BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 62, No. 2, 2014
DOI: 10.2478/bpasts-2014-0032

Formal model of time point-based sequential data for OLAP-like analysis

B. BĘBEL*, T. MORZY, Z. KRÓLIKOWSKI, and R. WREMBEL

Institute of Computing Science, Poznan University of Technology, 3a Piotrowo St., 90-965 Poznań, Poland

Abstract. Numerous nowadays applications generate huge sets of data, whose natural feature is order, e.g,. sensor installations, RFID devices, workflow systems, Website monitors, health care applications. By analyzing the data and their order dependencies one can acquire new knowledge. However, nowadays commercial BI technologies and research prototypes allow to analyze mostly set oriented data, neglecting their order (sequential) dependencies. Few approaches to analyzing data of sequential nature have been proposed so far and all of them lack a comprehensive data model being able to represent and analyze sequential dependencies. In this paper, we propose a formal model for time point-based sequential data. The main elements of this model include an *event* and a *sequence* of events. *Measures* are associated with events and sequences. Measures are analyzed in the context set up by *dimensions* in an OLAP-like manner by means of the set of operations. The operations in our model are categorized as: operations on sequences, on dimensions, general operations, and analytical functions.

Key words: OLAP, sequential data, sequential OLAP, formal model.

1. Introduction

On-line transaction processing systems (OLTP), sometimes called production data sources, generate data that are further analyzed by business intelligence (BI) systems [1] for the purpose of decision support. A typical BI system architecture is based on a central database that stores consolidated data from multiple OLTP systems. This central database is called a data warehouse (DW). Data in the DW are analyzed by means of the so-called on-line analytical processing applications (OLAP). DW data are organized into the so-called facts and dimensions. *Facts* represent data being analyzed, whereas *dimensions* set up a context for an analysis. Facts have features called *measures*, typically numerical, that quantify the facts.

Various DW architectures are available nowadays. In the typical ones, a DW is located on a secondary storage (e.g., Oracle11g, SybaseIQ, IBM DB2, and SQL Server). For increasing the performance of data analysis, main-memory (in-memory) DWs can be applied, where all the analyzed data reside in main memory (e.g., Targit X-bone Server, Oracle Exadata, SAP Hana, and IBM Netezza).

Nowadays, analytical needs extend beyond traditional data analysis (character strings, dates, and numbers). A plethora of devices and applications generate streams of data by means of RFID devices and sensors (e.g., good transportation monitoring, public transportation infrastructures, intelligent buildings, crude refineries, gas delivery pipelines, remote media consumption measurement). Such data form a stream whose an inherent feature is ordering, typically by time. Other types of applications that generate ordered data include for example stock exchange quotations, workflow management systems, web server logs with clickstream, and patient treatment records.

RFID technology is becoming widely used in supply chain management (e.g., just-in-time delivery) as well as in optimizing product transportation routes. RFID devices generate sequences of records identifying their current position and thus, allowing to be monitored on-line [2]. Some route changes may be then made on-line. In advanced public transportation infrastructures, e.g., Washington, Hong Kong [3–6] passengers use intelligent cards. Travels are automatically detected by various devices, generating tracking records. These records are next used for billing the passengers. They can also be used for analyzing the most frequently used routes and, thus, for discovering route bottlenecks, station bottlenecks, and rush hours in various districts. In intelligent installations, numerous sensors supply their data [7–10]. Based on the chronologically analyzed sensor data, one can discover fluctuation of a temperature within a given time period, discover idle time periods (e.g., heating in an intelligent building) or discover unnecessary power consumption (e.g., lights turned on). In workflow systems, objects arrive to a given task at certain points of time, they are processed there for a certain period of time, and leave the task for another task. The order of object movement in a workflow is important and it is part of the definition of the workflow. Thus, being able to analyze the workflow log data and explore orders between objects processing is an important requirement. For example, one may be interested in finding the average time an object spends in the path which connects two tasks (e.g., from task t_1 to task t_5). In Web log analysis, especially for e-commerce, an analysts may be interested in knowing the navigation path where a user (a potential buyer) spends the most of his/her time. The process of disease curing has also sequential nature represented by drug application, treatment, and the results of a therapy. Doctors may be interested in analyzing blood pa-

*e-mail: Bartosz.Bebel@put.poznan.pl

rameters [11] and correlate them with a chronological order of drug application.

Some of the data generated by the aforementioned applications and systems have the character of events that last an instant - a chronon [12], whereas some of them last for a given time period - an interval, but for all of them the order they were generated in is important. With this regard, sequential data can be categorized either as *time-point based* or *interval based* ones [13].

The data streamed into an analysis engine can be processed by means of traditional OLAP applications, however, by exploiting the sequential (ordering) nature of the data an analyst could mine a valuable additional knowledge. Unfortunately, traditional commercial and research BI architectures, although very advanced ones, allow to analyze set oriented data, but they are not capable of exploiting the existing order among the data. The research and technological advancements in the area of sequential data analysis have just been launched. Some extensions to traditional OLAP functionality have been proposed in the research literature, commonly known as Sequential OLAP (S-OLAP) [6, 14, 15].

The advancements, e.g., [6, 14–17] focus on analyzing time-point based sequential data. Conversely, [2, 18] focus on interval based sequential data but only from a storage point of view. Unfortunately, all of them lack a comprehensive data model being able to represent and analyze sequential dependencies. With this respect, there is an evident need for developing a formal model and a query language capable of analyzing such data.

Paper contribution. In this paper we focus on time-point based sequential data and we substantially extend our initial proposal of a data model for analyzing sequential data, presented in [19]. The data model that we contribute in this paper is based on the notion of an *event* and a *sequence* of events. Similarly as in the traditional OLAP, *measures* (associated with events and sequences) are analyzed in the context of *dimensions* by means of operations. The operations in our model are categorized as: operations on sequences, on dimensions, general operations, and analytical functions. Unlike the traditional OLAP, sequences are the elements that need to be created on demand from events. The same set of events may form various sequence sets, depending on an analysis. Moreover, measures that are associated with sequences need to be defined on demand as well. These features make processing sequential data challenging.

The rest of this paper is organized as follows. Section 2 defines a leading example used throughout the paper. Section 3 contributes the formal model of time point-based sequential OLAP. Section 4 outlines approaches related to storing and processing sequential data. Finally, Sec. 5 summarizes the paper and points out the directions for future work.

2. Leading example

As an example used throughout the paper let us consider data about cars failures and their repairs. We assume that initially the data are stored in a table composed of 8 attributes, as shown in Table 1. For each car failure a date of the failure and its description are stored. A car is described by its identifier, year of production, and mileage, while a repair is characterized by its description, cost, and name of a car service where the car was repaired.

In terms of data warehousing, the table stores fact data about car failures and repairs. Attributes *repair cost* and *car mileage* represent measures, whereas the other ones represent dimensions. The dimensions are defined as follows:

- **vehicle** with the hierarchy: *car identifier* → *model* → *make*,
- **failure type** with the hierarchy: *failure description* → *significance level*,
- **shop** with the hierarchy: *shop name* → *city*,
- **time** with the hierarchy: *failure date* → *month* → *quarter* → *year*.

Table 1
Example data on cars failures and their repairs

failure date	car identifier	production year	car mileage	failure description	repair description	repair cost	shop name
2012.04.04	BB111	2003	145 500	F1	R11	1 500	P1
2012.06.11	BB111	2003	160 000	F2	R21	800	P1
2012.06.12	AA222	2004	184 000	F3	R31	2 100	P2
2012.06.13	CC333	2007	80 000	F2	R22	790	P2
2012.07.27	BB111	2003	179 000	F3	R32	2 200	P1
2012.12.02	AA222	2004	201 123	F4	R41	650	P3
2012.12.08	CC333	2007	120 000	F4	R42	660	P2
2012.12.13	DD444	2005	110 000	F1	R12	1 400	P2
2013.01.30	EE555	2000	190 000	F3	R32	1 900	P1
2013.02.16	DD444	2005	121 000	F2	R21	850	P2
2013.06.10	EE555	2000	194 000	F4	R42	780	P1

According to the aforementioned definitions, the data can be organized as a logical data cube, cf. Fig. 1. The cube allows to analyze *repair cost* and *car mileage* in the context of: **vehicle**, **failure type**, **shop**, **time**.

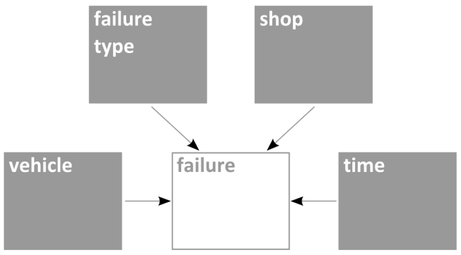

Fig. 1. A logical data cube for car repairs

Conventional OLAP analysis of the data cube could focus on: (1) finding the average mileage of Peugeots when transmission breaks, (2) finding the percentage of cars with respect to car models and production dates with major failures during the first three years of usage, or (3) finding car services competences, namely the top-3 failures most often repaired by a car service.

However, if we took into account the sequential dependencies between data, some new, non-typical analyses could be performed and a valuable information could be discovered. For example, we could: (1) find the percentage of cars in which failure *F1* occurs within 6 months after failure *F2*, or (2) find the most common ”pattern” of failures occurring in cars in their fifth and subsequent years of exploitation, or (3) find an average mileage distance between failures *F1* and *F2*, with respect to cars models and years of production, or (4) find ”hot spots” in car model life, i.e., mileage balance when the probability of a given failure is the highest.

We argue that in order to perform these and many other analyses a new data model and a query language are needed. For this reason, we developed a formal model for analyzing sequential data in an OLAP-like manner. The model is discussed in the next section.

3. Data model for sequential OLAP

The data model that we propose comprises *data elements* and *operations*. Both of them are detailed in this section.

3.1. Data elements. The three fundamental data elements of our model include an *event*, a *sequence*, and a *dimension*. A sequence is created from events by clustering and ordering them. Sequences and events have distinguished attributes – *measures* that can be analyzed in an OLAP-like manner in contexts set up by *dimensions*.

Event and its attributes. An *event* represents an elementary data item, whose duration is a chronon. Formally, event $e_i \in \mathbb{E}$, where $\mathbb{E}$ is the set of events. e_i is a n-tuple of attributes' values: $(a_{i1}, a_{i2}, ..., a_{in})$, where a_{ij} is the value of attribute A_j in event e_i. $A_j \in \mathbb{A}$, where $\mathbb{A}$ is the set of event attributes. Value a_{ij} is within the domain of attribute A_j: $a_{ij} \in dom(A_j)$. $dom(A_j) \subseteq \mathbb{V}$, where $\mathbb{V}$ is the set of atomic values (character string, date, number). $\mathbb{V}$ also contains a null value.

Attribute hierarchy. Similarly like in traditional OLAP, event attributes may have hierarchical structures. Let $\mathbb{L} = \{L_1, L_2, ..., L_k\}$ be the set of levels in the hierarchies of the event attributes. Pair $(\mathbb{L}_{A_i}, \rhd_{A_i})$ describes the hierarchy of attribute $A_i \in \mathbb{A}$, where $\mathbb{L}_{A_i} \subseteq \mathbb{L}$ and $\rhd_{A_i}$ is a partial order on set $\mathbb{L}_{A_i}$. The sets of levels values are subsets of $\mathbb{V}$.

Example 1. In the leading example (cf. Sec. 2), an event represents car failure and it is stored as one row in Table 1, i.e.,

$$\mathbb{E} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}\}$$

where

- e_1=(2012.04.04, BB111, 2003, 145 500, F1, R11, 1 500, P1),
- e_2=(2012.06.11, BB111, 2003, 160 000, F2, R21, 800, P1),
- ...
- e_{11}=(2013.06.10, EE555, 2000, 194 000, F4, R42, 780, P1).

The attributes of the events are as follows:

$\mathbb{A}$ = {failureDate, carIdentifier, productionYear, carMileage, failureDescription, repairDescription, repairCost, shopName}

Four attributes, namely A_1, A_2, A_5 and A_8, have the hierarchies defined:

$$\mathbb{L} = \mathbb{L}_{A_1} \cup \mathbb{L}_{A_2} \cup \mathbb{L}_{A_5} \cup \mathbb{L}_{A_8}$$

and the structures of the hierarchies are as follows:

- $\mathbb{L}_{A_1}$ = {failureDate, month, quarter, year} and $\rhd_{A_1}$: failureDate $\rightarrow$ month $\rightarrow$ quarter $\rightarrow$ year (example values: 26.04.2011 $\rightarrow$ April 2011 $\rightarrow$ Q2 2011 $\rightarrow$ 2011),
- $\mathbb{L}_{A_2}$ = {carIdentifier, model, make} and $\rhd_{A_2}$: carIdentifier $\rightarrow$ model $\rightarrow$ make (example values: BB111 $\rightarrow$ 308 $\rightarrow$ peugeot, AA222 $\rightarrow$ Golf VI $\rightarrow$ volkswagen),
- $\mathbb{L}_{A_5}$ = {failureDescription, seriousness} and $\rhd_{A_5}$: failureDescription $\rightarrow$ seriousness (example values: F1 $\rightarrow$ major, F4 $\rightarrow$ minor),
- $\mathbb{L}_{A_8}$ = {shopName, city} and $\rhd_{A_8}$: shopName$\rightarrow$ city (exemplary values: P1 $\rightarrow$ Poznań, P2 $\rightarrow$ Warsaw).

Sequence and its measures. An ordered list of events that fulfill a given condition is called a *sequence*. The order of events in a sequence is defined by values of selected event attributes. Such attributes will further be called *ordering attributes*. A sequence is composed of the events that have the same value of another selected attribute (or attributes). Such attributes will further be called *forming attributes*. If a forming attribute has a hierarchy associated, then a level selected in the hierarchy can also be used as a forming attribute.

Formally, sequence $s_i \in \mathbb{S}$, where $\mathbb{S}$ is the set of sequences, is pair $(\mathbb{E}_i, \rhd)$, where $\mathbb{E}_i \subseteq \mathbb{E}$ and $\rhd$ is a partial order on $\mathbb{E}$.

Creating a sequence. For the purpose of creating the set of sequences from the set of events, we define operator *CreateSequence* as follows:

$$CreateSequence(\mathbb{E}, \mathbb{F}, \mathbb{A}_o, p) = \mathbb{S}$$

where

- $\mathbb{E}$ is the set of elementary events,
- $\mathbb{F}$ is the set of pairs (A_i, L_j), where $A_i \in \mathbb{A}$ is a forming attribute and $L_j \in \mathbb{L}_{A_i}$ is the level in the hierarchy of the forming attribute A_i, or (A_i, ϕ) if attribute A_i does not have a hierarchy,
- $\mathbb{A}_o$ is the set of ordering attributes, $\mathbb{A}_o \subseteq \mathbb{A}$,
- $p \in \mathbb{P}$ is a logical predicate which selects events to form sequences,
- $\mathbb{S}$ is the set of sequences created by the operator.

Notice that sequences are not defined statically, but their structure is dynamically constructed based on the features of analyses, for which the sequences are created. Algorithm 1 describes how *CreateSequence* works. First, events of $\mathbb{E}$ are filtered based on logical predicate p. Then, for each of the filtered events, two vectors are created, namely: (1) vector f that contains values of forming attributes for a given event, and (2) vector o that contains values of ordering attributes for a given event. Finally, the sequences are created as follows: events with the same vectors f form one sequence, events in the sequence are ordered by vectors o. Example 2 presents three different sequence sets, created by *CreateSequence*.

Algorithm 1: *CreateSequence*

```
Input: $\mathbb{E}, \mathbb{F}, \mathbb{A}_o, p$
Result: $\mathbb{S}$
$\mathbb{S} \longleftarrow \phi$; $\mathbb{C} \longleftarrow \phi$ /* $\mathbb{C}$ – set of triples */
foreach event $e \in \mathbb{E}$ do
    if $p(e)$ is true then
        $f \longleftarrow \phi$; /* $f$ – $|\mathbb{F}|$-element vector */
        $o \longleftarrow \phi$; /* $o$ – $|\mathbb{A}_o|$-element vector */
        $k \longleftarrow 0$; /* variable for indexing $f$ and $o$ */
        foreach $(A_i, L_j) \in \mathbb{F}$ do
            $k \longleftarrow k+1$
            if $L_j \neq \phi$ then
                $f[k] \longleftarrow$ value of $A_i$ in $e$ at level $L_j$
            end
            else
                $f[k] \longleftarrow$ value of $A_i$ in $e$
            end
        end
        $k \longleftarrow 0$
        foreach $A_m \in \mathbb{A}_o$ do
            $k \longleftarrow k+1$
            $o[k] \longleftarrow$ value of $A_m$ in $e$
        end
        $\mathbb{C} \longleftarrow \mathbb{C} \cup \{(e, f, o)\}$
    end
end
forall $(e, f, o) \in \mathbb{C}$ do
    put all $e$ with the same $f$ in order of $o$ into one
    sequence $s$
    $\mathbb{S} \longleftarrow \mathbb{S} \cup \{s\}$
end
```

Example 2. For the purpose of analyzing failure history of particular cars, all events describing failures of a given car are included into one sequence. They are further ordered by date when the failure occurred, as shown below.

$$CreateSequence(\mathbb{E}, \{(A_2, \text{carIdentifier})\}, \{A_1\}, null) = \{s_1, s_2, s_3, s_4, s_5\}$$

where

$$s_1 = \langle e_1, e_2, e_5 \rangle, \quad s_2 = \langle e_3, e_6 \rangle, \quad s_3 = \langle e_4, e_7 \rangle,$$

$$s_4 = \langle e_8, e_{10} \rangle, \quad s_5 = \langle e_9, e_{11} \rangle.$$

In order to analyze common patterns of failures with respect to car makes (e.g., "what is an average mileage of Volkswagens when transmission breaks") one should create sequences with events describing cars of the same make, ordered by mileage. In this case:

$$CreateSequence(\mathbb{E}, \{(A_2, \text{make})\}, \{A_4\}, null) = \{s_1, s_2\}$$

where

$$s_1 = \langle e_1, e_2, e_5, e_3, e_9, e_{11}, e_6 \rangle,$$
$$s_2 = \langle e_4, e_8, e_7, e_{10} \rangle.$$

In order to take into account 10 years old or younger cars, the *CreateSequence* operator should be extended with a logical predicate as shown below:

$$fCurrentYear - productionYear >= 10$$

where *fCurrentYear* is a function that returns the current year, and the set of sequences is as follows:

$$s_1 = \langle e_1, e_2, e_5, e_3, e_6 \rangle, \quad s_2 = \langle e_4, e_8, e_7, e_{10} \rangle.$$

Fact and measure. Similarly as in traditional OLAP, sequences are characterized by the values of their measures. A measure is denoted as m_i and it is an element of set $\mathbb{M}$ that denotes set of measures ($m_i \in \mathbb{M}$). Values of a measure are the subset of $\mathbb{V}$: $dom(m_i) \subseteq \mathbb{V}$. A measure can be either an attribute of an event or the property of the whole sequence. In order to treat measures uniformly, a measure is defined as function *ComputeMeasure* that associates an atomic value with a sequence, i.e., $ComputeMeasure : \mathbb{S} \times \mathbb{M} \rightarrow \mathbb{V}$. The syntax of the function is as follows:

ComputeMeasure(s, name, p)

where

- $s \in \mathbb{S}$ is a sequence, for which the values of the measure are computed,
- *name* is the name of the measure,
- $p \in \mathbb{P}$ is an expression that computes the values of the measure for a given sequence.

Examples of measures being event's attributes include: *car mileage* and *repair cost*, whereas *total cost of car repairs* is an example of a measure being the property of the whole sequence.

Dimension. A dimension sets up the context of an analysis and defines aggregation paths of facts. Let D_i denote a dimension and $\mathbb{D}$ denote the set of dimensions, thus $D_i \in \mathbb{D}$. A dimension can be either an event attribute or the property of the whole sequence. The *CreateContext* operator associates a dimension with either an event attribute or the whole sequence. It also defines a dimension hierarchy, namely the set of levels and a partial order on this set. The syntax of the operator is as follows:

$$CreateContext(name_{D_i}, A_{D_i}, p_{D_i}, \mathbb{H}_{D_i}) = D_i$$

where

- $name_{D_i}$ is the name of dimension D_i,
- A_{D_i} equals to $A_j \in \mathbb{A}$ if the dimension is an event attribute A_j or $A_{D_i} = \phi$ if the dimension is the property of the whole sequence,
- p_{D_i} equals to predicate $p \in \mathbb{P}$ if the dimension is the property of the whole sequence (p is an expression that computes the values of the dimension) or $p_{D_i} = \phi$ if the dimension is an event attribute (in this case values of an attribute A_{D_i} are taken as dimension values),
- $\mathbb{H}_{D_i}$ is the set of hierarchies of dimension D_i, composed of pairs $(\mathbb{L}_{D_i}, \triangleright_{D_i})$, where $\mathbb{L}_{D_i} \subseteq \mathbb{L}$ is the set of levels in the dimension hierarchy and $\triangleright_{D_i}$ is a partial order on set $\mathbb{L}_{D_i}$; $\mathbb{H}_{D_i} = \phi$ if dimension D_i does not have a hierarchy.

Example 3. An example of a dimension defined by means of attributes include **vehicle** with hierarchy: *carIdentifier* → *model* → *make*. The dimension is set up by the following operator:

$$CreateContext(vehicle, A_2, \phi, \{(\{\text{carIdentifier,model,make}\}, \text{carIdentifier} \rightarrow \text{model} \rightarrow \text{make})\})$$

However, in order to analyze the distribution of numbers of cars failures, the dimension should be defined as a function which calculates the length of a sequence describing failures of a single car. In this case the dimension would be defined as follows:

$$CreateContext(numberOfFailures, \phi, \forall \text{ sequences } s \in \mathbb{S} \text{ find } length(s), \phi)$$

3.2. Operations of the model. The OLAP-like analysis of sequential data is performed in our model by the set of *operations*, detailed in this chapter. The operations are classified into: (1) operations on sequences, (2) general operations, (3) operations on dimensions, and (4) analytical functions.

Operations on sequences. Operations on sequences transform the structure of one or more sequences. The operations include:

1. $First(s)$ – creates a new sequence from sequence $s \in \mathbb{S}$ by removing from s all events except the first one. For example, the analysis "find the oldest failure of car identified by BB111" will be expressed as $First(\langle e_1, e_2, e_5 \rangle) = \langle e_1 \rangle$.
2. $Last(s)$ – creates a new sequence from sequence $s \in \mathbb{S}$ by removing from s all events except the last one. For example, the analysis "find the latest failure of car identified by BB111" will be expressed as $Last(\langle e_1, e_2, e_5 \rangle) = \langle e_5 \rangle$.
3. $Subsequence(s, m, n)$ – creates a new sequence from sequence $s \in \mathbb{S}$ by removing from s all events prior to an event at position m and all events following an event at position n, e.g.,

$$Subsequence(s, length(s) - 1, length(s)) = \langle e_2, e_5 \rangle$$

   where $s = \langle e_1, e_2, e_5 \rangle$.
4. $Split(s, expression)$ – creates the set of new sequences by splitting initial sequence $s \in \mathbb{S}$ using a given *expression*. For example, let us assume that an initial sequence describes the failures of cars of given makes. In order to transform it to sequences describing failures of particular cars, one should use the following expression:

$$Split(\langle e_1, e_2, e_5, e_3, e_9, e_{11}, e_6 \rangle, \text{"the same values of } A_1\text{"}) = \{\langle e_1, e_2, e_5 \rangle, \langle e_3, e_6 \rangle, \langle e_9, e_{11} \rangle\}$$

   In the aforementioned example each element of the original sequence belongs to only one of the output sequences. However, it is not a rule. In some cases (cf. Example 3.2), the original sequence element can be shared by some output sequences.
5. $Combine(\mathrm{S})$ – creates a new sequence with events of all sequences in $\mathrm{S} \subseteq \mathbb{S}$, given as parameters. The events in the new sequence are ordered by the values of ordering attributes of the original sequences. For example, having an initial set of sequences that describe failures of given cars of a given make, the analysis of failures of a given make as a whole would be expressed as follows:

$$Combine(\{\langle e_1, e_2, e_5 \rangle, \langle e_3, e_6 \rangle, \langle e_9, e_{11} \rangle\}) = \langle e_1, e_2, e_5, e_3, e_9, e_{11}, e_6 \rangle$$

   We assume the total order on all events from the set of initial sequences.

Example 4. Let us assume that $F1 - F2 - F3 - F1 - F2$ represents the sequence of failures of a given car within one year. In order to find out how often the "pattern" of car failures occurred: two failures of the same element (e.g., front brakes) within the same year separated by at least one failure of another element (e.g., transmission), one should split $F1 - F2 - F3 - F1 - F2$ into two new sequences, namely: $F1 - F2 - F3 - F1$ and $F2 - F3 - F1 - F2$. Notice that, subsequence $F2 - F3$ is shared by two output sequences.

General operations. General operations allow to manipulate sets of sequences and they include: *SelectSequences*, *SelectEvents*, *GroupBy*, *Join*, and set operations (*union*, *difference*, *intersection*).

1. $SelectSequences(\mathrm{S}, expression)$ – filters sequences in $\mathrm{S} \subseteq \mathbb{S}$ that fulfill a given expression. The result of the operation is the set of sequences $\subseteq \mathbb{S}$. For example, the analysis of failures of the cars with at least four events (failures) would be expressed as follows:

$$SelectSequences(\mathrm{S}, \forall s \in \mathrm{S} : length(s) \geq 4)$$

SelectSequences is explained by Algorithm 2.

2. $SelectEvents(\mathrm{S}, expression)$ – removes from sequences in $\mathrm{S} \subseteq \mathbb{S}$ all events that do not fulfill a given expression. For example, the analysis of failures fixed in a given shop (e.g., P1), would be expressed as follows:

   $SelectEvents(\mathrm{S}, \forall s \in S \wedge e$ being element of $s : e(A_8) = P1)$

   SelectEvents is explained by Algorithm 3.

3. $GroupBy(\mathrm{S}, expression \mid D_i)$ – assigns sequences from $\mathrm{S} \subseteq \mathbb{S}$ to groups according to the results of a given grouping *expression* (case A) or to the values of dimension $D_i \in \mathbb{D}$ (case B). Sequences having the same value of the grouping expression or dimension value belong to the same group. The result of the operation is set $\mathbb{G}$ of pairs $(value, s)$, where:

   - *value* is a given value of grouping expression
   - $s \in \mathrm{S}$ is a sequence with *value* of a grouping expression (case A), or set of pairs $(value(D_i), s)$, where $value(D_i)$ is the value of dimension D_i (case B).

   GroupBy is explained by Algorithm 4.

4. $Join(\mathrm{S}_{in_1}, \mathrm{S}_{in_2},$ *join condition, filtering predicate)* – allows to merge events in sequences from one set (S_{in_1}) with events in sequences from another set (S_{in_2}). The structures of events in sequences from both sets can be different. The operator creates a new set of sequences $\mathbb{S}_{out}$. The structure of events (set $\mathbb{A}_{out}$ of attributes) in the sequences of $\mathbb{S}_{out}$ is a union of events attributes of sequences from both joined sets ($\mathbb{A}_{in_1} \cup \mathbb{A}_{in_2}$).
   The sequences in $\mathbb{S}_{out}$ are created as follows.

   - The operator takes a sequence from S_{in_1} (say, s_1) and adds the attributes from $\mathbb{A}_{in_2}$ to the structure of events in s_1. The values of the added attributes are null.
   - Next, for each event in s_1, the operator finds in sequences from S_{in_2} all events which fulfill the *join condition*. We assume that for each event in s_1 at most one event in sequences from S_{in_2} should be found. If more events in sequences from S_{in_2} fulfill the *join condition*, they should be filtered with the *filtering predicate*. If the *join condition* does not find events in sequences from S_{in_2}, values of events attributes of s_1 corresponding to events attributes of sequences from S_{in_2} are set to null.
   - Finally, sequence s_1 is added to $\mathbb{S}_{out}$. The forming attributes and ordering attributes for sequences in $\mathbb{S}_{out}$ are the same as forming attributes and ordering attributes for sequences in S_{in_1}.

   Join is described by Algorithm 5. Example 5 shows a possible application of the operation.

5. Set operations: union $\cup$, difference $\setminus$, and intersection $\cap$ – they are standard set operations that produce a new set of sequences, e.g., $\mathrm{S}_1 \cup \mathrm{S}_2$.

Algorithm 2: *SelectSequences*

```
Input: $S_{in} \subseteq \mathbb{S}, p$
Result: $S_{out} \subseteq \mathbb{S}$
$S_{out} \longleftarrow \phi$
foreach sequence $s \in S_{in}$ do
    if $p(s)$ is true then
        $S_{out} \longleftarrow S_{out} \cup \{s\}$
    end
end
```

Algorithm 3: *SelectEvents*

```
Input: $S_{in} \subseteq \mathbb{S}, p$
Result: $S_{out}$
$S_{out} \longleftarrow \phi$
foreach sequence $s \in S_{in}$ do
    foreach event $e$ in $s$ do
        if $p(e)$ is false then
            remove $e$ from $s$
        end
    end
    if $length(s) \geq 1$ then
        /* $s$ has at least one event */
        $S_{out} \longleftarrow S_{out} \cup \{s\}$
    end
end
```

Algorithm 4: *GroupBy*

```
Input: $S_{in} \subseteq \mathbb{S}, p$ or $D_i \in \mathbb{D}$
Result: $\mathbb{G}$
$\mathbb{G} \longleftarrow \phi$
foreach sequence $s \in S_{in}$ do
    if $p \neq \phi$ then
        $\mathbb{G} \longleftarrow \mathbb{G} \cup \{(p(s), s)\}$ /* $p(s)$ – value of
        expression $p$ for $s$ */
    end
    else if $D \neq \phi$ then
        $\mathbb{G} \longleftarrow \mathbb{G} \cup \{(D_i(s), s)\}$ /* $D_i(s)$ – value of $D_i$
        for $s$ */

end
```

Algorithm 5: *Join*

```
Input: $S_{in_1} \subseteq \mathbb{S}_{in_1}$, $S_{in_2} \subseteq \mathbb{S}_{in_2}$, join condition $p$,
       filtering predicate $f$
Result: $\mathbb{S}_{out}$
$\mathbb{E}_{out} \longleftarrow \phi$; $\mathbb{A}_{out} \longleftarrow \mathbb{A}_{in_1} \cup \mathbb{A}_{in_2}$; $\mathbb{S}_{out} \longleftarrow \phi$
foreach sequence $s_{in_1} \in S_{in_1}$ do
    create empty sequence $s_{out}$
    foreach event $e_{in_1}$ in $s_{in_1}$ do
        create empty event $e_{out}$ with attributes of $\mathbb{A}_{out}$
        copy values of attributes in $e_{in_1}$ to
        corresponding attributes in $e_{out}$
        $E_{temp} \longleftarrow \phi$
        foreach $e_{in_2}$ in sequences from $S_{in_2}$ do
            if $p(e_{in_1}, e_{in_2})$ is true then
                $E_{temp} \longleftarrow E_{temp} \cup \{e_{in_2}\}$;
            end
        end
        switch $|E_{temp}|$ do
            case 0
                leave values of attributes from $\mathbb{A}_{in_2}$ in
                $e_{out}$ empty
            end
            case 1
                copy values of attributes from
                $e_{temp} \in E_{temp}$ to corresponding
                attributes in $e_{out}$
            end
            otherwise
                find $e_{temp} \in E_{temp}$ with filtering
                predicate $f$
                copy values of attributes from $e_{temp}$ to
                corresponding attributes in $e_{out}$
            end
        end
        $\mathbb{E}_{out} \longleftarrow \mathbb{E}_{out} \cup \{e_{out}\}$;
        $Combine(s_{out}, \langle e_{out} \rangle)$ /* add $e_{out}$ at the end of
        $s_{out}$ */
    end
    $\mathbb{S}_{out} \longleftarrow \mathbb{S}_{out} \cup \{s_{out}\}$
end
```

Example 5. In order to analyze correlation between cars failures and weather conditions, the structure of events describing car failures should be merged with weather information. Let S_1 denote the set of sequences of particular cars failures limited to one sequence describing failures of car identified by *BB111* (sequence s_1 in Example 2). Notice that the structure of events in S_1 is the same as in Example 1. Let S_2 contain sequences that describe monthly weather conditions in 24-hour intervals.

The events structure of sequences in S_2 includes four attributes, namely: date (A_{w1}), day average air temperature (A_{w2}), day average humidity level (A_{w3}), and description of precipitation (A_{w4}). Let's assume that S_2 contains 12 sequences that describe weather conditions in consecutive months of 2012. For example, sequences describing weather conditions in April, June, and July are as follows:

- s_{2_4} = $\langle$(2012.04.01, 13°, 77%, light rains), ..., (2012.04.04, 15°, 70%,cloudy), ...$\rangle$,
- s_{2_6} = $\langle$(2012.06.01, 21°, 80%, sunny), ..., (2012.06.11, 22°, 98%,rain), ...$\rangle$,
- s_{2_7} = $\langle$(2012.07.01, 25°, 50%, sunny), ..., (2012.07.27, 21°, 98%,medium rain), ...$\rangle$.

The structure of events that constitute sequences resulting from the *Join* operator consists of 12 attributes, namely all attributes of the events in sequences from S_1 (A_1 to A_8) and all attributes of the events in sequences from S_2 (A_{w1} to A_{w4}).

The events of the joined sequence sets are matched using the following join condition: the value of A_1 (from S_1) is equal to the value of A_{w1} (from S_2). The output set contains one sequence only, i.e., $s = \langle e_1, e_2, e_3 \rangle$. The events of the sequence are shown below.

- e_1=(2012.04.04, BB111, 2003, 145500, ..., 2012.04.04, 15°, 70%,cloudy),
- e_2=(2012.06.11, BB111, 2003, 160000, ..., 2012.06.11, 22°, 98%,rain),
- e_3=(2012.07.27, BB111, 2003, 179000, ..., 2012.07.27, 21°, 98%,medium rain).

Operations on dimensions. Operations on dimensions allow to navigate in the hierarchy of a given dimension. Although these operations look similar to the standard *drill-down* and *roll-up* OLAP operation, their meaning is different. The operations include: *LevelUp* and *LevelDown*.

1. $LevelUp(D_i, S)$ navigates one level up in the hierarchy of dimension $D_i \in \mathbb{D}$ for all sequences in $S \subseteq \mathbb{S}$.
   An example of this operation may include changing the level of attribute A_8, being a dimension **shop**, from *shop name* to *city*, namely:
   $$LevelUp(shop, \{s_2\}) = \{s_2'\},\ s_2 = \langle e_3, e_6 \rangle,\ s_2' = \langle e_3', e_6' \rangle$$
   where:
   - e_3=(2012.06.12, AA222, 2004, 184000, F3, R31, 2100, **P2**) is transformed to e_3'=(2012.06.12, AA222, 2004, 184000, F3, R31, 2100, **Poznań)** (shop P2 is located in Poznań),
   - e_6=(2012.12.02,AA222,2004,201123,F4, R41, 650, **P3**) is transformed to e_6'=(2012.12.02,AA222,2004,201123,F4, R41, 650, **Warsaw** (shop P3 is located in Warsaw).
2. $LevelDown(D_i, S)$ navigates one level down in the hierarchy of dimension $D_i \in \mathbb{D}$ for all sequences in $S \subseteq \mathbb{S}$.

Algorithm 6: *LevelUp*

```
Input: $S_{in} \subseteq \mathbb{S}, D_i \in \mathbb{D}$
Result: $S_{out}$
$S_{out} \longleftarrow \phi$;
foreach sequence $s \in S_{in}$ do
    change value of $D_i$ in $s$ one level up;
    $S_{out} \longleftarrow S_{out} \cup \{s\}$
end
```

Algorithm 7: *LevelDown*

```
    Input: $S_{in} \subseteq \mathbb{S}, D_i \in \mathbb{D}$
    Result: $S_{out}$
    $S_{out} \longleftarrow \phi$;
    foreach sequence $s \in S_{in}$ do
        change a value of $D_i$ in $s$ one level down;
        $S_{out} \longleftarrow S_{out} \cup \{s\}$
    end
```

Analytical functions. The values of measures are aggregated along aggregations paths defined by dimensions. In our model we support standard OLAP aggregation functions, namely *Avg*, *Count*, *Max*, *Min*, and *Sum*. If we aggregate measures which are the features of whole sequences, the usage of the functions is the same as in traditional OLAP. If, however, one would like to aggregate a measure which is a feature of an event rather than of a sequence, the semantics of the aggregation should be provided. In our model, the semantics is defined by an algorithm. Example 6 illustrates the concept of aggregation semantic encoded in an algorithm.

The operator for aggregation is defined as follows:

$$Aggregate(F, S, m, p)$$

where

- F is an aggregation function, namely $F \in \{Avg, Count, Max, Min, Sum\}$,
- S is a set of sequences, $S \subseteq \mathbb{S}$,
- m is a measure to aggregate, $m \in \mathbb{M}$, this parameter can be a null in case of *Count* function,
- p is an algorithm that defines the semantics of the aggregation.

Example 6. Let us assume that: (1) the sequences being analyzed describe failures of given cars, (2) attribute A_4 (car mileage) of an event represents measure m_1. In order to "find average mileage of cars with a sequence of failures with pattern $F1-F2-F3$", the following steps have to be executed:

- find sequences with failure pattern $F1-F2-F3$,
- remove all events that are not included in the subsequences having pattern F1-F2-F3,
- aggregate the values of mileage (measure m_1) stored in the leading events (at the first position) of the sequences.

In this example, the algorithm that defines the semantics of function *Avg* retrieves the values of m_1 from the first event of every sequence.

3.3. Example application.

In order to illustrate the application of our model to sequential data analysis, let us consider two simple analyses.

Analysis 1. Find the number of cars that broke at least three times in 2008, and the "pattern" of failures was as follows: two failures of the same element were separated by the failure of another element (or elements).

In our model, this analysis is implemented by sequence of the three following operations:

1. create sequences for failures of particular cars which occurred in 2008:

$$\mathbb{S} = CreateSequence(\mathbb{E}, \{(A_2, \text{carIdentifier})\}, \{A_1\}, \forall e_i \in \mathbb{E} : a_{i1} \geq 2008.01.01 \wedge a_{i1} \leq 2008.12.31) \text{ where } a_{i1} \text{ denotes the value of } A_1 \text{ in event } e_i,$$

2. select sequences with at least three failures, two of them concern the same part, and their occurrences are separated by the occurrence of another failure:

$$\mathbb{S}' = SelectSequences(\mathbb{S}, \exists e_i, e_j, e_k \text{ being elements of } s \in \mathbb{S} : a_{i1} < a_{j1} < a_{k1} \wedge a_{i5} \neq a_{j5} \wedge a_{i5} = a_{k5} \wedge a_{j5} \neq a_{k5})$$

3. compute result: $Aggregate(Count, \mathbb{S}', null, null)$

Analysis 2. Find the percentage of cars in which failure F1 follows within 6 months the occurrence of failure F2.

1. create sequences for particular cars:

$$\mathbb{S} = CreateSequence(\mathbb{E}, \{(A_2, \text{carIdentifier})\}, \{A_1\}, \phi)$$

2. eliminate events that do not describe failure F1 or F2:

$$\mathbb{S}' = SelectEvents(\mathbb{S}, \forall e_i \text{ being elements of } s \in \mathbb{S} : a_{i5} = F1 \vee a_{i5} = F2)$$

3. select sequences, in which F1 and F2 occur and time distance between them 6 months or less:

$$\mathbb{S}'' = SelectSequences(\mathbb{S}', \exists e_i, e_j \text{ being elements of } s \in \mathbb{S}' : a_{i5} = F2 \wedge a_{j5} = F1 \wedge a_{i1} + 6 \text{ month} <= a_{j1})$$

4. compute result: $Aggregate(Count, \mathbb{S}'', null, null)/Aggregate(Count, \mathbb{S}, null, null)$

4. Related work

The research and technological areas related to processing sequential data include: (1) complex event processing (CEP) over data streams, (2) OLAP, and (3) sequential pattern mining.

The CEP technology has been developed for the purpose of continuous analysis of data streams to detect patterns, outliers, and generate alerts [20–23]. This technology has been developed for the analysis of current data and is unable to perform OLAP analysis. On the contrary, the OLAP technology [1] has been developed for the purpose of analyzing huge amounts of data organized in relations but it is unable to exploit the sequential nature of data.

With this respect, *Stream Cube* [24] has been developed in order to provide tools for OLAP analysis of stream data. [17] presents more advanced concept, called *E-Cube* allowing to execute OLAP queries on data streams. *E-Cube* includes a query language allowing to query events of a given pattern, a concept hierarchy allowing to compute coarser aggregates based on finer ones, hierarchical storage with data sharing, and a query optimizer.

Sequential data analysis has been researched since several years with respect to storage, e.g., [25–28]. In [27] sequences are modeled by an enhanced abstract data type, in an object-relational model, whereas in [25] sequences are modeled as sorted relations. The query languages proposed in [26,28] allow typical OLTP selects on sequences and do not support OLAP analyzes.

Further extension towards sequence storage and analysis have been made in [29] that proposes a general concept of a RFID warehouse. Unfortunately, no in-dept discussion on RFID data storage and analysis is provided.

[6,14–16] focus on analyzing time-point based sequential data. [6,15,16] focus on storage and analysis of event based sequences. [6] propose the set of operators for a query language for the purpose of analyzing patterns. [15,16] focus on an algorithm for supporting ranking pattern-based aggregate queries and graphical user interface. The drawback of these approaches is that they are based on relational data model and storage for sequential data. Conversely, [14] proposes a formal model for event based sequences and the set of operators for processing and analyzing sequences. The main drawback of this approach is that it does not provide strict meaning of operations defined for a model.

[2,18] focuses on interval based sequential data, generated by RFID devices. The authors propose a few techniques for reducing the size of sequential data, propose techniques for constructing RFID cuboids and computing higher level cuboids from lower level ones. They focus on relational implementation and propose three tables, called Info, Stay, and Map, for storing RFID data and their sequential orders. The proposed approach lacks a formal data model and a query language for analyzing sequences.

Substantial research efforts focused on mining sequential patterns either on data stored in a data warehouse, e.g., [30–32] or on data streams, e.g., [33–35]. The developed algorithms are able to discover patterns of time point based sequences but they do not support typical OLAP-like analyses of sequential data. Therefore, the approaches cover application areas other than the approach proposed in this paper.

5. Summary

In this paper, we presented a formal model suitable for processing time point-based sequential data in an OLAP-like manner. To this end, as data elements, the model uses an *event* and a *sequence*. The values of measures (either event's or sequence's) are analyzed in the context of dimensions, by means of operations. The model supports four classes of operations, namely: on sequences, general, on dimensions, and analytical functions. The unique feature of the model is that both measures and dimensions are created dynamically and may be associated with events or the whole sequences.

Currently we are implementing a prototype system based on our model. Future work will focus on indexing sequences and query optimization.

Acknowledgements. This work was supported from the Polish National Science Center (NCN), the grant No. 2011/01/B/ST6/05169.

REFERENCES

[1] S. Chaudhuri, U. Dayal, and V. Narasayya, "An overview of business intelligence technology", *Communications ACM* 54 (8), 88–98 (2011).

[2] H. Gonzalez, J. Han, and X. Li, "FlowCube: constructing RFID flowcubes for multi-dimensional analysis of commodity flows", *Proc. Int. Conf. on Very Large Data Bases (VLDB)* 1, 834–845 (2006).

[3] "Smart card alliance latin america & the caribbean", http://latinamerica.smartcardalliance.org/ (2012).

[4] "Smart card alliance", http://www.smartcardalliance.org (2012).

[5] "Octopus card", http://hong-kong-travel.org/Octopus/(2012).

[6] E. Lo, B. Kao, W.-S. Ho, S.D. Lee, C.K. Chui, and D.W. Cheung, "OLAP on sequence data", *Proc. ACM SIGMOD Int. Conf. on Management Data* 1, 649–660 (2008).

[7] M. Gorawski, P. Marks, and M. Gorawski, "Collecting data streams from a distributed radio-based measurement system", *Lecture Notes in Computer Science* 4947, 702–705 (2008).

[8] M. Gorawski, "Multiversion spatio-temporal telemetric data warehouse", *ADBIS Workshops* 1, 63–70 (2009).

[9] M. Gorawski, "Extended cascaded star schema and ECOLAP operations for spatial data warehouse", *Lecture Notes in Computer Science* 5788, 251–259 (2009).

[10] S. Szczepański, M. Wójcikowski, B. Pankiewicz, M. Kłosowski, and R. Żaglewski, "FPGA and ASIC implementation of the algorithm for traffic monitoring in urban areas", *Bull. Pol. Ac.: Tech.* 59 (2), 137–140 (2011).

[11] Z. Opilski, G. Konieczny, T. Pustelny, A. Gacek, R. Kustosz, and M. Gawlikowski, "Noninvasive acoustic blood volume measurement system for the POLVAD prosthesis", *Bull. Pol. Ac.: Tech.* 59 (4), 429–433 (2011).

[12] C. Dyreson, F. Grandi, W. Käfer, N. Kline, N. Lorentzos, Y. Mitsopoulos, A. Montanari, D. Nonen, E. Peressi, B. Pernici, J.F. Roddick, N.L. Sarda, M.R. Scalas, A. Segev, R.T. Snodgrass, M.D. Soo, A. Tansel, P. Tiberio, and G. Wiederhold, "A consensus glossary of temporal database concepts", *SIGMOD Record* 23 (1), 52–64 (1994).

[13] F. Mörchen, "Unsupervised pattern mining from symbolic temporal data", *SIGKDD Explor. Newsl.* 9 (1), 41–55 (2007).

[14] B. Bębel, P. Krzyżagórski, M. Kujawa, M. Morzy, and T. Morzy, "Formal model for sequential OLAP", *Information Technology and its Applications* 1, 1–11, ISBN 978-83-89529-82-4 (2011).

[15] C.K. Chui, B. Kao, E. Lo, and D. Cheung, "S-OLAP: an OLAP system for analyzing sequence data", *Proc. ACM SIGMOD Int. Conf. on Management of Data* 1, 1131–1134 (2010).

[16] C.K. Chui, E. Lo, B. Kao, and W.-S. Ho, "Supporting ranking pattern-based aggregate queries in sequence data cubes", *Proc. ACM Conf. on Information and Knowledge Management (CIKM)* 1, 997–1006 (2009).

[17] M. Liu, E. Rundensteiner, K. Greenfield, C. Gupta, S. Wang, I. Ari, and A. Mehta, "E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing", *Proc. ACM SIGMOD Int. Conf. on Management of Data* 1, 889–900 (2011).

[18] H. Gonzalez, J. Han, X. Li, and D. Klabjan, "Warehousing and analyzing massive RFID data sets", *Proc. Int. Conf. on Data Engineering (ICDE)* 1, 83–92 (006).

[19] B. Bebel, M. Morzy, T. Morzy, Z. Królikowski, and R. Wrembel, "OLAP-Like analysis of time point-based sequential data", *ER Workshops Lecture Notes in Computer Science* 7518, 153–161 (2012).

[20] A.P. Buchmann and B. Koldehofe, "Complex event processing", *Information Technology* 51 (5), 241–242 (2009).

[21] M. K. Chandy, "Event-driven applications: costs, benefits and design approaches", http://www.infospheres.caltech.edu/node/38 (2012).

[22] A.J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W.M. White, "Cayuga: a general purpose event monitoring system", *CIDR* 1, 412–422 (2007).

[23] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams", *Proc. ACM SIGMOD Int. Conf. on Management of Data* 1, 407–418 (2006).

[24] J. Han, Y. Chen, G. Dong, J. Pei, B.W. Wah, J. Wang, and Y.D. Cai, "Stream cube: an architecture for multi-dimensional analysis of data streams", *Distributed and Parallel Databases* 18 (2), 173–197 (2005).

[25] R. Ramakrishnan, D. Donjerkovic, A. Ranganathan, K.S. Beyer, and M. Krishnaprasad, "SRQL: Sorted Relational Query Language", *Proc. Int. Conf. on Scientific and Statistical Database Management (SSDBM)* 1, 84–95 (1998).

[26] P. Seshadri, M. Livny, and R. Ramakrishnan, "Sequence query processing", *SIGMOD Record* 23 (2), 430–441 (1994).

[27] P. Seshadri, M. Livny, and R. Ramakrishnan, "The design and implementation of a sequence database system", *Proc. Int. Conf. on Very Large Data Bases (VLDB)* 1, 99–110 (1996).

[28] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi, "Optimization of sequence queries in database systems", *Proc. ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)* 1, 71–81(2001).

[29] S.S. Chawathe, V. Krishnamurthy, S. Ramachandran, and S. Sarma, "Managing RFID data", *Proc. Int. Conf. on Very Large Data Bases (VLDB)* 1, 1189–1195 (2004).

[30] J.-W. Han, J. Pei, and X.-F. Yan, "From sequential pattern mining to structured pattern mining: a pattern-growth approach", *J. Comput. Sci. Technol.* 19 (3), 257–279 (2004).

[31] N.R. Mabroukeh and C.I. Ezeife, "A taxonomy of sequential pattern mining algorithms", *ACM Comput. Surv.* 43 (1), 3:1–3:41 (2010).

[32] F. Masseglia, M. Teisseire, and P. Poncelet, "Sequential pattern mining", in *Encyclopedia of Data Warehousing and Mining*, pp. 1800–1805, IGI Global, London, 2009.

[33] A. Marascu and F. Masseglia, "Mining sequential patterns from data streams: a centroid approach", *J. Intell. Inf. Syst.* 27 (3), 291–307 (2006).

[34] L.F. Mendes, B. Ding, and J. Han, "Stream sequential pattern mining with precise error bounds", *Proc. IEEE Int. Conf. on Data Mining (ICDM)* 1, 941–946 (2008).

[35] Q. Zheng, K. Xu, and S. Ma, "When to update the sequential patterns of stream data?", *Proc. Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD)* 1, 545–550 (2003).