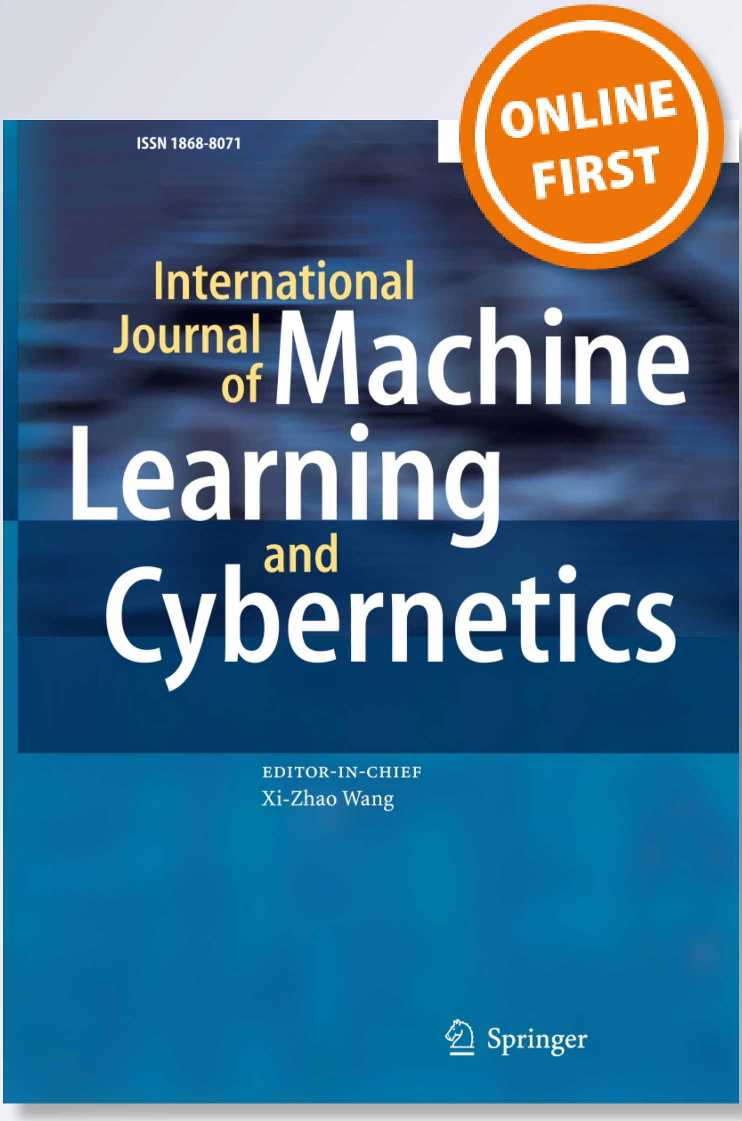
Speeding up maximal fully-correlated itemsets search in large databases

Lian Duan & W. Nick Street

International Journal of Machine Learning and Cybernetics

ISSN 1868-8071

Int. J. Mach. Learn. & Cyber.
DOI 10.1007/s13042-014-0290-9



Springer

Int. J. Mach. Learn. & Cyber.
DOI 10.1007/s13042-014-0290-9

ORIGINAL ARTICLE

Speeding up maximal fully-correlated itemsets search in large databases

Lian Duan · W. Nick Street

Received: 16 January 2014 / Accepted: 26 July 2014

Abstract Finding the most interesting correlations among items is essential for problems in many commercial, medical, and scientific domains. Our previous work on the maximal fully-correlated itemset (MFCI) framework can rule out the itemsets with irrelevant items and its downward-closed property helps to achieve good computational performance. However, to calculate the desired MFCIs in large databases, there are still two computational issues. First, unlike finding maximal frequent itemsets which can start the pruning from 1-itemsets, finding MFCIs must start the pruning from 2-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 can be very high. Second, users usually need to try different correlation thresholds for different desirable MFCIs. Therefore, the cost of processing the Apriori procedure each time for a different correlation threshold is also very high. Consequently, we proposed two techniques to solve these problems. First, we identify the correlation upper bound for any good correlation measure to avoid unnecessary IO query for the support of pairs, and make use of their common monotone property to prune many pairs even without computing their correlation upper bounds. In addition, 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. Experimental results show that our algorithm can be an order of magnitude faster than the original MFCI algorithm.

Keywords Correlation · Maximal fully-correlated itemset · Apriori

L. Duan (✉)
New Jersey Institute of Technology, Newark, NJ 07102, USA
e-mail: lian.duan@njit.edu

W. N. Street
The University of Iowa, Iowa City, IA 52242, USA
e-mail: nick-street@uiowa.edu

1 Introduction and related work

The analysis of relationships between items is fundamental in many data mining problems [9, 16, 23]. Most of the published work with regard to correlation is related to finding correlated pairs [6, 12, 20, 21]. Related work with association rules [3, 4, 18] is a special case of correlated pairs since each rule has a left- and right-hand side. 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. The first related technique is frequent itemset mining [17, 22]. 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 [20] compared 21 different measures for correlation. Only four of the 21 measures can be used to measure the correlation within a given k-itemset. Dunning [11] 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 itemsets with irrelevant items. Jermaine [15] extended Dunning's work and examined the computational issue of probability ratio and likelihood ratio. No existing correlation measures satisfying the three primary correlation properties [7, 10] have the downward-closed property to facilitate the search. To find the top-k correlated itemsets is a NP-hard problem. 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 problems, others proposed efficient correlation measure like all-confidence [18]. All-confidence is as fast as support; however, the correlation is still measured in a sub-optimal way [7]. 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 maximal fully-correlated itemsets [7] can not only decouple the correlation measure from the need for efficient search, but also rule out the itemsets with irrelevant items. With the MFCI framework, we only need to focus on effectiveness side when selecting correlation measures. However, to calculate the desired MFCIs, there are still two computational issues. First, unlike finding maximal frequent itemsets which can start the pruning from 1-itemsets, finding MFCIs must start the pruning from 2-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 can be high. Second, users usually need to try different correlation thresholds for different desirable MFCIs. For example, when we set up the threshold 15,000 by using likelihood ratio on 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 on 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 8,000. However, the cost of processing the Apriori procedure each time for a different correlation threshold is very high.

The remainder of this paper is organized as follows. Section 2 introduces basic concepts of correlation measures and MFCIs. As the search for MFCIs must start the pruning from pairs instead of single items, we briefly introduce our related published pair search speeding-up techniques [8] in Sect. 3. Section 4 introduces an enumeration tree structure to keep the fully-correlated value (FCV) for all the MFCIs of the previous search which can speed up the next MFCI search. The experiment results are shown in Sect. 5. Finally, we conclude the paper and suggest future directions.

2 Basic notions

In this section, some basic notions of our previous work on MFCI are introduced. It helps the understanding of how to improve the performance of the original MFCI algorithm. In addition, the math symbols used throughout this paper are summarized in Table 1.

2.1 Correlation measure properties

To find high-correlation itemsets, we should find a reasonable correlation measure first. Since it is impossible to compare against every possible measure [12, 21], in this paper we use the following three commonly accepted criteria to generalize the correlation measure, such as χ^2-statistic, probability ratio, leverage, and likelihood ratio.

Given an itemset $S = \{I_1, I_2, \ldots, I_m\}$, a correlation measure M must satisfy the following three properties [19]:

- 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.

In Sect. 3, we will make use of the above 3 properties to facilitate our correlated pair search. Without losing generality, we use the best correlation measure [7], likelihood ratio, which measures the ratio of the likelihood of k out of n transaction containing the itemset S when the single trial probability is the true probability, $tp = P(S)$, to that when the single trial probability is the expected probability, $ep = P(I_1) * P(I_2) * \ldots * P(I_m)$, if all the items in S are independent from each other to demonstrate our search in the rest of the paper.

2.2 Maximal fully-correlated itemsets

To the best of our knowledge, no correlation measure alone can either detect whether a given itemset contains items that are independent from the other items or have the

Table 1 Math symbols

Name	Description
S	An itemset
I_k	The k-th item in a given itemset
M	The function to measure the correlation of a given itemset
$P(S)$	The support of the itemset S
tp	The probability of a randomly selected transaction containing S
ep	The expected probability of a randomly selected transaction containing S under the assumption of independence
$P(I)$	The probability of a randomly selected transaction containing I
$\rho(S)$	The fully-correlated value of a given itemset defined in Sect. 4
T	An enumeration tree

Table 2 An independent case

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

downward-closed property. To achieve that, we use fully-correlated itemsets defined as follows.

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, i.e., any subset of S is closely correlated with all other subsets and no irrelevant items can be removed, this itemset S is a fully-correlated itemset.

This definition has three very important properties. First, it can be incorporated with any correlation measure. Second, it helps to rule out an itemset with independent items. For example in Table 2, 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 fully-correlated itemset should be $\{B, C\}$ whose likelihood ratio is 17.93. Third, there is a desirable downward closure property which can help us to prune unsatisfied itemsets quickly like Apriori [1]. Given a fully-correlated itemset S, if there is no other item that can be added to generate a new fully-correlated itemset, then S is a *maximal fully-correlated itemset*. MFCI shows more compact information.

3 Correlated pair search

Since finding MFCIs must start the pruning from 2-itemsets instead of 1-itemsets, a good correlated pair search algorithm can speed up the generation of MFCIs a lot. As we will make use of our related published pair search speeding-up techniques [8] here, we briefly introduce the related concept. More details can be found in the related paper.

3.1 A correlation upper bound search

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

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

3.2 1-Dimensional search

The upper bound search can save a lot of unnecessary IO cost; however, it needs to calculate the upper bound of all the possible $n * (n-1)/2$ pairs which is still computationally expensive. The 1-Dimensional search which can save a lot of unnecessary upper bound checkings for very high thresholds. In order to facilitate the search, we sort the items according to their supports in an increasing order.

Theorem 2 *Given a user-specified threshold* θ, *and the item list* $\{I_1, I_2, \ldots, I_m\}$ *according to their supports in an increasing order, the correlation upper bound of* $\{I_a, I_c\}$ is less than θ *if the correlation upper bound of* $\{I_a, I_b\}$ *is less than* θ *and* $P(I_a) \leq P(I_b) \leq P(I_c)$.

We specify the reference item A in the first loop and start a search within each branch in the second loop. 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 an increasing order. By Theorem 2, the upper bound of the correlation 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 upper bound of the correlation is 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.

3.3 2-Dimensional search

If we can save the correlation value of pairs whose correlation is greater than θ in the database, we can get the satisfied pairs for any threshold higher than θ by a simple query. Ideally, saving all the pair correlation values is the most flexible way to users by choosing $\theta = -\infty$; however, we can only choose the θ under which time allows us to finish the search. Before the actual search, we can use the 1-D search to calculate how many pairs' support needs to be retrieved for a given threshold to conduct an estimation.

However, when the threshold is very low, the 1-D search might still calculate the upper bound of all the possible $n*(n-1)/2$ pairs. In order to get faster estimation, we use the 2-D search for the Pearson Correlation measure. But we will use different search sequences for three different types of correlation measures.

Given three items I_a, I_b, and I_c with $P(I_a) \geq P(I_b) \geq P(I_c)$, the correlation measure M is

- Type 1 if $Corr_{upper}(I_a \cap I_b) \geq Corr_{upper}(I_a \cap I_c)$.
- Type 2 if $Corr_{upper}(I_a \cap I_b) \leq Corr_{upper}(I_a \cap I_c)$.
- Type 3 if $Corr_{upper}(I_a \cap I_b) = Corr_{upper}(I_a \cap I_c)$.

The simplified χ^2-statistic, probability ratio, leverage, and likelihood ratio are all the type 1 correlation measures. However, we can coin the type 2 correlation measure which satisfies all the three properties like

$$Correlation_{Type2} = \begin{cases} \frac{\sqrt{tp-ep}}{ep}, when\, tp > ep \\ -\frac{\sqrt{ep-tp}}{ep}, when\, tp < ep \end{cases}$$

and the type 3 correlation measure like $Correlation_{Type3} = \frac{tp-ep}{ep}$.

The 1-D search algorithm starts upper bound computing from the beginning of each branch. The key difference between 1-D search and 2-D search algorithm is that the 2-D search algorithm records the turning point in the previous branch and starts the computing from the recorded point instead of the beginning of the current branch. If we sort items according to their support values and calculate the correlation value for each pair, Tables 3, 4, and 5 show the typical pattern of different types of correlation measures. For different types of correlation measures, we use different search sequences.

Table 3 Type 1 correlation upper bound pattern

	A	B	C	D	E	F
A		5	4	3	2	1
B			5	4	3	2
C				5	4	3
D					5	4
E						5
F						

Table 4 Type 2 correlation upper bound pattern

	A	B	C	D	E	F
A		9	8	7	6	5
B			7	6	5	4
C				5	4	3
D					3	2
E						1
F						

Table 5 Type 3 correlation upper bound pattern

	A	B	C	D	E	F
A		5	4	3	2	1
B			4	3	2	1
C				3	2	1
D					2	1
E						1
F						

For the type 1 correlation measures, the upper-right corner has the lowest upper bound value. The upper bound value decreases from left to right and from bottom to top. 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 right to it. At the same time, we can conclude that the upper bound of the cell under the current cell is also above the threshold θ because their upper bound is no less than that of the current cell. If the upper bound of the current cell is below the threshold θ, we will check the cell under it instead of the cell right to it. At the same time, we can conclude that all the upper bounds of the cells right to the current cell is lower than the threshold θ because their upper bound is no more than that of the current cell. Take the threshold 3.5 in the type 1 table for example, the search sequence is (A, B), (A, C), (A, D), (B, D), (B, E), (C, E), (C, F), (D, F). The upper bound of any cell below this boundary is greater than the threshold θ.

For the type 2 correlation measure, the lower-right corner has the lowest upper bound value. The upper bound value decreases from left to right and from top to bottom. We start the search from the upper-right corner instead of the upper-left corner in type 1. If the upper bound of the current cell is greater than the threshold, then we will check the cell below the current cell. At the same time, we can conclude that the upper bound of the cell left of the current cell is also above the threshold θ because their upper bound is no less than that of the current cell. If the upper bound of the current cell is less than the threshold, we will check the cell left of the current cell. At the same time, we can conclude that all the correlation upper bounds under the current cell is lower than the threshold θ because their correlation upper bound is no more than that of the current cell. Take the threshold 4.5 in the type 2 table for example, the search sequence is (A, F), (B, F), (B, E), (C, E), (C, D). The upper bound of any cell above this boundary is greater than the threshold θ.

For the type 3 correlation measure, 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. If the current column is below the threshold, we stop the search.

By using the 2-D search, we can easily calculate how many support of pairs need to be retrieved for any given threshold θ and the computational complexity is $O(n)$. According to the IO speed of retrieving the support of pairs and how much time we are allowed for the search, we can evaluate whether we can finish the search for a given threshold.

3.4 The current pair generation from the previous search

Given the correlation threshold $\theta < \theta_1$, the set of satisfied pairs under the threshold θ_1 is the subset of the set of satisfied pairs under the threshold θ. Therefore, if we can save the correlation value of pairs whose correlation is greater than θ in the database, we can get the satisfied pairs for any threshold higher than θ by a simple query. However, if we only save the correlation value of pairs whose correlation is greater than θ and we want to find the satisfied pairs for the threshold θ_1 lower than θ, we have to calculate all the pairs whose upper bound is greater than θ_1. Instead, if we save the correlation value of all the pairs whose correlation upper bound is greater than θ since we have already calculated those, we only need to calculate the pairs whose upper bound is between θ and θ_1 and then retrieve the pairs whose correlation value is greater than θ_1 in the database. If we use 2-D search algorithm to find the turning points of each branch for different thresholds θ and θ_1, the points between the two different turning points of each branch are the pairs whose upper bound is between θ and θ_1.

4 Enumeration tree structure

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 can directly get this fully-correlated k-itemset given 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.

4.1 Fully-correlated value

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 3 *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 the 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 $\rho((k-1) - Subset_i) \geq \beta$. 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 $\rho((k-1) - Subset_i) \geq \beta$. Since SS must be the subset of at least one of all the $(k-1)$-subset of the current itemset S, $\rho((k-1) - Subset_j) = \beta$ 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 with five items to save all the possible combination is shown in Fig. 1. In this tree, those itemsets are combined in a node which have the same prefix with regard to a fixed order of the items. With this structure, the itemsets contained in a node of the tree can be easily constructed in the following way: Take all the items with which the edges leading to the node are labeled and add an item that succeeds after the last edge label on the path in the fixed order of the items. In this way we only need one item to distinguish the itemsets after a particular node. In each node, we save the fully-correlated value for each itemset corresponding to it. Due to the way we construct this tree, we can conclude that the fully-correlated value saved in the current node is no less than the fully-correlated value saved in the any child node of the current node. By using the above property, given a threshold, if the fully-correlated value of the current node is less than the threshold, the fully-correlated value of any node rooted from the current node is also lower than the threshold. Then we can easily travel this tree to retrieve all the possible fully-correlated itemsets given a threshold.

4.2 MFCI generation from the enumeration tree structure

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 [5], Max-Miner [2], and GenMax [13]. Here, we use MAFIA to find the MFCIs given a threshold.

4.3 The current enumeration tree generation from the previous enumeration tree

Theorem 4 *Given an threshold* θ, *we can save all the fully-correlated itemsets in this enumeration tree structure. Let T1 be the enumeration tree for the threshold* θ_1, *T2 is the enumeration tree for the threshold* θ_2, *and* $\theta_1 > \theta_2$. *Then T1 is the subtree of T2.*

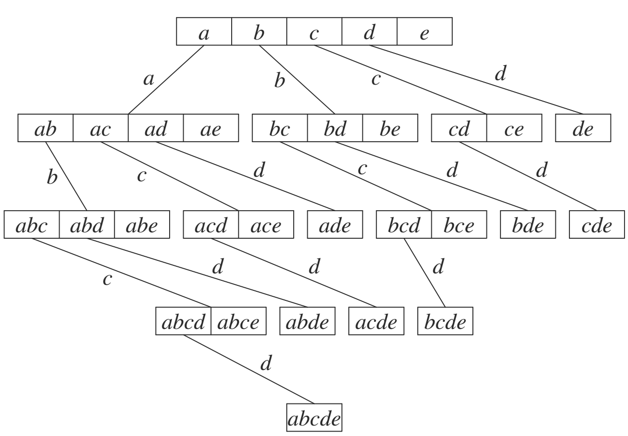


Fig. 1 An enumeration tree for five items

Proof Given any node in tree T1, the fully-correlated value of the corresponding itemset is greater than θ_1 according to the definition. Since $\theta_1 > \theta_2$, the fully-correlated value of the corresponding itemset S is also greater than θ_2. In other words, this node also exists in tree T2. Then T1 is the subtree of T2. □

The above property means the fully-correlated itemsets under a high threshold has already been contained by the fully-correlated itemsets under a low threshold. 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 level and their fully-correlated values in the enumeration tree T. Although we keep all the candidates generated from low level, 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 traveling the current enumeration tree T. For the threshold θ_2 less than θ, the current enumeration tree T is the 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 the beginning. 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 the repeat calculation which has already been finished when building the enumeration tree T.

5 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. Due to the 2-D 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. It is related to the number of satisfied pairs and the characteristic of the dataset. The best way to finish the second step is to choose a relative 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.

6 Experiments

In this section, we describe the performance study to evaluate our methods. The algorithm was run on two real life datasets. The first is the Netflix[1] data set which contains 17,770 movies and 480,000 transactions. The second is the retail data set from FIMI repository [2]containing 16,470 items and 88,000 transactions. Netflix contains many long correlation patterns because of movies in the same series and TV shows of many episodes; the retail data set contains fewer correlation patterns because those highly correlated items might have already been produced by manufactures as a package. The number of correlated pairs and MFCIs under different thresholds for these two data sets is shown in Fig. 2.

We implemented our algorithms using Java 7 on a Dell workstation with 2.4 GHz Dual CPU and 4G memory running on the Windows 7 operating system. The performance improvement comes from the improved pair search and the enumeration tree structure. When searching MFCI, we use them together. However, we only check the improvement from each technique independently in the following since the search on the enumeration tree structure is independent from the improved pair search. The improvement from using them together can be easily inferred.

6.1 Improvement from the pair search

Since we will keep pair correlation information for a relatively low threshold to facilitate the enumeration tree construction, we need to choose the affordable low threshold first, and then search pairs whose correlation is above the threshold.

6.1.1 Count the number of pair candidates

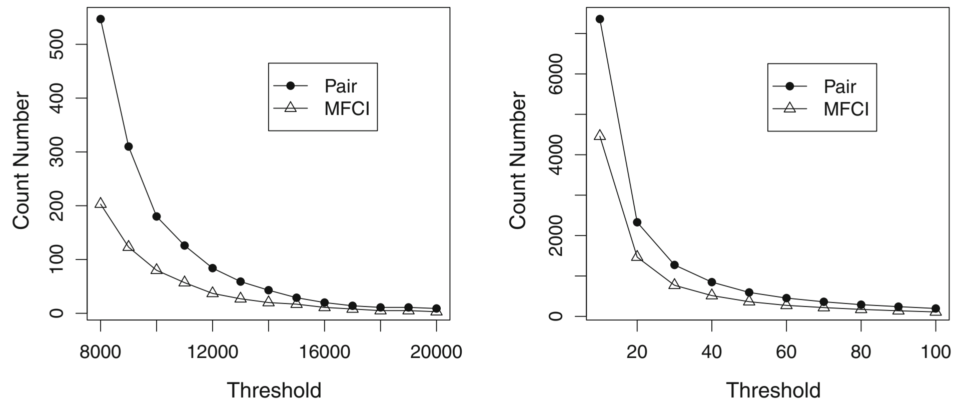
Before we determine the threshold to search the satisfied pairs, we can count how many pair's correlation upper bounds are above a tentative threshold. This number can help us to estimate the time we will spend on the satisfied pair search because the support of the pair needs to be retrieved if its correlation upper bound is above the threshold. If the IO cost of retrieving the pair support is 20 ms and we can spend one week on the satisfied pair search, we should not choose the threshold under which the number of pair candidates is above 30 million. The number of candidates and the time we spent on counting candidates under different thresholds are shown in Figs. 3 and 4 respectively. To get the number of pair candidate, 2-D search is linear and 1-D search is exponential with the threshold. When the threshold is 0, the 1-D search will check all the pairs. The runtime of 1-D search when the threshold is 0 is equal to the runtime of the brute-force method.

6.1.2 Search the satisfied pairs

Since the IO cost for calculating correlation of pairs is much higher than the time cost for calculating correlation upper bound, the upper bound calculation can save a lot of unnecessary IO cost when the threshold is high. The runtime of retrieving the satisfied pairs by calculating correlation upper bound first given different correlation thresholds is shown in Fig. 5. The runtime of the upper bound, 1-D, and 2-D search all decreases drastically as we increase the threshold. When the threshold is 0, the upper bound search will retrieve all the pair support. The runtime of upper bound search when the threshold is 0 is equal to the runtime of the brute-force method. From the graph, both 1-D and 2-D search take much less time than the upper bound search when the threshold is high because 1-D and 2-D search take less time to find the pairs whose correlation upper bound is higher than the threshold. However, the upper bound, 1-D, and 2-D search do not make too much difference for us when searching the satisfied pairs, since we need to keep pair correlation

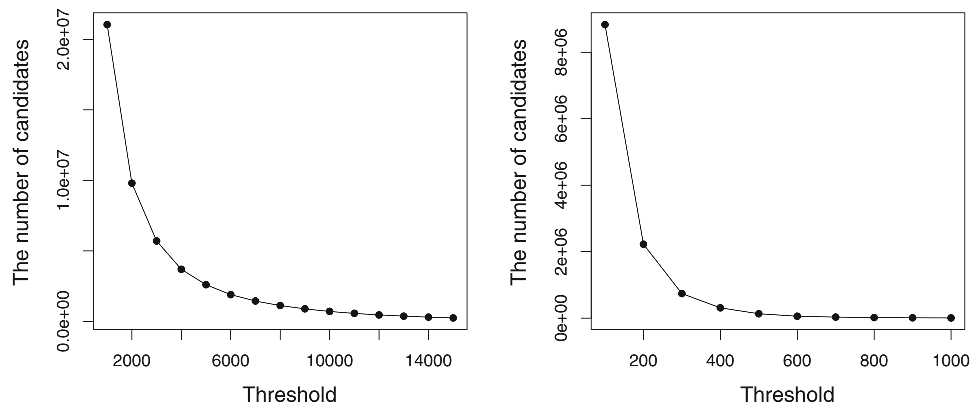
[1] http://www.netflixprize.com/.

[2] http://fimi.cs.helsinki.fi/data/.



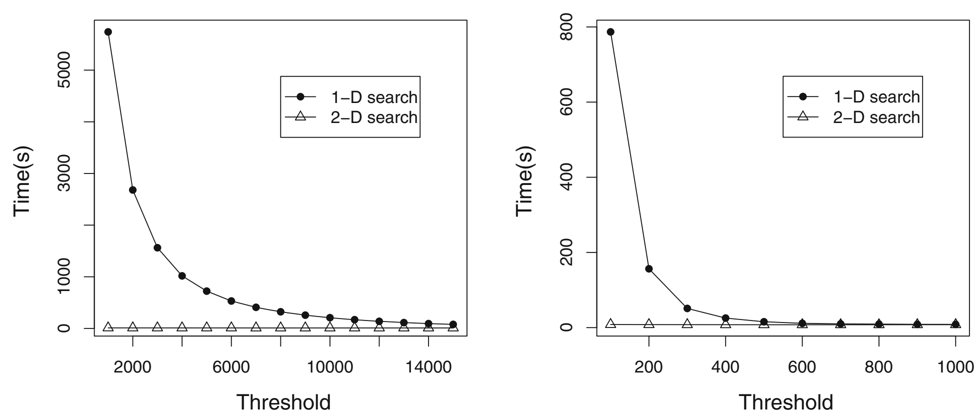
(a) Netflix **(b)** Retail

Fig. 2 The number of correlated pairs and MFCIs under different thresholds



(a) Netflix **(b)** Retail

Fig. 3 The number of candidates under different thresholds



(a) Netflix **(b)** Retail

Fig. 4 The runtime of getting the number of candidates under different thresholds

information for a relatively low threshold to facilitate the enumeration tree construction.

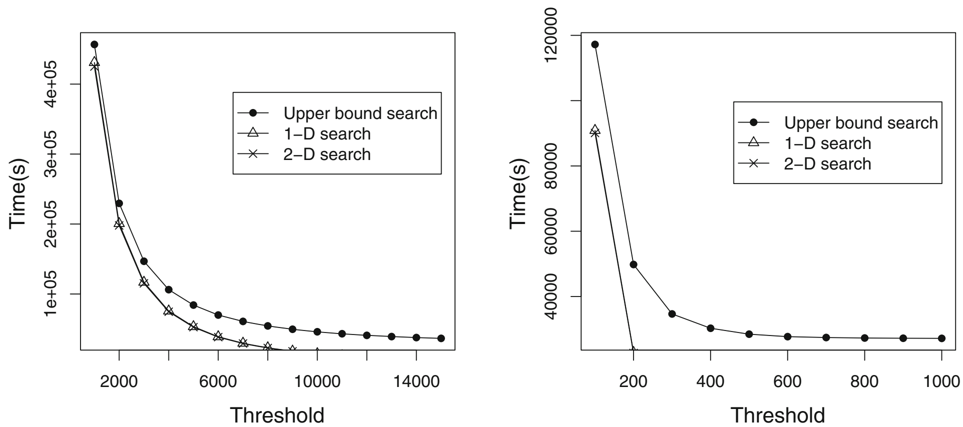
6.2 Improvement from the enumeration tree structure

The fully-correlated values saved in the enumeration tree benefit the next round search. We need to build the tree first, and then handily retrieve MFCIs according to fully-correlated values saved in the enumeration tree.

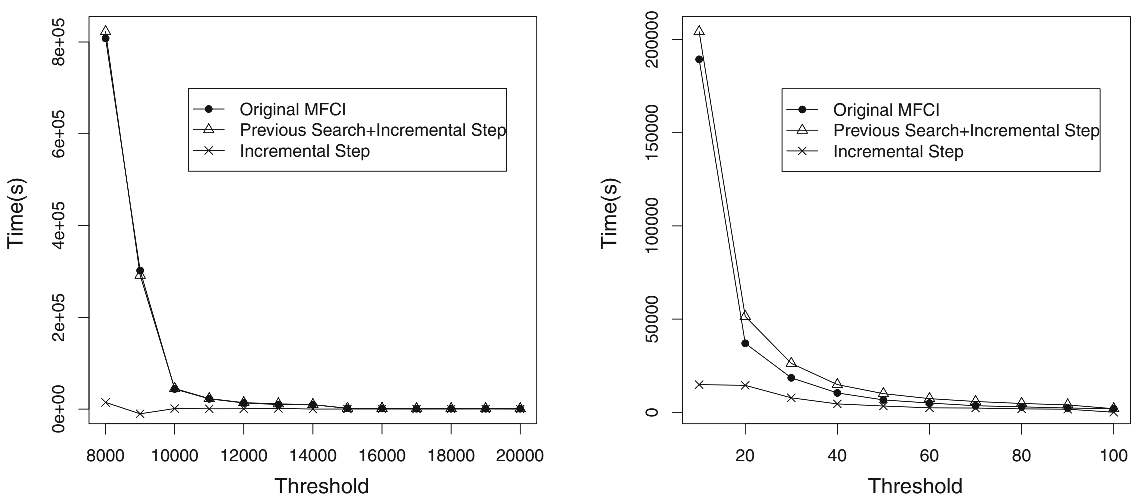
6.2.1 Build the enumeration tree

Since it is hard to estimate the time we will spend to build the enumeration tree for any given threshold, the best way

Fig. 5 The runtime for retrieving the satisfied pairs



(a) Netflix **(b)** Retail

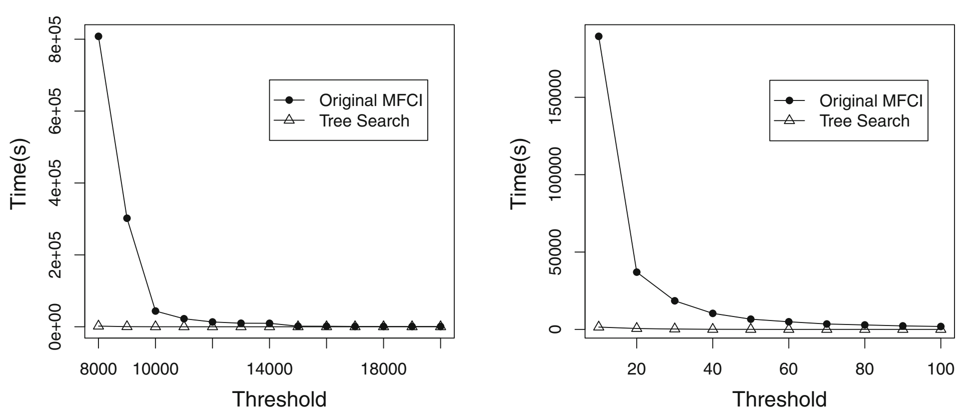


(a) Netflix when the threshold gap is 1000

(b) Retail when the threshold gap is 10

Fig. 6 The runtime of building the enumeration tree

Fig. 7 The runtime for generating MFCIs



(a) Netflix **(b)** Retail

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 tree from T_1 to T is c. Therefore, the runtime of original MFCI is a, and the total runtime of the incremental algorithm is $(b + c)$. The runtime of original MFCI and the incremental algorithm given the threshold gap g and different threshold is shown in Fig. 6. The total runtime of these two algorithms is very close, but the incremental algorithm is more flexible for user interactions.

6.2.2 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, we can apply any search technique for maximal frequent itemsets in our situation. Here, we only show how much improvement we get if we use MAFIA to generate MFCIs from the enumeration tree. Figure 7 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.

6.3 Combination of two algorithms

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-D search is used to evaluate the threshold θ under which time allows to retrieve all the correlated pairs. Second, upper bound search is used to find the correlated pairs and save their correlation value in the database. Third, we choose a relative 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 to θ. 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-D search with the incremental algorithm for the enumeration tree. From Figs. 5 and 6, we expect the runtime saving mainly comes from the 2-D search.

7 Conclusions

Our previous work on the maximal fully-correlated itemset framework can rule out the itemsets with irrelevant items and its downward-closed property helps to achieve good computational performance. However, unlike finding maximal frequent itemsets which can start the pruning from 1-itemset, finding MFCIs must start the pruning from 2-itemsets. In addition, the cost of processing the Apriori procedure each time for a different correlation threshold is also very high. In the paper, we proposed the correlation upper bound search algorithm and the enumeration tree structure saving the fully-correlated values to solve the above problems. Experimental results show that our algorithm can be an order of magnitude faster than the original MFCI. In the future, we search for the data structure which takes less memory saving the fully-correlated values and facilitate the search like FP-tree [14] for frequent itemsets.

References

1. Agrawal R, Imieliński T, Swami A (1993) Mining association rules between sets of items in large databases. In: SIGMOD '93: Proceedings of the ACM SIGMOD international conference on management of data. ACM, New York, pp 207–216
2. Bayardo RJ, Jr. (1998) Efficiently mining long patterns from databases. In: SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on management of data. ACM, New York, pp 85–93
3. Brin S, Motwani R, Silverstein C (1997) Beyond market baskets: Generalizing association rules to correlations. In: SIGMOD '97: Proceedings ACM SIGMOD international conference on management of data. ACM, New York, pp 265–276
4. Brin S, Motwani R, Ullman JD, Tsur S (1997) Dynamic itemset counting and implication rules for market basket data. In: SIGMOD '97: Proceedings of the ACM SIGMOD international conference on management of data. ACM, New York, pp 255–264
5. Burdick D (2001) Mafia: A maximal frequent itemset algorithm for transactional databases. In: ICDE '01: Proceedings of the 17th international conference on data engineering. IEEE Computer Society, Washington, DC, p 443
6. Duan L, Khoshneshin M, Street W, Liu M (2013) Adverse drug effect detection. IEEE J Biomed Health Inf 17(2):305–311
7. Duan L, Street WN (2009) Finding maximal fully-correlated itemsets in large databases. In: ICDM '09: Proceedings of the 9th international conference on data mining. IEEE Computer Society, Miami, pp 770–775
8. Duan L, Street WN, Liu Y (2013) Speeding up correlation search for binary data. Pattern Recognit Lett 34(13):1499–1507
9. Duan L, Street WN, Liu Y, Lu H (2014) Community detection in graphs through correlation. In: The 20th ACM SIGKDD conference on knowledge discovery and data mining (accepted)
10. Duan L, Street WN, Liu Y, Xu S, Wu B (2014) Selecting the right correlation measure for binary data. ACM transactions on knowledge discovery from data (accepted)
11. Dunning T (1993) Accurate methods for the statistics of surprise and coincidence. Comput Linguist 19(1):61–74
12. Geng L, Hamilton HJ (2006) Interestingness measures for data mining: A survey. ACM Comput Surv 38(3):9
13. Gouda K, Zaki MJ (2005) Genmax: An efficient algorithm for mining maximal frequent itemsets. Data Min Knowl Discov 11(3):223–242
14. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on management of data. ACM, New York, pp 1–12
15. Jermaine C (2005) Finding the most interesting correlations in a database: How hard can it be? Inf Syst 30(1):21–46
16. Liu M, Hinz ERM, Matheny ME, Denny JC, Schildcrout JS, Miller RA, Xu H (2013) Comparative analysis of pharmacovigilance methods in the detection of adverse drug reactions using

electronic medical records. J Am Med Inform Assoc 20(3):420–426

17. Mohamed MH, Darwieesh MM (2013) Efficient mining frequent itemsets algorithms. Int J Mach Learn Cybern. doi:10.1007/s13042-013-0172-6
18. Omiecinski ER (2003) Alternative interest measures for mining associations in databases. IEEE Trans Knowl Data Eng 15(1):57–69
19. Piatetsky-Shapiro G (1991) Discovery, analysis, and presentation of strong rules. AAAI/MIT Press, Cambridge
20. Tan P-N, Kumar V, Srivastava J (2004) Selecting the right objective measure for association analysis. Inf Syst 29(4):293–313
21. Tew C, Giraud-Carrier C, Tanner K, Burton S (2014) Behavior-based clustering and analysis of interestingness measures for association rule mining. Data Min Knowl Discov 28(4):1004–1045
22. Vo B, Le T, Coenen F, Hong T-P (2014) Mining frequent itemsets using the n-list and subsume concepts. Int J Mach Learn Cybern. doi:10.1007/s13042-014-0252-2
23. Zhou W, Zhang H (2013) Correlation range query for effective recommendations. World Wide Web. doi:10.1007/s11280-013-0265-x