Efficient algorithms for sequence segmentation

Evimaria Terzi * Panayiotis Tsaparas †

Abstract

The sequence segmentation problem asks for a partition of the sequence into k non-overlapping segments that cover all data points such that each segment is as homogeneous as possible. This problem can be solved optimally using dynamic programming in $O(n^2k)$ time, where n is the length of the sequence. Given that sequences in practice are too long, a quadratic algorithm is not an adequately fast solution. Here, we present an alternative constant-factor approximation algorithm with running time $O(n^{4/3}k^{5/3})$. We call this algorithm the DNS algorithm. We also consider the recursive application of the DNS algorithm, that results in a faster algorithm ($O(n \log\log n)$ running time) with $O(\log n)$ approximation factor, and study the accuracy/efficiency tradeoff. Extensive experimental results show that these algorithms outperform other widely-used heuristics. The same algorithms can speed up solutions for other variants of the basic segmentation problem while maintaining constant their approximation factors. Our techniques can also be used in a streaming setting, with sublinear memory requirements.

1 Introduction

Recently, there has been an increasing interest in the data-mining community for mining sequential data. This is due to the existence of abundance of sequential datasets that are available for analysis, arising from applications in telecommunications, stock-market analysis, bioinformatics, text processing, click-stream mining and many more . The main problem associated with the analysis of these datasets is that they consist of huge number of data points. The analysis of such data requires efficient and scalable algorithms.

A central problem related to time-series analysis is the construction of a compressed and concise representation of the data, so that it is handled efficiently. One commonly used such representation is the *piecewise-constant* approximation. A piecewise-constant representation approximates a time series T of length n using k non-overlapping and contiguous segments that span the whole sequence. Each segment is represented by a single (constant) point, e.g., the mean of the points in the segment. We call this point the *representative* of the segment, since it represents the points in the segment. The error in this approximate representation is measured using some *error function*, e.g. the sum of squares. Different error functions may be used depending on the application. Given an error function, the goal is to find the segmentation of the sequence and the corresponding representatives that minimize the error in the representation of the underlying data. We call this problem a *segmentation problem*. Segmentation problems, particularly for multivariate time series, arise in many data mining applications, including bioinformatics [5, 15, 17] and context-aware systems [10].

This basic version of the sequence-segmentation problem can be solved optimally in time $O(n^2k)$ using dynamic programming [3], where n is the length of the sequence and k the number of segments. This quadratic algorithm, though optimal, is not satisfactory for data-mining applications where n is usually very large. In practice, faster heuristics are used. Though the latter are usually faster ($O(n \log n)$ or $O(n)$), there are no guarantees on the quality of the solutions they produce.

In this paper, we present a new *divide and segment* (DNS) algorithm for the sequence segmentation problem. The DNS algorithm has sub-quadratic running time, $O(n^{4/3}k^{5/3})$, and it is a 3-approximation algorithm for the segmentation problem. That is, the error of the segmentation it produces is provably no more than 3 times that of the optimal segmentation. Additionally, we explore several more efficient variants of the algorithm and we quantify the accuracy/efficiency tradeoff. More specifically, we define a variant that runs in time $O(n \log\log n)$ and has an $O(\log n)$ approximation ratio. All algorithms can be made to use sublinear amount of memory, making them applicable to the case that the data needs to be processed in an streaming fashion. We also propose an algorithm that requires logarithmic space, and linear time, albeit, with no approximation guarantees.

Extensive experiments on both real and synthetic datasets demonstrate that in practice our algorithms perform significantly better than the worst-case theoretical upper bounds. It is often the case that the more efficient variants of our algorithms are the ones that produce the best results, even though they are inferior in theory. In many cases our algorithms give results equivalent to the optimal algorithm. We also compare our algorithms against different popular heuristics that are known to work well in practice. Although these heuristics output results of good quality our algorithms still

*HIIT, Basic Research Unit Department of Computer Science University of Helsinki, Finland, email:terzi@cs.helsinki.fi

†HIIT, Basic Research Unit Department of Computer Science University of Helsinki, Finland, email:tsaparas@cs.helsinki.fi

perform consistently better. This can often be achieved with computational cost comparable to the cost of these heuristics. Finally, we show that the proposed algorithms can be applied to variants of the basic segmentation problem, like for example the one defined in [7]. We show that for this problem we achieve similar speedups for the existing approximation algorithms, while maintaining constant approximation factors.

1.1 Related Work. There is a large body of work that proposes and compares segmentation algorithms for sequential (mainly time-series) data. The papers related to this topic follow usually one of the following three trends: (i) Propose heuristic algorithms for solving a segmentation problem faster than the optimal dynamic-programming algorithm. Usually these algorithms are fast and perform well in practice. (ii) Devise approximation algorithms, with provable error bounds, and (iii) Propose new variations of the basic segmentation problem. These variations usually impose some constraint on the structure of the representatives of the segments. Our work lies in the intersection of categories (i) and (ii) since we provide fast algorithms with bounded approximation factors. At the same time, we claim that our techniques can be used for solving problems proposed in category (iii) as well.

The bulk of papers related to segmentations are in category (i). Since the optimal algorithm for solving the sequence segmentation problem is quadratic, faster heuristics that work well in practice are valuable. The most popular of these algorithms are the top-down and the bottom-up greedy algorithms. The first runs in time $O(n)$ while the second needs time $O(n \log n)$. Both these algorithms work well in practice. In Section 5.1 we discuss them in detail, and we evaluate them experimentally. Online versions of the segmentation problems have also been studied [11, 16]. In this case, new data points are coming in an online fashion. The goal of the segmentation algorithm is to output a good segmentation (in terms of representation error) at all points in time. In some cases, like for example in [11], it is assumed that the maximum tolerable error is also part of the input.

The most interesting work in category (ii) is presented in [8]. The authors present a fast segmentation algorithm with provable error bounds. Our work has similar motivation, but approaches the problem from a different point of view.

Variations of the basic segmentation problem have been studied extensively. In [7], the authors consider the problem of partitioning a sequence into k contiguous segments under the restriction that those segments are represented using only $h < k$ distinct representatives. We will refer to this problem as the (k, h)-*segmentation problem*. Another restriction that is of interest particularly in paleontological applications, is *unimodality*. In this variation the representatives of the segments are required to follow a unimodal curve, that is, a curve that changes curvature only once. The problem of finding unimodal segmentations is discussed in [9]. This problem can be solved optimally in polynomial time, using a variation of the basic dynamic-programming algorithm.

1.2 Roadmap. The rest of the paper is structured as follows. Section 2 provides the necessary definitions, and the optimal dynamic-programming algorithm. In Section 3 we describe the basic DNS algorithm, and we analyze its running time and approximation ratio. In Section 4 we consider a recursive application of our approach, resulting in more efficient algorithms. Section 5 includes a detailed experimental evaluation of our algorithms, and comparisons with other commonly used heuristics. Section 6 considers applications of our techniques to other segmentation problems. We conclude the paper in Section 7.

2 Preliminaries

Let $T = (t_1, t_2, \ldots, t_n)$ be a d-dimensional sequence of length n with $t_i \in \mathbb{R}^d$, i.e., $t_i = (t_{i1}, t_{i2}, \ldots, t_{id})$.

A k-*segmentation* S of a sequence of length n is a partition of $\{1, 2, \ldots, n\}$ into k non-overlapping contiguous subsequences (segments), $S = \{s_1, s_2, \ldots, s_k\}$. Each segment s_i consists of $|s_i|$ points. The representation of sequence T when segmentation S is applied to it, collapses the values of the sequence within each segment s into a single value μ_s (e.g., the mean). We call this value the *representative* of the segment, and each point $t \in s$ is "represented" by the value μ_s. Collapsing points into representatives results in less accuracy in the sequence representation. We measure this loss in accuracy using the error function E_p. Given a sequence T, the error of segmentation S is defined as

$$E_p(T, S) = \left(\sum_{s \in S} \sum_{t \in s} |t - \mu_s|^p \right)^{\frac{1}{p}}.$$

We consider the cases where $p = 1, 2$. For simplicity, we will sometimes write $E_p(S)$ instead of $E_p(T, S)$, when the sequence T is implied.

The *segmentation problem* asks for the segmentation that minimizes the error E_p. The representative of each segment depends on p. For $p = 1$ the optimal representative for each segment is the median of the points in the segment; for $p = 2$ the optimal representative of the points in a segment is their mean. Depending on the constraints one imposes on the representatives, one can consider several variants of the segmentation problem. We first consider the basic k-*segmentation problem*, where no constraints are imposed on the representatives of the segments. In Section 6 of the paper we consider the (k, h)-*segmentation problem*, a variant of the k-segmentation problem defined in [7], where only h distinct representatives can be used, for some $h < k$.

2.1 The segmentation problem. We now give a formal definition of the segmentation problem, and we describe the optimal algorithm for solving it. Let $\mathcal{S}_{n,k}$ denote the set of all k-segmentations of sequences of length n. For some sequence T, and for error measure E_p, we define the optimal segmentation as

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

That is, S_{opt} is the k-segmentation S that minimizes the $E_p(T,S)$. For a given sequence T of length n the formal definition of the k-segmentation problem is the following:

PROBLEM 1. (OPTIMAL k-SEGMENTATION) *Given a sequence T of length n, an integer value k, and the error function E_p, find $S_{opt}(T,k)$.*

Problem 1 is known to be solvable in polynomial time [3]. The solution consists of a standard dynamic-programming (DP) algorithm and can be computed in time $O(n^2k)$. The main recurrence of the dynamic-programming algorithm is the following:

$$\begin{aligned}(2.1)\quad & E_p(S_{\text{opt}}(T[1\ldots n],k)) = \\ & \min_{j<n} \quad \{E_p\left(S_{\text{opt}}(T[1\ldots j],k-1)\right) \\ & \qquad +E_p\left(S_{\text{opt}}(T[j+1,\ldots,n],1)\right)\}\ .\end{aligned}$$

where $T[i\ldots,j]$ denotes the subsequence of T that contains all points in positions from i to j, with i,j included. We note that the dynamic-programming algorithm can also be used in the case of weighted sequences, where each point is associated with a weight. Then the representatives (mean or median) are defined to be the weighted representatives (weighted mean or median).

3 Divide and Segment for the k-segmentation problem

3.1 DIVIDE&SEGMENT **algorithm.** In this section we describe the DIVIDE&SEGMENT (DNS) algorithm for Problem 1. The algorithm is faster than the standard dynamic-programming algorithm and its approximation factor is constant. 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 recurrence 2.1 is a building component of DNS. The output of the algorithm is a k-segmentation of the input sequence. Algorithm 1 shows the outline of DNS. In step 1, the input sequence T is partitioned into χ disjoint subsequences. Each one of them is segmented optimally using dynamic programming. For subsequence T_i, the output of this step is a segmentation S_i of T_i and a set M_i of k weighted points. These are the representatives of the segments of segmentation S_i, weighted by the length of the segment they represent. All the χk representatives of the χ subsequences are concatenated to form the (weighted) sequence T'. Then the dynamic-programming algorithm is applied on T'. The k-segmentation of T' is output as the final segmentation.

Algorithm 1 The DNS algorithm

Input: Sequence T of n points, number of segments k, value χ.

Ouput: A segmentation of T into k segments.

1: Partition T into χ disjoint intervals $T_1,\ldots,T_\chi$.
2: **for all** $i\in\{1,\ldots,\chi\}$ **do**
3: (S_i,M_i) = DP(T_i,k)
4: **end for**
5: Let $T' = M_1\oplus M_2\oplus\cdots\oplus M_\chi$ be the sequence defined by the concatenation of the representatives, weighted by the length of the interval they represent.
6: Return the optimal segmentation of (S,M) of T' using the dynamic programming algorithm.

The following example illustrates the execution of DNS.

EXAMPLE 1. *Consider the time series of length $n=20$ that is shown in Figure 3.1. We show the execution of the* DNS *algorithm for $k=2$ and using $\chi=3$. In step 1 the sequence is divided into three disjoint and contiguous intervals T_1, T_2 and T_3. Subsequently, the dynamic-programming algorithm is applied to each one of those intervals. The result of this are the six weighted points on which dynamic-programming is applied again. For this input sequence, the output 2-segmentation found by the* DNS *algorithm is the same as the optimal segmentation.*

The running time of the algorithm is easy to analyze.

THEOREM 3.1. *The running time of the* DNS *algorithm is at most $O(n^{4/3}k^{5/3})$ for $\chi=(\frac{n}{k})^{2/3}$.*

Proof. Assume that DNS partitions T into χ equal-length intervals. The running time of the DNS algorithm as a function of χ is

$$\begin{aligned}R(\chi) &= \chi\left(\frac{n}{\chi}\right)^2 k + (\chi k)^2 k \\ &= \frac{n^2}{\chi}k + \chi^2 k^3.\end{aligned}$$

The minimum of function $R(\chi)$ is achieved when $\chi_0 = \left(\frac{n}{k}\right)^{\frac{2}{3}}$ and this gives $R(\chi_0) = 2n^{4/3}k^{5/3}$. □

Throughout this paper we assume that the representative of each segment can be computed in constant time in the dynamic-programming subroutine of the algorithm. For the E_2-error function, it is possible to compute the mean in

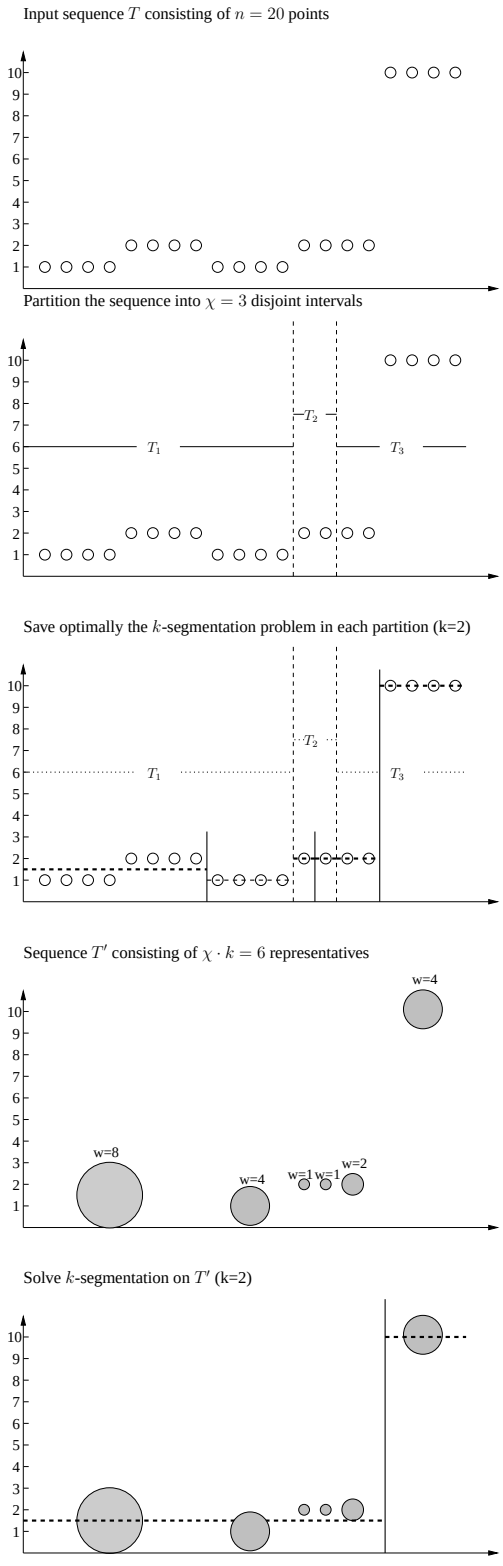

Figure 1: Illustration of the DNS algorithm

constant time by storing the partial sums of squares and squares of partial sums. For the E_1-error function, we are not aware of any method that computes the median of the segment in constant time. In the sequel, we assume that in this case a preprocessing step of computing medians of all possible segments is performed.

We note that the DNS algorithm can also be used in the case where the data must be processed in a streaming fashion. Assuming that we have an estimate of the size of the sequence n, then the algorithm processes the points in batches of size n/χ. For each such batch it computes the optimal k-segmentation, and stores the representatives. The space required is $M(\chi) = n/\chi + \chi k$. This is minimized for $\chi = \sqrt{n/k}$, resulting in space $M = 2\sqrt{nk}$.

3.1.1 Analysis of the DNS algorithm. For the proof of the approximation factor of the DNS algorithm we first make the following observation.

OBSERVATION 1. *Let $S_i = S_{opt}(T_i, k)$ for $i = 1, \ldots \chi$, and $S_{opt} = S_{opt}(T, k)$. If $\overline{t}$ is the representative assigned to point $t \in T$ by segmentation S_i after the completion of the for loop (Step 2) of the* DNS *algorithm, then we have*

$$\sum_{t \in T} d_p(t, \overline{t})^p = \sum_{i=1}^{\chi} E_p\left(T_i, S_i\right)^p \leq E_p\left(T, S_{opt}\right)^p .$$

Proof. For each interval T_i consider the segmentation points of S_{opt} that lie within T_i. These points together with the starting and ending points of interval T_i define a segmentation of T_i into k_i' segments with $k_i' \leq k$. Denote this segmentation by S_i'. Then for every interval T_i and its corresponding segmentation S_i' defined as above we have that: $E_p(T_i, S_i) \leq E_p(T_i, S_i')$. This is true since S_i is the optimal k-segmentation for subsequence T_i and $k_i' \leq k$. Thus we have

$$E_p\left(T_i, S_i\right)^p \leq E_p\left(T_i, S_i'\right)^p .$$

Summing over all T_i's we get

$$\begin{aligned} \sum_{t \in T} d_p(t, \overline{t})^p &= \sum_{i=1}^{\chi} E_p\left(T_i, S_i\right)^p \\ &\leq \sum_{i=1}^{\chi} E_p\left(T_i, S_i'\right)^p \\ &= E_p\left(T, S_{\text{opt}}\right)^p . \end{aligned}$$

□

We now prove the approximation factors for E_1 and E_2 error measures.

THEOREM 3.2. *For a sequence T and error measure E_1 let $\text{OPT}_1 = E_1(S_{opt}(T,k))$ be the E_1-error for the optimal k-segmentation. Also let DNS_1 be the E_1-error for the k-segmentation output by the* DNS *algorithm. We have that $\text{DNS}_1 \leq 3 \cdot \text{OPT}_1$.*

Proof. Let S be the segmentation of sequence T output by the $\text{DNS}(T,k,\chi)$ algorithm, and let μ_t be the representative assigned to some point $t \in T$ in S. Also, let λ_t denote the representative of t in the optimal segmentation $S_{\text{opt}}(T,k)$. The E_1-error of the optimal segmentation is

$$\text{OPT}_1 = E_1(S_{\text{opt}}(T,k)) = \sum_{t \in T} d_1(t,\lambda_t) \ .$$

The E_1 error of the DNS algorithm is given by

$$\text{DNS}_1 = E_1(T,S) = \sum_{t \in T} d_1(t,\mu_t).$$

Now let $\overline{t}$ be the representative of the segment to which point t is assigned after the completion of the for loop in Step 2 of the DNS algorithm. Due to the optimality of the dynamic-programming algorithm in Step 4 of the algorithm we have

$$\sum_{t \in T} d_1(\overline{t},\mu_t) \quad \leq \quad \sum_{t \in T} d_1(\overline{t},\lambda_t) \ . \tag{3.2}$$

We can now obtain the desired result as follows

$$\begin{aligned}
\text{DNS}_1 &= \sum_{t \in T} d_1(t,\mu_t) \\
&\leq \sum_{t \in T} \big(d_1(t,\overline{t}) + d_1(\overline{t},\mu_t)\big) && (3.3) \\
&\leq \sum_{t \in T} \big(d_1(t,\overline{t}) + d_1(\overline{t},\lambda_t)\big) && (3.4) \\
&\leq \sum_{t \in T} \big(d_1(t,\overline{t}) + d_1(\overline{t},t) + d_1(t,\lambda_t)\big) && (3.5) \\
&\leq 2 \cdot \sum_{t \in T} d_1(t,\lambda_t) + \sum_{t \in T} d_1(t,\lambda_t) && (3.6) \\
&= 3 \cdot \text{OPT}_1 \ .
\end{aligned}$$

Inequalities 3.3 and 3.5 follow from the triangular inequality, inequality 3.4 follows from Equation 3.2, and inequality 3.6 follows from Observation 1. □

Next we prove the 3-approximation result for E_2. For this, we need the following simple fact.

FACT 3.1. *(*DOUBLE TRIANGULAR INEQUALITY*) Let d be a distance metric. Then for points x, y and z and $p \in \mathbb{N}^+$ we have*

$$d(x,y)^2 \leq 2 \cdot d(x,z)^2 + 2 \cdot d(z,y)^2 \ .$$

THEOREM 3.3. *For a sequence T and error measure E_2 let $\text{OPT}_2 = E_2(S_{opt}(T,k))$ be the E_2-error for the optimal k-segmentation. Also let DNS_2 be the E_2-error for the k-segmentation output by the* DNS *algorithm. We have that $\text{DNS}_2 \leq 3 \cdot \text{OPT}_2$.*

Proof. Consider the same notation as in Theorem 3.2. The E_2 error of the optimal dynamic-programming algorithm is

$$\text{OPT}_2 = E_2(S_{\text{opt}}(T,k)) = \sqrt{\sum_{t \in T} d_2(t,\lambda_t)^2} \ .$$

Let S be the output of the $\text{DNS}(T,k,\chi)$ algorithm. The error of the DNS algorithm is given by

$$\text{DNS}_2 = E_2(T,S) = \sqrt{\sum_{t \in T} d_2(t,\mu_t)^2} \ .$$

The proof continues along the same lines as the proof of Theorem 3.2 but uses Fact 3.1 and Cauchy-Schwartz inequality.

Using the triangular inequality of d_2 we get

$$\begin{aligned}
\text{DNS}_2^2 &= \sum_{t \in T} d_2(t,\mu_t)^2 \\
&\leq \sum_{t \in T} \big(d_2(t,\overline{t}) + d_2(\overline{t},\mu_t)\big)^2 \\
&= \sum_{t \in T} d_2(t,\overline{t})^2 + \sum_{t \in T} d_2(\overline{t},\mu_t)^2 \\
&\quad + 2 \cdot \sum_{t \in T} d_2(t,\overline{t}) \cdot d_2(\overline{t},\mu_t) \ .
\end{aligned}$$

From Observation 1 we have that

$$\sum_{t \in T} d_2(t,\overline{t})^2 \leq \sum_{t \in T} d_2(t,\lambda_t)^2 = \text{OPT}_2^2 \ .$$

Using the above inequality, the optimality of dynamic programming in Step 4 of the algorithm, and Fact 3.1 we have

$$\begin{aligned}
\sum_{t \in T} d_2(\overline{t},\mu_t)^2 &\leq \sum_{t \in T} d_2(\overline{t},\lambda_t)^2 \\
&\leq 2 \cdot \sum_{t \in T} \big(d_2(\overline{t},t)^2 + d_2(t,\lambda_t)^2\big) \\
&\leq 4 \cdot \sum_{t \in T} d_2(t,\lambda_t)^2 \\
&= 4 \cdot \text{OPT}_2^2 \ .
\end{aligned}$$

Finally using the Cauchy-Schwartz inequality we get

$$\begin{aligned} 2 \cdot \sum_{t \in T} d_2(t,\bar{t}) \cdot d_2(\bar{t},\mu_t) &\leq 2 \cdot \sqrt{\sum_{t \in T} d_2(t,\bar{t})^2} \\ &\quad \cdot \sqrt{\sum_{t \in T} d_2(\bar{t},\mu_t)^2} \\ &\leq 2 \cdot \sqrt{\textsc{Opt}_2^2} \cdot \sqrt{4 \cdot \textsc{Opt}_2^2} \\ &= 4 \cdot \textsc{Opt}_2^2 \, . \end{aligned}$$

Combining all the above we conclude that

$$\textsc{DnS}_2^2 \leq 9 \cdot \textsc{Opt}_2^2 \, .$$

□

4 Recursive DnS algorithm

The DnS algorithm applies the "divide-and-segment" idea once, splitting the sequence into subsequences, partitioning each of subsequence optimally, and then merging the results. We now consider the recursive DnS algorithm (RDnS) which recursively splits each of the subsequences, until no further splits are possible. Algorithm 2 shows the outline of the RDnS algorithm.

Algorithm 2 The RDnS algorithm

Input: Sequence T of n points, number of segments k, value χ.

Ouput: A segmentation of T into k segments.

1: **if** $|T| \leq B$ **then**
2: Return the optimal partition (S,M) of T using the dynamic-programming algorithm.
3: **end if**
4: Partition T into χ intervals $T_1, \ldots, T_\chi$.
5: **for all** $i \in \{1, \ldots, \chi\}$ **do**
6: (S_i, M_i) = RDnS(T_i, k, χ)
7: **end for**
8: Let $T' = M_1 \oplus M_2 \oplus \cdots \oplus M_\chi$ be the sequence defined by the concatenation of the representatives, weighted by the length of the interval they represent.
9: Return the optimal partition (S,M) of T' using the dynamic-programming algorithm.

The value B is a constant that defines the base case for the recursion. Alternatively, one could directly determine the depth ℓ of the recursive calls to RDnS. We will refer to such an algorithm, as the ℓ-RDnS algorithm. For example, the simple DnS algorithm, corresponds to the 1-RDnS algorithm. We also note that at every recursive call of the RDnS algorithm the number χ of intervals into which we partition the sequence may be a function of sequence length. However, for simplicity we use χ instead of $\chi(n)$.

As a first step in the analysis of the RDnS we consider the approximation ratio of the ℓ-RDnS algorithm. We can prove the following theorem.

THEOREM 4.1. *The ℓ-RDnS algorithm is an $O(2^\ell)$ approximation algorithm for the E_1-error function, and an $O(6^{\ell/2})$ approximation algorithm for the E_2-error function, with respect to Problem 1.*

Proof. (Sketch) The proof in both cases follows by induction on the value of ℓ. The exact approximation ratio is $2^{\ell+1} - 1$ for E_1, and $\sqrt{\frac{9}{5}6^\ell - \frac{4}{5}}$ for E_2. We will sketch the proof for E_1. The proof for E_2 follows along the same lines.

From Theorem 3.2, we have that the theorem is true for $\ell = 1$. Assume now that it is true for some $\ell \geq 1$. We will prove it for $\ell + 1$. At the first level of recursion the $(\ell+1)$-RDnS algorithm, breaks the sequence T into χ subsequences $T_1, \ldots, T_\chi$. For each one of these we call the ℓ-RDnS algorithm, producing a set R of χk representatives. Similar to the proof of Theorem 3.2, let $\bar{t} \in R$ denote the representative in R that corresponds to point t. Consider also the optimal segmentation of each of these intervals, and let O denote the set of χk representatives. Let $\widetilde{t} \in O$ denote the representative of point t in O. From the inductive hypothesis we have that

$$\sum_{t \in T} d_1(t,\bar{t}) \leq \left(2^{\ell+1} - 1\right) \sum_{t \in T} d_1(t,\widetilde{t})$$

Now let μ_t be the representative of point t in the segmentation output by the $(\ell+1)$-RDnS algorithm. Also let λ_t denote the representative of point t in the optimal segmentation. Let $\textsc{RDnS}_1$ denote the E_1-error of the $(\ell+1)$-RDnS algorithm, and $\textsc{Opt}_1$ denote the E_1-error of the optimal segmentation. We have that

$$\textsc{RDnS}_1 = \sum_{t \in T} d_1(t,\mu_t) \quad \text{and} \quad \textsc{Opt}_1 = \sum_{t \in T} d_1(t,\lambda_t)$$

From the triangular inequality we have that

$$\begin{aligned} \sum_{t \in T} d_1(t,\mu_t) &\leq \sum_{t \in T} d_1(t,\bar{t}) + \sum_{t \in T} d_1(\bar{t},\mu_t) \\ &\leq \left(2^{\ell+1} - 1\right) \sum_{t \in T} d_1(t,\widetilde{t}) + \sum_{t \in T} d_1(\bar{t},\mu_t) \end{aligned}$$

From Observation 1, and Equation 3.2, we have that

$$\sum_{t \in T} d_1(t,\widetilde{t}) \leq \sum_{t \in T} d_1(t,\lambda_t)$$

$$\sum_{t \in T} d_1(\bar{t},\mu_t) \leq \sum_{t \in T} d_1(\bar{t},\lambda_t)$$

Using the above inequalities and the triangular inequality we obtain

$$\begin{aligned}
\text{RDNS}_1 &= \sum_{t\in T} d_1(t,\mu_t) \\
&\le \left(2^{\ell+1}-1\right)\sum_{t\in T} d_1(t,\lambda_t) + \sum_{t\in T} d_1(\bar{t},\lambda_t) \\
&\le \left(2^{\ell+1}-1\right)\sum_{t\in T} d_1(t,\lambda_t) \\
&\quad + \sum_{t\in T} d_1(t,\bar{t}) + \sum_{t\in T} d_1(t,\lambda_t) \\
&\le 2^{\ell+1}\sum_{t\in T} d_1(t,\lambda_t) + \left(2^{\ell+1}-1\right)\sum_{t\in T} d_1(t,\widetilde{t}) \\
&\le \left(2^{\ell+2}-1\right)\sum_{t\in T} d_1(t,\lambda_t) \\
&= \left(2^{\ell+2}-1\right)\text{OPT}_1
\end{aligned}$$

The proof for the E_2 follows similarly. Instead of using the binomial identity as in the proof of Theorem 3.3, we obtain a more clean recursive formula for the approximation error by applying the double triangular inequality. □

We now consider possible values for χ. First, we set χ to be a constant. We can prove the following theorem.

THEOREM 4.2. *For any constant χ the running time of the* RDNS *algorithm is $O(n)$, where n is the length of the input sequence. The algorithm can operate on data that arrive in streaming fashion using $O(\log n)$ space.*

Proof. (Sketch) The running time of the RDNS algorithm is given by the following recursion

$$R(n) = \chi R\left(\frac{n}{\chi}\right) + (\chi k)^2 k.$$

Solving the recursion we get that $R(n) = O(n)$.

In a case that the data arrive in a stream, the algorithm can build the recursion tree online, in a bottom-up fashion. At each level of the recursion tree, we only need to maintain at most χk entries that correspond to the leftmost branch of the tree. The depth of the recursion is $O(\log n)$, resulting in $O(\log n)$ space overall. □

Therefore, for constant χ, we obtain an efficient algorithm, both in time and space. Unfortunately, we do not have any approximation guarantees, since the best approximation bound we can prove using Theorem 4.1 is $O(n)$. We can however obtain significantly better approximation guarantees if we are willing to tolerate a small increase in the running time. We set $\chi = \sqrt{n}$, where n is the length of the input sequence at each specific recursive call. That is, at each recursive call we split the sequence into $\sqrt{n}$ pieces of size $\sqrt{n}$.

THEOREM 4.3. *For $\chi = \sqrt{n}$ the* RDNS *algorithm is an $O(\log n)$ approximation algorithm for Problem 1 for both E_1 and E_2 error functions. The running time of the algorithm is $O(n \log\log n)$, using $O(\sqrt{n})$ space, when operating in a streaming fashion.*

Proof. (Sketch) It is not hard to see that after ℓ recursive calls the size of the input segmentation is $O(n^{1/2^\ell})$. Therefore, the depth of the recursion is $O(\log\log n)$. From Theorem 4.1 we have that the approximation ratio of the algorithm is $O(\log n)$. The running time of the algorithm is given by the recurrence

$$R(n) = \sqrt{n}R\left(\sqrt{n}\right) + nk^3.$$

Solving the recurrence we obtain running time $O(n \log\log n)$. The space required is bounded by the size of the top level of the recursion, and it is $O(\sqrt{n})$. □

The following corollary is an immediate consequence of the proof of Theorem 4.3 and it provides an accuracy/efficiency tradeoff.

COROLLARY 4.1. *For $\chi = \sqrt{n}$, the ℓ-*RDNS *algorithm is an $O(2^\ell)$ approximation algorithm for the E_1-error function, and an $O(6^{\ell/2})$ approximation algorithm for the E_2-error function, with respect to Problem 1. The running time of the algorithm is $O(n^{1+1/2^\ell} + n\ell)$.*

5 Experiments

5.1 Segmentation heuristics. Since sequence segmentation is a basic problem particularly in time-series analysis several algorithms have been proposed in the literature with the intention to improve the running time of the optimal dynamic-programming algorithm. These algorithms have been proved very useful in practice, however no approximation bounds are known for them. For completeness we briefly describe them here.

The TOP-DOWN greedy algorithm (TD) starts with the unsegmented sequence (initially there is just a single segment) and it introduces a new boundary at every greedy step. That is, in the i-th step it 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 change is made to the existing $i-1$ boundary points. The splitting process is repeated until the number of segments of the output segmentation reaches k. This algorithm, or variations of it with different stopping conditions are used in [4, 6, 14, 18]. The running time of the algorithm is $O(nk)$.

In the BOTTOM-UP greedy algorithm (BU) initially each point forms a segment on 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 complexity of the bottom-up algorithm is

$O(n \log n)$. BU performs well in terms of error and it has been used widely in time-series segmentation [9, 16].

The LOCAL ITERATIVE REPLACEMENT (LiR) and GLOBAL ITERATIVE REPLACEMENT (GiR) are randomized algorithms for sequence segmentations proposed in [10]. Both algorithms 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. The algorithms repeat these steps until they converge, i.e., they cannot improve the error of the output segmentation. The two algorithms differ in the types of replacements of the segmentation boundaries they are allowed to do. Consider a segmentation $s_1, s_2, \ldots, s_k$. Now assume that both (LiR) and (GiR) pick segment boundary s_i for replacement. LiR is only allowed to put a new boundary between points s_{i-1} and s_{i+1}. On the other hand, GiR is allowed to put a new segment boundary anywhere on the sequence. Both algorithms run in time $O(In)$, where I is the number of iterations necessary for convergence.

Although extensive experimental evidence shows that these algorithms perform well in practice, there is no known guarantee of their worst-case error ratio.

5.2 Experimental setup. We show the qualitative performance of the proposed algorithms via an extensive experimental study. For this, we compare the family of "divide-and-segment" algorithms with all the heuristics described in the previous subsection. We also explore the quality of the results given by RDNS compared to DNS for different parameters of the recursion (i.e., number of recursion levels, value of χ).

For the study we use two types of datasets: (a) synthetic and (b) real data. The synthetic data are generated as follows: First we fix the dimensionality d of the data. Then we select k segment boundaries, which are common for all the d dimensions. For the j-th segment of the i-th dimension we select a mean value μ_{ij}, which is uniformly distributed in $[0, 1]$. Points are then generated by adding a noise value sampled from the normal distribution $\mathcal{N}(\mu_{ij}, \sigma^2)$. For the experiments we present here we have fixed the number of segments $k = 10$. We have generated datasets with $d = 1, 5, 10$, and standard deviations varying from 0.05 to 0.9.

The real datasets were downloaded from the UCR time-series data mining archive [12][1].

5.3 Performance of the DNS algorithm. Figures 2 and 3 show the performance of different algorithms for the synthetic datasets. In particular we plot the error ratio $\frac{\text{A}}{\text{OPT}}$ for A being the error of the solutions found by the algorithms DNS, BU, TD, LiR and GiR. OPT represents the error of the optimal solution. The error ratio is shown as a function of the number of segments (Figure 2), or the variance of the generated datasets (Figure 3). In all cases, the DNS algorithm consistently outperforms all other heuristics, and the error it achieves is very close to that of the optimal algorithm. Note that in contrast to the steady behavior of DNS the quality of the results of the other heuristics varies for the different parameters and no conclusions on their behavior on arbitrary datasets can be drawn.

This phenomenon is even more pronounced when we experiment with real data. Figure 4 is a sample of similar experimental results obtained using the datasets *balloon*, *darwin*, *winding*, *xrates* and *phone* from the UCR repository. The DNS performs extremely well in terms of accuracy, and it is again very robust across different datasets for different values of k. Overall, GiR performs the best among the rest of the heuristics. However, there are cases (e.g., the balloon dataset) where GiR is severely outperformed.

5.4 Exploring the benefits of the recursion. We additionally compare the basic DNS algorithm with different versions of RDNS. The first one, FULL-RDNS (full recursion), is the RDNS algorithm when we set the value of χ to be a constant. This algorithm runs in linear time (see Theorem 4.2). However, we have not derived any approximation bound for it (other than $O(n)$). The second one, SQRT-RDNS, is the RDNS algorithm when we set χ to be $\sqrt{n}$. At every recursive call of this algorithm the parental segment of size s is split into $O(\sqrt{s})$ subsegments of the same size. This variation of the recursive algorithm runs in time $O(n \log \log n)$ and has approximation ratio $O(\log n)$ (see Theorem 4.3). We study experimentally the tradeoffs between the running time and the quality of the results obtained using the three different alternatives of "divide-and-segment" methods on synthetic and real datasets. We also compare the quality of those results with the results obtained using GiR algorithm. We choose this algorithm for comparison since it has proved to be the best among all the other heuristics. In Figures 5 and 6 we plot the error ratios of the algorithms as a function of the number of segments and the variance for the synthetic datasets. Figure 7 shows the experiments on real datasets.

From the results we can make the following observations. First, all the algorithms of the divide-and-segment family perform extremely well, giving results close to the optimal segmentation and usually better than the results obtained by GiR. The full recursion (FULL-RDNS) does harm the quality of the results. However, we note that in order to study the full effect of recursion on the performance of the algorithm we set $\chi = 2$, the minimum possible value. We believe that for larger values of χ the performance of FULL-RDNS will be closer to that of DNS (for which we have $\chi = (n/k)^{2/3}$). Finally, there are cases where SQRT-

[1] The interested reader can find the datasets at http://www.cs.ucr.edu/∼eamonn/TSDMA/.

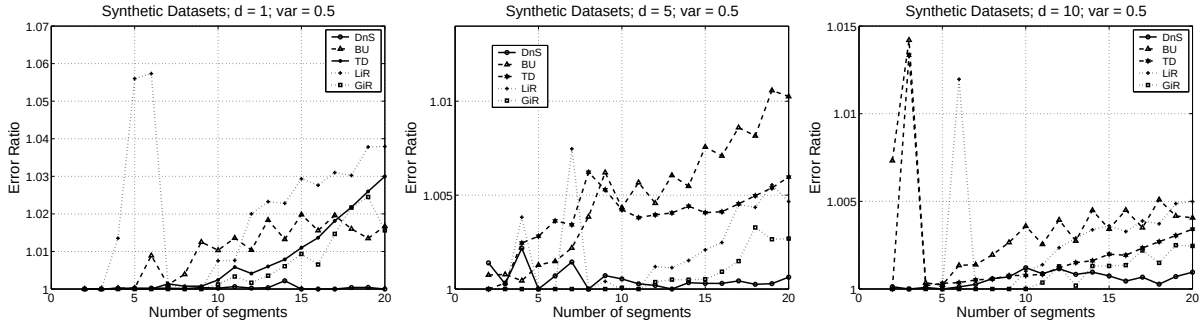

Figure 2: Error ratio of different algorithms with respect to OPT as a function of the number of segments

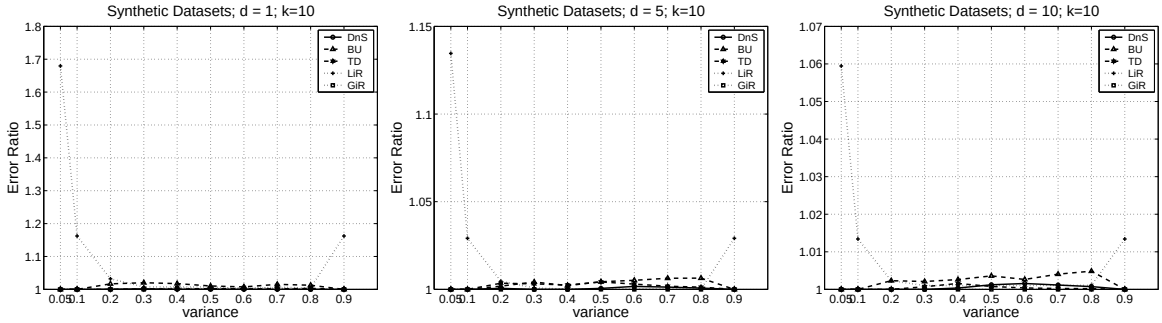

Figure 3: Error ratio of different algorithms with respect to OPT as a function of the variance of the generated datasets

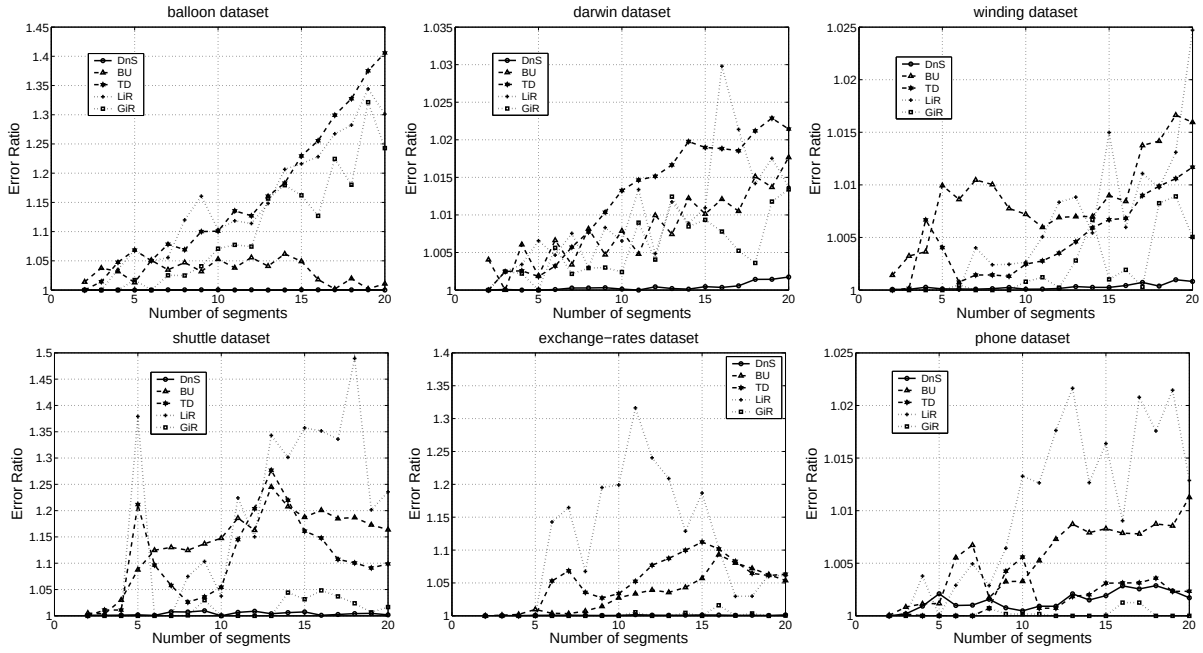

Figure 4: Error ratio of different algorithms with respect to OPT as a function of the number of segments for different real datasets

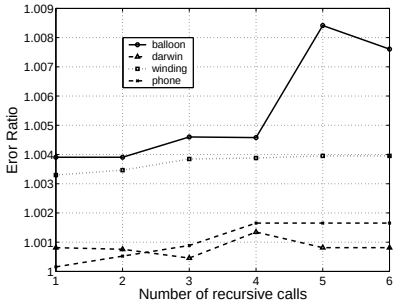

Figure 8: Error ratio of ℓ-RDNS for different number of recursion calls; Real datasets.

RDNS (and in some settings FULL-RDNS) performs even better than simple DNS. This phenomenon is due to the difference in the number and the positions of the splitting points the two algorithms pick for the division step. It appears that, in some cases, performing more levels of recursion helps the algorithm identify better segment boundaries, and thus produce segmentations of lower cost.

Figure 8 shows how the error of the segmentation output by ℓ-RDNS changes for different number of recursion levels, for four real datasets (*balloon*, *darwing*, *phone* and *winding*). Note that even for 5 levels of recursion the ratio never exceeds 1.008.

6 Applications to other segmentation problems

Here, we discuss the application of the simple DNS algorithm for a variant of the k-segmentation problem, namely the (k,h)-segmentation [7]. Similar to the k-segmentation, the (k,h)-segmentation of sequence T asks again for a partition of T in k segments. The main difference is that now the representatives of each segment are not chosen independently. In the (k,h)-segmentation only $h < k$ distinct representatives can be used to represent the k segments. We have picked this problem to demonstrate the usefulness of the DNS algorithm because of the applicability of (k,h)-segmentation to the analysis of long genetic sequences. For that kind of analysis, efficient algorithms for the (k,h)-segmentation problem are necessary.

Let S be a (k,h)-segmentation of the sequence T. For each segment s of the segmentation S, let ℓ_s be the representative for this segment (there are at most h representatives). The error E_p of the (k,h)-segmentation is defined as follows

$$E_p(T,S) = \left(\sum_{s \in S} \sum_{t \in s} |t - \ell_s|^p \right)^{\frac{1}{p}} .$$

Let $\mathcal{S}_{n,k,h}$ denote the family of all segmentations of sequences of length n into k segments using h representatives. In a similar way to the k-segmentation, for a given sequence T of length n and error measure E_p, and for given $k, h \in \mathbb{N}$ with $h < k$, the optimal (k,h)-segmentation is defined as

$$S_{\text{opt}}(T,k,h) = \arg \min_{S \in \mathcal{S}_{n,k,h}} E_p(T,S) . \quad (6.7)$$

Therefore, the optimal (k,h)-segmentation is defined as follows:

PROBLEM 2. (OPTIMAL (k,h)-SEGMENTATION) *Given a sequence T of length n, integer values k and h with $h < k \leq n$ and error function E_p, find $S_{opt}(T,k,h)$.*

6.1 Algorithms for the (k,h)-segmentation problem

The (k,h)-segmentation problem is known to be NP-Hard for $d \geq 2$ and $h < k$, since it contains clustering as its special case [7]. Approximation algorithms, with provable approximation guarantees are presented in [7] and their running time is $O(n^2(k+h))$. We now discuss two of the algorithms presented in [7]. We subsequently modify these algorithms, so that they use the DNS algorithm as their subroutine.

Algorithm SEGMENTS2LEVELS: The algorithm initially solves the k-segmentation problem obtaining a segmentation S. Then it solves the (n,h)-segmentation problem obtaining a set L of h levels. Finally, the representative μ_s of each segment $s \in S$ is assigned to the level in L that is the closest to μ_s.

Algorithm CLUSTERSEGMENTS: As before, the algorithm initially solves the k-segmentation problem obtaining a segmentation S. Each segment $s \in S$ is represented by its representative μ_s weighted by the length of the segment $|s|$. Finally, a set L of h levels is produced by clustering the k weighted points into h clusters.

6.2 Applying DNS to the (k,h)-segmentation problem

Step 1 of both SEGMENTS2LEVELS and CLUSTERSEGMENTS algorithms uses the optimal dynamic-programming algorithm for solving the k-segmentation problem. Using DNS instead we can achieve the following approximation results:

THEOREM 6.1. *If algorithm* SEGMENTS2LEVELS *uses* DNS *for obtaining the k-segmentation, and the clustering step is done using an α approximation algorithm, then the overall approximation factor of* SEGMENTS2LEVELS *is $(6+\alpha)$ for both E_1 and E_2-error measures.*

When the data points are of dimension 1 ($d = 1$) then clustering can be solved optimally using dynamic programming and thus the approximation factor is 7 for both E_1 and E_2 error measures. For $d > 1$ and for both E_1 and E_2 error measures the best α is $1+\epsilon$ using the algorithms proposed in [1] and [13] respectively.

THEOREM 6.2. *Algorithm* CLUSTERSEGMENTS *that uses* DNS *in for obtaining the k-segmentation, has approximation factor $\sqrt{29}$ for E_2-error measure, and 11 for E_1-error measure.*

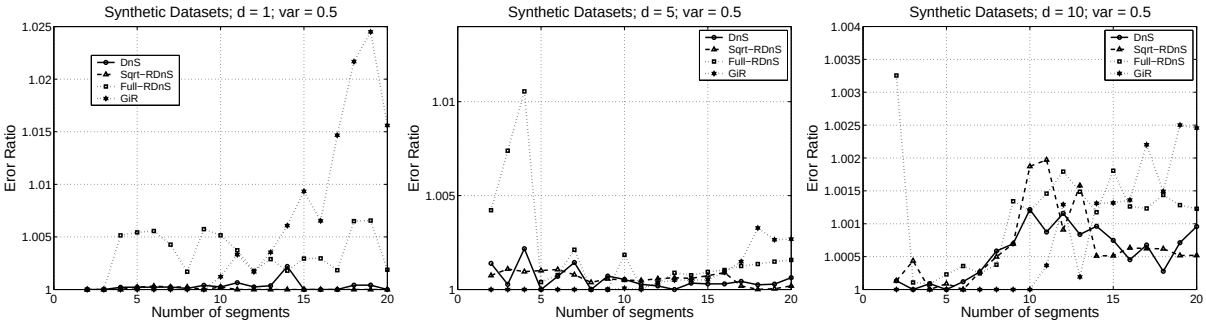

Figure 5: Error ratio of DNS and RDNS algorithms with respect to OPT for synthetic datasets.

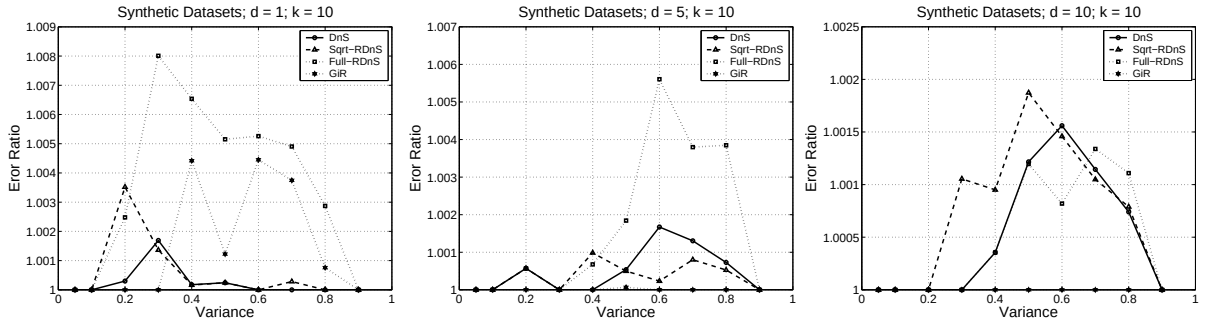

Figure 6: Error ratio of DNS and RDNS algorithms with respect to OPT for synthetic datasets.

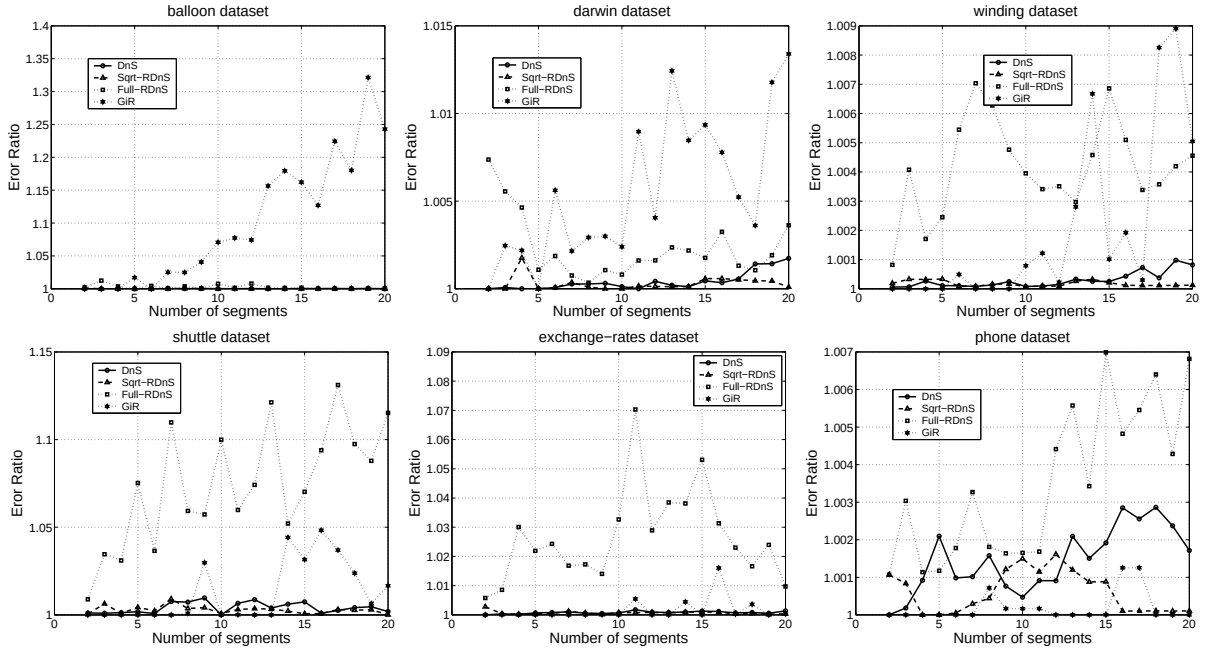

Figure 7: Error ratio of DNS and RDNS algorithms with respect to OPT for real datasets

Notice that the clustering step of the CLUSTERSEGMENTS algorithm does not depend on n and thus one can assume that clustering can be solved optimally in constant time, since usually $k << n$. However, if this step is solved approximately using the clustering algorithms of [1] and [13], the approximation ratios of the CLUSTERSEGMENTS algorithm that uses DNS for segmenting, becomes $11 + \epsilon$ for E_1 and $\sqrt{29 + \epsilon}$ for E_2.

Given Theorem 3.1 and using the linear-time clustering algorithm for E_2 proposed in [13] and the linear-time version of the algorithm proposed in [2] for E_1 we get the following result:

COROLLARY 6.1. *Algorithms* SEGMENTS2LEVELS *and* CLUSTERSEGMENTS *when using* DNS *in their first step run in time* $O(n^{4/3}k^{5/3})$ *for both* E_1 *and* E_2 *error measure.*

In a similar way one can derive the benefits of using the DNS and R-DNS algorithms to other segmentation problems (like for example unimodal segmentations [9]).

7 Conclusions

In this paper we described a family of approximation algorithms for the k-segmentation problem. The most basic of those algorithms (DNS) works in time $O(n^{4/3}k^{5/3})$ and has is a 3-approximation algorithm. We have described and analyzed several variants of this basic algorithm that are faster, but have worse approximation bounds. Furthermore, we quantified the accuracy versus speed tradeoff. Our experimental results on both synthetic and real datasets show that the proposed algorithms outperform other heuristics proposed in the literature and that the approximation achieved in practice is far below the bounds we obtained analytically.

Acknowledgments

We would like to thank Aris Gionis and Heikki Mannila for helpful discussions and advice.

References

[1] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-medians and related problems. In *STOC*, pages 106–113, 1998.

[2] V. Arya, N. Garg, R. Khandekar, K. Munagala, and V. Pandit. Local search heuristic for k-median and facility location problems. In *STOC*, pages 21–29. ACM Press, 2001.

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

[4] P. Bernaola-Galvan, R. R.-R. R, and J. Oliver. Compositional segmentation and long-range fractal correlations in dna sequences. *Phys Rev E Stat Phys Plasmas Fluids Relat Interdiscip Topics*, 53(5):5181–5189, 1996.

[5] H. J. Bussemaker, H. Li, and E. D. Siggia. Regulatory element detection using a probabilistic segmentation model. In *ISMB*, pages 67–74, 2000.

[6] D. Douglas and T. 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.

[7] A. Gionis and H. Mannila. Finding recurrent sources in sequences. In *RECOMB*, pages 115–122, Berlin, Germany, 2003.

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

[9] N. Haiminen and A. Gionis. Unimodal segmentation of sequences. In *ICDM*, pages 106–113, 2004.

[10] J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmäki, and H. Toivonen. Time series segmentation for context recognition in mobile devices. In *ICDM*, pages 203–210, 2001.

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

[12] E. Keogh and T. Folias. The UCR time series data mining archive, 2002.

[13] A. Kumar, Y. Sabharwal, and S. Sen. A simple linear time $(1+\epsilon)$ -approximation algorithm for k-means clustering in any dimensions. In *FOCS*, pages 454–462, 2004.

[14] V. Lavrenko, M. Schmill, D. Lawrie, P. Ogilvie, D. Jensen, and J. Allan. Mining concurrent text and time series. In *In proceedings of the 6th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining Workshop on Text Mining*, pages 37–44, 2000.

[15] W. Li. DNA segmentation as a model selection process. In *RECOMB*, pages 204–210, 2001.

[16] T. Palpanas, M. Vlachos, E. J. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *ICDE*, pages 338–349, 2004.

[17] M. Salmenkivi, J. Kere, and H. Mannila. Genome segmentation using piecewise constant intensity models and reversible jump MCMC. In *ECCB*, pages 211–218, 2002.

[18] H. Shatkay and S. 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.