Theoretical Computer Science 590 (2015) 106–117

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Efficient sampling of non-strict turnstile data streams

Neta Barkay *, Ely Porat *, Bar Shalem *

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

ARTICLE INFO

Article history:
Received 19 November 2013
Received in revised form 16 July 2014
Accepted 16 January 2015
Available online 21 January 2015

Keywords:
Sampling
Inverse distribution
Data streams

ABSTRACT

We study the problem of generating a large sample from a data stream $\mathbb{S}$ of elements (i, v), where i is a positive integer key, v is an integer equal to the count of key i, and the sample consists of pairs (i, C_i) for $C_i = \sum_{(i,v)\in\mathbb{S}} v$. We consider strict turnstile streams and general non-strict turnstile streams, in which C_i may be negative. Our sample is useful for approximating both forward and inverse distribution statistics, within an additive error ϵ and provable success probability $1-\delta$.

Our sampling method improves by an order of magnitude the known processing time of each stream element, a crucial factor in data stream applications, thereby providing a feasible solution to the sampling problem. For example, for a sample of size $O(\epsilon^{-2}\log(1/\delta))$ in non-strict streams, our solution requires $O((\log\log(1/\epsilon))^2 + (\log\log(1/\delta))^2)$ operations per stream element, whereas the best previous solution requires $O(\epsilon^{-2}\log^2(1/\delta))$ evaluations of a fully independent hash function per element.

We achieve this improvement by constructing an efficient K-elements recovery structure from which K elements can be extracted with probability $1-\delta$. Our structure enables our sampling algorithm to run on distributed systems and extract statistics on the difference between streams.

1. Introduction

In the *turnstile* data stream model [23], the general representation of data streams, the data is a sequence of N elements composed of a key i and a count v, $(i, v) \in U \times R$, where $U = \{1, \ldots, u\}$ and $R = \{-r, \ldots, r\}$. The vector of cumulative counts $\boldsymbol{C} = (C_1, \ldots, C_u)$ starts as $\boldsymbol{C} = \boldsymbol{0}$ and for every input element (i, v) it is updated by $C_i \leftarrow C_i + v$. Let $\boldsymbol{C}(t)$ be the state of $\boldsymbol{C}$ after processing the t'th element in the stream. For every t, i, $C_i(t) \in R$.

The function $f: U \to R$ where $f(i) = C_i$ describes the *forward distribution* of the stream. A common use of data stream algorithms is to calculate stream statistics on f, or find its frequent items. Obtaining a synopsis of the data by sampling the stream and then querying the sample is a basic technique to perform this task. For example, in order to approximate queries on f, we can use an *ϵ-approximate sampling algorithm* for $\epsilon \in (0, 1)$, which outputs $S \subseteq \{(i, f'(i)) : f(i) \neq 0\}$, where $f'(i) \in [(1-\epsilon)f(i), (1+\epsilon)f(i)]$. Note that we consider a key i with $C_i = 0$ to be a *deleted* element that should not affect stream statistics, and therefore must not appear in a sample of the stream.

However, another approach should be taken when answering queries on the *inverse distribution* function f^{-1}, defined as $f^{-1}(C) = \frac{|\{i:C_i=C\}|}{|\{i:C_i\neq 0\}|}$ for $C \neq 0$, i.e. the fraction of distinct keys with a cumulative count C. In order to answer queries on f^{-1}, an *exact sample* $S \subseteq \{(i, f(i)) : f(i) \neq 0\}$ is required. To illustrate this, assume $f^{-1}(C) = \alpha$ for a fraction $\alpha \in (0, 1)$, and in the

* Corresponding authors.
E-mail addresses: netabarkay@gmail.com (N. Barkay), porately@gmail.com (E. Porat), barshalem@gmail.com (B. Shalem).

http://dx.doi.org/10.1016/j.tcs.2015.01.026
0304-3975/

ϵ-approximate sample for every i with $f(i) = C$ the approximated cumulative count is $(1+\epsilon)C$. In that case one might get $f^{-1}(C) = 0$ instead of α, a significant change to f^{-1}.

The algorithms we present in this paper provide an exact sample for *strict turnstile* and *non-strict turnstile* data streams, defined as follows: In the *strict turnstile* model, $\forall t, i,\ C_i(t) \geq 0$, while in the *non-strict turnstile* model, $C_i(t)$ may obtain negative values. A sample S drawn at time t is $S(t) \subseteq \{(i, C_i(t)) : C_i(t) \neq 0\}$. To simplify the notation we consider sampling only at the end of the stream, and denote $C_i = C_i(N)$ and $S = S(N)$.

Since the sample S that we generate is exact, it is useful for calculating both forward and inverse distribution statistics. Its applications include network traffic analysis [20] and geometric data streams [8,9]. For example, in [20] inverse distribution statistics were used for earlier detection of content similarity, an indicator of malicious IP traffic. Another use is detecting DoS attacks, specifically SYN floods, which are characterized as flows with a single packet.

Previous work. Most previous work on sampling dynamic streams that support deletions was limited to approximating the forward distribution [3,13]. Works on the inverse distribution include a restricted model where $\forall i,\ C_i \in \{0, 1\}$ [12,28], and minwise-hashing, which samples the set of keys uniformly but does not support deletions [6]. The work in [14] supports only a few deletions.

Inverse distribution queries in streams with multiple deletions were supported in a work by Frahling et al. [8,9], who developed a solution for strict turnstile data streams and used it in geometric applications. Cormode et al. [5] developed a solution for both strict turnstile and non-strict turnstile streams. However, they did not analyze the required randomness or the algorithm's error probability in the non-strict model. L_p samplers, and specifically L_0 samplers for non-strict streams were built by Monemizadeh and Woodruff [22] and Jowhari et al. [18], however [18] lacked the time analysis of the sparse recovery.

Our results. Previous works [5,8,9,18,22] constructed basic structures for sampling only a single element. For applications that require a sample of size K, one has to use $O(K)$ independent instances of their structure, obtaining a K-size independent sample. Running the sampling procedure $O(K)$ times in parallel means that each stream element is inserted as an input to $O(K)$ instances, requiring a long time to process. To illustrate this, consider stream queries which require a sample of size $K = \Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$, where the results are ϵ-approximated, and $1-\delta$ is the success probability of the process. For $\epsilon = 10^{-2}$ and $\delta = 10^{-6}$, the number of operations per stream element is multiplied by about 200,000. The structures of [5,8,9,18,22] cannot be used to generate a K-size sample due to this unfeasible process load.

To solve this problem, we construct a single K-elements recovery structure from which an entire sample of size K can be extracted. The problem of K-recovery has been studied in different variants. For example, it is similar to the sparse recovery problem [15,26] and to IBLT [16]. In sparse recovery we have a signal that is sparse or consists of a small number of high magnitude elements and the rest of the elements are negligible. The goal is to acquire a small amount of information about this signal in a linear, nonadaptive way and then use that information to quickly recover the high magnitude elements. The difference between our problem and sparse recovery is there is no tail noise and we limit the amount of randomness. IBLT is a randomized data structure that supports insertion, deletion, lookup of key-value pairs and a full listing of the pairs it contains with high probability, as long as the number of key-value pairs does not exceed a certain threshold. The IBLT is based on a set of random hash functions that determine where key-value pairs are stored, and a set of counters in each array cell that resolve collisions. In comparison to IBLT, our solution achieves $1-\delta$ success probability in a shorter processing time, and with reduced randomness as opposed to IBLT's fully independent hash functions. Hence, our structures, especially ϵ-FRS (ϵ-Full Recovery Structure), may be of independent interest.

Our contributions are as follows:

- We reduce significantly the processing time per stream element. Our optimization enables applications that were previously limited to gross approximations to obtain much more accurate approximations in feasible time.
- We present solutions for both strict and non-strict turnstile data streams. Our algorithms have proven success probability for non-strict streams. We accomplish this by developing a structure called the *Non-strict Bin Sketch*.
- We provide efficient algorithms in terms of the randomness requirements. Our algorithms do not require fully independent or min-wise independent hash functions, or pseudorandom generators which have a drawback of large evaluation time, or are impractical to use due to the memory required to represent them. The use of fully independent hash functions is impractical because they require $O(u\log(r))$ bits just to keep a function of the form $h : [u] \to [r]$. Min-wise independent hash functions, or their ϵ-min-wise approximation, require $O(\log(1/\epsilon))$-wise independence and are evaluated in $O(\log(1/\epsilon))$ time. Pseudorandom generators such as Nisan's generator take $\log(n)$ time to evaluate.
- To the best of our knowledge, we are the first to reduce time complexity of hash functions by choosing $\Theta(\log\frac{1}{\delta})$-wise independent hash functions and evaluating them efficiently with the multi-point evaluation method. We generate a sample with $1-\delta$ success probability for any $\delta > 0$. Our method outperforms the traditional approach of increasing the success probability from a constant to $1-\delta$ by $\Theta(\log\frac{1}{\delta})$ repetitions. The traditional approach takes $O(\log\frac{1}{\delta})$ time, while our approach reduces processing time to $O((\log\log\frac{1}{\delta})^2)$.

Table 1
Comparison of our sampling algorithms to previous work.

Alg.	Memory size	Update time given hash oracle	Overall update time per element	Sample extraction time	Model
[8,9]	$K \log^2(\frac{ur}{\delta})$	$K \log u$	K polylog u	K	S
[9]	$K \log u \frac{1}{\epsilon} \log \frac{1}{\delta} \log(\frac{ur}{\epsilon})$	$K \frac{1}{\epsilon} \log \frac{1}{\delta} \log u$	$K \frac{1}{\epsilon} \log \frac{1}{\delta} \log u$	$K \frac{1}{\epsilon} \log \frac{1}{\delta}$	S
[5]	$K \log^2 u \log(ur)$	$K \log u$	$K \log u$	$K \log^2 u$	S
[5]	$K \log u \log \frac{1}{\delta} \log(ur)$	$K \log \frac{1}{\delta}$	–	$K \log u \log \frac{1}{\delta}$	NS
[18]	$K \log^2 u \log \frac{1}{\delta}$	K	$K \log u$	$K \log u \log \frac{1}{\delta}$	NS
[22]	$K \log^2 u \log(ur)$	$K \log^2 u$	$K \log^3 u$	$K \log u$	NS
FRS	$K \log u \log \frac{K}{\delta} \log(ur)$	$\log \frac{K}{\delta}$	$\log \frac{K}{\delta}$	$K \log \frac{K}{\delta}$	S/NS
ϵ-FRS	$K \log u \log(\frac{ur}{\delta})$	1	$(\log \log \frac{K}{\delta})^2$	K	S/NS

Notes: FRS and ϵ-FRS support sample sizes $K = \Omega(\log \frac{1}{\delta})$ and $K = \Omega(\frac{1}{\epsilon} \log \frac{1}{\delta})$ respectively. Memory size is given in bits. Update time is the number of operations per element. Sample extraction time is for a K-size sample. S and NS denote the strict and non-strict models respectively. In [5] S is the deterministic approach, NS is the probabilistic approach, and the hash function is fully independent. In [18] the extraction time assumes sparse recovery in linear time.

Our approach overcomes a few challenges.

- The sample requires some independence in order to be representative. Previous works used multiple independent instances, generating an independent sample at the cost of multiplying the number of random bits. Since we use a single structure, the independence depends on its hash functions. We define the notion of a (t, ϵ)-*partial sample* and show how to generate a sample for $t = \Omega(\log \frac{1}{\delta})$ using simple hash functions with low randomization and low evaluation time.
- Achieving $1 - \delta$ success probability is challenging for inputs of small size of $o(\log \frac{1}{\delta})$. For this purpose, we construct the *Small Recovery Structure*, based on techniques from coding theory.

A comparison of our algorithms to previous work is presented in Table 1. We introduce two algorithms, denoted FRS (Full Recovery Structure) and ϵ-FRS. The table shows our improvement of the update and sample extraction times.

2. Preliminaries

2.1. Problem definition

Given a data stream of N elements $\{(i, v)\}$, which is either a strict or non-strict turnstile stream, assume there is an application that performs queries on the stream such as inverse distribution queries. This application allows an $\epsilon \in (0, 1)$ additive approximation error, and a $1 - \delta$ for $\delta \in (0, 1)$ success probability of the process. The application predefines the size of the sample it requires to be K, where K might depend on ϵ and δ. The input to our sampling algorithm is the data stream, K and δ.

Let $D = \{(i, C_i) : C_i \neq 0\}$ be the set of keys and their cumulative counts in the stream at the time we sample. The output of our sampling algorithm is a sample $S \subseteq D$ of size $|S| = \Theta(K)$, generated with probability $1 - \delta$. Note that the size of the sample returned is of order K and not precisely K.

We define the following two types of samples.

Definition 1. A *t-wise independent sample* is a random set $S \subseteq D$ in which each subset of t distinct elements in D has equal probability to be in S.

Definition 2. Let $X \subseteq D$ be a sample obtained by a t-wise independent sampling algorithm, and let $\epsilon \in (0, 1)$. A subset $S \subseteq X$ of size $(1 - \epsilon)|X| \leq |S| \leq |X|$ is a (t, ϵ)-*partial sample*.

FRS returns a t-wise independent sample, where $t = \Omega(\log \frac{1}{\delta})$. ϵ-FRS returns a (t, ϵ)-partial sample. The key insight is that both samples can provide the same order of approximation as an independent sample, i.e. they have similar quality. For example, these samples of size $K = \Omega(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ enable the ϵ-approximation of inverse distribution queries. The (t, ϵ)-partial sample has a fractional bias of at most ϵ, adding an error of at most ϵ to the approximation (see Section 5).

2.2. Analysis techniques

We denote by $\mathcal{H}_t(A, B)$ the family of t-wise independent hash functions $h : A \to B$, and by $h \in_R \mathcal{H}$ a function selected randomly from $\mathcal{H}$. In this paper we use $\mathcal{H}_t$ for $t = \Theta(\log \frac{1}{\delta})$ and $t = \Theta(\log \frac{K}{\delta})$. We use the following techniques in our analysis.

Bounding error probabilities. The Moment Inequality for high moments enables us to bound error probabilities as follows: for a random variable Z and an even number l,

$$\Pr\left[\left|Z - E[Z]\right| \geq t\right] = \Pr\left[\left(Z - E[Z]\right)^l \geq t^l\right] \leq \frac{\Delta^l}{t^l}$$

where $\Delta^l = E[(Z - E[Z])^l]$ is the l'th central moment of Z. We use the estimation of Indyk [17] to Δ^l, where Z is a sum of l-wise independent indicator variables: $\Delta^l \leq 8(6l)^{(l+1)/2}E[Z]^{(l+1)/2}$.

This implies:

$$\Pr\left[\left|Z - E[Z]\right| \geq \alpha E[Z]\right] \leq \frac{48l}{\alpha}\left(\frac{6l}{\alpha^2 E[Z]}\right)^{(l-1)/2}.$$

Lemma 1. *Let Z be a sum of l-wise independent indicator variables with $E[Z] \geq \frac{c}{\epsilon^2}\log\frac{1}{\delta}$ for some $0 < \epsilon, \delta \in (0, \frac{1}{4})$, $\epsilon > \text{poly}(\delta)$, and constant c. Let $l = \tilde{c}\log\frac{1}{\delta}$ for some constant $\tilde{c}$ be an even number. Then for appropriate constants $c, \tilde{c}$, $\Pr[|Z - E[Z]| \geq \epsilon E[Z]] < \delta$.*

Proof.

$$\begin{aligned}\Pr\left[\left|Z - E[Z]\right| \geq \epsilon E[Z]\right] &\leq \frac{48\tilde{c}\log\frac{1}{\delta}}{\epsilon}\left(\frac{6\tilde{c}\log\frac{1}{\delta}}{\epsilon^2\frac{c}{\epsilon^2}\log\frac{1}{\delta}}\right)^{(\tilde{c}\log\frac{1}{\delta}-1)/2} \\ &= \frac{48\tilde{c}\log\frac{1}{\delta}}{\epsilon}\left(\frac{6\tilde{c}}{c}\right)^{(\tilde{c}\log\frac{1}{\delta}-1)/2} \qquad (*)\end{aligned}$$

$\epsilon > \text{poly}(\delta)$, i.e. for some predefined constant c', $\epsilon > \delta^{c'}$. So for a constant $\tilde{c} \geq \max(45, 8(c'+1))$ we get,

$$\begin{aligned}\log\left(\frac{1}{\epsilon}\cdot 48\tilde{c}\log\frac{1}{\delta}\right) &< c'\log\frac{1}{\delta} + \log 48\tilde{c} + \log\frac{1}{\delta} \\ &= \left(c'+1\right)\log\frac{1}{\delta} + \log 48\tilde{c} < \frac{\tilde{c}}{4}\log\frac{1}{\delta}\end{aligned}$$

for any $\delta \in (0, \frac{1}{4})$. And for $c \geq 12.77\tilde{c}$,

$$(*) < (2)^{\frac{\tilde{c}}{4}\log\frac{1}{\delta}} \cdot \left(\frac{6\tilde{c}}{c}\right)^{\frac{\tilde{c}}{4}\log\frac{1}{\delta}} = \left(\left(\frac{12\tilde{c}}{c}\right)^{\frac{\tilde{c}}{4}}\right)^{\log\frac{1}{\delta}} < \delta. \qquad \square$$

Note that if instead of an ϵ-approximation we only need approximation by a constant α, in order to prove $\Pr[|Z - E[Z]| \geq \alpha E[Z]] < \delta$ we require only $E[Z] = \Omega(\log\frac{1}{\delta})$.

Hash evaluations. A family of t-wise independent hash functions can be constructed from the set of polynomials of degree less than t over a finite field. Using this construction, we evaluate the hash functions efficiently by using the polynomials multipoint evaluation algorithm [11]. In this method, the time complexity of evaluating a polynomial of degree less than t on t points is $O(t\log^2 t)$. Thus, when we batch t points together we can evaluate $h \in \mathcal{H}_t$ in $O(\log^2 t)$ amortized time per point.

3. Overview

In this section we provide an overview of our sampling algorithm. Our goal is to return a sample of size K. Since the number of distinct keys in the input stream is unknown in advance, we keep an L_0 estimation structure to estimate it. We map the elements into levels in a way that enables us to extract a sample of size $\Theta(K)$ from one of the levels. The level is chosen based on the L_0 estimation (see Fig. 1). We describe the data structure hierarchy, explain how the data structure is updated for each stream element, and finally show how the output sample is extracted.

The data structure is composed of several levels. Each level has an FRS (or an ϵ-FRS) that is composed of several arrays, and every array cell holds a block called Bin Sketch (BS). Each input stream element (i, v) is first mapped to a level l. Then it is inserted to one BS in each of the arrays of level l.

FRS and ϵ-FRS are our two recovery structures, from which a sample is extracted. Assume a structure (FRS or ϵ-FRS) contains the set of elements X, where $K \leq |X| \leq \tilde{K}$ and $\tilde{K} = \Theta(K)$. FRS enables us to extract X, returning a t-wise independent sample, for $t = \Theta(\log\frac{1}{\delta})$. ϵ-FRS enables us to extract $X' \subseteq X$ where $|X'| \geq (1-\epsilon)|X|$, returning a (t, ϵ)-partial sample.

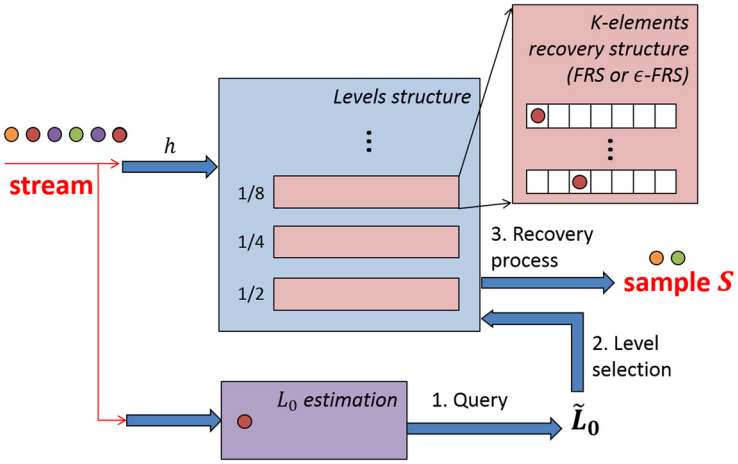

Fig. 1. Updating the data structure: each stream element is mapped to a level from the levels structure (center). Each level holds a recovery structure (top right). **Extracting the output sample:** First L_0, the number of unique keys, is estimated (1). Based on the L_0 estimation a level with about K elements is selected (2). Finally all elements are recovered in the level selected (3).

In order to select the level from which the sample is extracted we use a separate L_0 estimation structure that estimates the number of distinct keys in the stream. This structure is updated per each stream element, in parallel to the process described above.

To generate the output sample S we query the L_0 estimation structure, and select a level l^* that has $\Theta(K)$ elements with probability $1-\delta$. We extract the elements of X from l^*, or at least $(1-\epsilon)|X|$ of them, depending on the recovery structure, with probability $1-\delta$. These extracted elements are the final output sample S (see Fig. 1).

4. Sampling algorithms

4.1. Levels structure

The first operation on each stream element is mapping it to a single level in a levels structure. We take $h \in_R \mathcal{H}_t(U,[u])$ and derive $h_l(x) = \lfloor 2^l h(x)/u \rfloor$, for $l \in [L]$ where $L = \log_2 u$. (i,v) is mapped to level $l \Leftrightarrow (h_l(i) = 0 \wedge h_{l+1}(i) \neq 0)$.

Our sample S consists of the elements from one of the levels. Two issues need to be addressed. First, S should be t-wise independent, and hence we select $h \in_R \mathcal{H}_t$. Second, the level selection should be independent of our specific mapping in order to avoid a biased sample, as appeared in some previous works. Hence, we use an estimation of L_0, the number of distinct keys in the stream (see Section 4.2), and select a level where $\Theta(K)$ elements are expected.

Lemma 2. *Let $h \in_R \mathcal{H}_t$ for $t = \Omega(\log \frac{1}{\delta})$ be the function that maps the elements to levels. Assume we have an estimation of L_0, $L_0 \leq \tilde{L}_0 \leq \alpha L_0$ for $\alpha > 1$, and assume $K = \Omega(\log \frac{1}{\delta})$. Let l^* be the level for which $\frac{1}{\alpha}\tilde{L}_0(\frac{1}{2})^{l^*+2} < 2K \leq \frac{1}{\alpha}\tilde{L}_0(\frac{1}{2})^{l^*+1}$, and X be the elements in l^*. Then* $\Pr[K \leq |X| \leq (4\alpha+1)K] \geq 1-\delta$.

Proof. Let X_l be a random variable that indicates the number of elements in level l, i.e. $X_l = |\{i : C_i \neq 0 \wedge (h_l(i) = 0 \wedge h_{l+1}(i) \neq 0)\}|$. First we calculate $E[X_{l^*}]$ and then we show that X_{l^*} cannot be far from $E[X_{l^*}]$ with high probability.

$\forall l \in [L]$, key i, $p_l = \Pr_{h\in\mathcal{H}}[h_l(i) = 0 \wedge h_{l+1}(i) \neq 0] = (\frac{1}{2})^l - (\frac{1}{2})^{l+1} = (\frac{1}{2})^{l+1}$. $E[X_l] = L_0 p_l$. From $L_0 \leq \tilde{L}_0 \leq \alpha L_0$ and $\frac{1}{\alpha}\tilde{L}_0 p_{l^*+1} < 2K \leq \frac{1}{\alpha}\tilde{L}_0 p_{l^*}$ we derive: $2K \leq \frac{1}{\alpha}\tilde{L}_0 p_{l^*} \leq L_0 p_{l^*} \leq \tilde{L}_0 p_{l^*} \leq 4\alpha K$. Hence for level l^*, $2K \leq E[X_{l^*}] \leq 4\alpha K$.

Since $E[X_{l^*}] = \beta K$ for some $2 \leq \beta \leq 4\alpha$, using the variant of Lemma 1 with constant ϵ we have: $\Pr[|X_{l^*} - E[X_{l^*}]| \geq K] = \Pr[|X_{l^*} - E[X_{l^*}]| \geq \frac{1}{\beta}E[X_{l^*}]] < \delta$. We can use the lemma since X_{l^*} is a sum of t-wise independent variables, $t = \Theta(\log \frac{1}{\delta})$, and $K = \Omega(\log \frac{1}{\delta})$. □

We extract the sample from level l^* as defined in Lemma 2. We denote the maximal number of elements in l^* by $\tilde{K} = (4\alpha+1)K$.

4.2. L_0 estimation

We estimate the *Hamming norm* $L_0 = |\{i : C_i \neq 0\}|$ [4], the number of distinct keys in a stream with deletions, as follows. We use the structure of Kane et al. [19], which provides an ϵ-approximation $\tilde{L}_0$ with 2/3 success probability in both strict and non-strict streams. It uses $O(\frac{1}{\epsilon^2}\log u(\log\frac{1}{\epsilon} + \log\log(r)))$ bits of memory and has $O(1)$ update and report times.

In order to increase the success probability to $1-\delta$ we use $h \in_R \mathcal{H}_t(U,[\tau])$ for $t=\Theta(\log^2\frac{1}{\delta})$ to map each stream element to one of $\tau=\Theta(\log\frac{1}{\delta})$ instances of Kane et al. structure. Let the vector $\boldsymbol{B}=(B_0,\ldots,B_{\tau-1})$ be the number of elements in each instance. If $|\{i\in[\tau]\colon B_i \text{ is a constant approximation to } \frac{L_0}{\tau}\}|>c\tau$ for a large constant fraction c, the median of the estimations multiplied by τ is a constant approximation to L_0 with probability $1-\delta$.

When $L_0=\Omega(\log^2\frac{1}{\delta})$, with probability $1-\delta$ all B_i's are approximately equal. When $L_0<n=c'\log^2\frac{1}{\delta}$ for a constant c', the elements are mapped to instances independently since h is $\Theta(\log^2\frac{1}{\delta})$-wise independent. Since the elements are mapped independently, vector $\boldsymbol{B}$ satisfies the *negative association* and *negative regression* conditions [7]. Therefore we can use the Chernoff–Hoeffding bounds to estimate the number of elements in each instance. We get the required result when $L_0=\Omega(\log\frac{1}{\delta})$.

Thus, we can estimate L_0 when $L_0=\Omega(\log\frac{1}{\delta})$ with $1-\delta$ success probability using $O(\log u(\log\log(r)+\log\frac{1}{\delta})\log\frac{1}{\delta})$ bits of space, since we can have a constant approximation error, and $O((\log\log\frac{1}{\delta})^2)$ update time, since one $h\in\mathcal{H}_{\Theta(\log^2\frac{1}{\delta})}$ is evaluated. We can report L_0 in $O(1)$ time when the process is performed lazily.

4.3. Collisions detection in data streams with deletions

In this section we present the *Bin Sketch* (BS), a building block in our data structure. Given a stream $\{(i_j,v_j)\}_{j\in[N]}$, the input to BS is a substream $\{(i_j,v_j)\}_{j\in J}$ where $J\subseteq[N]$. BS maintains a sketch of J, identifies if J contains a single key i, and if so, retrieves (i,C_i). Note that a single key i can be obtained from a long stream of multiple keys if $\forall j\neq i$, $C_j=0$ at the end.

Strict Bin Sketch ($\mathrm{BS_s}$). For strict turnstile streams we use the *Strict Bin Sketch* ($\mathrm{BS_s}$) [9,10]. Given $\{(i_j,v_j)\}_{j\in J}$, $\mathrm{BS_s}$ consists of 3 counters: $T_1=\sum_{i\in I}C_i$, $T_2=\sum_{i\in I}C_i\cdot i$, and $T_3=\sum_{i\in I}C_i\cdot i^2$ where $I=\{i\colon\exists j\in J\ i_j=i\}$. For each (i,v), $\mathrm{BS_s}$ is updated by: $T_1\leftarrow T_1+v$, $T_2\leftarrow T_2+v\cdot i$ and $T_3\leftarrow T_3+v\cdot i^2$. $\mathrm{BS_s}$ holds a single element $\Leftrightarrow(T_1\neq 0\wedge T_1\cdot T_3=T_2^2)$. This element is (i,C_i), where $i=T_2/T_1$ and $C_i=T_1$. $\mathrm{BS_s}$ is empty $\Leftrightarrow T_1=0$. $\mathrm{BS_s}$ uses $O(\log(ur))$ bits of space.

Non-strict Bin Sketch ($\mathrm{BS_{ns}}$). $\mathrm{BS_s}$ cannot be used in non-strict streams since it might not distinguish between three or more elements and a single element when there are negative cumulative counts. A previous attempt to solve the problem in non-strict streams used a deterministic structure [5], which held the binary representation of the elements. However, it falsely identifies multiple elements as a single element on some inputs.[1] We provide a generalization to Bin Sketch and adjust it to non-strict streams, using the following new sketch.

Lemma 3. *Let $T=\sum_i C_i h(i)$ for $h\in_R\mathcal{H}_t(U,[q])$ maintain a sketch of substream J. Assume J contains at most $t-1$ distinct keys. Then $\forall(i',C_{i'})$, if $(i',C_{i'})$ is not the single element in J, $\Pr_{h\in\mathcal{H}_t}[T=C_{i'}h(i')]\leq 1/q$.*

Proof. If T is not a sketch of $(i',C_{i'})$ only, then $T'=T-C_{i'}h(i')$ is a sketch of 1 to t elements. The hashes of those elements are independent because h is t-wise independent. Therefore T' is a linear combination of independent uniformly distributed elements in the range $[q]$, and thus $\Pr_{h\in\mathcal{H}_t}[T'=0]\leq 1/q$. □

The *Non-strict Bin Sketch* ($\mathrm{BS_{ns}}$) consists of the counters T_1, T_2 and T as defined in Lemma 3, which are updated per each (i,v) inserted to $\mathrm{BS_{ns}}$. Updating T by $T\leftarrow T+v\cdot h(i)$ requires evaluation of $h\in\mathcal{H}_t$ in $O(\log^2 t)$ operations. Thus the space of $\mathrm{BS_{ns}}$ is $O(\log(urq))$ bits, and its insertion time is $O(\log^2 t)$. When there are at most $t-1$ elements in $\mathrm{BS_{ns}}$, the probability of falsely identifying a single element in $\mathrm{BS_{ns}}$ is at most $1/q$.

BS is placed in each cell in each of the arrays of our data structure. It identifies *collisions*, which occur when more than one element is mapped to a cell. $\mathrm{BS_{ns}}$ is used in the non-strict ϵ-FRS (see Section 4.5) to bound the probability of false identification in collisions of a small number of elements.

4.4. Full Recovery Structure (FRS)

In this section we present the Full Recovery Structure (FRS) that can be placed in each of the levels (see Fig. 2). FRS is a K-elements recovery structure that extracts the K elements it contains with probability $1-\delta$. Since it uses only pairwise independent hash functions, it can be easily implemented. We refer to the entire algorithm with FRS in each level as the FRS algorithm, and summarize its properties in the following theorem.

Theorem 1. *Given a required sample size $K=\Omega(\log\frac{1}{\delta})$ and $\delta\in(0,1)$, FRS sampling algorithm generates a $\Theta(\log\frac{1}{\delta})$-wise independent sample S with $1-\delta$ success probability. The sample size is $K\leq|S|\leq\tilde{K}$, where $\tilde{K}=\Theta(K)$. For both strict and non-strict*

[1] For example, for every set $\{(2i,1),(2i+1,-1),(2i+2,-1),(2i+3,1)\}$, the structure is zeroed and any additional element $(i',C_{i'})$ will be identified as a single element.

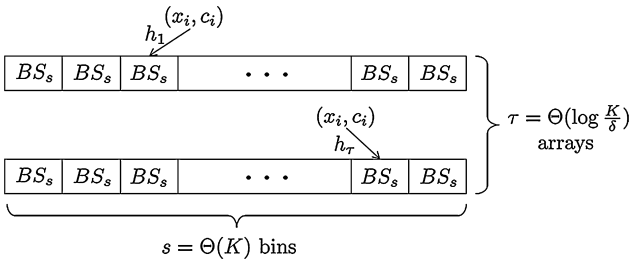

Fig. 2. FRS schema.

data streams FRS *uses* $O(K \log \frac{K}{\delta} \log(ur) \log(u))$ *bits of space,* $O(\log \frac{K}{\delta})$ *update time per element,* $O(\log \frac{K}{\delta} \log u)$ *random bits and* $O(K \log \frac{K}{\delta})$ *time to extract the sample* S.

FRS is composed of $\tau = O(\log \frac{K}{\delta})$ arrays of size $s = O(K)$ with an instance of BS_s in each cell (also called *bin*), in each of the τ arrays. Let $h_1 \dots h_\tau \in_R \mathcal{H}_2(U, [s])$ be τ hash functions chosen independently. For each array $a \in [\tau]$, element (i, v) is mapped to bin $h_a(i)$ in a. The same set of functions $h_1 \dots h_\tau$ can be used in the instances of FRS in all levels. A construction of FRS for strict streams appears in [10]. We describe it briefly and then show our generalization to non-strict streams.

We denote FRS for at most $\tilde{K}$ elements by $\text{FRS}_{\tilde{K}}$. For strict streams, let $\text{FRS}_{\tilde{K}}$ have $\tau = \log \frac{\tilde{K}}{\delta}$ arrays of size $s = 2\tilde{K}$. Let $\text{FRS}(X)$ denote the state of FRS structure after insertion of elements X. Then for each $i \in X$, there is a bin in $\text{FRS}(X)$ in which i is isolated with probability $1 - \delta$. In strict streams BS_s identifies isolated elements, and thus we can extract X from $\text{FRS}(X)$ with probability $1 - \delta$.

Non-strict FRS. We present our generalization of FRS to non-strict streams. We use BS_s to detect collisions and identify elements, and add to our sample only elements that are consistently identified in multiple arrays.

Lemma 4. *Let* $\text{FRS}_{\tilde{K}}$ *have* $\tau = 5 \log \frac{\tilde{K}}{\delta}$ *arrays of size* $s = 8\tilde{K}$. *For non-strict streams, there is a recovery procedure* $rec_{\text{ns}}: \text{FRS}(X) \to S$ *such that for each elements set* X, *where* $|X| \le \tilde{K}$, $\Pr[rec_{\text{ns}}(\text{FRS}_{\tilde{K}}(X)) = X] \ge 1 - \delta$.

Proof. rec_{ns} works as follows. We extract a set $\mathcal{A}$ of candidate elements from all BS_ss with $(T_1 \neq 0 \wedge T_1 \cdot T_3 = T_2^2)$ in the first $\log \frac{\tilde{K}}{\delta}$ arrays of $\text{FRS}_{\tilde{K}}(X)$. Similar to the argument in strict streams, $\Pr[X \subseteq \mathcal{A}] \ge 1 - \delta/2$ (here we increase the arrays size which reduces the probability of a collision). Thus, $\mathcal{A}$ contains X, and falsely detected elements as a result of collisions, and overall $|\mathcal{A}| \le \tilde{K} \log \frac{\tilde{K}}{\delta}$.

For $(i, C_i) \in \mathcal{A}$, let n_i be the number of arrays in which (i, C_i) is identified in the $\tau' = 4 \log \frac{\tilde{K}}{\delta}$ remaining arrays of $\text{FRS}_{\tilde{K}}(X)$. The sample is $S = \{(i, C_i) \in \mathcal{A} : n_i \ge \tau'/2\}$. The errors that might occur are: (1) $(i, C_i) \in X \wedge (i, C_i) \notin S$, (2) $(i, C_i) \notin X \wedge (i, C_i) \in S$.

(1) For each $(i, C_i) \in X$ and array a, $\Pr[i$ collides with another element in $a] < \frac{\tilde{K}}{s} = \frac{1}{8}$. The hash functions of the arrays are independent and hence $\forall (i, C_i) \in X$, $\Pr[n_i < \tau'/2] \le \binom{\tau'}{\tau'/2} (\frac{1}{8})^{\tau'/2} < (\frac{\delta}{\tilde{K}})^2$. Thus $\Pr[\exists (i, C_i) \in X : n_i < \tau'/2] < \tilde{K} \cdot (\frac{\delta}{\tilde{K}})^2 < \frac{\delta}{4}$.

(2) $(i, C_i) \notin X$ can be identified only as a result of a collision in its bins. There are collisions in its bins only if other elements are mapped there. Hence $\forall (i, C_i) \notin X$, $\Pr[n_i \ge \tau'/2] \le (\frac{\delta}{\tilde{K}})^2$, and $\Pr[\exists (i, C_i) \in \mathcal{A} : i \notin X, n_i \ge \tau'/2] \le \tilde{K} \log \frac{\tilde{K}}{\delta} \cdot (\frac{\delta}{\tilde{K}})^2 < \frac{\delta}{4}$. □

4.5. ϵ-Full Recovery Structure (ϵ-FRS)

In this section we present the ϵ-Full Recovery Structure (ϵ-FRS), our efficient K-elements recovery structure (see Fig. 3). ϵ-FRS requires only $O((\log \log \frac{K}{\delta})^2)$ processing time per element, and enables us to recover almost all elements inserted to it with probability $1 - \delta$. We refer to the entire algorithm with ϵ-FRS placed in each of the levels as ϵ-FRS algorithm, and summarize it to the following theorem.

Theorem 2. *Given a required sample size* $K = \Omega(\frac{1}{\epsilon} \log \frac{1}{\delta})$, *for* $\delta \in (0, 1)$ *and* $\epsilon \in (0, 1)$, ϵ-FRS *sampling algorithm generates a* (t, ϵ)*-partial sample* S *for* $t = \Theta(\log \frac{1}{\delta})$ *with* $1 - \delta$ *success probability. The sample size is* $(1 - \epsilon)K \le |S| \le \tilde{K}$, *where* $\tilde{K} = \Theta(K)$. *For both strict and non-strict data streams* ϵ-FRS *requires* $O((\log \log \frac{K}{\delta})^2)$ *update time per element,* $O(\log \frac{K}{\delta} \log u)$ *random bits and* $O(K)$ *time to extract the sample* S. *The space is* $O((K \log(ur) + \log^2 \frac{1}{\delta}) \log(u))$ *bits for strict data streams and* $O(K \log(\frac{ur}{\delta}) \log(u))$ *bits for non-strict streams.*

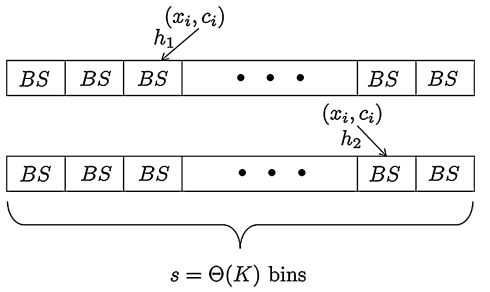

Fig. 3. ϵ-FRS schema.

ϵ-FRS is composed of $\tau = 2$ arrays of size $s = O(K)$, where in each bin of each array we keep an instance of $\mathrm{BS_s}$ or $\mathrm{BS_{ns}}$ according to the input data stream. As in FRS, each input element is mapped to one bin in each of the arrays, using two independent functions $h_1, h_2 \in_R \mathcal{H}_t(U, [s])$ for $t = \Theta(\log \frac{K}{\delta})$.

Let ϵ-FRS(X) denote an ϵ-FRS structure after insertion of a set of elements X. A *fail set* $F \subseteq X$ with respect to ϵ-FRS(X), is a set of f elements, such that each element in F collides with other elements from F in both its bins. The elements in a fail set F cannot be extracted from ϵ-FRS(X).

We bound $\Pr[\exists$ fail set F in ϵ-FRS$(X), |F| \geq t]$ using the following (revised) lemma of Pagh and Pagh [25]. The analysis of this lemma is similar to checking the size of the 2-core of the graph that represents ϵ-FRS.

Lemma 5. *(See [25], Lemma 3.4.) For two functions $i_1, i_2 : U \to [R]$ and a set $S \subseteq U$, let $G(i_1, i_2, S) = (A, B, E)$ be the bipartite graph that has left vertex set $A = \{a_1, \ldots, a_R\}$, right vertex set $B = \{b_1, \ldots, b_R\}$ and edge set $E = \{e_x \mid x \in S\}$, where $e_x = (a_{i_1(x)}, b_{i_2(x)})$. For each set S of size n, and for $i_1, i_2 : U \to [4n]$ chosen at random from a family that is t-wise independent on S, $t \geq 32$, the probability that the fail set F of the graph $G(i_1, i_2, S)$ has size at least t is $n/2^{\Omega(t)}$.*

Corollary 1. *Let ϵ-FRS$_{\tilde{K}}$ have 2 arrays of size $s = 4\tilde{K}$. Let the input elements be mapped to bins by $h_1, h_2 \in \mathcal{H}_t$ for $t = c \log \frac{\tilde{K}}{\delta}$, constant c and $t \geq 32$. Then for each elements set X, where $|X| \leq \tilde{K}$, $\Pr[\exists$ fail set F in ϵ-FRS$_{\tilde{K}}(X), |F| \geq t] < \delta$.*

Proof. More elements in ϵ-FRS increase the probability for a fail set of some fixed predefined size. The probability is therefore bounded by $\tilde{K}/2^{c' c \log \frac{\tilde{K}}{\delta}} \leq \delta$ for constants c, c'. □

Lemma 6. *Let $F_{max} \subseteq X$ be the maximal fail set of X with respect to ϵ-FRS(X). I.e., F_{max} is the union of all fail sets in X. Assume ϵ-FRS contains in each bin a sketch of the substream of elements mapped to that bin. This sketch can identify if the substream contains a single element (i, C_i) and can extract that single element. Then given ϵ-FRS(X), the following graph peeling algorithm returns the output sample $S = \{(i, C_i) : i \in X \wedge i \notin F_{max}\}$ in $O(K)$ time.*

1. Initialize the output sample $S = \emptyset$ and a queue $Q = \emptyset$.
2. Scan the arrays in ϵ-FRS. For each bin b, if it holds a single element, *Enqueue*(Q, b).
3. While $Q \neq \emptyset$:
   3.1. $b \leftarrow$ *Dequeue*(Q).
   3.2. If b holds a single element:
      3.2.1. Extract the element (i, C_i) from the sketch in b.
      3.2.2. Add it to the output sample $S = S \cup \{(i, C_i)\}$.
      3.2.3. Subtract (i, C_i) from the sketch in $\tilde{b}$, the other bin i is hashed to.
      3.2.4. *Enqueue*$(Q, \tilde{b})$.
4. Return S.

Proof. *Correctness.* Let UE be the set of unidentified elements at the end of the algorithm's run. If UE is not a fail set, then after removing all other elements from ϵ-FRS, there is $x \in UE$ isolated in bin b. If x was isolated prior to the first scan over all bins, bin b would have been identified in the first scan. If x became isolated during the recovery process, then all other elements in b were identified beforehand. When the last of them was identified and removed from b, x became isolated, and b was inserted to the queue. Each bin inserted to the queue is removed from the queue before the process ends, and hence when b was removed, x was identified. Identifying x results in a smaller set of unidentified elements. The identification process can proceed until all elements are identified or none of them are isolated, i.e. they all belong to a fail set.

Runtime. The initial scan takes $O(K)$ time, linear in the number of bins in ϵ-FRS. $O(K)$ bins are inserted to Q in the process, since a bin is inserted only when an element is extracted and added to the sample. Thus, apart from identifying per each element its other bin, all other operations take a total of $O(K)$ time.

In order to identify the other bin an element is hashed to, we use an additional counter T_{loc} in each bin sketch. Let h_1, h_2 map the elements to the bins in the two arrays. When element (i, v) is inserted to a bin in array $a \in \{1, 2\}$, we perform $T_{loc} \leftarrow T_{loc} + v \cdot h_{3-a}(i)$. The space and update time required by the algorithm remain of the same order, since the update costs an additional $O(1)$ time and T_{loc} takes $O(\log(ur))$ bits.

If the sketch in a bin in array a contains a single element (i, C_i), then $T_{loc} = C_i h_{3-a}(i)$. Thus, if (i, C_i) is extracted from array a we obtain its location $h_{3-a}(i)$ in array $3 - a$. Thus, the recovery algorithm takes $O(K)$ operations for all $O(K)$ elements. □

For every elements set X, $K \leq |X| \leq \tilde{K}$, $\tilde{K} = \Theta(K)$, in order to recover from ϵ-FRS$_{\tilde{K}}(X)$ all but $\epsilon|X|$ elements we require $K = \Omega(\frac{1}{\epsilon}\log\frac{1}{\delta})$. If K is smaller, we recover all but at most $O(\max\{\epsilon|X|, f\})$ elements, where $f = O(\log\frac{K}{\delta})$.

Strict ϵ-FRS. Plugging in BS$_s$ as the sketch in each bin in ϵ-FRS, the algorithm in Lemma 6 is a recovery procedure for strict streams.

Lemma 7. *Let ϵ-FRS$_{\tilde{K}}$ have 2 arrays of size $s = 4\tilde{K}$. Let the input elements be mapped to bins by $h_1, h_2 \in \mathcal{H}_t$ for $t = c\log\frac{\tilde{K}}{\delta}$, constant c and $t \geq 32$. Let each bin contain* BS$_s$*. Then for strict turnstile streams the recovery process returns all items except at most t with probability $1 - \delta$.*

Proof. According to Corollary 1, $|F_{max}| < t$ with probability $1 - \delta$. For strict streams, BS$_s$ identifies with probability 1 if a bin contains a single element and can extract that single element. Hence the recovery algorithm in Lemma 6 can be applied, and all elements that are not in F_{max} are recovered. □

Non-strict ϵ-FRS. In non-strict turnstile streams BS$_s$ cannot determine if a bin contains a single element. Hence, in the non-strict ϵ-FRS construction, we replace BS$_s$s with BS$_{ns}$s. In each of the BS$_{ns}$ we use a hash function $h \in_R \mathcal{H}_{t'}(U, [q])$ for $q = \Theta(\frac{K}{\delta})$ and $t' = \Theta(\log\frac{K}{\delta})$. The same h can be used in all BS$_{ns}$s. The analysis of the non-strict ϵ-FRS is as follows.

Let n denote the number of elements in BS$_{ns}$. Recall that if $n = 1$, the single element BS$_{ns}$ contains can be extracted successfully; if $1 < n < t'$, an event called a *small collision*, the probability of an error is at most $1/q$; if $n \geq t'$, an event called a *large collision*, we do not have a guarantee on the probability of an error. The recovery procedure in Lemma 6 can be used also for non-strict streams, once we prove that with probability $1 - \delta$ we do not have any errors.

Lemma 8. *Let ϵ-FRS$_{\tilde{K}}$ have 2 arrays of size $s = 4\tilde{K}$. Let the input elements be mapped to bins using the functions $h_1, h_2 \in_R \mathcal{H}_t$ for $t = 2\log\frac{\tilde{K}}{\delta}$. Let each bin contain* BS$_{ns}$ *with $q = \frac{4\tilde{K}}{\delta}$ and $t' = t$. Then for each set of elements X, where $|X| \leq \tilde{K}$, there are no false detections during the entire recovery process with probability at least $1 - \delta$.*

Proof. We begin by bounding the probability of a large collision in ϵ-FRS. Let ϵ-FRS have $|X| \leq \tilde{K}$ elements. Let $\mathcal{E}_b$ indicate that a large collision in bin b occurred. Since $t = t'$, every t' elements that participated in a large collision are mapped there independently. Thus, $\Pr[\mathcal{E}_b] \leq \binom{X}{t'}(\frac{1}{s})^{t'} \leq (\frac{eX}{t'})^{t'}(\frac{1}{4\tilde{K}})^{t'} \leq (\frac{e}{4t'})^{t'} < (\frac{\delta}{\tilde{K}})^2$, and $\Pr[\exists b, \mathcal{E}_b] \leq 8\tilde{K} \cdot (\frac{\delta}{\tilde{K}})^2 < \delta/2$.

We now bound the probability of a small collision. At most $2\tilde{K}$ bins with collisions are inspected during the recovery process. Hence, the probability of detecting at least one false element due to a small collision is at most $2\tilde{K}\frac{1}{q} = \delta/2$. We get that the total probability of false detections either from a small collision or from a large collision is at most δ. □

Recovering small sets. ϵ-FRS is superior to FRS in almost every aspect. Its disadvantage is that since the fail set's elements cannot be recovered, in order to extract all but $\epsilon|X|$ elements, ϵ-FRS requires $K = \Omega(\frac{1}{\epsilon}\log\frac{1}{\delta})$, where FRS requires only $K = \Omega(\log\frac{1}{\delta})$. ϵ-FRS is also not suitable when there are $\Omega(\log\frac{1}{\delta})$ distinct keys with a non-zero C at the time of the sample extraction. As a solution to the problem of sampling small sample sizes, we construct the *Small Recovery Structure* (SRS). Combining SRS and ϵ-FRS results in a sampling algorithm with $O((\log\log\frac{K}{\delta})^2)$ processing time per stream element, that supports sample sizes $K = \Omega(\log\frac{1}{\delta})$ instead of $K = \Omega(\frac{1}{\epsilon}\log\frac{1}{\delta})$, and even smaller sample sizes when the number of distinct stream elements is as small as $O(\log\frac{1}{\delta})$.

SRS with parameter n, denoted by SRS$_n$, is a deterministic structure that keeps a sketch of substream J, and recovers all of the elements in J if at the recovery time $L_0(J) \leq n$. It can be used for both strict and non-strict streams. Since the update time of SRS$_n$ depends on n, we use small n values.

For a substream of elements $\{(i_j, v_j)\}_{j \in J}$, SRS$_n$ keeps a sketch of substream J defined as follows. Let $GF(p)$ be a finite field for a prime $p > \max(u, 2r + 1)$ where u is the size of the universe of keys and $[-r, r]$ is the range of the total counts. SRS$_n$ consists of $2n + 1$ counters defined as follows:

$$T_k = \sum_{i \in I} C_i \cdot i^k, \quad k = 0 \ldots 2n,$$

where $I = \{i : \exists j \in J, i_j = i\}$, and the calculations are performed over $GF(p)$.

SRS design and analysis are based on Reed–Solomon (RS) codes [27]. Its properties are summarized in the following lemma.

Lemma 9. *SRS_n that keeps a sketch of stream I, can recover all I's elements if I contains at most n distinct keys. It uses $O(n\log(ur))$ bits of space, and is updated in $O(\log^2 n)$ operations amortized.*

Proof. Assume that the counters $\{T_k\}_{k=0}^{2n}$ are updated by substream I with $L_0(I)=\mu\le n$. Let $\{x_j\}_{j=1}^{\mu}$ be the elements' keys and $\{C_{x_j}\}_{j=1}^{\mu}$ be their corresponding total counts. The counters can be written as: $T_k=\sum_{j=1}^{\mu} C_{x_j}\cdot {x_j}^k$, $k=0\ldots 2n$. Then the keys and their total counts can be recovered similarly to the syndrome decoding of Reed–Solomon codes over $GF(p)$, as done in [2]. First, the error locator polynomial $\Lambda(Z)=\prod_{j=1}^{\mu}(1-Zx_j)$ is calculated using Berlekamp–Massey. The input syndromes are the $2n$ counters $\{T_k\}_{k=1}^{2n}$. Then $\Lambda(Z)$ is factored, and the inverse of its roots are $\{x_j\}_{j=1}^{\mu}$—the keys of the elements in the stream. At last, the total counts of the elements $\{C_{x_j}\}_{j=1}^{\mu}$ are found by multiplication with the inverse transpose Vandermonde matrix.

Looking at the counters' updates in batches, the update procedure can be performed as follows. Let $\{(i_j,C_{i_j})\}_{j=1}^{2n+1}$ be a batch of elements with distinct keys that should be added to $\{T_k\}_{k=0}^{2n}$. The keys in each batch are kept distinct by sorting the elements in $O(n\log n)$ operations and uniting elements with the same key. The updates are $T_k\leftarrow T_k+U_k$ for $j=0\ldots 2n$ where

$$\begin{pmatrix} i_1^0 & i_2^0 & \cdots & i_{2n+1}^0\\ i_1^1 & i_2^1 & \cdots & i_{2n+1}^1\\ \vdots & \vdots & \ddots & \vdots\\ i_1^{2n} & i_2^{2n} & \cdots & i_{2n+1}^{2n}\end{pmatrix}\begin{pmatrix} C_{i_1}\\ C_{i_2}\\ \vdots\\ C_{i_{2n+1}}\end{pmatrix}=\begin{pmatrix} U_0\\ U_1\\ \vdots\\ U_{2n}\end{pmatrix}.$$

This update of the $2n+1$ counters with $O(n)$ elements is a multiplication of a transposed Vandermonde matrix by a vector. This multiplication takes $O(n\log^2 n)$ operations [1], i.e. $O(\log^2 n)$ operations per stream element amortized.

A counter's maximal value is $p>\max(u,2r+1)$. Taking $p=O(u+r)$, $O(n\log(ur))$ bits are required for $O(n)$ counters in SRS_n. □

SRS can be combined in our sampling algorithm as follows. In each level an instance of $\mathrm{SRS}_{\Theta(\log(K/\delta))}$ is placed, and it is updated in parallel to ϵ-FRS. The update takes $O((\log\log\frac{K}{\delta})^2)$ amortized operations per stream element. When a sample is extracted from ϵ-FRS, the extracted elements are removed from SRS (removing is the same as updating). When a fail set F is reached, all its $f=O(\log\frac{K}{\delta})$ elements are extracted from $\mathrm{SRS}_{\Theta(\log(K/\delta))}$. Note that using SRS increases the recovery time because of the polynomial factorization phase in the syndrome decoding process. Using this construction, the sample generated is t-wise independent for $t=O(\log\frac{1}{\delta})$, instead of a (t,ϵ)-partial sample, and sample sizes $K=\Omega(\log\frac{1}{\delta})$ are supported.

5. Applications and extensions

Inverse distribution. The samples generated by FRS and ϵ-FRS can be used to approximate various inverse distribution queries within an additive ϵ error and with $1-\delta$ success probability. One example is *inverse point queries*, which return $f^{-1}(C)$ for a given frequency C.

Lemma 10. *Let S be a (t,ϵ')-partial sample of size $|S|=\Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ for $\epsilon\in(0,1)$, $\epsilon'=\Theta(\epsilon)$, $\delta\in(0,1)$, and $t=\Omega(\log\frac{1}{\delta})$. For a given frequency i let the estimator of $f^{-1}(i)$ be $f_S^{-1}(i)=\frac{|\{k\in S:C_k=i\}|}{|S|}$. Then $\Pr[|f^{-1}(i)-f_S^{-1}(i)|<\epsilon]\ge 1-\delta$.*

Proof. First, we prove that $f_S^{-1}(i)$ is an ϵ-approximation to $f^{-1}(i)$ when all elements are recovered, i.e. when FRS is used. Then we relax the assumption that all elements are recovered, and prove that we get an ϵ-approximation also when we use ϵ-FRS. Thus, an ϵ-approximation is provided with $1-\delta$ success probability when using both our sampling algorithms.

The elements recovered are from a specific level l^* that depends on L_0. For each value k where $C_k\neq 0$, we define an indicator random variable Y_k. $Y_k=1$ if (k,C_k) is mapped to level l^*. We denote $p_{l^*}=\Pr[Y_k=1]=(\frac{1}{2})^{l^*+1}$.

Let $Y=\sum_k Y_k$ be the number of elements in level l^*. For now assume that all elements in the level are recovered, i.e. $Y=|S|$. Later we relax this assumption. $E[Y]$ is the expected sample size obtained from level l^*. Thus $E[|S|]=E[Y]=p_{l^*}\cdot L_0$. We choose l^* as a level with $\Theta(K)$ elements, i.e. $E[Y]=\Theta(K)=\Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$.

$\{Y_k\}$ are t-wise independent, where $t=\Theta(\log\frac{1}{\delta})$. $E[Y]=\Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$. From Lemma 1 we obtain $\Pr[|Y-E[Y]|>\epsilon' E[Y]]<\delta'$, for $\epsilon'=\Theta(\epsilon)$ and $\delta'=\Theta(\delta)$ that will be determined later. Thus, with probability $1-\delta'$,

$$\big(1-\epsilon'\big)E\big[|S|\big]\le|S|\le\big(1+\epsilon'\big)E\big[|S|\big].$$

Let $F_i = \sum_{k:C_k=i} Y_k$ be the number of elements in S with frequency i. $E[F_i] = |\{k: C_k = i\}| \cdot p_i^* = f^{-1}(i) \cdot E[|S|]$. We get $f^{-1}(i) = \frac{E[F_i]}{E[|S|]}$. If all elements in the level are recovered, $f_S^{-1}(i) = \frac{F_i}{|S|}$.

Hence we would like to prove:

$$\Pr\left[\left| f^{-1}(i) - \frac{F_i}{|S|}\right| \geq \epsilon\right] \leq \delta \qquad (*)$$

$\frac{E[F_i]}{|S|}$ is an ϵ'-approximation to $\frac{E[F_i]}{E[|S|]}$, i.e. $\Pr[|f^{-1}(i) - \frac{E[F_i]}{|S|}| \geq \epsilon'] \leq \delta'$ since

$$\frac{E[F_i]}{E[|S|]} - \epsilon' \leq \frac{E[F_i]}{(1+\epsilon')E[|S|]} \leq \frac{E[F_i]}{|S|} \leq \frac{E[F_i]}{(1-\epsilon')E[|S|]} \leq \frac{E[F_i]}{E[|S|]} + \epsilon'$$

with probability $1-\delta'$ when $f^{-1}(i) = \frac{E[F_i]}{E[|S|]} \leq 1-\epsilon'$.

$\frac{E[F_i]}{|S|}$ is an ϵ'-approximation to $\frac{F_i}{|S|}$ since

$$\begin{aligned}\Pr\left[\left|\frac{E[F_i]}{|S|} - \frac{F_i}{|S|}\right| > \epsilon'\right] &= \Pr\left[\left|E[F_i] - F_i\right| > \epsilon'|S|\right] \\ &< \Pr\left[\left|E[F_i] - F_i\right| > \frac{\epsilon'}{2}E\left[|S|\right]\right] \\ &= \Pr\left[\left|E[F_i] - F_i\right| > \frac{\epsilon'}{2f^{-1}(i)}E[F_i]\right] < \delta'.\end{aligned}$$

The last inequality follows from Lemma 1. Lemma 1 holds since F_i is the sum of t-wise independent indicator variables $\{Y_k\}_{k:C_k=i}$, where $t = \Theta(\log \frac{1}{\delta})$.

$E[F_i] = f^{-1}(i) \cdot E[|S|] = \Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta} \cdot f^{-1}(i)) > \Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta} \cdot f^{-1}(i)^2)$. Thus we get (*), for a constant c, $2\epsilon'$-additive error and $1-2\delta'$ success probability.

Now we relax the assumption that all elements in the level are recovered. The maximal bias occurs when $\epsilon'|S|$ elements are not recovered, and all unrecovered elements have frequency i. I.e. $|\{k \in S: C_k = i\}| = F_i \pm \epsilon'|S|$. Thus, there is an additional additive error of ϵ'. Setting $\epsilon' = \epsilon/3$, $\delta' = \delta/3$ completes the proof. □

Additional variations of inverse distribution queries such as inverse range queries and inverse heavy hitters were previously suggested and approximated using a sample in [5]. These queries can be approximated by the samples generated by FRS and ϵ-FRS as well. For the rest of the section, assume S is a (t,ϵ')-partial sample of size $|S| = \Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ for $\epsilon \in (0,1)$, $\epsilon' = \Theta(\epsilon)$, $\delta \in (0,1)$, and $t = \Omega(\log\frac{1}{\delta})$.

Corollary 2. Inverse range queries *get an input range* $[i,j]$ *and return the fraction of values with frequencies in the range* $[i,j]$, $\frac{|\{k: i \leq C_k \leq j\}|}{|\{k: C_k \neq 0\}|}$. *The estimator* $\frac{|\{k \in S: i \leq C_k \leq j\}|}{|S|}$ *provides an additive* ϵ*-approximation to inverse range queries for range* $[i,j]$ *with probability at least* $1-\delta$.

Proof. Following the notation of the proof of Lemma 10, let $F_i = \sum_{k:C_k=i} Y_k$ be the number of elements in S with frequencies in range $[i,j]$. The same claims follow. The maximal bias occurs when the ϵ' fraction of unrecovered elements all have frequencies in the range $[i,j]$. □

Corollary 3. Inverse heavy hitters queries *get an input threshold* ϕ *and return* $I = \{i: f^{-1}(i) \geq \phi\}$. *The set* $I_S = \{i: f_S^{-1}(i) \geq \phi\}$ *is an* ϵ*-approximation to* I *with probability* $1-\delta$: $\forall i \in I$ *if* $f^{-1}(i) \geq \phi + \epsilon$ *then* $i \in I_S$ *and* $\forall i$ *if* $f^{-1}(i) < \phi - \epsilon$ *then* $i \notin I_S$.

Proof. Similar to [5] and derives from the approximation of the frequencies in Lemma 10. □

Distributed computing. Our sampling structure is *strongly history independent* [21,24] since it supports additions and deletions and is oblivious to their order in the stream. Hence, if we use the same random bits, we can split the stream into multiple substreams, and process each substream with an instance of our sampling structure. Then we can combine the data in the structures by summing all BSs, and generate a unified sample.

Difference. Given two streams S_1, S_2, their *difference stream* is $S' = \{(i, C_i): C_i = C_i^{S_1} - C_i^{S_2}, C_i \neq 0\}$. If S_1, S_2 are represented using the same random bits, a structure for S' can be generated by subtracting all BSs of S_2 from S_1, and extracting a sample of S' from this structure. Note that the difference of two strict streams might be a non-strict stream. Hence, our structures for the non-strict model are useful for both non-strict input streams and for sampling the difference.

References

[1] J.F. Canny, E. Kaltofen, L. Yagati, Solving systems of nonlinear polynomial equations faster, in: Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, ACM, 1989, pp. 121–128.
[2] R. Clifford, K. Efremenko, E. Porat, A. Rothschild, From coding theory to efficient pattern matching, in: SODA, 2009, pp. 778–784.
[3] E. Cohen, G. Cormode, N.G. Duffield, Don't let the negatives bring you down: sampling from streams of signed updates, in: SIGMETRICS, 2012, pp. 343–354.
[4] G. Cormode, M. Datar, P. Indyk, S. Muthukrishnan, Comparing data streams using hamming norms (how to zero in), IEEE Trans. Knowl. Data Eng. 15 (3) (2003) 529–540.
[5] G. Cormode, S. Muthukrishnan, I. Rozenbaum, Summarizing and mining inverse distributions on data streams via dynamic inverse sampling, in: VLDB, 2005, pp. 25–36.
[6] M. Datar, S. Muthukrishnan, Estimating rarity and similarity over data stream windows, in: ESA, 2002, pp. 323–334.
[7] D.P. Dubhashi, D. Ranjan, Balls and bins: a study in negative dependence, Random Structures Algorithms 13 (2) (1998) 99–124.
[8] G. Frahling, P. Indyk, C. Sohler, Sampling in dynamic data streams and applications, in: Symposium on Computational Geometry, 2005, pp. 142–149.
[9] G. Frahling, P. Indyk, C. Sohler, Sampling in dynamic data streams and applications, Internat. J. Comput. Geom. Appl. 18 (1/2) (2008) 3–28.
[10] S. Ganguly, Counting distinct items over update streams, Theoret. Comput. Sci. 378 (3) (2007) 211–222.
[11] J. von zur Gathen, J. Gerhard, Modern Computer Algebra, Cambridge University Press, New York, NY, USA, 1999.
[12] R. Gemulla, W. Lehner, P.J. Haas, A dip in the reservoir: maintaining sample synopses of evolving datasets, in: VLDB, 2006, pp. 595–606.
[13] R. Gemulla, W. Lehner, P.J. Haas, Maintaining Bernoulli samples over evolving multisets, in: PODS, 2007, pp. 93–102.
[14] P.B. Gibbons, Distinct sampling for highly-accurate answers to distinct values queries and event reports, in: VLDB, 2001, pp. 541–550.
[15] A.C. Gilbert, Y. Li, E. Porat, M.J. Strauss, Approximate sparse recovery: optimizing time and measurements, in: STOC, 2010, pp. 475–484.
[16] M.T. Goodrich, M. Mitzenmacher, Invertible Bloom lookup tables, in: 2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton), IEEE, 2011, pp. 792–799.
[17] P. Indyk, A small approximately min-wise independent family of hash functions, J. Algorithms 38 (1) (2001) 84–90.
[18] H. Jowhari, M. Saglam, G. Tardos, Tight bounds for L_p samplers, finding duplicates in streams, and related problems, in: PODS, 2011, pp. 49–58.
[19] D.M. Kane, J. Nelson, D.P. Woodruff, An optimal algorithm for the distinct elements problem, in: PODS, 2010, pp. 41–52.
[20] V. Karamcheti, D. Geiger, Z.M. Kedem, S. Muthukrishnan, Detecting malicious network traffic using inverse distributions of packet contents, in: MineNet, 2005, pp. 165–170.
[21] D. Micciancio, Oblivious data structures: applications to cryptography, in: STOC, 1997, pp. 456–464.
[22] M. Monemizadeh, D.P. Woodruff, 1-pass relative-error L_p-sampling with applications, in: SODA, 2010, pp. 1143–1160.
[23] S. Muthukrishnan, Data streams: algorithms and applications, Found. Trends Theor. Comput. Sci. 1 (2) (2005).
[24] M. Naor, V. Teague, Anti-persistence: history independent data structures, IACR Cryptology ePrint Archive 36, 2001.
[25] A. Pagh, R. Pagh, Uniform hashing in constant time and optimal space, SIAM J. Comput. 38 (1) (2008) 85–96.
[26] E. Porat, O. Lipsky, Improved sketching of hamming distance with error correcting, in: CPM, 2007, pp. 173–182.
[27] I. Reed, S. Golomb, Polynomial codes over certain finite fields, SIAM J. Sci. Comput. 8 (2) (1960) 300–304.
[28] Y. Tao, X. Lian, D. Papadias, M. Hadjieleftheriou, Random sampling for continuous streams with arbitrary updates, IEEE Trans. Knowl. Data Eng. 19 (1) (2007) 96–110.