Theoretical Computer Science 554 (2014) 82–94

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

How to catch L_2-heavy-hitters on sliding windows

Vladimir Braverman [a], Ran Gelles [b,*], Rafail Ostrovsky [b,c]

[a] Department of Computer Science, Johns Hopkins University, United States
[b] Department of Computer Science, University of California, Los Angeles, United States
[c] Department of Mathematics, University of California, Los Angeles, United States

ARTICLE INFO

Article history:
Received 8 September 2013
Received in revised form 13 March 2014
Accepted 4 June 2014
Available online 10 June 2014

Keywords:
Data streams
Approximation algorithms
Heavy hitters
Sliding window model

ABSTRACT

Finding heavy-elements (heavy-hitters) in streaming data is one of the central, and well-understood tasks. Despite the importance of this problem, when considering the *sliding windows* model of streaming (where elements eventually expire) the problem of finding L_2-heavy elements has remained completely open despite multiple papers and considerable success in finding L_1-heavy elements.

Since the L_2-heavy element problem doesn't satisfy certain conditions, existing methods for sliding windows algorithms, such as smooth histograms or exponential histograms are not directly applicable to it. In this paper, we develop the first polylogarithmic-memory algorithm for finding L_2-heavy elements in the sliding window model.

Our technique allows us not only to find L_2-heavy elements, but also heavy elements with respect to any L_p with $0 < p \leq 2$ on sliding windows. By this we completely "close the gap" and resolve the question of finding L_p-heavy elements in the sliding window model with polylogarithmic memory, since it is well known that for $p > 2$ this task is impossible.

We demonstrate a broader applicability of our method on two additional examples: we show how to obtain a sliding window approximation of the similarity of two streams, and of the fraction of elements that appear exactly a specified number of times within the window (the α-rarity problem). In these two illustrative examples of our method, we replace the current *expected* memory bounds with *worst case* bounds.

1. Introduction

A *data stream* S is an ordered multiset of elements $\{a_0, a_1, a_2, \ldots\}$ where each element $a_t \in \{1, \ldots, u\}$ arrives at time t. In the *sliding window model* we consider at each time $t \geq N$ the last N elements of the stream, i.e. the window $W = \{a_{t-(N-1)}, \ldots, a_t\}$. These elements are called *active*, whereas elements that arrived prior to the current window $\{a_i \mid 0 \leq i < t - (N-1)\}$ are *expired*. For $t < N$, the window consists of all the elements received so far, $\{a_0, \ldots, a_t\}$.

Usually, both u and N are considered to be extremely large so it is not applicable to save the entire stream (or even one entire window) in memory. The problem is to be able to calculate various characteristics about the window's elements using small amount of memory (usually, polylogarithmic in N and u). We refer the reader to the books of Muthukrishnan [1] and Aggarwal (ed.) [2] for extensive surveys on data stream models and algorithms.

One of the main open problems in data streams deals with the relations between the different streaming models [3], specifically between the unbounded stream model and the sliding window model. In this paper we provide another

* Corresponding author.
E-mail addresses: vova@cs.jhu.edu (V. Braverman), gelles@cs.ucla.edu (R. Gelles), rafail@cs.ucla.edu (R. Ostrovsky).

http://dx.doi.org/10.1016/j.tcs.2014.06.008
0304-3975/

important step in clarifying the connection between these two models by showing that finding L_p-heavy hitters is just as doable on sliding windows as on the entire stream.

We focus on approximation-algorithms for certain statistical characteristics of the data streams, specifically, finding frequent elements. The problem of finding frequent elements in a stream is useful for many applications, such as network monitoring [4] and DoS prevention [5–7], and was extensively explored over the last decade (see [1,8] for a definition of the problem and a survey of existing solutions, as well as [9–17]).

We say that an element is *heavy* if it appears more times than a constant fraction of some L_p norm of the stream. Recall that for $p > 0$, the L_p norm of the frequency vector[1] is defined by $L_p = (\sum_i n_i^p)^{1/p}$, where n_i is the frequency of element $i \in [u]$, i.e., the number of times i appears in the window. Since different L_p can be considered, we obtain several different ways to define a "heavy" element. Generally speaking (as mentioned in [19]), when considering frequent elements (heavy-hitters) with respect to L_p, the higher p is, the better. Specifically, identifying frequent elements with respect to L_2 is better than L_1 since an L_1 algorithm can always be replaced with an L_2 algorithm, with less or equal memory consumption (but not vice versa).

Naturally, finding frequent elements with respect to the L_2 norm is a more difficult task (memory-wise) than the equivalent L_1 problem. To demonstrate this fact let us regard the following example: let S be a stream of size N, in which the element a_1 appears $\sqrt{N}$ times, while the rest of the elements $a_2, \ldots, a_{N-\sqrt{N}}$ appear exactly once in S. Say we wish to identify a_1 as a heavy element. Note that $n_1 = \frac{1}{\sqrt{N}} L_1$ while $n_1 = cL_2$, where c is a constant, lower bounded by $\frac{1}{\sqrt{2}}$. Therefore, as N grows, $n_1/L_1 \to 0$ goes to zero, while n_1/L_2 is bounded by a constant. If an algorithm finds elements which are heavier than γL_p with memory $poly(\gamma^{-1}, \log N, \log u)$, then for $p = 2$ we get a polylogarithmic memory, while for $p = 1$ the memory consumption is super-logarithmic.

We focus on solving the following L_2-heaviness problem:

Definition 1 *((γ, ϵ)-approximation of L_2-frequent elements).* For $0 < \epsilon, \gamma < 1$, output any element $i \in [u]$ such that $n_i > \gamma L_2$ and no element such that $n_i < (1-\epsilon)\gamma L_2$.

The L_2 norm is the most powerful norm for which we can expect a polylogarithmic solution, for the frequent-elements problem. This is due to the known lower bound of $\Omega(u^{1-2/p})$ for calculating L_p over a stream [20,21].

There has been a lot of progress on the question of finding L_1-frequent elements, in the sliding window model [14,16,17], however those algorithms cannot be used to find L_2-frequent elements with an efficient memory. In 2002, Charikar, Chen and Farach-Colton [9] developed the CountSketch algorithm that can approximate the "top k" frequent-elements on *an unbounded stream*, where k is given as an input. Formally, their algorithm outputs only elements with frequency larger than $(1-\epsilon)\phi_k$, where ϕ_k is the frequency of the kth most frequent element in the stream, using memory proportional to $L_2^2/(\epsilon\phi_k)^2$. Since the "heaviness" in this case is relative to ϕ_k, and the memory is bounded by the fraction $L_2^2/(\epsilon\phi_k)^2$, Charikar et al.'s algorithm finds in fact heaviness in terms of the L_2 norm. A natural question is whether one can develop an algorithm for finding frequent-elements that appear at least γL_2 times in the *sliding window model*, using $poly(\gamma^{-1}, \log N, \log u)$ memory.

Our results. We give the first polylogarithmic algorithm for finding an ϵ-approximation of the L_2-frequent elements in the sliding window model. Our algorithm is able to identify elements that appear within the window a number of times which is at least a γ-fraction of the L_2 norm of the window, up to a multiplicative factor of $(1-\epsilon)$. In addition, the algorithm guarantees to output *all* the elements with frequency at least $(1+\epsilon)\gamma L_2$.

Theorem 1. *There exists an efficient sliding window algorithm that outputs a (γ, ϵ)-approximation of the L_2-frequent-elements, with probability at least $1-\delta$ and memory $poly(\epsilon^{-1}, \gamma^{-1}, \log N, \log \delta^{-1})$.*

We note that the CountSketch algorithm works in the unbounded model and does not apply directly on sliding windows. Moreover, CountSketch solves a slightly different (yet related) problem, namely, the top-k problem, rather than the L_2 heaviness. To achieve our result on L_2 heavy hitters, we combine in a non-trivial way the scheme of Charikar et al. with a sliding-window approximation for L_2 as given by Braverman and Ostrovsky [22]. Variants of these techniques sufficient to derive similar results were known since 2002,[2] however no algorithm for L_2 heavy hitters was reported despite several papers on L_1 heavy hitters.

Our solution gives another step in the direction of making a connection between the unbounded and sliding window models, as it provides an answer for the very important question of heavy hitters in the sliding window model. The result joins the various solutions of finding L_1-heavy hitters in sliding windows [11,14,24,7,16,17,25], and can be used in various algorithms that require identifying L_2 heavy hitters, such as [26,27] and others. More generally, our paper resolves the

[1] Throughout the paper we use the term "L_p norm" to indicate the L_p norm of the frequency vector, i.e., the p-th root of the p-th frequency moment $F_p = \sum_i n_i^p$ [18], rather than the norm of the data itself.

[2] Indeed, we use the algorithm of Charikar et al. [9] that is known since 2002. Also, it is possible to replace (with some non-trivial effort) our smooth histogram method for L_2 computation with the algorithm of Datar, Gionis, Indyk and Motwani [23] for L_2 approximation.