Improved counter based algorithms for frequent pairs mining in transactional data streams

Konstantin Kutzkov

konk@itu.dk

IT University of Copenhagen

Abstract. A straightforward approach to frequent pairs mining in transactional streams is to generate all pairs occurring in transactions and apply a frequent items mining algorithm to the resulting stream. The well-known counter based algorithms FREQUENT and SPACE-SAVING are known to achieve a very good approximation when the frequencies of the items in the stream adhere to a skewed distribution.
Motivated by observations on real datasets, we present a general technique for applying FREQUENT and SPACE-SAVING to transactional data streams for the case when the transactions considerably vary in their lengths. Despite of its simplicity, we show through extensive experiments that our approach is considerably more efficient and precise than the naïve application of FREQUENT and SPACE-SAVING.

1 Introduction

Mining heavy items from data streams is a fundamental problem in knowledge discovery. It has been widely studied from both theoretical and practical point of view, see [10] for an overview of achieved results. Motivated by applications for finding associations among purchased items in market baskets, many algorithms have been developed for mining frequent k-itemsets, for $k \geq 2$. In the case when transactions are revealed one at a time in a streaming fashion, a straightforward approach to mining the frequent k-itemsets is to generate all size-k subsets in each transaction and then apply a frequent items mining algorithm to the resulting stream. However, this approach does not make use of the additional information that for a transaction of length ℓ we implicitly know the next $\binom{\ell}{k}$ k-itemsets in the stream. In this paper we show that for many real datasets this information allows us to design significantly more efficient and robust algorithms than the naïve application of counter-based frequent items mining algorithm.

Transactional data streams. Classic algorithms like Apriori [1] and FP-Growth [14] need several passes over the input to find the frequent itemsets. This implies that we need a persistently stored database and for many applications this is not a feasible requirement. Historically, Manku and Motwani [25] first recognized the necessity for frequent itemset mining algorithms over high-speed transactional data streams. They heuristically generalized their LOSSY COUNTING algorithm and observed that it provides good estimates for the most frequent itemsets. Association rule mining from transactional data streams has been recognized a major research problem due to continuously increasing data volume, see [15] for an overview of the area.

Assuming that transactions are either generated by a random process or arrive in a random order, researchers have designed randomized algorithms for frequent itemset mining [8, 30]. The theoretical analysis of the algorithms performance thus utilizes Chernoff bounds in order to show quality estimates with high probability. However, experiments indicate [8] that such assumptions are too optimistic for real datasets and the results are not nearly as good as suggested by the theoretical analysis.

Frequent items mining. Algorithms for frequent items mining in data streams can be roughly divided in two categories: sketch-based and counter-based algorithms. Sketch-based algorithms work by hashing the items to a small sketch of the data stream processed so far and updating a corresponding counter. The frequency of individual items can be then estimated by reading a counter in the sketch. The two well-known algorithms COUNT-SKETCH [9] and COUNT-MIN [11] are based on this approach. The sketches consist of $O(1/\varepsilon)$ counters. COUNT-SKETCH provides an additive approximation of $O((\varepsilon F_2)^{\frac{1}{2}})$ and COUNT-MIN of $O(\varepsilon F_1)$ where F_p is the p-norm of the frequency vector of the stream. In order to guarantee that the approximation is correct with high probability, one runs several copies of the algorithm in parallel and returns the median of the estimates.

Counter based algorithms are deterministic. They maintain a summary of the items processed so far. The summary consists of a small subset of the items with associated counters approximating the frequency of the item in the stream. For a summary maintaining $O(1/\varepsilon)$ entries, they provide an additive approximation εF_1. It was experimentally observed that counter based algorithms provide better guarantees than sketch based algorithms but the reasons have been unclear. In a recent work Berinde et al. [4] present an analysis of so called *heavy tolerant counter based* algorithms, including FREQUENT [13, 17, 27] and SPACE-SAVING [26]. They show that both algorithms are clearly superior to sketch based algorithms, see the third paragraph of Section 2 for a detailed discussion.

Mining frequent k-itemsets. In this paper we consider frequent pairs mining in transactional data streams. Our algorithm can be extended in a straightforward way to k-itemsets mining but for the ease of presentation we concentrate on results for frequent pairs mining. Note that the generation of the frequent pairs is considered to be the most time consuming phase in Apriori [1]. As noted by Park et al. [28] *"...the initial candidate set generation, especially for the large 2-itemsets, is the key issue to improve the performance of data mining".*

Our contribution. The straightforward application of frequent items mining algorithms is the only known approach to mining frequent itemsets from transactional data streams with rigorously understood behaviour. The focus in our work is on improving the approach for counter based frequent items algorithms. Our main contributions can be summarized as follows:

– We make the observation that for many real-life datasets the length of the transactions considerably vary and most of the pairs in the transactional stream are generated by a fraction of the transactions.
– Utilizing the main idea behind the FREQUENT algorithm, we present a simple technique for adjusting FREQUENT and SPACE-SAVING to mining frequent pairs in transactional data streams.
– We prove that our modified approach yields an additive approximation error at least as good as the original algorithms. However, through extensive experiments on real datasets, we obtain considerable improvements in both running time and accuracy of the estimates.

Building upon FREQUENT, Jin and Agrawal [16] presented a method for improving the space requirements of Apriori. Thus, our algorithm automatically yields an improved version of Apriori for the considered class of datasets.

2 Preliminaries

Notation. Let $\mathcal{I}$ be a set of n items. We assume a total order on $\mathcal{I}$ and for the ease of presentation we set $\mathcal{I} = \{0, 1, 2, \ldots, n-1\}$. A *weighted stream* $\mathcal{S}$ over $\mathcal{I}$ is a sequence of m entries (i, v) for an item $i \in \mathcal{I}$ and weight $v \in \mathbb{R}^+$. When for all entries (i, v) in $\mathcal{S}$, $v = 1$ holds, we will refer to $\mathcal{S}$ as an *unweighted* stream. A *transaction* T is a subset of items of $\mathcal{I}$. The *length* of a transaction T, denoted as $|T|$, is the number of items occurring in it. A *transactional stream* $\mathcal{T}$ consists of m transactions revealed one at a time, $p = (i, j) \subset \mathcal{I}$ for $i < j$ is a *pair*.

The *frequency* or *weight* of an item i is defined as $f_i = \sum_{(i,v)\in\mathcal{S}} v$ and $\mathbf{f}_\mathcal{S} = (f_0, \ldots, f_{n-1})$ is the *frequency vector* of the stream $\mathcal{S}$. For unweighted streams the frequency is simply the number of occurrences of i in $\mathcal{S}$. For transactional streams the frequency of a pair p is the number of transactions containing p, $|\{T_j : p \subseteq T_j\}|, 1 \leq j \leq m$. The ℓ-norm of a weighted stream $\mathcal{S}$ over a set $\mathcal{I}$ is denoted as $F_\ell(\mathcal{S}) = (\sum_{i\in\mathcal{I}} f_i^\ell)^{\frac{1}{\ell}}$ for $\ell \geq 0$. For an unweighted stream $\mathcal{S}$, $F_1(\mathcal{S})$ is simply the length, i.e. the number of entries, in $\mathcal{S}$. Without loss of generality, we assume that the items $i \in \mathcal{I}$ are ordered by decreasing frequency such that $f_1 \geq f_2 \geq \ldots \geq f_n$. The kth item in this sequence is the item *of rank* k. The *k-residual ℓ-norm* of a stream $\mathcal{S}$ is then defined as $F_\ell^{res(k)}(\mathcal{S}) = (\sum_{i=k+1}^n f_i^\ell)^{\frac{1}{\ell}}$. Clearly, $F_\ell^{res(0)}(\mathcal{S}) = F_\ell(\mathcal{S})$. Items with weight above εF_1, for a user-defined $0 < \varepsilon < 1$, will be called *ε-heavy hitters* or just *heavy hitters* when ε is clear from the context.

Heavy tolerant counter based algorithms. Let us first briefly recall how the algorithms FREQUENT and SPACE-SAVING work for an unweighted stream of items. Both algorithms maintain a summary B of b entries. The summary consists of items $i \in \mathcal{I}$ with associated counter c_i serving as an estimate of the frequency of

i in the stream processed so far. We will refer to the *size of the summary* as the number of entries that can be recorded in the summary. When a new item j arrives, we check whether j is recorded in B. If so, we increment c_j by 1. If not, we check whether all b slots are occupied by an entry and if not, we insert the entry $(j, 1)$ to B. The algorithms proceed differently when j is not recorded in B. In this case FREQUENT decrements by 1 the counters of all entries in the summary and then removes entries with a counter equal to 0. SPACE-SAVING replaces an entry (k, c_k) with the smallest counter by the new entry $(j, c_k + 1)$. After processing the stream one estimates the frequency of a given item i by checking the corresponding entry in the summary. If i is not recorded in the summary, FREQUENT returns 0 as an estimate of i's frequency and SPACE-SAVING returns minimum counter in the summary. A simple analysis shows that after processing the whole stream, FREQUENT guarantees that the frequency of each item is underestimated by at most F_1/b, and SPACE-SAVING provides an upper bound of the overestimation of each item by F_1/b. This leads to the following

Definition 1 *For a stream $\mathcal{S}$, a counter based algorithm $\mathcal{A}$ with a summary of size $b = \lceil 1/\varepsilon \rceil$ provides an ε-*heavy hitter guarantee *if $\delta_i \leq \lceil \varepsilon F_1 \rceil$ for all items $i \in \mathcal{I}$ where δ_i is the absolute additive approximation error. $\mathcal{A}$ provides then an ε-*approximation *of the weight of each item and it is called an ε-*heavy tolerant *counter based algorithm.*

The above implies that after processing a stream $\mathcal{S}$, a counter based algorithm providing an ε-heavy hitter guarantee will correctly detect all ε-heavy hitters. The algorithms can be generalized in a straightforward way to handle weighted updates such that the approximation guarantee is maintained. The generalization poses a challenge only for the processing time of each incoming item, see the discussion on the running time of our algorithm in Section 3 for more on this.

Counter based vs. sketch based frequent items mining algorithms. Skewness in the frequency distribution of items in data streams is ubiquitous, see for example [10, 12]. Therefore, an algorithm providing an approximation guarantee in terms of the residual norm of a stream can be considered superior to algorithms with approximation error depending on the norm of the whole stream. The analysis of randomized sketch based algorithms naturally applies to obtaining strong k-tail approximation guarantees, i.e. depending on $F_1^{res(k)}$, for the frequency distribution of individual items. For data streams adhering to a skewed distribution such guarantees are much stronger than the heavy hitter guarantees provided by the naïve analysis of counter based analysis. Building upon work by Bose et al. [5], Berinde et al. [4] recently presented a deeper analysis of the behaviour of heavy tolerant counter-based algorithms. In particular, their results show that FREQUENT and SPACE-SAVING yield a k-tail guarantee of $(\varepsilon/k)F_1^{res(k)}$ using a summary with $O(k/\varepsilon)$ entries as opposed to sketch based algorithms requiring $O((k/\varepsilon)\log n)$ counters. This result closed the discrepancy between the better results yielded by counter based algorithms on real and synthetic data sets as compared to counter based algorithms, clearly indicating that counter based algorithms are superior.

Approximation guarantees of FREQUENT *and* SPACE-SAVING. In the following we list several results about the approximation guarantees provided by counter based algorithms. The first one is a generalization of the correctness argument of FREQUENT.

Lemma 1 *After decreasing d times the weight of at least $s > b = \frac{1}{\varepsilon}$ distinct items in a stream $\mathcal{S}$, such that no item has negative weight, all ε-heavy hitters are guaranteed to have positive weight.*

Proof. Since no item can have negative weight, $d \leq F_1(\mathcal{S})/s$ holds. This implies $f_i - d > 0$ for all $i : f_i \geq \varepsilon F_1(\mathcal{S})$.

Lemma 2 *[4, 26] After processing a weighted stream $\mathcal{S}$ by either* FREQUENT *or* SPACE-SAVING *the following hold*

- *The sum of all counters in the summary of* SPACE-SAVING *is exactly $F_1(S)$.*
- *For a summary of size $b = O(k/\varepsilon)$, the underestimation of the approximation returned by* FREQUENT *of the weight of each entry is bounded by $(\varepsilon/k)F_1^{res(k)}$.*
- *For a summary of size $b = O(k/\varepsilon)$, the overestimation of the approximation returned by* SPACE-SAVING *of the weight of each entry is bounded by $(\varepsilon/k)F_1^{res(k)}$.*

From the above we derive the following

Definition 2 *A counter based algorithm with a summary of size $O(k/\varepsilon)$ provides a (k, ε)-tail guarantee if the additive approximation of the weight of each item is bounded by $(\varepsilon/k)F_1^{res(k)}$.*

3 Our approach

Motivation. Before presenting our improved algorithm, let us give some informal motivation. Figure 1 presents the distribution of transaction lengths for the Kosarak dataset and the number of pairs generated from transactions with the given length. It turns out that more than 90% of the pairs are generated by less than 10% of the transactions. Thus, by efficiently processing the long transactions we can achieve considerable improvement in the running time of a counter based algorithm applied to a stream of pairs occurring in transactions. A naïve solution would be to simply not consider long transactions. This approach would yield incomplete results since it is often the case the long transactions contain more specific but still important information. In Section 4 we show that for datasets from various real-life domains the pairs per transaction distribution follows this pattern.

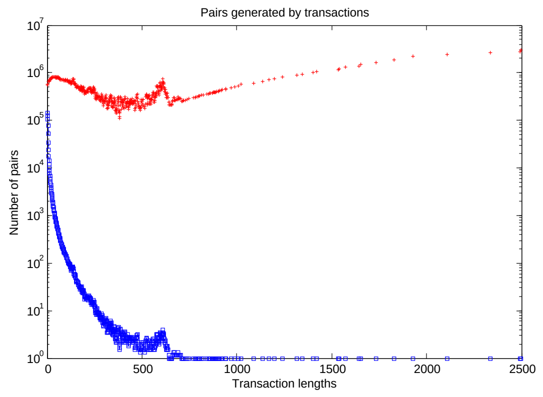

Fig. 1. The number of pairs in the Kosarak dataset generated by transactions with the given length.

The basic idea. At the heart of our improved counter based algorithms is the following simple observation. Assume we want to find an ε-approximation of the frequencies of the items in a given unweighted stream. If we repeatedly remove $s \geq 1/\varepsilon$ distinct items from the stream, and then apply a counter based frequent items mining algorithm to the updated stream, we will again achieve an ε-approximation for all items in the stream. An informal correctness argument is that each item removal is witnessed by the removal of $s-1$ other distinct items and thus the total decrease in the item frequency is bounded by $F_1/s \leq \varepsilon F_1$, thus the heavy hitter guarantee is maintained. Essentially, this is a reformulation of the main idea of the FREQUENT algorithm.

For mining frequent pairs in transactional streams the above idea may be useful because we know that a transaction of length ℓ contains $\binom{\ell}{2}$ distinct pairs. This suggests a more efficient way for processing the long transactions.

The algorithm.

A high-level pseudocode description of our algorithm FREQUENTPAIRSMINER is presented in Figure 2. The input consists of a counter based algorithm $\mathcal{A}$ providing an (ε, k)-approximation, a stream of transactions $\mathcal{T}$ and a user-defined parameter t. We will distinguish between the cases when $\mathcal{A}$ is either FREQUENT or SPACE-SAVING. $\mathcal{A}$ maintains a summary B consisting of b entries. We assume that $\mathcal{A}$ is capable of handling weighted updates. We proceed a transaction $T \in \mathcal{T}$ as follows: If $|T| \leq t$ we generate all $\binom{t}{2}$ pairs occurring in T and feed them into $\mathcal{A}$. Otherwise if $|T| > t$ we add T to a batch of long transactions $\mathcal{L}$ and check whether the number of distinct pairs in $\mathcal{L}$ is bigger than ℓ/ε for $\ell \geq 1$. If so, we find the number of occurrences of all pairs occurring more than ℓ times in $\mathcal{L}$, decrease their weight by ℓ, and feed the resulting weighted stream into $\mathcal{A}$.

```
function FrequentPairsMiner
Input: stream of transactions $\mathcal{T}$, a counter based algorithm $\mathcal{A}$, a threshold $t$
 1: for each transaction $T \in \mathcal{T}$ do
 2:     if $|T| \leq t$ then
 3:         Generate the set $\mathcal{P}_T$ of all pairs in $T$
 4:         for each pair $p \in \mathcal{P}_T$ do
 5:             call $\mathcal{A}.update(B, p, 1)$
 6:     else
 7:         add $T$ to a batch $\mathcal{L}$
 8:         if there are $\ell$ groups of at least $b$ distinct pairs in $\mathcal{L}$ then
 9:             Compute the set $P_{\mathcal{L}}$ of weighted pairs occurring in $\mathcal{L}$ more than $\ell$
                times
10:             for each $(p, w_p) \in P_{\mathcal{L}}$ do
11:                 call $\mathcal{A}.update(B, p, w_p - \ell)$
12: Report all pairs in the summary $B$ and their estimated frequency.
```

Fig. 2. A high-level pseudocode description of the algorithm.

Theorem 1. *Let $\mathcal{T}$ be a transactional stream and $\mathcal{A} = \{$*Frequent, Space-Saving*$\}$ with a summary of size $O(k/\varepsilon)$. Then for the stream of pairs in $\mathcal{T}$* FrequentPairsMiner *provides a (k, ε)-tail guarantee.*

Proof. We first show that FrequentPairsMiner provides an ε-approximation. Assume that d times we eliminate at least $1/\varepsilon$ distinct pairs in lines 8-11 and decrease the weight of a pair by at most d. Then, we feed $\mathcal{A}$ with a stream with $F_1 - d/\varepsilon$ pairs. $\mathcal{A}$ provides an ε-approximation, thus we have that the absolute additive approximation for each pair is bounded by $\varepsilon(F_1 - d/\varepsilon) + d = \varepsilon F_1$.

Next we prove that if δ is an upper bound on the absolute additive approximation, then $\frac{\delta k + F_1^{res(k)}}{b}$ is also an upper bound on the additive approximation of each pair for any $0 \leq k \leq P$ where P is the total number of distinct pairs in the stream. We consider two cases:

1. $\mathcal{A}$ =Frequent. In this case we can charge the underestimation only from pairs occurrences that have been deleted from the stream. Clearly, their number is upper-bounded by $k\delta + F_1^{res(k)}$ and since each deletion is witnessed by the deletion of at least $b+1$ distinct pairs, the bound follows.
2. $\mathcal{A}$ =Space-Saving. In the following we denote by $\mathcal{S}$ the stream of pairs generated from $\mathcal{T}$. Let $\mathcal{P}^k$ be the set of the k most frequent pairs in the transactional data stream. The frequency of each pair $p \in \mathcal{P}^k$ is approximated within absolute additive error of δ. Denote by $\mathcal{S}_{\mathcal{A}}$ the stream fed into $\mathcal{A}$ and by $\mathcal{S}_{\mathcal{L}}$ the stream of the remaining pairs in the transactional stream $\mathcal{S}$. By Lemma 2 the total sum of counters in the summary of $\mathcal{A}$ is then $F_1(\mathcal{S}_{\mathcal{A}})$. Now consider the k counters in the summary of $\mathcal{A}$ with the largest value. Each of them approximates a pair with an additive error at most δ. Also, the ith such counter, $i \leq k$, has a value of at least the weight of the ith heaviest pair in $\mathcal{S}_{\mathcal{A}}$, which in turn implies that its value is at least the weight of the

ith heaviest pair in $\mathcal{S}$ restricted to $\mathcal{S}_\mathcal{A}$. Therefore, together with Lemma 2 we obtain that the total sum of the counters in the summary, different from the largest k counters, is bounded by $F_1^{res(k)}(\mathcal{S}_\mathcal{A})$. Also, we have deleted $F_1(\mathcal{S}_\mathcal{L})$ pairs occurrences from $\mathcal{S}$. Therefore, the total approximation error for all pairs amounts to $k\delta + F_1^{res(k)}(\mathcal{S}_\mathcal{A}) + F_1^{res(k)}(\mathcal{S}_\mathcal{L})$. By Lemma 2 the overestimation of each pair in the stream $\mathcal{S}_\mathcal{A}$ is bounded by $\frac{k\delta + F_1^{res(k)}(\mathcal{S}_\mathcal{A})}{b}$. In the stream $\mathcal{S}_\mathcal{L}$ we delete d times at least b distinct pairs, thus together with Lemma 1 we obtain that we can charge to each pair an approximation error of at most $\frac{k\delta + F_1^{res(k)}(S_\mathcal{A}) + F_1^{res(k)}(S_\mathcal{L})}{b} \leq \frac{k\delta + F_1^{res(k)}}{b}$.

Once we have the recursive form for the upper bound, we can continue iterating in this way setting $\delta_i = \frac{k\delta_{i-1} + F_1^{res(k)}(\mathcal{S})}{b}$ while $\delta_i \leq \delta_{i-1}$ for the approximation error in the ith step holds. We either reach a state where no progress is made or, since the approximation error is lower bounded by $F_1^{res(k)}(\mathcal{S})$, we have $\delta_i \to \delta$ as $i \to \infty$. In either case the claim follows. Setting $b = O(k/\varepsilon)$ concludes the proof. $\square$

Running time. We present efficient algorithms for the steps in FREQUENT-PAIRSMINER. We assume that the summary can be updated in constant time. For FREQUENT a hashtable implementation guarantees a constant amortized processing time per pair. More sophisticated data structures [13, 17] improve this to constant time in the worst case. For SPACE-SAVING a linked list implementation of the summary [26] guarantees constant worst case processing time per pair. Using the modification from [7] one can achieve constant processing time for weighted pairs such that the approximation error is increased by at most a factor of 2. Using similar techniques one can obtain constant time for weighted updates for FREQUENT.

The running time crucially depends on how we process a batch of long transactions and mine the frequent pairs in the batch. Under the assumption that many items will occur only once in the batch a suitable choice would be a classic frequent itemset mining algorithm like Apriori [1] or FP-growth [14]. However, in this case we have a relatively small number of long transactions. Under the assumption that most pairs will occur at most once in the batch, we present another approach aimed at minimizing the running time:

For each combination of two transactions in the batch $\mathcal{L}$ we compute their intersection, generate the pairs occurring in the intersection and store them in a hashtable associating a set of transactions IDs with each pair. Assuming transactions are given in sorted order we can compute the intersection of two transactions in linear time. Therefore, for a batch of q transactions, such that the total number of items in the transactions in the batch is r, the running time is upper bounded by $O(qr)$: for each transactions we compute in time $O(r)$ the intersections with the other transactions. Let $\mathcal{P}^\mathcal{L}$ be the pairs occurring in $\mathcal{L}$ and $f^\mathcal{L}$ be the frequency vector of $\mathcal{P}^\mathcal{L}$. Assuming a given pair p occurs $f^\mathcal{L}(p)$ times in the batch $\mathcal{L}$, it will occur in $\binom{f^\mathcal{L}(p)}{2}$ transaction intersections. Then the number of pairs generated from all transactions is bounded by $\sum_{p \in \mathcal{P}^\mathcal{L} : f^\mathcal{L}(p) \geq 2} f^\mathcal{L}(p)^2$.

We can efficiently estimate whether there exist ℓ groups in $\mathcal{L}$, each of them containing at least b distinct pairs, by a simple modification of the approach presented in [2]. Building upon work by Bar-Yossef et al. [3], in [2] the authors present an efficient algorithm for estimating the number of distinct pairs in transactional data streams. Essentially, the algorithm hashes each pair in the stream to a random number in (0,1], keeps track of the k smallest hash values and returns as an unbiased estimate of the number of distinct pairs $\lceil k/r_k \rceil$ where r_k is the kth smallest hash value. Using a pairwise independent hash function one shows that we obtain with constant error probability an $(1 \pm \varepsilon)$-approximation of the number of distinct pairs for $k = O(1/\varepsilon^2)$. Running $O(1/\delta)$ copies of the algorithm in parallel and returning the median of the results guarantees an error probability of at most δ. A clever construction of the hash function allows each transaction to be processed in expected linear time.

The above approach admits a natural generalization for estimating the number of pairs occurring exactly i times in $\mathcal{L}$. As shown in [2] the hash function is injective with high probability and thus instead the hash value we can also store the pair with the corresponding hash value. At the end we obtain a sample of pairs with their exact frequency in $\mathcal{L}$ and using standard approaches we estimate the desired quantities.

4 Observations on datasets

Table 1 summarizes the results for several real datasets. Assuming the transactions are sorted by their length in decreasing order, each row stands for a dataset and the columns give the fraction of pairs generated by the first k percent of the transactions, $1 \leq k \leq 10$. As can be clearly seen from the values in the table, a small ratio of the transactions contributes the majority of pairs. We argue that this is a ubiquitous pattern for many real-life domains.

The first datasets are taken from the FIMI repository. Kosarak contains anonymized click-stream data of a Hungarian on-line news portal, provided by Ferenc Bodon. Webdocs [24] is built from a spidered collection of web html documents. BMS-Web-View1 [18] is built from purchase data of a legwear and legcare web retailer. MovieActors was built from the IMDB database and lists the actors acting in a given movie [7].

The next datasets were built from graphs available at the Stanford Large Network Dataset Collection (http://snap.stanford.edu/data/). For a (directed or undirected) graph given as a set of edges we created a transaction from the neighbors of each vertex. Arxiv COND-MAT [23] (Condense Matter Physics) collaboration network is from the e-print arXiv and covers scientific collaborations between authors papers submitted to Condense Matter category. If an author i co-authored a paper with author j, the graph contains a undirected edge from i to j. Therefore for each author there exists a transaction representing the set of her coauthors. The next two datasets, WikiTalk and WikiVote, are built from the online encyclopedia Wikipedia. For WikiTalk for each Wikipedia user i a transaction records the set of other users j whose talk page was edited at least once by i. WikiVote represents popularity of users. For a user i a transaction lists all users j who voted on user i [20, 21]. In Web-BerkStan for a page i from the berkely.edu and stanford.edu domains a transaction contains all pages j linked to by i [22]. Email-EU-All is built from email data from a large European research institution. For an email address i a transaction records all email addresses j if i sent at least one message to j [23]. The last dataset, soc-Epinions1, uses data from the general consumer review site Epinions.com. This is a who-trust-whom online social network and members of the site can decide whether to "trust" each other. For a member i a transaction contains all the users trusted by i [29].

Dataset	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Kosarak	0.7834	0.8634	0.9012	0.9237	0.9386	0.9493	0.9572	0.9633	0.9681	0.9719
Webdocs	0.7741	0.8189	0.8467	0.8663	0.8811	0.8930	0.9027	0.9111	0.9182	0.9244
BMS-Web-View1	0.7434	0.8009	0.8347	0.8582	0.8766	0.8920	0.9035	0.9144	0.9224	0.9296
MovieActors	0.8555	0.8991	0.9163	0.9271	0.9351	0.9414	0.9467	0.9512	0.9551	0.9586
CA-CondMat	0.3917	0.5011	0.5709	0.6225	0.6633	0.6965	0.7246	0.7487	0.7695	0.7879
WikiTalk	0.9817	0.9950	0.9979	0.9989	0.9994	0.9996	0.9997	0.9998	0.9998	0.9999
WikiVote	0.5797	0.7179	0.7933	0.8412	0.8749	0.9004	0.9203	0.9352	0.9465	0.9556
web-BerkStan	0.3734	0.5234	0.6005	0.6551	0.6886	0.7179	0.7470	0.7761	0.7997	0.8213
Email-EU-All	0.9895	0.9944	0.9962	0.9973	0.9978	0.9983	0.9986	0.9987	0.9989	0.9991
soc-Epinions1	0.7432	0.8462	0.8977	0.9289	0.9488	0.9619	0.9711	0.9777	0.9825	0.9859

Table 1. The ratio of pairs from the top-k percent longest transactions to the total number of pairs for several datasets.

5 Experiments

We choose the two datasets Kosarak and Webdocs for our experiments. Kosarak consists of 990,002 transactions over 41,270 items. The total number of pairs is more than 10^8 and the number of distinct pairs is more than 10^7. We took a prefix of Webdocs consisting of the first 100,000 transactions. There are over 5,267,656 items, almost 10^{10} pairs in total and the number of distinct pairs is slightly more than 10^9. Clearly, an exact solution using a hashtable to compute the support of all pairs is infeasible since it will require several Gigabytes of memory. As reported in [6], the distribution of pairs frequencies in both datasets adheres to a power law, therefore an approximation guarantee depending on the residual norm of the frequency vector will yield high quality estimates.

Implementation details. FrequentPairsMiner has been implemented in Java. Experiments have been run on a Mac Pro desktop equipped with Quad-Core Intel 2.8GHz and 8 GB RAM. For Space-Saving we implemented the algorithm as described in [26]. For Frequent we observed that the simpler solution guaranteeing constant amortized cost per update is more efficient than the the linked list data structure presented in [13, 17]. We worked with standard Java data structures which required the use of objects. A cache optimized implementation can achieve better space complexity by using only primitive data types. However, the goal of the present paper is to compare our approach with the naïve application of Frequent and Space-Saving to the transactional data stream. Therefore, we don't employ optimization tricks that will equally benefit both approaches.

Parameters. Of crucial importance for the achieved results, both in terms of complexity and accuracy, are the various parameters we set. In particular, how many pairs will the summary record, which transactions will be considered long and how many transactions are we going to keep in the batch. Frequent and Space-Saving need a summary of size $O(k/\varepsilon)$ to provide an (ε, k)-guarantee. However, a frequent pair is defined in terms of the number of transactions containing it. Therefore, without prior knowledge on the distribution of transaction lengths it is impossible to predict how long will be the stream of pairs and thus how many pairs we need to record in the summary in order to guarantee that frequent pairs will be in the summary after processing the transactional stream. Jin and Agrawal [16] claim that by adjusting the size of the summary of Frequent for each incoming transactions we can guarantee that ε-heavy hitters will be reported but this is erroneous. As a counter example consider a stream of m transactions such that all transactions have length 2. Assume each pair occurs exactly d times in the stream and we have a summary of size $d-1$, such that after processing of the m transactions no pair is recorded in the summary. Since transactions have the same length length we will not update the size of the summary. Then for fixed m, ε and d it is easy to construct a stream of $f(m, \varepsilon, d)$ longer transactions such that some of the pairs become ε-heavy hitters but have exactly the same frequency as pairs that have not appeared among the first m transactions. Thus, the algorithm has no way to distinguish between those pairs. Therefore in the following we assume that the summary size is a user-defined parameter which is to be chosen depending on the available memory. (Note that the parameters k and ε in the theoretical analysis of the algorithm are not user defined but simply used to obtain the best possible bounds on the approximation error.)

From the discussion about the complexity of the algorithm it is clear that a large batch of long transactions will dominate the running time since we need to compute the intersection of each two transactions. Therefore, we implemented the following heuristic: Let b be the size of the summary. In order to achieve good running time, we will keep q long transactions in the batch $\mathcal{L}$, each of them of length at least t, such that $q < t$ and $qt^2 \geq cb$ for some small constant $c > 1$. This will ensure that the running time for finding the intersections of the long transactions is of the order $q^2 \sum t_i$ opposed to $q \sum t_i^2$ for the explicit generation of all pairs in the long transactions. Assuming there are not many intersections among the long transactions, the factor c guarantees that we will (implicitly) find at least b distinct pairs in the batch. If this fails, i.e. it turns out that we have less than b distinct pairs in the batch, we explicitly generate all pairs in the transactions and update the summary with each of them.

Further we observed that the frequent pairs in the batch rarely occur more than a few times. Thus, we simply kept track of a few summary entries with the lowest estimates and when needed, replaced one of them with a heavy entry not recorded in the summary.

Overview of experiments We compared the straightforward application of Frequent and Space-Saving to the stream of pairs to FrequentPairsMining with respect to the following criteria:

1. Running time: For different summary sizes we measured the running time.
2. Precision: How many of the reported pairs in the summary are among the top-k pairs for various k.
3. Recall: The ratio of the top-k pairs reported in the summary for various k.
4. Average relative error: Following [10], for the most frequent k pairs we plot $\delta^{rel} = \frac{|f(p)-\widehat{f}(p)|}{f(p)}$, i.e. the absolute difference of the estimate $\widehat{f}(p)$ and the exact count $f(p)$ scaled by $f(p)$.

In all experiments for 2)–4) we set the size of the summary to be 50,000 for kosarak and 100,000 for webdocs. In general, we observe better running times and more precise estimates. The former is a result of the fast processing of long transaction. The latter is due to the fact that in many cases we find a number of distinct pairs which is much bigger than the size of the summary. In order to concentrate on the approximation guarantees achieved by the naïve application of Space-Saving and our optimized approach, we did not implement the book-keeping extension proposed in the original work [26] providing a lower bound on the estimates.

5.1 Experiments for $\mathcal{A}$ =Frequent

Running time. As can be seen from Figure 3 we achieve considerably better running time for summary sizes that are not too big. Note that all summary sizes guarantee reasonably good estimates. For larger summaries the advantages of our modified approach become less pronounced since less transactions are considered long.

Precision and recall. In Figures 4 and 5 we plot the precision and recall for the top-k pairs for growing k. Note that the number of pairs explicitly recorded in the summary varies, as we do not explicitly record pairs with a counter set to 0, and for the naïve implementation of Frequent it is usually smaller. Nevertheless, FrequentPairsMiner achieves better precision.

Relative approximation error. The approximation achieved by FrequentPairsMiner is also considerably better which can be explained with the smaller number of pair weights decrements.

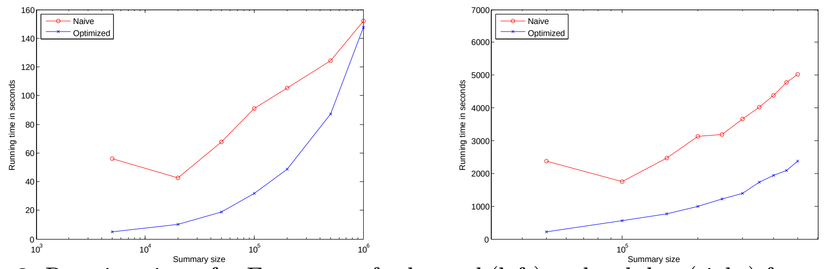

Fig. 3. Running times for Frequent for kosarak(left) and webdocs(right) for growing summary size.

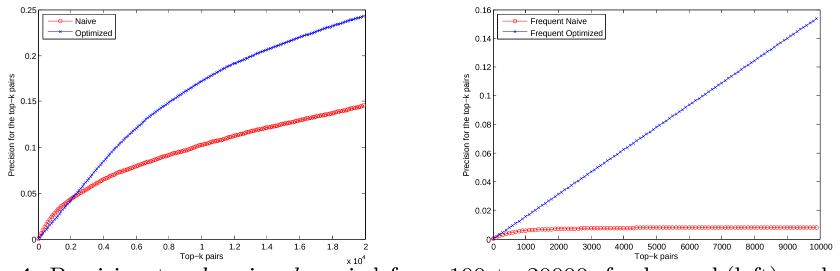

Fig. 4. Precision top-k pairs, k varied from 100 to 20000, for kosarak(left) and webdocs(right).

5.2 Experiments for $\mathcal{A}$ =Space-Saving

Running time. The running time measurements are presented in Figure 7. The larger difference compared to Frequent seems to be due to the fact that the summary size for Space-Saving remains constant.

Recall. The number of returned pairs in Space-Saving always equals the size of the summary, thus results on precision will be redundant from results on recall. Figure 8 shows the recall for the two considered datasets.

Relative approximation error. Figure 9 presents results for the relative approximation error. The straight line parallel to the x-axis is from pairs that have not been reported.

6 Further directions

We presented evidence that meanwhile classic frequent items mining streaming algorithms can be considerably improved when applied to transactional streams when the transactions lengths are skewed. A more thorough research is needed to fully employ the benefits of our approach. In particular, other approaches for counting exactly frequent pairs in a few long transaction should be possible. Note that the current approach relies on the observation on real datasets that

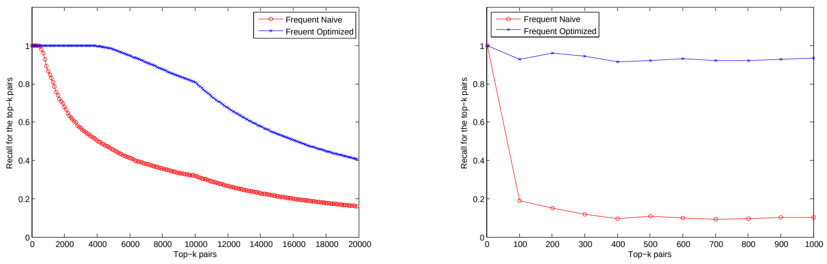

Fig. 5. Recall for Frequent for top-k pairs, k varied from 100 to 20000, for kosarak(left) and webdocs(right).

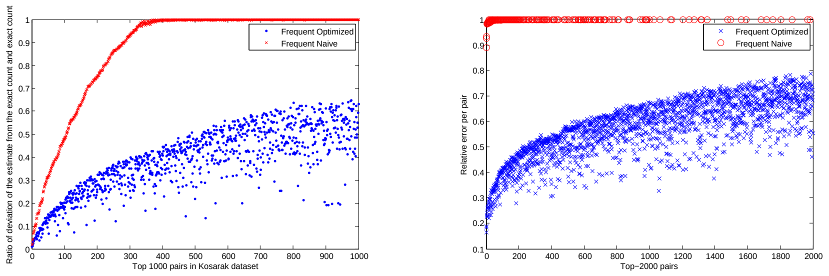

Fig. 6. Average relative error for Frequent for kosarak(left) and webdocs(right).

the overlap in the long transactions in the batch is small. However, for other datasets this assumption might not be feasible and the intersections approach will produce poor results. In such a case applying a standard approach like Apriori to the transactions in the batch might be more suitable.

Another direction is to apply our technique to mining frequent items from transactional data streams over sliding windows. In the sliding window model, we are interested in pairs occurring a certain number of times in the last t transactions for a user-defined t. A natural candidate to apply (a modification of) our approach is the algorithm by Lee and Ting [19]. This algorithm builds upon Frequent and detects frequent items among the most recently seen t items.

Acknowledgements. I would like to thank my supervisor Rasmus Pagh for helpful discussions.

References

1. R. Agrawal, R. Srikant: Fast Algorithms for Mining Association Rules in Large Databases. *VLDB* 1994: 487–499
2. R. R. Amossen, A. Campagna, R. Pagh: Better Size Estimation for Sparse Matrix Products. *APPROX-RANDOM 2010*: 406–419
3. Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, L. Trevisan. Counting Distinct Elements in a Data Stream. *RANDOM 2002*: 1–10

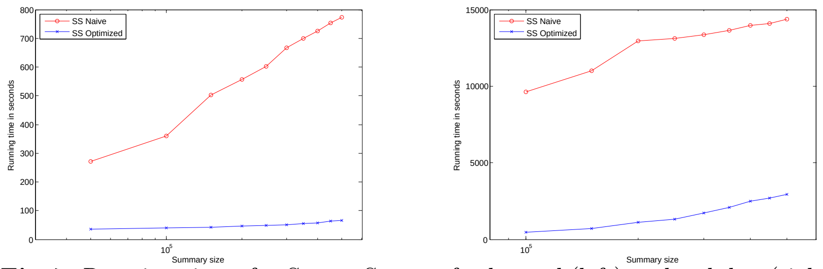

Fig. 7. Running times for Space-Saving for kosarak(left) and webdocs(right).

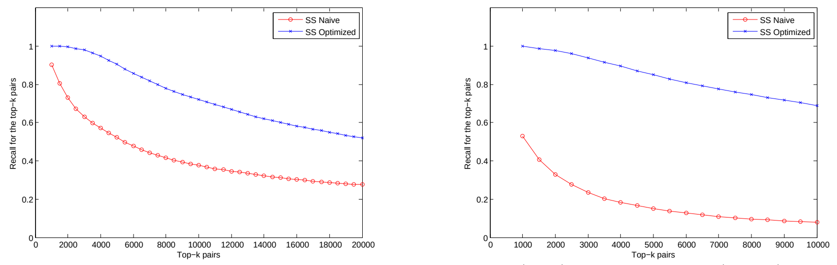

Fig. 8. Recall for Space-Saving for kosarak(left) and webdocs(right).

4. R. Berinde, P. Indyk, G. Cormode, M. J. Strauss: Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst.* 35(4): 26 (2010)
5. P. Bose, E. Kranakis, P. Morin, Y. Tang. Bounds for Frequency Estimation of Packet Streams. *SIROCCO 2003*: 33–42
6. A. Campagna, K. Kutzkov, R. Pagh. Frequent Pairs in Data Streams: Exploiting Parallelism and Skew. *ICDM Workshops 2011*: 145–150
7. A. Campagna and R. Pagh Finding Associations and Computing Similarity via Biased Pair Sampling. *ICDM 2009*: 61–70
8. A. Campagna, R. Pagh On Finding Similar Items in a Stream of Transactions. *ICDM Workshops 2010*: 121–128
9. M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci*, 312(1):3–15, 2004
10. G. Cormode, M. Hadjieleftheriou. Finding the frequent items in streams of data. *Commun. ACM 52(10)*: 97–105 (2009)
11. G. Cormode, S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms 55(1)*: 58–75 (2005)
12. G. Cormode, S. Muthukrishnan. Summarizing and Mining Skewed Data Streams. *SDM* 2005
13. E. D. Demaine, A. López-Ortiz, J. I. Munro. Frequency Estimation of Internet Packet Streams with Limited Space. *ESA 2002*: 348–360
14. J. Han, J. Pei, Y. Yin, R. Mao. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Min. Knowl. Discov.* 8(1): 53–87 (2004)
15. N. Jiang, L. Gruenwald. Research issues in data stream association rule mining. *SIGMOD Record* 35(1): 14–19 (2006)
16. R. Jin, G. Agrawal. An Algorithm for In-Core Frequent Itemset Mining on Streaming Data. *ICDM 2005*: 210–217

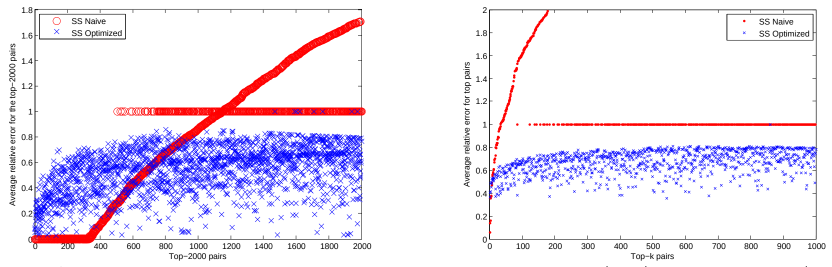

Fig. 9. Average relative error for Space-Saving for kosarak(left) and webdocs(right).

17. R. M. Karp, S. Shenker, C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst. 28*: 51–55 (2003)
18. R. Kohavi, C. E. Brodley, B. Frasca, L. Mason, Z. Zheng. KDD-Cup 2000 Organizers' Report: Peeling the Onion. *SIGKDD Explorations 2(2)*: 86–98 (2000)
19. L. K. Lee, H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. *PODS 2006*: 290–297
20. J. Leskovec, D. Huttenlocher, J. Kleinberg. Signed Networks in Social Media. *CHI* 2010.
21. J. Leskovec, D. Huttenlocher, J. Kleinberg. Predicting Positive and Negative Links in Online Social Networks. *WWW* 2010.
22. J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks. Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Mathematics* 6(1) 29–123, 2009.
23. J. Leskovec, J. Kleinberg and C. Faloutsos. Graph Evolution. Densification and Shrinking Diameters. *ACM TKDD*, 1(1), 2007.
24. C. Lucchese, S. Orlando, R. Perego, F. Silvestri. WebDocs: a real-life huge transactional dataset. *FIMI* 2004
25. G. S. Manku, R. Motwani. Approximate Frequency Counts over Data Streams. *VLDB 2002*: 346–357
26. A. Metwally, D. Agrawal, A. El Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Trans. Database Syst.* 31(3): 1095–1133 (2006)
27. J. Misra, D. Gries. Finding Repeated Elements. *Sci. Comput. Program.* 2(2): 143–152 (1982)
28. J. S. Park, M.-S. Chen, P. S. Yu. Using a Hash-Based Method with Transaction Trimming for Mining Association Rules. *IEEE TKDE* 9(5): 813–825 (1997)
29. M. Richardson and R. Agrawal and P. Domingos. Trust Management for the Semantic Web. *ISWC*, 2003.
30. J. X. Yu, Z. Chong, H. Lu, Z. Zhang, A. Zhou. A false negative approach to mining frequent itemsets from high speed transactional data streams. *Inf. Sci.* 176(14): 1986–2015 (2006)