Mining Frequent Item Sets More Efficiently Using ITL Mining

R Hemalatha A Krishnan R Hemamathi

Abstract

Correlated The discovery of association rules is an important problem in data mining. It is a two-step process consisting of finding the frequent itemsets and generating association rules from them. Most of the research attention is focused on efficient methods of finding frequent itemsets because it is computationally the most expensive step. This paper presents a new data structure and a more efficient algorithm for mining frequent itemsets from typical data sets. The improvement is achieved by scanning the database just once and by reducing item traversals within transactions. The performance comparisons of the algorithm against the fastest Apriori implementation and the recently developed H-Mine algorithm are given here. These results show that the algorithm outperforms both Apriori and H-mine on several widely used test data sets.

Keywords : Data Mining, Data Structure.

0. Introduction

Association Rules are used to identify relationships among sets of items. They are relevant to several domains such as the analysis of market basket transactions in retail stores, target marketing, fraud detection, finding patterns in telecommunication alarms, etc. In retail stores for example, this information is useful to increase the effectiveness of advertising, marketing, inventory control, and stock location on the shop floor. Since the introduction of association rules a decade ago [1], a large number of increasingly efficient algorithms have been proposed [2,3,4,5,6,7].

The process of mining association rules consists of two steps :

1) Find the frequent itemsets that have *minimum support*;

2) Use the frequent itemsets to generate association rules that meet the *confidence threshold*.

Between these two steps, step 1 is the most expensive since the number of itemsets grows exponentially with the number of items. The strategies developed to speed up this process can be divided into two categories. The first is the candidate generation-and-test approach. Algorithms in this category include Apriori and its several variations [1,2,13,14,15,16]. They use the Apriori property also known as antimonotone property that any subset of a frequent item set must be a frequent item set. In this approach, a set of candidate item sets of length $n + 1$ is generated from the set of item sets of length n and then each candidate item set is checked to see if it meets the *support threshold*. The second approach of pattern-growth has been proposed more recently.

It also uses the Apriori property, but instead of generating candidate item sets, it recursively mines patterns in the database counting the support for each pattern. Algorithms in this category include TreeProjection [7], FP-Growth [3,14,15] and H-Mine [4]. Algorithms based on candidate generation and test such as Apriori runs very slowly on long pattern data sets because of the huge number of candidate itemsets it has to generate and test.

Testing the candidate itemsets for minimum support requires scanning the whole database many times, although for some small to moderate size databases, the scan can be made faster by storing the database in main memory. The problem of generating candidates can be avoided by using the pattern-growth approach.

For most data sets, these algorithms perform better than Apriori. Among the existing pattern-growth algorithms, H-Mine runs faster than TreeProjection and FP-Growth on several commonly used test data sets. H-mine scans the transaction database twice. It performs repeated horizontal traversals of the transactions in memory while generating frequent itemsets. H-Mine also needs to continually re-adjust the links between transactions in its H-struct data structure during the mining process. If these costs are reduced, the mining process will be improved further.

A more efficient algorithm based on the pattern-growth approach is introduced here. It reduces the cost by scanning the database only once, by significantly reducing the horizontal traversals of the transactions in memory and keeping the links between transactions in memory unchanged during the mining process. To achieve these reductions in cost, a new data structure called Item-Trans Link (ITL) and the new algorithm called ITL-Mine is presented.

The performance of ITL-Mine is compared with Apriori and H-Mine algorithms and found that ITL-Mine performs better on a range of widely used test data sets. H-mine was chosen for performance comparisons because it is an improvement over FP-Growth and TreeProjection algorithms. The H-Mine algorithm is based on its description in [4]. The Apriori program used is from [11], which is generally acknowledged as the fastest Apriori implementation available.

Tid	Items
1	3 4 5 6 7 9
2	1 3 4 5 13
3	1 2 4 5 7 11
4	1 3 4 8

Tid	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	0	1	1	1	1	1	0	1	0	0	0	0
2	1	0	1	1	1	0	0	0	0	0	0	0	1
3	1	1	0	1	1	0	1	0	0	0	1	0	0
4	1	0	1	1	0	0	0	1	0	0	0	0	0

Fig. 1. The Transaction Database

1. Definitions and ITL Data Structure

This section defines the terms used for describing association rule mining. The conceptual basis for design of the data representation is presented followed by a description of the ITL data structure.

1.1 Definition of Terms

The basic terms needed for describing association rules using the formalism of [1,13,14,15]. Let I={i1, i2, … , in} be a set of items, and *D* be a set of transactions, where a transaction *T* is a subset of *I* (*T* Í *I*). Each transaction is identified by a *TID*. An association rule is an expression of the form *X* Þ *Y*, where *X* Ì *I*, *Y* Ì *I* and *X* Ç *Y* = Æ. Note that each of X and Y is a set of one or more items and the quantity of each item is not considered. *X* is referred to as the *body* of the rule and *Y* as the *head*.

An example of association rule is the statement that 80% of transactions that purchase A also purchase B and 10% of all transactions contain both of them. Here, 10% is the *support* of the itemset {A, B} and 80% is the *confidence* of the rule *A* Þ *B*. An itemset is called a *frequent itemset* if its *support* is greater than or equal to a *support threshold* specified by the user, otherwise the itemset is *not frequent*. A *k-frequent itemset* is a frequent itemset that contains *k* items.

1.2 Binary Representation of Transactions

The transactions in the database could be represented as a binary table [1] as shown in Figure 1. Counting the support for an item can be considered as counting the number of 1's for that item in all the transactions. In most datasets, the number of items in each transaction is much smaller than the total number of items, and therefore the binary table representation will not allow efficient use of memory. Therefore, use a more efficient representation scheme.

1.3 Item-Trans Link (ITL) Data Structure

Researchers have proposed various data representation schemes for association rule mining. They can be broadly classified as horizontal data layout, vertical data layout, and a combination of the two. Most candidate generation and test algorithms (e.g. Apriori) use the horizontal data layout and most pattern-growth algorithms like FP-Growth and H-Mine use a combination of vertical and horizontal data layouts. A data structure called **Item-Trans Link (ITL)** that combines the vertical and horizontal data layouts is proposed (see Figure 2).

The data representation used by the algorithm is based on the following observations :

1. Item identifiers may be mapped to a range of integers.
2. Transaction identifiers can be ignored provided the items of each transaction are linked together.

ITL consists of an item table (ItemTable) and the transactions linked to it (TransLink) as follows:

1. ItemTable: It contains all the items and the support of each item. It also has a link to the first occurrence of each item in the transactions of TransLink described below.
2. TransLink: It represents the items of every transaction for all the transactions in the database. The items of a transaction are arranged in sorted order.

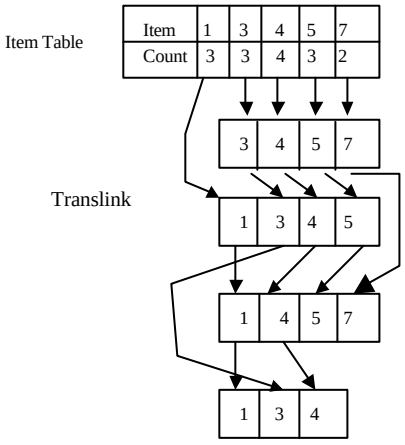

Fig. 2. The Item-Trans Link (ITL) Data Structure

For each item in a transaction, it contains a link to the next occurrence of that item in another transaction. In other words, this link will represent all the 1's for each item so that the counting can be done quickly. For example, in Figure 2, to check the occurrences of item 7, go to the cell of 7 in tid 1 in the TransLink and then directly to the next occurrence of 7 in tid 3 without traversing tid 2. Since ITL has features of both horizontal and vertical data layouts, it is general and flexible enough to be used by algorithms that need horizontal, vertical or combined data layout. It also makes it possible to combine existing ideas for efficient mining based on both layouts.

ITL is similar to H-struct proposed in [4], except for the vertical links between the occurrences of each item in the transactions. In H-struct, the links always point to the first item of the transaction, and therefore to get a certain item, traverse the transaction from the beginning. ITL points directly to the occurrence of the item which makes it faster to traverse all occurrences of an item.

2. ITL-Mine Algorithm

This section describes the ITL-Mine algorithm, and provides a running example. ITL-Mine assumes that the ItemTable and TransLink will fit into main memory. With the availability of increasingly larger sizes of main memory that currently approach gigabytes, many small to moderate databases will fit in the main memory. However, the extension of this algorithm to mine very large databases is currently in progress.

There are three steps in the ITL-Mine algorithm as follows:

- Construct ItemTable and TransLink: In this step, the ItemTable and TransLink are constructed by a single scan of the transaction database. At the end of this step, the 1-frequent itemsets will be identified in the ItemTable by the support count.
- Prune: Using the anti-monotone property, the infrequent items are pruned or deleted from the TransLink since infrequent items will not be useful in the next step.
- Mine Frequent Itemsets: In this step, all the frequent itemsets of two or more items are mined using a recursive function as described further in this section.

Example 1 : The ITL-Mine algorithm is illustrated by this example. Let Table 1 be the transaction database and suppose the user wants to get the Frequent Itemsets with minimum support = 50% (minimum 2 transactions). In Step 1, all transactions in the database are read in a single scan to construct the ItemTable and TransLink. For each item in a transaction, the existence of the item in the ItemTable is checked. If the item is not present in the ItemTable, it is entered with an initial count of 1, otherwise the count of the item is incremented. After that, the item is entered in the proper location in the TransLink.

3. Procedure Construct-ITL

For all transactions in the DB

For all items in transaction

If item in ItemTable

Increment count of item

Else

Insert item with count = 1

End If
Insert Item into TransLink
Connect link of previous
occurrence to this item
End For
End For

Procedure Prune-ITL
For all xÎItemTable where count(x)< min_sup
Delete x from ItemTable and TransLink
End For

Procedure Mine-FI
For all xÎItemTable
Add x to the set of Frequent Itemsets
Prepare and fill tempList for x
For all yÎtempList where count(y)³min_sup
Add xy to the set of Frequent Itemsets
For all zÎtempList after y
where count(z) ³min_sup
RecMine (xy, z)
End For
End For
End For

Procedure RecMine(prefix, test_item)
tlp:= tid-list of prefix
tli:= tid-list of test_item
tl_current = Intersect(tlp,tli)
If size(tl_current) ³?min_sup
new_prefix:= prefix + test_item
Add new_prefix to the set of Frequent
Itemsets

For all zÎ tempList after test_item

Where count (z) ³ ?min_sup

RecMine(new_prefix, z)

End For

End If

Fig. 4. Algorithms for Construct, Prune and Mine-FI and the links are made.

Prefix	**TempList (count)**	**Freq-Itemsets (count)**
1	3 (2), 4 (3), 5 (2), 7 (1)	1 (3), 1 3 (2), 1 4 (3), 1 5 (2)
1 3	4 (2), 5 (1)	1 3 4 (2)
1 4	5(2), 7(1)	1 4 5 (2)
3	4 (3), 5 (2), 7 (1)	3(3), 3 4 (3), 3 5 (2)
3 4	5 (2), 7(1)	3 4 5 (2)
4	5 (3), 7(2)	4 (4), 4 5 (3), 4 7 (2)
4 5	7 (2)	4 5 7 (2)
5	7(2)	5 (3), 5 7 (2)
7	None	7(2)

Fig. 3. Mining Frequent Itemsets Recursively Support of frequent itemsets shown in brackets)

On completing the database scan, the 1-frequent itemsets can be identified in the ItemTable as {1, 3, 4, 5, 7}. In Step 2, all items in the ItemTable are traversed to prune the infrequent items. If the support count of an item in the ItemTable is below the minimum, it is removed from both the ItemTable and the TransLink. The pruning is done by following the link from the ItemTable to traverse all the occurrences of that item in the TransLink. After the pruning, the TransLink is as shown in Figure 2.

In the last Step, each item in the ItemTable will be used as a starting point to mine all longer frequent itemsets for which it is a prefix. As an example, starting with item 1, follow the link to get all other items that occur together with item 1. Items that occur together with item 1 will be registered in a simple table that called TempList together with their support count and list of tids. As in Figure 3, for prefix 1, items {3, 4, 5} that are frequent (their support ³ 2). Generating the frequent patterns for this step involves simply concatenating the prefix with each frequent-item.

As an example, the frequent itemsets for this step are 1 3 (2), 1 4 (3) and 1 5 (2). After generating the 2-frequent-itemsets for prefix 1, since we have got the tid-list of each item in the TempList, we can recursively use the tid intersection scheme to generate the subsequent frequent itemsets. For example, we can use the tid-list of 3 and intersect with tid list of 5 to generate frequent itemsets 1 3 5. At the end of recursive calls with prefix item 1, all frequent itemsets that contains item 1 will be generated: 1 (3), 1 3 (2), 1 4 (3), 1 5 (2), 1 3 4 (2), 1 4 5 (2). In the next sequence, item 3 will be used to generate all frequent itemsets that contain item 3 but does not contain item 1. Then item 5 will be used to generate all frequent itemsets that contain item 5 but does not contain items 1 and 3. The algorithm of ITL-Mine is shown in Figure 4.

3. Performance Study

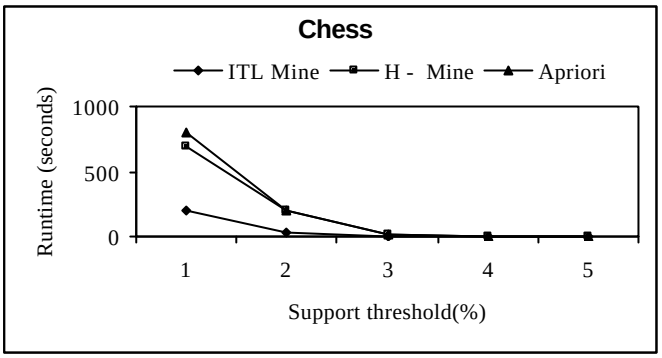

Fig. 5. Performance comparison of Apriori, H-Mine and ITL-Mine on chess datasets

In this section, the performance evaluation of ITL-Mine is presented. All the tests were performed on an 866MHz Pentium III PC, with 128 MB RAM and 30 GB HD running Microsoft Windows 2000. ITL-Mine is written in Microsoft Visual C++ 6.0. In this paper, the runtime includes both CPU time and I/O time.

The chess dataset has been used to test the performance Chess (3196 trans, max 37 items per trans). The Chess dataset is derived from the steps of Chess games. Chess is dense dataset since it produces many long patterns of frequent itemsets for very high values of support. It is downloaded from [9]. Comparison of ITL-Mine with the fastest available Apriori implementation [12] and the implementation of H-Mine. Performance comparison of Chess dataset is shown in Figure 5. The result shows that ITL-Mine outperforms Apriori and H-Mine on these datasets for the given parameters.

4. Discussion

The algorithm follows the pattern growth approach without candidate generation, so it is compared with H-Mine, the most recently proposed algorithm of this class. H-Mine is also considered to be more efficient than TreeProjection and FP-growth algorithms [4]. The better performance of the algorithm is:

- This algorithm traverses the database only once and performs the rest of the mining process using ITL. H-mine performs two scans of the database to build the H-struct data structure.
- After the ITL data structure is constructed and pruned to remove infrequent 1-itemsets, it remains unchanged while mining all of the frequent patterns. In H-Mine, the pointers in the H-struct need to be continually readjusted during the extraction of frequent patterns and so needs additional computation.
- ITL-Mine uses a simple temporary table called TempList during the recursive extraction of frequent patterns. ITL-Mine stores the tid-list of each item in the TempList and uses tid intersection scheme to generate frequent patterns. The TempList stores only the information for the current recursive call and the space will be reused for the subsequent recursive calls. H-Mine builds a series of header tables linked to the H-struct and it needs to change pointers to create or re-arrange queues for each recursive call.

- The recursive calls in H-Mine also involve repeated traversals of relevant parts of the H-struct. ITL-Mine avoids these repeated traversals by using the tid intersections. So the additional computation required by H-Mine to extract the frequent itemsets from Hstruct are more than for ITL-Mine. It supports more efficient interactive mining, where the user may experiment with different values of minimum support levels. In this case the pruning step would be skipped. Using the constructed ItemTable and TransLink in the memory, if the user wants to change the value of support threshold, there is no need to re-read the transaction database.

5. Conclusion

In this paper, a generic data structure called Item-Trans Link (ITL) and a new algorithm called ITL-Mine for discovering frequent itemsets is presented. The algorithm needs to scan the transaction database only once. The performance of ITL-Mine is compared against Apriori and H-Mine on chess dataset and the result show that ITL-Mine outperforms both Apriori and H-Mine on the data set for the given ranges of support levels. Assume that the ItemTable and TransLink will fit into main memory.

However, this assumption will not apply for huge databases. The extension of ITL-Mine for very large databases is currently underway. Several researchers have investigated ways to reduce the size of frequent itemsets to manageable levels for users and to allow greater user focus in the mining process [8]. User-specified constraints on the mining process represent a promising approach to this problem.

6. References

1. R. Agrawal, T. Imielinski, and A. Swami, “Mining Association Rules between Sets of Items in Large Databases”, Proc. of the ACM SIGMOD Conf., Washington DC, 1993.
2. R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules”, Proc. of the 20th Int. Conf. on VLDB, Santiago, Chile, 1994.
3. J. Han, J. Pei, and Y. Yin, “Mining Frequent Patterns without Candidate Generation”, Proc. of the ACM SIGMOD Conf., Dallas, TX, 2000.
4. J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, “H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases,” Proc. of the 2001 IEEE ICDM, San Jose, California, 2001.
5. M. J. Zaki, “Scalable Algorithms for Association Mining,” IEEE Transactions on Knowledge and Data Engineering, vol. 12, pp. 372-390, May/June 2000.
6. P. Shenoy, J. R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah, “Turbo-charging Vertical Mining of Large Databases,” Proc. of the ACM SIGMOD Conf., Dallas, TX USA, 2000.
7. R. Agarwal, C. Aggarwal, and V. V. V. Prasad, “A Tree Projection Algorithm for Generation of Frequent Itemsets”, Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining), 2000.
8. J. Pei, J. Han, and L. V. S. Lakshmanan, “Mining Frequent Itemsets with Convertible Constraints”, Proc. of 17th ICDE, Heidelberg, Germany, 2001.
9. Irvine Machine Learning Database Repository: http://www.ics.uci.edu/~mlearn/ MLRepository.html
10. http://www.ecn.purdue.edu/KDDCUP
11. http://www.cs.sfu.ca/ ~peijian/personal/publications
12. Apriori version 4.01, available at http://fuzzy.cs.unimagdeburg.de/~borgelt/).

13. A. Krishnan, R. Hemalatha, C. Senthamarai, “Mining of Association Rules in Distributed Database Using Partition Algorithm”, International Conference on Systemic, Cybernetics & Informatics (ICSCI 2004), February, 2004

14. A. Krishnan, R. Hemalatha, “Parallel Association Rule Mining – Finding Frequent Patterns Without Candidate Generation”, National Level Conference, Tech Fete 2004 on “Intelligence Techniques”, February 2004.

15. A. Krishnan, R. Hemalatha, R. Hemamalini, “Mining Frequent Patterns Without Candidate Generation in Distributed Databases”, National Conference on Distributed Database and Computing, March 2004.

16. A. Krishnan, R. Hemalatha, C. Senthamarai, “Association Rule mining with the Pattern Repository”, National Conference on Data Mining, December 2004.

About Authors

Mrs. R Hemalatha is a Lecturer in Department of Computer Science in K. S. R. College of Technology, Tiruchengode. Namakkal Dt., Tamil Nadu. She holds M.Sc. (I.T.)
E-mail : hemaa_msc@yahoo.com

Dr. A Krishnan is a Principal in the R. R. Engineering College, Tiruchengode. Namakkal Dt., Tamil Nadu.
E-mail : a_krishnan26@hotmail.com

Ms. R Hemamathi is a MCA Final Year Student at M. Kumarasamy College of Engineeing, Thalavappalayam, Karur Dist., Tamil Nadu.
E-mail : hemamathi_mca@rediffmail.com