# EventSummarizer: A tool for summarizing large event sequences

[Demo Paper]

Jerry Kiernan
IBM Almaden
San Jose,CA
jkiernan@us.ibm.com

Evimaria Terzi
IBM Almaden
San Jose, CA
eterzi@us.ibm.com

## ABSTRACT

We present EventSummarizer - a tool for extracting comprehensive summaries from large event sequences. EventSummarizer takes as input a sequence with events of different types that occur during an observation period, and creates a partitioning of this time period into contiguous non-overlapping intervals such that each interval can be described by a simple model. Within each interval local associations between events of different types are reported. EventSummarizer runs on top of any Relational DataBase Management System (RDBMS), on tables with a timestamp attribute. Our system is parameter free and has a visual interface that provides the user with a global view of the input sequence via the segmentation of the timeline. The easy-to-use interface provides the user with the option to further examine the activity and associations of event types within each segment.

## 1. INTRODUCTION

Monitoring the activities of systems and users produces large *event sequences*, i.e., logs where each event has an associated time of occurrence. Network traffic and activity logs are examples of large event sequences. Summarization of such data is largely motivated by administrators' pressing need to understand the usage and activity of their systems over time.

The objective of our demonstration is to present EventSummarizer, a flexible tool that showcases the event summarization algorithms that we developed in [3] and demonstrates their effectiveness on real data sets and in different usage scenarios. Thus, this demo paper should be viewed as an extension of our work in [3]. The underpinnings of the methods used in EventSummarizer are described in [3]. Due to space limitations here we only give a high-level description of these methods and mostly focus on the description of our running prototype.

To the best of our knowledge, our methodology developed in [3] is the first to address the need for short and comprehensive summaries that give a global overview of the history of events, as well as local associations among events of different types. Previous data-mining methods for event sequences mostly focus on finding recurring local structures, e.g., episodes (see for example [1, 2, 4, 5, 7]). Moreover, such data-mining algorithms usually output too many patterns which may be overwhelming for data analysts. Our method *summarizes* event sequences instead of focusing on pattern extraction. These summaries exhibit the following characteristics:

- **Brevity and accuracy:** The summaries are *short* and *accurately* describe the input data. This is achieved by using the *Minimum Description Length* (MDL) principle ([6]) that seeks a balance between summaries' length and descriptions' accuracy. Using MDL in its core, EventSummarizer becomes parameter free and thus no extra tuning is required in order to get informative and useful results.
- **Global data description:** The summaries give an indication of the global structure of the event sequence and its evolution through time. This is achieved by using a *segmentation model* that provides a high-level view of the sequence by identifying intervals on the timeline. The events appearing within each interval exhibit local regularity.
- **Local pattern identification:** The summaries reveal information about local patterns. Each interval in the segmented timeline is described by a *local model* similar to clustering. That is, event types with similar rates of appearance within the interval are grouped together.

The above advantages of our method allowed us to develop EventSummarizer into a tool that is extremely *simple to use* by any non-expert. More specifically:

- EventSummarizer is *parameter free* and therefore, no extra knowledge about the specifics of the dataset or the details of the underlying methods is required by the user.
- EventSummarizer has an *easy-to-use interface.* It only assumes familiarity of the user with very basic classical SQL syntax, required for loading the data.

The rest of the interaction between the user and EventSummarizer is based on simple mouseclicks. Finally, the visualization of the output is simple and yet indicative of the trends appearing in the dataset.

- **Cross-platform operation**: EventSummarizer can be used on top of any RDBMS. This *cross-platform* flexibility does not restrict the usage of the tool to a specific domain or application.

## 2. SEGMENTAL GROUPINGS OF EVENT SEQUENCES

In this section we provide a high-level description of the models and algorithms used in order to implement the core functionality of EventSummarizer. The details of these methods are given in [3].

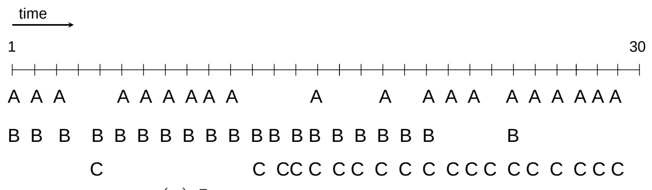
Let $\mathbf{S}$ be an event sequence that records occurrences of events over a time interval $[1, n]$. All the events appearing on $\mathbf{S}$ belong to a finite set of event types. At a high-level, a *segmental grouping* is a model of $\mathbf{S}$ that partitions the observation interval $[1, n]$ into segments of local activity. Within each segment, event types that exhibit similar frequency of occurrence in the segment are grouped together .

Figure 1 shows an example of an input event sequence and the output of our method (that is, a segmental grouping) for this particular sequence. The input sequence is shown in Figure 1(a). The sequence contains events from a set of three event types $\{A, B, C\}$ and it spans timeline $[1, 30]$ that consists of 30 discrete timestamps.

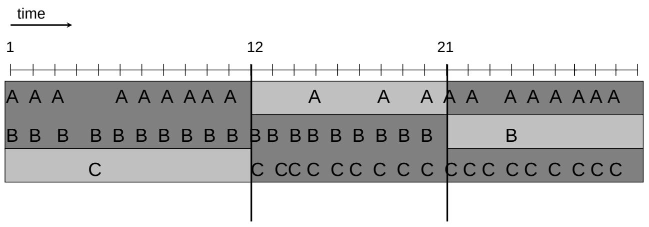
Figure 1(b) shows the actual *segmental grouping* produced by our method. Three segments are identified: $[1 \ldots 11]$, $[12 \ldots 20]$ and $[21 \ldots 30]$. Within each segment, the events are grouped into two groups: event types with similar frequency in a segment are grouped together. In the first segment, the two groups consist of event types $\{A, B\}$ and $\{C\}$; $A$ and $B$ are grouped together as they appear much more frequently than $C$ in the interval $[1 \ldots 11]$. Similarly, the groups in the second and third segment are $\{\{A\}, \{B, C\}\}$ and $\{\{A, C\}, \{B\}\}$ respectively.

Finally, Figure 1(c) pictorially illustrates the concept of segmental grouping. The coloring of the groups within a segment is indicative of the probability of appearance of the events in the group; darker colors correspond to higher occurrence probabilities (higher frequency).

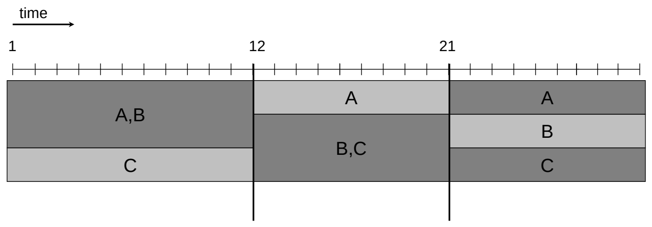
For any given event sequence $\mathbf{S}$ there are exponentially many possible segmental groupings. In [3] we tackled the problem of finding the *best* of them. We achieved that by assigning a cost to every segmental grouping and then solving the corresponding optimization problem. Our cost function was based on the MDL principle. By penalizing both complex and simple models, we developed a parameter-free methodology and provided polynomial-time algorithms that optimally solve the corresponding optimization problem. More specifically, we developed an optimal polynomial-time algorithm that we called `DP-DP`. For a timeline with $n$ timestamps and $m$ distinct event types this algorithm runs in time $O(n^2m^2)$. Additionally, we developed sub-optimal but faster heuristics like `DP-Greedy`, `Greedy-DP` and `Greedy-Greedy` algorithms that run in $O(n^2m \log m)$, $O(n \log nm^2)$ and $O(n \log nm \log m)$ time respectively. The experiments in [3] demonstrated that our algorithms are much faster and their performance with respect to the objective function is not far from the optimal.


time
1
30
A A A A A A A A A A A A A A A A A A A A
B B B B B B B B B B B B B B B B B B B B
C C CC C C C C C C C C C C C C C C C C

(a) Input event sequence



(b) Illustration of the segmental grouping



(c) Output summary

**Figure 1: Visual representation of an event sequence that contains events of three event types $\{A, B, C\}$ and spans timeline $[1, 30]$. Figure 1(a) shows the input sequence; Figure 1(b) shows the segmental grouping and Figure 1(c) shows the high-level view of our summary. Same tone of gray correspond to same group.**

## 3. THE EventSummarizer DEMONSTRATION

In this section we first give an overall review of the EventSummarizer's architecture and then demonstrate its functionality by going through an example run of the system.

### 3.1 System architecture

The EventSummarizer architecture is shown in Figure 2. The system is built in Java and it runs on top of any RDBMS through JDBC. In the examples we give here, we use DB2. Using a Graphical User Interface (GUI), the user can mainly perform two tasks: *load* the data and *summarize* it. The data loading is done using standard SQL queries. The summarization task is then performed on the retrieved data. For the summarization, the user with simple mouseclicks specifies the *summary attribute* that defines the timeline, and the *summarization algorithm* to be used for producing the segmental grouping.

Notice that for segmental groupings to be meaningful, the

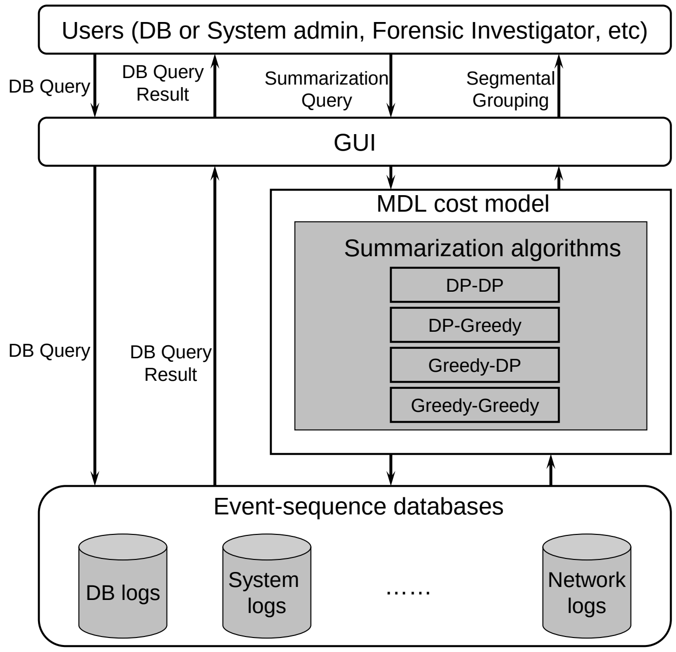


**Figure 2: The EventSummarizer architecture**

summary attribute should define a timeline and thus should be of type `timestamp`. Otherwise, our system simply sorts the tuples on the specified summary attribute. The user can specify the desired algorithm by selecting the appropriate option on EventSummarizer's menu bar. There are four algorithmic options: the `DP-DP`, the `DP-Greedy`, the `Greedy-DP` and the `Greedy-Greedy` algorithms. The details of these algorithms, as well as their respective advantages and disadvantages, are thoroughly discussed in [3]. Once the best segmental grouping of the input sequence is computed, the visual presentation of the results is shown to the user. The results include both the segmentation of the input timeline into intervals and the illustration of the groups within every segment. Details on the graphical illustration of segmental groupings is given in the next section.

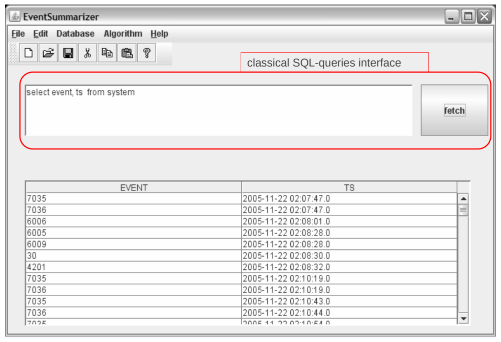
## 3.2 Demonstrated functionality

In this section we give a description of EventSummarizer's functionality and present an illustrative example of how it can be used. The dataset we use here is from the **system log** file displayed by the Windows XP Event Viewer on our machines. The **system log** contains events logged by Windows XP system components. The records of the corresponding **system** table contain four fields: `Type`, `Source`, `Event`, and `TS`. The `TS` attribute is of type *timestamp*.

In the example we will present in the rest of this section, we project the **system** table on attributes `Event` and `TS`. Since `TS` is of type timestamp we use it as a summary attribute. The timestamps appearing in `TS` span the period from November 2005 to November 2007.

Figure 3 shows EventSummarizer's graphical interface. The same figure also shows the part of the interface that allows the user to select the data for summarization using standard SQL queries. The matching tuples to the imposed query (in this case `select event, ts from system` ) are rendered in the lower part of the interface.

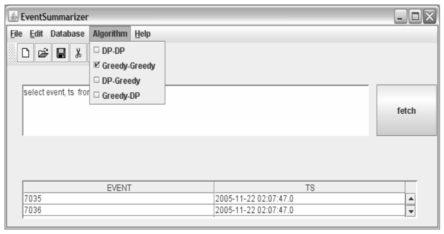
Once the selected data is fetched, the user can then summarize it. The summarization algorithm is selected on the menu bar of EventSummarizer. Figure 4 shows how this selection can be made. In our example, we use the `Greedy-Greedy` summarization algorithm.


EventSummarizer
File Edit Database Algorithm Help
classical SQL-queries interface
fetch

select event, ts  from system

| EVENT | TS |
|---|---|
| 7035 | 2005-11-22 02:07:47.0 |
| 7036 | 2005-11-22 02:07:47.0 |
| 6006 | 2005-11-22 02:08:01.0 |
| 6005 | 2005-11-22 02:08:28.0 |
| 6009 | 2005-11-22 02:08:28.0 |
| 30 | 2005-11-22 02:08:30.0 |
| 4201 | 2005-11-22 02:08:32.0 |
| 7035 | 2005-11-22 02:10:19.0 |
| 7036 | 2005-11-22 02:10:19.0 |
| 7035 | 2005-11-22 02:10:43.0 |
| 7036 | 2005-11-22 02:10:44.0 |

**Figure 3: Data selection in EventSummarizer using standard SQL queries**



☐ DP-DP
[✓ Greedy-Greedy]
☐ DP-Greedy
☐ Greedy-DP

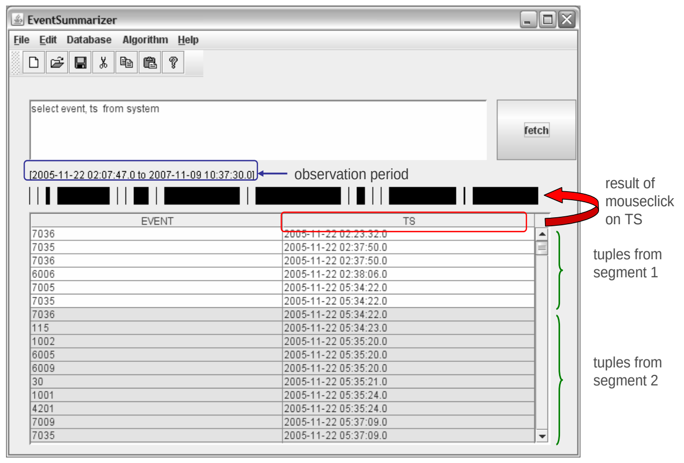
select event, ts  fro

| EVENT | TS |
|---|---|
| 7035 | 2005-11-22 02:07:47.0 |
| 7036 | 2005-11-22 02:07:47.0 |

**Figure 4: EventSummarizer's menu bar; algorithm's selection**

Once the data and the algorithm are selected the actual summarization task can be performed. Figure 5(a) shows the segmentation of the input timeline that is produced as a result of summarization on (summary) attribute `TS`. The specification of the summary attribute is done by a simple mouseclick on the attribute's name as it appears on the rendered data. In our example, a mouseclick on the `TS` attribute results in the segmentation bar shown in Figure 5(a). The bar shows the partition of the input timeline. The actual segments correspond the black-colored segments. Here, there are five relatively large intervals and thirteen smaller ones. The original input data is still shown in the lowest part of the screen. However, the tuples are ordered on `TS` and then colored according to the segment they belong to.

Figure 5(b) shows the actual grouping of the tuples within a single (in this case the last) segment. The segment is selected by a simple mouseclick on it. The grouping of the event types within this interval is rendered below the segmentation bar in a table form. The first attribute of this new table is the `Key` attribute and it corresponds to the event type. The second attribute is the `Count` attribute and it shows for every event type the number of its occurrences within

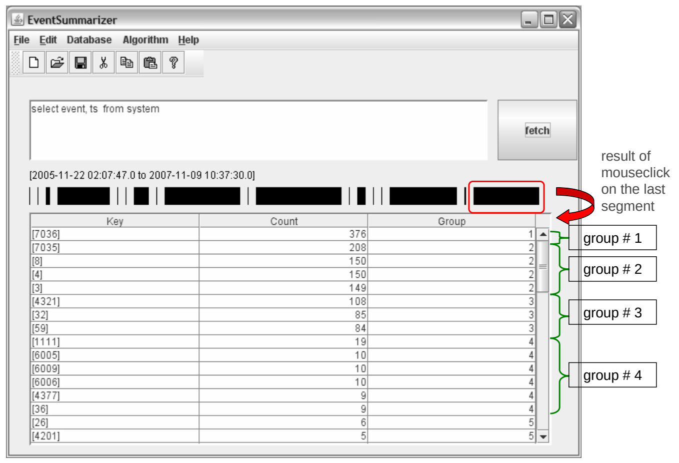


select event, ts from system

[2005-11-22 02:07:47.0 to 2007-11-09 10:37:30.0]

| EVENT | TS |
|---|---|
| 7036 | 2005-11-22 02:23:32.0 |
| 7035 | 2005-11-22 02:37:50.0 |
| 7036 | 2005-11-22 02:37:50.0 |
| 6006 | 2005-11-22 02:38:06.0 |
| 7005 | 2005-11-22 05:34:22.0 |
| 7035 | 2005-11-22 05:34:22.0 |
| 7036 | 2005-11-22 05:34:22.0 |
| 115 | 2005-11-22 05:34:23.0 |
| 1002 | 2005-11-22 05:35:20.0 |
| 6005 | 2005-11-22 05:35:20.0 |
| 6009 | 2005-11-22 05:35:20.0 |
| 30 | 2005-11-22 05:35:21.0 |
| 1001 | 2005-11-22 05:35:24.0 |
| 4201 | 2005-11-22 05:35:24.0 |
| 7009 | 2005-11-22 05:37:09.0 |
| 7035 | 2005-11-22 05:37:09.0 |

(a) EventSummarizer's visualization of the timeline's segmentation


EventSummarizer
File Edit Database Algorithm Help
fetch
result of mouseclick on the last segment
group # 1
group # 2
group # 3
group # 4

select event, ts from system

[2005-11-22 02:07:47.0 to 2007-11-09 10:37:30.0]

| Key | Count | Group |
|---|---|---|
| [7036] | 376 | 1 |
| [7035] | 208 | 2 |
| [8] | 150 | 2 |
| [4] | 150 | 2 |
| [3] | 149 | 2 |
| [4321] | 108 | 3 |
| [32] | 85 | 3 |
| [59] | 84 | 3 |
| [1111] | 19 | 4 |
| [6005] | 10 | 4 |
| [6009] | 10 | 4 |
| [6006] | 10 | 4 |
| [4377] | 9 | 4 |
| [36] | 9 | 4 |
| [26] | 6 | 5 |
| [4201] | 5 | 5 |

(b) Grouping of event types within a segment

**Figure 5: Demonstration of the functionality of EventSummarizer. Figure 5(a) shows the visualization of the segmentation output by EventSummarizer and Figure 5(b) shows the grouping of event types within a selected segment.**

the examined time interval. The last attribute, `Group`, takes integer values that denote the group-id in which the different event types belong to. Local associations are identified by event types sharing the same group-Id. In the illustrated example, group # 1 contains just a single event type, group # 2 contains four event types and so on. Notice that event types that belong in the same group have similar occurrence counts within the interval.

## 4. CONCLUSIONS

In this demo paper, we presented EventSummarizer, a tool for summarizing large event sequences. The algorithmic principles used in the implementation of the tool are described in detail in [3]. Here we focused on presenting an overview of the EventSummarizer architecture and demonstrating its functionality from the user's point of view. For example, a system administrator can use EventSummarizer as a standalone application to periodically review system activities. In the paper, we illustrated the usefulness of EventSummarizer by focusing on a selected event sequence example and demonstrated its main functionalities using this dataset. In the demo itself, we will show that EventSummarizer is easy to use and gives easy-to-interpret results. We will further showcase its usefulness using a diverse set of event sequences that arise in practical settings.

## 5. REFERENCES

[1] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. of the 11th Int'l Conference on Data Engineering*, Taipei, Taiwan, March 1995.

[2] D. Chudova and P. Smyth. Pattern discovery in sequences under a markov assumption. In *KDD*, pages 153–162, 2002.

[3] J. Kiernan and E. Terzi. Constructing comprehensive summaries of large event sequences. In *KDD*, pages 417–425, 2008.

[4] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 146–151, 1996.

[5] J. Pei, J. Han, and W. Wang. Constraint-based sequential pattern mining: the pattern-growth methods. *J. Intell. Inf. Syst.*, 28(2):133–160, 2007.

[6] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[7] J. Yang, W. Wang, P. S. Yu, and J. Han. Mining long sequential patterns in a noisy environment. In *ACM SIGMOD International Conference on Management of Data*, pages 406–417, 2002.