Theoretical Computer Science 562 (2015) 23–32

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Competitive analysis of maintaining frequent items of a stream ☆,☆☆

Yiannis Giannakopoulos *, Elias Koutsoupias

University of Oxford, Department of Computer Science, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

ARTICLE INFO

Article history:
Received 31 May 2013
Accepted 10 September 2014
Available online 17 September 2014
Communicated by G. Ausiello

Keywords:
Online algorithms
Competitive analysis
Frequent items
Data streams

ABSTRACT

We study the classic frequent items problem in data streams, but from a competitive analysis point of view. We consider the standard worst-case input model, as well as a weaker distributional adversarial setting. We are primarily interested in the single-slot memory case and for both models we give (asymptotically) tight bounds of $\Theta(\sqrt{N})$ and $\Theta(\sqrt[3]{N})$ respectively, achieved by very simple and natural algorithms, where N is the stream's length. We also provide lower bounds, for both models, in the more general case of arbitrary memory sizes of $k \geq 1$

1. Introduction

The *frequent items* problem [2] is one of the most well-studied ones in the area of data streams [3–7]. Informally, the problem is that of observing a stream (sequence) of values and trying to discover those that appear most frequently. Many applications in packet routing, telecommunication logging and tracking keyword queries in search machines are critically based upon such routines.

More formally, in the most basic version of the classic frequent items problem, we are given a stream $a_1, a_2, \ldots, a_N$ of items from some universe, as well as a frequency threshold ϕ, $0 < \phi < 1$, and we are asked to find and/or maintain all items that occur more than ϕN times throughout the stream. For real-life applications there are some restricting assumptions the algorithms need to respect: the size N of the stream, as well as the rate at which the items arrive, far exceed the computational capabilities of our devices. Consequently, we usually require streaming algorithms to use $O(polylog(N))$ memory and allow approximate solutions within a factor of ε (additive, with respect to ϕ), since exact solutions would require linear space [2], a totally unrealistic option in some domains. Furthermore, we are usually interested in algorithms that make as few passes as possible over the input stream and, in particular, *single-pass* algorithms that process the input stream in an online way, i.e. each item sequentially, making decisions on-the-fly.

So, traditionally data stream problems have been essentially approached as space complexity optimization problems: given an input stream and an approximation guarantee of ϵ, we try to minimize the memory used (see, e.g., the seminal work of [8]). However, despite their intrinsic *online* nature, these problems have not been studied within the predominant framework for studying online problems, i.e. that of competitive analysis [9].

☆ A preliminary version of this paper, not including the results in Section 5 about ephemeral frequencies, appeared in [1].

☆☆ This work was supported by ERC Advanced Grant 321171 (ALGAME) and the ESF-NSRF research program Thales (ALGONOW). A significant part of it was done while the authors were at the Department of Informatics of the University of Athens.

* Corresponding author. Tel.: +44 1865273891.
E-mail addresses: ygiannak@cs.ox.ac.uk (Y. Giannakopoulos), elias@cs.ox.ac.uk (E. Koutsoupias).

http://dx.doi.org/10.1016/j.tcs.2014.09.011
0304-3975/

Only recently, Becchetti and Koutsoupias [10] did that, using competitive analysis to study another important streaming problem, namely that of maintaining the maximum value within a sliding window [11]. Following a similar approach, we formulate an online version of the classic frequent items problem: given an input stream and a memory of at most k slots, try to optimize the online algorithm's performance with respect to that of an optimal offline algorithm that knows the entire input stream in advance. This is, in a way, the inverse of the traditional space complexity optimization objective mentioned above. We also use a similar aggregate-through-time perspective (instead of an unrealistic optimization-at-every-step requirement) upon defining our objective function (see Section 1.1). As argued in [10], the competitive analysis approach combined with an aggregate objective, seem more appropriate for economic applications as well as a decision-making under uncertainty framework.

1.1. Setting

We are observing a stream $A = a_1, a_2, \ldots, a_N$ of N elements drawn from some universe $\mathcal{U}$, in a sequential, *online* fashion. It is natural to assume that $|\mathcal{U}| \gg N$. We consider algorithms that, at every time step $t = 1, 2, \ldots, N$, maintain in memory a set $S_t(A) \subseteq \{a_1, a_2, \ldots, a_t\}$ of at most k items from the part of the input stream A observed so far.[1] Furthermore, we assume that the only way in which our algorithms can update these memory sets throughout the execution, is to make an *irrevocable* decision, at every time point t, of whether or not to store the newly arrived element a_t in memory, i.e. $S_t \subseteq S_{t-1} \cup \{a_t\}$. An *online* algorithm can base its decision just[2] on the knowledge of its current memory state S_{t-1} and the current time point t, while an offline algorithm can have access to the entire input stream A. Notice that, since $|S_t| \leq k$, if at some point t we want to store a new element $a_t \notin S_{t-1}$, then we may need to discard some previously stored item $a_j \in S_{t-1}$, $j < t$ and then $S_t \subseteq \{a_t\} \cup S_{t-1} \setminus \{a_j\}$. For the special case of $k = 1$, which will be our main concern in this paper for the most part (we will consider general memory sizes of $k \geq 1$ again in Section 4) we will denote by s_t the unique item in the algorithm's memory at time point t, i.e. $S_t = \{s_t\}$.

Given an input stream A and an element $a \in A$ we define its *frequency* as $f^A(a) = \frac{n^A(a)}{N}$, where $n_A(a) = |\{i \mid a_i = a\}|$ is the number of instances of a in the stream.[3] Intuitively, we want our algorithms, at every time, to maintain the most frequent items possible. We formalize this, by defining the *aggregate frequency* objective as the sum, across the entire execution, of the frequencies of all *distinct* items in memory, i.e. $\sum_{t=1}^{N} \sum_{a \in S_t} f(a)$. Notice here a fine point: we treat S_t as a set and *not* as a multi-set, i.e. multiple occurrences of the same element in memory can only contribute *once* towards our objective. We measure an online algorithm's performance on a given input A by comparing its total gain (i.e. the value of the aggregate frequency objective on stream A) to that of an offline algorithm that knows the entire input stream A in advance. The *competitive ratio* of the online algorithm is the maximum value of this ratio among all possible inputs,

$$\max_A \frac{\sum_{t=1}^{N} \sum_{a \in S'_t} f^A(a)}{\sum_{t=1}^{N} \sum_{a \in S_t} f^A(a)},$$

where S_t, S'_t are the memory sets of the online and optimal offline algorithm, respectively. The competitive ratio for our online frequent items problem, is the best (minimum) competitive ratio we can achieve over all online algorithms.

Finally, whenever we deal with randomized algorithms in this paper, we are always silently assuming the standard, oblivious adversary [12,9] model, i.e. the adversary decides an input A knowing the online algorithm but not the actual results of its coin tosses.

1.2. Organization of the paper and results

In this paper we are mostly interested in the special case of single-slot memories ($k = 1$). That is the case for Sections 2 and 3. We do not deal with general memory sizes of $k \geq 1$ until Section 4.

In Section 2, we prove that the competitive ratio of the online frequent items problem is $\Theta(\sqrt{N})$, by providing a lower bound proof of $\frac{1}{3}\sqrt{N}$ and showing that the most simple algorithm that myopically accepts every element that arrives achieves an (asymptotically) tight $\sqrt{N}$ competitive ratio. Furthermore, in Section 2.1, we show that the well known Majority algorithm for the classical frequent items problem performs very poorly from a competitive analysis point of view, since it has (asymptotically) the worst possible competitive ratio an online algorithm can demonstrate, namely $\Theta(N)$.

Also, we consider weaker adversarial inputs and in particular the case of the input stream being generated i.i.d. from a probability distribution, known only to the adversary. In Section 3 we show how a very simple and natural algorithm, called Eager, that essentially waits until it sees some element appearing twice, achieves a competitive ratio of $O(\sqrt[3]{N})$, asymptotically matching a lower bound of $\Omega(\sqrt[3]{N})$ again providing a tight competitive ratio (of $\Theta(\sqrt[3]{N})$) for the case of $k = 1$.

[1] To keep notation light, we will simply use S_t instead of $S_t(A)$ whenever it is clear to which input stream we are referring to.

[2] In a different online setting, we could have also assumed that the online algorithm has complete knowledge of the past. This, though, seems as a rather unrealistic assumption to make, especially in a streaming setting.

[3] Again, we will drop the superscript A whenever this causes no confusion.