h1-max: A Domain Independent Heuristic for Real-Time Frequent Pattern Mining

Rajanish Dass

Indian Institute of Management Ahmedabad
e-mail: *rajanish@iimahd.ernet.in*

Ambuj Mahanti

Indian Institute of Management Calcutta
e-mail: *am@iimcal.ac.in*

Abstract

In a number of areas of doing business, real-time frequent pattern mining is in need. The need is being felt more due to the possibility of real-time knowledge discovery along with the gradual acceptance of technologies like RFID and grid computing and the huge amount of possibilities they promise for real-time decision making like supply-chain optimization. This has made Real-time frequent pattern mining for business intelligence systems currently in the focal area of research. In this paper, we have described a domain-independent heuristic, *h1-max* and a heuristic search algorithm, *BDFS(b)-h1-max* for real-time frequent pattern mining using limited computer memory. Empirical evaluations show that the techniques being presented can make a fair estimation of the set of the probable frequent patterns and completes the search 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. The issue is, if the historic data can help us make better decisions, how real-time data can improve the decision making process [1] in a fast changing world, where real-time optimization techniques have become the focus of both academia and industry. Frequent pattern mining for large databases of business data, such as transaction records, is of great interest in knowledge discovery and data mining, since its inception by Agrawal et al. in 1993 [2]. 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.

International Conference on Management of Data
COMAD 2005b, Hyderabad, India, December 20-22, 2005

Researchers have generally focused on mining of frequent patterns as it is complex and the search space needed is colossal. A number of efficient algorithms have been proposed in this area in the last decade. However, most of the successful algorithms in this area have been bounded by their limitation of making the frequent pattern mining an offline analytical task. These algorithms stop only after finding the exhaustive (optimal) set of frequent patterns 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. Until recently, researchers have introduced techniques for real-time frequent pattern mining looking at its huge applicability in various facets of decision making and management [3-5]. One such area of thrust may be real-time optimization techniques in supply-chain management using real-time data from RFID detectors. A plethora of other areas of application demands real-time frequent pattern mining. Among them, real-time customer relationship management, real-time recommender systems and real-time fraud detection systems and group-decision support systems are to name a few. However, all these real-time frequent pattern mining techniques have used a brute-force approach and have not incorporated any domain-independent heuristic that enhances this real-time search for frequent patterns.

In this paper, we introduce a heuristic named *h1-max* and describe BDFS(b)-h1max, a real-time heuristic search technique for real-time frequent pattern mining. We also show the edge of this heuristic search technique, BDFS(b)-h1 max, over its brute force version BDFS(b), which in turn has been found to show substantial performance improvement over existing algorithms like Apriori, FP-Growth, Eclat, dEclat etc. when run to completion and outputs exhaustive set of frequent patterns[3-6] .

The rest of the paper is organized as follows. In the next section, we discuss the importance of real-time frequent pattern mining in businesses, with a special reference to supply-chain management. In section 3, we provide a review of the previous work done in developing heuristics in frequent pattern mining. In section 4, we present the heuristic h1 max and in section 5, we introduce BDFS(b)-h1 max. Section 6

contains the empirical evaluation of our algorithm. Finally we conclude the paper in section 7.

2. Applications of Real-Time Frequent Pattern Mining in Business

With more and more companies maintaining customer and transactional databases and data warehouses, data mining has become an important and integral component of doing business in a global context. In today's scenario of the huge amount of data being maintained by the business houses and with the advent and progress of e-commerce, thousands and millions of data can easily take place in a single day. Embedded in these data is the valuable hidden knowledge about the behavior of the customers [7]. There are numerous areas where real-time decision making plays a critical role. One such area of focus is real-time supply chain management systems.

In today's demanding business environment, timely access to data to generate business information is the key to competitive advantage. Real-time data at the right time (***right-time data***) may change operational processes and stimulate process innovation. However, real-time data feeds to legacy systems or ERP may not be productive. Adaptability may be enhanced if decision systems can access information at the right time based on real-time data (real-time analytics) which may be acquired from diverse sources (RFID, sensors, GPS, barcodes). The argument over format (electronic produce code or EPC, universal identifier code or UID) may continue but that should not inhibit the thinking pre-requisite for process innovation to make adequate use of (Figure 1) right-time data [8].

This level of information, properly controlled, creates the opportunity for effective management throughout the supply chain: whether it is reading the bar code on a product, tracking that product's movement through the network or capturing the data at various transit points throughout the chain.

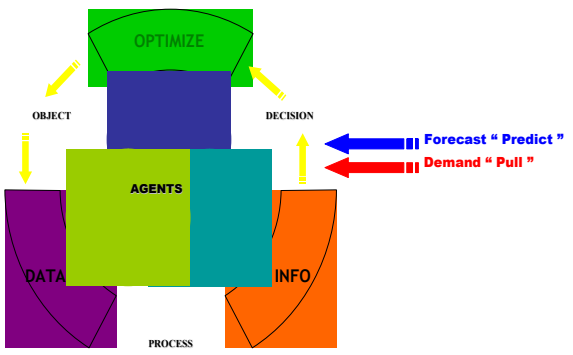

Figure 1. Connectivity of real-time data to process (real-time analytics) may improve decisional information *Source [8]*

The level of management control and real time decision making and the efficiencies this produces gives a return on investment from controlling processes real-time through a manufacturing line, to effective warehouse and inventory management, from tracking the movement of goods in the distribution and delivery chain with real-time product and shipment location dates, right through into the retail store and beyond. No major organization can ignore this return on investment. It has been estimated that companies, which generally spend about 10 percent of revenue on logistics and inventory overhead, could halve that by taking full advantage of effective supply chain management planning. Quite simply, 'increased information used effectively equals increased profits' [9]. In the next 5 years or, more likely, in the next 25-50 years, when we may migrate from adaptive to ***predictive*** status of operations, we will require other concepts and tools that may be unknown, today [8].

There are numerous other areas where real-time decision making becomes important. These include areas like real-time customer relationship management and real-time recommender systems, real-time enterprise risk management [10], real-time fraud detection [11], real-time negotiations and other areas like real-time dynamic pricing. 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 [12, 13].

The most well known and influential algorithm is Apriori[14]. Many variants of this algorithm have been designed. Other ways of solving the problem were using various partitioning methods and sampling methods. Su & Lin [15] have concluded that the most salient features of these algorithms are their *counting strategy*, *search direction* and *search strategy* (figure 2).

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			

Figure 2. Classification of prevailing algorithms

Throughout the last decade, researchers have tried applying a number of heuristics for frequent pattern

mining. One such heuristic that has the greatest influence in frequent pattern mining is the Apriori heuristic [14]. A number of other attempts have been made for proposing heuristics for frequent patterns, which, however have not been found to be as effective and accurate as expected to be [12]. The major areas of development of these heuristics are: a.) to formulate heuristics for forecasting the probable frequency of a *k-length* pattern b.) to decide the switching from the horizontal database format to the vertical tid-lists depending on the database density c.) and also to predict the maximum number of candidate patterns and the level till which frequent patterns for a given support threshold can occur.

The heuristic proposed for shifting from the Apriori to the AprioriTid in the AprioriHybrid algorithm by [16] was found to fail in substantial number of cases. Another such attempt was done by [17] proposing a mixture of Apriori and Eclat and a switching between them in the Hybrid algorithm. None of these heuristics have, however, been able to grab the centre-stage of frequent pattern mining.

Forecasting the probable frequency of pattern based on other patterns has been a central question in the arena of frequent pattern mining. One such attempt was to find frequencies of *k-length* frequent patterns based on the heuristically defined values of patterns of *(k-1)-length* patterns as was done in DIC [18] used. However, later studies have found that the DIC heuristic was not performing as promised. Moreover, DIC landed up in counting much more number of candidate patterns than was actually expected. CARMA[19] produced many more candidate patterns compared to Apriori and DIC and was not performing as the authors claimed [12]. The first algorithm that had used a support lower bounding technique was MaxMiner and Apriori-LB as proposed by Bayardo [20]. Other than this, researchers have attempted have been to tightly forecast the number of candidate patterns that yet remain to be checked as has been proposed by [21].

4. Heuristic h1-max for Real-Time Frequent Pattern Mining

In this paper, we propose a heuristic, h1-max. The h1-max heuristic has been designed for heuristically determining the frequency of a handful of patterns, provided that it satisfies some constraints (which will be discussed later in this paper), and actually not checking their frequencies and continue searching for other frequent patterns.

Researchers have found that the problem of finding all frequent patterns from the database becomes complex as it is an exhaustive search and is NP-hard [22]. Moreover with increasing number of records in the database and with the increasing number of parameters, the search space becomes huge. Keeping these challenges of mining all frequent patterns in mind, our basic objective for in designing a heuristic will be as follows:

- check lesser number of patterns for arriving at the total number of the frequent patterns once the search is completed
- take lesser time to complete the search
- will need a heuristic that will give us a solution set once the search for all frequent patterns is complete.
- real-time performance of the heuristic search technique will be better than that of the brute force version as we are considering the patterns depending on their individual merit (where by merit we mean higher length patterns with a better promise or chance of being frequent).

4.1 Heuristic *h1-max*

The heuristic h1-max consists of finding a lower-bound value and estimating an upper-bound value of each k-length pattern generated by merging two (k-1)-length patterns with common prefix and heuristically assigning a frequency to a pattern that can be said as heuristically frequent provided it satisfies the condition of being frequent as discussed later in this section. The support lower bounding technique that has been used is the lower bounding had been proposed by [20]

- **Lower Bound Support of a pattern.**

Let X, Y, Z $\subseteq$ *I be itemsets then,*
Lower Bound support (X $\cup$ *Y* $\cup$ *Z)*
= lb (X $\cup$ *Y* $\cup$ *Z)*
$\geq$ *support (X* $\cup$ *Y) + support (X* $\cup$ *Z) – support (X)*
……(1)

In practice, this lower-bound support value of a pattern can be obtained and used in the following way. Say, for example, we have a pattern *abc* by merging 2-length frequent patterns *ab* and *ac*. Then, the lower-bounded support value of this pattern *abc* will be given by equation (1) as follows:
Lower-bound of support of abc = lb(abc) =support of (ab) + support of (ac) – support of(a)……(2)
However, for calculating the lower-bounded support value of a pattern, we should exactly know the frequency of two of its generating subsets (generating subsets are subsets from which the pattern was generated by merging) and also the frequency of the immediate common subset of these generating subsets. Nevertheless, researchers till now have suggested that we have to have the exact count of these required sets (in case of equation 2, we need to know the count of ab, ac and a), and hence not much mileage can be achieved by using the lower-bounded support value and predicting a pattern as frequent if the calculated lower-bounded support value is found to be more than the given user-defined support threshold. In this context, we propose a heuristic upper-bounded value using which we can assign a heuristic frequency to a pattern.

- **Upper Bound Support of a pattern using heuristic h1-max**

Let X, Y, Z $\subseteq$ *I be itemsets then,*

Upper Bound support (X $\cup$ *Y* $\cup$ *Z)*

= ub (X $\cup$ *Y* $\cup$ *Z)*

= h1 (X $\cup$ *Y* $\cup$ *Z)*

= minimum {support(X $\cup$ *Y), support (X* $\cup$ *Z) …} * k, 0* $\leq$ *k* $\leq$ *1……. (3)*

Here h1 is the heuristically calculated upper bound of the pattern (X $\cup$ *Y* $\cup$ *Z) from the degrading factors of its immediate subsets. The upper-bound value using heuristic h1 is calculated using the support of its immediate subsets either found frequent or has been made heuristically frequent till the point of time when the upper bound support of the pattern under consideration is being calculated. If any subset becomes frequent at a later point of time, the upper bound value is updated.*

The degrading factor, k, of a pattern is defined as

$$k = \begin{cases} 1 & \text{if pattern-length} = 2 \\ \dfrac{(\textit{actual}\ \text{frequency of the pattern})}{(\textit{estimated}\ \text{frequency of the pattern})} & \text{for all other cases} \end{cases} \quad \ldots\ldots\ldots (4)$$

The upper-bound support value of a pattern is calculated taking into consideration its generating subsets. If however, other subsets of this pattern are found to be frequent, while we are checking for the presence of all its subsets, the upper bound value has to be updated taking into account the support (i.e the frequency) of its other subsets. Thus, if we generate the pattern *abc* by merging two of its subsets *ab* and *ac*, both having a common prefix, we say that the upper-bound support value of abc as follows:

Upper-bound support of abc

= ub(abc) =h1 (abc)

*= minimum { support(ab), support(ac)} * k, where k is the degrading factor.*

Once we are checking for the presence of all the immediate subsets of the pattern *abc* and find that its immediate subset *bc* is also frequent, we update the upper-bound support of *abc*.

The degrading factor, *k*, plays a very crucial role in the upper-bound support value of a pattern. Let us consider a pattern of *n-length*. This means that this pattern will have *n* subsets of *(n-1)-length*. Each of these subsets will have a degrading factor (as calculated from equation 4) and let us designate the degrading factor k_i for the *i-th (n-1)-length* subset of the *n-length* pattern under consideration. Thus k_1, k_2, k_3 ….. k_n represents the degrading factor of these n subsets. Now the degrading factor for calculating the upper-bound support can be as follows:

h_1 *max* = h1 heuristic using maximum degrading factor of the item sets in the same level that are subsets of the same superset (and this superset has been found to be frequent)

= max { k1, k2, k3 … kn} ……(5)

Let us assume that the pattern *abcd* is formed by merging its immediate subsets with a common prefix, *abc* and *abd*. Let the heuristically calculated frequency of the patterns *abc* and *abd* be *u(abc)* and *u(abd)* respectively. Thus, the degrading factor of the pattern *abc* and *abd* will be:

k_{abc} *= (actual frequency of abd)/u(abd) ……(8)*

k_{abd} *= (actual frequency of abd)/u(abd)……(9)*

Using heuristic h_1 *max*:

k max(abcd) = max { $k_{(abc)}, k_{(abd)}$} *……(10)*

Thus the upper-bound frequency calculated using h1 max will be:

ub(abcd) = min { supp (abc), supp (abd)}* *k* max(abcd) = minimum {support (abc), support (abd)} * max {$k_{(abc)}, k_{(abd)}$}……. (11)

5. BDFS(b)-h1-max: An Efficient Heuristic Search Algorithm for Real-Time Frequent Pattern Mining Using Heuristic h1-max

BDFS(b)-h1-max is a heuristic search algorithm for finding frequent patterns. We use the vertical database format of tid-lists for intersecting and finding out the frequency of a particular pattern. In BDFS(b)-h1-max we calculate the lower-bound and the upper-bound value of a pattern the moment it is generated. If we find that the calculated lower-bound support value is more than that of the user-defined support threshold and none of its subsets has been found to be infrequent till this point of time, we go ahead and mark this pattern as *Heuristically Frequent*, assign the calculated upper-bound support value as its heuristically assigned frequency and move this pattern to the set of the frequent itemsets of corresponding length. The upper-bound support value (and hence the heuristically assigned frequency or support) of a pattern is updated the moment any immediate subset of this pattern is found to be frequent. As we now have the lower-bound and the upper-bound support value for each pattern, we keep the global pool of candidate itemsets sorted in descending order of length. Patterns of same length are sorted on their lower-bound values and then patterns with same length and same lower-bound values are sorted on their upper-bound values. 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. Once the user specifies a desired support value, all frequent patterns of length 1 and 2 (where F(n) means frequent pattern of length-n) are obtained from M. Then BDFS(b)-h1-max starts its search for frequent patterns of higher lengths from this point forward

The most salient features of BDFS(b)-h1-max 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. Once a pattern is newly generated, it finds the lower-bound and upper-bound support values using the heuristic h1. For patterns, whose lower-bound support value is greater than the given support threshold, it marks these patterns as heuristically frequent and move to the set of frequent items of corresponding length and assigns the calculated upper-bound value as the heuristic frequency of this pattern and the degrading factor the same as that of the degrading factor of the pattern. 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 (later in the paper, we have shown how the performance of BDFS(b)-h1 is affected with changing block size b) for purposes of academic curiosity to find how it affects the performance of the algorithm. We intend to implement in a future work, the automatic decision of block b depending on various decision making criterion. We have implemented this algorithm with the prefix based tree, called TRIE, data structure for implementing BDFS(b)-h1. We have kept the nodes of TRIE lexicographically ordered.

Let us consider the following example to show how BDFS(b)-h1-max works.

Algorithm BDFS(b)-h1-max:

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. For patterns of same length, sort candidate patterns according to their lower-bound values. Resolve ties by giving higher preference to candidate patterns with higher upper-bound values. 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 any immediate superset of the pattern is present in the Border Set increase its sub-itemset counter. If the sub-itemset 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 and Heuristically Frequent marked patterns from list of frequent patterns.*
4. *Generate all patterns of next higher length from the newly obtained frequent patterns at step 2 by merging them with frequent patterns of same length found previously or from the current block. Calculate the lower-bound and upper-bound value of these newly generated patterns using h1-max-heuristic. Check for all immediate subsets of the newly generated pattern.*
5. *If the lower-bound value of the newly generated pattern > given minimum support value, mark the pattern as Heuristically Frequent and put it in the list of frequent patterns along with its heuristically estimated value, even if all its immediate subsets are not found to be frequent as yet. If the support value > lower-bound value, and all immediate subsets of the newly generated pattern found till this point of time are frequent then put the pattern in the global pool of candidate patterns along with its calculated lower-bound and upper-bound values else put it in the Border Set if the pattern length is > 3.*
6. *Take a block of most promising b candidate patterns from the global pool.*
7. *If block b is empty and no more candidate patterns left, output frequent patterns and exit.*
8. *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)-h1-max

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

Transaction ID	Transaction of items
1	abcde
2	acde
3	ade
4	bcde
5	bde
6	abd
7	abd
8	abcd
9	de
10	acde
11	abcde
12	ace

Figure 4. An example of transaction data, D.

Now we proceed as follows:

Step I. Given this set of transactions D, we create a support-independent two dimensional lower triangular matrix M (as in figure 6.4) using procedure *Create_Matrix* (as shown in figure 6.3) and the transaction id lists (as shown in figure 6.5). This step of creating the support-independent lower triangular and the tid-list of 1-length patterns is support independent step and we refer this step through out this paper as a *pre-processing step*. Once this support independent step is conducted, the matrix M and the tid-list of 1-length items are stored in the hard-drive once and for all for all future references.

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.*

Figure 4. Procedure *Create_Matrix*

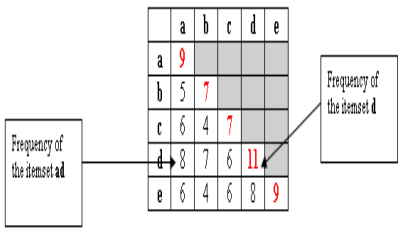

Figure 5. Matrix M

Step II. Let the absolute support, ξ (abs), given for running BDFS(b)-h1-max be 3. Cells of Matrix M (figure 6.4) 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. We calculate the degrading factor for the patterns of length 2. We calculate the upper-bound frequencies of the 2-length patterns approximated from the F(1). Thus, we have:

ub(ab) = min (a,b) = min (9,7) = 7. Actual frequency of ab = 5

Thus the degrading factor for ab, ***k(ab)* = 5/7**

Similarly, we find the degrading factors of other 2-length patterns. Thus we have: ***k(ac)* = 6/7; *k(ad)* = 8/9; *k(ae)* = 6/9; *k(bc)* = 4/7; *k(bd)* = 1; *k(be)* = 4/7; *k(cd)* = 6/7; *k(ce)* = 6/7; *k(de)* = 8/9**

Step III. Two 2-length patterns are merged if their first elements match. Thus we have from equation 2:

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

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 6. The tid-lists of the items

Step IV. Find if all the subsets of new merged patterns are frequent. For any 3-length newly merged pattern, if all its 2-length subsets are not present, then the pattern is pruned (using the support monotonicity property [14] If all its 2-length subsets are present we calculate the lower-bound and the upper-bound heuristics of these patterns. For all patterns, whose lower-bound heuristic value is ≥ support threshold; we directly mark the pattern as a *Heuristically Frequent* pattern of corresponding length and assign it a heuristic frequency for further references. We move this pattern to the list of frequent patterns along with is heuristically assigned frequency and marking it as *Heuristically Frequent*. Otherwise we check if all of its subsets are frequent. If yes, we move it to the global pool of candidate itemsets, else we move it to the Border set of corresponding length and wait till all its subsets become frequent. If however, any of the subsets of this border-set pattern becomes infrequent, this border-set pattern in turn is made infrequent and is pruned. However, where lb = ub, we take the approx freq = lb. We will take the value rounded off to the nearest integer. Thus, we have:

abc: lb = ab + ac – a = 5 + 6 – 9 = 2;
ub = min (ab, ac, bc) * *k*max = min(5,6,4)* max (k_{ab}, k_{ac}, k_{bc}) = 4 * max (5/7, 6/7, 4/7)
= 4* 6/7 ≈ 3

abd: lb = ab + ad – a = 5 + 8 – 9 = 4;
ub = min (ab,ad, bd) * *k*max = min(5,8,7) * max (k_{ab}, k_{ad}, k_{bd}) = 5 * max (5/7, 8/9, 1) = 5

abe: lb = ab + ae – a = 5 + 6 – 9 = 2; ub = min (ab, ae, be) * *k* max = min(5,6,4) * max (k_{ab}, k_{ae}, k_{be}) = 4 * max (5/7, 6/9, 4/7) ≈ 3

Similarly: **acd:** lb = 5; ub = 5; **ace:** lb = 3; ub = 5.; **ade:** lb = 5; ub = 5.; **bcd:** lb = 4; ub = 4.; **bce:** lb = 1; ub = 3.; **bde:** lb = 4; ub = 4.; **cde:** lb = 5; ub = 5.

Thus in short we have the following patterns with the lower bound and upper bound denoted in the parentheses [where the first number in the parentheses denotes the lb and the second the ub]:

newly merged patterns = {abc (2,3), abd (4,5), abe(2,3), acd(5,5), ace(3,5), ade(5,5), bcd(4,4), bce(1,3), bde(4,4), cde(5,5)}…….. (4)

As mentioned above, for all patterns, whose lower bound heuristic value ≥ support threshold, we mark the pattern as *Heuristically Frequent* and assign it its upper-bound value as a heuristically found frequency. For all other patterns, we check if all its subsets are frequent and move it to the global pool of candidates. Thus, we have that the patterns *abd, acd, ace, ade, bcd, bde, cde* which have their lower-bound heuristic values greater than the given minimum support threshold are marked as *Heuristically Frequent* patterns of length 3, and is moved to the list of frequent patterns of length 3, *F(3)*, along with the heuristically assigned frequency (its upper-bound support) and the mark that it is heuristically frequent. The rest of the patterns are moved to the global pool of candidate patterns. Thus, we have:

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

C( ) = {abe (2,4), abc (2,4), bce (1,3)} ……(6)

We keep the global pool of candidates sorted on the length, lb and ub respectively and have assigned the *Heuristically Frequent* patterns with the calculated upper-bound as their heuristically assigned frequency.

Step V. We start finding the frequencies of the patterns lying in the global pool of candidate itemsets. If the block size b is assumed to be 4, then we have that all the candidate patterns are checked in the first block b itself. *b = { abc, abe, bce} ……(5)*

We intersect the tid-lists and find the frequencies of these itemsets. Thus, we have:

b = {abc(3), abe(2), bce(3)} ……(6)

As the given support threshold is 3, we prune ace Thus from this block, the frequent patterns obtained are:

F(3) = { abc(3), bce(3)}……(7)

We merge the newly found 3-length frequent patterns from the current block with the existing set of frequent patterns of length-3. Thus, we have:

F(3) = { abd(HF, 5), acd(HF, 5), ace(HF, 5), ade(HF, 5), bcd(HF, 4), bde(HF, 4), cde(HF, 5), abc(3), bce(3)}……(8)

Step VI We now merge the frequent patterns of length-3 in the F(3) to find higher-length patterns of length-4 and calculate the lower-bound and upper-bound support values of these patterns. Thus we have:

newly merged patterns = {abcd, acde, bcde} ……(9)

We now calculate the lower-bound and upper-bound support of these patterns newly merged.

abcd: lb = abc + abd – ab = 3 + 5 – 5 = 3;
ub = min supp of {abc, abd, acd, bcd) * *k* max
= min (3, 5, 5, 4) * max (1,1,8/9,1) = 3. .... *(10)*

acde: lb = acd + ace – ac = 5 + 5 – 6 = 4
ub = min support of (acd, ace, ade, cde) * *k* max
= 5 * max (8/9, 6/7,8/9,8/9) ≈ 4 …… *(11)*

bcde: lb = bcd + bce – bc = 4 + 3 – 4 = 3
ub = min supp of (bcd, bce, bde, cde) * *k* max
= min (4, 3, 4, 5) * max (k_{bcd}, k_{bce}, k_{bde}, k_{cde}) = 3 ... *(12)*

From equations 10, 11 and 12, we find that all the three patterns have their lower-bound heuristics calculated to be more than the user-defined support threshold. Hence these patterns are made *Heuristically Frequent* and are moved to the frequent set of 4-length patterns along with their upper-bound values as the heuristically assigned values. Thus, we have:

F(4) = { abcd(HF,3), acde(HF, 4), bcde(HF, 3)} ...(13)

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, or if the total executable time τ, has come to an end, 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) = { abd(HF, 5), acd(HF, 5), ace(HF, 5), ade(HF, 5), bcd(HF, 4), bde(HF, 4), cde(HF, 5), abc(3), bce(3)}
F(4) = { abcd(HF,3), acde(HF, 4), bcde(HF, 3)} …….. (14)

If the user now wants to have the set of the frequent patterns only, then in another iteration we can find the actual frequencies of the patterns with the flag HF (i.e. *Heuristically Frequent*) in the set of frequent patterns. The block size b can now be varied to show how it affects the execution time of the algorithm. In a later section, we show and discuss this effect. BDFS(b)-h1 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.

6. 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 the BDFS(b)-h1-max heuristic search technique, we have tested it on various datasets, both sparse and dense, which on the other hand were both taken from synthetic and real-life datasets. These includes real-life datasets like BMS-Webview-1[23], Kosarak[1], CHESS[2] and a host of synthetic datasets like T10I8D100K, T10I4D100K, T25I10D100K[3] etc. The experiments were performed on The experiments were performed on a Red-Hat Linux machine with 1GB RAM and 20 GB HD with Pentium IV 2.24Ghz processor.

6.1 Comparison of BDFS(b)-h1-max with Existing Algortihms

In order to show how BDFS(b)-h1-max performs when it is allowed to run to complete execution, we have chosen to compare it with existing bench-marked algorithms like the Apriori, FP-Growth, Eclat, dEclat and DIC[4]. Since FP-Growth, Eclat and dEclat is known to perform and scale better than that of the Apriori algorithm, we have chosen to take FP-Growth, Eclat and dEclat as the benchmark and compare the execution time of these algorithms with that of BDFS(b)-h1-max. In some cases, where we have found Apriori to perform better, we have compared our work with Apriori as well. For the sake of curiosity of the number of patterns being checked (as this actually decides the total running time of an algorithm in frequent pattern mining cases), we have chose to compare the performance of our technique with Apriori.

In figures 7, 10 and 12 we have compared the run-time of Eclat and dEclat with BDFS(b)-h1-max for three different datasets and found that our technique compares well with Eclat and dEclat. In figures 7, 10 and 12 we have compared the run-time of Eclat and dEclat with BDFS(b)-h1-max for three different datasets and found that our technique compares well with Eclat and dEclat.

[1] These datasets are publicly available at http://fimi.cs.helsinki.fi/data/
[2] These datasets are publicly available at http://fimi.cs.helsinki.fi/data/
[3] The data generator is available from http://www.almaden.ibm.com/cs/quest//syndata.html#assocSynData
[4] The FP-growth code used for comparison is publicly available at www.cse.cuhk.edu.hk/~kdd/program.html ;
The Eclat code used for comparison is publicly available at http://fuzzy.cs.uni-magdeburg.de/~borgelt/eclat.html;
The Apriori, dEclat and DIC codes used for comparison are publicly available at http://www.cs.helsinki.fi/u/goethals/software/index.html

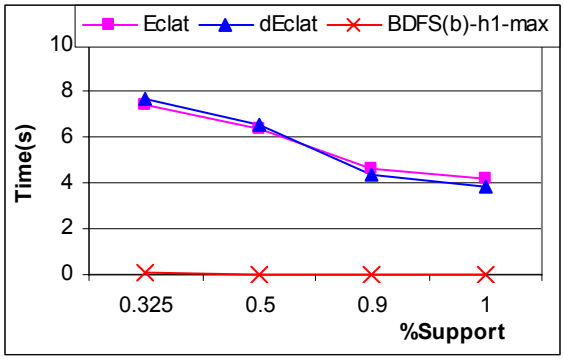

Figure 7. Time comparison of BDFS(b)-h1-max with Eclat and dEclat for T10I8D100K, b=100K

In figures 8, 11 and 13 we have compared the performance of BDFS(b)-h1-max with FP-Growth to find that FP-Growth gets significantly outperformed in all the cases. In figure 9 and 10 we have shown that BDFS(b)-h1-max performs better than that of DIC and Apriori respectively for the corresponding datasets.
Comparing the number of patterns being checked by Apriori and BDFS(b)-h1-max, as shown in figures 14, 15 and 16 for three datasets, we find that in all the

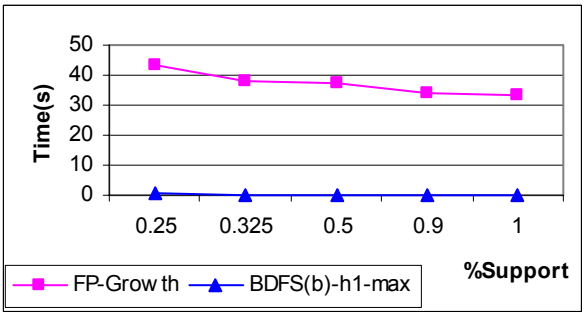

Figure 8. Time comparison of BDFS(b)-h1-max with FP-Growth for T10I8D100K, b=100K.

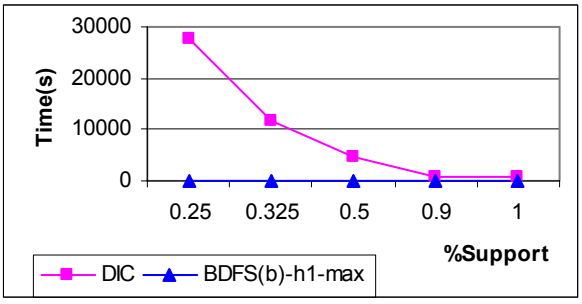

Figure 9. Time comparison of DIC and BDFS(b)-h1-max for varying supports of T10I8D100K, b=100K

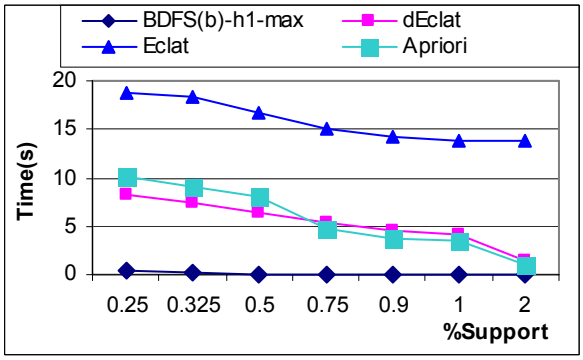

Figure 10. Time comparison of BDFS(b)-h1-max with Apriori, Eclat and dEclat for T10I4D100K, b=100K

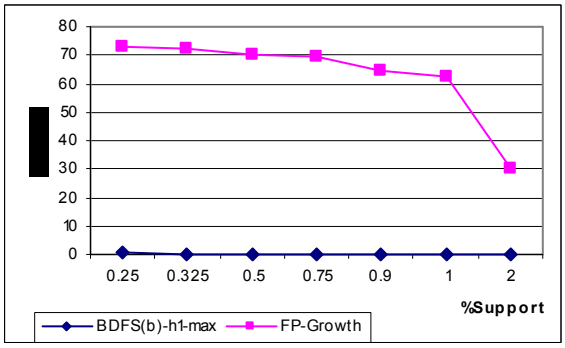

Figure 11. Time comparison of FP-Growth and BDFS(b)-h1-max for T10I4D100K, b=100K

cases BDFS(b)-h1-max checks significantly lesser number of patterns than that of Apriori.
In figures 17, 18 and 19 we have shown the %accuracy of the output for all-frequent patterns that BDFS(b)-h1-max provides for three different datasets under consideration. In all the three cases we find that the accuracy of the result provided is high.

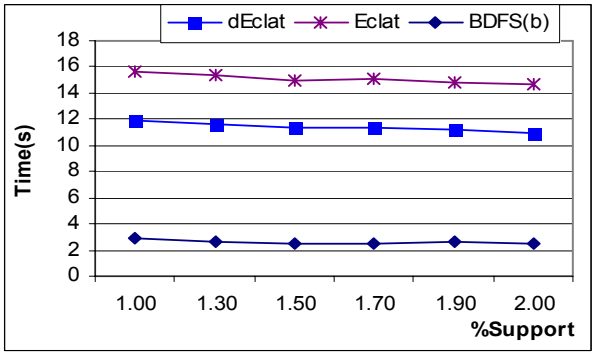

Figure 12. Time comparison of BDFS(b)-h1-max, b=41.2K, with Eclat and dEclat for varying supports of Kosarak

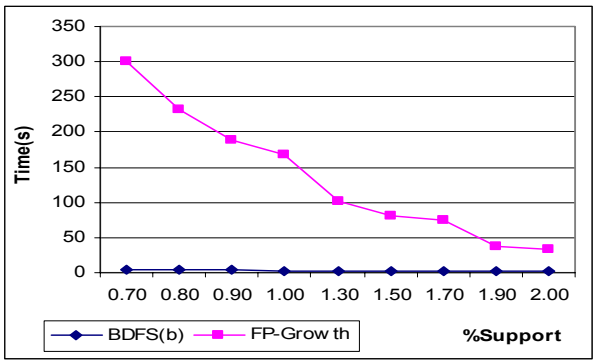

Figure 13. Time comparison of BDFS(b)-h1-max, b=41.2K, with FP-Growth for varying supports of Kosarak

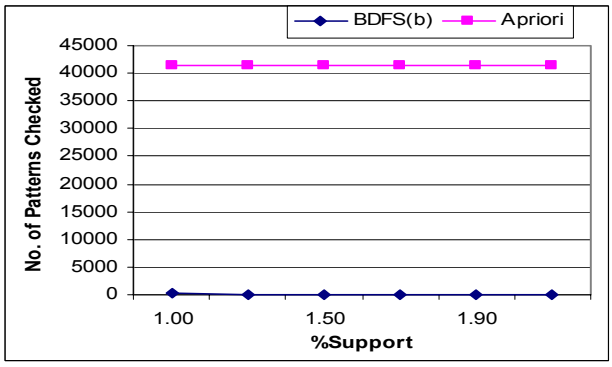

Figure 14. Comparison of number of patterns checked by Apriori and BDFS(b)-h1-max for varying supports of Kosarak

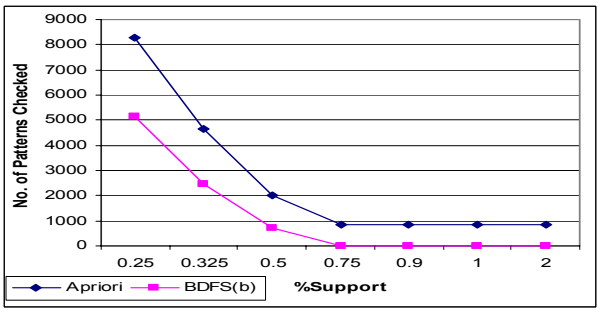

Figure 15. Comparison of number of patterns checked by Apriori and BDFS(b)-h1-max for T10I4D100K

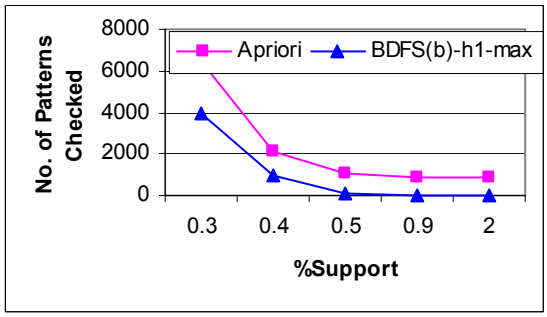

Figure 16. Comparison of number of patterns checked by Apriori and BDFS(b)-h1-max for T10I8D100K

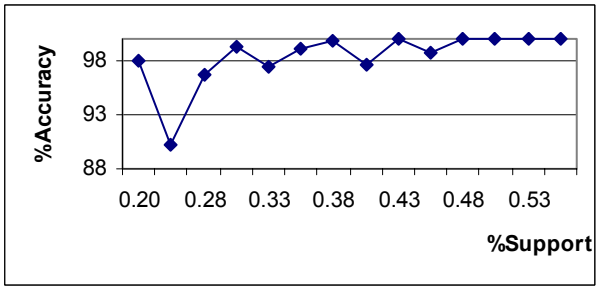

Figure 17. %Accuracy of output by BDFS(b)-h1-max for varying supports of T10I8D100K

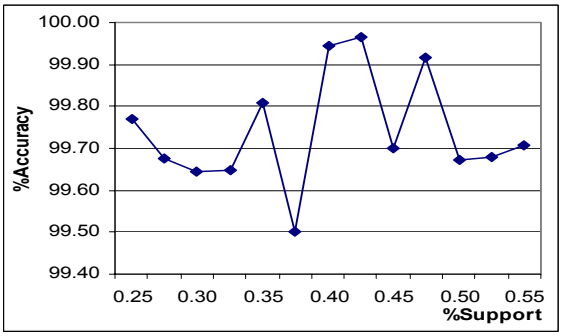

Figure 18. %Accuracy of output by BDFS(b)-h1-max for varying supports of T10I4D100K

The accuracy increases with the increase in the support threshold. This may be easily explained as follows: with the increase in the support threshold the number of frequent patterns of higher lengths drastically decreases and hence there is a significant increase in the accuracy of the output being provided by BDFS(b)-h1-max.

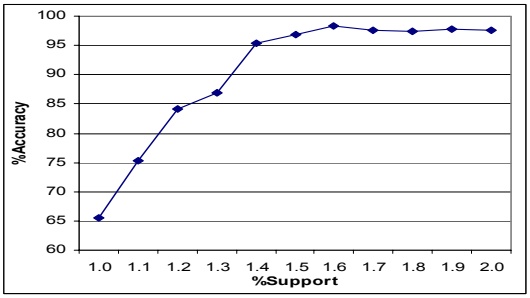

Figure 19. %Accuracy of output bt BDFS(b)-h1-max or varying supports of Kosarak

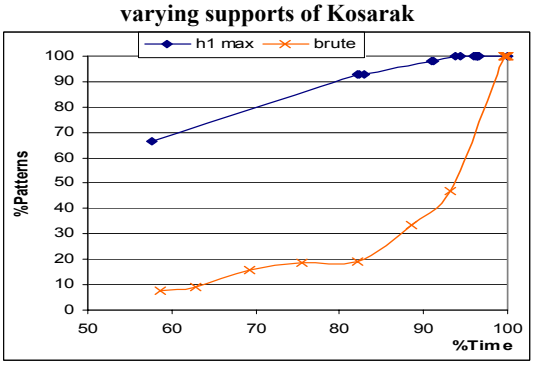

Figure 20. Time-Pattern% of BDFS(b)-h1-max as compared to BDFS(b) brute force search for 0.40% support of Kosarak and b=41270

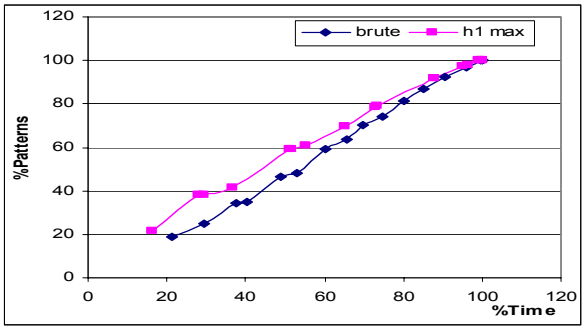

Figure 21. Time-pattern% of BDFS(b)-h1-max as compared to BDFS(b) brute force search for 69% support of Chess and b=7600

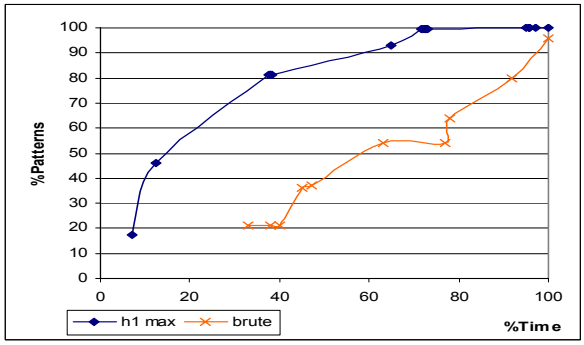

Figure 22. Time-pattern% of BDFS(b)-h1-max as compared to BDFS(b) brute force for 0.08% support of BMS-Webview-I and b=497K

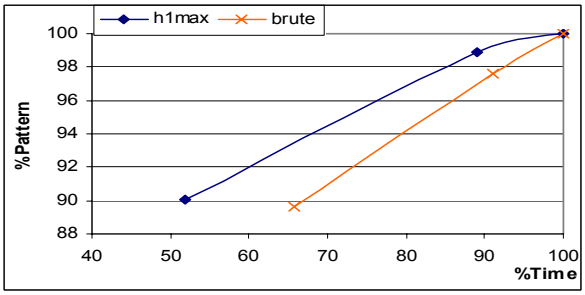

Figure 23. Time-pattern% of BDFS(b)-h1-max as compared to BDFS(b) brute force for 0.775% support of T25I10D100K and b=1K

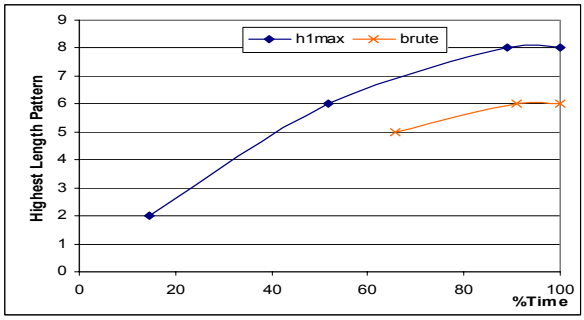

Figure 24. Highest length pattern-time% for 0.775% support of T25I10D100K

6.2 Real-Time Performance of BDFS(b)-h1-max

Real-time performance of BDFS(b)-h1-max has been summarized in figures 20, 21, 22 and 23. In these figures, we have shown the real-time performance of BDFS(b)-h1-max both in the context of real-life and synthetic datasets. We have also compared the real-time performance with brute force BDFS(b). In every cases, we find that the real-time performance has been better than that of the brute force version. From fig 22, for example, we find that the heuristic version provides 80% patterns in 40% time. In fig. 24, we have shown the highest length patterns being found frequent in %completion time. We have intentionally not presented the comparison of the real-time output of existing algorithms as they are not meant for performing in real-time, not even for the sake of academic curiosity. Authors may be contacted for these results.

7. Conclusion

Traditionally frequent pattern mining has been treated as an offline task. Real-time frequent pattern mining was introduced in [3, 4] looking at the needs in real-life business scenarios [24]. However, the approach to this was a brute-force search. In this paper, we have introduced the heuristic h1-max along with the heuristic search technique BDFS(b)-h1-max, which can respond to the real-time requirements more efficiently. Real-time frequent pattern mining will have a great impact in real-time decision making, especially in the scenarios like real-time optimization for managing supply chains and real-time predictive modeling. This algorithm may be modified for matching the memory and time constraints dynamically. It may be worthwhile to extend this work in scenarios of grid-computing where data is being gathered from a multi-agent grid-network. More domain specific heuristics for business, if developed, and used along with the technique proposed in this paper, will make real-time frequent pattern mining more efficient.

8. Reference:

[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] R. Dass and A. Mahanti, "Frequent Pattern Mining in Real-Time – First Results," presented at TDM2004/ACM SIGKDD 2004, Seattle, Washington USA, 2004.

[4] 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.

[5] R. Dass and A. Mahanti, "Implementing BDFS(b) with Diff-Sets for Real-Time Frequent

Pattern Mining in Dense Datasets – First Findings," presented at UDM 2005/IEEE International Conference of Data Engineering 2005 (ICDE 2005), Tokyo, Japan, 2005.
[6] R. Dass and A. Mahanti, "Fast Frequent Pattern Mining in Real-Time," presented at Proceedings of the Eleventh International Conference on Management of Data (COMAD 2005), Goa, India, 2005.
[7] A. Das, W.-K. Ng, and Y.-K. Woon, "Rapid Association Rule Mining," presented at CIKM 2001, Atlanta, Georgia, USA, 2001.
[8] S. Dutta, "Adapting Decisions, Optimizing Facts and Predicting Figures: Can Confluence of Concepts, Tools, Technologies and Standards Catalyze Innovation?," Massachusetts Institute of Technology, Massachusetts August 2004 2004.
[9] J. Dyche, "Real Time or Right Time-Explaining The Real Time Enterprise," vol. 2004: CRM Guru, 2003.
[10] OpenServiceInc., "Real-Time Enterprise Risk and Vulnerability Management," vol. 2004: Open Service Incorporation, 2004.
[11] 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.
[12] B. Goethals, "Survey on Frequent Pattern Mining," vol. 2004. Helsinki, 2003, pp. 43.
[13] 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.
[14] 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.
[15] 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.
[16] R. Agarwal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *Proceedings of the 20th International Conference on Very Large Databases (VLDB)*, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. Santiago, Chile: Morgan Kaufmann, 1994, pp. 487-499.
[17] J. Hipp, U. Guntzer, and G. Nakhaeizadeh, "Mining Association Rules: Deriving a Superior Algorithm by Analyzing Today's Approaches," in *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, vol. 1910 of Lecture Notes in Computer Science, D. A. Zighed, H. J. Komorowski, and J. M. Zytkow, Eds.: Springer, 2000.
[18] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," in *Proceedings of the ACM SIGMOD Conference*, 1997, pp. 255-264.
[19] C. Hidber, "Online Association Rule Mining," in *Proceedings ACM SIGMOD International Conference on Management of Data*. Philadephia, Pennsylvania: ACM, 1999, pp. 145-156.
[20] R. J. Bayardo Jr., "Efficiently mining long patterns from databases.," in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, vol. 27(2) of SIGMOD Record, L. M. Haas and A. Tiwary, Eds.: ACM Press, 1998, pp. 85-93.
[21] F. Goerts, B. Goethals, and J. Van den Bussche, "A Tight Upper Bound on the Number of Candidate Patterns," in *Proceedings of the 2001 IEEE International Conference on Data Mining*, N. Cercone, T. Y. Lin, and X. Wu, Eds.: IEEE, 2001, pp. 155-162.
[22] M. J. Zaki and M. Ogihara, "Theoratical Foundations of Association Rules," presented at DMKD'98, Seattle, 1998.
[23] R. Kohavi and F. Provost, "Applications of Data Mining to Electronic Commerce," *Journal of Data Mining and Knowledge Discovery*, vol. 5, pp. 5-10, 2001.
[24] M. S. Rahman, N. L. Martin, and S. Paul, "Data Mining, Group Memory, Group Decision Making: A Theoretical Framework," presented at Ninth Americas Conference on Information Systems, 2003.