Analysis of Complexities for finding efficient Association Rule Mining Algorithms

R.Rathinasabapathy, R.Bhaskaran
Madurai Kamraj University, Madurai
Email: ramalingam.rathinasabapathy@gmail.com

Abstract: Several algorithms for association rule mining, have been implemented including a variation of Apriori, an algorithm using hash functions for finding large 2-itemsets and 3-itemsets and direct search method for finding other large k-itemsets, and another variation of Eclat algorithm using perfect hash functions for 2-itemsets and 3-itemsets and the method of vertical mining for finding other large k-itemsets. All these algorithms were compared, among themselves by finding time complexities of each algorithm, for finding factors affecting the efficiency of the algorithms. By observing the complexities we are able to (i) find factors that is influencing execution times (ii) find why one algorithm works faster than the other and (iii) find convincing method of choosing best algorithm with the given constraints such as database composition, maximum frequent set size and itemset size factors.

Keyword: Clustering, classification, and association rules, Data mining, Analysis of algorithms, Graph theory.

1. INTRODUCTION

Data mining is a widely researched area wherein we extract information, also known as knowledge, from large amount of data which is accumulated over several years, even decades. We have to preprocess the data, structure the data, analyze the data using various statistical methods, use rigorous algorithms over the data and these steps may have to be repeated several times, that too not in the order of their appearance. Obviously, the task is not easy. The knowledge is hidden and it is viewable only for those who have the expertise of extracting it. There are various methods of extracting the knowledge. We are utilizing several techniques and algorithms[2][15] such as classification, clustering, regression, association rules mining, neural networks, decision trees, genetic algorithms, support vector machines, fuzzy logic and so on for extracting useful and interesting patterns from the data. The data may have been stored in structured arrays, structured files, relational databases, image databases, text files, hyper text and multimedia databases. Depending on the type of data, type of database and kinds of knowledge extracted we have to use various methods and techniques. Obviously, we have to use various algorithms to extract the desired knowledge.

2 ASSOCIATION RULE MINING ALGORITHMS

Association rule mining [3][5][12][13][14][17][20][22][26][27][28][31][33][37][40] is the process of extracting patterns of sets of elements, or items occurring with specified probability in a set of transactions. The association rule mining is introduced in [27]. Another algorithm was proposed in [26] for association rule mining known as apriori algorithm which represents the database differently by storing attributes themselves in each transaction. Apart from the measures support and confidence, several other measures[39] may also be used and it may not only reduce search space of the rules but also include measures needed by variety of knowledge to be extracted. The problem of finding association rules can be divided into the following. (i) Finding the frequent itemsets having the minimum support and they are called as large sets (ii) From the frequent item-sets, we can easily form association rules[26].

Now, there are lot many algorithms already proposed and implemented for association rule mining. The algorithms are broadly classified as horizontal data mining algorithms[3][26][27], vertical data mining algorithms[22][23][25] and algorithms using tree structures[29](such as fp-growth tree)[13][14] depending on how we are representing the elements of the database. The horizontal data mining algorithms are keeping the data base as a set of transaction identifier versus set of transactions. In vertical data mining, we keep the database as itemwise. That is, we are representing the database as items versus transaction identifier lists that are having the items. In frequent pattern growth tree it is represented in tree structures.

3 HASH BASED ALGORITHMS

3.1 Introduction

In this paper, several algorithms have been implemented utilizing perfect hash functions for finding large 2-itemsets and 3-itemsets that improves the efficiency of association rule mining algorithms considerably. Suppose we have a set S= {i1, i2, i3, … in} be set of m elements. Let C2 be set of 2-itemsets, formed using two any elements of S and C3 be set of 3-itemsets formed using any three elements of S. Then there will be m*(m-1)/2 elements in the set C2 and m(m-1)(m-2)/6 elements in the set C3. Then, if the elements of C2 and C3 are unordered and to search for an element in those sets we have to search the whole set of elements till we reach the element. That is we will have to search at most all the m(m-1)/2 elements for C2 and at most all the m(m-1)(m-2)/6 elements for C3. If it is ordered, then there will be a binary search of at most log2(m(m-1)/2) times for C2.
and \( \log_2(m(m-1)(m-2)/6) \) times for \( C_3 \). But if we use perfect hash functions, it will be just one search\( O(1) \). We provide perfect hash functions on 2-itemsets and 3-itemsets and use this function in several algorithms to make them efficient.

### 3.2 Hash functions for finding large 2-itemset and 3-itemset

The hash functions for 2-itemset and 3-itemset have been provided in [32]. Define the perfect hash function \( H_2 \) such that,

\[
H_2(e_i, e_j) = (i-1)*(2*r-i)/2 + (j-i),
\]

where \( e_i \) and \( e_j \) are elements of frequent 1-itemsets and \( r \) is the total number of frequent 1-itemsets.

Define the perfect hash function \( H_3 \) such that

\[
H_3(e_i, e_j, e_k) = \left[ (i-1)*r*(r-1)/2 + (j-1)*(2*r-j)/2 + (k-1)*(3*r-k)/2 + (i)*(i+1)*(i+2-3*r)/6 + (j)*(j+1)*(j+2-3*r)/6 + (k-1)*(k+1)*(k+2-3*r)/6 + \right]
\]

where \( e_i, e_j \) and \( e_k \) are elements of frequent 1-itemsets and \( r \) is the total number of frequent 1-itemsets. We can directly find 1-itemset frequent set from the database. Using above hash functions, 2-itemsets and 3-itemsets can be mapped into consecutive integers and that can be utilized in finding large 2-itemsets and 3-itemsets.

### 3.3 Algorithms for association rule mining

The disadvantage of the apriori algorithm is that if number of elements in the set of candidate item-sets is large, it may degrade the efficiency of the apriori algorithm. Some researchers have indicated that the candidate item-sets generated in the initial iterations dominate the cost of data mining[18]. There have been many algorithms proposed and implemented to overcome the problem of large number of candidate set generated. These algorithms provide solutions to overcome the problem of oversize candidate sets. They tried to reduce the number of generation of candidate sets. DHP algorithm uses hash function to reduce the number of candidates to be searched for finding large item-sets. The search for existence of candidate set can be done away with using perfect hash functions. There are algorithms using perfect hash functions such as PHS algorithm and MPIP algorithm. The algorithm[5] utilizes apriori method, perfect hash function and encoding scheme. It proposes a perfect hashing and data shrinking method to solve the association rule mining problem. The perfect hash function is used by employing encoding mechanism to fix the length of each candidate itemset at two which allows the perfect hash function to be applied on itemsets of length more than two. In MPIP algorithm a hashing scheme has been given. This algorithm uses perfect hash function initially to find large itemsets. Later it shifts its method by taking up apriori with pruning. This algorithm generates frequent 2-itemsets directly from one scan over the database without generating \( C_1, L_1 \) and \( C_2 \). Perfect hash function for 3-itemsets have also been provided in [32]. So, without generating \( C_2, L_2 \) and \( C_3 \) we can find large 3-itemsets. Using these hash functions for 2-itemsets and 3-itemsets, several algorithms have been provided in this paper and not all algorithms are efficient in all circumstances. So, their complexities have been analysed to find factors affecting the efficiency of algorithms. There is another method whereby we can improve the efficiency of the apriori algorithm. Usually, for each transaction we are taking all the k-item-sets and search it over the candidate set usually using a hash tree[38]. So, for each \( \binom{m}{k} \) number of k-item-sets (m is column size of the database and k is item-set size) for each row in the database we have to search the hash tree of the candidate set. On the other way, we can find the frequency of candidate k-item-set by comparing each candidate k-item-set against the columns of the database for each row in which case we do not need to have a hash tree. But we have to do it for all elements of the candidate set. So if the size of the candidate set is large it will take much time reducing the efficiency. On the other hand, if there are many columns in the database it will increase the value of \( \binom{m}{k} \). So for less number of candidate sets the apriori algorithm of direct search will work better than apriori algorithm with hash tree implementation. This will be more evident when we discuss about time complexities of algorithms. We have incorporated this method in the algorithm AprioriDirectSearchHash. We have implemented the Eclat, an algorithm for vertical mining of association rules and AprioriHashTree an algorithm for apriori algorithm using hash tree[8] and MPIPTwoTreeHash, an algorithm using perfect hash functions for finding large 2-itemsets and 3-itemsets and using hash tree for finding other large k-itemsets. we have also provided AprioriDirectSearchHash, an algorithm using hash functions for finding large 2-itemsets and 3-itemsets and direct search method for finding other large k-itemsets, EclatHash, an algorithm using perfect hash functions for 2-itemsets and 3-itemsets and the method of vertical mining for finding other large k-itemsets, but all these algorithms were compared among themselves by finding time complexities of each algorithm.

### 4 COMPLEXITY ANALYSIS OF ALGORITHMS

#### 4.1 complexities of algorithms

The time complexities, Big oh notations and analysis of algorithms are explained in detail in [1][7][6][9][11][16][30][34][35][36]. Counting exact number of instructions required by the algorithm is not essential. So, the big oh notations ignore multiplicative constants and the details that do not impact the comparison of algorithms. The time
complexity of a program is generally some function of instance characteristics. This function is very useful in determining how time requirements vary as the instance characteristics changes. Computational complexities of frequent itemset mining algorithms have been discussed in[22] and [23]. The relationship between transactions and itemsets can be represented using bipartite graph[10][22][24] G=(U,V,E) where U is set of itemsets, V is set of transactions and E is the set of all edges (u,v) such that u∈U and v∈V. The problem of enumerating maximal frequent itemsets is the same as enumerating all maximal bipartite cliques in the bipartite graph G and the same is NP-complete[22]. Many theoretically intractable problems can be solved efficiently by taking advantage of small parameters[4][19][21]. The problems which are intractable theoretically may be having parameters, and for such problems for smaller parameters, we can solve the problem in polynomial time and they are called Fixed Parameter Tractable (FPT).

More formally, we consider a parameterized language to be a subset L ⊆Σ∗ X Σ∗. If L is a parameterized language and <σ,k> ∈ L then we refer to σ as the main part and k as the parameter. A parameterized language L is said to be Fixed Parameter Tractable (FPT) if membership in L can be determined by an algorithm whose running time on instance <σ,k> is bounded by f(k) |σ|α, where f is a polynomial function with parameter k and α is independent of both σ and the parameter k. For example, the vertex cover problem is well known to be NP-complete[10]. It leads us to conclude that the problem of mining maximal frequent itemsets is NP-hard. For sufficiently large support count, practically the size of maximal frequent set is small. It can be shown that problem of enumerating the maximal frequent itemsets for such problem as FPT. Besides, the algorithm is not only affected by the size of maximal frequent set(a small parameter), but also other functions affecting the time complexity of the algorithm by considerable amount of factor which can be found using the complexity expressions of the algorithms by the method provided in[1]. We have implemented the following algorithms (i) AprioriHashTree (ii) MPIPTwoThreeHash (iii) AprioriDirectSearchHash (iv) Eclat and (v) EclatHash for mining association rules. The experiments were done on a PC with 2GB RAM and 2.4 GHz processor running Windows Vista using C++. We used the data sets(Chess, Mushroom and Connect) available in frequent itemset mining implementations repository <http://fimi.cs.helsinki.fi>. The time complexities of the above algorithms have been arrived at and it is given below here. We have found time complexities of the algorithms and provide them as Big O notation. Using these complexities we can analyse the factors that is influencing the algorithms. The complexities are shown below.

| | \[Row\]| and tidSize | Number of rows in the database |
| | Column | Number of columns in the database |
| | ItemSize | Item size |
| | L-freq(k) | Number of frequent k-itemsets |
| | L-freq(1) | Number of k-frequent sets to the power r. |
| | C-join(k) | Number of k-itemset candidate sets obtained by way of joining |
| | C-prun(k) | Number of candidates after pruning |

Table 4.2 Symbols used in complexity expression.

### Complexity for AprioriHashTree

\[ O(|row|*|Column|) + O(|itmSze|) + O(|Lfreq(1)|) + \sum_{k=1}^{k} \{ O(|Lfreq(k-1)|^2*k) + O(|C_join(k)| * |Lfreq(k-1)| ^k*k) + O(|C_prun(k)|) + O(|row|*|Column|) \}, \]

for k=2,3,4,… Until last k non-empty frequent set.

### Complexity for MPIPTwoThreeHash

\[ O(|row|*|Column|) + O(|itmSze|) + O(|Lfreq(1)|) + O(|Lfreq(1)|^2) + O(|row| * ((|Column|)^2)) + O(|Lfreq(1)|^1) + O(|row| * ((|Column|)^3)) + \sum_{k=1}^{k} \{ O(|Lfreq(k-1)|^2*k) + O(|C_join(k)| * |Lfreq(k-1)| ^k*k) + O(|C_prun(k)|) + O(|C_join(k)|) + O(|row|*|Column|)^2 \}, \]

for k=4,5,6,… until last k having nonempty frequent set.

### Complexity for AprioriDirectSearchHash.

\[ O(|row|*|Column|) + O(|itmSze|) + O(|Lfreq(1)|) + O(|row| * ((|Column|)^2)) + O(|Lfreq(1)|^2) + O(|row| * ((|Column|)^3)) + \sum_{k=1}^{k} \{ O(|Lfreq(k-1)|^2*k) + O(|C_join(k)| * |Lfreq(k-1)| ^k*k) + O(|row|*|Column|*|C_join(k)|) + O(|Lfreq(k)|*k) + O(|Lfreq(k)|) + O(|C_join(k)|) \}, \]

Where k = 4,5,6,… Until last k having nonempty frequent set.

### Complexity for Eclat

\[ O(|row|*|Column|) + O(|itmSze*|tidSze|) + O(|Lfreq(1)|*|tidSze|) + \sum_{k=1}^{k} \{ O((|Lfreq(k-1)|)^2*C_2 * (k+|tidSze|) ) + O(|C_join(k)|) * (k + |tidSze|) ) + O(|Lfreq(k)|) * (k + |tidSze|) ) \}

Until last k having non empty frequent set.

### Complexity for EclatHash

\[ O(|row|*|Column|) + O(|itmSze|) + O(|Lfreq(1)|) + O(|Lfreq(1)|^3) + O(|row| * ((|Column|)^2)) + O(|Lfreq(1)|^2) + O(|row| * ((|Column|)^3)) \]

+ O(|Lfreq(3)|*(3+tidSize))
+ \sum_{k=4}^{n} \{ O(|Lfreq(k-1)|C_2 *(k+tidSize) ) 
+ O(|C_{join}(k)|*(k + tidSize) )
+ O(|Lfreq(k)|*(k + tidSize) )

Until last k having non empty frequent set.

4.2 Analysis of algorithms

We have already shown the time complexities for AprioriHashTree, MPIPTwoThreeHash, AprioriDirectSearchHash, Eclat and EclatHash. These are represented using Big O notation. Using this we can understand why an algorithm runs faster than the other one and what are all the factors affecting the algorithms.

**AprioriHashTree vs AprioriDirectSearchHash**

We can observe from the time complexities of AprioriHashTree and AprioriDirectSearchHash the efficiency of the algorithms depends on the 1-frequent itemsets, 2-frequent itemsets, number of candidate sets, number of candidates after pruning, rows and columns of the database. The difference in the complexity does not provide exact execution time difference. But it can be used to analyse the algorithm. The factors affecting the algorithms are the parameters with regard to itemset factors(such as number of candidate sets, number of the itemsets remaining after pruning and number of frequent sets) and number of rows and columns. As far as AprioriDirectSearchHash algorithm is concerned if the parameters relating to itemsets factors are high it is very much affecting the execution time as the factors getting less in number it drastically comes down than that of AprioriHashTree. It is because if the itemset factors reduces to a minimum and from the time complexities, we can observe that for less number of frequent sets and candidate sets, the time complexities will be influenced by the expressions.

(i) \Sigma k=4.. \{ O(|row|*(|column|)C_k ) \} and
(ii) \Sigma k=4.. \{ O(|row|*(|column|)C_prune( k ) ) \}

As the number of columns increase in the database, the value of first expression become very large compared to second expression. It is usually the case that as support increases and length of the frequent set size increases, the number of frequent sets and candidate sets decreases. So, as support and frequent set size increase, the algorithm AprioriHashTree takes more execution time than the algorithm AprioriDirectSearchHash.

The AprioriDirectSearchHash algorithm uses hash functions for 2-itemsets and 3-itemsets. Apart from this, we are also using a technique of direct search to find frequent sets. So, the influence of hash functions added with a technique of direct search makes the algorithm efficient. The effect of hash functions will be more evident when we discuss about MPIPTwoThreeHash algorithm.

**MPIPTwoThreeHash versus AprioriHashTree**

The MPIPTwoThreeHash uses hash functions for 2-itemsets and 3-itemsets, otherwise it is same as AprioriHashTree. By comparing the time complexities we can observe that how much the hash functions impact the efficiency of the algorithms. From the complexities we can infer that finding the large 2-itemsets and 3-itemsets using AprioriHashTree involves number of large 1-itemset, number of large 2-itemsets, number of 2-itemsets joined and pruned and the number of 3-itemsets joined and pruned apart from rows and columns. In MPIPTwoThreeHash it depends on large 1-itemsets apart from rows and columns. As number of frequent large 1-itemset and their squares and cubes put together is considerably less than number of large 1-itemset, number of large 2-itemsets, number of 2-itemsets joined and pruned and the number of 3-itemsets joined and pruned the MPIPTwoThreeHash will take less time than the AprioriHashTree.

**AprioriDirectSearchHash versus Eclat**

We can observe that the time complexities of Eclat have many factors multiplied with tidsize. So while observing the two complexities we can infer that for very large value of tidsize the Eclat algorithm will take more execution times than AprioriDirectSearchHash algorithm.

**Eclat versus EclatHash**

When we observe the complexities of Eclat and EclatHash, we can find that the complexities involved in both the algorithms relating to 4-itemset onwards are the same. So, the difference in the complexities of two algorithms is the complexities contributing for finding large 1-itemset, 2-itemset and 3-itemset. By observing both the complexity expressions for finding 1-itemset, 2-itemset and 3-itemset we can observe that EclatHash very much influenced by the size of the database. The other contributing factors are number of large 1-itemset, its square, its cube and the number of columns of the database. For less number of columns, less number of large 1-itemset, and large number of rows the major contributing factor would be tidsize. But for Eclat, from the initial step itself it is the product of number of itemset factors multiplied with the size of the database. A small reduction in the number of itemset factors will make a much impact on the execution time of the algorithm if it contains large number of rows. So, the Eclat algorithm will take more execution time than EclatHash, for large number of rows and itemset factors.

**Conclusion**

Usually, in large databases we have to search for an element often. If the element to be searched is over the sorted set of n elements, the search complexity will be O(log n). If it is not sorted, it will be O(n). So to overcome such a scenario, we are using various methods such as hash functions to reduce the search space. In this paper, we have used two hash functions, to search over 2-itemsets and 3-itemsets for association rule mining. Whether it is sorted or not, searching for an element in either 2-itemset or 3-itemset search space, using these hash
functions will take computational complexity of $O(1)$ instead of having $O(\log_2 n)$ or be used in large number of 2-itemsets and 3-itemsets, we have tremendous improvement in execution time over other methods. Using hash functions we have implemented several hybrid algorithms for association rule mining. We analyzed all these algorithms with their computational complexities. By observing the complexities we are able to (i) find factors that is influencing execution times (ii) find why one algorithm works faster than the other and (iii) find convincing method of choosing best algorithm with given constraints such as database composition, maximum frequent set size and itemset size factors. It is evident from the complexities of algorithms, that the major influencing factors are row size (tidsize), column size, maximal frequent set size and item size factors. The hash functions on 2-itemsets and 3-itemsets are reducing the time complexities of the algorithm, if the dataset is having small maximal frequent size and large number of frequent 2-itemset and 3-itemsets. By applying these hash functions on 2-itemsets and 3-itemsets, initially thereby reducing the itemsize factors and then applying the direct search on the candidate set and since the maximal frequent item size is small, AprioriDirectSearchHash algorithm will give us good results. When row size is large, it may adversely affect the vertical mining algorithms since the complexities involve multiplication of item size factors by row size. If item size factors reduced to a threshold minimum then row size will not have much impact. As far as column size is concerned, the AprioriHashTree algorithm will be very much affected because it has the complexity of $O( ((\text{Column})C_k) )$ as $k$ increases, this value will increase affecting the algorithm. This problem is mitigated by AprioriDirectSearchHash, but it depends on the item set factors produced in the pass. Once item set factors reached a threshold the AprioriDirectSearch algorithm will perform well. Maximum frequent size affects all the algorithms because it decides the number of passes the algorithm to go through. If item size factors reduced during the later part of the passes, the algorithm is not further affected by the number of passes of the algorithm. The hash functions on 2-itemsets and 3-itemsets can be used in large number of applications such as graph algorithms, clustering algorithms. These functions can be used as index for arbitrary combinations of 2-items and 3-items. In such a case it will have time complexity of $O(1)$ instead of having $O(\log n)$ or $O(n)$ depending on whether they are sorted or not. There are lot many algorithms are available for association rule mining. In all these algorithms we can use these hash functions and their complexities can be analysed. There are also algorithms for closed itemset mining, maximal itemset mining. These can also be implemented and experimented in the same way.

REFERENCES

[14] Jiawei Han and Micheline Kamber, DATA MINING: Concepts and techniques.
International workshop and 8th annual conference of the EACSL on computer science Logic.


[38] Xiaowei Yan, Shichao Zhang and Chengqi Zhang, ON DATA STRUCTURES FOR ASSOCIATION RULE DISCOVERY, Applied Artificial Intelligence, vol 21, no 2, 2007 [pages 57-79].
