Speeding up Correlation Search for Binary Data

Lian Duan and W. Nick Street
lian-duan@uiowa.edu
Management Sciences Department
The University of Iowa

Abstract

Finding the most interesting correlations in a collection of items is essential for problems in many commercial, medical, and scientific domains. Much previous research focuses on finding correlated pairs instead of correlated itemsets in which all items are correlated with each other. Though some existing methods find correlated itemsets of any size, they suffer from both efficiency and effectiveness problems in large datasets. In our previous paper [10], we propose a fully-correlated itemset (FCI) framework to decouple the correlation measure from the need for efficient search. By wrapping the desired measure in our FCI framework, we take advantage of the desired measure's superiority in evaluating itemsets, eliminate itemsets with irrelevant items, and achieve good computational performance. However, FCIs must start pruning from 2-itemsets unlike frequent itemsets which can start the pruning from 1-itemsets. When the number of items in a given dataset is large and the support of all the pairs cannot be loaded into the memory, the IO cost $O(n^2)$ for calculating correlation of all the pairs is very high. In addition, users usually need to try different correlation thresholds and the cost of processing the Apriori procedure each time for a different threshold is very high. Consequently, we propose two techniques to solve the efficiency problem in this paper. With respect to correlated pair search, we identify a 1-dimensional monotone property of the upper bound of any good correlation measure, and different 2-dimensional monotone properties for different types of correlation measures. We can either use the 2-dimensional search algorithm to retrieve correlated pairs above a certain threshold, or our new Token-Ring algorithm to find top-k correlated pairs to prune many pairs without computing their correlations. In addition, in order to speed up FCI search, we build an enumeration tree to save the fully-correlated value (FCV) for all the FCIs under an initial threshold. We can either efficiently retrieve the desired FCIs for any given threshold above the initial threshold or incrementally grow the tree if the given threshold is below the initial threshold.

1 Introduction and Related Work

The analysis of relationships between items is fundamental in many data mining problems. For example, the central task of association analysis [2] is to discover a set of binary

variables (called items) that co-occur frequently in a transaction database. Correlation search is useful for sales promotions, website catalog design, and store shelf layout. Regardless of how the relationships are defined, such analysis requires a proper measure such as support, confidence [2], or lift [6] to evaluate the interestingness of association patterns. Although we are, in general, interested in correlated sets of arbitrary size, most of the published work with regard to correlation is related to finding correlated pairs [19, 13]. Related work with association rules [6, 7, 17] is a special case of correlation pairs since each rule has a left- and right-hand side. Given an association rule $X \Rightarrow Y$ where X and Y are itemsets, $support = P(X \cap Y)$ and $confidence = P(X \cap Y)/P(X)$ [2, 17] are often used to represent its significance. However, these can produce misleading results because of the lack of comparison to the expected probability under the assumption of independence. In order to overcome the shortcoming, lift [6], conviction [7], and leverage [18] are proposed. $Lift = P(X \cap Y)/(P(X)P(Y))$ measures the ratio of X and Y's actual co-occurrence to the expected value under the independence assumption. $Conviction = (P(X)P(\bar{Y}))/P(X \cap \bar{Y})$ compares the probability that X appears without Y if they were independent with the actual frequency of the appearance of X without Y. $Leverage = P(X \cap Y) - P(X)P(Y)$ measures the difference of X and Y appearing together in the data set and what would be expected if X and Y were statistically independent.

However, there are some applications in which we are specifically interested in correlated itemsets rather than correlated pairs. For example, we are interested in finding sets of correlated stocks in a market, or sets of correlated gene sequences in a microarray experiment. But finding correlated itemsets is much harder than finding correlated pairs because of three major problems. First, computing correlation for each possible itemset is an NP-complete problem [16]. Second, if there are some highly correlated items within an itemset and the rest are totally independent items, most correlation measures still indicate that the itemset is highly correlated. No existing measure provides information to identify the itemsets with independent items. Third, there is no guarantee that the itemset has high correlation if any of its strict subsets are highly correlated. Since the correlated pair search can be considered a special case of the correlated itemset search for 2-itemsets, we focus on the correlated itemset search in the rest of the paper. The first related technique for correlated itemset search is frequent itemset mining. By using support, the search is fast. However, co-occurrence is related to two factors: the correlation and the single item support within the itemset. In other words, co-occurrence is related but not equal to correlation. The second technique is to find the top-k correlated itemsets. Tan [19] compared 21 different measures for correlation. Only six of the 21 measures can be used to measure the correlation within a given k-itemset. Dunning [12] introduced a more statistically reliable measure, likelihood ratio, which outperforms other correlation measures. It measures the overall correlation within a k-itemset, but cannot identify the itemsets with irrelevant items. Jermaine [16] extended Dunning's work and examined the computational issue of probability ratio and likelihood ratio. Bate [3] proposed a correlation measure called Bayesian confidence propagation neural network (BCPNN) which is good at searching for correlated patterns occurring rarely in the whole dataset. The above correlation measures can search for more meaningful correlated patterns than support, but do not have the downward-closed property to reduce the computational expense. A property ρ is downward-closed if for every set with property ρ,

all its subsets also have property ρ [2]. Once we can find a set which does not satisfy a given downward-closed property, we can prune the exponential superset search space immediately. Since no existing correlation measures satisfying the three primary correlation properties [10] have the downward-closed property to facilitate the search, finding the top-k correlated itemsets is very computationally expensive. To sum up, frequent itemset mining is efficient, but not effective; top-k correlated itemset mining is effective, but not efficient. In order to solve the problem, others proposed efficient correlation measure like all-confidence [17]. All-confidence is as fast as support; however, the correlation is still measured in a sub-optimal way. Requiring the correlation measure to be both effective and efficient is too demanding. Instead of proposing another efficient correlation measure, we proposed the framework of fully-correlated itemsets (FCI) [10], in which any two subsets are correlated. This framework can not only decouple the correlation measure from the need for efficient search, but also rules out the itemsets with irrelevant items. With it, we only need to focus on effectiveness when selecting correlation measures.

Even though the FCI framework can impose the downward-closed property for any given correlation measure, there are still two computational issues to find the desired maximal fully-correlated itemsets (MFCIs). First, unlike finding maximal frequent itemsets which can start pruning from 1-itemsets, finding MFCIs must start pruning from 2-itemsets. However, as the number of items and transactions in the dataset increases, calculating the correlation values for all the possible pairs is computationally expensive. Since there is no monotone property for correlation measures which can help prune, the brute-force method is straightforward. When a database contains 10^5 items, a brute-force approach requires computing the correlation value of $0.5 * 10^{10}$ pairs. Even worse, when the support of all the pairs cannot be loaded into the memory, the IO cost for retrieving supports is much more expensive than the computational cost for calculating correlations. Therefore, an efficient correlated pair search algorithm can speed up the MFCI search. The most significant progress on correlated pair search was made by Xiong [22, 23]. He made use of the upper bound of the Pearson correlation coefficient (ϕ-coefficient) for binary variables. The computation of this upper bound is much cheaper than the computation of the exact correlation because this upper bound is a function of single item supports. In addition, the upper bound has special 1-dimensional and 2-dimensional properties that prune many pairs from the search space without the need to compute their upper bounds. The algorithm TAPER [23] makes use of the 1-dimensional and 2-dimensional properties to retrieve correlated pairs above a given threshold. The algorithm TOP-COP [22] uses the 1-dimensional property and a diagonal traversal method, combined with a refine-and-filter strategy, to efficiently find the top-k pairs. However, this work is only related to the ϕ-coefficient, which is not the only or the best correlation measure. Second, users usually need to try different correlation thresholds for different desirable MFCIs. For example, when we set the likelihood ratio threshold to $15,000$ using the Netflix dataset, we successfully retrieve the series of "Lord of the Rings 1, 2, and 3" in one MFCI; however, we only retrieve several pairs of the TV show "Sex and the City" like {1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}. In order to get the whole series of "Sex and the City 1, 2, 3, 4, 5, 6" in one MFCI, we have to lower the threshold to 8000. However, the cost of processing the Apriori procedure each time for a different correlation threshold is very high. Since the framework of FCI is relatively new, there is no related work on improving its efficiency.

Given an itemset $S = \{I_1, I_2, ..., I_m\}$ with m items in a dataset with sample size n, the actual probability is $tp = P(S)$, the expected probability is $ep = \prod_{i=1}^{m} P(I_i)$. Since itemset mining has a long history from 1993 when frequent itemset mining [2] was proposed, a lot of related concepts are proposed such as closed itemset [24], contrast itemset [1], and discriminative itemset [9]. These concepts are related to each other in a certain degree; however, in this paper, we only focus on the performance issue and the general framework of finding correlated itemsets where the correlation measure must explicitly use the measures tp and ep. In addition, there are several other issues related to correlation that we also don't address here, such as effectiveness of correlation measures, statistical type-1 and type-2 error reduction, error-tolerant methods using sampling, and the existing optimization methods on itemset mining. In [11], we carefully studied 19 correlation measures and provided four extra properties for correlation measures from the statistical point of view which can help users to choose the correlation measure retrieving results closer to human intuition. Webb [21] proposed a framework of reducing the type-1 and type-2 error of itemset mining which can be applied to our pattern search framework. Zhang [25] studied the distribution of the ϕ-coefficient and relaxed the upper bound in TAPER in order to speed up search. Guns [15] formulated the traditional itemset mining problem, such as frequent, closed, and discriminative itemset mining, as constraint programming problems, and then applied an existing solver for constraint programming to speed up the search. However, traditional itemset mining can easily retrieve the value of a given itemset S, while the fully-correlated itemset mining cannot retrieve the value of a given itemset S without calculating the value of all the subsets of S. Therefore, there is no obvious way of formulating our problem as a constraint programming problem.

In addition, we differentiate our search on the corresponding itemset cell from that on the itemset family. Given the itemset family $\{A,B,C\}$, it includes 8 cells, such as cell $\{A,B,C\}$, cell $\{A,B,\bar{C}\}$, and so on. Some measures like leverage [18] and the simplified χ^2-statistic [16] evaluate the correlation corresponding to a cell, but other measures like entropy [20] and χ^2-statistic evaluate the overall correlation of all the cells related to a given itemset family. We are not interested in the search of itemset family for two reasons. First, it messes up the positive correlation and negative correlation. If A, B, and $\bar{C}$ are positively correlated with each other, the search on itemset family can only tell us the dependence of items A, B, and C, but we don't know whether they are positively correlated. Second, for an itemset S with the size m, we need to calculate the value for 2^m cells. Though there are some smart ways of avoiding redundant calculation [20], it is still computationally expensive for large itemsets.

The rest of this paper is organized as follows. Section 2 presents basic notions of correlation properties, correlation upper bound, 1-dimensional and 2-dimensional properties, and the fully-correlated itemset framework. We propose several methods to speed up correlated pair and correlated itemset search in Section 3. Section 4 shows the experimental results. Finally, we draw a conclusion in Section 5.

2 Basic Properties

In this section, some basic properties of correlation are introduced to better explain the improved performance of correlation search.

2.1 Correlation Measure Properties

To find highly correlated itemsets, we should find a reasonable correlation measure first. Since it is impossible to compare against every possible measure [13], we use the following three commonly accepted criteria to generalize the correlation measures, such as ϕ-coefficient, the simplified χ^2-statistic, probability ratio, leverage, and likelihood ratio.

Given an itemset $S = \{I_1, I_2, ..., I_m\}$, a correlation measure M must satisfy the following three properties [18] :

P1: M is equal to a certain constant number C when all the items in the itemset are statistically independent.

P2: M monotonically increases with the increase of $P(S)$ when all the $P(I_i)$ remain the same.

P3: M monotonically decreases with the increase of any $P(I_i)$ when the remaining $P(I_k)$ and $P(S)$ remain unchanged.

The first property requires a fixed reference for independence. The last two properties are based on the following intuition.The correlation value increases when the expected probability stays the same while the actual probability goes up. The correlation value decreases when the expected probability goes up while the actual probability stays the same. In the following sub-sections, we will make use of the above three properties to infer correlation upper bound properties. Interested readers can find more details related to correlation properties in [11]. Without loss of generality, in this paper we do experiments by using the best correlation measure [10], likelihood ratio, which measures the ratio of the likelihood of k out of n transactions containing the itemset S when the single trial probability is the true probability to that when the single trial probability is the expected probability if all the items in S are independent from each other.

2.2 Correlation Upper Bound for Pairs

Theorem 1. *Given any pair $\{I_i, I_j\}$ and support values $P(I_i)$ for item I_i and $P(I_j)$ for item I_j, the correlation upper bound $CUB(I_i, I_j)$, i.e. the highest possible correlation value of the pair $\{I_i, I_j\}$, is the correlation value for $\{I_i, I_j\}$ when $P(I_i \cap I_j) = min\{P(I_i), P(I_j)\}$.*

Proof. The upper bound of the support value $P(I_i \cap I_j)$ for the 2-itemset $\{I_i, I_j\}$ is $min\{P(I_i), P(I_j)\}$ and the lower bound is $max\{0, P(I_i) + P(I_j) - 1\}$. For the given $P(I_i)$ and $P(I_j)$, any correlation measure reaches its upper bound when $P(I_i \cap I_j) = min\{P(I_i), P(I_j)\}$ and its lower bound when $P(I_i \cap I_j) = max\{0, P(I_i) + P(I_j) - 1\}$ according to correlation property 2. □

The calculation of correlation upper bound (CUB) for pairs only needs the support of each item which can be saved in memory even for large datasets; however, the calculation of correlation for pairs needs the support of pairs, incurring a high IO cost for large datasets. Given a 2-itemset $\{I_i, I_j\}$, if its correlation upper bound is lower than the correlation threshold we specify, there is no need to retrieve the support of the 2-itemset $\{I_i, I_j\}$, because the correlation value for this pair is definitely lower than the threshold no matter what the

	I_1	I_2	I_3	I_4	I_5	I_6
I_1		X	X	X	X	X
I_2			X	X	X	X
I_3				X	X	X
I_4					X	X
I_5						X
I_6						

Table 1: Pair Matrix

support is. If we only retrieve the support of a given pair in order to calculate its correlation when its upper bound is greater than the threshold, we will save a lot of unnecessary IO cost when the threshold is high.

2.3 1-Dimensional Property

Although the correlation upper bound calculation can save a lot of unnecessary IO cost, it still requires the correlation upper bound calculation for all the possible pairs. Therefore, we make use of the 1-dimensional property to eliminate unnecessary upper bound checks. It is motivated by the search algorithm TAPER [23] for the ϕ-coefficient. In order to fit the situation for any good correlation measure, we sort the items according to their supports in increasing order instead of decreasing order as in TAPER.

Theorem 2. *Given a user-specified threshold θ and an item list $\{I_1, I_2, ..., I_m\}$ sorted by support in increasing order, the correlation upper bound of $\{I_i, I_k\}$ is less than θ if the correlation upper bound of $\{I_i, I_j\}$ is less than θ and $i < j < k$.*

Proof. Since $i < j < k$ and the item list $\{I_1, I_2, ..., I_m\}$ is sorted by support in increasing order, $P(I_i) \leq P(I_j) \leq P(I_k)$. Then, the support upper bound of both $\{I_i, I_j\}$ and $\{I_i, I_k\}$ is equal to $P(I_i)$. For the pair $\{I_i, I_j\}$, $P_{upper}(I_i \cap I_j) = P(I_i)$ and $P_{expected}(I_i \cap I_j) = P(I_i)P(I_j)$. For the pair $\{I_i, I_k\}$, $P_{upper}(I_i \cap I_k) = P(I_i)$ and $P_{expected}(I_i \cap I_k) = P(I_i)P(I_k)$. Therefore, $P_{upper}(I_i \cap I_k) = P_{upper}(I_i \cap I_j)$, and $P_{expected}(I_i \cap I_k) \geq P_{expected}(I_i \cap I_j)$ because $P(I_k) \geq P(I_j)$. According to correlation property 3, we get $CUB(I_i, I_k) \leq CUB(I_i, I_j)$. Since $CUB(I_i, I_j) < \theta$, $CUB(I_i, I_k) < \theta$. □

To get all the pair correlation upper bounds, we need to calculate an $n \times n$ matrix for an item list sorted by support as shown in Table 1. Since this matrix is symmetrical, we only need to calculate the upper part above the diagonal. If the data set contains n items, $(n-1)$ branches (rows) need to be calculated. The pairs in branch i are $\{I_i, I_j\}$ where $i+1 \leq j \leq n$. The reference item I_i is fixed in each branch i and it has the minimum support value due to the way we construct the branch. Since items in each branch are also sorted based on their support in increasing order, the correlation upper bound of $\{I_i, I_j\}$ monotonically decreases with the increase of j by Theorem 2. In other words, $CUB(I_i, I_k) < \theta$ when $CUB(I_i, I_j) < \theta$ and $j + 1 \leq k \leq n$.

2.4 2-Dimensional Property

When the threshold is low, we might still calculate a lot of correlation upper bounds by using the 1-dimensional property. In order to avoid too many correlation upper bound checks, a 2-dimensional property similar to TAPER [23] for the ϕ-coefficient is used. However, we present different 2-dimensional properties and use different search sequences for three different types of correlation measures.

Given three items I_i, I_j, and I_k with $P(I_i) \geq P(I_j) \geq P(I_k)$, the correlation measure M is

- Type 1 if $CUB(I_i, I_j) \geq CUB(I_i, I_k)$.
- Type 2 if $CUB(I_i, I_j) \leq CUB(I_i, I_k)$.
- Type 3 if $CUB(I_i, I_j) = CUB(I_i, I_k)$.

Given an itemset $S = \{I_1, I_2, ..., I_m\}$ with m items, the actual probability is $tp = P(S)$, the expected probability is $ep = \prod_{i=1}^{m} P(I_i)$. The simplified χ^2-statistic, leverage, likelihood ratio, and ϕ-coefficient are all type 1 correlation measures. Although we know of no existing type 2 correlation measure, we can construct a type 2 correlation measure which satisfies all the three mandatory correlation properties like

$$Correlation_{type2} = \begin{cases} \frac{\sqrt{tp-ep}}{ep}, & \text{when } tp \geq ep \\ -\frac{\sqrt{ep-tp}}{ep}, & \text{when } tp < ep \end{cases} \quad (1)$$

Probability ratio is a type 3 correlation measure.

If we sort items according to their support values in increasing order and calculate the upper bound for each pair, Tables 2, 3, and 4 show the typical patterns of different types of correlation measures. For different types of correlation measures, we get different 2-dimensional properties. For type 1 correlation measures, the upper bound value decreases from left to right and from bottom to top. If the upper bound of the current cell $\{I_i, I_j\}$ is below θ, then $CUB(I_p, I_q) < \theta$ when $1 \leq p \leq i$ and $j \leq q \leq n$. In the example, if the correlation upper bound of the randomly selected cell $\{I_2, I_5\}$ is below $\theta = 40$, the cells in the gray area are all below θ. For type 2 correlation measures, the upper bound value decreases from left to right and from top to bottom. If the upper bound of the current cell $\{I_i, I_j\}$ is below θ, then $CUB(I_p, I_q) < \theta$ when $i \leq p \leq n-1$ and $max(p+1, j) \leq q \leq n$. In the example, if the correlation upper bound of the randomly selected cell $\{I_2, I_5\}$ is below $\theta = 60$, the cells in the gray area are all below θ. For type 3 correlation measures, the rightmost column has the lowest upper bound value. If the upper bound of the current cell $\{I_i, I_j\}$ is below θ, then the upper bounds of any cell to its right is below θ. In the example, if the correlation upper bound of the randomly selected cell $\{I_2, I_5\}$ is below $\theta = 40$, the cells in the gray area are all below θ.

2.5 Fully-correlated Itemset Framework

Correlation search for arbitrary size itemset has two more problems than that for pairs. First, we need a way to rule out independent items from a given itemset. Second, we need

	I_1	I_2	I_3	I_4	I_5	I_6
I_1		43	39	35	26	13
I_2			56	41	33	25
I_3				42	37	29
I_4					59	37
I_5						46
I_6						

Table 2: Type 1 Correlation Upper Bound Pattern

	I_1	I_2	I_3	I_4	I_5	I_6
I_1		92	84	76	69	51
I_2			75	67	58	41
I_3				55	42	38
I_4					36	23
I_5						17
I_6						

Table 3: Type 2 Correlation Upper Bound Pattern

	I_1	I_2	I_3	I_4	I_5	I_6
I_1		53	41	37	23	17
I_2			41	37	23	17
I_3				37	23	17
I_4					23	17
I_5						17
I_6						

Table 4: Type 3 Correlation Upper Bound Pattern

	C		not C	
	B	not B	B	not B
A	25	25	25	425
not A	25	25	25	425

Table 5: An independent case

a framework or a measure with downward-closed property to speed up search. Since no existing correlation measures satisfying the three correlation properties can solve either of the above problems, we proposed the fully-correlated itemset framework [10].

Definition 1. *Given an itemset S and a correlation measure, if the correlation value of any subset containing no less than two items of S is higher than a given threshold θ, this itemset S is a fully-correlated itemset.*

This definition has two very important properties. First, it can be incorporated with any correlation measure for itemsets to impose the downward-closed property which can help us to prune unsatisfied itemsets quickly like Apriori [2]. If a given itemset S is not a fully-correlated itemset, it must contain a subset S' whose correlation is below the threshold. Since the subset S' is a subset of any superset of S, any superset of S is not a fully-correlated itemset either. With this framework, we only need to focus on effectiveness side when selecting correlation measures. Second, it helps to rule out an itemset with independent items. For example in Table 5, B is correlated with C, and A is independent from B and C. Suppose we use the likelihood ratio and set the correlation threshold to be 2. The likelihood ratio of the set $\{A, B, C\}$ is 8.88 which is higher than the threshold. But the likelihood ratio of its subset $\{A, B\}$ which is 0 doesn't exceed the threshold. According to our definition, the set $\{A, B, C\}$ is not a fully-correlated itemset. The only fully-correlated itemset in this example is $\{B, C\}$ whose likelihood ratio is 17.93.

3 Correlation Search

As the number of items and transactions in the data set increases, calculating the correlation values for all the possible pairs is already computationally expensive, and it is much harder to calculate correlation for all the possible itemsets. We propose several methods to handle the efficiency problem.

3.1 Correlated Pair Search

Correlated pair search is a special case of correlated itemset search. Unlike frequent itemset search which can start pruning for 1-itemsets, correlated itemset search even with the downward-closed property has to start pruning with 2-itemsets. Therefore, correlated pair search is the cornerstone of correlated itemset search. In general, there are two types of correlated pair searches: correlated pairs above a certain threshold and top-k correlated pairs. Although the correlation itself doesn't have a monotone property to help prune, the correlation upper bound does. Consequently, we make use of the 1-dimensional monotone

property of the upper bound of any good correlation measure, and different 2-dimensional monotone properties for different types of correlation measures. We can either use the 2-dimensional search algorithm to retrieve correlated pairs above a certain threshold, or our new Token-Ring algorithm to find the top-k correlated pairs, to prune many pairs without the need to compute their correlations. Although the 2-dimensional search algorithm can efficiently find correlated pairs above a given threshold, it is difficult for users to provide an appropriate correlation threshold in real-world applications since different data sets have different characteristics. Instead, we try to find the top-k correlated pairs. However, when using the Token-Ring algorithm to find the top-k correlated pairs, we could spend a lot of time on calculation and end up with trivial results in a data set that doesn't contain strong positive correlations. Therefore, we propose a user procedure to combine the 2-dimensional search algorithm and the Token-Ring algorithm to reconcile the two tasks.

3.1.1 Correlated Pairs above a Certain Threshold

In this subsection, we focus on task 1: finding correlated pairs above a certain threshold. The easiest way of speeding up search is calculating the upper bound before the real correlation. If the upper bound is already below the threshold, there is no need to retrieve the pair support to calculate the real correlation. Therefore, we greatly reduce IO cost for high thresholds. Upper bound checking needs to calculate the upper bound of all the possible $n(n-1)/2$ pairs which is still computationally expensive. The 1-dimensional property can be used to save a lot of unnecessary upper bound checks for very high thresholds. We specify the reference item A and start a search within each branch. The reference item A is fixed in each branch and it has the minimum support value due to the way we construct the branch. Items in each branch are also sorted based on their support in increasing order. By Theorem 2, the correlation upper bound of $\{A, B\}$ monotonically decreases with the increase of the support of item B. Therefore, if we find the first item B, the turning point, which results in an correlation upper bound less than the user-specified threshold, we can stop the search for the current branch. If the upper bound is greater than the user-specified threshold, we calculate the exact correlation and check whether this pair is really satisfied. Furthermore, the 2-dimensional property can be used to prune cells faster. The key difference between 1-dimensional search and 2-dimensional search is that the 2-dimensional search algorithm records the turning point in the previous branch and starts computing from that point instead of the beginning of the current branch. But we will use different search sequences for three different types of correlation measures. For type 1 correlation measures, we start the search from the upper-left corner. If the upper bound of the current cell is above the threshold θ, we will check the cell to the right of it; else we will instead check the cell under it. Take the threshold 36 in Table 2 for example, the search sequence is (I_1, I_2), (I_1, I_3), (I_1, I_4), (I_2, I_4), (I_2, I_5), (I_3, I_5), (I_3, I_6), and (I_4, I_6). The upper bound of any cell below this boundary is greater than the threshold θ. For type 2 correlation measures, we start the search from the upper-right corner. If the upper bound of the current cell is greater than θ, we check the cell below the current cell; else, we check the cell to the left of the current cell. Take the threshold 45 in Table 3 for example, the search sequence is (I_1, I_6), (I_2, I_6), (I_2, I_5), (I_3, I_5), and (I_3, I_4). The upper bound of any cell above this boundary is greater than θ.

For type 3 correlation measures, the rightmost column has the lowest upper bound value. We only need to search the first branch. If the current column is above the threshold, we continue the search of the right-side column until the current column is below the threshold.

Performance Analysis

Theorem 3. *The number of upper bound calculations in the 2-dimensional search is between* $n-1$ *and* $2n-3$ *for the first type,* $n-1$ *for the second type, and between* 1 *and* $n-1$ *for the third type.*

Proof. For the 2-dimensional search in type 1, the number of calculations is determined by the cell where we stop on the right most column. If the algorithm stops at the upper-right corner, the whole search moves from left to right $n-1$ times. If the algorithm stops at the lower-right corner, the whole search moves from left to right $n-1$ times and from up to down $n-2$ times, so there are $n-1+n-2=2n-3$ movements. Since the search might stop at any cell on the most right column, the calculation for type 1 is between $n-1$ and $2n-3$.

For the second type, the search starts at the upper-right corner and stops at one of the border cells along the diagonal, that is, cell $C_{i,i+1}$ where $i=1,2,...,n-1$. From the upper-right corner cell $C_{1,n}$ to the cell $C_{i,i+1}$, we have to make i movements from up to down and $n-i-1$ movements from right to left. In all, we need to make $i+n-i-1=n-1$ movements no matter in which cell we stop the search.

For the third type, we stop the search between the calculation for the first column and that of the last column, so the number of calculation is between 1 and $n-1$. □

Different methods of facilitating correlated pair search above a certain threshold are discussed in this section. If we don't make use of correlation upper bound at all, we need to calculate the correlation for all the possible pairs to find correlated pairs above a threshold. This brute-force method will have $n(n-1)/2$ support IO cost and $n(n-1)/2$ correlation calculations. Here, we call a pair a candidate if its correlation upper bound is greater than the user-specified threshold θ. For a given dataset and θ, the number of candidates, α, is fixed. For each candidate, we have to calculate its true correlation to determine whether its correlation is above the threshold or not. No matter which method we use, the IO cost of retrieving support and the computing cost of correlation for α candidates are inevitable. The only difference is the number of correlation upper bound checks. If it is just upper bound calculation, $n(n-1)/2$ upper bounds need to be calculated. If it is 1-dimensional search, α plus an extra β upper bounds need to be calculated where β is between 0 and $n-1$ depending on in how many branches we check all the cells. If it is 2-dimensional search, γ upper bounds need to be calculated which is between $n-1$ and $2n-3$ for type 1, $n-1$ for type 2, and between 1 and $n-1$ for type 3.

Here, we further study the difference between 1-dimensional and 2-dimensional search. If we want to find correlated pairs above a given threshold, α IO cost and correlation calculation is inevitable because each candidate must be checked. The difference is that 1-dimensional has $\alpha+\beta$ upper bound calculations and 2-dimensional has γ upper bound calculations. Since α is much greater than β and γ, and IO is much more expensive than calculation,

the α IO cost dominates the computational cost. There is no significant difference between 1-dimensional and 2-dimensional search with regard to finding pairs above a threshold. However, it is hard to determine which threshold is proper for a given dataset. Normally, this threshold is determined by intuition. But we can use the number of candidates for the given threshold to do the evaluation. For example, we may know how much time we spend on retrieving support and calculating correlation for a pair and how long we are allowed to search. We can estimate how many candidates we can afford to search. When just finding the number of candidates, we can avoid the α support IO cost and α correlation calculation. In this case, 2-dimensional search can find the number of candidates in linear time which is much better than 1-dimensional search.

3.1.2 Top-k Correlated Pairs

Although 2-dimensional search can efficiently find correlated pairs above the threshold θ, it is hard for users to provide a proper correlation threshold in many applications. Instead, finding top-k correlated pairs is more meaningful for users.

TOP-COP Search The original TOP-COP algorithm [22] introduced a diagonal traversal method for efficiently searching the top-k correlated pairs of ϕ-coefficient. While it exploits the 2-dimensional monotone property of the upper bound of the ϕ-coefficient, it needs $O(n^2)$ space to save pair status indicating whether or not the pair needs to be checked. Saving pair status only takes 3% of the space of saving support, but it is still not feasible for large datasets. If items are sorted by their support in increasing order, the upper bound of pairs for any good correlation measure decreases from left to right for each row in Table 1. The search starts from the diagonal consisting of $\{I_i, I_{i+1}\}$ which is the closest to the main diagonal, then goes to the diagonal consisting of $\{I_i, I_{i+2}\}$ which is the second closest to the main diagonal, and so on. During the iterative search process, this method maintains a top-k list and a pair is pushed into this list if its correlation coefficient is greater than the current minimum correlation coefficient in the top-k list. The search stops if the maximal upper bound of all the pairs in a diagonal is less than the current minimum correlation coefficient in the top-k list.

Token-Ring Search TOP-COP calculates all the pairs in a diagonal to find the maximal upper bound. In fact, the upper bound calculation of some pairs in the diagonal can be avoided if the upper bound of their left pair is already less than the current minimum correlation coefficient τ in the top-k list. In order to achieve that, we propose a token-ring search algorithm. We treat all the $n-1$ branches as nodes in the token-ring. Only the branch that gets the token can calculate the upper bound of the leftmost uncalculated pair in this branch and compare its upper bound against τ. If the upper bound is above τ, we will check the correlation value of this pair. This pair will be pushed into this list if its correlation coefficient is greater than τ. If the upper bound is below τ, this branch will be removed from the token-ring because the upper bounds of the uncalculated pairs in this branch must be less than τ according to the 1-dimensional property. In addition, a branch will also be removed from the token-ring if all the pairs in this branch are calculated. The algorithm

	I_1	I_2	I_3	I_4	I_5	I_6
I_1		15	14	13	8	5
I_2			36	23	14	9
I_3				48	31	19
I_4					49	31
I_5						45
I_6						

Table 6: Correlation upper bound of the coined data

	I_1	I_2	I_3	I_4	I_5	I_6
I_1		15	14	3	-2	1
I_2			36	3	12	0
I_3				48	2	10
I_4					36	29
I_5						20
I_6						

Table 7: Correlation of the coined data

will stop when there is no branch in the token-ring. According to the way token-ring search works, the upper bound of all the pruned pairs must be less than the current minimum correlation coefficient in the top-k list, so the current top-k list contains the pairs with the k highest correlation values in the data set.

We coined a data set with 6 items. The correlation upper bounds and correlations of all the pairs are shown in Tables 6 and 7 respectively. An example of the Token-Ring search for the top-5 pairs in this data set is shown as follows. After traveling the first diagonal from $\{I_1, I_2\}$ to $\{I_5, I_6\}$, all the five pairs are pushed into the top-5 list, and the current minimum correlation coefficient in the top-5 list is 15. Since the current $\tau = 15$ is less than the maximal upper bound in the current diagonal 49, we continue the search. When checking $\{I_1, I_3\}$, the upper bound 14 is less than the current minimum correlation coefficient in the top-5 list 15, so the first branch quits the Token-Ring. The upper bound of $\{I_2, I_4\}$ is 23 which is greater than 15, but its correlation is less than 15. Therefore, the second branch stays in the Token-Ring and the current top-5 list is not changed. A similar thing happens to $\{I_3, I_5\}$. After checking $\{I_4, I_6\}$, the current minimum correlation coefficient in the top-5 list is 20 and the maximal upper bound in the current diagonal is 31. Therefore, we continue the search in the third diagonal. Branch 1 is already pruned, so $\{I_1, I_4\}$ won't be checked. We only check $\{I_2, I_5\}$ and $\{I_3, I_6\}$. After traveling the third diagonal, the current minimum correlation coefficient in the top-5 list is 20 and the maximal upper bound in the current diagonal is 19. Finally, we stop the search at cell $\{I_3, I_6\}$.

Theorem 4. *Both TOP-COP and Token-Ring calculate the same number of correlations. The only performance difference between TOP-COP and Token-Ring is that Token-Ring reduces the number of correlation upper bound computations.*

Proof. The correlation upper bound check for the cell $\{I_i, I_{i+j}\}$ in the i-th branch and the

j-th diagonal line can be avoided if the correlation upper bound of the cell $\{I_i, I_{i+j-1}\}$ in the i-th branch and the $(j-1)$-th diagonal line is below the minimum correlation coefficient in the top-k list at that time. Therefore, Token-Ring avoids the upper bound checks of TOP-COP for those pairs that cannot affect the current top-k list, but the current top-k list changes at the same cells for both Token-Ring and TOP-COP. In all, both TOP-COP and Token-Ring calculate the same number of correlations, but they check a different number of correlation upper bounds. □

Theorem 5. *For the Token-Ring algorithm, the difference between the number of computed correlation upper bounds and the number of computed correlations is no more than $n-1$.*

Proof. For the Token-Ring algorithm, there are $n-1$ branches at the beginning. If the i-th branch is pruned because all the pairs in this branch are calculated, we calculate the same number of upper bounds as correlations. If the i-th branch is pruned because the upper bound is lower than the current minimum correlation coefficient in the top-k list, we calculate one more upper bound than correlations for this branch. Therefore, the difference between the number of computed correlation upper bounds and the number of computed correlations is no more than $n-1$. □

3.1.3 Combination of Two Tasks

The 2-dimensional search can efficiently find correlated pairs above θ, but it is difficult for users to provide an appropriate correlation threshold. When using the Token-Ring algorithm to find the top-k correlated pairs, we could spend a lot of time on calculation and end up with trivial results in a data set that doesn't contain strong positive correlations. In the extreme case, any item in a transaction appears alone, which means there is no positive correlation for any pair. Both TOP-COP and Token-Ring will calculate the correlations of all the pairs and end up with the trivial top-k correlated pairs. In order to solve this problem, we propose a user procedure to combine 2-dimensional search and Token-Ring search which can stop searching wisely on its way finding the top-k correlated pairs when there are few strong positive correlations in the data set.

Before we try to find the correlated pairs, we can calculate the number of candidates for a given threshold and use this number to estimate the time we will spend. For example, if we hope the algorithm is completed in one day and the time for processing one candidate is 20ms, then we should choose the threshold under which the number of candidates is less than 4 million. Due to the existence of 2-dimensional search, we can get the number of candidates for any threshold in the linear time $O(n)$. We can choose the lowest threshold θ_{min} under which the number of candidates is less than 4 million, and then search the correlated pairs by using 2-dimensional search. The advantage is that the algorithm will stop on time if the data set contains few strong positive correlations. However, there are two disadvantages. First, we might retrieve too many pairs and we are only interested in the top-k correlated pairs. Second, we will definitely spend one day to search the correlated pairs, while we might only need to spend one hour to find the top-k correlated pairs by using the Token-Ring algorithm. Instead of searching the correlated pairs above θ_{min} by the 2-dimensional search algorithm, we integrate the threshold θ_{min} into the Token-Ring algorithm. We treat all the $n-1$

branches as nodes in the token-ring. Only the branch which gets the token can calculate the upper bound of the leftmost uncalculated pair in this branch and compare its upper bound against the value $\psi = max\{\theta_{min}, \tau\}$ where τ is the current minimum correlation coefficient in the top-k list. If the upper bound is above ψ, we will check the correlation value of this pair. This pair will be pushed into this list if its correlation coefficient is greater than ψ. If the upper bound is below ψ, this branch will be removed from the token-ring. The algorithm will stop when there is no node in the token-ring. In that way, even if the current minimum correlation coefficient τ in the top-k list never exceeds the threshold θ_{min}, the algorithm will process all the candidates under the threshold θ_{min} which won't cost more than the maximal time we allow. It stops searching wisely on its way to find the top-k correlated pairs when there are few strong positive correlations in the data set. If there are a lot of strong positive correlations in the data set, the algorithm will find the top-k list and stop. For example, if we try to find the top-5 pairs and set up the threshold 40 in the coined data set shown in Tables 6 and 7, the search sequence is as follows: $\{I_1, I_2\}$, $\{I_2, I_3\}$, $\{I_3, I_4\}$, $\{I_4, I_5\}$, $\{I_5, I_6\}$, $\{I_3, I_5\}$, and $\{I_4, I_6\}$. We end up with the only strong pair $\{I_3, I_4\}$.

3.2 Correlated Itemset Search

In a large dataset, it is impossible to calculate correlation for all the possible itemsets in order to retrieve the top-k correlated itemsets or correlated itemsets above a threshold. Even the best measures, such as leverage and likelihood ratio, only penalize itemsets with independent items, but cannot detect whether a given itemset contains items that are independent of the others. Therefore, we make use of the fully-correlated itemset framework [10] for correlated itemset search. Given a fully-correlated itemset S, if no other item can be added to generate a new fully-correlated itemset, then S is a *maximal fully-correlated itemset*. MFCIs provide more compact information for FCI as maximal frequent itemset for frequent itemset.

When going through the Apriori procedure, we will discard the $(k-1)$-level information after we generate the k-level information. However, if we generate a fully-correlated k-itemset given a correlation threshold θ and keep that information, we know that this itemset satisfies any correlation threshold lower than θ. To achieve that, we build an enumeration tree to save the fully-correlated value for all the MFCIs under a given initial correlation threshold. We can either efficiently retrieve the desired MFCIs for any given threshold above the initial threshold or incrementally grow the tree if the given threshold is below the initial threshold.

Definition 2. *Given an itemset S, the fully-correlated value $\rho(S)$ for this itemset is the minimal correlation value of all its subsets.*

Given an itemset S and the correlation threshold θ, if the current correlation threshold $\theta \leq \rho(S)$, the current itemset is a fully-correlated itemset because the correlation of any subset is no less than θ, i.e., any subset is highly correlated. If the current correlation threshold θ is greater than $\rho(S)$, the current itemset is not a fully-correlated itemset because the correlation of at least one subset is lower than θ, i.e., at least one subset is uncorrelated.

Theorem 6. *Given a k-itemset S ($k \geq 3$), the fully-correlated value $\rho(S)$ for this itemset*

is the minimal value α among its correlation value and the fully-correlated values for all its $(k-1)$-subsets.

Proof. For any true subset SS of the current k-itemset S, SS must be the subset of at least one of all the $(k-1)$-subset of the current itemset S. Among all the subsets of S, either S itself or one true subset SS has the minimal correlation value β.

(i) If the S itself has the minimal correlation value β, then the correlation of any true subset SS is no less than β. According to the definition of fully-correlated values, $\rho(S) = \beta$ and $\rho((k-1)\text{-}Subset_i) \geq \beta$. According to the definition of α in this theorem, $\alpha = \beta$. Therefore, $\rho(S) = \alpha$.

(ii) If one true subset SS has the minimal correlation value β, then the correlation of S and any true subset is no less than β. According to the definition of fully-correlated values, $\rho(S) = \beta$ and $\rho((k-1)\text{-}Subset_i) \geq \beta$. Since SS must be the subset of at least one of all the $(k-1)$-subset of the current itemset S, $\rho((k-1)\text{-}Subset_j) = \beta$ at least for one j. According to the definition of α in this theorem, $\alpha = \beta$. Therefore, $\rho(S) = \alpha$. □

By making use of the above theorems, we can calculate the fully-correlated value for each itemset by Algorithm 1.

Algorithm 1 Find the Fully-Correlated Value

```
Main: CalculateFullyCorrelatedValue()
  ρ(pairs) = CalCorrelation(pairs);
  for k = 3; k ≤ n; k++ do
    for each possible k-itemset C do
      for i = 1; i ≤ k; i++ do
        θ_i is the fully-correlated value of the i-th (k-1)-subset of C
      end for
      ρ(C) = min{CalCorrelation(C), θ_1, θ_2, ..., θ_k}
    end for
  end for
```

An enumeration tree will be used to store the fully-correlated value. This enumeration tree is commonly used for itemset search, and more details can be found in [5]. If the fully-correlated value of the current node is less than a given threshold θ, the fully-correlated value of any descendant of the current node is also lower than θ which can be avoided when traversing the tree. Then we can easily traverse this tree to retrieve all possible fully-correlated itemsets given a threshold. Since finding the maximal fully-correlated itemsets from the enumeration tree saving fully-correlated value information is exactly the same as finding the maximal frequent itemsets from the enumeration tree saving support information, we can apply any technique of searching maximal frequent itemsets to MFCI search, such as MAFIA [8], Max-Miner [4], and GenMax [14]. Here, we use MAFIA to find the MFCIs given a threshold.

The current enumeration tree generation from the previous enumeration tree Given threshold θ, we can save all the fully-correlated itemsets with FCV greater than θ in this enumeration tree structure. Therefore, we can modify the original MFCI algorithm as follows. Given the threshold θ, instead of only getting the fully-correlated itemsets for each level, we keep all the candidates generated from lower levels and their fully-correlated

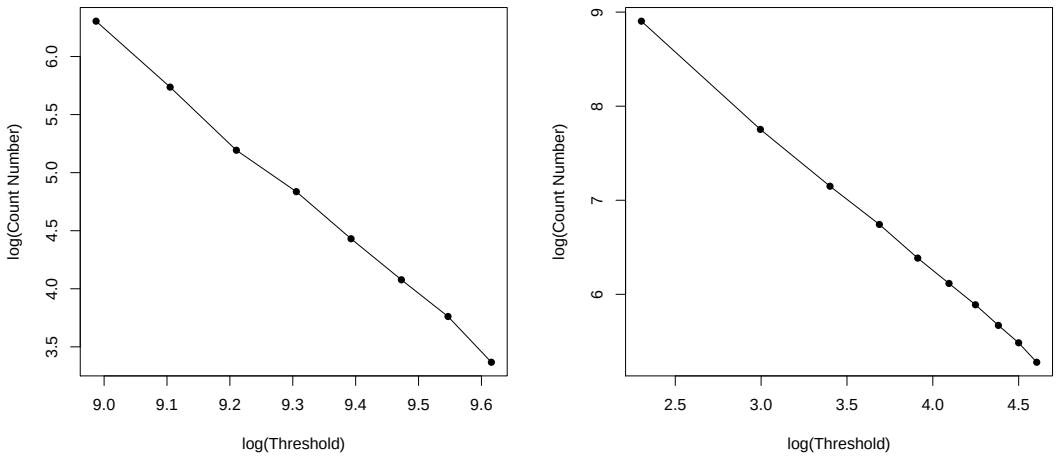
values in the enumeration tree T. Although we keep all the candidates generated from low levels, we only use the fully-correlated itemsets instead of the candidates in the current level to generate the next-level candidates. For any threshold θ_1 greater than θ, we can easily get the corresponding fully-correlated itemsets and the corresponding enumeration tree T_1 by traversing the current enumeration tree T. For a threshold $\theta_2 < \theta$, the current enumeration tree T is a subtree of the target enumeration tree T_2. In this case, we only need to increment the current enumeration tree T instead of building the target enumeration tree T_2 from scratch. To generate k-itemset candidates from fully-correlated $(k-1)$-itemsets, the candidates generated by the fully-correlated itemsets whose fully-correlated value is greater than θ have already been saved in the enumeration tree T. In order to generate the remaining candidates, we generate the candidates involving at least one $(k-1)$-itemset whose fully-correlated value is between θ and θ_2 to increment the enumeration tree T. In this way, we can generate the target enumeration tree T_2 and avoid repeating calculations that were made when building the enumeration tree T.

User Interaction Procedure The best way to get the desired MFCIs is to select a relatively low threshold θ to build a corresponding enumeration tree to keep the fully-correlated values first. Then for any threshold above θ, we can easily generate the corresponding MFCIs from this enumeration tree. The core problem is how to determine this relative low threshold θ. If the threshold is too high, there might be some MFCIs which we are interested in but are not contained in the enumeration tree. We have to do extra work to increment the current enumeration tree. If the threshold is too low, we will keep much unnecessary information and might not have enough memory to generate or save the enumeration tree. Therefore, several proper user interaction procedures are needed in order to select the proper threshold.

We split the enumeration tree construction into two steps. First, we keep relevant pair correlation information for a relatively low threshold θ. By doing that, we can get the satisfied pairs for any threshold higher than θ by a simple query. Second, we construct the enumeration tree by trying different thresholds. Using the 2-dimensional search algorithm, we can easily count the number of pair candidates whose correlation upper bound is above a tentative threshold. Since we need to retrieve the support of all the pair candidates to calculate their correlation, we can estimate the time we will spend in the first step by the count for any given threshold. We will choose the threshold under which the computational time is affordable to us. For the second step, it is hard to estimate the time we will spend for any given threshold. The best way to finish the second step is to choose a relatively high threshold first, and then gradually grow the enumeration tree by lowering the threshold until we run out of time to update the tree for a threshold.

4 Experiments

The efficiency performance of our methods was tested on several real-life datasets. Since the results were similar for all the datasets, we only present results on two typical datasets. The first is the Netflix data set which contains 17,770 movies and 480,000 transactions. The second is a retail data set from the FIMI repository containing 16,470 items and 88,000



(a) Netflix (b) Retail

Figure 1: The number of correlated pairs under different likelihood ratio thresholds

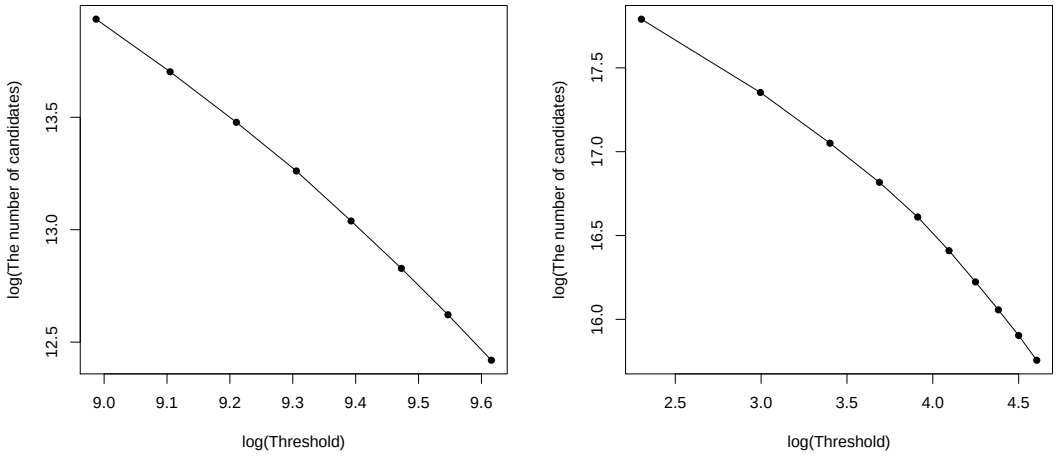
transactions. Netflix contains many correlated patterns because of movies in the same series and TV shows of many episodes, while the retail data set contains fewer correlated patterns because those highly correlated items might be already offered by manufactures as a package. The number of correlated pairs under different likelihood ratio thresholds for these two data sets is shown in Figure 1. We transformed both x-axis and y-axis into the log scale, and it is easy to see that the number of correlated pairs and the likelihood ratio threshold follow a power-law distribution. We implemented our algorithm using Java 1.6.0 on a Dell workstation with 2.4GHz Dual CPU and 4G memory running on the Vista operating system. In the following, we will check the performance improvement from each algorithm.

4.1 Correlated pair search

4.1.1 Finding correlated pairs above a certain threshold

Here, we focus on task 1 of correlated pair search: finding correlated pairs above a given threshold θ.

Count the number of pair candidates Before we determine the threshold to search the satisfied pairs, we can count how many pairs' CUBs are above a tentative threshold. This number can help us to estimate the time we will spend on the satisfied pair search because the support of the pair needs to be retrieved if its CUB is above the threshold. The number of candidates and the time spent on counting candidates under different thresholds are shown in Figure 2 and 3 respectively. Figure 2 is transformed to the log-log scale. The number of candidate and the likelihood ratio threshold roughly follow a power-law distribution. To

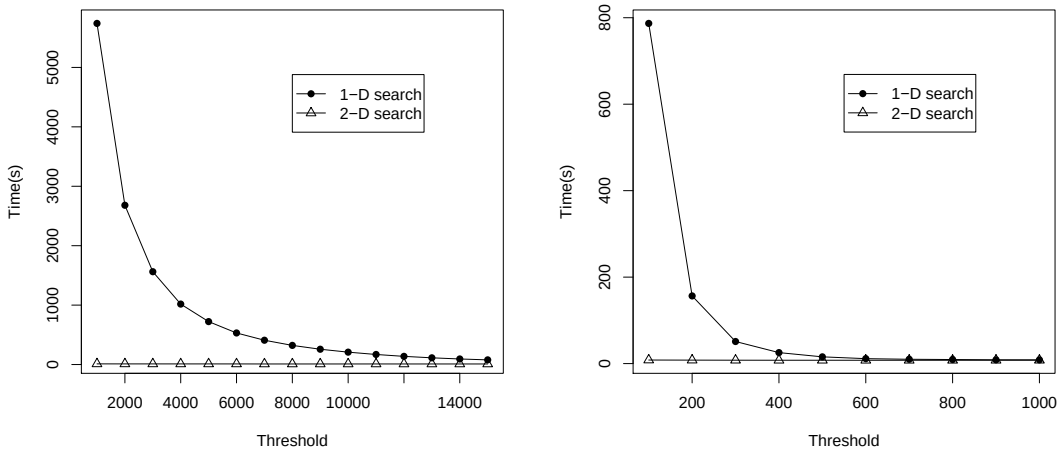


(a) Netflix (b) Retail

Figure 2: The number of candidates under different thresholds

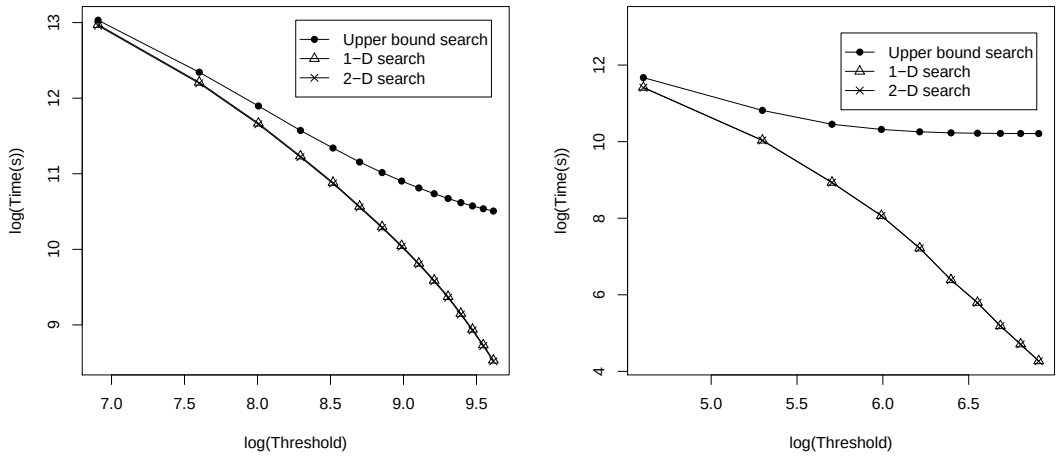
get the number of pair candidates, 2-dimensional search is linear and 1-dimensional search is exponential with the threshold. When the threshold is 0, the 1-dimensional search will check all the pairs. Therefore, the runtime of 1-dimensional search with threshold 0 is equal to the runtime of the upper bound calculation method. The 1-dimensional search takes less time than the upper bound calculation method when the threshold is high, but the 2-dimensional search is always an order of magnitude faster than the just upper bound calculation method.

Search the satisfied pairs Since the IO cost for calculating correlation of pairs is much higher than the time cost for calculating CUB, the CUB calculation can save a lot of unnecessary IO cost when the threshold is high. The log runtime of retrieving the satisfied pairs by calculating CUB first given different log correlation thresholds is shown in Figure 4. The 1-dimensional and 2-dimensional search are almost identical. The runtime of the CUB, 1-dimensional, and 2-dimensional search all decreases drastically as we increase the threshold. Both 1-dimensional and 2-dimensional search take much less time than the CUB search when the threshold is high because 1-dimensional and 2-dimensional search take less time to find the pairs whose CUB is higher than the threshold. For the dataset which contains less correlation, the correlation upper bound check for high thresholds dominates the calculation. In that case, the speed-up of 1-dimensional and 2-dimensional search comparing to the CUB search is more obvious like in the retail dataset. However, the CUB, 1-dimensional, and 2-dimensional search do not make too much difference when the threshold is low.



(a) Netflix (b) Retail

Figure 3: The runtime of determining the number of candidates under different thresholds



(a) Netflix (b) Retail

Figure 4: The runtime for retrieving the satisfied pairs

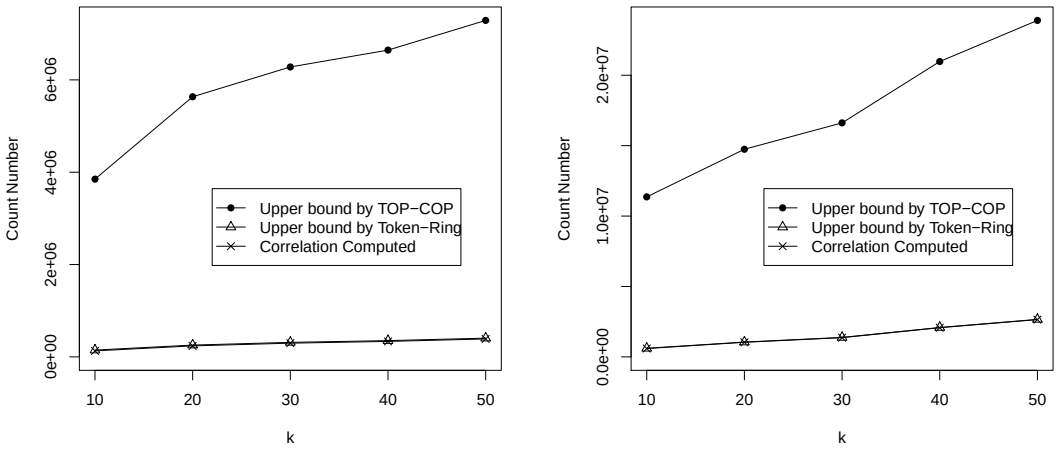
4.1.2 Finding top-k correlated pairs

For finding the top-k correlated pairs, the runtime of the brute-force method is determined by the number of correlations we need to compute, while the runtime of TOP-COP or Token-Ring is determined by the number of correlations and the number of CUBs we need to compute. The brute-force method will calculate the correlation of all the possible pairs which is not feasible for large datasets. According to Theorem 4, Token-Ring computes fewer CUBs than TOP-COP, but computes the same number of correlations. Therefore, the runtime difference between Token-Ring and TOP-COP is determined by the difference between the number of CUB calculations. The number of correlation calculations and CUB calculations under different k is shown in Figure 5. According to Theorem 5, the number of correlation calculations is almost equal to that of CUB calculations in the Token-Ring algorithm which is verified in Figure 5. Token-Ring can save a huge number of unnecessary CUB checks compared to TOP-COP. The runtime of TOP-COP and Token-Ring for top-k correlated pairs is shown in Figure 6. When the IO cost of retrieving pair support is much more expensive than the CUB calculation for a dense dataset like Netflix, the total runtime is dominated by the time spent on correlation calculation and there is little difference between Token-Ring and TOP-COP. When the IO cost is not that expensive compared to the CUB calculation for a sparse dataset like retail, we can see Token-Ring took significantly less time than TOP-COP. The advantage of Token-Ring over TOP-COP is obvious only for datasets where the gap between the true correlation and the correlation upper bound is large.

The top-k search method is more flexible for finding the desired patterns compared to the 2-dimensional method. Assuming we can get the top-k pair correlation ahead of time and use it to search the top-k pairs with the 2-dimensional method, Figure 7 shows the number of correlation calculations using 2-dimensional search and those using our top-k search method. The gap between the 2-dimensional method and the top-k search method is not huge and the gap gradually increases with the k. The top-k search method is more flexible and does not require much more time than the threshold search method.

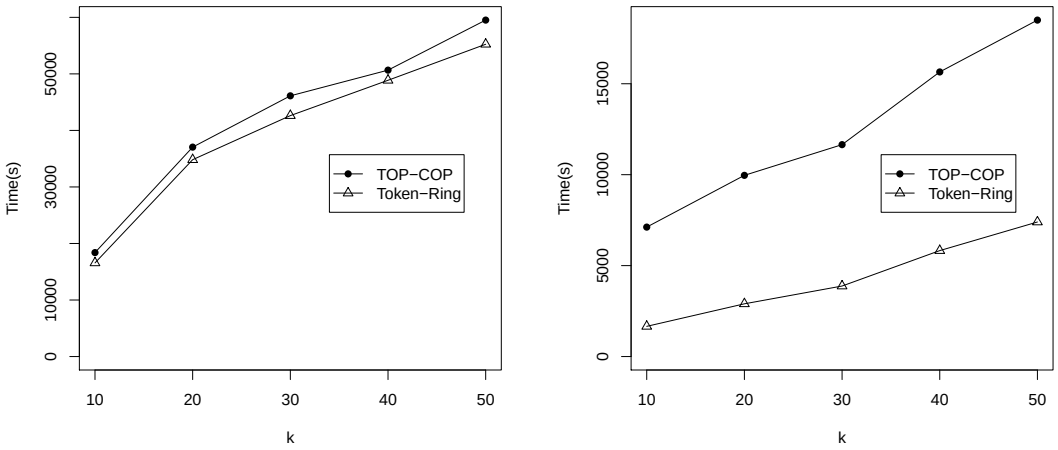
4.1.3 Combination of Two Tasks

We proposed a user procedure to combine the 2-dimensional search algorithm and the Token-Ring algorithm to reconcile the two tasks of searching for pairs above a certain threshold and for top-k pairs. The performance of the user procedure really depends on the parameters we set. If the threshold θ is so high that the k-th correlation in the dataset is less than θ, the runtime is equal to the 1-dimensional search under the threshold θ because the modified Token-Ring checks the same number of correlations and the same number of CUBs as the 1-dimensional search algorithm. If k is small and the data set contains many strong positive correlations, the runtime is close to the original Token-Ring algorithm. From Figures 4 and 6, we can easily estimate the runtime of the modified Token-Ring for different parameters. For example, when the threshold θ is 1000 and k is 10 in Netflix, the runtime will be close to 18,000 seconds.



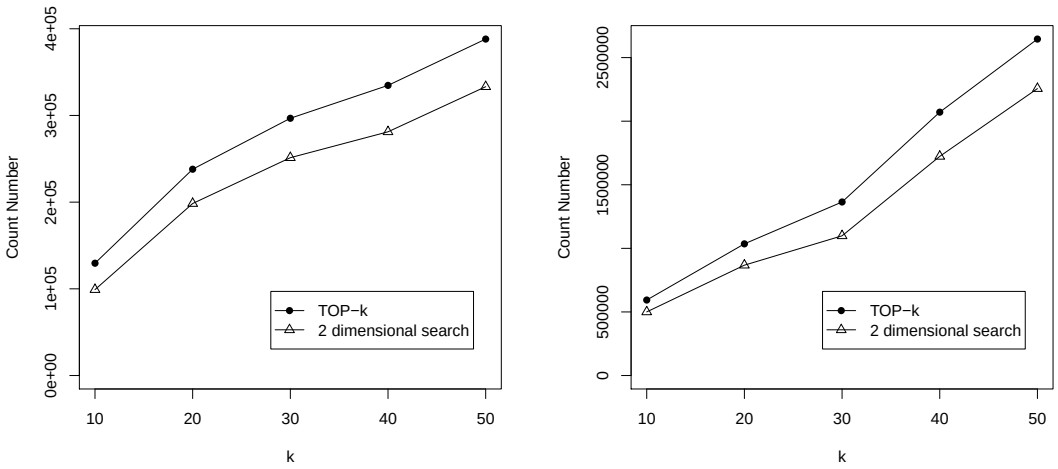
(a) Netflix (b) Retail

Figure 5: The number of correlation and correlation upper bound checks



(a) Netflix (b) Retail

Figure 6: The runtime for top-k algorithms



(a) Netflix

(b) Retail

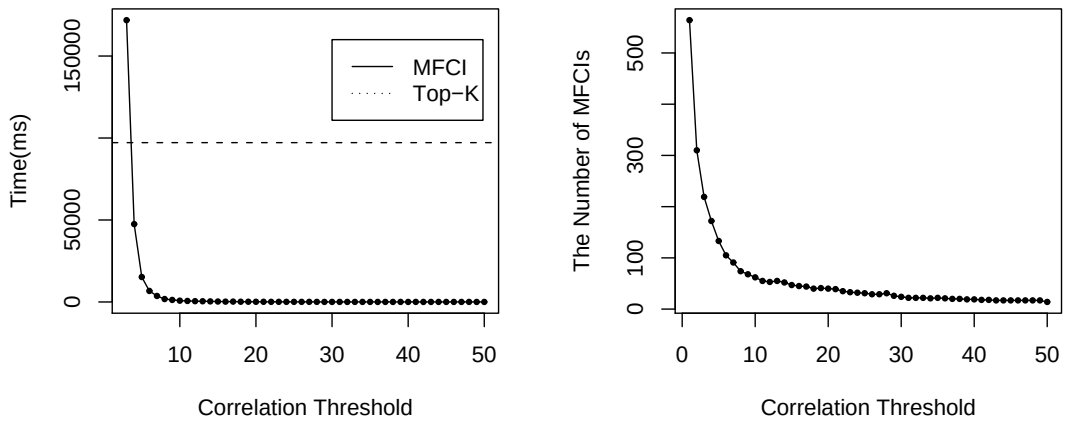
Figure 7: Correlation checks for top-k search and threshold search

4.2 Correlated itemset search

4.2.1 Maximal Fully-correlated Itemset Search

Since it is impossible to find the top-k correlated itemsets on both datasets and we want to compare the top-k correlated itemsets with MFCIs, we use a subset of the Netflix dataset which only contains the first 100 movies according to the code sequence in the Netflix dataset and test on it. The performance of our algorithm depends on the characteristic of the data set. In the extreme case, among all the n items in the data set, if the first $(n-1)$ items always show up together and the remaining item appears alone, our algorithm will start with $\binom{n-1}{2}$ 2-itemsets and end with the only $(n-1)$-maximal fully-correlated itemset. In other words, all the $2^{(n-1)}$ possible combinations will be checked. It is still an NP-hard problem. But in reality, most data sets are sparse, so most of the search space can be pruned.

The runtimes of our algorithm on the Netflix subset data given different correlation thresholds are shown in Figure 8(a). The runtime decreases drastically as we increase the threshold. The runtime of the top-k method is also shown. When running the top-k method, we only checked the itemsets which occurred at least once in the dataset, so about 2 million itemsets were checked instead of all 2^{100} possible itemsets. In the worst case, our algorithm checks all the itemsets that occur at least once. Therefore, the runtime of our algorithm is at least as good as the top-k method if both use Apriori. Even though we use the prefix tree structure [5] to find the top-k correlation itemsets, the runtime of our method is less than the top-k method when the threshold is larger than 3. In addition, the number of maximal fully-correlated itemsets corresponding to each correlation threshold is shown in Figure 8(b). It shows even if we use a small threshold like 1, we still get a relatively small number of compact itemsets.



(a) Runtime for each threshold

(b) MFCIs for each threshold

Figure 8: Performance results for Netflix

Besides the gain from the efficiency side, we can also make use of the characteristics of the Netflix dataset to evaluate the effectiveness of the MFCI framework. Like maximal frequent itemsets, there is no ranking of maximal fully-correlated itemsets. Rather, we only get a different number of them under different thresholds. Therefore, we try to find the threshold under which only 15 MFCIs are retrieved to compare with top-20 correlation itemsets which are retrieved by calculating every possible combination. The 15 maximal fully-correlated itemsets are shown in Table 8, and the top-20 correlated sets are listed in Table 9. Our maximal fully-correlated itemsets have several advantages over the top-20 correlated sets. First, some top-k correlated sets are redundant since they are subsets of other top-k correlated sets. For example, the first correlated set of Netflix is a subset of the second one. There is no need to show redundant information. Second, some top-k correlated itemsets contain irrelevant information. Among all the top-20 correlated itemsets, 16 of them are subsets of the 15 maximal fully-correlated itemsets. Four remaining itemsets all contain one more movie "Something's Gotta Give (2003)" than the corresponding maximal correlation sets. In fact, "Something's Gotta Give (2003)" is the most favored movie among all the 100 films. It almost has no correlation with any other movie. If we remove "Something's Gotta Give (2003)" from these 4 itemsets, we can get higher correlation values. For each of these four itemsets, if we treat "Something's Gotta Give (2003)" as set A and the remaining movies as set B, all the correlation values between A and B are close to 0 which means there is no correlation. Especially in the 17th correlation set, {Dragonheart (1996), Congo (1995)} and {Something's Gotta Give (2003)} are negatively correlated. The probability of favoring "Something's Gotta Give (2003)" is 55.28%. The conditional probability of favoring "Something's Gotta Give (2003)" given {Dragonheart (1996), Congo (1995)} is 47.02%. Since favoring "Dragonheart (1996)" and "Congo (1995)" will decrease the probability of

favoring "Something's Gotta Give (2003)", putting them together is not a wise choice. As mentioned above, the top-k method contains some redundant itemsets. One way to solve this problem is to only keep the itemsets that do not have any superset in the top-k list. However, because of the existence of irrelevant items like "Something's Gotta Give (2003)", the itemsets that have no superset within the top-k itemsets might contain irrelevant items. The maximal fully-correlated itemsets are more reasonable than the top-k correlated itemsets. We get similar patterns by testing other correlation measures. Some of the movies in MFCIs do not seem to go together due to the weak connection among a small set of items. However, if MFCIs are considered bad, the top-k correlated itemsets are even worse. Since it is impossible to compare MFCI with the top-k correlated itemsets on the whole dataset, we instead chose the traditional maximal frequent itemsets to compare with. We tried different thresholds and stopped when only five patterns are retrieved. Table 10 shows the 5 maximal frequent itemsets and the 5 MFCIs on the whole Netflix dataset. We naively assume that movies and TV shows in the same series are more correlated. MFCIs get better results than maximal frequent itemsets.

ID	Maximal Fully-Correlated Itemsets
1	Character (1997), Mostly Martha (2002)
2	My Favorite Brunette (1947), The Lemon Drop Kid (1951)
3	7 Seconds (2005), Never Die Alone (2004)
4	Immortal Beloved (1994), Richard III (1995)
5	Aqua Teen Hunger Force: Vol. 1 (2000), Invader Zim (2004)
6	Rudolph the Red-Nosed Reindeer (1964), Jingle All the Way (1996)
7	The Bad and the Beautiful (1952), The Killing (1956)
8	Richard III (1995), The Killing (1956)
9	Dragonheart (1996), Jingle All the Way (1996)
10	The Rise and Fall of ECW (2004), WWE: Armageddon (2003), WWE: Royal Rumble (2005)
11	The Rise and Fall of ECW (2004), WWE: Royal Rumble (2005), ECW: Cyberslam '99 (2002)
12	Screamers (1996), Dragonheart (1996), Congo (1995)
13	Immortal Beloved (1994), Spitfire Grill (1996), Silkwood (1983)
14	Lilo and Stitch (2002), Justice League (2001), The Powerpuff Girls Movie (2002)
15	Lilo and Stitch (2002), Dragonheart (1996), Congo (1995)

Table 8: Maximal fully-correlated itemsets from Netflix subset using likelihood ratio

4.2.2 Improvement from the enumeration tree structure

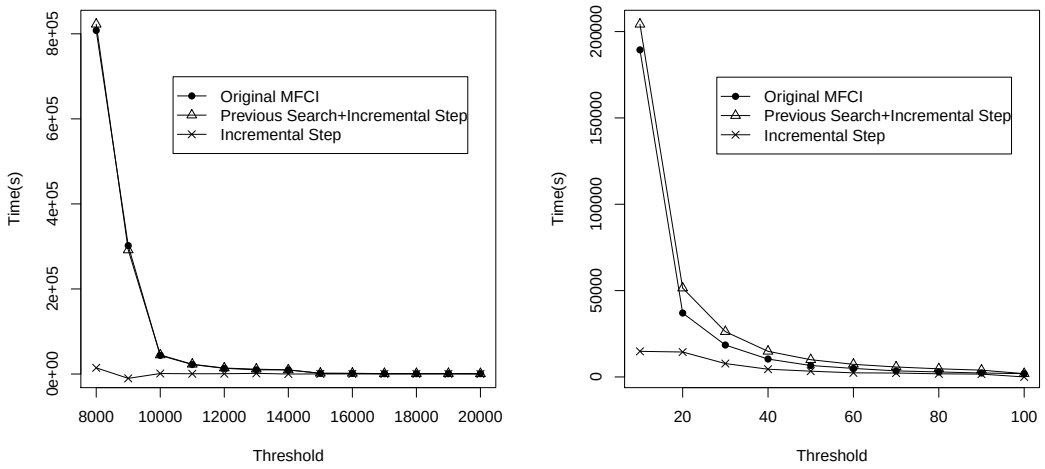
The performance improvement for MFCI comes from the improved pair search and the enumeration tree structure. We have checked the improvement from the improved pair search. In the following, we check the improvement from the enumeration tree structure. The fully-correlated values saved in the enumeration tree benefit the next search round. We need to build the tree first, and then handily retrieve MFCIs according to fully-correlated values saved in the enumeration tree.

Ranking	Correlated Itemset
1	Dragonheart (1996), Congo (1995)
2	Lilo and Stitch (2002), Dragonheart (1996), Congo (1995)
3	The Rise and Fall of ECW (2004), WWE: Royal Rumble (2005)
4	Spitfire Grill (1996), Silkwood (1983)
5	My Favorite Brunette (1947), The Lemon Drop Kid (1951)
6	Immortal Beloved (1994), Spitfire Grill (1996), Silkwood (1983)
7	Screamers (1996), Dragonheart (1996), Congo (1995)
8	Lilo and Stitch (2002), Dragonheart (1996)
9	Lilo and Stitch (2002), Something's Gotta Give (2003), Dragonheart (1996), Congo (1995)
10	The Rise and Fall of ECW (2004), WWE: Royal Rumble (2005), ECW: Cyberslam '99 (2002)
11	Aqua Teen Hunger Force: Vol. 1 (2000), Invader Zim (2004)
12	Something's Gotta Give (2003), Spitfire Grill (1996), Silkwood (1983)
13	The Bad and the Beautiful (1952), The Killing (1956)
14	Rudolph the Red-Nosed Reindeer (1964), Jingle All the Way (1996)
15	Immortal Beloved (1994), Richard III (1995)
16	Immortal Beloved (1994), Something's Gotta Give (2003), Spitfire Grill (1996), Silkwood (1983)
17	Something's Gotta Give (2003), Dragonheart (1996), Congo (1995)
18	The Rise and Fall of ECW (2004), WWE: Armageddon (2003), WWE: Royal Rumble (2005)
19	The Rise and Fall of ECW (2004), ECW: Cyberslam '99 (2002)
20	Screamers (1996), Dragonheart (1996)

Table 9: Top-20 correlated itemsets for Netflix subset using likelihood ratio

ID	Maximal Frequent Itemsets	Maximal Fully-Correlated Itemsets
1	Forrest Gump (1994) The Green Mile (1999)	Harry Potter (2001) Harry Potter (2002)
2	Forrest Gump (1994) Shawshank Redemption (1994)	Kill Bill 1 (2003) Kill Bill 2 (2004)
3	Pirates of the Caribbean (2003) The Lord of the Rings (2001)	The Lord of the Rings (2001) The Lord of the Rings (2002) The Lord of the Rings (2003)
4	Pirates of the Caribbean (2003) The Lord of the Rings (2002)	The Sopranos: Season 1 (1999) The Sopranos: Season 2 (2000) The Sopranos: Season 3 (2001)
5	The Lord of the Rings (2001) The Lord of the Rings (2002) The Lord of the Rings (2003)	Star Wars IV (1977) Star Wars V (1980) Star Wars VI (1983)

Table 10: Maximal Fully-Correlated Itemsets with Different Measures



(a) Netflix when the threshold gap is 1000 (b) Retail when the threshold gap is 10

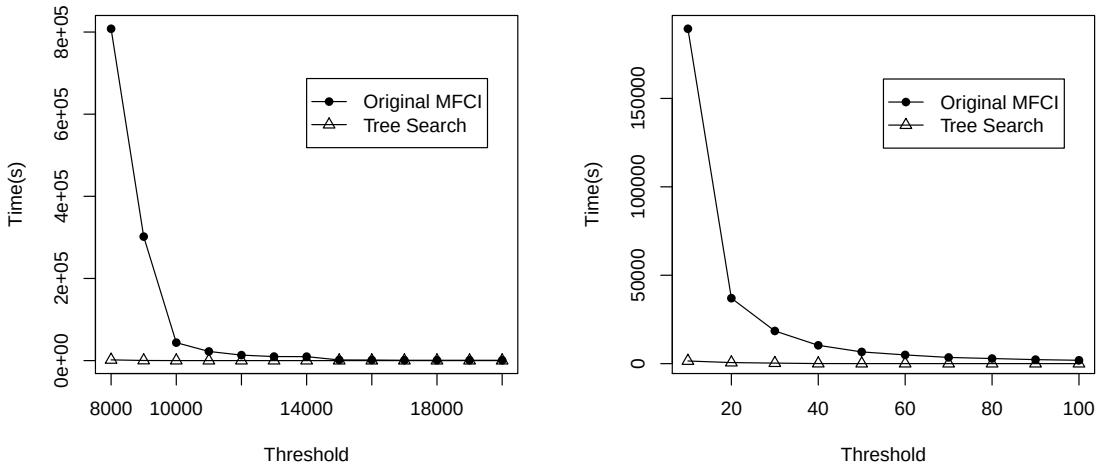
Figure 9: The runtime of building the enumeration tree

4.2.3 Build the enumeration tree

Since it is hard to estimate the time we will spend to build the enumeration tree for any given threshold, the best way to build the enumeration tree is to choose a relative high threshold first, and then gradually increment the enumeration tree by lowering the threshold. Given the threshold θ and the threshold gap g, the time to build the enumeration tree T for the threshold θ is a, the time to build the enumeration tree T_1 for the threshold $(\theta + g)$ is b, and the time for incrementing the tree from T_1 to T is c. Therefore, the runtime of the original MFCI is a, and the total runtime of the incremental algorithm is $(b + c)$. The runtime of the original MFCI and the incremental algorithm given the threshold gap g and different thresholds are shown in Figure 9. The total runtime of these two algorithms is very close, but the incremental algorithm is more flexible for user interactions.

4.2.4 Generate MFCIs from the enumeration tree

The enumeration tree saving fully-correlated values has exactly the same downward-closed property of the enumeration tree saving support, so we can apply any search technique for maximal frequent itemsets. Here, we only show how much improvement we get if we use MAFIA to generate MFCIs from the enumeration tree. Figure 10 shows the runtime of the original MFCI algorithm and the MFCI generation from the enumeration tree. The MFCI generation from the enumeration tree is much faster than the original MFCI algorithm.



(a) Netflix

(b) Retail

Figure 10: The runtime for generating MFCIs

4.2.5 User Interaction Procedure

If the goal is to build a relatively large enumeration tree which can facilitate the search of MFCIs under different thresholds, we use the following procedure. First, 2-dimensional search is used to evaluate the threshold θ under which time allows to retrieve all the correlated pairs. Second, CUB search is used to find the correlated pairs and save their correlation value in the database. Third, we choose a relatively high threshold, and then gradually update the enumeration tree by lowering the threshold until we run out of time to grow the tree or lowering the threshold. After that, we can easily retrieve MFCIs under any threshold above the threshold of the enumeration tree.

If the goal is to identify the highly correlated patterns, we can combine 2-dimensional search with the incremental algorithm for the enumeration tree. From Figure 4 and 9, we expect the runtime saving mainly comes from the 2-dimensional search.

5 Conclusion

In the paper, we propose an FCI framework to decouple the correlation measure from the need for efficient search. By wrapping the desired measure in our FCI framework, we take advantage of the desired measure's superiority in evaluating itemsets, make use of the property of FCI to eliminate itemsets with irrelevant items, and still achieve good computational performance. Further, we facilitated search for both pairs and itemsets. For pairs, we can either use the 2-dimensional search to retrieve correlated pairs above a certain threshold, or our new Token-Ring algorithm to find top-k correlated pairs. For itemsets,

we build an enumeration tree to save the fully-correlated value. In that way, we can either efficiently retrieve the desired FCIs for any given threshold above the initial threshold or incrementally grow the tree if the given threshold is below the initial threshold.

References

[1] C. C. Aggarwal, J. Pei, and B. Zhang. On privacy preservation against adversarial data mining. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 510–516, New York, NY, USA, 2006. ACM.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 207–216, New York, NY, USA, 1993. ACM.

[3] A. Bate, M. Lindquist, I. R. Edwards, S. Olsson, R. Orre, A. Lansner, and R. M. De Freitas. A bayesian neural network method for adverse drug reaction signal generation. *European Journal of Clinical Pharmacology*, 54(4):315–321, 1998.

[4] R. J. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 85–93, New York, NY, USA, 1998. ACM.

[5] C. Borgelt. Efficient implementations of Apriori and Eclat. In *ICDM '03: Proc. 3rd IEEE Int. Conf. on Data Mining, Workshop on Frequent Itemset Mining Implementations*, 2003.

[6] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *SIGMOD '97: Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 265–276, New York, NY, USA, 1997. ACM.

[7] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD '97: Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 255–264, New York, NY, USA, 1997. ACM.

[8] D. Burdick. Mafia: A maximal frequent itemset algorithm for transactional databases. In *ICDE '01: Proceedings of the 17th International Conference on Data Engineering*, page 443, Washington, DC, USA, 2001. IEEE Computer Society.

[9] H. Cheng, X. Yan, J. Han, and C.-W. Hsu. Discriminative frequent pattern analysis for effective classification. In *ICDE'07*, pages 716–725, 2007.

[10] L. Duan and W. N. Street. Finding maximal fully-correlated itemsets in large databases. In *ICDM '09: Proc. Int. Conf. on Data Mining*, pages 770–775, Miami, FL, USA, 2009.

[11] L. Duan and W. N. Street. Selecting the right correlation measure for binary data. *Under review*, http://dollar.biz.uiowa.edu/∼ldn/research/paper/selecting_the_right_correlation_measure.fdf.

[12] T. Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993.

[13] L. Geng and H. J. Hamilton. Interestingness measures for data mining: A survey. *ACM Computing Surveys*, 38(3):9, 2006.

[14] K. Gouda and M. J. Zaki. Genmax: An efficient algorithm for mining maximal frequent itemsets. *Data Min. Knowl. Discov.*, 11(3):223–242, 2005.

[15] T. Guns, S. Nijssen, and L. De Raedt. Itemset mining: A constraint programming perspective. *Artif. Intell.*, 175:1951–1983, August 2011.

[16] C. Jermaine. Finding the most interesting correlations in a database: How hard can it be? *Information Systems*, 30(1):21–46, 2005.

[17] E. R. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):57–69, 2003.

[18] G. Piatetsky-Shapiro. *Discovery, Analysis, and Presentation of Strong Rules.* AAAI/MIT Press, 1991.

[19] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right objective measure for association analysis. *Information Systems*, 29(4):293–313, 2004.

[20] N. Tatti. Maximum entropy based significance of itemsets. *Knowl. Inf. Syst.*, 17:57–77, October 2008.

[21] G. I. Webb. Discovering significant patterns. *Mach. Learn.*, 68:1–33, July 2007.

[22] H. Xiong, M. Brodie, and S. Ma. TOP-COP: Mining top-k strongly correlated pairs in large databases. In *ICDM '06: Proc. sInt. Conf. on Data Mining*, pages 1162–1166, Washington, DC, USA, 2006.

[23] H. Xiong, S. Shekhar, P.-N. Tan, and V. Kumar. TAPER: A two-step approach for all-strong-pairs correlation query in large databases. *IEEE Trans. on Knowl. and Data Eng.*, 18(4):493–508, 2006.

[24] S. B. Yahia, T. Hamrouni, and E. M. Nguifo. Frequent closed itemset based algorithms: a thorough structural and analytical survey. *SIGKDD Explor. Newsl.*, 8:93–104, June 2006.

[25] J. Zhang and J. Feigenbaum. Finding highly correlated pairs efficiently with powerful pruning. In *CIKM '06: Proc. ACM CIKM Int. Conf. on Information and Knowledge Management*, pages 152–161, New York, NY, USA, 2006. ACM.