Fast Frequent Pattern Mining in Real-Time

Rajanish Dass
Indian Institute of Management Calcutta
email: rajanish@iimcal.ac.in

Ambuj Mahanti
Indian Institute of Management Calcutta
email: am@iimcal.ac.in

ABSTRACT

Finding frequent patterns from databases has been the most time consuming process of the association rule mining. Till date, a large number of algorithms have been proposed in the area of frequent pattern generation. However, all of these algorithms produce output only at the completion and are not amenable to the real-time need. The need for real-time frequent pattern mining for online tasks and real-time decision-making is increasingly being felt. In this paper, we describe BDFS(b), an algorithm to perform real-time frequent pattern mining using limited computer memory. Empirical evaluations show that our algorithm can make a fair estimation of the probable frequent patterns and reaches some of the longest frequent patterns much faster than the existing algorithms.

1. INTRODUCTION

Since its inception in 1993 by Agarwal et al., association rule mining for large databases of business data, such as transaction records, is of great interest in data mining and knowledge discovery [1]. An association rule is an expression of the form $X \Rightarrow Y$, where *X* and *Y* are sets of items. Such a rule reveals that the transactions in the database, containing items in *X* tend to contain items in *Y*, and the probability, measured as the fractions of the transactions containing *X* also containing *Y*, is called the confidence of the rule. The support of the rule, is the fraction of the total transactions that contain all items both in *X* and *Y*.

ADVANCES IN DATA MANAGEMENT 2005
Jayant Haritsa, T.M. Vijayaraman (Editors)

For an association rule to hold, the support and the confidence of the rule should satisfy a *user-specified* minimum support and minimum confidence. The problem of mining association rules is to discover all rules that satisfy this user-specified minimum support and minimum confidence. In this paper, we assume that the reader knows the basic assumptions and terminologies of association mining.

However, it is noteworthy at this point that the work of association rule mining can be decomposed into two phases:

1. Frequent itemsets generation: Find out all itemsets (or group of parameters) that exceed the given minimum support
2. Rules construction: From the frequent itemsets generated in step 1 above, generate all association rules having confidence higher than the given minimum confidence.

As the second phase mentioned above is straightforward and less-expensive, researchers have generally focused on the first phase itself. The search space needed for finding all frequent itemsets is undoubtedly huge [6]. A number of efficient algorithms have been proposed in the last few years to make this search fast and accurate[7]. However, most of the algorithms stop only after finding the exhaustive (optimal) set of frequent itemsets. These algorithms have been very efficient and scalable for many real-life applications and are based on the "collect-store-analyze" model. In all these, data mining is typically considered to be an offline analytical task. These algorithms do not have the ability to run under user defined real-time constraints and produce some satisficing (interesting sub-optimal) solutions. With the increasing demand of real-time applications in various fields of business today, development of real-time data mining algorithms demand attention.

In this paper, we describe BDFS(b) (adopted from [20]), a real-time frequent pattern mining algorithm which runs under limited computer memory. We also show its edge over existing efficient association mining algorithms such as FP tree, when it runs to completion and outputs exhaustive set of frequent patterns.

The rest of the paper is organized as follows. In the next section we discuss on the importance of real-time frequent pattern mining in businesses. In Section 3, we present a review of the previous work in association rule mining. In Section 4, we introduce algorithm BDFS(b). Section 5 contains the empirical evaluation of our algorithm. Finally, we conclude the paper in Section 6.

2. NEED FOR REAL-TIME FREQUENT PATTERN MINING IN BUSINESS

In recent years, business intelligence systems are playing pivotal roles in fine-tuning business goals such as improving customer retention, market penetration, profitability and efficiency. In most cases, these insights are driven by analyses of historic data. Now the issue is, if the historic data can help us make better decisions, how real-time data can improve the decision making process [8] ?

An offline approach to data mining reflects sound practice because the data have to be cleaned, checked for accuracy, etc. However, in a scenario of cutthroat competition, the organizations cannot afford to show the attitude of not keeping abreast with the latest changing demands and trends of their customers and get satisfied with periodical data. They have to act on the latest data that is available to them to react not only to the fierce global competition, but also market products keeping in mind of the latest customer wishes. In such a scenario, the concept of a real-time enterprise has creped into the corporate boardrooms of a number of organizations. Using up-to-date information, getting rid of delays, and using speed for competitive advantage is what the real-time enterprise is about [5].

In the following sub-sections, we discuss the importance of real-time frequent pattern mining in some common business applications.

2.1 Real-Time CRM

Due to the change in the focus of marketing from mass marketing to more of relationship marketing, Customer Relationship Management (CRM) has become a major focus and thrust area for most of the companies, both online and offline. It costs five to seven times more to find new customers than to retain current customers. A 5% reduction in customer defection can result in profit increases from 30% to 85%. If companies increase their customer retention by 2%, it is the equivalent of cutting their operating expenses by 10% [3].

In this context, the extraction of hidden patterns from large databases help the organizations to identify customers, predict their future behaviors and enable firms to take proactive and knowledge-driven decisions. However, one important thing to be noted is that the companies do not have infinite time to run data mining tools on huge transactional databases and data warehouses to look into the patterns or come up with offers for the customers who come to visit their stores. This applies to both online and offline business outlets. Companies have understood the need for real-time CRM and that real-time analysis of the buying habits is desperately needed to make relevant offers to a particular concerned customer, before the customer leaves the outlet and ends the transaction. Researchers [15] believe that real-time personalization technology will proactively offer a particular customer products and services that will fit into her needs exactly. A real-time analytical engine will work in real-time, analyzing web clicks or sales rep interactions and matching them with the past purchasing history to make the offerings.

2.2 Real-Time Recommender Systems

The explosive growth of the world-wide-web and the emergence of e-commerce have led to the development of *recommender systems* [8], which is a variant of real-time CRM. Recommender systems are personalized information filtering technology used to either predict if a particular user will like a particular item (*prediction problem)* or to identify a set of N-items that will be of interest to a particular user *(top-N recommendation system)* [29].

In recent years, recommender systems have been used in a number of different applications. Recommending products that a customer is most likely to buy such as movies, books, music, TV programs, and restaurant recommendations, demonstrating the wide range of application domains of existing recommender systems a user will find enjoyable, identifying the web pages that will be of interest, or even suggesting alternate ways of searching for information[18]. In particular, Schafer et al. [24] made an elegant survey on major existing systems and approaches to e-commerce recommendation.

Despite the popularity of the recommender systems, they have a number of limitations related to their scalability and real-time performance. As a typical example, the SmartPad system developed at IBM [16] makes recommendations using a model of association rules. The model is built on the past eight weeks of data from Safeway Stores and is updated weekly or quarterly to reflect seasonal differences. Hence, Shen et al. [26] observes that most existing data mining approaches to e-commerce recommendation are past data model-based in the sense that they first build a preference model from a past dataset and then apply the model to current customer situations. Such approaches are not suitable for applications where fresh data should be collected and

processed instantly since it reflects changes to customer preferences over some products.

There are numerous other areas where real-time decision making plays a crucial role. These include areas like real-time supply chain management [13], real-time enterprise risk and vulnerability management [21], real-time stock management and vendor inventory [25], real-time operational management with special applications in mission critical real-time information as is used in the airlines industry, real-time intrusion and real-time fraud detection [17], real-time negotiations and other areas like real-time dynamic pricing and discount offering to customers in real-time. More than that, real-time data mining will have tremendous importance in areas where a real-time decision can make the difference between life and death – mining patterns in medical systems.

3. PREVIOUS WORK

Association rule mining was introduced by Agrawal et al. [1]. A detailed discussion about the various algorithms of frequent pattern mining and their performance can be found in the literature surveys of frequent pattern mining [7, 9, 12].

It is noteworthy at this point that the total search space for all frequent itemsets is huge. Instead of generating and counting the supports of all possible itemsets at once, which is obviously infeasible, several solutions have been proposed to perform a more directed search by iteratively generating and counting sets of *candidate* itemsets [6].

The most well known and influential algorithms are Apriori[1] and FP-growth [10]. Apriori uses an a-priori knowledge of frequent *k*-itemsets to generate candidate itemsets of length *(k+1)* and employs an innovative technique for pruning non-promising candidates. However, the most discussed drawback of this algorithm is that when the cardinality of the longest frequent itemsets is *k*, Apriori needs *k* passes of database scans. FP-growth, however, uses a depth-first strategy for finding frequent patterns without generating any candidate itemset. It constructs an FP-tree with itemsets above the user-given support and then recursively mines the constructed Fp-tree to find out all patterns. FP-growth makes only 2 scans of the database for finding all the frequent patterns. Many variants of Apriori algorithm have been designed. Other ways of solving the problem were using various partitioning methods and sampling methods. These implementations included algorithms like Apriori-TID[2], Apriori-Hybrid[2], Partition [23], Sampling [28], DIC [4], CARMA [11], ECLAT [19], and Top-Down [19]among others. Although CARMA [Continuous Association Rule Mining Algorithm) uses a similar technique like DIC (to divide the database into intervals of a specific size) reducing the interval size to 1. More specifically, candidate itemsets are generated on the fly from every transaction. After reading a transaction, it increments the supports of all candidate itemsets contained in that transaction and it generates a new candidate itemset contained in that transaction, if all of its subsets are suspected to be relatively frequent with respect to the number of transactions that has already been processed. As a consequence, CARMA generates a lot more candidate itemsets than Apriori. Again, to determine the exact supports of all generated itemsets, another full scan of the database is required [7].

Majority of the algorithms in this area have been classified according to their strategy to traverse the search space and by their strategy to determine the support values of the itemsets [12]. However, [27] has concluded that the most salient features of these algorithms are their *counting strategy*, *search direction* and *search strategy*. Horizontal counting or vertical intersections are used for counting the occurrences of candidate itemsets. Most of the algorithms have generally used a bottom up approach in the search strategy. While applying the search strategy, the algorithms have used a breadth first or a depth first search. The above points may be summarized in the following table:

Counting Strategy	**Search Direction**			
	Bottom-up		*Top-Down*	
	Search Strategy		**Search Strategy**	
	Depth-first	*Breadth-first*	*Depth-first*	*Breadth-first*
Counting	FP-Growth	Apriori		Top-Down
Intersection	ECLAT	Partition		

Figure 1. Classification of prevailing algorithms *[27]*

4. BDFS(b): AN EFFICIENT TECHNIQUE OF FREQUENT PATTERN MINING IN REAL-TIME

4.1 Algorithm Basics

In this study, we propose a brute force algorithm, which is a variant of the Block Depth First Search[20]. We call the algorithm as BDFS(b). BDFS(b) explores the given search space in stages. The search is conducted in a depth first manner, which ensures that patterns of greater length will be preferred over those of comparatively shorter lengths.

We assume that a lower triangular frequency matrix M is created in a pre-processing step, which stores the support independent frequencies of all 1-length and 2-length patterns. Once the user specifies a desired support value, all frequent patterns of length 1 and 2 (meaning F(1) and F(2), where F(n) means frequent pattern of

length-n) are obtained from M. Then BDFS(b) starts its search for frequent patterns of higher lengths from this point forward.

The most salient features of BDFS(b) are:

(a) It conducts search in stages and uses back-tracking strategy to run to completion and ensure optimal solution.

(b) It takes a block of candidate patterns b from a global pool, conducts the search by checking the frequency of these patterns in the database. It generates the possible candidate patterns (explained later with an example) of the next higher length from the currently known frequent patterns. These candidate patterns are continued to be explored in a systematic manner until all frequent patterns are generated. The value of the block b is defined by the user using her knowledge and experience (later in the paper, we have shown how the performance of BDFS(b) is affected with changing block size b).

A possible state space diagram of BDFS(b) is shown in figure 2.

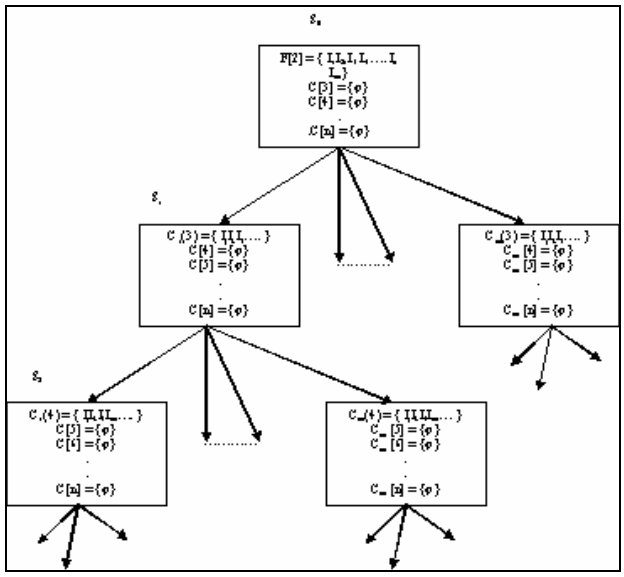

Figure 2. State space diagram for BDFS(b).

The initial state (or the root node) in the state-space is denoted by S_0, which contains the complete set of 2-length frequent patterns F(2). In S_0, the set of all candidate patterns of length 3 or more are set to ϕ. In general, by the expansion of a node (which is a block of candidate patterns in this case) we mean:

i. Counting the support frequency of all candidate patterns in the state from the database.
ii. Generating the candidate patterns or patterns of border set of next higher level (explained later in the algorithm and its working through example).
iii. Arranging the candidate patterns according to their merits (explained later) and group them into blocks containing b-patterns each. If the block has empty space, it gets candidate patterns from the previous level. This can be handled using a global pool of candidate patterns that has been sorted in descending order of length. We resolve ties arbitrarily.

4.2 Algorithm Details

Algorithm BDFS(b):

Initialize the allowable execution time τ.
Let the initial search frontier contain all 3-length candidate patterns. Let this search frontier be stored as a global pool of candidate patterns. Initialize a set called Border Set to null.
Order the candidate patterns of the global pool according to their decreasing length (resolve ties arbitrarily). Take a group of most promising candidate patterns and put them in a block b of predefined size.

- *Expand (b)*
  Expand (b: block of candidate patterns)
  If not last_level
  then
  begin
  Expand$_1$(b)
  end.

Expand$_1$(b):

1. *Count support for each candidate pattern in the block b by intersecting the t-id list of the items in the database.*
2. *When a pattern becomes frequent, remove it from the block b and put it in the list of frequent patterns along with its support value. If the pattern is present in the Border Set increase its subitemset counter. If the subitemset counter of the pattern in Border Set is equal to its length move it to the global pool of candidate patterns.*
3. *Prune all patterns whose support values < given minimum support. Remove all supersets of these patterns from Border Set.*
4. *Generate all patterns of next higher length from the newly obtained frequent patterns at step 3. If all immediate subsets of the newly generated pattern are frequent then put the pattern in the global pool of candidate patterns else put it in the Border Set if the pattern length is > 3.*
5. *Take a block of most promising b candidate patterns from the global pool.*
6. *If block b is empty and no more candidate patterns left, output frequent patterns and exit.*
7. *Call Expand(b) if enough time is left in τ to expand a new block of patterns, ,else output frequent patterns and exit.*

Figure 3. Algorithm BDFS(b)

Let us consider the following example to show how BDFS(b) works. Let us consider some market basket data to illustrate its working, which has been a conventional technique in describing many frequent pattern mining algorithms.

Let the following table represent a set of 12 transactions, where the items are represented by a, b, c …

1. a b c d e	2. a c d e	3. a d e	4. b c d e
5. b d e	6. a b d	7. a b d	8. a b c d
9. d e	10. a c d e	11. a b c d e	12. ace

Figure 4. An example of transaction data

Now we proceed as follows:

Step I. Given this set of transactions D, we create a two dimensional lower triangular matrix M using procedure *Create_Matrix* and the transaction id lists.

I. Create a lower triangular adjacency matrix, M, for n-items (Total storage required: n(n+1)/2). M stores the frequencies of 1-at-a-time and 2-at-a-time combinations of all items.*
II. In M, M(i,j) represents the number of occurrences of the item-pair i and j, ∀ i = 1,2…n and ∀ j = 1,2,3…i and M(i,i) represents the total number of occurrences of item i.

Figure 5. Procedure *Create_Matrix*

The created matrix M is depicted in figure 6. This step of creating the matrix M and the tid-list (figure 7) is a support independent step and we refer this step through out this paper as a *pre-processing step*.

Step II. Let the absolute support ξ (abs) given for running BDFS(b) be 3. This means that we are interested only in patterns, which have frequency greater than or equal to 3. Cells of Matrix M are visited to find F(1) and F(2) [where F(n) is frequent pattern of length n]. Thus we have:

F(1) = { a(9), b(7), c(7), d(11), e(9)} …… (1)

F(2) = { ab(5), ac(6), ad(8), ae(6), bc(4), bd(7), be(4), cd(6), ce(6), de(8)} …… (2)

Frequency of each pattern is shown within parentheses. Thus the pattern *e* of F(1) has frequency 9 and *bd* of F(2) has frequency 7.

Step III. Two 2-length patterns are merged if their first elements match.

Thus *newly merged patterns = { abc, abd, abe, acd, ace, ade, bcd, bce, bde, cde }* …… (3)

Step IV. Find if all the subsets of new merged patterns are frequent. For any 3-lemgth newly merged pattern, if all its 2-length subsets are not present, then the pattern is pruned (using the support monotonicity property[18]). Otherwise, if all its 2-length subsets are present the pattern becomes a *candidate-pattern* and it is moved to the *global-pool* of candidate patterns *C( )*. The global-pool of candidate patterns is sorted on length and any tie between two same length patterns is resolved arbitrarily.

C ( ) = { abc, abd, abe, acd, ace, ade, bcd, bce, bde, cde } …… (4)

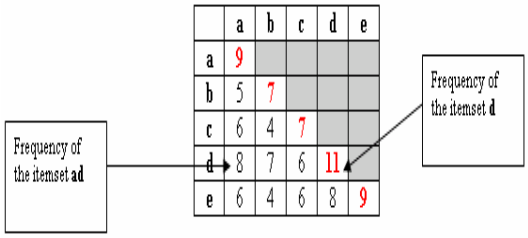

Figure 6. Matrix M

Item	Transaction Ids										
a	1	2	3	6	7	8	10	11	12		
b	1	4	5	6	7	8	11				
c	1	2	4	8	10	11	12				
d	1	2	3	4	5	6	7	8	9	10	11
e	1	2	3	4	5	10	11	12			

Figure 7. The tid-list of the items

Step V. Let us assume that the block size b is 4, which means that we can take at most 4 patterns into a block for checking their frequency. This means as the 3-length candidate patterns are pushed into the global pool, 4 of these patterns namely, *abc, abd, abe* and *acd,* will be put in the next block b.

Step VI. We now check the frequency of these patterns by intersecting the tid-lists of the items.

b = {abc (3), abd (5), abe (2), acd (5)} ..... (5)

As frequency of *abe* is less than the support threshold, it gets pruned.

F (3) = {abc (3), abd (5), acd (5)} …… (6)

Step VII. We now merge the newly found frequent patterns in *F(3)* and test these newly merged patterns generated for the presence of their immediate subsets..

Newly merged patterns = { abcd } …… (7)

We now find that all immediate subsets of the pattern *abcd* are not present in *F(3)*. But only three immediate subsets are present. Hence we move the pattern *abcd* to border set of length 4, *BS (4)*, with a sub-itemset counter of 3.

BS (4) = { abcd (sub-itemset = 3) } ……(8)

Patterns *ace, ade, bcd, bce* are taken in the next block b from the global-pool of candidate patterns.

b={ace(5),ade(5),bcd(4),bce(3)}..... (9)

We find that all these items have frequency greater than ξ(abs) = 3 and are hence frequent. Thus from the new block

F(3)={ ace(5) ,ade(5), bcd(4), bce(3)}……..... (10)

For each pattern in the current *F(3)*, we search *BS (4)* to see if any of the immediate supersets are waiting in the border set. We find that the pattern *abcd* is in *BS (4)* with sub-itemset counter = 3. Hence we increase the sub-itemset counter of *abcd* and make it 4. The pattern *abcd* is of the highest length among the candidate patterns in the global-pool and is put in the next block b.

Newly merged patterns (4) = {acde ,bcde } ……(11)

The number of frequent immediate subsets of *acde* and *bcde* are 3 and 2 respectively. Hence they are moved to *BS (4).*

BS (4) = { acde (sub-itemset = 3), bcde (sub-itemset = 2)}……. (12)

The patterns *abcd, bde* and *cde* go to the current block b. After intersecting the tid-list of these patterns, we find that

F (4) = {abcd (3)} ……..…… (13)
F (3) = {bde (3),cde (5)} …… (14)

Similarly we search the BS (4) with newly found F(3) patterns and merge the patterns in the newly found F(3)'s with previous F(3)'s to generate higher length patterns. We find that *acde* and *bcde* move from BS (4) to global pool of patterns and moves into the block b. By intersecting the tid-lists of the items, we find that

F(4)={acde (4), bcde (3)} ……..…. (15)

As no higher length patterns can be generated and the number of patterns in block b becomes zero and also the number of candidate patterns in the global pool of candidate patterns becomes zero, the algorithm stops executing here. Thus, the set of all frequent patterns are:

F(1) = { a(9), b(7), c(7), d(11), e(9)}

F(2) = { ab(5), ac(6), ad(8), ae(6), bc(4), bd(7), be(4), cd(6), ce(6), de(8)}

F(3) = { abc (3), abd (5), acd (5), ace(5) ,ade(5), bcd(4), bce(3), bde (3),cde (5)}

F(4) = { abcd (3), acde (4), bcde (3)}

The block size b can now be varied to show how it affects the execution time of the algorithm. In the next section, we show and discuss this effect. BDFS(b) has the capability to run in real-time. Whenever it is stopped before its natural completion, it outputs frequent patterns of various lengths it had obtained up to that point of execution time.

5. EMPIRICAL EVALUATIONS

Legend:
T= Average size of transaction; I= Average size of the maximal potentially large itemset; D= No. of transactions in the database; N= Number of items.

In order to show how BDFS(b) performs, when it is run to generate all frequent patterns, we have chosen to compare it with FP-growth and Apriori. Since FP-growth is known to be an order faster and scales better than Apriori[10], we have taken FP-growth as the benchmark and compared its execution time with that of BDFS(b), ,implemented with a TRIE data structure. For the sake of curiosity we have also compared Apriori and BDFS(b) but for their number of patterns checked. The experiments were performed on a Linux machine with 1GB RAM and 20 GB HD.

5.1 Performance of BDFS(b) on Synthetic Datasets

Experimental evaluation of BDFS(b) has been performed on several synthetic datasets like: T10I8D100K, T10I8D10K, T10I8D1K, T10I2D100K, T6I5D10K, T5I4D1K, T5I4D10K, T5I4D100K (all these datasets have 1K number of items), T5I4N500D1K, T5I4N500D10K, T5I4N500D100K (with number of items being 500), T5I4N100D1K, T5I4N100D10K, T5I4N100D100K (where the number of items is 100) etc. These datasets were generated using the IBM synthetic data generator[1] [2].

5.1.1 Comparison of BDFS(b) with Existing Algorithms

In figures 8 and 9, we have compared the run-time of FP-growth[2] and BDFS(b) for two different datasets and found that BDFS(b) compares well with FP-growth in all the

[1] The data generator is available from http://www.almaden.ibm.com/cs/quest//syndata.html#assocSynData.
[2] The FP-growth code used for comparison is publicly available at www.cse.cuhk.edu.hk/~kdd/program.html

cases. In figure 10, we have tested the scalability of the FP-growth and BDFS(b). We have observed that both the algorithms are well scalable with time and number of transactions in the database. Here again BDFS(b) takes relatively much less time than FP-growth over the same databases.

Comparing the number of patterns being checked by Apriori and BDFS(b), as shown in figure 11, it is found that BDFS(b) checks much lesser number of patterns than Apriori[3]. Although the Apriori code used here for comparison is an efficient version of Apriori with several optimizations added and the hash-tree data structure replaced with TRIE, but as BDFS(b) has the capability to find out one and two-length frequent patterns from the pre-processed matrix itself, the total number of patterns checked by BDFS(b) is lesser compared to Apriori.

5.1.2 Real-time Performance of BDFS(b)

Performance of BDFS(b) for varying values of block size b is shown in figure 12. We find that for b = 1K, 10K and 100K BDFS(b) is well scalable. When the block size is too small, say b = 1K, then it takes more running time for completion compared to b = 10K or 100K. For b = 10K or above it gives similar performance. This is quite intuitive, because a successor block can then accommodate all the candidate patterns of a parent block.

Figures 13 and 14 summarize the real-time behavior of BDFS(b) by depicting the percentage of frequent patterns generated with the percentage of total execution time. These include F(1) and F(2) obtained from the frequency matrix. Figure 14 particularly shows how the percentage of patterns generated by BDFS(b) increases with the increasing values of support for a particular percentage of its running time. Next three figures namely, 15,16 and 17 present three different scenarios of real-time performance of BDFS(b) by showing the number of patterns of different lengths obtained at different time slices (expressed as % of total execution time), using three different block sizes.

5.2 Performance of BDFS(b) on Real-life Datasets

To evaluate the performance of BDFS(b), we have tested it on various datasets. This includes real-life datasets like BMS-WebView-1, BMS-WebView-2 and BMS-POS [14]

5.2.1 Comparison of BDFS(b) with Existing Algorithms

In figures 18 and 19, we have compared the run-time of BDFS(b) and FP-growth on two different real-life databases, BMS-POS and BMS-WebView-2. We have found that BDFS(b) performs well on these datasets too.

5.2.2 Real-time Performance of BDFS(b)

In figures 20,21,22,23 and 24, we present the real-time performance of BDFS(b) on three real-life datasets namely, BMS-WebView-2, BMS-POS and BMS-WebView-1 [22]. In figure 20 we show how BDFS(b) performs on the dataset BMS-POS. Again, in figures 21 and 22, we observe the real-time performance of BDFS(b) on real-life datasets BMS-WebView-2 and BMS-WebView-1respectively. In these figures, 20, 21 and 22 we show percentage of frequent patterns generated with percentage execution time having F(1) and F(2) included and excluded in two respective curves. Similarly, in figures 23 and 24, we particularly see that the efficiency of the algorithm enhances with increase in the support value.

Figure 25 makes a tabular presentation of real-time outputs showing length-wise frequent patterns, border sets, and candidate sets, at different time slices (expressed as % of total execution time). It may be seen from the output that all the F(8) patterns (which are of maximal length in this case) were outputted only in 4.17% time. It may be noted that the over all percentage of output is almost always ahead of percentage execution time.

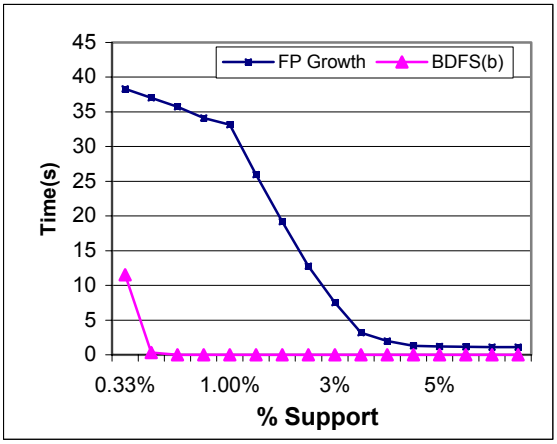

Figure 8. Time Comparison of FP-growth and BDFS(b) T10I8D100K,b =100K

[3] The Apriori code used for comparison is publicly available at http://www.cs.helsinki.fi/u/goethals/software/index.html

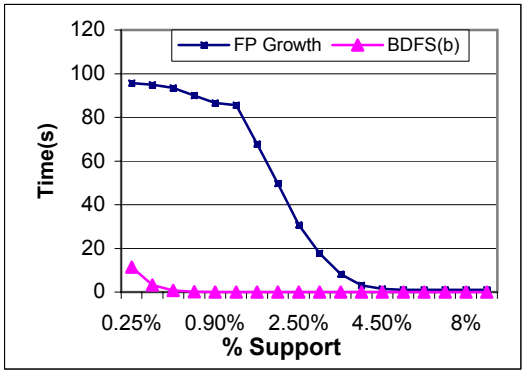

Figure 9. Time Comparison of FP-growth and BDFS(b) T10I2D100K, b=100K

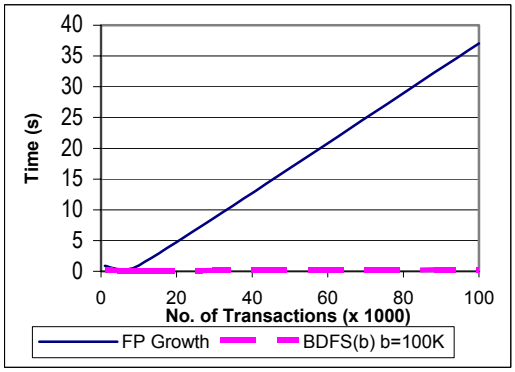

Figure 10. Scalability evaluation of FP-growth and BDFS(b) (b=100K) with varying number of transactions for T10I8 with support=0.5%

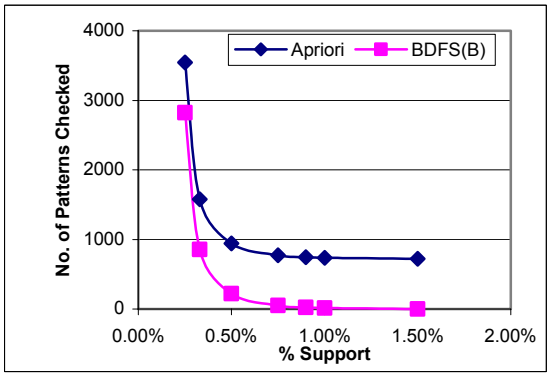

Figure 11. Number of patterns checked by Apriori and BDFS(b) for T10I2D100K with varying supports

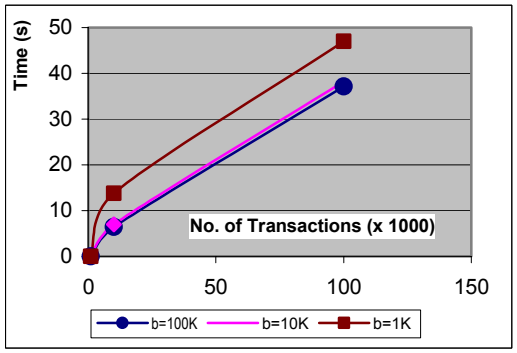

Figure 12. Time Scalability of BDFS(b) with increasing no. of transactions for T5I4D1K,10K,100K and b=1K,10K and 100K and support 0.5%

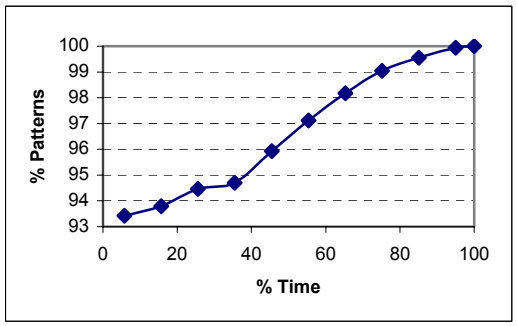

Figure 13. Time-Patterns % for 0.05% support of BDFS(b) for T6I5D10K, BDFS(b), b=1000

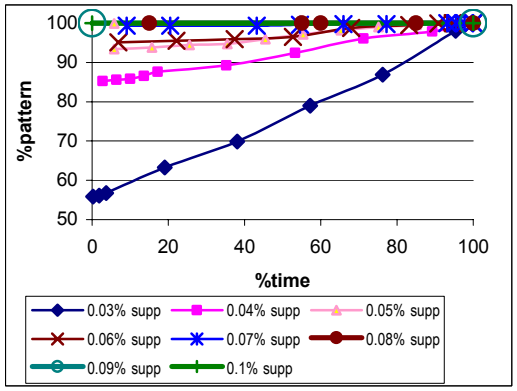

Figure 14. Time-Patterns % for varying support for T6I5D10K, BDFS(b), b=1000

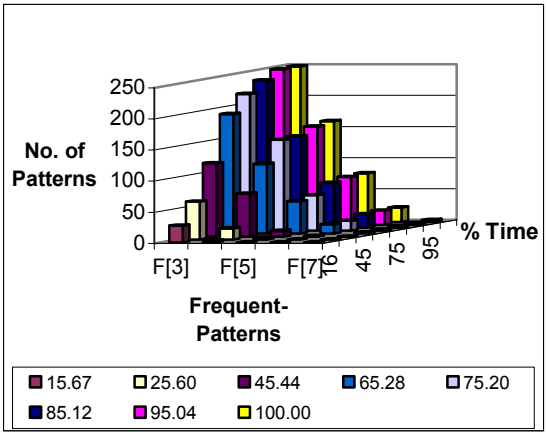

Figure 15. Real-time output of frequent patterns of T6I5D10K, support 0.05%, BDFS(b), b=1000

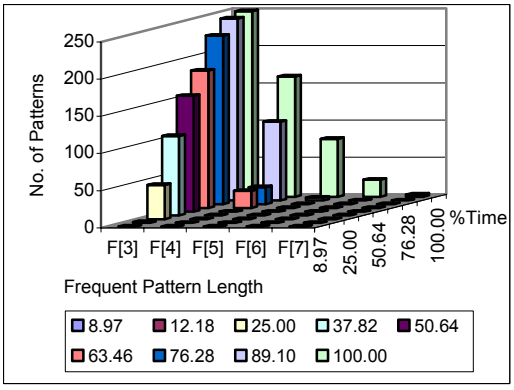

Figure 16. Real-Time output of frequent patterns of T6I5D10K, support 0.05%, BDFS(b), b=10000

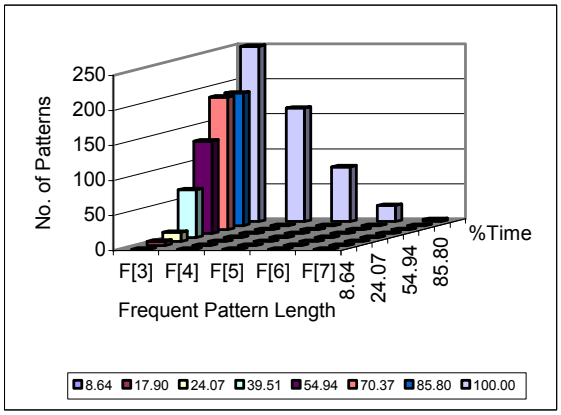

Figure 17. Real-Time output of frequent patterns of T6I5D10K, support 0.05%, BDFS(b), b=100000

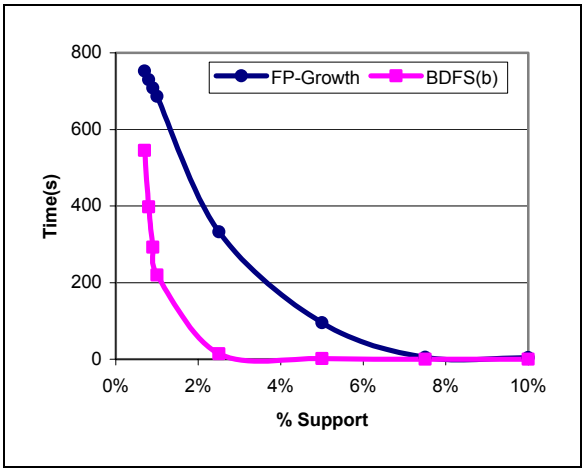

Figure 18. Time Comparison of FP-growth and BDFS(b) for BMS-POS, T=7.5, D = 515,597, N=1658

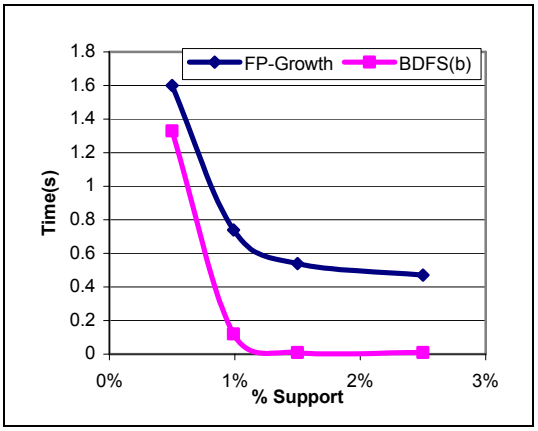

Figure 19. Time Comparison of FP-growth and BDFS(b) for BMS-Web-View-2, T=5.6, D = 77512, N=3341.

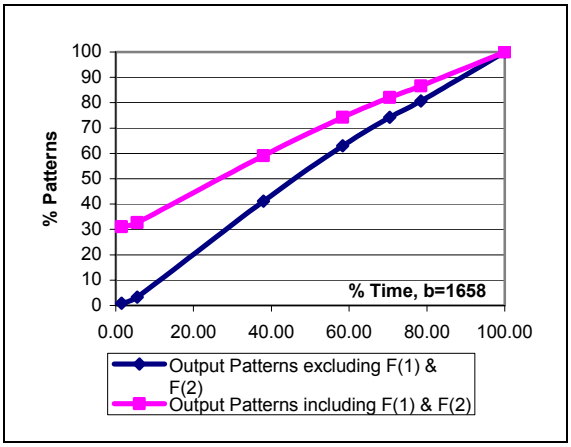

Figure 20. Time-Pattern % for 0.5% support of BDFS(b) for BMS-POS, T=7.5, D = 515,597, N=1658.

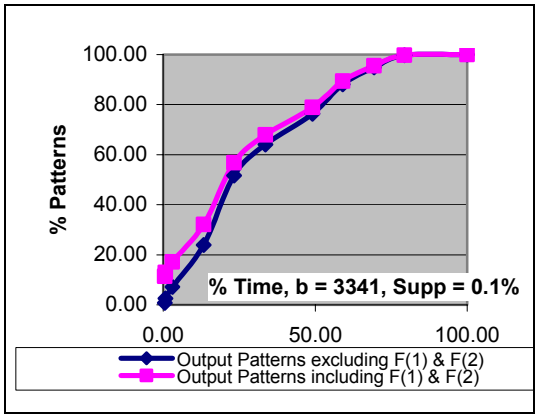

Figure 21. Time-Patterns % for 0.1% support of BDFS(b) for BMS-Web-View-2, T = 5.6, D = 77512, N =3341

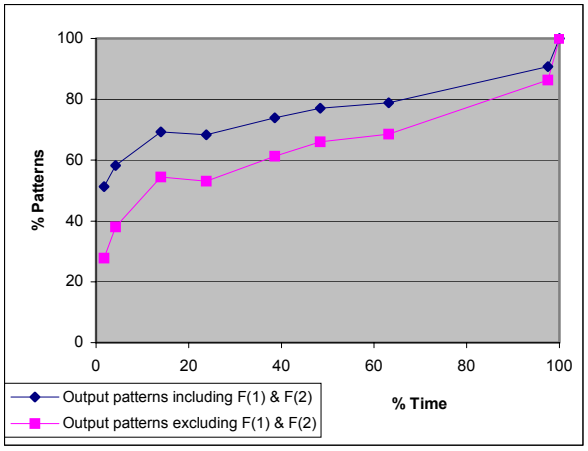

Figure 22. Time-Patterns % of BDFS(b) with b = 497 for BMS-WebView-1, N= 497, T=2.5, D=59602 with support 0.08%

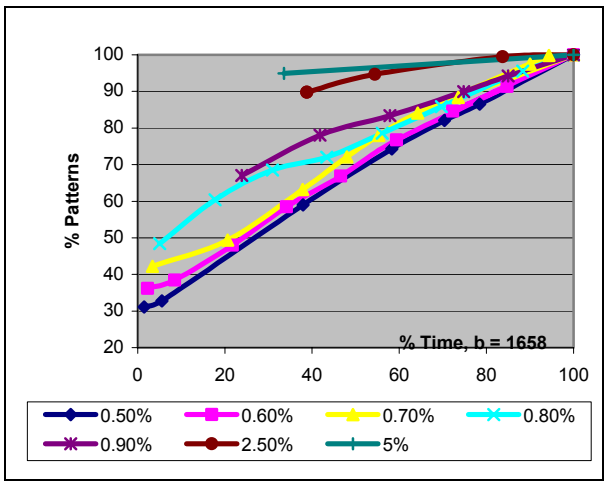

Figure 23. Time-Patterns % of BDFS(b) for varying support for BMS-POS T = 7.5, D = 515597, N = 1658

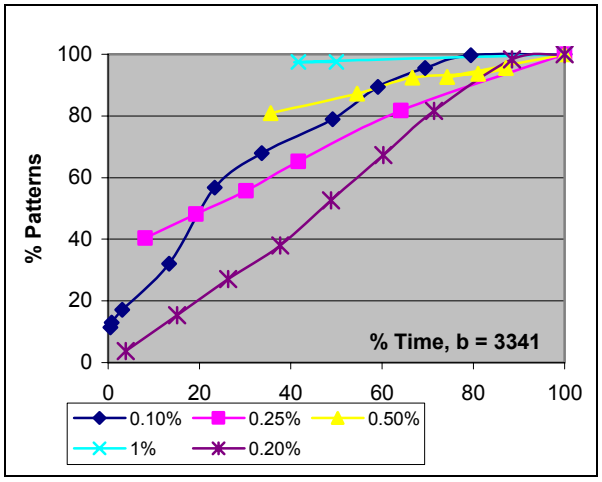

Figure 24.Time-Patterns % of BDFS(b) for varying support for BMS-Web-View-2, T = 5.6, D = 77512, N = 3341

% Time	1.72	4.18	14.01	23.83	38.58	48.41	63.15	97.55	100.00
	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350	F[1] : 350
	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319	F[2] : 2319
	F[3] : 612	F[3] : 1028	F[3] : 1542	F[3] : 1488	F[3] : 1783	F[3] : 1935	F[3] : 2078	F[3] : 2695	F[3] : 3049
	F[4] : 488	F[4] : 607	F[4] : 953	F[4] : 944	F[4] : 1070	F[4] : 1163	F[4] : 1250	F[4] : 1456	F[4] : 1790
	F[5] : 299	F[5] : 301	F[5] : 347	F[5] : 331	F[5] : 363	F[5] : 376	F[5] : 390	F[5] : 445	F[5] : 508
	F[6] : 134	F[6] : 134	F[6] : 134	F[6] : 134	F[6] : 134	F[6] : 136	F[6] : 136	F[6] : 137	F[6] : 141
	F[7] : 1	F[7] : 31	F[7] : 31	F[7] : 31	F[7] : 31	F[7] : 31	F[7] : 31	F[7] : 31	F[7] : 31
		F[8] : 3	F[8] : 3	F[8] : 3	F[8] : 3	F[8] : 3	F[8] : 3	F[8] : 3	F[8] : 3
Border Sets	BS[4] : 524	BS[4] : 879	BS[4] : 2469	BS[4] : 2446	BS[4] : 2890	BS[4] : 3159	BS[4] : 3676	BS[4] : 4820	BS[4] : 8127
	BS[5] : 285	BS[5] : 310	BS[5] : 625	BS[5] : 625	BS[5] : 697	BS[5] : 747	BS[5] : 771	BS[5] : 927	BS[5] : 1865
	BS[6] : 90	BS[6] : 90	BS[6] : 106	BS[6] : 101	BS[6] : 110	BS[6] : 118	BS[6] : 118	BS[6] : 136	BS[6] : 170
	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21	BS[7] : 21
Candidate Sets	C[3] : 17180	C[3] : 14570	C[3] : 10755	C[3] : 11698	C[3] : 9632	C[3] : 8504	C[3] : 7345		

Figure 25. Frequent Pattern Output along with Border Sets and Candidates Patterns of BDFS(b) for BMS-Web View-1 with support 0.08%. N= 497, T=2.5, D=59602 and b=497.

6. CONCLUSION

Traditionally, the frequent pattern mining has been kept as an offline analytical task, where the frequent patterns are found on the data captured for a specific time period, few weeks, months or even years. But with the changing scenario in the business environment and with improvement in the communications technology and the Internet, and with the more and more business processes going online, frequent pattern mining for real-time decision making has become a thrust area of research [22].

Real-time frequent pattern mining will have great impact on the way knowledge is gathered from patterns from the databases. It has the capability to affect all aspects of doing business in today's world. It will provide decision makers with more accuracy and reduced time lag and help in real-time decision-making.

In this paper, we have proposed an algorithm BDFS(b), which is a brute force version of the Block Depth First Search(BDFS) [20]. First we have compared the performance of BDFS(b) with FP-Growth and Apriori and shown that it does significantly better than both.

Moreover, by adjusting its block size properly, BDFS(b) has the extra ability to run with limited available memory, which often becomes a point of concern in other algorithms. We have then shown that while running under real-time constraints it outputs large chunks of frequent patterns with fractional execution times. We have made detailed performance evaluation based on empirical analysis using several commonly used synthetic datasets and one real-life dataset.

Thus, we have demonstrated that real-time frequent pattern mining can be done successfully using BDFS(b).

Further research in this direction may include design of powerful heuristics to enhance the efficiency of BDFS(b) under different scenarios. We believe this study will encourage use of AI heuristic search techniques in real-time frequent pattern mining.

7. REFERENCE

1. Agarwal, R., Imielinski, T. and Swami, A. Mining Association Rules Between Sets of Items in Large Datasets. in *Proceedings of the ACM SIGMOD Conference on Management of Data*, ACM, Washington,D.C., 1993, 207-216.
2. Agrawal, R. and Srikant, R. Mining Sequential Patterns. in *Proceedings of the 11th IEEE International Conference on Data Engineering*, IEEE, Taipei, Taiwan, 1995.
3. Bhote, K.R. Beyond Customer Satisfaction to Customer Loyalty. in *Proceedings of the American Management Association*, New York, NY, 1996.
4. Brin, S., Motwani, R., Ullman, J.D. and Tsur, S. Dynamic Itemset Counting and Implication Rules for Market Basket Data. in *Proceedings of the ACM SIGMOD Conference*, 1997, 255-264.
5. Gartner. The Real-Time Enterprise, 2004.
6. Goethals, B. Memory Issues in Frequent Pattern Mining. in *Proceedings of SAC'04*, ACM, Nicosia, Cyprus, 2004.
7. Goethals, B. Survey on Frequent Pattern Mining, 2003.
8. Gonzales, M.L. Unearth BI in Real-time, Teradata, 2004.
9. Grossman, R.L., Kamath, C., Kegelmeyer, P., Kumar, V. and Namburu, R. *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers, 2001.
10. Han, J., Pei, J. and Yin, Y. Mining Frequent Patterns Without Candidate Generation. in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ACM, Dallas,TX, 2000, 1-12.
11. Hidber, C. Online Association Rule Mining. in *Proceedings ACM SIGMOD International Conference on Management of Data*, ACM, Philadephia, Pennsylvania, 1999, 145-156.
12. Hipp, J., Guntzer, U. and Nakhaeizadeh, G. Algorithms for Association Rule Mining -- A general Survey and Comparision. in *Proceedings of ACM SIGKDD*, ACM, 2000, 58-64.
13. Kalakota, R., Stallaert, J. and Whinston, A.C., Implementing Real time Supply Chain Optimization Systems. in *Supply Chain Management*, (Hong Kong, 1995).
14. Kohavi, R. and Provost, F. Applications of Data Mining to Electronic Commerce. *Journal of Data Mining and Knowledge Discovery*, *5* (2001). 5-10.
15. Langenfeld, S. CRM and the Customer Driven Demand Chain, 2004.
16. Lawrence, R., Almasi, G., Kotlyar, V., Viveros, M. and Duri, S. Personalization of Supermarket Product Recommendations. *Journal of Data Mining and Knowledge Discovery*, *5* (2001). 11-32.
17. Lee, W., Stolfo, S.J., Chan, P.K., Eskin, E., Fan, W., Miller, M., Hershkop, S. and Zhang, J., Real time data mining-based intrusion detection. in *DARPA Information Survivability Conference & Exposition II*, (Anaheim, CA , USA, 2001), IEEE Xplore, 89-100 vol. 101.
18. Lin, W. Association Rule Mining for Collaborative Recommender Systems *Computer Science*, Worcester Polytechnique Institute www.wpi.edu/Pubs/ETD/Available/etd-0515100-145926/ unrestricted/wlin.pdf, 2000, 64.
19. M.J.Zaki Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering*, *12* (2). 372-390.
20. Mahanti, A., Ghosh, S. and Pal, A.K. A High Performance Limited-Memory Admissible and Real Time Search Algorithm for Networks, University of Maryland at College Park, MD 20742, Maryland, College Park, 1992, 1-15.
21. OpenServiceInc. Real-Time Enterprise Risk and Vulnerability Management, Open Service Incorporation, 2004.
22. Rahman, M.S., Martin, N.L. and Paul, S., Data Mining, Group Memory, Group Decision Making: A Theoretical Framework. in *Ninth Americas Conference on Information Systems*, (2003).
23. Savasere, A., Omiecinski, E. and Navathe, S.B. An Efficient Algorithm for Mining Association Rules in Large Databases. in *Proceedings of the 21nd International Conference on Very Large Databases*, Zurich, Swizerland, 1995, 432-444.
24. Schafer, J., Konstan, J. and Riedl, J. Electronic Commerce Recommender Applications. *Journal of Data Mining and Knowledge Discovery*, *5* (2001). 115-152.
25. SeeBeyond. Real-Time Stock Management and VMI, 2004.
26. Shen, Y.D., Yang, Q., Zhang, Z. and Lu, H. Mining the Customer's Up-To-Moment Preferences for E-commerce Recommendation. in Whang, K.-Y., Jeon, J., Shim, K. and Srivastava, J. eds. *Proceedings of the Advances in Knowledge Discovery and Data Mining: 7th Pacific-Asia Conference, PAKDD 2003, Seoul, Korea, April 30 - May 2*, Springer-Verlag, Heidelberg, 2003, 166-177.
27. Su, J.-H. and Lin, W.Y. CBW: An Efficient Algorithm for Frequent Itmeset Mining. in *Proceedings of the 37th Hawaii International Conference on System Sciences*, IEEE, Hawaii, 2004.
28. Toivonen, H. Sampling Large Databases for Association Rules. in *Proceedings of the 22nd International Conference on Very Large Databases*, Mumbai, India, 1996, 134-145.