Fast Discovery of Sequential Patterns Using Materialized Data Mining Views

Tadeusz Morzy, Marek Wojciechowski, Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
{morzy,marek,mzakrz}@cs.put.poznan.pl

Abstract. Most data mining techniques consist in discovery of frequently occurring patterns in large data sets. From a user's point of view, data mining can be seen as advanced querying, where each data mining query specifies the source data set and the requested class of patterns. Unfortunately, current data mining systems consume minutes or hours to answer simple queries, which makes them unsuitable for interactive use. In this paper we present the concept of materialized data mining views and their application to fast discovery of sequential patterns. We show how materialized data mining views can be used to optimize processing of sequential pattern queries.

1 Introduction

Data mining, also referred to as database mining or knowledge discovery in databases (KDD), aims at discovery of useful patterns from large databases or warehouses. Currently we are observing the evolution of data mining environments from specialized tools to multi-purpose data mining systems offering some level of integration with existing database management systems. From a user's point of view, data mining can be seen as an interactive and iterative process of advanced querying: a user specifies the source data set and the requested class of patterns, the system chooses the right data mining algorithm and returns discovered patterns to the user. The most serious problem concerning data mining queries is a long response time. Current systems consume minutes or hours to answer simple queries.

One of the most popular data mining methods is sequential pattern discovery. Sequential patterns are the most frequently occurring subsequences in sequential data. Their applications include analysis of telecommunication systems, discovering frequent buying patterns, analysis of patients' medical records, etc.

In this paper we discuss optimization of the sequential pattern discovery problem. We propose using periodically refreshed *materialized data mining views* (MDMVs) for repetitive data mining queries in the same manner as materialized views are used in relational database management systems to store results of complex and time consuming queries. We notice that it is obvious that MDMVs can be used to answer queries identical to the queries over which they have been defined, therefore we focus on processing queries that differ in their syntax.

1.1 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. Let D be a set of variable length sequences, where for each sequence $S = <X_1 X_2 ... X_n>$, a timestamp is associated with each X_i.

With no time constraints we say that a sequence $X = <X_1 X_2 ... X_n>$ is *contained* in a sequence $Y = <Y_1 Y_2 ... Y_m>$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i1}$, $X_2 \subseteq Y_{i2}$, ..., $X_n \subseteq Y_{in}$. We call $<Y_{i1} Y_{i2} ... Y_{in}>$ an *occurrence* of X in Y. We consider the following user-specified time constraints while looking for occurrences of a given sequence: minimal and maximal gap allowed between consecutive elements of an occurrence of the sequence (called *min-gap* and *max-gap*), maximal duration (called *time window*) of the occurrence and time *tolerance* that allows a group of consecutive elements of the occurrence to be merged and treated as a single element.

A *sequential pattern* is a sequence whose statistical significance in D is above user-specified threshold. We consider two alternative measures of statistical significance for sequential patterns: *support* and *number of occurrences*. The support for a sequential pattern $<X_1 X_2 ... X_n>$ in D is the fraction of sequences in D that contain the pattern. While counting the support it is not important how many times a pattern occurs in a given data sequence. This makes support unsuitable when sequential patterns are mined over a single data sequence ($|D| = 1$). In such case, the number of occurrences is more useful as a statistical measure.

1.2 Data Mining Queries

MineSQL [8] is a declarative language for expressing data mining problems by means of *data mining queries*. It serves as a *SQL*-based interface between a client application and a data mining system. In this section we present elements of *MineSQL* that are used later in the paper. The detailed syntax of *MineSQL* can be found in [8].

MineSQL language defines a set of new *SQL* data types, which are used to deal with sequences and sequential patterns. The *SEQUENCE OF* data type family is used to represent sequences of sets of items. Sequences are ordered collections of *(timestamp, value)* pairs, where timestamp is usually of date and time type and value can be of any type. In order to convert a collection of *(timestamp, value)* pairs into a *SEQUENCE OF* value, we use a new *SQL* group function called *SEQUENCE*. The *PATTERN OF* data types family is used to represent sequential patterns and their statistical significance (support or number of occurrences).

MineSQL defines a set of the following *SQL* functions and operators that operate on sequences or patterns: *s CONTAINS t* returning *TRUE* if the sequence or pattern *s* contains the sequence *t*, *SUPPORT(x)* returning the support of the pattern *x*, and *OCCURRENCES(x)* returning the number of occurrences of the pattern *x*.

The central statement of the *MineSQL* language is *MINE*. *MINE* is used to discover frequent itemsets, association rules and sequential patterns from the database. *MINE* also specifies a set of predicates to be satisfied by the returned rules or patterns. In order to discover sequential patterns we use the following syntax of *MINE* statement.

```
MINE pattern_expression [, pattern_expression…]
[WINDOW window][MAXGAP maxgap][MINGAP mingap][TOLERANCE tolerance]
FOR column FROM subquery
WHERE pattern_predicate [AND pattern_predicate…];
```

In the above syntax, *pattern_expression* represents the keyword *PATTERN* or a function operating on *PATTERN* (*PATTERN* represents a single sequential pattern being discovered). The clauses *window*, *maxgap*, *mingap*, and *tolerance* are used to specify time constraints. *Column* is the name of the query column of the type *SEQUENCE OF*, containing sequences to be mined. *Subquery* is the *SQL* subquery, returning the sequences to be mined. *Pattern_predicate* is a Boolean predicate on a function which operates on *PATTERN,* to be satisfied by returned sequential patterns.

The following *MINE* statement uses the table *CUST_TRANSACTIONS(C_ID, T_TIME, ITEM)* to discover all sequential patterns, whose support is greater than 0.1.

```
MINE PATTERN, SUPPORT(PATTERN)
FOR X FROM (SELECT SEQUENCE(T_TIME, ITEM) AS X
            FROM CUST_TRANSACTIONS GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.1;
```

1.3 Related Work

The problem of mining frequent patterns in a set of data sequences together with a few mining algorithms was first introduced in [2]. The class of patterns considered there, called sequential patterns, had a form of sequences of sets of items. The statistical significance of a pattern (called support) was measured as a percentage of data sequences containing the pattern. In [11], the problem was generalized by adding taxonomy on items and time constraints such as min-gap, max-gap and sliding window (in this paper called tolerance).

Another formulation of the problem was given in [7], where discovered patterns (called episodes) could have different type of ordering: full, none or partial and had to appear within a user-defined time window. The episodes were mined over a single event sequence and their statistical significance was measured as a percentage of windows containing the episode (frequency) or as a number of occurrences.

In [10], an issue of incremental and interactive sequence mining was addressed. An algorithm was proposed for finding sequential patterns in the expanded database using the old frequent patterns. Another contribution was a method for handling interactive sequential pattern queries. The goal was achieved by adding a preprocessing step that consisted in discovering patterns with a low support threshold and storing them in a form suitable for efficient retrieval according to user-specified query conditions.

Another approach to incremental mining of sequential patterns was presented in [12]. The algorithm introduced there was applicable not only for expanded but also for reduced database. The algorithm required some extra information to be stored together with the discovered patterns.

In [9], an issue of interactive mining of association rules [1] was addressed and the concept of *knowledge cache* was introduced. The cache was designed to hold frequent itemsets that were discovered while processing other queries. An important contribution was an algorithm, which used itemsets discovered for higher support

thresholds in the discovery process for the same task, but with a lower support threshold. The frequent itemsets discovered in previous tasks were stored in cache and were used for determining support of some candidate itemsets without checking them against the database. Although the method was proposed in the context of frequent itemsets, it can also be applied to sequential patterns.

The idea of precomputing frequent itemsets in a partitioned database and using them while discovering association rules in the whole database or parts of it was discussed in [13]. The proposed method exploited the property that an itemset can be frequent in the union of partitions if and only if it is frequent in at least one of the partitions. Thus itemsets that were frequent in at least one of the partitions of the mined data set, formed the set of candidates for one verifying database pass.

The notion of data mining queries (or *KDD* queries) was introduced in [5]. The need for Knowledge and Data Management Systems (KDDMS) as second generation data mining tools was expressed. The ideas of application programming interfaces and data mining query optimizers were also mentioned. Several data mining query languages that are extensions of *SQL* were proposed [3][4][6][8].

2 Data Mining Views and Materialized Data Mining Views

In relational database systems views are used to simplify access to frequently used data sets that are results of complex queries. A view presents the results of the *SQL* query hidden in its definition. When a user selects data from a view, its defining query has to be executed. In case of materialized views, results of defining queries are stored in the database, which significantly shortens the response time.

Since data mining tasks are repetitive in nature and the syntax of data mining queries may be complicated, we propose to extend the usage of views to handle both *SQL* queries and *MineSQL* queries. Any *SQL* query concerning a data mining view involves performing the data mining task according to the data mining query that defines the view. This guarantees access to up-to-date patterns but leads to long response times, since data mining algorithms are time consuming.

To address the above issue, we propose materialized data mining views (MDMVs). A materialized data mining view is a database object containing patterns discovered as a result of a data mining query. It contains patterns that were valid at a certain point of time. MDMVs can be used for further selective analysis of discovered patterns with no need to re-run mining algorithms. They can be automatically refreshed according to a user-defined time interval in order to keep the set of patterns up-to-date. In most cases when a MDMV is being refreshed, it can be refreshed efficiently with one of the algorithms for incremental mining.

The following statement creates a MDMV containing all sequential patterns with support greater than 0.1, discovered in the set of sequences from *CUST_TRANSACTIONS* table. The view is to be refreshed once a week.

```
CREATE MATERIALIZED VIEW SEQ_PATTERNS
REFRESH 7 AS MINE PATTERN, SUPPORT(PATTERN)
             FOR X FROM (SELECT SEQUENCE(T_TIME, ITEM) AS X
                         FROM CUST_TRANSACTIONS GROUP BY C_ID)
             WHERE SUPPORT(PATTERN)>0.1;
```

In the defining statement of a data mining view, there are two classes of constraints: database constraints and mining constraints. *Database constraints* are located within the *SELECT* statement in the *FROM* clause of the *MINE* statement. Database constraints are used to apply selection conditions on the source data set that is being mined. *Mining constraints* are located in the *WHERE* clause of the *MINE* statement and are used to specify selection conditions on the set of patterns to be discovered.

3 Discovery of Sequential Patterns in Presence of Materialized Data Mining Views

MDMVs can be also used to reduce execution time of data mining queries, which are not identical to those, on which the views were built. Consider the following example: we are given a materialized data mining view ($MDMV_1$) and a data mining query issued by a user (DMQ_1).

DMQ_1:

```
MINE PATTERN
FOR X FROM
(SELECT SEQUENCE(T_TIME, ITEM)
 AS X FROM CUST_TRANSACTIONS
 GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.1
AND PATTERN CONTAINS
    TO_PATTERN('<(10 20) (30)>');
```

$MDMV_1$:

```
CREATE MATERIALIZED VIEW MDMV1
AS MINE PATTERN
FOR X FROM
(SELECT SEQUENCE(T_TIME, ITEM)
 AS X FROM CUST_TRANSACTIONS
 GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.1;
```

Notice that in order to execute the query, we can simply filter the actual contents of the materialized data mining view $MDMV_1$, without running a data mining algorithm. Thus, MDMVs can play a similar role to data mining queries, as indexes or materialized views do to database queries. Application developers can create MDMVs to transparently decrease execution times of their applications' data mining queries.

We need formal methods for determining data mining query execution plans, which use MDMVs to reduce time complexity. First, we define four relations, which may occur between two data mining queries, DMQ_1 and DMQ_2. We say that:

1. *DMQ_1 extends database constraints* of DMQ_2, if DMQ_1 does one of the following: appends a *WHERE* or *HAVING* clause of database constraints to DMQ_2; appends an additional *ANDed* condition to a *WHERE* or *HAVING* clause of database constraints of DMQ_2; removes an *ORed* condition from a *WHERE* or *HAVING* clause of database constraints of DMQ_2; tightens one or more conditions from a *WHERE* or *HAVING* clause of database constraints of DMQ_2.

2. *DMQ_1 reduces database constraints* of DMQ_2, if DMQ_1 does one of the following: removes a *WHERE* or *HAVING* clause of database constraints from DMQ_2; appends an additional *ORed* condition to a WHERE or HAVING clause of database constraints of DMQ_2; removes an *ANDed* condition from a *WHERE* or *HAVING* clause of database constraints of DMQ_2; relaxes one or more conditions from a *WHERE* or *HAVING* clause of database constraints of DMQ_2.

Example. The following data mining query DMQ_1 extends database constraints of the data mining query DMQ_2 (DMQ_2 reduces database constraints of DMQ_1).

DMQ_1:

```
MINE PATTERN
FOR X FROM
(SELECT SEQUENCE(T_TIME, ITEM)
 AS X FROM CUST_TRANSACTIONS
 WHERE T_TIME > '10-Jan-2000'
 GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.1;
```

DMQ_2:

```
MINE PATTERN
FOR X FROM
(SELECT SEQUENCE(T_TIME, ITEM)
 AS X FROM CUST_TRANSACTIONS
 GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.1;
```

Intuitively, extension of database constraints means narrowing the mined data set whereas reduction of database constraints means extending the mined data set.

3. *DMQ_1 extends mining constraints* of DMQ_2, if DMQ_1 does one of the following: decreases *MAXGAP*, *WINDOW* or *TOLERANCE*, or increases *MINGAP*; appends a *WHERE* or *HAVING* clause of mining predicates to DMQ_2; appends an additional *ANDed* condition to a *WHERE* or *HAVING* clause of mining constraints of DMQ_2; removes an *ORed* condition from a *WHERE* or *HAVING* clause of mining constraints of DMQ_2; replaces *SUPPORT(PATTERN)>x* with *SUPPORT(PATTERN)>y* in DMQ_2, where $x<y$; replaces *OCCURRENCES(PATTERN)>x* with *OCCURRENCES(PATTERN)>y* in DMQ_2, where $x<y$; replaces *PATTERN CONTAINS X* with *PATTERN CONTAINS Y* in DMQ_2, where $X \subset Y$; replaces *PATTERN NOT CONTAINS X* with *PATTERN NOT CONTAINS Y* in DMQ_2, where $Y \subset X$.

4. *DMQ_1 reduces mining constraints* of DMQ_2, if DMQ_1 does one of the following: increases *MAXGAP*, *WINDOW* or *TOLERANCE*, or decreases *MINGAP*; removes a *WHERE* or *HAVING* clause of mining constraints from DMQ_2; appends an additional *ORed* condition to a *WHERE* or *HAVING* clause of mining constraints of DMQ_2; removes an *ANDed* condition from a *WHERE* or *HAVING* clause of mining constraints of DMQ_2; replaces *SUPPORT(PATTERN)>x* with *SUPPORT(PATTERN)>y* in DMQ_2, where $x>y$; replaces *OCCURRENCES(PATTERN)>x* with *OCCURRENCES(PATTERN)>y* in DMQ_2, where $x>y$; replaces *PATTERN CONTAINS X* with *PATTERN CONTAINS Y* in DMQ_2, where $Y \subset X$; replaces *PATTERN NOT CONTAINS X* with *PATTERN NOT CONTAINS Y* in DMQ_2, where $X \subset Y$.

Example. The following data mining query DMQ_1 extends mining constraints of the data mining query DMQ_2 (DMQ_2 reduces mining constraints of DMQ_1).

DMQ_1:

```
MINE PATTERN
FOR X FROM
(SELECT SEQUENCE(T_TIME, ITEM)
 AS X FROM CUST_TRANSACTIONS
 WHERE T_TIME > '10-Jan-2000'
 GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.2;
```

DMQ_2:

```
MINE PATTERN
FOR X FROM
(SELECT SEQUENCE(T_TIME, ITEM)
 AS X FROM CUST_TRANSACTIONS
 WHERE T_TIME > '10-Jan-2000'
 GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.1;
```

Intuitively, extension of mining constraints means narrowing the resulting set of discovered patterns whereas reduction of mining constraints means expanding the resulting set of discovered patterns.

We also define four classes of mining methods, which will be used to execute data mining queries over MDMVs: full mining, incremental mining, complementary mining, and verifying mining. *Full mining* (FM) refers to executing a complete algorithm for discovering frequent patterns (e.g. [11]). This method is used if MDMV's contents cannot support processing of the data mining query. *Incremental mining* (IM) refers to discovering frequent patterns in an incremented data set (e.g. [10]). It can be used for data mining queries which reduce database constraints. *Complementary mining* (CM) refers to discovering frequent patterns using currently materialized patterns which are guaranteed to remain frequent (e.g. [9]). This method can be used for data mining queries which reduce mining constraints. Finally, we have *verifying mining* (VM), that simply consists in pruning those materialized patterns, which do not satisfy mining constraints. It is used for data mining queries, which extend mining constraints.

If two relations occur between a data mining query and a data mining query on which a MDMV is based, then we use the compatibility table (see Table 1) to decide which mining method to use.

Table 1. Compatibility table for using materialized data mining views

	reduction of database constraints	extension of database constraints	-
reduction of mining constraints	CM, IM	FM	CM
extension of mining constraints	VM, IM	FM	VM
-	IM	FM	-

Example. We are given the following data mining query DMQ_1 and the materialized data mining view $MDMV_1$.

DMQ_1:

```
MINE PATTERN
FOR X FROM
(SELECT SEQUENCE(T_TIME, ITEM)
 AS X FROM CUST_TRANSACTIONS
 GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.3;
```

$MDMV_1$:

```
CREATE MATERIALIZED VIEW MDMV1
AS MINE PATTERN
FOR X FROM
(SELECT SEQUENCE(T_TIME, ITEM)
 AS X FROM CUST_TRANSACTIONS
 WHERE T_TIME > '10-Jan-2000'
 GROUP BY C_ID)
WHERE SUPPORT(PATTERN)>0.2;
```

Since DMQ_1 extends mining constraints (higher minimum support) and reduces database constraints (removed WHERE clause) of the data mining query of $MDMV_1$, we perform verifying mining (*VM*), and then incremental mining (*IM*). The verifying mining prunes all materialized patterns, whose support value is not above 0.3, while the incremental mining discovers frequent patterns using the information on frequent patterns discovered in a subset of the mined data set. It was proven in the literature that the execution time of the above mining algorithms would be shorter than when performing full mining.

4 Conclusions and Future Work

In this paper we have presented the concept of materialized data mining views and their application to fast discovery of sequential patterns. We have proposed several rules for optimization of data mining queries in environments, where MDMVs, containing results of other data mining queries are available. These rules can serve as a basis for rule-based data mining query optimizers. An important advantage of the solutions we propose is that the algorithms required to implement our optimization framework have already been introduced and verified.

In the future we plan to address the problem of cost-based data mining query optimization, especially concentrating on situations when there are several MDMVs that can be used to optimize the processing of a given data mining query. Another topic that we plan to discuss is concurrent refreshing of several MDMVs. We believe that in such case, sometimes it might be desirable to combine mining tasks associated with several MDMVs to optimize the global performance of the refresh operation.

References

1. Agrawal R., Imielinski T., Swami A.: Mining Association Rules Between Sets of Items in Large Databases. Proc. of the 1993 ACM SIGMOD Conf. on Management of Data (1993)
2. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th ICDE Conf. (1995)
3. Ceri S., Meo R., Psaila G.: A New SQL-like Operator for Mining Association Rules. Proc. of the 22nd VLDB Conference (1996)
4. Han J., Fu Y., Wang W., Chiang J., Gong W., Koperski K., Li D., Lu Y., Rajan A., Stefanovic N., Xia B., Zaiane O.R.: DBMiner: A System for Mining Knowledge in Large Relational Databases. Proc. of the 2nd KDD Conference (1996)
5. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996)
6. Imielinski T., Virmani A., Abdulghani A.: Datamine: Application programming interface and query language for data mining. Proc. of the 2nd KDD Conference (1996)
7. Mannila H., Toivonen H., Verkamo A.I.: Discovering frequent episodes in sequences. Proc. of the 1st KDD Conference (1995)
8. Morzy T., Wojciechowski M., Zakrzewicz M.: Data Mining Support in Database Management Systems. Proc. of the 2nd DaWaK 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)
10. Parthasarathy S., Zaki M.J., Ogihara M., Dwarkadas S.: Incremental and Interactive Sequence Mining. Proc. of the 8th CIKM Conference (1999)
11. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th EDBT Conference (1996)
12. Wang K., Tan J.: Incremental Discovery of Sequential Patterns. ACM-SIGMOD's 96 Data Mining Workshop: On Research Issues on Data Mining and Knowledge Discovery (1996)
13. Wojciechowski M., Zakrzewicz M.: Itemset Materializing for Fast Mining of Association Rules. Proc. of the 2nd ADBIS Conference (1998)