Finding Highly Correlated Pairs Efficiently with Powerful Pruning

Jian Zhang*
Computer Science Department
Stanford University
Stanford, CA 94305
jz@cs.stanford.edu

Joan Feigenbaum†
Computer Science Department
Yale University
New Haven, CT 06520
jf@cs.yale.edu

ABSTRACT

We consider the problem of finding highly correlated pairs in a large data set. That is, given a threshold not too small, we wish to report all the pairs of items (or binary attributes) whose (Pearson) correlation coefficients are greater than the threshold. Correlation analysis is an important step in many statistical and knowledge-discovery tasks. Normally, the number of highly correlated pairs is quite small compared to the total number of pairs. Identifying highly correlated pairs in a naive way by computing the correlation coefficients for all the pairs is wasteful. With massive data sets, where the total number of pairs may exceed the main-memory capacity, the computational cost of the naive method is prohibitive. In their KDD'04 paper [15], Hui Xiong *et al.* address this problem by proposing the TAPER algorithm. The algorithm goes through the data set in two passes. It uses the first pass to generate a set of candidate pairs whose correlation coefficients are then computed directly in the second pass. The efficiency of the algorithm depends greatly on the selectivity (pruning power) of its candidate-generating stage.

In this work, we adopt the general framework of the TAPER algorithm but propose a different candidate-generation method. For a pair of items, TAPER's candidate-generation method considers only the frequencies (supports) of individual items. Our method also considers the frequency (support) of the pair but does not explicitly count this frequency (support). We give a simple randomized algorithm whose false-negative probability is negligible. The space and time complexities of generating the candidate set in our algorithm are asymptotically the same as TAPER's. We conduct experiments on synthesized and real data. The results show that our algorithm produces a greatly reduced candidate set—one that can be several orders of magnitude smaller than that generated by TAPER. Because of this, our algorithm uses much less memory and can be faster. The former is critical for dealing with massive data.

Categories and Subject Descriptors

H.2.8 [**DATABASE MANAGEMENT**]: Database Applications—*Data mining*

General Terms

Algorithms

Keywords

correlation, statistical computing, massive data set

1. INTRODUCTION

Finding highly correlated pairs in a large set of items is an important basic operation in many knowledge-discovery tasks. For example, in building Bayesian networks from massive data sets, many algorithms [12, 8] consider the strongly dependent (correlated) variables first to speed up the search process. In this paper, we examine correlations in market-basket-like data. Here "market basket" is an abstraction that can be used to model many data sets involving two concepts: a set of items and a set of relations (baskets). In statistical linguistics, one may view words as items and sentences as relations (baskets) to establish the connections among words. In time-series analysis, one may view events as items and certain time windows as baskets to discover the dependencies among events. Correlation analysis in market-basket data is important in knowledge discovery.

Correlation has been well studied in statistics, but statisticians focus mainly on measures of correlations. Efficient computation is often of little concern. Computation does not pose a problem when handling normal amounts of data. However, with massive data sets, efficiency problems arise. Consider a common query in correlation analysis: Given a threshold, find all the pairs whose correlation coefficients are above the threshold. The query can be answered in a naive way: Compute the correlation coefficients of all the pairs, and pick out the highly correlated ones. With massive data sets, the number of items may be very large, and the total number of pairs can be huge. (For a data set with a million items, the total number of pairs is nearly a trillion.)

*Supported by NSF grant 0331640.

†Supported in part by HSARPA grant ARO-1756303, ONR grants N00014-01-1-0795 and N00014-04-1-0725, and NSF grants 0331548, 0428422, and 0534052.

CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.

Computing correlation coefficients for such a huge number of pairs can be prohibitively expensive. Moreover, in many cases, there will not be enough main memory to hold all the pairs. Although one can turn to external-memory computations, the performance deteriorates to an unacceptable level. Hence, in these situations, it is critical for the memory requirement of an algorithm to be much smaller than the size of the input data set. This is possible because it is often the case for real-world data that the number of highly correlated pairs is much smaller than the total number of possible pairs. Storing all the pairs and computing all the correlation coefficients are wasteful.

To address the efficiency and the memory-shortage problem in finding correlated pairs, Hui Xiong *et al.* [15] proposed the TAPER algorithm. The algorithm makes two passes through the data set. In the first pass, it prunes many pairs that are not highly correlated and generates a set of candidate pairs whose correlation coefficients may be above the threshold. The correlation coefficients of these pairs in the candidate set are computed in the second pass. The pairs that are indeed highly correlated are then identified. The size of the candidate set can be much smaller than the total number of pairs. By paying a fairly small price (one more pass through the data), the algorithm saves the computation for the pruned pairs. Furthermore, the algorithm requires less memory than the naive one, making it more suitable for massive data.

Clearly, the efficiency and the memory requirement of the algorithm depend strongly on the effectiveness of the pruning method used in the candidate-generation stage. To decide whether a pair (a, b) should be pruned, the TAPER algorithm uses a rule that considers only the frequencies of individual items a and b (or, from an association-rule-mining point of view, the supports of a and b). Computational simplicity is the advantage of this pruning rule. One needs to count only the supports of the individual items. The disadvantage is that a relatively large group of uncorrelated pairs is missed by the pruning rule. Intuitively, the pruning rule used in TAPER can be viewed as the following: We prune the pair (a, b) if the support of a and the support of b differ a lot. Consider a data set with items' support distribution shown in Fig. 1. (The x axis is the value of the supports, and the y axis is the number of items with that support value.) Consider two areas L and S. Items that fall in L have large supports, and items that fall in S have small supports. TAPER's pruning rule removes pairs that consist of one item from L and the other from S. The pairs that draw both items from S (or L) are selected by TAPER as candidates. However, two items a and b of similar small supports are often not correlated because the item pair (a, b) has extremely small support. There can be many such pairs. In particular, in a large database, most of the items have similar small supports, but many are not correlated. Because TAPER's pruning rule includes these pairs in the candidate set, the candidate set is relatively large. We observe that TAPER's pruning can be further improved by considering the supports of the pairs.

However, there is a basic obstacle to making such improvement. At first glance, it seems the new pruning process requires that we get the supports of all the pairs. On the other hand, the goal of the pruning is exactly to avoid doing so. A pruning process is meaningless if it needs to get the statistics of all the pairs. That is the crux of the challenge we address here: We want to consider the supports of the pairs in a pruning rule without actually counting these supports. In this paper, we show that this can be done. We propose a pruning rule that involves the supports of the pairs. Meanwhile, we give a pruning method for this rule that does not require explicitly counting these supports.

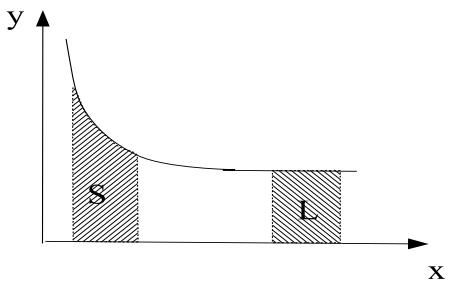

Figure 1: Distribution of Items' Support

Our pruning method is based on min-hash functions [7, 11]. In [11], min-hash functions are used to identify association rules with high confidence. Though related, confidence for association rules and correlation coefficients are two different quantities. In particular, the fact that the correlation coefficient of a pair is above a certain threshold does not mean that the confidence of the corresponding association rule is above that threshold. Thus we cannot use the algorithms in [11] to identify all the highly correlated pairs. Another alternative is to use the min-hash to estimate the supports of the pairs. One can go through all the possible pairs, use the estimation to compute the coefficient correlations for all the pairs, and identify the highly correlated ones. However, this approach still needs to consider all the possible pairs and hence is not efficient. We use the min-hash function in a different way. For two items a and b, we consider the Jaccard distance between the set of baskets containing a and the set of baskets containing b. (The Jaccard distance between two sets is defined to be one minus the ratio of the size of the intersection over the size of the union.) We show a strong connection between this distance and the correlation coefficient of the pair (a, b). In particular, we observe that, if the pair (a, b) has a large correlation coefficient, then its Jaccard distance must be small. We use min-hash to estimate the Jaccard distance and only select pairs within small distance to the candidate set. Note that the Jaccard distance is a relative value of a pair's support. Hence, our pruning process utilizes the information about the support of the pairs and removes unwanted pairs that are missed in TAPER's pruning phase.

Finally, we remark that our pruning process is a randomized algorithm. The pruning process may produce false negatives, i.e., pairs with high correlations might get pruned. However, the false-negative probability can be controlled in our algorithm and can be set arbitrarily small.

We conduct experiments on real and synthesized data. The results show that, by adding information about pairs' supports, our pruning removes more unwanted pairs than TAPER and produces a greatly reduced candidate set—one that can be several orders of magnitude smaller than that generated by TAPER. Because it produces a smaller candidate set, our algorithm is faster. More importantly, as we discussed earlier, with massive data sets that exceed the

memory capacity of computers, an algorithm's efficiency and applicability strongly depend on its memory requirement. We did not use extremely large data sets in the experiment. Instead, we closely examined the memory requirements of the algorithms being compared. The experiments showed that our algorithm has a much smaller (several orders of magnitude smaller) memory requirement than other algorithms. Hence, it is much more usable and efficient in dealing with massive data sets.

1.1 Related Work

"Market basket" is a powerful abstraction used for modeling data. The market-basket abstraction is particularly useful for identifying association (correlation) patterns in the data. Mining association rules [1] is an often encountered problem in many data-mining tasks. There are many algorithms [2, 3, 4, 9] that exploit different constraints for efficient mining of association rules. Two measures are often considered when judging the significance of an association rule: its support and its confidence. Other quantities that measure the interestingness of association rules are studied in [6, 13, 14].

Identifying associations (correlations) according to statistical measures is also an important problem. It has been considered in statistics, but not primarily from an algorithmic point of view. The computations in the statistics literature are often done by brute-force algorithms. Previously, Xiong *et al.* [15] considered the problem of identifying strongly correlated pairs, using Pearson coefficients as the measure of correlation. In this paper, we propose an algorithm that adopts the general framework of the TAPER algorithm in [15] but uses a different method for pruning weakly correlated pairs.

1.2 Overview of the Paper

The rest of this paper is organized as follows: In section 2, we introduce a rule for generating a candidate set and give an efficient candidate-generation method. In section 3, we present the whole algorithm and analyze the time and space complexity of our candidate-generation method. In section 4, we provide experimental results. The results show that our algorithm generates a much smaller candidate set and is faster than TAPER. Section 5 summarizes the paper and discusses some future work.

2. A PRUNING METHOD BASED ON SUPPORTS OF PAIRS

Our pruning process is based on the following observation: For a pair (a, b), if the correlation coefficient of the pair is above a threshold θ, the Jaccard distance between the set of baskets containing a and the set of baskets containing b must be smaller than $f(\theta)$, for a function f that we will specify later. With this observation, to generate candidates is to select pairs of items that are close in the Jaccard space. We use min-hash to estimate the Jaccard distances. A second hash is used to group together the pairs that are close to each other in the Jaccard space. The candidate set then consists of the pairs in such groups. In this way, we use the supports of the pairs in the pruning process without having to count these supports. The process does not need to consider all the possible pairs.

Now we describe the pruning process in detail. First, we need some notation. Data in a market-basket model can be viewed as a table T with entries "0" or "1." The columns of the table represent the items, and the rows represent the baskets. If basket i contains item j, $T(i, j) = 1$; otherwise, $T(i, j) = 0$. We denote by n the number of items and m the number of baskets. We also use $[m]$ to represent the set $\{1, 2, \ldots, m\}$. For an item a, we define $R(a)$ to be the set $\{r \in [m] | T(r, a) = 1\}$. That is, $R(a)$ is the set of rows (baskets) that contain item a. For two items a and b, we denote by $sp(a)$ the support of item a (i.e., $sp(a) = |R(a)|/m$), $sp(b)$ the support of item b, and $sp(ab)$ the support of the pair (a, b). Our task is to find pairs of items whose correlation coefficients are above a threshold θ. Table 1 summarizes the notations that we will use throughout the paper.

T	A table of "0"s and "1"s, representing the market-basket data. The columns of the table represent the items, and the rows represent the baskets. Basket i contains item j when $T(i, j) = 1$.
n	The number of items
m	The number of baskets
$R(a)$	$\{r \in [m] \| T(r, a) = 1\}$, i.e., the set of rows (baskets) that contain item a
$sp(a)$	The support of item a, i.e., $\|R(a)\|/m$
θ	The threshold that determines whether a pair is highly correlated

Table 1: Notations

For a pair of items a and b, the Pearson correlation coefficient (also called the ϕ correlation coefficient) can be expressed in terms of the supports:

$$\phi_{(a,b)} = \frac{sp(ab) - sp(a)sp(b)}{\sqrt{sp(a)sp(b)(1 - sp(a)) \cdot (1 - sp(b))}}.$$

We are looking for pairs (a, b) such that $\phi_{(a,b)} \geq \theta$. Intuitively, we know that, if a and b are highly correlated, the set $R(a) \cap R(b)$ should not be too small, compared to $R(a)$ or $R(b)$. We now establish this relationship formally. Without loss of generality, we assume that $sp(b) \geq sp(a)$ (equivalently $|R(b)| \geq |R(a)|$). Assume that a and b are highly correlated:

$$\frac{sp(ab) - sp(a)sp(b)}{\sqrt{sp(a)sp(b)(1 - sp(a)) \cdot (1 - sp(b))}} \geq \theta. \qquad (1)$$

Because $sp(a) \geq sp(ab)$, we can replace $sp(ab)$ with $sp(a)$ in Inequality 1 to obtain

$$\sqrt{\frac{sp(a) \cdot (1 - sp(b))}{sp(b) \cdot (1 - sp(a))}} \geq \theta. \qquad (2)$$

This is also the pruning rule used in the TAPER algorithm. Let $S = \sqrt{\frac{sp(a) \cdot (1 - sp(b))}{sp(b) \cdot (1 - sp(a))}}$. By Inequality 2, $S \geq \theta$. By the assumption that $sp(a) \leq sp(b)$, $S \leq 1$. Now, from Inequality 1

and the fact that $sp(ab) = \frac{|R(a)\cap R(b)|}{m}$, we have

$$\begin{aligned}\frac{|R(a)\cap R(b)|}{|R(b)|} &\geq \theta\cdot\sqrt{\frac{sp(a)\cdot(1-sp(b))}{sp(b)\cdot(1-sp(a))}}\cdot(1-sp(a))+sp(a)\\ &\geq \theta\cdot S\cdot(1-sp(a))+sp(a)\\ &= \theta\cdot S+(1-\theta\cdot S)sp(a)\\ &\geq \theta\cdot S.\end{aligned}$$

The last inequality comes from the fact that $\theta, S \leq 1$. Similarly,

$$\begin{aligned}\frac{|R(a)\cap R(b)|}{|R(a)|} &\geq \theta\cdot\sqrt{\frac{sp(b)\cdot(1-sp(a))}{sp(a)\cdot(1-sp(b))}}\cdot(1-sp(b))+sp(b)\\ &\geq \frac{\theta}{S}\cdot(1-sp(b))+sp(b)\\ &= \frac{\theta}{S}+(1-\frac{\theta}{S})sp(b)\\ &\geq \frac{\theta}{S}.\end{aligned}$$

Here the last inequality comes from the fact that $S \geq \theta$. Now consider the ratio $\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|}$. We have

$$\begin{aligned}\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|} &= \frac{|R(a)\cap R(b)|}{|R(a)|+|R(b)|-|R(a)\cap R(b)|}\\ &= \frac{1}{\frac{|R(a)|}{|R(a)\cap R(b)|}+\frac{|R(b)|}{|R(a)\cap R(b)|}-1}\\ &\geq \frac{1}{S/\theta+1/(\theta\cdot S)-1}\\ &= \frac{\theta}{S+1/S-\theta}\end{aligned}$$

Given $1 \geq S \geq \theta$, this ratio achieves its minimum value of θ^2 when $S = \theta$. Our rule is thus to prune when $\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|} < \theta^2$.

Candidate-Generation Rule: *We put the pair (a, b) into the candidate set only when $\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|} \geq \theta^2$. That is, (a, b) is selected if the Jaccard distance between $R(a)$ and $R(b)$ is smaller than $1-\theta^2$.*

Note that this bound is tight. That is, if we prune a pair (a, b) when $\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|}$ is slightly larger than θ^2, we may remove a pair whose correlation coefficient is slightly above θ. To see this, consider two items a and b with $R(a)\cap R(b) = R(a)$. Also, assume $sp(a)$ and $sp(b)$ are very small. In this case, $(1-sp(a)) \to 1$, and $(1-sp(b)) \to 1$. The correlation coefficient reduces to $\sqrt{\frac{sp(a)}{sp(b)}}$, i.e., $\phi_{(a,b)} \approx \sqrt{\frac{sp(a)}{sp(b)}}$. Because $\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|} = \frac{sp(a)}{sp(b)} = \theta^2$, $\phi_{(a,b)} \approx \theta$. Furthermore, $\phi_{(a,b)}$ increases with $\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|}$. Hence, if we prune a pair when $\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|}$ is above θ^2, we may have removed a pair whose correlation coefficient is above θ.

Next, we describe a method that can efficiently prune (or generate candidates) according to our rule. The candidate-generation method uses min hashing, which was introduced in [7, 11]. A min-hash function h_{min} maps an item in the data set to a number and has the following property: Given two items a and b,

$$\mathbb{P}r(h_{min}(a) = h_{min}(b)) = \frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|}$$

The following is a simple min-hash function from [11]: Let h be a general hash function that maps a number in $[m]$ to a random number in $[m^2]$. (The domain of the hash values is made much larger than the domain of the keys so that the probability of collision is small.) Then

$$h_{min}(a) = min_{r\in R(a)}\{h(r)\}$$

EXAMPLE 1. *Here is an example of min-hash. Assume that there are 10 rows (baskets) in total and that we choose the following values of h:*

r	0	1	2	3	4	5	6	7	8	9
h(r)	17	21	9	44	5	16	1	20	37	8

Also assume that item 3 appears in baskets 2, 5, and 8.

$$h_{min}(3) = min\{h(2) = 9, h(5) = 16, h(8) = 37\} = 9.$$

With min hashing, the larger the ratio $\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|}$, the more likely it is that the two items a and b will be hashed to the same value. This suggests a simple candidate-generation method: We can put every pair of items that have the same min-hash value in the candidate set. In this way, the pairs that satisfy our rule ($\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|} \geq \theta^2$) will be placed into the candidate set with high probability and the pairs that do not satisfy our rule ($\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|} < \theta^2$) will be selected with low probability.

However, the gap between the two probabilities is not wide enough to result in powerful pruning while maintaining a small false-negative probability. We use another technique from [11, 10] to widen the gap. We use k independent min-hash functions and define an equivalence relation "$\simeq$." For two items a and b, $a \simeq b$ if and only if a and b have the same min-hash values for all the k hash functions. The equivalence relation can be used to partition the items into equivalence classes. If, with one min-hash function $\mathbb{P}(a \simeq b) = x$, then, with k independent functions, $\mathbb{P}(a \simeq b) = x^k << x$. We repeat the whole process t times, each time with a different set of k min-hash functions. (A total of $k\cdot t$ independent min-hash functions is required.) The probability that a and b belong to the same equivalence class in at least one of the trials is $1-(1-x^k)^t$. We put into the candidate set all the pairs whose two items belong to the same equivalence class.

EXAMPLE 2. *We give an example that shows how candidates are generated after we obtain the min-hash values. Assume $k = 2$ and $t = 3$. Consider the items and their min-hash values in Table 2. Each item has 6 min-hash values. The hash values are grouped into 3 vectors, v_1, v_2 and v_3, each of size 2. The candidate-generation process takes $t = 3$*

item	Min-Hash Values		
	v_1	v_2	v_3
3	3, 8	12, 7	4, 8
9	1, 7	9, 20	17, 6
17	3, 8	10, 22	17, 6
19	5, 7	7, 22	6, 4

rounds. In round 1, vector v_1 is considered. v_1 of item 3 is equal to v_1 of item 17. Hence 3 and 17 are in an equivalence class and the pair (3, 17) is put in the candidate set. In the second round, vector v_2 is considered. This time, no two vectors are equal. No candidate is generated. In the third round, v_3 of item 9 is equal to v_3 of item 17. Hence (9,17)

is put in the candidate set. The process then stops. Note that the pair (3, 9) is not in the candidate set. This agrees with the fact that there is no vector of item 3 that is equal to a corresponding vector of item 9.

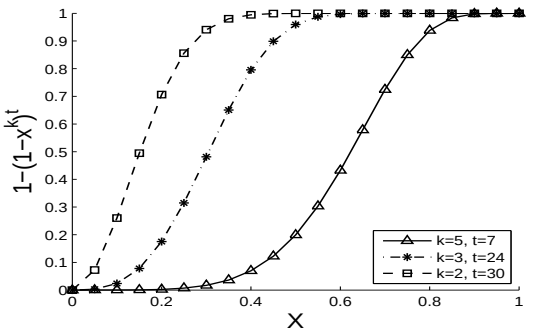

Figure 2: Shape of the Function $f(x) = 1 - (1 - x^k)^t$

Fig. 2 shows the function $f(x) = 1 - (1 - x^k)^t$ for several values of k and t. $f(x)$ has an "s" shape and approximates a threshold function, i.e., a function $g(x)$ that takes the value 1 when x is larger than the threshold and 0 when x is smaller than the threshold. $f(x)$ is an approximation of such a threshold function. We observe that k determines the sharpness of the transition from 0 to 1. t works with k to determine where the transition happens. For an ideal pruning result, we want the transition to be sharp, i.e., f to behave more like g. By choosing values for k and t properly, we can place into the candidate set the pairs that satisfy our rule ($\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|} \geq \theta^2$) with probability close to one and pairs that do not satisfy our rule ($\frac{|R(a)\cap R(b)|}{|R(a)\cup R(b)|} < \theta^2$) with probability close to zero.

3. FINDING ALL PAIRS OF HIGHLY CORRELATED ITEMS

We describe our full algorithm in this section. Our algorithm adopts the general framework of TAPER. To find the highly correlated pairs, it makes two passes through the data set. After the first pass, we generate a set of candidates. In the second pass, the correlation coefficients of the candidates are computed, and the pairs that are truly correlated are identified.

In the first pass, our algorithm performs min hashing for each item. The min-hash value of an item a is simply the minimum of the set $\{h(r)|r \in R(a)\}$. It is easy to see that this computation can be performed in one pass through the data. When using multiple min-hash functions, all hashes of an item can be computed at the same time—there is no need for multiple passes. Recall that we use a total of $k \cdot t$ independent hash functions. The parameters k and t are used to control the pruning and the probability of false negatives. k controls the effectiveness of the pruning. By using a large k, we can prune more pairs that do not satisfy our rule. In general, k can be determined by the limits on the available computational resources. t controls the probability of false negatives. The larger the t, the smaller the probability that we drop a pair satisfying our rule.

Once we have a value for k, we choose t according to the threshold θ and false-negative tolerance τ. We first dis-

Algorithm 1 Finding Correlated Pairs

We use $k \cdot t$ hash functions $h_0, h_1 \ldots, k_{k \cdot t-1}$ and a hash table HT. HT uses a hash function $\hat{h}$ that maps a vector of k integers (the key) to one of its buckets.

C: The set of highly correlated pairs
S: The candidate set
H: $H[i]$ is a vector that stores the $k * t$ min-hash values of item i. We also use $H[i][u : v]$ to mean the vector $(H[i][u], H[i][u+1], \ldots, H[i][v])$.

Compute Min-Hash:
for each item i and u from 0 to $k * t - 1$ **do**
  $H[i][u] \leftarrow m$
end for
In one pass through the data set:
for each item i in row j **do**
  for u from 0 to $k * t - 1$ **do**
    if $H[i][u] > h_u(j)$ **then**
      $H[i][u] \leftarrow h_u(j)$
    end if
  end for
end for

Generate Candidate Pairs:
for i from 0 to $t - 1$ **do**
  set all buckets of HT to ϕ
  for each item j **do**
    $v \leftarrow H[j][i * k : (i + 1) * k - 1]$
    $HT[\hat{h}(v)] \leftarrow HT[\hat{h}(v)] \cup j$
  end for
  for all bucket $HT[u]$ **do**
    if $HT[u]$ has more than one item **then**
      for every pair p of items in $HT[u]$ **do**
        $S \leftarrow S \cup p$
      end for
    end if
  end for
end for

Identify Truly Correlated Pairs:
In one pass through the data set:
for each pair (a, b) in S **do**
  Get $sp(ab)$
  $\phi_{(a,b)} \leftarrow \frac{sp(ab)-sp(a)sp(b)}{\sqrt{sp(a)sp(b)(1-sp(a))\cdot(1-sp(b))}}$
  if $\phi_{(a,b)} > \theta$ **then**
    $C \leftarrow C \cup (a, b)$.
  end if
end for
Output C.

cuss the meaning of τ and then describe how we choose its value. Note that, when we fix the values of both k and t, the stronger a pair's correlation is, the more likely it is that this pair will be placed in the candidate set. This is determined by the probability function in Fig. 2. A pair (a, b) has probability $1 - (1 - x^k)^t$ to be in the candidate set, where $x = \frac{|R(a) \cap R(b)|}{|R(a) \cup R(b)|}$. The stronger the pair's correlation is, the larger $\frac{|R(a) \cap R(b)|}{|R(a) \cup R(b)|}$, and therefore the pair is more likely to be in the candidate set. In other words, our candidate generation favors strongly correlated pairs. We choose t such that, for a pair (a, b), $\frac{|R(a) \cap R(b)|}{|R(a) \cup R(b)|} = \theta^2$, the probability of omitting this pair from the candidate set is below τ. (This is why we call τ the "false-negative tolerance.") Note this does not mean that our candidate generator misses every highly correlated pair with a probability τ. Only the pairs (a, b) that are at the border ($\frac{|R(a) \cap R(b)|}{|R(a) \cup R(b)|} = \theta^2$) (with relative low correlation) have probability of τ being left out. The other pairs with stronger correlation have false-negative probability smaller than τ. The formula for choosing t is then $t = log_{(1-\theta^{2k})}\tau$.

After going through the data set, we have $k \cdot t$ hash values for each item. They are grouped into t vectors each of size k. Using the equivalence relation $\simeq$ defined in the previous section, we partition the items into equivalence classes. We can do so with the help of a hash table. The hash table takes a vector of size k as a key and hashes the item into one of its buckets. The items in the same bucket form an equivalence class, and we add to the candidate set the pairs formed by two items in the same class. Note that each item has t vectors and thus t keys. Therefore, this process is repeated t times, each time using a different key. Once we have the candidate set, the second pass is straightforward. The whole algorithm is presented in Algorithm 1.

We now analyze the space and time complexity of our candidate-generation method. We first consider the memory requirement. For each item, we store $k \cdot t$ min-hash values. Because k and t are two constants determined only by the threshold θ and the false-negative tolerance τ, the total memory required by our candidate-generation method is $O(k \cdot t \cdot n) = O(n)$.

Now we consider the time complexity. While going through the data set, for each item i contained in basket (row) j, the row number j is hashed, and the hash value is compared to the currently stored minimum value. The current minimum is replaced if necessary. Clearly, this process requires time only necessary to go through the input. After going through the data set, we partition the set of items into equivalence classes according to the equivalence relation. Using a hash table, this can be done in $O(n)$ time. Since we repeat the partition process t times, the total time requirement is again $O(t \cdot n) = O(n)$.

In summary, the time and space complexity of our candidate-generation method are asymptotically the same as that of TAPER's candidate-generation process.

4. EXPERIMENT RESULTS

In this section, we report the results of testing our algorithm on several data sets. The experimental results show that our candidate-generation method prunes the unwanted pairs effectively, producing a small candidate set, and therefore achieves small overall running time. We first show, in Section 4.1, the execution time of our algorithm, comparing it with that of the TAPER algorithm. We then discuss the effectiveness of our pruning method in Section 4.2. In Section 4.3, we investigate how much we can gain by combining our pruning rule with TAPER's rule. In Section 4.4 we test the scalability of our algorithm. Finally, in Section 4.5, we investigate the tradeoff between running time and false-negative rate. (The false-negative rate is the ratio of the truly correlated pairs removed by our algorithm to the total number of truly correlated pairs.)

Before presenting the results, we first describe the data sets used in the experiments. We used both real and synthesized data. The real data were taken from several application domains. Table 2 summarizes the characteristics of these data sets. "pumsb" is a binarized census data set.

Data Set	# Items	# Records
pumsb	2113	49046
pumsb*	2089	49046
retail	16470	88162

Table 2: Characteristics of Real Data Sets

"pumsb*" is "pumsb" with items whose support is larger than 80% removed. "retail" contains market-basket data from an anonymous retail store [5]. We took these data sets from the FIMI website.[1]

Synthesized data were also used in the experiments. These data sets were produced using a generator included in the software package published by the ILLIMINE group.[2] It is essentially the same as the generator from the IBM Almaden Quest research group. Table 3 lists the characteristics of the three synthesized data sets that we used.

Data Set	# Items	# Records
SD1	15306	49240
SD2	29354	49181
SD3	42414	49233

Table 3: Characteristics of Synthesized Data Sets

In all experiments except the one in which we investigated the tradeoff between the running time and the false-negative rate, we set the false-negative tolerance τ to be smaller than 0.005. The experiments showed that the false-negative rate was indeed below 0.005. Furthermore, when the number of truly correlated pairs was small, our algorithm produced no false negatives using such a tolerance. Table 4 lists the values of k and t for different coefficient threshold θ.

θ	k	t
0.9	4	10
0.7	2	18
0.5	2	47
0.3	2	370

Table 4: Parameters of the Algorithm Used in the Experiments

We tested our algorithm on a Linux system, with an Intel Xeon CPU, 3.2GHz, and 2G memory.

[1] http://fimi.cs.helsinki.fi/data/
[2] http://illimine.cs.uiuc.edu/

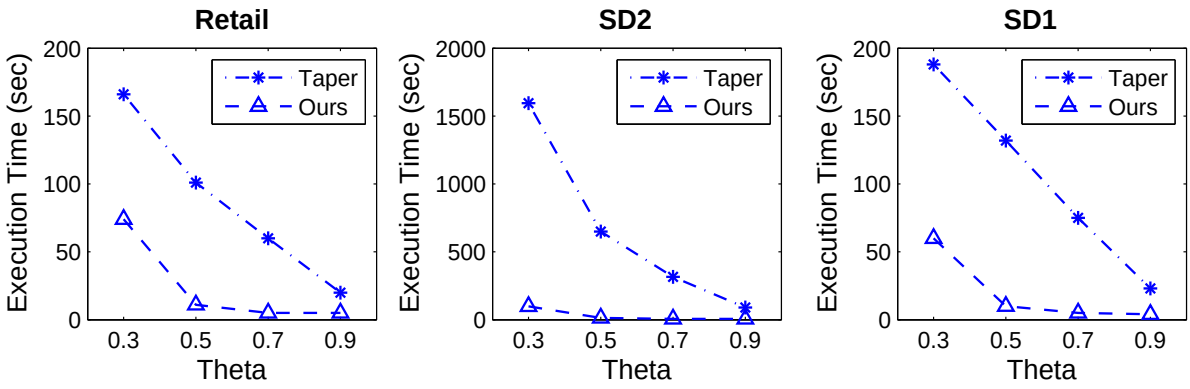

Figure 3: Overall Execution Time

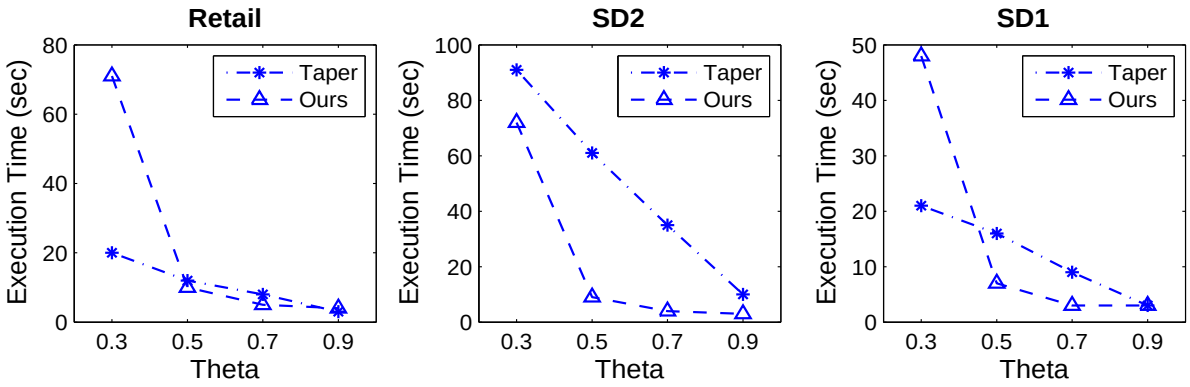

Figure 4: Candidate-Generation Time

4.1 Running Time

We measured two types of running times. One is the overall running time of the program. The other is the running time of the candidate-generation phase. In Fig. 3 and Fig. 4, we plot the overall and the candidate-generation running times of our algorithm, respectively, and compare them with those of the TAPER algorithm.

Our algorithm achieves smaller overall running time in these test cases. This is mainly due to the very small candidate set generated by the algorithm. In most cases, because the candidate set is so small, the second pass of our algorithm takes only a few seconds.

The comparison of the running times for candidate generation is interesting. In some cases (e.g., $\theta = 0.3$ for the "retail" data set), our candidate-generation process takes much longer than TAPER's candidate-generation process, while in some other cases (e.g., $\theta = 0.5, 0.7$ for the "retail" data set), our candidate-generation process takes less time.

In both algorithms, there are two steps in the candidate-generation process. First, they need to go through the input data in one pass. Second, they generate the candidate set. When TAPER goes through the input data, it counts the frequencies of the items. In our algorithm, we need to compute the min-hash values for the items. For each item, we may compute several min-hash values. Therefore, in the first step, our algorithm takes more time than TAPER, particularly when θ is small and we compute many min-hash values. In the second step, for each pair generated as a candidate, TAPER needs to look up the frequencies of its items. On the other hand, our algorithm simply puts the pairs of items that have the same hash value into the candidate set. Hence, in the second step, our algorithm requires less time. In summary, with small θ, our algorithm requires many min-hash values and spends more time to compute them. With large θ, although computing hash values still takes more time than counting the frequency, it can be compensated for by the savings in the second step, resulting in a candidate-generation process of the same or slightly faster speed.

4.2 Effectiveness of the Pruning

A small candidate set provides several advantages, particularly with massive data sets that have large numbers of items. A small candidate set reduces the number of pairs one needs to go through in order to identify the correlations. This improves efficiency. Second, one needs to store only a small number of pairs instead of all the pairs. This lowers the memory requirement. A memory requirement much smaller than the size of the data is critical if the algorithm is to work on massive data. It enables us to carry out computation in main memory, whereas external memory computations can be many orders of magnitude slower. Furthermore, if data are distributed among many machines, a small candidate set means small communications. Therefore, we will closely examine the size of the candidate sets, aka the pruning powers, of the algorithms.

To quantify pruning power, we define two measurements.

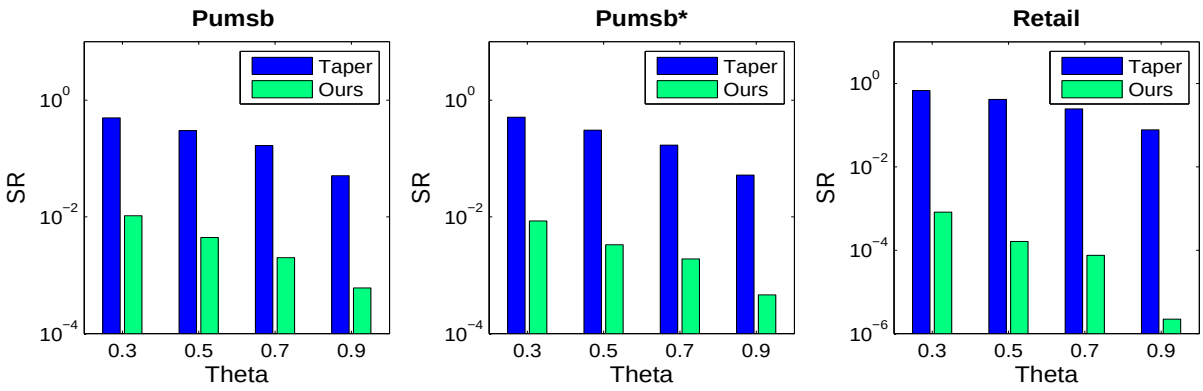

Figure 5: Shrink Ratio of the Candidate Sets

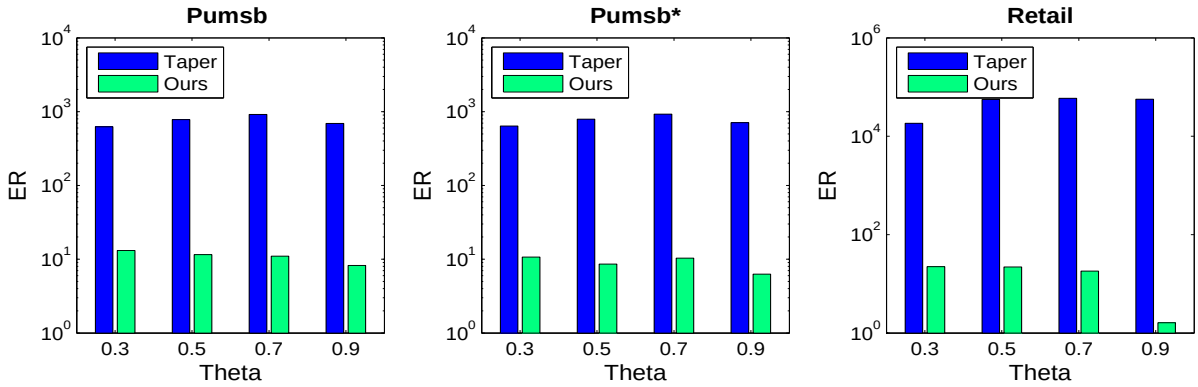

Figure 6: Expand Ratio of the Candidate Sets

The first one measures how small the candidate set is, compared to the set of all possible pairs. We call this quantity the *shrink ratio* (SR) of the candidate set. It is defined to be the ratio of the size of the candidate set to the total number of pairs. Note that the measurement called "pruning ratio" defined in [15] is exactly $1 - SR$. Our second quantity measures how large the candidate set is, comparing to the number of the truly correlated pairs. By truly correlated pairs, we mean the pairs whose coefficients are above the threshold. We call this measurement the *expand ratio* (ER) of the candidate set. A good candidate set should have both small ER and small SR. A small SR means we save a lot by pruning many unwanted pairs. A small ER means that the candidate set does not contain too many false positives. We remark that the false negative rate of our candidate sets in these experiments is below 0.005. That is, our candidate set misses less than 5/1000 of the truly correlated pairs. Hence, when our algorithm achieves a small ER (SR), it is indeed a small ER (SR), not one made by removing a lot of truly correlated pairs.

We first examine the shrink ratio of the candidate set generated by our algorithm and compare it to TAPER's ratio. We plot the shrink ratios in Fig. 5. The values of the ratios are represented by the heights of the bars. For each threshold θ, we plot the candidate set's SR from the two algorithms side by side. Note that the Y axis is in log scale. We make several observations here. First, the result shows that our pruning rule is much more powerful for these data sets than TAPER's. The candidate sets generated by our algorithm are often one order of magnitude smaller. For large data sets and large θ values, our candidate set can be several orders of magnitude smaller. For example, our candidate set for the "retail" data set, when $\theta = 0.9$, is about 10^{-5} of the number of all possible pairs. As we discussed earlier, with massive data sets, the number of total pairs can be huge and deep pruning is often sought, our pruning approach can be useful in these situations. Second, as we analyzed in the introduction, there is a group of uncorrelated pairs that are elusive to TAPER's pruning. These pairs contain two items whose frequency values are close. But the items in the pairs show up together in no or very few baskets. The experiment results show that, in the tested data sets, this type of pair is abundant. It also shows that our pruning method is effective in removing many such pairs. In many cases, pruning by our algorithm yields a candidate set whose size is even smaller than the number of items.

We now consider the expand ratio ER of the candidate set generated by both algorithms. We plot ER in Fig. 6. As in Fig. 5, the values of the ratios are represented by the heights of the bars. For each threshold θ, we plot the candidate set's ER from the two algorithms side by side. We see that our candidate set has quite a small ER (in many cases, around 10). For the "retail" data set, for large θ values, the ER of our candidate set is about 3. That is, the candidate set contains a number of false-positive pairs that is only twice

the number of truly correlated pairs. This shows that our algorithm has very strong pruning power.

4.3 Combine Two Pruning Rules

We investigate a pruning process that uses both rules. Both our and TAPER's pruning rules produce candidate sets with false positives. We analyzed how a pair can escape TAPER's pruning. We gave an efficient pruning method to catch these pairs. However, there are pairs that are elusive to our rule as well. In particular, a pair's passing our test does not necessarily mean that it also satisfies TAPER's rule. It can be expected that pruning using both rules is more effective. Note that applying the TAPER pruning rule in our algorithm is simple. It can be done by adding a step in our candidate-generation process. Namely, before putting a pair into the candidate set, we check the two items against TAPER's rule. Only those that pass the test can be put in the candidate set. We applied this double-rule pruning on the "pumsb" data set. The result is plotted in Fig. 7. For each threshold θ, we plot, side by side, the ER of the candidate sets generated by our rule and by the combined rules.

The result shows that the combination of the two rules is powerful. The size of the candidate set generated by the combined rule is of the same order as the number of truly correlated pairs. For large θ values, the ER is less than 2. This means that the number of false-positive pairs in the candidate set is less than the number of truly correlated pairs.

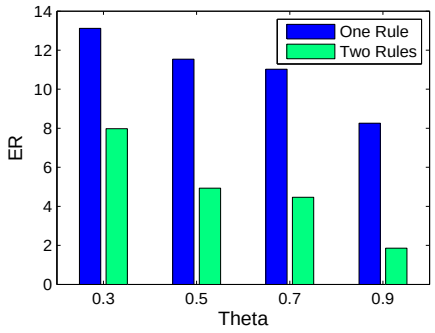

Figure 7: ER of the Candidate Sets Generated by Double Pruning and Our Pruning

4.4 Scalability of Our Algorithm

We used synthesized data sets listed in Table 3 to examine the scalability of our algorithm. Fig. 8 plots the execution time of the algorithm against the number of items in the data set. It shows that the execution time increases linearly with the number of items.

We also examine the scalability of the algorithm's pruning power. That is, we investigate whether the algorithm's effectiveness in pruning changes with the size of the data set. In Fig. 9, we plot the expand ratio ER of the candidate set against the number of items in the data set. We observe that, when the threshold θ is large, the ER values stay essentially the same for data sets with different numbers of items. This means that, for these θ values, our algorithm's pruning power is independent of the number of items in the data set. For small θ values, the candidate set's ER increases slightly. But the absolute value is still quite small.

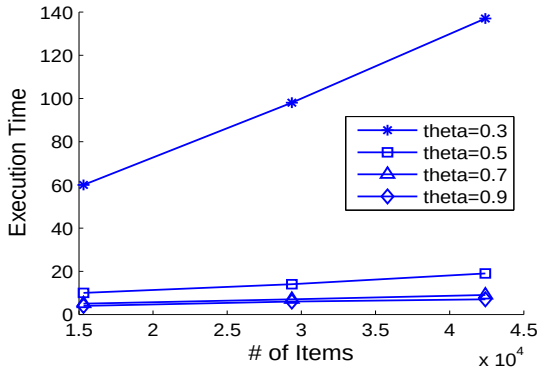

Figure 8: Scalability of the Algorithm

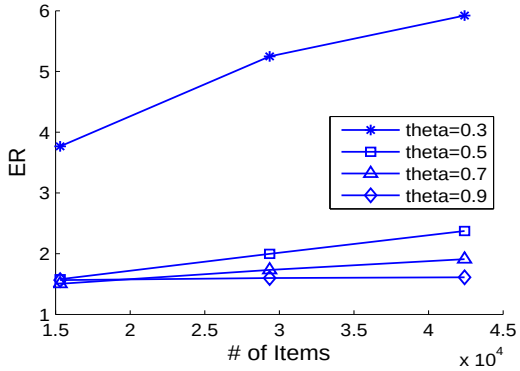

Figure 9: Scalability of the Algorithm's Pruning Power

4.5 Tradeoff between Running Time and False-Negative Rate

One difference between our pruning rule and TAPER's is that we produce false negatives, i.e., our pruning rule may remove truly correlated pairs. Such false negatives can be controlled in our algorithm by using multiple min-hash values. We set parameters as shown in Table 4 to bound the false-negative tolerance below 0.005. Our experimental results show that the false-negative rates of our candidate sets are indeed smaller than 0.005.

In some situations, such a small false-negative rate may not be necessary. With a larger false-negative tolerance, we can use fewer min-hashes. This in turn will speed up the algorithm. Therefore, there is a tradeoff between how fast the algorithm is and how many false negatives it produces. To investigate this tradeoff, we fix θ and change the parameters k and t. For different sets of k and t values, we measure the false-negative rate and the running time of our algorithm. The false-negative rate is plotted against the running time in Fig. 10. We observe that the false-negative rate decreases almost exponentially with the increase in running time. Hence, we can achieve a certain false-negative rate without increasing the running time too much.

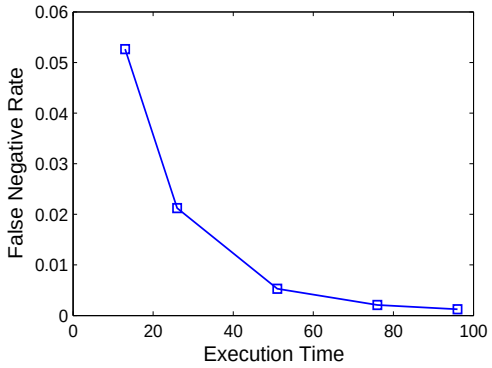

Figure 10: Tradeoff between Running Time and False Negative Rate

5. CONCLUSION AND FUTURE WORK

In this paper, we consider the problem of finding all pairs of highly correlated items in a large data set. In many cases, the number of highly correlated pairs is much smaller than the total number of pairs. It is wasteful to compute the coefficients of all the pairs in the process of identifying the highly correlated ones. For massive data sets, the computational cost of the naive method is prohibitive. Xiong *et al.* in [15] addressed this problem and proposed a two-step algorithm: TAPER. During the first step, they prune many of the unwanted pairs by simple computation. Only the coefficients of the remaining pairs are computed during the second step.

We examine the pruning rule in TAPER and observe that there is a relatively large group of uncorrelated pairs that are elusive to TAPER's pruning. To remove these pairs, a rule needs to consider the support of the pair in addition to the supports of individual items. A straightforward pruning method using such a rule will require computing the supports of all the pairs, which makes it meaningless. We propose a pruning rule and a pruning method that avoid this problem. Our experimental results show that our pruning method yields a candidate set much smaller than the ones produced by TAPER. Therefore, it achieves much larger savings of computational resources.

There are two directions for future work. First, we use the function $1-(1-x^k)^t$ to approximate a threshold function. When the threshold is close to one, this approximation is efficient, in the sense that k and t can take reasonably small values. When the threshold is close to zero, we need large k and t. If we had a more efficient approximation for small thresholds, our algorithm could be improved. Also, there are several other quantities that measure associations/correlations. It will be interesting to see whether our algorithm could be used for them.

6. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1993, pp. 207-216.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases*, 1994, pp. 487–499.

[3] R. Rayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. *Data Mining and Knowledge Discovery*, 4(2-3):217-240, 2000.

[4] C. Bucila, J. Gehrke, D. Kifer, and W.M. White. Dualminer: a dual-pruning algorithm for itemsets with constraints. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002, pp. 241–272.

[5] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: a case study. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999, pp. 254-260.

[6] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, 1997, pp. 255-264.

[7] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55:441-453, 1997.

[8] N. Friedman, I. Nachman and D. Peer. Learning Bayesian network structure from massive datasets: the Sparse Candidate algorithm. In *Proceedings of 15th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999.

[9] G. Grahne, L.V. Lakshmanan, and X. Wang. Efficient mining of constrained correlated sets. In *Proceedings of the 16th International Conference on Data Engineering*, 2000, pp 512–521.

[10] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, 1998, pp. 604-613.

[11] R. Motwani, E. Cohen, M. Datar, S. Fujiware, A. Gionis, P. Indyk, J. Ullman, and C. Yang. Finding interesting associations without support pruning. *IEEE Transactions on Knowledge and Data Engineering (special issue),* 13:64-78, 2001.

[12] Mehran Sahami. Learning limited dependence Bayesian classifiers. In *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 335–338.

[13] C. Silverstein, S. Brin, and R. Motwani. Beyond market baskets: generalizing association rules to dependence rules. *Data Mining and Knowledge Discovery* 2(1):39-68, 1998.

[14] P. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002, pp. 32–41.

[15] H. Xiong, S. Shekhar, P. Tan, and V. Kumar. Exploiting a support-based upper bound of Pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp 334–343.