Chapter 8
Finding Segmentations of Sequences

Ella Bingham

Abstract We describe a collection of approaches to inductive querying systems for data that contain segmental structure. The main focus in this chapter is on work done in Helsinki area in 2004-2008. Segmentation is a general data mining technique for summarizing and analyzing sequential data. We first introduce the basic problem setting and notation. We then briefly present an optimal way to accomplish the segmentation, in the case of no added constraints. The challenge, however, lies in adding constraints that relate the segments to each other and make the end result more interpretable for the human eye, and/or make the computational task simpler. We describe various approaches to segmentation, ranging from efficient algorithms to added constraints and modifications to the problem. We also discuss topics beyond the basic task of segmentation, such as whether an output of a segmentation algorithm is meaningful or not, and touch upon some applications.

8.1 Introduction

Segmentation is a general data mining technique for summarizing and analyzing sequential data. It gives a simplified representation of data, giving savings in storage space and helping the human eye to better catch an overall picture of the data. Segmentation problems arise in many data mining applications, including bioinformatics, weather prediction, telecommunications, text processing and stock market analysis, to name a few.

The goal in segmentation is to decompose the sequence, such as a time series or a genomic sequence, into a small number of homogeneous non-overlapping pieces, segments, such that the data in each segment can be described accurately by a

Ella Bingham
Helsinki Institute for Information Technology, University of Helsinki and Aalto University School of Science and Technology
e-mail: ella.bingham@hiit.fi

S. Džeroski, *Inductive Databases and Constraint-Based Data Mining,*
DOI 10.1007/978-1-4419-7738-0_8,

simple model. In many applications areas, this is a natural representation and reveals the high-level characteristics of the data by summarizing large scale variation. For example, in a measurement time series, each segment s_j could have a different mean parameter μ_j such that the measurement values x in segment s_j are modeled as $x = \mu_j + \text{noise}$.

Segmentation algorithms are widely used for extracting structure from sequences; there exist a variety of applications where this approach has been applied [3, 4, 6, 23, 29, 30, 32, 37, 38]. Sequence segmentation is suitable in the numerous cases where the underlying process producing the sequence has several relatively stable states, and in each state the sequence can be assumed to be described by a simple model. Naturally, dividing a sequence into homogeneous segments does not yield a perfect description of the sequence. Instead, a simplified representation of the data is obtained — and this is often more than welcome.

One should note that in statistics the question of segmentation of a sequence or time series is often called the change-point problem.

If no constraints are made between different segments, finding the optimal segmentation can be done for many model families by using simple dynamic programming [2] in $O(n^2k)$ time, where n is the length of the sequence and k is the number of segments. Thus one challenge lies in adding constraints that relate the segments to each other and make the end result more interpretable for the human eye. Another challenge is to make the computational task simpler. We will discuss both of these challenges in this chapter, and many more.

This chapter is a survey of segmentation work done in the Helsinki area: at Helsinki Institute for Information Technology, which is a joint research institute of University of Helsinki and Aalto University (part of it formerly known as Helsinki University of Technology), roughly between the years 2004 and 2008.

Notation. We assume that our data is a d-dimensional sequence T consisting of n observations, that is, $T = \langle t_1, \ldots, t_n \rangle$ where $t_i \in \mathbb{R}^d$. A *k-segmentation* S of T is a partition of $\langle 1, 2, \ldots, n \rangle$ into k non-overlapping contiguous subsequences (segments), $S = \langle s_1, \ldots, s_k \rangle$ such that $s_i = \langle t_{b(i)}, \ldots, t_{b(i+1)-1} \rangle$ where $b(i)$ is the beginning of the i:th segment. In its simplest case, segmentation collapses the values within each segment s into a single value μ_s which is e.g. the mean value of the segment. We call this value the *representative* of the segment. Collapsing points into representatives results in a loss of accuracy in the sequence representation. This loss of accuracy is measured by the *reconstruction error*

$$E_p(T,S) = \sum_{s \in S} \sum_{t \in s} ||t - \mu_s||^p.$$

The *segmentation problem* is that of finding the segmentation minimizing this reconstruction error. In practice we consider the cases $p = 1, 2$. For $p = 1$, the optimal representative of each segment is the median of the points in the segment, for $p = 2$ it is the mean of the points.

Depending on the constraints one imposes on the representatives, one can consider several variants of the segmentation problem, and we will discuss many of

them later in this chapter. Also, instead of representing the data points in a segment by a single representative, one can consider simple functions of the data points. Extensions of the methods presented in this chapter into functional representatives is often straightforward.

Figure 8.1 shows an example of a signal and its segmentation. In this simple case, the segments are represented by the mean values of the points belonging to a segment.

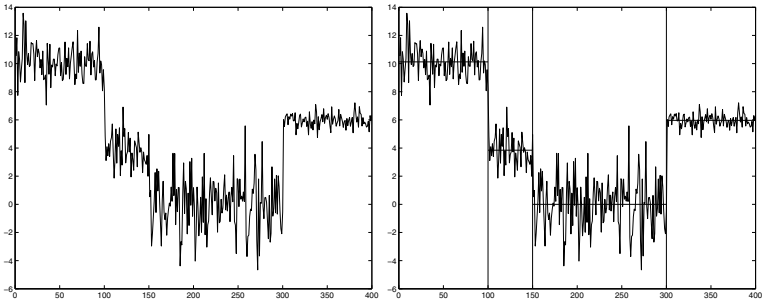

Fig. 8.1 Left: a signal. Right: the result of segmenting into $k = 4$ segments. The vertical bars show the segment boundaries ($b(1) = 0$, $b(2) = 100$, $b(3) = 150$, $b(4) = 300$). The horizontal lines show the representatives of the segments which in this case are the mean values of the points in each segment: $\mu_1 = 10.12$, $\mu_2 = 3.85$, $\mu_3 = 0.00$, $\mu_4 = 5.97$.

Optimal segmentation. Let us start by giving the optimal algorithm for solving the plain segmentation problem. Let $\mathscr{S}_{n,k}$ denote the set of all k-segmentations of sequences of length n. For some sequence T and error measure E_p, we define the optimal segmentation as

$$S_{opt}(T,k) = \arg\min_{S \in \mathscr{S}_{n,k}} E_p(T,S).$$

We sometimes write $E(S)$ instead of $E_p(T,S)$ as the dependence of the data T is obvious, and p is often clear from the context. Finding the optimal segmentation S_{opt} for a given T of length n and for given k and E_p can be done in time $O(n^2k)$ [2] by a standard dynamic programming (DP) algorithm. The dynamic programming algorithm proceeds in an incremental fashion, using a table A of size $n \times k$, where the entry $A[i,\ell]$ denotes the error of segmenting the sequence $T[1,i]$ using at most ℓ segments. Here $T[1,i]$ denotes the subsequence of T that contains all points between 1 and i: $T[1,i] = \langle t_1, \ldots, t_i \rangle$. Let $E(S_{opt}(T[j,i],1))$ be the minimum error that can be obtained for the subsequence $T[j,i]$ when representing it as one segment. The computation of the entries of table A is based on the equation ([17, 25] etc.)

$$A[i,\ell] = \min_{1 \le j \le i} (A[j-1,\ell-1] + E(S_{opt}(T[j,i],1)))$$

The table is first initialized with entries $A[i,1] = E(S_{opt}(T[1,i],1))$ for all $i = 1,\ldots,n$. The best k-segmentation is found by storing the choices for j at each step in the recursion in another table B, and by reconstructing the sequence of choices starting from $B[n,k]$. Note that the table B also gives the optimal k-segmentations for all $k' < k$, which can be reconstructed by starting from $B[n,k']$.

The resulting segmentation is optimal in the sense that the representation error (8.1) between the original sequence and a piecewise constant representation with k segments is minimized. A piecewise constant representation is one in which the representatives of the segments are constants (in practice, means or medians of the data points in the segment). In fact, the above algorithm can be used to compute optimal k-segmentations with piecewise polynomial models. In a piecewise polynomial representation, a polynomial of a given degree is fitted to each segment separately.

We note that the dynamic programming algorithm can also be used in the case of weighted sequences in which each point is associated with a weight. Then the representatives are defined to be the weighted representatives.

Related work. There is a large body of work in segmentation algorithms for sequential data. Terzi and Tsaparas [43] have found three main approaches to segmentation in the literature:

1. Heuristics for solving a segmentation problem faster than the optimal dynamic programming algorithm, with promising experimental results but no theoretical guarantees about the quality of the result.
2. Approximation algorithms with provable error bounds, that is, theoretical upper bounds for the error compared to the optimal error.
3. New variations of the basic segmentation problem, imposing some modifications or constraints on the structure of the representatives of the segments.

The majority of the papers published on segmentation fall into Category 1, fast heuristics. The most popular of these algorithms are the top-down and bottom-up greedy algorithms. The *top-down greedy algorithm* is used in e.g. [4, 11, 29, 40] and briefly discussed in [43]: The algorithm starts with an unsegmented sequence and introduces a new boundary at every greedy step. That is, in the i-th step the algorithm introduces the i-th segment boundary by splitting one of the existing i segments into two. The new boundary is selected in such a way that it minimizes the overall error. No changes are made to the existing $i-1$ boundary points. The splitting is repeated until the number of segments reaches k. The running time of the algorithm is $O(nk)$.

In the *bottom-up greedy algorithm*, each point initially forms a segment of its own. At each step, two consecutive segments that cause the smallest increase in the error are merged. The algorithm stops when k segments are formed. The time complexity of the bottom-up algorithm is $O(n\log n)$. The algorithm performs well in terms of error and it has been used widely in time-series segmentation [18, 35, 43].

Yet another fast heuristics is presented by Himberg et al [23]: two slightly different randomized algorithms that start with a random k-segmentation. At each step they pick one segment boundary (randomly or in some order) and search for the best position to put it back. This is repeated until the representation error converges.

Both algorithms run in time $O(In)$ where I is the number of iterations needed until convergence.

For the algorithms in Category 1 there is empirical evidence that their performance is often very good in practice. However, there are no guarantees of their worst-case error ratio. This is in contrast to algorithms in Category 2 for which error bounds can be shown. In Category 2, an interesting contribution is that of Guha et al [16]: a fast segmentation algorithm with provable error bounds. Terzi and Tsaparas [43] have a similar motivation but different point of view, and we will take a closer look at this in Section 8.2. Category 3, variations of the basic segmentation problem, is studied extensively, and several approaches will be described in the following sections.

Online versions of the segmentation problem have also been studied ([26, 36] and others). In this setting, new observations arrive continuously in a streaming manner, making the data a streaming time series.

An interesting restriction on the segmentation problem in the online case is to require more accuracy in the representation of new observations, as opposed to those which arrived further away in the past. This representation is called *amnesic* as the fidelity of approximation decreases with time, and we are willing to answer queries about the recent past with greater precision. Palpanas et al [36] use a piecewise linear segmentation method to this end. The error of the approximation is always kept under some user-specified, time-dependent threshold.

An abstract framework for the study of streaming time series is recently given by Gandhi et al [12]. They present theoretical results for the space-quality approximation bounds. Both data streams, amnesic approximations and out-of-order streams are discussed in their paper. The case of out-of-order time series will also be discussed in Section 8.6 but only in the case of non-streaming, offline segmentation.

A task related to segmentation is time series approximation or summarization. Similarly to the task of segmentation, the goal here is again to simplify the representation of a sequence. Classical signal processing approaches to time series approximation include Discrete Fourier Transform, Discrete Cosine Transform and Discrete Wavelet Transform; common to these tree methods is that a segment-wise presentation is not sought but the characteristics of the sequence are represented using an existing "dictionary" of finer and coarser building blocks. Instead, methods such as Piecewise Aggregate Approximation [46], Adaptive Piecewise Constant Approximation [8], Piecewise Linear Approximation [7, 28] and Piecewise Quadratic Approximation [22] etc. are segmentation methods, and the representatives of the segments are simple functions of the data points in the segment. An interesting comparison on all of these methods is given in Palpanas et al. [36], by measuring their reconstruction accuracy on several real world data sets. A perhaps surprising result was that there was little difference between all the approaches; similar results have been reported elsewhere, too [8, 27, 45]. The take-home message in this respect is that we should not choose the representation method based on approximation fidelity but rather on other features [36]. This is a guiding principle behind the methods described in this chapter, too.

An alternative approach to analyzing sequential data is a Hidden Markov Model (HMM): the observed data is generated by an unknown process that takes several (unobserved) states, and different states output different observations. Churchill [9] was among the first to apply HMMs to sequence segmentation.

Organization. In this chapter, we describe various approaches to segmentation, ranging from efficient algorithms to added constraints and beyond. We start with an efficient approximation algorithm with provable error bounds in Section 8.2. In Sections 8.3 to 8.5 we discuss three different constraints to make the segmentation result more tractable. Sections 8.6 and 8.7 discuss interesting variations in the basic problem setting. Sections 8.8 to 8.10 touch upon other topics related to segmentation, such as determining the goodness of an output of a segmentation algorithm, model selection issues, and bursty event sequences. Finally, Section 17.7 gives a brief conclusion.

8.2 Efficient Algorithms for Segmentation

In the general case, an optimal segmentation for a sequence can be found using dynamic programming [2] in $O(n^2k)$ time, where n is the length of the sequence and k is the number of segments. In practice, sequences are typically very long, and a quadratic algorithm is seldom adequate. Faster heuristics with $O(n \log n)$ or $O(n)$ running time have been presented (see Section 16.1), but there are often no guarantees of the quality of the solutions they produce.

Instead, Terzi and Tsaparas [43] have presented a constant-factor approximation algorithm whose optimal running time is $O(n^{4/3}k^{5/3})$, called the *divide and segment* (DnS) algorithm. The error of the segmentation it produces is provably no more than 3 times that of the optimal segmentation; we thus say that the approximation ratio is 3. The main idea of the algorithm is to divide the problem into smaller subproblems, solve the subproblems optimally and combine their solutions to form the final solution:

- The algorithm starts by partitioning the sequence T into m disjoint subsequences T_i (of equal length, typically).
- Then each T_i is segmented optimally by dynamic programming, yielding a segmentation S_i and a set M_i of k weighted points $M_i = \langle \mu_{i1}, \ldots, \mu_{ik} \rangle$: these are the representatives of the segments (means or medians), weighted by the length of the segment they represent.
- All the mk representatives of the m subsequences are concatenated to form the weighted sequence $T' = \langle \mu_{11}, \ldots, \mu_{1k}, \mu_{21}, \ldots, \mu_{mk} \rangle$, and dynamic programming is then applied on T', outputting the final segmentation.

Assuming that the subsequences are of equal length, the running time of the algorithm depends on m, the number of subsequences. The optimal running time is $2n^{4/3}k^{5/3}$ and it is achieved at $m = (n/k)^{2/3}$ [43].

Terzi and Tsaparas [43] also explore several more efficient variants of the algorithm and quantify the accuracy/efficiency tradeoff. More specifically, they present a recursive application of the DnS algorithm, resulting in a faster algorithm with $O(n \log\log n)$ running time and $O(\log n)$ approximation ratio. All presented algorithms can be made to use a sublinear amount of memory, making them applicable to the case when the data needs to be processed in a streaming fashion (not stored in main memory). Assuming that one has an estimate of n, the size of the sequence, then the algorithm processes the points in batches of size n/m. For each such batch it computes the optimal k-segmentation, and stores the representatives. The space required is $M = n/m + mk$ and this is minimized for $m = \sqrt{n/k}$, resulting in space $M = 2\sqrt{nk}$.

Extensive experiments on both real and synthetic datasets demonstrate that in practice their algorithms perform significantly better than the worst-case theoretical upper bounds, in terms of reconstruction error. Also, the algorithms perform consistently better than fast heuristic algorithms, and the computational costs are comparable [43]. The synthetic datasets are generated by first fixing the dimensionality of the data ($d = 1, 5, 10$) and the segment boudaries ($k = 10$), and then drawing the mean of each segment in each dimension from a Uniform distribution, and adding Gaussian noise whose standard deviation varies from 0.05 to 0.9. The real datasets *balloon, darwin, winding, xrates* and *phone* are from the UCR Time Series Data Mining Archive[1].

8.3 Dimensionality Reduction

Let us then start discussing the various constraints and modifications we add to the problem of segmentation to make the end result more tractable. The first natural constraint that we wish to incorporate in the segmentation arises from dimensionality reduction in multidimensional time series: the multidimensional mean parameters μ_j of the segments should lie within a subspace whose dimensionality is smaller than that of the original space.

Bingham et al. [5] have stated the problem as follows. Given a multidimensional time series, find a small set of latent variables and a segmentation of the series such that the data in each segment can be explained well by some (linear) combination of the latent variables. We call this problem the *basis segmentation problem.*

Our problem formulation allows decomposing the sequences into segments in which the data points are explained by a model unique to the segment, yet the whole sequence can be explained adequately by the vectors of the basis.

Following the notation presented in Section 16.1, our data is a sequence consisting of n observations of d-dimensional vectors. For convenience, we now stack the vectors into a matrix X that contains in its rows the n observations, each of which

[1] `http://www.cs.ucr.edu/~eamonn/TSDMA/`

is d-dimensional, so X is an $n \times d$ matrix. As previously, the n observations will be partitioned into k segments $S = \langle s_1, \ldots, s_k \rangle$.

We will consider basis-vector representations of the data. We denote by $V = \{v_1, \ldots, v_m\}$ the set of m basis vectors $v_\ell \in \mathbb{R}^d$, $\ell = 1, \ldots, m$. The number of basis vectors m is typically significantly smaller than the dimensionality d of the data points. In matrix form, V is an $m \times d$ matrix containing the basis vectors as its rows. Also, for each segment S_j we have a set of coefficients $a_{j\ell} \in \mathbb{R}$ for $\ell = 1, \ldots, m$ that tell us how to represent the data using the basis vectors in V. In matrix notation, $A = (a_{j\ell})$ is a $k \times m$ matrix of coefficients. V and A will be found by Principal Component Analysis (PCA, discussed more in the sequel).

We approximate the sequence with *piecewise constant* linear combinations of the basis vectors, i.e., all observations in segment s_j are represented by a single vector

$$u'_j = \sum_{\ell=1}^{m} a_{j\ell} v_\ell. \tag{8.1}$$

The problem we consider is the following.

Problem 8.1. Denote by $j(i) \in \{1, \ldots, k\}$ the segment to which point i belongs. Given a sequence $T = \langle t_1, \ldots, t_n \rangle$, and integers k and m, find a basis segmentation (S, V, A) that uses k segments and a basis of size m, so that the reconstruction error

$$E(T; S, V, A) = \sum_{i=1}^{n} ||t_i - u'_{j(i)}||^2$$

is minimized. The constant vector $u'_{j(i)}$ for approximating segment S_j is given by Equation (8.1).

To solve the basis segmentation problem, we combine existing methods for sequence segmentation and for dimensionality reduction: (*i*) k-segmentation by dynamic programming, discussed in Section 16.1, and (*ii*) Principal Component Analysis (PCA), one of the most commonly used methods for dimensionality reduction. Given a matrix Z of size $n \times d$ with data points as rows, the goal in PCA is to find a subspace of dimension $r < d$ so that the residual error of the points of Z projected onto the subspace is minimized. The PCA algorithm computes a matrix Y of rank r, and the decomposition $Y = AV$ of Y into the orthogonal basis V of size r, such that

$$Y = \operatorname*{arg\,min}_{\text{rank}(Y') \le r} ||Z - Y'||$$

which holds for all matrix norms induced by L_p vector norms. PCA is typically accomplished by Singular Value Decomposition (SVD) on the data matrix Z. The basis vectors $v_1, \ldots, v_m$ are the right singular vectors of the data matrix.

We suggest three different algorithms for solving the basis segmentation problem, all of which combine k-segmentation and PCA in different ways:

- Seg-PCA: First partition into k segments in the full d-dimensional space, to obtain segments $S = \langle s_1, \ldots s_k, \rangle$ and d-dimensional vectors $u_1, \ldots, u_k$ representing

the points in each segment. Then consider the set $U_S = \{(u_1, |s_1|), \ldots, (u_k, |s_k|)\}$ where each vector u_j is weighted by $|s_j|$, the length of segment s_j. Perform PCA on the set of weighted points U_S, outputting for each segment vector u_j an approximate representation u'_j as in (8.1). Bingham et al [5] show that the Seg-PCA algorithm yields a solution to the basis segmentation problem such that the reconstruction error is at most 5 times the reconstruction error of the optimal solution. Experiments demonstrate that in practice, the approximation ratios are smaller than 5.

- Seg-PCA-DP: First segment into k segments, then find a basis of size m for the segment means, similarly to above. Then refine the segmentation boundaries by using the discovered basis by a second application of dynamic programming. As the first two steps of the algorithm are identical to the Seg-PCA algorithm, and the last step can only improve the cost of the solution, the same approximation ratio of 5 holds also for Seg-PCA-DP.
- PCA-Seg: First do PCA to dimension m on the whole data set. Then obtain the optimal segmentation of the resulting m-dimensional sequence. This gives computational savings, as the segmentation is not performed on a high-dimensional space.

Experiments on synthetic and real datasets show that all three algorithms discover the underlying structure in the data [5]. Prototype implementations are available to the public at `http://www.cs.helsinki.fi/hiit_bru/software/`.

A somewhat related problem setting, restricting the complexity of the representatives of the segments, will be considered in the next section.

8.4 Recurrent Models

Whereas in Section 8.3 we represented the segments as different combinations of a small set of global basis vectors, we now wish to use a small set of models to predict the data values in the segments.

Often in a sequence with segmental structure, similar types of segments occur repeatedly: different models are suitable in different segments. For example, high solar radiation implies clear skies, which in the summer means warm temperatures and in the winter cold ones. As another example, the inheritance mechanism of recombinations in chromosomes mean that a genome sequence can be explained by using a small number of ancestral models in a segment-wise fashion. In these examples, the model used to explain the target variable changes relatively seldom, and has a strong effect on the sequence. Moreover, the same models are used repeatedly in different segments: the summer model works in any summer segment, and the same ancestor contributes different segments of the genome.

In an earlier contribution by Gionis and Mannila [13], the idea for searching for recurrent models was used in the context of finding piecewise *constant* approximations in the so called (k,h) segmentation problem. In their paper it was assumed that the sequence can be segmented into k pieces, of which only h are distinct. In other

words, there are h hidden sources such that the sequence can be written as a concatenation of $k > h$ pieces, each of which stems from one of the h sources. This problem was shown to be NP-hard, and approximate algorithms were given [13]. The "Segments2Levels" algorithm runs in time $O(n^2(k+h))$ and gives a 3-approximation for $p = 1,2$ for dimension 1. For higher dimensions, the approximation guarantees are $3+\varepsilon$ for $p = 1$ and $\alpha+2$ for $p = 2$ where α is the best approximation factor for the k-means problem. The "ClusterSegments" algorithm yields approximation ratios 5 and $\sqrt{5}$ for $p = 1$ and 2, respectively; the running time is again $O(n^2(k+h))$. The "Iterative" algorithm is inspired by the EM algorithm and provides at least as good approximations as the two previous ones. Its running time is $O(In^2(k+h))$ where I is the number of iterations.

The goal in (k,h) segmentation is similar to, although the technique is different from, using a Hidden Markov Model (HMM) to sequence segmentation, originally proposed by Churchill [9].

In a new contribution by Hyvönen et al [25], this approach was used to arbitrary *predictive models*, which requires considerably different techniques than those in (k,h) segmentation [13]. To find such recurrent predictive models, one must be able to do segmentation based not on the target to be predicted itself, but on *which model* can be used to predict the target variable, given the input measurements. The application areas discussed above, the temperature prediction task and ancestral models in a genome sequence, call for such a recurrent predictive model.

Given a model class $\mathscr{M}$, the task is to search for a small set of h models from $\mathscr{M}$ and a segmentation of the sequence T into k segments such that the behavior of each segment is explained well by a single model. It is assumed that $h < k$, i.e., the same model will be used for multiple segments. More precisely, the data $D = (T,y)$ consist of a multidimensional sequence $T = \langle t_1,\ldots,t_n\rangle$, $t_i \in \mathbb{R}^d$ and corresponding scalar outcome values $y = \langle y_1,\ldots,y_n\rangle$, $y_i \in \mathbb{R}$. We denote a subsequence of the input sequence between the i-th and j-th data point as $D[i,j]$. A model M is a function $M : \mathbb{R}^d \to \mathbb{R}$ that belongs to a model class $\mathscr{M}$. Given a subsequence $D[i,j]$ and a model $M \in \mathscr{M}$ the prediction error of M on $D[i,j]$ is defined as

$$E(D[i,j],M) = \sum_{\ell=i}^{j} ||M(t_\ell) - y_\ell||^2. \tag{8.2}$$

For many commonly used model classes $\mathscr{M}$ one can compute in polynomial time the model $M^* \in \mathscr{M}$ that minimizes the error in (8.2). For example, for the class of linear models, the optimal model can be found using least squares. For probabilistic models one can estimate the maximum likelihood model. For some model classes such as decision trees, finding the optimal model is computationally difficult, but efficient heuristics exist. It is thus assumed that one can always find a good model for a given subsequence.

One should note that the task of predicting a given output value y_i for a multidimensional observation t_i using a model $M \in \mathscr{M}$ is now different from the basic segmentation task in which the "output" or the representative of the segment is not

given beforehand. In the latter, the task is to *approximate* the sequence rather than *predict*.

Now, let us first define the "easy" problem:

Problem 8.2. Given an input sequence D, a model class $\mathscr{M}$, and a number k, partition D into k segments $D_1,\ldots,D_k$ and find corresponding models $M_1,\ldots,M_k \in \mathscr{M}$ such that the overall prediction error $\sum_{j=1}^{k} E(D_j,M_j)$ is minimized.

The above problem allows for different models in each of the k segments. Our interest, however, is in the *recurrent predictive modeling* problem which is a more demanding task in that it only allows for a small number of h distinct models, $h < k$. Thus, some of the models have to be used in more than one segment. More formally, we define the following problem.

Problem 8.3. Consider a sequence D, a model class $\mathscr{M}$, and numbers k and h, $h < k$. The task is to find a k-segmentation of D into k segments $D_1,\ldots,D_k$, h models $M_1,\ldots,M_h \in \mathscr{M}$, and an assignment of each segment j to a model $M_{m(j)}$, $m(j) \in \{1,\ldots,h\}$ so that the prediction error $\sum_{j=1}^{k} E(D_j,M_{m(j)})$ is minimized.

For any but the simplest model class the problem of finding the best h models is an NP-hard task, so one has to resort to approximate techniques.

Given a sequence D and a class of models $\mathscr{M}$, dynamic programming [2] is first used to find a good segmentation of the sequence into k segments. Thus each segment will have its unique predictive model. The method for finding the model describing a single segment depends, of course, on the model class $\mathscr{M}$. After that, from the k models found in the segmentation step, one selects a smaller number of h models that can be used to model well the whole sequence. In case parameters k and h can be fixed in advance, selecting a smaller number of models is treated as a clustering problem, and solved using the k-median [31] or k-means algorithm. Finally, an *iterative improvement algorithm* that is a variant of the EM algorithm is applied: iteratively fit the current models more accurately in the existing segments, and then find a new segmentation given the improved models. The iteration continues until the error of the solution does not improve any more.

In the more general case, the parameters k and h are not given, but need to be determined from the data. This model selection problem is addressed using the Bayesian Information Criterion (BIC). Selecting a smaller number of models is again a clustering problem, and using the facility location approach [24] one only has to iterate over the number of segments k: For each value of k, the corresponding value of h that minimizes the BIC score is automatically selected by the facility location algorithm.

In [25] the method of recurrent models was applied to two sets of real data, meteorological measurements, and haplotypes in the human genome. The experimental results showed that the method produces intuitive results. For example, in a temperature prediction task, the meteorological time series consisting of 4 consecutive winters and 3 summers was first found to contain $k = 7$ segments — not perhaps surprisingly — and these 7 segments were found to be generated by $h = 2$ recurring models, a winter model and a summer model.

8.5 Unimodal Segmentation

We will discuss another restriction of the basic segmentation problem. In *unimodal* segmentation, the representatives of the segments (for example, means or medians) are required to follow a unimodal curve: the curve that is formed by all representatives of the segments has to change curvature only once. That is, the representatives first increase until a certain point and then decrease during the rest of the sequence, or the other way round. A special case is a *monotonic* curve. Examples of unimodal sequences include (i) the size of a population of a species over time, as the species first appears, then peaks in density and then dies out or (ii) daily volumes of network traffic [18].

In contrast to other segmentation methods discussed in this chapter, the sequence now takes scalar values instead of multidimensional values. Haiminen and Gionis [18] show how this problem can be solved by combining the classic "pool adjacent violators" (PAV) algorithm [1] and the basic dynamic programming algorithm [2] (see Section 16.1). The time complexity of their algorithm is $O(n^2k)$ which is the same as in the unrestricted k-segmentation using dynamic programming.

Haiminen and Gionis [18] also describe a more efficient greedy-merging heuristic that is experimentally shown to give solutions very close to the optimal, and whose time complexity is $O(n \log n)$: the expensive dynamic programming step is replaced with a greedy merging process that starts with m segments and iteratively merges the two consecutive segments that yield the least error, until reaching k segments.

The authors in [18] also give two tests for unimodality of a sequence. The first approach compares the error of an optimal unimodal k-segmentation to the error of an optimal unrestricted k-segmentation. If the sequence exhibits unimodal behaviour, then the error of its unimodal segmentation does not differ very much from the error of its unrestricted segmentation — in other words, requiring for unimodality did not hurt. Instead, if the sequence is not unimodal in nature, then forcing the segments to follow a unimodal curve will increase the representation error. The authors compute the ratio between the error of unrestricted k-segmentation and the error of unimodal segmentation and find a data-dependent threshold value that helps to differentiate between unimodal and non-unimodal sequences.

The second approach for testing for unimodality is to randomly permute the unimodal segments in the data, and to see if the error of unimodal k-segmentation on the permuted sequence is comparable to the error on the original sequence — the random permutation will destroy the unimodal structure of the sequence, if such exists. If the original sequence was indeed unimodal, then the error of the permuted sequences should be larger in a statistically significant way.

After discussing three different restrictions on the representatives of the segments in Sections 8.3, 8.4 and 8.5, we then turn to other modifications of the basic problem of sequence segmentation in the next section.

8.6 Rearranging the Input Data Points

The majority of related work on segmentation primarily focuses on finding a segmentation S of a sequence T taking for granted the order of the points in T. However, more often than not, the order of the data points of a sequence is not clear-cut but some data points actually appear simultaneously or their order is for some other reason observed only approximately correctly. In such a case it might be beneficial to allow for a slight rearrangement of the data points, in order to achieve a better segmentation. This was studied by Gionis and Terzi [15]: in addition to partitioning the sequence they also apply a limited amount of reordering, so that the overall representation error is minimized.

The focus now is to find a rearrangement of the points in T such that the segmentation error of the reordered sequence is minimized. The operations used to rearrange an input sequence consist of bubble-sort swaps and moves (single-element transpositions). The task is to find a sequence of operations O minimizing the reconstruction error on the reordered input sequence T_O:

$$O = \arg\min_{O'} E(S_{opt}(T_{O'}, k))$$

where $S_{opt}(T,k)$ is the optimal segmentation of T into k segments, and there is an upper limit on the number of operations: $|O| \leq C$ for some integer constant C.

The problem of segmentation with rearrangements is shown to be NP-hard to solve or even approximate. However, efficient algorithms are given in [15], combining ideas from linear programming, dynamic programming and outlier-detection algorithms in sequences. The algorithms consist of two steps. In the first step, and optimal segmentation S of the input sequence T into k segments is found. In the second step, a good set of rearrangements is found, such that the total segmentation error or the rearranged sequence is minimized. The latter step, the rearrangement, can be done in several ways, and the authors discuss the task in detail. In one possible formulation, the rearrangement task is a generalization of the well known NP-hard Knapsack problem for which a pseudopolynomial-time algorithm is admittable [44]. For the special case of bubble-sort swaps only, or moves only, a polynomial-time algorithm for the rearrangement is obtained. The authors also present a greedy heuristic with time complexity $O(Ink)$ where I is the number of iterations of the greedy algorithm in [15].

The problem formulation has applications in segmenting data collected from a sensor network where some of the sensors might be slightly out of sync, or in the analysis of newsfeed data where news reports on a few different topics are arriving in an interleaved manner. The authors show experiments on both synthetic data sets and on several real datasets from the UCR time series archive[2].

[2] `http://www.cs.ucr.edu/~eamonn/TSDMA`

8.7 Aggregate Segmentation

Whereas in Section 8.6 we refined the input data, we now turn our attention to the output of one or more segmentation algorithms.

A sequence can often be segmented in several different ways, depending on the choice of the segmentation algorithm, its error function, and in some cases, its initialization. The multitude of segmentation algorithms and error functions naturally raises the question: given a specific dataset, what is the segmentation that best captures the underlying structure of the data?

Thus a natural question is, given a number of possibly contradicting segmentations, how to produce a single *aggregate segmentation* that combines the features of the input segmentations.

Mielikäinen et al [33] adopt a democratic approach that assumes that all segmentations found by different algorithms are correct, each one in its own way. That is, each one of them reveals just one aspect of the underlying true segmentation. Therefore, they aggregate the information hidden in the segmentations by constructing a consensus output that reconciles optimally the differences among the given inputs.

Their approach results in a proof that for a natural formalization of this task, there is an optimal polynomial-time algorithm, and a faster heuristic that has good practical properties. The algorithms were demonstrated in two applications: clustering the behavior of mobile-phone users, and summarizing different segmentations of genomic sequences.

More formally, the input is a set of m different segmentations $S_1, \ldots, S_m$, and the objective is to produce a single segmentation $\hat{S}$ that agrees as much as possible with the input segmentations. The number of segments in $\hat{S}$ is learned during the process. In the discrete case a *disagreement* between two segmentations S and S' is defined as a pair of points (x, y) placed in the same segment by S but in different segments by S', or vice versa. Denoting the total number of disagreements between the sequences S and S' by $D_A(S, S')$, the formal objective is to minimize

$$\sum_{j=1}^{m} D_A(S_j, \hat{S}),$$

the grand total number of disagreements between the input segmentations S_j and the output segmentation $\hat{S}$. In the continuous case, disagreements are defined similarly for intervals instead of discrete points. The polynomial-time exact algorithm is based on the technique of dynamic programming [2], and the approximation algorithm on a greedy heuristic.

Segmentation aggregation can prove useful in many scenarios. We list some of them below, suggested by Mielikäinen et al [33].

In the analysis of genomic sequences of a population one often assumes that the sequences can be segmented into blocks such that in each block, most of the haplotypes fall into a small number of classes. Different segmentation algorithms have successfully been applied to this task, outputting slightly or completely differ-

ent block structures; aggregating these block structures hopefully sheds light on the underlying truth.

Segmentation aggregation adds to the robustness of segmentation results: most segmentation algorithms are sensitive to erroneous or noisy data, and thus combining their results diminishes the effect of missing or faulty data.

Segmentation aggregation also gives a natural way to cluster segmentations: the representative of a cluster of segmentations is then the aggregate of the cluster. Furthermore, the disagreement distance is now a metric, allowing for various distance-based data mining techniques, together with approximation guarantees for many of them.

Other scenarios where segmentation aggregation can prove useful include segmentation of multidimensional categorical data and segmentation of multidimensional data having both nominal and numerical dimensions; summarization of event sequences; and privacy-preserving segmentations [33].

In this section we assessed and improved the quality of the output of the segmentation by aggregating the output of several segmentation methods. A related point of view is to assess the quality of the segmentations by measuring their statistical significance — this nontrivial task will be considered in Section 8.8.

8.8 Evaluating the Quality of a Segmentation: Randomization

An important question is how to evaluate and compare the quality of segmentations obtained by different techniques and alternative biological features. Haiminen et al [20] apply *randomization* techniques to this end.

Consider a segmentation algorithm that given as input a sequence T outputs a segmentation P. Assume that we a priori know a groundtruth segmentation S^* of T. Then, we can say that segmentation P is good if P is similar to S^*. In more exact terms, P is a good segmentation if the entropy of P given S^*, $H(P \mid S^*)$, and the entropy of S^* given P, $H(S^* \mid P)$, are small. However, a natural question is, how small is small enough?

Before we proceed, let us give some more details on the notation. Consider a segmentation P consisting of k segments $P = \langle p_1, \ldots, p_k \rangle$. If we randomly pick a point t on the sequence, then the probability $t \in p_i$ is $Pr(p_i) = |p_i|/n$ where n is the length of the sequence. The *entropy* of a segmentation P is now

$$H(P) = -\sum_{i=k}^{k} Pr(p_i) \log Pr(p_i).$$

The maximum value that the entropy of a segmentation can have is $\log n$, and this value is achived when all segments are of equal length and thus the probabilities of a random point belonging to any of the segments are equal.

Consider now a pair of segmentations P and Q of sequence S. Assume that P and Q have k_p and k_q segments, respectively: $P = \langle p_1, \ldots, p_{k_p} \rangle$ and $Q = \langle q_1, \ldots, q_{k_q} \rangle$.

The *conditional entropy* [10] of P given Q is defined as

$$\begin{aligned} H(P \mid Q) &= \sum_{j=1}^{k_q} Pr(q_j) H(P \mid q_j) \\ &= -\sum_{j=1}^{k_q} Pr(q_j) \sum_{i=1}^{k_p} Pr(p_i \mid q_j) \log Pr(p_i \mid q_j) \\ &= -\sum_{j=1}^{k_q} \sum_{i=1}^{k_p} Pr(p_i, q_j) \log Pr(p_i \mid q_j) \end{aligned}$$

That is, the conditional entropy of segmentation P given segmentation Q is the expected amount of information we need to identify the segment of P into which a point belongs, given that we know the segment of this point in Q.

Haiminen et al [20] give an efficient algorithm for computing the conditional entropies between two segmentations: Denote by U the union of two segmentations P and Q, that is, the segmentation defined by the segment boundaries that appear in P or in Q. The conditional entropy of P given Q can be computed as $H(P \mid Q) = H(U) - H(Q)$. The algorithm runs in time $O(k_p + k_q)$.

Let us now return to our original problem setting: Assuming we know a groundtruth segmentation S^* of T, then P is a good segmentation if $H(P \mid S^*)$ and $H(S^* \mid P)$ are small. But how small is small enough? Or, is there a threshold in the values of the conditional entropies below which we can characterize the segmentation P as being correct or interesting? Finally, can we set this threshold universally for all segmentations?

The generic methodology of *randomization techniques* (see [14], [34] among others) that are devised to answer these questions is the following. Given a segmentation P and a ground-truth segmentation S^* of the same sequence, we first compute $H(P \mid S^*)$ and $H(S^* \mid P)$. We compare the values of these conditional entropies with the values of the conditional entropies $H(R \mid S^*)$ and $H(S^* \mid R)$ for a random segmentation R. We conclude that P is similar to S^*, and thus interesting, if the values of both $H(P \mid S^*)$ and $H(S^* \mid P)$ are smaller than $H(R \mid S^*)$ and $H(S^* \mid R)$, respectively, for a large majority of random segmentations R. Typically, 10 000 or 100 000 random segmentations are drawn, and if $H(P \mid S^*) < H(R \mid S^*)$ in all but a couple of cases, then P is deemed interesting. The percentage of cases violating $H(P \mid S^*) < H(R \mid S^*)$ can be interpreted as a p value, and a small value denotes statistical significance.

The example applications in [20] include isochore detection and the discovery of coding-noncoding structure. The authors obtain segmentations of relevant sequences by applying different techniques, and use alternative features to segment on. They show that some of the obtained segmentations are very similar to the underlying true segmentations, and this similarity is statistically significant. For some other segmentations, they show that equally good results are likely to appear by chance.

8.9 Model Selection by BIC and Cross-validation

One of the key questions in segmentation is choosing the number of segments to use. Important features of the sequence may be lost when representing it with too few segments, while using too many segments yields uninformative and overly complex segmentations.

Choosing the number of segments is essentially a model selection task. Haiminen and Mannila [19] present extensive experimental studies on two standard model selection techniques, namely Bayesian Information Criterion (BIC) and cross-validation (CV).

Bayesian Information Criterion (BIC) seeks a balance between model complexity and the accuracy of the model by including a penalty term for the number of parameters. BIC is defined as [39] $BIC = -2\ln L + K\ln N + C$ where L is the maximized likelihood of the model with K free parameters, N is the sample size and C is a small constant that is often omitted. The model with the smallest BIC is optimal in terms of complexity.

Cross-validation is an intuitive iterative method for model selection: A subset of the data is used to train the model. The goodness of fit on the remaining data, also called test data, is then evaluated. This is repeated for a number of times, in each of which the data are randomly split into a training set and test set. In an outer loop, the complexity of the model (here, the number of segments) is varied. When the model complexity is unnecessarily high, the model overfits the training data and fails to represent the test data. Alternatively, if the model complexity is too low, the test data cannot be faithfully represented either. CV is a very general method in that no assumptions regarding the data are made, and any cost function can be used. The use of CV has been discussed by e.g. Stone [42] and Smyth [41].

The results in [19] show that these methods often find the correct number of piecewise constant segments on generated real-valued, binary, and categorical sequences. Also segments having the same means but different variances can be identified. Furthermore, they demonstrate the effect of linear trends and outliers on the results; both phenomena are frequent in real data.

The results indicate that BIC is fairly sensitive to outliers, and that CV in general is more robust. Intuitive segmentation results are given for real DNA sequences with respect to changes in their codon, G+C, and bigram frequencies, as well as copy-number variation from CGH data.

8.10 Bursty Sequences

In the earlier sections, we have assumed some specific constraints on the segments, making the problem more applicable to the human eye. Haiminen et al [21] have also studied constraining the nature of the sequence itself, outside the task of segmenting the sequence. An intuitive subset of sequences is one in which *bursts of activity* occur in time, and a natural question then is, how to formally define and measure

this. The problem setting applies to *event sequences*: Given a set of possible event types, an event sequence is a sequence of pairs (r,t), where r is an event type and t is the occurrence location, or time, of the event. Moreover, a *bursty event sequence* is one in which "bursts" of activity occur in time: different types of events often occur close together.

Bursty sequences arise, e.g., when studying potential transcription factor binding sites (events) of certain transcription factors (event types) in a DNA sequence. These events tend to occur in bursts. Tendencies for co-occurrence of binding sites of two or more transcription factors are interesting, as they may imply a co-operative role between the transcription factors in regulatory processes.

Haiminen et al [21] measure the co-occurrence of event types r and r' either by (i) dividing the sequence into non-overlapping windows of a fixed length w and counting the number of windows that contain at least one event of type r and at least one event of type r', or by (ii) counting the number of events of type r that are followed by at least one event of type r' within distance w, or by (iii) counting the number of events of type r that are followed or preceded by at least one event of type r' within distance w.

In order to determine the significance of a co-occurrence score, we need a *null model* to estimate the distribution of the score values and then decide the significance of an individual value. Haiminen et al [21] define three such null models that apply to any co-occurrence score, extending previous work on null models. These models range from very simple ones to more complex models that take the burstiness of sequences into account. The authors evaluate the models and techniques on synthetic event sequences, and on real data consisting of potential transcription factor binding sites.

8.11 Conclusion

In this chapter, we have discussed several variants of the problem of sequence segmentation. An optimal segmentation method, applicable when no specific restrictions are assumed, is segmentation using dynamic programming [2]. However, this is computationally burdensome for very long sequences. Also, it is often the case that by adding some constraints on the output segmentation, or by making small modifications to the problem, the output of the segmentation is more interpretable for the human eye. This chapter is a survey of different approaches for segmentation suggested by researchers at Helsinki Institute for Information Technology during the years 2004 to 2008.

In Section 8.2 we described an efficient segmentation method, with a proven quality of the solution it provides when representing the original data [43]. We then discussed three constraints on the problem setting of segmentation, to make the end result more tractable, in Sections 8.3 to 8.5. Using these constraints we wish to restrict the values that the representatives (that is, means or medians, typically) of the segments can assume. First, the representatives of multidimensional segments can

be presented as different combinations of a small set of basis vectors [5]. Secondly, a small set of models can be used to predict the data values in the segments [25]. Thirdly, one can require the representatives of the sequences to follow a unimodal or monotonic curve [18].

Then, instead of constraints on the representatives per se, we discussed small modifications to the basic problem setting in Sections 8.6 and 8.7: By allowing small reorderings of the data points in a sequence, we can decrease the reconstruction error of the segmentation, and in some application areas these reorderings are very natural [15]. In some cases there is a need to choose between several different outputs of segmentation algorithms, and a way to overcome this is to combine the outputs into one aggregate segmentation [33]. A very important and nontrivial task is to characterize the quality of a segmentation in statistical terms, and randomization provides an answer here [20] (Section 8.8). Choosing the number of segments is a question of model selection, and experimental results were discussed in Section 8.9 [19]. Finally in Section 8.10, instead of constraining the representatives of the sequences, we constrained the nature of the sequence itself, when determining when an event sequence is bursty or not [21].

Acknowledgements The author would like to thank Aristides Gionis, Niina Haiminen, Heli Hiisilä, Saara Hyvönen, Taneli Mielikäinen, Evimaria Terzi, Panayiotis Tsaparas and Heikki Mannila for their work in the papers discussed in this survey. The comments given by the anonymous reviewers have greatly helped to improve the manuscript.

References

1. Miriam Ayer, H. D. Brunk, G. M. Ewing, W. T. Reid, and Edward Silverman. An empirical distribution function for sampling with incomplete information. *Annals of Mathematical Statistics*, 26(4):641–647, 1955.
2. Richard Bellman. On the approximation of curves by line segments using dynamic programming. *Communications of the ACM*, 4(6), 1961.
3. K.D. Bennett. Determination of the number of zones in a biostratigraphical sequence. *New Phytologist*, 132(1):155–170, 1996.
4. Pedro Bernaola-Galván, Ramón Román-Roldán, and José L. Oliver. Compositional segmentation and long-range fractal correlations in dna sequences. *Phys. Rev. E*, 53(5):5181–5189, 1996.
5. Ella Bingham, Aristides Gionis, Niina Haiminen, Heli Hiisilä, Heikki Mannila, and Evimaria Terzi. Segmentation and dimensionality reduction. In *2006 SIAM Conference on Data Mining*, pages 372–383, 2006.
6. Harmen J. Bussemaker, Hao Li, and Eric D. Siggia. Regulatory element detection using a probabilistic segmentation model. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 67–74, 2000.
7. A. Cantoni. Optimal curve fitting with piecewise linear functions. *IEEE Transactions on Computers*, C-20(1):59–67, 1971.
8. K. Chakrabarti, E. Keogh, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems*, 27(2):188–228, 2002.
9. G.A. Churchill. Stochastic models for heterogenous dna sequences. *Bulletin of Mathematical Biology*, 51(1):79–94, 1989.

10. Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley, 1991.
11. David Douglas and Thomas Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, 1973.
12. Sorabh Gandhi, Luca Foschini, and Subhash Suri. Space-efficient online approximation of time series data: Streams, amnesia, and out-of-order. In *Proceedings of the 26th IEEE International Conference on Data Engineering (ICDE)*, 2010.
13. Aristides Gionis and Heikki Mannila. Finding recurrent sources in sequences. In *Proceedings of the Sventh Annual International Conference on Computational Biology (RECOMB 2003)*, 2003.
14. Aristides Gionis, Heikki Mannila, Taneli Mielikäinen, and Panayiotis Tsaparas. Assessing data mining results via swap randomization. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(3), 2007. Article No. 14.
15. Aristides Gionis and Evimaria Terzi. Segmentations with rearrangements. In *SIAM Data Mining Conference (SDM) 2007*, 2007.
16. S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Symposium on the Theory of Computing (STOC)*, pages 471–475, 2001.
17. Niina Haiminen. *Mining sequential data — in search of segmental structure*. PhD Thesis, Department of Computer Science, University of Helsinki, March 2008.
18. Niina Haiminen and Aristides Gionis. Unimodal segmentation of sequences. In *ICDM '04: Proceedings of the Fourth IEEE International Conference on Data Mining*, pages 106–113, 2004.
19. Niina Haiminen and Heikki Mannila. Evaluation of BIC and cross validation for model selection on sequence segmentations. *International Journal of Data Mining and Bioinformatics*. In press.
20. Niina Haiminen, Heikki Mannila, and Evimaria Terzi. Comparing segmentations by applying randomization techniques. *BMC Bioinformatics*, 8(171), 23 May 2007.
21. Niina Haiminen, Heikki Mannila, and Evimaria Terzi. Determining significance of pairwise co-occurrences of events in bursty sequences. *BMC Bioinformatics*, 9:336, 2008.
22. Trevor Hastie, R. Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2001.
23. J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmäki, and H. T.T. Toivonen. Time series segmentation for context recognition in mobile devices. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 203–210, 2001.
24. Dorit S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22(1):148–162, 1982.
25. Saara Hyvönen, Aristides Gionis, and Heikki Mannila. Recurrent predictive models for sequence segmentation. In *The 7th International Symposium on Intelligent Data Analysis*, Lecture Notes in Computer Science. Springer, 2007.
26. Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 289–296, 2001.
27. Eamonn Keogh and S. Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *Proceedings of the ACM SIGKDD '02*, pages 102–111, July 2002.
28. Eamonn Keogh and Michael J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Proceedings of the ACM SIGKDD '98*, pages 239–243, August 1998.
29. Victor Lavrenko, Matt Schmill, Dawn Lawrie, Paul Ogilvie, David Jensen, and James Allan. Mining of concurrent text and time series. In *In proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining Workshop on Text Mining*, pages 37–44, 2000.
30. W. Li. DNA segmentation as a model selection process. In *Proceedings of the Fifth Annual International Conference on Computational Biology (RECOMB 2001)*, pages 204 – 210, 2001.

31. Jyh-Han Lin and Jeffrey Scott Vitter. ε-approximations with minimum packing constraint violation. In *Proc. ACM Symposium on Theory of Computing (STOC'92)*, pages 771–781, 1992.
32. Jun S. Liu and Charles E. Lawrence. Bayesian inference on biopolymer models. *Bioinformatics*, 15(1):38–52, 1999.
33. Taneli Mielikäinen, Evimaria Terzi, and Panayiotis Tsaparas. Aggregating time partitions. In *The Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006)*, pages 347–356, 2006.
34. Markus Ojala, Niko Vuokko, Aleksi Kallio, Niina Haiminen, and Heikki Mannila. Randomization of real-valued matrices for assessing the significance of data mining results. In *Proc. SIAM Data Mining Conference (SDM'08)*, pages 494–505, 2008.
35. T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *ICDE 2004: Proceedings of the 20th International Conference on Data Engineering*, pages 338–349, 2004.
36. Themis Palpanas, Michail Vlachos, Eamonn Keogh, and Dimitrios Gunopulos. Streaming time series summarization using user-defined amnesic functions. *IEEE Transactions on Knowledge and Data Engineering*, 20(7):992–1006, 2008.
37. V.E. Ramensky, V.J. Makeev, M.A. Roytberg, and V.G. Tumanyan. DNA segmentation through the Bayesian approach. *Journal of Computational Biology*, 7(1-2):215–231, 2000.
38. Marko Salmenkivi, Juha Kere, and Heikki Mannila. Genome segmentation using piecewise constant intensity models and reversible jump MCMC. *Bioinformatics (European Conference on Computational Biology)*, 18(2):211–218, 2002.
39. G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.
40. Hagit Shatkay and Stanley B. Zdonik. Approximate queries and representations for large data sequences. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pages 536–545, 1996.
41. P. Smyth. Model selection for probabilistic clustering using cross-validated likelihood. *Statistics and Computing*, 9:63–72, 2000.
42. M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B*, 36(2):111–147, 1974.
43. Evimaria Terzi and Panayiotis Tsaparas. Efficient algorithms for sequence segmentation. In *2006 SIAM Conference on Data Mining*, pages 314–325, 2006.
44. V. Vazirani. *Approximation algorithms*. Springer, 2003.
45. Y.-L. Wu, D. Agrawal, and A. El Abbadi. A comparison of DFT and DWT based similarity search in time series databases. In *Proceedings of the Ninth ACM International Conference on Information and Knowledge Management (CIKM'00)*, pages 488–495, November 2000.
46. B. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary LP-norms. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB'00)*, pages 385–394, September 2000.