Mining various patterns in sequential data in an SQL-like manner*

Marek Wojciechowski

Poznan University of Technology, Institute of Computing Science,
ul. Piotrowo 3a, 60-965 Poznan, Poland
Marek.Wojciechowski@cs.put.poznan.pl

Abstract. One of the most important data mining tasks is discovery of frequently occurring patterns in sequences of events. Many algorithms for finding various patterns in sequential data have been proposed recently. Researchers concentrated on different classes of patterns, which resulted in many different models and formulations of the problem. In this paper a uniform formulation of the problem of mining frequent patterns in sequential data is provided together with an SQL-like language capable of expressing queries concerning all classes of patterns. An issue of materializing discovered patterns for further selective analysis is also addressed by introducing a concept of knowledge snapshots.

1 Introduction

One of the most important data mining problems is discovery of frequently occurring patterns in sequences of events. The problem was first introduced in [3], and then generalized in [4], as a problem of mining *sequential patterns* in a set of sequences. Another approach was presented in [5] and [6] where various types of patterns called *episodes* where mined in one sequence of events. Many algorithms for finding various patterns in sequential data have been proposed, optimized for particular classes of patterns considered interesting in particular applications. We claim that all data mining algorithms should be integrated into a powerful universal data mining engine and users should be provided with a language capable of specifying all data mining tasks including discovery of frequent patterns in sequential data.

As it was stated in [11], such a language could also serve as an Application Programming Interface (API) for building business applications involving knowledge discovery. One of the approaches was to develop such a language as an extension of SQL. Several extensions of SQL were proposed to handle association rules queries [11][12][13][14]. For the problem of mining association rules first formulated in [1], an algorithm integrating user-defined constraints in the process of mining to reduce the execution time was presented [2]. Such a possibility proves that a query language can be used not only to select the desired subset of patterns from a larger set of pre-

* This work was partially supported by the grant no. KBN 43-1309 from the State Committee for Scientific Research (KBN), Poland.

discovered patterns but also to guide the mining algorithm in order to improve performance.

As for mining patterns in sequential data, no language capable of specifying all variations of the problem has been proposed. One of the reasons for that is probably the lack of the uniform formulation of the problem. Researchers concentrated on different classes of patterns, which resulted in many different models. Some kind of a language, based on logical predicates designed for specifying the structure of requested patterns, was presented in [7].

In this paper, a uniform formulation of the problem of mining frequent patterns in sequential data is provided together with an SQL-like language capable of expressing queries concerning all classes of patterns. We refer to patterns in sequential data as to *episodes*, following the terminology from [5]. An episode is defined as a collection of events that are consistent with a given partial order [5]. As input data, we consider sequential data that can be seen as a sequence or a set of sequences of events. We assume that the input data is in a relational form (a set of tuples describing events) and whether it is seen as one event sequence or a set of sequences depends on the user's interpretation.

The language presented in the paper is designed as an extension of MineSQL language, which was introduced in [14] as a query language for mining various kinds of rules. We assume that a user can mine patterns (episodes) having different type of ordering, occurring in a single event sequence or in a set of sequences and satisfying all sorts of constraints that current algorithms can handle.

An issue of materializing discovered patterns for further selective analysis is also addressed in this paper by introducing a concept of knowledge snapshots as an alternative to the solution presented in [14], where discovered rules were to be stored directly in database tables. The idea for knowledge snapshots comes from table snapshots used in Oracle mostly to replicate data from remote tables in a distributed environment [15]. We introduce the idea of intelligent storage objects called knowledge snapshots intended to contain discovered knowledge of any form that was valid at a certain point in time.

1.1 Related Work

The problem of mining frequent patterns in a set of data sequences together with a few mining algorithms was first introduced in [3]. 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 [4], the problem was generalized by adding taxonomy (is-a hierarchy) 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 [5], where discovered patterns (called episodes) could have different type of ordering: full (serial episodes), none (parallel episodes) 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 [6], the model was extended to handle episodes described by a set of unary and binary predicates on event attributes. In [7], a language capable of specifying episodes of interest based on logical predicates was presented and a few extensions to the model were added.

In [8], the issue of integrating data mining with current database management systems was addressed and a concept of KDD queries was introduced. Several extensions of SQL were presented to handle association rules queries [11][12][13][14]. In addition to the query language, in [14] a framework for materializing discovered knowledge in a relational database was provided.

The idea of integrating data mining with relational databases by formulating mining tasks as SQL queries in order to exploit optimization techniques offered by relational database management systems was discussed in [9] and [10].

1.2 Organization of the Paper

The paper is organized as follows. We give a uniform formulation of the problem of mining frequent episodes in sequential data in Section 2. In Section 3 we present the MINE EPISODE statement as an extension of the data mining query language MineSQL introduced in [14]. Section 4 describes in detail a concept of knowledge snapshots intended for storing discovered knowledge for further analyses. Section 5 contains final conclusions and discusses possible future work.

2 Mining frequent episodes in sequential data

In this section we generalize the problem of mining frequent episodes in sequential data in the following way:

We are given an event sequence in a form of a set of tuples. Each tuple corresponds to a single event and is described by a set of event attributes R, a set of *sequence partitioning attributes* $P \subset R$ and an *ordering attribute* $T \in R$. An ordering attribute determines how tuples should be sorted to form a sequence. A set of sequence partitioning attributes allows to treat the input data as a set of event sequences (if $P = \emptyset$ then we are dealing with a single event sequence).

The goal is to find all episodes having desired type of *ordering*, a user-defined *statistical significance*, satisfying user-defined *ordering conditions* and *content conditions*, taking into account user-defined taxonomies on event attributes and user-defined *time constraints*.

In general, we assume partial order on events within an episode but we allow a user to specify that the discovered episodes should be *serial* or *parallel*. An episode is said to be serial when for each pair of events within the episode either one of them precedes the other or they occur simultaneously. An episode is parallel if no constraints are imposed on the order in which events forming an episode should occur.

We assume that a user specifies the requested statistical significance of mined episodes by means of expressions involving one of the following statistical measures: *support*, *frequency* or *number of occurrences*. Support is a percentage of input se-

quences containing a given pattern. Frequency can be measured only if a user specifies a size of the time window within which an episode has to occur. Its value is the percentage of windows containing a given episode.

Ordering and content conditions are used to specify the requested structure of mined episodes. By content conditions we mean arbitrary predicates on events forming an episode, concerning their attributes. Ordering conditions are used to specify that within a given episode one event should precede some other event or that two events should occur simultaneously.

Time constrains determine what should be taken into account while checking whether a given episode occurs in a given input sequence or not. We consider the following time constraints: *window-size*, *max-gap*, *min-gap* and *tolerance*. Window-size is used to specify that all events forming an episode have to occur within a given time period. Max-gap, min-gap and tolerance constraints can be used only in mining serial episodes and they mean maximum and minimum time gap allowed between consecutive events and the maximum time gap between two events when they can still be treated as occurring simultaneously.

3 SQL-like language for mining patterns in sequential data

In MineSQL a user specifies the requested class of patterns by means of various forms of the MINE statement. Here we present the MINE EPISODE statement capable of specifying all variations of data mining tasks concerning patterns in event sequences mentioned in the previous section. The MINE EPISODE statement has the following syntax:

```
MINE EPISODE episode_expr [, ...]
[ORDERING {FULL|PARTIAL|NONE}]
FROM {table|SQL_select_statement}
ON attribute [USING tax_name][, ...]
SORTED BY attribute
[PARTITIONED BY attribute [, ...]]
[MAXGAP numeric_expr]
[MINGAP numeric_expr]
[TOLERANCE numeric_expr]
[WINDOW numeric_expr]
[CONTAINING EVENTS alias1 [, ...]]
[WHERE {order_condition|stat_condition
|content_condition}
[{AND|OR} {order_condition|stat_condition
|content_condition}] ...]
[ORDER BY episode_expr [{ASC|DESC}]]
```

In the above syntax *episode_expr* denotes an episode expression and can be a pseudo-attribute *EPISODE* or any of allowed statistical metrics applied to the *EPISODE* pseudo-attribute, that is *support(EPISODE)*, *frequency(EPISODE)* or *occurrences(EPISODE)*. Episode expressions are used in the main clause of the statement to determine what should appear as the output and in the *ORDER BY* clause to deter-

mine how the results should be sorted. The clause *ORDERING* is used to specify the type of ordering that the discovered set of episodes should be consistent with. The *FROM* clause specifies the source data for the mining process, which can be a single table or the result of an SQL query. The *ON* clause determines which attributes are to be used as a description of an element of the pattern, with a possibility of specifying taxonomy on each attribute (*tax_name* denotes the name of a taxonomy object, each taxonomy object contains one is-a hierarchy). The *SORTED BY* clause determines the attribute according to which the input data should be sorted to form a sequence. The *PARTITIONED BY* clause is used to specify that the input data should be treated not as a single event sequence but as a set of sequences (each sequence is formed from tuples sharing the same values of all attributes listed in this clause). The *MAXGAP*, *MINGAP* and *TOLERANCE* clauses can appear only if full ordering was requested and they refer to max-gap, min-gap and tolerance time constraints respectively. When any of the above three clauses is omitted, the corresponding time constraint is not taken into account in the mining process. The clause *WINDOW* is used to specify that events forming an episode should occur within a given time window. The clause *WHERE* is used to specify thresholds for statistical measures (support, frequency and number of occurrences) as well as the structure of required episodes by means of ordering and content conditions. In ordering and content conditions, events are referred to by means of aliases specified in the *CONTAINING EVENTS* clause.

Let us consider the following example. We are given a database of telecommunication network alarms in a form of the table ALARMS(alarm_type, module, time). Let us assume that a user wants to find all episodes (partially ordered) occurring within an hour, involving occurring of the alarm of type 2134 before the alarm of type 1015 in the same module, having frequency higher then 1%. This leads to the following query:

```
MINE EPISODE episode, frequency(episode)
FROM alarms
ON alarm_type, module
SORTED BY time
WINDOW 1/24
CONTAINING EVENTS a, b
WHERE a.alarm_type = 2134
AND b.alarm_type = 1015
AND a.time < b.time
AND a.module = b.module
AND frequency (episode) >= 0.01
ORDER BY frequency(episode);
```

4 Knowledge snapshots

A knowledge snapshot is a database object containing knowledge in form of rules, patterns, etc., discovered as a result of a KDD query. It represents the knowledge that was valid at a certain point of time. The main purpose of knowledge snapshots is materialization of mining results for further selective analysis instead of running

mining algorithms each time when a user wants to browse previously discovered knowledge.

Knowledge snapshots can be refreshed on demand or automatically according to a user-defined time interval. This might be useful when a user is interested in a set of rules or patterns, whose specification does not change in time, but always wants to have access to relatively recent information.

Information about knowledge snapshots should be stored in a database's data dictionary. A knowledge snapshot is described by its unique name, defining KDD query, time of last refresh, and optional time interval according to which a snapshot should be automatically refreshed. Access to such descriptions of knowledge snapshots is necessary to interpret the contents of a given knowledge snapshot.

Knowledge snapshots are created by means of the CREATE KNOWLEDGE SNAPSHOT statement and dropped by means of the DROP KNOWLEDGE SNAPSHOT statement. Creation of a new knowledge snapshot always involves knowledge discovery. A user can specify that a knowledge snapshot should be refreshed automatically but can also request an immediate refresh by means of the REFRESH KNOWLEDGE SNAPSHOT statement. We propose the following syntax of the CREATE KNOWLEDGE SNAPSHOT statement:

```
CREATE KNOWLEDGE SNAPSHOT knowledge_snap_name
[REFRESH time_interval]
AS mine_statement
```

In the above syntax *knowledge_snap_name* is the name of a knowledge snapshot, *time_interval* denotes the time interval between two consecutive refreshes of the knowledge snapshot, and *mine_statement* denotes any variation of the MINE statement. The *REFRESH* clause is optional since a user might not want a knowledge snapshot to be refreshed automatically.

To allow a user to select the desired set of patterns from those stored in a knowledge snapshot, we propose to permit a user to specify the name of a knowledge snapshot instead of the name of a table or SQL query in the *FROM* clause of the MINE statement. Execution of a MINE statement against a knowledge snapshot should generate an error when a set of requested patterns is not guaranteed to be a subset of the set of patterns stored in the knowledge snapshot. Moreover, only clauses that specify pattern selection constraints or affect the presentation of results should be allowed (in case of the MINE EPISODE statement: *CONTAINING EVENTS*, *WHERE* and *ORDER BY*).

5 Conclusions

In this paper, a uniform formulation of the problem of mining frequent patterns in sequential data covering all classes of patterns proposed in previous works has been provided. Based on this formulation, an extension to the SQL-like data mining language MineSQL capable of specifying queries concerning mining patterns in sequential data has been presented. A novel, general approach to materializing data mining results has also been proposed by introducing a concept of knowledge snapshots.

The biggest challenge for future work is probably implementation of knowledge snapshots. One of the questions is where the discovered knowledge should be stored. The solution presented here assumed storing discovered patterns in the same database where the input data was stored but it should be taken into account that data mining algorithms are often run on data warehouses instead of operational databases. Another interesting problem concerning knowledge snapshots is how to refresh them automatically. One of the possible approaches is to dedicate this task to a special background process operating as a part of the data mining system.

References

1. Agrawal R., Imielinski T., Swami A.: Mining Association Rules Between Sets of Items in Large Databases. Proc. of the ACM SIGMOD Conference on Management of Data (1993).
2. Srikant R., Vu Q., Agrawal R.: Mining Association Rules with Item Constraints. Proc. of the 3rd Int'l Conference on Knowledge Discovery and Data Mining (1997).
3. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th Int'l Conference on Data Engineering (1995).
4. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th Int'l Conf. on Extending Database Technology (1996).
5. Mannila H., Toivonen H., Verkamo A.I.: Discovering frequent episodes in sequences. Proc. of the 1st Int'l Conference on Knowledge Discovery and Data Mining (1995).
6. Mannila H., Toivonen H.: Discovering generalized episodes using minimal occurrences. Proc. of the 2nd Int'l Conference on Knowledge Discovery and Data Mining (1996).
7. Guralnik V., Wijesekera D., Srivastava J.: Pattern Directed Mining of Sequence Data Proc. of the 4th Int'l Conference on Knowledge Discovery and Data Mining (1998).
8. Imielinski T., Mannila H.: A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11 (1996).
9. Sarawagi S., Thomas S., Agrawal R.: Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. Proc. of the ACM SIGMOD International Conference on Management of Data (1998).
10. Thomas S., Sarawagi S.: Mining Generalized Association Rules and Sequential Patterns Using SQL Queries. Proc. of the 4th Int'l Conference on Knowledge Discovery and Data Mining (1998).
11. Imielinski T., Virmani A., Abdulghani A.: Datamine: Application programming interface and query language for data mining. Proc. of the 2nd Int'l Conference on Knowledge Discovery and Data Mining (1996).
12. Ceri S., Meo R., Psaila G.: A New SQL-like Operator for Mining Association Rules. Proc. of the 22nd Int'l Conference on Very Large Data Bases (1996).
13. 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 Int'l Conference on Knowledge Discovery and Data Mining (1996).
14. Morzy T., Zakrzewicz M.: SQL-like Language for Database Mining. ADBIS'97 Symposium (1997).
15. Oracle Corporation: Oracle8™ Server Replication. Oracle8 Documentation (1997).