Access Paths for Data Mining Query Optimizer

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

Abstract

Data mining research has developed many pattern discovery algorithms dedicated to specific data and pattern characteristics. We argue that a user 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 techniques to select an appropriate algorithm to solve the user's problem. In this paper we discuss the process of data mining query optimization and we extend the list of choices the optimizer can make.

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.

Data mining research community has proposed many algorithms for discovering various types of patterns. Unfortunately, the algorithms exhibit significant computational complexity, resulting in long processing times. The computational cost of data mining 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 are dedicated to low cardinality attributes, some algorithms benefit from uniform data distribution, etc.

From a user's point of view, data mining can be seen as an interactive and iterative process of advanced querying: a user specifies a request to discover a specific class of patterns, and then a data mining system returns discovered patterns to the user. 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 context of our data mining research we follow a promising idea of integrating data mining into database management systems (DBMSs). We argue that DBMS functionality should be extended to completely support data mining applications. This 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 allow users to permanently store discovered patterns, and (3) query compiler/optimizer extension to generate alternative query execution plans and to choose the best one. In this paper we discuss the required DBMS query optimizer modifications needed to provide for data mining query optimization. Our research results are presented in the context of association 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 user-defined 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. Typically, the result of association discovery consists of the frequent itemsets and their support measures *sup(X,D)* (percentages of supporting itemsets in D).

Example. Consider the following database D:

Itemset
{1, 5, 8, 10}
{2, 8, 10, 12}
{1, 10, 11}
{3, 5, 10}

Witold Abramowicz (ed.), Business Information Systems, Proceedings of BIS 2002, Poznan, Poland

Suppose the user-defined support threshold is 30%, what means that we want to find all itemsets that are supported by at least 30% of all the database itemsets (by at least 2 itemsets). The result of the association discovery will be: {1} (support=50%), {5} (support=50%), {8} (support=50%), {10} (support=100%), {1, 10} (support=50%), {5, 10} (support=50%), {8, 10} (support=50%).

1.2 Related Work

The problem of association rule mining was introduced in [1] and the algorithm called *AIS* was presented. In [2], a more effective algorithm, called *Apriori* was presented. Since that time, a number of association rule mining algorithms have been proposed [e.g. 5,7].

In [3] incremental refinement of association rule queries was addressed. The queries were expressed using the MINE RULE operator based on a complex constraint model. The equivalence, inclusion, and dominance relationships between queries were introduced, and syntactic differences resulting in these relationships were discussed. For the assumed constraint model, algorithms for answering an association rule query using the applicable results of a previous query were presented.

In [8] the concept of materialized data mining views was introduced. Materialized data mining views allow users to materialize results of data mining queries, with an option of periodic refreshing of their contents in order to reflect the possible changes in the source data set. In the paper, general ideas of application of materialized data mining views to optimization of the processing of the incoming queries were also presented.

Cost-based query optimization is widely used in database management systems [4]. The cost-based optimizer chooses the query execution plan with the lowest estimated cost. The cost of a given execution strategy is estimated using known cost functions for the algorithms being used and certain statistics maintained for the database.

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. The data mining query describes: (1) a data source on which to perform data mining, (2) a support threshold to determine frequent itemsets, (3) filtering predicates to narrow source data set, and (4) filtering predicates to narrow the set of discovered frequent itemsets. For example, a data mining query 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 that three items". A data mining query can be expressed by means of a declarative language. Using the SQL language extension we introduced in [8], the presented example data mining query takes the following 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 data mining query is sent to the DBMS. The DBMS has to compile the query into a microprogram and then to execute the microprogram, delivering the results to the user or to the user's application. We will show that a data mining query can be compiled into a number of alternative microprograms and that a query optimizer is needed to efficiently choose the best one. First we discuss available access paths for data mining and then we explain the steps of the optimization process.

3 Access paths for data mining

A data mining query can be executed in many different ways. We classify them into three groups:

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 data mining query. 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*.

3.1 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.

3.2 Materialized View Scan

Weak performance of many of the regular data mining algorithms is caused by the need to make multiple passes over the 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. Database technology already offers a data structure that can be efficiently used to reduce the original database: *materialized views*. A materialized view (MV) is a database view, having its contents permanently stored in a database. Materialized views are normally created by users to support data-intensive operations. We propose to use materialized views to support data mining algorithms.

Since not every materialized view guarantees a correct data mining result (according to a full table scan performed on the original database), we define the following types of materialized views.

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 materialized view which is strong pattern preserving, we can use it as an alternative data source for a data mining algorithm. If we are given a materialized view 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. Consider the following illustrative examples of materialized views.

Example 1. Given is the database table ISETS and the materialized view ISETS_MV1 created by means of the following SQL statement.

```
create materialized view isets_mv1
 as
select set, count(*)
from isets
group by set
```

```
ISETS:                ISETS_MV1:

sid  set              set              count(*)
---  -------------    --------------   --------
  1  {5, 11, 18}      {5, 11, 18}       2
  2  {1, 5, 14, 18}   {1, 5, 14, 18}    1
  3  {5, 11, 18}
```

For the materialized view ISETS_MV1, we can intuitively define the support measure as the weighted function of the number of supporting itemsets multiplied by their corresponding *count(*)* values.

According to our definitions, the view ISETS_MV1 is a strong pattern preserving view (notice that e.g. sup({5, 18}, V) = 3 and sup({5, 18}, D) = 3). It can be used by a data mining algorithm instead of the original table ISETS, possibly reducing the number of required I/O operations.

Example 2. 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:

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

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 analyzed 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.

□

Materialized views can be created explicitly by users, or implicitly by a DBMS during the first step of a data mining algorithm execution. The implicit generation introduces some additional cost, but for large data mining queries this cost can be negligible.

3.3 Materialized Data Mining View Scan

Since data mining users 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 data mining query.

Since not every materialized data mining view guarantees a correct data mining result (according to a full table scan performed on the original database), we define the following relations between data mining queries.

1. Two data mining queries are *equivalent* if for all data sets they both return the same set of frequent itemsets and the support values for each frequent itemset are the same in both cases.
2. A data mining query DMQ_1 *includes* a data mining query DMQ_2 if for all data sets each frequent itemset in the result of DMQ_2 is also returned by DMQ_1 with the same support value.
3. A data mining query DMQ_1 *dominates* a data mining query DMQ_2 if for all data sets each frequent itemset in the result of DMQ_2 is also returned by DMQ_1, and for frequent itemset returned by both data mining queries its support value evaluated by DMQ_1 is not less than is case of DMQ_2.

Equivalence is a particular case of inclusion, and inclusion is a particular case of dominance. Equivalence, inclusion, and dominance meet the transitivity property.

If for a given data mining query, results of a data mining query equivalent to it, including it, or dominating it are available, the data mining query can be answered without running a costly mining algorithm. In case of equivalence no processing is necessary, since the queries have the same results. In case of inclusion, one scan of the materialized data mining query 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). Consider the following illustrative examples of materialized data mining views.

Example 1. Given the ISETS2 table, a user has issued a data mining query to discover all frequent patterns having their support values equal to at least 30%. The results of the data mining query have been permanently stored in the database in the form of the materialized data mining view ISETS_DMV1, created by means of the following statement.

```
create materialized view isets_dmv1
 as
mine itemset
from isets2
where support(set)>=0.3
```

```
ISETS2              ISETS_DMV1

sid  set            itemset (support)
---  ------------   -----------------
  1  5, 6, 7, 22    {5}(0.75)
  2  5, 6, 17       {6}(0.75)
  3  7, 22          {7}(0.5)
  4  2, 5, 6        {22}(0.5)
                    {5,6}(0.75)
                    {7,22}(0.5)
```

Using the above results we can answer an *included* data mining query simply by filtering the frequent patterns. Assume a user issued the following data mining query.

```
mine itemset
from isets2
where support(set)>=0.6
```

Using the materialized data mining view ISETS_DMV1, the above data mining query can be automatically rewritten to the form of:

```
select itemset
from isets_dmv1
where support(itemset)>=0.6
```

Finally, the result of the user query is shown below.

```
ISETS_DMV1            Result of the new data
                      mining query:

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

Example 2. Given the ISETS3 table, a user has issued a data mining query 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 data mining query 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              ISETS_DMV2

sid  set            itemset (support)
---  ------------   -----------------
  1  5, 6, 7, 22    {5}(0.75)
  2  5, 6, 17       {6}(0.75)
   3  7, 22         {7}(0.5)
   4  2, 5, 6       {22}(0.5)
   5  2, 6, 22      {5,6}(0.75)
   6  6, 22         {7,22}(0.5)
```

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

```
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 data mining query 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
```

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 must be evaluated using the *full table scan* method or the *materialized view scan* method (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         Frequent patterns minsup=0.30
---  ---------   -----------------------------
   5  2, 6, 22   {6}(1.00)
   6  6, 22      {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
```

□

Generally, materialized data mining views are created explicitly by users. However, there are applications where implicit view generation can significantly improve system performance. Assume batch environment, where a number of data mining queries are submitted by users for asynchronous execution. If some of these queries are similar, it can be helpful to organize the queries in a way that materialized results of one query are used to execute another one more efficiently. We refer to such process as to *multiple data mining query optimization*.

3.4 Architecture of the Data Mining Query Optimizer

We have described various methods to execute a data mining query. However, we do not expect users to make decisions which method to use for a particular query. Instead, following the idea of RDBMS query optimizers, we emphasize the need to develop the concept of a *data mining query optimizer*. A *data mining query optimizer* is a software component which chooses the fastest method to execute every data mining query submitted by users. Meaning of "the fastest" can be application dependent –

typically the optimization goal is to maximize throughput or to minimize response time.

In order to chose the appropriate method, a data mining query optimizer must be able to estimate execution costs for potential methods. In order to calculate the disk and cpu costs, database statistical model is needed. The model contains such coefficients as: number of disk blocks of a database and pattern histograms.

The statistical model of a database can be generated in three ways:

1. Users can explicitly and regularly run a statistics gatherer over the database.
2. At the beginning of a data mining query execution, a database sample can be used to dynamically gather the required statistical information.
3. Each data mining query being executed can automatically gather statistical information for the future use.

Since evaluating the complexity of many data mining algorithms requires detailed actual information on data characteristics, sampling can be the most promising method to build a precise statistical model. Although this additional sampling introduces some overhead to the query execution time, it allows us to make a decision which can spare hours of processing time.

5 Conclusions

We have showed the directions of extending a regular DBMS query optimizer to provide for data mining query optimization. Apart from using a traditional data mining algorithm, frequent itemsets can be discovered with help of materialized views and materialized data mining views. 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 table (or a part of it) can be gathered using a preliminary step of sampling.

For the future work we plan to develop efficient statistics gathering algorithms, which will not introduce significant overhead to the optimization process. Performance models for popular data mining algorithms must also be created to allow the optimizer to consider them during optimization.

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 Knowledge Discovery in Databases (KDD'94) (1994)
8. Morzy T., Wojciechowski M., Zakrzewicz M.: Materialized Data Mining Views. Proc. of the 4th PKDD Conference (2000)
9. Nag B., Deshpande P.M., DeWitt D.J.: Using a Knowledge Cache for Interactive Discovery of Association Rules. Proc. of the 5th KDD Conference (1999)