Estimating Size of the Union of Sets in Streaming Model*

Kuldeep S. Meel ⓡ
National University of Singapore

N. V. Vinodchandran ⓡ
University of Nebraska, Lincoln

Sourav Chakraborty
Indian Statistical Institute, Kolkata

ABSTRACT

In this paper we study the problem of estimating the size of the union of sets $S_1, \ldots, S_M$ where each set $S_i \subseteq \Omega$ (for some discrete universe Ω) is implicitly presented and comes in a streaming fashion. We define the notion of Delphic sets to capture class of streaming problems where membership, sampling, and counting calls to the sets are efficient. In particular, we show our notion of Delphic sets capture three well known problems: Klee's measure problem (discrete version), test coverage estimation, and model counting of DNF formulas.

The Klee's measure problem is the problem of computing the volume of a union of multi-dimensional axis aligned rectangles. Test coverage estimation problem, in the context of combinatorial testing, focuses on the computation of the coverage measure for a given testing array. Combinatorial testing is a fundamental technique in hardware and software testing. Finally, given a DNF formula $\varphi = D_1 \vee D_2 \vee \ldots \vee D_M$, model counting problem seeks to compute the number of satisfying assignments of φ.

The primary contribution of our work is a simple and efficient sampling-based algorithm, called APS-Estimator, for outputing an (ε, δ)-approximation the cardinality of the union of (Delphic) sets in streaming setting. Our algorithm has the space complexity of $O(R \log |\Omega|)$ and the update time complexity of $O(R \log R \cdot \log(M/\delta) \cdot \log |\Omega|)$ where, $R = O\left(\log(M/\delta) \cdot \varepsilon^2\right)$. Consequently, for streaming Klee's measure problem, our algorithm provides the first algorithm with update time complexity that linearly depends on the dimension d for $d > 1$. This settles an open problem of Tirthpura and Woodruff (PODS-12). Furthermore, a straightforward application of our algorithm results in efficient algorithms for the coverage estimation problem and the DNF model counting problem in streaming setting.

We then investigate whether the space complexity for coverage estimation can be further improved, and in this context, we present another streaming algorithm that uses near-optimal space complexity but uses an update algorithm that is in P^{NP}, thereby showcasing an interesting time vs space trade-off in the streaming setting.

It is worth remarking that we view a key strength of our work is the simplicity of both the algorithm and its theoretical analysis, which makes it amenable to practical implementation and easy adoption.

CCS CONCEPTS

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

ACM Reference Format:
Kuldeep S. Meel ⓡ, N. V. Vinodchandran ⓡ, and Sourav Chakraborty. 2021. Estimating Size of the Union of Sets in Streaming Model. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3452021.3458333

1 INTRODUCTION

Estimating the size of the union of sets is a fundamental problem in Computer Science. The goal, usually, is to design an efficient randomized algorithm that can output an (ε, δ)-approximation of the size of the union of sets. We say that a random variable Z is an (ε, δ)-approximation of Y if $\Pr[|Z - Y| \leq \varepsilon |Y|] \geq 1 - \delta$.

In this paper, we focus on estimating the union of sets in a streaming setting. We consider a family of sets which we call *Delphic Sets* (see Definition 1.4), for which membership, sampling, and counting queries can be implemented *efficiently*. To showcase the generality of Delphic sets, we first present three problems arising in diverse domains that can be captured by Delphic sets: Klee's measure problem, test coverage estimation, and model counting for DNF formulas. The three problems are defined as follows:

Klee's Measure Problem

We define the discrete version of Klee's Measure Problem (KMP) in streaming setting.

Definition 1.1. *A d-dimensional axis aligned rectangle $\boldsymbol{r}$ over an universe $\Omega = [\Delta]^d$, where Δ is a totally ordered discrete set, is defined as $[a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$. Given a rectangle $\boldsymbol{r}$, let* $\mathsf{Range}(\boldsymbol{r})$ *denote 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 rectangle can be succinctly represented by the tuple $(a_1, b_1, \cdots a_d, b_d)$. The streaming Klee's Measure Problem is the following: Given a stream $\mathcal{R}$ of size M such that $\mathcal{R} = \langle \boldsymbol{r}_1, \boldsymbol{r}_2, \cdots \boldsymbol{r}_M \rangle$, where each item $\boldsymbol{r}_i$ is a d-dimensional rectangle, output a (ε, δ)-approximation of the volume of $\mathcal{R}$, where*

$$\mathsf{Volume}(\mathcal{R}) = |\cup_{1 \leq i \leq M} \mathsf{Range}(\boldsymbol{r}_i)|$$

Motivation. Klee's Measure Problem (KMP) is well investigated in computational geometry. Klee in 1977 introduced the one-dimensional version of the problem over reals: given n intervals in $\mathbb{R}$, compute the size of their union [32]. Klee presented an $O(n \log n)$ time algorithm for the problem in the traditional RAM model, which was later proved to be optimal by Fredmen and Weide [24]. Since its introduction, KMP has been studied extensively in computational geometry with the goal of designing efficient algorithms in the

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PODS '21, June 20–25, 2021, Virtual Event, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8381-3/21/06…$15.00
https://doi.org/10.1145/3452021.3458333

traditional RAM model. As certain data objects in databases can be represented by axis parallel multi-dimension rectangles, the KMP problem is significant in data bases [10, 34, 44, 52, 58]. In addition to computer science areas, algorithms for KMP have recently found applications in a varied range of practical areas including in environmental chemistry [20] and lunar archaeology [6].

The discrete version of KMP that we consider in this paper is studied in the streaming community as *range efficient* $\mathbb{F}_0$ computation [45, 55]. Other than its natural appeal, this problem is interesting because several significant problems, including max-dominance norm [21], counting triangles in graphs [3], and distinct summation problem [19] can be reduced to computing $\mathbb{F}_0$ over multi-dimensional discrete ranges.

Test Coverage Estimation Problem

Definition 1.2. *For an n-bit binary string $\boldsymbol{a} = a_1 a_2 \cdots a_n \in \{0,1\}^n$, its t-*coverage*, denoted by $\mathrm{Cov}_t(\boldsymbol{a})$, is defined as*

$$\mathrm{Cov}_t(\boldsymbol{a}) = \big\{(T, \boldsymbol{y}) \mid T \subset [n], |T| = t, \boldsymbol{y} \in \{0,1\}^t \text{ and the restriction of } a_i \text{ to indices in } T \text{ gives } \boldsymbol{y}\big\}$$

The streaming coverage estimation problem is defined as follows. Given a stream $\mathcal{A}$ of size M such that $\mathcal{A} = \langle \boldsymbol{a}_1, \ldots, \boldsymbol{a}_M \rangle$ where $\boldsymbol{a}_i \in \{0,1\}^n$, output an (ε, δ)-approximation of $|\mathrm{Cov}_t(\mathcal{A})|$ for any given t ($1 \le t \le n$), where $\mathrm{Cov}_t(\mathcal{A}) = \bigcup_{1 \le i \le M} \mathrm{Cov}_t(\boldsymbol{a}_i)$.

Motivation. Over the past half-century, the widespread adoption of software in diverse areas has necessitated the design of highly configurable systems wherein the end-user can choose a configuration of interest by setting the desired values to the configuration parameters. The space of configurations is often astronomical. For example, the possible number of configuration options of an embedded Linux for microcontrollers is 7.7×10^{417} [46]. Since it is not feasible to examine the behavior of a System Under Test (SUT) for all possible configurations, combinatorial testing has emerged as a dominant paradigm [33]. A configuration is specified by assigning values to the configuration parameters[1]. Motivated by the observation that most often the errors in system's results are due to the interaction of a relatively small number of parameters, the combinatorial testing paradigm focuses on covering t-combinations of parameters for a given t. Note that when each of the parameters takes binary values, the total number of possible t-combinations of parameters over a test of size n is $\binom{n}{t}2^t$.

In the critical area of software testing, there has been a long line of work in the design of test suite generators [9, 18, 35, 37, 42, 53, 54]. The earliest works focused on the design of the smallest size of test suites such that all the $\binom{n}{t}2^t$ combinations are covered. The optimal constructions still require test suites of size $O(t2^t \log n)$, which is intractable for large enough t. Consequently, one is often interested in achieving as high a coverage as possible within a given budget. Moreover, such generators are heuristic-based and fail to provide any rigorous guarantees on the quality of their generated test suites. In this context, it is critical to rigorously estimate the coverage of a given test suite. Note that in the context of binary parameters, a test can be specified as a binary string $\mathbf{a} = a_1 a_2 \ldots a_n \in \{0,1\}^n$. For many practical systems, n, the number of configurable parameters, is very large. Therefore, it is not practical to store all the tests. In addition, it is useful to have estimation methods that deal with a growing test suit. In particular, from a practical perspective, one envisions a coverage estimator to be a *monitor* with as small resource overhead as possible. These issues lend themselves to investigating the coverage estimation problem in a data streaming framework.

[1]While techniques developed in our work extends to scenarios wherein each of the parameters take values in finite domains, for simplicity of exposition, we will focus on the case where each of the parameters take a binary value.

Model Counting for DNF

Definition 1.3. *Let X be a set of n Boolean variables. A literal is a variable or its negation. A term is a conjunction of literals. A DNF formula φ over X is a disjunction of terms. For a DNF formula φ over n variables, let $\mathrm{Sol}(\varphi)$ denote the set of satisfying assignments of φ. The streaming DNF model counting problem is the following: Given a stream $\Phi = \langle D_1, D_2 \ldots, D_M \rangle$, where D_i is a term of n variables, output an (ε, δ)-approximation of $|\mathrm{Sol}(\varphi)|$ where φ is the DNF formula $\phi = D_1 \vee D_2 \vee \ldots \vee D_M$.*

Motivation. #DNF (model counting for DNF, also referred to as DNF counting, is often denoted by #DNF in the literature) is a fundamental problem in computer science with a wide variety of applications. Dalvi and Suici [23] showed that queries in probabilistic databases reduce to #DNF. Another important application of #DNF arises from the domain of network unreliability: given a graph $G = (V, E)$, wherein each edge e_i fails with probability p_i, we are interested in computing the probability that s and t are disconnected. Karger's seminal work reduces the computation of network unreliability to #DNF wherein each term represents a min-cut [29]. The past few years have witnessed a surge in interest for designing efficient FPRAS techniques for #DNF [39, 40].

1.1 Our Results

We define the notion of *Delphic sets* (Definition 1.4) and formulate the problem of estimating the size of the union of these sets in the streaming setting (Problem 1.5). This generic framework captures many union-of-sets size estimation problems, including all the three problems mentioned above. Then (in Theorem 1.6) we present an efficient algorithm for the problem. The efficiency of our algorithm is measured in terms of its worst case space complexity and its worst case per-item update time. Finally we show how our algorithm for Delphic sets gives new efficient algorithms for all the three problems mentioned above. The algorithm for KMP solves an open problem from the literature.

Delphic Sets as a Unifying Model

Let $\{\Omega_n\}_{n \ge 1}$ be a family of discrete sets indexed by n. We define the notion of Delphic family over $\{\Omega_n\}_{n \ge 1}$.

Definition 1.4. *Let $\mathcal{F} = \{\mathcal{F}_n\}_{n \ge 1}$ where $\mathcal{F}_n \subseteq 2^{\Omega_n}$. We call $\mathcal{F}$ a Delphic family if the following holds: for every n and for every set $S \in \mathcal{F}_n$ if the following queries can be done in $O(\log |\Omega_n|)$ time.*

(1) Know the size of the set S,
(2) Draw a uniform random sample from S, and
(3) Given any x check if $x \in S$.

Next, we define the following streaming problem.

Problem 1.5. *Fix a Delphic family $\mathcal{F}$. Given a stream $\mathcal{S} = \langle S_1, S_2, \ldots, S_M \rangle$ wherein each S_i belongs to $\mathcal{F}_n$, and $0 < \varepsilon, \delta < 1$ output an (ε, δ)-approximation of $|\bigcup_{i=1}^{M} S_i|$. An algorithm for solving this problem has only access to three queries listed in Definition 1.4.*

The main contribution of this is a work a new adaptive sampling-based algorithm for the above problem (Problem 1.5), as stated in the following theorem. We state it as a theorem.

Theorem 1.6. *There is a streaming algorithm, which we call,* APS-Estimator *that solves Problem 1.5. The algorithm has worst case space complexity $O(R \log |\Omega|)$ and update time is $O(R \log R \cdot \log(M/\delta) \cdot \log |\Omega|)$ where, $R = O\left(\log(M/\delta) \cdot \varepsilon^{-2}\right)$.*

Remark 1.7. *Tirthapura and Woodruff [55] (PODS-12) employed the notation $O^*(f)$ as a place holder for $O\left(f \cdot (\varepsilon^{-1} \cdot \log(\frac{|\Omega|M}{\delta}))^{O(1)}\right)$. Therefore, we can state the space and update time complexity of* APS-Estimator *as $O^*(1)$ in Tirthapura and Woodruff's notations.*

Klee's Measure Problem

We first observe that Klee's measure problem can be formulated as Delphic coverage by constructing set $S_i = \mathsf{Range}(\mathbf{r}_i)$ corresponding to each $\mathbf{r}_i$ and observing that each such S_i belongs to the Delphic family. Therefore, the following Corollary follows from Theorem 1.6

Corollary 1.8. *There is a streaming algorithm that given any reals numbers $\varepsilon, \delta < 1$, and a stream $\mathcal{R} = \langle \boldsymbol{r}_1, \boldsymbol{r}_2, \cdots \boldsymbol{r}_M \rangle$ where each $\boldsymbol{r}_i$ is a d-dimensional rectangle over $\Omega = [\Delta^d]$, computes an (ε, δ)-approximation of* Volume$(\mathcal{R})$. *The algorithm has worst case space $O\left(d(\log \Delta) \cdot \log(M/\delta) \cdot \varepsilon^{-2}\right)$ and update time complexity $O\left(d(\log \Delta) \cdot (\log(M/\delta))^2 \log\log(M/\delta) \cdot \varepsilon^{-2} \log \varepsilon^{-1}\right)$.*

Remark 1.9. *Corollary 1.8 provides the first efficient algorithm with linear dependence on d for Klee's measure problem in the streaming model. This resolves an open problem from Tirthapura and Woodruff [55]. We would like to remark that Tirthapura and Woodruff claimed an algorithm for KMP with space and update time complexity $O(d(\log \Delta) \cdot (\log M) \cdot \varepsilon^{-2} \cdot \log(\frac{1}{\delta}))$. However, they have retracted their claim [57]. Their method only yields per-item worst case update time complexity of $\mathrm{poly}((\log \Delta)^d, \varepsilon^{-1}, \log \frac{1}{\delta})$.*

Multi-Dimensional Arithmetic Progression. We then focus on generalization of Klee's measure problem by generalizing the notion of range $[a_i, b_i]$ to arithmetic progressions, which was studied previously by Pavan and Tirthapura [45] for $d = 1$. Let $[a, b, c]$ represent the arithmetic progression with common difference c in the range $[a, b]$, i.e., $a, a + c, a + 2c, a + jc$, where j is the largest integer such that $a + jc \leq b$. Consider a stream $\mathcal{R} = [\mathbf{r}_1, \mathbf{r}_2, \ldots \mathbf{r}_m]$ wherein each $\mathbf{r}_i = [a_1, b_1, c_1] \times \cdots \times [a_d, b_d, c_d]$. We generalize $\mathsf{Range}(r)$ to denote the set of tuple $\{(x_1, \ldots x_d)\}$ where $a_i \leq x_i \leq b_i$ and $x_i = a_i + k \cdot c_i$ for some positive integer k. Similarly, $\mathsf{Volume}(\mathcal{R}) = |\bigcup_{i=1}^{m} \mathsf{Range}(\mathbf{r}_i)|$. By observing that $\mathsf{Range}(\mathbf{r}_i)$ for d-dimensional arithmetic progress belongs to Delphic family, we obtain the following streaming algorithm.

Corollary 1.10. *There is a streaming algorithm that given any positive reals numbers $\varepsilon, \delta < 1$, and a stream $\mathcal{R} = \langle \boldsymbol{r}_1, \boldsymbol{r}_2, \cdots \boldsymbol{r}_M \rangle$ consisting of d-dimensional arithmetic progressions, computes an (ε, δ)-approximation of* Volume$(\mathcal{R})$. *The algorithm has worst case space complexity $O\left(d(\log \Delta) \cdot \log(M/\delta) \cdot \varepsilon^{-2}\right)$ and worst case update time complexity $O\left(d(\log \Delta) \cdot (\log(M/\delta))^2 \log\log(M/\delta) \cdot \varepsilon^{-2} \log \varepsilon^{-1}\right)$.*

Observe that Klee's measure problem is a special case of multi-dimensional arithmetic progress wherein $c_i = 1$. It is worth remarking that in comparison to prior work that focused on the special case of arithmetic progression for d=1, the algorithm for multi-dimensional ranges (i.e., Klee's measure problem) can be simply lifted to multi-dimensional arithmetic progression, and the space and time complexity does not change.

Test Coverage Estimation

Next, we observe that Test Coverage Estimation problem can be also formulated as Delphic coverage by constructing $S_i = \mathsf{Cov}(\mathbf{a}_i)$ for each $\mathbf{a}_i$, and again observing that each such S_i belongs to Delphic family.

Corollary 1.11. *There is a streaming algorithm* APS-Estimator *that given any reals numbers $\varepsilon, \delta < 1$, and a stream $\mathcal{A} = \langle \boldsymbol{a}_1, \cdots, \boldsymbol{a}_M \rangle$, where $\boldsymbol{a}_i \in \{0,1\}^n$, computes an (ε, δ)-approximation of $|\mathsf{Cov}_t(\mathcal{A})|$. The algorithm has space complexity $O\left(t(\log n) \cdot \log(M/\delta) \cdot \varepsilon^{-2}\right)$ and worst case update time complexity*

$$O\left(t(\log n) \cdot (\log(M/\delta))^2 \log\log(M/\delta) \cdot \varepsilon^{-2} \log \varepsilon^{-1}\right).$$

We also investigate whether the space complexity can be further improved in the context of test coverage estimation. We present a hashing-based algorithm that trades off improvement in space complexity with an increase in the overhead for time complexity that cane be captured via relying on oracles. The oracles we need is in the class NTISP(poly, LIN). This is the class of languages that can be solved by polynomial-time non-deterministic algorithms that also run in space linear in the input size.

Theorem 1.12. *There is a streaming algorithm* HashingEstimator *that takes a language $L \in$ NTISP(poly, LIN) as an oracle that given a stream $\mathcal{A} = \langle \boldsymbol{a}_1, \cdots, \boldsymbol{a}_M \rangle$, and real numbers $0 < \varepsilon, \delta < 1$, where each $\boldsymbol{a}_i \in \{0,1\}^n$ computes an (ε, δ)-approximation of $|\mathsf{Cov}_t(\mathcal{A})|$. The algorithm takes $O(t \log n \cdot \varepsilon^{-2} \cdot \log \frac{1}{\delta})$ space and $\mathrm{poly}(n, t, 1/\varepsilon)$ update time. Thus, if* NTISP(poly, LIN) $=$ DTISP(poly, LIN)*, then the algorithm will run in space $O(t \log n \cdot \varepsilon^{-2} \cdot \log \frac{1}{\delta})$ and time $\mathrm{poly}(n, t, 1/\varepsilon)$.*

The space complexity of the above algorithm, in general, is tight up to a $O(\log n)$ factor as for $t = n$. The problem reduces the problem of $\mathbb{F}_0$ computation in the traditional insertion-only data streams for which a lower bound of $\Omega(n+\varepsilon^{-2})$ is known [28] (notice that in our case items are from $\{0,1\}^n$). Therefore, the problem of designing algorithms that achieve tight bounds from both space and time complexity perspective remains open and it is difficult to establish a lower bounds on the space complexity other than the ones known for standard streaming as this will lead to a complexity class separation NTISP(poly, LIN) $\neq$ DTISP(poly, LIN),.

DNF Counting

We again observe that DNF counting can be formulated as Delphic coverage by constructing set $S_i = \mathsf{Sol}(D_i)$ corresponding to each term D_i, and observing that each S_i belongs to Delphic family. Therefore, the following corollary follows from Theorem 1.6.

Corollary 1.13. *There is a streaming algorithm that given any positive reals numbers $\varepsilon, \delta < 1$, and a stream $\langle D_1, D_2, \ldots D_M \rangle$ of terms over n variables, computes an (ε, δ)-approximation of $|\mathsf{Sol}(\varphi)|$ wherein $\varphi = D_1 \vee D_2 \vee \ldots \vee D_M$. The algorithm takes $O(n \cdot \log(M/\delta) \cdot \varepsilon^{-2})$ space and $O(n \cdot (\log(M/\delta))^2 \log\log(M/\delta) \cdot \varepsilon^{-2} \log \varepsilon^{-1})$ update time.*

1.2 Techniques

Our main algorithm, APS-Estimator, is a simple and elegant sampling-based algorithm. The core idea is that one can estimate $|\cup_i S_i|$ if it was possible to sample every element from $\cup_i S_i$ identically and independently with probability p where p is appropriately chosen (more formally, a number that is of the order of $\frac{1}{\varepsilon^2 \cdot |\cup_i S_i|}$. For now, let us focus on how we would sample every element of $\cup_i S_i$ identically and independently with probability p for some fixed value of p. The key insight is that we can maintain a bucket $\mathcal{X}$ of elements with an invariant that

$$\text{After processing } S_j, \text{ we have every element of } \cup_{i=1}^{j} S_i \text{ sampled independently and identically with probability } p \tag{1}$$

As a starting point when S_1 arrives, we can sample every element of S_1 with probability p by first sampling N_i from $\mathsf{Bin}(|S_i|, p)$ where Bin refers to the Binomial distribution. Now, consider the aforementioned invariant holds when S_{j+1} arrives in the stream. If we were to simply sample every element of S_{j+1} independently and identically with probability p, we would run the risk of biasing $\mathcal{X}$ to contain elements that appear in multiple S_i. It turns there is simple but powerful idea to handle the issue of elements that appear in multiple S_i: when S_{j+1} arrives, we first remove all elements of $\mathcal{X}$ that are in S_{j+1} and then sample every element of S_{j+1} with probability p and add them to $\mathcal{X}$. Such a step ensures that the probability $x \in S_{j+1}$ and $x \in \mathcal{X}$ after processing S_{j+1} is p. Now, we return to the issue of how do we determine p. Since we don't know p a priori, we can first set $p = 1$ and proceed as mentioned above. Whenever we have $|\mathcal{X}| \geq \mathsf{thresh}$, we throw every element of $\mathcal{X}$ identically and independently with probability $\frac{1}{2}$ and set $p = p/2$. It is worth observing that p itself is now a random variable, and therefore, an assertion such as "every element of $\cup_{i=1}^{j+1} S_i$ sampled independently and identically with probability p" is no longer a meaningful statement. Accordingly, our technical analysis defines suitable random variables to argue that $\frac{|\mathcal{X}|}{p}$ provides (ε, δ)-guarantees.

Note that for the sake of presenting a unified framework, we derive time complexity solely based on the three properties of Delphic sets, which allows us only black-box access to a random sample. The time complexity analysis requires generation of samples without repetition, which can be be performed more efficiently in specific contexts of $\mathsf{Cov}(\mathbf{a}_i)$, $\mathsf{Range}(\mathbf{r}_i)$, $\mathsf{Sol}(D_i)$. We leave a refined analysis of the time complexity for specific problems as future work.

We view a key strength of our work is the simplicity of both the algorithm, APS-Estimator, and its theoretical analysis. Our algorithm's simplicity makes it amenable to practical implementation, while our analysis's simplicity lends itself to easy verification and adoption.

Organization. The rest of the paper is organized as follows: We discuss related work in Section 2. We then discuss notations and preliminaries in Section 3. The paper's primary technical contribution is presented in Section 4, which consists of four parts. Section 4.1 provides the proof of the main technical theorem 1.6. We then demonstrate Theorem 1.6 can be applied in different contexts, Klee's Measure Problem (Section 4.2), coverage estimation (Section 4.3), and DNF counting (Section 4.4) to obtain efficient streaming algorithms. We finally conclude in Section 5.

2 RELATED WORK

Starting with the seminar work of Alon, Matias, and Szegedy [2], the streaming model of computation has emerged as an important area of research in theoretical computer science. This model is well suited to investigate algorithmic problems that arise from real life situations dealing with large data. A central focus of investigation has been on estimating the frequency moments $\mathbb{F}_k$ of a stream of data items. In particular, considerable work has been done in designing algorithms for estimating the the 0^{th} frequency moment ($\mathbb{F}_0$), the number of distinct elements in the stream, culminating in the development of an algorithm with optimal space complexity $O(\log|\Omega| + \frac{1}{\varepsilon^2})$ and $O(1)$ update time [28], where Ω is the universe.

In the case when items in the data stream succinctly represent a set of elements of the universe, $\mathbb{F}_0$ estimation becomes estimation of size of the *union of sets*. The union-of-sets problem has been studied in approximate counting literature. The line of work that is closest to ours is the work on DNF counting problem initiated by Karp and Luby [30]. Since the work of Karp and Luby, substantial research has gone into understanding various aspects of DNF counting problem including designing hashing-based algorithms [13, 22, 31, 39–41]. We note that Karp-Luby algorithm can be adapted to counting union of sets in the streaming setting to get an algorithm with space and time complexity $O(\frac{M \log|\Omega|}{\epsilon^2} \log M \log n)$. In comparison, we achieve only a logarithmic dependence on M.

As mentioned in the introduction, Klee's Measure Problem (KMP) is a fundamental problem that is well investigated in computational geometry with the focus of designing efficient algorithms in the traditional RAM model. Klee introduced the one-dimensional version of the problem over reals and presented $O(n \log n)$ time algorithm where n is the number of line segments [32]. Fredmen and Weide showed that this is optimal in time (under certain model) [24]. Since then substantial work has gone into extending the algorithms to multidimensional case [5, 8, 14, 16, 17, 26, 43] and also with space complexity considerations [16, 56].

Discrete version of KMP over streaming model has been considered before. However the success has been limited [3, 45, 51, 55]. Pavan and Thirthapura considered the problem for one dimensional ranges over the discrete domain $\{1, \ldots, n\}$ and gave an algorithm with $O(\varepsilon^{-2} \log n \log 1/\delta)$ space and $O(\log n/\varepsilon \log 1/\delta)$ update time [45]. Sharma, Busch, Vaidyanathan, Rai, and Trahan considered the two-dimensional version but only gave a $O(\sqrt{\log U})$-approximation for the general case where U is the total number of discrete points in the space [47]. Thirthapura and Woodruff [55] considered the general d dimensional problem. They presented an algorithm, based of range efficient implementations of *count sketch* algorithm [15] and recursive sketches [7, 27], which is efficient in

space complexity. However the update time of the algorithm has exponential dependency on the dimension d (see Remark 1.9). In a concurrent work, Pavan ⓡ Vinodchandran ⓡ Bhattacharyya ⓡ Meel[2] also proposed another hashing-based technique with exponential dependence on the dimension d.

3 NOTATIONS AND PRELIMINARIES

We will denote by $[n]$ the set $\{1, 2, \ldots, n\}$ and by $\binom{[n]}{t}$ the set of all subsets of $[n]$ of size t. For any $n \in \mathbb{N}$ and any $p \in [0, 1]$ we will also use $\mathsf{Bin}(n, p)$ to denote the Binomial distribution over the set of natural numbers $\{0, \ldots, n\}$ where probability of a number $0 \le m \le n$ is $\binom{n}{m} p^m (1-p)^{n-m}$.

At any point the input item is a length n string. However, as done in the case of traditional space bounded computations, for counting space, we will not include the space required to represent the input item. We will consider that input is available on a read-only input tape and do not contribute to the space used by the algorithm. In this paper we consider unit-cost model and assume all basic operations including arithmetic operations on words can be performed in unit time.

3.1 Concentration Inequalities

Our technical analysis employs the standard concentration inequalities: Chernoff bound, Chebychev's bound, and Paley-Zygmund Inequality.

Theorem 3.1 (Chernoff Bound). *Suppose $v_1, ..., v_n$ are independent random variables taking values in $\{0, 1\}$. Let $V = \sum_{i=1}^{n} v_i$ and $\mu = \mathbb{E}[V]$ then*

$$\Pr\left(|V - \mu| \ge \delta\mu\right) \le 2e^{-\frac{\delta^2\mu}{3}}$$

Theorem 3.2 (Chebychev's Inequality). *Let Z be a random variable with expected value μ and non-zero variance σ^2. Then for any real number $k > 0$,*

$$\Pr\left(|Z - \mu| \ge k\sigma\right) \le \frac{1}{k^2}.$$

Theorem 3.3 (Paley–Zygmund inequality). *If $Z \ge 0$ is a random variable with finite variance, and if $0 \le \theta \le 1$, then*

$$\Pr\left(Z > \theta\mathbb{E}[Z]\right) \ge (1-\theta)^2 \frac{\mathbb{E}[Z]^2}{\mathbb{E}[Z^2]}.$$

3.2 Coupon Collector Problem

For the proof of Theorem 1.6 we will need the following theorem, popularly known as the Coupon Collector Problem.

Theorem 3.4. *Given access to uniform random samples from a set T and a number $r \le |T|$, let Z_r be a random variable that stand for the number of independent uniform random samples from T needed before we get r distinct samples from T. Then*

$$\Pr\left[Z_r > \beta r \log r\right] \le r^{-\beta+1}.$$

[2] ⓡ refers to the randomized author ordering.

3.3 Independently picking elements from a set with a fixed probability

How to efficiently sample a subset $\mathcal{L}$ of a set S so that each element of S is in $\mathcal{L}$ independently with probability p for a given probability value p? The naive approach will be go over all the element of the set S and decide whether to pick that element in the set $\mathcal{L}$ or not independently. By one can achieve the same by the following sampling process $\mathcal{P}$: *first draw a number K according to the Binomial distribution $\mathsf{Bin}(|S|, p)$ and then draw K distinct elements at random from S.* The proof of correctness of this process is given below.

Claim 3.5. *The sampling process $\mathcal{P}$ samples each element of S independently with probability p.*

Proof. For any $x \in S$, the probability of choosing x is $\sum_k \frac{k}{|S|} \Pr[k \sim \mathsf{Bin}(|S|, p)]$. Using the definition of Binomial distribution we have

$$\begin{aligned}
\sum_{k=0}^{|S|} \frac{k}{|S|} \Pr\left[k \sim \mathsf{Bin}(|S|, p)\right] &= \sum_{k=0}^{|S|} \frac{k}{|S|} \binom{|S|}{k} p^k (1-p)^{|S|-k} \\
&= \sum_{k=1}^{|S|} \binom{|S|-1}{k-1} p^k (1-p)^{|S|-k} \\
&= p \cdot \sum_{k=0}^{|S|-1} \binom{|S|-1}{k} p^k (1-p)^{|S|-1-k} \\
&= p
\end{aligned}$$

Now, to prove that each element of S is chosen independently let us calculate the probability of choosing any specific $x_1, \ldots, x_t$ from S. Note that, when one picks a subset of size k from S, probability that all $x_1, \ldots, x_t$ is picked is 0 if $k < t$ and is $\binom{k}{t}/\binom{|S|}{t}$ otherwise. So, the probability that our process will of choose $x_1, \ldots, x_t$ is

$$\begin{aligned}
&\sum_{k=0}^{|S|} \frac{\binom{k}{t}}{\binom{|S|}{t}} \Pr\left[k \sim \mathsf{Bin}(|S|, p)\right] \\
= &\sum_{k=t}^{|S|} \frac{\binom{k}{t}}{\binom{|S|}{t}} \Pr\left[k \sim \mathsf{Bin}(|S|, p)\right] \\
= &\sum_{k=t}^{|S|} \frac{\binom{k}{t}}{\binom{|S|}{t}} k p^k (1-p)^{|S|-k} \\
= &\sum_{k=t}^{|S|} \binom{|S|-t}{k-t} p^k (1-p)^{|S|-k} \\
= &\; p^t \cdot \sum_{k=0}^{|S|-t} \binom{|S|-t}{k} p^k (1-p)^{|S|-t-k} \\
= &\; p^t
\end{aligned}$$

Thus, for any set of t elements in S probability that the t elements are chosen is p^t. This proves that all the items of S are chosen independently with probability p.

□

3.4 Cascade Binomial Sampling

Our sampling process involves sampling the Binomial distribution $\mathsf{Bin}(n, p)$ for a positive integer n (cardinality of a set in the stream)

and a probability p that is adaptively chosen. In general to sample the distribution $\mathrm{Bin}(n, pq)$ the process we employ a cascading process: first sample $\mathrm{Bin}(n, p)$ to get a number l and then sample $\mathrm{Bin}(l, q)$. Let $\mathcal{S}$ denote this process. For completeness we give proof of correctness that $\mathcal{S}$ is same as sampling from $\mathrm{Bin}(n, pq)$.

Claim 3.6. *Let n be a positive integer and $0 \le p, q \le 1$ be probability values. Consider the following sampling process $\mathcal{S}$: First get l according to $\mathrm{Bin}(n, p)$ and then get k according to $\mathrm{Bin}(l, q)$. Then the sampling process $\mathcal{S}$ is same as sampling $\mathrm{Bin}(n, pq)$.*

Proof. We will show that $\Pr(k \leftarrow \mathcal{S}) = \Pr(k \leftarrow \mathrm{Bin}(n, pq)) = \binom{n}{k}(pq)^k(1-pq)^{n-k}$.

$$
\begin{aligned}
& \Pr(k \leftarrow \mathcal{S}) \\
= & \sum_{l=0}^{n} \Pr\left(k \leftarrow \mathrm{Bin}(l, q) \mid l \leftarrow \mathrm{Bin}(n, p)\right) \cdot \Pr(l \leftarrow \mathrm{Bin}(n, p)) \\
= & \sum_{l=0}^{n} \binom{l}{k} q^k (1-q)^{l-k} \cdot \binom{n}{l} p^l (1-p)^{n-l} \\
= & \sum_{l=0}^{n} \binom{n}{l}\binom{l}{k} p^l (1-p)^{n-l} q^k (1-q)^{l-k} \\
= & \sum_{l=0}^{n} \binom{n}{k}\binom{n-k}{l-k} p^l (1-p)^{n-l} q^k (1-q)^{l-k} \\
= & \binom{n}{k} \sum_{l \ge k}^{n} \binom{n-k}{l-k} p^l (1-p)^{n-l} q^k (1-q)^{l-k} \\
= & \binom{n}{k} \sum_{r=0}^{n-k} \binom{n-k}{r} p^{r+k} (1-p)^{n-r-k} q^k (1-q)^r \\
= & \binom{n}{k} (pq)^k \sum_{r=0}^{n-k} \binom{n-k}{r} p^r (1-p)^{n-r-k} (1-q)^r \\
= & \binom{n}{k} (pq)^k \sum_{r=0}^{n-k} \binom{n-k}{r} p^r (1-q)^r (1-p)^{n-r-k} \\
= & \binom{n}{k} (pq)^k \left((1-q)p + 1 - p\right)^{n-k} \\
= & \binom{n}{k} (pq)^k (1-pq)^{n-k} \\
= & \Pr\left(k \leftarrow \mathrm{Bin}(n, pq)\right)
\end{aligned}
$$

□

3.5 Independently picking elements from a set combined with cascade Binomial Sampling

Using Claims 3.6 and 3.5 we can combine the two random processes to have different random processes with the same output. We will crucially use this in our proofs.

Claim 3.7. *For any S, p the two random processes* RandomProcess1 *and* RandomProcess2 *are same. That is, the distribution of their output is same.*

Algorithm 1 RandomProcess1

1: $N \leftarrow \mathrm{Bin}(|S|, p)$
2: $N \leftarrow \mathrm{Bin}(N, 1/2)$
3: Let T be a subset of S obtained by drawing N distinct samples from S

Algorithm 2 RandomProcess2

1: $N \leftarrow \mathrm{Bin}(|S|, p)$
2: Let T be a subset of S obtained by drawing N distinct samples from S
3: Throw each of the elements in T independently by probability $1/2$ drawn

Proof. By Claims 3.6 at the end of line 2 in RandomProcess1 the number N is distribution according to $\mathrm{Bin}(S, p/2)$ and so by Claim 3.5 the set T in RandomProcess1 is a subset of S where each element of S is picked independently with probability $p/2$. On the other, by Claim 3.5 in RandomProcess2 at the end of line 2 the set T is a subset of S where each element of S is picked independently with probability p. And finally in line 3 since each element of T is thrown independently with probability $1/2$ so the final T is a subset of S where each element of S is picked independently with probability $p/2$ which is also what RandomProcess1 produces. □

3.6 Pairwise Independent Hash functions

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

Definition 3.8. *A family of hash functions $\mathcal{H}(n, m)$ is 2−wise independent if $\forall \alpha_1, \alpha_2 \in \{0,1\}^m$, distinct $x_1, x_2, \in \{0,1\}^n, h \xleftarrow{R} \mathcal{H}(n, m)$,*

$$\Pr[(h(x_1) = \alpha_1) \wedge (h(x_2) = \alpha_2)] = \frac{1}{2^{2m}} \tag{2}$$

Explicit families. In this work, one hash family of particular interest is $\mathsf{H}_{\mathsf{Teop}}(n, m)$, which is known to be 2-wise independent [11]. The family is defined as follows: $\mathsf{H}_{\mathsf{Teop}}(n, m) \triangleq \{h : \{0,1\}^n \to \{0,1\}^m\}$ is the family of functions of the form $h(x) = Ax + b$ with $A \in \mathbb{F}_2^{m \times n}$ and $b \in \mathbb{F}_2^{m \times 1}$ where A is a uniformly randomly chosen Toeplitz matrix of size $m \times n$ while b is uniformly randomly matrix of size $m \times 1$. it is worth noticing that $\mathsf{H}_{\mathsf{Teop}}$ can be represented with $\Theta(n)$-bits.

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.

4 THE ALGORITHMS

4.1 Coverage of Delphic Sets

We restate the theorem that we prove in this section.

Theorem 1.6. *There is a streaming algorithm, which we call,* APS-Estimator *that solves Problem 1.5. The algorithm has worst case space complexity $O(R \log |\Omega|)$ and update time is $O(R \log R \cdot \log(M/\delta) \cdot \log |\Omega|)$ where, $R = O\left(\log(M/\delta) \cdot \varepsilon^{-2}\right)$.*

Algorithm 3 APS-Estimator

1: Initialize $p \leftarrow 1$
2: Thresh $\leftarrow \max\left(12 \cdot \frac{\ln(24/\delta)}{\varepsilon^2}, 6(\ln \frac{6}{\delta} + \ln M)\right)$
3: Initialize $\mathcal{X} \leftarrow \emptyset$
4: **for** $i = 1$ to M **do**
5: **for** all $s \in \mathcal{X}$ **do**
6: **if** $s \in S_i$ **then**
7: remove s from $\mathcal{X}$
8: $N_i \leftarrow \mathsf{Bin}(|S_i|, p)$
9: **while** $N_i + |\mathcal{X}| \geq$ Thresh **do**
10: $N_i \leftarrow \mathsf{Bin}(N_i, 1/2)$
11: Throw away each element of $\mathcal{X}$ with probability $\frac{1}{2}$
12: $p = p/2$
13: Initialize $\ell \leftarrow 0; k \leftarrow 0$
14: **while** $\ell < 1 + N_i \log N_i \log(\frac{3M}{\delta})$ **do**
15: $\ell \leftarrow \ell + 1$
16: Draw a random sample y from S_i
17: **if** $y \notin \mathcal{X}$ **then**
18: Add y to $\mathcal{X}$; $k \leftarrow k + 1$
19: **if** $k = N_i$ **then break**
20: **if** $\ell = 1 + N_i \log N_i \log\left(\frac{3M}{\delta}\right)$ **then** Output 0
21: Output $\frac{|\mathcal{X}|}{p}$

Proof. We present the algorithm APS-Estimator and prove its correctness and we will then analyse its complexity.

We start with observing steps 13 to 20. Note that the purpose of the steps 13 to 20 is to pick N_i distinct samples from S_i and add then to the set $\mathcal{X}$. If 0 is not output in steps 20 then it means the algorithm has successfully sampled N_i distinct elements from S_i and added then to $\mathcal{X}$. Thus if we assume that APS-Estimator does not output 0 in steps 20 (for any i) then the algorithm can be simplified and steps 13 to 20 can be replaced with Algorithm 4

We now prove the correctness of the algorithm assuming that the algorithm does not output 0 in steps 20. Let Fail denote the event that the algorithm outputs 0 in steps 20.

Assuming $\overline{\mathsf{Fail}}$*:* As argued earlier, proving the correctness of APS-Estimator assuming $\overline{\mathsf{Fail}}$ is same as proving the correctness of Algorithm 4.

In steps 8 to 12 of Algorithm 4 the algorithm can be thought of as initially (in steps 8) deciding to pick N_i samples from S_i but there may not be enough space to store all the picked samples (along with the elements already in $\mathcal{X}$. So in steps 9 to 12 it keeps on reducing the size of $\mathcal{X}$ (in steps 11) and also the number of sampled items (in steps 10) until the sample items can be added to $\mathcal{X}$. To save the space required to temporarily store the sampled items before them may be thrown away in (in steps 10) in APS-Estimator the sampling is done at the very last after the number of samples that will be stored is determined. But if we are only concerned with the correctness of the algorithm and not the space complexity by Claim 3.7 the outputs of Algorithm 4 and Algorithm 5 have the same distribution.

Algorithm 4

1: Initialize $p \leftarrow 1$
2: Thresh $\leftarrow \max\left(12 \cdot \frac{\ln(24/\delta)}{\varepsilon^2}, 6(\ln \frac{6}{\delta} + \ln M)\right)$
3: Initialize $\mathcal{X} \leftarrow \emptyset$
4: **for** $i = 1$ to M **do**
5: **for** all $s \in \mathcal{X}$ **do**
6: **if** $s \in S_i$ **then**
7: remove s from $\mathcal{X}$
8: $N_i \leftarrow \mathsf{Bin}(|S_i|, p)$
9: **while** $N_i + |\mathcal{X}| \geq$ Thresh **do**
10: $N_i \leftarrow \mathsf{Bin}(N_i, 1/2)$
11: Throw away each element of $\mathcal{X}$ with probability $\frac{1}{2}$
12: $p = p/2$
13: Draw N_i distinct samples from S_i and add them to $\mathcal{X}$
14: Output $\frac{|\mathcal{X}|}{p}$

Algorithm 5

1: Initialize $p \leftarrow 1$
2: Thresh $\leftarrow \max\left(12 \cdot \frac{\ln(24/\delta)}{\varepsilon^2}, 6(\ln \frac{6}{\delta} + \ln M)\right)$
3: Initialize $\mathcal{X} \leftarrow \emptyset$
4: **for** $i = 1$ to M **do**
5: **for** all $s \in \mathcal{X}$ **do**
6: **if** $s \in S_i$ **then**
7: remove s from $\mathcal{X}$
8: $N_i \leftarrow \mathsf{Bin}(|S_i|, p)$
9: Draw N_i distinct samples from S_i and add them to $\mathcal{X}$
10: **while** $|\mathcal{X}| \geq$ Thresh **do**
11: Throw away each element of $\mathcal{X}$ with probability $\frac{1}{2}$
12: $p = p/2$
13: Output $\frac{|\mathcal{X}|}{p}$

In short proving the correctness of APS-Estimator assuming $\overline{\mathsf{Fail}}$ is same as proving the correctness of Algorithms 5. We now proceed with proving the correctness of Algorithms 5.

Let $S^{(i)} = S_1 \cup \cdots \cup S_i$. To begin with, we will inductively define the array $Y_j^{(i)}$ of distributions that we may get over the sets $S^{(i)}$ during the execution of the algorithm for $i \in \{1, \ldots, m\}$, and $p = 1/2^j$. The set $Y_0^{(1)}$ is defined to be $S^{(1)}$. Then we consider two sampling moves:

Next set move: To obtain $Y_j^{(i+1)}$ from $Y_j^{(i)}$, we remove all elements $Y_j^{(i)}$ that are in S_{i+1} (corresponding to step 5 to 7 of Algorithm 5), and sample all elements of S_{i+1} independently with probability $\frac{1}{2^j}$ and add those to $Y_j^{(i)}$ (corresponding to step 9 of Algorithm 5).

Decrease p move: To obtain $Y^{(i)}_{j+1}$ from $Y^{(i)}_j$, we sample each element of $Y^{(i)}_j$ independently with probability 1/2. This corresponds to step 10 to step 12 in Algorithm 5

We make the following simple observation.

Observation 4.1. *For all i, j, the random set $Y^{(i)}_j$ is distributed as independent samples from set $S^{(i)}$, each element with probability 2^{-j}.*

First, observe that Algorithm 5 can be viewed as inducing distribution over $(\mathcal{X}, p)$ such that after processing S_i, we have $\mathcal{X} = Y^i_j$ where $p = 2^{-j}$. It is worth remarking that both $\mathcal{X}$, p are random variables.

For $1 \le i \le M$, and $j \ge 0$, we define the event $E^{(i)}_j$ as the event that after the algorithm processes the first i sets, the value of p is 2^{-j}. These events can depend arbitrarily on all the random sets $Y^{(i)}_j$'s sampled during the execution of the algorithm. This corresponds to an arbitrary sequence of the aforementioned moves over the array of distributions $Y^{(i)}_j$ depending on the decisions of the Algorithm.

Let $A^{(i)}_j$ be the event that the cardinality of $Y^{(i)}_j$ is not in

$$\left[|S^{(i)}|2^{-j}(1-\varepsilon), |S^{(i)}|2^{-j}(1+\varepsilon)\right].$$

Let j^* be the smallest j such that $\frac{1}{2^j} < \frac{\mathsf{Thresh}}{4|S^{(M)}|}$.

We note that the probability that the algorithm outputs a number not in $\left[|S^{(M)}|(1-\varepsilon), |S^{(M)}|(1+\varepsilon)\right]$ is

$$\Pr\left[\bigcup_{j=0}^{\infty}(E^{(M)}_j \wedge A^{(M)}_j)\right] \le \sum_{j=0}^{j^*-1}\Pr\left[A^{(M)}_j\right] + \Pr\left[\bigcup_{j\ge j^*}(E^{(M)}_j\right].$$

We bound $\sum_{j=0}^{j^*-1}\Pr[A^{(M)}_j]$ and $\Pr[\bigcup_{j\ge j^*}E^{(M)}_j]$.

By Theorem 3.1, we have that since $|S^{(M)}|\frac{1}{2^{j^*-1}} > \frac{\mathsf{Thresh}}{4}$

$$\sum_{j=0}^{j^*-1}\Pr[A^{(M)}_j] \le 2e^{-\varepsilon^2\mathsf{Thresh}/12} + 2e^{-\varepsilon^2\mathsf{Thresh}/6} + \cdots$$
$$< 4e^{-\varepsilon^2\mathsf{Thresh}/12} \le \frac{\delta}{6},$$

where we use that $\mathsf{Thresh} > \frac{12\ln(24/\delta)}{\varepsilon^2}$.

To bound $\Pr[\bigcup_{j\ge j^*}E^{(M)}_j]$, we notice that the event $\bigcup_{j\ge j^*}E^{(M)}_j$ occurs only if the decrease p move happens from $Y^{(i)}_{j^*-1}$ to $Y^{(i)}_{j^*}$ for some i. This happens if $\mathsf{Bin}(|S^{(i)}|, \frac{1}{2^{j^*-1}})$ is larger than Thresh. Since $\frac{1}{2^{j^*-1}} < \frac{\mathsf{Thresh}}{2|S^{(M)}|}$, and so $\mathsf{Thresh} - |S^{(i)}|\cdot\frac{1}{2^{j^*-1}} > \frac{\mathsf{Thresh}}{2}$, by Lemma 3.1, the probability of this event is at most

$$e^{-\mathsf{Thresh}/6} \le \frac{\delta}{6M},$$

where we use that $\mathsf{Thresh} \ge 6\left(\ln\frac{6}{\delta} + \ln M\right)$. By a union bound over i, we have that

$$\Pr\left[\bigcup_{j\ge j^*}E^{(M)}_j\right] \le M\cdot\frac{\delta}{6M} = \frac{\delta}{6}.$$

Therefore, we have

$$\Pr\left[\bigcup_{j=0}^{\infty}(E^{(M)}_j \wedge A^{(M)}_j)\right] \le \frac{\delta}{3}. \tag{3}$$

Note that the above probability is assuming $\overline{\mathsf{Fail}}$.

Probability of Fail*:* For any i the algorithm will output 0 in steps 20 is the algorithm is unable to get N_i distinct samples from S_i even after drawing $N_i \log N_i \log(3M/\delta)$ samples from S_i. From Theorem 3.4 we know that the probability of this happening is at most $\delta/3M$. So using union bound over all i the probability of Fail is at most $\delta/3$.

Correctness of the algorithm. By Equation 3 and the fact that $\Pr[\mathsf{Fail}] \le \delta/3$ we have that the algorithm outputs an (ϵ, δ)-estimate of $\left|\cup_{i=1}^{M}S_i\right|$ with probability at least $(1-\delta)$.

Space and Update Time Complexity. Firstly, it is easy to see that the set $\mathcal{X}$ cannot ever cross Thresh, and so the space complexity is upper bounded by $O(\mathsf{Thresh}\cdot(\log|\Omega|))$. For the update time, we observe that for steps 5 to 7 the algorithm makes at most $|\mathcal{X}|$ number of queries (number (3) query from Definition 1.4). For steps 8 the algorithm APS-Estimator makes 1 query (number (1) query from Definition 1.4). And finally, in steps 14 to 19 the algorithm may have to make more than $N_i \log N_i \log(3M/\delta)$ queries (number (2) query from Definition 1.4) to obtain N_i distinct elements from S_i. So the update time of the algorithm is bounded by $O(\mathsf{Thresh}\log(\mathsf{Thresh})\cdot\log(3M/\delta)\cdot(\log|\Omega|))$. So the worst case space and update time complexity of APS-Estimator is as stated in the statement of Theorem 1.6.

□

4.2 Klee's Measure Problem

We return to Klee's measure problem in streaming setting and show that a straightforward application of Theorem 1.6 leads to the first space and update time efficient algorithm. As a first step, we show that the set $\mathsf{Range}(\mathbf{r})$ of a rectangle $\mathbf{r}$ is Delphic.

Lemma 4.2. *For each rectangle r,* $\mathsf{Range}(r)$ *belongs to Delphic family.*

Proof. Note that $\mathsf{Range}(\mathbf{r}) \subseteq [\Delta]^d$

(1) For a given $\mathbf{r}$, the size of the set is simply $\prod_{i=1}^{d}(b_i - a_i)$ can be computed in $O(d)$ time.
(2) To draw a uniform random sample $(x_1, \ldots x_d)$, x_i is sampled uniformly at random from $[a_i, b_i]$, which can be accomplished in $O(d\log\Delta)$
(3) Given $x = (x_1, \ldots x_d)$, we can check if $x \in \mathsf{Range}(\mathbf{r})$ by checking if for all i, $x_i \in [a_i, b_i]$, which can be accomplished in $O(d)$

□

Now, upon observing that each $\mathbf{r}_i$ implicitly represents $\mathsf{Range}(\mathbf{r}_i)$, the following corollary immediately follows from Theorem 1.6.

Corollary 1.8. *There is a streaming algorithm that given any reals numbers $\varepsilon, \delta < 1$, and a stream $\mathcal{R} = \langle \mathbf{r}_1, \mathbf{r}_2, \cdots \mathbf{r}_M\rangle$ where each $\mathbf{r}_i$ is a d-dimensional rectangle over $\Omega = [\Delta^d]$, computes an (ε, δ)-approximation of* $\mathsf{Volume}(\mathcal{R})$*. The algorithm has worst case space*

$O\left(d(\log\Delta)\cdot\log(M/\delta)\cdot\varepsilon^{-2}\right)$ *and update time complexity* $O\left(d(\log\Delta)\cdot(\log(M/\delta))^2\log\log(M/\delta)\cdot\varepsilon^{-2}\log\varepsilon^{-1}\right)$.

The notion of range $[a_i, b_i]$ can be generalized to arithmetic progressions, which was studied previously by Pavan and Tirthpura for $d = 1$. Our sampling model allows us to derive streaming algorithms for a more general model comprising of d-dimensional arithmetic progressions: Let $[a, b, c]$ represent the arithmetic progression with common difference c in the range $[a, b]$, i.e., $a, a + c, a + 2c, a + jc$, where j is the largest integer such that $a + jd \leq b$. Consider a stream $\mathcal{R} = \langle \mathbf{r}_1, \mathbf{r}_2, \ldots \mathbf{r}_M \rangle$ wherein each $\mathbf{r}_i = [a_1, b_1, c_1] \times \cdots \times [a_d, b_d, c_d]$. We generalize $\mathsf{Range}(\mathbf{r})$ to denote the set of tuple $\{(x_1, \ldots x_d)\}$ where $a_i \leq x_i \leq b_i$ and $x_i = a_i + k \cdot c_i$ for some positive integer k. Similarly, $\mathsf{Volume}(\mathcal{R}) = |\bigcup_{i=1}^{m} \mathsf{Range}(\mathbf{r}_i)|$. To apply Theorem 1.6, we first show that $\mathsf{Range}(\mathbf{r})$ of a d-dimensional arithmetic progressions is Delphic.

Lemma 4.3. *For each d-dimensional arithmetic progressions $\mathbf{r}$, the set* $\mathsf{Range}(\mathbf{r})$ *belongs to Delphic family.*

Proof. Note that $\mathsf{Range}(\mathbf{r}) \subseteq [\Delta]^d$

(1) For a given $\mathbf{r}$, the size of the set is simply $\prod_{i=1}^{d}\left(\lfloor\frac{b_i-a_i}{c_i}\rfloor+1\right)$, which can be computed in $O(d)$ time.
(2) To draw a uniform sample $(x_1, \ldots x_d)$, x_i is sampled uniformly at random from $[a_i, b_i, c_i]$. Note that to sample from $[a_i, b_i, c_i]$, we first draw a number k uniformly at random from $(\lfloor\frac{b_i-a_i}{c_i}\rfloor+1)$, and then return $a_i + k \cdot c_i$
(3) Given $x = (x_1, \ldots x_d)$, we can check if $x \in \mathsf{Range}(\mathbf{r})$ by checking if for all i, $x_i \in a_i + k \cdot c_i$ for some positive integer k and $x_i \leq b_i$, which can be accomplished in $O(d)$ time.

□

We can now invoke Theorem 1.6 to obtain the following result.

Corollary 1.10. *There is a streaming algorithm that given any positive reals numbers $\varepsilon, \delta < 1$, and a stream $\mathcal{R} = \langle \mathbf{r}_1, \mathbf{r}_2, \cdots \mathbf{r}_M \rangle$ consisting of d-dimensional arithmetic progressions, computes an (ε, δ)-approximation of* $\mathsf{Volume}(\mathcal{R})$. *The algorithm has worst case space complexity $O\left(d(\log\Delta)\cdot\log(M/\delta)\cdot\varepsilon^{-2}\right)$ and worst case update time complexity $O\left(d(\log\Delta)\cdot(\log(M/\delta))^2\log\log(M/\delta)\cdot\varepsilon^{-2}\log\varepsilon^{-1}\right)$.*

4.3 Test Coverage Estimation

We now focus on test coverage estimation problem and provide the proof for Corollary. We begin with the following lemma.

Lemma 4.4. *For each $\boldsymbol{a}_i$, $\mathsf{Cov}(\boldsymbol{a}_i)$ belogns to Delphic family*

Proof. Note that $\mathsf{Cov}(\mathbf{a}_i) \subseteq \{0,1\}^{t\log n+t}$.

(1) $|\mathsf{Cov}(\mathbf{a}_i| = \binom{n}{t}$
(2) Drawing a uniform sample is simply drawing a $\log_2(\binom{n}{t})$-bit strings uniformly at random, which can be accomplished in $O\left(\log(\binom{n}{t})\right)$.
(3) Finally, to check if $x = (T, y) \in \mathsf{Cov}(\mathbf{a}_i)$, we simply compute the restriction of a_i over T and perform a string equivalence check.

□

Upon observing that each $\mathbf{a}_i$ implicitly represents $\mathsf{Cov}(\mathbf{a}_i)$, the following corollary immediately follows from Theorem 1.6

Corollary 1.11. *There is a streaming algorithm* APS-Estimator *that given any reals numbers $\varepsilon, \delta < 1$, and a stream $\mathcal{A} = \langle \boldsymbol{a}_1, \cdots, \boldsymbol{a}_M \rangle$, where $\boldsymbol{a}_i \in \{0,1\}^n$, computes an (ε, δ)-approximation of $|\mathsf{Cov}_t(\mathcal{A})|$. The algorithm has space complexity $O\left(t(\log n)\cdot\log(M/\delta)\cdot\varepsilon^{-2}\right)$ and worst case update time complexity*

$$O\left(t(\log n)\cdot(\log(M/\delta))^2\log\log(M/\delta)\cdot\varepsilon^{-2}\log\varepsilon^{-1}\right).$$

A natural question one wonders is whether the space complexity of the above algorithm is tight. The following observation illustrates that the space complexity can be improved but at the cost of update time.

Observation 4.5. *There is a streaming algorithm that given a stream $\mathcal{A} = \langle \boldsymbol{a}_1, \cdots, \boldsymbol{a}_M \rangle$, computes an (ε, δ)-approximation of $|\mathsf{Cov}_t(\mathcal{A})|$. The algorithm takes $O(t\log n + \varepsilon^{-2})\log\frac{1}{\delta})$ space and $O(n^t)$ update time.*

Proof. For the item $\mathbf{a}_i$, produce a steam with $\binom{n}{t}$ elements $\mathbf{b} \in \{0,1\}^t$ that are covered by $\mathbf{a}_i$ and use the algorithm due to Kane, Nelson, and Woodruff [28] to compute the F_0 of the resulting stream. The reduction takes $O(t\log n)$ space and $O(n^t)$ time. The $\mathbb{F}_0$ computation on the resulting stream takes $O((t\log n + \varepsilon^{-2})\log 1/\delta))$ and $O(1)$ processing time per item. □

One wonders whether we can exploit the trade off between space and time complexity to arrive at an algorithm that has better space complexity than Corollary 1.11 and time complexity that is better than Observation 4.5. To this end, we present a hashing-based strategy that can replace the time complexity of $O(n^t)$ with linearly many calls to NP oracle.

Before, we state the result, we take a detour and formally introduce the notion of pairwise independent hash functions.

Hashing-based Coverage Estimation

We begin with an alternate view of $\mathsf{Cov}_t(a)$. We can view $\mathbf{a}$ as a string $a_1a_2\cdots a_n \in \{0,1\}^n$. Then,

$$\mathsf{Cov}_t(\mathbf{a}) = \{\langle i_1, i_2, \cdots, i_t; b_1, b_2, \cdots, b_t\rangle \mid i_1 < i_2 < \cdots < i_t \text{ and } a_{i_j} = b_j \forall 1 \leq j \leq t\}.$$

Therefore, each element of $\mathsf{Cov}_t(\mathbf{a})$ can be seen as a binary string of length $(t+1)\log n$ and we will focus on the universe $\{0,1\}^{(t+1)\log n}$.

We will now prove the following theorem that improves upon the space complexity of Corollary 1.11 when allowed access to an oracle in NTISP(poly, LIN).

Theorem 1.12. *There is a streaming algorithm* HashingEstimator *that takes a language $L \in$ NTISP(poly, LIN) as an oracle that given a stream $\mathcal{A} = \langle \boldsymbol{a}_1, \cdots, \boldsymbol{a}_M \rangle$, and real numbers $0 < \varepsilon, \delta < 1$, where each $\boldsymbol{a}_i \in \{0,1\}^n$ computes an (ε, δ)-approximation of $|\mathsf{Cov}_t(\mathcal{A})|$. The algorithm takes $O(t\log n\cdot\varepsilon^{-2}\cdot\log\frac{1}{\delta})$ space and* poly$(n, t, 1/\varepsilon)$ *update time. Thus, if* NTISP(poly, LIN) = DTISP(poly, LIN)*, then the algorithm will run in space $O(t\log n\cdot\varepsilon^{-2}\cdot\log\frac{1}{\delta})$ and time* poly$(n, t, 1/\varepsilon)$.

Algorithm 6 can be viewed as an adaptation of Gibbons and Tirthapura's algorithm for F_0 estimation wherein every element of the stream a represents the associated set $\mathsf{Cov}_t(a)$. The algorithm first selects a $T = O(\log(1/\delta))$ pairwise independent hash functions. We then maintain two arrays of size T: array of sketches $\mathcal{X}$

and an associated array of levels (represented as integers) m. In particular, corresponding to every hash function $H[i]$, we maintain an associated level $m[i]$ and the corresponding $\mathcal{X}[i]$. The core underlying idea is that at every point, $\mathcal{X}[i]$ consists of the all $y \in \{0,1\}^{(t+1)\log n}$ covered by the stream so far such that for each y, we have $H[i]_{m[i]}(y) = 0^{m[i]}$. Furthermore, to avoid storing exponentially many elements, we ensure that $\mathcal{X}[i] < \text{thresh}$. To this end, we first check whether $|(H[i]^{-1}_{m[i]}(0^{m[i]}) \cap \text{Cov}_t(a)) \cup \mathcal{X}[i]|$ is less than thresh, and in such a case, we increment the value of $m[i]$ until the check in line 9 succeeds. Note that whenever we increment the value of $m[i]$ in line 13, we also refine the set of $\mathcal{X}[i]$ to ensure that for all the elements $y \in \mathcal{X}[i]$, it is indeed the case that $H[i]_{m[i]}(y) = 0^{m[i]}$.

Algorithm 6 HashingEstimator

```
1:  thresh ← 1 + 9.84(1 + 1/ε²)
2:  T ← 35 log(1/δ)
3:  m[1 : T] ← 0; X[1 : T] ← 0
4:  H ← ChooseHashFunctions(T)
5:  while !EndStream do
6:      a ← input()
7:      for i ∈ {0, 1, ..., T} do
8:          while true do
9:              if |(H[i]^{-1}_{m[i]}(0^{m[i]}) ∩ Cov_t(a)) ∪ X[i]| < thresh
    then
10:                 X[i] = X[i] ∪ {H[i]^{-1}_{m[i]}(0^{m[i]}) ∩ Cov_t(a)}
11:                 break;
12:             else
13:                 m[i]+ = 1
14:                 X[i] = X[i] ∩ H[i]^{-1}_{m[i]}(0^{m[i]})
15: return Median ( { size(X[i]) × 2^{m[i]} }_i )
```

Lemma 4.6. *Let* $c = \text{Median}\left(\left\{\text{size}(\mathcal{X}[i]) \times 2^{m[i]}\right\}_i\right)$. *Then*

$$\Pr\left[\frac{\text{Cov}_t(\mathcal{A})}{1+\varepsilon} \le c \le \text{Cov}_t(\mathcal{A})(1+\varepsilon)\right] \ge 1-\delta$$

While allowing larger constants in the expression of thresh would allow us to use the arguments of Gibbons and Tirthapura [25], we provide an alternate proof building on Chakraborty, Meel, and Vardi [13] and Meel (r) Akshay [38] that allows us to obtain better constants. We believe the proof is of independent interest as it exploits the nested properties of the sets $H[i]^{-1}_{m[i]}(0^{m[i]})$, which have shown to provide significant performance improvements in the context of model counting [13]. The proof is deferred to Appendix.

Now, the key question remains is of the time complexity. In essence, we are interested in the time complexity of the check in line 9. Observe that for all $y \in \{0,1\}^{(t+1)\log n}$, the check $y \in \text{Cov}_t(a)$ can be performed in $O((t+1)\log n)$ time. The following proposition follows from Lemma 3.7 of Bellare, Goldreich, and Petrank [4].

Lemma 4.7. *Given a hash function* $h \in \mathsf{H}_{\mathsf{Teop}}(n,m)$ *and* $\mathcal{X}$, *there is a polynomial time algorithm* $\mathcal{A}$ *and NP sets* M_1, M_2 *such that* $\mathcal{A}^{M_1,M_2}(h,a,m,p,\mathcal{X})$ *outputs 0 if* $|(h^{-1}(0^m) \cap \text{Cov}_t(a)) \cup \mathcal{X}| \ge p$ *and* $h^{-1}(0^m) \cap \text{Cov}_t(a)$ *otherwise. The algorithm makes* $O(p(t+1)\log n)$ *calls to NP oracle and uses* $O(p(t+1)\log n)$ *space.*

Proof. The proof follows similar structure to Lemma 3.7 of [4].

$M_1 = \{(a,h,m,p) \mid \exists y_1, \ldots y_p$ such that $y_1, \ldots y_p$ are distinct and
$\forall i \in [p] : y_i \in \text{Cov}_t(a) \wedge h(y_i) = 0^m \wedge y_i \notin \mathcal{X}\}$

$M_2 = \{(a,h,m,p',i',j') \mid \exists y_1 \prec y_2 \ldots \prec y_{p'}$ such that
$y_1 \cdot y_{p'}$ are distinct and $\forall i \in [p'] : y_i \in \text{Cov}_t(a) \wedge$
$h(y_i = 0^m \wedge (i' \le p') \wedge (j' \le |y_{i'}|$ and j'-th bit of $y_{i'}$ is 0 $\}$

Note that $\prec$ denotes a total ordering on $\{0,1\}^{(t+1)\log n}$. The algorithm $\mathcal{A}$ first invokes M_1 to output 0 if $|(h^{-1}(0^m) \cap \text{Cov}_t(a) \cup \mathcal{X}| < p$. Otherwise, $\mathcal{A}$ queries M_2 to determine all the $y \in (h^{-1}(0^m) \cap \text{Cov}_t(a)) \cup \mathcal{X}$ bit by bit. Finally is easy to verify that the languages M_1 and M_2 can be accepted by a non-deterministic polynomial-time machine that uses only linear space. □

Observe that Lemma 4.6 and 4.7 imply the desired theorem.

It is worth remarking that the usage of SAT oracle does not necessarily imply that the algorithm HashingEstimator would not be able handle practical instances. The past three decades have witnessed a sustained development of algorithmic techniques that allow modern SAT solvers to handle problems involving millions of variables [36]. Furthermore, the queries to SAT oracle in HashingEstimator can be expressed as conjunction of CNF and XOR constraints, also known as CNF-XOR formulas. Owing to the critical importance of CNF-XOR formulas in the context of hashing-based techniques for model counting [12, 13, 50], there has been a renewed focus on the design of efficient techniques for CNF-XOR formulas [48, 49]. We leave design of practical tools for future work.

4.4 Model Counting of DNF

Consider a DNF formula $\varphi = D_1 \vee D_2 \ldots \vee D_M$ wherein each D_i is a term defined a set of n Boolean variables.. We denote the set of all satisfying assignments of φ by $\text{Sol}(\varphi)$. Given φ, the problem of model counting seeks to compute $|\text{Sol}(\varphi)|$. We focus on the problem of model counting for DNF formulas in streaming setting, i.e., where the terms D_i arrive one by one over a stream. We first begin with the following lemma.

Lemma 4.8. *For each* D_i, $\text{Sol}(D_i)$ *belongs to Delphic family.*

Proof. Note that $\text{Sol}(D_i) \in \{0,1\}^n$. Let $|D_i|$ denote the size of the term D_i

- $\text{Sol}(D_i) = 2^{n-|D_i|}$, which can computed in $O(n)$ time.
- Drawing a uniform sample from $\text{Sol}(D_i)$ is simply draw $n-|D_i|$ bits uniformly at random, which can be accomplished in $O(n)$ time.
- Finally, check $x \in \text{Sol}(D_i)$ can be accomplished in $O(n)$ time.

□

Since each D_i implicitly represents $\text{Sol}(D_i)$, the following corollary follows from Theorem 1.6.

Corollary 1.13. *There is a streaming algorithm that given any positive reals numbers* $\varepsilon, \delta < 1$, *and a stream* $\langle D_1, D_2, \ldots D_M \rangle$ *of terms over n variables, computes an* (ε,δ)*-approximation of* $|\text{Sol}(\varphi|)$

wherein $\varphi = D_1 \vee D_2 \vee \ldots \vee D_M$. *The algorithm takes* $O(n \cdot \log(M/\delta) \cdot \varepsilon^{-2})$ *space and* $O(n \cdot (\log(M/\delta))^2 \log\log(M/\delta) \cdot \varepsilon^{-2} \log \varepsilon^{-1})$ *update time.*

5 CONCLUSION AND FUTURE OUTLOOK

To summarize, our investigations led us to design a surprisingly simple yet efficient scheme for computation of size of union of sets belonging to Delphic family. We then show that the notion of Delphic sets can capture three fundamental problems in streaming setting: Klee's measure problem, coverage estimation problem, and DNF counting. For each of these problems, we provide efficient streaming algorithms.

Crucially, we believe the simplicity of our scheme should make it amenable to practical implementation and adoption. From technical perspective, we sketch out three directions of interest:

Generalization of Delphic Sets In this work, we limited our focus to three problems to showcase the generalizability of the notion of Delphic sets. As a future work, an interesting direction of work would be to study other streaming problems that reduce to Delphic sets. To this end, one line of work would be to relax the requirement of $O(\log |\Omega|)$ time to $O(\log |\Omega|)^{O(1)}$ for membership, counting, and sampling queries.

Higher Moments the computation of union of sets corresponds to $\mathbb{F}_0$ (0-th frequency moment) estimation. In this context, a natural question would be whether we can generalize our sampling-based strategy for $\mathbb{F}_k$, i.e., k-th frequency moment estimation.

Beyond Insertion-Only Streams The framework presented in this paper handles insertion only streams. The past two decades have witnessed a long line of work on richer turnstile models that allow deletion. Therefore, an interesting direction of future work would be to explore sampling-based framework for turnstile model.

Complexity independent of M The space and time complexity of APS-Estimator has logarithmic dependence on M, which is in line with the $O^*(1)$ notation introduced by Tirthapura and Woodruff [55]. However, observe that we can cast the distinct element problem as a special case of union of Delphic sets wherein every set is simply a singleton. In case of distinct element problem, the algorithms without dependence on M are known. Therefore, an interesting direction for future work would be to investigate whether sampling-based framework can lead to algorithms whose space and update time complexity are independent of M.

ACKNOWLEDGMENTS

The revised (and arguably much cleaner) proof of APS-Estimator is based on the proof sketch of Divesh Aggarwal and Maciej Obremski. We thank the anonymous reviewers of PODS 21 for valuable comments. Meel was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme[NRF-NRFFAI1-2019-0004] and AI Singapore Programme [AISG-RP-2018-005], and NUS ODPRT Grant [R-252-000-685-13]. Vinodchandran was supported in part by NSF CCF-184908 and NSF HDR:TRIPODS-1934884 awards.

REFERENCES

[1] Dimitris Achlioptas and Panos Theodoropoulos. 2017. Probabilistic model counting with short XORs. In *Proc. of SAT*. Springer, 3–19.
[2] 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. https://doi.org/10.1006/jcss.1997.1545
[3] 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.
[4] M. Bellare, O. Goldreich, and E. Petrank. 2000. Uniform Generation of NP-witnesses using an NP-oracle. *Information and Computation* 163, 2 (2000), 510–526.
[5] Jon Louis Bentley. 1977. *Algorithms for Klee's rectangle problems.* Technical Report. Technical Report, Computer.
[6] Valentin Tertius Bickel, Jordan Aaron, Andrea Manconi, Simon Loew, and Urs Mall. 2020. Impacts drive lunar rockfalls over billions of years. *Nature Communications* 11, 2862 (2020).
[7] Vladimir Braverman and Rafail Ostrovsky. 2010. Recursive Sketching For Frequency Moments. *CoRR* abs/1011.2571 (2010).
[8] Karl Bringmann and Tobias Friedrich. 2010. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Comput. Geom.* 43, 6-7 (2010), 601–610.
[9] Renée C Bryce and Charles J Colbourn. 2009. A density-based greedy algorithm for higher strength covering arrays. *Software Testing, Verification and Reliability* 19, 1 (2009), 37–53.
[10] Mengchu Cai, Dinesh Keshwani, and Peter Z Revesz. 2000. Parametric rectangles: A model for querying and animation of spatiotemporal databases. In *International Conference on Extending Database Technology*. Springer, 430–444.
[11] 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.
[12] S. Chakraborty, K. S. Meel, and M. Y. Vardi. 2013. A Scalable Approximate Model Counter. In *Proc. of CP*. 200–216.
[13] 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*.
[14] Timothy M Chan. 2010. A (slightly) faster algorithm for Klee's measure problem. *Computational Geometry* 43, 3 (2010), 243–250.
[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] Eric Y Chen and Timothy M Chan. 2005. Space-efficient algorithms for Klee's measure problem. *algorithms* 3, 5 (2005), 6.
[17] Bogdan S Chlebus. 1998. On the Klee's measure problem in small dimensions. In *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer, 304–311.
[18] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. 1997. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering* 23, 7 (1997), 437–444.
[19] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. 2004. Approximate aggregation techniques for sensor databases. In *Proceedings. 20th International Conference on Data Engineering*. IEEE, 449–460.
[20] Pedro J. Copado-Méndez, Carlos Pozo, Gonzalo Guillén-Gosálbez, and Laureano Jiménez. 2016. Enhancing the ε-constraint method through the use of objective reduction and random sequences: Application to environmental problems. *Computers & Chemical Engineering* 87 (2016), 36 – 48. https://doi.org/10.1016/j.compchemeng.2015.12.016
[21] Graham Cormode and Shanmugavelayutham Muthukrishnan. 2003. Estimating dominance norms of multiple data streams. In *European Symposium on Algorithms*. Springer, 148–160.
[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] Nilesh Dalvi and Dan Suciu. 2007. Efficient query evaluation on probabilistic databases. *The VLDB Journal* 16, 4 (2007), 523–544.
[24] Michael L Fredman and Bruce Weide. 1978. On the complexity of computing the measure of ⋃[ai, bi]. *Commun. ACM* 21, 7 (1978), 540–544.
[25] Phillip B Gibbons and Srikanta Tirthapura. 2001. Estimating simple functions on the union of data streams. In *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*. 281–291.
[26] Joachim Gudmundsson and Rasmus Pagh. 2017. Range-Efficient Consistent Sampling and Locality-Sensitive Hashing for Polygons. In *28th International Symposium on Algorithms and Computation, ISAAC 2017, December 9-12, 2017, Phuket, Thailand (LIPIcs)*, Yoshio Okamoto and Takeshi Tokuyama (Eds.), Vol. 92. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 42:1–42:13.
[27] Piotr Indyk and David P. Woodruff. 2005. Optimal approximations of the frequency moments of data streams. In *Proc. of STOC*. ACM, 202–208.
[28] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. 2010. An optimal algorithm for the distinct elements problem. In *Proc. of PODS*. ACM, 41–52.

[29] David R Karger. 2001. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM review* 43, 3 (2001), 499–522.

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

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

[32] Victor Klee. 1977. Can the Measure of be Computed in Less than O (n log n) Steps? *The American Mathematical Monthly* 84, 4 (1977), 284–285.

[33] D Richard Kuhn, Raghu N Kacker, and Yu Lei. 2013. *Introduction to combinatorial testing*. CRC press.

[34] Iosif Lazaridis and Sharad Mehrotra. 2001. Progressive approximate aggregate queries with a multi-resolution tree structure. *Acm sigmod record* 30, 2 (2001), 401–412.

[35] Robert Mandl. 1985. Orthogonal Latin squares: an application of experiment design to compiler testing. *Commun. ACM* 28, 10 (1985), 1054–1058.

[36] Joao Marques-Silva, Inês Lynce, and Sharad Malik. 2009. Conflict-driven clause learning SAT solvers. In *Handbook of satisfiability*. ios Press, 131–153.

[37] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Rohit Gheyi, and Sven Apel. 2016. A comparison of 10 sampling algorithms for configurable systems. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 643–654.

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

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

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

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

[42] Changhai Nie and Hareton Leung. 2011. A survey of combinatorial testing. *ACM Computing Surveys (CSUR)* 43, 2 (2011), 1–29.

[43] Mark H Overmars and Chee-Keng Yap. 1991. New upper bounds in Klee's measure problem. *SIAM J. Comput.* 20, 6 (1991), 1034–1045.

[44] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. 2001. Efficient OLAP operations in spatial data warehouses. In *International Symposium on Spatial and Temporal Databases*. Springer, 443–459.

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

[46] Tobias Pett, Thomas Thüm, Tobias Runge, Sebastian Krieter, Malte Lochau, and Ina Schaefer. 2019. Product sampling for product lines: the scalability challenge. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019*. ACM, 14:1–14:6.

[47] Gokarna Sharma, Costas Busch, Ramachandran Vaidyanathan, Suresh Rai, and Jerry L. Trahan. 2015. Efficient transformations for Klee's measure problem in the streaming model. *Computational Geometry* 48, 9 (2015), 688 – 702. https://doi.org/10.1016/j.comgeo.2015.06.007

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

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

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

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

[52] Yufei Tao and Dimitris Papadias. 2004. Range aggregate processing in spatial databases. *IEEE Transactions on Knowledge and Data Engineering* 16, 12 (2004), 1555–1570.

[53] Keizo Tatsumi. 1987. Test case design support system. In *Proc. International Conference on Quality Control (ICQC'87)*. 615–620.

[54] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys (CSUR)* 47, 1 (2014), 1–45.

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

[56] Jan Vahrenhold. 2007. An in-place algorithm for Klee's measure problem in two dimensions. *Information processing letters* 102, 4 (2007), 169–174.

[57] David Woodruff. 2020. personal communication.

[58] Donghui Zhang, Alexander Markowetz, Vassilis J Tsotras, Dimitrios Gunopulos, and Bernhard Seeger. 2008. On computing temporal aggregates with range predicates. *ACM Transactions on Database Systems (TODS)* 33, 2 (2008), 1–39.

APPENDIX

We provide a Proof of Lemma 4.6. We first restate the lemma below.

Lemma 4.6. *Let* $c = \text{Median}\left(\left\{\text{size}(\mathcal{X}[i]) \times 2^{m[i]}\right\}_i\right)$. *Then*

$$\Pr\left[\frac{\text{Cov}_t(\mathcal{A})}{1+\varepsilon} \le c \le \text{Cov}_t(\mathcal{A})(1+\varepsilon)\right] \ge 1-\delta$$

Proof. While allowing larger constants in the expression of thresh would allow us to use the arguments of Gibbons and Tirthapura [25], we provide an alternate proof building on Chakraborty, Meel, and Vardi [13] and Meel⓻Akshay [38] that allows us to obtain better constants. We believe the proof is of independent interest as it exploits the nested properties of the sets $H[i]^{-1}_{m[i]}(0^{m[i]})$, which have shown to provide significant performance improvements in the context of model counting [1].

Let $Y = \{y_1, y_2, \ldots y_{F_0}\}$ be F_0 distinct elements covered by the input stream. For a fixed $i \in [|H|]$, let us use Bad_i to denote the event $\text{size}(\mathcal{S}[i]) \times 2^{m[i]}$ does not lie in the interval

$$I_{\text{Good}} = \left[\frac{|\text{Cov}_t(\mathcal{A})|}{1+\varepsilon}, |\text{Cov}_t(\mathcal{A})|(1+\varepsilon)\right].$$

Let $\text{Cnt}_{\langle i,j\rangle}$ denote $|Y \cap H[i]^{-1}_j(0^{m[i]})|$. For $j \in \{1,\ldots,n\}$, let $T_{i,j}$ denote the event that $\left(\text{Cnt}_{\langle i,j\rangle} \le \text{thresh} - 1\right)$, and let $L_{i,j}$ and $U_{i,j}$ denote the events $\left(\text{Cnt}_{\langle i,j\rangle} < \frac{|\text{Cov}_t(\mathcal{A})|}{(1+\varepsilon)2^j}\right)$ and $\left(\text{Cnt}_{\langle i,j\rangle} > \frac{|\text{Cov}_t(\mathcal{A})|}{2^j}(1+\frac{\varepsilon}{1+\varepsilon})\right)$, respectively.

Now, for Bad_i to happen, either $L_{i,j}$ or $U_{i,j}$ must happen alongside the event $T_{i,j} \cap \overline{T_{i,j-1}}$. Thus, we obtain

$$\Pr[\text{Bad}_i] \le \Pr\left[\bigcup_{j\in\{1,\ldots n\}}\left(\overline{T_{i,j-1}} \cap T_{i,j} \cap (L_{i,j} \cup U_{i,j})\right)\right] \quad (4)$$

Note that we only get an upper bound (and not an equality) above because the interval I_{Good} considered has upper bound $|\text{Cov}_t(\mathcal{A})|(1+\varepsilon)$, while U_i and thresh are defined using the factor $(1+\frac{\varepsilon}{1+\varepsilon}) \le 1+\varepsilon$.

Our next goal is to simplify this upper bound. Let m^* be the smallest j such that $\frac{|\text{Cov}_t(\mathcal{A})|}{2^j}(1+\varepsilon) \le \text{thresh} - 1$. Now, by substituting the chosen value of thresh and simplifying, we obtain

$$m^* = \left\lfloor \log_2|\text{Cov}_t(\mathcal{A})| - \log_2\left(4.92 \cdot \rho \cdot \left(1+\frac{1}{\varepsilon}\right)^2\right)\right\rfloor \quad (5)$$

We make use of the following simple but crucial observation:

$$\forall j \in \{1,\ldots,n\}, T_j \implies T_{j+1} \quad (6)$$

Following [38], we can now state the claim that we can upper bound Bad_i by considering only five events, namely, $T_{i,m^*-3} L_{i,m^*-2}$, L_{i,m^*-1}, L_{i,m^*} and U_{i,m^*}.

Claim 5.1. $\Pr[\text{Bad}_i] \le \Pr[T_{i,m^*-3}] + \Pr[L_{i,m^*-2}] + \Pr[L_{i,m^*-1}] + \Pr[L_{i,m^*} \cup U_{i,m^*}]$

Proof. Let $\mu_j = \frac{|\text{Cov}_t(\mathcal{A})|}{2^j}$. We make three observations, labeled O1, O2 and O3 below, which follow from the definitions of m^*, thresh and μ_j, and from the monotonicity of $\text{Cnt}_{\langle i,j\rangle}$ with respect to j for a fixed i.

O1: $\forall j \leq m^* - 3$, it is guaranteed that $\frac{|\mathrm{Cov}_t(\mathcal{A})|}{2^j(1+\varepsilon)} \geq \mathsf{thresh}$. From this it follows that (a) $T_{i,j} \cap U_{i,j} = \emptyset$ and (b) $T_{i,j} \cap L_{i,j} = T_{i,j}$. Therefore,

$$\begin{aligned}
&\bigcup_{j\in\{1,\ldots m^*-3\}} \left(\overline{T_{i,j-1}} \cap T_{i,j} \cap (L_{i,j} \cup U_{i,j})\right) \\
&\subseteq \bigcup_{j\in\{1,\ldots m^*-3\}} \left(\overline{T_{i,j-1}} \cap T_j\right) \\
&\subseteq \bigcup_{j\in\{1,\ldots m^*-3\}} T_{i,j} \subseteq T_{i,m^*-3}
\end{aligned}$$

where the last containment follows from Equation 6 . Hence,

$$\Pr\left[\bigcup_{j\in\{1,\ldots m^*-3\}} \left(\overline{T_{i,j-1}} \cap T_{i,j} \cap (L_{i,j} \cup U_{i,j})\right)\right] \leq \Pr[T_{i,m^*-3}].$$

O2: For $j \in \{m^* - 2, m^* - 1\}$, it similarly follows that $\mathsf{thresh} \leq \frac{|\mathrm{Cov}_t(\mathcal{A})|}{2^j}(1+\frac{\varepsilon}{1+\varepsilon})$, we have $T_{i,j} \cap U_{i,j} = \emptyset$. Since, $T_{i,j} \cap L_{i,j} \subseteq L_{i,j}$, we have

$$\begin{aligned}
&\Pr\left[\bigcup_{j\in\{m^*-2,m^*-1\}} \left(\overline{T_{i,j-1}} \cap T_{i,j} \cap (L_{i,j} \cup U_{i,j})\right)\right] \\
&\leq \Pr\left[L_{i,m^*-2}\right] + \Pr\left[L_{i,m^*-1}\right].
\end{aligned}$$

O3: For $j \geq m^*$, it can be shown in the same vein that $\mathsf{thresh} \geq \frac{|\mathrm{Cov}_t(\mathcal{A})|}{2^j}(1+\frac{\varepsilon}{1+\varepsilon})$, which implies that $\overline{T_{i,j}} \subseteq U_{i,j}$. Now, from Equation 6, it follows that for all j, $\overline{T_{i,j}} \subseteq \overline{T_{i,j-1}}$. This implies that

$$\begin{aligned}
&\Pr[\bigcup_{j\in\{m^*,\ldots|S|\}} \overline{T_{i,j-1}} \cap T_{i,j} \cap (L_{i,j} \cup U_{i,j})] \\
&\leq \Pr[\overline{T_{i,m^*}} \cup (\overline{T_{i,m^*-1}} \cap T_{i,m^*} \cap (L_{i,m^*} \cup U_{i,m^*}))] \\
&\leq \Pr[\overline{T_{i,m^*}} \cup L_{i,m^*} \cup U_{i,m^*}] \\
&\leq \Pr[L_{i,m^*} \cup U_{i,m^*}]
\end{aligned}$$

Using O1, O2 and O3, we get

$$\Pr[\mathsf{Bad}_i] \leq \Pr[T_{i,m^*-3}]+\Pr[L_{i,m^*-2}]+\Pr[L_{i,m^*-1}]+\Pr[L_{i,m^*}\cup U_{i,m^*}].$$

□

Claim 5.2. *The following bounds hold:*

(1) $\Pr[L_{i,m^*} \cup U_{i,m^*}] \leq \frac{1}{4.92}$
(2) $\Pr[L_{i,m^*-1}] \leq \frac{1}{10.84}$
(3) $\Pr[L_{i,m^*-2}] \leq \frac{1}{20.68}$
(4) $\Pr[T_{i,m^*-3}] \leq \frac{1}{62.5}$

Proof. Note that $\Pr[T_{i,j}] = \Pr[\mathsf{Cnt}_{\langle i,j\rangle} \leq \mathsf{thresh}]$ and $\Pr[L_{i,j}] = \Pr\left[\mathsf{Cnt}_{\langle i,j\rangle} \leq (1+\varepsilon)^{-1}\mu_j\right]$. Furthermore,
$\Pr[L_{i,j} \cup U_{i,j}] = \Pr\left[\mathsf{Cnt}_{\langle i,j\rangle} - \mu_j| \geq \frac{\varepsilon}{1+\varepsilon}\mu_j\right]$ To obtain bounds, we substitute values of m^*, thresh, μ_j, and we apply Chebyshev and Payley-Zygmund inequalities. □

Noting that $c = \mathrm{Median}\left(\left\{\mathrm{size}(\mathcal{S}[i]) \times 2^{m[i]}\right\}_i\right)$, we use the Chernoff bounds to obtain the desired bound. □