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 $O((\log\log\frac{1}{\delta})^2)$.