Improved Frequent Pattern Mining Algorithm using Divide and Conquer Technique with Current Problem Solutions

Nirav Patel¹ Kiran Amin²
¹PG Student (I. T.) ²Head of Department
¹Dept. of Info. Technology ²Department of Computer Science
¹, ²Ganpat University, Kherva, Gujarat, India

Abstract—Frequent patterns are patterns such as item sets, subsequences or substructures that appear in a data set frequently. A Divide and Conquer method is used for finding frequent item set mining. Its core advantages are extremely simple data structure and processing scheme. Divide the original dataset in the projected database and find out the frequent pattern from the dataset. Split and Merge uses a purely horizontal transaction representation. It gives very good result for dense dataset. The researchers introduce a split and merge algorithm for frequent item set mining. There are some problems with this algorithm. We have to modify this algorithm for getting better results and then we will compare it with old one. We have suggested different methods to solve problem with current algorithm. We proposed two methods (1) Method I and (2) Method II for getting solution of problem: We have compared our algorithm with the currently worked algorithm SaM. We examine the performance of SaM and Modified SaM using real datasets. We have taken results for both dense and sparse datasets.

I. INTRODUCTION

In, few years the size of database has increased rapidly. The term data mining or knowledge discovery in database has been adopted for a field of research dealing with the automatic discovery of implicit information or knowledge within the databases. The implicit information within databases, mainly the interesting association relationships among sets of objects that lead to association rules may disclose useful patterns for decision support, financial forecast, marketing policies, even medical diagnosis and many other applications.

Frequent itemsets play an essential role in many data mining tasks that try to find interesting patterns from databases such as association rules, sequences, clusters and many more of which the mining of association rules is one of the most popular problems. The original motivation for searching association rules came from the need to analyze called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products. Association rules describe how often items are purchased together.

II. FREQUENT ITEMSET MINING

Studies of Frequent Itemset (or pattern) Mining[1,7] is acknowledged in the data mining field because of its broad applications in mining association rules, correlations, and graph pattern constraint based on frequent patterns, sequential patterns, and many other data mining tasks. Efficient algorithms for mining frequent itemsets are crucial for mining association rules as well as for many other data mining tasks. The major challenge found in frequent pattern mining is a large number of result patterns. As the minimum threshold becomes lower, an exponentially large number of itemsets are generated. Therefore, pruning unimportant patterns can done effectively in mining process and that becomes one of the main topics in frequent pattern mining. Consequently, the main aim is to optimize the process of finding patterns of which should be efficient, scalable and can detect the important of patterns which can be used in various ways.

III. RELATED WORK

A. Apriori

The most popular frequent item set mining called the Apriori algorithm was introduced by [1]. The item sets are check in the order of increasing size (breadth first/level wise traversal of the prefix tree). The canonical form of item sets and the induced prefix tree are use to ensure that each candidate item set is generated at most once. The already generated levels are used to execute Apriori [1] pruning of the candidate item sets (using the Apriori property). Apriori [1,7]: before accessing the transaction database to determine the support Transactions are represented as simple arrays of items (so-called horizontal transaction representation, see also below). The support of a candidate item set is computing by checking whether they are subsets of a transaction or by generating and finding subsets of a transaction. For more detail refer [10].

B. Eclat

Eclat [6, 9, 10] algorithm is basically a depth-first search algorithm using set intersection. It uses a vertical database layout i.e. instead of explicitly listing all transactions; each item is stored together with its cover (also called TIDList) and uses the intersection based approach to compute the support of an item set. In this way, the support of an item set X can be easily computed by simply intersecting the covers of any two subsets Y, Z ⊆ X, such that Y U Z = X. It states that, when the database is stored in the vertical layout, the support of a set can counted much easier by simply intersecting the covers of two of its subsets that together give the set itself. It essentially generates the candidate itemsets using only the join step from Apriori [1]. Again all the items in the database is reordered in ascending order of support to reduce the number of candidate itemsets that is generated, and
hence, reduce the number of intersections that need to be computed and the total size of the covers of all generated itemsets. Since the algorithm does not fully exploit the monotonicity property, but generates a candidate itemset based on only two of its subsets, the number of candidate itemsets that are generate is much larger as compared to a breadth-first approach such as Apriori. As a comparison, Eclat essentially generates candidate itemsets using only the join step from Apriori [4], since the itemsets necessary for the prune step are not available.

C. SaM

The Split and Merge algorithm [3,8] is a simplification of the already fairly simple RElim (Recursive Elimination) algorithm[2]. While RElim represents a (conditional) database by storing one transaction list for each item (partially vertical representation), the split and merge algorithm employs only a single transaction list (purely horizontal representation), stored as an array. This array is process with a simple split and merge scheme, which computes a conditional database recursively. An occurrence counter and a pointer to the sorted transaction (array of contained items). This data structure is then processed recursively to find the frequent item sets. The basic operations of the recursive processing is based on depth-first/divide-and-conquer scheme. In, split steps given array is split with respect to the leading item of the first transaction. All array elements referring to transactions starting with this item are transfer to a new array. The new array created in the split step and the rest of the original arrays are combining with a procedure that is almost identical to one phase of the well-known merge sort algorithm. The main reason for the merge operation in SaM [3,8] is to keep the list sorted, so that, (1)All transactions with the same leading item are grouped together and (2) Equal transactions (or transaction suffixes) can be combined, thus reducing the number of objects to process.

Fig. 1 The example database: (1) original form, (2) item frequencies, (3) transactions with sorted items, (4) lexicographically sorted transactions, and the used (5) data structure

Fig. 2: The basic operations of the Split and Merge algorithm: split (left) and merge (right).

The steps illustrated in Fig. 1 for a simple example transaction database are below [3,8]:

1) Step 1: Shows the transaction database in its original form.
2) Step 2: The frequencies of individual items are determined from this input in order to be able to discard infrequent items immediately. If we assume a minimum support of three transactions for our example, there are no infrequent items, so all items are kept.
3) Step 3: The (frequent) items in each transaction are sorting according to their frequency in the transaction database, since it well known that processing the items in the order of increasing frequency usually leads to the shortest execution times.
4) Step 4: The transactions are sorted lexicographically into descending order, with item comparisons again being decided by the item frequencies, although here the item with the higher frequency precedes the item with the lower frequency.
5) Step 5: The data structure on which SaM operates is built by combining equal transactions and setting up an array, in which each element consists of two fields: an occurrence counter and a pointer to the sorted transaction. This data structure is then processed recursively to find the frequent item sets.

The basic operations in divide-and-conquer scheme reviewed [3,2] in Fig. 3.3.2. In the split step (see the left part of Figure) the given array is split w.r.t. the leading item of the first transaction (item e in our example): all array elements referring to transactions starting with this item are transferred to a new array. In this process, the pointer (in) to the transaction is advance by one item, so that the common leading item will remove from all transactions. Obviously,
IV. PROBLEM WITH CURRENT SAM

Here we will focus on frequent item set mining using divide and conquer technique in split and merge algorithm. As we have discussed on example how split is select and then merge item set is use for finding frequent. Some problems are arrives when taken results. This problem is critical at initial point. It creates problems at select item from item set and generates affected result.

We will discuss problem with example for specific situation like this.

![Fig. 3: Problems with SaM](Image)

Here one example is identifying the problem. There are 10 different transactions as shown in Fig. 4.1(Left). Now, each item frequency is initializing in shown in figure 4.1(Right).

For e=3, a=3, c=5, b=8, d=8. Now, e and a have frequency are same. Then how can select first split item for algorithm. In, first step both frequency are same. So these controversy is created to select e or select a. From initial point, we have to stop the calculation if we have this type of situation. SaM algorithm given affected result when this type of situation is created. We identify this problem and still work on find solution for SaM algorithm. When we get solution, we will present our result.

V. MODIFIED MECHANISM

As we have discussed in problem identification, when there is situation like first both items have same frequency then result is not proper. So now we have to find solution for that. We have solution for this. For this type of situation we have proposed one solution. For n different items if we want to use this algorithm for finding frequent item set, we have to consider first two same frequency counts with passing support. Among them which we have to select is dependent on number of transaction it contains. Suppose, here \( E \) has 3 transaction and \( A \) has 4 transaction, then we have to select least of them, i.e \( E \) is selected.

![Fig. 4: Problem Solution](Image)

We have to modify existing algorithm for reducing total execution time. In current algorithm too much scanning and sorting is used. So execution time is more. We have to modify this algorithm in such a way that result is not affected but execution time will decrease. We have made some modification for that. First check this modified algorithm steps. First two steps are as it was in Split and Merge algorithm. As discussed in problem with current split and merge algorithm. We have solved that problem with this algorithm.

- After Second Step, First assign all items which passes minimum support in array.
- Then according to transaction assign remaining items for each item. If any item is not starting with transaction then put it as it is.
- Remove least frequency item (single) with all its transaction.
- Copy and store all transaction items.
- Remove next least frequency item with all is transaction.
- Copy and store all transaction items.
- Repeat this until transaction is empty.

VI. EXPERIMENTS AND PERFORMANCE COMPARISON

We present our experimental results that show that the modified split and merge method achieves reasonably good result in terms of time. We processed three datasets. Algorithm has been implemented in C and platform used is Ubuntu 11.04 - the Natty Narwhal - released in April 2011. CPU with 2GB of RAM, 8 Processor and 20GB of hard drive space is used.

A. Dataset Information

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Chess</th>
<th>Mushroom</th>
<th>PUMBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Donated by</td>
<td>Roberto Bayardo</td>
<td>Roberto Bayardo</td>
<td>Roberto Bayardo</td>
</tr>
<tr>
<td>Total instances</td>
<td>1,18,252</td>
<td>1,86,852</td>
<td>36,29,404</td>
</tr>
<tr>
<td>Total Columns</td>
<td>37</td>
<td>23</td>
<td>74</td>
</tr>
<tr>
<td>Total Transaction</td>
<td>3196</td>
<td>8124</td>
<td>49046</td>
</tr>
<tr>
<td>Attributes type</td>
<td>Numeric</td>
<td>Numeric</td>
<td>Numeric</td>
</tr>
<tr>
<td>No of instances processed</td>
<td>All instances</td>
<td>All instances</td>
<td>All instances</td>
</tr>
<tr>
<td>Description</td>
<td>This data was collected from Roberto Bayardo from the UCI datasets. In this dataset, moves of chess game in numeric values stored. Total no of transaction are 3196. This is one type of dense dataset. The data set listing chess.</td>
<td>This data was collected from Roberto Bayardo from the UCI datasets. In this dataset numeric values stored. Total no of transaction are 8124. This is one type of sparse dataset. The data set describing poisonous and</td>
<td>This data was collected from Roberto Bayardo from PUMBS. In this dataset numeric values stored. Total no of transaction are 49046. This is one type of sparse dataset.</td>
</tr>
</tbody>
</table>
end game positions for king vs. king and rook. Edible mushrooms by different attributes.

Table 1: Dataset Information [11]

B. Results

We have taken results with different datasets with support threshold. We run algorithm on C framework and platform used is Ubuntu 11. CPU with 2GB of RAM, 8 Processor and 20GB of hard drive space is used. Describe results in below Table. We have found average result of execution time for Modified SaM and SaM algorithm [1, 3, 6]. We have compared our results with Eclat algorithm also. We have used item sets like Chess, Mushroom, PUMSB [12, 13, 14]. We have taken result for Eclat [3] algorithm for comparison. Eclat algorithm is used for finding Frequent Itemset Mining. We have compared this algorithm with our modified SaM and original SaM. Let us see the result of that.

Table 2: Execution Time of Chess dataset

As shown in Table 2, we have taken results for different support threshold for chess dataset. Here we compared support 50%-80% with total execution time. We have compared Eclat algorithm with our modified SaM and original SaM algorithm. The time of execution is decreased with the increase support threshold. Modified SAM gives good result as compared to other. Results show that Eclat’s performance is not good as compared to other.

Table 3: Execution Time of Mushroom dataset

Fig. 6: Execution Time of Mushroom dataset

Fig. 6 shows that the execution time of SaM and Modified SaM algorithm is nearby but it can also be analyzed that the execution time of SaM, Modified SaM and Eclat is comparatively same for higher support threshold. As experimental results SaM algorithm performs excellently on dense data sets, but shows certain weaknesses on sparse data sets.

As shown in Table 2.3, we have taken results for different support threshold for PUMSB dataset. Here we compared support 60%-80% with total execution time. The time of execution is decrease with the increase support threshold. Modified SaM performs better than Sam on sparse dataset. In sparse dataset SaM cannot perform good because of too much scanning and filtering. So Modified SaM gives good results for both sparse and dense dataset. Eclat performs averaged for PUMSB dataset.

Table 4: Execution Time PUMSB dataset

Fig. 7: Execution Time of PUMSB dataset

Fig. 7 shows that the execution time of SaM and Modified SaM algorithm is nearby but it can also be analyzed that the execution time of SaM, Modified SaM and Eclat is comparatively same for higher support threshold. As experimental results SaM algorithm performs excellently on dense data sets, but shows certain weaknesses on sparse data sets.
As shown in Fig. 7 shows the execution time for all the algorithms with different support threshold for PUMSB data set. The time of execution is decrease with the increase support threshold. Modified SaM gives good result as compared to SaM. For lower support our modified SaM does not give good performance for PUMSB dataset.

VII. CONCLUSION AND FUTURE ENHANCEMENT

In this paper, we study the frequent itemset mining and we study some of the basic algorithm of frequent itemset mining along with the one of the better algorithm for Split and Merge. After analysis of the all the things till now, we can say that SaM can’t work with some of the occasion. So we modify the current algorithm to find out the frequent itemset. We have observed frequent pattern mining algorithm with their execution time for specific datasets. In this thesis, an in-depth analysis of few algorithms is done which made a significant contribution to the search of improving the efficiency of frequent Itemset mining. By comparing our result to classical frequent item set mining algorithms like SaM and Eclat the strength and weaknesses of these algorithms were analyzed. As experimental results modified SaM algorithm performs excellently on dense data sets as well as sparse dataset up some support limit.

We have found different problems in this algorithm. If this problem is not solved then result is affected. So we suggest two different methods for getting better results. As experimental results modified SaM algorithm performs excellently on data sets as compared to original SaM and Eclat. We can also compare our algorithm to another classical frequent itemset mining algorithm.

Modified SaM works really better at the moment with compare to all other algorithms but we have planned to develop the algorithm which is more efficient and faster than the current version of Modified SaM and our main aim is to develop the Modified SaM such a way that consumes the less execution time compared to current version. One idea to make it more effective in terms of execution time, we have to reduce scanning and sorting such a way that preprocessing is less as compared to current. Second extension of the Modified SaM is that we can use some of the taxonomy, which eliminates some of the items, which are not frequent, at the beginning of the stage or user can decide which type of patterns he/she wants. So it will not waste the time and memory.

REFERENCES


