that the two communities focus on seemingly different efficiency objectives: in streaming algorithms, space complexity is of major concern while in the context of model counting, time (especially NP query complexity in the context of CNF formulas) is of primary concern. Therefore, it is striking to observe that our trans-

This is a minor revision of the paper entitled "Model Counting Meets F_0 Estimation", Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Virtual Event, China, June 20-25, 2021. ACM 2021, ISBN 978-1-4503-8381-3. https://dl.acm.org/doi/10.1145/3452021.3458311

formation recipe leads to the design of *efficient* algorithms for F_0 estimation as well as model counting wherein *efficient* is measured by the concern of the corresponding community. We further investigate this observation and demonstrate that the space complexity of streaming algorithms provides an upper bound on the query complexity of model counting algorithms.

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 [32]. This motivated researchers to investigate approximate model counting and in particular to design (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 [30]. In contrast, when it is represented as a DNF formula, model counting admits an FPRAS (fully polynomial-time approximation scheme) [21, 22]. We will use #CNF to refer to the case when φ is a CNF formula and #DNF to refer to the case when φ is a DNF formula.

For #CNF, Stockmeyer [30] provided a hashing-based randomized procedure that can compute an (ε, δ)-approximation with running time $\text{poly}(|\varphi|, 1/\varepsilon, 1/\delta)$, 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. The practical implementations substitute the NP oracle with a SAT solver. 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 [29].

Karp and Luby [21] proposed the first FPRAS scheme for #DNF, which was improved in subsequent works [22, 9]. Chakraborty, Meel, and Vardi [5] demonstrated that the hashing-based framework can be extended to #DNF, thereby providing a unified framework for both #CNF and #DNF. Meel, Shrotri, and Vardi [24, 25] subsequently improved the complexity of the hashing-based approach for #DNF and observed that hashing-based techniques achieve better scalability than Monte Carlo techniques.

Zeroth Frequency Moment Estimation

Estimating (ε, δ)-approximation of the k^{th} frequency moments (F_k) of a stream is a central problem in the data streaming model [1]. 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. For streaming algorithms, the primary resource concerns are space complexity and processing time per element. In general, for a streaming algorithm to be considered efficient, these should be $\text{poly}(\log N, 1/\epsilon)$ where N is the size of the universe (we assume δ to be a small constant and ignore $O(\log(\frac{1}{\delta}))$ factors in this discussion).

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 [15]. 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 [1]. Subsequent investigations into hashing-based schemes by Gibbons and Tirthapura [16] and Bar-Yossef, Kumar, and Sivakumar [3] provided (ε, δ)-approximation algorithms with space and time complexity $\log N \cdot \text{poly}(\frac{1}{\varepsilon})$. Later, Bar-Yossef et al. proposed *three algorithms* with improved space and time complexity [2]. 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 to estimate the F_0 of a stream [20].

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 a particular focus to handle the cases $a_i \subseteq \{1, 2, \cdots, N\}$ such as a list or a multidimensional range [3, 26, 31].

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 [30] that is further refined by Chakraborty et al. [5] and Gibbons and Tirthapura's F_0 estimation algorithm [16], 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 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 $\mathsf{Thresh}+1$ number of solutions or we have exhaustively enumerated all the solutions. If 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. Without loss of generality, we assume that N is a power of two thus identify $[N]$ with $\{0,1\}^n$. The algorithm maintains a bucket of size 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 $O(\frac{1}{\varepsilon^2})$. To reduce the error probability to $1/\delta$,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.

First, 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. Our 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.

Building on the connection between model counting and F_0 estimation algorithms, 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. the size of $\cup_i S_i$. In this scenario, the goal is to design algorithms whose per-item time (time to process each S_i) is poly-logarithmic in the size of the universe. Structured set streams that are considered in the literature include 1-dimensional and multidimensional ranges [26, 31]. Several interesting problems, including max-dominance norm [6] and counting triangles in graphs [3], 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 DNF sets, affine spaces, and multidimensional ranges. Prior work on structured sets had to rely on involved analysis for each of the specific instances, while our work provides a general recipe for both analysis and implementation.

A natural question that arises from the transformation is the relationship between the space complexity of the streaming algorithms and the query complexity of the obtained model counting algorithms. We establish a relationship between these two quantities by showing that the space complexity is an upper bound on the query complexity.

2. NOTATION

We will assume the universe $[N] = \{0,1\}^n$. *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 interested in a *probably approximately correct* scheme that returns an *(ε, δ)-estimate c* of F_0, i.e.,

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

Model Couting: Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean variables. For a Boolean formula φ over variables X, let $\mathsf{Sol}(\varphi)$ denote the set of all satisfying assignments of φ. The *propositional model counting problem* is to compute $|\mathsf{Sol}(\varphi)|$ for a given formula φ. As in the case of F_0, we are interested in a *probably approximately correct* algorithm 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 of $|\mathsf{Sol}(\varphi)|$ i.e., $\Pr\left[\frac{|\mathsf{Sol}(\varphi)|}{1+\varepsilon} \le c \le (1+\varepsilon)|\mathsf{Sol}(\varphi)|\right] \ge 1 - \delta.$

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 $\{0,1\}^m$.

DEFINITION 1. *A family of hash functions $\mathcal{H}(n,m)$ is k-wise independent, denoted $\mathcal{H}_{\mathsf{k\text{-}wise}}(n,m)$, if $\forall \alpha_1, \alpha_2, \ldots, \alpha_k \in \{0,1\}^m$, for all distinct $x_1, x_2, \ldots x_k \in \{0,1\}^n$,*

$$\Pr_{h \in \mathcal{H}(n,m)}[(h(x_1) = \alpha_1) \wedge (h(x_2) = \alpha_2) \ldots (h(x_k) = \alpha_k)] = \frac{1}{2^{km}}$$

Explicit families. An explicit hash family that we use is $\mathcal{H}_{\mathsf{Toeplitz}}(n,m)$, which is known to be 2-wise independent [4]. 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 $b \in \mathbb{F}_2^{m \times 1}$, and $\mathbb{F}_2$ is the finite field of size 2. For a hash function $h : \{0,1\}^n \to \{0,1\}^m$, $h_\ell : \{0,1\}^n \to \{0,1\}^\ell$, $\ell \in \{1, \ldots, m\}$, is the function where $h_\ell(y)$ is the first ℓ bits of $h(y)$.

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 [2]. Their first algorithm, the Bucketing algorithm discussed above, is a modification of an F_0 estimation algorithm due to Gibbons and Tirthapura [16]. The second algorithm, which we call Minimum, is based on the idea that if we hash all the items of the stream, then $\mathcal{O}(1/\varepsilon^2)$-th minimum of the hash values can be used to compute a good estimate of F_0. The third algorithm, which we call Estimation, chooses a set of k functions, $\{h_1, h_2, \ldots, h_k\}$, such that each h_j is picked randomly from an $\mathcal{O}(\log(1/\varepsilon))$-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. 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 independently chosen hash functions from the family $\mathcal{H}$. We use H to denote the collection of hash functions returned, this collection is viewed as either a 1-dimensional array or as a 2-dimensional array. When H is a 1-dimensional array, $H[i]$ denotes the ith hash function of the collection and when H is a 2-dimensional array $H[i][j]$ is the $[i, j]$th hash function.

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 $\mathcal{O}(1/\varepsilon^2)$), then we increment the m_i as in line 8.

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

Algorithm 1 ComputeF0(n, ε, δ)

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

Sketch Properties. For each of the three algorithms, their corresponding sketches can be viewed as arrays of size $35 \log(1/\delta)$. 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 Each $\mathcal{S}[i]$ holds the lexicographically Thresh many smallest elements of $\{H[i](x) \mid x \in \mathbf{a}\}$.

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

Algorithm 2 ChooseHashFunctions(n, Thresh, t)

```
1: switch AlgorithmType do
2:     case AlgorithmType==Bucketing
3:         $H \leftarrow$ PickHashFunctions($\mathcal{H}_{\mathsf{Toeplitz}}(n, n), t$)
4:     case AlgorithmType==Minimum
5:         $H \leftarrow$ PickHashFunctions($\mathcal{H}_{\mathsf{Toeplitz}}(n, 3n), t$)
6:     case AlgorithmType==Estimation
7:         $s \leftarrow 10 \log(1/\varepsilon)$
8:         $H \leftarrow$ PickHashFunctions($\mathcal{H}_{s-\mathrm{wise}}(n, n), t \times$ Thresh)
   return $H$
```

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

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

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 (and the hash functions chosen) match, then 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.
2. View the formula φ as symbolic representation of the unique set $\mathbf{a}_u$ represented by the stream $\mathbf{a}$ such that $\mathsf{Sol}(\varphi) = \mathbf{a}_u$.
3. Given a formula φ and set of hash functions H, design an algorithm to construct sketch $\mathcal{S}$ such that the property $\mathcal{P}(\mathcal{S}, H, \mathsf{Sol}(\varphi))$ holds. Using the sketch $\mathcal{S}$, $|\mathsf{Sol}(\varphi)|$ can be estimated.

By applying the above recipe to the three F_0 estimation algorithms, we can derive corresponding model counting algorithms. In particular, applying the above recipe to

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

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

the $\mathsf{Bucketing}$ algorithm leads us to the state of the art hashing-based model counting algorithm, $\mathsf{ApproxMC}$, proposed by Chakraborty et al. [5]. Applying the above recipe to $\mathsf{Minimum}$ and $\mathsf{Estimation}$ allows us to obtain different model counting schemes. In this extended abstract we illustrate this transformation for the $\mathsf{Minimum}$-based algorithm.

3.2 Example Application of Recipe: Minimum-based Algorithm

We showcase the application of the recipe in the context of minimum-based algorithm. For a given multiset $\mathbf{a}$ (e.g.: a data stream or solutions to a model), we now specify the property $\mathcal{P}(\mathcal{S}, H, \mathbf{a}_u)$ as follows:

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 $\min(\frac{96}{\varepsilon^2}, |\mathbf{a}_u|)$. $\mathcal{P}$ is the property that specifies this relationship.

The following lemma due to Bar-Yossef et al. [2] establishes the relationship between the property $\mathcal{P}$ and the number of distinct elements of a multiset. Let $\max(S_i)$ denote the largest element of the set S_i.

LEMMA 1. *[2] Let $\mathbf{a} \subseteq \{0,1\}^n$ be a multiset. Let $H \subseteq \mathcal{H}_{\mathsf{Toeplitz}}(n, 3n)$ where each $H[i]$ is independently drawn from $\mathcal{H}_{\mathsf{Toeplitz}}(n, 3n)$, and $|H| = O(\log 1/\delta)$. Let $\mathcal{S}$ be such that $\mathcal{P}(\mathcal{S}, H, a_u)$ 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 $\mathsf{Minimum}$ algorithm for F_0 estimation to that of model counting given access to a subroutine that can compute $\mathcal{S}$ such that $\mathcal{P}(\mathcal{S}, H, \mathsf{Sol}(\varphi))$ holds. The following proposition establishes the existence and complexity of such a subroutine, called $\mathsf{FindMin}$.

PROPOSITION 1. *There is an algorithm $\mathsf{FindMin}$ that, given φ over n variables, $h \in \mathcal{H}_{\mathsf{Toeplitz}}(n, m)$, and p as input, returns a set, $\mathcal{B} \subseteq h(\mathsf{Sol}(\varphi))$ so that if $|h(\mathsf{Sol}(\varphi))| \leq p$, then $\mathcal{B} = h(\mathsf{Sol}(\varphi))$, otherwise $\mathcal{B}$ is the p lexicographically minimum elements of $h(\mathsf{Sol}(\varphi))$. Moreover, if φ is a CNF formula, then $\mathsf{FindMin}$ makes $\mathcal{O}(p \cdot m)$ calls to an NP oracle, and if φ is a DNF formula with k terms, then $\mathsf{FindMin}$ takes $\mathcal{O}(m^3 \cdot n \cdot k \cdot p)$ time.*

Equipped with Proposition 1, we are now ready to present the algorithm, called $\mathsf{ApproxModelCountMin}$, for model counting. Since the complexity of $\mathsf{FindMin}$ is PTIME when φ is in DNF, we have $\mathsf{ApproxModelCountMin}$ as a FPRAS for DNF formulas.

Algorithm 5 $\mathsf{ApproxModelCountMin}(\varphi, \varepsilon, \delta)$

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

THEOREM 1. *Given φ, ε,δ, $\mathsf{ApproxModelCountMin}$ returns Est such that*

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

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

We now give a proof of Proposition 1 by describing the subroutine $\mathsf{FindMin}$.

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 using 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 $\mathcal{H}_{\mathsf{Toeplitz}}(n, m)$ defined by a $m \times n$ binary matrix A. Let $\mathcal{C}$ be the set of hashed values of the satisfying assignments for φ: $\mathcal{C} = \{h(x) \mid x \models \varphi\} \subseteq \{0,1\}^m$. Let $\mathcal{C}_p$ be the first p elements of $\mathcal{C}$ in the lexicographic order. Our goal is to compute $\mathcal{C}_p$.

We illustrate an algorithm with running time $O(m^3np)$ to compute $\mathcal{C}_p$ when the formula is just a term T. This algorithm can easily be generalized to formulas with k-terms. Let T be a term with width w (number of literals) and $\mathcal{C} = \{Ax \mid x \models T\}$. By fixing the variables in T we get a vector b_T and an $N \times (n-w)$ matrix A_T so that $\mathcal{C} = \{A_T x + b_T \mid x \in \{0,1\}^{(n-w)}\}$. 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 $\mathcal{C}_p$ iteratively as follows: assuming we have computed the $(q-1)^{th}$ minimum of $\mathcal{C}$, we will compute the q^{th} minimum using a prefix-search strategy. We will use a subroutine to solve the following basic prefix-search 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 $\mathcal{C}$. 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 $\mathcal{C}$, 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 $\mathcal{C}_p$ in $O(m^3np)$ time. $\square$

3.3 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 [12, 11, 13, 14]. From a practical perspective, given the centrality of #DNF in the context of probabilistic databases [28, 27], a distributed DNF counting algorithm 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. [7]. 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 $\mathbf{a}$ while minimizing the total communication. This general framework has several direct applications and has been studied extensively (see [8, 18, 33] and the references therein).

In distributed DNF counting problem, 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 algorithms discussed in Section 3 can be extended to the distributed setting. We briefly describe the distributed implementation of the minimum based algorithm.

Distributed implementation of the minimum-based algorithm. The coordinator chooses hash functions $H[1], \ldots, H[t]$ from $\mathcal{H}_{\mathsf{Toeplitz}}(n, 3n)$ and sends them 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 $\log(\delta^{-1})$.

A straightforward implementation of the Bucketing algorithm leads to a distributed DNF counting algorithm whose communication cost is $\tilde{O}(k(n + 1/\varepsilon^2) \cdot \log(1/\delta))$ and time complexity per site is polynomial in n, ε^{-1}, and $\log(\delta^{-1})$. Similarly, the estimation based algorithm leads to a distributed algorithm with $\tilde{O}(k(n + 1/\varepsilon^2) \log(1/\delta))$ communication cost. However, we do not know a polynomial time algorithm to implement the last algorithm on 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 [33] 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 the 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.

4. FROM COUNTING TO STREAMING

In this section we consider the *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 $|\cup_i S_i|$ - the 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. We discuss two types of structured sets *DNF Sets* and *Affine Spaces*. As we mentioned in the introduction, other structured sets studied in the literature are single and multi-dimensional ranges. Our techniques also give algorithms for estimating F_0 of such structured set streams, which we omit in this extended abstract.

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 be a DNF formula over n variables. Then the DNF set corresponding to φ be 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 the 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.

THEOREM 2. *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 the Minimum-value based algorithm from Section 3.2 to this setting. The algorithm picks a hash function $h \in \mathcal{H}_{\mathsf{Toeplitz}}(n, 3n)$ and 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 1, the 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, the update time for the item φ_i is $O(n^4 \cdot k \cdot t)$. For obtaining an (ε, δ)-approximation, 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 space, each hash function uses $O(n)$ bits and and the algorithm stores $O(\frac{1}{\varepsilon^2})$ minimums, resulting in overall space usage of $O(\frac{n}{\varepsilon^2} \cdot \log \frac{1}{\delta})$. The proof of correctness follows from Lemma 1. $\square$

Instead of the Minimum-value based algorithm, we could also adapt the Bucketing-based algorithm to obtain an algorithm with similar space and time complexities.

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 A is a $n \times n$ matrix. Thus an affine stream consists of $\langle A_1, B_1\rangle, \langle A_2, B_2\rangle, \cdots$, where each $\langle A_i, B_i\rangle$ succinctly represents a set $\{x \in \{0,1\}^n \mid A_i x = B_i\}$. Here operations are over the finite field of size 2. For a $n \times n$ Boolean matrix A and a zero-one vector B, let $\mathsf{Sol}(\langle A, B\rangle)$ denote the set of all x that satisfy $Ax = B$.

PROPOSITION 2. *Given (A, B), $h \in \mathcal{H}_{\mathsf{Toeplitz}}(n, 3n)$, and t as input, there is an algorithm,* $\mathsf{AffineFindMin}$*, that returns a set, $\mathcal{B} \subseteq h(\mathsf{Sol}(\langle A, B\rangle))$ so that if $|h(\mathsf{Sol}(\langle A, B\rangle))| \leq t$, then $\mathcal{B} = h(\mathsf{Sol}(\langle A, B\rangle))$, otherwise $\mathcal{B}$ is the t lexicographically minimum elements of $h(\mathsf{Sol}(\langle A, B\rangle))$. The time taken by this algorithm is $O(n^4 t)$ and the space taken by the algorithm is $O(tn)$.*

The above proposition together with the minimum-based algorithm gives the following theorem.

THEOREM 3. *There is a streaming algorithm that computes a (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.*

5. RELATING SKETCH SPACE COMPLEXITY AND NP QUERY COMPLEXITY

Our investigations reveal surprising connections between algorithms for F_0 estimation and model counting that are of interest to two different communities.

It is noteworthy that the two communities often have different concerns: in the context of model counting, one is focused on the NP-query complexity while in the context of streaming, the focus is on the space complexity. This begs the question of whether the connections are a matter of happenstance or there is an inherent relationship between the space complexity in the context of streaming and the query complexity for model counting. We detail our investigations on the existence of such a relationship.

In the following, we will fold the hash function h also in the sketch S. With this simplification, instead of writing $P(S, h, \mathsf{Sol}(\varphi))$ we write $P(S, \mathsf{Sol}(\varphi))$.

We first introduce some complexity-theoretic notation. For a complexity class $\mathcal{C}$, a language L belongs to the complexity class $\exists \cdot \mathcal{C}$ if there is a polynomial $q(\cdot)$ and a language $L' \in \mathcal{C}$ such that for every x, $x \in L \Leftrightarrow \exists y, |y| \leq q(|x|), \langle x, y\rangle \in L'$.

Consider a streaming algorithm for F_0 that constructs a sketch such that $P(S, a_u)$ holds for some property P using which we can estimate $|a_u|$, where the size of S is poly-logarithmic in the size of the universe and polynomial in $1/\varepsilon$. Now consider the following *Sketch-Language*

$$L_{sketch} = \{\langle \varphi, S\rangle \mid P(S, \mathsf{Sol}(\varphi)) \text{ holds}\}.$$

THEOREM 4. *If L_{sketch} belongs to the complexity class $\mathcal{C}$, then there exists a $\mathrm{FP}^{\exists \cdot \mathcal{C}}$ model counting algorithm that estimates the number of satisfying assignments of a given formula φ. The number of queries made by the algorithm is bounded by the sketch size.*

Let us apply the above theorem to the minimum-based algorithm. The sketch language consists of tuples of the form $\langle \varphi, \langle h, v_1, \cdots, v_t\rangle\rangle$ where $\{v_1, \cdots v_t\}$ is the set of t lexicographically smallest elements of the set $h(\mathsf{Sol}(\varphi))$. It can be see that this language is in coNP. Since $\exists \cdot \mathrm{coNP}$ is same as the class Σ_2^{P}, we obtain a $\mathrm{FP}^{\Sigma_2^{\mathrm{P}}}$ algorithm. Since $t = O(1/\varepsilon^2)$ and h maps from n-bit strings to $3n$-bit strings, it follows that the size of the sketch is $O(n/\varepsilon^2)$. Thus the number of queries made by the algorithm is $O(n/\varepsilon^2)$.

Interestingly, all the model counting algorithms that were obtained following our recipe are probabilistic polynomial-time algorithms that make queries to languages in NP. The above generic transformation gives a deterministic polynomial-time algorithm that makes queries to a Σ_2^{P} language. Precisely characterizing the properties of the sketch that lead to probabilistic algorithms making only NP queries is an interesting direction to explore.

6. CONCLUSION AND FUTURE OUTLOOK

Our investigation led to a diverse set of results that unify over two decades of work in model counting and F_0 estimation. The viewpoint presented in this work has the potential to spur several new interesting research directions.

Higher Moments. There has been a long line of work on estimation of higher moments, i.e. F_k over data streams. A natural direction of future research is to adapt the notion of F_k in the context of model of counting and explore its applications. We expect extensions of the framework and recipe presented in this work to derive algorithms for higher frequency moments in the context of model counting.

Sparse XORs. In the context of model counting, the performance of underlying SAT solvers strongly depends on the size of XORs. The standard constructions lead to XORs of size $\Theta(n)$ and an interesting line of research has focused on the design of sparse XOR-based hash functions [17, 19, 10] culminating in showing that one can use hash functions of the form $h(x) = Ax + b$ wherein each entry of the m-th row of A is 1 with probability $\mathcal{O}(\frac{\log m}{m})$ [23]. 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.

Acknowledgements

We thank the anonymous reviewers of PODS 21 for valuable comments. We are grateful to Phokion Kolaitis for suggesting exploration beyond the transformation recipe that led to results in Section 5. Bhattacharyya was supported in part by the NRF Fellowship Programme [NRF-NRFFAI1-2019-0002] and an Amazon Research Award. Meel was supported in part by the NRF Fellowship Programme[NRF-NRFFAI1-2019-0004] and the AI Singapore Programme [AISG-RP-2018-005], and NUS ODPRT Grant [R-252-000-685-13]. Vinod was supported in part by NSF CCF-2130608, NSF CCF-184908 and NSF HDR:TRIPODS-1934884 awards. Pavan was supported in part by NSF CCF-2130536, NSF CCF-1849053 and NSF HDR:TRIPODS-1934884 awards.

7. REFERENCES

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[2] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Proc. of RANDOM*, volume 2483, pages 1–10, 2002.

[3] Z. Bar-Yossef, R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. of SODA*, pages 623–632. ACM/SIAM, 2002.

[4] J. L. Carter and M. N. Wegman. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112. ACM, 1977.

[5] S. Chakraborty, K. S. Meel, and M. Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proc. of IJCAI*, 2016.

[6] G. Cormode and S. Muthukrishnan. Estimating dominance norms of multiple data streams. In G. D. Battista and U. Zwick, editors, *Proc. of ESA*, volume 2832 of *Lecture Notes in Computer Science*, pages 148–160. Springer, 2003.

[7] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. *ACM Transactions on Algorithms (TALG)*, 7(2):1–20, 2011.

[8] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Continuous sampling from distributed streams. *Journal of the ACM (JACM)*, 59(2):1–25, 2012.

[9] P. Dagum, R. Karp, M. Luby, and S. Ross. An optimal algorithm for monte carlo estimation. *SIAM Journal on computing*, 29(5):1484–1496, 2000.

[10] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman. Low-density parity constraints for hashing-based discrete integration. In *Proc. of ICML*, pages 271–279, 2014.

[11] W. Feng, T. P. Hayes, and Y. Yin. Distributed symmetry breaking in sampling (optimal distributed randomly coloring with fewer colors). *arXiv preprint arXiv:1802.06953*, 2018.

[12] W. Feng, Y. Sun, and Y. Yin. What can be sampled locally? *Distributed Computing*, pages 1–27, 2018.

[13] W. Feng and Y. Yin. On local distributed sampling and counting. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 189–198, 2018.

[14] M. Fischer and M. Ghaffari. A simple parallel and distributed sampling technique: Local glauber dynamics. In *32nd International Symposium on Distributed Computing*, 2018.

[15] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985.

[16] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In A. L. Rosenberg, editor, *Proc. of SPAA*, pages 281–291. ACM, 2001.

[17] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. From sampling to model counting. In *Proc. of IJCAI*, pages 2293–2299, 2007.

[18] Z. Huang, K. Yi, and Q. Zhang. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *Proc. of PODS*, pages 295–306, 2012.

[19] A. Ivrii, S. Malik, K. S. Meel, and M. Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints*, pages 1–18, 2015.

[20] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proc. of PODS*, pages 41–52. ACM, 2010.

[21] R. Karp and M. Luby. Monte-carlo algorithms for enumeration and reliability problems. *Proc. of FOCS*, 1983.

[22] R. M. Karp, M. Luby, and N. Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429 – 448, 1989.

[23] K. S. Meel and S. Akshay. Sparse hashing for scalable approximate model counting: Theory and practice. In *Proc. of LICS*, 2020.

[24] K. S. Meel, A. A. Shrotri, and M. Y. Vardi. On hashing-based approaches to approximate dnf-counting. In *In Proc. of FSTTCS*, 2017.

[25] K. S. Meel, A. A. Shrotri, and M. Y. Vardi. Not all fprass are equal: Demystifying fprass for dnf-counting (extended abstract). In *Proc. of IJCAI*, 8 2019.

[26] A. Pavan and S. Tirthapura. Range-efficient counting of distinct elements in a massive data stream. *SIAM J. Comput.*, 37(2):359–379, 2007.

[27] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *Proceedings of the VLDB Endowment*, 1(1):797–808, 2008.

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

[29] M. Soos and K. S. Meel. 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)*, 2019.

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

[31] S. Tirthapura and D. P. Woodruff. Rectangle-efficient aggregation in spatial data streams. In *Proc. of PODS*, pages 283–294. ACM, 2012.

[32] L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[33] D. P. Woodruff and Q. Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 941–960, 2012.