# An Efficient Algorithm for Real-Time Frequent Pattern Mining for Real-Time Business Intelligence Analytics

**Rajanish Dass**
Indian Institute of Management Ahmedabad
*email: rajanish@iimahd.ernet.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 in data mining tasks, like association rule mining. Frequent pattern mining in real-time is of increasing thrust in many business applications such as e-commerce, recommender systems, and supply-chain management and group decision support systems, to name a few. A plethora of efficient algorithms have been proposed till date, among which, vertical mining algorithms have been found to be very effective, usually outperforming the horizontal ones. However, with dense datasets, the performances of these algorithms significantly degrade. Moreover, these algorithms are not suited to respond to the real-time need. In this paper, we describe BDFS(b)-diff-sets, an algorithm to perform real-time frequent pattern mining using diff-sets and limited computing resources. 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

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 [1].

Frequent pattern mining for large databases of business data, such as transaction records, is of great interest in data mining and knowledge discovery [2], since its inception in 1993, by Agrawal et al. In this paper, we assume that the reader knows the basic assumptions and terminologies of mining all frequent patterns.

Researchers have generally focused on the frequent pattern mining, as it is complex and the search space needed for finding all frequent itemsets is huge [2]. A number of efficient algorithms have been proposed in the last few years to make this search fast and accurate[3]. Among these, a number of effective vertical mining algorithms have been recently proposed, that usually outperforms horizontal approaches [4]. Despite many advantages of the vertical format, the methods tend to suffer, when the tid-list cardinality gets very large as in the case of dense datasets [4]. Again, these algorithms have limited themselves to either breadth first or depth first search techniques. Hence, most of the algorithms stop only after finding the exhaustive (optimal) set of frequent itemsets and do not promise to run under user defined real-time constraints and produce some satisficing (interesting sub-optimal) solutions due to their limiting characteristics[5, 6].

In this paper, we describe BDFS(b)-diff-sets (adopted from[5, 6]), a real-time frequent pattern mining algorithm which runs under limited execution time and has the capability of running under limited memory as well in cases of dense datasets. BDFS(b)-diff-sets does not limit itself to either of breadth-first or a depth-first search, but uses a search technique, which is a good mix of the staged search and depth-first search (discussed later in section 4.1), adopted from [7]. We have adopted the diff-sets concept as introduced by [4] as it has been found to be very effective in cases of dense datasets.

In this paper, we also show the edge of BDFS(b)-diff-sets over existing efficient association mining algorithms such as Apriori [8], FP-Growth [9], Eclat [10] and dEclat [4], 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 present business issues of real-time frequent pattern mining in brief. In Section 3, we discuss a review of the previous work in association rule mining. In Section 4, we introduce algorithm BDFS(b)-diff-sets implemented using diff-sets[4]. Section 5 contains the empirical evaluation of our algorithm. Finally, we conclude the paper in Section 6.

## 2. Business Intelligence Issues of Real-Time Frequent Pattern Mining

An offline analytic 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 [11]. Moreover, in cases where there is a ubiquitous flow of data from various sources, like that of RFID sensor networks, it becomes impossible for the human analyst to sift through huge volumes of data for decision making in real-time.

Frequent pattern mining has been extensively used for market basket analysis of data, to find out the hidden patterns that lie in the transactional database. To promote a particular product, if a retailer decides to go for dynamic pricing or for dynamic discount, she must do it before the customer actually moves out of the store. Hence, the retailer cannot afford to make run on the huge dataset again and again to depict the correct association rule for a particular customer before she moves out of the store. Again, the strategy of making the association mining an offline task and refer to the patterns for a particular time period may also prove to be ineffective because the customer preference may considerably change over time. Hence, dynamic pricing or offering dynamic discounts will not be able to fetch the necessary returns from the customer(s), if the whole exercise is based on patterns that were obtained previously. With competition growing at a break-neck speed, organizations have started appreciating the real-time analysis and real-time decision making for the particular concerned customer [12]. The importances for real-time solutions have been felt more lately due to the introduction and development of online businesses (although for offline businesses as well, the thrust remains the same). Researchers [14] believe that real-time personalization technology will proactively offer a particular customer products and services that will fit into their need 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.

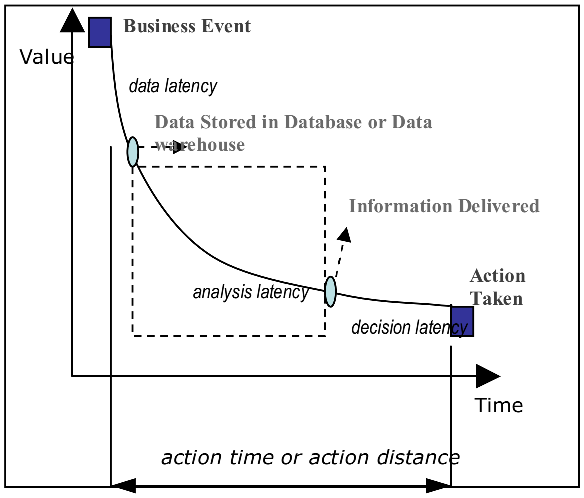


**Exhibit 1.** Framework for real-time business intelligence. Organizations must manage three distinct processes that create latency in an analytic environment to support real-time decision making. ***Source [13]***

In cases of event based information management systems, as the example in the previous paragraph, current approaches of business intelligence systems using various data mining techniques make organizations face some serious latency problems, which they must overcome. These are: *data latency, analysis latency and decision latency* [13]. The following exhibit will make the point clearer.

Once a business event happens, users face *data latency*, meaning the time taken for various pre-processing steps for storing this data into the corresponding database or data warehouse. On this data, various analytic processes have to run for discovering the relevant information and delivering it to the right user for the purpose of decision making. This phase, referred to as analytic latency in Exhibit 1, refers to the time taken by various algorithms to run on the corresponding database or data warehouse. Once the information is delivered, the user may take some time before she can take any action on this delivered information. This is referred to as *decision latency*, in Exhibit 1. As pertinent from the above figure, the majority of the action time is caused due to the *analytic latency* only. Hence the major challenge to bye-pass these latencies and delivering *right information* to the *right user* within *right time* is the *analytic latency*. This means that the existing technologies hinder in responding to the real-time need of the business user due to their in-built limitations as they do not have the capability to respond to the real-time need. This real-time time bound, as described by various authors as *right time*, will vary from user-to-user and from industry to industry. In an research carried by TDWI (The Data Warehousing Institute) [15], based on the responses of 383 respondents world wide, who have deployed various data mining related systems in organizations, it has been found that the major factors that create the bottle-neck of reducing the *analytic latency* and real-time business intelligence are lack of tools for doing real-time processing, immature technology and performance issues in Exhibit 2.

| | |
|---|---|
| Lack of tools for doing real-time processing | 35% |
| Immature technology | 28% |
| Performance and scalability | 24% |

**Exhibit 2.** Obstacles to real-time business intelligence *Source[16]*

There are numerous areas where real-time decision making plays a crucial role. These include areas like real-time customer relationship management [17-19], real-time supply chain management systems [20] real-time enterprise risk and vulnerability management [21], real-time stock management and vendor inventory [22], real-time recommender systems[23], 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 [24], 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 Done

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 [3, 25, 26]. 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 [25]. However, Su & Lin [27] have concluded that the most salient features of these algorithms are their *counting strategy*, *search direction* and *search strategy* (**Table 1**). Recently, a number of vertical mining algorithms have been proposed[4, 10, 28]. In a vertical database, each item is associated with its corresponding set of transactions where the particular item appears [4], called tid-list. However, in dense datasets, the method suffers since the intersection time becomes very high. Furthermore, the scalability of these algorithms gets affected, when the vertical tid-lists become too large for memory. Zaki [4] has introduced the concept of d*iff-sets,* that only keeps track of the differences in the tids of a candidate pattern from its generating frequent patterns. This diff-set implementation drastically cut down the size of the memory and tid-list intersections are done significantly faster (as diff-sets are a small fraction of the size of tid-lists).

| **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 of tid-lists* | Eclat | Partition | | |
| *Intersection of Diff-Sets* | dEclat | | | |

**Table 1.** Classification of prevaililng algorithms

## 4. BDFS(b)-diff-sets: An Efficient Technique of Frequent Pattern Mining In Real-Time Using Diff-Sets

### 4.1 Algorithm Basics

In this study, we propose a brute force algorithm BDFS(b)-diff-sets, which is a variant of the Block Depth First Search [7] and inducted into the domain of frequent pattern mining [5, 6]. Block Depth First Search is a search algorithm, based on a novel combination of the staged search and the depth first search [29]. As a result, it has the merits of both best-first search and the depth-first-branch-and-bound (DFBB) search [30], ,and at the same time, avoids bad features of both. BDFS(b)-diff-sets 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 for a given database is created in a support-independent pre-processing step and kept in the hard-disk, 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)-diff-sets starts its search for frequent patterns of higher lengths from this point forward by intersecting the diff-set tid-lists of corresponding items. The most salient features of BDFS(b)-diff-sets 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. In this paper, we keep the block b variable and the value to be defined by the user using her knowledge and experience depending on the available computer memory. A possible state space diagram of BDFS(b)-diff-sets is shown in. Fig. 1

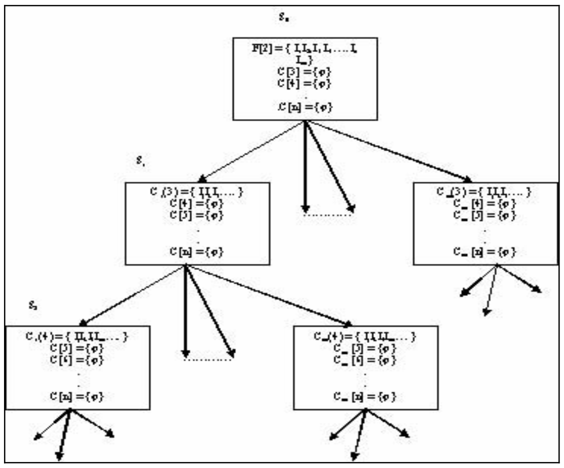


**Fig. 1.** State space representation of BDFS(b)-diff-sets

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 $\phi$. 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 by intersecting the diff-sets of the corresponding items.
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.

We have implemented this algorithm with diff-sets as proposed by [4] and have used the prefix based tree, called trie, data structure for implementing BDFS(b)-diff-sets.

## 4.2 Algorithm Details

```
Algorithm BDFS(b)-diff-sets:

 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 diff-set 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.
```

**Fig. 2.** Algorithm BDFS(b)-diff-sets

Let us consider the following example to show how BDFS(b)-diff-sets work.

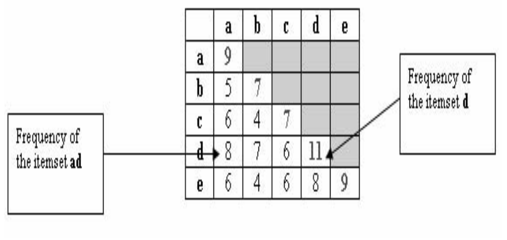
Let the following table (fig. 3) represent a set of 12 transactions, where the items are represented by a, b, c …

| | | | |
|---|---|---|---|
| 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 |

**Fig. 3** Given transaction dataset

*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, $\forall$ i = 1,2…n and $\forall$ j = 1,2,3…i and M(i,I) represents the total number of occurrences of item i.*

**Fig. 4**. Procedure *Create_Matrix*


Frequency of the itemset ad
Frequency of the itemset d

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 9 | | | | |
| b | 5 | 7 | | | |
| c | 6 | 4 | 7 | | |
| d | 8 | 7 | 6 | 11 | |
| e | 6 | 4 | 6 | 8 | 9 |

**Fig. 5** Matrix M

Now we proceed as follows:

***Step I.*** Given this set of transactions D, create a two-dimensional lower triangular matrix M using procedure *Create_Matrix* (fig. 4) and the diff-set transaction id lists. This diff-set tid-list (fig. 5) contains the transaction numbers corresponding to which the particular item does not occur. The created matrix M is depicted in fig. (5). This creating of the matrix M and the diff-set tid-list and storing in the hard-drive is a *support independent step* and we will refer this step through out this paper as a *support-independent pre-processing step*.

***Step II.*** Let the absolute support $\xi$ (abs) be 3. Cells of Matrix M are visited to find F(1) and F(2) [where F(n) is frequent pattern of length n]. With the frequency being in parentheses, 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)

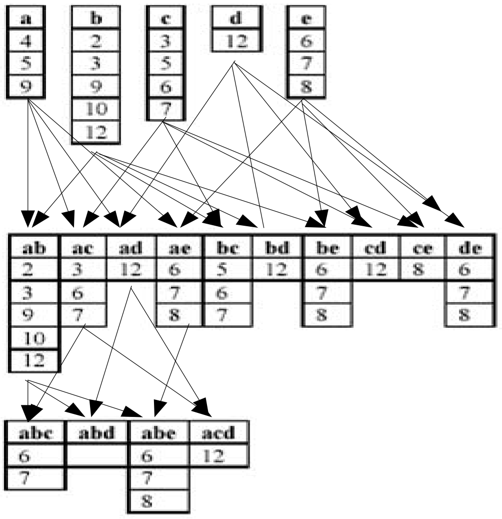


**Fig. 6 The diff-set list of the items (a snap-shot till first block of patterns)**

***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. If all its 2-length subsets are not present, then the pattern is pruned (using the support monotonicity property[8]), else 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)

***Step V.*** Let the block size b is 4. 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.*** From the diff-sets of the two-length patterns we calculate the diff-sets of the three length patterns as shown in the figure(4d) as follows: If d(ab) and d(ac) represents the diff-set of ab and ac respectively, then we can get d(abc) = d(ac) – d(ab) [as suggested by Zaki [4]] and the frequency of the pattern abc can be found from freq(abc) = freq(ab) - |d(abc)|. We now check the frequency of these patterns by intersecting the diff-set 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)

All immediate subsets of the pattern *abcd* are not present in *F(3)*. 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)

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)*, search *BS (4)* to see if any of the immediate supersets are waiting in the border set. Pattern *abcd* is in *BS (4)* with sub-itemset counter = 3. Hence 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. Merge newly found *k-length* frequent patterns with previously found *k-length* frequent patterns to make patterns of higher length.

*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 diff tid-list of these patterns,

*F (4) = {abcd (3)}*……………….…… (13)

*F (3) = {bde (3),cde (5)}* ………….. (14)

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

*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)-diff-sets 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 Evaluation

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.
To evaluate the performance of BDFS(b)-diffsets with on dense datasets, we have tested it on various dense datasets. This includes real-life dense datasets like CHESS, Connect-4, PUMSB and PUMSB*1and synthetic datasets like: T10I8D100K, T10I8D10K, T10I8D1K (N=1K). These datasets were generated using the IBM synthetic data generator2 [2]. The experiments were performed on a Red-Hat Linux machine with 1GB RAM and 20 GB HD with Pentium IV 2.24Ghz processor.

### 5.1 Comparison of BDFS(b)-diff-sets with existing algorithms

In order to show how BDFS(b)-diff-sets performs on dense datasets, when it is run to generate all frequent patterns, we have chosen to compare it with dEclat[3], Eclat[4], FP-growth[5] and Apriori[6.] Since FP-growth is known to be an order faster and scales better than Apriori[9], we have compared Apriori and BDFS(b)-diff-sets but for their number of patterns checked. In figures 7, 8, 9 and 10, we have compared the run-time of FP-Growth, dEclat and Eclat with BDFS(b)-diff-sets for dense datasets Pumsb, T10I8D100K and Pumsb* respectively and found that BDFS(b)-diff-sets significantly out-performs all the three algorithms in these cases. In figure 11, we have tested the scalability of Eclat and dEclat and BDFS(b)-diff-sets. We have observed that all the algorithms are scalable with time and number of transactions in the database, but BDFS(b)-diff-sets takes strikingly much less time than dEclat, and Eclat over the same databases. Comparing the number of patterns being checked by Apriori and BDFS(b)-diff-sets, as shown in figure 12, it is found that BDFS(b)-diff-sets checks much lesser number of patterns than Apriori. The performance imperatives come from the efficient search strategy of the block depth first search that BDFS(b)-diff-sets utilizes and combines the power of the diff-sets approach. It is worth mentioning at this point that the codes we have obtained from the public domains are highly optimized in respect to implementation.

[1] These datasets are publicly available at http://fimi.cs.helsinki.fi/data/

[2] The data generator is available from http://www.almaden.ibm.com/cs/quest//syndata.html#assocSynData

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

[4] The Eclat code used for comparison is publicly available at http://fuzzy.cs.uni-magdeburg.de/~borgelt/eclat.html

[5] The FP-growth code used for comparison is publicly available at www.cse.cuhk.edu.hk/~kdd/program.html

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

### 5.2 Real-Time Performance of BDFS(b)-diff-sets

Figures 13, 14, 15 and 16 summarize the real-time behavior of BDFS(b)-diff-sets by depicting the percentage of frequent patterns generated with percentage execution time having F(1) & F(2) included and excluded in two respective curves. This we have done to show how the real-time performance is affected by the two-dimensional matrix M. It may be noted that the over all percentage of output is almost always ahead of percentage execution time. In figure 13, we find out that we have approximately 95% of the frequent patterns in 25% of completion time. We have also observed that our proposed algorithm perform quite well on real-life dense dataset connect-4 and highest length patterns can be obtained in lesser than 50% of total execution time.

Although it can be argued that all the existing frequent pattern mining algorithms will give some output if the execution is stopped at a user-defined time, but we have found that their performance in the real-time output is not promising as they use either a breadth-first or a depth-first search only and do not try to promise real-time performance. In figures 17, 18 and 19, we do a comparison of the real-time output of the existing algorithms. In all the cases, we find that BDFS(b)-diff-sets outperforms al existing techniques in providing real-time output. From figure 19, we find that BDFS(b)-diff-sets can provide 70% of the frequent patterns in just 40% of execution time. Whereas depth-first search techniques like FP-Growth and dEclat provides much lesser patterns corresponding to the given time. Its worth mentioning at this point that BDFS(b)-diff-sets takes much lesser time for complete execution as shown before. In this case, the percentage time taken for a particular algorithm is the slice of its own total execution time. Had the comparison been done in a scale of absolute time, the real-time performance edge pf BDFS(b)-diff-sets would have been much more prominent. This can be explained by the fact that BDFS(b)-diff-sets is using an intelligent and informed stages search strategy and is able to rank the nodes and continue searching in an intelligent manner, as compared to other methods that are using a blind depth-first or breadth-first search technique. The datasets on which the performance has been measured happen to be dense datasets[7]

Figures 20-22 shows the performance of BDFS(b)-diff-sets when the block size is varied. We find that for smaller block size we get higher length patterns quickly. This signifies that a better real-time output is obtained with smaller block sizes. Fig 23 gives a tabular representation of the actual output. From figure 23 we find that all F[15] patterns are found only in 34% of completion time.

[7] Comparison on sparse datasets can be obtained in [5,6]

## 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, advent of GRID based computing scenarios and in cases where data is being captured by AIT (like RFID) agent based sensor networks, frequent pattern mining for real-time decision making will become a thrust area of research. 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, bye-passing the analytic latency as discussed in [13].

In this paper, we have proposed an algorithm BDFS(b)-diff-sets, a brute force version of the Block Depth First Search(BDFS) [7] and implemented with diff-sets [4]. First we have compared the performance of BDFS(b)-diff-sets with dEclat, Eclat, FP-Growth and Apriori and shown that it compares well with others. Moreover, by adjusting its block size properly, BDFS(b)-diff-sets 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 commonly used synthetic and real-life dense datasets. Thus, we have demonstrated that real-time frequent pattern mining can be done successfully using BDFS(b)-diff-sets. 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.

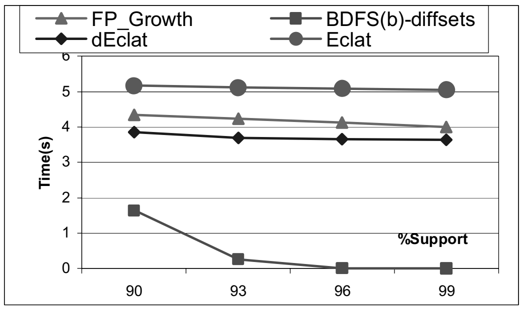


**Fig. 7.** Time (in seconds) comparison of FP-Growth, Eclat and dEclat with BDFS(b)-diffsets (b= 20880) on PUMSB, N=2113, T=74, D=49046

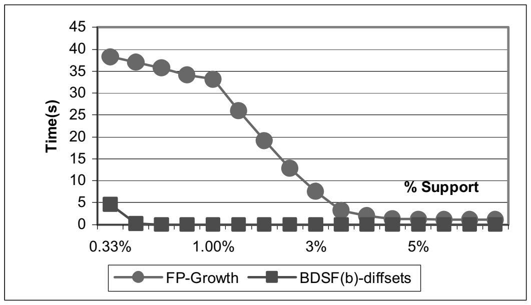


**Fig. 8.** Time (in seconds) comparison of FP-Growth with BDFS(b)-diffsets for T10I8D100K, b=100K. In most cases BDFS(b)-diffsets took in milli seconds only.

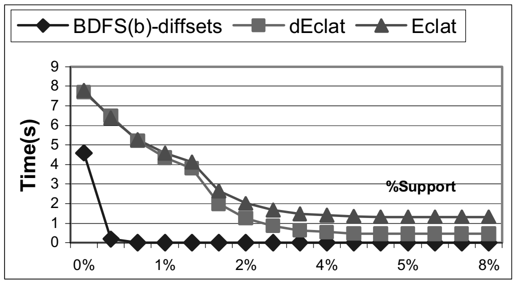


**Fig. 9.** Time (in seconds) comparison of Eclat and dEclat with BDFS(b)-diffsets for T10I8D100K, b=100K

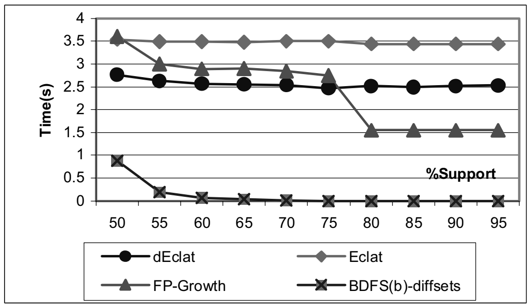


**Fig. 10.** Time (in seconds) comparison of FP-Growth, Eclat and dEclat with BDFS(b)-diffsets (b=2088K) for PUMSB*, N=2088 T= 50.5, D = 49046

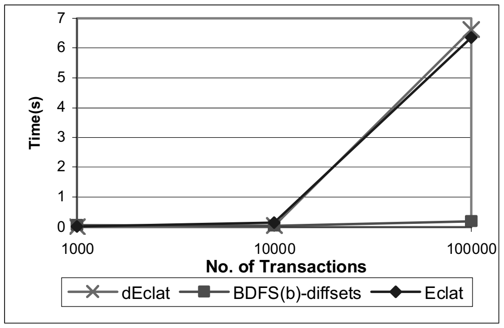


**Fig. 11.** Scalability evaluation of BDFS(b)-diffsets with Eclat and dEclat supp=0.5%, b = 100K for T10I8D1K,10K and 100K (Time in seconds)

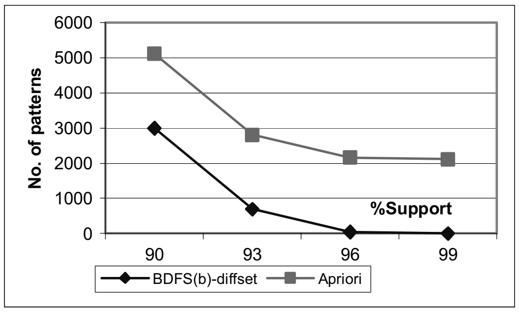


**Fig. 12.** Number of patterns checked by Apriori and BDFS(b)-diffsets (b=208800) for Pumsb, N=2113,T=74, D=49046, with varying support

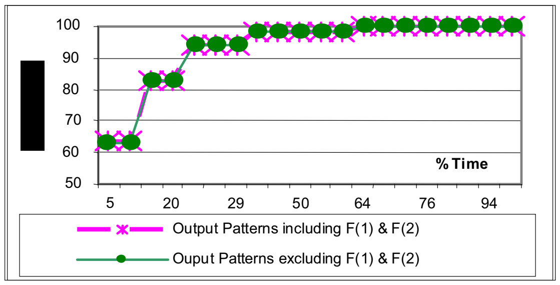


**Fig. 13** Time-Patterns % of BDFS(b) for b=75K and 65% supp for Chess (N=75, T=37, D=3196)

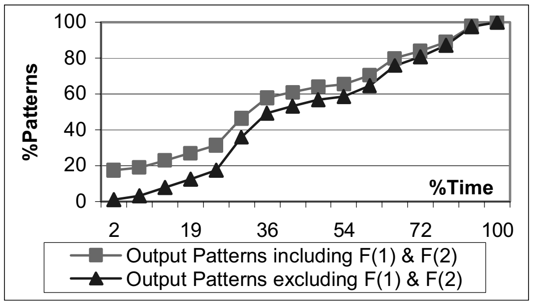


**Fig. 14** Time-Patterns % for b=75K and 65% supp for T10I8D100K

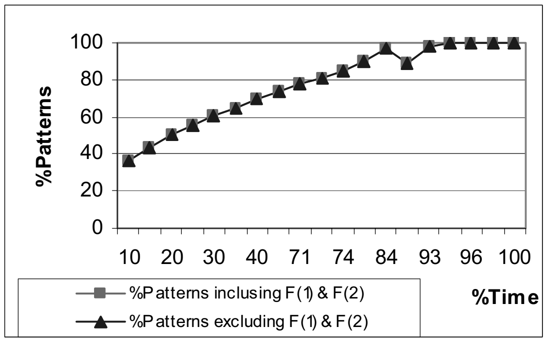


**Fig. 15.** Time-pattern% of BDFS(b), b=129, for 75% supp of Connect-4

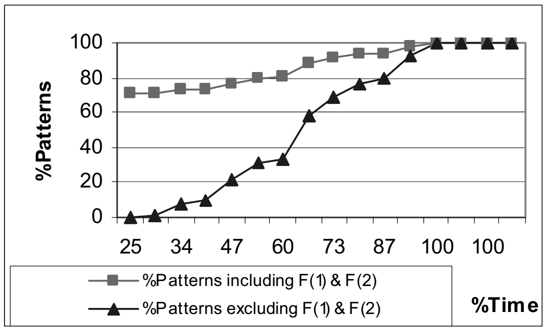


**Fig. 16**. Time-pattern% of BDFS(b), b=1K, for T25I20D100K

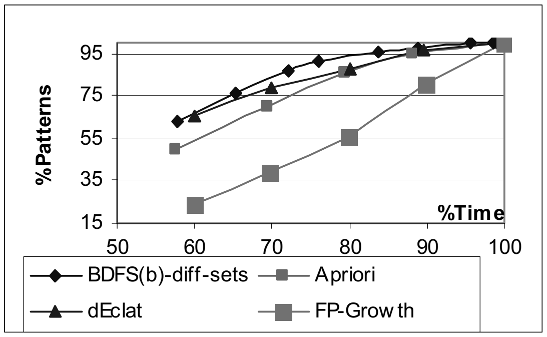


**Fig. 17.** Time-pattern% comparison of dEclat, Apriori, FP-Growth with BDFS(b)-diff-sets, b=1K, for 0.15% supp of T10I8D100K

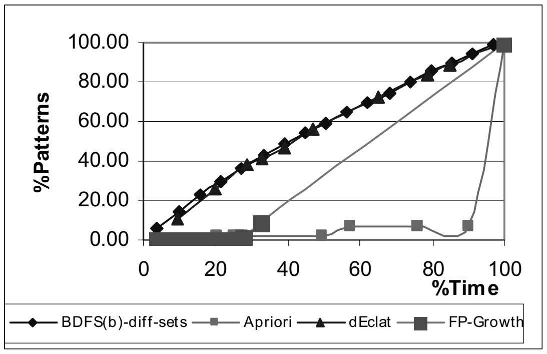


**Fig. 18**. Time-pattern% comparison of dEclat, Apriori, FP-Growth with BDFS(b)-diff-sets, b=2113, for 75% supp of PUMSB

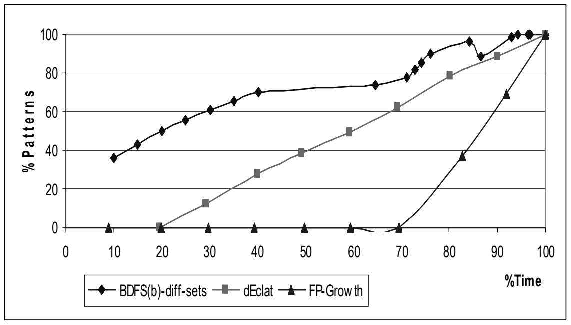


**Fig. 19**. Time-pattern% comparison of dEclat, FP-Growth with BDFS(b)-diff-sets, b=380, for 75% supp of Connect-4

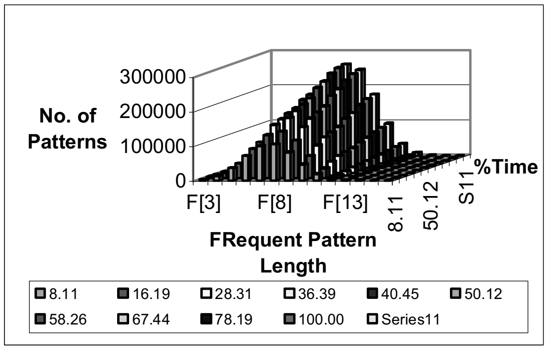


**Fig. 20.** Real-time output of frequent patterns by BDFS(b)-diff-sets, b=76, for 50% support of CHESS

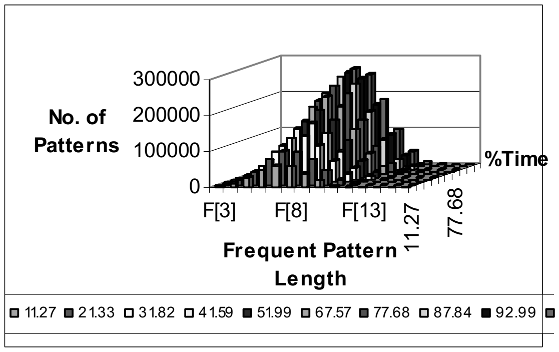


**Fig. 21**. Real-time output of frequent patterns by BDFS(b)-diff-sets, b=760, for 50% support of CHESS

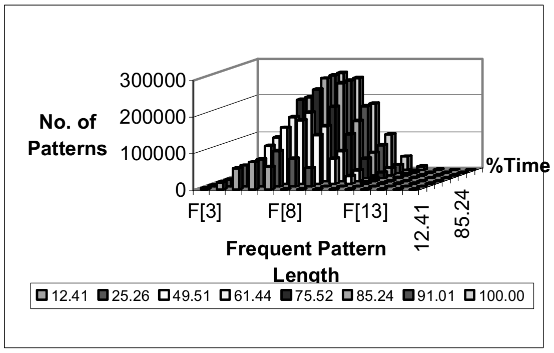


**Fig. 22**. Real-time output of frequent patterns by BDFS(b)-diff-sets, b=7600, for 50% support of CHESS

## 7 References

[1] M. L. Gonzales, "Unearth BI in Real-time," vol. 2004: Teradata, 2004.
[2] B. Goethals, "Memory Issues in Frequent Pattern Mining," in *Proceedings of SAC'04*. Nicosia, Cyprus: ACM, 2004.
[3] B. Goethals, "Survey on Frequent Pattern Mining," vol. 2004. Helsinki, 2003, pp. 43.
[4] M. J. Zaki and K. Gouda, "Fast Vertical Mining Using Diffsets," presented at 9th International Conference on Knowledge Discovery and Data Mining, Washington, DC, 2003.
[5] R. Dass and A. Mahanti, "Frequent Pattern Mining in Real-Time – First Results," presented at TDM2004/ACM SIGKDD 2004, Seattle, Washington USA, 2004.
[6] R. Dass and A. Mahanti, "An Efficient Technique for Frequent Pattern Mininig in Real-Time Business Applications," presented at 38th IEEE Hawaii International Conference on System Sciences (HICSS 38), Big Island, 2005.
[7] A. Mahanti, S. Ghosh, and A. K. Pal, "A High Performance Limited-Memory Admissible and Real Time Search Algorithm for Networks," University of Maryland at College Park, MD 20742, Maryland, College Park, Computer Science Technical Report Series CS-TR-2858 UMIACS-TR-92-34, March 1992 1992.
[8] R. Agarwal, T. Imielinski, and A. Swami, "Mining Association Rules Between Sets of Items in Large Datasets," in *Proceedings of the ACM SIGMOD Conference on Management of Data*. Washington,D.C.: ACM, 1993, pp. 207-216.
[9] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns Without Candidate Generation," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. Dallas,TX: ACM, 2000, pp. 1-12.
[10] M. J. Zaki, "Scalable Algorithms for Association Mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, pp. 372-390, 2000.
[11] Gartner, "The Real-Time Enterprise," vol. 2004, 2004.
[12] B. Riggs and P. McDougal, "Real-Time Analysis of Buying Habits," in *Information week*, vol. October 18, 1999, pp. 117.
[13] R. Hackathorn, "Minimizing Action Distance," in *The Data Administration Newsletter*, vol. 23, 2003.
[14] S. Langenfeld, "CRM and the Customer Driven Demand Chain," vol. 2004, 2004.
[15] W. Eckerson, "The "Soft Side" of Real-Time BI," in *DM Review*, vol. 14, 2004, pp. 30-32.
[16] TDWI, "The Real Time Enterprise Report," TDWI 2003.
[17] M. J. A. Berry and G. S. Linoff, *Data Mining Techniques:For Marketing, Sales, and Customer Support*: John Wiiley & Sons, 1997.
[18] C. Rygielski, J. C. Wang, and D. C. Yen, "Data Mining Techniques for Customer Relationship Management," *Technology and Society*, vol. 24, pp. 483-502, 2002.
[19] Y. D. Shen, Q. Yang, Z. Zhang, and H. Lu, "Mining the Customer's Up-To-Moment Preferences for E-commerce Recommendation," in *Proceedings of the Advances in Knowledge Discovery and Data Mining: 7th Pacific-Asia Conference, PAKDD 2003, Seoul, Korea, April 30 - May 2*, vol. 2637 / 2003, *Lecture Notes in Computer Science*, K.-Y. Whang, J. Jeon, K. Shim, and J. Srivastava, Eds. Heidelberg: Springer-Verlag, 2003, pp. 166-177.
[20] R. Kalakota, J. Stallaert, and A. C. Whinston, "Implementing Real time Supply Chain Optimization Systems," presented at Supply Chain Management, Hong Kong, 1995.
[21] OpenServiceInc., "Real-Time Enterprise Risk and Vulnerability Management," vol. 2004: Open Service Incorporation, 2004.
[22] SeeBeyond, "Real-Time Stock Management and VMI," vol. 2004, 2004.
[23] J. Schafer, J. Konstan, and J. Riedl, "Recommender Systems in e-Commerce," presented at Proceedings of the ACM E-Commerce, 1999.
[24] W. Lee, S. J. Stolfo, P. K. Chan, E. Eskin, W. Fan, M. Miller, S. Hershkop, and J. Zhang, "Real time data mining-based intrusion detection," presented at DARPA Information Survivability Conference & Exposition II, Anaheim, CA , USA, 2001.
[25] J. Hipp, U. Guntzer, and G. Nakhaeizadeh, "Algorithms for Association Rule Mining -- A general Survey and Comparision," *SIGKDD Explorations*, vol. 2, pp. 58-64, 2000.
[26] R. L. Grossman, C. Kamath, P. Kegelmeyer, V. Kumar, and R. Namburu, *Data Mining for Scientific and Engineering Applications*: Kluwer Academic Publishers, 2001.
[27] J.-H. Su and W. Y. Lin, "CBW: An Efficient Algorithm for Frequent Itmeset Mining," in *Proceedings of the 37th Hawaii International Conference on System Sciences*. Hawaii: IEEE, 2004.
[28] P. Shenoy, J. R. Haritsa, S. Sudarshan, G. Bhalotia, and M. Bawa, "Turbo-charging Vertical Mining of Large Datasets," presented at ACM SIGMOD International Conference Management of Data, 2000.
[29] N. J. Nilson, *Artificial Intelligence: A New Synthesis*. Los Altos, CA: Morgan Kaufmann, 1998.
[30] V. N. Rao and V. Kumar, "Analysis of Heuristic Search Algorithms," University of Minnessota, Technical Report Csci TR 90-40, 1990.

| %Time | 100 | 91 | 85 | 74 | 62 | 51 | 45 | 34 | 28 | 22 | 16 | 10 | 4 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | |
| F[1] | 27 | 27 | 27 | 27 | 27 | 27 | 27 | 27 | 27 | 27 | 27 | 27 | 27 |
| F[2] | 312 | 312 | 312 | 312 | 312 | 312 | 312 | 312 | 312 | 312 | 312 | 312 | 312 |
| F[3] | 2192 | 2192 | 2192 | 2068 | 2003 | 2003 | 2003 | 2003 | 2003 | 2003 | 2003 | 2003 | 2003 |
| F[4] | 10210 | 10210 | 10210 | 8992 | 8875 | 8629 | 8629 | 7320 | 7224 | 6847 | 5875 | 5098 | 2999 |
| F[5] | 32977 | 32370 | 30444 | 28262 | 27701 | 26009 | 25514 | 21224 | 20402 | 18717 | 15621 | 12238 | 5565 |
| F[6] | 76345 | 72996 | 67866 | 63903 | 61003 | 55906 | 52542 | 44037 | 40963 | 35870 | 28835 | 20554 | 7906 |
| F[7] | 128208 | 120747 | 111497 | 106074 | 95856 | 86421 | 79894 | 65110 | 58179 | 47548 | 37440 | 25085 | 7612 |
| F[8] | 155445 | 144715 | 135617 | 127761 | 110673 | 95111 | 85711 | 67868 | 57777 | 45049 | 32520 | 18881 | 5669 |
| F[9] | 135148 | 125864 | 121436 | 107890 | 89616 | 73018 | 63703 | 48178 | 37159 | 28139 | 19434 | 9716 | 2975 |
| F[10] | 83291 | 78102 | 77396 | 61661 | 49369 | 36766 | 32538 | 22193 | 15192 | 11559 | 7072 | 3220 | 1079 |
| F[11] | 35699 | 34219 | 34178 | 22678 | 17775 | 12344 | 11124 | 6297 | 3882 | 3018 | 1597 | 685 | 198 |
| F[12] | 10347 | 10141 | 10141 | 5240 | 3838 | 2692 | 2495 | 1237 | 596 | 493 | 234 | 96 | 29 |
| F[13] | 1951 | 1941 | 1941 | 718 | 511 | 367 | 348 | 179 | 55 | 48 | 24 | 7 | 2 |
| F[14] | 225 | 225 | 225 | 52 | 40 | 29 | 28 | 19 | 2 | 2 | 1 | 0 | 0 |
| F[15] | 13 | 13 | 13 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | | | | |
| C[3] | 0 | 0 | 0 | 147 | 226 | 226 | 226 | 226 | 226 | 226 | 226 | 226 | 226 |
| C[4] | 0 | 0 | 0 | 285 | 118 | 366 | 366 | 1703 | 1800 | 2186 | 3202 | 4018 | 6233 |
| C[5] | 0 | 622 | 2664 | 35 | 8 | 303 | 812 | 100 | 440 | 280 | 254 | 509 | 1050 |
| C[6] | 0 | 0 | 599 | 661 | 929 | 99 | 1995 | 86 | 58 | 308 | 500 | 604 | 194 |
| C[7] | 0 | 0 | 626 | 215 | 0 | 333 | 104 | 727 | 428 | 0 | 0 | 87 | 0 |
| C[8] | 0 | 0 | 717 | 94 | 0 | 370 | 0 | 705 | 0 | 0 | 0 | 2275 | 0 |
| C[9] | 0 | 0 | 16 | 0 | 0 | 75 | 0 | 322 | 0 | 0 | 0 | 153 | 0 |
| C[10] | 0 | 0 | 0 | 0 | 0 | 1103 | 0 | 285 | 0 | 0 | 0 | 8 | 0 |
| C[11] | 0 | 0 | 0 | 0 | 0 | 65 | 0 | 282 | 0 | 0 | 0 | 0 | 0 |
| C[12] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C[13] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C[14] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C[15] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Fig. 23**. Frequent output along with candidate sets of BDFS(b)-diff-sets for PUMSB data for 75% support and b=2113