Improved Streaming Algorithm for the Klee's Measure Problem and Generalizations [*]

Mridul Nandi ⓡ
Indian Statistical Institute, Kolkata, India

N. V. Vinodchandran ⓡ
University of Nebraska, Lincoln, USA

Arijit Ghosh ⓡ
Indian Statistical Institute, Kolkata, India

Kuldeep S. Meel ⓡ
University of Toronto, Canada

Soumit Pal ⓡ
Indian Statistical Institute, Kolkata, India

Sourav Chakraborty
Indian Statistical Institute, Kolkata, India

Abstract

Estimating the size of the union of a stream of sets $S_1, S_2, \ldots, S_M$ where each set is a subset of a known universe Ω is a fundamental problem in data streaming. This problem naturally generalizes the well-studied F_0 estimation problem in the streaming literature, where each set contains a single element from the universe. We consider the general case when the sets S_i can be succinctly represented and allow efficient membership, cardinality, and sampling queries (called a Delphic family of sets). A notable example in this framework is the Klee's Measure Problem (KMP), where every set S_i is an axis-parallel rectangle in d-dimensional spaces ($\Omega = [\Delta]^d$ where $[\Delta] := \{1, \ldots, \Delta\}$ and $\Delta \in \mathbb{N}$). Recently, Meel, Chakraborty, and Vinodchandran (PODS-21, PODS-22) designed a streaming algorithm for (ϵ, δ)-estimation of the size of the union of set streams over Delphic family with space and update time complexity $O\left(\frac{\log^3|\Omega|}{\varepsilon^2} \cdot \log \frac{1}{\delta}\right)$ and $\widetilde{O}\left(\frac{\log^4|\Omega|}{\varepsilon^2} \cdot \log \frac{1}{\delta}\right)$, respectively.

This work presents a new, sampling-based algorithm for estimating the size of the union of Delphic sets that has space and update time complexity $\widetilde{O}\left(\frac{\log^2|\Omega|}{\varepsilon^2} \cdot \log \frac{1}{\delta}\right)$. This improves the space complexity bound by a $\log|\Omega|$ factor and update time complexity bound by a $\log^2|\Omega|$ factor.

A critical question is whether quadratic dependence of $\log|\Omega|$ on space and update time complexities is necessary. Specifically, can we design a streaming algorithm for estimating the size of the union of sets over Delphic family with space and complexity linear in $\log|\Omega|$ and update time $\mathsf{poly}(\log|\Omega|)$? While this appears technically challenging, we show that establishing a lower bound of $\omega(\log|\Omega|)$ with $\mathsf{poly}(\log|\Omega|)$ update time is beyond the reach of current techniques. Specifically, we show that under certain hard-to-prove computational complexity hypothesis, there is a streaming algorithm for the problem with optimal space complexity $O(\log|\Omega|)$ and update time $\mathsf{poly}(\log(|\Omega|))$. Thus, establishing a space lower bound of $\omega(\log|\Omega|)$ will lead to break-through complexity class separation results.

2012 ACM Subject Classification Theory of computation → Sketching and sampling

Keywords and phrases Sampling, Streaming, Klee's Measure Problem

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2024.26

Category APPROX

[*] The authors chose to abandon the traditional alphabetical ordering of authors in favor of a randomized ordering, marked by ⓡ. The publicly verifiable documentation of this randomization can be found at: `https://tinyurl.com/2mb96dea`

Funding *N. V. Vinodchandran*: Vinodchandran's research is partially supported by NSF grants 2130608 and 2342244.
Arijit Ghosh: Arijit Ghosh is partially supported by the Science & Engineering Research Board of the DST, India, through the MATRICS grant MTR/2023/001527.
Kuldeep S. Meel: This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019- 0004].

Acknowledgements We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) [RGPIN-2024-05956].

1 Introduction

The past three decades bear witness to significant developments in the field of data streaming. The widespread adoption of computing systems has led to the era of big data, wherein the ubiquity of sensors has allowed the collection of a large amount of data. Consequently, the data streaming model and the design of algorithms that balance time and space efficiency in this model are of significant interest to theoreticians and practitioners alike.

In this paper, we focus on one of the fundamental problems in data streaming: Given a stream of sets $S_1, S_2, \dots S_M$ where each S_i is a subset of a universe Ω, output an (ε, δ)-estimate (see the Definition 5) of the size of the union of the sets, that is, $|\bigcup_i S_i|$. Note that when sets are singletons, the problem boils down to the estimation of the zeroth frequency moment (F_0) of the stream of items and is well-studied in streaming literature. In particular, a long line of work culminated in the development of algorithms for F_0-estimation with optimal space complexity $O(\log|\Omega| + \frac{1}{\varepsilon^2})$ and $O(1)$ update time complexity [16] (for a constant error probability δ). In this work, we will focus on the Delphic family of sets which is a general framework for computational problems over abstract sets.

▶ **Definition 1** (Delphic family). *Let Ω be a discrete universe. A set $S \subseteq \Omega$ belongs to a Delphic family*[1] *if the following queries can be done in $O(\log|\Omega|)$ time: (1)* Membership: *Given any $x \in \Omega$ check if $x \in S$, (2)* Cardinality*: Determine the size of S, that is $|S|$, (3)* Sampling: *Draw a uniform random sample from S.*

The notion of the Delphic family is general enough to capture several well-known problems, such as Klee's Measure Problem (KMP) [23, 25], test coverage estimation, and DNF counting. For example, the streaming version of Klee's Measure Problem (KMP) refers to the case where the sets in a stream are represented by an axis-parallel rectangle in $[\Delta]^d$ where Δ is a natural number. KMP is a naturally occurring and fundamental topic that has been extensively researched in computational geometry [4, 6, 8, 10, 9, 12, 14, 17, 21]. While *Delphic Sets* was coined in [19], the notion has been implicit in prior work stretching to the early 1980's. We are interested in designing streaming algorithms for estimating the size of the union of sets over a Delphic family. We also call this problem F_0 estimation problem over Delphic sets.

▶ **Problem 2.** *Given a stream $\mathcal{S} = \langle S_1, S_2, \dots, S_M\rangle$ wherein each $S_i \subseteq \Omega$ belongs to a Delphic family, and $0 < \varepsilon < 1$ output an $(\varepsilon, 1/3)$*[2] *approximation of* $\mathsf{F}_0(\mathcal{S}) := \left|\bigcup_{i=1}^{M} S_i\right|$.

[1] As observed in [18], every Delphic set can be represented by a circuit of size $O(\log|\Omega| \log\log|\Omega|)$.
[2] c is $(\varepsilon, 1/3)$ approximation of $\mathsf{F}_0(\mathcal{S})$ if $\Pr[\mathsf{F}_0(\mathcal{S})(1-\varepsilon) \leqslant c \leqslant \mathsf{F}_0(\mathcal{S})(1+\varepsilon)] \geqslant 2/3$. Note that the choice of 1/3 is arbitrary. The probability 1/3 can be boosted to an arbitrary δ by using standard boosting technique.

Table 1 In the table $n := \log|\Omega|$, $m := \log M$ and $c := O(1)$. The Big-Oh notation (O) in the above comparison table shows only dependency on n and m and hides the $\mathsf{poly}(1/\varepsilon, 1/\delta)$ factors on ε and δ.

Comparison of Complexity Bounds with Previous Works				
Algorithm	Technique Type	Set Type	Space Complexity	Update Time
[22]	Hashing based	KMP $d = 1$	$O(\varepsilon^{-2} \cdot n)$	$O(n)$
[25]	Hashing based	KMP $d \geqslant 2$	$O(\varepsilon^{-1} \cdot d \cdot n)^c$	$O(n^d)$
[19]	Sampling based	Delphic	$O(\varepsilon^2 \cdot m \cdot n)$	$O(\varepsilon^2 \cdot \log m \cdot m^2 \cdot n)$
[18]	Sampling based	Delphic	$O(\varepsilon^{-2} \cdot n^3)$	$O(\varepsilon^{-2} \cdot n^4)$
This Paper	Sampling based	Delphic	$O(\varepsilon^{-2} \cdot n^2)$	$O(\varepsilon^{-2} \cdot n^2)$

The design goal is to optimize the algorithm's update time and space complexity, wherein the update time complexity refers to the amount of time spent processing an item (a set in our case) of the stream.

1.1 Prior Work and Technical Challenges

Since the work of Alon, Matias, and Szegedy [1], streaming algorithms gained considerable interest from algorithm design community. However, the problem of estimating distinct elements in a stream of items has been investigated prior to the work of [1]. In particular, the seminal work of Flajolet and Martin [11] pioneered a sketching-based framework for streaming algorithms for the case wherein every element of the stream is a singleton. The sketching-based techniques crucially rely on the use of pairwise independent hash functions. In the early 2000s, the sketching-based approach emerged as a principal technical tool in the design of streaming algorithms. Particularly for F_0 estimation of single-item streams, a series of hash-function-based algorithms led to the development of both space-efficient and update-time optimal algorithms [16, 5].

F_0 estimation over set streams has garnered interest from researchers due to the natural extension from singletons to sets. Notably, Pavan and Tirthapura [22] and Sun and Poon [24] explored range-efficient F_0 estimation, which addresses a specific case of the KMP in one dimension. For the broader KMP, Tirthapura and Woodruff [25] developed an algorithm with optimal space complexity. However, the update time for their algorithm was $O(|\Omega|)$, which is exponential in terms of set representation. Subsequently, Pavan, Vinodchandran, Bhattacharyya, and Meel [23] proposed an alternative technique, yet it similarly faced an update time complexity of $O(|\Omega|)$.

All the above-mentioned algorithms employed hash function based approaches and failed to yield an algorithm with update time complexity polynomial in representation of sets for the general case of Delphic sets. The primary technical barrier to such approaches arises from the fact that sketching-based techniques crucially rely on checking whether for a function h, randomly chosen from a pairwise independent hash family, there exists an element $x \in S_i$ such that $h(x) = 0$. Nevertheless, whether a pairwise independent hash family exists that supports such checking in time $\text{poly}(\log|\Omega|)$ is unknown.

In [19], the authors introduced a sampling-based technique for F_0 estimation that eschews traditional hash functions. Their method involves maintaining a bucket $\mathcal{X}$, where each stream element is selected independently with probability p. To handle element repetitions, new elements from a set S_i replace those in $\mathcal{X}$, with all elements in S_i sampled independently at the same probability p. To manage the bucket's size, elements are discarded with a probability of 1/2 once a threshold based on ε^{-1} is reached, halving p simultaneously. This approach yields a space complexity of $\tilde{O}\left(\frac{\log|\Omega|\cdot\log M}{\varepsilon^2}\right)$, with a logarithmic dependence on the stream size M. In a subsequent work [18], the authors modified this approach by allowing p to vary, not just decrease. Each tuple in $\mathcal{X}$ then included the element and its sampling probability at arrival. This adjustment removed the dependency on M, leading to space complexity of $\tilde{O}\left(\frac{\log^3|\Omega|}{\varepsilon^2}\right)$ and update time complexity of $\tilde{O}\left(\frac{\log^4|\Omega|}{\varepsilon^2}\right)$. This contrasts with the optimal algorithm for singleton sets S_i by Kane, Nelson, and Woodruff, which has a space complexity of $O(\log|\Omega| + \frac{1}{\varepsilon^2})$.

The above-mentioned line of research leads to the following significant open question:

Can we design algorithm for F_0 estimation over Delphic sets with space complexity $O(\log|\Omega|)$ and update-time complexity $\mathsf{poly}(\log|\Omega|)$ (ignoring dependency on ε and δ)?

Remark: We note that if we have no restriction on the update time complexity, then the problem over Delphic sets reduces to F_0 estimation of singleton streams: when a set S_i arrives, we can cycle through all elements in the universe Ω and use membership testing only to stream elements of S_i. This leads to an algorithm with optimal space complexity of $O\left(\log(|\Omega|) + \varepsilon^{-2}\right)$ but with update time that depends linearly on $|\Omega|$. An asymptotic lower bound of $\Omega\left(\log(|\Omega|) + \varepsilon^{-2}\right)$ is known for the space complexity [15, 1].

It is worth observing that if we were to follow the approach suggested in [19, 18], then ensuring that the value of p at least $\frac{1}{\varepsilon^2\cdot\mathsf{F}_0(\mathcal{S})}$ with sufficiently high probability does entail the space complexity of $\tilde{O}(\frac{\log^3|\Omega|}{\varepsilon^2})$. Therefore, an improvement in space complexity must require a new approach.

1.2 Our Results

As our first contribution, we report progress towards the above question. In particular, we establish the following:

▶ **Theorem 3** (Main Theorem). *There is a streaming algorithm that given a stream $\mathcal{S} = \langle S_1, S_2, \ldots, S_M\rangle$ wherein each S_i belongs to Delphic family, and $0 < \varepsilon < 1$ outputs an $(\varepsilon, 1/3)$-estimation of $\left|\bigcup_{i=1}^{M} S_i\right|$ with space complexity $O\left(\frac{\log^2|\Omega|}{\varepsilon^2}\cdot\log\frac{1}{\varepsilon}\right)$ and update time complexity $O\left(\frac{\log^2|\Omega|}{\varepsilon^2}\cdot\log^3\frac{1}{\varepsilon}\right)$.*

The above theorem improves the space complexity by a factor of $\log|\Omega|$ and the update time complexity by a factor of $\log^2|\Omega|$. We note that for special cases such as KMP, we can bring the update time complexity factor of $\log^3\frac{1}{\varepsilon}$ to $\log\frac{1}{\varepsilon}$, resulting in update time complexity of $O\left(\frac{d^2\log^2\Delta}{\varepsilon^2}\cdot\log\frac{1}{\varepsilon}\right)$ where $\Omega = [\Delta]^d$.

Theorem 3 leads to the natural question of whether further improvement is possible towards solving the open question of designing a streaming algorithm for F_0 estimation over Delphic sets with space complexity $O(\log|\Omega|)$ (ignoring the dependence on ε and δ) and update time complexity significantly smaller than $|\Omega|$. This seems hard and will require new techniques: since storing a single element takes $O(\log|\Omega|)$ space implies that one can only store a constant number of elements at any point of time. This appears to be a major technical restriction for sampling-based approaches like our algorithm in Theorem 3 and we even conjecture that $\omega(\log|\Omega|)$ space is required if we restrict the update time to be $\mathsf{poly}(\log|\Omega|)$. Unfortunately, establishing such a lower bound has a major computational complexity bottleneck. In particular, we show that establishing a lower bound on space other than the lower bound known for computing F_0 for singleton streams will lead to major separation result in computational complexity.

NTISP(poly,LINSPACE) is the class of language accepted by non-deterministic Turning machines in polynomial time and linear space simultaneously. Likely, DTISP(poly,LINSPACE) is the class of languages accepted by deterministic Turing machines in polynomial time and linear space simultaneously. Whether or not NTISP(poly, LINSPACE) = DTISP(poly, LINSPACE) is an open problem in complexity theory. If NTISP(poly,LINSPACE) = DTISP(poly,LINSPACE) then P=NP. The other implication is yet to be discovered. However, separating NTISP(poly, LINSPACE) from DTISP(poly,LINSPACE) is a hard open question. The best-known lower bound for time-space complexity classes is that SAT (which is in NTISP(poly,LINSPACE)) cannot be solved in DTISP($n^{1.8}, o(n)$) [26]. Any improvement on this will be a major result in complexity theory. We show the following:

▶ **Theorem 4.** *There is a streaming algorithm,* DelphicWithNP*, that given, a stream $\mathcal{S} = \langle S_1, \cdots, S_M \rangle$, where each set $S_i \subseteq \Omega$ is a member of a Delphic family, $0 < \varepsilon$, and an oracle access to a language belonging to* NTISP(poly,LINSPACE)*, computes a $(\varepsilon, 1/3)$-approximation of $\mathsf{F}_0(\mathcal{S})$. Moreover,* DelphicWithNP *has the following properties:*
(1) *it takes $O(\log|\Omega| \cdot \varepsilon^{-2})$ space and $\mathrm{poly}(\log|\Omega|, \frac{1}{\varepsilon})$ update time,*
(2) *the queries it makes to the oracle are of size $O(\log|\Omega| \cdot \varepsilon^{-2})$.*

Thus, if NTISP(poly,LINSPACE) = DTISP(poly,LINSPACE)*, there is an algorithm (without oracle calls) for $(\varepsilon, 1/3)$-estimation of F_0 of a set stream over Delphic family with $O(\log|\Omega| \cdot \varepsilon^{-2})$ space and $\mathrm{poly}(\log|\Omega|, 1/\varepsilon)$ update time.*

Thus, for constant ε, we get a space optimal algorithm for estimating F_0 over Delphic set streams as $\log|\Omega|$ space is required even in the singleton case. In fact, the above theorem is true for a very relaxed version of Delphic sets where we only require one of the conditions for membership in Delphic sets to be decided in time linear in the representation of the set. Theorem 4 implies that establishing a space lower bound of $\omega(\log|\Omega|)$ for estimating $\mathsf{F}_0(\mathcal{S})$ of Delphic set streams with $\mathsf{poly}(\log|\Omega|)$ will lead to proving NTISP(poly,LINSPACE) $\neq$ DTISP(poly,LINSPACE), thus resolving a significant lower bound in complexity theory.

Our work leaves a tantalizing open question:

What is the optimal space complexity for F_0 estimation of set streams over Delphic family with $\mathrm{poly}(\log|\Omega|)$ update time complexity?

1.3 Technical Overview

Key Ideas for Theorem 3

The high-level idea is to maintain a bucket for every level $k \in \{1, \ldots, \log|\Omega|\}$ such that the bucket, $\mathcal{X}^{(k)}$, at level k consists of elements from $\bigcup_i S_i$ that were independently selected with a probability of 2^{-k}. To handle repetitions, i.e., when a new set S_i arrives, we remove all the elements of $\mathcal{X}^{(k)}$ that are present in S_i. This process ensures that the event of an element s being in $\mathcal{X}^{(k)}$ at the end of the stream depends only on whether s was independently chosen from the set S_i, where S_i is the last set containing s, and there is no S_j with $j > i$ such that $s \in S_j$.

To bound the space complexity, we establish a threshold, denoted by thresh, limiting the maximum size of the bucket. Whenever $|\mathcal{X}^{(k)}| = \mathsf{thresh}$, no additional elements can be added to $\mathcal{X}^{(k)}$ (i.e., we must wait until some elements are removed from $\mathcal{X}^{(k)}$). We set $\mathsf{thresh} := O\left(\frac{1}{\varepsilon^2}\right)$. Note that storing any element requires $O(\log|\Omega|)$ space, and thus storing $\mathsf{thresh}\log|\Omega|$ elements necessitates $O\left(\frac{\log^2|\Omega|}{\varepsilon^2}\right)$ space.

The key technical insight from [19] is that if $\mathsf{thresh} = O\left(\frac{\log M}{\varepsilon^2}\right)$, it can be demonstrated that, with sufficiently high probability and for $k > \log(\mathsf{F}_0(\mathcal{S}))$, $\mathcal{X}^{(k)}$ would not be full (i.e., $|\mathcal{X}^{(k)}| < \mathsf{thresh}$) at all times. Furthermore, in [18], it was observed that $\mathsf{thresh} = O\left(\frac{\log|\Omega|}{\varepsilon^2}\right)$, then it can be demonstrated that, with sufficiently high probability and for $k > \log(\mathsf{F}_0(\mathcal{S}))$, $\mathcal{X}^{(k)}$ would not be full (i.e., $|\mathcal{X}^{(k)}| < \mathsf{thresh}$) when an element $s \in \bigcup_i S_i$ appears for the last time in a stream.

The major technical advancement in our analysis hinges on a crucial observation: while it is plausible for $\mathcal{X}^{(k)}$ to be full when an element s makes its final appearance in the stream, the resulting relative error of our estimator $|\mathcal{X}^{(k)}| \cdot 2^k$ remains manageable even when k is approximately logarithmic in the order of magnitude of the size of the stream, that is $k \sim \log(\mathsf{F}_0(\mathcal{S}))$. The ensuing technical analysis is complex due to its dependence on a meticulous formulation of significant events and the construction of a sum of products of random variables. We introduce two sets of random variables, denoted as $X_{i,r}^{(k)}$ and $Y_{i,r}^{(k)}$. The random variable $X_{i,r}^{(k)}$ indicates whether an element is sampled from the j-th set S_j in the stream. The random variables $Y_{i,r}^{(k)}$ indicate whether a set of sampled elements is included in the buckets. We then define a series of random variables $Z_r^{(k)}$, dependent on the aforementioned random variables $X_{i,r}^{(k)}$ and $Y_{i,r}^{(k)}$. To leverage concentration inequalities effectively, we maintain a collection of $O\left(1/\varepsilon^2\right)$ buckets at each level k. Additionally, we set the size of each bucket to be $O\left(\log(1/\varepsilon)\right)$. At the end of the stream, for each level k, we calculate the average number of elements in the buckets at that level, denoted as $\overline{Z}^{(k)}$.

Finally, it is important to note that our technical analysis only ensures that the relative error is small for $k \approx \log(\mathsf{F}_0(\mathcal{S}))$. Since $\mathsf{F}_0(\mathcal{S})$ is unknown a priori, we must identify a method to choose an optimal k^* for returning the final estimate $\overline{Z}^{(k^*)} \cdot 2^{k^*}$. A key observation is that $\overline{Z}^{(k)}$ is less than 1 for $k >> \log(\mathsf{F}_0(\mathcal{S}))$. Therefore, finding the largest k for which $\overline{Z}^{(k)}$ exceeds 1 should suffice.

Key Ideas for Theorem 4

The main idea is to adapt one of the standard hashing-based algorithms (e.g., Gibbons and Tirthapura's algorithm) for F_0 estimation for singleton streams that takes $O\left(\log|\Omega| \cdot \frac{1}{\varepsilon^2}\right)$ space. The algorithm starts with picking a hash function h from a pairwise independent family and keeps a bucket $\mathcal{X}$ which is initially set to empty and has a capacity of $O\left(\frac{1}{\varepsilon^2}\right)$. The computational challenge is to implement the update step. For this, when a new set S comes, we need to add all elements $x \in S$ so that the first m bits of $h(x)$ are all 0s for a variable m that the algorithm keeps. If the addition of these elements makes the bucket $\mathcal{X}$ overflow, m is incremented and deletes all elements $\mathcal{X}$ with $m+1$st bit of hash value is non-zero. The main computational challenge is to implement this step. However, we show how this step can be implemented by making linear size queries to disjoint union of two languages in NTISP(poly,LINSPACE). Thus if NTISP(poly,LINSPACE) = DTISP(poly,LINSPACE), then queries to this language can be simulated in space $O(\log|\Omega| \cdot \frac{1}{\varepsilon^2})$ and update time $\mathsf{poly}(\log|\Omega|, \varepsilon^{-1})$.

1.4 Paper Organization

The remainder of this paper is structured as follows. Section 2 introduces key notations and fundamental concepts. In Section 3, we present our primary Algorithm 2 for Delphic sets with WOR sampling, along with its correctness analysis. Subsequently, in subsection 3.2, we present the algorithm for the general Delphic set. Section 4 contains the proof of Theorem 4. The appendix is divided as follows: Section A presents the concentration bounds used in the correctness analysis; Section B provides the missing proofs for Proposition 15; Section C details the algorithm for KMP along with its correctness analysis; and finally, Section D offers some basic probability results.

2 Preliminaries

2.1 F_0-Estimator and Klee's Measure Problem

For a stream of sets, $\mathcal{S} := \langle S_1, S_2, \ldots, S_M \rangle$ where each set in the stream is a subset of Ω, recall that we denote by $\mathsf{F}_0(\mathcal{S})$ the size of the union of the sets: $\mathsf{F}_0(\mathcal{S}) = \left|\bigcup_{j=1}^{M} S_j\right|$

▶ **Definition 5.** *A random variable X is an (ε, δ)-approximation of c if* $\mathsf{Pr}[(1-\varepsilon)c \leqslant X \leqslant (1+\varepsilon)c] \geqslant (1-\delta)$. *We also simply write the event $(1-\varepsilon)c \leqslant X \leqslant (1+\varepsilon)c$ as $X = (1 \pm \varepsilon)c$ (or $X \neq (1 \pm \varepsilon)c$ to denote the complement event).*

Finally an F_0-estimator is a streaming algorithm that on input $\varepsilon, \delta \in (0,1)$ and access to a stream of set $\mathcal{S} := \langle S_1, S_2, \ldots, S_M \rangle$ outputs an (ε, δ) approximation of $\mathsf{F}_0(\mathcal{S})$.

There are two main complexity measures that we are interested about an F_0-estimator - space complexity and update time complexity. The space complexity is the amount of work space needed by the algorithm. The update time complexity is the amount of time spent by the algorithm for processing a single set in the stream. The goal is to design an F_0-estimator while trying to minimize both the space complexity and the update time complexity of the algorithm.

The notion of Delphic sets captures several well-known problems, such as Klee's Measure Problem, which we define below.

Let Δ be a natural number, consider the following set $[\Delta] = \{1, 2, \ldots, \Delta\}$. A d-dimensional axis-aligned rectangle $\mathbf{r}$ over $[\Delta]^d$ is a subset of $[\Delta]^d$, succinctly represented by the tuple $(a_1, b_1, \cdots a_d, b_d)$, and contains all the tuples $\{(x_1, \ldots, x_d)\}$ where $a_i \leqslant x_i \leqslant b_i$ and $x_i \in [\Delta]$. Formally, we write $\mathbf{r} = \{(x_1, x_2, \ldots, x_d) : a_i \leqslant x_i \leqslant b_i, x_i \in [\Delta] \textit{ for all } i \in [d]\}$

▶ **Definition 6** (Klee's Measure Problem (KMP) in Streaming Setting)**.** *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, compute a (ε, δ)-approximation of $\left|\bigcup_{i=1}^{M} \boldsymbol{r}_i\right|$.*

Note that KMP is a special case of Problem 2 since set of d-dimensional axis-aligned rectangles over $[\Delta]^d$ forms a Delphic family.

2.2 Notations

For any positive integer k, we write $[k] = \{1, 2, \ldots, k\}$. In the rest of this paper, we will assume that $\mathcal{S} := \langle S_1, S_2, \ldots, S_M \rangle$ is a stream of sets, where each set is a subset of universe Ω, i.e., $S_j \subseteq \Omega$ for all $j \in [M]$. We denote by $s_j := |S_1| + \cdots + |S_j|$ for all $j \in [M]$ and $s_0 := 0$. We write $s := s_M$ to denote the total number of elements counting with repetition that appeared in the stream $\mathcal{S}$. For $j \in [M]$, let $\mathcal{I}_j = \{s_{j-1}+1, s_{j-1}+2, \ldots, s_j\}$, the set $\mathcal{I}_j$ denotes the indices of elements in the set S_j. Note that $\mathcal{I}_j$'s are disjoint and $\bigcup\limits_{j=1}^{M} \mathcal{I}_j = [s]$. Clearly, for every $i \in [s]$, we can assign a *streaming-index* $j := j(i)$ (which is unique) such that $i \in \mathcal{I}_j$.

Although, total order of the elements on Ω is not needed for our algorithms, we will assume an ordering of the elements of Ω; this will help us in the presentation of the correctness proofs of the algorithm. We can open the set stream into element streams and write a sequence $\langle x_1, \ldots, x_s \rangle$, where $S_j = \{x_i : i \in \mathcal{I}_j\}$, $\quad x_{s_{j-1}+1} < \cdots < x_{s_j} \forall j \in [M]$. For any $k \in [|S_j|]$ we will denote by $S_j[k]$ to the element $x_{s_{j-1}+k}$.

At any point, say after the sets $S_1, \ldots, S_j$ have arrived in the stream and for any element x in the stream $\langle x_1, \ldots, x_{s_j} \rangle$, that is, the stream we have seen till now, we will be interested in the last time the element appeared in the stream. The set of indices in $[s_{j-1}]$ that contains the last appearance of any element is called the set of *final indices with respect to* j*th set* S_j and is denoted by $\mathcal{F}_j$.

More formally, let $j \in [M]$ be a stream index. We call $i \leqslant s_{j-1}$ a *final index with respect to* j*th set* S_j if $x_i \notin \{x_{i+1}, \ldots, x_{s_j}\}$. Let

$$\mathcal{F}_j = \{i \leqslant s_{j-1} : i \text{ is a final indices w.r.t. } j\text{th set } S_j\}.$$

When $j = 1$, the set $\mathcal{F}_1 = \varnothing$.

Now we make the following simple but valuable observations:

▶ **Observation 7.**
1. *For all $j \in [M]$, (i) $\mathcal{F}_j$ and $\mathcal{I}_j$ are disjoint i.e., $\mathcal{F}_j \cap \mathcal{I}_j = \varnothing$ and (ii) $\{x_i : i \in \mathcal{F}_j\}$ is disjoint from S_j.*
2. *$\{x_i : i \in \mathcal{F}_j\} \sqcup S_j \subseteq S$. The equality occurs for $j = M$, i.e., $\{x_i : i \in \mathcal{F}\} = S$ where $\mathcal{F} = \mathcal{F}_M \sqcup \mathcal{I}_M$. Note, $|\mathcal{F}| = \mathsf{F}_0(\mathcal{S})$.*
3. *For all $j \in [M-1]$, $\mathcal{F}_{j+1} \subseteq \mathcal{F}_j \sqcup \mathcal{I}_j$.*

2.3 Delphic Family with WOR Sampling

The definition of Delphic Sets (Definition 1) allows one to draw a uniform random sample from a Delphic Set S. However, if one needs to uniformly draw a set of k distinct samples from S, the only option is to draw independent samples from S until one gets k distinct samples. One can use the coupon collector theorem to ensure that with high probability, one has to draw at most $O(k \log k)$ samples. Nevertheless, this "high probability" statement may not be sufficient if one has to repeat this process multiple times - a slightly more involved calculation may be necessary.

One way of dealing with this issue is assuming one is allowed to draw samples "without replacement (WOR)". Formally, we define a Delphic family with WOR sampling as follows:

▶ **Definition 8** (Delphic family with WOR sampling). *A set $S \subseteq \Omega$ belongs to the Delphic family with WOR sampling if the following queries can be done:*
(1) *know the size of the set S, in $O(\log |\Omega|)$ time*
(2) *given any x check if $x \in S$, in $O(\log |\Omega|)$ time*
(3) *for any $k \leqslant |S|$, we can draw in $O(k \log |\Omega|)$ time a uniformly random subset S of size k.*

Note that, in contrast to Definition 8, the original Delphic family (Definition 1) can be called "Delphic family with WR sampling", where WR stands for "with replacement".

2.4 Random Processes and Distributions

We will be drawing samples according to different distributions. All the distributions are standard in the literature. The following are the distributions we will be using in this paper.

▶ **Definition 9** (Binomial Distribution). *Given any $n \in \mathbb{N}$ and any $p \in [0,1]$ the Binomial Distribution over the set of integers $\{0,1,2,\ldots,n\}$ is denoted as $\mathsf{Bin}(n,p)$ where probability of a number $0 \leqslant k \leqslant n$ is $\binom{n}{k}p^k(1-p)^{n-k}$.*

▶ **Definition 10** (Bernoulli Process). *Given any finite set S and any $p \in [0,1]$, a Bernoulli sample of probability p for the set S is denoted as $\mathsf{Ber}(S,p)$ such that every element $x \in S$ is picked with probability p. If X denotes the sampled subset of S with probability p then for any set $T \subseteq S$, $Pr(X = T) = p^{|T|}(1-p)^{|S|-|T|}$.*

By abuse of notation, we will use $\mathsf{Ber}(p)$ to denote the distribution over $\{0,1\}$ where the probability of 1 is p. So, the Bernoulli process is nothing but independent copies of the Bernoulli distribution where 1 (or 0) represents that the element is picked (or not picked respectively).

Clearly, if $X \sim \mathsf{Ber}(S,p)$ then $|X| \sim \mathsf{Bin}(|S|,p)$. We now describe how to sample the Bernoulli process. We first sample $d \sim \mathsf{Bin}(|S|,p)$, then we sample d many elements, denoted as T, in a without replacement (WOR) manner. Then, $T \sim \mathsf{Ber}(S,p)$. One can sample WOR through with replacement (WR) sampling (this is well-known as the coupon collection problem). We have the following lemma using the coupon collector theorem (see Appendix D).

▶ **Lemma 11.** $\mathsf{WRSample}(S,r,t)$ *(Algorithm 1) outputs a subset L of the set S with the following guarantee*

- *With probability $\geqslant \left(1 - r^{\frac{t}{r\log r}+1}\right)$ the algorithm outputs a set of size r distinct samples, each drawn uniformly from the set S.*
- *With the remaining probability at most $r^{\frac{t}{r\log r}+1}$ the algorithm outputs an empty set.*
- *The maximum number of samples drawn from S is t.*

Algorithm 1 $\mathsf{WRSample}(S,r,t)$.

```
1: Input Set $S$ $t, r \in \mathbb{N}$
2: Initialize $L = \varnothing$;
3: for $1 \leqslant i \leqslant t$ do
4:     if $|L| < r$ then
5:         Draw a random sample $x$ from $S$ (with replacement)
6:         if $x \notin L$ then $L = L \cup \{x\}$
7: if $|L| \neq r$ then $L = \varnothing$
8: Output $L$
```

3 F_0-Estimator for Delphic sets

In this section, we prove Theorem 3. We prove it in two steps. In Section 3.1, we present Algorithm 2 and prove that it is an F_0-estimator for Delphic Sets with WOR sampling. Then, in Section 3.2, we show how the Algorithm 2 can be modified to obtain an F_0-estimator for general Delphic Sets and hence prove Theorem 3.

3.1 Handling Delphic Sets with WOR sampling

In this section, we will prove the following theorem:

▶ **Theorem 12.** *If $\mathcal{S}$ is a stream of Delphic Sets with WOR sampling, then the Algorithm 2 is an $(\varepsilon, 1/3)$-F_0-estimator. Also, the space and update time complexity of Algorithm 2 is* $O\left(\frac{\log^2 |\Omega|}{\varepsilon^2} \cdot \log \varepsilon^{-1}\right)$.

Before we present the proof of the correctness of the algorithm and the analysis of its complexities (in Section 3.1.1 and Section 3.1.2 respectively), we will give a brief description of the algorithm.

Description of the Algorithm

Let $0 < \varepsilon \leqslant 1/2$, and we set $\mathsf{thresh} := \max\left\{18, \lceil \log \frac{2}{\varepsilon} \rceil\right\}$ and $\mathsf{Reps} = 90/\varepsilon^2$. We will now give an efficient F_0-estimator for Delphic sets. At each level k, where $k \in \{1, \ldots, \log |\Omega|\}$, there is a collection of Reps many buckets $\mathcal{X}_r^{(k)}$ with $r \in \{1, \ldots, \mathsf{Reps}\}$. Each bucket $\mathcal{X}_r^{(k)}$ can contain at most thresh many samples from the universe $[n]$. At the beginning of the algorithm, all the Reps many buckets in each level k is an empty set. Let the input stream S of Delphic sets be the following: $\mathcal{S} := \langle S_1, \ldots, S_M \rangle$, where $S_j \subseteq \Omega$ for all $j \in [M]$. Note that we want to estimate the size of the set $S = \bigcup_{j=1}^{M} S_j$. Suppose that our algorithm has already processed the $S_1, \ldots, S_i$ from the stream and the buckets $\mathcal{X}_r^{(k)}$ in each level k contains a sample of elements from the set S. Now, our algorithm receives a new set S_{i+1} over the data stream. Our algorithm will perform the following steps to process the new set S_{i+1}:

Step 1: For each buckets $\mathcal{X}_r^{(k)}$, where $k \in [\log |\Omega|]$ and $r \in [\mathsf{Reps}]$, we will first begin by removing all the elements in $\mathcal{X}_r^{(k)}$ that are also present in the new set S_{i+1}, see **lines** 6 to 9 from the Algorithm 2. Observe that the time complexity for this step will be $O\left(\mathsf{Reps}\ \mathsf{thresh} \log^2 |\Omega|\right)$

Step 2: Ideally, we would like to add an element from S_{i+1} independently with probability 2^{-k} to each bucket $\mathcal{X}_r^{(k)}$, but this may not be possible as the size of each bucket is thresh. To work around this problem, for each $k \in [\log |\Omega|]$ and $r \in [\mathsf{Reps}]$, we will first sample $d_r^{(k)} \sim \mathsf{Bin}(|S_{i+1}|, 2^{-k})$. By slight abuse of notation, we will denote by $|\mathcal{X}_r^{(k)}|$ the current size of the bucket $\mathcal{X}_r^{(k)}$. If $d_r^{(k)} \geqslant \mathsf{thresh} - |\mathcal{X}_r^{(k)}|$ then we will keep $\mathcal{X}_r^{(k)}$ unchanged. Otherwise, as S_{i+1} is a Delphic set with WOR sampling, we will sample without replacement $d_r^{(k)}$ many samples from S_{i+1} and add them to $\mathcal{X}_r^{(k)}$. See **lines** 10 to 13 from the Algorithm 2.

Once we have processed the whole stream $\mathcal{S} = \langle S_1, \ldots, S_M \rangle$, for all $k \in [\log |\Omega|]$, we will calculate the average size $\overline{Z}^{(k)}$ of the buckets in each level k, (See **lines** 14 to 15 from Algorithm 2) that is,

$$\overline{Z}^{(k)} = \frac{1}{\mathsf{Reps}} \sum_{r=1}^{\mathsf{Reps}} |\mathcal{X}_r^{(k)}|$$

Once all the averages have been computed for each level $k \in [\log |\Omega|]$ the algorithm computes the maximum k, let us call it k^* for which $\overline{Z}^{(k^*)} \geqslant 1$. Finally, the algorithm outputs $2^{k^*} \cdot \overline{Z}^{k^*}$. See **lines** 16 to 17 from Algorithm 2.

3.1.1 Correctness of Algorithm 2 for Delphic Sets with WOR sampling

If we replace the **lines** 10-13 in Algorithm 2 by the following lines, then it is easy to see that it functionally behaves identically.

```
10: Draw $d_r^{(k)}$ from $\mathsf{Bin}(|S_j|, 1/2^k)$
```

Algorithm 2 [F_0-Estimator for Delphic Sets with WOR sampling.]

```
1:  **Input** Stream $= \langle S_1, S_2, \ldots, S_M \rangle$, $\varepsilon$, $\delta$
2:  **Initialize**
3:  $p \leftarrow 1$; thresh $\leftarrow \max\{\lceil \log 2/\varepsilon \rceil,\ 18\}$; Reps $\leftarrow \lceil 90/\varepsilon^2 \rceil$;
4:  **for** $(1 \leqslant k \leqslant \log|\Omega|)$ and $(1 \leqslant r \leqslant \text{Reps})$ **do**
5:      $\mathcal{X}_r^{(k)} \leftarrow \varnothing$;

    STREAMING PHASE:
6:  **for** $j = 1$ to $M$ **do**.
7:      **for** $(1 \leqslant k \leqslant \log|\Omega|)$ and $(1 \leqslant r \leqslant \text{Reps})$.   **do**
8:          **for** all $s \in \mathcal{X}_r^{(k)}$ **do**
9:              **if** $s \in S_i$ **then** remove $s$ from $\mathcal{X}_r^{(k)}$
10:         Draw $d_r^{(k)}$ from $\mathsf{Bin}(|S_j|, 1/2^k)$
11:         **if** $|\mathcal{X}_r^{(k)}| + d_r^{(k)} \leqslant$ thresh **then**
12:             $\mathcal{L}_r^{(k)}$ is a set of $d_r^{(k)}$ samples drawn from $S_j$ WOR
13:             $\mathcal{X}_r^{(k)} = \mathcal{X}_r^{(k)} \cup \mathcal{L}_r^{(k)}$

    POST-STREAMING PHASE:
14: **for** $1 \leqslant k \leqslant \log|\Omega|$ **do**.
15:     $\overline{Z}^{(k)} = \frac{1}{\text{Reps}} \sum_{r=1}^{\text{Reps}} |\mathcal{X}_r^{(k)}|$
16: $k^* = \max\{k \in [\log n] : \overline{Z}^{(k)} \geqslant 1\}$
17: **Output** $\overline{Z}^{(k^*)} \cdot 2^{k^*}$
```

```
11: $\mathcal{L}_r^{(k)}$ is a set of $d_r^{(k)}$ samples drawn $S_j$ WOR
12: **if** $|\mathcal{X}_r^{(k)}| + d_r^{(k)} \leqslant$ thresh **then**
13:     $\mathcal{X}_r^{(k)} = \mathcal{X}_r^{(k)} \cup \mathcal{L}_r^{(k)}$
```

Note that the differences between the above lines and the lines in the Algorithm 2 is that in the Algorithm 2 we do not sample the set $\mathcal{L}_r^{(k)}$ unless we are sure that we will use the set $\mathcal{L}_r^{(k)}$, that is the condition $|\mathcal{X}_r^{(k)}| + d_r^{(k)} \leqslant \text{thresh}$ is satisfied. We do this in the actual algorithm (more like a "lazy sampling") to have better control on the worst-case update time complexity. But, for the presentation of the proof of correctness of the theorem it is useful to use the above three lines (instead of **lines** 10-13 in Algorithm 2) where we first sample $\mathcal{L}_r^{(k)}$ and then decide whether to use it.

Let us consider the jth round of the streaming phase of the algorithm. That is when the set S_j arrives in the stream. By the construction of the set $\mathcal{L}_r^{(k)}$ it follows that all elements of S_j's are chosen in $\mathcal{L}_r^{(k)}$ independently with probability 2^{-k}. For the sake of presentation, let $\mathcal{L}_{j,r}^{(k)}$ denote the set $\mathcal{L}_r^{(k)}$ during the jth round of the streaming phase of the algorithm [3]. Recall that the elements in the set S_j, that is, the elements $x_{s_{j-1}+1}, \ldots, x_{s_j}$ and for any $i \in [s_j]$ we denote by $j(i)$ the stream index such that $i \in \{s_{j(i)-1} + 1, \ldots, s_{j(i)}\}$ (refer to the notations in the Section 2.2). For any $i \in [s_j]$ let us denote by $X_{i,r}^{(k)}$ the random variable that takes value 1 if x_i is included in $\mathcal{L}_{j(i),r}^{(k)}$ and 0 otherwise, that is, the element x_i is in the set $\mathcal{L}_r^{(k)}$ during the jth round of the streaming phase of the algorithm. Clearly, for any i, r, k $X_{i,r}^{(k)} \overset{i.i.d.}{\sim} \mathsf{Ber}(2^{-k})$.

[3] We ignore the streaming index j in the Algorithm 2 as we use the same state to update the random set over different indices j.

However, the random set is actually included in $\mathcal{X}_r^{(k)}$ provided $|\mathcal{X}_r^{(k)}| + d_r^{(k)} \leqslant \mathsf{thresh}$. To indicate whether an element is actually included in $\mathcal{X}_r^{(k)}$ we define the random variable $Y_{i,r}^{(k)}$, for any $i \in [s_j]$. We define the random variable $Y_{i,r}^{(k)}$ as $Y_{i,r}^{(k)} = 1$ if $|\mathcal{X}_r^{(k)}| + d_r^{(k)} \leqslant \mathsf{thresh}$, 0 otherwise (i.e., x_i is eventually included in $\mathcal{X}_r^{(k)}$ if $X_{i,r}^{(k)} Y_{i,r}^{(k)} = 1$).

Formally, the random variable $Y_{i,r}^{(k)}$ can be defined recursively. First we note that since in **lines** 8-9 (of Algorithm 2) we remove all the elements in $\mathcal{X}_r^{(k)}$ that are in S_j, so at that time (after **line** 9)

$$\mathcal{X}_r^{(k)} \subseteq \left(\cup_{k=1}^{j-1} S_k\right) \backslash S_j = \{x_i : i \in \mathcal{F}_j\}.$$

Thus the size of $\mathcal{X}_r^{(k)}$ after **line** 9 is $\sum_{a \in \mathcal{F}_j} X_{a,r}^{(k)} Y_{a,r}^{(k)}$. Thus,

$$Y_{i,r}^{(k)} = \begin{cases} 0 & \text{if } \sum\limits_{a \in \mathcal{F}_{j(i)}} X_{a,r}^{(k)} Y_{a,r}^{(k)} + \sum\limits_{a \in \mathcal{I}_{j(i)}} X_{a,r}^{(k)} > \mathsf{thresh} \\ 1 & \text{otherwise.} \end{cases}$$

Now, in the post-streaming phase, we set

$$Z_r^{(k)} = \sum_{i \in \mathcal{F}} X_{i,r}^{(k)} Y_{i,r}^{(k)}, \forall r \in [\mathsf{Reps}], \quad \overline{Z}^{(k)} = \frac{Z_1^{(k)} + \cdots + Z_{\mathsf{Reps}}^{(k)}}{\mathsf{Reps}}.$$

Note that $\mathcal{F}$ denote the set of all final indices of the elements of S. Thus, $Z_r^{(k)}$'s are the sizes of the sets $\mathcal{X}_r^{(k)}$ after processing the stream. Moreover, these are independent for all r and k.

▶ **Lemma 13.** *For any k such that* $\mathsf{thresh} \geqslant 6c$ *where* $c := \frac{\mathsf{F}_0(\mathcal{S})}{2^k}$.

$$c(1 - 2^{-\mathsf{thresh}}) \leqslant \mathbb{E}(Z_r^{(k)}) \leqslant c \tag{1}$$

$$\mathsf{Var}(Z_r^{(k)}) \leqslant c(1 + 2^{-\mathsf{thresh}} c). \tag{2}$$

Proof. The upper bound of expectation directly follows from the linearity of expectation. For the lower bound, we first note that

$$\begin{aligned} \mathsf{Pr}(Y_{i,r}^{(k)} = 0 \mid X_{i,r}^{(k)} = 1) &\leqslant \mathsf{Pr}\left(\sum_{a \in \mathcal{F}_j} X_{a,r}^{(k)} Y_{a,r}^{(k)} + \sum_{a \in \mathcal{I}_j} X_{a,r}^{(k)} \geqslant \mathsf{thresh} + 1 \;\middle|\; X_{i,r}^{(k)} = 1 \right) \\ &\leqslant \mathsf{Pr}(\mathsf{Bin}(\mathsf{F}_0(\mathcal{S}) - 1, p) \geqslant \mathsf{thresh}) \leqslant 2^{-\mathsf{thresh}}. \end{aligned}$$

Note that the last inequality follows from Lemma 20. Thus, $\mathsf{Ex}(X_{i,r}^{(k)} Y_{i,r}^{(k)}) = \mathsf{Pr}(X_{i,r}^{(k)} = 1, Y_{i,r}^{(k)} = 1) \geqslant p(1 - 2^{-\mathsf{thresh}})$ for all i. Hence, the lower bound is established by linearity of expectation. To bound the variance, we first bound the second moment:

$$\mathsf{Ex}((Z_r^{(k)})^2) = \mathsf{Ex}(Z_r^{(k)}) + \sum_{i \neq j} \mathsf{Ex}(X_{i,r}^{(k)} Y_{i,r}^{(k)} X_{j,r}^{(k)} Y_{j,r}^{(k)}) \leqslant \mathsf{Ex}(Z_r^{(k)}) + \mathsf{F}_0(\mathcal{S})^2 p^2.$$

Hence we obtain,

$$\begin{aligned} \mathsf{Var}(Z_r^{(k)}) &\leqslant \mathsf{Ex}(Z_r^{(k)})(1 - \mathsf{Ex}(Z_r^{(k)}) + \mathsf{F}_0(\mathcal{S})^2 p^2 \\ &\leqslant c\left(1 - \left(1 - 2^{-\mathsf{thresh}}\right) c\right) + c^2 \leqslant c + c^2 2^{-\mathsf{thresh}} \end{aligned}$$ ◀

Before we complete the proof of the correctness of our Algorithm 2, we first state a core result based on bucket-load random variables (defined below) that would be used.

▶ **Definition 14** (Bucket-load). *A (c, α)-bucket-load is a nonnegative random variable Z such that the first and second moments of Z satisfy the following relations:*

$$c(1-\alpha) \leqslant \mathsf{Ex}(Z) \leqslant c \qquad AND \qquad \mathsf{Var}(Z) \leqslant c + \alpha c^2.$$

From Lemma 13 we note that for any k with $\mathsf{thresh} \geqslant 6\mathsf{F}_0(\mathcal{S})/2^k$, $Z_r^{(k)}$'s are (c, α)-bucket-load random variables for $c = \mathsf{F}_0(\mathcal{S})/2^k$ and $\alpha = 2^{-\mathsf{thresh}} \leqslant \min\{\varepsilon/2, 2^{-18}\}$. Due to independence of the random variables $Z_1^{(k)}, \ldots, Z_{\mathsf{Reps}}^{(k)}$, the random variable $\overline{Z}^{(k)}$ has mean approximately c and variance $O(c/\mathsf{Reps})$. Thus, for an appropriately large Reps, we can apply Chebyshev's inequality to show that $\overline{Z}^{(k)}$ is indeed very close to c with high probability. The following proposition makes this intuition formal. Let k_0 be the unique negative power of 2 such that

$$\frac{3}{4} < c_0 = \frac{\mathsf{F}_0(\mathcal{S})}{2^{k_0}} \leqslant \frac{3}{2} \tag{3}$$

▶ **Proposition 15.** *Let $\mathsf{Reps} = \lceil 90/\varepsilon^2 \rceil$ and let k_0 be the number defined in Equation 3. For every $k \geqslant k_0 - 1$, let $Z_1^{(k)}, \ldots, Z_{\mathsf{Reps}}^{(k)}$ be the Reps many independent $(\mathsf{F}_0(\mathcal{S})/2^k, \varepsilon/2)$-bucket load random variables. Then,*

1. *$\overline{Z}^{(k_0)}$ and $\overline{Z}^{(k_0-1)}$ are $(\varepsilon, 1/24)$-approximation of c_0 and $2c_0$ respectively. Hence, $2^{k_0} \cdot \overline{Z}^{(k_0)}$ and $2^{k_0-1} \cdot \overline{Z}^{(k_0-1)}$ are $(\varepsilon, 1/24)$-approximations of $\mathsf{F}_0(\mathcal{S})$.*
2. *$\mathsf{Pr}(\overline{Z}^{(k_0-1)} < 1) + \sum_{i \geqslant 1} \mathsf{Pr}(\overline{Z}^{(k_0+i)} \geqslant 1) \leqslant 1/4$.*

We postpone the proof of Proposition 15 to the Appendix B.

Item 2 of the Proposition 15 proves that $k^* \in \{k_0, k_0 - 1\}$ with probability at least 3/4 (where k^* is defined in **line** 16). The first part proves that $\overline{Z}^{(k_0)} \cdot 2^{k_0}$ and $\overline{Z}^{(k_0-1)} \cdot 2^{k_0-1}$ are $(\varepsilon, 1/24)$-approximations of $\mathsf{F}_0(\mathcal{S})$. Hence, we observe that $\overline{Z}^{(k^*)} 2^{k^*}$ is an $(\varepsilon, 1/3)$-approximation of $\mathsf{F}_0(\mathcal{S})$ (as this can fail either when $k^* \notin \{k_0, k_0 - 1\}$, or $\overline{Z}^{(k_0)} \cdot 2^{k_0}$ or $\overline{Z}^{(k_0-1)} \cdot 2^{k_0-1}$ fails to approximate $\mathsf{F}_0(\mathcal{S})$, the probability of any one of these can happen with probability at most 1/3). This proves that the Algorithm 2 is an F_0-estimator for Delphic Sets with WOR sampling.

3.1.2 Complexity of the Algorithm 2

The space complexity and the update time complexity are clear from the pseudo-code of the algorithm. The size of $\mathcal{X}_r^{(k)}$ is upper bounded by thresh. And since r ranges from 1 to Reps and k ranges from 1 to $\log|\Omega|$, so the maximum number of elements of Ω stored is $O\left(\frac{\log|\Omega|}{\varepsilon^2}\right)$. Nevertheless to store an element of Ω, one needs $O(\log|\Omega|)$ bits. So the total space required is $O\left(\frac{\log^2|\Omega|}{\varepsilon^2} \cdot \log \varepsilon^{-1}\right)$. The additional space required for bookkeeping is $O(\log|\Omega|)$.

By the definition of Delphic Sets with WOR sampling, knowing the size of a set S_j and checking if an element is in S_j can be done in $O(\log|\Omega|)$ time. So the for any j, r and k the **lines** 8 to 9 can be done is at most $O(|\mathcal{X}_r^{(k)}| \log(|\Omega|)) = O(\mathsf{thresh} \log(|\Omega|))$ step. The number of samples drawn from any set S_j (in **line** 12 is at most thresh. So the time taken in **line** 12 is also $O(\mathsf{thresh} \log(|\Omega|))$. We should note here that we assume **line** 10 that is drawing a sample from $\mathsf{Bin}(|S_j|, 1/2^k)$ can be done in order $O(\mathsf{thresh} \log(|\Omega|))$ steps, since $|S_j| \leqslant |\Omega|$ and $k \leqslant \log|\Omega|$. We have to do these steps for all k and r. Thus in total the update time complexity is $O(\mathsf{thresh} \log(|\Omega|) \cdot (\log|\Omega|) \cdot \mathsf{Reps}) = O\left(\frac{\log^2|\Omega|}{\varepsilon^2} \cdot \log \varepsilon^{-1}\right)$.

Thus, we have the Theorem 12.

3.2 F_0- Estimator for General Delphic Sets

Algorithm 2 in **line** 12 uses the ability to sample from the sets in the stream with WOR sampling. This property is not available in general Delphic Sets. However, with a little modification to Algorithm 2, we can also have an F_0-estimator for general Delphic Sets, which is what Theorem 3 states.

Proof of Theorem 3. To have an F_0-estimator for general Delphic sets, we need to make two important changes to Algorithm 2. Firstly, we set the thresh to be $\max\{\lceil \log 4/\varepsilon \rceil,\ 18\}$ (instead of $\max\{\lceil \log 2/\varepsilon \rceil,\ 18\}$). Secondly, we change the **line** 12 to the following:

12: $\mathcal{L}_r^{(k)} = \mathsf{WRSample}(S, d_r^{(k)}, d_r^{(k)} \log d_r^{(k)} \log(\frac{4}{\varepsilon}))$

While in Algorithm 2 in **line** 12 the set $|\mathcal{L}_r^{(k)}| = d_r^{(k)}$ with probability 1, in the modified algorithm this is not the case. In other words, if we try to prove the correctness of the modified algorithm in the same line as the proof of Algorithm 2 we note that the random variable $Y_{i,r}^{(k)}$ takes value slightly differently.

The random variable $Y_{i,r}^{(k)}$ can take value 1 for two possible cases.

1. Due to the overflow of the bucket, which can now happen with probability at most $2^{-\mathsf{thresh}} \leqslant \varepsilon/4$ (This is due to the modified setting of thresh so that $2^{-\mathsf{thresh}} \leqslant \varepsilon/4$).
2. Due to $\mathsf{WRSample}$ returning an empty set. By Lemma 11 and by the setting of the inputs of $\mathsf{WRSample}$ this can happen also with probability at most $\varepsilon/4$.

Thus we can prove a lemma corresponding to Lemma 13 and hence, $Z_r^{(k)}$ is (c, α)-bucket-load where $\alpha = \min\{2^{-18}, \varepsilon/2\}$. Now, we can apply our core Proposition 15 to conclude the correctness of the algorithm.

It is easy to see that the space and time complexity of the modified algorithm is $O\left(\frac{\log^2 |\Omega|}{\varepsilon^2} \cdot \log \varepsilon^{-1}\right)$ and $O\left(\frac{\log^2 |\Omega|}{\varepsilon^2} \cdot \log^3 \varepsilon^{-1}\right)$ respectively. ◀

4 Space Optimal Algorithm with an Oracle in NTISP(poly, LINSPACE)

In this section, we give a sketch of the proof of Theorem 4. We will implement the F_0-estimation algorithm by Gibbons and Tirthapura [13] for singleton streams as described in [2]. The proof that the algorithm gives the correct estimation follows from their proof and is hence omitted. The algorithm picks a random hash function h from a pairwise-independent family and keeps a bucket $\mathcal{X}$ of hash values of a subset of elements in the stream and a number z of leading zeros of all the elements in the bucket. The size of the bucket is bounded by $O(\frac{1}{\varepsilon^2})$. When a new item x comes, if the number of leading zeros of $h(x) \geqslant z$, the algorithm updates the bucket to $\mathcal{X} \cup \{h(x)\}$. If the bucket overflows, then z is updated to $z+1$, and all elements from $\mathcal{X}$ with leading zeros $\leqslant z$ are removed. At the end of the stream, the algorithm outputs $|\mathcal{X}| \cdot 2^z$ as the estimate. In adapting their algorithm for the case of set streams, the computational challenge is implementing the update. The central intuition is that this can be accomplished by querying a language in NTISP(poly,LINSPACE) with query size $O(\log |\Omega| \cdot \frac{1}{\varepsilon^2})$. Thus if NTISP($\mathsf{poly}$,LINSPACE) =DTISP($\mathsf{poly}$,LINSPACE), then queries to the language can be simulated in space $O(\log |\Omega| \cdot \frac{1}{\varepsilon^2})$ and time $\mathsf{poly}(\log |\Omega|, \varepsilon^{-1})$.

We will use n to denote $\log |\Omega|$, the length of the representation of any element in the universe Ω. For any binary string y, Zero(y) denotes the number of leading zeros of y. We will use the standard Toeplitz family of pairwise-independent hash functions denoted by $\mathsf{H}_{\mathsf{Teop}}(n, k)$ (see Appendix D for details). Let h be a hash function. 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 $h(y)$. If h is from $\mathsf{H}_{\mathsf{Teop}}(n,k)$, both h and h_m are efficiently computable and take linear space to represent.

For a member $S \subseteq \Omega$ of the Delphic set, we will assume a representation of size $O(|\Omega|)$ and denote it by r_S. This is required for computational purposes, and all the Delphic families discussed in earlier works have such representations (e.g., rectangles in KMP). We use the notation $x \in r_S$ to mean $x \in S$. Note that since S is a Delphic set, membership can be checked in time and space linear in the representation.

Oracle Languages

We will define two languages, L and L_{pref}, and show that they are in NTISP(poly,LINSPACE). We use these languages to implement Gibbons and Tirthapura algorithms. The first language, L, checks whether the bucket overflows by adding new elements from the current set S with the current value of the leading zeros count.

$$\begin{aligned} L =&\{\langle r_S, h, \mathcal{X}, \ell, m, 1^{\ell \log n}\rangle \mid \exists x_1 \prec x_2 \prec \ldots \prec x_k \text{ such that } x_i \in r_S \\ &\& \ \forall i (h_m(x_i) = 0^m) \ \& \ \ |\{h(x_1), \ldots, h(x_k)\} \cup \mathcal{X}| > \ell\}. \end{aligned}$$

The following language is (close to) the *prefix language* of L that can be used to construct witnesses $x_1, \ldots, x_k \in r_S$ if adding new elements does not result in an overflow of $\mathcal{X}$.

$$\begin{aligned} L_{\text{pref}} =&\big\{\langle r_S, h, \mathcal{X}, \ell, m, 1^{\ell \cdot \log n}, i, j\rangle \mid \exists x_1 \prec x_2 \prec \ldots \prec x_k \text{ such that} \\ &\quad x_i \in r_S \ \& \ \forall i (h_m(x_i) = 0^m) \& \ |\{h(x_1), \ldots, h(x_k)\} \cup \mathcal{X}| \leqslant \ell \ \& \ i^{th} \text{ bit of } x_j \text{ is } 1\big\}. \end{aligned}$$

To implement the algorithm, we must update $\mathcal{X}$ with new elements from S (represented by r_S), the current set in the stream. For this, the algorithm uses a subroutine Construct that in turn uses L_{pref} to search for a set of $\leqslant l$ the elements x_is in S so that $h_m(x_i) = 0^m$. We will ensure that we will use Construct only when we are guaranteed that adding the hash values on these elements will not make $\mathcal{X}$ overflow. For this, we will use membership in L. The Algorithm 3 is described below. Theorem 4 follows from Claim 16, Claim 17, and the guarantee of Gibbons and Tirthapura's algorithm.

Algorithm 3 DelphicWithNP($\mathcal{S}, \varepsilon$).

```
1: $h \xleftarrow{R} \mathsf{H}_{\mathsf{Teop}}(n,n)$
2: $\mathcal{X} \leftarrow \phi$, Const $\leftarrow \frac{100}{\varepsilon^2}$, $m \leftarrow 0$
3: while not End-of-Stream do
4:     On item $r_S$
5:     while $\langle r_S, h, \mathcal{X}, \mathsf{Const}, m, 1^{\mathsf{Const} \cdot \log n}\rangle \notin L$ do
6:         Remove all $y$s from $\mathcal{X}$ for which Zero($y$) $= m$
7:         $m \leftarrow m + 1$
8:     $\mathcal{X} \leftarrow \mathcal{X} \cup$ Construct($\langle r_S, h, \mathcal{X}, \mathsf{Const}, 1^{\mathsf{Const} \cdot \log n}\rangle$)
9: Output $|\mathcal{X}|2^m$
```

▷ Claim 16. Both L and L_{pref} are in NTISP(poly,LINSPACE).

Proof. We show membership of L in NTISP(poly,LINSPACE). The proof of membership of L_{pref} is similar. Consider the following non-deterministic machine M_L. M_L on input $\{\langle r_S, h, \mathcal{X}, 1^\ell, m\rangle$ first guesses a number $1 \leqslant k \leqslant l$ and $x_1 \prec x_2 \prec \ldots \prec x_k$ in Ω. Then it verifies the following: (1) $\forall i\ x_i \in r_S$, (2) $\forall i\ h_m(x_i) = 0^m$, (3) $|\{h(x_1), \dots, h(x_k)\} \cup \mathcal{X}| > \ell$. The input size is $O(|\mathcal{X}| + \ell \cdot \log n)$. Since all the steps: guessing and verification (1), (2), and (3), can be done in time polynomial in the input length, M_L runs in polynomial time. The critical observation is that M_L can write down all the guesses in space linear in the input (at most l x_is from Ω that takes $O(\ell \log n)$ bits to store). Thus, the overall space used is linear in the input length. ◁

Let k be the maximum value of $|\{h(x_1), \dots, h(x_t)\} \cup \mathcal{X}|$ so that the following property $\mathcal{P}$ holds:

1. $\exists x_1 \prec x_2 \prec \dots \prec x_t$ such that $x_i \in r_S$
2. $\forall i (h_m(x_i) = 0^m)$.

▷ Claim 17. There is a deterministic algorithm Construct, takes $\langle r_S, h, \mathcal{X}, \mathsf{Const}, 1^{\mathsf{Const}\cdot \log n}\rangle$ as input and L and L_{pref} as oracles and if $k \leqslant \mathsf{Const}$, it returns a set $\{h(x_1), \dots, h(x_k)\}$ (if non-empty) so that $\forall i (h_m(x_i) = 0^m)$ and $x_i \in r_S$. Construct runs in time polynomial and space linear in the input length and makes only queries that are linear in the input length.

Proof (Sketch). Construct first queries L to check $k > \mathsf{Const}$. If $k > \mathsf{Const}$, it rejects. Otherwise, it first finds k and uses $k \log n$ queries to L_{pref} to construct all x_is in S with the property $\mathcal{P}$. The complexity bounds are clear from the language's description and definition. ◁

5 Conclusion

In this paper, we present a new elegant algorithm that improves both the space and update time complexities for the computation of the size of the union of Delphic sets. The space and time complexities of our algorithm are $\widetilde{O}\left(\log^2(|\Omega|) \cdot \frac{\log(1/\delta)}{\varepsilon^2}\right)$. To complement our algorithm, we also show that, given access to NTISP(poly, LINSPACE) oracle, there exists a *hashing* based algorithm for approximating the size of the union of Delphic sets with space complexity $O\left(\log(|\Omega|) \cdot \varepsilon^{-2} \cdot \delta^{-1}\right)$ and $\mathsf{poly}\left(\log(|\Omega|), \varepsilon^{-1}, \log\frac{1}{\delta}\right)$ update time complexity. Note $\Omega\left(\log(|\Omega|)\right)$ lower bound for our problem follows directly from the lower bound of F_0-estimation problem, but the above result implies that if the complexity of our problem is $\omega\left(\log(|\Omega|)\right)$, then we should not expect this to be proved soon.

References

1 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. doi:10.1006/jcss.1997.1545.

2 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proc. of RANDOM*, pages 1–10, 2002. doi:10.1007/3-540-45726-7_1.

3 Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Phys. Rev. E*, 71:036113, March 2005. doi:10.1103/PhysRevE.71.036113.

4 Jon Louis Bentley. Algorithms for klee's rectangle problems. Technical report, Technical Report, Computer, 1977.

5 Jaroslaw Blasiok. Optimal streaming and tracking distinct elements with high probability. In *Proc. of SODA*, 2018. doi:10.1137/1.9781611975031.156.

6 Karl Bringmann and Tobias Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Comput. Geom.*, 43(6-7):601–610, 2010. doi:10.1016/j.comgeo.2010.03.004.

7 Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979. doi:10.1016/0022-0000(79)90044-8.

8 Timothy M. Chan. A (slightly) faster algorithm for klee's measure problem. *Comput. Geom.*, 43(3):243–250, 2010. doi:10.1016/j.comgeo.2009.01.007.

9 Eric Y Chen and Timothy M Chan. Space-efficient algorithms for klee's measure problem. *algorithms*, 3(5):6, 2005.

10 Bogdan S. Chlebus. On the klee's measure problem in small dimensions. In Branislav Rovan, editor, *SOFSEM '98: Theory and Practice of Informatics, 25th Conference on Current Trends in Theory and Practice of Informatics, Jasná*, volume 1521, pages 304–311, 1998. doi:10.1007/3-540-49477-4_22.

11 Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.*, 31(2):182–209, 1985. doi:10.1016/0022-0000(85)90041-8.

12 Michael L Fredman and Bruce Weide. On the complexity of computing the measure of $\bigcup$[ai, bi]. *Communications of the ACM*, 21(7):540–544, 1978.

13 E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research.* Lecture Notes in Computer Science 2500. Springer, 2002.

14 Joachim Gudmundsson and Rasmus Pagh. Range-efficient consistent sampling and locality-sensitive hashing for polygons. In *28th International Symposium on Algorithms and Computation, ISAAC*, volume 92 of *LIPIcs*, pages 42:1–42:13, 2017. doi:10.4230/LIPIcs.ISAAC.2017.42.

15 Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 283–288. IEEE Computer Society, 2003. doi:10.1109/SFCS.2003.1238202.

16 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS*, pages 41–52, 2010. doi:10.1145/1807085.1807094.

17 Victor Klee. Can the measure of be computed in less than o (n log n) steps? *The American Mathematical Monthly*, 84(4):284–285, 1977.

18 Kuldeep S. Meel, Sourav Chakraborty, and N. V. Vinodchandran. Estimation of the size of union of delphic sets: Achieving independence from stream size. In Leonid Libkin and Pablo Barceló, editors, *PODS*, pages 41–52. ACM, 2022. doi:10.1145/3517804.3526222.

19 Kuldeep S. Meel, N. V. Vinodchandran, and Sourav Chakraborty. Estimating the size of union of sets in streaming models. In *Proc. of PODS*, pages 126–137, 2021. doi:10.1145/3452021.3458333.

20 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis.* Cambridge University Press, USA, 2nd edition, 2017.

21 Mark H Overmars and Chee-Keng Yap. New upper bounds in klee's measure problem. *SIAM Journal on Computing*, 20(6):1034–1045, 1991. doi:10.1137/0220065.

22 A. Pavan and Srikanta Tirthapura. Range-efficient counting of distinct elements in a massive data stream. *SIAM J. Comput.*, 37(2):359–379, 2007. doi:10.1137/050643672.

23 Aduri Pavan, N. V. Vinodchandran, Arnab Bhattacharya, and Kuldeep S. Meel. Model counting meets f_0 estimation. In *PODS'21: Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 299–311. ACM, 2021. doi:10.1145/3452021.3458311.

24 He Sun and Chung Keung Poon. Two improved range-efficient algorithms for f_0 estimation. *Theor. Comput. Sci.*, 410(11):1073–1080, 2009. doi:10.1016/j.tcs.2008.10.031.

25 Srikanta Tirthapura and David P. Woodruff. Rectangle-efficient aggregation in spatial data streams. In *Proc. of PODS*, pages 283–294. ACM, 2012. doi:10.1145/2213556.2213595.

26 Richard Ryan Williams. Time-space tradeoffs for counting np solutions modulo integers. *computational complexity*, 17:179–219, 2007. URL: https://api.semanticscholar.org/CorpusID:8815358.

A Concentration Bounds

▶ **Lemma 18.** *Let Z be a random variable with $c(1-\varepsilon/2) \leqslant \mathsf{Ex}(Z) \leqslant c$ for some $c > 0$ and $0 \leqslant \varepsilon < 1$. Then,*

$$\Pr(Z \neq (1 \pm \varepsilon)c) \leqslant \frac{4\ \mathsf{Var}(Z)}{c^2\varepsilon^2}.$$

Note that the lower tail event $Z \leqslant c - c\varepsilon$ implies $Z - \mathsf{Ex}(Z) \leqslant c\varepsilon/2$. Thus, $|Z - c| > c\varepsilon$ implies that $|Z - \mathsf{Ex}(Z)| > c\varepsilon/2$. Thus, the above result follows from the Chebyshev's inequality. The following lemma directly follows from Chebyshev's inequality.

▶ **Lemma 19.**

1. *Let Z be a random variable with $\mathsf{Ex}(Z) \geqslant \mu$. Then,*

$$\Pr(Z \leqslant \mu - t) \leqslant \frac{\mathsf{Var}(Z)}{t^2}.$$

2. *Let Z be a random variable with $\mathsf{Ex}(Z) \leqslant \mu$. Then,*

$$\Pr(Z \geqslant \mu + t) \leqslant \frac{\mathsf{Var}(Z)}{t^2}.$$

We will also need the following Chernoff bound.

▶ **Lemma 20** (See Theorem 4.4 from [20])**.** *Suppose $T \sim \mathsf{Bin}(n,p)$ and $L \geqslant 6np$ then*

$$\Pr(T \geqslant L) \leqslant 2^{-L}.$$

B Proof of Proposition 15

We first observe the simple properties of the first and second moments of averages of the independent bucket-loads.

▶ **Lemma 21.** *Let c_0 and k_0 be defined as in Equation 3. Then,*

- $c_0(1-\varepsilon/2) \leqslant \mathbb{E}(\overline{Z}^{(k_0)}) \leqslant c_0$,
- $\mathsf{Var}(\overline{Z}^{(k_0)}) \leqslant V_0 := \frac{c_0 + c_0^2 2^{-18}}{\mathsf{Reps}} \leqslant \frac{1}{80}$.
- $\mathsf{Var}(\overline{Z}^{(k_0-1)}) \leqslant V_{-1} := \frac{2c_0 + 4c_0^2\varepsilon/8}{\mathsf{Reps}} \leqslant 2V_0 \leqslant \frac{1}{40}$.
- *For all $i \geqslant 1$,*

$$\mathsf{Var}(\overline{Z}^{(k_0+i)}) \leqslant V_i := \frac{2^{-i} \cdot c_0 + 2^{-2i}c_0^2\varepsilon/2}{\mathsf{Reps}} \leqslant V_0/2^i$$

The above follows directly from Lemma 13 along with the choice of $\mathsf{Reps} = 99/\varepsilon^2$ and $\varepsilon \leqslant 0.5$. Now, by using Lemma 18,

$$\Pr(\overline{Z}^{(k_0)} \neq (1 \pm \varepsilon)c_0) \leqslant \frac{4\mathsf{Var}(\overline{Z}^{(k_0)})}{c_0^2\varepsilon^2} \leqslant \frac{2}{\varepsilon^2\mathsf{Reps}}(\frac{2}{c_0} + \varepsilon) \leqslant \frac{(8/3) + 2\varepsilon}{\varepsilon^2\mathsf{Reps}} \leqslant 1/24.$$

The last inequality follows from the choice of $\mathsf{Reps} = 90/\varepsilon^2$ and we assume that $\varepsilon \leqslant 0.5$. So, $\overline{Z}^{(k_0)}$ is an $(\varepsilon, 1/24)$-approximation of c_0. Similarly, by using Lemma 18, $\overline{Z}^{(k_0-1)}$ is an $(\varepsilon, 1/24)$-approximation of $2c_0$. This completes the proof of Item 1 in Proposition 15.

Now we prove the Item 2 of Proposition 15.

We start by bounding $\mathsf{Pr}(\overline{Z}^{(k_0-1)} < 1)$. Note that $\mathbb{E}(\overline{Z}^{(k_0-1)}) \geqslant 2c_0(1-2^{-\mathsf{thresh}}) \geqslant 3/2 \times (1-2^{-18}) \geqslant 3/2 - \beta$, where β is a negligible real number such that $3/2^{19} \leqslant \beta \leqslant 2^{-17}$. Now it is easy to see that there exists some $\alpha \in \mathbb{R}$ with $2^{-16} \leqslant \alpha \leqslant 2^{-15}$ and $\alpha^2 + 4\alpha < 1$ such that $\frac{1}{2+\alpha} \leqslant 1/2 - \beta$. By using Lemma 19,

$$\mathsf{Pr}(\overline{Z}^{(k_0-1)} < 1) \leqslant (2+\alpha)^2\ \mathsf{Var}(\overline{Z}^{(k_0-1)}) = (4+4\alpha+\alpha^2)\ V_{-1} \leqslant 5V_{-1} = 10V_0 \tag{4}$$

Now let us bound $\mathsf{Pr}(\overline{Z}^{(k_0+i)} \geqslant 1)$ for every $i \geqslant 1$. Note that $\mathbb{E}(\overline{Z}^{(k_0+1)}) \leqslant c_0/2 \leqslant 3/4$. So, by Lemma 19 we obtain

$$\mathsf{Pr}\left(\overline{Z}^{(k_0+1)} \geqslant 1\right) \leqslant 16V_1 = 8V_0 \tag{5}$$

Note that for $i \geqslant 2$ we have $\mathbb{E}(\overline{Z}^{(k_0+i)}) \leqslant c_0/2^i \leqslant 1/2$. Applying Lemma 19 working out in detail we obtain,

$$\mathsf{Pr}\left(\overline{Z}^{(k_0+i)} > 1\right) \leqslant 4\mathsf{Var}\left(\overline{Z}^{(k_0+i)}\right) \leqslant 2^{-i+2}V_0$$

Thus using the infinite geometric sum we obtain

$$\sum_{i\geqslant 2} \mathsf{Pr}(\overline{Z}^{(k_0+i)} \geqslant 1) \leqslant 2V_0 \tag{6}$$

Hence from the inequalities 5 and 6 we obtain

$$\sum_{i\geqslant 1} \mathsf{Pr}(\overline{Z}^{(k_0+i)} \geqslant 1) \leqslant 10V_0 \tag{7}$$

Finally, combining the inequalities 4 and 7 we conclude

$$\mathsf{Pr}(\overline{Z}^{(k_0-1)} < 1) + \sum_{i\geqslant 1} \mathsf{Pr}(\overline{Z}^{(k_0+i)} \geqslant 1) \leqslant 1/4$$

This completes the proof of Item 2 of Proposition 15. ◀

C Further Improvements for Klee's Measure Problem

Recall, every d-dimensional rectangle is a Delphic set with $\Omega = [\Delta]^d$. Therefore, Theorem 3 provides an algorithm that has space complexity $O\left(\frac{d^2\log^2\Delta}{\varepsilon^2}\cdot\log\varepsilon^{-1}\right)$ and has update time complexity $O\left(\frac{d^2\log^2\Delta}{\varepsilon^2}\cdot\log^3\varepsilon^{-1}\right)$ for Klee's Measure Problem. We now discuss how we can further improve the dependence on ε. The key idea behind such an improvement is to observe that we can avoid the overhead due to WOR by by replacing with sampling from Geometric distribution. Such a replacement is possible only because there exists a total ordering for all the elements in $\Omega = [\Delta]^d$ such that we can access i-th element in time $\log(|\Omega|)$. Interestingly, all the known classes of Delphic sets satisfy the above property and therefore, we formalize such a family below:

▶ **Definition 22.** *A set $S \subseteq \Omega$ belongs to Ordered Delphic family if there exists a total ordering $\preceq$ for all the elements in the Ω and the following queries can be done in $O(\log|\Omega|)$ time.*

1. *Know the size of the set S,*
2. *For any $1 \leqslant i \leqslant |S|$ the ith element of the set S can be obtained,*
3. *Given any x check if $x \in S$.*

In Section 2.4 we discussed how one can sample a Bernoulli process from a Delphic Set by using the coupon collector theorem. There is an alternate way to sample a Bernoulli process if the set is a Ordered Delphic Set. The set L returned by the algorithm $Sample(S,p)$ follows $\mathsf{Ber}(S,p)$. Since there is an order to the elements in a Ordered Delphic set, the process is basically same as sampling a Bernoulli process from $[|S|]$. This is a well known result in probability theory and details can be found in [3].

Algorithm 4 GeometricSample(n,p).

```
1: **Input** Set $n$  $p \in [0,1]$.
2: **Initialize** $L = \emptyset$; $t = 0$
3: **for** $t \leqslant |S|$ **do**
4:     Sample $g$ from $Geo(p)$
5:     $t = t + g$
6:     **if** $t \leqslant n$ **then** $L = L \cup \{t\}$
7: Output $L$
```

▶ **Lemma 23.** *The subroutine* GeometricSample(n,p) *outputs a subset of* $[n]$ *according to the Bernoulli process. More precisely, if we set* $I_j = 1$ *if* $i \in L$*, 0 otherwise then* $I_1, \ldots, I_n$ *follow identically and independently distributed to* $\mathsf{Ber}(p)$.

Using the subroutine GeometricSample we can now present the improved algorithm (Algorithm 5) for the Ordered Delphic Sets.

Algorithm 5 [F$_0$-Estimator for Ordered Delphic Sets.]

```
1: **Input** Stream = $\langle S_1, S_2, \ldots, S_M \rangle$, $\varepsilon$, $\delta$
2: **Initialize**
3: $p \leftarrow 1$; thresh $\leftarrow \max\{\lceil \log 2/\varepsilon \rceil,\ 18\}$; Reps $\leftarrow \lceil 90/\varepsilon^2 \rceil$;
4: **for** $(1 \leqslant k \leqslant \log n)$ and $(1 \leqslant r \leqslant \text{Reps})$ **do**
5:     $\mathcal{X}_r^{(k)} \leftarrow \emptyset$;
   STREAMING PHASE:
6: **for** $j = 1$ to $M$ **do**
7:     **for** $(1 \leqslant k \leqslant \log n)$ and $(1 \leqslant r \leqslant \text{Reps})$ **do**
8:         **for** all $s \in \mathcal{X}_r^{(k)}$ **do**
9:             **if** $s \in S_i$ **then** remove $s$ from $\mathcal{X}_r^{(k)}$
10:        Draw $D_r^{(k)}$ from GeometricSample$(|S_j|, 1/2^k)$
11:        **if** $|\mathcal{X}_r^{(k)}| + |D_r^{(k)}| \leqslant$ thresh **then**
12:            $\mathcal{L}_r^{(k)} = \{S_j[\ell] \mid \ell \in D_r^{(k)}\}$
13:            $\mathcal{X}_r^{(k)} = \mathcal{X}_r^{(k)} \cup \mathcal{L}_r^{(k)}$
   POST-STREAMING PHASE:
14: **for** $1 \leqslant k \leqslant \log n$ **do**
15:     $\bar{z}^{(k)} = \frac{1}{\text{Reps}} \sum_{r=1}^{\text{Reps}} |\mathcal{X}_r^{(k)}|$
16: $k^* = \max\{k \in [\log n] : \bar{z}^{(k)} \geqslant 1\}$
17: **Output** $\bar{z}^{(k^*)}\dot{2}^{k^*}$
```

Algorithm 5 is the streaming algorithm for the Ordered Delphic Sets. For proving the correctness of Algorithm 5, we have the following theorem:

▶ **Theorem 24** (correctness theorem of Algorithm 5). *If $\mathcal{S}$ is a stream of Ordered Delphic Set the Algorithm 5 is an $(\varepsilon, 1/3)$-F_0-estimator. Also, the space and update time complexity of Algorithm 5 is $O\left(\frac{\log^2|\Omega|}{\varepsilon^2} \cdot \log \varepsilon^{-1}\right)$.*

Proof of Theorem 24. Note that, the only difference between Algorithm 5 and Algorithm 2 is in **lines** 10, 11 and 12. Also note that the Algorithm 5 behaves identically if **lines** 10 to 13 is replaced by

```
10: Draw $D_r^{(k)}$ from GeometricSample$(|S_j|, 1/2^k)$
11: $\mathcal{L}_r^{(k)} = \{S_j[\ell] \mid \ell \in D_r^{(k)}\}$
12: if $|\mathcal{X}_r^{(k)}| + |D_r^{(k)}| \leqslant$ thresh then
13:     $\mathcal{X}_r^{(k)} = \mathcal{X}_r^{(k)} \cup \mathcal{L}_r^{(k)}$
```

By Lemma 23 the size of $D_r^{(k)}$ is distributed according to the binomial distribution $\mathsf{Bin}(|S_k|, 1/2^k)$, that is the distribution of $d_r^{(k)}$ in Algorithm 2 is same as the distribution of $|D_r^{(k)}|$ in Algorithm 5. Thus, $L_r^{(k)}$ is distributed according to $\mathsf{Ber}(S_j, 1/2^k)$ which is also the same distribution if $|D_r^{(k)}|$ samples are drawn from S_j using WOR sampling.

Thus the correctness of the Algorithm 5 follows from the correctness of Algorithm 2. The complexities of Algorithm 5 is also identical to that of Algorithm 2. ◀

Recall that every d-dimensional rectangle can be viewed as an Ordered Delphic set, therefore, Theorem 24 implies the following result in the context of Klee's Measure Problem.

▶ **Corollary 25** (Improved Algorithm for KMP). *There is a streaming algorithm for the Klee's Measure Problem with space and update time complexity $O\left(\frac{d^2 \log^2 \Delta}{\varepsilon^2} \cdot \log \frac{1}{\varepsilon}\right)$.*

D Basic Probability Results

▶ **Definition 26** (Geometric Distribution). *Given any $p \in (0, 1]$ the Geometric Distribution over the set of positive integers $\{1, 2, 3, \ldots\}$ is denoted as $\mathsf{Geo}(p)$ where probability of a positive integer k is $(1-p)^{k-1}p$.*

▶ **Theorem 27** (Coupon Collection Problem). *Given access to uniform random samples from a set T and a number $r \leqslant |T|$, let W_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 $\mathsf{Pr}\left(W_r > \beta r \log r\right) \leqslant r^{-\beta+1}$.*

▶ Remark 28 (Pairwise Independent Hash Function). We will use pairwise independent hash functions. For an integer n, k and $\mathcal{H}(n, k) \triangleq \{h : \{0,1\}^n \to \{0,1\}^k\}$ be a family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^k$. We use $h \xleftarrow{R} \mathcal{H}(n,k)$ to denote the probability space obtained by choosing a function h uniformly at random from $\mathcal{H}(n,k)$. A family of hash functions $\mathcal{H}(n,k)$ is 2−wise independent if $\forall \alpha_1, \alpha_2 \in \{0,1\}^k$, distinct $x_1, x_2, \in \{0,1\}^n, h \xleftarrow{R} \mathcal{H}(n,k)$, $\Pr[(h(x_1) = \alpha_1) \wedge (h(x_2) = \alpha_2)] = \frac{1}{2^{2k}}$. Several families of 2-wise independent hash functions are known. We will use $\mathsf{H}_{\mathsf{Teop}}(n,k)$, which is known to be 2-wise independent [7]. The family is defined as follows: $\mathsf{H}_{\mathsf{Teop}}(n,k) \triangleq \{h : \{0,1\}^n \to \{0,1\}^k\}$ is the family of functions of the form $h(x) = Ax + b$ with $A \in \mathbb{F}_2^{k \times n}$ and $b \in \mathbb{F}_2^{k \times 1}$ where A is a uniformly randomly chosen Toeplitz matrix of size $k \times n$ while b is uniformly randomly matrix of size $k \times 1$. It is worth noticing that $\mathsf{H}_{\mathsf{Teop}}$ can be represented with $\Theta(n)$-bits and hash value of any element $x \in \{0,1\}^n$ can be computed in $O(nk)$ time.