On Finding Similar Items in a Stream of Transactions

Andrea Campagna and Rasmus Pagh
IT University of Copenhagen, Denmark
Email: {acam,pagh}@itu.dk

***Abstract*—While there has been a lot of work on finding frequent itemsets in transaction data streams, none of these solve the problem of finding similar pairs according to standard similarity measures. This paper is a first attempt at dealing with this, arguably more important, problem.**

We start out with a negative result that also explains the lack of theoretical upper bounds on the space usage of data mining algorithms for finding frequent itemsets: Any algorithm that (even only approximately and with a chance of error) finds the most frequent k-itemset must use space $\Omega(\min\{mb, n^k, (mb/\varphi)^k\})$ bits, where mb is the number of items in the stream so far, n is the number of distinct items and φ is a support threshold.

To achieve any non-trivial space upper bound we must thus abandon a worst-case assumption on the data stream. We work under the model that the transactions come in random order, and show that surprisingly, not only is small-space similarity mining possible for the most common similarity measures, but the mining accuracy *improves* with the length of the stream for any fixed support threshold.

***Keywords*-algorithms; streaming; sampling; data mining; association rules.**

I. Introduction

Imagine that we have a set of m sets ("transactions"), each a subset of $\{1, \ldots, n\}$, and that we want to find interesting associations among items in these transactions. This problem is often framed in a "market basket" model where we are interested in finding those pairs of items that are frequently bought together. Whether a pattern is really interesting or not is a problem dependent question, and for this reason various similarity measures other than number of co-occurrences have been introduced. Some of the most common measures are *Jaccard* [1], *cosine*, and *all_confidence* [2], [3]. Besides these measures we are also interested in association rules, which are intimately related to the *overlap coefficient* similarity measure. See [4, Chapter 5] for background and discussion of similarity measures.

We initiate the study of this problem in the *streaming model* where transactions arrive one by one, and we are allowed limited time per transaction and very small space. The latter constraint implies we cannot hope to store much information regarding pairs that are not similar and, moreover, we cannot store the input. In particular, classical frequent item set algorithms such as Apriori [5] and FP-growth [6] that work in several passes over the data cannot be used. The survey of Jiang and Gruenwald [7] gives a good overview of the challenges in data stream association mining.

Previous works on transaction data streams have focused on finding frequent itemsets, and can be classified in the following way [8]:

Landmark model: The frequent itemsets are searched for in the whole stream, so that itemsets that appeared in the far past have the same importance as recent ones;

Damped model: This model is also called *Time-Fading*. Recent transactions have a higher weight than the older ones, so nearer itemsets are considered more interesting than the further;

Sliding window: Only a part of the stream is considered at a given time in this model, the one falling in the sliding window. This implies storing information concerning the transactions falling within the window, since whenever a transaction gets out of the window span, it has to be removed from the counts of the itemsets.

The last two models make the problem of achieving low space usage simpler, since most of the information in the stream has little or no effect on the mining result. The challenge is instead to handle the real-time requirements of data stream settings.

All these approaches look for frequent items and do not try to compute any similarity, relying on the tacit assumption that whatever is frequent is automatically interesting. This assumption is not always true:

Example: Suppose we have item 1 appearing in 20% of transactions, item 2 appearing in 20% of transactions, and the pair $\{1, 2\}$ appears in 10% of transactions. Suppose moreover that the pair $\{3, 4\}$ appears in only 5% of transactions and that these transactions are the only ones in which 3 and 4 appear. The set $\{1, 2\}$ has a frequency that is two times the one of $\{3, 4\}$. But looking at the similarity function *cosine*, we can easily realize that $s(1, 2) = 10/20 = 0.5$ while $s(3, 4) = 5/5 = 1$. If we base the idea of similarity only on frequencies, we are likely to miss the pair $\{3, 4\}$ which holds a much higher similarity than the more frequent pair $\{1, 2\}$.

Notice also that $\{3, 4\}$ holds a higher similarity for *all* the measures we are addressing, so the example shows how frequencies alone do not suffice to infer similarity properties of pairs. $\circ$

Our contributions: In this paper we address the problem of finding similar pairs in a stream of transactions. We first show a negative result, which is that a worst-case stream does not allow solutions with non-trivial space usage: To approximate even the simplest similarity measure one essentially needs space that would be sufficient to store either the number of occurrences of all pairs or the contents of the stream itself. Imposing a minimum support φ for the items we are interested in alleviates the problem only when φ is close to the number of transactions.

Theorem 1: Given a constant $k > 0$, and integers m, n, φ, consider inputs of m transactions of total size mk with n distinct items. Let $s_{\max}$ denote the highest support among k-itemsets where each item has support φ or more. Any algorithm that makes a single pass over the transactions and estimates $s_{\max}$ within a factor $\alpha < 2$ with error probability $\delta < 1/2$ must use space $\Omega(\min(m, n^k, (m/\varphi)^k))$ bits in expectation on a worst-case input distribution. ○

This lower bound extends and strengthens a lower bound for single-item streams presented in [9].

Of course, many data streams may not exhibit worst-case behavior. Several papers have considered models of data streams where the items are supposed to be independently chosen from some distribution, or presented in random order [10]–[13]. We present an upper bound that works for a worst-case set of transactions under the condition that it is presented in random order, which is sufficient to bypass the lower bound. Our method is general in the sense that it can evaluate the similarity of pairs according to several well-established measure functions.

We note that outside the streaming domain, distributed sorting algorithms, such as the one built into MapReduce, can be used to permute transactions in random order (by using random values as keys). It seems likely that our approach can also be used in a 1-pass MapReduce implementation.

Theorem 2: Let $\delta > 0$ be constant, and s, $M > 1$ be integers. We consider a data stream of transactions (subsets of $\{1, \ldots n\}$) of maximum size M, where in each prefix the set of transactions appears in random order. For all the similarity measures in figure 1 there is a streaming algorithm (depending on s and M) that maintains a "$1 \pm \delta$ approximation" of the s most similar high-support pairs in the stream, as follows: Within the m transactions seen so far, let Δ be the sth highest similarity among pairs $\{i, j\}$ where both i and j appear at least φ times, where φ can be any function of m. There exists $L = O(\log(mn))$ such that if $\Delta > \frac{L}{\varphi} \max\left\{\sqrt{\frac{mbM}{s}}, M\right\}$, then the pairs maintained all have similarity at least $(1-\delta)\Delta$ with high probability, and all such pairs with similarity $(1+\delta)\Delta$ or more are reported. To process a prefix of mb items, the algorithm uses time $O(mb \log(nm))$, with high probability, and space $O(n+s)$. ○

It is worth noticing that s can be chosen as $O(n)$, which yields a space usage linear in the number of distinct items. Conversely, choosing s smaller does not improve the space usage, so we may assume $s \geq n$. In absence of a known bound on the maximum transaction size, one can use $M = n$. Then the algorithm guarantees to detect pairs with similarity at least $\frac{L}{\varphi} \max\left\{\sqrt{mb}, n\right\}$. Using $s \geq n$ and ignoring the logarithmic factor L this means that up to input size $mb = n^2$ we can detect similarity n/φ, and after this point we can detect similarity $\sqrt{mb}/\varphi$. Assuming that φ is chosen as a linear function of m (relative support threshold), we see that the accuracy improves with the length of the stream.

A. Previous work

Denote by m the number of transactions seen up to the moment in which we want to report the similar pairs. Let n indicate the number of distinct items that can appear in transactions. Without loss of generality we can assume these items are in the set $\{1, \ldots, n\}$. Parameter b is the average length of transactions (such that mb is the size of the data set seen so far).

Most of the algorithms we describe actually consider the problem of finding frequent objects in a stream of items, so they do not focus on itemsets, like we do. But given a stream of transactions we can of course generate the stream of all pairs occurring in these transactions, and feed them to a frequent item algorithm. (We do not consider here that this might not be possible for large transactions in settings where real-time constraints are important.) In the following we let M_2 denote the length of the derived stream of pairs.

Landmark model: Many papers have addressed the problem of frequent items in a stream. Starting from the seminal paper [14] streaming algorithms have started to flow in recent years. Many important contributions to the problem of frequent items (and indirectly frequent itemsets) have thus been presented.

In several independent papers [10], [15], [16] algorithms have been presented that can find all pairs with support at least k using space $M_2/(k-1)$ and constant time per pair in the stream. These algorithms may generate false positives, i.e., it is only known that the output will contain the frequent pairs.

Cormode and Muthukrishnan [9] consider the problem of reporting *hot items* in a fully dynamic database scenario. The space usage is similar to the schemes above, but the error probability can be reduced arbitrarily (at the cost of space).

Also in [9] is a lower bound on the number of bits of memory necessary in order to answer queries that concern reporting the items with frequencies over a certain threshold. This lower bound is extended and generalized by our lower bound in theorem 1.

In [17] the COUNT SKETCH algorithm tackles the problem of reporting the k most frequent itemsets. For worst-case distributions their algorithm has similar performance to those mentioned above, but for skewed distributions they are able to detect itemsets with smaller frequencies in the same amount of space.

A false negative approach: Yu et al. [11] present algorithms directly addressing the problem of finding frequent itemsets in a transaction stream. The algorithm does not find itemsets that are similar by means of measure functions other than support. Under the assumption that items occur independently (which is arguably quite strong, since we are assuming that there may be dependencies resulting in frequent sets) the authors show upper bounds on space usage similar to those of [9]. The performance is tested on artificial data sets where the independence assumption holds. For itemsets of size two (or more) the paper lacks a theoretical analysis of the proposed algorithm, but claims an empirical space usage bounded by m^3/k^3.

Sampling according to the similarity: Our algorithms builds on top of an idea presented in [18], [19]. The sampling technique used in that algorithm is such that pairs are sampled a number of times that is proportional to their similarity. (A more technical explanation can be found in section III-A where we improve the sampling procedure to make it suitable for a streaming environment.) The algorithms presented in [18], [19] have near-optimal running time, when no information on the distribution of similarities are given. As a matter of fact, the running time is linear in the size of the input and output (when there are many pairs of roughly the same similarity). The methods presented are highly general and apply to many measure functions that are linear in the number of occurrences of a pair. However, the method does not directly apply to a streaming setting since it needs two passes over the data.

II. LOWER BOUND

There are two naïve approaches to handling k-itemset support counting in a data stream setting: One consists in storing all the transactions seen (possibly trying to compress the representation), and the other one maintains support counts for all k-itemsets seen so far.

Theorem 1 says that it is not possible to beat the best of these approaches in the worst case (with support threshold $\varphi = 1$). The proof is a reduction from communication complexity:

Proof: The inputs considered for the lower bound have m transactions of size k. Let $n' = \min(n, \lfloor mk/(2\varphi) \rfloor) - 1$ be the largest possible number of items that can appear φ times in $m/2$ transactions, minus 1. We pick an arbitrary set F of n' items, and will form an input stream that consists of two parts:

- In the first $m/2$ transactions we ensure that each item in F appears φ times or more, while no k-subset of F appears. This can be done by putting one item not in F in each transaction.
- In the last $m/2$ transactions we encode information that will require many bits to store, as detailed below.

Consider the first $s = \min(m/2, \binom{n'}{k})$ transactions in the second part. Since $s \leq \binom{n'}{k}$ we can map the numbers $\{1, \ldots, s\}$ to unique k-itemsets in F. In particular, any bit string $x \in \{0,1\}^s$ can be mapped to the unique set of transactions corresponding to the positions of 1s in x. In this data set, each k-itemset from F appears at most once.

Suppose we have an algorithm that can determine the support of the most frequent itemset within a factor $\alpha < 2$ with probability $1 - \delta$. This implies that, on inputs where no itemset appears more than twice, the algorithm can distinguish (with probability $1-\delta$) the cases where the most frequent itemset appears once and twice. Given $x \in \{0,1\}^s$ we consider the memory configuration after the algorithm has seen the set of transactions that correspond to x. This can be seen as a "message" that encodes sufficient information on x that allows us to determine if one of the itemsets we have seen appears later in the stream. Lower bounds from communication complexity (see [20, Example 3.22]) tell us that even when we allow error probability $\delta < 1/2$ the amount of communication to determine whether $x, y \in \{0,1\}^s$ have a 1 in the same position (corresponding to the same k-itemset appearing twice) is $\Omega(s)$ bits in expectation. This means that the memory representation (even if it is compressed) must use $\Omega(s)$ bits. Using the estimate $\binom{n'}{k} \geq \min(\binom{n}{k}, \binom{mk/(3\varphi)}{k}) = \Omega(\min(n^k, (m/\varphi)^k))$ we get the lower bound stated in the theorem. ■

Corollary 3: Any deterministic algorithm that determines the highest support in a transaction data stream must, after having processed transactions of total size mb, use space $\Omega(\min(mb, n^k))$ bits on a worst-case input. ○

III. OUR ALGORITHM

We present a new algorithm for extracting similar pairs from a set of transactions using only one pass over the data. The algorithm is approximate, so false negatives and false positives occur. Most of our discussion will concern space usage, but we are also aiming for very low per-item time complexity of the algorithm. In particular, we will not allow anything like iterating through all pairs in a transaction.

The measures we will address are reported in Figure 1, and are all symmetric. This means that we are interested only in looking at pairs (i, j) where $i < j$. For this reason we will use set notation for the pairs, so instead of (i, j) we will write $\{i, j\}$.

Parameters of the algorithm: We recall that φ is the item support threshold, and M is the maximal transaction size. Increasing φ will decrease the minimum similarity the algorithm will be able to spot. M is a characteristic of the transactions, supplied as a parameter to the algorithm. In absence of a known bound on M, one can set $M = n$. The

Measure	$s(i,j)$	$f(\lvert S_i\rvert, \lvert S_j\rvert)$
Cosine	$\frac{\lvert S_i \cap S_j\rvert}{\sqrt{\lvert S_i\rvert \lvert S_j\rvert}}$	$1/\sqrt{\lvert S_i\rvert \cdot \lvert S_j\rvert}$
Dice	$\frac{\lvert S_i \cap S_j\rvert}{\lvert S_i\rvert + \lvert S_j\rvert}$	$1/(\lvert S_i\rvert + \lvert S_j\rvert)$
All_confidence	$\frac{\lvert S_i \cap S_j\rvert}{\max(\lvert S_i\rvert, \lvert S_j\rvert)}$	$1/\max(\lvert S_i\rvert, \lvert S_j\rvert)$
Overlap_coef	$\frac{\lvert S_i \cap S_j\rvert}{\min(\lvert S_i\rvert, \lvert S_j\rvert)}$	$1/\min(\lvert S_i\rvert, \lvert S_j\rvert)$

Figure 1. *Measures that we cover with our algorithm and the corresponding functions. The overlap coefficient measure has the property that finding pairs having similarity over a certain threshold implies finding all association rules with confidence over that the same threshold. As argued in [18], [19],* Jaccard *similarity can be handled via dice similarity.*

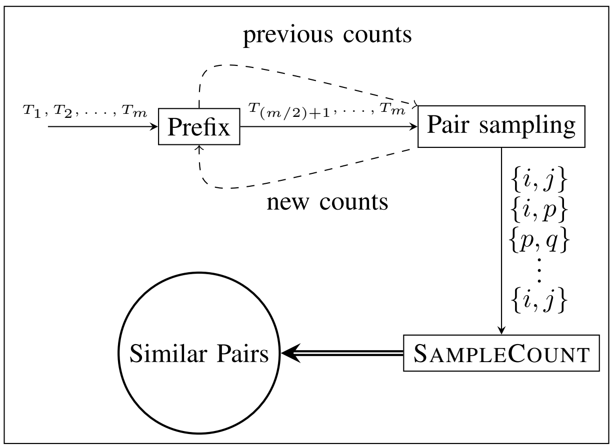

Figure 2. Overview of the algorithm with all its components.

parameter s determines the space usage of the algorithm, which is $O(n+s)$ words.

Notation: In the streaming framework, the total number of transactions is not known. In order to address this issue, we consider sets of transactions, *prefixes*, of the stream of increasing size. Suppose that so far we have seen m transactions $T_1, \ldots, T_m \subseteq \{1, \ldots, n\}$.

The *current* prefix has length 2^t, $t \in \mathbf{N} \cup \{0\}$ when m falls in the interval $[2^t, 2^{t+1})$. Our algorithm maintains counts of all items and store copies of the counts every time the current prefix changes (that is: every time the number of transactions seen is two times the length of the current prefix). Each time the current prefix changes, we update our estimate of the most similar pairs, and use this estimate until the next change of current prefix.

The algorithm is based on two pipelined stages: a stream of pairs generation phase and a store and count phase. We will describe the two phases separately, since the output of the former phase will constitute the input of the latter. Figure 2 gives an overview of the algorithm.

The prefixes of the stream are fed to a *pair sampling* stage that uses the stored counts from the previous prefix to compute sampling probabilities. Given the current prefix, the counts relative to that prefix will be used in order to sample pairs in the stream, until a new set of counts is stored for the prefix of length 2^{t+1} The idea is that, since transactions come in random order, the sampling probabilities should be approximately the same as for the BiSam sampling procedure (which bases the sampling probabilities on exact item frequencies).

In section IV we show how this technique samples, with high probability, the pairs having a high enough similarity. In fact, we show that a stronger property holds with high probability: Even when we split the stream into κ chunks, each with the same number of transactions, we will sample these pairs sufficiently often in each chunk to reliably estimate their similarity.

A. Pair sampling

We base our technique on the sampling method of the BiSam algorithm [18], [19]. For each transaction the pairs are sampled according to their support, such that the pair $\{i,j\}$ is sampled with probability $\tau f(|S_i|, |S_j|)$, where f is a function that depends on the similarity measure considered, and τ is a parameter that is used to control the sampling rate. We fix $\tau = \frac{4\varphi}{M}$, where the number of chunks κ is given by equation (6).

BiSam idea: The idea is that after both i and j have appeared φ times, the expected number of times $\{i,j\}$ is sampled is proportional to $s(i,j)$. Also, the number of samples follows a highly concentrated (binomial) distribution, so the true similarity can be estimated reliably for pairs that are sampled sufficiently often. For any f that is non-increasing in both parameters, the BiSam algorithm performs the sampling in time that is expected linear in the transaction size plus the number of samples. However, the time to process a transaction may be quadratic with non-negligible probability, which is problematic for application in a streaming context. We refer to [18], [19] for details.

Streaming adaptation: Two things allow us to arrive at a version suitable for streaming:

- While BiSam produces dependent samples, in the sense that the number of times two different itemsets is sampled is not independent, we show how to make the samples produced independent. This will ensure that the number of samples from each transaction is highly concentrated around its expectation.
- The requirement of minimum support φ will ensure that processing of a single transaction takes "linear time with high probability." More precisely: Any set of consecutive transactions with a total of $\log m$ items will require linear time with high probability.

To achieve independence we will change the sampling probabilities by rounding them down to the nearest negative power of 2. This means that the expected number of times $\{i,j\}$ is sampled is no longer exactly proportional to $s(i,j)$, but is changed by a factor $\gamma_{i,j} \in [1,2]$. However, since the sampling probability is known, which means that $\gamma_{i,j}$ will be constant for any given $\{i, j\}$, we can still use the sample counts to reliably estimate similarity.

Details: For a transaction T_t we can visualize the pairs in $T_t \times T_t$ as a 2-dimensional table, with rows and columns sorted by support, where we are interested in the pairs below the diagonal (index $i < j$). Since f is non-increasing the sampling probabilities are decreasing in each row and column. This means that for any $k > 0$, in time $O(|T_t|)$ we can determine what interval in each row of the table is to be sampled with probability 2^{-k}. To produce the part of the sample for one such interval, we describe a method for producing a random sample of $S = \{1, \ldots, \phi\}$, for a given integer ϕ, where each number is sampled with the same probability p. Since $p\phi$ may be much smaller than ϕ, we want the time to depend on the number of samples, rather than on ϕ. This can be achieved using a simple recursive procedure similar to the one used in efficient implementations of reservoir sampling: With probability $(1-p)^{\phi}$ we return an empty sample. Otherwise, we choose one random element x from S, and recursively take a sample of the set $S \backslash \{x\}$ with sampling probability p. The set S can be maintained in an array, where sampled numbers are marked. In case more than half of the numbers are marked, we construct a new array containing only unmarked numbers (the amortized cost of this is constant per marking). To select a random unmarked number we sample until one is found, which takes expected $O(1)$ time because no more than half of the numbers are marked.

In summary, for each sampling probability 2^{-k} we can compute the corresponding part of the sample in expected time $O(|T_i| + z_k)$, where z_k is the number of samples. This is done for $k = 1, 2, \ldots, 2\log(nm)$. Sampling probabilities smaller than $(nm)^{-2}$ are ignored, since the probability that any such pair would be sampled in any transaction is less than $1/m$. That is, with high probability ignoring such pairs does not influence the sample. To state our result, let $2^{-\mathbf{N}}$ denote the set of negative integer powers of 2.

Lemma 4: Let $\tilde{f} : \mathbf{N} \times \mathbf{N} \to 2^{-\mathbf{N}}$ be non-increasing in both parameters. Given a transaction T_t and support counts $|S_i|$ for its items, in expected time $O(|T_t|\log(nm) + z)$ we can produce a random sample of z 2-subsets of T_t such that:

- $\{i, j\}$ is sampled with probability $\tilde{f}(|S_i|, |S_j|)$ if $\tilde{f}(|S_i|, |S_j|) > (nm)^{-2}$, and otherwise with probability 0, and
- the samples are independent. ○

For all similarity measures in figure 1 and any feasible value of τ, the minimum support requirement will ensure that the expected number of samples in a transaction is at most $|T_t|$. This means that for each transaction T_t, the time spent is $O(|T_t|\log(nm))$ with high probability.

B. SampleCount

This phase sees the stream of pairs generated by the pair sampling, and has to filter out as many low similarity pairs as possible, while successfully identifying high similarity pairs. By the properties of pair sampling, this is essentially the task of identifying frequent pairs in the stream of samples. We aim for space usage that is smaller than that of standard algorithms for frequent item mining in a data stream. In order to accomplish this we use a modification of an algorithm by Demaine et al. [10]. Their algorithm finds frequent items in a randomly permuted stream of items, and so does not directly apply to our setting where only the transactions are assumed to come in random order. Demaine et al. are able to sample random elements by simply taking the first elements from the stream. This would not work in our setting, where all these elements might be pairs coming from the same transaction.

Reservoir sampling: Instead, we use a *reservoir sampling* method [22]. We sketch the mechanism here and we refer to the original paper for a complete description. Suppose we have a sequence of d items and we want to sample a random subset of the sequence. We first of all put in the sample the first s elements that we see. For each subsequent element, in position $t > s$, we will put it in the sample with probability s/t. When a new element has to be included in the sample, another one that is already part of the sample has to be evicted. Each element of the set of samples will be chosen as the victim with probability $1/s$. This technique ensures we will end up with a set of samples that is a true random sample of size s.

SampleCount: We consider the stream of pairs divided into κ chunks. The pair sampling generates these chunks such that each chunk corresponds to some set of transactions (i.e., all the pairs sampled from each transaction end up in the same chunk).

We run reservoir sampling on every other chunk to produce a truly random sample of size $s/2$. We then proceed to count the occurrences of the elements of the sample in the next chunk. Assume in the following that we number chunks by $[\kappa]$, such that reservoir sampling is done on even-numbered chunks, indexed by $[\kappa_{\text{even}}]$.

When doing the above, whenever we see a pair $\{i, j\}$ whose count must be updated, we weigh the sample by the factor $\gamma_{i,j}$ that got "lost" during the pair sampling phase, so as to consider an expected number of samples exactly proportional to $s(i, j)$. At the end of a counting chunk we estimate the similarities of all pairs sampled, and keep the $s/2$ largest similarities seen so far. At the end of the stream the similarity estimates found are returned to supersede the previous estimates. Pseudocode for the SampleCount algorithm is shown in figure 1.

Algorithm 1 Pseudocode for the SAMPLECOUNT phase.

procedure SAMPLECOUNT($P, s, size$) ▷ P is a stream of pairs, each of which has associated a similarity value. The length of P is known.
    $S_{\text{out}} \leftarrow \emptyset$
    while There are elements in P **do**
        $S' \leftarrow \emptyset$
        $S \leftarrow \emptyset$
        $t \leftarrow 0$
        $S \leftarrow$ the first $s/2$ elements in P
        while $(t < \frac{size}{2} - s/2)$ **do**
            $i \leftarrow$ the next element in P
            Choose uniformly at random a number $r \in [0, 1]$
            if $r \leq s/(s + 2t + 2)$ **then**
                Choose uniformly at random a victim from S and substitute it with i
            end if
            $t \leftarrow t + 1$
        end while
        initialize(S', S) ▷ S' is an associative array indexed on the distinct items present in S; initializing it means putting all its entries to 0
        while $(t < size)$ **do**
            $i \leftarrow$ the next element in P
            if $i \in S$ **then**
                $S'(i) \leftarrow S'(i) + \gamma_i$
            end if
            $t \leftarrow t + 1$
        end while
        Choose the s topmost distinct items between S'_{out} and S', and assign them to S'_{out}
    end while
    Return S'_{out}
end procedure

IV. ANALYSIS

Let S_i denote the set of transactions containing the element i. This means that $S_i \cap S_j$ is the set of transactions containing the pair $\{i, j\}$. Let S_i^1 denote the set of transactions containing i in the current prefix of the stream. Similarly, S_i^k will denote the set of transactions containing i in C_k, the chunk k of the suffix of the stream up to the point in which a new current prefix changes the counts of items occurrences. So $S_i^k = S_i \cap C_k$

Definition 5: Given $x, y \in \mathbf{R}$ we say that x (δ, L)-approximates y, written $x \overset{\delta,L}{\simeq} y$, if and only if $x \geq L$ implies $x \in [(1-\delta)y; (1+\delta)y]$. ○
The notation extends in the natural way to approximate inequalities.

In what follows we will use (δ, L)-approximations, where $L = C \log(mn)$ for a suitably large constant C (depending on the accuracy δ in Theorem 2). The task is to analyze the accuracy of the new approximation computed when the current prefix changes. We introduce two random events, GOODPERMUTATION (GP) and GOODBISAMSAMPLE (GBS), and bound the probability that they do not happen.

A permutation of the transactions is called *good* for $\{i, j\}$, denoted $\text{GP}_{i,j}$, if and only if the following conditions hold (for the new prefix):

1) $|S_i^1| \overset{\delta,L}{\simeq} |S_i|/2$ and $|S_j^1| \overset{\delta,L}{\simeq} |S_j|/2$;
2) $\forall k. |S_i^k \cap S_j^k| \overset{\delta,L}{\simeq} |S_i \cap S_j|/2k$;

Essentially, goodness means that the frequencies of individual items are close in the first and second half of the new prefix and the frequency of the pair is evenly spread over the chunks in the second part of the new prefix.

Lemma 6: Given $\delta \in [0; 1] \subseteq \mathbf{R}$, we have:

$$\Pr[\text{GP}_{i,j}] \geq 1 - 6 \cdot e^{\frac{-|S_i|\delta^2}{6}}$$

Proof: An interesting property of the random variables $|S_i^1|$ and $|S_i^k \cap S_j^k|$ is that they are negatively dependent [23]. First of all we bound the probability that $|S_i^1|$ is far from $|S_i/2|$. Using Chernoff bounds we can write:

$$\Pr[|S_i^1| - |S_i|/2| \leq \delta |S_i|/2] \leq 2 \cdot e^{-\frac{|S_i|\delta^2}{6}} \quad (1)$$

Looking at $|S_i^k \cap S_j^k|$ we can write:

$$\Pr[||S_i^k \cap S_j^k| - |S_i \cap S_j|/2\kappa| \leq \delta |S_i \cap S_j|/2\kappa] \leq 2 \cdot e^{-\frac{|S_i \cap S_j|\delta^2}{6\kappa}} \quad (2)$$

We use the fact that Chernoff bounds also holds for negatively dependent random variables. Since the last bound is the weakest of the three, the lemma follows. ■

We want $\mathrm{GP}_{i,j}$ to hold with probability $1-o(1/n^2)$ whenever items i and j both have support φ. From Lemma 6 we get that this holds if $|S_i \cap S_j| > C\kappa \log n$, for some constant C (depending on δ). If $s(i,j) > 2\kappa L f(\varphi,\varphi) \geq \kappa L/\varphi$ then $|S_i \cap S_j| \geq 2\kappa L$. Hence, a sufficient condition for the similarity is

$$s(i,j) > \kappa L/\varphi \; . \quad (3)$$

It remains to understand what is the probability that, given a good permutation, the pair sampler will take a number of samples for a given pair in each chunk k that leads to a $(1 \pm \delta)$-approximation of $s(i,j)$. We denote the latter event by $\mathrm{GBS}_{i,j,k}$, and want to bound the quantity $\Pr[\mathrm{GBS}_{i,j,k}|\mathrm{GP}_{i,j}]$.

For this purpose consider the random variable $X_{i,j,k}$ defined as the number of times we sample the pair $\{i,j\}$ in chunk k. Assuming $\mathrm{GP}_{i,j}$ we have that (over the randomness in the pair sampling algorithm) $\mathrm{E}[X_{i,j,k}] \overset{\delta,L}{\simeq} \tilde{f}(|S_i^1|,|S_j^1|)\tau|S_i \cap S_j|/2\kappa$. Since the occurrences of $\{i,j\}$ are independently sampled, we can apply a Chernoff bound to conclude $X_{i,j,k} \overset{\delta,L}{\simeq} \mathrm{E}[X_{i,j,k}]$. This leads to the conclusion:

Lemma 7: $X_{i,j,k} \overset{\delta,L}{\simeq} \tilde{f}(|S_i^1|,|S_j^1|)\tau|S_i \cap S_j|/2\kappa$ ◦

Suppose that $X_{i,j,k}$ is close to its expectation. Then we can use it, with $(1 \pm \delta)$-approximations of $|S_i|$ and $|S_j|$, to compute a $(1 \pm O(\delta))$-approximation of $s(i,j)$. This follows by analysis of the concrete functions f of the measures in Figure 1.

A sufficient condition on the similarity needed for a $(1 \pm \delta)$-approximation of $X_{i,j,k}$ can be inferred from lemma 7. If $s(i,j) \geq 4\kappa L/\tau$ then $\mathrm{E}[X_{i,j,k}] \geq s(i,j)\tau/4\kappa \geq L$. So it suffices to enforce:

$$s(i,j) \geq 4\kappa L/\tau \; . \quad (4)$$

In order to have $O(mb)$ pairs produced by the pair sampling phase, we will choose $\tau = 4\varphi/M$. The expected number of pair samples from T_t is less than $|T_t|^2 \tau f(\varphi,\varphi)$, using that f is decreasing. For all measures we consider, $f(\varphi,\varphi) \leq 1/\varphi$, so $|T_t|^2 \tau f(\varphi,\varphi) \leq |T_t|^2/M \leq |T_t|$.

It remains to understand which is the probability that a pair of items, each with support at least φ, is not sampled by SampleCount. Let the random variable $X_{\cdot,\cdot,k}$ represent the total number of samples taken in chunk k. The probability that a $\{i,j\}$ is sampled in chunk k is $X_{i,j,k}/X_{\cdot,\cdot,k}$, so the probability that it does not get sampled in any (even-numbered) chunk is $\prod_{k \in [\kappa_{\text{even}}]}(1 - X_{i,j,k}/X_{\cdot,\cdot,k})^s$. We have seen before that $X_{i,j,k} \overset{\delta,L}{\geq} s(i,j)\tau/4\kappa$. For what concerns $X_{\cdot,\cdot,k}$ using a Chernoff bound we can get: $X_{\cdot,\cdot,k} \overset{\delta,L}{\simeq} \mathrm{E}[X_{\cdot,\cdot,k}] \leq mb/\kappa$, by means of having shown that we expect a linear number of samples. So we can compute:

$$\prod_{k \in [\kappa_{\text{even}}]} (1 - X_{i,j,k}/X_{\cdot,\cdot,k})^s \leq \left(1 - \frac{s(i,j)\tau\kappa}{2\kappa\gamma_{i,j} mb}\right)^{s\kappa/2}$$

$$\leq \left(1 - \frac{s(i,j)\tau}{4mb}\right)^{s\kappa/2} \leq C \exp\left[-\frac{s(i,j)\tau s\kappa}{8mb}\right]$$

In order for this probability to be small enough ($O(1/m^2)$), we need to bound the similarity to

$$s(i,j) \geq \frac{8mbL}{s\kappa\tau} \quad (5)$$

To choose the best value of κ we balance constraints (3) and (5), getting:

$$\frac{\kappa L}{\varphi} = \frac{mbL}{s\kappa\tau} \Rightarrow \kappa = \sqrt{\frac{mbM}{s}} \quad (6)$$

From which we can deduce:

$$s(i,j) = \frac{L}{\varphi} \max\left\{\sqrt{\frac{mbM}{s}}, M\right\} \; . \quad (7)$$

V. Dataset characteristics

We have computed, for a selection of the datasets hosted on the FIMI web page[1], the ratios between the number of occurrences of single items and pairs in the first half of the transactions and the total number of occurrences of the same items or pairs. The values of some of this ratios, the most representative, are plotted figure 3; on the x-axis items or pairs are spread evenly, after they have been sorted according to their associated ratio. The y-axis represents the value of the ratios. We have taken into account only items and pairs whose support is over 20 occurrences in the whole dataset, in order to avoid the noise that could be generated by very rare elements. As we can see, the number of occurrences and co-occurrences are not so far from what would be expected under a random permutation of the transactions. The synthetic data set behaves exactly like we would expect under a random permutation, with the ratio being very close to $1/2$ for almost all items/pairs.

This means that even for real data sets, where the order of transactions is not random, the sampling probabilities used in the pair sampling are reasonably close to the ones that would be obtained under the random permutation assumption.

VI. Conclusions

We presented the first study concerning the problem of mining similar pairs from a stream of transactions that does rely on the similarity of items and not only on the frequency of pairs. A thorough experimental study of (carefully engineered versions of) the presented algorithm remains to be carried out.

[1] `http://fimi.cs.helsinki.fi/`

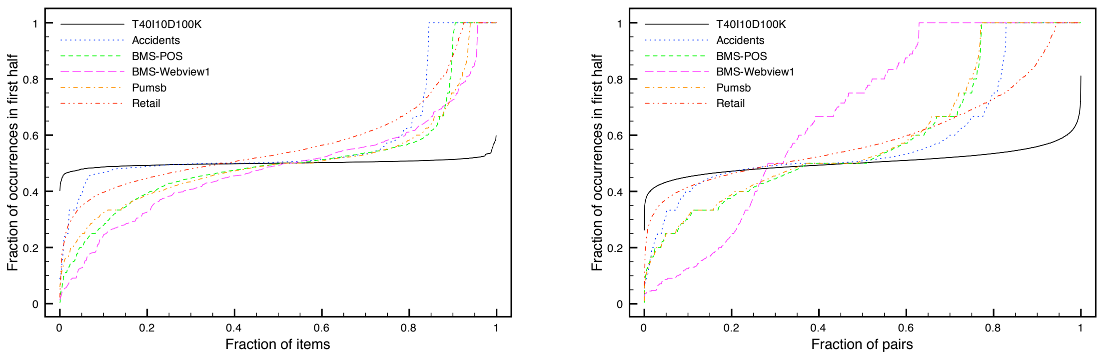

Figure 3. Plots of the ratios $|S_i^1|/|S_i|$ and $|S_i^1 \cap S_j^1|/|S_i \cap S_j|$.

References

[1] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang, "Finding interesting associations without support pruning," *IEEE Trans. Knowl. Data Eng*, vol. 13, no. 1, pp. 64–78, 2001.

[2] Y.-K. Lee, W.-Y. Kim, Y. D. Cai, and J. Han, "Comine: Efficient mining of correlated patterns," in *Proc. IEEE International Conference on Data Mining (ICDM 2003)*. IEEE Computer Society, 2003, pp. 581–584.

[3] E. Omiecinski, "Alternative interest measures for mining associations in databases," *IEEE Trans. Knowl. Data Eng*, vol. 15, no. 1, pp. 57–69, 2003.

[4] J. Han and M. Kamber, *Data Mining: Concepts and Techniques, 2nd edition*. Morgan Kaufmann, 2006.

[5] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proc. International Conference On Very Large Data Bases (VLDB 1994)*. Morgan Kaufmann Publishers, Inc., Sep. 1994, pp. 487–499.

[6] 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.

[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." Morgan Kaufmann, 2002, pp. 358–369.

[9] G. Cormode and S. Muthukrishnan, "What's hot and what's not: tracking most frequent items dynamically," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 249–278, 2005.

[10] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency estimation of internet packet streams with limited space," in *Proc. 10th Annual European Symposium Algorithms (ESA 2002)*, 2002, pp. 348–360.

[11] 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.

[12] A. Chakrabarti, G. Cormode, and A. McGregor, "Robust lower bounds for communication and stream computation," in *STOC*, C. Dwork, Ed. ACM, 2008, pp. 641–650.

[13] S. Guha and A. McGregor, "Stream order and order statistics: Quantile estimation in random-order streams," *SIAM Journal on Computing*, vol. 38, no. 5, pp. 2044–2059.

[14] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *J. Comput. Syst. Sci.*, vol. 58, no. 1, pp. 137–147, 1999.

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

[16] 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.

[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] A. Campagna and R. Pagh, "Finding associations and computing similarity via biased pair sampling," in *Proc. 9th IEEE International Conference on Data Mining (ICDM 2009)*.

[19] ——, "Finding associations and computing similarity via biased pair sampling," *Pending journal publication*, 2010.

[20] E. Kushilevitz and N. Nisan, *Communication complexity*. New York: Cambridge University Press, 1997.

[21] A. Campagna and R. Pagh, "Finding associations and computing similarity via biased pair sampling," *Knowledge and Information Systems (to appear)*.

[22] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.

[23] D. Dubhashi and D. Ranjan, "Balls and bins: a study in negative dependence," *Random Struct. Algorithms*, vol. 13, no. 2, pp. 99–124, 1998.