Materialized Views in Data Mining

Bogdan Czejdo
Department of Mathematics and Computer Science, Loyola University
czejdo@loyno.edu

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

Abstract

Data mining is an interactive and iterative process. It is highly probable that a user will issue a series of similar queries until he or she receives satisfying results. Currently available mining algorithms suffer from long processing times depending mainly on the size of the dataset. As the pattern discovery takes place mainly in the data warehouse environment, such long processing times are unacceptable from the point of view of interactive data mining. On the other hand, the results of consecutive data mining queries are usually very similar. This observation leads to the idea of reusing materialized results of previous data mining queries in order to improve performance of the system. In this paper we present the concept of materialized data mining views and we show how the results stored in these views can be used to accelerate processing of data mining queries. We demonstrate the use of materialized views in the domains of association rules discovery and sequential pattern search.

1. Introduction

Data mining is a non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in large volumes of data. Data mining systems are evolving from systems dedicated to and specialized in particular tasks or domains to general-purpose systems, which are tightly coupled with the existing database technology. This integration allows for the development of universal data mining environments that constitute a set of knowledge discovery algorithms and a data warehouse. Data warehouses form excellent data sources for several mining techniques but require a powerful back-end database engine.

From a user's point of view the execution of a data mining algorithm and the discovery of a set of patterns is an answer to a sophisticated database query. A user limits the mined dataset (e.g. by the means of a standard *SQL* query) and determines the values of parameters that control a given algorithm. In return, the system discovers the patterns and presents them to the user. When the process starts, the user does not know the exact goal of the exploration. Rather, he achieves satisfying results in several consecutive steps. In each step the user verifies the discovered patterns and, suitably to his needs, expectations, and experience, modifies either the mined dataset, or algorithm parameters, or both. In other words, the user discovers interesting and useful results in a series of runs, with the run environment slightly tuned in each run.

The basic problem in data mining is the processing time of data mining queries. Data mining algorithms often require minutes or hours to answer a simple query. On the other hand, mining practice shows that the majority of data mining queries are only minor modifications of previous queries. Given these circumstances, data mining systems should try to exploit the results of previous queries, instead of running a complete mining algorithm for each query.

In this paper we discuss techniques of reusing materialized results of previous queries stored in materialized views in the context of frequent itemsets, association rules, and sequential patterns. Materialized views have been thoroughly examined and successfully applied in traditional database management systems. We propose to follow this path and introduce materialized views to data mining systems. Considering data mining queries, finding materialized views suitable for answering a given query is difficult. Query defining a materialized view can differ from the current query both in algorithm parameters and the mined dataset. We show when and how materialized views can be used to answer a data mining query guaranteeing correctness of the answer. All examples presented in this paper are expressed in *MineSQL*, a declarative *SQL*-like data mining language [12].

2. Basic definitions

2.1. Frequent sets and association rules

Let $L = l_1, l_2, ..., l_m$ be a set of literals, called *items*. An *itemset* X is a non-empty set of items ($X \subseteq L$). The *size* of an itemset X is the number of items in X. Let D be a set of variable size itemsets, where each itemset T in D has a unique identifier and is called a *transaction*. We say that a transaction T *contains* an item $x \in L$ if x is in T. We say that a transaction T *contains* an itemset $X \subseteq L$ if T contains every item in the set X. The *support* of the itemset X is the percentage of transactions in D that contain X. The problem of mining frequent itemsets in D consists in discovering all itemsets whose support is above a user-defined support threshold.

An *association rule* is an implication of the form $X \rightarrow Y$, where $X \subseteq L$, $Y \subseteq L$, $X \cap Y = \emptyset$. We call X the *body* of a rule and Y the *head* of a rule. The *support* of the rule $X \rightarrow Y$ in D is the support of the itemset $X \cup Y$. The *confidence* of the rule $X \rightarrow Y$ is the percentage of transactions in D containing X that also contain Y. The problem of mining association rules in D consists in discovering all association rules whose support and confidence are above user-defined minimum support and minimum confidence thresholds.

2.2. Sequential patterns

Let $L = l_1, l_2, ..., l_m$ be a set of literals called *items*. An *itemset* is a non-empty set of items. A *sequence* is an ordered list of itemsets and is denoted as $< X_1 X_2 ... X_n >$, where X_i is an itemset ($X_i \subseteq L$). X_i is called an *element* of the sequence. The *size* of a sequence is the number of items in the sequence. The *length* of a sequence is the number of elements in the sequence.

We say that a sequence $X =< X_1 X_2 ... X_n >$ is a *subsequence* of a sequence $Y =< Y_1 Y_2 ... Y_m >$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i_1}, X_2 \subseteq Y_{i_2}, ..., X_n \subseteq Y_{i_n}$. We call $< Y_{i_1} Y_{i_2} ... Y_{i_n} >$ an *occurrence* of X in Y.

Let D be a set of variable length sequences (called *data-sequences*), where for each sequence $S =< S_1 S_2 ... S_n >$, a timestamp is associated with each S_i. With no time constraints we say that a sequence X is *contained* in a data-sequence S if X is a subsequence of S. We consider the following user-specified time constraints while looking for occurrences of a given sequence in a given data-sequence: minimal and maximal gap allowed between consecutive elements of an occurrence of the sequence (called *min-gap* and *max-gap*), and time window that allows a group of consecutive elements of a data-sequence to be merged and treated as a single element as long as their timestamps are within the user-specified *window-size*.

The *support* of a sequence $< X_1 X_2 ... X_n >$ in D is the fraction of data-sequences in D that contain the sequence. A *sequential pattern* is a sequence whose support in D is above the user-specified minimum support threshold.

2.3. Related work

The work on materialized views started in the 80s. The basic concept was to use materialized views as a tool to speed up queries and serve older copies of data. Multiple algorithms for view maintenance were developed [8][15].

The problem of association rule discovery was introduced in [1]. In the paper, discovery of frequent itemsets was identified as the key step in association rule mining. In [2], the authors proposed an efficient frequent itemset discovery algorithm called *Apriori* that became the basis for many other mining algorithms.

The idea of sequential pattern discovery was first presented in [3]. In [4], time constraints were incorporated into the problem and a sequential pattern discovery algorithm called *GSP* was introduced.

Incremental mining was first discussed in [7] in the context of association rules. A novel algorithm called *FUP* was proposed to efficiently discover frequent itemsets in an incremented dataset, exploiting previously discovered frequent itemsets. Incremental mining was also analyzed in the context of sequential patterns (e.g. [14]).

The notion of interactive and iterative knowledge discovery first appeared in [13]. The authors postulated to create a knowledge cache that would keep recently discovered frequent itemsets along with their support value. Besides presenting the notion of knowledge cache the authors introduced several maintenance techniques for such cache, and discussed using the cache contents when answering new frequent set queries. To facilitate interactive and iterative sequential pattern discovery, [14] proposed to materialize patterns discovered with the least restrictive selection criteria, and answer incoming queries by filtering the materialized pattern collection.

The concept of Knowledge and Data Management Systems was first introduced in [9]. In the opinion of the authors, KDMSs should replace contemporary database management systems by integrating data and knowledge related activities in one central place. The authors also defined the notion of a data mining query and suppressed the need to tightly integrate knowledge discovery systems with the existing database and data warehouse infrastructure to provide a framework for advanced applications.

3. Data mining queries

3.1. Queries

Several declarative data mining query languages have been proposed so far [6][10][12]. Such languages can be used to separate user applications from data mining algorithms. In this paper we use a multipurpose data mining query language called *MineSQL* [12] to formulate example queries. *MineSQL* employs the concept of data mining queries to express data mining tasks. *MineSQL* syntax mimics that of standard *SQL* and allows for tight and seamless integration of data mining queries with traditional database queries. *MineSQL* currently allows to issue commands that discover frequent itemsets, association rules and sequential patterns. *MineSQL* defines a set of additional data types (e.g., SET, SEQUENCE, RULE) as well as a set of operators and functions for those data types (e.g., CONTAINS, BODY(x), HEAD(x), SIZE(x), LENGTH(x)). The following data mining query discovers all frequent itemsets with support higher than 20% and containing an item *milk*. Mining takes place in the part of the database that contains transactional data for the 4th quarter of 2001.

```
MINE ITEMSET
FROM (SELECT SET(PURCHASED_ITEM)
FROM PURCHASES
WHERE DATE_OF_PURCHASE >= '01.10.2001'
AND DATE_OF_PURCHASE <= '31.12.2001'
GROUP BY TRANSACTION_ID)
WHERE SUPPORT(ITEMSET) > 0.2
AND ITEMSET CONTAINS TO_SET('milk');
```

Two classes of constraints can be seen in the above example. *Database constraints* are placed within WHERE clause in the SELECT subquery. Database constraints define the source dataset, i.e., the subset of the original database in which data mining is performed. *Mining constraints* are placed within the WHERE clause in the MINE statement. Mining constraints define the conditions that must be met by discovered patterns.

3.2. Relationships between results of data mining queries

In [5] three relationships which occur between two data mining queries Q_1 and Q_2 have been identified. Two data mining queries are *equivalent* if for all datasets they both return the same set of patterns and the values of statistical significance measures (e.g., support) for each pattern are the same in both cases. A data mining query Q_1 *includes* a data mining query Q_2 if for all datasets each pattern in the results of Q_2 is also returned by Q_1 with the same values of the statistical significance measures. A data mining query Q_1 *dominates* a data mining query Q_2 if for all datasets each pattern in the results of Q_2 is also returned by Q_1, and for each pattern returned by both queries its values of the statistical significance measures evaluated by Q_1 are not less than is case of Q_2.

Relationships described above occur between the results of data mining queries and can be used to identify the situations in which a query Q_1 can be efficiently answered using the materialized results of another query Q_2. Those relationships are general in nature and can be applied to various types of patterns (frequent sets, association rules, sequential patterns) and various constraint models. General idea of using materialized query results is the following. If for a given query, results of a query equivalent to it, including it, or dominating it are available, the query can be answered without running a costly mining algorithm. In case of equivalence no processing is required, since the queries have the same results. In case of inclusion, one scan of the materialized query results is necessary to filter out patterns that do not satisfy constraints of the included query. In case of dominance, one scan of the source dataset is necessary to evaluate the statistical significance of materialized patterns (filtering out the patterns that do not satisfy constraints of the dominated query is also required).

3.3. Data mining views

Traditional views are used mainly to hide difficult query structures from a user. Views also provide independence of applications from the schema changes occurring in the database. All changes must be reflected only in the definition of the view and no modification is required in end-user applications. Every access to the view triggers the execution of the query that defines the view.

Data mining is an interactive and iterative process and data mining queries tend to be fairly complicated. Data mining views hide the complexity of the algorithm from an application and simplify access to discovered patterns. The notion of a data mining view was introduced in [11]. The following *MineSQL* statement creates a data mining view *V_SEQ_PATS*. The view presents sequential patterns discovered in the *CUST_TRANSACTIONS* table, having the support exceeding 0.2, using the following time constraints: max-gap of 100, min-gap of 1, and no window-size (the default value of 0 is used).

```
CREATE VIEW V_SEQ_PATS AS
MINE PATTERN MAXGAP 100 MINGAP 1
FROM (SELECT SEQUENCE(T_TIME, ITEM)
FROM CUST_TRANSACTIONS
GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.2;
```

Data mining views provide additional independency layer between the database and the end-user application. Slight modifications of algorithm parameters or explored dataset are reflected only in the view definition whilst the application does not notice any changes. Besides, the user is separated from the technical details of the algorithm and can perform repetitive data mining tasks without knowing the details of syntax of the MINE statement. As with traditional views, every access to the data mining view triggers the execution of the underlying algorithm.

The algorithms for pattern discovery are usually time-consuming. Processing time of a data mining query could easily become unacceptable from the point of interactive mining. The solution to this problem is materialization of previously obtained results of data mining queries. A materialized data mining view is a database object storing the results of a data mining query (frequent sets, association rules, sequential patterns). With every materialized view a time period can be associated, after which the view is automatically refreshed. The following statement creates the materialized data mining view *MV_SEQ_PATS*. The view is to be refreshed automatically once a week.

```
CREATE MATERIALIZED VIEW MV_SEQ_PATS
REFRESH 7 AS
MINE PATTERN MAXGAP 100 MINGAP 1
FROM (SELECT SEQUENCE(T_TIME, ITEM)
FROM CUST_TRANSACTIONS
GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.2;
```

Materialized data mining views can be refreshed either automatically or on user's demand. In most cases such refresh can be performed by one of the incremental mining algorithms instead of running the complete discovery algorithm. Additional advantage of materialized views is the fact that data mining usually takes place in a data warehouse where changes to base relations (and thus to the stored patterns) do not happen continually over time but are accumulated and loaded to the data warehouse during data warehouse refresh process. The patterns discovered and stored in the materialized view remain valid for a long period of time until next data warehouse refresh. Validation of patterns can be postponed until next warehouse refresh event.

4. Using materialized data mining views in data mining query execution

In many cases contents of the materialized view can be used to answer a query that is similar to the query defining the view. For example, if the query defining the view Q_v includes a given query Q then the latter can be answered by simply reading the contents of the view and pruning those patterns that do not meet the conditions formulated in Q. The key issue is identification of syntactic differences leading to situations in which one query can be efficiently answered using the results of another query.

In our analysis we consider only materialized views containing frequent sets and sequential patterns. Even if the final goal is discovery of association rules, we propose to materialize frequent sets for two reasons. As it was also observed by other researchers: generation of rules from itemsets is straightforward [1], and materialized itemsets can be used to answer more itemset and rule queries [13].

4.1. Frequent sets

Given two queries Q_1 and Q_2 we say that Q_2 *extends database constraints* of Q_1 if syntactic differences between the queries imply that the source dataset for Q_2 will always be a subset of the source dataset for Q_1.

We say that Q_2 *extends mining constraints* of Q_1 if for an arbitrary collection of patterns, filtering it according to the outer WHERE clause of Q_2 will lead to a subset of the results of filtering it according to the outer WHERE clause of Q_1.

Depending on circumstances, given the query Q and the results of the query Q_v stored in a materialized view, several mining methods are available.

Incremental mining refers to the situation when one of the incremental discovery algorithms is executed on extended dataset. This method is used when the query Q_v extends database constraints of Q.

Another possibility is *complementary mining*. This method can be utilized when the query Q_v extends mining constraints of Q (all patterns available in the view will be present in the answer to the query). In this case, there is no need to compute the support of some patterns because it can be read from the materialized view.

Verifying mining is possible when Q extends pattern constraints of Q_v and has the same database constraints (this case corresponds to the inclusion relationship from [5]). The method consists in reading materialized view and pruning away those patterns that do not satisfy extended mining constraints of Q.

Finally, *full mining* refers to running a complete data mining algorithm that does not exploit results of previous queries. This method has to be applied when for a given query materialized views supporting incremental, complementary, or verifying mining are not available.

Let us consider the following example. We are given the following definition of a materialized data mining view Q_v:

```
MINE ITEMSET
FROM (SELECT SET(PURCHASED_ITEM)
FROM PURCHASES
GROUP BY TRANSACTION_ID
HAVING COUNT(*) >= 5)
WHERE SUPPORT(ITEMSET) > 0.3;
```

and the following data mining query Q:

```
MINE ITEMSET
FROM (SELECT SET(PURCHASED_ITEM)
FROM PURCHASES
GROUP BY TRANSACTION_ID)
WHERE SUPPORT(ITEMSET) > 0.5;
```

The query Q extends mining constraints of Q_v by setting a higher value of minimum support. On the other hand, the query Q_v extends database constraints of Q by adding the HAVING clause. To answer Q using the contents of Q_v the following steps need to be taken. First, verifying mining is performed to prune patterns with support not exceeding 0.5. Next, incremental mining is performed on the part of the database consisting of transactions shorter than 5 items.

4.2. Sequential patterns

Similarly to frequent itemset discovery, materialized views can be successfully utilized in sequential pattern search. Apart from the mining and database constraints, in sequential pattern discovery *time constraints* are also present in data mining queries. These time constraints are *min-gap*, *max-gap* and *window-size*.

The relationships of extending mining and database constraints defined in the previous section carry over to sequential patterns. Additionally, we say that Q_2 *extends time constraints* of Q_1 if it tightens at least one of the time parameters without relaxing any remaining parameters.

All the query processing techniques involving materialized views: incremental mining, complementary mining, and verifying mining are also valid for sequential patterns, provided that the current query Q and the query Q_v defining a materialized view have the same time constraints.

If Q_v extends time constraints of Q full mining has to be performed. On the other hand, if Q extends time constraints of Q_v with the same mining and database constraints, Q can be answered by re-evaluating the support of patterns returned by Q_v using the time constraints of Q and pruning the patterns not satisfying the minimum support threshold of Q (this case corresponds to dominance relationship from [5]). This technique can be used as the initial step before incremental, complementary, and verifying mining, if differences in mining and database constraints suggest a given method, and additionally Q extends time constraints of Q_v.

5. Conclusions

We have addressed the problem of employing materialized data mining views to optimize data mining queries in large data warehouses. Materialized data mining views are physical data warehouse structures, created explicitly or implicitly, used to store precomputed results of selected data mining queries.

We showed that in some situations, a new data mining query can be mapped to an existing materialized data mining view and it can be answered without the need to run a complete data mining algorithm. We classified mining methods exploiting materialized results of previous data mining queries, and identified situations in which those methods are applicable in the context of two main data mining techniques: frequent itemset discovery and sequential pattern discovery.

In practical data mining systems, a cost-based query optimizer should be responsible for using the described methods for seamless rewriting of users' queries to shorten their execution time. Designing such a data mining query optimizer is one of our future research goals.

References

[1] R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the 1993 ACM SIGMOD Conference*, 1993.

[2] R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules. In *Proc. of the 20th VLDB Conference*, 1994.

[3] R. Agrawal, R. Srikant. Mining Sequential Patterns. In *Proc. of the 11th ICDE Conference*, 1995.

[4] R. Agrawal, R. Srikant. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proc. of the 5th EDBT Conference*, 1996.

[5] E. Baralis, G. Psaila. Incremental refinement of mining queries. In *Proc. of the 1st DaWaK Conference*, 1999.

[6] S. Ceri, R. Meo, G. Psaila. A New SQL-like Operator for Mining Association Rules. In *Proc. of the 22nd VLDB Conference*, 1996.

[7] D. W.-L. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proc. of the 12th ICDE Conference*, 1996.

[8] A. Gupta, I. S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin, Special Issue on Materilaized Views and Data Warehousing*, 18(2), 1995.

[9] T. Imielinski, H. Mannila. A Database Perspective on Knowledge Discovery. *Communications of the ACM*, 39(11), 1996.

[10] T. Imielinski, A. Virmani, A. Abdulghani. Datamine: Application programming interface and query language for data mining. In *Proc. of the 2nd KDD Conference*, 1996.

[11] T. Morzy, M. Wojciechowski, M. Zakrzewicz. Materialized Data Mining Views. In *Proc. of the the 4th PKDD Conference*, 2000.

[12] T. Morzy, M. Wojciechowski, M. Zakrzewicz. Data Mining Support in Database Management Systems. In *Proc. of the 2nd DaWaK Conference*, 2000.

[13] B. Nag, P. Deshpande, D. J. DeWitt. Using a Knowledge Cache for Interactive Discovery of Association Rules. In *Proc. of the 5th KDD Conference*, 1999.

[14] S. Parthasarathy, M. J. Zaki, M. Ogihara, S. Dwarkadas. Incremental and interactive sequence mining. In *Proc. of the 1999 ACM CIKM Conference*, 1999.

[15] N. Roussopoulos. Materialized Views and Data Warehouses. *SIGMOD Record*, 27(1), 1998.