Generating (Fuzzy) Frequent Itemsets by a Bitmap-based Algorithm – the Word's Most Compact Frequent Itemset Miner

János Abonyi

Department of Process Engineering, University of Pannonia
POB. 158, H-8200 Veszprém, Hungary
abonyij@fmt.uni-pannon.hu

Abstract: Mining frequent itemsets in databases is an important and widely studied problem in data mining research. The problem of mining frequent itemsets is usually solved by constructing candidates of itemsets, and identifying those itemsets that meet the requirement of frequent itemsets. This paper proposes a novel algorithm based on BitTable (or bitmap) representation of the data. Data - related to frequent itemsets - are stored in spare matrices. Simple matrix and vector multiplications are used to calculate the support of the potential n+1 itemsets. The main benefit of this approach is that only bitmaps of the frequent itemsets are generated. The concept is simple and easily interpretable and it supports a compact and effective implementation (in MATLAB). An application example related to the BMS-WebView-1 benchmark data is presented to illustrate the applicability of the proposed algorithm.

Keywords: frequent itemsets, BitTable

1 Introduction

Mining frequent itemsets is an important and widely studied problem in the field of data mining research [1]. When the minimum support is small, and hence the number of frequent itemsets is very large, most algorithms either run out of memory or run over of allowed disk space due to the huge number of frequent itemsets. Most of early researches focused on the reduction of the amount of candidate itemsets and the time required to database-scanning. A lot of these algorithms adopt an Apriori-like candidate itemsets generation and support count approach which is a time-demanding process and needs a huge memory capacity.

Among the wide range of the developed algorithms this paper focuses on bitmap based solutions, like MAFIA [2] and BitTableFI [3], where BitTable is used for compressing the database horizontally and vertically for quick candidate itemsets generation and support count. Experiments with both synthetic and real databases

show that BitTableFI outperforms Apriori and CBAR which uses ClusterTable for quick support count. Wei Song at al. has developed this concept resulting the Index-BitTableFI algorithm [4].

In these algorithms the task of mining frequent itemsets is still solved by constructing candidate itemsets, then, identifying the itemsets that meet the frequent itemset requirement. The motivation of this paper is to develop an extremely simple and easily implementable algorithm based on the bitmap-like representation of the frequent itemsets. The key idea is simple: store the data related to a given itemset in a binary vector. Hence, data related to frequent itemsets is stored in spare matrices, simple matrix and vector multiplications are used to calculate the support of the potential *k+1* itemsets. The main benefit of this approach is that only bitmaps of the frequent itemsets are generated based on the elementwise products of the binary vectors corresponding the building *k-1 frequent itemsets*. Furthermore, when fuzzy membership values are stored in the bitmap-like matrices, the algorithm can directly be used to generate fuzzy frequent itemsets. The concept is simple and easily interpretable, so it supports the compact and effective implementation of the algorithm (in MATLAB).

The paper is organized as follows. In Section 2 we are going to briefly revisit the problem definition of frequent itemset mining by basic definitions and show the details of the proposed algorithm. In Section 3, an application for web usage mining will be presented. Finally, the results and the advantages of the proposed method will be summarized.

2 The Proposed Algorithm

2.1 Definitions – Matrix Representation

The problem of finding frequent itemsets can be formally stated by the following way: let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of distinct literals, called items. Let $D = \{T_1, T_2, \ldots, T_N\}$ be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. A transaction T is said to support an itemset X if it contains all items of X, i.e., $X \subseteq T$. The support of an itemset X is the number (or percentage) of transactions that support X. An itemset is frequent if its support is greater or equal to a user-specified minimum support threshold, denoted *MinSup*. Frequent itemsets are also known as large itemsets. An itemset X is called *k-itemset* if it contains k items from I.

Items	
Tid	**items**
T_1	a, c, d
T_2	b, c, e
T_3	a, b, c, e
T_4	b, e

Items					
Tid	1(a)	2(b)	3(c)	4(d)	5(e)
T_1	1	0	1	1	0
T_2	0	1	1	0	1
T_3	1	1	1	0	1
T_4	0	1	0	0	1
Sum	**2**	**3**	**3**	**1**	**3**

Table 1
Illustrative example for a transactional dataset and its binary incidence matrix representation

An illustrative example for *D* transactional database is shown in Table 1(a). The transactional database can be transformed into a bitmap-like matrix representation, where if an item $i=1,\ldots,m$ appears in transaction T_j $j=1,\ldots,N$, the bit i of the j-th row of the binary incidence matrix will be marked as one (as seen in Table 1).

For mining association rules, non-binary attributes have to be mapped to binary attributes. The straightforward mapping method is to transform the metric attributes to k ordinal attributes by building categories (e.g., an attribute income might be transformed into a ordinal attribute with the three categories: “low”, “medium” and “high”). Then, in a second step, each categorical attribute with categories k is represented by k binary dummy attributes which correspond to the items used for mining. An example application using questionnaire data can be found in [5].

As the support of an itemset is a percentage of the total number of transactions, the sum of the columns of this $\mathbf{B}^0_{N\times n}$ matrix represent the support of the $j=1,\ldots,n$ items. (see the bottom of Table 1(b)) Hence, if $\mathbf{b}^0_j$ represents the j-th column of

$\mathbf{B}^0_{N\times n}$ which is related to the occurrence of the i_j -th item, then the support of the i_j item can be easily calculated as

$$\sup\left(X = i_j\right) = \left(\mathbf{b}_j^0\right)^T \mathbf{b}_j^0 / N \tag{1}$$

(in this case the result is given in percentage). Similarly, the support of an $X_{i,j} = \{i_i, i_j\}$ itemset can be easily calculated by a simple vector product of the two related bitvectors, since when both i_i and i_j items appear in a given transaction the product of the two related bits can represent the AND connection of the two items:

$$\sup\left(X_{i,j} = \{i_i, i_j\}\right) = \left(\mathbf{b}_i^0\right)^T \mathbf{b}_j^0 / N \tag{2}$$

The matrix representation allows the effective calculation of all of the itemsets:

$$\mathbf{S}^2 = \left(\mathbf{B}^0\right)^T \mathbf{B}^0 \tag{3}$$

where the *i,j*-th element of the $\mathbf{S}^2$ matrix represents the support of the $X_{i,j} = \{i_i, i_j\}$ 2-itemset. Of course, only the upper triangular elements of this, this symmetrical matrix has to be checked, whether the $X_{i,j} = \{i_i, i_j\}$ *2-itemsets* are frequent or not.

Fuzzy membership values can also be stored in the same matrix structure, where the columns represent the items and the rows the transactions (see Table 2). Hence, beside the analysis of classical transactional datasets, the analysis of fuzzy data is also considered in this paper. In this case let $D = \{t_1, t_2, \ldots, t_N\}$ be a transformed fuzzy dataset of N tuples (data points) with a set of variables $Z = \{z_1, z_2, \ldots, z_n\}$ and let $c_{i,j}$ be an arbitrary fuzzy interval (fuzzy set) associated with attribute z_i in *Z*. Use the notation $\langle z_i : c_{i,j} \rangle$ for an *attribute-fuzzy interval pair*, or simply *fuzzy item*, (i.e.: $\langle Age : young \rangle$). For *fuzzy itemsets*, we use expressions like $\langle Z : C \rangle$ to denote an ordered set $Z \subseteq Z$ of attributes and a corresponding set *C* of some fuzzy intervals, one per attribute, i.e. $\langle Z : C \rangle$.

Table 2
Example database containing membership values

	Items		
	$\langle \text{Balance}:\text{medium} \rangle$	$\langle \text{Credit}:\text{high} \rangle$	$\langle \text{Income}:\text{high} \rangle$
T_1	0.5	0.6	0.4
T_2	0.8	0.9	0.4
T_3	0.7	0.8	0.7
T_4	0.9	0.8	0.3
T_5	0.9	0.7	0.6

Equation (3) can also be used in case the $\mathbf{B}^0_{N\times n}$ matrix stores fuzzy membership values. A fuzzy support reflects how the record of the dataset supports the itemset. In the literature, the fuzzy support value has been defined in different ways. Some of the researchers suggest the minimum operator as in fuzzy intersection, others prefer the product operator. They can be defined formally as follows: value $T_k(z_i)$ for attribute z_i, then the ***fuzzy support*** of $\langle Z:C \rangle^2$ with respect to *D* is defined as

$$FS(Z:C) = \frac{\sum_{k=1}^{N} \prod_{\langle z_i : c_{i,j} \rangle \in \langle Z:C \rangle} T_k(z_i)}{N} \tag{4}$$

The following example illustrates the calculation of the fuzzy support value. Let $\langle X:A \rangle = [\langle \text{Balance}:\text{medium} \rangle \cup \langle \text{Income}:\text{high} \rangle]$ be a fuzzy itemset, the dataset shown in Table 2. The fuzzy support of $\langle X:A \rangle$ is given by:

$$FS(X:A) = \frac{0.5 \cdot 0.4 + 0.8 \cdot 0.4 + 0.7 \cdot 0.7 + 0.9 \cdot 0.3 + 0.9 \cdot 0.6}{5} = 0.364 \tag{5}$$

An itemset $\langle Z:C \rangle$ is called ***frequent*** if its fuzzy support value is higher than or equal to a user-defined minimum support threshold σ.

2.2 Mining Frequent Itemsets Based on Bitmap-like Representation

2.2.1 Apriori Algorithm for Mining Frequent Itemsets

The best-known and most commonly applied frequent pattern mining algorithm *Apriori* was developed by Agrawal et al. [6]. The name is based on the fact that

the algorithm uses prior knowledge of the already determined frequent itemsets. It is an iterative, breadth-first search algorithm, based on generating stepwise longer *candidate* itemsets, and clever pruning of non-frequent itemsets. Pruning possesses the advantage of the so-called *apriori* (or *upward closure*) *property* of frequent itemsets: all subsets of a frequent itemset must also be frequent. Each candidate generation step is followed by a counting step where the supports of candidates are checked and non-frequent ones deleted.

Given a user-specified *MinSup*, Apriori passes multiple times over the database to find all frequent itemsets. In the first pass, Apriori scans the transaction database to count the support of each item and identify the frequent *1-itemsets* marked as L_1. In a subsequent *k*-th pass, Apriori establishes a candidate set of frequent *k-itemsets* (which are itemsets of length *k*) marked as C_k from L_{k-1}. Two arbitrary L_{k-1} join each other, when their first *k-1* items are identical. Then, the downward closure property is applied to reduce the number of candidates. This property refers to the fact that any subset of a frequent itemset must be frequent. Therefore, the process deletes all the *k-itemsets* whose subsets with length *k - 1* are not frequent. Next, the algorithm scans the entire transaction database to check whether each candidate *k-itemset* is frequent.

Generation and counting alternate, until at some step all generated candidates turn out to be non-frequent. A high-level pseudocode of the algorithm used for mining fuzzy frequent itemsets based on the *apriori* principle is given in Table 3.

Table 3
Algorithm: *Mining Frequent Itemsets* (minimum support σ, dataset *D*)

$k = 1$
  $(C_k; D_F) = Transform(D)$
  $F_k = Count(C_k, D_F, \sigma)$
while $|C_k| \neq 0$ do
  $inc(k)$
  $C_k = Generate(F_{k-1})$
  $C_k = Prune(C_k)$
  $F_k = Count(C_k, D_F, \sigma)$
  $F = F \bigcup F_k$
end

In case of mining fuzzy frequent itemsets the subroutines are outlined as follows:

Transform(*D*): Generates a fuzzy database D_F from the original dataset *D*. At the same time the complete set of candidate items C_1 is found.

Count(C_k, D_F, σ): In this subroutine the fuzzy database is scanned and the fuzzy support of candidates in C_k is counted. If this support is not less than minimum support σ for a given itemset, we put it into the set of frequent itemsets F_k.

Generate(F_{k-1}): Generates candidate itemsets C_k from frequent itemsets F_{k-1}, discovered in the previous iteration *k-1*. For example, if $F_1 = \left\{\left\langle \text{Balance} : \text{high} \right\rangle, \left\langle \text{Income} : \text{high} \right\rangle\right\}$ then $C_2 = \left\{\left\langle \text{Balance} : \text{high} \right\rangle \cup \left\langle \text{Income} : \text{high} \right\rangle\right\}$

Prune(C_k): During the prune step, the itemset will be pruned if one of its subsets does not exist in the set of frequent itemsets *F*.

2.2.2 The Proposed Algorithm for Mining Frequent Itemsets

The proposed algorithm has similar philosophy as the Apriori TID [7], which is does not revisit the original table of data, $\mathbf{B}^0_{N\times n}$, for computing the supports larger itemsets, but transforms the table as it goes along with the generation of the k-itemsets, $\mathbf{B}^1_{N_1\times n_1}$ … $\mathbf{B}^k_{N_k\times n_k}$ $N_k < N_{k-1} < \mathrm{K} < N$.

$\mathbf{B}^1_{N_1\times n_1}$ represents the data related to the 1-frequent itemsets. This table is generated from $\mathbf{B}^0_{N\times n}$, by erasing the columns related to the non-frequent items, to reduce the size of a Bittable and improve the performance of the generation process, because all non-frequent 1-itemsets are not useful for further analysis. The rows of $\mathbf{B}^k_{N_k\times n_k}$ which do not contain frequent itemsets (the sum of the row is zero) are also deleted from the table. This concept is taken from the Apriori TID algorithm based on using new data structure called *counting_base* to store the transactions which can support the actual list of candidates.

If a column remains, the index of its original position is written into matrix that stores the indexes ("pointers") of the element of the itemsets, $\mathbf{L}^1_{N_1\times 1}$.

Data related to frequent itemsets are stored in spare matrices. Simple matrix and vector multiplications are used to calculate the support of the potential *k+1* itemsets:

$$\mathbf{S}^k = \left(\mathbf{B}^{k-1}\right)^T \mathbf{B}^{k-1} \tag{6}$$

where the *i,j*-th element of the $\mathbf{S}^k$ matrix represent the support of the $X_{i,j} = \left\{\mathbf{L}^{k-1}_i, \mathbf{L}^{k-1}_j\right\}$ itemset. Of course, only the upper triangular elements of this symmetrical matrix has to be checked, whether the *itemsets* are frequent or not. The main benefit of this approach is that only the BitTables of the frequent itemsets are generated, by forming the columns of the $\mathbf{B}^k_{N_k\times n_k}$ as the element wise products of the columns of the $\mathbf{B}^{k-1}_{N_{k-1}\times n_{k-1}}$, $\mathbf{b}^k_i$, $\mathbf{b}^k_j$.

The concept is simple and easily interpretable that supports the compact and effective implementation (see the appendix for the MATLAB code).

The above presented approach is related to the lazy version of the algorithm. The performance of the support count can be significantly decreased when only the relevant blocks of the $\mathbf{B}^{k-1}_{N_{k-1}\times n_{k-1}}$ matrix are multiplied by their transpose. When $L^{k-1}_{N_{k-1}\times k-1}$ matrices related to the indexes of the k-1-itemsets are ordered it is easy to follow the heuristics of the apriori algorithm, as only the itemsets L_{k-1} join each other, when their first *k-1* items are identical (the set of these itemsets form the blocks of the $\mathbf{B}^{k-1}_{N_{k-1}\times n_{k-1}}$ matrix).

3 Application Example

The main benefit of the proposed algorithm is that it can be effectively implemented in tools tailored to perform matrix manipulations. In the appendix of this paper the full implementation of the algorithm is shown as a MATLAB function. This code with 15 lines is optimized to give a reasonable calculation time. Hence, the matrix multiplications are performed in block-wise manner and the unnecessary transactions (rows) are removed from the $\mathbf{B}^{k}_{N_k\times n_k}$ matrices.

The proposed algorithm and MATLAB code has been applied to the BMS-WebView-1 benchmark problem, [8] where data taken from the www.gazelle.com web portal is analyzed. This database contains 59,602 transactions and 497 items (webpages). The maximal size of a basket is 267, while its average size is 2.5 (length of the average visit of the portal).

The calculation time is shown in Figure 1 is quite reasonable, considering the MATLAB framework is not optimized to calculation speed. The results agree with the results of other applications [8] (see Figure 2 for the number of the mined itemsets). It is interesting to see Figure 3 that nicely illustrates the key element of the proposed approach, the bitmap of the 2nd itemset of the studied benchmark data.

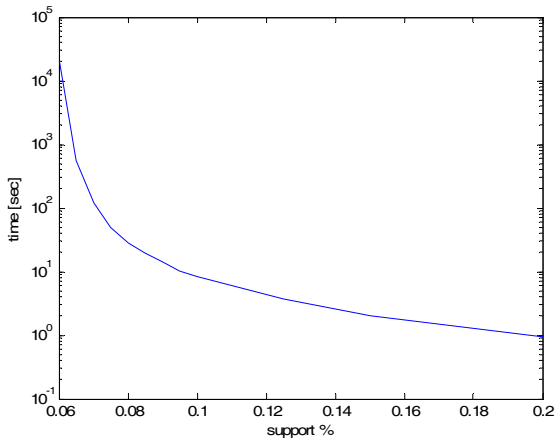

Figure 1
Time required to mine frequent itemsets with a given support

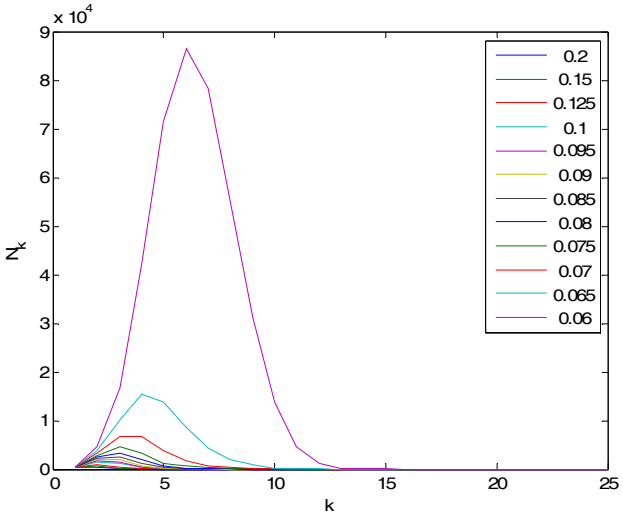

Figure 2
Number of frequent itemsets related to different support treshold (MinSup).
As can be seen, at smaller support values the number of itemsets can be really huge.

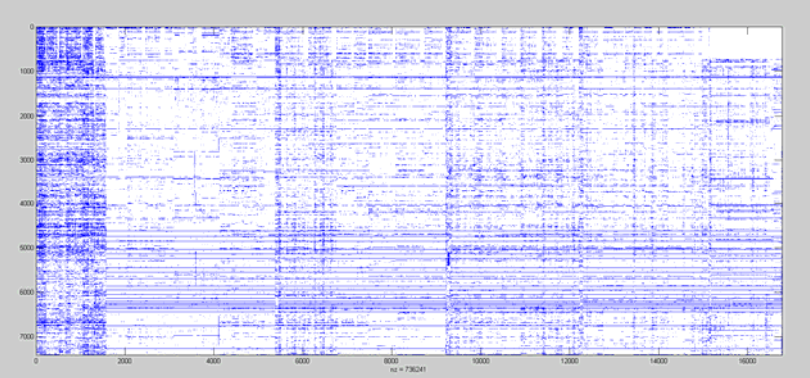

Figure 3
Data related to frequent itemsets are stored in "Bitmaps" like shown in this figure, where the columns represent the itemsets and the dots in the rows represen the given itemset is in the transaction related to the row of the matrix

Conclusions

This paper proposed a novel algorithm for mining frequent itemsets. The key idea is to store the data related to a given itemset in a binary vector. Hence, data related to frequent itemsets are stored in spare matrices and simple matrix and vector multiplications are used to calculate the support of the potential $k+1$ itemsets.

The main benefit of this approach is that only bitmaps of frequent itemsets are generated based on the elementwise products of the binary vectors corresponding the building k-1 frequent itemsets, since bitwise AND operation is greatly faster than comparing each item in two frequent itemsets (as at Apriori). Furthermore, when fuzzy membership values are stored in the bitmap-like matrices, the algorithm can directly be used to generate fuzzy frequent itemsets. The concept is simple and easily interpretable, so it supports the compact and effective implementation of the algorithm (in MATLAB). The application example related to the BMS-WebView-1 benchmark problem demonstarted that applicability of the developed compact MATLAB code that can be easily used by medium-sized firms having around 100 000 transactions and several tousand items.

Acknowledgement

The financial support from the TAMOP-4.2.2-08/1/2008-0018 (Livable environment and healthier people – Bioinnovation and Green Technology research at the University of Pannonia, MK/2) project is gratefully acknowledged.

Appendix: the word's most compact frequent itemset miner in MATLAB

```
function [items,Ai]=bittable(A, suppp)
[N,n]=size(A);
items{1}=find(sum(A,1)>=suppp)';
k=1; Ai{1}=A(:,items{1});
while ~isempty(items{k})
  k=k+1; Ai{k}=[]; items{k}=[];
 index=[0; find(sum(abs(diff((items{k-1}(:,1:end-1)))),2)~=0); size(items{k-1},1)];
    for i=1:length(index)-1
        v=[index(i)+1:index(i+1)];  m=Ai{k-1}(:,v)'*Ai{k-1}(:,v);
        m=triu(m,1); [dum1,dum2]=find((m)>suppp);
        for j=1:length(dum1)
         items{k}=[items{k}; [items{k-1}(v(dum1(j)),:) items{k-1}(v(dum2(j)),end) ] ];
         Ai{k}= [ Ai{k} Ai{k-1}(:,v(dum1(j))).*Ai{k-1}(:,v(dum2(j)))];
        end
    end
    [items{k},I]=sortrows(items{k}); Ai{k}=Ai{k}(:,I);
end
```

References

[1] J. Abonyi et al., Adatbányászat – a hatékonyság eszköze. Computerbooks, 2006

[2] D. Burdick, M. Calimlim, J. Gehrke, MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases Department of Computer Science, Cornell University

[3] J. Dong, M. Han BitTableFI: An efficient mining frequent itemsets algorithm, Science Direct, 2006

[4] W. Song, B. Yang, Z. Xu, Index-BitTableFI: An improved algorithm for mining frequent itemsets; Knowledge-based Systems journal homepage: elsevier.com/locate/knosys, 2008

[5] Hastie et al. (2001)

[6] R. Agrawal, R. Srikant, Fast algorithm for mining association rules in large databases, in: Proceedings of 1994 International Conference on VLDB, pp. 487–499. 1994

[7] J. Han, M. Kamber, Data Mining Concepts and Techniques, Elsevier, 2001

[8] Z. Zheng, R. Kohavi, L. Mason, Real World Performance of Association Rule Algorithms; ACM, 2001