Mining heavy hitters with Space-Saving

Daniele Cono D'Elia

Dept. of Computer, Control and Management Engineering
Sapienza University of Rome

delia@dis.uniroma1.it

August 11, 2013

1 Introduction

In the last ten years a lot of effort has been put on the design of algorithms able to analyze massive data streams on a near-real time basis; in such streams input data often come at a high rate and cannot be stored due to their possibly unbounded size [6]. This line of research has been mainly motivated by networking and database applications, in which streams may be very long and stream items may be drawn from a very large universe. Streaming algorithms are designed to address these requirements, providing approximate answers in contexts where it is impossible to obtain an exact solution using limited space.

The entries in the query logs of a search engine represent a good example of massive data stream: items (i.e. submitted queries) belong to a possibly unbounded universe (i.e. the set of all possible strings) and the length of the stream itself is very high. Identifying the most popular search terms is clearly an instance of the top-k problem, in which the top-k elements are the k elements occurring with highest frequency. Unfortunately, the exact solution of this problem requires complete knowledge about the frequency of all elements [2], hence a space that is linear in the number of distinct queries is required. Even if it is possible to keep all elements in memory, performance would be clearly weak due to poor access locality

In this work we implemented the Space-Saving algorithm and analyzed its accuracy with respect to the exact solution. Space-Saving [5] is a deterministic algorithm able to solve the *ε-Deficient Frequent Elements* problem [4] using fixed space; we will show how to use this algorithm also to approximate the top-k problem yielding accurate results.

The rest of this document is organized as follows. Section 2 gives theoretical definitions on the frequent items problem and the Space-Saving algorithm. Section 3 focuses on implementation aspects. Section 4 presents the outcome of

our experimental study. Finally, in Section 5 we discuss additional remarks to be shared with the reader.

2 Background

2.1 The Frequent Items problem

The *Frequent Items* (a.k.a. heavy hitters) problem [5] has been extensively studied in data streaming computational models. Given a frequency threshold $\phi \in [0, 1]$ and a data stream of length N, the problem is to find all items that appear more than $\lceil \phi N \rceil$ times, i.e., having frequency greater than ϕ. For instance, for $\phi = 0.01$ the problem seeks all items that appear at least 1% of the times.

It can be proved that any algorithm that provides an exact solution requires $\Omega(N)$ space [6]. Therefore, research focused on solving an approximate version of the problem [4, 5]:

Definition 1. *ε-Deficient Frequent Elements problem. Any algorithm that solves this problem will behave as follows:*

1. *All items whose true frequency exceeds ϕN are output (no false negatives);*
2. *No item whose true frequency is less than $(\phi - \varepsilon)N$ is output;*
3. *The difference between estimated and true frequency is at most εN.*

Clearly, $\phi, \varepsilon \in [0, 1]$ and $\varepsilon \ll \phi$.

2.2 Space-Saving

Space-Saving monitors a set of $m = \lceil 1/\varepsilon \rceil$ triples of the form (*item*, *count*, ϵ) initialized respectively by the first $1/\varepsilon$ distinct items, their exact counts, and zero. After the init phase, when an element e is observed in the stream the update operation works as follows:

- if e is already monitored, the corresponding counter is incremented;
- if e is not monitored, the (*item*, *count*, ϵ) triple with the smallest count is chosen as a victim and has its item replaced with e and its count incremented. Moreover, ϵ is set to the count value of the victim triple - it is used to measure the maximum possible estimation error.

The update time is bounded by the dictionary operation of checking whether an item is monitored. Heavy hitters queries are answered by returning entries in the set such that $count > \lceil \phi N \rceil$.

Space-Saving has the following properties (see [5] for proofs):

1. The minimum counter value *min* is no greater than $\lfloor \varepsilon N \rfloor$;
2. For any monitored element, $0 \leq \epsilon \leq min$, that is, $f \leq (f + \epsilon) = count \leq f + min$. In other words, *count* is overestimated by at most *min* with respect to the true frequency f;

3. Any element whose true frequency is greater than *min* is monitored;

4. The algorithm uses a number of counters $\min\{|A|, \lceil 1/\varepsilon \rceil\}$, where $|A|$ is the cardinality of the universe from which the items are drawn; this can be proved to be optimal;

5. Space-Saving solves the *ε-Deficient Frequent Elements* problem, by reporting all elements with frequency greater than $\lceil \phi N \rceil$, and no additionally reported element has true frequency less than $\lceil (\phi - \varepsilon)N \rceil$;

6. Items whose true frequencies are greater than $\lceil (\phi - \varepsilon)N \rceil$ but less than $\lceil \phi N \rceil$ could be output. They are the so-called false positives.

3 Implementation

3.1 Space-Saving

As suggested by the authors [5], we realized a C implementation of the Stream-Summary (SS) data structure using a doubly linked list for buckets and linked lists for counters. In SS, all elements with the same counter value are linked together in a linked list; also, they all point to a parent bucket in which the count value is stored. Every bucket points to the first element of the linked list of counters, and buckets are kept in a doubly linked list, sorted by their values.

Initially, m empty counters are created and attached to a single parent bucket with value 0. When an element of the stream is processed, the `SS_lookup` routine verifies whether an element is already monitored using a hash table (`GHashTable` from the `GLib` library[1]) and returns a pointer to the counter. If the element is not actually monitored, the `SS_update` routine deals with the selection of the victim counter as described in Section 2.2. Finally, the `increment_counter` operation links the counter to the proper bucket. In particular, it first checks the bucket's neighbor with the larger value. If its value is equal to the new value of the element, the element is detached from its current list and inserted as first child in the neighbor's list. Otherwise, a new bucket with the correct value is created and then properly inserted in the ordered bucket list; then the element is attached to the new bucket. The old bucket is deleted if it now points to an empty child list.

Heavy hitters queries are answered as described in Section 2.2; items are output in decreasing order, while the *guaranteed* flag [5] tells whether there might be false positives reported.

3.2 Exact Solution

Beside the Space-Saving algorithm, we implemented a simple tool that calculates the exact frequency of all elements. This tool is needed to evaluate the accuracy of the results provided by Space-Saving, as we will see in Section 4.

A `GHashTable` is used to maintain the (*item*, *count*) pairs and for each element of the stream it is updated as follows: if an element appears for the first time in the stream, an entry is added to the table with *count* equal to 1; if an element has already been observed in the past, the associated entry is

[1] `GLib` Reference Manual - `http://developer.gnome.org/glib/`.

updated by incrementing *count* by 1. Heavy hitters queries are answered by returning those entries for which $count > \lceil \phi N \rceil$.

4 Experimental Evaluation

4.1 Data Sets and Parameters

As data sets for our experiments we chose the query logs of 4 consecutive days of the AOL search engine [7]. These logs track the first four days of March 2006 and have been analyzed both individually and in the aggregated version – see Table 1 for their characteristics.

Our source code has been compiled and executed on a Linux laptop running Ubuntu 12.10 with with kernel 3.5.0, `gcc` 4.7.2 and `GLib` 2.34.

Table 1: Characteristics of the data sets

Day	Stream length (N)	Distinct items	Distinct items/N
1	267,887	197,651	0.73781
2	272,935	202,926	0.74349
3	246,136	183,567	0.74579
4	266,687	198,063	0.74268
Aggregated	1,053,645	696,027	0.66059

In our analysis we considered four accuracy measures:

1. *Precision*, defined as the fraction of retrieved instances that are relevant (i.e., the fraction of heavy hitters returned by Space-Saving that are actual heavy hitters);

2. *Recall*, defined as the fraction of relevant instances that are retrieved (i.e., the fraction of actual heavy hitters that are returned by Space-Saving);

3. *Mean Absolute Percentage Error* (MAPE), calculated by summing the ratios between overestimation error and true frequency for the actual heavy hitters, then multiplying this value by 100 and dividing it by the number of actual heavy hitters;

4. *Maximum Percentage Error* (MPE), defined as the maximum value for the ratio between overestimation error and true frequency calculated among actual heavy hitters - this ratio is then multiplied by 100.

These indices are calculated every 20000 processed items of the stream, in order to simulate an on-line querying process. We will focus on their average values calculated among the iterations.

4.2 Mining the top-k items

The choice of the ϕ parameter is strictly related to the number of top-k items we would like to retrieve. Our approach to approximate a solution to the top-k

problem using an ε-Deficient Frequent Elements algorithm required a preliminary analysis of the frequencies of the most frequent items[2]. In Figure 1 we plot the number of actual heavy hitters for different values of ϕ in the interval $[0.0001; 0.001]$. Clearly, the great number of distinct items with respect to the stream length implies that most frequent items have frequencies quite low from a strictly numerical point of view.

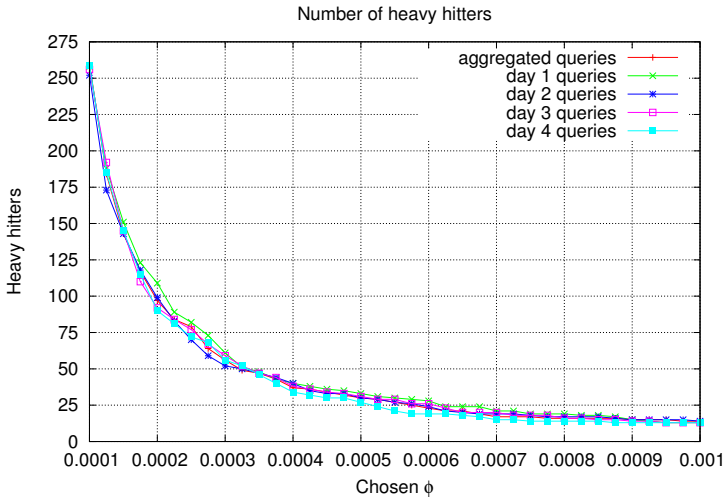

Figure 1: Number of heavy hitters for different values of ϕ.

The most interesting fact in our opinion is that the number of retrieved items at the frequencies we considered is roughly the same in the four logs we considered and, in particular, also in the aggregated log. Hence, the characteristics of the distributions of submitted queries allows us to use the same values of ϕ for streams of increasing length, while the number of counters kept in memory by Space-Saving remains the same. In other words, it is possible to achieve greater reductions in space usage as long as the length of the stream increases and new elements appear.

Figure 2 emphasizes this fact: heavy hitters account for a (small) percentage of the distinct items observed in the stream that decreases when considering bigger data streams (i.e., the one made up of aggregated queries).

We identified five values of ϕ suited to retrieve the top-k items (and possibly other heavy hitters) for k chosen in $\{10, 25, 50, 75, 100\}$: such values are 0.001, 0.00055, 0.0003, 0.00025, and 0.000175, respectively. In particular, we identified the biggest possible value of ϕ such that at least k elements are output for a given value of k. In Section 4.5 we will analyze the correlation between space usage and the value of ϕ: however, it should already be clear that our choice aims at minimizing space usage. The sum of the frequencies of items retrieved for these values of ϕ accounts for 3.61%, 4.49%, 5.63%, 6.27%, and 7.03% of the length of the stream of aggregated queries, respectively.

[2]The authors of Space-Saving also proposed[5] a version of the algorithm tailored to the top-k problem. This algorithm is slightly more complex than the one discussed in this work and might be of interest for further reading.

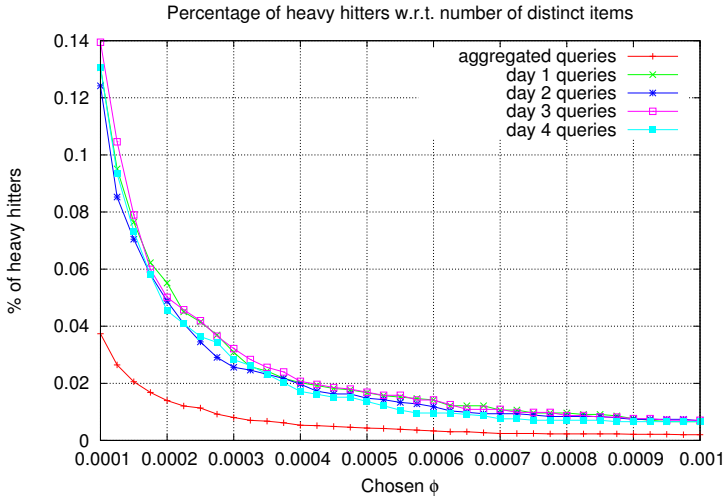

Figure 2: Number of heavy hitters as percentage of the total number of distinct items in the stream.

4.3 Precision and Recall

Precision and recall are very used measures in research on information retrieval. In Figure 3 we plot both of them, calculated on the stream of aggregated queries using different values for ϕ. The ε parameter has been chosen such that the ratio between ϕ and ε is equal to 5; the reason behind this choice will be explained in Section 4.5.

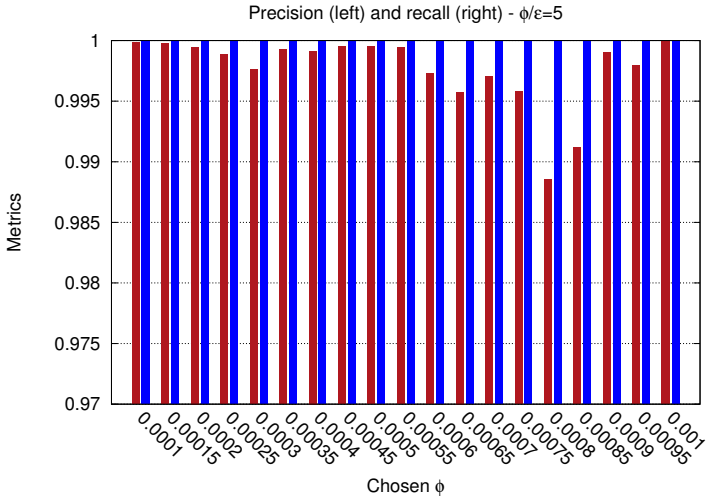

Figure 3: Precision and recall for different values of ϕ.

Precision is always greater than 0.985: this means that the fraction of false positives among heavy hitters output by Space-Saving is very little. Moreover, this fraction is often 0: in other words, the algorithm most of the times returns

actual heavy hitters only.

Claim 1. *Recall is always 1 for the Space-Saving algorithm.*

Proof. *Space-Saving solves in a deterministic way the ε-Deficient Frequent Elements problem by monitoring all items whose actual frequency is greater than $\lfloor \varepsilon N \rfloor$ and reporting all elements with frequency greater than $\lceil \phi N \rceil$ (see Section 2.2 or [5]). The intersection between the sets of retrieved and relevant instances coincides with the set of relevant instances, hence recall is always* 1.

In fact, recall is important mainly for stochastic counter-based and sketch-based approaches to the problem [1], so we will not discuss it anymore in the remaining part of this work.

4.4 Average and Maximum Error

MAPE (*Mean Absolute Percentage Error*) can be used as a measure of accuracy to evaluate the global behavior of the Space-Saving algorithm with respect to the overestimation of the frequencies of actual heavy hitters. Since the estimation error is non-negative, we can also refer to MAPE as average percentage error.

The MPE (*Maximum Percentage Error*) index, instead, provides a clue on the worst-case overestimation scenario for the algorithm. In fact, in many cases using only MAPE might not be sufficient to evaluate accuracy, hence also metrics such as MPE or MSE (*Mean Squared Error*) should be taken into account.

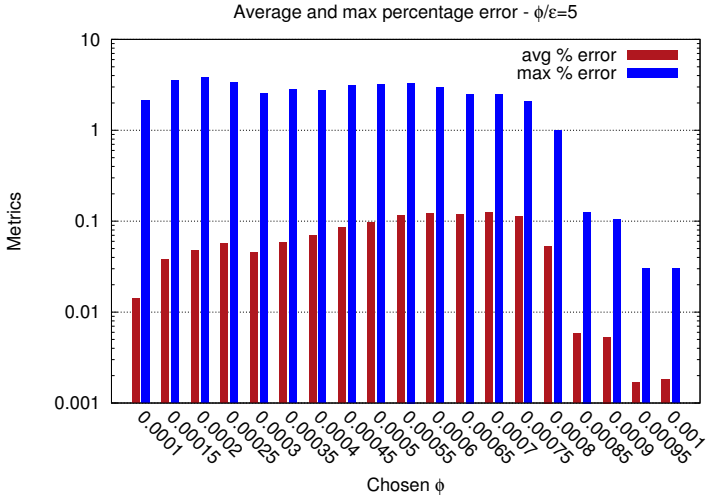

Figure 4: MAPE and max percentage error for different values of ϕ.

Figure 4 shows the two indices evaluated on the stream of aggregated queries for different values of ϕ, with the ratio between ϕ and ε fixed to 5. Data are plotted on a logarithmic scale and the peak values are 0.125 ($\phi = 0.0007$) and 3.84 ($\phi = 0.0002$) for average and maximum percentage error, respectively. Note that the upper bound on maximum percentage error is given by the ratio between ε and ϕ multiplied by 100 - in fact, an actual heavy hitter should occur more than $\lceil \phi N \rceil$ times in the stream to be reported and the maximum error on

the estimated frequency of a counter is at most $\lfloor \varepsilon N \rfloor$ - and is equal to 20% for this choice of parameters.

4.5 (ϕ/ε) trade-off

In this section we discuss the problem of choosing the right value for the ε parameter. Clearly, there is a trade-off between accuracy and space: with a higher value of ε less counters are kept in memory but, on the other hand, overestimations could become worse; conversely, with a lower value of ε results would be very accurate at the price of using much more space. Moreover, the choice of ε is strictly related to the ϕ parameter, since $\varepsilon \ll \phi$.

A rule of thumb validated by previous experimental studies [3] suggests that it is sufficient to choose them such that $\phi/\varepsilon = 10$ to obtain a small number of false positives and high counter accuracy. We found this choice overly pessimistic in our scenario: the characteristics of the streams we analyzed make it possible to use much smaller values for this ratio without sacrificing accuracy. In this section we will focus on the stream of aggregated queries because of its length: while the memory usage of Space-Saving depends only on the choice of the ε parameter, the exact solution to the problem requires space growing linearly in the number of items. This number, shown in Table 1, is much bigger in the aggregated log than in the logs of single days: in fact, on a short time interval, a lot of new unpopular queries are submitted day by day.

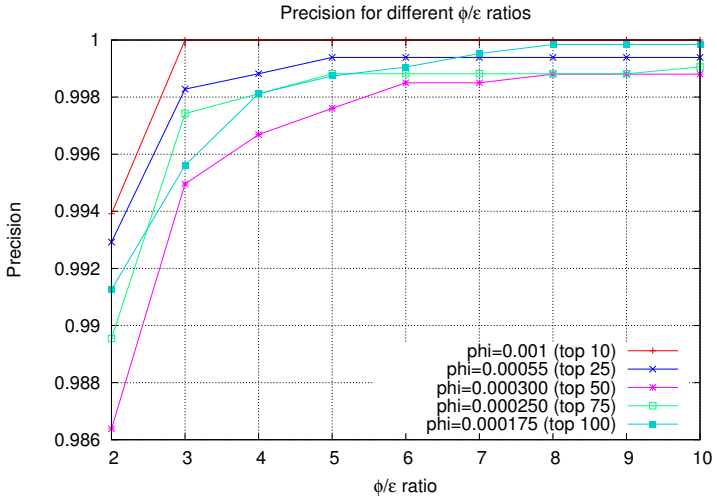

Figure 5: Precision for different choices of ϕ/ε ratio.

The first accuracy measure we will analyze is precision. Figure 5 shows how this index varies for different choices of the ϕ/ε ratio. In particular, values range from 2 to 10 with unit increments; plotted curves are calculated for the values of ϕ discussed in Section 4.2. Precision is not significantly affected by the choice of smaller values for the ratio: the index is very close to 0.99 already for $\phi/\varepsilon = 2$ and greater than 0.9949 for $\phi/\varepsilon \geq 3$. Note that for $\phi = 0.001$ precision is equal to 1 already for $\phi/\varepsilon \geq 3$: in other words, no false positives are reported in any iteration.

In Figure 6 we report the average percentage error calculated with the same settings of the previous experiment. We observe that MAPE is more affected than precision by the choice of smaller values for the ratio. However, the range of variation is very limited: the maximum average percentage error among the five curves is equal to 0.366% ($\phi = 0.00055$) and becomes lower than 0.1% for all of the five curves when choosing $\phi/\varepsilon \geq 6$. These results suggest that even $\phi/\varepsilon = 2$ for the ratio is a good choice from the MAPE perspective.

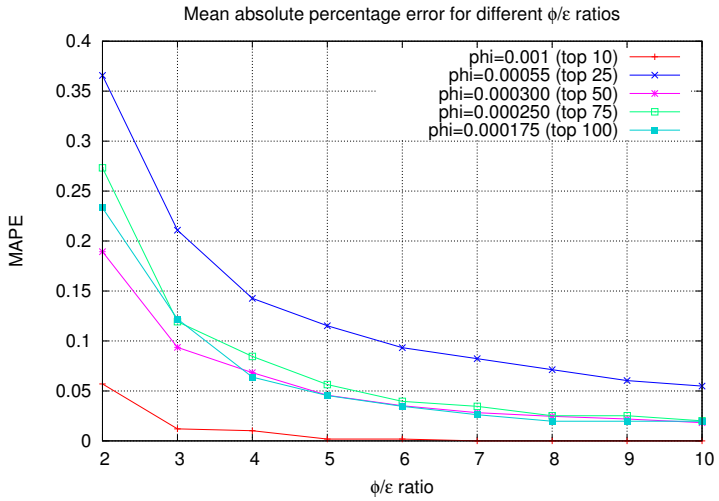

Figure 6: MAPE for different choices of ϕ/ε ratio.

Finally, Figure 7 shows how the maximum percentage error varies for different values of ϕ/ε. Among the three accuracy indices we considered, MPE is the most affected by the choice of smaller values for this ratio.

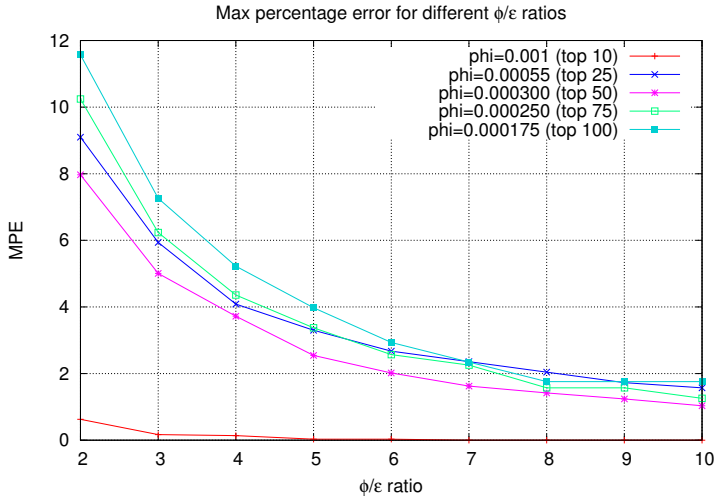

Figure 7: MPE for different choices of ϕ/ε ratio.

A straightforward explanation to this phenomenon is that by using lower values of ε more counters are kept in memory: since the victim counters for a sequence of replacement operations can be chosen from a possibly larger set of counters with low frequencies, the value of min increases more slowly – hence the overestimation for new items is lower. For this reason, when an actual heavy hitter is observed for the first time in the stream, it is added to the Stream-Summary with a lower initial overestimation. The same consideration also holds when an actual heavy hitter is observed only a few times in the initial part of the stream (so it is more likely to be removed from the SS) and then starts to occur much frequently: from now on, it is added to SS with an overestimation equal to min and then never removed.

The MAPE index suggests that significant overestimations are present only in a few output items, since the average error is very low. However, the values reported in Figure 7 are much lower than the respective theoretical upper bounds for the overestimation error: for instance, for $\phi/\varepsilon = 2$ the upper bound on actual heavy hitters is equal to 50% while the maximum error reported among the five curves is equal to 11.57%. Moreover, for $\phi = 0.001$ the maximum percentage error goes quickly to zero.

Starting from the considerations made in this section for the accuracy measures, we chose $\phi/\varepsilon = 5$ in the experiments reported in Figure 3 and 4 in order to achieve very good results in all of the three indices while keeping at the same time only a reasonably small number of counters in main memory.

From the space usage perspective, a possible further optimization consists in tailoring the choice of the ϕ/ε ratio also to the value of ϕ used in the experiment. For instance, with $\phi = 0.001$ it is possible to achieve very high accuracy already for $\phi/\varepsilon = 2$: in this scenario only $\lceil 1/\varepsilon \rceil = 2000$ counters are kept in memory, while the exact solution would require nearly $700k$ counters.

5 Conclusions

Streaming algorithms can provide very space-efficient solutions for scenarios in which the input is presented as a long sequence of items and can be examined only in a single pass and limited memory. Space-Saving proved to be very efficient and accurate in our experimental evaluation, and we believe it can be implemented on a nearly real-time basis since the number of operations required to process an event in the stream is very low. In this final section we would like to share with the reader two additional remarks.

The choice of parameters and the accuracy results discussed in Section 4 should be compared to the outcome of an execution of the algorithm on a much larger data set (e.g., the query logs of a whole month). We suppose that the characteristics of a larger cumulative distribution would be a little bit different from those of the aggregated log we analyzed: in fact, the ratio between number of distinct items and stream length will become possibly lower, since on a larger time interval it is unlikely that users submit with a steady frequency queries on items never observed in the past. However, we believe that Space-Saving can deal very well with this differences: since in our experiments the initial overestimation of heavy hitters showed to be low, once a frequent element begins to be monitored it is unlikely that it will be removed from the Stream-Summary and then re-added later. Hence, overestimation errors can become larger only

for elements that do not appear with a regular rate over time (e.g., a query submitted many times in a couple of consecutive days, for instance related to some sport event).

Note also that Space-Saving during its execution uses a fixed quantity of space that does not depend on the characteristics of the analyzed stream, but only on the choice of the ε parameter. Other streaming algorithms, on the other hand, have worse theoretical bounds on space usage but they might perform well in practice [3]. In particular, we believe that Lossy Counting [4] can be a very promising one. This algorithm stores in main memory all items observed in a window of fixed length and periodically deletes items with low frequencies: this simple approach might deal well with the characteristics of the query logs of search engine, in which many items appear a very small number of times. A comparison between Space-Saving and Lossy Counting would represent a natural extension of this work.

References

[1] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proc. VLDB Endow.*, 1(2):1530–1541, August 2008.

[2] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA '02, pages 348–360, London, UK, UK, 2002. Springer-Verlag.

[3] Nishad Manerikar and Themis Palpanas. Frequent items in streaming data: An experimental evaluation of the state-of-the-art. *Data Knowl. Eng.*, 68(4):415–430, April 2009.

[4] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 346–357. VLDB Endowment, 2002.

[5] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Trans. Database Syst.*, 31(3):1095–1133, September 2006.

[6] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[7] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *Proceedings of the 1st international conference on Scalable information systems*, InfoScale '06, New York, NY, USA, 2006. ACM.