Frequent Pairs in Data Streams: Exploiting Parallelism and Skew

Andrea Campagna*, Konstantin Kutzkov*, Rasmus Pagh*
*IT University of Copenhagen
Copenhagen, Denmark
Email: acam,konk,pagh@itu.dk

***Abstract*—We introduce the Pair Streaming Engine (PairSE) that detects frequent pairs in a data stream of transactions. Our algorithm finds the most frequent pairs with high probability, and gives tight bounds on their frequency. It is particularly space efficient for skewed distribution of pair supports, confirmed for several real-world datasets. Additionally, the algorithm parallelizes easily, which opens up for real-time processing of large transactions. Unlike previous algorithms we make no assumptions on the order of arrival of transactions and pairs.**

Our algorithm builds upon approaches for frequent items mining in data streams. We show how to efficiently scale these approaches to handle large transactions.

We report experimental results showcasing precision and recall of our method. In particular, we find that often our method achieves excellent precision, returning identical upper and lower bounds on the supports of the most frequent pairs.

***Keywords*-data stream; association rule; parallel; shared-nothing; algorithm**

I. Introduction

A fundamental tasks in knowledge discovery in databases is the mining of high quality association rules from transactional databases over a set of items. The pioneering Apriori [1] algorithm proposed about two decades ago has paved the way for many important contributions to the problem. Algorithms with much better space and time complexity have since been proposed [2]–[5] and shown to efficiently handle large amounts of data.

In this work we concentrate on discovering frequent *pairs*, or 2-itemsets, in a high speed stream of transactions. Our algorithm can be generalized in a straightforward way to k-itemsets, but the analysis becomes more complex. Also, it has been observed that already the case of 2-itemsets captures the main challenge of frequent itemset mining: *"... the initial candidate set generation, especially for the large 2-itemsets, is the key issue to improve the performance of data mining"* [4].

A. Mining data streams

Classical approaches such as Apriori [1] and FP-growth [3] require several passes over the transactional database and thus it is necessary to have access to a storage system containing the database. As observed by Manku and Motwani [6] this requirement is not practical for many real life applications where we want to mine frequent patterns in only one pass from a high speed stream of transactions. Since this seminal work, many researchers have considered the special requirements of data stream association mining. We refer the reader to the survey of Jiang and Gruenwald [7] for an introduction to this area, and references to many central works. We restrict our attention to the fundamental case of mining frequent pairs over the entire stream ("landmark model", in the classification of [8]). This problem is fundamentally different from mining over a sliding window, as the challenge is to use far less space than the size of the stream *and* keep up with the data stream in real-time.

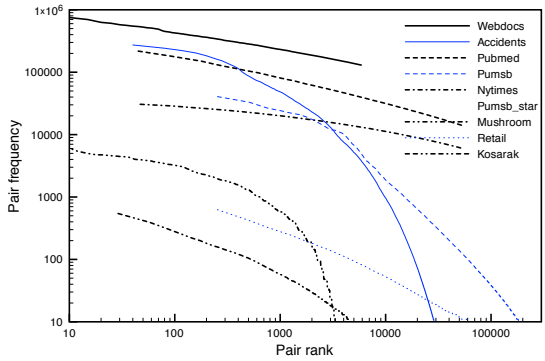

Figure 1. The frequency distribution for the most frequent pairs, on doubly-logarithmic scale. All start off with a straight line.

We present algorithms that succeed with high probability, and return upper and lower bounds on the number of occurrences rather than precise counts. For example, in the `webdocs` dataset there are around 700 million distinct pairs of items, and keeping all their counts in a hash table would require at least 8 GB of memory. In contrast, we obtain accurate results using a sketch data structure of a few megabytes that fits in L2 cache.

B. Pair similarity distribution

Skewed distributions are common for real-life datasets. We conjectured that the frequency of the pairs for many data sets will adhere to a power law, or more precisely to a Zipfian distribution. While it is well-known that this is true for single items in many data sets, it is not obvious that this assumption holds for streams of pairs generated from a stream of transactions. For this we computed the exact count

of the most significant pairs for several well-studied data sets using Borgelt's Apriori implementation [1] [9], and plotted them in decreasing order. Figure 1 shows the supports of the most frequent pairs for our datasets. In all cases we see that the curve starts of as approximately a straight line. The length of this line varies from one data set to another, from a few hundred pairs to hundreds of thousands. Observe also that in all cases where the curve deviates from a line it drops below, i.e., the distribution is dominated by a powerlaw distribution.

This motivates the design and analysis of algorithms aimed at efficient frequent pairs mining for pairs following Zipfian distribution.

C. Related work

Heuristic algorithms: Manku and Motwani [6] first recognized the necessity for efficient algorithms targeted at frequent itemsets in transaction streams. They generalized their STICKYSAMPLING algorithm to a heuristic for transaction streaming and showed empirically that it reliably estimates the frequency of the most frequent itemsets on several benchmarks. The basic idea is to process the data set in memory-sized chunks, mining each chunk for frequent itemsets to determine which itemsets should be counted in the next chunk. However, this method is vulnerable to large itemsets that are "temporarily frequent". An itemset of size k that is frequent in a chunk will have all its subsets counted in the following chunk, using space 2^k. For this reason it does not seem suitable for general use.

Reduction to the single-item case: Another approach to mining of frequent pairs (mentioned, but dismissed, in [6], [10]) is to reduce the problem to that of mining frequent items, which is well-studied in a data stream context. For a transaction $T \subseteq [n] = \{1, \ldots, n\}$, this approach generates all $\binom{|T|}{2} = \Theta(|T|^2)$ pairs occurring in T and feeds the resulting stream S, where the items of S are the pairs generated, into a frequent items algorithm. Let F_2 denote the length of the stream generated in this way. It is known that using space s one can compute the frequency of items (which are in fact pairs in our case) with an additive error of F_2/s [11]–[13].

This means that all pairs with frequency above F_2/s can be reported, with computed upper and lower bounds on the frequency that differ by at most F_2/s. While this is optimal over a worst-case data stream where all pairs occur with frequency about F_2/s, some methods, notably the SPACESAVING algorithm [14], have been observed to produce even tighter bounds on the highest frequencies in practice. However, to our best knowledge, SPACESAVING and related algorithms have never been experimentally investigated in the context of finding frequent itemsets. Frequent items algorithms aim for using small time per item, and as a matter of fact, the best methods use constant time per item; therefore the time usage for the whole stream is $O(F_2)$. The most space-efficient methods do not parallelize efficiently, as they rely on a single data structure, any part of which may be updated for a particular transaction. In contrast, we show how to parallelize efficiently without any need for shared memory.

Muthukrishan and Cormode [15] considered finding frequent items space-efficiently in a stream that is highly skewed (Zipfian distribution with parameter greater than 1). In this case they are able to improve the space needed to identify the most frequent items. However, looking at the stream of all pairs, none of the data sets that we considered exhibited large enough skew for their result to apply.

Algorithms for random streams: Yu et al. [16] presented another algorithm for transaction stream mining. The main idea in their approach is to keep a list of potentially frequent itemsets, and to update the list in a clever way when advancing the stream. They show also theoretical bounds for the quality of their estimates. In order to derive these bounds, however, they need to assume that transactions are generated *independently at random* by some process, and their analysis crucially depends on the Chernoff bounds that become applicable because of this assumption.

Campagna and Pagh [10] make the arguably weaker, but still questionable, assumption that the *order* of the transactions in the stream is random. Again, this enables them to utilize Chernoff bounds in analogy with a frequent item mining algorithm by Charikar et al. [17].

It is already clear from the experiments of [10] that such optimistic assumptions do not even approximately hold for many data sets. For both schemes [10], [16] it is easy to find an ordering of essentially any transaction stream that breaks the randomness assumption, and makes it perform much worse than the theoretical bounds.

D. Our contribution

The main contribution of the present work is a randomized algorithm that returns with high probability a correct estimate of the frequency of the most frequent pairs. We build upon well-known streaming algorithms and show how to extend them to transaction streaming.

The complexity as well as the quality of the output is determined by the Zipfian distribution parameters and the space allowed. The space usage is a user-defined parameter.

We show through extensive experiments on real and synthetic datasets that our algorithm achieves very good estimates and scales particularly well when parallelized on several cores.

II. NOTATION

The transaction stream is denoted by $S = T_1, .., T_m$ where $T_i \subseteq [n]$ A subset $p = \{i, j\} \subset [n]$ is called a pair. The set of pairs is denoted by $\mathcal{P}$, while the number of distinct pairs

[1] http://www.borgelt.net/apriori.html

occurring in the stream S is represented using $d \leq \binom{n}{2}$. Furthermore the number of frequent pairs by f, where the meaning of *frequent* will be specified in the given context.

The *support* of a pair p is the number of transactions containing p: $\sup(p) = |\{T_j : p \subseteq T_j\}|, 1 \leq j \leq m$.

A hash function $h : \mathcal{P} \to [k]$ for $k \in \mathbb{N}$ is t-wise independent if and only if $\Pr[h(p_1) = c_1 \wedge h(p_2) = c_2 \wedge \cdots \wedge h(p_t) = c_t] = k^{-t}$ for distinct pairs p_i, $1 \leq i \leq t$, and $c_i \in [k]$.

Zipfian distribution with parameters C and z is defined as $f_i = C/i^z$ for the frequency f_i of the $i-$th most frequent pair.

III. Our approach

A. Background and intuition

Before formally describing our algorithm let us give some technical background and intuition. An algorithm detecting the frequent items in an item stream can be generalized in a straightforward way in order to find frequent pairs in a stream of transactions: Simply generate all subsets of size 2 of each transaction and treat them as items. In particular, the two well known algorithms COUNT-SKETCH and SPACESAVING can be generalised as described.

In COUNT-SKETCH [17] every item i is hashed by a hash function $h : [n] \to [k]$ to a bucket B containing a counter c_B. Upon arrival of an item i the corresponding counter is updated by a uniform sign hash function $s(i)$ evaluating i to either 1 or -1. After processing the stream the frequency of a given item i can be estimated as $c_B \cdot s(i)$ where $B = h(i)$. The intuition is that the contribution from other items will cancel out. Both h and s are pairwise independent and this is sufficient to show that for an appropriate number of buckets the algorithm produces good estimates where the error is measured with respect to the $2-$norm of the vector of item frequencies. For skewed distribution of the stream frequencies this gives high quality estimates of the heaviest pairs. One can amplify the probability for correct estimates by working with more than one hash functions. Upon a query for the frequency of a given item COUNT-SKETCH returns the median of the estimates, i.e., the counters in the buckets the item hashes to.

The SPACESAVING algorithm [14] offers upper and lower frequency bounds, rather than an unbiased estimator. It keeps a list of ℓ triples $(\text{item}_j, \text{count}_j, \text{overestimation}_j)$, $1 \leq j \leq \ell$. If not all ℓ slots are already full, it inserts a new triple as $(i, 1, 0)$ for an arriving item i. The ℓ triples are sorted according to their *count* value. Once a new item arrives it checks if it is already in the list. If yes, it increases the corresponding counter by 1 and updates the order in the list. Otherwise, it replaces the last triple $(\text{item}_\ell, \text{count}_\ell, \text{overestimation}_\ell)$ with a new triple $(\text{item}_{new}, \text{count}_\ell + 1, \text{count}_\ell)$ where item_{new} is the newly arrived item. The intuition behind is that heavy items will get early on the "pull positions" and won't be evicted from the list until the end, thus for skewed data we will get an accurate estimation. However, this relies on the assumption that the most frequent pair is also frequent in an early prefix of the stream, so this will not be true in the worst case.

Our algorithm can be seen as a twofold refinement of the above direct approach:

1) In order to address the issue of having a quadratic number of pairs in each transaction, hence a quadratic number of hash values to produce, we use parallelism. In this way we are able to distribute the computation among several cores in a way such that each core efficiently computes the pairs hashing to a given subset of the hash table.
2) Assuming Zipfian distribution we want to use the fact that the most frequent pairs will not collide and thus we keep track of the most frequent pair hashing to a given bucket. We will use an important property of SPACESAVING namely, that in a stream of items, an item having relative frequency at least $1/2$ will end up in the first position of the SPACESAVING data structure.

B. Our algorithm

The skeleton of our algorithm is the following:

- Hash each pair to a bucket.
- Keep track of the most frequent pair in each bucket.
- Return an estimate of the frequency of the most frequent pair for each bucket.

In the parallel version, each processor keeps track of an interval of the hash table, and the total space remains fixed. Thus, we are in a *shared nothing* model with no need for a shared memory – the only requirement is that each processor sees the input stream. It is well-known that this kind of parallel algorithms scales extremely well compared to algorithms that rely on interprocess communication or shared data structures. Even for the largest data sets that we looked at, it is feasible to keep the entire hash table in L2 cache of the involved processors on a large workstation, resulting in extremely fast processing.

A crucial property is that most frequent pairs do not collide, and thus we obtain high quality estimates on their frequency. We combine two different ways for estimating the frequency of the heaviest pairs based on the COUNT-SKETCH and SPACESAVING algorithms. In particular, we use a distribution hash function $h : [n] \times [n] \to \{1, \ldots, k-1\}$ to split the set of pairs into k parts, and use a SPACESAVING sketch on each part. The size k of the hash table and the size of the SPACESAVING sketch determines the accuracy of the sketch.

Parallelizing processing of pairs: Naïvely we could just iterate through all pairs of each transaction T_t, but we would like an algorithm that runs in linear time when the number of pairs hashing to $[i, j)$ is small. This will allow us to split the task of computing the sketch among several cores, all the

way to the point where each core processes a transaction in linear time. In other words, given sufficient parallelism we can handle a given data rate even if the transactions are huge.

Lemma 1: Let $h : \mathcal{P} \to \{0, \ldots, k-1\}$ be a pairwise independent hash function. Given a transaction T of size t and a subset $\mathcal{L} \subseteq \{0, \ldots, k-1\}$ we can construct $\mathcal{P}^T_{\mathcal{L}}$, the set of pairs occurring in T hashing to a value in $\mathcal{L}$, in time $O(|\mathcal{P}^T_{\mathcal{L}}| + t)$.

In order to improve the algorithm's accuracy we may run several copies of the algorithm in parallel and report the median.

At the end a pair is reported frequent if it has "won" in at least $t/2$ of its corresponding SPACESAVING data structures. Our experimental results will be for a single run, so the reported accuracy can be improved at the cost of time and space.

The second estimate of the algorithm is based on the COUNT-SKETCH algorithm by Charikar, Chen and Farach-Colton [17]. Here, we have a counter serving as an unbiased estimator for the frequency of the heaviest pair, where *unbiased* means that the estimate does not depend on the order of arrival of pairs.

As in the original COUNT-SKETCH algorithm we will work with an additional pairwise independent hash functions: the sign function $s : \mathcal{P} \to \{-1, 1\}$. With each bucket B we associate a counter c_B. The counter serves the same purpose as in the original algorithm [17].

Upon arriving of a new pair p we update the corresponding bucket, we abuse notation and denote it as $h(p)$, as follows: $c_{h(p)} = c_{h(p)} + s(p)$. The intuition is that the heaviest pair will contribute with the same sign, and contributions from other pairs will cancel out. At the end the algorithm returns $s(p) \cdot c_{h(p)}$ as estimated frequency for the pair where p is the first pair in the SPACESAVING data structure. As we show in the next section, if $\sup(p) > m/2$, then a high-quality estimation of p's frequency is returned.

In order to reduce the error we can work again with several independent hash functions and report the median of the results.

IV. EXPERIMENTS

Table I summarizes the data sets that we use for experiments. In all cases, we use the order in which the transactions are given as the stream order. We worked with two implementations, a simple Python implementation, and a cache-optimized Java implementation that was 10–20 times faster. In both cases, we used the built-in random number generator of the language to store hash values in a table.

A. Accuracy of results

Our first set of experiments shows results on the precision of the counts obtained by PairSE using a SPACESAVING data structure of size 2. The accuracy is of course influenced by the amount of space used, as well as the number of pairs you are interested in reporting. We made one experiment fixing the space usage, and looking at results for pairs of decreasing rank (computed exactly), and one that varies the space usage and considers the top-100 pairs.

Dataset	# of pairs (F_2)	# of distinct pairs
Mushroom	$22.4 \cdot 10^5$	$3.65 \cdot 10^3$
Pumsb	$1360 \cdot 10^5$	$536 \cdot 10^3$
Pumsb_star	$638 \cdot 10^5$	$485 \cdot 10^3$
Kosarak	$3130 \cdot 10^5$	$33100 \cdot 10^3$
Retail	$80.7 \cdot 10^5$	$3600 \cdot 10^3$
Accidents	$187 \cdot 10^5$	$47.3 \cdot 10^3$
Webdocs	$2.0 \cdot 10^{11}$	$> 7 \cdot 10^{10}$
Nytimes	$1.0 \cdot 10^{10}$	$> 5 \cdot 10^8$
Pubmed	$1.6 \cdot 10^{10}$	$> 6 \cdot 10^8$
Wikipedia	$5.17 \cdot 10^{11}$	$> 5.8 \cdot 10^9$

Table I
INFORMATION ON DATA SETS FOR OUR EXPERIMENTS. NYTIMES AND PUBMED ARE TAKEN FROM THE UCI MACHINE LEARNING REPOSITORY (BAG OF WORDS DATA SET). THE WIKIPEDIA DATASET HAS BEEN CRAFTED ACCORDING TO WHAT IS DESCRIBED IN [18, PAGE 14]. FOR THE LAST THREE DATA SETS THE NUMBER OF DISTINCT PAIRS WAS ESTIMATED USING A HASHING TECHNIQUE FROM [19]. THE DATASETS PUMSB, AND PUMSB STAR WERE PREPARED BY ROBERTO BAYARDO FROM THE UCI DATASETS AND PUMBS. KOSARAK CONTAINS (ANONYMIZED) CLICK-STREAM DATA OF A HUNGARIAN ON-LINE NEWS PORTAL, PROVIDED BY FERENC BODON. RETAIL CONTAINS THE (ANONYMIZED) RETAIL MARKET BASKET DATA FROM A BELGIAN RETAIL STORE (BRIJS ET AL., 1999). ACCIDENTS CONTAINS (ANONYMIZED) TRAFFIC ACCIDENT DATA (GEURTS ET AL., 2003)

Fixed space usage: In practice, it may be hard to foresee how much space will be needed for a particular stream, so probably one will tend to use as much space as feasible with respect to running time (ensure in-cache hash table), or what amount of memory can be made available on the system. A consequence of this will be even more precise results. The results of our experiments on the Nytimes data set can be seen in Figures 2 and 3. The former zooms in on the zone where the lower and upper bounds computed by SPACESAVING are very accurate.

Varying space usage: We now investigate what happens to the quality of results when the space usage of PairSE is pushed to, and beyond, its limits. For this, we chose to work with 3 representative data sets, namely Mushroom, Retail and Accidents, for decreasing space usage, plotting the ratio between the upper and lower bounds for the top-100 pairs returned by our algorithm. This is shown in Figure 4 and we can see how the transition between very good and very poor quality is fairly fast.

B. Count-Sketch Estimates

The result of the unbiased COUNT-SKETCH estimator for the Kosarak dataset with 50000 buckets and 2 pairs per bucket is presented in Figure 5. We ran the algorithm 11 times and for each pair reported at least 6 times we return the median of its estimates. The plot shows the ratio of our

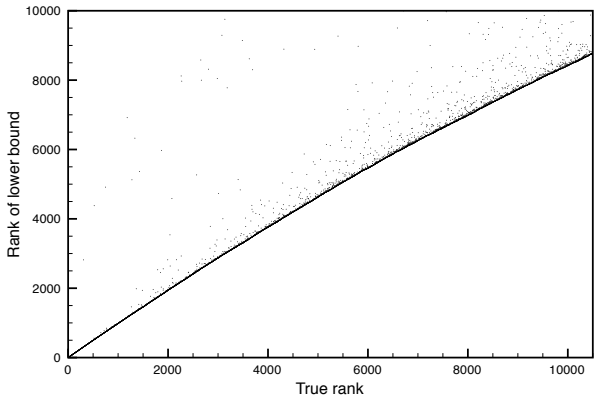

Figure 2. Top frequent pairs for Nytimes, and their rank according to the frequency lower bound computed by PairSE using 10^6 buckets. As can be seen, recall is initially high, but decreases with the support.

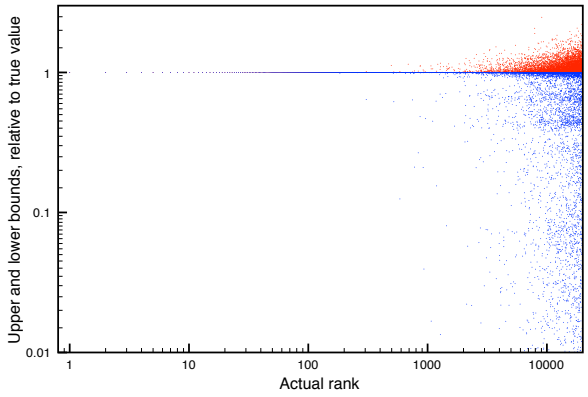

Figure 3. Upper and lower bounds for Nytimes computed by PairSE using 10^6 buckets. Values are normalized by dividing by true support. Upper bounds shadow lower bounds, exact bounds are visible only as a red dot with no blue dot below. As can be seen, upper bounds are generally tighter than lower bounds.

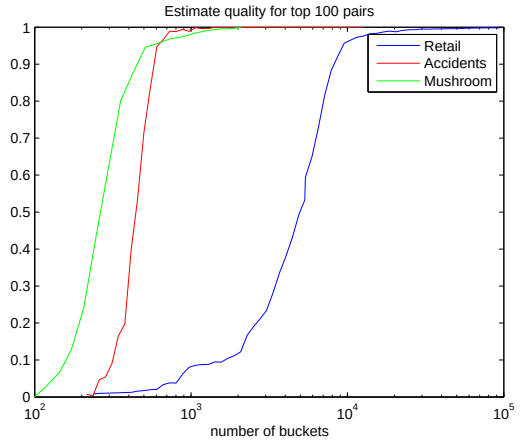

Figure 4. Average ratio of lower and upper bound for top-100 pairs, for three representative data sets, as function of number of buckets. As can be seen, there is a quick transition from poor to excellent precision.

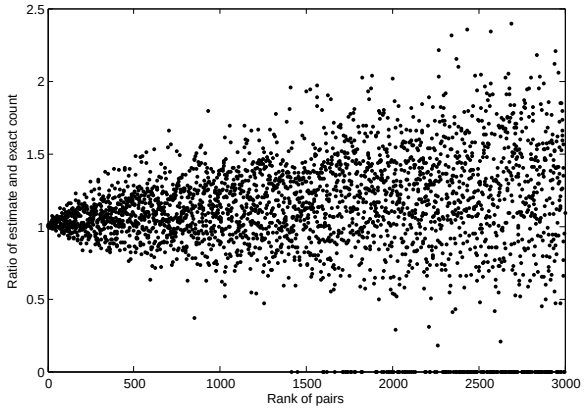

Figure 5. Ratio of estimates and true count for the top 3000 pairs of Kosarak. All top 1400 pairs are reported by our algorithm and for most of the pairs the estimates are within of factor 2.

estimates and the exact count of the 3000 pairs with highest support in the dataset. Not reported pairs have ratio 0.

C. Performance and scalability

Experiments have been carried out in order to verify how the algorithm scales, in terms of time, when parallel computations are used. We ran the algorithm on various datasets using several different number of cores. In this way it has been possible to highlight the parallel nature of the algorithm, hence, its capability of being very time efficient when many cores are at hand. Table II reports some of the results we obtained. The machine we used is described in the caption of the table. For large datasets such as nytimes, with a high number of pairs, our simple Java implementation processed almost 100 million pairs per second on an 8-core Mac Pro. The throughput of a competing hash table solution can be upper bounded by the number of updates of random memory locations possible (disregarding time for hash function computation, and other overheads). On the Mac Pro the number of such updates per second was estimated to around 50 millon per second, when updating a 1 GB table using 8 cores. This means that we are at least a factor of 2 faster than any implementation based on a large, shared data structure.

Acknowledgement

We wish to thank Blue Martini Software for contributing the KDD Cup 2000 data.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB*, 1994, pp. 487–499.

[2] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," in *SIGMOD Conference*. ACM Press, 1997, pp. 255–264.

Dataset	# of cores	*ms* on # cores	*ms* 1 core
Kosarak	8	1551	2881
	4	1586	
	2	1997	
Webdocs	8	299153	891565
	4	357679	
	2	482111	
Nytimes	8	443119	1313698
	4	524553	
	2	689058	
Wikipedia	8	27526403	93477243
	4	35397110	
	2	53795313	

Table II
EXPERIMENTS RAN ON AN INTEL XEON E5570 2.93 GHZ EQUIPPED WITH 23 GB OF RAM; THE OS IS GNU/LINUX, KERNEL VERSION 2.6.18. TIMES ARE GIVEN IN MILLISECONDS (*ms*). THE NUMBER OF BUCKETS IS $\approx 2^{20}$.

[3] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Min. Knowl. Discov*, vol. 8, no. 1, pp. 53–87, 2004.

[4] J. S. Park, M.-S. Chen, and P. S. Yu, "An effective hash-based algorithm for mining association rules," *SIGMOD Record*, vol. 24, no. 2, pp. 175–186, Jun. 1995.

[5] A. Savasere, E. Omiecinski, and S. B. Navathe, "An efficient algorithm for mining association rules in large databases," in *VLDB*, 1995, pp. 432–444.

[6] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *VLDB '02*. Morgan Kaufmann Publishers, 2002, pp. 346–357.

[7] N. Jiang and L. Gruenwald, "Research issues in data stream association rule mining," *SIGMOD Record*, vol. 35, no. 1, pp. 14–19, 2006.

[8] Y. Zhu and D. Shasha, "Statstream: Statistical monitoring of thousands of data streams in real time," in *(VLDB)*, 2002, pp. 358–369.

[9] C. Borgelt, "Recursion pruning for the apriori algorithm," in *IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2004.

[10] A. Campagna and R. Pagh, "On finding similar items in a stream of transactions," in *ICDM Workshops*, 2010, pp. 121–128.

[11] J. Misra and D. Gries, "Finding repeated elements," *Sci. Comput. Program.*, vol. 2, no. 2, pp. 143–152, 1982.

[12] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency estimation of internet packet streams with limited space," in *ESA 2002*, 2002, pp. 348–360.

[13] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Trans. Database Syst.*, vol. 28, pp. 51–55, 2003.

[14] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *ICDT*, pp. 398–412.

[15] G. Cormode and S. Muthukrishnan, "Summarizing and mining skewed data streams," in *SIAM International Conference on Data Mining*, 2005.

[16] J. X. Yu, Z. Chong, H. Lu, Z. Zhang, and A. Zhou, "A false negative approach to mining frequent itemsets from high speed transactional data streams," *Inf. Sci.*, vol. 176, no. 14, pp. 1986–2015, 2006.

[17] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theor. Comput. Sci.*, vol. 312, no. 1, pp. 3–15, 2004.

[18] E. Abusland and M. Markovics, "Implementing and evaluating a sampling-based approach to association mining on mapreduce," Master's thesis, IT University of Copenhagen, 2011.

[19] R. R. Amossen, A. Campagna, and R. Pagh, "Better size estimation for sparse matrix products," in *APPROX–RANDOM 2010*, ser. Lecture Notes in Computer Science, vol. 6302. Springer, 2010, pp. 406–419.