Competent Accomplishment of Diffset Based Mining Algorithm

Dr.R.Sabitha[1] Ms.T.Mythili[2] Ms.R.D.Priyanka[3]
[1]Associate Professor [2,3]Assistant Professor
[1,2]Department of Information Technology [3]Department of Computer Science & Engineering
[1,2,3]Info Institute of Engineering, Kovilpalayam, Coimbatore 641 107, Tamilnadu, India

Abstract— *Data mining is the process of automatically discovering useful information in large data repositories. One of the important problems in data mining is discovering association rules from databases of transactions where each transaction consists of a set of items. Frequent itemsets play an important role in many data mining tasks. The task of discovering all frequent itemsets is a fundamental problem in data mining. In this paper we examine the problem of finding frequent dEclat algorithm. The Mushroom dataset contains characteristics of various species of mushrooms, and was originally obtained from the UCI Repository of Machine Learning Databases [27]. dEclat is a best-known algorithm for mining frequent item sets in a set of transactions. In this paper we describe implementations of this algorithm that use several optimizations to achieve maximum performance, with respect to both execution time and memory usage. The dEclat implementation uses (sparse) bit matrices to represent transactions lists and to filter closed and maximal item sets.*

Keywords— Diffsets, Frequent Itemsets, Association Rule Mining, dEclat

I. INTRODUCTION

Mining frequent patterns or itemsets is a fundamental and essential problem in many data mining applications. These applications include the discovery of association rules, strong rules, correlations, sequential rules, episodes, multi-dimensional patterns, and many other important discovery tasks [10]. The problem is formulated as follows: Given a large data base of item transactions, find all frequent itemsets, where a frequent itemset is one that occurs in at least a user-specified percentage of the data base. Mining frequent itemsets (or frequent patterns) has been studied extensively in the research area of data mining. Particularly, it has played an important role in mining association rules, correlations, causality, sequential patterns episodes ,Multidimensional patterns, max-patterns, partial periodicity, emerging patterns and many other important data mining tasks. Most of the previous works on mining associations are based on the traditional horizontal transactional database format a variant of Apriori-like approach. However, a number of vertical mining algorithms have been proposed recently for mining associations. In the vertical format, each item is associated with its corresponding tidset, the set of all transactions (or tids) which contains that item. Mining algorithms on the vertical format have shown to be very effective and usually outperform horizontal approaches. This advantage stems from the fact that frequent patterns can be counted via tidset intersections, instead of using complex internal data structures (Candidate generation and counting occur in a single step). Vertical mining approaches take advantages on no distinction of candidate generation and support counting phases. Diffset [5, 25] is a vertical data representation that only keeps track of the difference in the tids of a candidate pattern from its generating frequent patterns. It drastically cut down the size of memory required to store intermediate results. The initial database stored in the diffset format, instead of the tidset, can also reduce the total database size.

In this paper, we implement this novel vertical mining algorithm dEclat on mushroom dataset, which can mine a complete set of frequent patterns on Diffset structure. It utilizes a pattern growth strategy for efficiently enumerating all complete sets of frequent patterns. This algorithm recursively generates all subsets from 1-itemset condition patterns without computing the support values of all their subsets and, also, there are no candidate generation procedure and no complex data structures such as hash trees. It can be instantiated using either the traditional vertical tidset or the diffset structure. The remaining of the paper is organized as follows. Section 2 introduces the concept of diffset structure and its construction method. Section 3 explains the frequent pattern-mining algorithm, called dEclat on the Diffset structure. Section 4 presents the result of our experimentation. Section 5 summarizes our work and discusses future research issues.

II. CONCEPTS AND PRE-LIMINARIES

A. Common Data Formats

Figure 1 illustrates some of the common data formats used in association mining. In the traditional horizontal approach, each transaction has a tid along with the itemset comprising the transaction. In contrast, the vertical format maintains for each item its tidset, a set of all tids where it occurs. Most of the past research has utilized the traditional horizontal database format for mining; some of these methods include Apriori [1], which mines frequent itemsets, and MaxMiner [4] and DepthProject [2] which mine maximal itemsets. No table exceptions to this trend are the approaches that use a vertical database format, which include Eclat [22], Charm [21], and Partition [18]. Viper [20] and MaFia [6] use compressed vertical bit vectors instead of tidsets.

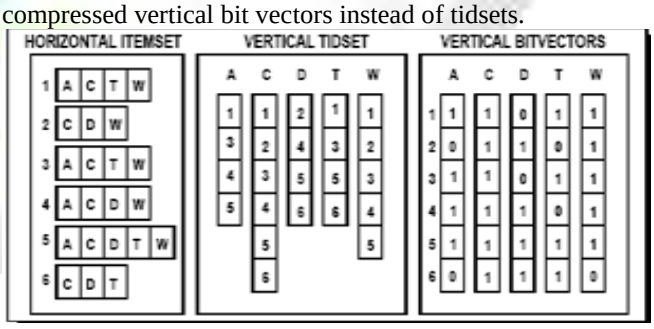

Fig. 1: Common Data Formats

B. Diffsets

Let I = {a1, a2… am} be a set of items, and DB = {T1, T2… Tn} be a transaction database, where Ti (i = {1, …, n}) is a transaction which contains a set of items in I. In addition, an index i is a unique identifier (called tid) of a transaction Ti. The support (or occurrence frequency) of an

itemset (or a pattern) X, denoted by σ(X), is the number of transactions in DB that contain X. An itemset X is called a frequent itemset (or a frequent pattern) if its support is no less than a predefined minimum support threshold, ξ %. [25].

To illustrate the concept of Diffset structure, it is assumed that DB = {1, 2, 3, 4, 5}, as shown in Table 1, be an example of a transaction database, and I = {A, B, C, D, E} be a set of five different items in the database. Figure 2 depicts a common data format that has been used often in mining associations. In this traditional horizontal approach, each transaction has a tid along with the itemset comprising the transaction. In contrast, the vertical format maintains for each item its tidset, the set of all tids where the item occurs. The Diffset structure is based on the concept of vertical database format. It avoids storing the entire tidset of each item by keeping track of only the differences in the tids between the tidset of each item and the whole tidset (the set of all tids). These differences in tids are stored in what has been called the diffset. When we set a minimum support to the Diffset structure, an item ai whose corresponding tidset has the cardinality more than DB - DB * ξ /100, must be cut off. Figure 2(a) shows Diffset structure with ξ = 50%. In Figure 3(b), Diffset structure is sorted by the support in an ascending order and, hence, it has a better chance that more postfix can be shared. [25]

TID	Item Bought
1	A,T,W
2	C,D,W
3	A,C,T,W
4	A,C,D,W
5	A,C,D,T,W
6	C,D,T

Table 1: A transaction database

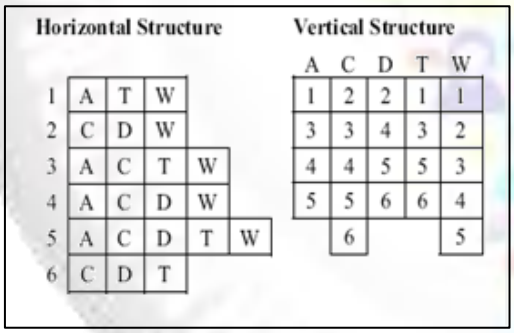

Fig. 2: Data Format

Figure 4 shows the different regions for the tidset and diffsets of item a and b.Let t(a) denote the tidset of an element a. Also, the diffset of an element a is denoted by d(a) and defined by d(a) = t(a') = U - t(a) , and the diffset of an itemset ab defined by d(ab) = t(a) - t(b) = t(b') - t(a'). We can compute the support of an itemset ab on diffset structure by evaluating the cardinality of Area (Δ) shown in Figure 4.

Thus, the support of itemset ***ab*** is

$$\sigma(ab) = |t(a) \cap t(b)|$$
$$= |\sigma(a)| - |d(ab)|$$

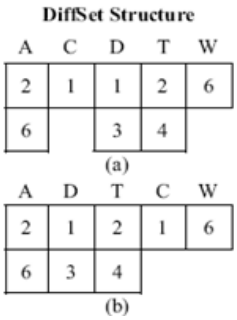

Fig. 3: Diffset Data Format

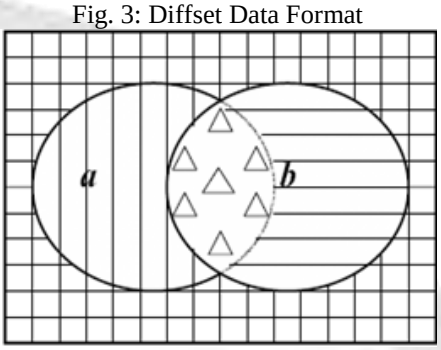

Fig. 4: Compute Support

III. DECLAT - DIFFSET BASED MINING

Eclat performs a depth-first search of the subset tree. Our experiments show that diffsets allow it to mine on much lower supports than other methods like Apriori and the base Eclat method.

Frequent itemsets are generated by computing diffsets for all distinct pairs of itemsets and checking the support of the resulting itemset. A recursive procedure call is made with those itemsets found to be frequent at the current level. This process is repeated until all frequent itemsets have been enumerated. In terms of memory management it is easy to see that we need memory to store intermediate diffsets for at most two consecutive levels within a class. Once all the frequent itemsets for the next level have been generated, the itemsets at the current level within a class can be deleted.

Our dEclat implementation represents the set of transactions as a (sparse) bit matrix and intersects rows to determine the support of item sets. The search follows a depth first traversal of a prefix tree as it is shown in Figure5.

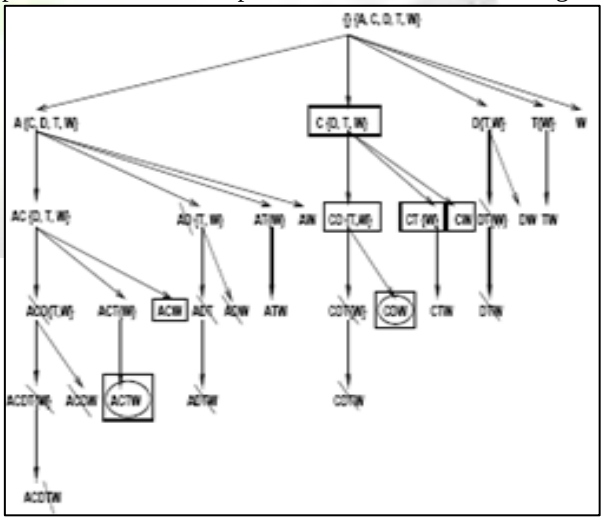

Fig. 5: Subset Search Tree

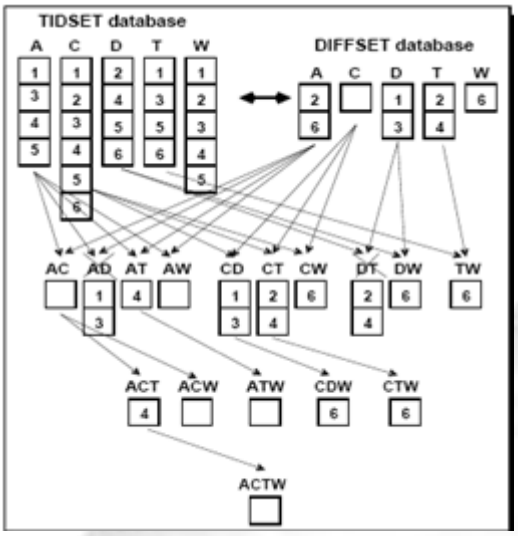

Fig. 6: Diffsets for pattern Counting

A. *Bit Matrices*

A convenient way to represent the transactions for the dEclat algorithm is a bit matrix, in which each row corresponds to an item, each column to a transaction (or the other way round). A bit is set in this matrix if the item corresponding to the row is contained in the transaction corresponding to the column, otherwise it is cleared. There are basically two ways in which such a bit matrix can be represented: Either as a true bit matrix, with one memory bit for each item and transaction, or using for each row a list of those columns in which bits are set. Which representation is preferable depends on the density of the dataset. On 32 bit machines the true bit matrix representation is more memory efficient if the ratio of set bits to cleared bits is greater than 1:31. A sparse representation should be used even if the ratio of set bits to cleared bits is greater than 1:31. In our current implementation a sparse representation is preferred if the ratio is greater than 1:7, but this behavior can be changed by a user. A more sophisticated option would be to switch to the sparse representation of a bit matrix during the search once the ratio of set bits to cleared bits exceeds 1:31. [26]

B. *Search Tree Traversal*

dEclat searches a prefix tree like the one shown in Figure 5 in depth first order. The transition of a node to its first child consists in constructing a new bit matrix by intersecting the first row with all following rows. For the second child the second row is intersected with all following rows and so on. The item corresponding to the row that is intersected with the following rows thus is added to form the common prefix of the item sets processed in the corresponding child node. Of course, rows corresponding to infrequent item sets should be discarded from the constructed matrix, which can be done most conveniently if we store with each row the corresponding item identifier rather than relying on an implicit coding of this item identifier in the row index. Intersecting two rows can be done by a simple logical *and* on a fixed length integer vector if we work with a true bit matrix. During this intersection the number of set bits in the intersection is determined by looking up the number of set bits for given word values (i.e., 2 bytes, 16 bits) in a precomputed table. For a sparse representation the column indices for the set bits should be sorted ascendingly for efficient processing. Then the intersection procedure is similar to the merge step of merge sort. In this case counting the set bits is straightforward. After the traversal the difference in tids is calculated as shown in figure [26].

C. *Filtering Item Sets*

Determining item sets with dEclat is slightly more difficult than with Apriori, because due to the backtrack dEclat "forgets" everything about a frequent item set once it is reported. In order to filter for item sets, one needs a structure that records these sets, and which allows to determine quickly whether in this structure there is an item set that is a superset of a newly found set (and whether this item set has the same support if closed item sets are to be found). In our implementation we use the following approach to solve this problem: Frequent item sets are reported in a node of the search tree *after* all of its child nodes have been processed. In this way it is guaranteed that all possible supersets of an item set that is about to be reported have already been processed.

Consequently, we can maintain a repository of already found (closed or maximal) item sets and only have to search this repository for a superset of the item set in question. The repository can only grow (we never have to remove an item set from it), because due to the report order a newly found item set cannot be a superset of an item set in the repository. For the repository one may use a bit matrix in the same way as it is used to represent the transactions: Each row 4 corresponds to an item, each column to a found (closed or maximal) frequent item set. The superset test consists in intersecting those rows of this matrix that correspond to the items in the frequent item set in question. If the result is empty, there is no superset in the repository, otherwise there is (at least) one. To include the information about the support for closed item sets, an additional row of the matrix is constructed, which contains set bits in those columns that correspond to item sets having the same support as the one in question.

With this additional row the intersection process is started. It should be noted that the superset test can be avoided if any direct descendant (intersection product) of an item set has the same support (closed item sets) or is frequent (maximal item set). In our implementation the repository bit matrix uses the same representation as the matrix that represents the transactions. That is, either both are true bit matrices or both are sparse bit matrices. [26]

IV. EXPERIMENTAL RESULTS

In this section, we describe the experimental results of using dEclat algorithm for generating frequent patterns. We performed experiments on benchmark mushroom dataset, obtained from UCI Repository of Machine Learning Databases [27]. Typically, these real datasets are very dense, i.e., they produce many long frequent itemsets even for very high values of support.

A. *Dataset Description*

Title : Mushroom Database
No of instances : 8124
No of attributes : 22 (all nominally valued)

Attribute information: (classes: edible=e, poisonous=p) cap_shape, cap-surface, cap color, bruishes, odor, gill-attachment, gill-spacing, gill-size, gill-color, stalk-shape, stalk, root, stalk_surface_along ring, stalk color_above_ring, stalk_color_ below ring, veil type , veil color, ring_number, ring_type spore_print_colour, population, habitat.

B. Class Distribution

Edible : 4208(51.8%)
Poisonous: 3916(48.2%)

All experiments were tested on an 800 MHz, 256MB memory, Pentium IV processor running on Win98. Algorithms were coded in JAVA. Furthermore, the times for all the vertical methods include all costs, including the conversion of the original database from a horizontal to a vertical format required for the vertical algorithms. The algorithm was tested for its efficiency in finding the frequent pattern. We tested our program for different values of minimum support and plotted a graph as shown in figure 7 and figure 8.

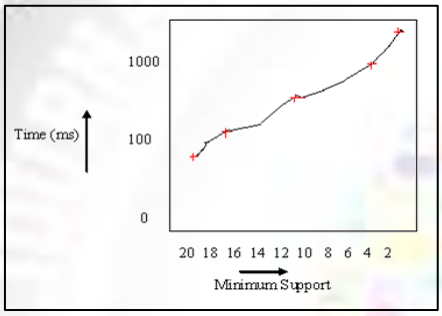

Fig. 7: Frequent pattern Mining

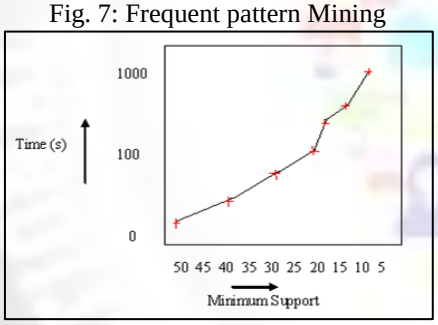

Fig. 8: Execution Time

V. CONCLUSION

The overall performance of mining association rule is determined by finding all frequent itemsets. From the experiment which we have conducted we would able to find frequent itemsets from the mushroom dataset depending upon support and confidence values which were the input given by the user. The dEclat algorithm implementation uses (sparse) bit matrices to represent transactions lists and to filter item sets. And we have shown the performance of the algorithm in the graph. We can also use this extension and compare it with other ARM algorithms to shown its increasing performance.

REFERENCES

[1] R. Agrawal, et al., "Fast discovery of association rules", In U. Fayyad and et al (eds.), Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996.

[2] Ramesh Agrawal, Charu Aggarwal, and V.V.V. Prasad, "Depth First Generation of Long Patterns", In 7th Int'l Conference on Knowledge Discovery and Data Mining, August 2000.

[3] Jay Ayres, J. E. Gehrke, Tomi Yiu, and Jason Flannick, "Sequential pattern mining using bitmaps", In SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, July 2002.

[4] R. J. Bayardo, "Efficiently mining long patterns from databases", In ACM SIGMOD Conf. Management of Data, June 1998.

[5] M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets", Technical Report 01-1, Rensselaer Polytechnic Institute, USA, 2001.

[6] D. Burdick, M. Calimlim, and J. Gehrke, "MAFIA: a maximal frequent itemset algorithm for transactional databases", In Intl. Conf. on Data Engineering, April 2001.

[7] B. Dunkel and N. Soparkar, "Data organization and access for efficient data mining", In 15th IEEE Intl. Conf. on Data Engineering, March 1999.

[8] K. Gouda and M. J. Zaki, "Efficiently mining maximal frequent itemsets", In 1st IEEE Int'l Conf. on Data Mining, November 2001.

[9] D. Gunopulos, H. Mannila, and S. Saluja, "Discovering all the most specific sentences by randomized algorithms", In Intl. Conf. on Database Theory, January 97.

[10] J. Han and M. Kamber, "Data Mining: Concepts and Techniques", Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[11] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation", In ACM SIGMOD Conf. Management of Data, May 2000.

[12] D-I. Lin and Z. M. Kedem, "Pincer-search: A new algorithm for discovering the maximum frequent set", In 6th Intl. Conf. Extending Database Technology, March 98.

[13] J-L. Lin and M. H. Dunham, "Mining association rules: Anti-skew algorithms", In 14th Intl. Conf. on Data Engineering, February 1998.

[14] J. S. Park, M. Chen, and P. S. Yu, "An effective hash based algorithm for mining association rules", In Intl. Conf. Management of Data, May 1995.

[15] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules", In 7th Intl. Conf. on Database Theory, January 1999.

[16] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. In SIGMOD Int'l Workshop on Data Mining and Knowledge Discovery, May 2000.

[17] S. Sarawagi, S. Thomas, and R. Agrawal, "Integrating association rule mining with databases: alternatives and implications", In ACM Intl. Conf. Management of Data, June 1998.

[18] A. Savasere, E. Omiecinski, and S. Navathe, "An efficient algorithm for mining association rules in large databases", In 21st VLDB Conf., 1995.

[19] J. Shafer, R. Agrawal, and M. Mehta "Sprint: A scalable parallel classifier for data mining", In 22nd VLDB Conference, March 1996.
[20] P. Shenoy, et al., "Turbo-charging vertical mining of large databases", In Intl. Conf. Management of Data, May 2000.
[21] M. J. Zaki, "Generating non- redundant association rules", In Int'l Conf. Knowledge Discovery and Data Mining, August 2000.
[22] M. J. Zaki, "Scalable algorithms for association mining", IEEE Transactions on Knowledge and Data Engineering, 12(3):372-390, May-June 2000.
[23] M. J. Zaki and C.-J. Hsiao, "Charm: An efficient algorithm for closed itemset mining", In 2nd SIAM Int'l Conf. on Data Mining, April 2002.
[24] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules", In 3rd Intl. Conf. on Knowledge Discovery and Data Mining, August 1997.
[25] Mohammed J. Zaki, Karam Gouda, "Novel vertical mining using diffsets", Conference on Knowledge Discovery in Data Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining Pages: 326 - 335 , 2003 .
[26] C.Borgelt, "Efficient implementations of Apriori and Eclat", In FIMI'03: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, November 2003.
[27] C.L. Blake and C.J. Merz. UCI Repository of Machine Learning Databases. Dept. of Information and Computer Science, University of California at Irvine, CA, USA 1998
http://www.ics.uci.edu/mlearn/MLRepository.html