ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/296873041

A Compressed Vertical Binary Algorithm for Mining Frequent Patterns.

Chapter · November 2008

CITATIONS
0

READS
336

4 authors, including:

José Hernández-Palancar
Advanced Technologies Applications Center, (CENATAV), Cuba
75 PUBLICATIONS 244 CITATIONS
SEE PROFILE

Raudel Hernández-León
Centro de Aplicaciones de Tecnologias de Avanzada
44 PUBLICATIONS 91 CITATIONS
SEE PROFILE

José E. Medina Pagola
University of Information Sciences
64 PUBLICATIONS 338 CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

High resolution palmprint feature extraction and matching View project

Data Mining View project

All content following this page was uploaded by Raudel Hernández-León on 06 March 2016.

The user has requested enhancement of the downloaded file.

A Compressed Vertical Binary Algorithm for Mining Frequent Patterns

J. Hdez. Palancar[1], R. Hdez. León[1], J. Medina Pagola[1], and A. Hechavarría[1]

Advanced Technologies Application Center (CENATAV), 7a ♯ 21812 e/ 218 y 222, Rpto. Siboney, Playa, C.P. 12200, Ciudad de la Habana, Cuba.
{jpalancar,rhernandez,jmedina,ahechavarria}@cenatav.co.cu

1 Introduction

Mining association rules in transaction databases have been demonstrated to be useful and technically feasible in several application areas [14], [18], [21] particularly in retail sales, and it becomes every day more important in applications that use document databases [11], [16], [17]. Although research in this area has been going on for more than one decade; today, mining such rules is still one of the most popular methods in knowledge discovery and data mining.

Various algorithms have been proposed to discover large itemsets [2], [3], [6], [9], [11], [19]. Of all of them, Apriori has had the biggest impact [3], since its general conception has been the base for the development of new algorithms to discover association rules.

Most of the previous algorithms adopt an Apriori-like candidate set generation-and-test approach. However, candidate set generation is still costly, especially when there are many items, when the quantity of items by transaction is high, or the minimum support threshold is quite low. These algorithms need to scan the database several times to check the support of each candidate, and it is a time-consuming task for very sparse and huge databases. The weak points of the Apriori algorithm are these aspects: the candidate generation and the count of each candidate support.

Performance of the algorithm could be significantly improved if we find a way to reduce the computational cost of the tasks above mentioned.

Although in [3] the way in which the transactions are represented is not mentioned by the authors, this aspect influences decisively in the algorithm. In fact, it has been one of the elements used by other authors, including us, in the formulation of new algorithms [4], [6], [7], [9], [19].

We face the problem in the following way:

- Other authors represent the transaction database as sorted lists (or array-based), BTree, Trie, etc., using items that appear in each transaction;

others use horizontal or vertical binary representations. We will use a compressed vertical binary representation of the database.
- The efficiency count of each candidate's support in this representation can be improved using logical operations, which are much faster than working with non-compact forms.

A new algorithm suitable for mining association rules in databases is proposed in this paper; this algorithm is named as CBMine (Compressed Binary Mine).

The discovery of large itemsets (the first step of the process) is computationally expensive. The generation of association rules (the second step) is the easiest of both. The overall performance of mining association rules depends on the first step; for this reason, the comparative effects of the results that we present with our algorithm covers only the first step.

In the next section we give formal definitions about association rules and frequent itemsets. The third section is dedicated to related work. The fourth section contains the description of CBMine algorithm. The experimental results are discussed in the fifth section.

The new algorithm shows significantly better performance than several algorithms, like Bodons Apriori algorithms, and in sparse databases than MAFIA, and in a general way it is applicable to those algorithms with an Apriori-like approach.

2 Preliminaries

Let be $I = \{i_1, i_2, ..., i_n\}$ a set of elements, called items (we prefer to use the term elements instead of literals [2], [3]). Let D be a set of transactions, where each transaction T is a set of items, so that $T \subseteq I$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. The association rule $X \Rightarrow Y$ holds in the database D with certain quality and a support s, where s is the proportion of transactions in D that contain $X \cup Y$. Some quality measures have been proposed, although these are not considered in this work.

Given a set of transactions D, the problem of mining association rules is to find all association rules that have a support greater than or equal to the user-specified minimum (called *minsup*) [3]. For example, beer and disposable diapers are items so that beer $\Rightarrow$ diaper is an association rule mined from the database if the co-occurrence rate of beer and disposable diapers (in the same transaction) is not less than *minsup*.

The first step in the discovery of association rules is to find each set of items (called *itemset*) that has co-occurrence rate above the minimum support. An itemset with at least the minimum support is called a *large* itemset or a *frequent* itemset. In this paper, as in others, the term frequent itemset will be used. The size of an itemset represents the number of items contained in the

itemset, and an itemset containing k items is called a k-itemset. For example, beer, diaper can be a frequent 2-itemset. If an itemset is frequent and no proper superset is frequent, we say that it is a *maximally* frequent itemset.

Finding all frequent itemsets has received a considerable amount of research effort in all these years because it is a very resource-consuming task. For example, if there is a frequent itemset with size l, then all $2^l - 1$ nonempty subsets of the itemset have to be generated.

The set of all subsets of I (the powerset of I) naturally forms a lattice, called the *itemset lattice* [10], [22]. For example, consider the lattice of subsets of $I = \{i_1, i_2, i_3, i_4\}$, shown in Fig. 1 (the empty set has been omitted). Each maximal frequent itemset of the figure is in bold face and in an ellipse.

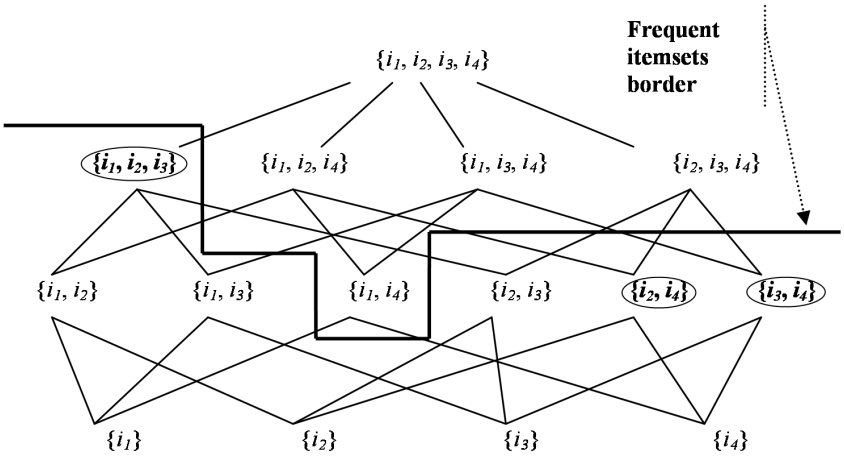

Fig. 1. Lattice of subsets of $I = \{i_1, i_2, i_3, i_4\}$

Due to the *downward closure* property of itemset support - meaning that any subset of a frequent itemset is frequent - there is a border, so that all frequent itemsets lie below the border, while all infrequent itemsets lie above it. The border of frequent itemsets is shown with a bold line in Fig. 1.

An optimal association mining algorithm must only evaluate the frequent itemsets traversing the lattice in some way. This one can be done considering an equivalence class approach. The equivalence class of an itemset a, expressed as $E(a)$, is given as:

$$E(a) = \{b : |a| = k, |b| = k, Prefix_{k-1}(b) = Prefix_{k-1}(a)\}, \qquad (1)$$

where $Prefix_k(c)$ is the prefix of size k of c, i.e., its k first items in a lexicographical order.

Assuming equivalence classes, the itemset lattice of Fig. 1 can be structured in a forest, shown in Fig. 2, clustering itemsets of same equivalence classes.

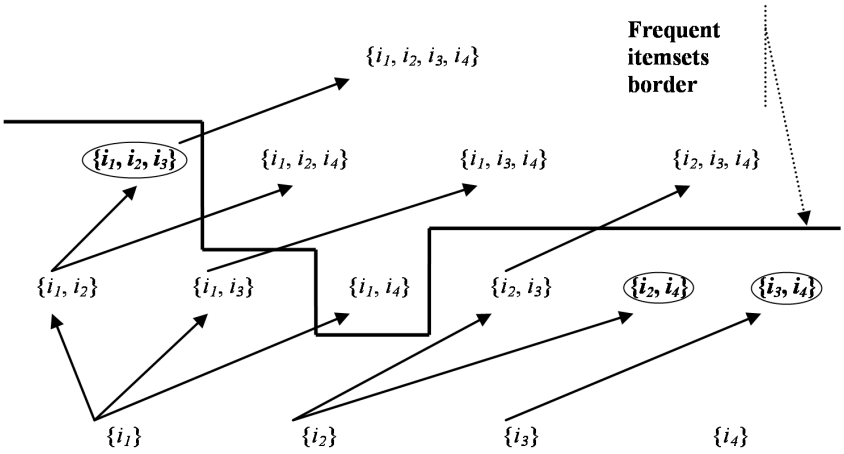

Fig. 2. Search forest of subsets of $I = \{i_1, i_2, i_3, i_4\}$

In order to traverse the itemset space, a convenient strategy should be chosen. Today's common approaches employ either breath-first search or depth-first search. In a breadth strategy the support values of all (k-1)-itemsets are determined before counting the support values of k-itemsets, and in a depth one recursively descends following the forest structure defined through equivalence classes [10].

The way itemsets are represented is decisive to compute their supports. Conceptually, a database is a two-dimensional matrix where the rows represent the transactions and the columns represent the items. This matrix can be implemented in the following four different formats [20]:

- Horizontal item-list (**HIL**): The database is represented as a set of transactions, storing each transaction as a list of item identifiers (item-list).
- Horizontal item-vector (**HIV**): The database is represented as a set of transactions, but each transaction is stored as a bit-vector (item-vector) of 1's and 0's to express the presence or absence of the items in the transaction.
- Vertical tid-list (**VTL**): The database is organized as a set of columns with each column storing an ordered list (tid-list) of only the transaction identifiers (TID) of the transactions in wich the item exists.
- Vertical tid-vector (**VTV**): This is similar to VTL, except that each column is stored as a bit-vector (tid-vector) of 1's and 0's to express the presence or absence of the items in the transactions.

Many association rule mining algorithms have opted for a list-based (horizontally or vertically) layout since, in general, this format takes less space than the bit-vector approach. In other way, it could be noticed that computing the

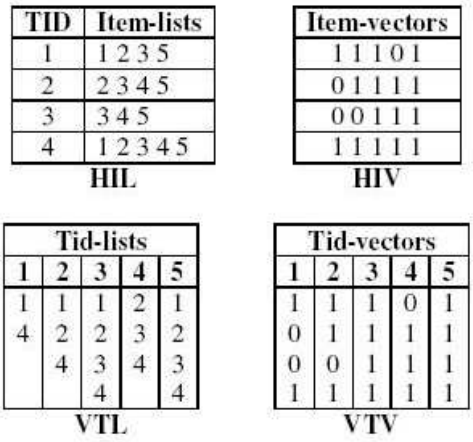

Fig. 3. Examples of database layouts

supports of itemsets is simpler and faster with the vertical layout (VTL or VTV) since it involves only the intersections of tid-lists or tid-vectors.

In a general point of view, rule mining algorithms employ a combination of a traverse strategy (breadth-first or depth-first) and a form of the database layout. Examples of algorithms for horizontal mining with a breadth strategy previously presented are Apriori, AprioriTID, DIC [3] and with a depth strategy are different version applying FP-Trees [1]. Other algorithms considering a VTL layout are Partition, with a breadth strategy [10], and Eclat, with a depth strategy [22].

In this paper we evaluate a compressed form of the VTV layout, improving the performance of the itemset generation, applicable to those algorithms with an Apriori-like approach.

The problem of mining frequent itemsets was first introduced by Agrawal *et al.* [2]. To achieve efficient mining frequent patterns, an antimonotonic property of frequent itemsets, called the Apriori heuristic, was formulated in [3]. The Apriori heuristic can dramatically prune candidate itemsets.

Apriori is a breadth-first search algorithm, with a HIL organization, that iteratively generates two kinds of sets: C_k and L_k. The set L_k contains the large itemsets of size k (k-itemsets). Meanwhile, C_k is the set of candidate k-itemsets, representing a superset of L_k. This process continues until a null set L_k is generated.

The set L_k is obtained scanning the database and determining the support for each candidate k-itemset in C_k. The set C_k is generated from L_{k-1} with the following procedure:

$$C_k = \{c | Join(c, L_{k-1}) \wedge Prune(c, L_{k-1})\} \tag{2}$$

where:

$$Join(\{i_1,..,i_k\}, L_{k-1}) \equiv \langle \{i_1,..,i_{k-2},i_{k-1}\} \in L_{k-1} \wedge \{i_1,..,i_{k-2},i_k\} \in L_{k-1} \rangle, \quad (3)$$

$$Prune(c, L_{k-1}) \equiv \langle \forall s[s \subset c] \wedge |s| = k-1 \rightarrow s \in L_{k-1} \rangle. \quad (4)$$

Observe that the Join step (3) takes two (k-1)-itemsets of a same equivalence class to generate a k-itemsets.

The Apriori algorithm is presented in Fig. 4.

Algorithm: Apriori
Input: Database
Output: Large itemsets

```
1)  L1 = {large 1-itemsets};
2)  for ( k = 2; Lk-1 ≠ ∅; k++ ) do begin
3)    Ck = apriori_gen(Lk-1);  // New candidates
4)    forall transaction t ∈ D do begin
5)      Ct = subset(Ck, t);  // Candidates in t
6)      forall candidate c ∈ Ct do
7)        c.count++;
8)      end
9)    Lk = {c ∈ Ck | c.count ≥ minsup};
10)   end
11) end
12) Answer = ∪k Lk;
```

Fig. 4. Pseudo code of Apriori Algorithm

The procedure apriori_gen used in step 3 is described in (2).

3 Related work

The vertical binary representations (VTV) and the corresponding support counting method have been investigated by other researchers [7], [8], [10], [12], [22].

Zaki *et al.* proposed several algorithms using vertical binary representations in 1997 [22]. Their improvements are obtained clustering the database and applying an Apriori-like method with simple tid-vectors.

Gardarin *et al.* proposed two breadth-first search algorithms using vertical binary representations named N-BM and H-BM in 1998 [8]. N-BM considers simple (uncompressed) vertical binary representations for itemsets. Meanwhile, H-BM uses, besides, an auxiliary bit-vector, where each bit represents a group of bits of the original bit-vector. In order to save the memory, every 1-itemset has both, while every large itemset keeps only the auxiliary bit-vector. H-BM first performs the AND between auxiliary bit-vectors and only non-zero

groups are considered to the final count. However, as in large itemsets only auxiliary bit-vectors are stored; the bit values of the items considered in the itemset need to be checked.

Burdick *et al.* proposed a depth-first search algorithm using vertical binary representation named MAFIA in 2001 [7]. They use bit-vectors, and it is an efficient algorithm; but only for finding maximal frequent itemsets, and especially in dense databases. Mining only maximal frequent itemsets has the following deficiency: From them we know that all its subsets are frequent, but we do not know the exact value of the supports of these subsets. Therefore, we can not obtain all the possible association rules from them.

Shenoy *et al.* proposed another breath-first search algorithm, called VIPER, using vertical representations. Although VIPER used a compressed binary representation on disk, when these compressed vectors are processed in memory they are converted right away into tid-lists, not considering advantages of boolean operations over binary formats [20].

Many other researchers have proposed other vertical binary algorithms, although the above mentioned are to the best of our knowledge the most representatives.

The method we present in this paper, CBMine, obtains all frequent itemsets faster than these well-known Apriori and vertical binary implementations, outperforming them considerably, especially for sparse databases.

4 CBMine algorithm

A new method applied to Apriori-like algorithms, named CBMine (Compressed Binary Mine), is analyzed in this section.

4.1 Storing the transactions

Let us call the itemset that is obtained by removing infrequent items from a transaction the filtered transaction. The size of the filtered transactions is declared to be "substantially smaller than the size of database". Besides, all frequent itemsets can be determined even if only filtered transactions are available.

The set of filtered transactions can be represented as an m x n matrix where m is the number of transactions and n is the number of frequent items (see Fig. 5 for an 8x5 matrix). We can denote the presence or absence of an item in each transaction by a binary value (1 if it is present, else 0).

This representation has been considered as a logical view of the data. Nevertheless, some researchers have employed it for counting the support for an item and for generating the set of 1-frequent itemsets [15].

To reduce I/O cost and speed up the algorithm, the filtered transactions could be stored in main memory instead of on disk. Although this is a reasonable solution, any data structure could require a considerable - and probably a prohibitive - memory space for large databases.

Tid	Items
1	1 2 3 5
2	2 3 4 5
3	3 4 5
4	1 2 3 4 5
5	4 5
6	2 3 4
7	2 4 5
8	1 2 4 5

1 2 3 4 5
1 1 1 0 1
0 1 1 1 1
0 0 1 1 1
1 1 1 1 1
0 0 0 1 1
0 1 1 1 0
0 1 0 1 1
1 1 0 1 1

Fig. 5. Horizontal layout of the database

1 2 3 4 5
1 1 1 0 1
0 1 1 1 1
0 0 1 1 1
1 1 1 1 1
0 0 0 1 1
0 1 1 1 0
0 1 0 1 1
1 1 0 1 1

Item	Tid-vector	Array
1	1 0 0 1 0 0 0 1	{0x91}
2	1 1 0 1 0 1 1 1	{0xD7}
3	1 1 1 1 0 1 0 0	{0xF4}
4	0 1 1 1 1 1 1 1	{0x7F}
5	1 1 1 1 1 0 1 1	{0xFB}

Fig. 6. Vertical binary representation of a transaction database (word size = 8)

Considering the standard binary representation of the filtered transactions, we propose to represent these transactions vertically and store them in main memory as an array of integer numbers (a VTV organization). It should be noticed that these numbers are not defined by row but by column (see Fig. 6). The reasons for this orientation will be explained later on.

If the maximum number of transactions were not greater than a word size, the database could be stored as a simple set of integers; however, a database is normally much greater than a word size, and in many cases very much greater. For that reason, we propose to use a list of words (or integers) to store each filtered item.

Let T be the binary representation of a database, with n filtered items and m transactions. Taking from T the columns associated to frequent items, each item j can be represented as a list I_j of integers (integer-list) of word size w, as follows:

$$I_j = \{W_{1,j}, ..W_{q,j}\}, q = \lceil m/w \rceil, \tag{5}$$

where each integer of the list can be defined as:

$$W_{s,j} = \sum_{r=1}^{min(w,m-(s-1)*w)} 2^{(w-r)} * t_{((s-1)*w+r),j}. \quad (6)$$

The upper expression $min(w, m-(s-1)*w)$ is included to consider the case in which the transaction number $(s-1)*w+r$ doesn't exist due to the fact that it is greater than m.

This binary representation for items, as noted Burdick *et al.*, naturally extends to itemsets [7]. Suppose we have an integer-list A_X for an itemset X, the integer-list $A_{X\cup\{j\}}$is simply the bitwise AND of the integer-lists A_X and I_j. If an itemset has a single item then its integer-list is I_j.

The weakness of a the vertical binary representation is the memory spending, especially in sparse databases. An alternative representation is considering only non null integers. This compressed integer-lists could be represented as an array CA of pairs $< s, B_s >$ with $1 \leq s \leq q$ and $B_s \neq 0$.

4.2 Algorithm

CBMine is a breadth-first search algorithm with a VTV organization, considering compressed integer-lists for itemset representation.

This algorithm iteratively generates a prefix list PL_k. The elements of this list have the format: $< Prefix_{k-1}, CA_{Prefix_{k-1}}, Suffixes_{Prefix_{k-1}} >$, where $Prefix_{k-1}$ is a (k-1)-itemset, $CA_{Prefix_{k-1}}$ is the corresponding compressed integer-list, and $Suffixes_{Prefix_{k-1}}$ is the set of all suffix items j of k-itemsets extended with the same $Prefix_{k-1}$, where j is lexicographically greater than every item in the prefix and the k-itemsets extended are frequent . This representation not only reduces the required memory space to store the integer-lists but also eliminates the Join step described in (3).

CBMine guarantees a significant memory reduction. Other algorithms with VTV representation have an integer-list for each itemset, meanwhile CBMine has an integer-list only for each equivalent class (see Fig. 7).

The Prune step (4) is optimized generating PL_k as a sorted list by the prefix field and, for each element, by the suffix field.

In order to determine the support of an itemset with a compressed integer-list CA, the following expression is considered:

$$Support(CA) = \sum_{<s,B_s>\in CA} BitCount(B_s), \quad (7)$$

where $BitCount(B_s)$ represents a function that calculates the Hamming Weight of each B_s.

Although this algorithm uses compressed integer-lists of non null integers (CA) for itemset representation, in order to improve the efficiency, we maintain the initial integer-lists (including the null integers) $I_j = \{W_{1,j}, .., W_{q,j}\}$ associated with each large 1-itemset j. This consideration allows directly accessing in I_j the integer position defined in CA.

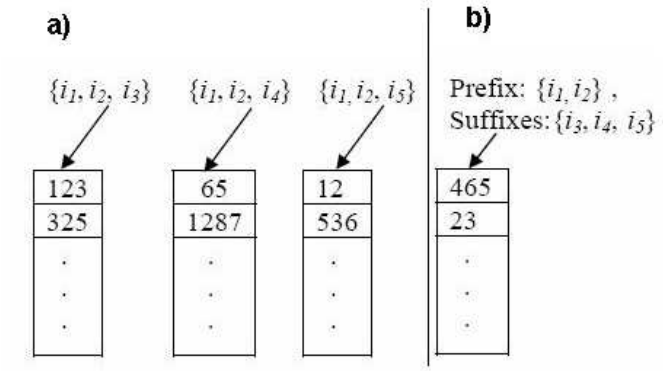

Fig. 7. a) others algorithms with VTV representations, b) CBMine

The above consideration allows defining the formule (8). Notice that this function represents a significant difference and improvement respect other methods.

$$CompAnd(CA, I_j) = \{< s, B_s^{'} >:< s, B_s > \in CA, \\ B_s^{'} = B_s and W_{s,j}, B_s^{'} \neq 0\}. \tag{8}$$

It could also be noticed that the cardinality of CA is reduced with the increment of the size of the itemsets due to the downward closure property. It allows the improvement of the above processes (7) and (8).

The CBMine algorithm is presented in Fig. 8.

The step 2 of the pseudo code shown in Fig. 8 (the process for $k = 2$) is performed in a similar way to that for $k \geq 3$, except the Prune procedure because it is unnecessary in this case. This procedure Prune, used in step 9, is similar to (4). Notice that this algorithm only scans the database once in the first step.

5 Experimental results

Here we present the experimental results of an implementation of the CBMine algorithm. It was compared with the performance of two Apriori implementations made by Ferenc Bodon (Simple and Optimized algorithms) [5], and MAFIA [7].

Four known databases were used: T40I10D100K, T10I4D100K, generated from the IBM Almaden Quest research group, Chess and Pumsb*, prepared by Roberto Bayardo from the UCI datasets and PUMSB (see Table 1).

Our tests were performed on a PC with a 2.66 GHz Intel P4 processor and 512 Mbytes of RAM. The operating system was Windows XP. Running times were obtained using standard C functions. In this paper, the runtime includes both CPU time and I/O time.

Algorithm: CBMine
Input: Database
Output: Large itemsets

```
1)  L1 = {large 1-itemsets}; // Scanning the database
2)  PL2 = {< Prefix1, CA_Prefix1, Suffixes_Prefix1 >};
3)  for ( k = 3; PL_k-1 ≠ ∅; k++ ) do
4)    forall <Prefix, CA, Suffixes> ∈ PL_k-1 do
5)      forall item j ∈ Suffixes do begin
6)        Prefix' = Prefix ∪ {j};
7)        CA' = CompAnd(CA, I_j);
8)        forall (j' ∈ Suffixes) && (j' > j) do
9)          if Prune(Prefix' ∪ {j'}, PL_k-1) &&
10)             Support(CompAnd(CA', I_j')) ≥ minsup
11)         then Suffixes'= Suffixes' ∪ {j'};
12)     if Suffixes' ≠ ∅
13)     then PL_k = PL_k ∪ {<Prefix', CA', Suffixes'>};
14)     end
15) Answer = ∪_k L_k; // L_k is obtained from PL_k
```

Fig. 8. Pseudo code of CBMine Algorithm

Table 1. Database characteristics

	T10I4D100K	T40I10D100K	Pumsb*	Chess
AvgTS	10.1	39.54	50	37
MaxItems	870	942	2.087	75
Transactions	100000	100000	49046	3196

Table 2 and the following graphics present the test results of the Apriori implementations, MAFIA and CBMine with these databases. Each test was carried out 3 times; the tables and graphics show the averages of the results. Bodon's Apriori implementations were versions on April 26th, 2006 [5]. MAFIA implementation was a version 1.4 on February 13th, 2005; it was run with "-fi" option in order to obtain all the frequent itemsets [13].

CBMine beats the other implementations in almost all the times. It performs best results independently of the support threshold in sparse databases (T40I10D100K and T10I4D100K). Nevertheless, we have verified that MAFIA beats CBMine for low thresholds in less sparse databases (Chess and Pumsb*).

Table 2. Performance results (in seconds)

databases	*minsup*	CBMine	Simple-Apriori	Optimized-Apriori	MAFIA
T10I4D100K	0.0004	17	32	20	52
	0.0003	25	49	39	64
	0.0002	51	198	86	104
	0.0001	135	222	192	287
T40I10D100K	0.0500	3	3	3	8
	0.0400	5	6	6	11
	0.0300	6	7	7	16
	0.0100	17	31	20	38
	0.0090	28	95	57	56
	0.0085	32	104	66	67
Pumsb*	0.7	2	2	1	2
	0.6	2	5	1	2
	0.5	2	11	5	2
	0.4	2	28	24	2
	0.3	23	106	47	11
Chess	0.9	0	3	2	0
	0.8	0	8	2	0
	0.7	1	45	3	1
	0.6	2	92	22	2
	0.5	16	163	53	7

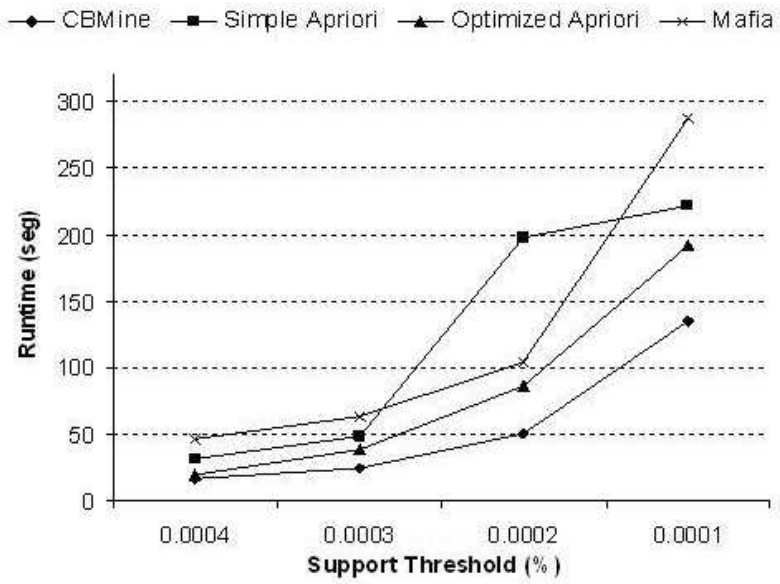

Fig. 9. T10I4D100K database

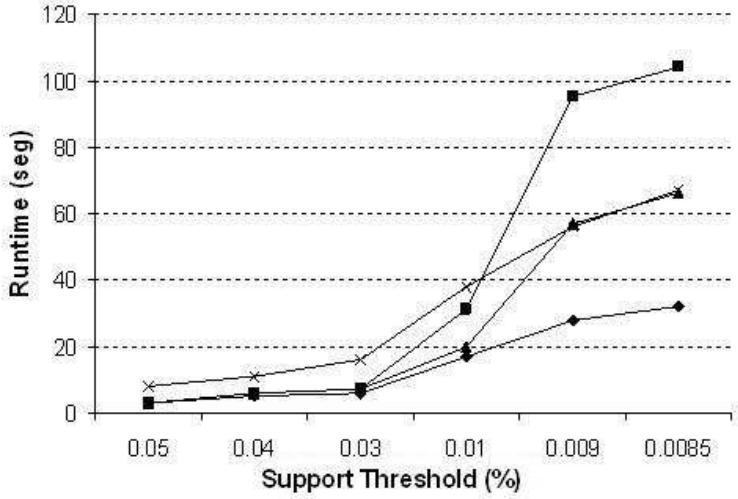

Fig. 10. T40I10D100K database

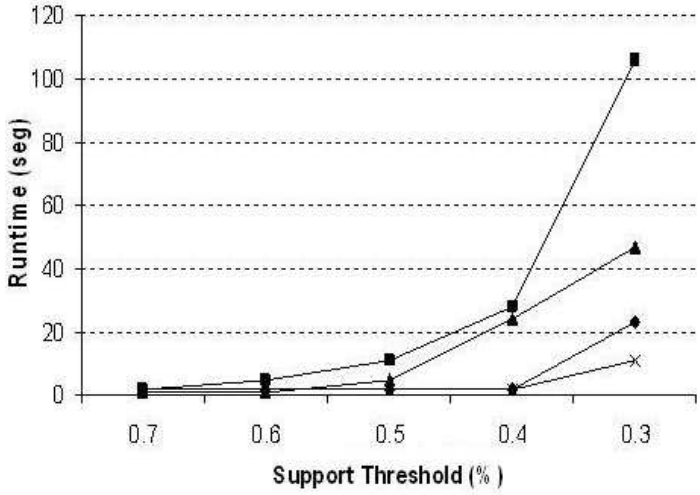

Fig. 11. Pumsb* database

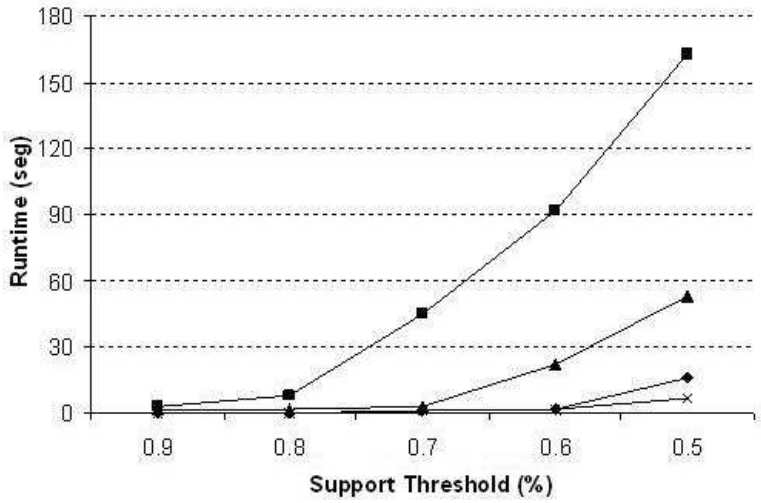

Fig. 12. Chess database

6 Conclusions

The discovery of frequent objects (itemsets, episodes, or sequential patterns) is one of the most important tasks in data mining. The ways databases and its candidates are stored cause a crucial effect on running times and memory requirements.

In this paper we have presented a compressed vertical binary approach for mining several kinds of databases. Our experimental results show that the inclusion of this representation in Apriori-like algorithms makes them more efficient and scalable.

We presented a method that obtains frequent itemset faster than other well-known Apriori implementations and vertical binary implementations, outperforming them considerably, especially for sparse databases.

There are many other issues to be analyzed using a vertical compressed binary approach. This is our goal, and these issues will be included in further papers.

References

1. Fast algorithms for frequent itemset mining using fp-trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(10):1347–1362, 2005. Member-Gosta Grahne and Student Member-Jianfei Zhu.
2. Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Ja-

jodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 26–28 1993.
3. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.
4. Ferenc Bodon. Surprising results of trie-based fim algorithms. In Bart Goethals, Mohammed J. Zaki, and Roberto Bayardo, editors, *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'04)*, volume 126 of *CEUR Workshop Proceedings*, Brighton, UK, 1. November 2004.
5. Ferenc Bodon. Trie-based apriori implementation for mining frequent itemsequences. In Bart Goethals, Siegfried Nijssen, and Mohammed J. Zaki, editors, *Proceedings of ACM SIGKDD International Workshop on Open Source Data Mining (OSDM'05)*, pages 56–65, Chicago, IL, USA, August 2005.
6. Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. In Joan Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 255–264. ACM Press, 05 1997.
7. Douglas Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 443–452, Washington, DC, USA, 2001. IEEE Computer Society.
8. G. Gardarin, P. Pucheral, and F. Wu. Bitmap based algorithms for mining association rules, 1998.
9. Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.
10. Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining — a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, July 2000.
11. John D. Holt and Soon M. Chung. Multipass algorithms for mining association rules in text databases. *Knowl. Inf. Syst.*, 3(2):168–183, 2001.
12. Tsau Young Lin. Data mining and machine oriented modeling: A granular computing approach. *Appl. Intell.*, 13(2):113–124, 2000.
13. Calimlim M. and Gehrke J. Himalaya data mining tools: Mafia. http://himalaya-tools.sourceforge.net, May 2006.
14. Fayyad U. M., Piatetsky-Shapiro G., and Smyth P. From data mining to knowledge discovery: An overview. In Fayyad U. M., Piatetsky-Shapiro G., Smyth P., and Uthurusamy R., editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI Press, 1996.
15. Gopalan R. P. and Sucahyo Y. G. High performance frequent patterns extraction using compressed fp-tree. In *Proceedings of the SIAM International Workshop on High Performance and Distributed Mining*, Orlando, USA, 2004.
16. Feldman R. and Hirsh H. *Machine Learning and Data Mining: Methods and Applications*, chapter Finding Associations in Collections of Text, pages 223–240. John Wiley and Sons, 1998.
17. Feldman R., Dagen I., and Hirsh H. Mining text using keyword distributions. *Journal of Intelligent Information Systems*, 10(3):281–300, 1998.

View publication stats

18. Chen M. S., Han J., and Yu P. S. Data mining: An overview from a data-base perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, 1996.
19. Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In *The VLDB Journal*, pages 432–444, 1995.
20. Pradeep Shenoy, Jayant R. Haritsa, S. Sundarshan, Gaurav Bhalotia, Mayank Bawa, and Devavrat Shah. Turbo-charging vertical mining of large databases. pages 22–33, 2000.
21. Cheung D. W., Han J., Ng V. T., and Wong C Y. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of the 12th IEEE International Conference on Data Engi-neering*, pages 106–114. IEEE, 1996.
22. Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. Technical Report TR651, 1997.