A SCALABLE ASSOCIATION RULES MINING ALGORITHM BASED ON SORTING, INDEXING AND TRIMING

CHUANG-KAI CHIOU, JUDY C. R TSENG

Department of Computer Science and Information Engineering, Chung Hua University, Hsinchu 300, Taiwan
E-MAIL: d09524003@chu.edu.tw, judycrt@chu.edu.tw

Abstract:
Apriori is an influential and well-known algorithm for mining association rules. However, the main drawback of Apriori algorithm is the large amount of candidate itemsets it generates. Several hash-based algorithms, such as DHP and MPIP, were proposed to deal with the problem. DHP employs hash functions to filter out potential-less candidate itemsets. MPIP further improves DHP by employing minimal perfect hashing functions to avoid generation of candidate itemsets. Though MPIP results in a very promising mining efficiency, the memory space required in MPIP increases rapidly as the number of items grows.

To obtain even better mining efficiency while reducing the memory space required, a Sorting-Indexing-Trimming (SIT) algorithm for mining association rules is proposed in this paper. SIT uses the sorting, indexing, and trimming techniques to reduce the amount of itemsets to be considered. Then, to utilize both the advantages of Apriori and MPIP, a revised MPIP algorithm is employed to deal with 2-itemsets, and a revised Apriori algorithm to deal with \( k \)-itemsets for \( k > 2 \).

Though the memory space required in SIT is less than MPIP, from the experiment results, SIT outperforms both Apriori and MPIP.

Keywords:
- Data mining; Association rule; Apriori algorithm

1. Introduction

With the popularization of computers, digital data grows in an exponential rate. To find out the knowledge behind the huge amounts of data becomes an important issue. For example, it is helpful to develop marketing management strategies according to the relationships among products found in the transaction database of an enterprise. That’s why data mining becomes a more and more important research topic. 

Among data mining methodologies, “Association Rule Mining” is the most widely applied technique. The most well-known method to mining association rules is the Apriori algorithm [1,2]. In recent years, many association rule mining algorithms proposed are also Apriori-based.

Apriori algorithm employs a “level-wise search” approach. The \( k \)-itemsets is used to find the \((k+1)\)-itemsets. During the searching, a set of candidate \((k+1)\)-itemsets is generated by joining frequent \( k \)-itemsets with itself. Each candidate itemset is verified against the database. This algorithm terminates when no further candidate itemsets can be found. It also implies that lots of candidate itemsets will be generated during the mining process, as well as the database will be accessed frequently, which results in a low performance.

Many researches tried to find efficient methods to improve Apriori. For instance, FP-Tree [3,4], CHARM-L [5], and m-FTP [6] employ special tree structures for mining the frequent itemsets. Partition [7] and the techniques proposed by Jin etc. [8] employ the concept of parallelization to share the loading of memory computing. Some algorithms, such as DHP [9], Princer-Search [10], avoid generating potential-less candidate itemsets, but the efficiency is not well. It still depends on the large amount of candidate itemsets, especially in low support cases.

Some algorithms find out the frequent itemsets without generating candidate itemsets, like FP-Tree and MPIP [11]. Although FP-Tree algorithm can find out the frequent itemsets directly, there still exist a lot of branches. The improvement is still limited. MPIP uses a hash function to guarantee no collision will be occurred. It is suitable for dealing with candidate 2-itemsets, which is the most time-consuming process in association rules mining. MPIP performs well at time efficiency, however, the memory requirement grows in exponential rate at later phase. MPIP improves the mining performance only when finding frequent 2-itemsets.

To obtain even better mining efficiency while reducing the memory space required, a Sorting-Indexing-Trimming (SIT) algorithm for mining association rules is proposed in this paper. SIT uses the sorting, indexing, and trimming techniques to reduce the amount of itemsets to be
considered. Then, to utilize both the advantages of Apriori and MPIP, a revised MPIP algorithm is employed to deal with 2-itemsets, and a revised Apriori algorithm to deal with k-itemsets for k>2.

Though the memory space required in SIT is less than MPIP, from the experiment results, SIT outperforms both Apriori and MPIP.

2. Related Works

The most well-known method to mining association rules is the Apriori algorithm. In recent years, many association rule mining algorithms are proposed, but many of them are Apriori-based.

Apriori algorithm employs a “level-wise search” approach. It also implies that lots of candidate itemsets will be generated during the mining process, as well as the database will be accessed frequently, which results in a low performance.

Some algorithms are focus on reducing the large amount of candidate itemsets, such as DHP and MPIP.

In Section 2.1, we will introduce the basic algorithm—Apriori algorithm. DHP, which employs hash functions to reduce the candidate itemsets, will be introduced in Section 2.2. A revised DHP algorithm—MPIP, which employs minimal perfect hashing functions, will be introduced in Section 2.3.

2.1. Apriori Algorithm

Apriori-based algorithm is the most representative type of mining association rules. As the name of Apriori algorithm, it utilizes prior knowledge to generate the candidate itemsets. It means the k-itemsets are used to explore (k+1)-itemsets.

For finding frequent itemsets, Apriori algorithm makes multiple passes over the database. Each pass consists of two phase. (1)Candidate Generation: Let frequent k-itemsets denotes as \(L_k\) and the potentially frequent itemsets denotes as \(C_k\). The \(C_k\) is generated by joining \(L_{k-1}\) with itself. This procedure ensures the \(C_k\) is a superset of the set of all frequent k-itemsets. (2)Support Counting: For each transaction, the candidate itemsets \(C_k\) will be determined with their support counts. If the support is less than the minimum support, the corresponding candidate itemsets will be eliminated. The remaining candidate itemsets \(C_k\) becomes the k-itemsets. The algorithm terminates when \(L_k\) or \(C_{k+1}\) becomes empty.

Apriori algorithm can be realized and implemented easily but two bottlenecks that have to be dealt with. One is the large candidate itemsets. The candidate \(C_2\) is generated by joining of any two 1-itemsets. If there are \(k\) items in 1-itemsets, \(k \times (k-1)/2\) candidate itemsets will be generated. Assume that if there are 100 items in \(L_1\), 4,950 candidate items will be generated in \(C_2\). Another bottleneck is the huge scanning times. Each pass of Candidate Generation will determine the support values of candidate itemsets by scanning the database which results in the high scanning frequency in database.

2.2. DHP Algorithm

For dealing with the low performance of Apriori-based algorithm, Chen et al. proposed DHP (Direct Hashing and Pruning) algorithm. DHP employ hash functions to generate candidate item sets efficiently, and DHP also employs effective pruning techniques to reduce the size of database. In early stage of candidate generation, DHP enables us to trim the transaction database. The potential-less itemsets will be filtered out, and the scanning of database will be avoided. Since DHP does not scan over the database all the time, the performance is also enhanced. A simple example is shown in Figure 1.

![Figure 1. Generating frequent 2-itemsets by DHP](image)
2.3. MPIP Algorithm

The performance of DHP is affected significantly with the collision rate of the hash functions. A minimal perfect hashing scheme, MPIP, is proposed to cope with such problems. The minimal perfect hashing function promises that no collisions will occur in the hash table (compare Figure 2 with Figure 1). MPIP employs a minimal perfect hashing function and the time needed for scanning and searching data items can be reduced. Moreover, by employing the minimal perfect hashing function and a transaction pruning strategy, MPIP is capable of reducing the execution cost without increasing the requirement for memory space.

**Figure 2. Generating frequent 2-itemsets by MPIP**

MPIP generates frequent 2-itemsets from one scan over the database without generating $C_2$. MPIP reduces many disk I/O time and the requirement of memory size can be estimated before starting the mining operations. In another aspect, MPIP algorithm can also be integrated with other database pruning or Scan-reduction methods (e.g. the pruning method of DHP) to achieve better performance under different circumstances.

MPIP employs the minimal perfect hashing function for mining $L_1$ and $L_2$ and employs the Apriori algorithm for finding the frequent $k$-itemsets for $k>2$. With the increasing of value $k$, the requirement of memory will increase in an exponential rate. If there are large amount of itemsets in 3-itemsets, the performance still could not be improved.

The association rule approaches we mention in this chapter have their advantage and drawbacks. Apriori is simple and save the memory space, but the performance is poor. DHP uses a hash function to avoid generating $C_2$, but the improvement is not well. MPIP deals with the collision problem suffered in DHP, but the requirement of memory space is too large for mining frequent $k$-itemsets for $k=2$. After analyzing these algorithms, we found that the critical part is to find out an efficient approach to filter the large candidate itemsets.

3. SIT Algorithm

For mining association rules, we propose a revised algorithm, Sorting-Indexing-Trimming (SIT) approach. SIT approach can avoid generating potential-less candidate itemsets and enhance the performance via Sorting, Indexing and Trimming. The details will be shown in the following sections.

3.1. Sorting

Sorting brings benefits for the algorithms of association rule mining as it in the binary searching tree. For a better performance, SIT sorts the items by the occurring frequency in increasing order which is opposed to the others. The increasing will be useful in Trimming phase and the details will be explained in section 3.3. The sorting process in SIT is shown in Figure 3.

**Figure 3. Sorting process in SIT**

(1) There is the original transaction database.
(2) Count the occurred frequency.
(3) Sort the items by the counts in increasing order and build a mapping table.
(4) Translate the items into mapping numbers (for instance $a \rightarrow 1$, $c \rightarrow 16$).
(5) Re-sort the item ordering in each transaction.

3.2. Indexing

Indexing is a useful technique in data retrieval, for instance, indexing can speed up the searching in database system. It can also be applied in association rule mining.
The upper left one in Figure 4 is the original database and the right one is the \( C_2 \) and the counting of comparison times. For example, if we want to ensure whether the itemset \( \{a, b\} \) is the subset of \( T_1 \) (\( Tid=1 \)) or not, we need to compare it with \( T_1 \) twice. First, we compare the first element in \( T_1 \) with \( a \), and we find it. Then we compare the second element in \( T_1 \) with \( b \), but we get \( c \). That means itemset \( \{a, b\} \) is not the subset of \( T_1 \). Keeping on this process, we use 69 comparison times to ensure whether all the itemsets in \( C_2 \) is the subset of \( T_1 \) or not.

In SIT, we use an index table to determine the comparison range. The first three items in \( T_1 \) are \( 1, 3 \) and \( 5 \) and they all indexing as 0, so there is no comparison needed. The 4th item is \( 12 \) and it indexed from 1–5 that means the candidate itemsets from position 1–5 (i.e., candidate itemsets \{12,13\}, \{12,14\}, \{12,15\}, \{12,16\}, \{12,17\}) need to be compared. Because we ensure the item \( 12 \) exists in \( T_1 \), we only need to compare from the 5th item–14.

For comparing \{12, 13\}, we compare 13 with 14, and we ensure that itemset \{12, 13\} doesn’t exist in \( T_1 \). For comparing the other four itemsets, we compare 1, 2, 3, 4 times respectively. We only compared 21 times to ensure whether all the itemsets in \( C_2 \) is the subset of \( T_1 \) or not what is less than 69 times in Apriori. Hence, the Sorting and Indexing processes can reduce the comparison times in Apriori and enhance the performance.

3.3. Trimming

Because Indexing has filtered out the potential-less itemsets, the improvement with Trimming for the Apriori algorithm is small but it is significant for MPiP. If the minimum support is \( 3 \), all the items with frequency less than 3 will be trimmed. As we have mentioned in previous chapter, if we sort the itemsets in decreasing order, it is difficult to trimming the database. For reserving the data, physical trimming will be avoided. We just record the starting position, and generate the hash table from this position. The trimming result of the database is shown in Figure 6.

3.4. SIT Algorithm

SIT utilizes both the advantage of MPiP and Apriori. The processes of SIT algorithm is shown as the following:

For finding \( L_1 \) and \( L_2 \):

1. Employ the Sorting, Indexing and Trimming techniques to the original database.

2. Employ MPiP algorithm to find \( L_1 \) and \( L_2 \)

For finding the \( k \)-itemsets for \( k \geq 2 \):

1. Employ Apriori algorithm to database which has been sorted, indexed and trimmed.

2. Find out the frequent itemsets.
Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, Hong Kong, 19-22 August 2007

In SIT, *Sorting* is the essential technique for all the other processes. Even though the minimum support value is changed, we still need not to sort the database again unless the content is changing.

4. Experiment and Evaluation

To evaluate the performance of SIT, we perform several experiments. The equipment we use is a PC with Pentium IV 2.4G processor with 1 GB memory and a Microsoft SQL Sever2000 database. The sorting process in SIT is operated by the SQL, and the sorted database is built by INSET and SELECT instruction in SQL. In our experiment, the testing data is generated by the data generator from IBM’s Almaden project[8] which is used in many researchers. The meaning of parameters is shown as follows:

\[
\begin{align*}
N & : \text{Number of data items} \\
|D| & : \text{Number of transactions in the database} \\
|T| & : \text{Average size of transactions} \\
|I| & : \text{Average size of the maximal potentially frequent itemsets used to generate the datasets} \\
|L| & : \text{Number of maximal potentially frequent itemsets}
\end{align*}
\]

The following experiments are focus on two parts to evaluate the SIT algorithm. (1) Performance of Apriori, SI+Apriori, MPIP, and SIT. (2) Performance of SIT and MPIP under different transaction qualities and length.

(1) Performance of SIT, Apriori, MPIP, and SI+Apriori.

In this part, four methods will be executed with minimum support from 2%~0.25%. The SI+Apriori means a revised Apriori algorithm with *Sorting* and *Indexing* processing. SI+Apriori and SIT, we used here, both were pre-sorted and pre-indexed. The result is shown in Figure 7.

![Figure 7. Experiment on T10.I6.D100K](image)

Obviously, SIT is the best one, because SIT utilizes both the advantages of MPIP and SI+Apriori. SIT also get a stable performance even in a low minimum support situation. By comparing with Apriori and SI+Apriori employing *Sorting* and *Indexing* process can enhance the performance of Apriori indeed.

(2) Performance of SIT and MPIP under different transaction qualities and length.

We can easily distinguish the Apriori and the others in the first part, so the following part we are focus on the comparison between SIT and MPIP. Figure 8 and 9 are the experiments with minimum support S=0.5% and 0.25%. The time of pre-sorting and pre-indexing are taken into consideration in SIT2. The performance of SIT is always better than MPIP, especially with the large amount of transaction and low support value. SIT2 is better than MPIP, while the support value is less than 0.25%.

![Figure 8. Experiment on T10.I4 with S=0.5%](image)

![Figure 9. Experiment on T10.I4 with S=0.25%](image)

Experiment presented in Figure 10 is focus on the different transaction length. When the transaction length is less than 10, the difference of SIT and MPIP are little but the difference is very large for the length bigger than 20.

![Figure 10. Experiment on 16.D100K with S=0.5%](image)
According to the experiment results, we find that Apriori is not suitable for the transaction which will generate large amount of candidate itemsets. MPIP can solve the problem of large $C_2$. MPIP also has a nice performance with different support value, transaction amount and length while the amount of candidate itemsets after $C_3$ is small. Sorting and Indexing can reduce the comparisons in a large transaction database. SIT is more efficiently than MPIP while there are large amount of candidate itemsets.

5. Conclusion and Future works

In this paper, we proposed a SIT approach to enhance the performance of associate rule mining. SIT approach revises MPIP and Apriori with Sorting and Indexing for finding the $2$-itemsets and the frequent $k$-itemsets for $k>2$. SIT reduces the amount of candidate itemsets which results a low performance in Apriori algorithm, and SIT also avoids generating potential-less candidate itemsets. The performance of SIT is better than Apriori, DHP and MPIP.

Although SIT is more efficiently than the other approach, several problems still need to be dealt with for the practices. One is the incremental of data sets. When the data sets in the data warehouse are increasing, we need to sort and index again for association rule mining. However, it is difficult for some data warehouses which updates frequently. If the amount of each candidate itemsets can be stored, we just need to determine the incremental parts that can reduce the computation time efficiently. Although we can not use the Sorting and Trimming in SIT approach, the time required for determining incremental parts will be less than Sorting and Trimming.

Another one is the long transaction length. Items in SIT will be mapped into corresponding index number. This process is time-consuming for the long transaction length. If we do not consider the requirement of memory space, creating a mapping matrix for each potential itemset will gain a better performance than SIT and MPIP. For enhancing the memory space, we can employ the Partition method [4] to partition the database into several parts in different servers. The candidate itemsets can be determined separately, and then summarizing by accumulating the counts of each candidate itemsets table. This method not only reduces the memory space but also suits for disturbed computing environment.

Acknowledgements

The authors would like to thank Mr. Shen-Wan Lo for his helps in our experiments.

References

[3] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation”, ACM SIGMOD, pp. 1-12, Dallas, TX, May 2000.