Efficient Analytical and Implementation work on Frequent Pattern Mining Algorithms

Prof. Paresh Tanna[#1], Dr. Yogesh Ghodasara[*2]

[#1]*School of Engineering – MCA Department, RK. University, Rajkot, Gujarat, India*

[*2]*College of Information Tech., Anand Agriculture University, Anand, Gujarat, India*

1paresh.rkcet@gmail.com

2yrghodasara77@yahoo.co.uk

***Abstract*—Apriori, DHP and ECLAT are the best-known basic algorithms for mining frequent patterns in a dataset. Here we describe implementations of these three algorithms that use several optimizations to achieve maximum performance with efficiency. The Apriori implementation is based on a k-1 itemset prefix and uses a recursive scheme to count the transactions. The DHP uses the hashing technique to reduce number of candidate set and dataset size in each step. The ECLAT implementation uses bit matrices to represent transactions lists.**

***Keywords*— Apriori, DHP, ECLAT, frequent pattern mining, association rule.**

I. Introduction

Finding frequent itemsets in a set of transactions is a popular method for so called market basket analysis, which aims at finding regularities in the shopping behaviour of customers of supermarkets, mail order companies, online shops etc. In particular, it is tried to identify sets of products that are frequently bought together[6].

The main problem of finding frequent itemsets, i.e., itemsets that are contained in a user specified minimum number of transactions, is that there are so many possible sets, which renders approaches infeasible due to their unacceptable execution time[4,6]. Among the more sophisticated approaches three algorithms known under the names of Apriori[1], DHP[2] and ECLAT[3] are most popular. All rely on a top down search in the subset lattice of the items. An example of such a subset lattice for five items is shown in Table 1, 2, 3 and 4.

To structure the search, all three algorithms organize the subset lattice as a prefix tree, which for five items is shown in Table 1, 2, 3 and 4. In this, those itemsets are combined in a set which have the same prefix w.r.t. to some arbitrary, but fixed order of the items (in the five items example, this order is simply a, b, c, d, e). With this structure, the itemsets contained in a node of the tree can be constructed easily in the following way: Take all the items with which the edges leading to the node are labelled (this is the common prefix) and add an item that succeeds, in the fixed order of the items, the last edge label on the path. Note that in this way we need only one item to distinguish between the itemsets represented in one node, which is relevant for the implementation of three algorithms[6].

The main differences between Apriori, DHP and ECLAT are how they traverse this prefix tree and how they determine the *support* of an itemset, i.e., the number of transactions the itemset is contained in. Apriori traverses the prefix tree in breadth first order, that is, it first checks itemsets of size 1, then itemsets of size 2 and so on. Apriori determines the support of itemsets either by checking for each candidate itemset which transactions it is contained in, or by traversing for a transaction all subsets of the currently processed size and incrementing the corresponding itemset counters. The latter approach is usually preferable[1].

Apriori is a classic algorithm for frequent itemset mining and association rule learning over transactional databases[5]. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger itemsets as long as those itemsets appear sufficiently often in the database. The frequent itemsets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis[1,5].

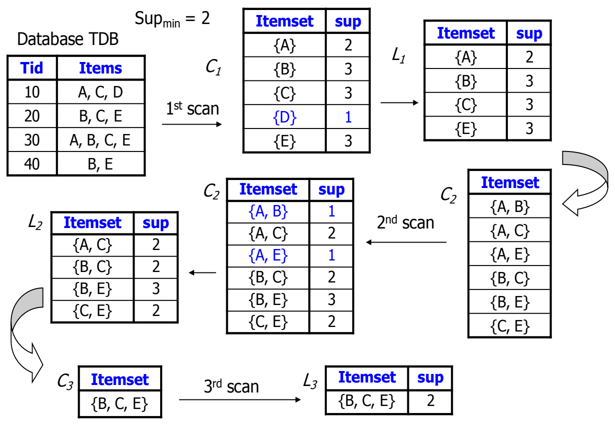
DHP can be used for efficient large itemset generation. It has two major features: efficient generation for large itemsets and effective reduction on transaction database. It uses hashing technique. In particular, for the large 2-itemsets, where the number of candidate large itemsets generated by DHP is, in orders of magnitude, smaller than that of by Apriori method[2]. Thus improving the performance bottleneck of the whole process. It uses pruning technique to reduce the size of the database progressively [2].

ECLAT, on the other hand, traverses the prefix tree in depth first order. That is, it extends an itemset prefix until it reaches the boundary between frequent and infrequent itemsets and then backtracks to work on the next prefix (in lexicographic order w.r.t. the fixed order of the items)[3,6]. ECLAT determines the support of an itemset by constructing the list of identifiers of transactions that contain the itemset. It does so by intersecting two lists of transaction identifiers of two itemsets that differ only by one item and together form the itemset currently processed[3].

II. APRIORI IMPLEMENTATION

Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation). Each transaction is seen as a set of items (an itemset). Given a threshold C, the Apriori algorithm identifies the itemsets which are subsets of at least C transactions in the database. Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found. Apriori uses breadth-first search and a Hash tree structure to count candidate itemsets efficiently. It generates candidate itemsets of length k from itemsets of length k-1. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k-length itemsets. After that, it scans the transaction database to determine frequent itemsets among the candidates[1,6].

Table :1 : The Apriori Algorithm—An Example



Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

Itemset
{B, C, E}

Itemset	sup
{B, C, E}	2

The Apriori Algorithm[1]

C_k: Candidate itemset of size k
L_k : frequent large itemset of size k

```
$L_1$ = {find frequent large 1-itemsets}
for( $k=2;\ L_{k-1} \neq \phi;\ k{+}{+}$ ){
        $C_k$ = apriori-gen ($L_{k-1}$);
        for each transaction $t \in D$ {
                // Scan D for counts
                $C_t$ = subset($C_k$, t)
// get the subsets of t that are candidate
                forall candidates $c \in C_t$ do
                        c.count++;
        }
        $L_k = \{ c \in C_k | c.count \geq minsup \}$
}
return $L = \cup_k L_k$;
```

Candidate Generation : Join Step

```
insert into $C_k$
select $p.item_1, p.item_2, p.item_{k-1}, q.item_{k-1}$
from $L_{k-1}\, p, L_{k-1}\, q$
where $p.item_1 = q.item_1, \ldots, p.item_{k-2}$
$= q.item_{k-2}$, $p.item_{k-1} < q.item_{k-1}$
```

Candidate Generation : Prune Step

```
forall itemsets $c \in C_k$ do
        forall (k-1)-subsets s of c do
                if ($s \notin L_{k-1}$) then
                        delete c from $C_k$
```

Considering an example for joining and pruning : Let L_3 = { {1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4} } After joining : { {1 2 3 4}, {1 3 4 5} } and After pruning : {1 2 3 4} since {1 4 5} and {3 4 5} are not in L_3.

Also Apriori algorithm can be modified to improve its efficiency (computational complexity) by hashing, removal of transactions that do not contain frequent itemsets, sampling of the data, partitioning of the data, and mining frequent itemsets without generation of candidate itemsets[6].

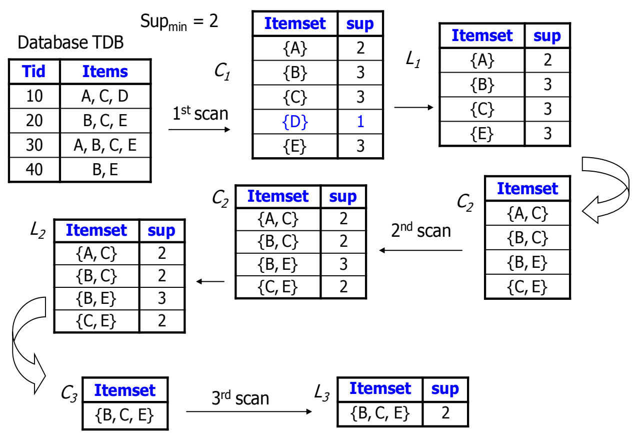
III. DHP IMPLEMENTATION

These algorithms generate candidate k+1- itemsets from large k-itemsets by counting the occurrence of candidate k+1-itemsets in the dataset. DHP utilizes a hashing technique to filter the unnecessary itemsets to generate next candidate itemsets[2,6].

The set of large k-itemsets, L_k, is used to generate a set of candidate k+1-itemsets, C_{k+1}, by joining L_k with itself on k-1 denoted by, $L_k * L_k$, to find the common items for next pass. Increasing number of items in the C_{k+1} will increase the processing cost of finding the L_{k+1}. Scanning all database transactions and testing each transaction to determine L_k from C_k is very expensive process[2].

DHP algorithm constructs smaller size C_k than Apriori algorithm. Therefore it is faster in counting C_k from database to determine L_k. The size of L_k decreases rapidly as k increases. A smaller L_k will lead to smaller C_{k+1}, so lower corresponding processing cost. DHP reduces the corresponding processing cost of determining L_k from C_k by reducing the number of itemsets to be explored in C_k in initial iteration significantly. DHP algorithm has two major features; making efficient generation of large itemsets and reducing transaction database size in effective way[2]

Table :2 : The DHP Algorithm - Example



$Sup_{min} = 2$

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

C_1 (1st scan)

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, C}
{B, C}
{B, E}
{C, E}

C_2 (2nd scan)

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

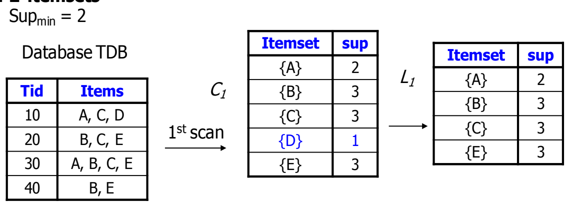
Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

Part : 1 : Gets a set of large 1-itemsets and makes a hash table (i.e. H_2) for 2-itemsets



$Sup_{min} = 2$

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

C_1 (1st scan)

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

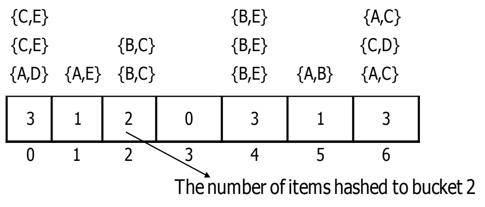
Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

Making Hash Table $Sup_{min} = 2$

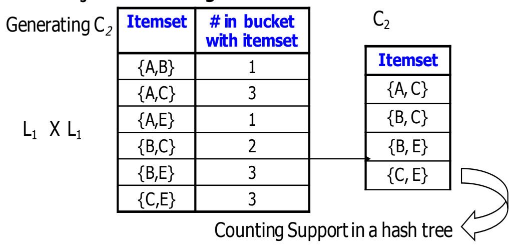
Tid	K-Itemsets for Tid (K=2)
10	{A,C}, {A,D}, {C,D}
20	{B,C}, {B,E}, {C,E}
30	{A,B}, {A,C}, {A,E}, {B,C}, {B,E}, {C,E}
40	{B,E}

H{{x,y}}=((order of x)*10 + (order of y)) mod 7



{C,E}				{B,E}		{A,C}
{C,E}		{B,C}		{B,E}		{C,D}
{A,D}	{A,E}	{B,C}		{B,E}	{A,B}	{A,C}
3	1	2	0	3	1	3
0	1	2	3	4	5	6

Part : 2 : Generates the set of candidate itemsets C_2 based on the hash table (H_2), determines the set of large 2-itemsets L_2 . Also reduces the size of database for the next large itemsets & makes h_3 for next C_3 candidate large itemsets

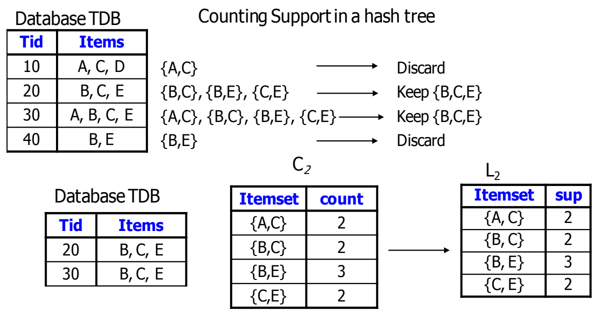


Generating C_2 (L_1 X L_1)

Itemset	# in bucket with itemset
{A,B}	1
{A,C}	3
{A,E}	1
{B,C}	2
{B,E}	3
{C,E}	3

C_2

Itemset
{A, C}
{B, C}
{B, E}
{C, E}



Database TDB — Counting Support in a hash tree

Tid	Items		
10	A, C, D	{A,C}	Discard
20	B, C, E	{B,C}, {B,E}, {C,E}	Keep {B,C,E}
30	A, B, C, E	{A,C}, {B,C}, {B,E}, {C,E}	Keep {B,C,E}
40	B, E	{B,E}	Discard

Database TDB

Tid	Items
20	B, C, E
30	B, C, E

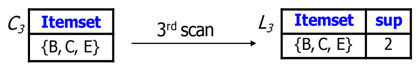
C_2

Itemset	count
{A,C}	2
{B,C}	2
{B,E}	3
{C,E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

Part : 3 : Further process same as Apriori method. But it provides database reduction in each pass .



C_3

Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

In DHP algorithm, we find support count of C_k by scanning the database. The algorithm also accumulates information about candidate k+1-itemsets. That means all possible k+1 subset of items of each transaction after pruning item are less than min_support from hash table. Each entry in hash table consists of number of items that have been hashed to this entry. Thus far this table will be used to determine C_{k+1}- itemsets from L_k as Apriori algorithm. Each bucket in the hash table consists of number to present how many itemset have been hashed to this bucket. A bit vector can be constructed. If the number of corresponding entry of the hash table is greater than or equal to s, set the value of a bit vector to one. The hash function is a black box which produces an address every time you drop in a key. H(k) transforms key, k, into the correct address, that is used to store and retrieve set of records[2].

Table 2 shows a representation of the data structure to provide a direct access to the returned value of a hash function. For many cases, the address generated by the hash function is a random value and depends in the architecture of the table. A collision is occurred when two different keys are transformed to the same address. In fact, it is impossible to hold two records in the same space address[2].

Hash Table Construction

Consider two items sets, all items are numbered as I1, I2, …In. For any pair (x, y), has according to Hash function bucket # = h({x y}) = ((order of x)*10+(order of y)) % 7. Example: Items = A, B, C, D, E, Order = 1, 2, 3 4, 5, then H({C, E})= (3*10 + 5)% 7 = 0. Thus, {C, E} belong to bucket 0.

How to trim candidate itemsets

In k-iteration, hash all "appearing" k+1 itemsets in a hashtable, count all the occurrences of an itemset in the correspondent bucket. In k+1 iteration, examine each of the candidate itemset to see if its correspondent bucket value is above the support (necessary condition)[2].

Examples for Trim and Reduction in transaction

(I) In transaction 10 (A, C, D) , a single candidate AC is found in C2. Occurrence frequencies of all the items are : a[0] = 1, a[1] = 1, a[2] = 0. Since all the values of a[i] are less than k (k=2), this transaction is deemed not useful for generating large 3-itemsets and thus discarded. (II) In transaction 30 (A, B, C, E), has four candidate 2-items (AC, BC, BE, CE) found in C2. Occurrence frequencies of all the items are : a[0] = 1, a[1] = 2, a[2] = 2, a[3] = 2. Since all the values of a[0] are less than k (k=2), and remaining are >=2, this transaction will be reduced to (B, C, E) and A is thus discarded.

Effective Database Pruning

Apriori - don't prune database but prune C_k by support counting on the original database, while DHP -Its m*ore efficient support counting can be achieved on pruned database[2,4]*

IV. ECLAT Implementation

ECLAT implementation represents the set of transactions as a (sparse) bit matrix and intersects rows to determine the support of itemsets. The search follows a depth first traversal of a prefix tree as it is shown in Table 3 and 4.

A convenient way to represent the transactions for the ECLAT 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[3,6].

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. (Obviously the latter representation is equivalent to using a list of transaction identifiers for each item.). ECLAT searches a prefix tree like the one shown in Table 3 and 4 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.[3] The item corresponding to the row that is intersected with the following rows thus is added to form the common prefix of the itemsets processed in the corresponding child node. Of course, rows corresponding to infrequent itemsets 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[3,6].

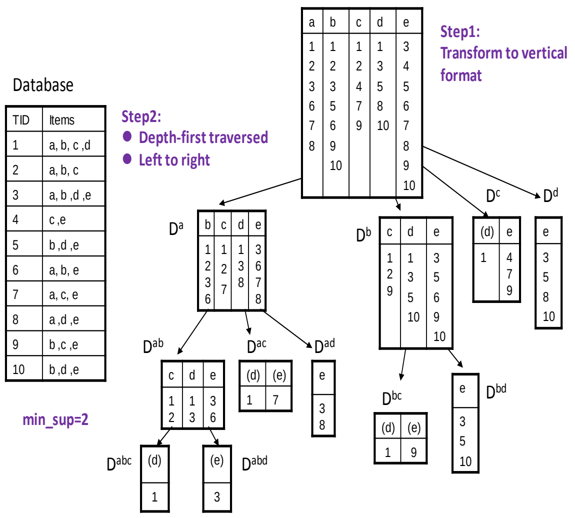
Intersecting two rows can be done by a simple logical *and* on a fixed length integer vector if we work with a true bit matrix. During this intersection the number of set bits in the intersection is determined by looking up the number of set bits for given word values (i.e., 2 bytes, 16 bits) in a precomputed table. For a sparse representation the column indices for the set bits should be sorted ascending order for efficient processing. Then the intersection procedure is similar to the merge step of merge sort.

As for Apriori the way in which items are coded has an impact on the execution time of the ECLAT algorithm[4]. The reason is that the item coding not only affects the number and the size of gaps in the counter vectors for Apriori, but also the structure of the *pruned* prefix tree and thus the structure of ECLAT's search tree. Sorting the items usually leads to a better structure. For the sorting there are basically the same options as for Apriori.

The ECLAT Algorithm[3]

```
Input: $D, \sigma, I \subseteq T$
Output: $F[I](D, \sigma)$
------------------------------
$F[I] := \{\}$
for all $i \in T$ occurring in D do
    $F[I] := F[I] \cup \{I \cup \{i\}\}$
    // Create $D^i$
    $D^i := \{\}$
    for all $j \in T$ occurring in D such that $j > i$ do
        $C := cover(\{i\}) \cap cover(\{j\})$
        if $|C| \geq \sigma$ then
            $D^i := D^i \cup \{(j, C)\}$
        end if
    end for
    // Depth-first recursion
    Compute $F[I \cup \{i\}](D^i, \sigma)$
    $F[I] := F[I] \cup F[I \cup \{i\}]$
end for
```

Table :3 : The ECLAT Algorithm - Example



TID	Items
1	a, b, c ,d
2	a, b, c
3	a, b ,d ,e
4	c ,e
5	b ,d ,e
6	a, b, e
7	a, c, e
8	a ,d ,e
9	b ,c ,e
10	b ,d ,e

a	b	c	d	e
1 2 3 6 7 8	1 2 3 5 6 9 10	1 2 4 7 9	1 3 5 8 10	3 4 5 6 7 8 9 10

D^a

b	c	d	e
1 2 3 6	1 2 7	1 3 8	3 6 7 8

D^b

c	d	e
1 2 9	1 3 5 10	3 5 6 9 10

D^c

(d)	e
1	4 7 9

D^d

e
3 5 8 10

D^{ab}

c	d	e
1 2	1 3	3 6

D^{ac}

(d)	(e)
1	7

D^{ad}

e
3 8

D^{bc}

(d)	(e)
1	9

D^{bd}

e
3 5 10

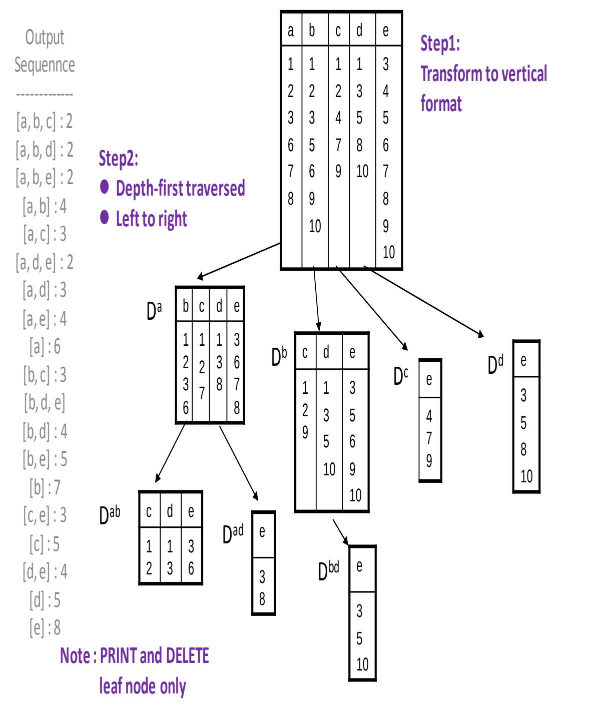
D^{abc}

(d)
1

D^{abd}

(e)
3

Table :4 : The ECLAT Algorithm – Example with Final Result



V. Experimental Results

We ran experiments with three programs on a dataset, which so that the advantages and disadvantages of the three approaches and the different optimizations can be observed. The data sets we used is: T10I4D10K (an artificial data set generated with IBM's data generator [7]). The results for these dataset is shown in Figure 1. We can easily understood from the figure 1 i.e. Apriori is much slower than DHP. Also ECLAT takes only around 1/10 time of taken by Apriori. ECLAT also gives result in much higher speed than Apriori and DHP. Between Apriori and DHP, DHP is still somewhat speedy than Apriori.

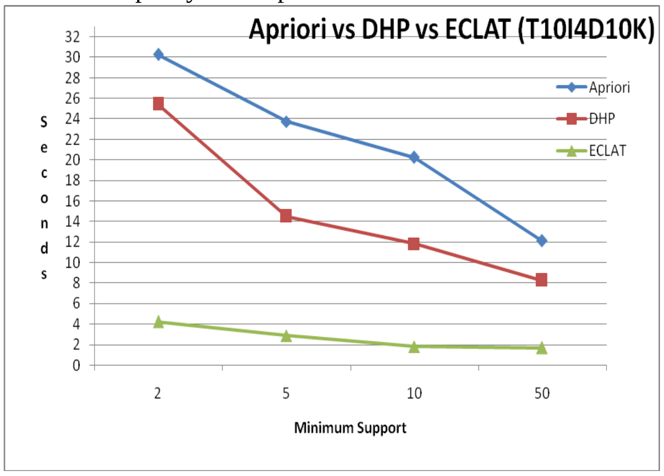


Fig. 1 Apriori Vs DHP Vs ECLAT (T10I4D10K)

VI. Conclusions

Apriori is best for frequent pattern mining approach for newer algorithm development. But after implementation you can find some challenges like multiple scans of transaction database, huge number of candidates, tedious workload of support counting for candidates and we can improve Apriori with effective hash-based algorithm for the candidate itemset generation i.e. a two phase transaction database pruning and much more efficient (time & space) than Apriori algorithm. DHP is different only in step of 2-itemset generation than Apriori. After that 2+ -itemsets, DHP takes lots time to traverse to each transaction for support count and candidate set generation. While ECLAT outperforms with all compare to Apriori and DHP. ECLAT is best and most suitable for larger datasets. The only problem with ECLAT is that it consumes lots more memory during execution by storing bit matrix for each transaction .

References

[1] R. Agrawal and S. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases", Proceedings of the 20th International Conference on Very Large Data Bases, September 1994.

[2] J. Park, M. Chen and Philip Yu, "An Effective Hash-Based Algorithm for Mining Association Rules", Proceedings of ACM Special Interest Group of Management of Data, ACM SIGMOD'95, 1995.

[3] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, " New Algorithms for Fast Discovery of Association Rules", Proc. 3rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'97, Newport Beach, CA), 283-296 AAAI Press, Menlo Park, CA, USA 1997

[4] Shruti Aggarwal, Ranveer Kaur, "Comparative Study of Various Improved Versions of Apriori Algorithm", International Journal of Engineering Trends and Technology (IJETT) - Volume4Issue4- April 2013

[5] Agrawal, R., T. Imielin´ ski, and A. Swami (1993). Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, SIGMOD '93, New York, NY, USA, pp. 207–216. ACM.

[6] Data Mining: Concepts and Techniques, Jiawei Han and Micheline Kamber, MORGAN KAUFMANN PUBLISHER, An Imprint of Elsevier

[7] Synthetic Data Generation Code for Associations and Sequential Patterns. http://fimi.cs.helsinki.fi