Data Access Paths for Frequent Itemsets Discovery

Marek Wojciechowski, Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
{marekw, mzakrz}@cs.put.poznan.pl

Abstract. Many frequent itemset discovery algorithms have been proposed in the area of data mining research. The algorithms exhibit significant computational complexity, resulting in long processing times. Their performance is also dependent on source data characteristics. We argue that users should not be responsible for choosing the most efficient algorithm to solve a particular data mining problem. Instead, a data mining query optimizer should follow the cost-based optimization rules to select the appropriate method to solve the user's problem. The optimizer should consider alternative data mining algorithms as well as alternative data access paths. In this paper, we use the concept of materialized views to describe possible data access paths for frequent itemset discovery.

1 Introduction

Data mining is a relatively new database research field, which focuses on algorithms and methods for discovering interesting patterns in large databases [6]. An interesting pattern is typically a description of strong correlation between attributes of a data object. Many data mining methods developed in the area have proved to be useful in decision support applications: association discovery, sequential pattern discovery, classifier discovery, clustering, etc. One of the most popular data mining methods is frequent itemset discovery, which aims at finding the most frequent subsets of database itemsets. Unfortunately, frequent itemset discovery algorithms exhibit significant computational complexity, resulting in long processing times. The computational cost of the algorithms is usually influenced by a need to perform multiple passes over the source data and to perform a significant amount of in-memory operations. Moreover, the algorithms' performance is also dependent on source data characteristics - for example, some algorithms perform better for long patterns, some algorithms benefit from uniform data distribution, etc.

Users perceive data mining as an interactive and iterative process of advanced querying: users specify requests to discover a specific class of patterns, and a data mining system returns the discovered patterns. A user interacting with a data mining system has to specify several constraints on patterns to be discovered. However, usually it is not trivial to find a set of constraints leading to the satisfying set of patterns. Thus, users are very likely to execute a series of similar data mining queries before they find what they need.

In the scope of our data mining research we follow the idea of integrating data mining mechanisms into database management systems (DBMSs). We argue that DBMS functionality should be extended to completely support data mining applications. This integration involves the following aspects: (1) query language extension to allow users to formulate their specific data mining problems, (2) logical and physical structure extensions to permanently store discovered patterns, and (3) query optimizer extension to generate alternative query execution plans and to choose the best one. In this paper we show that a DBMS query optimizer can consider various data access paths for alternative data mining algorithms. We present our research in the context of frequent itemsets discovery.

1.1 Preliminaries

Let L be a set of items. An itemset I is a subset of L. A database D is a set of itemsets. Consider two itemsets I_1 and I_2. We say that I_1 supports I_2 if $I_1 \subseteq I_2$. A frequent itemset X is an itemset which is supported by more than a given number of itemsets in D. Given a user-defined support threshold *minsup*, the problem of association discovery is to find all frequent itemsets in D.

1.2 Data Mining Query Processing

We assume the following model of user interaction with a DBMS extended with data mining functions. A user defines his/her data mining problem in the form of a *data mining query* (DMQ). The data mining query describes: (1) a data source, (2) a support threshold, (3) filtering predicates to narrow source data set, and (4) filtering predicates to narrow the set of discovered frequent itemsets. For example, a DMQ can state that a user is interested in "processing last month's sale transactions from the SALES table to find all frequent itemsets having support at least 2% and containing more than three items". Using the SQL language extension we introduced in [9], the presented example DMQ takes the form:

```
mine itemset from (select set(product) from sales
      where date between '1-06-01' and '30-06-01' group by trans_id)
where support(itemset)>=0.02 and length(itemset)>3
```

Next, the DMQ is sent to the DBMS. The DBMS has to compile the query into a microprogram and then to execute the microprogram. We will show that a DMQ can be compiled into many alternative microprograms and that a query optimizer is needed to efficiently choose the best one.

2 Data Access Paths

A DMQ can be executed using different data access methods:
1. A traditional data mining algorithm can be used to discover interesting patterns directly from the original database. We will refer to this method as to *Full Table Scan*.

2. A materialized view of the original database can be used by a data mining algorithm instead of the original database itself. A materialized view can introduce some form of data reduction (lossless or lossy), thus reducing I/O activity of a data mining algorithm. We will refer to this method as to *Materialized View Scan*.
3. Existing data mining results can be used to execute a new DMQ. Data mining results can be stored in a form of a data mining view, therefore we will refer to this method as to *Materialized Data Mining View Scan*.

2.1 Full Table Scan

The Full Table Scan method involves regular data mining algorithms like Apriori to discover all interesting patterns by counting their occurrences in the original database. Due to the large size of the original database, the performance of the algorithms is relatively bad. Moreover, many algorithms perform good only in certain conditions related to: data values distribution, support threshold value, available memory, etc. In a typical scenario, the user is responsible for selecting an appropriate (in terms of performance) data mining algorithm.

2.2 Materialized View Scan

Weak performance of many of the regular data mining algorithms is caused by the need to make multiple passes over the large original database. If we could reduce or compress the original database, the passes would be less costly since they would use less I/O operations. Databases already offer a data structure that can be efficiently used to reduce the original database: *materialized views* (MV). MV is a database view, having its contents permanently stored in a database. MVs are normally created by users to support data-intensive operations. We propose to use MVs to support data mining algorithms.

Since not every MV guarantees a correct data mining result (compared to a full table scan performed on the original database), we define the following types of MVs and their use for data mining.

Definition (Strong pattern preserving view). Given the original database *D*, the *minsup* threshold and the view *V*, we say the *V* is a *strong pattern preserving view* if for each pattern *p* having $sup(p,D)>minsup$, we have $sup(p,V)=sup(p,D)$.

Definition (Weak pattern preserving view). Given the original database *D*, the *minsup* threshold and the view *V*, we say the *V* is a *strong pattern preserving view* if for each pattern *p* having $sup(p,D)>minsup$, we have $sup(p,V)>=sup(p,D)$.

According to the above definitions, if we are given a MV which is strong pattern preserving, we can use it as an alternative data source for a data mining algorithm. If we are given a MV which is a weak pattern preserving, we can use it to discover *potentially* frequent patterns, but then we have to use the original database to make the final verification of their support values.

Example 1. Given is the database table ISETS and the materialized view ISETS_MV2 created by means of the following SQL statement.

```
create materialized view isets_mv2 as
  select signature(set,10) from isets;
```

where the user-defined function *signature()* computes the following binary signature for an itemset of integers:

$$signature(\{x_1, x_2, .., x_k\}, N) = 2^{x_1 \bmod \mathrm{N}} \; AND \; 2^{x_2 \bmod \mathrm{N}} ... AND \; 2^{x_k \bmod \mathrm{N}}$$

where *AND* is a bit-wise *and* operator.

```
ISETS                           ISETS_MV2

sid  set                        sid  signature(set,10)
---  --------------             ---  --------------------
  1  {5, 7, 11, 22}              1    0110010100
  2  {4, 5, 6, 17}               2    0000111100
  3  {7, 22}                     3    0010000100
```

For the materialized view ISETS_MV2, we can intuitively define the support measure as the percentage of signatures that have their bits set to '1' on at least the same positions as the signature for the counted itemset.

According to our definitions, the view ISETS_MV2 is a weak pattern preserving view (notice that e.g. sup({5, 17}, V) = 2 while sup({5, 17}, D) = 1). It can be used by a data mining algorithm to find a superset of the actual result, but the original table ISETS must be also used to perform the final verification.

2.3 Materialized Data Mining View Scan

Since usually data mining users must execute series of similar queries before they get satisfying results, it can be helpful to exploit materialized results of previous queries when answering a new one. We use the term *materialized data mining view* to refer to *intentionally gathered* and *permanently stored* results of a DMQ.

Since not every materialized data mining view guarantees a correct data mining result (compared to a full table scan performed on the original database), we define, according to [3], the following relations between data mining queries and materialized data mining views:

1. A materialized data mining view $MDMV_1$ *includes* a data mining query DMQ_1, if for all data sets, each frequent itemset in the result of DMQ_2 is also contained in $MDMQ_1$ with the same support value. According to our previous definitions, in this case $MDMV_1$ is a *strong pattern preserving view*.
2. A materialized data mining view $MDMV_1$ *dominates* a data mining query DMQ_1, if for all data sets, each frequent itemset in the result of DMQ_1 is also contained in $MDMQ_1$, and for a frequent itemset returned by both DMQ_1 and $MDMV_1$, its support value evaluated by $MDMV_1$ is not less than in case of DMQ_1. According to our previous definitions, in this case $MDMV_1$ is a *weak pattern preserving view*.

If for a given DMQ, results of a DMQ including or dominating it are available, the DMQ can be answered without running a costly mining algorithm on the original database. In case of inclusion, one scan of the materialized DMQ result is necessary to filter out frequent itemsets that do not satisfy constraints of the included query. In case of dominance, one verifying scan of the source data set is necessary to evaluate the support values of materialized frequent itemsets (filtering out the frequent itemsets that do not satisfy constraints of the dominated query is also required).

Example 2. Given the ISETS3 table, a user has issued a DMQ to analyze only the rows 1,2,3,4 to discover all frequent patterns having their support values equal to at least 30%. The results of the DMQ have been permanently stored in the database in the form of the materialized data mining view ISETS_DMV2, created by means of the following statement.

```
create materialized view isets_dmv2 as
  mine itemset from (select set from isets2 where sid in (1,2,3,4))
  where support(set)>=0.3
```

ISETS3

sid	*set*
1	*5, 6, 7, 22*
2	*5, 6, 17*
3	*7, 22*
4	*2, 5, 6*
~~*5*~~	~~*2, 6, 22*~~
~~*6*~~	~~*6, 22*~~

ISETS_DMV2

itemset (support)
{5}(0.75)
{6}(0.75)
{7}(0.5)
{22}(0.5)
{5,6}(0.75)
{7,22}(0.5)

Using the above results we can answer a DMQ over the whole database table ISETS3. Assume a user issued the following DMQ.

```
mine itemset from (select set from isets2 where sid in (1,2,3,4,5,6))
where support(set)>=0.3
```

Notice that the above DMQ is *dominated* by the union of the following two data mining queries (every itemset which is frequent in the whole table must also be frequent in at least one portion of it):

```
mine itemset from (select set from isets2 where sid in (1,2,3,4))
where support(set)>=0.3
union
mine itemset from (select set from isets2 where sid in (5,6))
where support(set)>=0.3
```

The above union represents a weak pattern preserving view. We can rewrite the first part of the above union to use the materialized data mining view ISETS_DMV2. The second part of the union can be evaluated using the *full table scan* method or the *materialized view scan* method (because of lack of a suitable materialized data mining view). However, since the result of the union is a superset of the actual result of the user's query, we still need to perform additional support evaluation and final filtering.

We use a traditional data mining algorithm to discover frequent itemsets in the remaining part of the original database:

sid	*set*
5	*2, 6, 22*
6	*6, 22*

Frequent patterns minsup=0.30
{6}(1.00)
{22}(1.00)
{6,22}(1.00)

Next, we merge the two sets of frequent itemsets and count their actual support by performing another scan over the database table ISETS3. The itemsets which do not appear to be frequent are then removed from the result (not the case here).

```
ISETS3                      Frequent patterns minsup=0.30
                            -----------------------------
sid  set                    {5}(0.5)
---  ------------           {6}(0.83)
  1  5, 6, 7, 22            {7}(0.33)
  2  5, 6, 17               {22}(0.67)
  3  7, 22                  {5,6}(0.5)
  4  2, 5, 6                {6,22}(0.5)
  5  2, 6, 22               {7,22}(0.33)
  6  6, 22
```

4 Conclusions

We have showed that a frequent itemsets discovery algorithm can use one of three methods for data access. An existing materialized view or a materialized data mining view can be employed by the algorithm to reduce its I/O complexity. We have defined rules for chosing views which are applicable to a given DMQ. The data access methods were presented in the context of frequent itemsets discovery, however, they can be easily mapped to other areas of data mining, e.g. sequential pattern discovery or association rules discovery.

The choice of the most efficient method should be done by the data mining query optimizer, using a model of a data mining method as well as a statistical model of the database table. The statistical model of the database (or a part of it) can be gathered using a preliminary step of sampling. Thus, transparently to the user, every data mining query execution can use its fastest implementation.

References

1. Agrawal, R., Imielinski, T. , Swami, A.: Mining Association Rules Between Sets of Items in Large Databases. Proc. of 1993 ACM-SIGMOD Int. Conf. Management of Data (1993)
2. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. Proc. 1994 Int. Conf. Very Large Databases (1994)
3. Baralis E., Psaila G.: Incremental Refinement of Mining Queries. Proc. of the 1st DaWaK Conference (1999)
4. Elmasri R., Navathe S.B.: Fundamentals of Database Systems, Second Edition (1994)
5. Houtsma, M., Swami, A.: Set-oriented Mining for Association Rules in Relational Databases. Proc. of 1995 Int. Conf. Data Engineering (1995)
6. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
7. Mannila, H., Toivonen, H., Verkami, A.I.: Efficient Algorithms for Discovering Association Rules. Proc. AAAI'94 Workshop on KDD (1994)
8. Morzy T., Wojciechowski M., Zakrzewicz M.: Materialized Data Mining Views. Proc. of the 4th PKDD Conference (2000)
9. Morzy T., Zakrzewicz M.: SQL-like Language for Database Mining. ADBIS’97 Symposium