Top-Down Mining of Frequent Patterns from Very High Dimensional Data

Hongyan Liu[†] Jiawei Han[‡] Dong Xin[‡] Zheng Shao[‡]
[†]*Department of Management Science and Engineering, Tsinghua University*
hyliu@tsinghua.edu.cn
[‡]*Department of Computer Science, University of Illinois at Urbana-Champaign*
{hanj, dongxin, zshao1}@uiuc.edu

1. Introduction

Many real world applications deal with transactional data, characterized by a huge number of transactions (tuples) with a small number of dimensions (attributes). However, there are some other applications that involve rather high dimensional data with a small number of tuples. Examples of such applications include bioinformatics, survey-based statistical analysis, text processing, and so on. High dimensional data pose great challenges to most existing data mining algorithms. Although there are numerous algorithms dealing with transactional data sets, there are few algorithms oriented to very high dimensional data sets with a relatively small number of tuples. Taking frequent pattern mining [1, 2, 3, 4] as an example, most of the existing algorithms are column (i.e., item) enumeration-based algorithms, which take the combinations of columns (items) as search space. Due to the exponential number of column combinations, this method is not suitable for very high dimensional data.

To address the above issue, a row enumeration-based method [5] was proposed recently to handle this kind of very high dimensional data. Based on this row-enumeration philosophy, several algorithms are developed to find frequent patterns or classification rules. By searching through the row enumeration space instead of column enumeration space, these algorithms outperform their counterparts in very high dimensional data. However, since these algorithms exploit a bottom-up search strategy to check row combinations from the smallest to the largest, they cannot make full use of the minimum support threshold to prune search space. As a result, experiments show that such a method often cannot run to completion in a reasonable time for large *microarray data*, and it sometimes runs out of memory before completion. To solve these problems, we propose a new *top-down* search strategy for row enumeration-based mining algorithm. To show its effectiveness, we design an algorithm, called *TD-Close*, for top-down row-enumeration mining of a complete set of frequent closed itemsets, and compare it with row enumeration-based bottom-up mining algorithms, *Carpenter* [5].

2. Problem formulation

Let T be a discretized data table (or data set), composed of a set of rows, $\mathcal{S} = \{r_1, r_2, \ldots, r_n,\}$, where r_i (i = 1, …, n) is called a row ID, or *rid* in short. Each row corresponds to a sample consisting of *k* discrete values or intervals. For simplicity, we call each of this kind of values or intervals an *item*. We call a set of *rids* a *rowset*, and a rowset having *k rids* a *k-rowset*. Likewise, we call a set of items an *itemset*. A *k-rowset* is called *large* if *k* is no less than a user-specified threshold which is called *minimum size threshold*.

Let TT be the transposed table of T, in which each row corresponds to an item i_j and consists of a set of *rids* which contain i_j in T. For clarity, we call each row of TT a tuple.

Table 2.1 An example table T

r_i	A	B	C	D
1	a_1	b_1	c_1	d_1
2	a_1	b_1	c_2	d_2
3	a_1	b_1	c_1	d_2
4	a_2	b_1	c_2	d_2
5	a_2	b_2	c_2	d_3

Table 2.1 shows an example table T with 4 attributes (columns): A, B, C and D. The corresponding transposed table TT is shown in Table 2.2. For simplicity, we use number i (i = 1, 2, …, n) instead of r_i to represent each *rid*.

Table 2.2 Transposed table TT of T

itemset	rowset
a_1	1, 2, 3
a_2	4, 5
b_1	1, 2, 3, 4
c_1	1, 3
c_2	2, 4, 5
d_2	2, 3, 4

Originally, we want to find all of the frequent closed itemsets which satisfy the minimum support threshold *minsup* from table T. After transposing T to the transposed table TT, the constraint *minimum support threshold for itemsets* becomes *the minimum size threshold for rowsets*. Therefore, the mining task becomes finding all of the large closed rowsets which satisfy minimum size threshold *minsup* from table TT.

3. Bottom-up vs. top-down search strategy

Existing column-enumeration based mining algorithms use bottom-up search strategy. Taking Table 2.1 as example, Figure 3.1 shows a row enumeration tree that uses the bottom-up search strategy. In this Figure, each node represents a rowset. By bottom-up searching we mean that along every search path, the row enumeration space is searched from small rowsets to large ones. Both depth-first and breadth-first search of the tree shown in Figure 3.1 belong to this search strategy. For example, by breadth-first search style, 1-rowsets {1}, {2}, {3}, {4} and {5} will be checked first, and then 2-rowsets such as {12}, {13} and {14} will be checked, and so on.

Our mining task is to discover all of the large closed rowsets. So the main constraint is the size of rowset. Since it is monotonic in terms of the bottom-up search order, it is hard to prune the row enumeration search space early. For example, suppose *minsup* is set to 3, although obviously all of the nodes in the first two levels from the root cannot satisfy this constraint, these nodes still need to be checked. In addition, the memory cost for this kind of bottom-up search is also big. Take *Carpenter* as an example. Similar to several other column-enumeration based algorithms, *Carpenter* uses a pointer list to point to each tuple belonging to an *x-conditional* transposed table. For a table with *n* rows, the maximum number of different levels of pointer lists needed to remain in memory is *n*, although among them the first (*minsup* – 1) levels of pointer lists will not contribute to the final result. As a result, as the *minsup* increases, the time needed to complete the mining process cannot decrease rapidly. This limits the application of this kind of algorithms to real situations.

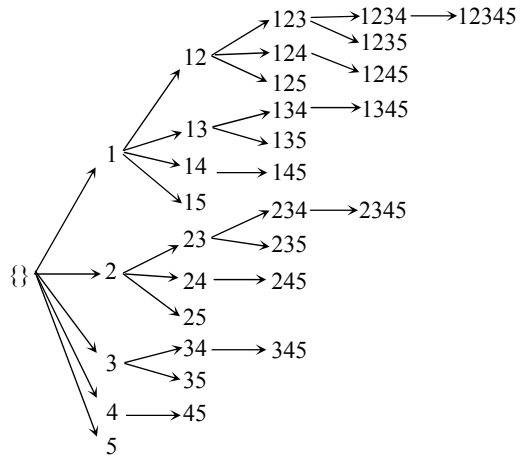

Figure 3.1 Bottom-up row enumeration tree

In order to solve the problem of bottom-up searching strategy for transposed data set, we notice that almost all of the frequent pattern mining algorithms dealing with the data set without transposition use the anti-monotonicity property of *minsup* to speed up the mining process. For transposed data set, the *minsup* constraint maps to the size of rowset. In order to stop further search when *minsup* is not satisfied, we propose to exploit a top-down search strategy instead of the bottom-up one. Contrary to the bottom-up search strategy, the top-down searching strategy means that along each search path the rowsets are checked from large to small ones. To do so, we need to design a new row enumeration tree following this strategy.

There are two concerns for designing such a tree. First, it should have the following property. With this kind of row enumeration tree, given a table T with *n* rows and the user-specified *minsup*, we can stop further search of it at level (*n* – *minsup*) for mining frequent itemsets. We regard the level of the root node of this tree is 0. Second, in order to enhance the efficiency, it would be better if we could develop a tree such that the transposed table TT can be mined by a divide-and-conquer style. Satisfying these two requirements, we propose a novel row enumeration tree, called *top-down row enumeration tree*, to guide the mining of the transposed table TT.

In order to find closed frequent itemsets, we need a method to check whether an itemset is closed. Existing methods usually need to scan either the mining data set or the result set, which makes them not very efficient. To solve this problem, we develop a new method, called *closeness-checking*. Unlike other

existing methods, it collects enough information for *closeness-checking* while searching the row enumeration space. So whenever a frequent itemset is found, it does not need to scan the mining data set again, and the result set does not need to be checked either. Also, it is easy to be integrated with the top-down search process. In addition, in order to further speed up the searching process, several pruning strategies are also developed to prune rowsets which cannot contribute to the result.

4. Experimental study

Based on the top-down row enumeration tree and the *closeness-checking* method as well as several pruning strategies, we design an algorithm, called ***TD-Close,*** to mine all of the frequent closed patterns from table T. In this algorithm, the top-down enumeration tree is traversed by depth-first order. Whenever a node of this tree is visited, the largest rowset in the table corresponding to this node will be checked. Then this table is divided into several sub-tables, each of which will be processed recursively in the same way.

We conducted experiments for both synthetic data sets and real *microarray data* sets. *Carpenter* has already shown that on such data sets it has much better performance than the column enumeration-based algorithms, such as CLOSET [2] and CHARM [3]. So we only compare our algorithm with *Carpenter* [5]. All experiments were performed on a PC with a Pentium-4 1.5 Ghz CPU, 1GB RAM, and 30GB hard disk.

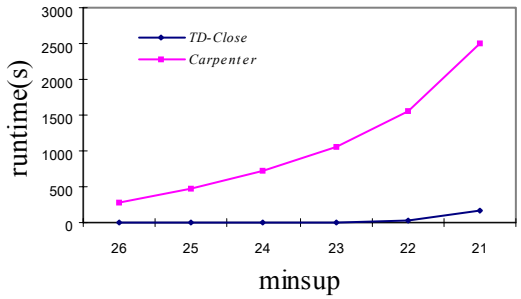

Figure 4.1 Runtime for Lung Cancer

Figure 4.1 is the result of the experiments for a real *microarray* data set: Lung Cancer. Originally, it has 12533 dimensions and 32 rows for training data set. After discretization using entropy based method [6], it has 2173 dimensions and 32 rows. Figure 4.1 shows the change of runtime as *minsup* decreases for algorithm *TD-Close* and *Carpenter*. Each runtime plotted in this figure includes both computation time and I/O time.

We can see from Figure 4.1 that as *minsup* decreases, the runtime of each algorithm increases, and *TD-Close* is much faster than *Carpenter*, especially when *minsup* becomes relatively small. Same results are also obtained for other data sets.

5. Conclusions

In this paper we propose a top-down search strategy for mining interesting patterns from very high dimensional data with a small number of rows. Unlike existing algorithms, such as *Carpenter* and several other algorithms of similar nature, we develop a top-down row enumeration method to search the row enumeration space, which makes the pruning power of minimum support threshold (*minsup*) stronger than using bottom-up search style. Integrating with this search method, an effective and efficient *closeness-checking* method is also proposed. Moreover, several pruning strategies are also developed to speed up searching. Finally, a new algorithm, *TD-Close*, is designed and implemented to show the effectiveness of these methods. Both analysis and experimental results show that this algorithm outperforms *Carpenter* for very high dimensional data.

6. Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant No. 70471006 and 70321001, and by the U.S. National Science Foundation NSF IIS-02-09199 and IIS-03-08215.

7. References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases* (VLDB'94), pp. 487–499, Sept. 1994.

[2] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery* (DMKD'00), pp. 11-20, Dallas, TX, May 2000.

[3] M. Zaki and C. Hsiao. Charm: An efficient algorithm for closed association rule mining. In *Proc. of 2002 SIAM Data Mining Conf.*, 2002.

[4] G. Grahne and J. Zhu. Efficiently Using Prefix-trees in Mining Frequent Itemsets. In *Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations* Melbourne, Florida, Nov., 2003.

[5] F. Pan, G. Cong, A. K. H. Tung, J. Yang, and M. J. Zaki. CARPENTER: Finding closed patterns in long biological datasets. In *Proc. 2003 ACM SIGKDD Int. Conf. On Knowledge Discovery and Data Mining*, 2003.

[6] http://www.sgi.com/tech/mlc/.