Constructing comprehensive summaries of large event sequences

JERRY KIERNAN
IBM Silicon Valley Lab

and
EVIMARIA TERZI
IBM Almaden Research Center

Event sequences capture system and user activity over time. Prior research on sequence mining has mostly focused on discovering local patterns appearing in a sequence. While interesting, these patterns do not give a comprehensive summary of the entire event sequence. Moreover, the number of patterns discovered can be large. In this paper, we take an alternative approach and build *short* summaries that describe an entire sequence, and discover local dependencies between event types.

We formally define the summarization problem as an optimization problem that balances shortness of the summary with accuracy of the data description. We show that this problem can be solved optimally in polynomial time by using a combination of two dynamic-programming algorithms. We also explore more efficient greedy alternatives and demonstrate that they work well on large datasets. Experiments on both synthetic and real datasets illustrate that our algorithms are efficient and produce high-quality results, and reveal interesting local structures in the data.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*; I.5.3 [**Pattern Recognition**]: Clustering—*Algorithms*; E.4 [**Coding and Information Theory**]: Data Compaction and compression

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: event sequences, summarization, log mining

1. INTRODUCTION

Monitoring of systems' and users' activities produces large *event sequences*, i.e., logs where each event has an associated occurrence time as well as other attributes. Network traffic and activity logs are examples of large event sequences. Off-the-shelf data-mining methods for event sequences though successful in finding recurring local structures, e.g., episodes, can prove inadequate in providing a global model of the data. Moreover, data-mining algorithms usually output too many patterns that may overwhelm the data analysts. In this paper, we reveal a new aspect of event sequence analysis, namely how to concisely summarize such event sequences.

From the point of view of a data analyst, an event-sequence summarization system should have the following properties.

Brevity and accuracy:	The summarization system should construct *short* summaries that *accurately* describe the input data.
Global data description:	The summaries should give an indication of the global structure of the event sequence and its evolution through

time.

Local pattern identification: The summary should reveal information about local patterns; normal or suspicious events or combinations of events that occur at certain points in time should be identified by just looking at the summary.

Parameter free: No extra tuning should be required by the analyst in order for the summarization method to give informative and useful results.

Despite the bulk of work on the analysis of event-sequences, to the best of our knowledge, there is no technique that satisfies all requirements discussed above. In this paper, we present a summarization technique that exhibits all these characteristics. More specifically,

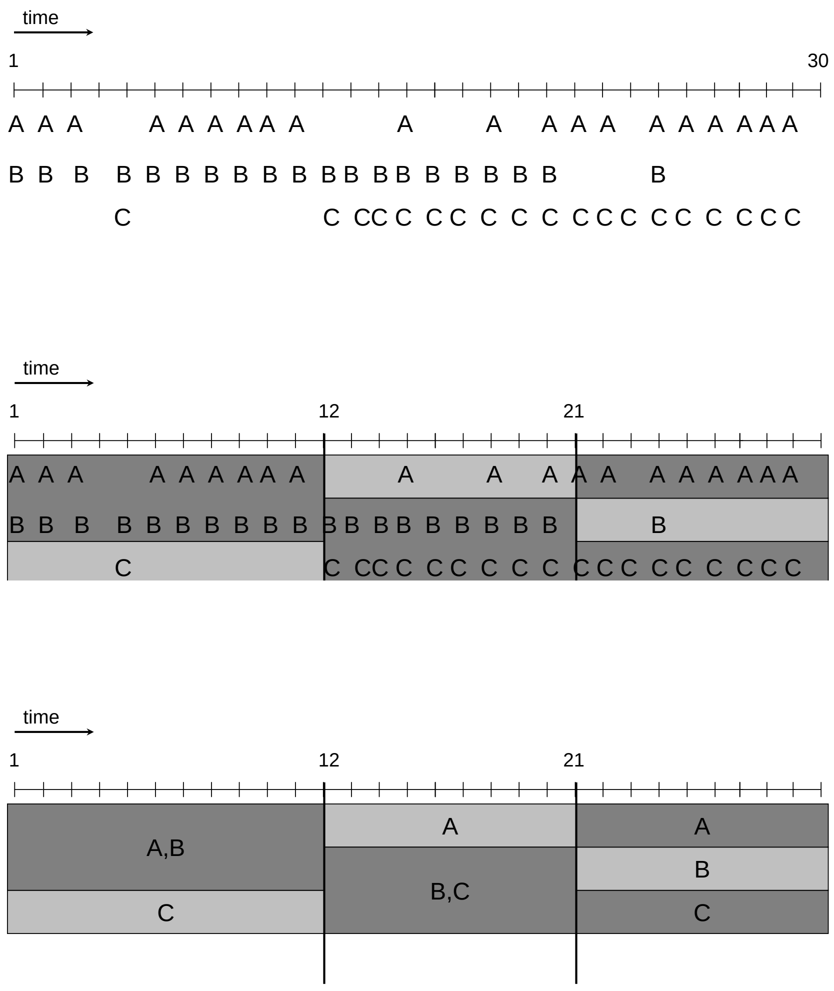
—We use the *Minimum Description Length* (MDL) principle to find a balance between summaries' length and descriptions' accuracy.
—We adopt a *segmentation model* that provides a high-level view of the sequence by identifying global intervals on the timeline. The events appearing within each interval exhibit local regularity.
—Each interval in the segmented timeline is described by a *local model* similar to clustering. Our local model groups event types with similar rates of appearance within the interval; in this way local associations among event types are captured.
—The usage of MDL penalizes both complex models that over-fit the data and simple models that over-generalize. This makes our methodology parameter-free and thus increases its practical utility.

Our methodology makes two assumptions: (a) the appearances of events of different types are independent and (b) the appearances of events of the same time at different timestamps are also independent.

Example 1. *Figure 1 shows an example of an input event sequence and the output of our method for this particular sequence. The input sequence is shown in Figure 1(a). The sequence contains three event types* $\{A, B, C\}$ *and it spans timeline* $[1, 30]$ *that consists of* 30 *discrete timestamps. A possible instantiation of these event types one can think that events of type A correspond to logins of user "Alice", events of type B to logins of user "Bob", and events of type C to logins of user "Cynthia" in the system.*

Figure 1(b) shows the actual segmental grouping that our method finds. Three segments are identified: $[1, 11]$, $[12, 20]$ *and* $[21, 30]$. *Within each segment the events are grouped into two groups; event types with similar frequency of appearance within 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* $[1, 11]$. *Similarly, the groups in the second segment are* $\{A\}$ *and* $\{B, C\}$ *and in the third segment* $\{A, C\}$ *and* $\{B\}$.

Finally, Figure 1(c) shows what the output of the summarization method conceptually looks like. 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.



(c) Output summary

Fig. 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. The same tone of gray identifies group membership.

1.1 Problem Statement and Approach

We address the following problem: assume an event sequence $\mathbf{S}$ that records occurrences of events over a time interval $[1, n]$. Additionally, let $\mathcal{E}$ denote the distinct event types that appear in the sequence. Our goal is to partition the observation interval into segments of local activity that span $[1, n]$; within each segment identify groups of event types that exhibit similar frequency of occurrence in the segment. We use the term *segmental grouping* for this data-description model. For the purposes of this paper we only consider discrete timelines. We additionally assume that events of different types are generated at every distinct timestamp independently from some stationary probability that depends on the event type and the segment itself.

We formally define the problem of finding the best segmental grouping as an optimization problem. By penalizing both complex and simple models, we develop a parameter-free methodology and provide polynomial-time algorithms that optimally solve the above summarization problem. Dynamic-programming is at the core of these optimal algorithms. The computational complexity of our algorithms depends only on the number of timestamps at which events occur and not on the total length of the timeline n.

Although the main motivation for our work is the forensic analysis of large audit logs, the techniques presented herein can also be applied to other diverse domains; appearances of words within a stream of documents could be considered as event sequences and useful summaries can be constructed for these domains using our methodology.

1.2 Roadmap

The rest of the paper is organized as follows. We review the related work in Section 2. In Section 3 we give some basic notational conventions. In Section 4 we formally describe our summarization scheme and the corresponding optimization problem of finding the best summary. The algorithms for solving the problem are presented in Section 5. In Section 6 we study some variants of the original problem. Experiments are given in Section 7; Section 8 presents an application which exploits the algorithms presented in the paper and illustrates intuitive visual metaphors for rendering the algorithms' results. We conclude in Section 9.

2. RELATED WORK

Although we are not aware of any work that proposes the same summarization model for event sequences, our work clearly overlaps with work on sequence mining and time-series analysis.

Closely related to ours is the work on mining episodes and sequential patterns ([Agrawal and Srikant 1995; Bettini et al. 1998; Chudova and Smyth 2002; Mannila and Toivonen 1996; Mannila et al. 1997; Pei et al. 2007; Srikant and Agrawal 1996; Yang et al. 2002]). That work mostly focused on developing algorithms that identify configurations of discrete events clustered in time. Although these algorithms identify local event patterns they do not focus on providing a global description of the data sequence. Moreover, these methods usually output all local patterns that satisfy certain properties. In contrast, the focus of our work is to provide an overall description of the event sequence and identify local associations of events, keeping the whole description short and accurate.

Summarization of event sequences via a segmentation model is proposed in [Mannila and Salmenkivi 2001]. However, the technique presented there can only model sequences of single event types; within each local interval, the appearances of events are modelled by a constant intensity model. The model of [Mannila and Salmenkivi 2001] cannot be extended to handle more than one event types. In fact, one can think of our model as a generalization of the model proposed in [Mannila and Salmenkivi 2001] since in fact we split the event types into groups of constant intensities.

Also related is the segmentation framework developed by [Koivisto et al. 2003] in order to identify block structures in genetic sequences. A minimum description length approach is also used there for identifying the number and positions of segment boundaries. However, the models built within each block serve the particular modelling requirements of the genetic sequences under study. For example, in the case of [Koivisto et al. 2003] finding the local model in each segment is an NP-hard task, while in our case this task is polynomial. At a high-level, our work is related to the general problem of finding homogeneous DNA sequences. This problem has been extensively studied in bioinformatics leading to a variety of segmentation algorithms [Gionis and Mannila 2003; Li 2001a; 2001b; Ruzzo and Tompa 1999]. There is only high-level connection between these pieces of work and ours: although we both deal with segmentation problems and in many cases we use dynamic programming as our main algorithmic tool, each paper studies a different model and tries to optimize a different optimization function.

Periodicity detection in event sequences has been the focus of many sequential data-analysis techniques (e.g., [Elfeky et al. 2004; Han et al. 1998; Han et al. 1999; Ma and Hellerstein 2001]). Although periodicity detection in event sequences is an interesting topic by itself, it is not the focus of our paper. We focus on finding local associations across different event types rather than finding combinations of event types that exhibit some periodic behavior.

Identifying time intervals at which an event or a combination of events makes bursty appearances has been the focus of many papers associated with mining and analysis of document streams, e.g., [Allan et al. 2001; Brants and Chen 2003; Kleinberg 2003; Swan and Allan 2000; Yang et al. 2000]. In that setting events correspond to specific words appearing on a stream of documents. Our methodology can also be applied to the analysis of document streams. Since the class of models we are considering are different from those proposed before, one can think of our methodology as complementary to the existing methods for document analysis.

At a high level there is an obvious connection between our model and the standard segmentation model used for time-series segmentation (see [Guha et al. 2001; Karras et al. 2007; Keogh et al. 2001; Papadimitriou and Yu 2006; Terzi and Tsaparas 2006] for an indicative, though not complete, set of references). Those models partition the time series into contiguous non-overlapping segment, so that within each segment the data exhibits some kind of homogeneity. Different definitions of homogeneity lead to different optimization problems and different models for describing the data within each segment. Usually linear models or piecewise-constant approximations are used for that. Despite the high-level similarity between this work and ours we point out that our focus is on event sequences rather than on time series. Moreover, the local model we consider to represent the data within a segment is a special type of clustering model. To the best of our knowledge we are not aware of any prior work that considers such local models.

There is an equally interesting line of work that deals with the discovery of local patterns in time-series data, e.g., [Papadimitriou and Yu 2006; Sakurai et al. 2005; Zhu and Shasha 2002]. However, the connection to our work remains at a high level since we focus on event sequences and not on time series, and the local per segment models we consider are quite distinct from the models considered before. Same high-level connection exists between our model and HMMs [Rabiner and Juang 1986]. However, the assumptions behind HMMs are different. For example, we assume that every segment (state) in our model is independent of any other state. To the contrary the HMMs assume that the Markov property holds between the states.

3. PRELIMINARIES

Event sequences consist of *events* that occur at specific points in time. That is, every event in the sequence has an associated time of occurrence. We assume a set $\mathcal{E}$ of m different *event types*. An event is a pair (E, t), where $E \in \mathcal{E}$ is an event type and t is the *(occurrence) time* of the event on a timeline. We consider *discrete* timelines in which occurrence times of events are positive integers in the interval $[1, n]$. That is, the timeline consists of n different evenly spaced timestamps at which events of different types might occur.

We represent an event sequence by an $m \times n$ array $\mathbf{S}$ such that $\mathbf{S}(i,t) = 1$ if an event of type E_i has occurred at time point t. At a certain time t, events of different types can occur simultaneously. That is, each column of $\mathbf{S}$ can have more than one 1-entries. However, at any time t, only one event of each type can occur (If multiple events of the same type do occur at a point t, they can be ignored as duplicates).

Figure 1(a) shows an event sequence $\mathbf{S}$ on which events of $m = 3$ different types appear; $\mathcal{E} = \{A, B, C\}$. The events occur on the timeline $[1, 30]$. That is, there are 30 timestamps at which any of the three event types can occur.

Given interval $I \subseteq [1, n]$, we use $\mathbf{S}[I]$ to denote the $m \times |I|$ projection of $\mathbf{S}$ on the interval I. Finally, for event type $E \in \mathcal{E}$ and interval $I \subseteq [1, n]$ we denote the number of occurrences of events of type E within the interval I with $n(E, I)$.

The core idea is to find a *segmentation* of the input timeline $[1, n]$ into contiguous, non-overlapping intervals that cover $[1, n]$. We call these intervals *segments*. More formally, we want to find a segmentation of $\mathbf{S}$ into segments denoted by $\mathbf{S} = (\mathbf{S}_1, \ldots, \mathbf{S}_k)$. Such a segmentation is defined by $k + 1$ *boundaries* $\{b_1, b_2, \ldots, b_k, b_{k+1}\}$ where $b_1 = 1$, $b_{k+1} = n + 1$ and each b_j, with $2 \leq j \leq k$ takes integer values in $[2, n]$. Therefore, the j-th segment corresponds to the subsequence $\mathbf{S}[b_j, b_{j+1} - 1]$. A segmentation of the input event sequence of Figure 1(a) is shown in Figure 1(b). The input sequence is split into 3 segments defined by the boundaries $\{1, 12, 21, 31\}$.

We now focus our attention on the data of a specific segment $\mathbf{S}_i$ defined over the time interval I. That is, $\mathbf{S}_i = \mathbf{S}[I]$. We describe the portion of the data that corresponds to $\mathbf{S}_i$ by the local model M_i. We consider model M_i to be a partitioning of event types $\mathcal{E}$ into groups $\{X_{i1}, \ldots, X_{i\ell}\}$ such that $X_{ij} \subseteq \mathcal{E}$ and $X_{ij} \cap X_{ij'} = \emptyset$ for every $j \neq j'$ with $1 \leq j, j' \leq \ell$. Each group X_{ij} is described by a single parameter $p(X_{ij})$ that corresponds to the probability of seeing an event of any type in X_{ij} at any timestamp within data segment S_i.

Consider for example the first segment of the output segmentation in Figure 1(b) (or Figure 1(c)) that defines segment $I_1 = [1, 11]$, with length $|I_1| = 11$. In this case the local model M_1 that describes the data in $\mathbf{S}_1 = \mathbf{S}[I_1]$ partitions $\mathcal{E}$ into groups $X_{11} = \{A, B\}$ and $X_{12} = \{C\}$ with

$$p(X_{11}) = \frac{1}{2}\frac{n(A, I_1) + n(B, I_1)}{|I_1|} = \frac{19}{22},$$

and

$$p(X_{12}) = \frac{n(C, I_1)}{|I_1|} = \frac{1}{11}.$$

The SUMMARIZATION **Problem.** Our overall goal is to identify the set of boundaries on the timeline that partition $\mathbf{S}$ into segments $(\mathbf{S}_1, \ldots, \mathbf{S}_k)$ and within each segment $\mathbf{S}_i$ identify a local model M_i that best describes the data in $\mathbf{S}_i$.

The partitioning of $\mathbf{S}$ into segments $(\mathbf{S}_1, \ldots, \mathbf{S}_k)$ and the corresponding local models $M_1, \ldots, M_k$ constitute the *segmental grouping* or *summary* of $\mathbf{S}$. For the rest of the discussion we use the terms summary and segmental grouping interchangeably.

In order to be able to devise algorithms for the SUMMARIZATION problem we first need to define the optimization function that best describes the objective of this informal problem definition. Our optimization function is motivated by the *Minimum Description Length* (MDL) principle.

4. SEGMENTATION MODEL FOR EVENT SEQUENCES

Before formally developing our model, we first review the *Minimum Description Length* (MDL) principle. Then, we show how to apply this principle to formalize the SUMMARIZATION problem.

4.1 Minimum Description Length Principle

The MDL principle [Rissanen 1978; 1989] allows us to transform the requirement of balance between over-generalizing and over-fitting into a computational requirement.

In brief the MDL principle states the following: assume two parties P and P' that want to communicate with each other. More specifically, assume that P wants to send event sequence $\mathbf{S}$ to P' using as less bits as possible. In order for P to achieve this minimization of communication cost, she has to select model M from a class of models $\mathcal{M}$, and use M to describe her data. Then, she can send to P' model M plus the additional information required to describe the data given the transmitted model.

Thus, party P has to encode the model M and then encode the data given this model. The quality of the selected model is evaluated based on the number of bits required for this overall encoding of the model and the data given the model.

MDL discourages complex models with minimal data cost and simple models with large data costs. It tries to find a balance between these two extremes. It is obvious that the MDL principle is a generic principle and it can have multiple instantiations that are determined by a set of modeling assumptions. It has been previously successfully applied in a variety of settings that range from decision-tree classifiers [Mehta et al. 1995], genetic-sequence modeling [Koivisto et al. 2003], patterns in sets of strings [Kilpeläinen et al. 1995] and many more. We devote the rest of the section to describe our instantiation of the MDL principle.

4.2 The Encoding Scheme

Recall that we model event sequences using a segmentation model that partitions the input observation interval $[1, n]$ into contiguous, non-overlapping intervals $I_1, \ldots, I_k$. Therefore, $\mathbf{S}$ is split into $(\mathbf{S}_1, \ldots, \mathbf{S}_k)$, where $\mathbf{S}_i = \mathbf{S}[I_i]$. The data in each $\mathbf{S}_i$ are described by *local model* M_i; the local model is in fact a grouping of the event types based on their frequency of appearance in $\mathbf{S}_i$.

Local encoding scheme: We start by describing the procedure that estimates the number of bits required to encode the data within a single segment $\mathbf{S}_i$.

Let model M_i partition the rows of $\mathbf{S}_i$ (which correspond to events of all types, present or not in $\mathbf{S}_i$) into ℓ groups $X_1, \ldots, X_\ell$. Each group X_j is described by a single parameter $p(X_j)$, the probability of appearance of any event type in X_j within subsequence $\mathbf{S}_i$. Given the X_j's, and corresponding $p(X_j)$'s for $1 \leq j \leq \ell$, and assuming independence of occurrences of events and event types, the *probability* of data $\mathbf{S}_i$ given model M_i is given by

$$\Pr(\mathbf{S}_i|M_i) = \prod_{j=1}^{\ell} \prod_{E \in X_j} p(X_j)^{n(E,I)} (1 - p(X_j))^{|I| - n(E,I)}.$$

Recall that quantity $n(E, I)$ refers to the number of events of type E that appear in interval I. The number of bits required to describe data $\mathbf{S}_i$ given model M_i is $-\log(\Pr(\mathbf{S}_i \mid M_i))$. This is because the number of bits required to encode an event that appears with probability q is $-\log(q)$. The more general model M_i is the less specific to the data S_i. In this case the probability $\Pr(\mathbf{S}_i \mid M_i)$ is small and many bits are required to further refine the description of $\mathbf{S}_i$ given M_i.

Therefore, *local data cost* of $\mathbf{S}_i$ given M_i is

$$\begin{aligned} \textsc{Ld}(\mathbf{S}_i|M_i) &= -\log \Pr(\mathbf{S}_i|M_i) \qquad (1) \\ &= -\sum_{j=1}^{\ell} \sum_{E \in X_j} \Big(n(E,I) \log p(X_j) + \\ &\quad + (|I| - n(E,I)) \log(1 - p(X_j)) \Big). \end{aligned}$$

Equation (1) gives the number of bits required to describe data in $\mathbf{S}_i$ given model M_i. For the encoding of $\mathbf{S}_i$ we also need to calculate the number of bits required to encode the model M_i itself. We call this cost (in bits) the *local model cost* $\textsc{Lm}(M_i)$. In order to encode M_i we need to describe the event types associated with every group X_j $(1 \leq j \leq \ell)$, and for each group X_j we need to specify parameter $p(X_j)$. Since $p(X_j)$ is a fraction with numerator an integer in $\{1, n\}$ and denominator the length of the segment. Therefore, we can describe each one of the $p(X_j)$'s using $\log n$ bits. This is because the denominator has been communicated in the description of the global model and the numerator is simply an integer that takes value at most n. Since there are ℓ groups we need a total of $\ell \log n$ bits to encode the ℓ different $p(X_j)$'s. The encoding of the partitioning is slightly more tricky; first we observe that if we fix an ordering of the event types that is consistent with the partitioning $X_1, \ldots, X_\ell$,[1] then we need $m \log m$ bits to specify the ordering and $\ell \log m$ bits to identify the ℓ partition points on that fixed order. This is because for this fixed order the partition points are integers in the range $[1, m]$ and thus $\log m$ bits are necessary for the description of each partition point. Summing up these costs we get the local model cost for M_i that is

$$\textsc{Lm}\left(M_i\right) \;=\; \ell \log n + \ell \log m + m \log m. \tag{2}$$

Therefore, the total local cost in bits for describing segment $\mathbf{S}_i$ is the number of bits required to describe $\mathbf{S}_i$ given model M_i and the cost of describing model M_i itself. By summing Equations (1) and (2) we get the valuation of the *local cost* Ll, which is

$$\textsc{Ll}\left(\mathbf{S}_i, M_i\right) \;=\; \textsc{Ld}\left(\mathbf{S}_i | M_i\right) + \textsc{Lm}\left(M_i\right). \tag{3}$$

Generative model: The above encoding schemes assume the following data-generation process; within segment $\mathbf{S}_i$ events of different types are generated independently. For each event type $E \in X_j$, with $1 \leq j \leq \ell$, an event of type E is generated at every time point $t \in I$ independently with probability $p\left(X_j\right)$.
Global Encoding Scheme: The global model is the segmental model M that splits $\mathbf{S}$ into segments $\mathbf{S}_1, \ldots, \mathbf{S}_k$; each segment is specified by its boundaries and the corresponding local model M_i. If for every segment i, the data in $\mathbf{S}_i$ is described using the encoding scheme described above, the only additional information that needs to be encoded for describing the global model is the positions of the segment boundaries that define the starting points of the segments on timeline $[1, n]$. Since there are n possible boundary positions the encoding of k segment boundaries would require $k \log n$ bits. Therefore, the total length of the description in bits would be

$$\textsc{Tl}\left(\mathbf{S}, M\right) \;=\; k \log n + \sum_{i=1}^{k} \textsc{Ll}\left(\mathbf{S}_i, M_i\right),$$

where $\textsc{Ll}\left(\mathbf{S}_i, M_i\right)$ is evaluated as in Equation (3).

4.3 Problem Definition Revisited

We are now ready to give the formal definition of the Summarization problem.

Problem 1. *(*Summarization*) Given event sequence* $\mathbf{S}$ *over observation period* $[1, n]$ *in which event types from set* $\mathcal{E}$ *occur, find integer* k *and a segmental grouping* M *of* $\mathbf{S}$ *into* $(\mathbf{S}_1, \ldots, \mathbf{S}_k)$ *and identify the best local model* M_i *for each* $\mathbf{S}_i$ *such that the total description length*

$$\textsc{Tl}\left(\mathbf{S}, M\right) \;=\; k \log n + \sum_{i=1}^{k} \textsc{Ll}\left(\mathbf{S}_i, M_i\right), \tag{4}$$

is minimized.

Problem 1 gives the optimization function that consists of the number of bits required to encode the data given the model, and the model itself. Note that the

[1] A trivial such ordering is the one that places first all the event types in X_1, followed by the event types in X_2 and so on.

total model cost can be decomposed in the cost for encoding the global segmentation model $k \log n$ plus the cost for encoding the different local models evaluated as in Equation (2). The cost of encoding the data given the model is simply the summation of the local data costs for every segment.

We use $\textsc{Ll}^*(\mathbf{S}_i)$ to denote the minimum value of $\textsc{Ll}(\mathbf{S}_i, M_i)$, over all possible local models M_i. Similarly, we use $\textsc{Tl}^*(\mathbf{S})$ to denote the minimum value of $\textsc{Tl}(\mathbf{S}, M)$ over all possible summaries M.

Since the definition of the optimization function is formed based on the MDL principle, the function is such that: (a) complex summaries are penalized because they over-fit the data and (b) simple summaries are also penalized since they over-generalize and fail to describe the data with the desired accuracy. Moreover, using the MDL principle allows for a problem formulation that is parameter-free; no parameter setting is required from the analyst who is attempting to extract knowledge from the input event sequence $\mathbf{S}$.

5. ALGORITHMS

Despite the apparent interplay between the local models picked and the positions of the segment boundaries on the timeline, we can show that, in fact, Problem 1 can be solved optimally in polynomial time.

Given data segment $\mathbf{S}_i$ we call the problem of identifying the local model that minimizes $\textsc{Ll}(\mathbf{S}_i, M_i)$ the LocalGrouping problem, and we formally define it as follows.

Problem 2. *(*LocalGrouping*) Given sequence* $\mathbf{S}$ *and interval* $I \subseteq [1, n]$ *find the optimal local model* M_i *that minimizes the local description length of* $\mathbf{S}_i = \mathbf{S}[I]$ *given* M_i*. That is, find* M_i *such that*

$$\begin{aligned} M_i &= \arg\min_{M_i'} \textsc{Ll}(\mathbf{S}_i, M_i') \\ &= \arg\min_{M_i'} \big(\textsc{Ld}(\mathbf{S}_i | M_i') + \textsc{Lm}(M_i')\big). \end{aligned}$$

In the rest of this section we give optimal polynomial-time algorithms for the Summarization problem. We also provide alternative sub-optimal, but practical and efficient, algorithms for the Summarization problem.

5.1 Finding the Optimal Global Model

We first present an optimal dynamic-programming algorithm for the Summarization problem. We also show that not all possible segmentations of interval $[1, n]$ are candidate solutions to the Summarization problem.

Theorem 1. *For any interval* $I \subseteq [1, n]$*, let* $\textsc{Ll}^*(\mathbf{S}[I]) = \min_{M_i} \textsc{Ll}(\mathbf{S}[I], M_i)$*. Then, Problem 1 can be solved optimally by evaluating the following dynamic-programming recursion. For every* $1 \leq i \leq n$*,*

$$\begin{aligned} &\textsc{Tl}^*(\mathbf{S}[1, i]) = \\ &= \min_{1 \leq j \leq i} \left\{ \textsc{Tl}^*(\mathbf{S}[1, j]) + \textsc{Ll}^*(\mathbf{S}[j+1, i]) \right\}. \end{aligned} \tag{5}$$

Proof. Recursion (5) is a standard dynamic-programming recursion that for every i ($1 \leq i \leq n$) goes through all j's ($1 \leq j \leq i$) and evaluates the quantity $\textsc{Tl}^*(\mathbf{S}[1,j]) + \textsc{Ll}^*(\mathbf{S}[j+1,i])$. Note that $\textsc{Tl}^*(\mathbf{S}[1,j])$ is simply a lookup of an already computed value. Therefore, the only additional computation that needs to be done is the evaluation of the second part of the summation, i.e., $\textsc{Ll}^*(\mathbf{S}[j+1,i])$. If that last part can be computed optimally, then the evaluation of Recursion (5) is also optimal. The time required for computing $\textsc{Tl}^*(\mathbf{S}[1,n])$ depends on the time required to evaluate function $\textsc{Ll}^*$ on every interval. □

We call the dynamic-programming algorithm that implements Recursion (5) the `Segment-DP` algorithm. If T_L is the time required to evaluate $\textsc{Ll}^*(\mathbf{S}[I])$, then the running time of the `Segment-DP` algorithm is $O(n^2 T_L)$.

Not all points on the interval $[1,n]$ are qualified to be segment boundaries in the optimal segmentation. In fact, only the timestamps on which an event (of any type) occurs are candidate segment boundaries. The following proposition summarizes this fact.

Proposition 1. *Consider event sequence* $\mathbf{S}$ *that spans interval* $[1,n]$ *and let* $T \subseteq \{1,2,\ldots,n\}$ *be the set of timestamps at which events have actually occurred. Then, the segment boundaries of the optimal segmentation model are subset of* T.

Proof. Let $(\mathbf{S}_1,\ldots,\mathbf{S}_i,\mathbf{S}_{i+1}\ldots,\mathbf{S}_k)$ be the optimal segmentation of event sequence $\mathbf{S}$ into k segments. Let segments $\mathbf{S}_i$ and $\mathbf{S}_{i+1}$ be such that they are defined by the following consecutive intervals $[P,x]$ and $[x,N]$ in $[1,n]$. Now let $P,N \in T$ and $x \notin T$. Our proof will be by contradiction to the optimality assumption of the segmentation. That is, we will show that moving boundary x to a new position x' will reduce the cost of the output segmentation. Let $P(x)$ and $N(x)$ be the first point to the left and right of x that is in T. That is, $x \in [P(x), N(x)]$.

The total number of bits required for encoding the data using segmentation $(\mathbf{S}_1,\ldots,\mathbf{S}_i,\mathbf{S}_{i+1}\ldots,\mathbf{S}_k)$ is

$$k\log n + \sum_{i=1}^{k} \textsc{Ll}\left(\mathbf{S}_i, M_i\right).$$

If we use C_S to represent the $\textsc{Ll}$ of intervals that do not have x as their end point, the above function can be written as a function of x as follows:

$$\begin{aligned} F(x) &= C_S + \textsc{Ll}\left(\mathbf{S}_i, M_i\right) + \textsc{Ll}\left(\mathbf{S}_{i+1}, M_{i+1}\right) \\ &= C_S - \sum_{j=1}^{\ell_i} \sum_{E \in X_{ij}} n\left(E, [P(x), x]\right) \log p\left(X_{ij}\right) \\ &\quad - \sum_{j=1}^{\ell_i} \sum_{E \in X_{ij}} \left(x - P(x) - n\left(E, [P(x), x]\right)\right) \log\left(1 - p\left(X_{ij}\right)\right) \\ &\quad - \sum_{j=1}^{\ell_{i+1}} \sum_{E \in X_{(i+1)j}} n\left(E, [x, N(x)]\right) \log p\left(X_{(i+1)j}\right) \\ &\quad - \sum_{j=1}^{\ell_{i+1}} \sum_{E \in X_{(i+1)j}} \left(N(x) - x - n\left(E, [x, N(x)]\right)\right) \log\left(1 - p\left(X_{(i+1)j}\right)\right). \end{aligned}$$

In the above equation X_{ij} and $X_{(i+1)j}$ refer to the j-th group of the i-th and $(i+1)$-th segment respectively. For every event type $E \in \mathcal{E}$, the above function is concave with respect to x, and therefore, function $F(x)$ is also concave with respect to x (as a summation of concave functions). Therefore, the value of x for which $F(x)$ takes its minimum value is either $P(x)$ or $N(x)$. That is, the original segmentation was not optimal, and its cost can be improved if boundary x is moved to coincide with a point in T.

Note that the statement about the concavity of $F(x)$ takes also into account the fact that $p\left(X_{ij}\right)$ and $p\left(X_{(i+1)j}\right)$ are also functions of x. However, for the sake of clarity of presentation we avoid presenting the full expression of $F(x)$. The proof of concavity of $F(x)$ is also trivial by a simple derivation. □

Proposition 1 offers a speedup of the `Segment-DP` algorithm from $O\left(n^2 T_L\right)$ to $O\left(|T|^2 T_L\right)$, where $|T| \leq n$. That is, the evaluation of Recursion (5) does not have to go through all the points $\{1, \ldots, n\}$, but rather all the points in T, on which events actually occur. Although in terms of asymptotic running time Proposition 1 does not give any speedup, in practice, there are many real data for which $|T| << n$ and therefore Proposition 1 becomes extremely useful. In our experiments with real datasets we illustrate this fact.

5.2 The `Greedy` Algorithm

The `Greedy` algorithm is an alternative to `Segment-DP` and computes a summary M of $\mathbf{S}$ in a bottom-up fashion. The algorithm starts with summary M^1, where all data points are in their own segment. At the t-th step of the algorithm, we identify boundary b in M^t whose removal causes the maximum decrease in $\textsc{Tl}\left(\mathbf{S}, M^t\right)$. By removing boundary b we obtain summary M^{t+1}. If no boundary that causes cost reduction exists, the algorithm outputs summary M^t.

Since there are at most $n-1$ boundaries candidate for removal the algorithm can have at most $n-1$ iterations. In each iteration the boundary with the largest reduction in the total cost needs to be found. Using a heap data structure this can be done in $O(1)$ time.

The entries of the heap at iteration t are the boundaries of summary M^t. Let these boundaries be $\{b_1,\ldots,b_l\}$. Each entry b_j is associated with the *impact*, $G(b_j)$, of its removal from M^t. The impact of b_j is the change in $\textsc{Tl}(\mathbf{S},M^t)$ that is caused by the removal of b_j from M^t. The impact may be positive if $\textsc{Tl}(\mathbf{S},M^t)$ is increased or negative if the total description length is decreased. For every point b_j at iteration t the value of $G(b_j)$ is

$$\begin{aligned} G(b_j) \;=\; & \textsc{Ll}^*\left(\mathbf{S}\left[b_{j-1},b_{j+1}-1\right]\right)+\log n \\ & -\textsc{Ll}^*\left(\mathbf{S}\left[b_{j-1},b_{j}-1\right]\right)-\log n \\ & -\textsc{Ll}^*\left(\mathbf{S}\left[b_{j},b_{j+1}-1\right]\right)-\log n. \end{aligned}$$

The positive terms in the first row of the above equation correspond to the cost of describing data $\mathbf{S}\left[b_{j-1},b_{j+1}-1\right]$ after removing b_j and merging segments $[b_{j-1},b_j]$ and $[b_j,b_{j+1}-1]$ into a single segment. The negative costs correspond to the cost of describing the same portion of the data using the two segments $[b_{j-1},b_j]$ and $[b_j,b_{j+1}-1]$.

Upon the removal of boundary b_j at iteration t, the impacts of boundaries b_{j-1} and b_{j+1} need to be updated. With the right bookkeeping this requires the evaluation of $\textsc{Ll}^*$ for two different intervals per update, and thus $O(2T_L)$ time. In addition to that, one heap update per iteration is required and takes $O(\log n)$ time. Therefore, the total running time of the `Greedy` algorithm is $O(T_L n\log n)$. Proposition 1 can again speedup the running time of the `Greedy` algorithm to $O(T_l|T|\log|T|)$.

5.3 Finding Optimal Local Models

In this section we show that the $\textsc{LocalGrouping}$ can also be solved optimally in polynomial time using yet another dynamic-programming algorithm. We call this algorithm the `Local-DP` algorithm.

The next proposition states that finding the optimal parameters $p\left(X_j\right)$ that minimize $\textsc{Ld}$ for local model M_i that partitions $\mathcal{E}$ into $X_1,\ldots,X_\ell$ is simple; the value of $p\left(X_j\right)$ is the mean of the occurrence probabilities of each event type $E\in X_j$ within segment I.

Proposition 2. *Consider interval $I\subseteq[1,n]$, and local model M_i for data in $\mathbf{S}_i=\mathbf{S}[I]$. Let M_i partition $\mathcal{E}$ into groups $X_1,\ldots,X_\ell$. Then, for every X_j, with $1\le j\le\ell$, the value of $p\left(X_j\right)$ that minimizes $\textsc{Ld}\left(\mathbf{S}_i|M_i\right)$ is*

$$p\left(X_j\right)=\frac{1}{|X_j|}\sum_{E\in X_j}\frac{n\left(E,I\right)}{|I|}.$$

Proof. The contribution of every group X_j to function $\textsc{Ld}\left(\mathbf{S}_i|M_i\right)$ is

$$-\sum_{E\in X_j}\Big[n\left(E,I\right)\log p\left(X_j\right)+\left(|I|-n\left(E,I\right)\right)\log\left(1-p\left(X_j\right)\right)\Big].$$

By substituting $p\left(X_j\right)$ with P_j, the above is basically a function of P_j. That is,

$$F\left(P_j\right)=-\sum_{E\in X_j}\Big[n\left(E,I\right)\log P_j+\left(|I|-n\left(E,I\right)\right)\log\left(1-P_j\right)\Big].$$

The first derivative of $F\left(P_j\right)$ with respect to P_j is

$$F'\left(P_j\right) = -\sum_{E\in X_j}\big(n\left(E,I\right)\frac{1}{P_j} - \left(|I| - n\left(E,I\right)\right)\frac{1}{1-P_j}\big).$$

In order to find the value of $P_j = p\left(X_j\right)$ that minimizes $\textsc{Ld}\left(\mathbf{S}_i|M_i\right)$ we need to find the value of P_j for which $F'\left(P_j\right)$ becomes equal to 0. By solving $F'\left(P_j\right) = 0$ we get that this value is

$$P_j = \frac{1}{|X_j|}\sum_{E\in X_j}\frac{n\left(E,I\right)}{|I|}.$$

□

The above proposition states that the optimal representative of the frequency of every group is the average of the frequencies of the event types in the group. A corollary of this proposition is the fact that the optimal grouping assumes an ordering of the event types within a segment in decreasing (or increasing) order of their frequencies. A formal statement of this observation is given below.

Observation 1. *Consider interval I and let $\mathbf{S}_i = \mathbf{S}[I]$. Without loss of generality assume that the events in $\mathcal{E}$ are ordered so that $n(E_1, I) \geq n(E_2, I) \geq \ldots \geq n(E_m, I)$. Additionally assume that the optimal local model M_i constructs ℓ groups $X_1, \ldots, X_\ell$. Then, we have the following: if $E_{j_1} \in X_l$ and $E_{j_2} \in X_l$, with $j_2 > j_1$, then for all $E_{j'}$'s such that $j' \in \{j_1+1, \ldots, j_2-1\}$ we have that $E_{j'} \in X_l$.*

For the rest of this section we will assume that event types in $\mathcal{E}$ are ordered according to the ordering described in Observation 1. Given this ordering, we use $\mathcal{E}(j)$ to denote the event type at the j-th position of the order and $\mathcal{E}(j,l)$ to denote the set of event types at positions $j, j+1, \ldots, l-1, l$ on that order. Moreover, given data segment $\mathbf{S}_i$ we use $\mathbf{S}_i\left[j,l\right]$ to denote the subset of the events in $\mathbf{S}_i$ that correspond to event types in $\mathcal{E}(j,l)$.

Given the ordering of the event types in $\mathcal{E}$ (Observation 1) the following dynamic-programming recursion computes the minimum number of bits required to encode $\mathbf{S}_i$.

$$\begin{aligned}&\textsc{Ll}^*\left(\mathbf{S}_i\left[1,j\right]\right) = m\log m + \\ &\min\nolimits_{1\leq l\leq j}\ \left\{\textsc{Ll}^*\left(\mathbf{S}_i\left[1,l\right]\right) + \mathrm{U}\left(\mathbf{S}_i\left[l+1,j\right]\right) + 2\log m\right\},\end{aligned} \tag{6}$$

where

$$\begin{aligned}&\mathrm{U}\left(\mathbf{S}_i\left[l+1,j\right]\right) = \\ &= -\sum_{E\in\mathcal{E}(l+1,j)} n\left(E,I_i\right)\log p^* \\ &\quad -\sum_{E\in\mathcal{E}(l+1,j)} \left(|I| - n\left(E,I\right)\right)\log(1-p^*),\end{aligned}$$

and by Proposition 2 p^* is given by

$$p^* = \sum_{E \in \mathcal{E}(l+1,j)} \frac{n(E,I)}{|I|}.$$

The $m \log m$ term in Recursion (6) corresponds to the cost of encoding the ordering of the event types in $\mathbf{S}_i$, while the term $2 \log m$ encodes the number of bits required to encode the occurrence probability of any event type in the group $\mathcal{E}(l+1,j)$ and the group itself. Note that the order of the event types needs to be sent only once per segment, while the probability of event appearance per group and the group information needs to be sent once per group.

Theorem 2. *The* `Local-DP` *algorithm that evaluates Recursion (6) finds the optimal local model for the data segment* $\mathbf{S}_i$ *in polynomial time.*

Proof. The proof of this theorem is a direct consequence of the fact that 1-dimensional clustering can be done in polynomial time using dynamic programming [Bellman 1961]. □

The running time of the `Local-DP` algorithm is $O(m^2)$. For every index j the algorithm recurses over all values of l in the interval $1 \leq l \leq j$. Since the largest value of j is m, the running time of the algorithm is $O(m^2)$. This quadratic running time is under the assumption that in a preprocessing step we can compute the values of the U() function for all the combination of indices j and l. In fact, the asymptotic term $O(m^2)$ also contains the hidden cost of sorting the event types in $\mathcal{E}$ based on their frequency of occurrence in $\mathbf{S}_i$, which is $O(m \log m)$.

Note that a proposition similar to Proposition 1 of Section 5.1 can also be applied here. Informally, this means that event types that do not occur in $\mathbf{S}_i$ can be ignored when evaluating Recursion (6).

5.4 The `LocalGreedy` Algorithm

Similar to the `Greedy` algorithm for finding the optimal segment boundaries in $[1,n]$ (see Section 5.2), we give here a greedy alternative to the `Local-DP` algorithm that we call the `LocalGreedy` algorithm. By using the same data structures as the ones described in Section 5.2 the running time of the `LocalGreedy` algorithm is $O(m \log m)$.

As the `Greedy` algorithm, `LocalGreedy` computes the global partitioning X of $\mathbf{S}_i$ in a bottom-up fashion. It starts with grouping X^1, where each event type is allocated its own group. At the t-th step of the algorithm grouping X^t is considered, and the algorithm merges the two groups that introduce the maximum decrease in $\textsc{Ll}(\mathbf{S}_i, M_i)$. This merge leads to partition X^{t+1}. If no merging that causes cost reduction exists, the algorithm stops and outputs partition X^t.

5.5 Putting the Algorithms Together

Both `Segment-DP` and `Greedy` algorithms require a function that evaluates $\textsc{Ll}^*$ for different data intervals. The value of $\textsc{Ll}^*$ can be evaluated using either `Local-DP` or `LocalGreedy` algorithms. This setting creates four different alternative algorithms for solving the Summarization problem; the `DP-DP` that combines `Segment-DP` with `Local-DP`, the `DP-Greedy` that combines `Segment-DP` with `LocalGreedy`, the

`Greedy-DP` that combines `Greedy` with `Local-DP` and `Greedy-Greedy` that combines `Greedy` with `LocalGreedy`. `DP-DP` gives the optimal solution to the SUMMARIZATION problem. However, all other combinations also provide high-quality results, while at the same time they give considerable computational speedups.

In terms of asymptotic running times the `DP-DP` algorithm requires $O(n^2m^2)$ time, the `DP-Greedy` $O(n^2m\log m)$, the `Greedy-DP` $O(m^2n\log n)$ and the `Greedy-Greedy` algorithm time $O(nm\,\log n\log m)$.

6. EXTENSIONS TO OVERLAPPING SEGMENTS AND SEGMENTS SEPARATED BY GAPS

In many cases, can be of interest to allow the segments of the segmental groupings to overlap or be separated by gaps – pieces of the sequence that do not belong to *any* segment. We call the segmental groupings that allow for overlapping segments *O-segmental groupings.* Similarly, we refer to the segmental groupings with gaps as *G-segmental groupings.* The focus of this section is on discussing the impact of these extensions to the algorithms we discussed before.

6.1 Finding O-segmental groupings

For O-segmental groupings, we allow for a total of B overlap between the segments in the output segmental grouping. In other words, there can be overlapping segments as long as the total number of timestamps in which overlap occurs is no more than B. We call the problem of finding the optimal O-segmental grouping with at most B gaps as the O-SUMMARIZATION problem. The formal definition of this problem is the following.

PROBLEM 3. *(O-*SUMMARIZATION*) Consider integer B and event sequence $\mathbf{S}$ over observation period $[1,n]$ in which event types from set $\mathcal{E}$ occur. Given these as input, find integer k and an O-segmental grouping M of $\mathbf{S}$ into $(\mathbf{S}_1,\ldots,\mathbf{S}_k)$. The segments $(\mathbf{S}_1,\ldots,\mathbf{S}_k)$ are allowed to overlap in at most B timestamps. Additionally, the best local model M_i for each segment $\mathbf{S}_i$ needs to be found so that the total description length*

$$\textsc{Tl}\left(\mathbf{S},M,B\right) \;=\; k\log n+\sum_{i=1}^{k}\textsc{Ll}\left(\mathbf{S}_i,M_i\right), \tag{7}$$

is minimized.

Note that in the above problem definition we have extended the list of arguments of function TL to additionally include the integer B, i.e., the upper bound on the total number of overlaps are allowed in the output O-segmental grouping. Apart from that, Problem 3 is very similar to the general SUMMARIZATION problem defined in Problem 1.

Next, we give a variant of the dynamic-programming recursion given by Equation 5 that can provide the optimal solution to the O-SUMMARIZATION problem. Recall, that for interval $I\subseteq[1,n]$ we use $\textsc{Ll}^*\left(\mathbf{S}\left[I\right]\right)$ to denote the minimum value of $\textsc{Ll}\left(\mathbf{S}\left[I\right],M_i\right)$, over all possible local models M_i. Similarly, we use $\textsc{Tl}^*\left(\mathbf{S},B\right)$ to denote the minimum value of $\textsc{Tl}\left(\mathbf{S},M\right)$ over all possible summaries M that allow for at most B overlaps between the segments. Then, Problem 3 can be solved optimally by evaluating the following dynamic-programming recursion. For every $1 \leq i \leq n$ and $0 \leq b_1, b_2 \leq B$,

$$\begin{aligned} &\textsc{Tl}^*\left(\mathbf{S}[1,i]\right) = &(8)\\ &= \min_{1\leq j\leq i} \min_{b_1+b_2\leq B} \left\{ \textsc{Tl}^*\left(\mathbf{S}[1,j], b_1\right) + \textsc{Ll}^*\left(\mathbf{S}\left[j+1-b_2, i\right]\right)\right) \right\}. \end{aligned}$$

Namely, from the total B overlaps that are allowed b_1 of them can be used for modelling the prefix sequence $\mathbf{S}[1,j]$ and b_2 of them can be used for the last segment that, when having no overlap, would start at timestamp $j+1$ and end at timestamp i. Note that not all B overlaps need to be used; the only constraint that at most B overlaps can appear.

If the `Local-DP` algorithm is used for the evaluation of $\textsc{Ll}^*$, then $\textsc{Tl}^*\left(\mathbf{S}, B\right)$ can be evaluated in time $O\left(n^2B^2m^2\right)$, where $B = O(n)$. On the other hand, if we use the suboptimal `LocalGreedy` algorithm for the evaluation of $\textsc{Ll}^*$, then Recursion (8) can be evaluated in $O\left(n^2B^2m\log m\right)$ time.

6.2 Finding G-segmental groupings

In the case of G-segmental groupings, we allow for a total of G gaps to exist between the segments of the segmental grouping. In other words, there can be at most G timestamps from the whole sequence that will not belong to *any* segment. We call the problem of finding the optimal G-segmental grouping with at most G gaps as the G-Summarization problem. This problem is formally defined as follows.

Problem 4. *(G-*Summarization*) Consider integer G and event sequence $\mathbf{S}$ over observation period $[1,n]$ in which event types from set $\mathcal{E}$ occur. Given these as input, find integer k and an G-segmental grouping M of $\mathbf{S}$ into $(\mathbf{S}_1,\ldots,\mathbf{S}_k)$. The segments $(\mathbf{S}_1,\ldots,\mathbf{S}_k)$ are allowed to have at most G gaps between them. Additionally, the best local model M_i for each segment $\mathbf{S}_i$ needs to be found so that the total description length*

$$\textsc{Tl}\left(\mathbf{S}, M, G\right) \;=\; k\log n + \sum_{i=1}^{k} \textsc{Ll}\left(\mathbf{S}_i, M_i\right), \tag{9}$$

is minimized.

As before, we have extended the list of arguments of function $\textsc{Tl}$ to additionally include the integer G, i.e., the upper bound on the total number of gaps that are allowed the output G-segmental grouping to have. Other than this addition, Problem 4 is very similar to the general Summarization problem defined in Problem 1.

Next, we give yet another variant of the dynamic-programming recursion given by Equation (5) that can provide the optimal solution to the G-Summarization problem. As before, for every interval $I \subseteq [1,n]$ we use $\textsc{Ll}^*\left(\mathbf{S}\left[I\right]\right)$ to denote the minimum value of $\textsc{Ll}\left(\mathbf{S}\left[I\right], M_i\right)$, over all possible local models M_i. Similarly, we use $\textsc{Tl}^*\left(\mathbf{S}, G\right)$ to denote the minimum value of $\textsc{Tl}\left(\mathbf{S}, M\right)$ over all possible summaries M that allow for at most G gaps between the segments. Then, Problem 4 can be solved optimally by evaluating the following dynamic-programming recursion. For every $1 \leq i \leq n$,

$$\textsc{Tl}^*\left(\mathbf{S}[1,i]\right) = \tag{10}$$
$$= \min\Big\{\textsc{Tl}^*\left(\mathbf{S}[1,i-1],G-1\right), \min_{1\leq j\leq i}\left(\textsc{Tl}^*\left(\mathbf{S}[1,j],G\right)+\textsc{Ll}^*\left(\mathbf{S}\left[j+1,i\right]\right)\right)\Big\}.$$

Namely, if a gap is used for modelling the i-th timestamp of $\mathbf{S}$, then the G-segmental grouping of the prefix sequence $\mathbf{S}[1,i-1]$ is allowed to contain at most $G-1$ gaps. Otherwise, a new segment spanning the subsequence $\mathbf{S}[j+1,i]$ is introduced and the maximum number of gaps allowed in modelling subsequence $\mathbf{S}[1,j]$ is still G.

Using the `Local-DP` algorithm for evaluating $\textsc{Ll}^*$ for every segment, the evaluation of Recursion (10) gives the optimal G-segmental grouping in time $O\left(Gn^2m^2\right)$. Although in practice the values of G are expected to be much smaller than n ($G << n$), in the worst case we have that $G = O\left(n\right)$. Therefore, the asymptotic running time of the dynamic-programming algorithm described by the above recursion is $O\left(n^3m^2\right)$. If we use `LocalGreedy` for the evaluation of $\textsc{Ll}^*$ for every segment, then the evaluation of Recursion (10) needs $O\left(Gn^2m\log m\right) = O\left(n^3m\log m\right)$ time.

7. EXPERIMENTAL EVALUATION

In this section we report our experiments on a set of synthetic and real datasets. The main goal of the experimental evaluation is to show that all four algorithms we developed for the Summarization problem (see Section 5) give high-quality results. That is, we show that even our non-optimal greedy-based algorithms (`DP-Greedy`, `Greedy-DP` and `Greedy-Greedy`) use close to the optimal number of bits to encode the input event sequences, while producing meaningful summaries. Moreover, the greedy-based methods provide enormous computational speedups compared to the optimal `DP-DP` algorithm.

The implementations of our algorithms are in Java Version 1.4.2. The experiments were conducted on a Windows XP SP 2 workstation with a 3GHz Pentium 4 processor and 1 GB of RAM.

We evaluate the quality of the solutions produced by an algorithm A, by reporting the *compression ratio* CR(A), where A is any of the algorithms: `DP-DP`, `DP-Greedy`, `Greedy-DP` and `Greedy-Greedy`. If M_A is the summary picked by algorithm A as a solution to the Summarization problem with input $\mathbf{S}$, then, we define the compression ratio of algorithm A to be

$$\mathrm{CR}(A) = \frac{\textsc{Tl}\left(\mathbf{S},M_A\right)}{\textsc{Tl}\left(\mathbf{S},M_{\mathrm{unit}}\right)}. \tag{11}$$

Summary M_{unit} is the model that describes every event on $\mathbf{S}$ separately; such a model has n segment boundaries (one segment per timestamp) and m groups per segment and it corresponds to the model where no summarization is done. By definition, compression ratio takes values in $[0,1]$; the smaller the value of CR(A) the better the compression achieved by algorithm A.

7.1 Experiments on Synthetic Data

In this section we give experiments on synthetic datasets. The goal of these experiments is threefold. First to demonstrate that our algorithms find the correct model

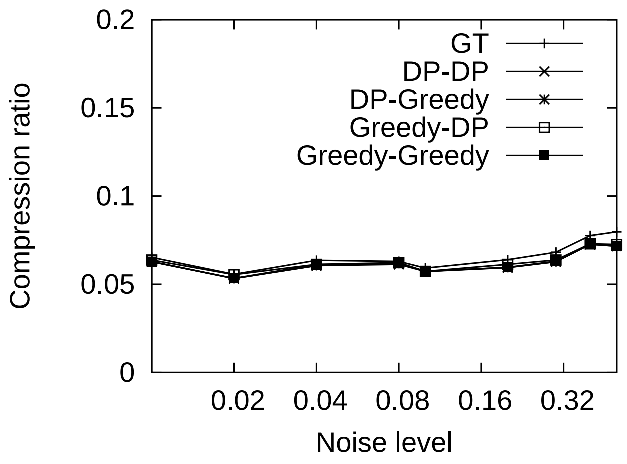


Fig. 2. Synthetic datasets: $n = 1000$, $m = 20$, $k = 10$; x-axis: noise level $V \in \{0.01, 0.02, 0.04, 0.08, 0.1, 0.2, 0.3, 0.4, 0.5\}$, y-axis: compression ratio for algorithms `DP-DP`, `DP-Greedy`, `Greedy-DP` and `Greedy-Greedy`.

used for the data generation; second to show that they significantly compress the input datasets; third to show that the greedy alternatives, though not provably optimal perform as well as the optimal `DP-DP` algorithm in practice.

The datasets: We generate synthetic datasets as follows: we first fix n, the length of the observation period, m, the number of different event types that appear in the sequence and k, the number of segments that we artificially "plant" in the generated event sequence. In addition to $\{0\}$ and $\{n+1\}$ we select $k-1$ other unique segment boundaries at random from points $\{2, \ldots, n\}$. These boundaries define the k segments. Within each segment $I_i = [b_i, b_{i+1})$ we randomly pick the number of groups to be formed. Each such group X_{ij}, is characterized by parameter $p(X_{ij})$, that corresponds to the probability of occurrence of each event type in X_{ij} in segment I_i. The values of $p(X_{ij})$ are normally distributed in $[0, 1]$.

Parameter V is used to control the noise level of the generated event sequence. When $V = 0$, for every segment I_i and every X_{ij} in I_i, events of any type $E \in X_{ij}$ are generated independently at every timestamp $t \in I_i$ with probability $p(X_{ij})$. For noise levels $V > 0$, any event of type $E \in X_{ij}$ is generated at each point $t \in I_i$ with probability sampled from the normal distribution $\mathcal{N}(p(X_{i,j}), V)$.

Accuracy of the algorithms: Figure 2 shows the compression ratio of the four different algorithms (`DP-DP`, `DP-Greedy`, `Greedy-DP` and `Greedy-Greedy`) as a function of the increasing noise level V that takes values in $[0.01, 0.5]$. For this experiment we fix $n = 1000$, $k = 10$ and $m = |\mathcal{E}| = 20$. In addition to our four algorithms, we also show the compression ratio of the *ground-truth* model (`GT`). This is the model that has been used in the data-generation process. From Figure 2 we observe that all four algorithms provide summaries with small CR values, close to 7%.[2] Furthermore, this compression ratio is very close the compression ratio achieved by

[2]Recall that the smaller the value of CR the better the summary produced by an algorithm.

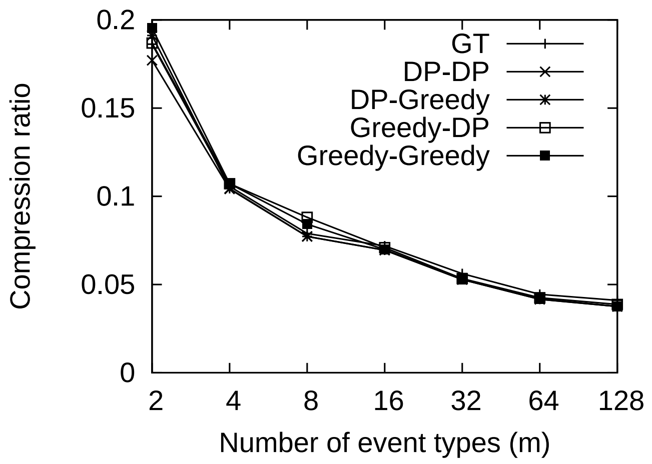
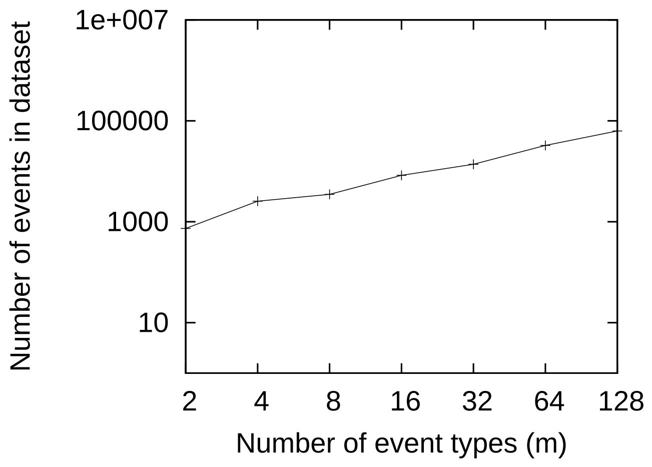


Fig. 3. Synthetic datasets: $n = 1000$, $V = 0.04$, $k = 10$; x-axis: number of event types $m \in \{2, 4, 8, 16, 32, 64, 128\}$, y-axis: compression ratio for algorithms `DP-DP`, `DP-Greedy`, `Greedy-DP` and `Greedy-Greedy`.

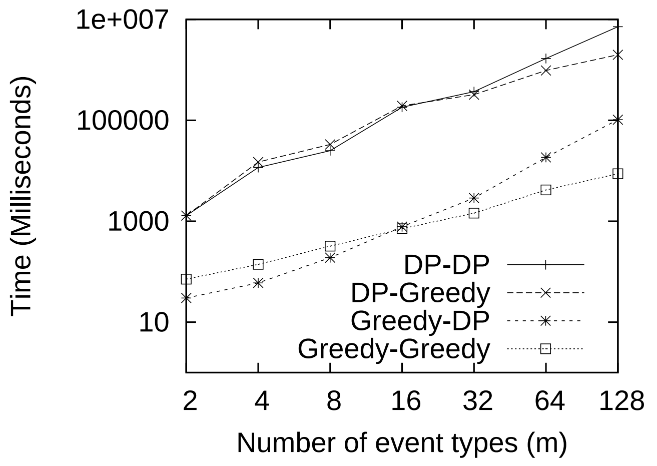
the ground-truth model. In fact for high noise levels ($V = 0.3, 0.4, 0.5$) the CR achieved by our algorithms is *better* than the CR of the ground-truth model. This is because for high noise levels, the data-generation model is less likely to be the optimal model to describe the data. Overall, even the greedy-based algorithms exhibit performance almost identical to the performance of the optimal `DP-DP` algorithm in terms of the number of bits required to encode the data.

Figure 3, shows the compression ratio of our algorithms as a function of the number of event types m that appear in the sequence. For this experiment, we vary m to take values $\{2, 4, 8, 16, 32, 128\}$ and fix the rest of the parameters of the data-generation process to $n = 1000$, $k = 10$ and $V = 0.04$. As in the previous experiment, we can observe that the compression ratio achieved by our algorithms is almost identical to the compression ratio achieved by the corresponding ground-truth model. Furthermore, we can observe that all our algorithms exhibit the same compression ratio and thus can be used interchangeably. Notice that as the number of event types increases, the compression ratio achieved by both the ground-truth model as well as the models discovered by our algorithms decreases, i.e., better summaries are found when compared to the raw data. This is because the more event types appear in the data, the more local patterns there are, which are discovered by our summarization methods. M_{unit} on the other hand, is oblivious to the existence of local groupings. As a result, for large number of event types the denominator in Equation 11 grows much faster than the numerator.

Increasing the number of event types also increases the total number of events appearing in the sequence, and thus leads to higher running times. Figure 4(a) shows the total number of events in the generated sequence as a function of the number of different event types, and Figure 4(b) shows the actual running times (in milliseconds) of our algorithms as a function of the different event types (and consequently as a function of the actual number of events occurring in the gener-



(a) Total number of events



(b) Noise level 0.04 and increasing number of event types $m = |E|$

Fig. 4. Performance measurements - synthetic datasets: $n = 1000$, $k = 10$, $V = 0.04$ and $m = |\mathcal{E}| \in \{2, 4, 8, 16, 32, 128\}$.

ated sequence). The figures illustrate the significant performance advantage of the greedy-based methods over the dynamic-programming based methods.

7.2 Experiments with Real Datasets

In this section we further illustrate the utility of our algorithms in a real-life scenario. By using event logs managed by Windows XP we show again that all four algorithms considerably compress the data and that they produce equivalent and intuitive models for the input sequences.

The real datasets consist of the **application log**, the **security log** and the

system log displayed by the Windows XP Event Viewer on our machines[3]. The **application log** contains events logged by application programs. The **security log** records events such as valid and invalid logon attempts, as well as events related to usage of resources. Finally, the **system log** contains events logged by Windows XP system components. Each one of the three log files we use stores log records with the following fields: (`Event_Type`, `Date`, `Time`, `Source`, `Category`, `Event`, `User`, `Computer`). We exported each one of the three log files into a separate file and processed them individually.

Our **application log** spans a period from June 2007 to November 2007, the **security log** the period from May 2007 to November 2007 and the **system log** the period from November 2005 to November 2007. For all these files we consider all the logged events found on our computer, without any modification.

	application	**security**	**system**
Observation Period	06/07-11/07	05/07 - 11/07	11/05-11/07
Observation Period (millisecs)	12,313,576,000	14,559,274,000	61,979,383,000
Number of events (N)	2673	7548	6579
Number of event types (m)	45	11	64
RUNNING TIMES (secs)			
`DP-DP`	3252	2185	34691
`CP-Greedy`	976	2373	8310
`Greedy-DP`	18	1	91
`Greedy-Greedy`	7	1	24
COMPRESSION RATIO CR(A)			
`DP-DP`	0.04	0.32	0.03
`DP-Greedy`	0.04	0.32	0.03
`Greedy-DP`	0.04	0.34	0.03
`Greedy-Greedy`	0.04	0.33	0.03

Table I. Experiments with real datasets

Considering as event types the unique combinations of `Event_Type`, `Source` and `Event` and as timestamps of events the combination of `Date` and `Time`, we get the datasets with characteristics described in Table I (upper part). Note that the system records the events at a millisecond granularity level. Therefore, the actual length of the timelines (n) for the **application**, **security** and **system** logs are $n = 12,313,576,000$, $n = 14,559,274,000$ and $n = 61,979,383,000$ respectively. However, this fact does not affect the performance of our algorithms which by Proposition 1 only depends on the number of unique timestamps N on which events actually occur; the values of N for the three datasets are are $N = 2673$, $N = 7548$ and $N = 6579$ respectively.

Elapse times for the computations are reported in seconds in Table I. For example, the elapse time for the `DP-DP` method with the system dataset is roughly 10 hours; which makes this method impractical for large datasets containing a large

[3]We use the default Windows configuration for logging, so similar datasets exist on all Windows machines.

number of event types. We see that the `Greedy-Greedy` algorithm ran in 24 seconds for the same dataset.

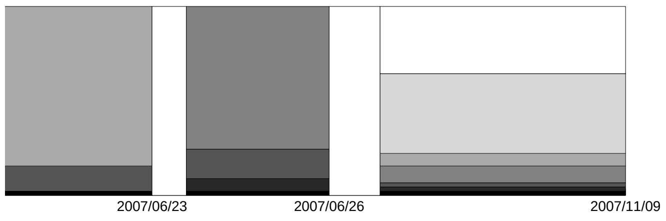
Finally, the compression ratio (CR) achieved for the three datasets by the four different algorithms are also reported in Table I. The results indicate that the greedy-based methods produce as good summaries as the optimal `DP-DP` algorithm. Therefore, the results of Table I further illustrate that despite the optimality of the solutions produced by `DP-DP`, the latter algorithm can prove impractical for very large datasets. Greedy-based algorithms on the other hand, give almost as accurate and condensed summaries and are much more efficient in practice.

Structural similarity of the results. We have observed that all our algorithms achieve almost identical compression ratios for the same data. A natural question to ask is whether the actual models they output are also *structurally* similar. In other words, do the reported segmental groupings have the same segment boundaries and are the groups within the reported segments similar?

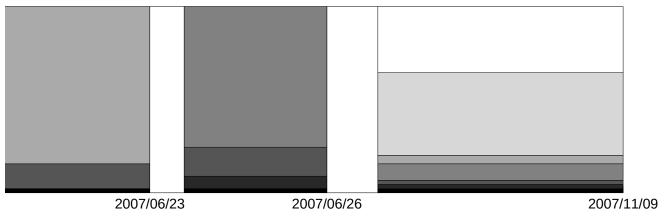
The goal of Figures 5 and 6 is to answer this question in an affirmative way. These two figures visualize the output segmental groupings reported by algorithms `DP-DP`, `DP-Greedy`, `Greedy-DP` and `Greedy-Greedy` (Figures 5(a) and 6(a), 5(b) and 6(b), 5(c) and 6(c), 5(d) and 6(d) respectively) for the **application log** and the **system log** datasets.

Each subfigure corresponds to the output of a different algorithm and should be interpreted as follows: the x-axis corresponds to the timeline that is segmented, with the vertical lines defining the segment boundaries on the timeline. Within each segment, different groups of event types are represented by different colors (darker colors represent groups that have higher probability of occurrence within a segment). The vertical length of each group is proportional to its size. The main conclusion that can be drawn from Figures 5 and 6 is that the output segmental groupings of `DP-DP` and `DP-Greedy` algorithms are almost identical, and the output of all four algorithms are very close to each other. The apparent similarity is that all segmentations have a large segment in the beginning of the observation period and an even larger segment towards its end. In these segments the same number of groups are observed. In the interval that is in-between these two large segments, the outputs of `DP-DP`, `DP-Greedy` and `Greedy-DP` exhibit very similar structure, by identifying almost identical segment boundaries. Seemingly different are the boundaries found by `Greedy-Greedy` algorithm. However, a closer look shows that these latter boundaries are not far from the boundaries identified by the other three algorithms; `Greedy-Greedy` in fact identified boundary positions very close to the boundary positions identified by the other three algorithms. Since the `Greedy-Greedy` algorithm is constrained to first choose boundaries between immediate neighbors before choosing larger boundaries, it overlooks intervals of points on a timeline which together exhibit better segmentation than intervals formed from exploring immediate neighbors.

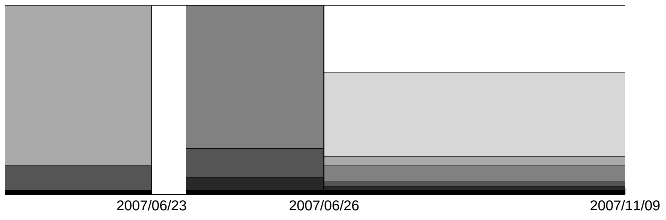
In fact, we have further looked at the segmental groupings output by the algorithms and judged their practical utility. For example, in the segmental grouping depicted in Figure 5(a) we noticed that the most frequent event types occurring in



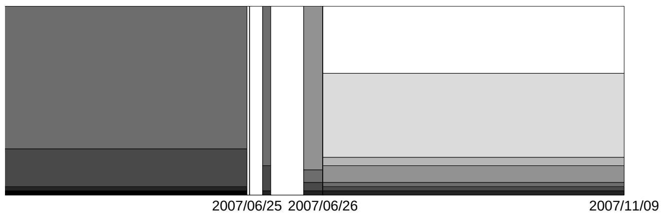
(a) `DP-DP` algorithm



(b) `DP-Greedy` algorithm



(c) `Greedy-DP` algorithm

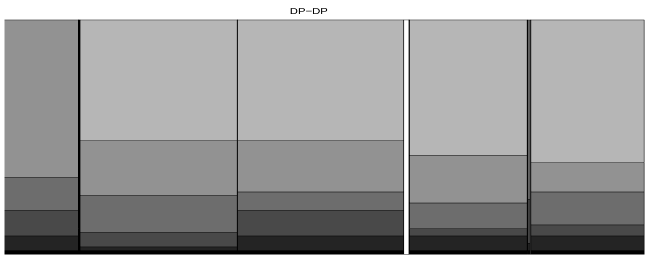


(d) `Greedy-Greedy` algorithm

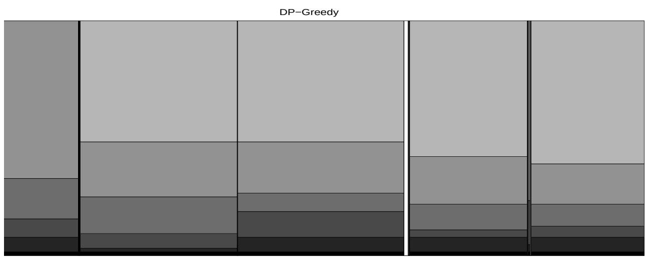
Fig. 5. Output segmental groupings of different algorithms for the **application log** data.

the first segment correspond to DB2-errors[4]. On the other hand, the most frequent group of event types occurring in the last segment correspond to DB-2 and other types of warnings. Moreover, the segments that appear to be free of occurrences of the majority of event types, correspond to segments where only few events occur; the type of these events suggest that security updates were happening in the system

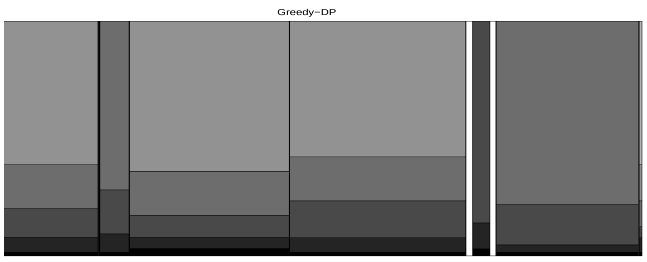
[4]DB-2 is the DataBase Management System (DBMS) used at IBM.



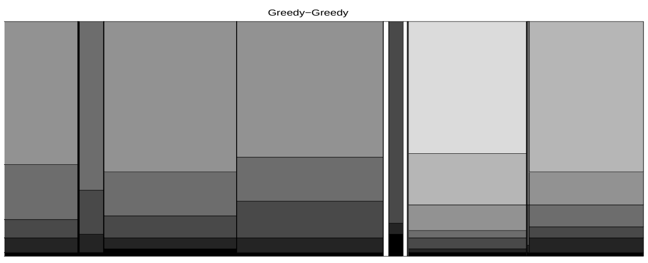
(a) `DP-DP` algorithm



(b) `DP-Greedy` algorithm



(c) `Greedy-DP` algorithm



(d) `Greedy-Greedy` algorithm

Fig. 6. Output segmental groupings of different algorithms for the **system log** data.

during these time intervals. The goal of this anecdote is mostly to highlight the fact that segmental groupings allow the system administrator to explore the available data without getting overwhelmed by it.

8. THE EVENTSUMMARIZER SYSTEM

In this section, we present a tool called EVENTSUMMARIZER [Kiernan and Terzi 2009] that exploits the algorithms given in this paper and which uses intuitive visual metaphors to render the segmentation found by the algorithms along with their local associations.

8.1 System architecture

The system is written in Java and runs on top of any RDBMS through JDBC. Using the Graphical User Interface (GUI), the user can: *load* the data and *summarize* it. The data is retrieved into memory using SQL queries. The summarization task is performed on the retrieved data. The user specifies the *summary attribute*, and the *summarization algorithm* used to produce the segmental grouping. The summary attribute should define a timeline and is of type `timestamp`.

Once the segmental grouping of the input sequence is computed, the results are presented to the user. They include both the segmentation of the input timeline into intervals and the description of the groups within every segment. Details on the display of segmental groupings is given in the next section.

8.2 EVENTSUMMARIZER Functionality

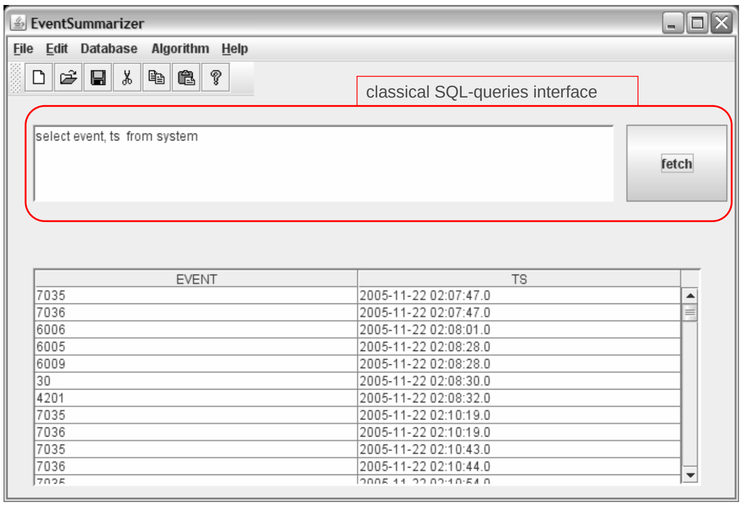
In this section, we present EVENTSUMMARIZER's functionality and illustrate it's usage. The dataset we use for this purpose is the **system log** file that was described earlier in Section 7.2.

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. As mentioned earlier, the timestamps appearing in `TS` span the period from November 2005 to November 2007.

Figure 7 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 SQL queries. The result tuples of the query (in this case `select event, ts from system`) are rendered in the lower part of the screen.

Once the selected data is fetched, the user can then summarize it. The summarization algorithm is selected on the menu bar of EVENTSUMMARIZER. The actual summarization task can be performed after the data and the algorithm have been selected. Figure 8(a) shows the segmentation of the input timeline that is produced as a result of summarization using the (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. The bar shows the partition of the input data's 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 8(b) shows the actual grouping of the tuples within a single (in this case the last) segment. The segment is selected by performing a simple mouseclick on it. The grouping of the event types within this interval is rendered below the segmentation bar and displayed as a table. The first attribute of this new table is the `Key`


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
7035	2005-11-22 02:10:54.0

Fig. 7. Data selection in EventSummarizer using standard SQL queries

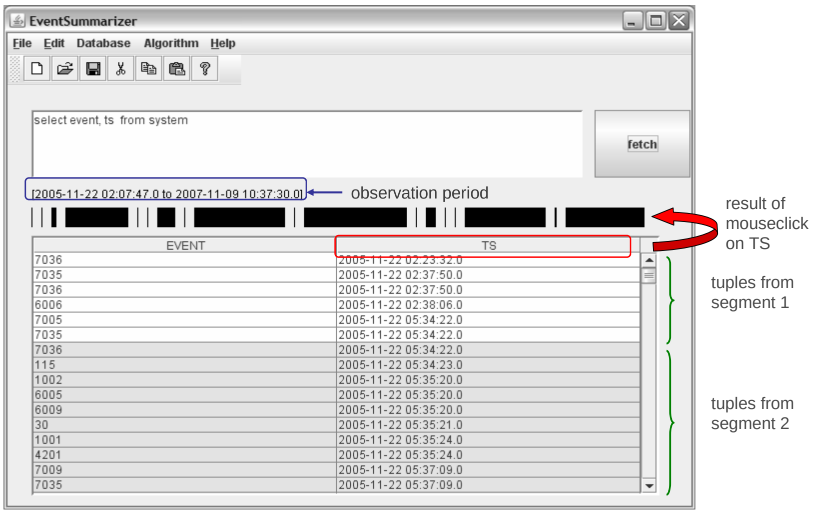
attribute and it corresponds to the event type. The second attribute is the `Count` attribute and it shows for every event type the number of its occurrences within the examined time interval. The last attribute, `Group`, takes integer values that denote the group-id to which the different event types belong. 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.

9. CONCLUSIONS

We proposed a framework and an algorithmic solution to the problem of summarizing large event sequences that represent activities of systems and individuals over time. Our framework is based on building segmental groupings of the data. A segmental grouping splits the timeline into segments; within each segment events of different types are grouped based on their frequency of occurrence in the segment. Our approach is based on the MDL principle that allows us to build summaries that are short and accurately describe the data without over-fitting.

We have presented EventSummarizer, a tool for summarizing large event sequences that exploits the algorithms given in this paper. We have created visual metaphors to render the segmentations found by the algorithms and designed mouse-click interactions to explore individual segments with their local associations. Through illustrations, we showed that EventSummarizer is easy to use and gives easy-to-interpret results.

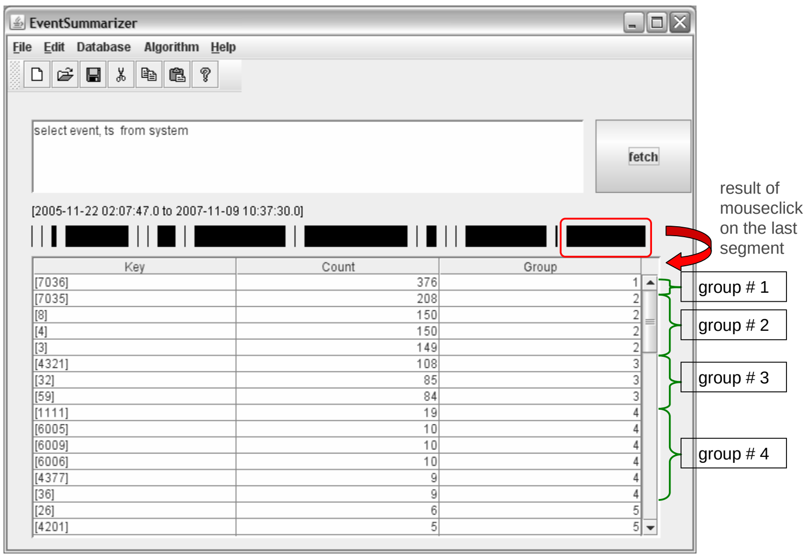
Our contribution is in the definition of the segmental groupings as a model for summarizing event sequences. This model when combined with the MDL principle allowed us to naturally transform the event-sequence summarization problem to a concrete optimization problem. We showed that this problem can be solved



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


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

Fig. 8. EventSummarizer functionality: Figure 8(a) shows the visualization of the segmentation output by EventSummarizer and Figure 8(b) shows the grouping of event types within a selected segment.

optimally in polynomial time using a combination of two dynamic-programming algorithms. Furthermore, we designed and experimented with greedy algorithms for the same problem. These algorithms, though not provably optimal, are extremely efficient and in practice give high-quality results. All our algorithms are parameter free and when used in practice produce meaningful summaries.

REFERENCES

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

Allan, J., Gupta, R., and Khandelwal, V. 2001. Temporal summaries of news topics. In *SIGIR*. 10–18.

Bellman, R. 1961. On the approximation of curves by line segments using dynamic programming. *Communications of the ACM 4,* 6.

Bettini, C., Wang, X. S., and Jajodia, S. 1998. Mining temporal relationships with multiple granularities in time sequences. *Data Engineering Bulletin 21,* 1, 32–38.

Brants, T. and Chen, F. 2003. A system for new event detection. In *International Conference on Research and Development in Information Retrieval (SIGIR)*. 330–337.

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

Elfeky, M., Aref, W., and Elmagarmid, A. 2004. Using convolution to mine obscure periodic patterns in one pass. In *Proc. of the Nineth Int'l Conference on Extending Database Technology (EDBT).* Heraklion, Crete.

Gionis, A. and Mannila, H. 2003. Finding recurrent sources in sequences. In *International Conference on Research in Computational Molecular Biology (RECOMB)*. 123–130.

Guha, S., Koudas, N., and Shim, K. 2001. Data-streams and histograms. In *Symposium on the Theory of Computing (STOC)*. 471–475.

Han, J., Dong, G., and Yin, Y. 1999. Efficient mining of partial periodic patterns in time series database. In *Proc. of the 15th Int'l Conference on Data Engineering.* Sydney, Australia.

Han, J., Gong, W., and Yin, Y. 1998. Mining segment-wise periodic patterns in time-related databases. In *KDD*. 214–218.

Karras, P., Sacharidis, D., and Mamoulis, N. 2007. Exploiting duality in summarization with deterministic guarantees. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 380–389.

Keogh, E. J., Chu, S., Hart, D., and Pazzani, M. J. 2001. An online algorithm for segmenting time series. In *IEEE International Conference on Data Mining (ICDM)*. 289–296.

Kiernan, J. and Terzi, E. 2009. Eventsummarizer: A tool for summarizing large event sequences. In *International Conference on Extending Database Technology (EDBT)*. To appear.

Kilpeläinen, P., Mannila, H., and Ukkonen, E. 1995. Mdl learning of unions of simple pattern languages from positive examples. In *EuroCOLT*. 252–260.

Kleinberg, J. 2003. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery 7*, 373–397.

Koivisto, M., Perola, M., Varilo, T., et al. 2003. An MDL method for finding haplotype blocks and for estimating the strength of haplotype block boundaries. In *Pacific Symposium on Biocomputing.* 502–513.

Li, W. 2001a. Dna segmentation as a model selection process. In *International Conference on Research in Computational Molecular Biology (RECOMB)*. 204–210.

Li, W. 2001b. New stopping criteria for segmenting dna sequences. *Physical Review Letters 86*, 5815–5818.

Ma, S. and Hellerstein, J. L. 2001. Mining partially periodic event patterns with unknown periods. In *Proc. of the 17th Int'l Conference on Data Engineering.* Heidelberg, Germany.

Mannila, H. and Salmenkivi, M. 2001. Finding simple intensity descriptions from event sequence data. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 341–346.

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

MANNILA, H., TOIVONEN, H., AND VERKAMO, A. I. 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery 1,* 3, 259–289.

MEHTA, M., RISSANEN, J., AND AGRAWAL, R. 1995. Mdl-based decision tree pruning. In *KDD*. 216–221.

PAPADIMITRIOU, S. AND YU, P. 2006. Optimal multi-scale patterns in time series streams. In *ACM SIGMOD International Conference on Management of Data*. 647–658.

PEI, J., HAN, J., AND WANG, W. 2007. Constraint-based sequential pattern mining: the pattern-growth methods. *J. Intell. Inf. Syst. 28,* 2, 133–160.

RABINER, L. R. AND JUANG, B. H. 1986. An introduction to Hidden Markov Models. *IEEE ASSP Magazine*, 4–15.

RISSANEN, J. 1978. Modeling by shortest data description. *Automatica 14*, 465–471.

RISSANEN, J. 1989. *Stochastic Complexity in Statistical Inquiry Theory.* World Scientific Publishing Co., Inc., River Edge, NJ, USA.

RUZZO, W. L. AND TOMPA, M. 1999. A linear time algorithm for finding all maximal scoring subsequences. In *ISMB*. 234–241.

SAKURAI, Y., PAPADIMITRIOU, S., AND FALOUTSOS, C. 2005. Braid: Stream mining through group lag correlations. In *ACM SIGMOD International Conference on Management of Data*. 599–610.

SRIKANT, R. AND AGRAWAL, R. 1996. Mining sequential patterns: Generalizations and performance improvements. In *EDBT*. 3–17.

SWAN, R. C. AND ALLAN, J. 2000. Automatic generation of overview timelines. In *SIGIR*. 49–56.

TERZI, E. AND TSAPARAS, P. 2006. Efficient algorithms for sequence segmentation. In *SIAM International Conference on Data Mining (SDM)*.

YANG, J., WANG, W., YU, P. S., AND HAN, J. 2002. Mining long sequential patterns in a noisy environment. In *ACM SIGMOD International Conference on Management of Data*. 406–417.

YANG, Y., AULT, T., PIERCE, T., AND LATTIMER, C. W. 2000. Improving text categorization methods for event tracking. In *International Conference on Research and Development in Information Retrieval (SIGIR)*. 65–72.

ZHU, Y. AND SHASHA, D. 2002. Statstream: Statistical monitoring of thousands of data streams in real time. In *Very Large Data Bases (VLDB) Conference*. 358–369.

Received Month Year; revised Month Year; accepted Month Year