Materialized Views in Data Mining

Bogdan Czejdo[1], Mikołaj Morzy[2], Marek Wojciechowski[2], Maciej Zakrzewicz[2]

Mathematics & Computer Science Department, Loyola University[1]
czejdo@loyno.edu

Institute of Computing Science, Poznań University of Technology[2]
{mmorzy,marek,mzakrz}@cs.put.poznan.pl

Abstract

Data mining is an interactive and iterative process. A user defines a set of interesting patterns choosing the dataset to be mined and setting the values of various parameters that drive mining algorithm. It is highly probable that a user will issue the same mining query several times until he receives satisfying results. During each run a user will slightly modify either the definition of the mined dataset or the parameters of the algorithm. 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 very similar. One possible solution is to reuse materialized results of previous data mining queries. 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

1.1. Overview

Data mining, also referred to as knowledge discovery in databases, is a non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data [8]. 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 relational 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. Most data mining queries are costly (in terms of processing cost) and differ significantly from the typical database queries. Hence, novel methods of query processing and optimization need to be developed in order to achieve satisfying data mining query performance.

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 given algorithm. In return the system discovers the patterns and presents them to a user. When the process starts, a 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, a user discovers interesting and useful results in a series of runs, with the run environment slightly tuned in each run. Mining practice shows that the vast majority of data mining queries are only minor modifications of former queries. Given these circumstances it is necessary for a user to be able to exploit the results of previous queries in answering given query. Knowledge discovery system should be capable of answering a query in an incremental manner where the results of previous queries are maintained and tested against current dataset and parameter set. In incremental mining base algorithm is run only on the difference set. This principle applies also to the situation when the mining algorithm is run after a data warehouse refresh to discover novel patterns. Usually the volume of new or changed data after refresh is significantly smaller

(and often negligible) when compared to the size of the original warehouse.

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. In addition, the size of the result can easily surpass the size of the queried database. Such properties of mining process make it unsuitable for interactive and iterative pattern discovery.

One possible solution to this problem is to use materialized views. Data mining query results can be materialized automatically or at user request. Knowledge discovery system should be capable of using these results and incorporating them into mining algorithms. 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 knowledge discovery systems.

In this paper we present algorithms that reuse materialized results of former queries and we show how such algorithms can accelerate processing times of queries in the discovery of frequent itemsets, association rules and sequential patterns. Experiments prove that the use of materialized views can shorten considerably query times for large class of queries in traditional database systems. Considering data mining queries, finding materialized views suitable for answering a given query is more difficult. Query defining a materialized view can differ from the actual query both in algorithm parameters and the mined database schema. In this paper we show how materialized views can be used to answer a data mining query and what additional steps must be taken to assure the correctness of the answer. All examples presented in this paper where expressed in MineSQL, a declarative data mining language developed in the Institute of Computing Science of Poznań University of Technology [15].

1.2. Outline

The article is organized as follows. Chapter 2 provides the definitions of basic notions and presents information on related work. In Chapter 3 the idea of data mining queries is presented and illustrated with examples. Different relations occurring between data mining queries are analyzed and the notion of materialized data mining view is introduced. Chapter 4 focuses on data mining query optimization using materialized views. We conclude in Chapter 5 with a brief discussion of open questions and future work agenda.

2. Basic Definitions

2.1. Traditional Views vs. Materialized Views

A view is a derived relation defined in terms of base relations. Formally, a view defines a function from the set of base relations to the derived relation. This function is usually computed on each reference to the view. A view can be materialized by storing tuples in the database. Because all data available in a materialized view are stored on a disk in a database, users can create indexes on materialized views, thus shortening the time needed to access tuples. This time can be significantly shorter than the time needed to recompute a view. In a way a materialized view resembles cache – it is a copy of the data that can be quickly accessed. The contents of a materialized view become invalid after any modification to base relations. In such cases view maintenance techniques are necessary to reflect the changes that happen in base relations of a materialized view. Sometimes updates of base relations affect only a part of a materialized view. In these cases recomputation of an entire view would be a waste of time and resources. It is faster and cheaper to perform incremental view maintenance, i.e., to recompute only the part of a materialized view affected by the base relation updates. It is worth noticing that in general incremental view maintenance requires some additional data and metadata to work properly, and in some cases (depending on the materialized view definition and base relation properties) incremental maintenance is impossible [10].

2.2. Frequent Sets

Let $L=\{l_1, l_2, ..., l_n\}$ be a set of literals called *items*. Let D be a set of variable length transactions and $\forall T \in D: T \subseteq L$. We say that the transaction T *supports* an item x if $x \in T$. We say that the transaction T *supports* an itemset X if T supports every element in X. The *support* of the itemset X is the ratio of the number of transactions supporting the itemset to the total number of transactions. An itemset containing k items is called a *k-itemset*.

$$support(X, D) = \frac{\left|\{T \in D : T \text{ supports } D\}\right|}{\left|D\right|}$$

The problem of discovering frequent itemsets can be formulated as follows. Given a database D and a minimum support threshold supplied by a user (called *minsup*) find all itemsets occurring in a database D with the support higher than *minsup*. An itemset with the support higher than *minsup* is called a *frequent itemset*.

2.3. Association Rules

An *association rule* is an implication of the form $X \rightarrow Y$ where $X \subset L$, $Y \subset L$ and $X \cap Y = \emptyset$. X is called the *head* of a rule whilst Y is called the *body* of a rule. Two statistical measures define its statistical significance and strength.

The *support* of a rule $X \rightarrow Y$ is the ratio of the number of transactions supporting the rule to the total number of transactions. In other words, a rule $X \rightarrow Y$ has the support of s in a database D if $s\%$ of transactions support $X \cup Y$.

$$support(X \rightarrow Y, D) = \frac{\left|\{T \in D : T \text{ supports } X \cup Y\}\right|}{\left|D\right|}$$

The *confidence* of a rule $X \rightarrow Y$ is the ratio of the number of transactions supporting the rule to the number of transactions that support the head of the rule. In other words, a rule $X \rightarrow Y$ has the confidence of c in a database D if $c\%$ of transactions supporting X also support Y.

$$confidence(X \rightarrow Y, D) = \frac{\left|\{T \in D : T \text{ supports } X \cup Y\}\right|}{\left|\{T \in D : T \text{ supports } X\}\right|}$$

The problem of discovering association rules can be formulated as follows. Given a database D and the minimum thresholds of support and confidence supplied by a user (called *minsup* and *minconf* respectively) find all association rules occurring in a database D with support and confidence higher than *minsup* and *minconf*.

2.4. Sequential Patterns

Let $L=\{l_1, l_2, ...,l_n\}$ be a set of literals called *items*. A *sequence* is an ordered list of sets of items and is denoted as

$<X_1, X_2, ...,X_n>$ where X_i is a set of items, $X_i \subseteq L$. Sets X_i are called *sequence elements*. The *size* of a sequence is the number of items in a sequence. The *length* of the sequence is the number of elements in a sequence. Each element has a timestamp associated with it. We say that the sequence $X=<X_1,...,X_n>$ is *contained* in the sequence $Y=<Y_1,...,Y_m>$ if there exist integer numbers $i_1<...<i_n$ such that $X_1<Y_{i1}, ..., X_n<Y_{in}$. The sequence $<Y_{i1},...,Y_{in}>$ is called an *occurrence* of X in Y. There are three main time constraints involved in sequential pattern discovery, namely, the minimum and maximum time gap between consecutive occurrences of elements in a sequence (*min-gap* and *max-gap* respectively) and the size of the time window which allows for merging identical sequence elements (provided the timestamps of those elements are contained in one window). The size of the window is denoted as *window-width*.

The *support* of a sequence $X=<X_1,...,X_n>$ in a database D is the ratio of the number of transactions containing the sequence to the total number of transactions. The problem of sequential pattern search can be formulated as follows. Given a database D and the minimum support threshold supplied by a user (called *minsup*) find all sequences occurring in the database D with support higher than *minsup*. A sequence with support higher than *minsup* is called a *sequential pattern*.

2.5. 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 [18]. Further research led to the creation of cost models for materialized view maintenance and determining the impact of materialized views presence on query processing performance. Some research has been conducted on applying views to force integrity constraints in databases. A summary of view maintenance techniques can be found in [9]. For a full presentation of subjects related to materialized views see [10].

The problem of association rule mining has been introduced in [1]. The notion of frequent set has been introduced in [2]. The authors proposed an algorithm called *Apriori* that became the basis for several data mining algorithms. The apriori principle reflects a simple observation: an itemset can be frequent if and only if all its subsets are frequent. In other words, only frequent sets are needed for generating larger frequent sets. The algorithm works as follows. In the first step all 1-itemsets are found and their support is determined during a full database scan. In all consecutive steps candidate itemsets (itemsets which are potentially large) of size n are generated based on frequent $(n-1)$ itemsets. Support values of all candidate itemsets of size n are determined during a database scan and the itemsets with support lower than *minsup* threshold are purged from the collection of frequent itemsets. The main drawback of the *Apriori* algorithm is the fact that it uses $(k+1)$ full database scans (which are costly and time-consuming operations) to find all frequent sets of size k.

In [6] a novel algorithm called FUP was proposed. This algorithm took over traditional *Apriori* technique by implementing an incremental frequent itemset search. FUP algorithm exploited previously discovered frequent itemsets and performed pattern search only in the modified part of the database. The next proposal presented in [19] performed incremental frequent itemset search in both reduced and extended databases. This algorithm exploited properties of the negative itemset boundary introduced in [20].

The idea of sequential patter discovery was first presented in [3]. An algorithm called *GSP* was introduced. *GSP* algorithm considered various time-related constraints and was capable of discovering a wide range of sequential pattern classes (including generalized sequential patterns). In [4] the authors proposed to materialize sequential patterns with reduced support and time constraints and to use these materialized patterns to answer incoming queries. Nevertheless, most work on sequential pattern discovery focused on improving the performance of the algorithm.

The notion of interactive and iterative knowledge discovery first appeared in [16]. The authors postulated to create a knowledge cache that would keep recently discovered frequent itemsets along with their support value. Such knowledge cache could be shared among multiple users and multiple applications, allowing them to use reciprocally partial results of their queries. Besides presenting the notion of knowledge cache the authors introduced several maintenance techniques for such cache.

In [21] an interesting idea of prior computation of frequent itemsets in database partitions was formulated. This method utilized the fact that an itemset can be frequent in the database if it is frequent in at least one partition. The authors presented an algorithm that divided the original database into several smaller partitions, each of them fitting into available main memory. The algorithm would find frequent itemsets in each partition independently. Frequent itemsets found during this process were materialized and used to identify itemsets that were frequent in an entire database.

The concept of Knowledge Data Management System was first introduced in [13]. In the opinion of the authors KDMS should replace contemporary database management systems by integrating data and knowledge related activities in one central place. The authors defined also 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

In [15] a declarative data mining language called MineSQL was introduced. *MineSQL* enables to express knowledge discovery problems in terms of data mining queries. This language separates user applications from a data mining algorithm. *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, ITEMSET, RULE) as well as set of operators and functions for those data types (e.g., CONTAINS, BODY(x), HEAD(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, **SUPPORT**(ITEMSET)
FOR ITEMS **FROM** (
SELECT SET(PURCHASED_ITEM) **AS** ITEMS
FROM PURCHASES
WHERE DATE_OF_PURCHASE > '01.07.2001'
AND DATE_OF_PURCHASE < '31.12.2001'
GROUP BY TRANSACTION_ID)
WHERE SUPPORT(ITEMSET) > 0.2
AND ITEMSET **CONTAINS TO_SET**('*milk'*);

Similarly one can use *MineSQL* to discover all association rules with support higher than 10%, confidence higher than 30% and containing item '*butter'* in the head of a rule.

MINE RULE R, **HEAD**(R), **BODY**(R)
FOR ITEMS **FROM** (
SELECT SET(PURCHASED_ITEM) **AS** ITEMS
FROM PURCHASES
GROUP BY TRANSACTION_ID)
WHERE SUPPORT(R) > 0.1
AND CONFIDENCE(R) > 0.3
AND HEAD(R) **CONTAINS TO_SET**('*butter*);

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 and described. These relationships include equality, containment and domination.

- We say that two data mining queries are *equal* if for every database the result sets of patterns returned by both queries are identical and for every pair of patterns the values of statistical coefficients (e.g., values of support and confidence) are equal
- We say that a data mining query Q_2 *contains* a query Q_1 if for every database each pattern returned by Q_1 is also returned by Q2 and the values of statistical coefficients are equal in both result sets
- We say that a data mining query Q_2 *dominates* a query Q_1 if for every database each pattern returned by Q_1 is also returned by Q_2 and the values of statistical coefficients determined by Q_1 are not less than the values of respective coefficients determined by Q_2

Equality of data mining queries is a special case of containment relation, and containment is a special case of more general dominance relation.

Relations 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 relations 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 Q_1 exist materialized results of another query Q_2 equal to Q_1 then no processing is required and Q_1 can be answered entirely from the results of Q_2 (recall that both queries return the same set of patterns for the same database). If materialized results are available from the query Q_2 containing the original query Q_1 then a full result set scan is required to filter out those patterns from Q_2 that do not satisfy constraints imposed on Q_1. If materialized results are available from the query Q_2 dominating the original query Q_1 then a full database scan is required to determine the values of statistical coefficients of patterns present in Q_2 (those values may vary from Q_1). Additionally, a scan of result set is required to filter out those patterns from Q_2 that do not satisfy the constraints imposed on Q_1.

3.3. Data mining views

Traditional views are used mainly to hide difficult query structures from a user and to simplify access data. Views also provide independence of application from the changes happening in the database. All changes must be reflected only in the definition of the view and no modification is required in the end-user application. Every access to the view triggers the execution of the query that defines this 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 [14]. Below is a *MineSQL* statement that creates a data mining view *V_ASSOC_RULES*.

CREATE VIEW V_ASSOC_RULES **AS**
MINE RULE, **BODY**(RULE), **SUPPORT**(RULE)
FOR ITEMS **FROM** (
SELECT SET(PURCHASED_ITEMS) **AS** ITEMS
FROM PURCHASES
WHERE TRANSACTION_DATA > '01.01.2001'
AND TRANSACTION DATA < '31.12.2001'
GROUP BY TRANSACTION_ID
HAVING COUNT(*) >= 3)
WHERE SUPPORT(RULE) > 0.2
AND HEAD(RULE) **CONTAINS TO_SET**('*bread*');

Two classes of constraints can be seen in the definition above. *Database constraints* are placed within WHERE clause in the SELECT subquery. Database constraints define a *data view*, 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.

The use of data mining view provides additional independency layer between the database and the end-user application. Slight modifications of algorithm parameters or explored data view 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. As with traditional views, every access to the data mining view triggers the execution of the underlying algorithm.

Algorithms for pattern discovery are usually very time-consuming. Processing time of a data mining query could easily become unacceptable from the point of view of interactive process of knowledge discovery. The solution of this problem is materialization of previously obtained results of data mining queries. A materialized data mining view is a database object storing patterns (frequent sets, association rules, sequential patterns) discovered during data mining queries. Every pattern in a materialized view has a timestamp representing its creation time and validity period. With every materialized view a time period can be associated, after which the contents of the view is automatically refreshed. Below is a MineSQL statement that creates the materialized data mining view *MV_ASSOC_RULES*.

CREATE MATERIALIZED VIEW
V_ASSOC_RULES **REFRESH** 7 **AS**
MINE RULE, **SUPPORT**(RULE), **CONFIDENCE**(RULE)
FOR ITEMS **FROM** (
SELECT SET(PURCHASED_ITEMS) **AS** ITEMS
FROM PURCHASES
WHERE ITEM_GROUP=*'beverages'*
GROUP BY TRANSACTION_ID)
WHERE SUPPORT(RULE) > 0.3
AND CONFIDENCE(RULE) > 0.5;

Materialized data mining view can be refreshed either automatically or on user's demand. In most cases such refresh can be performed by one of the incremental refresh algorithms [6,7,19] instead of running the algorithm from scratch. Additional advantage of materialized view is the fact that data mining usually takes place in the data warehouse environment in which 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. Data mining query optimization

4.1. Frequent sets and association rules

In many cases contents of the materialized view can be used to answer a query that is similar to a query defining the view. If the query defining the view Q_v dominates or contains given query Q then the later can be answered by simply reading the contents of the view and purging those patterns that do not meet the conditions formulated in Q. In order to use the contents of a materialized view for data mining query optimization additional formulations are required. First of all, it is necessary to define the conditions that must be met for an answer using materialized patterns to be correct. Those conditions are based on relations occurring between data mining queries. Below four main relationships regarding both database and mining constraints are identified.

Given materialized view based on query Q_v and a data mining query Q we say that:

- query Q extends database constraints of Q_v if
  - Q adds additional WHERE or HAVING clauses to the database constraints of Q_v
  - Q adds an ANDed condition to the database constraints of Q_v in the WHERE or HAVING clauses
  - Q removes an ORed condition from the database constraints of Q_v in the WHERE or HAVING clauses
- query Q reduces database constraints of Q_v if
  - Q removes WHERE or HAVING clauses from the database constraints of Q_v
  - Q removes an ANDed condition from the database constraints of Q_v in the WHERE or HAVING clauses
  - Q adds an ORed condition to the database constraints of Q_v in the WHERE or HAVING clauses
- query Q extends mining constraints of Q_v if
  - Q adds additional WHERE or HAVING clauses to the mining constraints of Q_v
  - Q adds an ANDed condition to the mining constraints of Q_v in the WHERE or HAVING clauses
  - Q removes an ORed condition from the mining constraints of Q_v in the WHERE or HAVING clauses
  - replaces mining constraint present in Q_v with a more restrictive constraint (e.g., higher *minsup* value)
- query Q reduces mining constraints of Q_v if
  - Q removes WHERE or HAVING clauses from the mining constraints of Q_v
  - Q removes an ANDed condition from the mining constraints of Q_v in the WHERE or HAVING clauses
  - Q adds an ORed condition to the mining constraints of Q_v in the WHERE or HAVING clauses
  - replaces mining constraint present in Q_v with a less restrictive constraint (e.g., lower *minsup* value)

Conceptually extending database constraints means shrinking the dataset whilst reducing database constraints means extending the data set to be mined. Extending mining constraints means narrowing whilst reducing mining constraints means extending the result set of patterns.

Depending on circumstances several mining methods are available. *Full mining* refers to the situation when the contents of a view cannot be used to answer the query and the mining algorithm must be run from scratch. This situation occurs when the query Q extends database constraints of the query Q_v defining the view. *Incremental mining* refers to the situation when one of the incremental discovery algorithms is executed on extended data view. This method is used when the query Q reduces database constraints of Q_v. Another possibility is *complementary mining*. Patterns are discovered based on previously discovered patterns. This method can be utilized when the query reduces mining constraints of Q_v (all patterns available in the view will be present in the answer to the query). Finally, *verifying mining* consists in reading materialized view and pruning away those patterns that do not satisfy extended mining constraints of Q. Below an example of using a materialized data mining view to answer a data mining query is presented.

Given the following definition of materialized data mining view Q_v:

MINE ITEMSET, **SUPPORT**(ITEMSET)
FOR ITEMS **FROM** (
SELECT SET(PURCHASED_ITEM) **AS** ITEMS
FROM PURCHASES
GROUP BY TRANSACTION_ID
HAVING COUNT(*) > 5)
WHERE SUPPORT(ITEMSET) > 0.3;

and the following data mining query Q:

MINE ITEMSET, **SUPPORT**(ITEMSET)
FOR ITEMS **FROM** (
SELECT SET(PURCHASED_ITEM) **AS** ITEMS
FROM PURCHASES
GROUP BY TRANSACTION_ID)
WHERE SUPPORT(ITEMSET) > 0.5
AND ITEMSET **CONTAINS TO_SET**('*butter*','*milk*');

The query Q extends mining constraints of Qv by setting a more restrictive (higher) value of minimum support and adding a clause containing mining constraints (narrowing pattern set to those patterns that contain items '*butter*' and '*milk*'). Conversely, the query Q reduces mining constraints of Q_v by removing a HAVING clause from the view definition. 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 lower than 0.5 and not containing items '*butter*' and '*milk*'. 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 and association rule discovery, materialized views can be successfully utilized in sequential pattern search. In order to be able to use the results materialized in a query it is necessary to first determine the type of relationship occurring between the two queries (the query being answered and the query defining a materialized view). This relationship depends strongly on liaisons between classes of constraints in the queries. Basic formulation of the sequential pattern search identifies three constraint classes.

- *database constraints* are used to limit the original database to the interesting subset of data
- *mining constraints* are the parameters of discovery algorithms, currently only minimal support threshold (denoted as *minsup*) is used
- *time constraints* are used to set the processing window size, currently only minimum and maximum gaps between consecutive elements are used along with the window width (denoted as *min-gap*, *max-gap* and *window-width* respectively)

For any data mining queries Q_1 and Q_2 two reciprocal relationships can be observed with respect to the above defined constraint classes.

- query Q_2 *extends mining constraints* of Q_1 if the mining constraints of Q_1 can be obtained from the mining constraints of Q_2 by adding new elementary predicates or replacing existing predicates of Q_2 by stronger (more restrictive) predicates
- query Q_2 *extends time constraints* of Q_1 if it tightens one of the time parameters *min-gap*, *max-gap* or *window-width* without relaxing any remaining parameters

These remarks concern the syntax of data mining queries. The implications of syntactic differences between data mining queries were examined in detail in [22]. With respect to sequential pattern queries the following relationships exist.

- let the queries Q_1 and Q_2 have identical data views (i.e., both queries operate on the same data set and have identical database constraints). Let the queries Q_1 and Q_2 have the same time constraints. If the query Q_2 extends mining constraints of Q_1 then Q_1 *contains* Q_2
- let the queries Q_1 and Q_2 operate on the same data set and have the same mining constraints. If the query Q_2 extends time constraints of Q_1 then Q_1 *dominates* Q_2
- let the queries Q_1 and Q_2 operate on the same data set. If the query Q_2 extends both time and mining constraints of Q_1 then Q_1 *dominates* Q_2

Those relationships form the basis for the algorithms that use materialized results of previous queries to answer a given query. Below we present a short description of various techniques using materialized data mining views. For a more detailed presentation of the subject please refer to [22]. In all examples Q denotes a data mining query expressing sequential pattern search and Q_v denotes the query defining the contents of a materialized data mining view *MV*.

If Q and Q_v operate on the same data set and have identical mining and time constraints then the equality occurs and no processing is required. The results of both queries are identical and entirely contained in the materialized view *MV*.

If Q and Q_v operate on the same data set and have the same time constraints and the query Q extends mining constraints of Qv then the query Q can be answered from the materialized view by purging those patterns from Q_v that do not satisfy the more restrictive constraints of Q (Q_v contains Q). The algorithm performs one scan of the materialized view and for each pattern it verifies whether the pattern satisfies additional mining constraints imposed on Q.

If Q and Q_v operate on the same data set and have identical mining constraints and the query Q extends time constraints of Q_v then Q can be answered by reading the contents of the view and verifying patterns materialized in the view against time constraints of Q (by definition Q_v dominates Q).

If Q and Q_v operate on the same data set and the query Q extends both mining and time constraints of Q_v (in other words, Q_v dominates Q) then Q can be answered by reading all patterns materialized in the view and recomputing their support using time constraints of Q and considering extended mining conditions of Q. The algorithm performs full scan of the materialized view and purges all patterns that do not satisfy the mining constraints of Q. Then the algorithm performs a full database scan and computes the support of the remaining patterns using time constraints of Q. Finally, patterns that have support value below the threshold imposed on Q are removed from the result set.

These methods allow for answering a sequential patterns data mining query without the need to run time-consuming algorithm. Rather, they reuse patterns found in previous sessions and materialized in views. Experiments show clearly that this approach leads to significant improvements in processing time of sequential pattern data mining queries.

5. Conclusions

In this paper we addressed the problem of data mining query optimization using materialized views. We presented methods for speeding up processing times in frequent itemset search, association rule discovery and sequential pattern search. Still many questions remain opened and several challenging issues need to be addressed. For example, most of the presented algorithms assume that interactive exploration takes place in the same data set from which the materialized view has been derived. Moreover, those methods assume that the shape of transactions remain constant. Questions concerning efficient view maintenance methods for data mining views remain unanswered. Another

open research area is the domain of cost models for data mining query optimization.

Our future work will focus on extending the usability of described methods to queries that differ from the view definition in explored data view and on constructing cost models for data mining query processing. A cost model is a necessary condition for ambitious plan of creating a fully featured cost-based data mining query optimizer module.

6. References

[1] Agrawal, R., Imielinski, T., Swami, A., "Mining association rules between sets of items in large databases", In Proc. of the ACM SIGMOD International Conference on Management of Data, Washington, USA, May 1993

[2] Agrawal, R., Srikant, R., "Fast Algorithms for Mining Association Rules", In Proc. of the 20th International Conference on Very Large Data Bases (VLDB'94), Santiago, Chile, 1994.

[3] Agrawal, R., Srikant, R., "Mining Sequential Patterns", In Proc. of the 11th International Conference on Data Engineering (IDCDE'95), Taipei, Taiwan, March 1995

[4] Agrawal, R., Srikant, R., "Mining Sequential Patterns: Generalizations and Performance Improvements", In Proc. of the 5th International Conference on Extending Database Technology (EDBT'96), Avignon, France, September 1996

[5] Baralis, E., Psaila, G., "Incremental refinement of mining queries", In Proc. of the 1st International Conference on Data Warehousing and Knowledge Discovery (DaWaK'99), Florence, Italy, September 1999

[6] Cheung, D.W., Han, J., Ng, V., Wong, C.Y., "Maintenance of discovered association rules in large databases: An incremental updating technique", In Proc. of the 12th International Conference on Data Engineering (ICDE'96), New Orleans, USA, February 1996

[7] Cheung, D. W., Lee, S. D. and Kao, B., "A General Incremental Technique for Maintaining Discovered Association Rules", In Proc. of the 5th International Conference on Database Systems for Advanced Applications (DASFAA'97), Melbourne, Australia, April 1997

[8] Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., (Eds.) Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press 1996,

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

[10] Gupta, A., Mumick, I.S., Materialized Views: Techniques, Implementations, and Applications, The MIT Press, 1999

[11] Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.-C., "FreeSpan: frequent pattern-projected sequential pattern mining", In Proc. of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'2000), Boston, USA, August 2000

[12] Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.-C., "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth", In Proc. of the 17th International Conference on Data Engineering (ICDE'01), Heidelberg, Germany, April 2001

[13] Imielinski, T., Mannila, H., „A Database Perspective on Knowledge Discovery", Communications of the ACM, Vol.39, No.11, 1996

[14] Morzy, T., Wojciechowski, M., Zakrzewicz, M., "Data Mining Query Optimization Using Materialized Views"

[15] Morzy, T., Zakrzewicz, M., "SQL-like Language for Database Mining", In Proc. of the 1st East European Symposium on Advances in Databases and Information Systems (ADBIS'97), St-Petersburg, Russia, September 1997

[16] Nag, B., Deshpande, P., DeWitt, D.J., "Using a Knowledge Cache for Interactive Discovery of Association Rules", In Proc. of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99), San Diego, USA, August 1999

[17] Parthasarathy, S., Zaki, M.J., Ogihara, M., Dwarkadas, S., "Incremental and interactive sequence mining", In Proc. of the ACM International Conference on Information and Knowedge Management (CIKM'99), November 1999

[18] Roussopoulos, N., "Materialized Views and Data Warehouses", SIGMOD Record vol.27 no 1, 1998,

[19] Thomas, S., Bodagala, S., Alsabti, K., Ranka, S., "An Ecient Algorithm for the Incremental Updation of Association Rules in Large Databases", In Proc. of the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'97), Newport Beach, USA, August 1997

[20] Toivonen, H., "Sampling large databases for association rules", In Proc. of the 22th International Conference on Very Large Data Bases (VLDB'96), Bombay, India, September 1996

[21] Wojciechowski, M., Zakrzewicz, M., "Itemset Materializing for Fast Mining of Association Rules", In Proc. of the 2nd East European Conference on Advances in Databases and Information Systems (ADBIS'98), Poznań, Poland, September 1998

[22] Wojciechowski, M., "Interactive Constraint-Based Sequential Pattern Mining", In Proc. of the 5th East European Conference on Advances in Databases and Information Systems (ADBIS'01), Vilnius, Lithuania, September 2001