Asian Journal of Information Management

ISSN 1819-334X

Asian Journal of Information Management 1 (1): 11-18, 2007
ISSN 1819-334X
© Academic Journals Inc., USA

Design of Algorithm for Frequent Pattern Discovery Using Lattice Approach

[1]Sanjeev Sharma, [1]Akhilesh Tiwari, [3]Sudhir Sharma and [4]K.R. Pardasani
[1]School of Information Technology,
[3]University Institute of Technology,
Rajiv Gandhi Technological University, Bhopal M.P. India
[3]MANIT, Bhopal M.P. India

Abstract: Data mining has recently attracted considerable attention from database practitioners and researchers because of its applicability in many areas, such as decision support, market strategy and financial forecasts combing techniques from the fields of machine learning, statistics and database, data mining enables us to find out useful and invaluable information from huge databases. Mining of association rules has received much attention among the various data mining problems. Most algorithms for association rule mining are the variants of the basic Apriori algorithm. One characteristic of these Apriori-based algorithms is that candidate itemsets are generated in rounds, with the size of the itemsets incremented by one per round. The number of database scan required by Apriori-based algorithms thus depends on the size of the largest large itemsets. In this research we proposed a new candidate set generation algorithm, which generates candidate itemsets of multiple sizes at each iteration by taking input as suggested large itemsets.

Key words: Data mining, apriori algorithm, new_gen, itemsets

Introduction

Data mining (Han and Kambe, 2001) has recently attracted considerable attention from database practitioners and researchers because of its applicability in many areas, such as decision support, market strategy and financial forecasts combing techniques from the fields of machine learning, statistics and database, data mining enables us to find out useful and invaluable information from huge databases. Mining of association rules has received much attention among the various data mining problems. The retail industry provides a classic example application (Frawley *et al*., 1991). Typically, a sales database of a supermarket stores, for each transaction, all the items that are bought in that transaction, together with other information such as the transaction time, customer-id etc. The association rule mining problem (Srikant *et al*., 1997) is to find out all inference rules such as: A customer who buys item X and item Y is also likely to buy item Z in the same transaction, Where X, Y and Z are not known beforehand. Such rules are very useful for marketers to develop and to implement customized marketing programs and strategies.

Problem Statement

The iterative nature of the Apriori algorithm implies that at least n database passes are needed to discover all the large itemsets. If the biggest large itemsets are of size n. Since database passes involve slow disk access, to increase efficiency, we should minimize the number of database passes during the

Corresponding Author: Sanjeev Sharma, School of Information Technology, Rajiv Gandhi Technological University, Bhopal M.P. (India)

mining process (Agrawal *et al*., 1993). One solution is to generate bigger-sized candidate itemsets as soon as possible, so that their supports can be counted early. With Apriori_Gen, unfortunately, the only piece of information that is useful for generating new candidate itemsets during the nth iteration is the size (n-1) large itemsets, L_{n-1}. Information from other L_i's (i<n-1) and Ci's (I• = n-1) are not useful because it is already subsumed in L_{n-1}. As a result, we cannot generate candidate itemsets large than n (Han and Kambe, 2001).

Now, suppose one is given a set of suggested large itemsets S. We can use this set as additional information to generate large itemsets early. During the first iteration, besides counting the supports of size-1 itemsets, we can also count the supports of the elements in S as well. After the first iteration, we thus have a (partial) set of large itemsets of various sizes, L. These itemsets include all large singletons, as well as those itemsets in S that are verified large. We can now follow the principle of apriori to generate candidate itemsets based on L. The only problem remains is how to generalize Apriori_Gen to compute a set of multiple-sized candidate itemsets from a set of multiple-sized large itemsets (L) efficiently.

Explanation

The problem of mining association rules could be decomposed into two sub problems:

- Find out all large itemsets and their support counts. A large itemset is a set of items which are contained in a sufficiently large number of transactions, with respect to a support threshold minimum support.
- From the set of large itemsets found, find out all the association rules (Agrawal and Shrikanth, 1994) that have a confidence value exceeding a confidence threshold minimum confidence.

Since the solution of the second subproblem is straightforward, here we are concentrating only on the first subproblem. Most of the algorithms devised to find large itemsets are based on the apriori algorithm (Ashok Savasere *et al*., 1995). The apriori algorithm finds out the large itemsets iteratively. In the ith iteration, Apriori generates a number of candidate itemsets of size i. Apriori then scans the database to find the support count of each candidate itemset. Itemsets whose support counts are smaller than the minimum support are discarded apriori terminates when no more candidate set can be generated.

The key of the Apriori algorithm is the apriori_Gen function which wisely generates only those candidate itemsets that have the potential of being large. However, at each database scan, only candidate itemsets of the same size are generated. Consequently, the number of database scans required by Apriori-based algorithms depends on the size of the largest large itemsets. As an example, if a database contains a size-10 large itemset, then at least 10 passes over the database are required. For large databases containing gigabytes of transactions, the I/O cost is dauntingly big.

The goal of this research is to analyze and to improve the I/O requirement of the apriori algorithm. In particular, we generalize apriori_Gen to a new candidates set generation algorithm, based on lattice theory.

The main idea is to relax Apriori's restriction that candidate itemsets generation must start from size one and that at each database scan, only candidate itemsets of the same size are generated. Instead, new lattice based algorithm takes a (partial) set of multiple sized large itemsets as a hint to generate a set of multiple-sized candidate itemsets. This approach allows us to take advantage of an educated guess, or a suggestion, of a set of large itemsets.

(*Example 1*)

As a simple example, suppose the itemset {a,b,c,d} and all its subsets are large in a database. Apriori will require four passes over the data to generate the itemset {a,b,c,d}. However, if one already know that the itemsets {a,b,c}, {a,b,d} and {c,d} were large, then we can use this piece of information to help us generate the itemset {a,b,c,d} early.

This new candidate set generation algorithm adopts the strategy of generating large candidate itemsets as soon as possible, using a suggested set of large itemsets as a hint. However, instead of using all large itemsets known so far to generate a batch of candidate itemsets, new algorithm uses only the maximal large itemsets for candidate generation (Mazlack, 2005; Shankar Pal *et al*., 2005). An (already known) large itemset is maximal if it is not a proper subset of another (already known) large itemset.

The advantage of new algorithm over the simple strategy are two fold.

- The set of maximal large itemsets is much smaller than the set of all large itemsets. This makes the candidate generation procedure much more efficient.
- No redundant candidate itemsets are generated.

In this study we present the new candidate set generation function New-Gen and the algorithm which uses New_Gen to discover large itemsets in a database.
We address the following questions:

- Will new algorithm generate more candidate itemsets than Apriori?
  New algorithm does not generate more candidate itemsets than Apriori does. It only generates them earlier and in fewer passes.
- Where can I find the set of suggested large itemsets for new algorithm?
  These suggested itemset can be obtained in different ways, for example a super market may keep a database of transactions of the past twelve months. At the end of every month, a new set of transactions are added to the database and transactions that are more than a month old are removed. To mine the association rules of the updated database, we can use the set of large itemsets found from the old database as the suggested set and apply the new algorithm. Alternatively, one can take a sample of the database, apply Apriori on the sample to get a rough estimate of the large itemsets (Dilly, 1995) and use it as the suggested set for new algorithm.
- How much performance gain can new algorithm achieve over Apriori?
  The number of I/O passes saved by new algorithm over Apriori depended on the accuracy of the suggested large itemsets. As an extreme case, if the suggested itemsets cover all the large itemsets in the database, then new algorithm requires only 2 database scans.

The Apriori Algorithm

Conceptually, finding large itemsets from database transactions involves keeping a count for every itemset. However, since the number of possible itemsets is exponential to the number of items in the database, it is impractical to count every subset we encounter in the database transaction (Ng and Han, 2002). The Apriori algorithm tackles this combinatorial explosion problem by using an iterative approach to count the itemsets. First, itemsets containing only one item (1-itemsets or singletons) are counted and the set of large 1-itemsets (L_1) is found. Then, a set of possible large 2 itemsets is generated using the function Apriori_Gen. Since for an itemset of size n to be large, all its size n-1 subsets must also be large, Apriori_Gen only generates those 2-itemsets whose size one subsets are all in L^1. This set is the candidate set of size-2 itemsets, C_2. For example, if L_1= {{c}, {e}, {g}, {j}}, C_2 would be {{c,e}, {c,g}, {c,j}, {e.g}, {e,j},{g,i}}.

After C_2 is generated the database is scanned once again to determine the support counts of the itemsets in C_2. Those with their support counts larger than the support threshold are put into the set of size–2 large itemsets, L2. L2 is then used to generate C3 in a similar manner: all size-2 subsets of every element in C_3 must be in L_2. So, if L2 in our previous example turns out to be {{c,e},{c,g},{c,j},{g,j}}, C_3 would be {{c,g,j}}. Note that the itemset {c,e,j} is not generated because not all of its size two subsets are in L_2. Again, the database is scanned once more to find L3 from C_3. This candidate set generation-verification process is continued until no more candidate itemsets can be generated. Finally the set of large itemsets is equal to the union of all the L_i's.

Algorithm for Finding Large Itemsets

```
Algorithm
1.  Function FindLarge (suggested Large)
2.  Set of large itemsets with associated counters, MaxLargeItemsets: = 0
3.  Iteration: =1
4.  CandidateSet: = (all 1-itemsets) ? (Us^a suggestedLarge ?s)
5.  Scan database and count occurrence frequency of every set in CandidateSet
6.  NewLargeItemsets: = Large itemsets in CandidateSet
7.  While (NewlargeItemsets • 0)
8.  Iteration: = Iteration +1
9.  MaxLargeItemsets: = Max (MaxLargeItemsets ? NewLargeItemsets)
10. Candidate Set: = New_Gen (MaxLargeItemsets, Iteration)
11. Count Occurrence frequency of every set in CandidateSet
12. NewLargeItemsets: = Large itemsets in Candidateset
13. end While
14. Return all subsets of elements in MaxLargeItemsets
```

Fig. 1: Algorithm for finding large itemsets

New_Gen (Helper Function)

We generalize Apriori_Gen to a new candidate set generation function called New_Gen based on Lattice Theory (Liu, 1985; Zaki, 2005). The main idea is to generate candidate itemsets of bigger sizes early using information provided by a set of suggested large itemsets. Before we describe our algorithm formally, let us first illustrate the idea with an example.

(*Example 2*)

Suppose we have a database whose large itemsets are {a,b,c,d,e,f}, {d,e,f,g}, {e,f,g,h}, {h,i,j} and all their subsets. Assume that the sets {a,b,d,e,f}, {e,f,g,h} and {d,g} are suggested large. During the first iteration, we count the supports of the singletons as well as those of the suggested itemsets. Assume that the suggested itemsets are verified large in the first iteration. In the second iteration, since {a,b,d,e,f} is large, we know that its subset {d,e} is also large. Similarly, we can infer from {e,f,g,h} that {e.g.,} is also large. Since {d,g} is also large, we can generate the candidate itemset {d,e,g} and start counting it. Similarly, the candidate itemset {d,f,g} can also be generated this way. Therefore, we have generated some size-3 candidate itemsets before we find out all size–two large itemset.

Our Algorithm Is Similar to Apriori Except That:

- It takes a set of suggested itemsets, hint as input and counts their supports during the first database scan.
- It replaces the Apriori_Gen function by New_Gen (helper function) which takes the set of maximal large itemsets (MaxLargeItemsets) as input.

The algorithm consists of two stages. The first stage consists of a single dababase scan (Fig. 1, lines 4-6). Singletons, as well as the suggested large itemsets and their subsets are counted. Any itemsets found to be large at this first stage is put into the set of newly found large itemsets (NewLargeItemsets).

The second stage of Proposed Algorithm is iterative. The iteration continues until no more new large itemsets can be found. At each iteration, algorithm generates a set of candidate itemsets based on the large itemsets it has already discovered.

The set of maximal large itemsets found is then passed to helper function New_Gen to generate candidate itemsets (Fig. 1, line 10). The crux is how to do the generation based on the compressed maximals only. We remark that the Apriori algorithm with Apriori_Gen is in fact displaying a special case of candidate generation with canonicalization. Recall that in Apriori, At the beginning of the n-th iteration, the set of large itemsets already discovered is L_{n-1}. Canonicalizing (Macqueen, 1967) the set gives max () = L_{n-1}. Interestingly, Apriori_Gen generates the candidate set C_n based solely on L_{n-1}.

The function New_Gen is shown in Fig. 2. By simple induction, one can show that at the beginning of the nth iteration of Proposed Algorithm, all large itemsets whose sizes are smaller than n are known. Hence, when New_Gen (helper function) is called at the nth iteration of Proposed Algorithm, it only generates candidate itemsets that are of size n or large.

New_Gen

```
1.  Function New_Gen (MaxLargeItemset,n)
2.  CandidateSet: = 0
3.  for each i,j ª MaxLargeItemsets, i • j,
4.  if (/ (i • j)/ >= n-2 )
        ⎧ | a1, a2, ......an-2 ∈ i∩j,
        ⎪ | $a_{n-1}$ ∈ i-j, $a_n$ ∈ j-i,
        ⎨ | ∀i ∈ [1, n] ∃t ∈ Max Large Itmesets s.t.     ⎬
        ⎪ | {a1, a2,.....$a_n$} - {$a_i$} ⊆ t
        ⎩ | ∃u ∈ Max Large Itmesets s.t.
            { a1, a2,......$a_n$}
5.  New Candidates: =
6.  CandidateSet: = CandidateSet U New Candidates
7.  End if
8.  End for each
9.  return CandidateSet
```

Fig. 2: Helper function

In helper function New_Gen, essentially given a target candidate itemset size n, we examines every pair of maximal large itemsets i,j whose intersection is at least n-2 in size (Fig. 2, lines 3-4). It then generates candidate itemsets of size n by picking n-2 items from the intersection between i and j, one item from the set difference i-j and another item from j-i. A candidate itemset so generated is then checked to see if all of its size (n-1) subsets are already known to be large. (That is, if all of them are subsets of certain maximal large itemsets) if not, the candidate itemset is discarded. The candidate itemsets so generated are collected in the set NewCandidates as shown in line 5.

(*Example 3*)

Let us consider the example, Suppose we are given that the itemsets {a, b, c}, {a, b, d} and {c, d,} are maximal large itemsets. To generate size 3 candidate itemsets, New_Gen needs to consider pairs whose intersection contain at least 3-2 = 1 item {e.g., a, b, c} and {c, d}}. Let i = {a, b, c} and j = (c, d), we have i • j = {c}, i -j = {a, b}, and j-i = {d}. The itemsets generated based on this i, j pair are {c, a, d} and {c,b,d}. Since all of their size-2 subsets are subsets of some maximal itemsets, they are included in the candidates set. Now, suppose that the support counts of {a, c, d} and {b, c, d} are larger than the support threshold, then in the second iteration, the maximal large itemsets are {a, b, c}, {a, b, d}, {a, c, d} and {b, c, d}. One can check that if we take any two of them to generate size-4 candidate itemsets, the set {a,b,c,d,} will be generated.

Analysis of Proposed Algorithm

In this subsection we have proved some lemmas related to proposed algorithm which shows that our algorithm for generating candidate item sets is correct.

Here, we are using symbol S for the representation of suggested large item set.

Lemma 1

Given a set of suggested large itemsets S, $C_{Apriori}$ • C_{New_Gen} U (Us^{a}S)

Since Proposed Algorithm (which uses New_Gen) counts the supports of all itemsets in C_{New_Gen} U (U_s^a S), Lemma 1 says that any candidate itemset that is generated by apriori will have its support counted by Proposed Algorithm. Hence, if Apriori finds out all large itemsets in the database, Proposed Algorithm does too. In other words, Proposed Algorithm is correct.

Lemma 2

C_{new_Gen} • $C_{Apriori}$.

Lemma 2 says that the set of candidate itemsets generated by New_Gen is a subset of that generated by Apriori._Gen. New_Gen thus does not generate any unnecessary candidate itemsets and waste resources in counting bogus ones. However recall that Proposed Algorithm counts the supports of the suggested large itemsets in the first database scan for verification. Therefore, if the suggested set contains itemsets that are actually small, Proposed Algorithm will count their supports superfluously. Fortunately, the number of large itemsets in a database is usually order of magnitude fewer than the number of candidate itemsets. The extra support counting is thus insignificant compared with support counting of all the candidate itemsets. Proposed Algorithm using New_Gen thus requires similar counting effort as Apriori does.

Lemma 3

if S = 0 then $C_{New\_Gen} = C_{Apriori}$.

Lemma 3 says that without suggested large itemsets, Proposed Algorithm reduces to Apriori. In particular, they generate exactly the same set of candidate itemsets.

Experimental Results

In this subsection we present a brief overview of the experimental results obtained by using proposed algorithm. All the experiments were performed on 933 MHz Pentium-III PC with 256 MB RAM. We perform the experiments on market basket data synthetically generated for the experiment (Fig. 3). The basic interface and operational module is coded in VB 6.0.

S. No.	Items	Support count
1.	Scanner	40
2.	Computer	30
3.	Printer	35
4.	FlashDrive	38
5.	UPS	38
6.	Computer UPS	23
7.	Printer UPS	26
8.	Computer Printer	22
9.	Computer Printer UPS	19
10.	Computer FlashDrive	22
11.	FlashDrive Printer	22
12.	FlashDrive UPS	27
13.	Computer FlashDrive Printer	16
14.	Computer FlashDrive UPS	17
15.	FlashDrive Printer UPS	19
16.	Computer FlashDrive Printer UPS	14

Fig. 3: Frequent patterns generated

Conclusions

In this research we described a new algorithm for Frequent Pattern Discovery for a transaction database, which takes a set of multiple-sized large itemsets to generate multiple-sized candidate itemsets. Given a reasonably accurate hint of suggested large itemsets, we can generate and process candidate itemsets of big-sized early.

References

Agrawal, R., T. Imielinski and A. Swami, 1993. Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIG- MOD International Conference on Management of Data, pp: 207-216, Washington, DC., May 26-28 1993.

Agrawal, R. and R. Shrikanth, 1994. Fast Algorithm for mining association rules. Proceedings of VLDB Conference, Santigo, Chile, pp: 487-449.

Dilly, R., 1995. Data Mining: An Introduction. Parallel Computer Center Queens, University Belfast. 1995.

Frawley, W.J., G. Piatetsky-Shapiro and C.J. Matheus, 1991. Knowledge Discovery in Databases: An Overview of Knowledge Discovery in Databases. AAI/MIT Press, 1991.

Han, J. and M. Kambe, 2001. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, Elsevier India.

Liu, C. L., 1985. Elements of Discrete Mathematics. Tata McGraw Hill, New Delhi, 2nd Ed., 1985.

MacQueen, J.B., 1967. Some methods for classification and analysis of multivariate observations. In Proceedings of the 5th Symposium on mathematical Statistics and Probability, Berkelely, CA, 1: 281-297.

Mazlack, L.J., 2005.Data Mining Fuzzy Nested Granular causal Complexes. Applied Artificial Intelligence Laboratory, University of Cincinnati, BISC, 2005.

Ng, R.T. and Han, 2002. CLARANS: A method for clustering objects for spatial data mining. In Proceedings of the IEEE Transactions on Knowledge and Data Engineering, 14: 1003-1016.

Savasere, A., E. Omiecinski and S. Navathe, 1995. An Efficient Algorithm for Mining Association Rules in Large Databases. Proceedingds of the 21st VLDB conference Zurich, Switzerland.

Shankar, K.P., B. Umashankar and P. Mitra, 2005. Granular computing, rough entropy and object extraction. Department of Computer Science and Engineering, IIT, Kanpur, india, Elsevier, July 28, 2005.

Srikant, R., Q. Vu and R. Agrawal, 1997. Mining Association Rules with Item Constraints. Proc. 3rd International Conference Knowledge Discovery and Data mining, KDD, 1997.

Zaki, M.J., Member, IEEE and Ching-Jui Hsiao, 2005. Efficient Algorithm for Mining Closed Itemsets and Their Lattice Structure. Vol. 17, No. 4, April 2005.