Efficient Sampling of Non-strict Turnstile Data Streams

Neta Barkay, Ely Porat, and Bar Shalem

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
{netabarkay,porately,barshalem}@gmail.com

Abstract. We study the problem of generating a large sample from a data stream of elements (i, v), where the sample consists of pairs (i, C_i) for $C_i = \sum_{(i,v)\in \text{stream}} v$. 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 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 [22], the general representation of data streams, the data is a sequence of N elements $(i, v) \in U \times [-r, r]$, where $U = \{1, \ldots, u\}$ and $[-r, r] = \{-r, \ldots, r\}$. 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 $C_i \leftarrow C_i + v$.

The function $f\colon U \to [-r, r]$ 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 $S \subseteq \{(i, f'(i))\colon f(i) \neq 0\}$, where $f'(i) \in [(1-\epsilon)f(i), (1+\epsilon)f(i)]$. Note that we consider a value 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 $f^{-1}(C) = \frac{|\{i\colon C_i = C\}|}{|\{i\colon C_i \neq 0\}|}$ for $C \neq 0$,

L. Gąsieniec and F. Wolter (Eds.): FCT 2013, LNCS 8070, pp. 48–59, 2013.

i.e. the fraction of distinct values with a cumulative count C. In order to answer queries on f^{-1}, an *exact sample* $S \subseteq \{(i, f(i)) \colon f(i) \neq 0\}$ is required. To illustrate this, assume $f^{-1}(C) = \alpha$ for a fraction $\alpha \in (0, 1)$, and in the 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: Let $\boldsymbol{C}(t)$ be the state of $\boldsymbol{C}$ after processing the t'th element in the stream, and assume that $\forall t, i,\ C_i(t) \in [-r, r]$. 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 $S(t) \subseteq \{(i, C_i(t)) \colon C_i(t) \neq 0\}$. 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 [19] and geometric data streams [6,7]. For example, in [19] 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 [1,11]. Works on the inverse distribution include a restricted model where $\forall i,\ C_i \in \{0, 1\}$ [10,27], and minwise-hashing, which samples the set of values uniformly but does not support deletions [4]. The work in [12] supports only a few deletions.

Inverse distribution queries in streams with multiple deletions were supported in a work by Frahling et al. [6,7], who developed a solution for strict turnstile data streams and used it in geometric applications. Cormode et al. [3] 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 [21] and Jowhari et al. [17], however [17] lacked the time analysis of the sparse recovery.

Our Results. Previous works [3,6,7,17,21] 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 typical 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 typical values $\epsilon = 10^{-2}$ and $\delta = 10^{-6}$, the number of operations per stream element is multiplied by about 200,000. The structures of [3,6,7,17,21] 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 [13,25] but in our case there is no tail noise and we limit the amount of randomness. In comparison to IBLT [15], 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 more efficient algorithms in terms of the randomness that they use. Our algorithms do not require fully independent or min-wise independent hash functions or PRGs.
- We introduce the use of $\Theta(\log \frac{1}{\delta})$-wise independent hash functions to 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. We utilize a method of fast evaluation of hash functions, which reduces our processing time to $O((\log\log \frac{1}{\delta})^2)$. The traditional approach takes $O(\log \frac{1}{\delta})$ 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 prove that for $t = \Omega(\log \frac{1}{\delta})$ it is sufficient for typical stream queries. We show how to generate this sample 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

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
[6,7]	$K\log^2(\frac{ur}{\delta})$	$K\log u$	$K\text{polylog}u$	K	S
[7]	$K\log u\frac{1}{\epsilon}\log\frac{1}{\delta}\log(\frac{ur}{\epsilon})$	$K\frac{1}{\epsilon}\log\frac{1}{\delta}\log u$	$K\frac{1}{\epsilon}\log\frac{1}{\delta}\log u$	$K\frac{1}{\epsilon}\log\frac{1}{\delta}$	S
[3]	$K\log^2 u\log(ur)$	$K\log u$	$K\log u$	$K\log^2 u$	S
[3]	$K\log u\log\frac{1}{\delta}\log(ur)$	$K\log\frac{1}{\delta}$	–	$K\log u\log\frac{1}{\delta}$	NS
[17]	$K\log^2 u\log\frac{1}{\delta}$	K	$K\log u$	$K\log u\log\frac{1}{\delta}$	NS
[21]	$K\log^2 u\log(ur)$	$K\log^2 u$	$K\log^3 u$	$K\log u$	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 [3] S is the deterministic approach, NS is the probabilistic approach, and the hash function is fully independent. In [17] the extraction time assumes sparse recovery in linear time.

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 $D=\{(i,C_i)\colon C_i\neq 0\}$ be the set of values 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.

Applications typically require a sample of size $K=\Omega(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ for an ϵ-approximation with $1-\delta$ success probability. Our algorithms FRS and ϵ-FRS, support even smaller sample sizes, $\Omega(\log\frac{1}{\delta})$ and $\Omega(\frac{1}{\epsilon}\log\frac{1}{\delta})$ respectively. 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 Sect. 5).

2.2 Hash Functions

We denote by $\mathcal{H}_t(A, B)$ the family of t-wise independent hash functions $h\colon A \to B$, 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. In order to bound the error probability we use the Moment Inequality for high moments and the estimation of [16]. For hash evaluations we use the multipoint evaluation algorithm of a polynomial of degree less than t on t points in $O(t \log^2 t)$ operations [9]. Thus, evaluation of $h \in \mathcal{H}_t$ takes $O(\log^2 t)$ amortized time per element by batching t elements together.

3 Overview

In this section we provide an overall view of our sampling algorithm. At first, we insert each stream element (i, v) to a level in our main structure. The goal in mapping to levels is to have a set of elements from which a sample of size K can be extracted, regardless of the number of distinct values in the stream. In each level we store a recovery structure suitable for K elements, which is the core of our algorithm. We present two alternative structures, FRS and ϵ-FRS, each consisting of several arrays with a building block called Bin Sketch (BS) in each cell of the arrays. Each (i, v) mapped to a level l is inserted to the instance of FRS (ϵ-FRS) in l, and in each of the arrays in that instance, to one BS.

The sample S is composed of the elements extracted from FRS (or ϵ-FRS) in a single level. 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.

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 values in the stream. This structure is updated per each stream element, in parallel to the process described above.

To generate 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 X from l^*, or at least $(1-\epsilon)\,|X|$ of them, depending on the recovery structure, with probability $1 - \delta$. The elements extracted are the final output sample S.

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 $h_l(x) = \left\lfloor 2^l h(x)/u \right\rfloor$, 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 values in the stream (see Sect. 4.2), and select a level where $\Theta(K)$ elements are expected.

Lemma 1. *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. See the full version.

We extract the sample from level l^* as defined in Lemma 1. 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* $L_0 = |\{i \colon C_i \neq 0\}|$ [2], the number of distinct values in a stream with deletions, as follows. We use the structure of Kane et al. [18], 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 $\left|\{i \in [\tau] \colon B_i \text{ is a constant approximation to } \frac{L_0}{\tau}\}\right| > c\tau$ 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 [5]. 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 value i, and if so, retrieves (i, C_i). Note that a single value i can be obtained from a long stream of multiple values if $\forall j \neq i$, $C_j = 0$ at the end.

Strict Bin Sketch ($\mathbf{BS_s}$). For strict turnstile streams we use the *Strict Bin Sketch* ($\mathrm{BS_s}$) [7,8]. 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 ($\mathbf{BS_{ns}}$). $\mathrm{BS_s}$ 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 [3], 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 2. *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 values. 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 2, 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 Sect. 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. 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.

[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.

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 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 $\mathrm{BS_s}$ 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. $\forall$ 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 [8]. We describe it briefly and then show our generalization to non-strict streams.

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

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

Lemma 3. *Let* $\mathrm{FRS}_{\tilde{K}}$ *have* $\tau = 5 \log \frac{\tilde{K}}{\delta}$ *arrays of size* $s = 8\tilde{K}$*. For non-strict streams, there is a recovery procedure* $rec_{ns}\colon \mathrm{FRS}(X) \to S$ *such that for each elements set* X*, where* $|X| \leq \tilde{K}$*,* $\Pr[rec_{ns}(\mathrm{FRS}_{\tilde{K}}(X)) = X] \geq 1-\delta$*.*

Proof. rec_{ns} works as follows. We extract a set $\mathcal{A}$ of candidate elements from all $\mathrm{BS_s}$s with $(T_1 \neq 0 \wedge T_1 \cdot T_3 = T_2^2)$ in the first $\log \frac{\tilde{K}}{\delta}$ arrays of $\mathrm{FRS}_{\tilde{K}}(X)$. Similar to the argument in strict streams, $\Pr[X \subseteq \mathcal{A}] \geq 1 - \delta/2$ (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 $|\mathcal{A}| \leq \tilde{K} \log \frac{\tilde{K}}{\delta}$.

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

(1) $\forall i \in X$, 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 $\forall i \in X$, $\Pr[n_i < \tau'/2] \leq \binom{\tau'}{\tau'/2} (\frac{1}{8})^{\tau'/2} < (\frac{\delta}{\tilde{K}})^2$. Thus $\Pr[\exists i \in X : n_i < \tau'/2] < \tilde{K} \cdot (\frac{\delta}{\tilde{K}})^2 < \frac{\delta}{4}$.

(2) $i \notin X$ 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 $\forall i \notin X$, $\Pr[n_i \geq \tau'/2] \leq (\frac{\delta}{\tilde{K}})^2$, and $\Pr[\exists i \in \mathcal{A}\colon i \notin X,\ n_i \geq \tau'/2] \leq \tilde{K} \log \frac{\tilde{K}}{\delta} \cdot (\frac{\delta}{\tilde{K}})^2 < \frac{\delta}{4}$.

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. 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 $(1-\epsilon)K \leq |S| \leq \tilde{K}$, 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.*

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 X where $|X| \leq \tilde{K}$ be the set of elements in ϵ-FRS. A *fail set* $F \subseteq X$ 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.

We bound $\Pr[\exists$ fail set $F,\ |F| \geq t]$ using the following (revised) lemma of Pagh and Pagh [24]. The analysis of this lemma is similar to checking the size of the 2-core of the graph that represents ϵ-FRS.

Lemma 4 ([24],Lemma 3.4). *For two functions $i_1, i_2 \colon U \to [R]$ 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-$\mathrm{FRS}_{\tilde{K}}$ have 2 arrays of size $s = 4\tilde{K}$. Let the mapping be performed by $h_1, h_2 \in \mathcal{H}_t$ for $t = c\log\frac{\tilde{K}}{\delta}$, constant c and $t \geq 32$. Then $\forall$ elements set X, where $|X| \leq \tilde{K}$, $\Pr[\exists$ fail set F in ϵ-$\mathrm{FRS}_{\tilde{K}}(X),\ |F| \geq t] < \delta$.*

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

Lemma 5. *Given ϵ-FRS(X), the following graph peeling algorithm returns the output sample $S = \{(i,C_i) \colon i \in X \wedge i \notin F\}$ in $O(K)$ time.*

1. Initialize the output sample $S = \emptyset$ and a queue $Q = \emptyset$.
2. Scan the arrays in ϵ-FRS. $\forall$ bin b, if BS holds a single element, $Enqueue(Q,b)$.
3. While $Q \neq \emptyset$:
   - 3.1. $b \leftarrow Dequeue(Q)$. If BS in b holds a single element:
     - 3.1.1. Extract the element (i,C_i) from BS in b. $S = S \cup \{(i,C_i)\}$.
     - 3.1.2. Subtract (i,C_i) from BS in $\tilde{b}$, the other bin i is hashed to.
     - 3.1.3. $Enqueue(Q,\tilde{b})$.

4. Return S.

Proof. Each BS in array $a \in \{1,2\}$ contains $T_{loc} = \sum C_i h_{3-a}(i)$ where $h_{3-a}(i)$ is the other bin i is hashed to. Thus, getting $h_{3-a}(i)$ takes $O(1)$ operations instead of a hash evaluation in $O((\log\log\frac{K}{\delta})^2)$. For more details see the full version.

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 $f = O(\log\frac{K}{\delta})$.

Non-strict ϵ-FRS. In non-strict turnstile streams, we keep in each bin $\mathrm{BS}_{\mathrm{ns}}$ with $h \in_R \mathcal{H}_{t'}(U,[q])$ for $q = \Theta(\frac{K}{\delta})$ and $t' = \Theta(\log\frac{K}{\delta})$. The same h can be used for all $\mathrm{BS}_{\mathrm{ns}}$s. Let n denote the number of elements in $\mathrm{BS}_{\mathrm{ns}}$. Recall that if $n = 1$, the single element is extracted successfully; if $n < t'$, 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 of the strict model can be used also for non-strict streams, once we prove that with probability $1-\delta$ we do not have any errors.

Lemma 6. *Let ϵ-FRS$_{\tilde{K}}$ have 2 arrays of size $s = 4\tilde{K}$, and let the mapping functions be $h_1, h_2 \in_R \mathcal{H}_t$ for $t = 2\log\frac{\tilde{K}}{\delta}$. Let each bin contain $\mathrm{BS}_{\mathrm{ns}}$ with $q = \frac{4\tilde{K}}{\delta}$ and $t' = t$. There are no false detections during the entire recovery process with probability at least $1-\delta$.*

Proof. First we bound the probability of a large collision in ϵ-FRS. Let ϵ-FRS have $|X| \leq \tilde{K}$ elements. Let $\mathcal{E}_b$ indicate that there is a large collision in bin b. Since $t = t'$, every t' elements that appear in a large collision are mapped there independently. Thus, $Pr[\mathcal{E}_b] \leq \binom{X}{t'}\left(\frac{1}{s}\right)^{t'} \leq \left(\frac{eX}{t'}\right)^{t'}\left(\frac{1}{4\tilde{K}}\right)^{t'} \leq \left(\frac{e}{4t'}\right)^{t'} < \left(\frac{\delta}{\tilde{K}}\right)^2$, and $\Pr[\exists b,\ \mathcal{E}_b] \leq 8\tilde{K}\cdot\left(\frac{\delta}{\tilde{K}}\right)^2 < \delta/2$. Hence we can consider only small collisions.

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 $2\tilde{K}\frac{1}{q} = \delta/2$, and the probability of any false detections is δ.

Recovering Small Sets. ϵ-FRS algorithm can be enhanced to enable recovery of the fail set elements as well. For this purpose we construct the *Small Recovery Structure* (SRS), a deterministic structure based on Reed-Solomon (RS) codes [26], from which all elements can be extracted with probability 1. SRS_n consists of $O(n)$ counters $T_j = \sum_{i\in I} C_i \cdot i^j$ that keep a sketch of substream I. All counters can be updated simultaneously in batches of $O(n)$ elements of I, by multiplication of a transposed Vandermonde matrix by a vector over $\mathbb{C}$, in $O(n\log^2 n)$ operations [14], $O(\log^2 n)$ per element. If $L_0(I) \leq n$, I's elements can be extracted in a similar way to the syndrome decoding of RS codes, with probability 1.

We can combine SRS in our sampling algorithm as follows. In each level we store an instance of $\mathrm{SRS}_{\Theta(\log(K/\delta))}$, updated in parallel to ϵ-FRS. The update takes

$O((\log\log\frac{K}{\delta})^2)$ operations per stream element. When we extract a sample from ϵ-FRS, we remove the extracted elements from SRS (removing is the same as updating). When we reach a fail set F, we extract all its $f = O(\log\frac{K}{\delta})$ elements from $\mathrm{SRS}_{\Theta(\log(K/\delta))}$. Note that this extraction increases the recovery time because of the polynomial factorization phase in the syndrome decoding process. Using this construction, the sample we generate 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. We formalize this for *inverse point queries*, which return $f^{-1}(C)$ for a given frequency C. Approximating inverse range queries, inverse heavy hitters and inverse quantiles queries is similar.

Lemma 7. *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 be $f_S^{-1}(i) = \frac{|\{k\in S:\, C_k=i\}|}{|S|}$. Then $\Pr[\left|f^{-1}(i) - f_S^{-1}(i)\right| < \epsilon] \geq 1-\delta$.*

Proof. See the full version.

Distributed Computing. Our sampling structure is *strongly history independent* [20,23] since it supports additions and deletions and is oblivious of their order in the stream. Hence, if we use the same random bits, we can split the stream to multiple substreams and process each one using an instance of our sampling structure. Then we can add the data structures, by addition of all BSs, and generate a unified sample.

Difference. Given streams D_1, D_2, $D' = \{(i,C_i)\colon C_i = C_i^{D_1} - C_i^{D_2}, C_i \neq 0\}$ is their difference. If D_1,D_2 are represented using the same random bits, we can generate a structure for D' by subtracting all BSs of D_1 from D_2, and extract a sample of D' 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. Cohen, E., Cormode, G., Duffield, N.G.: Don't let the negatives bring you down: sampling from streams of signed updates. In: SIGMETRICS, pp. 343–354 (2012)
2. Cormode, G., Datar, M., Indyk, P., Muthukrishnan, S.: Comparing data streams using hamming norms (how to zero in). IEEE Trans. Knowl. Data Eng. 15(3), 529–540 (2003)
3. Cormode, G., Muthukrishnan, S., Rozenbaum, I.: Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In: VLDB, pp. 25–36 (2005)
4. Datar, M., Muthukrishnan, S.M.: Estimating rarity and similarity over data stream windows. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 323–334. Springer, Heidelberg (2002)

5. Dubhashi, D.P., Ranjan, D.: Balls and bins: A study in negative dependence. Random Struct. Algorithms 13(2), 99–124 (1998)
6. Frahling, G., Indyk, P., Sohler, C.: Sampling in dynamic data streams and applications. In: Symposium on Computational Geometry, pp. 142–149 (2005)
7. Frahling, G., Indyk, P., Sohler, C.: Sampling in dynamic data streams and applications. Int. J. Comput. Geometry Appl. 18(1/2), 3–28 (2008)
8. Ganguly, S.: Counting distinct items over update streams. Theor. Comput. Sci. 378(3), 211–222 (2007)
9. von zur Gathen, J., Gerhard, J.: Modern computer algebra. Cambridge University Press, New York (1999)
10. Gemulla, R., Lehner, W., Haas, P.J.: A dip in the reservoir: Maintaining sample synopses of evolving datasets. In: VLDB, pp. 595–606 (2006)
11. Gemulla, R., Lehner, W., Haas, P.J.: Maintaining bernoulli samples over evolving multisets. In: PODS, pp. 93–102 (2007)
12. Gibbons, P.B.: Distinct sampling for highly-accurate answers to distinct values queries and event reports. In: VLDB, pp. 541–550 (2001)
13. Gilbert, A.C., Li, Y., Porat, E., Strauss, M.J.: Approximate sparse recovery: optimizing time and measurements. In: STOC, pp. 475–484 (2010)
14. Gohberg, I., Olshevsky, V.: Fast algorithms with preprocessing for matrix-vector multiplication problems. J. Complexity 10(4), 411–427 (1994)
15. Goodrich, M.T., Mitzenmacher, M.: Invertible bloom lookup tables. In: 2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 792–799. IEEE (2011)
16. Indyk, P.: A small approximately min-wise independent family of hash functions. J. Algorithms 38(1), 84–90 (2001)
17. Jowhari, H., Saglam, M., Tardos, G.: Tight bounds for lp samplers, finding duplicates in streams, and related problems. In: PODS, pp. 49–58 (2011)
18. Kane, D.M., Nelson, J., Woodruff, D.P.: An optimal algorithm for the distinct elements problem. In: PODS, pp. 41–52 (2010)
19. Karamcheti, V., Geiger, D., Kedem, Z.M., Muthukrishnan, S.: Detecting malicious network traffic using inverse distributions of packet contents. In: MineNet, pp. 165–170 (2005)
20. Micciancio, D.: Oblivious data structures: Applications to cryptography. In: STOC, pp. 456–464 (1997)
21. Monemizadeh, M., Woodruff, D.P.: 1-pass relative-error lp-sampling with applications. In: SODA, pp. 1143–1160 (2010)
22. Muthukrishnan, S.: Data streams: Algorithms and applications. Foundations and Trends in Theoretical Computer Science 1(2) (2005)
23. Naor, M., Teague, V.: Anti-persistence: History independent data structures. IACR Cryptology ePrint Archive 2001, 36 (2001)
24. Pagh, A., Pagh, R.: Uniform hashing in constant time and optimal space. SIAM J. Comput. 38(1), 85–96 (2008)
25. Porat, E., Lipsky, O.: Improved sketching of hamming distance with error correcting. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 173–182. Springer, Heidelberg (2007)
26. Reed, I., Golomb, S.: Polynomial codes over certain finite fields. Joint Society of Industrial and Applied Mathematics Journal 8(2), 300–304 (1960)
27. Tao, Y., Lian, X., Papadias, D., Hadjieleftheriou, M.: Random sampling for continuous streams with arbitrary updates. IEEE Trans. Knowl. Data Eng. 19(1), 96–110 (2007)