Constructing Comprehensive Summaries of Large Event Sequences

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

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

ABSTRACT

Event sequences capture system and user activity over time. Prior research on sequence mining has mostly focused on discovering local patterns. Though interesting, these patterns reveal local associations and fail to 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 the entire sequence, while revealing local associations among events.

We formally define the summarization problem as an optimization problem that balances between shortness of the summary and 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

Keywords

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 time of occurrence. Network traffic data, alarms in telecommunication networks, logging systems are examples of applications that produce 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 to provide a global model of the data. Moreover, data-mining algorithms usually output too many patterns that may be overwhelming for the data analysts. In this paper, we bring up a new aspect of event sequence analysis, namely how to concisely summarize such event sequences.

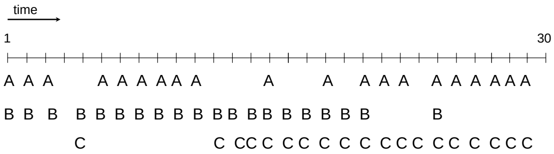
From the point of view of a data analyst, an event-sequence summarization system should have the following properties.

- **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 combination 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.

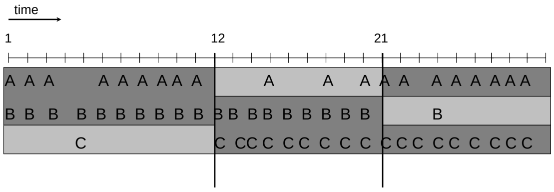
Despite the bulk of work on the analysis of event-sequences, to the best of our knowledge, there is no technique that satisfies all requirements discussed above. In this paper, we present a summarization technique that exhibits all these characteristics. More specifically,

- 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.

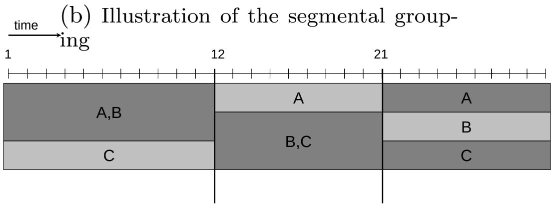
KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.



(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.

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); it contains 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 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.

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* to describe such 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, we conjecture that the techniques presented herein can also be applied to other diverse domains. For example, useful summaries can be constructed for biological data, data streams, and large documents using our methodology.

1.2 Roadmap

The rest of the paper is organized as follows; in Section 2 we give some basic notational conventions. In Section 3 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 4 and experiments are given in Section 5. We review the related work in Section 6 and conclude in Section 7.

2. 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 sub-

sequence $\mathbf{S}[b_j, b_{j+1}-1]$. 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} 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.

3. 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.

3.1 Minimum Description Length Principle

The MDL principle [15, 16] allows us to transform the requirement of balance between over-generalizing and overfitting 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 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 [11], genetic-sequence modeling [8], patterns in sets of strings [7] and many more. We devote the rest of the section to describe our instantiation of the MDL principle.

3.2 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

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

The number of bits required to describe data $\mathbf{S}_i$ given model M_i is $-\log(\Pr(\mathbf{S}_i|M))$[1]. Therefore, *local data cost* of $\mathbf{S}_i$ given M_i is

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

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* $\text{LM}(M_i)$. 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 $p(X_j)$. By standard arguments [15], we need $\log m$ bits to describe each one of the $p(X_j)$'s. Since there are ℓ groups we need a total of $\ell \log m$ 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

[1]By standard arguments the number of bits required for encoding an event with probability q is $-\log(q)$.

that is consistent with the partitioning $X_1, \ldots, X_\ell$,[2] 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

$$\textsc{Lm}(M_i) = 2\ell \log m + m \log m. \quad (2)$$

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

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

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 $p(X_j)$.

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

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

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

3.3 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*

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

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 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 $\mathbf{S}$.

4. 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*

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

Our algorithmic approach exploits the following statement.

Statement 1. *If the* LocalGrouping *problem (Problem 2) can be solved optimally in polynomial time, then the* Summarization *problem (Problem 1) can also be solved optimally in polynomial time.*

In the rest of this section we give optimal polynomial-time algorithms for the Summarization problem. The algorithms exploit the above statement. Moreover, we provide alternative sub-optimal, but practical and efficient, algorithms for the Summarization problem.

4.1 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* $1 \leq i \leq n$,

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

[2] 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.

The proof of optimality is omitted due to space constraints. However, a similar proof can be found in [2]. 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.

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.

4.2 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 $\textsc{Tl}\left(\mathbf{S}, M^t\right)$ that is caused by the removal of b_j from M^t. The impact may be positive if $\textsc{Tl}\left(\mathbf{S}, M^t\right)$ 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 $O(T_l |T| \log |T|)$.

4.3 Finding Optimal Local Models

In this section we show that the LOCALGROUPING can also be solved optimally in polynomial time using yet another dynamic-programming algorithm. We call this algorithm the `Local-DP` algorithm. The following proposition is at the core of the `Local-DP` algorithm.

PROPOSITION 2. *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* $E_{j'} \in X_l$.

Proposition 2 states that the grouping of the event types in interval I respects the ordering of event types with respect to their frequency of appearance in $\mathbf{S}[I]$.

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

PROPOSITION 3. *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* $1 \leq j \leq \ell$, *the value of* $p(X_j)$ *that minimizes* $\textsc{Ld}(\mathbf{S}_i | M_i)$ *is*

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

For the rest of this section we will assume that event types in $\mathcal{E}$ are ordered according to the ordering described in Proposition 2. 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 $\mathbf{S}_i[j, l]$ 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 $\mathcal{E}$ (Proposition 2) the following dynamic-programming recursion computes the minimum number of bits required to encode $\mathbf{S}_i$.

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

where

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

and by Proposition 3 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.*

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 4.1 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).

4.4 The `LocalGreedy` Algorithm

Similar to the `Greedy` algorithm for finding the optimal segment boundaries in $[1, n]$ (see Section 4.2), 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 4.2 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 $\text{LL}(\mathbf{S}_i, M_i)$. This merge leads to partition X^{t+1}. If no merging that causes cost reduction exists, the algorithm stops and outputs partition X^t.

4.5 Putting the Algorithms Together

Both `Segment-DP` and `Greedy` algorithms require a function that evaluates LL^* for different data intervals. The value of 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 `Local-Greedy`, the `Greedy-DP` that combines `Greedy` with `Local-DP` and `Greedy-Greedy` that combines `Greedy` with `Local-Greedy`. `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` $O(n^2 m \log m)$, the `Greedy-DP` $O(m^2 n \log n)$ and the `Greedy-Greedy` algorithm time $O(nm \log n \log m)$.

5. 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 4) 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* $\text{CR}(A)$, 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

$$\text{CR}(A) = \frac{\text{TL}(\mathbf{S}, M_A)}{\text{TL}(\mathbf{S}, M_{\text{unit}})}. \quad (7)$$

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 $\text{CR}(A)$ the better the compression achieved by algorithm A.

5.1 Experiments on Synthetic Data

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

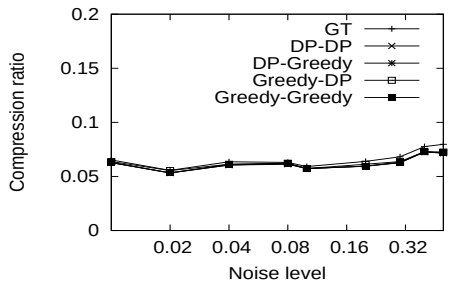


Figure 2: Synthetic datasets: $n = 1000$, $m = 20$, $k = 10$; x-axis: noise level $V \in \{0.01, 0.02, 0.04, 0.08, 0.1, 0.2, 0.3, 0.4, 0.5\}$, y-axis: compression ratio for algorithms `DP-DP`, `DP-Greedy`, `Greedy-DP` and `Greedy-Greedy`.

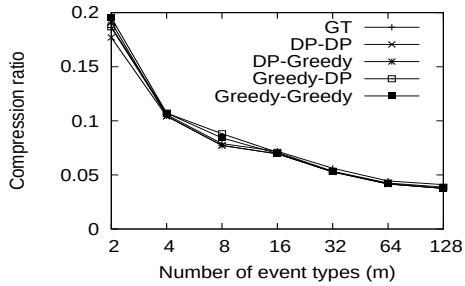


Figure 3: Synthetic datasets: $n = 1000$, $V = 0.04$, $k = 10$; x-axis: number of event types $m \in \{2, 4, 8, 16, 32, 64, 128\}$, y-axis: compression ratio for algorithms `DP-DP`, `DP-Greedy`, `Greedy-DP` and `Greedy-Greedy`.

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%.[3] Furthermore, this compression ratio is very close the compression ratio achieved by 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 7 grows much faster than the numerator.

5.2 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 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[4]. 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.

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 1 (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 1. 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 number of event types. We see that the `Greedy-`

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

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

	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 1: Experiments with real datasets

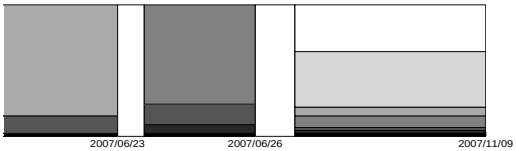
Greedy algorithm ran in 24 seconds for the same dataset.

Finally, the compression ratio (CR) achieved for the three datasets by the four different algorithms are also reported in Table 1. The results indicate that the greedy-based methods produce as good summaries as the optimal DP-DP algorithm. Therefore, the results of Table 1 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.

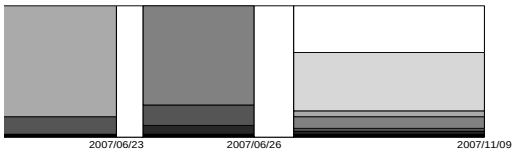
Structural similarity of the results. We have observed that all our algorithms achieve almost identical compression ratios for the same data. A natural question to ask is whether the actual models they output are also *structurally* similar. In other words, do the reported segmental groupings have the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

The goal of Figure 4 is to answer this question in an affirmative way. This figure visualizes the output segmental groupings reported by algorithms DP-DP, DP-Greedy, Greedy-DP and Greedy-Greedy (Figures 4(a), 4(b), 4(c), and 4(d) respectively) for the **application log** dataset.

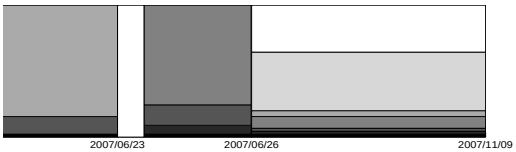
Each subfigure corresponds to the output of a different algorithm and should be interpreted as follows: the x-axis corresponds to the timeline that is segmented, with the vertical lines defining the segment boundaries on the timeline. Within each segment, different groups of event types are represented 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 Figure 4 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.



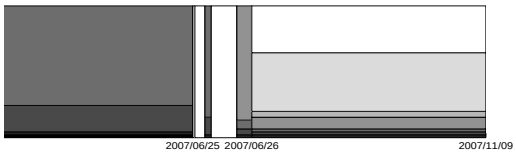
(a) DP-DP algorithm



(b) DP-Greedy algorithm



(c) Greedy-DP algorithm



(d) Greedy-Greedy algorithm

Figure 4: Output segmental groupings of different algorithms for the application log data.

6. 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 ([1, 3, 10, 13, 19]). That work mostly focuses on developing algorithms that identify configurations of discrete events clustered in time. Although those algorithms identify local event patterns, known as frequent episodes, they do not provide a global description of the event sequence neither do they care about the conciseness of the produced patterns.

Summarization of event sequences via a segmentation model is proposed in [9]. 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. In fact, one can think of our model as a generalization of the model proposed in [9] since in fact we split the event types into groups of constant intensities.

Also related is the segmentation framework developed by [8] 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 [8] finding the local model in each segment is an NP-hard task, while in our case this task is polynomial.

At a high level there is an obvious connection between our model and the standard segmentation model used for time-series segmentation (see [4, 5, 6, 18] and indicative, though not complete, set of references). Similarly, there is an equally interesting line of work that deals with the discovery of local patterns in time-series data, e.g., [12, 17, 20]. However, the connection to our work remains at a high level since we focus on event sequences and not on time series, while at the same time the local models we consider per segment are quite distinct from the models considered before. Same high-level connection exists between our model and HMMs [14]. However, the assumptions behind HMMs are different from the assumptions we make in this model.

7. CONCLUSIONS

We proposed a framework and an algorithmic solution to the problem of summarizing deluging event sequences that continuously record the activities of systems and individuals. 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.

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 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.

8. REFERENCES

[1] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *ICDE*, 1995.

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

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

[4] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *STOC*, pages 471–475, 2001.

[5] P. Karras, D. Sacharidis, and N. Mamoulis. Exploiting duality in summarization with deterministic guarantees. In *KDD*, pages 380–389, 2007.

[6] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001.

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

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

[9] H. Mannila and M. Salmenkivi. Finding simple intensity descriptions from event sequence data. In *KDD*, pages 341–346, 2001.

[10] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *KDD* , pages 146–151, 1996.

[11] M. Mehta, J. Rissanen, and R. Agrawal. Mdl-based decision tree pruning. In *KDD*, pages 216–221, 1995.

[12] S. Papadimitriou and P. Yu. Optimal multi-scale patterns in time series streams. In *SIGMOD* , pages 647–658, 2006.

[13] 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.

[14] L. R. Rabiner and B. H. Juang. An introduction to Hidden Markov Models. *IEEE ASSP Magazine*, pages 4–15, January 1986.

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

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

[17] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Braid: Stream mining through group lag correlations. In *SIGMOD*, pages 599–610, 2005.

[18] E. Terzi and P. Tsaparas. Efficient algorithms for sequence segmentation. In *SDM*, 2006.

[19] J. Yang, W. Wang, P. S. Yu, and J. Han. Mining long sequential patterns in a noisy environment. In *SIGMOD*, pages 406–417, 2002.

[20] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002.