

# Mining Top-K Frequent Items in a Data Stream with Flexible Sliding Windows

Hoang Thanh Lam  
Dept. of Math. and Computer Science  
TU Eindhoven  
t.l.hoang@tue.nl

Toon Calders  
Dept. of Math. and Computer Science  
TU Eindhoven  
t.calders@tue.nl

## ABSTRACT

We study the problem of finding the  $k$  most frequent items in a stream of items for the recently proposed max-frequency measure. Based on the properties of an item, the max-frequency of an item is counted over a sliding window of which the length changes dynamically. Besides being parameterless, this way of measuring the support of items was shown to have the advantage of a faster detection of bursts in a stream, especially if the set of items is heterogeneous. The algorithm that was proposed for maintaining all frequent items, however, scales poorly when the number of items becomes large. Therefore, in this paper we propose, instead of reporting all frequent items, to only mine the top- $k$  most frequent ones. First we prove that in order to solve this problem exactly, we still need a prohibitive amount of memory (at least linear in the number of items). Yet, under some reasonable conditions, we show both theoretically and empirically that a memory-efficient algorithm exists. A prototype of this algorithm is implemented and we present its performance w.r.t. memory-efficiency on real-life data and in controlled experiments with synthetic data.

## Categories and Subject Descriptors

H.2.8 [Database applications]: Miscellaneous

## General Terms

Algorithms, Performance

## Keywords

Data stream mining, top-k frequent items

## 1. INTRODUCTION

In this paper we study the problem of mining frequent items in a stream under the assumption that the number of different items can become so large that the stream summary cannot fit the system memory. This occurs, e.g., when

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD'10, July 25–28, 2010, Washington, DC, USA.

Copyright 2010 ACM 978-1-4503-0055-110/07 ...\$10.00.

monitoring popular search terms, identifying IP addresses that are the source of heavy traffic in a network, finding frequent pairs of callers in a telecommunication network, etc. In stream mining it is usually assumed that the data arrives at such a high rate that it is impossible to first store the data and subsequently mine patterns from the data. Typically there is also a need for the mining results to be available in real-time; at every moment an up-to-date version of the mining results must be available.

Most proposals for mining frequent items in streams can be divided into two large groups: on the one hand those that count the frequency of the items over the complete stream [3] and on the other hand, those based on the most recent part of the stream only [6, 5]. We are here concerned with the latter type. The current frequency of an item is in this context usually defined as either the weighted average over all transaction where the weight of a transaction decreases exponentially with the age of the transaction or by only considering the items that arrived within a window of fixed length (measured either in time or in number of transactions) from the current time. As argued by Calders *et al.* [1, 2], however, setting these parameters, the decay factor and the window size, is far from trivial. As often the case in data mining, the presence of free parameters rather represents additional burden on the user than it provides more freedom. Even worse, in many cases no single optimal choice for the parameters exists, as the most logical interval to measure frequency over may be item-dependent.

For these reasons, the max-frequency was introduced. The max-frequency of an item is parameterless and is defined as the maximum of the item frequency over all window lengths. As such, an item is given the “benefit of the doubt”; for every item its most optimal window is selected. The window that maximizes the frequency can grow and shrink again as the stream grows. Experiments have shown that “*this new stream measure turns out to be very suitable to early detect sudden bursts of occurrences of itemsets, while still taking into account the history of the itemset. This behavior might be particularly useful in applications where hot topics, or popular combinations of topics need to be tracked*” [1]

An algorithm was introduced in [1], based on the observation that even though the maximal window can shrink and grow again as the stream passes, only a few points in the stream are actually candidate for becoming the starting point of a maximal window; many time points can be discarded. Only for these candidates, called the *borders*, statistics are maintained in a summary. However, the algorithm scales linearly in the number of distinct items in the

stream. Therefore, this algorithm is clearly not suitable for systems with limited memory, such as for instance, a dedicated system monitoring thousands of streams from a sensor network. Under such circumstances, a very limited amount of memory is allocated for each stream summary. In this paper we extend the work on max-frequency by studying how this problem can be solved efficiently. Our contributions are as follows:

- We show a lower bound on the memory requirements for answering the frequent items query exactly. This bound shows that the dependence on the number of distinct items is inherent to the problem of mining all frequent items from a stream exactly.
- Therefore, we propose, instead of reporting all frequent items, to only mine the top- $k$  with the highest max-frequency. First we prove that in order to solve this problem exactly, again we need a prohibitive amount of memory (at least linear in the number of items). This negative result motivates why our study is extended to finding efficient approximation algorithms for the top- $k$  problem.
- Under some reasonable conditions, we show both theoretically and empirically that memory-efficient algorithms exists. Based on how much knowledge we have about the distribution generating the data stream, different algorithms are given.
- Prototype of the algorithms have been implemented and we present its memory-efficiency performance on real-life data and in controlled experiments with synthetic data.

The organization of the paper is as follows. Section 2 revisits some background knowledge of the recently proposed Max-Frequency measure. Section 3 presents the theoretical aspects of the top- $k$  frequent items mining problem in a data stream with flexible windows. An approximate algorithm is presented in this section with theoretical analysis to solve the top- $k$  mining problem effectively. In Section 4, independently from data distribution, a practical approximate algorithm is presented. Experimental results conducted with real-life datasets in Section 5 show that the memory requirements for the approximate algorithm are extremely small compared to the straightforward approach. Nevertheless, the accuracy of the top- $k$  results is preserved with high probability. Section 6 concludes the paper and presents possible extensions of the work.

## 2. BACKGROUND AND PRELIMINARIES

In this paper we use the following simple yet expressive stream model. We assume the existence of a countable set of items  $\mathcal{I}$ . A stream  $S$  over  $\mathcal{I}$  is a, possibly infinite, sequence of items from  $\mathcal{I}$ . We will adopt the notations given in Table 1.

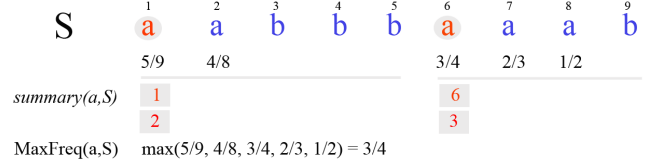
**DEFINITION 1 (MAX-FREQUENCY).** *Let  $S$  be a stream,  $s$  be an item, and  $n$  a positive integer. The max-frequency of  $s$  in  $S$  at time  $n$ , denoted  $MaxFreq(s, S_n)$  is defined as:*

$$MaxFreq(s, S_n) := \max_{k=1 \dots n} freq(s, S_{k,n})$$

*The smallest  $k$  that maximizes the fraction is called the maximal border and will be denoted  $MB(s, S_n)$ :*

**Table 1: Summary of Notations**

Notations	Descriptions
$s_i$	the $i^{th}$ element in the sequence
$S_{j,n}$	the sub-stream $\langle s_j, \dots, s_n \rangle$ of $S$
$S_n$	the sequence $S_{1,n}$
$count(s, S_n)$	the number of instances of $s$ in $S_n$
$\otimes_k(s)$	the sequence $ss \dots s$ with length $k$
$\epsilon$	an empty sequence
$freq(s, A)$	Relative Frequency of item $s$ in the suffix $A$ of $S$ : $freq(s, A) = \frac{count(s,A)}{ A }$



**Figure 1: Border points of  $a$  in  $S$  correspond to the highlight positions.  $Summary(a, S)$  and  $MaxFreq(a, S)$  are also shown in this figure.**

$$MB(s, S_n) := \min \operatorname{argmax}_{k=1 \dots n} freq(s, S_{k,n})$$

In other words, the max-frequency is defined as the maximal relative frequency of  $s$  in a suffix of  $S_n$ . For example, in Figure 1 we can see a stream of items  $S_n$  ( $n = 9$ ). The first ever occurrence of  $a$  is associated with a relative frequency equal to  $\frac{count(a, S_{1,9})}{|S_{1,9}|} = \frac{5}{9}$ . Similarly, the second occurrence of  $a$  is associated with a relative frequency equal to  $\frac{count(a, S_{2,9})}{|S_{2,9}|} = \frac{4}{8}$ . With every occurrence of  $a$  in  $S_n$  a relative frequency is associated. In this case  $MaxFreq(a, S_9) = \frac{3}{4}$  and the maximal border is  $MB(a, S_9) = 6$  (the third  $a$  from the beginning of the item stream). It is worth noting that the more traditional sliding window approaches consider only one prefix of a user-defined fixed length.

Obviously, for most data streams it is impossible to keep the whole stream into memory and to check, at every timestamp  $n$ , all suffixes of  $S_n$  in order to find the suffix which gives the maximal frequency. Luckily, however, not every point can become a maximal border. Points that can still potentially become maximal borders are called the *border points*:

**DEFINITION 2 (BORDERS).** *An integer  $1 \leq j \leq n$  is called a border for  $s$  in  $S_n$  if there exists a continuation of  $S_n$  in which  $j$  is the maximal border. The set of all borders for  $s$  in  $S_n$  will be denoted  $\mathcal{B}(s, S_n)$ :*

$$\mathcal{B}(s, S_n) := \{j \mid 1 \leq j \leq n \mid \exists T \text{ s.t. } S_n \text{ is a prefix of } T \text{ and } MB(s, T) = j\}$$

In Calders et. al. [1] the following theorem exactly characterizing the set of border points was proven:

**THEOREM 1 (CALDERS ET. AL. [1]).** *Let  $S$  be a stream,  $s$  be an item, and  $n$  a positive integer.  $j$  is in  $\mathcal{B}(s, S_n)$  if and only if for every suffix  $B$  of  $S_{j-1}$  and every prefix  $A$  of  $S_{j,n}$ ,*

$$freq(s, B) < freq(s, A)$$

Intuitively, the theorem states that a point  $j$  is a border for item  $a$  if and only if for any pair of blocks where the first

**Table 2: Border set summary**

$j_1$	$j_2$	...	$j_b$
$count(s, S_{j_1, j_2-1})$	$count(s, S_{j_2, j_3-1})$	...	$count(s, S_{j_b, n})$

block lies Before and the second After point  $j$ , the frequency of  $a$  in the first block should be lower than in the second one. For example, we can take a look at the data stream in Figure 1 again. Let us consider the fourth occurrence of  $a$  corresponding to the seventh position in the stream  $S_9$ . If we choose  $B = S_{6,6}$  and  $A = S_{7,9}$  we have  $freq(a, B) = 1 > freq(a, A) = \frac{2}{3}$  which does not satisfy the condition in Theorem 1 and hence this occurrence of  $a$  is not a border point. According to Theorem 1, of the 5 instances of  $a$  in  $S_9$  only the two highlighted positions could ever become maximal borders. As was shown in [1], this property is a powerful pruning criteria effectively reducing the number of border points that need to be checked.

Clearly, if  $j$  is not a border point in  $S_n$ , then neither it is in  $S_m$  for any  $m > n$ . Therefore, in [1], the following summary of  $S_n$  is maintained and updated whenever a new item arrives: let  $j_1 < \dots < j_b$  be the borders of  $s$  in  $S_n$ . The border set summary is depicted in Table 4. This summary can be maintained easily, is typically very small, and the max frequency can be derived immediately from it. For example, in Figure 1 we see the border set summary of item  $a$  in  $S_9$ .

The following theorem allows for reducing the border set summary even further, based on a minimal support threshold:

**THEOREM 2** (CALDERS ET. AL. [1]). *Let  $S$  be a stream,  $s$  be an item,  $n$  a positive integer, and  $j$  in  $\mathcal{B}(s, S_n)$ . Let  $T$  be such that  $S_n$  is a prefix of  $T$ , and  $j$  is the maximal border of  $s$  in  $T$ . Then,  $MaxFreq(s, T) \leq freq(s, A)$  for any prefix  $A$  of  $S_{j,n}$ .*

Hence, if there exists a block  $A$  starting on position  $j$  such that the frequency of  $s$  in  $A$  does not exceed a minimal threshold  $minfreq$ ,  $j$  can be pruned from the list of borders because whenever it may become the maximal border, its max-frequency won't exceed the minimal frequency threshold anyway.

### 3. THEORETICAL RESULTS

In this section, we are going to investigate some theoretical aspects of the *mining top-k frequent items with flexible windows problem*. First, we will prove that the number of border points in a data stream with multiple items can increase along with the data stream size and therefore this number will eventually reach the memory limit of any computing system. As a result, it is emerging to design a memory-efficient algorithm that maintains just partial information about the border point set while it is still able to answer the top-k queries with high accuracy.

Besides, we will also show that any deterministic algorithm solving our interested problem exactly requires a memory space with size being linear in the number distinct items of the stream. Thus, under the context that the system memory is limited, we need to design either a randomized exact algorithm or an approximate deterministic algorithm.

We propose a deterministic algorithm to approximate the problem's solutions. More importantly, we will show that

the proposed algorithm theoretically uses less memory than the straightforward approach storing the complete set of border points while it is able to answer the top-k queries with surprisingly high accuracy. "High accuracy" here refers to that the algorithm reports the top-k answers at any time point with a bounded low probability of having false negatives, without any false positives and the order between the items in the reported top-k list is always correct.

#### 3.1 The Number of Border Points

We first start by showing that there exists an item stream such that the number of border points increases along with the data stream size regardless how many distinct items in the data stream are. Hence, when the item stream evolves the number of border points will eventually reach the memory limit of any computing system.

Assume that we have a set of  $m + 1$  distinct items  $\mathcal{I} = \{a_1, a_2, \dots, a_m, b\}$  and let us denote  $w_l$  as the sequence  $\otimes_l(a_1) \otimes_l(a_2) \dots \otimes_l(a_m)b$ . Consider the following data stream:  $W_k = w_0 w_1 w_2 \dots w_k$  for  $k \geq 1$ . The data stream size  $|W_k|$  is equal to  $m \frac{k(k+1)}{2} + k + 1$ . The following lemma gives the number of border points of the data stream  $W_k$ :

**LEMMA 1.** *Data stream  $W_k$  has exactly  $mk + 1$  border points for any  $k \geq 1$*

**PROOF.** First, regarding the item  $b$  there is a unique border point corresponding to the position of the first occurrence of  $b$  in  $W_k$ . Moreover, we will prove that for every item  $a_i$  there are exactly  $k$  border points corresponding to the first elements of the sequences  $\otimes_j(a_i)$ , for  $j = 1, 2, \dots, k$ . These  $k$  occurrences of the item  $a_i$  divide  $W_k$  into  $k + 1$  portions.

For instance, let us consider a simple case with  $W_k = b a_1 a_2 a_3 b a_1 a_1 a_2 a_2 a_3 a_3 b a_1 a_1 a_1 a_2 a_2 a_2 a_3 a_3 a_3 b$ , that is when  $k = 3$  and  $m = 3$ . The three first occurrences of  $a_1$  in every sequence  $\otimes_j(a_1)$  divide  $S_k$  into 4 portions (separated by  $|$ ):  $b|a_1 a_2 a_3 b|a_1 a_1 a_2 a_2 a_3 a_3 b|a_1 a_1 a_1 a_2 a_2 a_2 a_3 a_3 a_3 b$ .

Let  $P_{i0}, P_{i1}, \dots, P_{i(k+1)}$  be the portions. The number of instances of the item  $a_i$  inside the portion  $P_{ij}$  is  $j$  and the size of  $P_{ij}$  is equal to  $m \cdot j + i$ . Hence, for every  $a_i$  we have a partition of  $S_k$  corresponding to a strictly increasing sequence of fractions:

$$0 < \frac{1}{m+i} < \frac{2}{2m+i} < \dots < \frac{k}{mk+i} \quad (1)$$

By the result of the paper [1], we can imply that all the considered positions correspond to the border points of the items  $a_i$  in  $W_k$ . Thus, for every item  $a_i$  we have exactly  $k$  border points and thus in total  $W_k$  has exactly  $m \cdot k + 1$  border points.  $\square$

A direct consequence of this lemma is that given a fixed number of distinct items  $m$  when the data stream evolves, the number of border points also evolves along with  $k$ . It is important to note that the upper bound on the number of border points presented in the paper [1] is only for a single item, from this result it is not easy to derive a similar upper bound for the multiple-item case. The result in this section is not as strong as the result presented in [1] for the single-item case. However, for the multiple-item case, it is enough to support the fact that there is no modern computing system being able to store the complete set of border points inside its limited memory when  $k$  becomes extremely large.

## 3.2 The Least Space Requirement for Deterministic Exact Algorithms

In this section, we will derive a lower bound on the memory usage that every deterministic algorithm will need in order to solve the top- $k$  frequent items mining problem exactly. In particular, if we assume that the data stream has at least  $m$  distinct items we can show that there is no deterministic algorithm solving the top- $k$  problem exactly with memory less than  $m$ . By the terminology deterministic algorithm we mean the model in which whenever a new item arrives in the data stream the algorithm has to decide whether it needs to evict an existing border point in the memory or just ignore it deterministically.

The main theoretical result of this section is shown in the following lemma:

**LEMMA 2.** *Let  $m$  be the number of distinct items in the data stream. If the system memory limit is  $m - 1$ , there is no deterministic algorithm being able to answer the top- $k$  ( $k > 1$ ) frequent items queries exactly all the time even for the special case  $k = 2$ .*

**PROOF.** First of all, at a time point  $t$ , if the system has information about an item we call it a *recorded item* and otherwise it is called a *missing item*. Given a data stream, the most recent item in the data stream is always the highest MaxFreq item, so in order to answer the top-2 query exactly this item must be recorded as long as it arrives in the stream.

Therefore, let us consider the following data stream with  $m$  distinct items:  $S = \otimes_m(a_m) \otimes_{m-1}(a_{m-1}) \cdots \otimes_1(a_1)$ . In this data stream, it is clear that at the moment  $a_1$  must be a recorded item. Since we have assumed that the system has limited memory which is less than  $m$  there is always at least one missing item in  $S$ . Without loss of generality we assume that  $a_n$  ( $n \neq 1$ ) is the missing item.

We extend the data stream  $S$  with  $l \geq 1$  instances of the item  $a_1$ , s.t.  $S = \otimes_m(a_m) \otimes_{m-1}(a_{m-1}) \cdots \otimes_2(a_2) \otimes_{l+1}(a_1)$ . In doing so, a deterministic algorithm does not have any information about  $a_n$  so far, hence, the item  $a_n$  remains missing for any value of  $l$ .

Moreover, every item  $a_i$  for  $i = 2, 3, \dots, m$  has a unique border point. This point corresponds to the maximum point of  $a_i$  the MaxFreq of which is equal to  $\frac{i}{\frac{i(i+1)}{2} + l}$ . We will prove that there exists  $l$  such that  $a_n$  can become the second highest MaxFreq. In fact, in order to prove this, we have to show that the following inequalities are true for some value of  $l$  for any  $i \neq n$  and  $n \geq 2$ :

$$\frac{n}{\frac{n(n+1)}{2} + l} > \frac{i}{\frac{i(i+1)}{2} + l} \quad (2)$$

We rewrite the above inequality as follows:

$$(n - i)(l - \frac{ni}{2}) > 0 \quad (3)$$

If  $l$  satisfies  $\frac{n(n+1)}{2} > l > \frac{n(n-1)}{2}$  (when  $n \geq 2$  there always exists such  $l$ ) then the inequality (2) is true for all  $i \neq n$  and  $n \geq 2$ . Hence, with such value of  $l$  the item  $a_n$  becomes the second highest MaxFreq. On the other hand,  $a_n$  so far is not a recorded item so it will be missing in the top-2 list reported by every deterministic algorithm.  $\square$

Lemma 2 allows to derive a lower bound as large as  $m$ , which is the number of distinct items in the stream, on the

memory usage of any deterministic algorithm for solving the top- $k$  problem exactly. When  $m$  is greater than the memory limit, solving the problem exactly with a deterministic exact algorithm is no longer possible. Therefore, in the following sections we will focus on approximate approaches which are memory-efficient.

## 3.3 Effective Approximate Algorithm

In this subsection we will propose an approximate algorithm for the top- $k$  frequent items mining problem. We show that the proposed algorithm can answer top- $k$  queries with high accuracy and consumes less memory than the straightforward approach of storing the complete set of border points. First, we assume that the item distribution in the data stream is known in advance and does not change over time. We make this assumption to simplify the analysis of the proposed algorithm. For the cases with unknown data distribution we propose another algorithm in the next section.

Prior to the description of the approximate algorithm we revisit a useful property of the border points stated in Theorem 2, Section 2. According to this theorem, if the tight lower bound on the MaxFreq of the top- $k$  frequent items is known in advance, then we can safely prune any border point  $p$  of an item  $a$  which has frequency being strictly less than this bound without effect on the accuracy of the top- $k$  results. We call the value that we use to do pruning the *pruning threshold*. Obviously, zero is a feasible lower bound. However, it is not a meaningful pruning threshold, because there is no border point with relative frequency being less than this threshold.

Indeed, let us revisit the data stream  $S$  shown in the proof of Lemma 2. If we extend the data stream  $S$  with  $l$  instances of  $a_1$  and let  $l$  go to infinite we will have a data stream in which the only feasible non-negative lower bound on the MaxFreq of the top- $k$  items is zero, in other words, there is no meaningful pruning threshold for this data stream. Thus, it is impossible to devise a meaningful pruning threshold value for every data stream such that there is no accuracy loss in the top- $k$  results. However, if the item distribution is known in advance we can estimate a good pruning threshold such that the accuracy loss is negligible. We first start with the following lemma:

**LEMMA 3.** *Given a time point  $n$  and a parameter  $k$ , we assume that  $Y_n$  is the size of the smallest suffix of the item stream  $S_n$  that contains exactly  $k$  distinct items. The top- $k$  frequent items of the data stream  $S_n$  have MaxFreq at least as big as  $\frac{1}{Y_n}$ .*

**PROOF.** Since the window with size  $Y_n$  contains  $k$  distinct items their relative frequency in this window is at least  $\frac{1}{Y_n}$ . On the other hand, the top- $k$  frequent items are always at least as frequent as the least frequent item in this window so the Lemma 3 holds.  $\square$

The consequence of Lemma 3 is that at every time point the reciprocal of the smallest suffix of  $S_n$  containing exactly  $k$  distinct items is the lower bound on the MaxFreq of the top- $k$  frequent items. This lower bound is not a fixed value but it may change when the data stream evolves. Let the size of the smallest suffix containing exact  $k$  distinct items be denoted by the random variable  $X_k$ . Estimating the expected value of  $X_k$  is well-known in the literature as the classical

---

**Algorithm 1** *MeanSummary*( $k, l$ )

---

```
1:  $\mathcal{B} \leftarrow \emptyset$ 
2: while New item  $a$  arrives do
3:   if previous occurred item is not  $a$  then
4:     create a new border point for  $a$  and include it into  $\mathcal{B}$ 
5:   end if
6:   Delete all the elements in  $\mathcal{B}$  that are no longer border points
7:   Update relative frequency of all elements in  $\mathcal{B}$ 
8:   Delete all the border points in  $\mathcal{B}$  that have relative frequencies strictly less than  $\beta = \frac{1}{lE(X_k)}$ 
9: end while
```

---

*Coupon Collector Problem* (CCP) [7]. We will revisit this problem later in the analysis part. At this point, we will start with the description of the MeanSummary algorithm summarizing the data stream as in Algorithm 1.

Recall that  $X_k$  is the random variable standing for the size of the smallest window containing exact  $k$  distinct items. Algorithm 1 assumes that  $E(X_k)$  is known in advance for any  $k$  and uses  $\frac{1}{lE(X_k)}$  as a pruning threshold, where  $k$  and  $l$  are two user-defined parameters. Having a stream summary the system just needs to take the  $k$  highest relative frequency items present in this summary and report this list as the answer to the top- $k$  query. To understand how precise the top- $k$  list produced by MeanSummary is we make some analyses in the subsequent part.

### 3.3.1 Accuracy Analysis

LEMMA 4. *Given two positive integers  $k$  and  $l$ , independently of the item distribution in the data stream, the probability that a top- $k$  item is missing in the answer list reported by MeanSummary is less than  $\frac{1}{l}$ .*

PROOF. Given a time point, since  $\frac{1}{X_k}$  is the lower bound on the MaxFreq of the top- $k$  frequent items, the event that the least frequent item in the top- $k$  frequent items has its MaxFreq less than  $\frac{1}{lE(X_k)}$  is less probable than  $\frac{1}{X_k} \leq \frac{1}{lE(X_k)}$ . On the other hands, according to the Markov's inequality  $Pr(X_k \geq lE(X_k)) \leq \frac{1}{l}$  which proves the lemma.  $\square$

By the result of Lemma 4, the probability of having false negatives is less than  $\frac{1}{l}$ , independently of the item distribution. In the next section we will derive a better bound for this type of error when the data set follows the uniform distribution in which the expectation and the variance of  $X_k$  are known.

Finally, we consider another interesting property of the proposed stream summary: MeanSummary is able to answer top- $k$  queries without false positives, that is, the reported top- $k$  list is always a subset of the right answer and the item order in the reported top- $k$  list is preserved.

LEMMA 5. *Using MeanSummary we are able to answer the top- $k$  queries without false positive, moreover, the item order in the reported top- $k$  list always correct.*

PROOF. Given a time point, assume that  $a$  is the  $k - th$  most frequent item of the item stream  $S$ . If  $a$  is not present in MeanSummary, that is, when  $MaxFreq(a, S) < \beta$ , the other items less frequent than  $a$  must also be absent in the summary. In this case, the top- $k$  list produced by taking

only items present in the summary will not contain any other items than the real top- $k$  items.

On the other hand, if  $a$  is present in the summary, that is when  $MaxFreq(a, S) > \beta$  and the MaxFreq of other top- $k$  frequent items must also larger than  $\beta$ , therefore, the border points corresponding to the maximum points of these items must be also present in the summary. In other words, the summary will report the right MaxFreq of them, and hence these items remain in the top- $k$  of the summary and are all present in the answer list.

Since the MaxFreqs of the top- $k$  items we report based on the data stream summary are always exact, the items in the answer list will always be in the correct order.  $\square$

### 3.3.2 Uniform Distribution

Algorithm 1 uses prior knowledge about the item distribution to define the pruning threshold. In particular, it requires the expected value of  $X_k$  for any  $k > 1$ . Estimating  $E(X_k)$  is well-known as the classical Coupon Collector Problem [7]. Unfortunately, to the best of our knowledge, there is no post work being able to present the value of  $E(X_k)$  in a closed form for all types of distributions [4, 8]. Indeed, in [4], the authors have tried to present  $E(X_k)$  in form of integral formulae which can be approximately estimated by classical numerical methods. These methods however are computationally demanding, especially when  $k$  is large [4]. Estimation of this expected value for any type of item distribution is out of the scope of this paper.

Fortunately, for the uniform distribution a simple presentation of  $E(X_k)$  is well-known [7, 4]. In addition to that the variance of  $X_k$ , denoted by  $\sigma_k^2$ , has a closed formula. Having knowledge of the expected value and the variance of  $X_k$  we can derive a tighter bound on the false negative error for this particular distribution as follows:

LEMMA 6.  *$Pr(X_k \geq lE(X_k)) \leq \frac{1}{(l-1)^2+1}$  for any  $l > 1$  and  $k < \frac{m}{2}$ .*

PROOF. First, we have the well-known formulae [7, 4] of  $E(X_k)$  and  $\sigma_k^2$ :

$$E(X_k) = \sum_{i=0}^{k-1} \frac{m}{m-i} \quad (4)$$

$$\sigma_k^2 = \sum_{i=0}^{k-1} \frac{mi}{(m-i)^2} \quad (5)$$

Recall that  $m$  is the number of distinct items in the data stream and that it is much larger than  $k$ . Since  $k < \frac{m}{2}$ , for any  $i < k$  we have  $\frac{i}{m-i} < 1$  and  $\sigma_k < \sqrt{E(X_k)} < E(X_k)$ .

By the one-sided version of Chebyshev's inequality we get:  $Pr(X_k - E(X_k) \geq (l-1)\sigma_k) \leq \frac{1}{(l-1)^2+1}$ , which implies  $Pr(X_k \geq lE(X_k)) \leq \frac{1}{(l-1)^2+1}$ .  $\square$

Lemma 6 gives a tighter bound on the false negative than the bound in Lemma 4: the false negative probability is less than  $Pr(\frac{1}{X_k} \leq \frac{1}{lE(X_k)}) = Pr(X_k \geq lE(X_k))$ , hence, less than  $\frac{1}{(l-1)^2+1}$ .

### 3.3.3 Experiment with Uniform Data

In order to demonstrate the effectiveness of the proposed algorithm we have carried out an experiment on a synthetic data set which follows the uniform distribution. We simulate

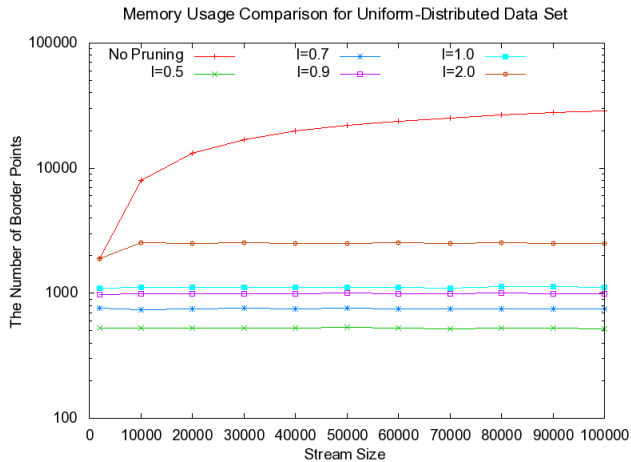


Figure 2: The comparison of memory usage in terms of the number of border points each algorithm has to store in memory. The plot shows how this number evolves when the data stream evolves. MeanSummary memory usage is shown for different values of  $l$ . It is clear from the figure that MeanSummary uses significantly less memory than the complete set of border points (No Pruning).

Table 3: Recall (%) of the top-1 000 answer produced by MeanSummary

Value of $l$	$l = 0.5$	$l = 0.7$	$l = 0.9$	$l = 1.0$	$l = 2.0$
Min	50.1	71.1	92.5	100	100
Average	51.2	72.4	94.5	100	100
Max	52.4	74.1	96.2	100	100

an item stream with 10 000 uniformly distributed distinct items until the stream size reaches the number 100 000. We have measured the performance of MeanSummary in terms of memory consumption and the quality of the top-1 000 answer. The results are reported in Figure 2 and Table 3.

According to Lemma 5, MeanSummary does not cause any false positives so it always produces the top- $k$  answer with maximum *Precision Value*, i.e. 100%. On the other hand, MeanSummary may produce false negatives which are usually measured by the *Recall Value*, i.e. the fraction of the number of real top- $k$  items reported by MeanSummary. In Table 3 we show the Average Recall of the top-1 000 answers produced by MeanSummary over different pruning thresholds when the data stream evolves. It is clear from the context that when the value of  $l$  is higher, MeanSummary is able to answer the top-1 000 queries more accurately on average. The average recall reaches its maximum value i.e. 100% when  $l$  is above 1. In order to show the stability of the obtained results we also report the Maximum and the Minimum Recall values of the top-1 000 answers in this table. It is clear from the context that the obtained results are also quite stable as the differences between the minimum, the maximum recall and the average recall are not large.

It is important to note that the higher the value of  $l$ , the lower the pruning threshold becomes, resulting in less border points being pruned. In other words, when  $l$  is high Mean-

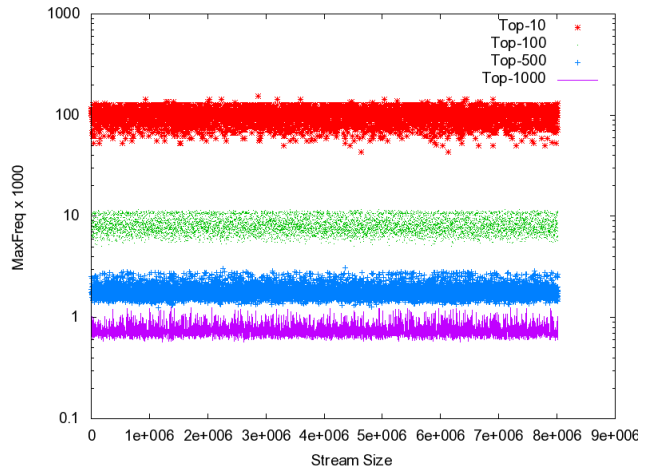


Figure 3: The MaxFreq of the  $k$ -th most frequent item of the Kosarak data stream over time.

Summary may use more memory but it will produce more accurate answers. This is a tradeoff between result quality and memory usage. In order to show the effectiveness of MeanSummary in terms of memory usage we plot the number of border points that MeanSummary has to store over different values of the pruning threshold in Figure 2. It is clear that when  $l$  is higher, more memory space is required. Fortunately, the number of border points that MeanSummary has to store seems to be bounded regardless of the stream size. Concretely, at the end of the algorithm execution the number of border points that MeanSummary has to store when  $l = 2.0$  is always less than 3 000 which is almost ten times smaller than the total number of border points of the data stream (28 793). It is important to notice that when  $l = 2.0$ , MeanSummary produces no errors at all for the top-1 000 queries (see Table 3). This fact emphasizes the extreme significance of MeanSummary.

#### 4. PRUNING ALGORITHM IN PRACTICE

We have seen the effectiveness of using MeanSummary in Algorithm 1 to answer the top- $k$  queries with the assumption that the expected value of  $X_k$  could be computed in advance. However, in practice, this effort may not be possible due to non-trivial expected value computation. Therefore, using  $\frac{1}{\mathbb{E}(X_k)}$  as a pruning threshold is impractical when the data distribution is unknown in advance or the data distribution changes over time. Even in the case that the data distribution is supposed to be known in advance, for instance, with the Zipfian distribution, there is no closed representation of  $E(X_k)$  that is easy to compute [4].

For the situation that the expectation of  $X_k$  is unknown, we now propose another summary algorithm which still has similar properties as MeanSummary. Before continuing with a detailed description of the proposed summary in Algorithm 2, we explain the crucial idea behind our proposal in Figure 3. In this figure, we plot the MaxFreq (multiplied by 1 000) of the  $k$ -th most frequent item of the Kosarak data stream (see section 5.1 for information about this dataset). Four lines correspond to different values of  $k$  from which we can observe that they are quite well-separated from each other. Intuitively, if we consider the upper bound on the top-100

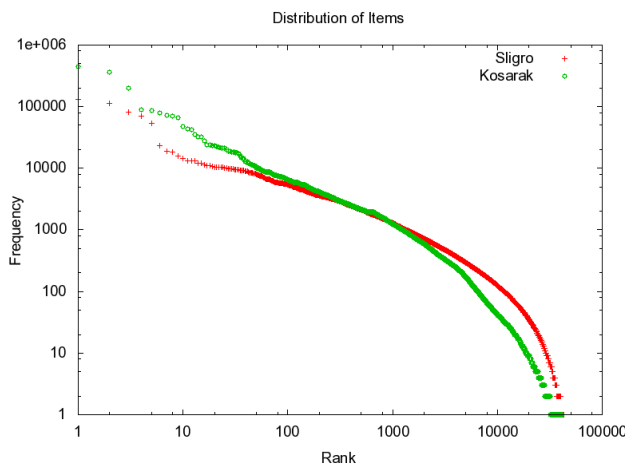
---

**Algorithm 2** *MinSummary*( $k, l$ )

---

```
1:  $\mathcal{B} \leftarrow \emptyset$ 
2:  $\beta \leftarrow 0$ 
3: while New item  $a$  arrives do
4:   if previous occurred item is not  $a$  then
5:     create a new border point for  $a$  and include it into
        $\mathcal{B}$ 
6:   end if
7:   Delete all the elements in  $\mathcal{B}$  that are no longer border
       points
8:   Update frequency of all elements in  $\mathcal{B}$ 
9:   Delete all the elements in  $\mathcal{B}$  that have frequencies
       strictly less than  $\beta$ 
10:   $\alpha \leftarrow$  MaxFreq of the least frequent item in  $\mathcal{B}$ 
11:   $|\mathcal{B}| \leftarrow$  the number of distinct items in  $\mathcal{B}$ 
12:  if  $|\mathcal{B}| \geq l$  AND  $\alpha > \beta$  then
13:     $\beta \leftarrow \alpha$ 
14:  end if
15: end while
```

---



**Figure 4: The Kosarak and Sligro data-sets follow Zipfian distribution but with different levels of skewed.**

line as a pruning threshold we can prune a lot of border points while the top-10 items are warranted to be present in the summary with high probability.

The aforementioned observation from the Kosarak data set is intuitive, allowing us to propose a summary algorithm which is then shown to be effective in the experiments with real-life datasets. The approach is briefly described in Algorithm 2. *MinSummary* is similar to *MeanSummary*, the only difference is that it uses a dynamic pruning threshold  $\beta$  instead of a static value. This threshold is updated in each step such that its value is monotonically increasing, thus, more border points are pruned in the further steps. The algorithm admits two user-defined parameters  $k$  and  $l$ . The bigger the value of  $l$ , the bigger the summary will become, but the more precise the top- $k$  queries will be answered.

In order to prevent  $\beta$  from increasing too fast we only update the value of  $\beta$  when the summary  $\mathcal{B}$  contains at least  $l$  different items. By doing so, we keep  $\beta$  increasing but always less than the upper bound of the  $l$ -th most frequent item. As a result,  $\beta$  is less than the frequency of  $k$ -th most frequent item with high probability as explained in the aforemen-

**Table 4: Data Sets Summary**

Data Sets	N. Trans	Stream Size	Size (MB)	N. Items
Kosarak	990002	8019015	32 MB	41269
Sligro	542194	7671058	38 MB	43082

tioned intuitive example. The following lemma shows that *MinSummary* has similar properties as *MeanSummary*:

**LEMMA 7.** *Using *MinSummary* we are able to answer top- $k$  queries without false positives and the item order in the reported top- $k$  items is preserved.*

It is important to note that  $\beta$  in *MinSummary* is kept increasing over time to let *MinSummary* have similar property like *MeanSummary*. In doing so, the proof of Lemma 7 proceeds in a very similar way as the proof of Lemma 5.

## 5. EXPERIMENTS AND RESULTS

### 5.1 Data sets

We use two real world data sets to conduct our experiments. The characteristics of them are summarized in Table 4. The Kosarak is a publicly available dataset<sup>1</sup> containing click-stream data of a Hungarian online news portal while the Sligro data set is released under restricted conditions containing information about products purchased by Sligro company’s customers in a specific city in the Netherlands from August 2006 to October 2008.

Both data sets follows a Zipfian distribution as we can see in Figure 4 in which we plot the item frequency (vertical axis) from the most frequent to the least frequent items (horizontal axis). Generally, the Sligro data set follows a Zipfian distribution but the occurrence of every specific item varies in different periods of the year. Some items may be suddenly frequent in a short period of time and may completely disappear in another period, e.g. seasonal products. We conduct the experiment on the Sligro dataset to see the behavior of *MinSummary* in the context of rapidly changing frequencies.

### 5.2 Accuracy of the Top-k Answer

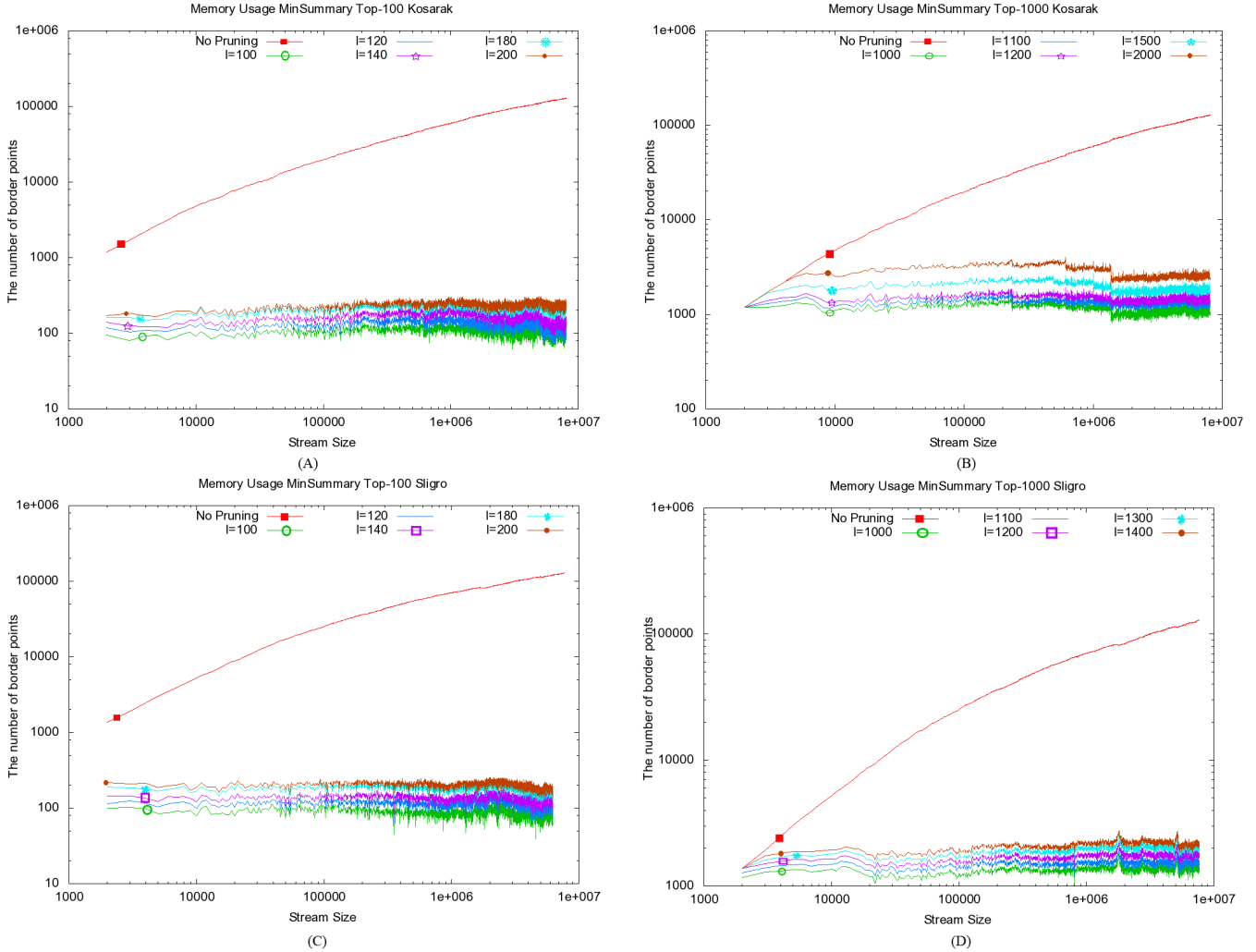
It is worth noting that by the results of Lemma 7, *MinSummary* always produces the top- $k$  list with the maximum precision value, i.e. 100%. So in order to measure the accuracy of *MinSummary* we just need to measure the recall values of the top- $k$  lists.

Concretely, each time when a new item arrives in the data stream we do querying from *MinSummary* for the top- $k$  frequent items. The obtained top- $k$  list from *MinSummary* will then be compared with the true top- $k$  set to estimate the recall value. We average the recall value over time and show the results in Table 5. We also present the maximum and the minimum recall values in this table to see the deviation of the recall from its average value.

We present results for different values of the parameters. The value of  $k$  is set to 100 and 1000 respectively while  $l$  is set to  $k$  and higher. Recall that  $l$  is a parameter that controls the memory usage in *MinSummary*. The higher value of  $l$  the more memory is used to maintain *MinSummary*.

In Table 5 we can observe that whenever  $l$  is increased the obtained top- $k$  list is more accurate. In particular, we are

<sup>1</sup><http://fimi.cs.helsinki.fi/data/>



**Figure 5: A comparison of memory usage in terms of the number of border points each algorithm has to store in memory. The plot shows how this number evolves when the data stream evolves. MinSummary’s memory usage is shown with different values of  $l$ . It is clear from the figure that MinSummary uses significantly less memory than the case with no border points are being pruned (No Pruning).**

able to reach the maximum recall value with proper choice of  $l$ . The obtained results with higher values of  $l$  are also very stable since the deviation of maximum and minimum recall from its average value is negligible. In summary, using MinSummary with a proper choice of its parameters we are able to answer the top- $k$  query with very high accuracy for these particular real world datasets.

### 5.3 Evolution of the Pruning Threshold

In order to illustrate the relation between the pruning threshold and the performance of MinSummary, we plot the value of  $\beta$  with different setups of parameter  $l$  in Figure 6. We also plot the MaxFreq of the  $k$ -th highest frequent item in the stream corresponding to  $k = 100$  and  $k = 1000$ .

It is clear from the context that  $\beta$  starts with a very low value and increases every time a new item arrives in the stream. With a proper choice of  $l$  we can let  $\beta$  increase up to a tight lower bound on the MaxFreq of the top- $k$  frequent items. In doing so, as the pruning threshold grows, we prune

more border points while preserving the accuracy of the top- $k$  answers. Concretely, assume that we intend to answer the top-1000 frequent item query,  $l = 2000$  or  $l = 3000$  are the proper choices because in these cases  $\beta$  (lines “ $l=2000$ ” and “ $l=3000$ ”) increases up to the lower bound on the MaxFreq of the top-1000 frequent items (line “Top-1000”). Obviously, when  $l = 3000$  we have a more accurate answer because there is no overlap between the line “ $l=3000$ ” and the line “Top-1000”, but MinSummary will consume more memory as compared to the case  $l = 2000$ . For top-100 queries it is clear that  $l = 1000$  is a very good choice.

### 5.4 Memory Usage

Following section 5.2 about the accuracy of the obtained top- $k$  lists, we plot the memory usage of MinSummary in Figure 5. In each plot we compare the size of MinSummary in terms of the number of border points that it has to store with the complete set of border points of the data stream (No Pruning line). We can observe that the size of Min-

Table 5: Recall value of top-k answers produced by MinSummary

		Kosarak					Sligro				
$k = 100$	Value of $l$	100	120	140	180	200	100	120	140	180	200
	Min	36	49	60	84	100	21	43	60	97	100
	Avg.	66.8	78.2	88.4	98.8	100	67.9	85.4	96.1	99	100
	Max	96	100	100	100	100	99	100	100	100	100
$k = 1000$	Value of $l$	1 000	1 100	1 200	1 500	2 000	1 000	1 100	1 200	1 300	1 400
	Min	42.8	53.4	56.8	70.1	92.5	61.1	85.9	100	100	100
	Avg.	60.3	65.7	70.7	85.6	99.4	83.7	91.5	100	100	100
	Max	100	100	100	100	100	99.9	100	100	100	100

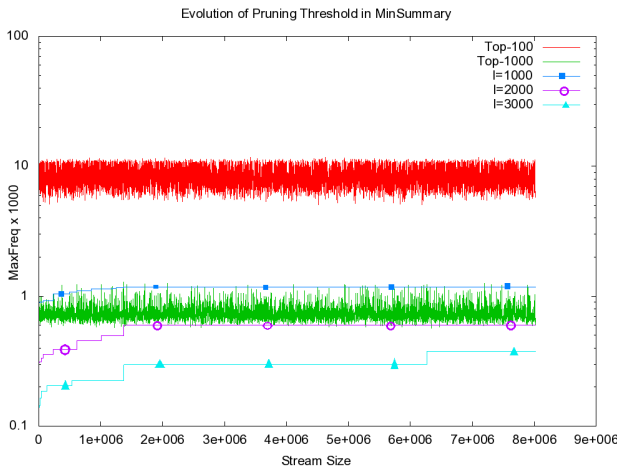


Figure 6: Evolution of the pruning threshold  $\beta$  over time.  $\beta$  starts with low value and increases up to the tight lower bound of the top-k MaxFreq

Summary increases with increasing  $l$ . Yet, if we compare these value with the complete set of border points we see that MinSummary consumes significantly less memory and the memory consumption seems to be independent from the stream size.

For instance, consider the case in Figure 5.B in which we plot the memory usage of the MinSummary with the Kosarak data stream and  $k = 1000$ . When the program finishes its execution, MinSummary stores less than 4000 border points for all values of  $l$  while the complete set of border points still keeps evolving and reaches its highest value 128 249. That means MinSummary is 300 times more memory-efficient than the straightforward approach. Moreover, we have seen in Table 5 that when  $l = 2000$ , MinSummary produces top-k answers with negligible errors. This fact confirms the significance of MinSummary in comparison with the straightforward approach.

The results with the Sligro dataset in Figure 5.C and 5.D confirm again the significant performance of MinSummary. Another important point we can observe is that given a value of  $l$ , the size of MinSummary seems to be bounded regardless of the stream size. This means when the stream evolves the set of the border points will eventually reach any limit but MinSummary size will not.

## 6. CONCLUSIONS AND FUTURE WORK

In this paper we have shown that an exact deterministic algorithm for computing the top- $k$  MaxFreq items inher-

ently has to use an amount of memory at least as large as the number of distinct items in the data stream. Thus, an exact deterministic algorithm is not feasible for applications with limited memory such as, e.g. mobile devices, sensor networks. However, fortunately memory-efficient approximate algorithm exists. We have proposed such an algorithm using significantly less memory as compared to the straightforward approach while preserving the high accuracy of the top-k results. The experiment conducted with real-life and syntactic datasets confirms the significance of the proposed algorithm. For future work we will consider extending the approach to the top-k frequent itemsets mining problem in the transaction stream with sliding windows. Another possible extension is considering the problem with a minimum windows constraint [2].

## Acknowledgement

We would like to thank Frank Takes and the anonymous reviewers for their useful comments to improve our work.

## 7. REFERENCES

- [1] T. Calders, N. Dexters, and B. Goethals. Mining frequent itemsets in a stream. In ICDM, pages 83-92, 2007.
- [2] T. Calders, N. Dexters, and B. Goethals. Mining frequent items in a stream using exible windows. *Intell. Data Anal.*, 12(3):293-304, 2008.
- [3] E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *ESA '02: Proceedings of the 10th Annual European Symposium on Algorithms*, pages 348-360, London, UK, 2002. Springer-Verlag.
- [4] P. Flajolet, D. Gardy, and L. Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Appl. Math.*, 39(3):207-229, 1992.
- [5] R. E. Giannella C., Han J. and L. Chao Mining frequent itemsets over arbitrary time intervals in data streams. In *Technical Report TR587 at Indiana University*, Bloomington, 37 pages, 2003.
- [6] L. K. Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD*, pages 290-297, New York, NY, USA, 2006. ACM.
- [7] R. Motwani and P. Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28(1):33-37, 1996.
- [8] A. N. Myers and H. S. Wilf. Some new aspects of the coupon collector’s problem. *SIAM J. Discret. Math.*, 17(1):1-17, 2004.