Finding associations and computing similarity via biased pair sampling

Andrea Campagna · Rasmus Pagh

Abstract Sampling-based methods have previously been proposed for the problem of finding interesting associations in data, even for low-support items. While these methods do not guarantee precise results, they can be vastly more efficient than approaches that rely on exact counting. However, for many similarity measures no such methods have been known. In this paper, we show how a wide variety of measures can be supported by a simple biased sampling method. The method also extends to find high-confidence association rules. We demonstrate theoretically that our method is superior to exact methods when the threshold for “interesting similarity/confidence” is above the average pairwise similarity/confidence, and the average support is not too low. Our method is particularly advantageous when transactions contain many items. We confirm in experiments on standard association mining benchmarks that we obtain a significant speedup on real data sets. Reductions in computation time of over an order of magnitude, and significant savings in space, are observed.

Keywords Algorithms · Sampling · Data mining · Association rules

1 Introduction

A central task in data mining is finding associations in a binary relation. Typically, this is phrased in a “market basket” setup, where there is a sequence of baskets (from now on “transactions”), each of which is a set of items. The goal is to find patterns such as “customers who buy diapers are more likely to also buy beer”. There is no canonical way of defining whether an association is interesting—indeed, this seems to depend on problem-specific factors not captured by the abstract formulation. As a result, a number of measures exist: In this paper,
we deal with some of the most common measures, including Jaccard [14], lift [2,8], cosine, and all confidence [28,33]. In addition, we are interested in high-confidence association rules, which are closely related to the overlap coefficient similarity measure. We refer to [24, Chap. 5] for general background and discussion of similarity measures.

In the discussion, we limit ourselves to the problem of binary associations, i.e., patterns involving pairs of items. There is a large literature considering the challenges of finding patterns involving larger itemsets, taking into account the aspect of time, multiple-level rules, etc. While some of our results can be extended to cover larger itemsets, we will for simplicity concentrate on the binary case. Previous methods rely on one of the following approaches:

1. Identifying item pairs \((i, j)\) that “occur frequently together” in the transactions—in particular, this means counting the number of co-occurrences of each such pair—or

2. Computing a “signature” for each item such that the similarity of every pair of items can be estimated by (partially) comparing the item signatures.

Our approach is different from both these approaches, and generally offers improved performance and/or flexibility. In some sense, we go directly to the desired result, which is the set of pairs of items with similarity measure above some user-defined threshold \(\Delta\). Our method is sampling based, which means that the output may contain false positives, and there may be false negatives. However, these errors are rigorously understood, and can be reduced to any desired level, at some cost of efficiency—our experimental results are for a false negative probability of less than 2%. The method for doing sampling is the main novelty of this paper, and is radically different from previous approaches that involve sampling.

The main focus in many previous association mining papers has been on space usage and the number of passes over the data set, since these have been recognized as main bottlenecks. We believe that time has come to also carefully consider CPU time. A transaction with \(b\) items contains \(\binom{b}{2}\) item pairs, and if \(b\) is not small the effort of considering all pairs is non-negligible compared to the cost of reading the itemset. This is true in particular if data resides in RAM, or on a modern SSD that is able to deliver data at a rate of more than a gigabyte per second. One remedy that has been used (to reduce space, but also time) is to require high support, i.e., define “occur frequently together” such that most items can be thrown away initially, simply because they do not occur frequently enough (they are below the support threshold). However, as observed in [14] this means that potentially interesting or useful associations (e.g., correlations between genes and rare diseases) are not reported. In this paper we consider the problem of finding associations without support pruning. Of course, support pruning can still be used to reduce the size of the data set before our algorithms are applied.

In the following sections, we first discuss the need for focusing on CPU time in data mining, and then elaborate on the relationship between our contribution and related works.

1.1 I/O versus CPU

In recent years, the capacity of very fast storage devices has exploded. A typical desktop computer has 4–16 GB of RAM, that can be read (sequentially) at a speed of at least 800 million 32-bit words per second. The flash-based ioDrive Duo of Fusion-io offers up to over a terabyte of storage that can be read at around 400 million 32-bit words per second. Thus, even massive data sets can be read at speeds that make it challenging for CPUs to keep up. An 8-core system must, for example, process 100 million (or 50 million) items per core per second. At 3 GHz this is 33 clock cycles (or 66 clock cycles) per item. This means that any kind of processing that is not constant time per item (e.g., using time proportional to the
size of the transaction containing the item) is likely to be CPU bound rather than I/O bound. For example, a hash table lookup requires on the order of 5–10 ns even if the hash table is L2 cache-resident (today less than 10 MB per core). This gives an upper limit of 100–200 million lookups per second in each core, meaning that any algorithm that does more than a dozen hash table operations per item (e.g., updating the count of some item pairs) is definitely CPU bound, rather than I/O bound. In conclusion, we believe it is time to carefully consider optimizing internal computation time, rather than considering all computation as “free” by only counting I/Os or number of passes. Once CPU efficient algorithms are known, it is likely that the remaining bottleneck is I/O. Thus, we also consider I/O efficient versions of our algorithm.

1.2 Previous work

1.2.1 Exact counting of frequent itemsets

The approach pioneered by the A-Priori algorithm [3,4], and refined by many others (see e.g., [9,23,29,34,35]), allows, as a special case, finding all item pairs \((i, j)\) that occur in more than \(k\) transactions, for a specified threshold \(k\). However, for the similarity measures we consider, the value of \(k\) must in general be chosen as a low constant, since even pairs of very infrequent items can have high similarity. This means that such methods degenerate to simply counting the number of occurrences of all pairs, spending time \(\Theta(b^2)\) on a transaction with \(b\) items. Also, generally the space usage of such methods (at least those requiring a constant number of passes over the data) is at least 1 bit of space for each pair that occurs in some transaction. An experimental comparison for some 2004 state-of-the-art algorithms for frequent itemset mining is carried out in [22].

The problem of counting the number of co-occurrences of all item pairs is in fact equivalent to the problem of multiplying sparse 0-1 matrices. To see this, consider the \(n \times m\) matrix \(A\) in which each row \(A_i\) is the incidence vector having 1 in position \(p\) iff the \(i\)th element in the set of items appears in the \(p\)th transaction. Each entry \(\tilde{A}_{i,j}\) of the \(n \times n\) matrix \(\tilde{A} = A \times A^T\) represents the number of transactions in which the pair \((i, j)\) appears. The best theoretical algorithms for (sparse) matrix multiplication [5,16,40] scale better than the A-Priori family of methods as the transaction size gets larger, but because of huge constant factors this is so far only of theoretical interest.

1.2.2 Sampling methods

Toivonen [36] investigated the use of sampling to find candidate frequent pairs \((i, j)\): Take a small, random subset of the transactions and see what pairs are frequent in the subset. This can considerably reduce the memory used to actually count the number of occurrences (in the full set), at the cost of some probability of missing a frequent pair. This approach is good for high-support items, but low-support associations are likely to be missed, since few transactions contain the relevant items.

Cohen and Lewis [15] present an algorithm for approximate matrix multiplication that can be used for finding similar pairs (in the same approximate sense that we pursue here) according to some similarity measures such as cosine and lift. In fact, for these similarity measures their algorithm will produce estimator random variables with the same distribution as ours (binomial), in a similar time bound. Our approach handles more general similarity measures, uses less space in addition to the input, and we couple the sampling with a space-efficient algorithm for finding the most similar pairs.
1.2.3 Locality-sensitive hashing

Cohen et al. [14] proposed the use of another sampling technique, called *min-wise independent hashing*, where a small number of occurrences of each item (a “signature”) is sampled. This means that occurrences of items with low support are more likely to be sampled. As a result, pairs of (possibly low-support) items with high *Jaccard coefficient* are found—with a probability of false positives and negatives. A main result of [14] is that the time complexity of their algorithm is proportional to the sum of all pairwise Jaccard coefficients, plus the cost of initially reading the data. Our main result exactly the same form, but has the advantage of supporting a wide class of similarity measures.

Min-wise independent hashing belongs to the class of *locality-sensitive hashing* methods [26]. Another such method was described by Charikar [12], who showed how to compute succinct signatures whose Hamming distance reflects angles between incidence vectors. This leads to an algorithm for finding item pairs with cosine similarity above a given threshold (again, with a probability of false positives and negatives), that uses linear time to compute the signatures, and $\Theta(n^2)$ time to find the similar pairs, where $n$ is the number of distinct items in all transactions. Charikar also shows that many similarity measures, including some measures supported by our algorithm, cannot be handled using the approach of locality-sensitive hashing.

1.2.4 Deterministic signature methods

In the database community, finding all pairs with similarity above a given threshold is sometimes referred to as a “similarity join.” Recent results on similarity joins include [6,13,38,39]. While not always described in this way, these methods can be seen as deterministic analogs of the locality-sensitive hashing methods, offering *exact* results. The idea is to avoid computing the similarity of every pair by employing succinct “signatures” that may serve as witnesses for low similarity. Most of these methods require the signatures of every pair of items to be (partially) compared, which takes $\Omega(n^2)$ time. However, the worst-case asymptotic performance appears to be no better than the A-Priori family of methods. A similarity join algorithm that runs faster than $\Omega(n^2)$ in some cases is described in [6]. However, this algorithm exhibits a polynomial dependence on the maximum number $k$ of differences between two incidence vectors that are considered similar, and for many similarity measures the relevant value of $k$ may be linear in the number $m$ of transactions.

1.2.5 Larger significant itemsets

Wu et al. [37] consider mining of significant itemsets according to a measure related to *lift*. In particular, their approach extends to negative associations. Zhang et al. [41] use the same approach, presenting also a fuzzy variant. However, both approaches require exact counting of the number of co-occurrences of all itempairs (where each item is above the support threshold). Therefore, the performance for finding significant/similar *pairs* is similar to the performance of other exact counting methods.

1.2.6 Streaming algorithms

In our recent paper [11], a modification of the technique presented in this paper finds similar pairs in a randomly ordered stream of transactions. In the streaming framework one can
Finding associations and computing similarity via biased pair sampling

see and store only one transaction at a time. In order to tackle the difficulties of the new environment in an efficient fashion, a support threshold is used in order to avoid sampling too many pairs. Moreover the sampling is carried out in a slightly different way, in order to guarantee that it happens in “real time”. Also, we replace the final step of the algorithm by the method of [20] that is particularly well suited to the setting of randomly ordered data. In [11], we also show a lower bound on the space that every algorithm mining similar \( k \)-itemsets must use, in a worst-case scenario, extending a previous lower bound in [19].

1.3 Our results

In this paper, we present a novel sampling technique to handle a variety of measures (including Jaccard, lift, cosine, and all_confidence), even finding similar pairs among low support items. The idea is to sample a subset of all pairs \((i, j)\) occurring in the transactions, letting the sampling probability be a function of the supports of \(i\) and \(j\), such that the expected number of times a pair is sampled is proportional to \(s(i, j)\). Given a threshold \(\Delta\), the sampling rate can be scaled such that any pair with similarity above \(\Delta\) is likely to be sampled several times, whereas pairs with similarity “far below” \(\Delta\) are likely not to be sampled. The number of times a pair is sampled follows a binomial distribution, which allows us to use the sample, in a filtering phase, to infer which pairs are likely to have similarity above the threshold, with rigorous bounds on false negative and false positive probabilities.

A naïve implementation of this idea would still use quadratic time for each transaction, but we show how to do the sampling time that is linear in the size of the transaction and number of sampled pairs. In turn, the expected number of samples is proportional the sum of all pairwise similarities between items. We will argue that this running time is the best one could hope for with no conditions on the distribution of pairwise similarities. Under reasonable assumptions, e.g., that the average support is not too low, this gives a speedup of a factor \(\Omega(b)\), where \(b\) is the average size of a transaction, compared to exact counting methods.

We show in extensive experiments on standard data sets for testing data mining algorithms that our approach (with sampling rate resulting in a 1.8% false negative probability) gives speedup factors in the vicinity of an order of magnitude, as well as significant savings in the amount of space required, compared to exact counting methods. We also present evidence that for data sets with many distinct items, our algorithm may perform significantly less work than methods based on locality-sensitive hashing 4.1.

1.4 Notation and framework

Let \(T_1, \ldots, T_m\) be a sequence of transactions, \(T_j \subseteq [n]\). For \(i = 1, \ldots, n\) let \(S_i = \{ j \mid i \in T_j \}\), i.e., \(S_i\) is the set of occurrences of item \(i\).

We are interested in finding associations among items, and consider a framework that captures the most common measures from the data mining literature. Specifically, we can handle a similarity measure \(s(i, j)\) if there exists a function \(f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}_+\) that is non-increasing in both parameters, and such that:

\[
s(i, j) = |S_i \cap S_j| \cdot f(|S_i|, |S_j|).
\]

Table 1 shows particular measures that fall within this framework. The monotonicity requirement on \(f\) holds for any reasonable similarity measure: It says that for a given value of \(|S_i \cap S_j|\), adding an occurrence of \(i\) or \(j\) should not increase the similarity. In the following, we assume that \(f\) is computable in constant time, which is clearly a reasonable assumption.
for the measures of Table 1. In the time analysis, we will further assume that $f$ is polynomial in the sense that changing the input by a constant changes the value of $f$ by a constant, specifically that $f(c_1, c_2) = O(f(c_1/2, c_2/2))$ for all $c_1, c_2$. This clearly holds for all similarity measures we consider, and arguably for any reasonable similarity measure.

We end with some observations on other measures that can be handled directly or indirectly by our framework.

### 1.4.1 Composite measures

Notice that if $f_1(|S_i|, |S_j|)$ and $f_2(|S_i|, |S_j|)$ are both non-increasing, then any linear combination $\alpha f_1 + \beta f_2$, where $\alpha, \beta > 0$, is also non-increasing. Similarly, $\min(\alpha f_1, \beta f_2)$ is non-increasing. This allows us to use BiSam to directly search for pairs with high similarity according to several measures (corresponding to $f_1$ and $f_2$), e.g., pairs with cosine similarity at least 0.7 and $\text{lift}$ at least 2.

### 1.4.2 Handling the Jaccard measure

We observe the following relationship between the Jaccard and dice similarity measures:

$$s_{\text{Jaccard}}(i, j) = s_{\text{dice}}(i, j)/(1 - s_{\text{dice}}(i, j)).$$

Observe that $s_{\text{Jaccard}}$ grows monotonically with $s_{\text{dice}}$, and that the derivative wrt. $s_{\text{dice}}$ is in the range $[1; 4]$. This means that most questions about Jaccard similarity can be translated into questions about dice similarity. For example, if we are interested in all pairs with a certain Jaccard similarity, this translates directly into all pairs with a certain dice similarity.

### 2 The BiSAM algorithm

For a given parameter $\tau > 0$, our goal is to sample pairs of items such that $(i, j)$ is sampled $\tau \cdot s(i, j)$ times in expectation. Also, we want the occurrences of an item pair to be sampled independently, such that the number of samples follow a (highly concentrated) binomial distribution.

The output of our algorithm, named BiSam (for biased sampling), will be an unordered sequence of samples. It will be convenient to work with weighted samples, i.e., with each sampled pair we associate a positive real number (which will be at least 1, but not necessarily integer). We define the number of occurrences of a pair $(i, j)$ as the sum of the associated numbers.
procedure BiSAM($T_1, \ldots, T_m; f, \tau$)
  $c := $ITEMCOUNT($T_1, \ldots, T_m$);
  $c' := $FLOORToPOWEROF2($c$);
  for $t := 1$ to $m$ do
    sort $T_t$ s.t. $c'(T_t[j]) \leq c'(T_t[j + 1])$ for $1 \leq j < |T_t|$;
    let $r$ be a random number in $[0; 1]$;
    for $i := 1$ to $|T_t| do$
      $j := i + 1$;
      while $j \leq |T_t|$ and $f(c'(T_t[i]), c'(T_t[j]))\tau > r$ do
        if $f(c(T_t[i]), c(T_t[j]))\tau > r$ then
          output $\{(T_t[i], T_t[j]), \max(1, f(c(T_t[i]), c(T_t[j]))\tau)\}$;
      end
    end
  end

Fig. 1 Pseudocode for the BiSAM algorithm. The procedure ITEMCOUNT(·) returns a function (hash map) that contains the number of occurrences of each item. FLOORToPOWEROF2(·) returns a function that is obtained from $c$ by rounding down occurrence counts to the nearest integer power of 2. $T_t[j]$ denotes the $j$th item in transaction number $t$.

We observe that all measures in Table 1 are symmetric, $s(i, j) = s(j, i)$, so it suffices to sample either $(i, j)$ or $(j, i)$. Our pseudocode will make this optimization, by dealing with sets $\{i, j\}$, but can easily be modified to also handle asymmetric measures.

2.1 Algorithm idea

Fig. 1 shows pseudocode for the BiSAM algorithm. In an initial pass over, the data it computes item support counts, stored in a (hash) map $c$. For convenience, we precompute item counts rounded down to the nearest power of 2, stored as $c'$.

After the initial pass, the algorithm iterates through the transactions once more. For each transaction $T_t$, some number of size-2 subsets of $T_t$ are output, with a weight associated with each pair. The processing of a transaction starts with sorting the items according to value of $c'$, i.e., they are “roughly sorted” according to support. Below, we discuss how this can be done in linear time by exploiting that $c'$ has only $\lceil \log m \rceil$ possible distinct values.

The main loop of the algorithm outputs those pairs $\{T_t[i], T_t[j]\}$ for which $f(c(T_t[i]), c(T_t[j]))\tau > r$, where $r$ is a random real number in $[0; 1)$. This can be seen as follows. For each value of $i$ the algorithm iterates through $j = i + 1, i + 2, \ldots$ until $j = |T_t|$ or $f(c'(T_t[i]), c'(T_t[j]))\tau \leq r$. In both cases we can conclude, since $f$ is non-increasing, that no more pairs with the current value of $i$ should be reported. The total time for the outer loop is $O(|T_t|)$, and the time for the inner loop is proportional to the number of pairs $\{i, j\}$ for which $f(c'(T_t[i]), c'(T_t[j]))\tau < r$.

A pair $\{T_t[i], T_t[j]\}$ is sampled with probability $\min(1, f(c(T_t[i]), c(T_t[j]))\tau)$. In cases where we sample with probability less than 1, we associate a weight of 1 with the sample; otherwise we assign the weight $f(c(T_t[i]), c(T_t[j]))\tau$. In either case, the expected weight assigned to the sample is $f(c(T_t[i]), c(T_t[j]))\tau$. Thus, we have the following:

Lemma 1 Let $M(i, j)$ denote the total weight of the pair $\{i, j\}$ in the output of BiSAM. For all pairs $\{i, j\}$, where $i \neq j$ and $c(i) \leq c(j)$, if $f(c(i), c(j))\tau < 1$ then at the end of the procedure, $M(i, j)$ has binomial distribution with $|S_i \cap S_j|$ trials and mean

$$|S_i \cap S_j| f(|S_i|, |S_j|)\tau = s(i, j)\tau.$$

If $f(c(i), c(j))\tau \geq 1$, then at the end of the procedure $M(i, j) = s(i, j)\tau$ with probability 1.
Table 2  Items in the example, with corresponding ITEMCOUNT values

| Item | Occurrences | Item | Occurrences |
|------|-------------|------|-------------|
| $i$  | $c(i)$      | $i$  | $c(i)$      |
| 1    | 66          | 6    | 31          |
| 2    | 66          | 7    | 28          |
| 3    | 65          | 8    | 5           |
| 4    | 60          | 9    | 5           |
| 5    | 58          | 10   | 3           |

Table 3  The table represents the function $c'$ obtained by the application of the functional FloorToPowerOf2 to the function $c$

| Item | FloorToPowerOf2 | Bucket | Item | FloorToPowerOf2 | Bucket |
|------|-----------------|--------|------|-----------------|--------|
| $i$  | $c'(i)$         | $i$    | $i$  | $c'(i)$         | $i$    |
| 1    | 64              | 5      | 6    | 16              | 3      |
| 2    | 64              | 5      | 7    | 16              | 3      |
| 3    | 64              | 5      | 8    | 4               | 2      |
| 4    | 32              | 4      | 9    | 4               | 2      |
| 5    | 32              | 4      | 10   | 2               | 1      |

Hence, the elements are grouped in 5 buckets

In other words, the weight $M(i, j)$ is an unbiased estimator for $s(i, j)\tau$ that is tightly concentrated. This allows us to give firm guarantees on the probability that $M(i, j)/\tau$ deviates substantially from $s(i, j)$, as discussed in Sect. 3.1.

2.2 Implementation

The best implementation of the subprocedure ITEMCOUNT depends on the relationship between available memory and the number $n$ of distinct items. If there is sufficient internal memory, it can be efficiently implemented using a hash table. In the following, we first consider the standard model (often referred to as the “RAM model”), where the hash tables fit in internal memory, and assume that each insertion takes constant time. Then, we consider the I/O model, for which an I/O efficient “sort-and-count” implementation is discussed (Sect. 2.3.2).

The use of the standard implementation of the sorting step would require time $O(|T_t| \log |T_t|)$. However, we observe that there are only $\lceil \log m \rceil$ possible values of $c'$, so this can be done more efficiently by bucket sorting (one bucket per value). In case $|T_t| < \log m$, we need a few extra tricks to get a linear time algorithm. We stress that these tricks are described for the purpose of the theoretical result, and are unlikely to yield an advantage in practice due to increased constant factors.

We modify standard bucket sort as follows: The buckets should be initialized in a lazy fashion, such that we do not use time on buckets that contain no elements. Also, when traversing the buckets to form the result we should not spend time on empty buckets. This can be achieved by maintaining a bit vector of length $\lceil \log m \rceil$ indicating which buckets are non-empty. Then, the non-empty buckets can be found in time $O(|T_t|)$ using a constant-time least-significant-bit computation.

Example  Suppose ITEMCOUNT has been run and the supports of items 1–6 are as shown in Table 2. Table 3 shows the values associated to each element by the function $c'$.  

Springer
Table 4 Pairs selected from $T_t$ in the example. Notice that after realizing the bucket pair $(2, 5)$ does not satisfy the inequality $f(c'(9), c'(3)) \tau > r$, the algorithm will not take into account the pairs of bucket $(2, 4)$

| $i$ | $j$ | $f(c(i), c(j)) \tau$ | $i$ | $j$ | $f(c(i), c(j)) \tau$ |
|-----|-----|---------------------|-----|-----|---------------------|
| 10  | 9   | 3.61                | 10  | 2   | 0.99                |
| 10  | 8   | 3.61                | 10  | 1   | 0.99                |
| 10  | 7   | 1.52                | 9   | 8   | 2.8                 |
| 10  | 6   | 1.45                | 9   | 7   | 1.18                |
| 10  | 5   | 1.06                | 9   | 6   | 1.12                |
| 10  | 4   | 1.04                | 8   | 7   | 1.18                |
| 10  | 3   | 1.00                | 8   | 6   | 1.12                |

Moreover, since $f(c'(7), c'(6)) \tau < r$ the pairs of buckets $(3, 3), (3, 4), (3, 5), (4, 4), (4, 5)$ and $(5, 5)$

Suppose now that the transaction $T_t = \{6, 4, 5, 3, 2, 1\}$ is given. Note that its items are written according to the mapping given by the function $c'$. Assuming the similarity measure is cosine, $\tau = 14$, and $r$ for this transaction equal to 0.9, the algorithm would sample from $T_t \times T_t$ the pairs shown in Table 4.

2.3 Analysis of running time

We provide a running time analysis both in the standard (RAM) model and in the I/O model of [1]. In the latter case, we present an external memory efficient implementation of the algorithm, IOBiSam. Let $b$ denote the average number of items in a transaction, i.e., there are $bm$ items in total. Also, let $z$ denote the number of pairs reported by the algorithm.

2.3.1 Running time in the standard model

The first part of the algorithm just goes through the input, using expected time $O(mb)$. The first and last part of the algorithm clearly runs in expected time $O(mb + z)$. The time for reporting the result is dominated by the time used for the main loop. The sorting of a transaction with $b_1$ items, performed as described above, takes $O(b_1)$ time, and in particular the total cost of all sorting steps is $O(mb)$. Similarly, the total cost of iterating through all transactions is $O(mb)$ if the cost of the while loop is not counted.

The time for the while loop is proportional to the number of pairs $\{i, j\}$ for which $f(c'(T_t[i]), c'(T_t[j])) \tau < r$. That is, the probability that we spend time $O(1)$ on the pair $\{i, j\}$ is min$(1, f(c'(T_t[i]), c'(T_t[j])) \tau)$. Summing over all pairs and all transactions we get an expected cost of at most:

$$\sum_i \sum_{\{i, j\} \subseteq T_t} f(c'(i), c'(j)) \tau = O \left( \sum_i \sum_{\{i, j\} \subseteq T_t} f(c(i), c(j)) \tau \right)$$

using the assumption that $f$ is polynomial. Reordering the terms of the sum, we get an expected cost of:

$$\sum_{\{i, j\}} |S_i \cap S_j| f(c(i), c(j)) \tau = \sum_{\{i, j\}} s(i, j) \tau.$$
Theorem 2 Suppose we are given transactions $T_1, \ldots, T_m$, each a subset of $[n]$, with $mb$ items in total, and that $f$ is a polynomial function such that $s(i, j) = |S_i \cap S_j| f(|S_i|, |S_j|)$. Then, the expected time complexity of BiSAM($T_1, \ldots, T_m; f, \tau$) in the standard model is $O(mb + \tau \sum_{1 \leq i < j \leq n} s(i, j))$.

Discussion In most of our experiments, the first of the two terms dominated the time complexity. This means that the running time is close to optimal, as $O(mb)$ is the time for just reading the input. However, we also found that for some data sets with mainly low-support items, the second term (the cost of reporting samples) dominated.

A comparison can be made with the complexity of schemes counting the occurrences of all pairs. Such methods use time $\Omega(mb^2)$, which is a factor $\Omega(b)$ larger than the first term. In fact, the difference will be larger if the distribution of transaction sizes is not even.

Similarity threshold The parameter $\tau$ should be chosen such that $s(i, j)$ is not too small, e.g., $s(i, j)\tau$ for the pairs that are considered highly similar. It is instructive to parameterize in terms of the threshold $\Delta$ for “interesting similarity”. To ensure that interesting pairs are reported with good probability, $\tau$ must be chosen such that $\tau \Delta$ is not too small, e.g., in our experiments we use $\tau \Delta \approx 15$.

A reasonable assumption is that $\Delta$ is greater than the average similarity, i.e., $\Delta \geq \sum_{1 \leq i < j \leq n} s(i, j)/\frac{n}{2}$. In many cases $\Delta$ will be much greater than the average similarity, as discussed in Sect. 4.1. But just using the above we can obtain the following simple (in some cases pessimistic) upper bound on the time complexity:

Corollary 3 If $\Delta = O(1/\tau)$ is no smaller than the average pairwise similarity, then expected time complexity of BiSAM is $O(mb + n^2)$.

This means that under the assumption of the corollary, we win a factor of at least $\min(b, m(b/n)^2)$ compared to the exact counting approach. If we let $\sigma = mb/n$ denote the average support, the speedup can be expressed as $\Omega(b \min(1, \sigma/n))$. So, if the average support is $n$ or more, we gain a factor $\Omega(b)$.

Independent items As further evidence for (or explanation of) why the time complexity of the second term may be close to linear, we consider an input where each item $i$ appears in a given transaction with probability $p_i$, independently of all other items. Thus, the probability that distinct items $i$ and $j$ appear in a transaction is $p_i p_j$. We observe that each similarity measure $s(i, j)$ in Table 1, with the exception of $\text{lift}$, satisfies $s(i, j) \leq \tilde{s}(i, j)$, where $\tilde{s}(i, j) = \frac{|S_i \cap S_j|}{|S_i|} + \frac{|S_i \cap S_j|}{|S_j|}$. Thus, we get an upper bound on running time for these measures by considering the similarity measure $\tilde{s}(i, j)$. Observe that the expected value of $\tilde{s}(i, j)$ is $p_i + p_j$ by linearity of expectation. Hence, the expected sum of similarities is:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} p_i + p_j \leq \sum_{i=1}^{n} p_i n + \sum_{j=1}^{n} np_j = 2n .$$

This means that the running time of BiSAM is $O(mb + \tau n)$ for independent items. Usually $mb \gg \tau n$, so the first term dominates.

2.3.2 Running time in the I/O model

We now present IOBiSAM, an I/O efficient implementation of the BiSAM algorithm. The rest of the paper can be read independently of this section. As before, we assume the similarity measure is represented as $s(i, j) = |S_i \cap S_j| f(|S_i|, |S_j|)$.
In order to compute the support of each item, which means computing the ITEMCOUNT function, a sorting of the dataset’s items is carried out. It is necessary to keep track of which transaction each item belongs to. To compute the sorted list of items, \( O(\frac{N}{B} \log \frac{m}{B} \frac{N}{M}) \) I/Os are needed [1], where \( N = mb \) is the number of pairs \( c = (\text{item}, \text{Transaction ID}) \). \( M \) is the number of such pairs that fit in memory, and \( B \) is the number of pairs that fit in a memory page, so it takes just \( O(\frac{N}{B}) \) I/Os to compute and store the tuples \( c(\text{item}, \text{support}, \text{Transaction ID}) \).

We then sort the tuples according to transaction ID, and secondarily according to support, again using \( O(\frac{N}{B} \log \frac{m}{B} \frac{N}{M}) \) I/Os. This gives us each transaction in sorted order, according to item supports. Assuming that each transaction fits in main memory\(^1\) it is simple to determine which pairs satisfy the inequality \( f(c(T_i[i]), c(T_i[j])) \tau > r \). When a pair satisfies the inequality, it is output, together with its weight \( \max\{1, f(c(T_i[i]), c(T_i[j])) \tau \} \). This operation has a cost of \( O(N/B) \) I/Os for the reads.

The most expensive steps are the sorting steps, implying that the following theorem holds:

**Theorem 4** Suppose we are given transactions \( T_1, \ldots, T_m \), each a subset of \([n]\), with \( N = mb \) items in total, and \( f \) is the function corresponding to the similarity measure \( s \). Also let \( |S_i \cap S_j| f(\{|S_i|, |S_j|\}) = s(i, j) \). The expected complexity of \( IOBiSam(T_1, \ldots, T_m; f, \tau) \) in the I/O model is

\[
O \left( \frac{N}{B} \log \frac{m}{B} \frac{N}{M} \right) \text{I/Os}.
\]

**3 How to use the BiSAM output**

Summing up the weight \( M(i, j) \) of a given pair \((i, j)\) in the output of BiSAM gives us, by Lemma 1, one of the following:

- Exactly \( s(i, j) \tau \) with probability 1, or
- The value of a random variable with binomial distribution and expectation \( s(i, j) \tau \).

In the former case, we obviously know the similarity of \( i \) and \( j \). In the latter case we can use statistical methods to derive bounds on likely and unlikely values of \( s(i, j) \). In our theoretical discussion, we will make use of the fact that the binomial distribution is closely approximated by the Poisson distribution (with the same mean) whenever the sampling probability is much smaller than the expectation. However, we stress that in concrete cases it is possible to do confidence calculations directly on the binomial distribution to get more accurate results.

Figure 2 shows confidence bounds for various observed values of a Poisson distributed random variable. We know that the mean value of \( M(i, j) \) is \( \tau s(i, j) \), so \( M(i, j) / \tau \) is an unbiased estimator for \( s(i, j) \). How likely is it that \( s(i, j) < \alpha M(i, j) / \tau \) for some \( \alpha < 1 \)? This depends on \( \alpha \) and \( M(i, j) \)—Fig. 2 considers the cases where \( M(i, j) \in \{1, 2, 4, 8, 16, 32\} \). For each value of \( M(i, j) \), the graph plotted is the probability of not observing a value as large as \( M(i, j) \) given that \( s(i, j) = \alpha M(i, j) / \tau \). This is the “confidence” we have in the assertion that \( s(i, j) > \alpha M(i, j) / \tau \). Larger values of \( M(i, j) \) yield higher confidences. Taking \( M(i, j) = 8 \) as an example we see that with 90% confidence the estimate

\(^1\) The assumption is made only for simplicity of exposition, since the result holds also without this assumption.
M(i, j)/τ is at most s(i, j)/0.59 ≈ 1.7s(i, j), and with 90% confidence M(i, j)/τ is at least s(i, j)/1.65 ≈ 0.6s(i, j).

3.1 Errors with respect to a reporting threshold

One case we will consider in particular is when there is a threshold ∆ such that we are interested in reporting all pairs with similarity ∆ or more. To report such pairs with reasonable probability we cannot simply choose the pairs with weight ∆τ or more, since this would give too many false negatives, i.e., pairs with s(i, j) ≥ ∆ that are not reported. The false negative probability can be decreased by lowering the weight threshold. In the following, we assume that pairs with weight ∆τ/2 or more are reported.

**Analysis of false negative probability** We first bound the probability that a pair {i, j} with s(i, j) ≥ ∆ is not reported by the algorithm. This happens if M(i, j) ≤ ∆τ/2 and M(i, j)f(c(i), c(j), ∆) < 1. If f(c(i), c(j))τ ≥ 1 then the pair {i, j} is reported with probability 1. Otherwise, since M(i, j) has binomial distribution, it follows from Chernoff bounds (see e.g., [32, Theorem 4.2] with δ = 1/2) that the probability of the former event is at most exp(−δ²μ/2) = exp(−μ/8). Solving for μ this means that we have error probability at most ε if μ ≥ 8 ln(1/ε). This bound is pessimistic, especially when ε is not very small. Tighter bounds can be obtained using the Poisson approximation to the binomial distribution, which is known to be precise when the number of trials is not too small (e.g., at least 100). Table 5 shows some values of μ and corresponding false negative probabilities, using the Poisson approximation.

**False positives** The probability that a pair {i, j} with s(i, j) < ∆ is reported depends on how far the mean s(i, j)τ is from ∆τ. If the ratio s(i, j)/∆ is close to 1, there is a high probability that the pair will be reported. However, this is not so bad since s(i, j) is close to the threshold ∆. On the other hand, when s(i, j)/∆ is close to zero we would like the probability that {i, j} is reported to be small. Again, we may use the fact that either f(c(i), c(j))τ ≥ 1 (in which case the pair is exactly counted and reported if and only if s(i, j) ≥ ∆/2). For s(i, j) < ∆/2 we can use Chernoff bounds, or the Poisson approximation, to bound the probability that M(i, j) > ∆τ/2. Figure 3 illustrates two Poisson distributions (one corresponding to an item pair with measure three times below the threshold, and one corresponding to an item pair with measure at the threshold).
Table 5 Values of $\tau \Delta$ and corresponding error probabilities $\epsilon$

| $\tau \Delta$ | $\epsilon$ | $\epsilon'$ |
|--------------|------------|------------|
| 3            | 0.199      | 0.0498     |
| 5            | 0.125      | 0.00674    |
| 10           | 0.0671     | 0.0000454  |
| 15           | 0.0180     | $< 10^{-6}$|
| 20           | 0.0108     | $< 10^{-8}$|
| 30           | 0.00195    | $< 10^{-13}$|

The error probabilities $\epsilon'$ are for the variant of the algorithm where we return the whole multiset $M$, and use a different method to filter false positives (see Sect. 3.2)

![Poisson distributions with mean 5 and 15](image)

Fig. 3 Illustration of false negatives and false positives for $\tau \Delta = 15$. The leftmost peak shows the probability distribution for the number of samples of a pair $\{i, j\}$ with $s(i, j) = \Delta/3$. With a probability of around 13% the number of samples is above the threshold (vertical line), which leads to the pair being reported (false positive). The rightmost peak shows the probability distribution for the number of samples of a pair $\{i, j\}$ with $s(i, j) = \Delta$. The probability that this is below the threshold, and hence not reported (false negative), is around 1.8%

3.2 Filtering of BiSAM output

The BiSAM algorithm generates a stream of weighted item pairs that may be very large. In order to obtain a more succinct output, we propose a filtering phase that eliminates pairs that are not similar enough. This task can be carried out in at least three ways:

**Exact threshold filtering:** a weight threshold $w$ can be set, depending on the similarity one is interested in, and can be used in order to filter out those pairs whose sum of weights is below the threshold. As discussed in the previous section, this gives an output where false positive and negative probabilities can be rigorously analyzed. This method requires that the filter stores a set of weighted samples $M$, e.g., using a hash table, keeping track of the current sum for each pair seen. In the I/O model, the best implementation is via sorting of the output produced by IOBiSAM. In the standard model where space is a bigger issue, the next methods may offer better guarantees at the cost of a more complex implementation;

**Checking similarities:** the weight threshold $w$ implies that we filter away those pairs whose similarity is far below $w/\tau$. An alternative is to spend more time on the pairs output by BiSAM, using a sampling method to obtain a more accurate estimate of $|S_i \cap S_j|$. A suitable technique could be to use min-wise independent hash functions $[10, 25]$ to obtain a sketch of each set $S_i$. It suffices to compare two sketches in order to have an approximation of the Jaccard similarities of $S_i$ and $S_j$, which in turn gives an approximation of $|S_i \cap S_j|$. Based on this we may decide if a pair is likely to be interesting, or if it is possible
to filter it out. The sketches could be built and maintained during the ItemCount procedure using, say, a logarithmic number of hash functions. Indyk [25] presents an efficient class of (almost) min-wise independent hash functions.

For some similarity measures such as lift and overlap coefficient the similarity of two sets may be high even if the sets have very different sizes. In such cases, it may be better to sample the smaller set, say, $S_i$, and use a hash table containing the larger set $S_j$ to estimate the fraction $|S_i \cap S_j|/|S_i|$. However, this requires that the whole data set fits in memory.

**Most frequent pairs in a stream**: this technique consists in the use of a streaming algorithm for finding the pairs whose sum of associated weights exceeds a given user-defined threshold. Many algorithms exist that address this problem, see [17] for a comprehensive treatment and an experimental comparison, but only some of them are able to manage weighted items. One algorithm suiting the needs of BiSam well is SPACEAVING [30]. See [31] for a more detailed description of the algorithm. In the following, we will describe a modification of SPACEAVING that takes into account the weights of pairs without adding any cost to the computational time.

### 3.2.1 Weighted SPACEAVING

Here, we describe a modification of SPACEAVING that supports weights in the stream $M$. Such a modification has already been presented in [18], but our approach is different in the sense that using some slackness in the space allowed, we get constant time updates for the underlying data structure. We will refer to our algorithm with the name WSPACEAVING. In the following, we consider the elements of $M$ as pairs, but the algorithm works for generic elements.

As pointed out before, we are interested in reporting only those pairs in $M$ whose sum of weights exceeds a certain threshold. Let $N := \sum_{(i,j) \in M} M(i,j)$; given a user-defined threshold $\phi$ we want to report those pairs $p = (i, j)$ whose sum of associated weights $M(i, j) = M(p)$ is larger than $\phi N$, for some $\phi > 0$. We discuss the choice of $\phi$ below.

In the following, we will call the sum of associated weights $M(i, j)$ of a pair $(i, j)$, the pair’s weight, and the threshold $\phi$ the cut weight. Moreover, we will denote the associated weight of a pair $(i, j) = p$ with $\omega$.

In order to have the desired result, we maintain a collection of entries, each of which contains a pair, plus an estimate of the weight. The estimate is denoted $count_v$. Moreover, we keep track of the minimum $count$ among all the recorded entries, and refer to this value with the name min.

The size $l$ of the collection has to be chosen according to the precision of the desired result, since the algorithm can output pairs whose weight is larger than $(\phi - 2/l)N$, and guarantees to output each pair having weight larger than $\phi N$.

The algorithm works in a pretty simple way: when a new pair $(i, j)$ arrives, we look for it in the collection; if it is already recorded in some position $v$, we update $count_v$ adding the associated weight $\omega$ (which is $\max\{1, f(|S_i|, |S_j|)\tau\}$ in the case of BiSam) to it and we move to the next pair. If the pair is not in the collection we put the pair in the data structure, replacing a pair among the ones having small estimated sum of associated weights. Suppose this pair appears at position $v$—then we put in that position the pair $(i, j)$, and assign $count_v = count_v + \omega$. Figure 4 reports the pseudocode for the updating procedure.

**Choosing the pair to replace** We describe how to implement WSPACEAVING in a way that only a constant number of operations are needed in case of an update to the data structure.
procedure WSPACESAVING(M, φ)
    While there is a new pair (i, j) from the stream
        if (i, j) is in the collection D at position v
            count_v = count_v + ω;
        end
    else
        Choose a pair from the bucket of low weight pairs; let v be its position;
        D_v = (i, j);
        count_v = count_v + ω;
    end
    ∀v, v ∈ {1, ..., l} : count_v ≥ φN output D_v;
end

Fig. 4 Pseudocode for the WSPACESAVING algorithm. We remark that, in the case of BiSam, ω = max{1, τf(|S_i|, |S_j|)}

We maintain the pairs along with their estimated sum of associated weights in buckets, each of which contains pairs with count in a certain range. The size of ranges is increasing by a factor of 2, so we only have to keep track of a logarithmic, with respect to N, number of buckets. In particular, we will have the ranges: [1, 2), [2, 4), [4, 8), ..., [2^k−1, 2^k), [2^k, φN), [φN, N]. In this way, when we need to find a pair whose count = min, we can go directly in the non-empty bucket with the lowest weight range, and pick up an arbitrary pair contained in it. Moreover, when it is necessary to move a pair in a new bucket, it is sufficient to move it in the bucket representing the next range, eventually initializing the bucket. This operations can easily be done in constant time per update. Once a bucket is empty it will never receive a new pair again, so we can directly switch to the next one.

For what concerns the correctness of the algorithm, we will first describe some properties.

Lemma 5 At any point in time \( \sum_{v \in \{1, ..., l\}} \text{count}_v = N \)

Proof The lemma can be proved via induction on the length of the stream. The main idea is that at each step, only one counter is incremented with the weight of the new arrived pair. □

Lemma 6 Among all counters, the minimum counter value, min, is no greater than N/l.

Proof We can write:

\[
\text{min} = l^{-1} \left( N - \sum_{v \in \{1, ..., l\}} (\text{count}_v - \text{min}) \right);
\]

since \( \forall v. \text{count}_v \geq \text{min} \) the summation has all nonnegative terms, thus the result. □

Theorem 7 \( \forall (i, j). (i, j) \in M \land M(i, j) > \phi N \Rightarrow (i, j) \) is recorded in the data structure.

Proof Assume (i, j) do not end up in the data structure; notice that \( M(i, j) > \text{min} \) at any point in time. Since (i, j) is not in the data structure there has to be a pair that caused the deletion of (i, j) one last time. Since (i, j) has been selected to be deleted, all the pairs in the data structure have to have an estimated frequency larger than \( \phi N \), so \( \text{min} > \phi N \); by means of Lemma 6, we also have \( \text{min} \leq N/l \), so \( \text{min} > \phi N \geq N/l \geq \text{min} \) which is absurd. □
Theorem 7 states that all pairs \((i, j)\) having frequency \(M(i, j) > \phi N\) are reported by the algorithm.

It remains to understand the entity of the error the algorithm introduces. The error depends on the maximum overestimation the algorithm allows. From Lemma 6, we know that \(\min \leq N/l\); the pair having \(count_i = \min\) pertains to a bucket whose range is \([a, b)\). Since \(b = 2(a - 1) < 2\min \leq 2N/l\) and since we can have overestimated the frequency of a pair using at most \(b\), we get an additive \(2/l\) approximation, hence it is possible that pairs, whose frequency falls in \([(\phi - 2/l)N, \phi N]\), are reported.

From the previous theorem, we get the corollary:

**Corollary 8** The space usage of BiSAM, when WSPACESAVING is used, is \(O(N/(\tau \cdot \Delta))\) and the computational time remains \(O(mb + \tau \sum_{0<i<j\leq n}s(i, j))\).

**Proof** For the time complexity, we have already pointed out that every update to the data structure takes constant time. The space bound is straightforward when we notice that, given the cut weight \(\phi\), we can have at most \(1/\phi\) frequent pairs, so we need at least the space for storing those pairs. For \(\phi = (\tau \cdot \Delta)/(2 \cdot N)\), the claim is verified. \(\square\)

### 4 Experiments

To make experiments fully reproducible and independent of implementation details and machine architecture, we focus our attention on the number of hash table operations, and the number of items in the hash tables. That is, the time for BiSAM is the number of items in the input set plus the number of pairs output. The space of BiSAM is the number of distinct items (for support counts) plus the space for the filtering algorithm. An exact threshold filter has space usage that is equal to the number of distinct pairs output by BiSAM, whereas the most frequent pairs algorithm has space usage that is equal to the output weight of BiSAM divided by the weight threshold (see Corollary 8). Similarly, the time for methods based on exact counting is the number of items in the input set plus the number of pairs in all transactions (since every pair is counted), and the space for exact counting is the number of distinct items plus the number of distinct pairs that occur in some transaction.

We believe that these simplified measures of time and space are a good choice for two reasons. First, hash table lookups and updates require hundreds of clock cycles unless the relevant key is in cache. This means that a large fraction of the time spent by a well-tuned implementation is used for hash table lookups and updates. Second, we are comparing two approaches that have a similar behavior in that they count supports of items and pairs. The key difference thus lies in the number of hash table operations, and the space used for hash tables. Also, this means that essentially any speedup or space reduction applicable to one approach is applicable to the other (e.g., using counting Bloom filters to reduce space usage).

**Data sets.** Experiments have been run on both real datasets and artificial ones. We have used most of the datasets of the Frequent Itemset Mining Implementations (FIMI) Repository.\(^2\) In addition, we have created three data sets based on the internet movie database (IMDB). Table 6 contains some key figures on the data sets.

\(^2\) http://fimi.cs.helsinki.fi/.
Table 6  Key figures on the data sets used for experiments

| Dataset      | Distinct items | Number of trans. | Avg. trans. size | Max. trans. size | Avg. items support | Avg. similarity |
|--------------|----------------|------------------|------------------|------------------|--------------------|-----------------|
| Chess        | 75             | 3,196            | 37               | 37               | 1,577              | 0.3148          |
| Connect      | 129            | 67,557           | 43               | 43               | 22,519             | 0.1626          |
| Mushroom     | 119            | 8,124            | 23               | 23               | 1,570              | 0.1523          |
| Pumsb        | 2113           | 49,046           | 74               | 74               | 1,718              | 0.0120          |
| Pumsb_star   | 2,088          | 49,046           | 50               | 63               | 1,186              | 0.0102          |
| Kosarak      | 41,270         | 990,002          | 8                | 2,498            | 194                | 0.0168          |
| BMS-WebView-1| 497            | 59,601           | 2                | 161              | 301                | 0.0307          |
| BMS-WebView-2| 3,340          | 59,601           | 2                | 161              | 107                | 0.0140          |
| BMS-POS      | 1,657          | 515,596          | 6                | 164              | 2,032              | 0.0044          |
| Retail       | 16,470         | 88,162           | 10               | 76               | 55                 | 0.0094          |
| Accidents    | 468            | 340,183          | 33               | 51               | 24,575             | 0.0248          |
| T10I4D100K   | 870            | 100,000          | 10               | 29               | 1,161              | 0.0137          |
| T40I10D100K  | 942            | 100,000          | 40               | 77               | 4,204              | 0.0230          |
| actors       | 128,203        | 51,226           | 31               | 1,002            | 12                 | 0.0618          |
| directorsActor| 50,645        | 3,783            | 1,221            | 8,887            | 90                 | 0.0978          |
| movieActors  | 51,226         | 133,633          | 12               | 2,253            | 33                 | 0.0380          |

The first 13 data sets are from the FIMI repository. The last 3 were extracted from the May 29, 2009 snapshot of the Internet Movie Database (IMDB). The datasets Chess, Connect, Mushroom, Pumsb, and Pumsb_star were prepared by Roberto Bayardo from the UCI datasets and PUMBS. Kosarak contains (anonymized) click-stream data of a hungarian online news portal, provided by Ferenc Bodon. BMS-WebView-1, BMS-WebView-2, and BMS-POS contain clickstream and purchase data of a legwear and legcare web retailer, see [27] for details. Retail contains the (anonymized) retail market basket data from a Belgian retail store [7]. Accidents contains (anonymized) traffic accident data [21]. The datasets T10I4D100K and T10I4D100K have been generated using an IBM generator from the Almaden Quest research group. Actors contains the set of rated movies for each male actor who has acted in at least 10 rated movies. DirectorActor contains, for each director who has directed at least 10 rated movies, the set of actors from Actors that this director has worked with in rated movies. MovieActor is the inverse relation of Actors, listing for each movie a set of actors.

4.1 Results and discussion

Tables 8, 9, 10 show the results of our experiments for the all_confidence measure. The time and space for BiSAM is a random variable. The reported number is an exact computation of the expectation of this random variable. Separate experiments have confirmed that observed time and space is relatively well concentrated around this value. The values of \( \tau \) used are shown in Table 7—they were chosen manually in each case to give a “human readable” output of around 1000 pairs. (For the IMDB data set Actor and the Kosarak data set this was not possible; for the latter this behavior was due to a large number of false positives.) Note that choosing a smaller \( \Delta \) would bring the performance of BiSAM closer to the exact algorithms; this is not surprising, since lowering \( \Delta \) means reporting pairs having a small values for the similarity measure, increasing in this way the number of samples taken. As noted before, we are usually interested in reporting pairs with high similarity, for almost any reasonable scenario.
Table 7  Chosen values of parameter $\tau$ and the corresponding output sizes

| Parameters and output size |         |         |         |
|---------------------------|---------|---------|---------|
| Dataset                   | $\tau$  | #output |
| Chess                     | 20      | 986     |
| Connect                   | 20      | 1,008   |
| Mushroom                  | 40      | 1,048   |
| Pumsb                     | 9       | 844     |
| Pumsb_star                | 14      | 1,012   |
| Kosarak                   | 6       | 1,710   |
| BMS-WebView-1             | 30      | 992     |
| BMS-WebView-2             | 21      | 1,002   |
| BMS-POS                   | 85      | 994     |
| Retail                    | 20      | 1,047   |
| Accidents                 | 30      | 1,030   |
| T10I4D100K                | 40      | 947     |
| T40I10D100K               | 30      | 1,087   |
| Actors                    | 8       | 200,445 |
| DirectorsActor            | 3       | 714     |
| MovieActors               | 13      | 1,213   |

The results for the other measures are omitted for space reasons, since they are very similar to the ones reported here. This is because the complexity of BiSAM is, in most cases, dominated by the first phase (counting item frequencies), meaning that fluctuations in the cost of the second phase have little effect. (Table 8)

Table 8  Experimental results for all_confidence measure concerning time

| Time | Dataset     | BiSAM    | Exact counting | Ratio |
|------|-------------|----------|----------------|-------|
|      | Chess       | $1.35 \times 10^5$ | $22.5 \times 10^5$ | 16.67 |
|      | Connect     | $29.3 \times 10^5$ | $639 \times 10^5$  | 21.82 |
|      | Mushroom    | $2.08 \times 10^5$ | $22.4 \times 10^5$ | 10.77 |
|      | Pumsb       | $36.8 \times 10^5$ | $1360 \times 10^5$| 36.96 |
|      | Pumsb_star  | $25.4 \times 10^5$ | $638 \times 10^5$  | 25.12 |
|      | Kosarak     | $108 \times 10^5$ | $3130 \times 10^5$| 28.98 |
|      | BMS-WebView-1| $2.06 \times 10^5$ | $9.64 \times 10^5$ | 4.68  |
|      | BMS-WebView-2| $5.66 \times 10^5$ | $24.4 \times 10^5$ | 4.31  |
|      | BMS-POS     | $35.1 \times 10^5$ | $246 \times 10^5$  | 7.01  |
|      | Retail      | $15.3 \times 10^5$ | $80.7 \times 10^5$ | 5.27  |
|      | Accidents   | $115 \times 10^5$ | $187 \times 10^5$  | 1.62  |
|      | T10I4D100K  | $11 \times 10^5$  | $62.8 \times 10^5$ | 5.72  |
|      | T40I10D100K | $42.6 \times 10^5$ | $841 \times 10^5$  | 19.74 |
|      | Actors      | $144 \times 10^5$ | $500 \times 10^5$  | 3.47  |
|      | DirectorsActor| $4,688 \times 10^5$| $81,500 \times 10^5$| 17.38 |
|      | MovieActors | $290 \times 10^5$ | $1,070 \times 10^5$| 3.69  |

The columns ratio represents the savings BiSAM gets with respect to an exact approach computing all pairs.
Table 9 Result of experiments for the all_confidence measure concerning space when the Exact Threshold Filtering is used

| Dataset        | BiSAM | Exact counting | Ratio |
|----------------|-------|----------------|-------|
| Chess          | $2.20 \times 10^3$ | $2.66 \times 10^3$ | 1.21  |
| Connect        | $4.14 \times 10^3$ | $6.96 \times 10^3$ | 1.68  |
| Mushroom       | $2.92 \times 10^3$ | $3.65 \times 10^3$ | 1.25  |
| Pumsb          | $36.7 \times 10^3$ | $536 \times 10^3$ | 14.6  |
| Pumsb_star     | $44.5 \times 10^3$ | $485 \times 10^3$ | 10.9  |
| Kosarak        | $2.306 \times 10^3$ | $33.100 \times 10^3$ | 14.35 |
| BMS-WebView-1  | $26.5 \times 10^3$ | $64.5 \times 10^3$ | 2.43  |
| BMS-WebView-2  | $163 \times 10^3$ | $725 \times 10^3$ | 4.45  |
| BMS-POS        | $89.4 \times 10^3$ | $381 \times 10^3$ | 4.26  |
| Retail         | $612 \times 10^3$ | $3600 \times 10^3$ | 5.88  |
| Accidents      | $10.9 \times 10^3$ | $47.3 \times 10^3$ | 4.35  |
| T10I4D100K     | $60.7 \times 10^3$ | $171 \times 10^3$ | 2.82  |
| T40I10D100K    | $168 \times 10^3$ | $433 \times 10^3$ | 2.57  |
| Actors         | $11.925 \times 10^3$ | $32.900 \times 10^3$ | 2.76  |
| DirectorsActor | $76.104 \times 10^3$ | $367,000 \times 10^3$ | 4.82  |
| MovieActors    | $22.317 \times 10^3$ | $55,400 \times 10^3$ | 2.48  |

Table 10 Result of experiments for the all_confidence measure concerning space when the version of WSPACESAVING presented in [18] is used

| Dataset        | BiSAM | Exact counting | Ratio |
|----------------|-------|----------------|-------|
| Chess          | $2.11 \times 10^3$ | $2.66 \times 10^3$ | 1.26  |
| Connect        | $2.90 \times 10^3$ | $6.96 \times 10^3$ | 2.40  |
| Mushroom       | $2.79 \times 10^3$ | $3.65 \times 10^3$ | 1.31  |
| Pumsb          | $9.33 \times 10^3$ | $536 \times 10^3$ | 57.48 |
| Pumsb_star     | $10.61 \times 10^3$ | $485 \times 10^3$ | 45.69 |
| Kosarak        | $387 \times 10^3$ | $33100 \times 10^3$ | 85.63 |
| BMS-WebView-1  | $7.5 \times 10^3$ | $64.5 \times 10^3$ | 8.60  |
| BMS-WebView-2  | $29.27 \times 10^3$ | $725 \times 10^3$ | 24.79 |
| BMS-POS        | $18.96 \times 10^3$ | $381 \times 10^3$ | 20.10 |
| Retail         | $94.33 \times 10^3$ | $3600 \times 10^3$ | 38.20 |
| Accidents      | $4.75 \times 10^3$ | $47.3 \times 10^3$ | 9.95  |
| T10I4D100K     | $12.52 \times 10^3$ | $171 \times 10^3$ | 13.65 |
| T40I10D100K    | $38.22 \times 10^3$ | $433 \times 10^3$ | 11.32 |
| Actors         | $1.731 \times 10^3$ | $32.900 \times 10^3$ | 19.00 |
| DirectorsActor | $58.974 \times 10^3$ | $367,000 \times 10^3$ | 6.32  |
| MovieActors    | $3.461 \times 10^3$ | $55,400 \times 10^3$ | 16.00 |

We see that the speedup obtained in the experiments varies between a factor 1.62 and a factor over 36. The largest speedups tend to come for data sets with the largest average transaction size, or data sets where some transactions are very large (e.g., Kosarak). However, as
our theoretical analysis suggests, large transaction size alone is not sufficient to ensure a large speedup—items also need to have support that is not too small. So, while the DirectorActor data set has very large average transaction size, the speedup is not as high as the ones observed for other datasets, because the support of items is low. In a nutshell, BiSAM gives the largest speedups when there is a combination of relatively large transactions and relatively high average support. The space results are shown in Tables 9 and 10. In particular, Table 9 refers to the algorithm when the Exact threshold filtering is applied and the space usage ranges from being quite close to the space usage for exact counting, to a decent reduction. Table 10 refers to the algorithm when WSPACESAVING is used. In particular, in this case, we are taking into account the version of the algorithm presented in [18], where the space usage would be $\frac{N}{(\tau \cdot \Delta)}$ at the cost of raising the time complexity to $O(mb + \tau \sum_{1 \leq i < j \leq n} s(i, j))$. As can be seen we may get much higher savings in space in this case, up to almost two orders of magnitude for some data sets. Especially, we get large savings for some data sets with many distinct items.

Though we have not experimented with methods based on locality-sensitive hashing (LSH), we observe that our method appears to have an advantage when the number $n$ of distinct items is large. This is because LSH in general (and in particular for cosine similarity) requires comparison of $\frac{n^2}{2}$ pairs of hash signatures. On the other hand, our algorithm uses time that is $\propto n^2$ times the average similarity (times a constant $\tau$ that is typically small, since we are looking for high-similarity pairs). Table 6 shows the average all_confidence similarity of each of our data sets, which is typically 1–2 orders of magnitude smaller than the similarity of the pairs we wish to report.

For the data sets Kosarak, Retail, BMS-Webview-2, Actors, and Movie-Actors, the ratio between the number of signature comparisons and the number of hash table operations required for BiSAM is in the range 9–1340. While these numbers are not necessarily directly comparable, it does indicate that BiSAM has the potential to improve LSH-based methods that require comparison of all signature pairs.

Acknowledgments We wish to thank Blue Martini Software for contributing the KDD Cup 2000 data. Also, we thank the reviewers of the ICDM submission for pointing out several related works.

References

1. Aggarwal A, Vitter JS (1988) The input/output complexity of sorting and related problems. Commun. ACM 31(9):1116–1127
2. Aggarwal CC, Yu PS (1998) A new framework for itemset generation. In: Proceedings of the ACM SIGACT–SIGMOD–SIGART symposium on principles of database systems (PODS '98). ACM Press, New York, pp 18–24
3. Agrawal R, Mehta M, Shafer JC, Srikant R, Arning A, Bollinger T (1996) The quest data mining system. In: Proceedings of the 2nd international conference of knowledge discovery and data mining (KDD '96). AAAI Press, CA, pp 244–249
4. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: International conference on very large data bases (VLDB '94). Morgan Kaufmann Publishers, Inc., CA, pp 487–499
5. Amossen RR, Pagh R (2009) Faster join-projects and sparse matrix multiplications. In: Proceedings of database theory—12th international conference (ICDT '09), vol 361 of ACM international conference proceeding series. ACM, New York, pp 121–126
6. Arasu A, Ganti V, Kaushik R (2006) Efficient exact set-similarity joins. In: Proceedings of the 32nd international conference on very large data bases (VLDB '06). ACM, New York, pp 918–929
Finding associations and computing similarity via biased pair sampling

7. Brijs T, Swinnen G, Vanhoof K, Wets G (1999) Using association rules for product assortment decisions: a case study. In: Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining (KDD ’99). ACM Press, New York, pp 254–260

8. Brin S, Motwani R, Silverstein C (1997) Beyond market baskets: generalizing association rules to correlations. SIGMOD Rec ACM Special Interest Group Manag Data 26(2):265–276

9. Brin S, Motwani R, Ullman JD, Tsur S (1997) Dynamic itemset counting and implication rules for market basket data. In: Proceedings of the ACM-SIGMOD international conference on management of data (SIGMOD ’97), vol. 26(2) of SIGMOD record (ACM special interest group on management of data). ACM Press, New York, pp 255–264

10. Broder AZ, Charikar M, Frieze AM, Mitzenmacher M (2000) Min-wise independent permutations. J Comput Syst Sci 60(3):630–659

11. Campagna A, Pagh R (2010) On finding similar items in a stream of transactions. In: Proceedings of the 10th IEEE international conference on data mining workshops (ICDMW 2010). IEEE Computer Society, Silver Spring, pp 121–128

12. Charikar MS (2002) Similarity estimation techniques from rounding algorithms. In: Proceedings of the thirty-fourth annual ACM symposium on theory of computing (STOC ’02). ACM, New York, pp 380–388

13. Chaudhuri S, Ganti V, Kaushik R (2006) A primitive operator for similarity joins in data cleaning. In: Proceedings of the 22nd international conference on data engineering (ICDE 2006). IEEE Computer Society, Silver Spring, p 5

14. Cohen E, Datar M, Fujiwara S, Gionis A, Indyk P, Motwani R, Ullman JD, Yang C (2001) Finding interesting associations without support pruning. IEEE Trans Knowl Data Eng 13(1):64–78

15. Cohen E, Lewis DD (1999) Approximating matrix multiplication for pattern recognition tasks. J Algorithms 30(2):211–252

16. Cormode G, Winograd S (1990) Matrix multiplication via arithmetic progressions. J Symb Comput 9(3):251–280

17. Cormode G, Hadjieletheriou M (2008) Finding frequent items in data streams. PVLDB 1(2):1530–1541

18. Cormode G, Korn F, Tirthapura S (2008) Exponentially decayed aggregates on data streams. In: Proceedings of the 22nd international conference on data engineering (ICDE 2008). IEEE, New York, pp 1379–1381

19. Cormode G, Muthukrishnan S (2005) What’s hot and what’s not: tracking most frequent items dynamically. ACM Trans Stream 30(1):249–278

20. Demaine ED, López-Ortiz A, Munro JI (2002) Frequency estimation of internet packet streams with limited space. In: Proceedings of the 10th annual European symposium algorithms (ESA ’02), pp 348–360

21. Geurts K, Wets G, Brijs T, Vanhoof K (2003) Profiling high frequency accident locations using association rules. In: Proceedings of the 82nd annual transportation research board, p 18

22. Goethals B, Zaki MJ (2004) Advances in frequent itemset mining implementations: report of fimi’03’. ACM SIGKDD Explor 6(1):109–117

23. Goethals B, Zaki MJ (eds) (2003) Proceedings of the ICDM 2003 workshop on frequent itemset mining implementations (FIMI ’03), Vol 90 of CEUR workshop proceedings. CEUR-WS.org

24. Han J, Kamber M (2006) Data mining: concepts and techniques, 2nd edn. Morgan Kaufmann, CA

25. Indyk P (1999) A small approximately min-wise independent family of hash functions. In: Proceedings of the 10th annual ACM-SIAM symposium on discrete algorithms (SODA’99), pp 454–456

26. Indyk P, Motwani R, Raghavan P, Vempala S (1997) Locality-preserving hashing in multidimensional spaces. In: Proceedings of the twenty-ninth annual ACM symposium on theory of computing (STOC ’97), pp 618–625

27. Kohavi R, Brodley C, Frasca B, Mason L, Zheng Z (2000) KDD-Cup 2000 organizers’ report: peeling the onion. SIGKDD Explor 2(2):86–98

28. Lee Y-K, Kim W-Y, Cai YD, Han J (2003) Comine: Efficient mining of correlated patterns. In: Proceedings of the IEEE international conference on data mining (ICDM ’03). IEEE Computer Society, Silver Spring, pp 581–584

29. Bayardo RJ Jr, Goethals B, Zaki MJ (eds) (2004) Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations (FIMI ’04), Vol 126 of CEUR workshop proceedings. CEUR-WS.org

30. Metwally A, Agrawal D, Abbadi AE (2005a) Efficient computation of frequent and top-k elements in data streams. In: Proceedings of database theory—10th international conference (ICDT 2005), Vol 3363 of lecture notes in computer science. Springer, Berlin, pp 398–412

31. Metwally A, Agrawal D, Abbadi AE (2005b) Efficient computation of frequent and top-k elements in data streams. Technical Report 23, University of California, Santa Barbara, USA

32. Motwani R, Raghavan P (1995) Randomized algorithms. Cambridge University Press, Cambridge
33. Omiecinski E (2003) Alternative interest measures for mining associations in databases. IEEE Trans Knowl Data Eng 15(1):57–69
34. Park JS, Chen M-S, Yu PS (1995) An effective hash-based algorithm for mining association rules. SIGMOD Rec ACM Special Interest Group Manag Data 24(2):175–186
35. Savasere A, Omiecinski E, Navathe SB (1995) An efficient algorithm for mining association rules in large databases. In: Proceedings of the 21st international conference on very large data bases (VLDB ’95). Morgan Kaufmann Publishers, CA, pp 432–444
36. Toivonen H (1996) Sampling large databases for association rules. In: Proceedings of the 22nd international conference on very large data bases (VLDB ’96). Morgan Kaufmann Publishers, pp 134–145
37. Wu X, Zhang C, Zhang S (2004) Efficient mining of both positive and negative association rules. ACM Trans Inf Syst 22:381–405
38. Xiao C, Wang W, Lin X, Shang H (2009) Top-k set similarity joins. In: Proceedings of the 25th international conference on data engineering. (ICDE ’09). IEEE, London, pp 916–927
39. Xiao C, Wang W, Lin X, Yu JX (2008) Efficient similarity joins for near duplicate detection. In: Proceedings of the 17th international conference on world wide web. (WWW ’08). ACM, New York, pp 131–140
40. Yuster R, Zwick U (2005) Fast sparse matrix multiplication. ACM Trans Algorithms 1(1):2–13
41. Zhang S, Wu X, Zhang C, Lu J (2008) Computing the minimum-support for mining frequent patterns. Knowl Inf Syst 15(2):233–257

Author Biographies

Andrea Campagna received a M.Sc. degree in computer engineering from the University of Rome “La Sapienza”, Italy, in 2008. From 2008 he has been a PhD student at the IT University of Copenhagen, under the supervision of associate professor Rasmus Pagh. His research interests are in theoretical foundations of computer science, with particular emphasis on algorithms, computational complexity and discrete mathematics.

Rasmus Pagh received a PhD from the computer science department at Aarhus University, Denmark, in 2002. He was assistant professor at the IT University of Copenhagen 2002–2006, and has been employed as associate professor since 2006. From 2007 he has been head of the Efficient Computation Group. His research interests are in theoretical foundations of data storage and processing, with particular emphasis on applications in indexing, database query processing, and data mining. He has been organizer or program committee member of numerous international conferences, and currently serves in the steering committee of the European Symposium on Algorithms.