Homogeneous String Segmentation using Trees and Weighted Independent Sets*

Peter Damaschke
Department of Computer Science and Engineering
Chalmers University, 41296 Göteborg, Sweden
ptr@chalmers.se

Abstract

We divide a string into k segments, each with only one sort of symbols, so as to minimize the total number of exceptions. Motivations come from machine learning and data mining. For binary strings we develop a linear-time algorithm for any k. Key to efficiency is a special-purpose data structure, called W-tree, which reflects relations between repetition lengths of symbols. For non-binary strings we give a nontrivial dynamic programming algorithm. Our problem is equivalent to finding weighted independent sets with certain size constraints, either in paths (binary case) or special interval graphs (general case). We also show that this problem is FPT in bounded-degree graphs.

Keywords: segmentation, dynamic programming, tree computations, weighted independent set, interval graphs, parameterized complexity

1 Introduction

String segmentation problems appear in language processing, classification, and machine learning. Application examples include the automatic segmentation of text in Asian languages into word, and segmentation of texts into parts dealing with one main subject. The putative quality of segmentations is measured by some objective function based on plausibility of segments. A main type of quality measure is similarity within the segments, see, e.g., [17].

Segmentation of more abstract (e.g., numerical) linearly ordered data is a related topic. Efficient algorithms for cutting time series into somehow homogeneous segments are given in [20], along with interesting motivations

*This is an improved version of an article presented at the *33rd International Workshop on Graph-Theoretic Concepts in Computer Science WG 2007*, Dornburg near Jena, *Lecture Notes in Computer Science* (Springer) 4769, pp. 214–225. The present manuscript slightly deviates from the journal version, due to small corrections.

for their use. A similar problem with other demands on the segments is studied in [19]. Segmentations of labeled point sets in higher dimensions by linear discriminators according to various homogeneity criteria lead to nontrivial computational problems which are relevant in neural network and decision tree learning. We refer to the series of papers [5, 6, 7, 8].

Dynamic programming is the obvious design technique for string segmentation (see [23, Chapter 6, Exercise 5]), but the precise complexity depends on the objective. In the present work we consider a specific objective. Let S be a string of n symbols from an alphabet of size b. The *segment* $S[i..j]$ is the substring of S from position i to position j. A *segmentation* splits S into segments and assigns a *designated symbol* to each. An *exception* in a segment is any occurence of a symbol distinct from the designated one. Our objective is:

Segmentation with Minimized Exceptions (SME)
Given a string S of n symbols from an alphabet of size b, and some $k < n$, compute a segmentation of S with at most k segments that minimizes the total number of exceptions. By SME-Binary we denote SME over alphabet $\{0, 1\}$.

A general motivation of SME-Binary comes from machine learning. Suppose that some binary feature f of objects depends randomly on a real-valued attribute x, in such a way that, for value x, we would observe $f = 1$ with probability $p(x)$. Function p is unknown but supposed to be "smooth", that is, p oscillates only a few times on the x-axis. Once we have sampled many values of f along the x-axis, these data, sorted by x, naturally form a string S of symbols 0 and 1. In order to predict f of yet unseen objects with other values x, we would split S into homogeneous segments, as good as possible, and always predict $f(x)$ to be the majority symbol in the segment containing the probed x. Then, the rate of prediction errors is roughly the total number of exceptions divided by $|S|$. Optimal segmentation can also serve as a subroutine in more complex machine learning tasks where real-valued attributes must be discretized, as in decision tree learning. A similar discussion applies to SME over larger alphabets. As for the choice of k, there is a trade-off between homogeneity of segments and reliability: A larger k results in fewer exceptions in the string S of training data, but on the other hand, predictions in smaller segments are less reliable due to overfitting. Thus we study SME with an arbitrary, user-defined parameter k.

String segmentation with entropy as the proposed homogeneity measure is studied in [15, 16] with similar motivations. Entropy minimization requires different algorithmic techniques. The total number of exceptions has also been introduced as the training set error, in a more general machine learning context [28, 1], and also considered for discretization of one-dimensional binary data sets [24]. A body of work on discretization followed, including structural properties (so-called well-behavedness), comparisons of the different homogeneity measures from a statistical point of view, and several practical algorithms and heuristics. The whole literature cannot be surveyed

here in detail, we refer to, e.g., [13, 14, 25]. Dynamic programming was rediscovered several times, but the question of fastest possible algorithms for SME has not been studied before, to our best knowledge. (Some algorithms were attributed "linear-time", but actually this refered to dynamic programming approaches which run in linear time only for fixed size of the parameters). In the present paper we accomplish time bounds that save some factors compared to the standard dynamic programming algorithms. The main finding is an algorithm for SME-Binary running in linear time regardless of the desired number of segments. Speed can make a difference in applications with very long strings, such as binarized signals with high sample rates, DNA strings, etc.

We came across with a concrete application of SME-Binary in biomedical engineering. The pattern of burst and suppression intervals in electroencephalograms (EEG) gives important information about the brain function. In an attempt to automatize this segmentation (in order to enable monitoring of patients), criteria for burst and suppression intervals have been learned by generic machine learning techniques, based on discretized spectral feartures of EEG signals and judgements made by EEG specialists [27]. Simple thresholding of spectral features would not yield the excpected segmentations, but experimental data suggest that SME-Binary could be an excellent approach to recognize bursts in a simpler and more transparent way: Bursts (suppression intervals) seem to be segments where high (low) amplitudes abound [26].

Various related topics are continuously interesting as well. The problem considered in [29] is to compute a single segmentation that is close to a set of already given segmentations. Dynamic programming is used to minimze the disagreement distance. Segmentation of sequences of numbers into histograms that minimize standard error functions is considered in [18]. Their algorithms find approximately optimal solutions on small space and can be used also for data streams. A practical heuristic for piecewise linear approximation of real-valued functions with desirable properties (from the data mining point of view) is proposed in [21]. Fast approximation algorithms for partitioning a sequence of points into segments represented by a single point, being close to all points in the segment, are given in [31].

Overview of the paper and our contributions: To keep the presentation self-contained we included proofs of the basic facts and simpler algorithms which later form the base of our actual contributions, but we presume that the reader knows standard graph-theoretic notions.

In Section 2 we observe that we never need to split the r *runs* of S, that is, the r maximal substrings consisting of only one sort of symbols. Runs and their lengths are trivially computed in an $O(n)$ time preprocessing, thus we state all further time bounds as functions of b, k, r, where we count arithmetic operations with integers as $O(1)$ time operations. The time used for the actual segmentation is significant especially if S comes from noisy data with many short runs, so that r remains large. SME is easy to solve by dynamic programming in $O(br \cdot \min\{k, r-k\})$ time, but we develop faster,

non-obvious algorithms. In Section 3 we show that SME-Binary is equivalent to computing a minimum weight independent set of exactly m vertices in a path of (essentially) r vertices, where $m \approx (r-k)/2$. Some greedy algorithm with a carefully specified greedy rule solves SME-Binary in $O(r + m \log m)$ time. In the practically relevant case $k \ll r$ this would become $O(r \log r)$. However, in Section 4 we use this greedy algorithm to show correctness of yet another algorithm that works in $O(r)$ time for any k. This result is achieved by a special data structure that we call the W-tree.

Very recently, a linear-time algorithm has been reported in [2, 4] for the k-cover problem, which is to determine k disjoint segments with maximum total score, in a sequence of n positive and negative numbers called scores. The problem is motivated from bioinformatics [10], and an earlier version of the algorithm using union-find was slightly slower [3]. However, it turns out that some reformulations reduce that problem in linear time to an instance of SME-Binary with n runs and $2k+1$ segments, so that a linear-time algorithm for SME-Binary also implies the result for k-covers. Moreover, from our W-trees we can read off optimal solutions for any desired number of segments in a simple and intuitive way.

Computing minimum weight independent sets of exactly m vertices in general graphs is interesting for its own. In Section 5 we show that this problem is fixed-parameter tractable (FPT), with m and the maximum degree d as combined parameters.

Back to SME, our method for SME-Binary breaks down if $b > 2$, and dynamic programming is still the best approach that we currently have in the non-binary case. Still we can improve upon the simple time bound. In Section 6 we reduce SME to the computation of minimum weight independent sets with length constraints, in special interval graphs. This intermediate problem which might also be of independent interest, allows us to derive an $O(r(\min\{k, r-k\})$ time algorithm for SME.

Section 7 discusses some extensions of SME to real number sequences and streams. Here we only propose some ways to use W-trees and dynamic programming, but we do not prove complexity results. Section 8 concludes the paper with some other open problems.

2 Runs and Exceptions

We collect a few simple properties of an optimal solution to SME. We call a segmentation *alternating* if the designated symbols of any two neighbored segments differ. A *run* in string S is a maximal contiguous substring with only one sort of symbol. We call a segmentation *regular* if every segment is a concatenation of runs. Throughout the paper, r is the number of runs in S.

Example: $S = 001010001110011$ has $r = 8$ runs. 00101000|111|0011 is some regular segmentation of S in 3 segments. With $0, 1, 0$ as their designated symbols, we have an alternating segmentation with 4 exceptions.

Lemma 1 *There exists an optimal segmentation which is alternating and regular, and where any symbol at an end of a segment is different from the designated symbol of the adjacent segment.*

Proof. In a non-alternating segmentation we could merge two neighbored segments having the same designated symbol, without creating new exceptions.

Next consider, in an alternating segmentation, any two neighbored segments $S[i..j]$ and $S[j+1..m]$., without loss of generality with designated symbol 0 and 1, respectively. Assume, e.g., that $S[j] = 1$, the designated symbol of the segment to the right. Then we could move the border and replace the two segments with $S[i..j-1]$ and $S[j..m]$, thereby reducing the number of exceptions. Hence we can always suppose $S[j] \neq 1$, and similarly $S[j+1] \neq 0$, in some optimal solution. We remark that, if $i = j$, the left segment becomes empty, in which case the number of segments decreases as well, and perhaps further segments with identical designated symbols can be merged.

Finally, if a segmentation with the above properties cuts through some run, then the symbol in the run is different from the designated symbols of two neighbored segments. Hence every position in the run is an exception in both segments, and we can move borders without changing the number of exceptions, until the run is entirely in one segment, or a segment disappears. □

Based on Lemma 1, straightforward dynamic programming on the sequence of runs gives:

Theorem 2 *SME can be solved in $O(br \cdot \min\{k, r-k\})$ time.*

Proof. First consider the case $k < r-k$. Let $E(i, j, s)$, with $i \leq k$, $j \leq r$, and symbol s, indicate the optimal number of exceptions in an alternating, regular segmentation of the first j runs into exactly i segments, with s as the designated symbol of the jth run. (By Lemma 1 we need to consider such segmentations only.) For $i > j$ let $E(i, j, s) = \infty$. It is easy to initialize $E(i, 1, s)$ for any i, s. For the $O(bkr)$ bound, we show for each $j > 1$ how to compute all $E(i, j, s)$ from all $E(i, j-1, s)$ in $O(bk)$ time. Let the jth run consist of l_j copies of symbol s_j. The following recursion is obvious:

$$E(i, j, s) = \min\{E(i, j-1, s), \min_{t \neq s}\{E(i-1, j-1, t)\}\} + q_{j,s} l_j,$$

where $q_{j,s} = 0$ if $s = s_j$, and 1 otherwise. For any fixed i we apply it simultaneously to all symbols s as follows. Find $\min_t\{E(i-1, j-1, t)\}$ in $O(b)$ time. The minimum is attained by some t_0, ties are broken arbitrarily. For each $s \neq t_0$ we have $\min_{t \neq s}\{E(i-1, j-1, t)\} = E(i-1, j-1, t_0)$, and $\min_{t \neq t_0}\{E(i-1, j-1, t)\}$ is also found in $O(b)$ time. Hence we can compute the second term in the recursion in $O(b)$ time for all s together, for any fixed i, and then we can evaluate the recursion in $O(bk)$ time for all i, s.

Case $k \geq r - k$ is similar, but now we let i count the runs that continue the preceding interval, and adjust the recursive formula for E accordingly. □

Later we will get rid of factor b, after a deeper problem analysis. In the end of this section we rule out another tempting dynamic programming approach to SME that easily comes to mind. Let us scan the given string S from left to right and fix some penalty $p > 0$ for changing the designated symbol, i.e., starting a new segment. Define the *score* of a segmentation as the number of exceptions plus the sum of penalties. A segmentation with minimum score is computable in $O(br)$ time: For each symbol s in the alphabet and each prefix of S, maintain a segmentation of this prefix, with s as the last designated symbol, and minimum score. (The $O(b)$ calculations needed for extending the prefix by a new run are obvious.) The catch is that we must find a penalty p which gives the desired number k of segments. It seems that binary search is inevitable, and the number of search steps needed is not clear. The simple penalty approach looks promising at first glance, but it does not give us an $o(brk)$ time algorithm, although typically we may succeed with fewer than k search steps.

3 Greedy Algorithm for Binary Strings

By Lemma 1 it suffices to consider regular segmentations which do not cut any run in S. We may suppose that S is already given as the sequence of lengths of its r runs, computed in a trivial $O(n)$ time preprocessing.

In a regular segmentation, we call a run x an *exception run* if the symbol in x differs from the designated symbol of the segment x belongs to. By Lemma 1, every instance (S, k) of SME-Binary has an optimal segmentation which is alternating and regular, and where no segment begins or ends with an exception run. (However, the first and last run of S may be exception runs.) Thus, no two exception runs are adjacent. Conversely, any set X of pairwise non-adjacent runs describes a segmentation with X as the set of exception runs. Thus we can henceforth characterize a segmentation simply by the set X of exception runs.

We refer to the first and last run of S as *outer runs*, while all others are *inner runs*. Addition of a new exception run x to X (not adjacent to any previous member of X) obviously reduces the number of segments by 2 if x is an inner run. The number of segments is reduced by 1 if x is an outer run. In order to get rid of this case distinction, we decide already in the beginning which of the two outer runs be in X. For any instance (S, k) we have only two options for this initial choice: If $r - k$ is even, either no or both outer runs must be in X. If $r - k$ is odd, exactly one outer run must be in X, either the first or the last one. After this bifurcation it remains to decide which inner runs be in X. We proceed as follows in each case: We remove the outer runs put in X from the instance. Thus, the neighbors of these removed outer runs become new outer runs, moreover they cannot be added to X anymore. Hence, this preprocessing on (S, k) yields two problem instances where *every* exception run is an inner run (and thus decreases the number of segments by

2). This way we have reduced our instance of SME-Binary to two instances of the following problem:
Given a binary string and a number m, find m pairwise non-adjacent inner runs of minimum total length.
The solution to (S, k) is finally obtained from optimal solutions to these two instances: Add the lengths of runs that were initially put in X, and take the minimum sum. As for m, we obviously have:
$m = (r-k)/2$ if $r-k$ is even, and the outer runs are not in X,
$m = (r-k-2)/2$ if $r-k$ is even, and the outer runs are in X,
$m = (r-k-1)/2$ if $r-k$ is odd.

The latter problem can be further rephrased as a "graph" problem, where the graph is merely a path: We represent every run as a vertex, weighted by the length of the run, and join any two neighbored runs by an edge. We call the first and last vertex of this path *outer vertices*, the others are *inner vertices*. The weight of vertex v, that is, the length of the run it represents, is denoted $l(v)$. By $l(X)$ we denote the total weight of a set X of vertices. By the *length* of a path we mean the number of vertices. (This should not be confused with lengths in the string.) What we have shown above is, in this notation:

Lemma 3 *As instance of SME-Binary is linear-time reducible to two instances of the following problem: Given a path P with r weighted vertices and some number $m < r/2$, compute a minimum weight independent set X with exactly m vertices, avoiding the two outer vertices of P.* □

Example: The exception runs (corresponding to elements of X) in the following segmentation (with designated symbols $0, 1, 0$) are printed in bold: 00**10**1000|111|00**11**. The outer runs are the first 00 and the last 11.

In order to split $S = 001010001110011$ into $k = 2, 4, \ldots$ segments, we either forbid the outer runs to be in X and consider a path with vertex weights $(\infty, 1, 1, 1, 3, 3, 2, \infty)$, or we put both outer runs in X and shorten the path to one with vertex weights $(\infty, 1, 1, 3, 3, \infty)$, where ∞ stands for a huge constant (see below).

In the following we use the "graph-theoretic" formulation of the problem and still call it SME-Binary, and we denote the number of vertices by r, this will not cause confusion. Since the weights of the outer vertices are irrelevant, we may set them to a huge constant. These two dummy vertices will simplify the presentation of the algorithms. We also remark that the vertex weights need not be integer anymore.

Greedy Algorithm for SME-Binary:
(1) Do the following m times. Merge an inner vertex v with minimum $l(v)$ and its neigbors u, w into a new vertex z (adjacent to the other neighbors of u, w), with $l(z) := l(u) + l(w) - l(v)$.
(2) After termination of (1) we have obtained a path of vertices labeled with odd-length subpaths of the original path, in the obvious sense. Output

as members of X all original vertices which have even positions in these subpaths.

Clearly, the segments of our segmentation correspond to the vertices of the path produced in (1), and these segments have always odd length. In the following we prove correctness of the greedy algorithm. Trivially, X is an independent set. Induction on m easily shows $|X| = m$. It remains to prove that $l(X)$ is minimized among all independent sets of size m.

Lemma 4 *Let v be some inner vertex with minimum $l(v)$. There exists an optimal solution X that contains either v or both its neighbors u, w.*

Proof. Let X be a solution with $v \notin X$. If also $u, w \notin X$, then replace any vertex in X with v and obtain another solution that is no worse. If $u \in X$ but $w \notin X$, replace u with v (the other case is symmetric). □

Theorem 5 *The greedy algorithm solves SME-Binary correctly and can be implemented to work in $O(r + (r-k)\log(r-k))$ time.*

Proof. We show that the first step of (1) transforms a given instance (P, m) into an equivalent instance $(Q, m-1)$, where Q is the path obtained from P by merging u, v, w into z. Consider any solution X to (P, m) that enjoys the property of Lemma 4. Let Y be the solution to $(Q, m-1)$ which contains all vertices of X other than u, v, w, and where $z \in Y$ iff $u, w \in X$. Since X consists of inner vertices only, so does Y. We also have $|Y| = m-1$ and $l(Y) = l(X) - l(v)$. Conversely, for any solution Y to $(Q, m-1)$, let X be the solution to (P, m) which contains all vertices of Y other than z, and where $u, w \in X$ if $z \in Y$, and $v \in X$ if $z \notin Y$. Note that X has the property as in Lemma 4, $|X| = m$, $l(X) = l(Y) + l(v)$, and X consists of inner vertices only (since Y does). This correspondence makes the problem instances (P, m) and $(Q, m-1)$ equivalent: By our definition of $l(z)$, the objective function is shifted in both directions by the same amount $l(v)$. In particular, the transformations turn an optimal X into an optimal Y and vice versa.

By induction, loop (1) eventually yields an equivalent trivial instance $(R, 0)$ with $\emptyset$ as the only solution. In order to get back an optimal solution to (P, m) we may trace back (1), expanding in each step the vertices merged by (1) and applying the above transformation. We show that (2) gives the same result in a simpler way. This is done by induction on the (odd) length of any subpath of P merged into one vertex of R: A subpath of one vertex has no vertices with even positions, and after any expansion of a vertex into three, the solution still contains exactly the vertices with even positions in their respective subpaths. This proves the correctness.

Phase (2) is done in $O(r)$ time. Phase (1) may be implemented with a doubly linked list for the vertices and a priority queue that returns the minimum and supports deletions and insertions in logarithmic time (in the maximum size of the queue) per operation. Note that the priority queue is needed only for the $m \approx (r-k)/2$ smallest weights. □

Example: A path with vertex weights $(\infty, 4, 1, 2, 1, 1, 3, 3, 2, \infty)$ is successively transformed as follows, if we (arbitrarily) prioritize the rightmost smallest weight in each step: $(\infty, 4, 1, 2, 3, 3, 2, \infty)$, $(\infty, 5, 3, 3, 2, \infty)$, $(\infty, 5, 3, \infty)$, (∞, ∞).

The greedy algorithm solves SME-Binary faster than dynamic programming (Theorem 2) if $k = \omega(\log r)$. Even better, in the next section we use it as a base to develop an $O(r)$ time algorithm.

Correctness of the greedy algorithm has some interesting structural consequences: Recall that, once the status of the outer runs (being an exception run or not) is decided, the parity of k is decided as well, and the optimal solutions are "nested" in the sense that some optimal solution with $k + 2$ segments is obtained from some for k segments by inserting new borders, preserving the existing ones. This is in general not true for numbers k with different parities:

Example: Consider the string $S = 00110001$. The only optimal segmentation with $k = 2$ is $0011000|1$, and the only optimal segmentation with $k = 3$ is $00|11|0001$ which has completely different borders.

Moreover, for the trade-off between the number of segments and exceptions we observe:

Corollary 6 *For any fixed status of the outer runs, let $x(k)$ be the optimal number of exceptions in a segmentation with k segments. (Note that argument k is restricted to either the even or the odd numbers.) Then, function $x(k)$ is monotone decreasing and convex.*

Proof. Both properties follow from correctness of the greedy algorithm (Theorem 5: Each step reduces k by 2, increases the number of exceptions by the weight of the middle vertex of the merged triple, and these weights increase. □

This sheds new light on the penalty algorithm for SME-Binary. First of all, we can force the penalty algorithm to output an even/odd number of segments, simply by appending an empty segment if the wrong parity is obtained. That is, we add another p to the score in that case. Recall that the score of a segmentation with k segments and x exceptions is $x + pk$. Convexity from Corollary 6 and simple geometric reasoning implies that, for any k of the considered parity (even or odd) there exists an *integer* p so that an optimal segmentation with k segments has also the optimal score with respect to penalty p. Thus, binary search can be restricted to integer p, but still this gives no hint which p yields the desired k. Actually, our optimal algorithm is very different: Next we develop an algorithm that achieves linear time with help of a special data structure.

4 Tree-Based Linear-Time Algorithm for Binary Strings

Remember that, after some preprocessing, an instance of SME-Binary is represented as a path of weighted vertices $(v_1, \ldots, v_r)$. We refer to it as the given string. Since $l(v_1), l(v_r)$ do not matter, we may set them to a huge number, e.g., larger than the sum of all other weights. Then $l(v_1), l(v_r)$ remain always maximal, even after any sequence of merge operations. In the following we build our supporting data structure.

W-trees: We multiply the weights of vertices in their given order alternatingly by -1 and $+1$, the start sign is arbitrary. Distinguish carefully between these *signed weights* and their absolute values, simply called *weights* further on. We will sometimes compare disjoint segments by their total (signed) weights. For tie-breaking when equality occurs, we apply some arbitrary priority rule. (For example, the left interval is always considered the "smaller" one in such cases.) In the sequence of weights of vertices, a weight is a local maximum (minimum) if it is larger (smaller) than the weights of both neighbors to the left and to the right.

Definition 7 *An ordered set of at least three vertices is a* W-segment *if the weights of all vertices are, alternatingly, local maxima and minima in the segment, where the weight of the first and last vertex, respectively, is larger than the weight of its only neighbor in the segment.*

Example: The maximal W-segments in $(\infty, 1, 4, 1, 3, 3, 2, \infty)$, adopting the above priority rule, are $(\infty, 1, 4, 1, 3)$ and $(3, 2, \infty)$. The signed weights are $(+\infty, -1, +4, -1, +3, -3, +2, -\infty)$, if we start with "+".

The name W-segment is inspired by the "zigzag" up-and-down pattern of weights. Now we construct a rooted and ordered tree which we call a *W-tree* of the string. To avoid confusion, we speak of *nodes* rather than vertices in the tree. The term *ordered tree* means that a left-to-right order is defined among the children of any node. This naturally induces an order on the leaves. A *sibship* is the orderd set of all children of some node.

Definition 8 *A rooted and ordered tree with signed and weighted nodes is called a* W-tree *of a string if it enjoys the following properties.*
(i) The leaf nodes correspond to the vertices in the given string, in the given left-to-right order and with the given signed weights.
(ii) The signed weight of any non-leaf is the sum of signed weights of its children.
(iii) The ordered set of children of any node is a W-segment (except perhaps the root which may have exactly two children).
(iv) The weight of a node is no larger than the weight of its parent.
(v) Every node (except the leaves and perhaps the root) has an odd number of children. In every sibship of odd size, we call a node odd/even *if it has an odd/even position in the sibship (where the leftmost node has number 1,*

etc).
(vi) The weight of an even node is never larger than the weights of its (odd) neighbored siblings.

Properties (v) and (vi) follow from (iii), but we stated them explicitly for later use.

Lemma 9 *We can construct a W-tree of a weighted path in $O(r)$ time.*

Proof. We give an algorithm that constructs a W-tree. At any moment we maintain an ordered set of trees and consider the sequence of their roots in their natural left-to-right order. Initally, all these trees consist of only one node, hence the sequence of roots is just the given string.

In every step we take an arbitrary W-segment of current roots and connect them to a *new* root, the signed weight of which is defined according to (ii). In the new sequence of roots, the signs of weights still alternate, this is obvious from Definition 7. Thus we can iterate this step. Since the two outermost roots have huge weights, we find some W-segment in every step, i.e., the process stops only when everything is merged into one tree (r odd) or two trees (r even). In the latter case we connect the two huge-weight roots again to a new root.

We obtain in fact a W-tree: (i) holds since the sequence of leaves and their signed weights are never changed. Properties (ii) and (iii) are true by construction. These and Definition 7 yield (iv), and the rest follows from (iii).

Finally we argue that the construction can be done in $O(r)$ time. The current sequence of roots is stored as a doubly-linked list to support fast local changes. Observe that some W-segment is found around an arbitrary local minimum in the sequence of weights. More precisely, a local minimum together with its two neighbors form a W-segment (which might be extended by pairs of further elements in both directions, if their weights satisfy Definition 7). Adding the signed weights and replacing a W-segment with a new root costs $O(1)$ time per node. By storing all local minima separately in a queue, we can take a local minimum in each step in $O(1)$ time. This is correct because each local minimum is preserved until we use the corresponding node in a W-segment: Since, by (iv), a parent's weight is not smaller than the children's weights, the weights of the neighbors of any local minimum can only increase. In summary, every node is processed $O(1)$ times. Since inner nodes of the W-tree have at least three children, the W-tree has size $O(r)$, which gives the $O(r)$ time bound. □

Tracing: Now we "trace" our greedy algorithm for SME-Binary: For every step we update our W-tree so as to obtain a W-tree of the reduced string.

Recall that the greedy algorithm merges some inner vertex v with minimum $l(v)$ and its neighbors u, w, and assigns the merged vertex the weight $l(u)+l(w)-l(v)$. Due to (iv) and (vi), we may always choose an even leaf as v. Let u' and w' be the siblings of v next to the left and right, respectively, in the W-tree. Either we have $u' = u$, or u' is an ancestor of u, and similarly

with w' and w. Moreover, u', w' are odd nodes. We transform the W-tree step by step as follows in the resulting cases.

(1) *None of u', w' is a leaf:* Then merge u' and w' to a new odd node z'. The children of z' are, from left to right, the children of u', node v, and the children of w'. The signed weight of z' is the sum of signed weights of u', v, w'.
(2) *Exactly one of u', w' is a leaf, say w' (the other case is symmetric):* Then u' adopts v and w' as two new rightmost children. To the signed weight of u' add the signed weights of v and w'.
(3) *Both u' and w' are leaves:* Then merge u', v, w' to a leaf z'. The signed weight of z' is the sum of signed weights of u', v, w'.

These steps are repeated, with updated u', w', until case (3) appears. Furthermore, whenever a node retains only one child, parent and child have the same signed weight, and we identify them immediately by contracting the edge.

The procedure will always terminate, as v "goes down the tree". Check that, due to the choice of v, every step preserves properties (i)-(vi) of a W-tree. Hence, upon termination we get a W-tree of the updated sequence, as claimed.

The following properties of the tracing procedure are crucial: An even node remains even until it is merged with its neighbored odd siblings into a new odd node. Odd nodes are merged into odd nodes only. Hence an even node never changes its (signed) weight until it disappears. The even node that disappears next is always one with smallest weight. It follows that the even nodes are processed in their fixed order of increasing weights. Hence, after any number of steps of the greedy algorithm, the W-tree obtained by tracing contains exactly those even nodes whose weights are above some threshold *thr*, plus certain odd nodes.

Getting the high nodes: Given a threshold *thr*, we call a node *high/low* if its weight is larger/smaller than *thr*. The following procedure *Extract(thr)* constructs, from the initial W-tree of the given string, the W-tree obtained by tracing until weight *thr*, but without actually performing the tracing steps.

Extract(thr):
(1) Retain the high even nodes and all their ancestors and siblings. Delete all other nodes.
(2) In every sibship that contains high even nodes, do the following. Between any two high even nodes, and on the left/right side of the leftmost/rightmost high even node in the sibship, merge all siblings into one node and and let the signed weight of the new node be the sum of signed weights of merged siblings.

Lemma 10 *Procedure* Extract(thr) *computes, in $O(r)$ time, a W-tree of the segmentation obtained by running the greedy algorithm for SME-Binary as long as the weight of the middle vertex in a merging step does not exceed* thr.

Proof. Every low even node v becomes a leaf during tracing. At this moment, v together with its two neighbored siblings is replaced with one new node in the sibship. It follows that all siblings between the high even nodes are eventually merged. Furthermore, invariant (ii) holds, so that adding the signed weights is also correct. Altogether it follows that *Extract(thr)* and tracing yield the same tree. The time bound is obvious. □

Once we have the W-tree, it is easy to reconstruct the segmentation in linear time, since its leaves represent the segments. Since the greedy algorithm is correct by Theorem 5, it follows that we obtain an optimal segmentation.

Number of even nodes and final result: To get out the desired number k of segments we have to use the appropriate *thr* in *Extract(thr)*.

Lemma 11 *A W-tree with e even nodes has $k = 2e+1$ leaves if the root has an odd number of children, and $k = 2e+2$ leaves if the root has two children.*

Proof. It suffices to show the first part. Then the second part follows immediately. We proceed by induction. The Lemma is true if the W-tree is just a root with an odd number of leaves as children. If we make some leaf the parent of a new sibship of, say, $2l+1$ nodes, then e and k increases by l and $2l$, respectively. □

By Lemma 11, all we need to know is the $(k-1)/2$ or $(k-2)/2$ even nodes with largest weights, and then we easily get a W-tree of an optimal segmentation into k segments. Summarizing the process, we can now state the main result. Regarding linear-time selection see, e.g., [11], or common textbooks on algorithm design. *Selection* means to determine a prescribed number of largest elements in an unsorted set of numbers.

Theorem 12 *SME-Binary can be solved in $O(r)$ time for any k.*

Proof. From the given string, given as the sequence of lengths of runs, build a W-tree as in Lemma 9. By selection, mark the number of high even nodes specified in Lemma 11. Let *thr* be the smallest weight of the high even nodes. Compute a W-tree of an optimal segmentation by *Extract(thr)* as in Lemma 10, and from the leaves of this W-tree straightforwardly recover the segmentation. □

There remain some practical issues. Selection in worst-case linear time suffers from a large hidden constant. We may use randomized selection with pivot elements and content ourselves with expected linear time. (However, in practice this should be faster than a deterministic selection algorithm. For recent results on selection see also [22].) In case $k \ll r$ we may circumvent selection algorithms and determine the high even nodes in time $O(r + k \log k)$ (with small hidden constant) using a priority queue. We would consider the even nodes of the W-tree at increasing distance from the root. Due to (iv)

we can ignore the subtree below a current node x if $l(x)$ is already smaller than the kth largest weight found so far. Thus we may typically need less than $O(r)$ time to determine the high even nodes.

The algorithm is easy to apply, as the segment borders correspond to subtrees rooted at the high even nodes.

Example: Suppose $S = 001010001110011$, k is even, and the outer runs are not exception runs. The dendrogram-like table shows a W-tree, drawn upside down. Note that the huge-weight dummy vertices before and after the string are not displayed. The even nodes (printed in bold) with largest weights yield the following segment borders for $k = 2, 4, 6$: 00101000|1110011, 00101000|111|00|11, 00|101|000|111|00|11. The other cases (where both outer runs are exception runs, or k is odd) give worse solutions for this string.

|	−2	|	+1	|	**−1**	|	+1	|	−3	|	+3	|	**−2**	|	+2 |
|		|			**+1**			|			|			**+3**		|
|						−4						|					|

5 A Parameterized Weighted Independent Set Problem

The graph problem specified in Lemma 3 is interesting in itself, for general graphs rather than just paths. For clarity we state the definition again:

Cardinality Minimum Weight Independent Set (CMWIS)
Given a graph and an integer m, compute a minimum weight independent set with exactly m vertices

Note that *minimum* may also be *maximum* without changing the problem, since m is fixed. By Theorem 5, CMWIS on paths of length r is solvable in $O(r)$ time. Slightly more generally, CMWIS is linear for graphs with maximum degree $d = 2$. By way of contrast we have:

Theorem 13 *CMWIS is NP-complete for graphs of any fixed maximum degree $d \geq 3$.*

Proof. We reduce Maximum Independent Set for degree $d \geq 3$ to CMWIS: Assign unit weights to every vertex. An independent set of size *at least* m exists iff an independent set of size *exactly* m exists. □

Due to this hardness result it is sensible to consider parameterized algorithms (see [12, 30] for introductions). CMWIS is (probably) not FPT with parameter m alone: If all weights are equal, we have to figure out existence of an independent set with m vertices, but this is known to be $W[1]$-complete. However, the same problem with dual parameter $n - m$ is a variation of Weighted Vertex Cover, thus it is FPT. Next we show that CMWIS is FPT in combined parameters m and d. Generalizing the observation from Lemma 4 we obtain:

Theorem 14 *CMWIS on graphs of maximum degree d can be solved in $O^*(x^m)$ time, where $x = (\sqrt{2d^2 - 2d + 1} + 1)/2 \approx d/\sqrt{2}$.*

Proof. We give a branching algorithm. Whenever a new vertex has been added to the independent set, remove this vertex and its neighbors from the graph, and set $m := m - 1$. Consider any rest graph (after removal of chosen vertices and their neighbors) with $m > 0$. Let v be a vertex with minimum $l(v)$. In one branch, add v to the solution. In all other branches (where v is not taken), we can add *at least two* of v's neighbors to the solution, by the following reasoning: If only one neighbor u is taken, we can replace u with v in any solution and get an independent set with smaller weight. Assume that no neighbor of v is taken. Let w be any vertex of the rest graph that belongs to a solution, w exists since $m > 0$. But now we can replace w with v and get an independent set with smaller weight. Altogether, we reduce m by 1 in one branch and by 2 in at most $\binom{d}{2}$ branches. Characteristic equation $x^2 = x + d(d-1)/2$ yields the claimed base x. □

We remark that CMWIS has been solved in [9] for the more general class of d-degenerate graphs, however the time as a function of the parameters is $O^*(2^{dm})$ there.

In Theorem 14, the worst case of $\binom{d}{2}$ branches with 2 chosen vertices appears only if v has degree exactly d, and all neighbors of v are pairwise non-adjacent. Moreover, we can delete further vertices during a branching step as follows. We can arrange the neighbors of v in an arbitrary order and decide on the first two neighbors of v to be added to X. Then we can remove, in every branch, those neighbors of v appearing before the second chosen vertex in this order. This heuristic does not help our worst-case upper bound, but it diminishes the rest graphs faster, in a simple way. Heuristics like these, together with kernelization, may lead to smaller time bounds in further research. A small problem kernel is very easy to obtain:

Theorem 15 *CMWIS has a polynomial-time computable kernel of at most $(d+1)(m-1)+1$ vertices.*

Proof. We claim that the set K of the $(d+1)(m-1)+1$ vertices with smallest weights (ties are broken arbitrarily) contains an optimal solution. To prove the claim, consider any X, $|X| = m$, such that $K \cap X$ is independent and a vertex $v \in X \setminus K$ exists. At most $m-1$ vertices of X are in K, each having at most d neighbors in K. Hence some $u \in K$ exists that is neither in X nor adjacent to any vertex of $K \cap X$. Since $l(u) \leq l(v)$, replacing v with u in X can only reduce $l(X)$, and $K \cap X$ is still independent. In particular, if we start from an independent set X, some iterations give an independent set of smaller or equal weight, contained in K. □

6 More than Two Symbols

Since Lemma 1 holds for arbitrary alphabet size b, SME can be rephrased similarly as SME-Binary. However, for $b > 2$, sets X corresponding to the

exception runs in regular segmentations are no longer restricted to independent sets of a specific size, and the characterization of (optimal) solutions X becomes more complicated.

We refer to any maximal subpath in X as a *block* of X. The length of a block is the number of vertices. (In case $b = 2$, all blocks had length 1.) Every block is adjacent to one or two vertices not in X. We say that the symbols assigned to these vertices are adjacent to the block. Hence, every block is adjacent to one or two different symbols. The following lemma is easily proved by arguments similarly as in Lemma 1.

Lemma 16 *There exists an optimal segmentation with the following properties: The segmentation is regular, moreover, each block of exception runs belongs entirely to a segment, and symbols in a block are always distinct from the symbols adjacent to this block.* □

Suppose that we add a block B to X, where B is not adjacent to any block in X. This reduces the number of segments by the length of B, if B has two different adjacent symbols or contains an end vertex of the path. It reduces the number of segments by the length of B plus 1, if B has two adjacent vertices that carry the same symbol (because we can merge two neighbored segments with the same designated symbol). In the latter case we say that B has a *bonus* vertex, in addition to its real vertices. With this notation, we have:

Lemma 17 *SME is equivalent to the following problem: Given a path of r vertices, each equipped with a positive weight and a symbol, so that, without loss of generality, neighbored vertices carry different symbols, and an integer k, compute a minimum weight set X such that the total number of vertices, including bonus vertices, in X is at least $m := r - k$, and symbols adjacent to a block do not occur inside the block. (A block B of contiguous vertices in X is said to have a bonus vertex if B is flanked by two vertices outside X with equal symbols.)* □

This reformulation may appear artificial, but actually this problem is a special case of some multicriteria independent set problem in interval graphs, and this fact will give us an efficient algorithm. Interval graphs are well-known as intersection graphs of families of intervals on the real axis. In the following we consider families of open intervals (not including their endpoints) whose endpoints have integer coordinates. The length of an interval is defined as usual.

Fixed Length Minimum Weight Independent Set (FLMWIS)
Given a family of F open intervals whose endpoints are integers in $[0, R]$, where every interval has a positive weight, and an integer $M \leq R$, compute a minimum weight independent set, i.e., a subset of pairwise disjoint intervals, with total length at least M.

Let $K := R - M$ be the total number of unit intervals in $[0, R]$ allowed to be outside the solution.

Theorem 18 *FLMWIS can be solved in $O((R+F)\cdot\min\{M,K\})$ time, provided that the intervals are already sorted by their right endpoints.*

Proof. We apply dynamic programming, treating the intervals in the order of increasing coordinates of right endpoints. First we prove the bound $O((R+F)M)$. For $i \leq M$ and $j \leq R$, let $w(i,j)$ be the weight of an optimal solution with length at least i, for the instance restricted to $[0,j]$. (That is, only intervals which are subsets of $[0,j]$ are considered.) If no such solution exists, let be $w(i,j) = \infty$. Initialization for $j = 0$ is trivial. When we step to the next j, we first set $w(i,j) := w(i,j-1)$ for all i, and then treat every interval ending at j. Appending interval (l,j) to a partial solution may improve some of the $w(i,j)$. Clearly, we simply have to add $w(i-(j-l),l)$ and the weight of (l,j), compare the sum to the current $w(i,j)$, and take the minimum. Thus we need $O(M)$ time for each endpoint and interval.

The $O((R+F)K)$ bound is achieved similarly, we only state what is different. For $i \leq K$ and $j \leq R$, let $w(i,j)$ be the weight of an optimal solution with length at least $j-i$, for the instance restricted to $[0,j]$. (That is, i bounds the number of unit intervals which are not part of the solution. It suffices to consider $i \leq K$, since we want to leave out at most K unit intervals in total.) When we step to the next j, we first set $w(i,j) := w(i,j-1) + 1$ for all i, corresponding to the case that no interval with endpoint j is inserted. Appending interval (l,j) to a partial solution may improve some of the $w(i,j)$. This time we add $w(i,l)$ and the weight of (l,j), compare the sum to the current $w(i,j)$, and take the minimum. The time is $O(K)$ for each endpoint and interval. □

Theorem 19 *SME can be solved in $O(r\cdot\min\{k,r-k\})$ time.*

Proof. From an instance of SME we construct a family of intervals in $[0,R]$, $R = 2r$, as follows. Vertices $1,2,\ldots,r$ of the given path correspond to *short* intervals $(0,2),(2,4),\ldots,(2r-2,2r)$. For any two vertices l and j ($l<j$) such that l,j have the same symbol s, but all symbols between l and j are different from s, we introduce the *long* interval $(2l-1,2j-1)$. (Terms *long* and *short* are only used for reference purposes.) The weight of a short interval is the weight of the corresponding vertex in the given path. The weight of a long interval is the sum of weights of vertices between l and j. Finally, define $M := 2m$.

From the intervals in a solution to FLMWIS we construct a solution X to SME (in the formulation of Lemma 17) as follows. (1) For every long interval as specified above, vertices between l and j are put in X, to form a block with bonus vertex, called a *bonus block*. For every short interval, the corresponding vertex is also put in X. (2) If some block B of vertices coming from short intervals contains vertices with symbols adjacent to B, we remove one such vertex from X. This step is repeated as long as possible.

We prove correctness of this reduction. In the following, *vertices in X* refers to both real and bonus vertices. Due to Lemma 17, a solution to SME consists of blocks such that symbols adjacent to a block do not occur inside the block. Now represent bonus blocks by long intervals, and vertices in other

blocks by short intervals. If X has at least m vertices, this gives a solution to FLMWIS with length at least $2m = M$ and the same weight as X. Note that the requested long intervals actually exist in our family of intervals, and that all selected intervals are disjoint.

Conversely, consider any solution to FLMWIS with length at least $M = 2m$. We claim that X constructed above has at least m vertices and at most the same weight. This implies that X is optimal.

The unit intervals $(2l-1, 2l)$ and $(2j, 2j-1)$ attached to every long interval (from l to j) have two roles. First, they ensure that vertices adjacent to the obtained bonus block in X are actually not in X. Second, they account for the bonus vertex, thus every vertex in X is now represented by two unit intervals, and $M = 2m$ yields that X has at least m vertices as desired. Also, X has so far the same weight as the FLMWIS solution.

Consider any block B of vertices coming from short intervals, containing vertices with symbols adjacent to B. Removal of such a vertex from X is compensated by one more bonus vertex, and the weight of X decreases. This proves the claim. Moreover, if the solution to FLMWIS was optimal, this case cannot appear, i.e., we can skip phase (2) of the reduction.

Thus, the actual algorithm is to construct the family of weighted intervals as above, solve FLMWIS to optimality as in Lemma 18, and get X simply by applying phase (1). We have r short intervals. The crucial point is that we also have $O(r)$ long intervals, since every vertex of the path is the right neighbor of at most one long interval. More precisely, $F < 2r$, and $R = 2r$, $M = 2m$, $K = 2k$ is clear by construction. For computing the weights of long intervals we have to sum up the weights between any two consecutive occurences of a symbol. This is easily managed in $O(r)$ time for all symbols in the alphabet: Compute all prefix sums of weights in the given path, and hence the start and end position of each run in the string. For every symbol in the alphabet, create the list of runs where this symbol occurs. Finally, get the weight of every long interval by one subtraction. □

Hence we need $O(kr)$ time in the "statistically relevant" case $k < r/2$. An open question is whether the factors can be further improved, as in SME-Binary, perhaps by an approach different from dynamic programming.

7 Binary Segmentations with Thresholds

In some applications as mentioned in Section 1, a sequence of n real numbers is given, and we want a segmentation into k segments each consisting of only large or only small numbers, with exceptions minimized. Large and small numbers, symbolized by 0 and 1, are separated by some threshold. That is, we have up to $n - 1$ instances of SME-Binary, for all (essentially different) thresholds. An appropriate threshold may be given by the application, but we may also be unsure about the threshold. A natural question is: How difficult is it to solve all resulting instances of SME-Binary? A naive method treats all $O(n)$ instances independently, using $O(n)$ time for each (since $r \leq n$). Sorting the given numbers costs $O(n \log n)$ time, hence we get away with $O(n^2)$ time.

However, it is intuitively clear that slight changes of the threshold mostly will not change the segmentations dramatically. This suggests to use W-trees more directly for this extended problem.

As the threshold grows, symbols 1 are successively turned into 0. Every switch affects at most three neighbored leaves in the W-tree. (One leaf is split in three, or a leaf of weight 1 disappears and its neighbors get merged, or one symbol moves to a neighbored leaf.) We also have to update the signed weights of ancestors of affected leaves, on a few rooted paths, where signed weights of inner nodes change by at most 2 units. We need to recompute only parts of the W-tree around these rooted paths, where the sibships may not be W-segments anymore. W-segments can be repaired in bottom-up direction, by local rearrangements. We do not go further into details, as we cannot prove a complexity result here. The difficulty is to bound the necessary changes in the W-tree. An amortized analysis seems to be an interesting and nontrivial question for further research.

In a data stream (or online) version of SME, symbols of the string arrive one by one, and we wish to maintain a structure that allows, at every desired moment, to quickly compute optimal segmentations of the prefix seen so far. This problem version could be interesting for permanent monitoring of trends or signals. If a W-tree is used, this leads essentially to the same update problem, where changes are initiated by the rightmost leaf. We have to leave complexity (space and amortized time) as an open question. It may turn out that dynamic programming is better suited for the stream version even if $b = 2$. For general b, obviously, we may apply dynamic programming as well, with a somewhat higher complexity than in Theorem 19, since it is not possible to compute bonus blocks in advance.

8 Conclusions

We devised a tree-based linear-time algorithm for partitioning binary strings into segments with one sort of symbols, so as to minimize the total number of exceptions. We also gave a dynamic programming algorithm with time bound $O(kr)$, for r runs and $k < r/2$ segments, independent of alphabet size b. Perhaps the time can be further improved. An intriguing question is whether similar ideas as for $b = 2$ can beat the dynamic programming approach and take away factor k also for general b. The simple exchange argument of Lemma 6 does not work for $b > 2$, hence already the greedy algorithm (the base of our optimal algorithm for $b = 2$) does not easily generalize. Other open questions concern the complexity of our segmentation problem extended to real-number sequences with varying thresholds, and to data streams.

One could look for problem transformations and data structures to obtain faster optimization algorithms also for other established homogeneity measures. Finally, segmentation should find more real-world applications, e.g., in data mining and automatic classification.

Computing *fixed cardinality minimum weight independent sets* in paths

is essentially equivalent to string segmentation for $b = 2$, and our dynamic programming algorithm for general b actually computes minimum weight independent sets satisfying another minimum size constraint in special interval graphs. The above problem on arbitrary graphs is certainly of independent interest. We gave an FPT result for bounded-degree graphs, and an obvious problem for further research is to improve the time bound.

Acknowledgements

This work was partially supported by the Swedish Research Council (Vetenskapsrådet), grant no. 2007-6437, "Combinatorial inference algorithms – parameterization and clustering".

I would like to thank the referees for various hints.

References

[1] P. Auer, R.C. Holte, W. Maass. Theory and applications of agnostic PAC-learning with small decision trees, In: *Proceedings of the 12th International Conference on Machine Learning ICML 1995*, pp. 21–29

[2] F. Bengtsson. Algorithms for aggregate information extraction from sequences, PhD thesis, Luleå University of Technology 2007

[3] F. Bengtsson, J. Chen. Computing maximum-scoring segments in almost linear time, In: *Proceedings of the 12th Annual International Conference on Computing and Combinatorics COCOON 2006*, *Lecture Notes in Computer Science* 4112, pp. 255–264

[4] F. Bengtsson, J. Chen. Computing maximum-scoring segments optimally, Research report, Luleå University of Technology 2007:3

[5] K.P. Bennett. Decision tree construction via linear programming, In: *Proceedings of the 4th Midwest AI and Cognitive Science Society Conference 1992*, pp. 97–101

[6] K.P. Bennett, O.L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets, *Optimization Methods and Software* 1 (1992), 23–34

[7] K.P. Bennett, O.L. Mangasarian. Bilinear separation of two sets in n-space, *Computational Optimization and Applications* 2 (1993), 207–227

[8] K.P. Bennett, O.L. Mangasarian. Multicategory separation via linear programming, *Optimization Methods and Software* 3 (1993), 27–39

[9] L. Cai, S.M. Chan, S.O. Chan. Random separation: A new method for solving fixed-cardinality optimization problems, In: *Proceedings of the 2nd International Workshop on Parameterized and Exact Compuation IWPEC 2006*, *Lecture Notes in Computer Science* 4169, pp. 239–250

[10] M. Csűrös. Maximum-scoring segment sets, *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 1 (2004), 139–150

[11] D. Dor. Selection algorithms, PhD thesis, Tel-Aviv University 1995

[12] R.G. Downey, M.R. Fellows. *Parameterized Complexity*, Springer 1999

[13] T. Elomaa, J. Rousu. General and efficient multisplitting of numerical attributes, *Machine Learning* 36 (1999), 201–244

[14] T. Elomaa, J. Rousu. On the computational complexity of optimal multisplitting, *Fundamenta Informaticae* 47 (2001), 35–52

[15] U. Fayyad, K.B. Irani. On the handling of continuous-valued attributes in decision tree generation, *Machine Learning* 8 (1992), 87–102

[16] U. Fayyad, K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning, In: *Proceedings of the 13th International Joint Conference on Artificial Intelligence IJCAI 1993*, pp. 1022–1027

[17] P. Fragkou, V. Petridis, A. Kehagias. A dynamic programming algorithm for linear text segmentation, *Journal of Intelligent Information Systems* 23 (2004), 179–197

[18] S. Guha, N. Koudas, K. Shim. Data-streams and histograms, In: *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing STOC 2001*, pp. 471–475

[19] N. Haiminen, A. Gionis. Unimodal segmentation of sequences, In: *Proceedings of the 4th IEEE International Conference on Data Mining ICDM 2004*, pp. 106–113

[20] J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmäki, H. Toivonen. Time series segmentation for context recognition in mobile devices, In: *Proceedings of the 1st IEEE International Conference on Data Mining ICDM 2001*, pp. 203–210

[21] E.J. Keogh, S. Chu, D. Hart, M.J. Pazzani. An online algorithm for segmenting time series, In: *Proceedings of the 1st IEEE International Conference on Data Mining ICDM 2001*, pp. 289–296

[22] K.C. Kiwiel. On Floyd and Rivest's SELECT algorithm, *Theoretical Computer Science* 347 (2005), 214–238

[23] J. Kleinberg, E. Tardos. *Algorithm Design*, Pearson/Addison-Wesley 2006

[24] R. Kohavi, M. Sahami. Error-based and entropy-based discretization of continuous features, In: *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD 1996*, pp. 114–119

[25] J. Kujala, T. Elomaa. Improved algorithms for univariate discretization of continuous features, In: *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases PKDD 2007*, pp. 188–199

[26] J. Löfhede, personal communication

[27] J. Löfhede, N. Löfgren, M. Thordstein, A. Flisberg, I. Kjellmer, K. Lindecrantz. Comparison of three methods for classifying burst and suppression in the EEG of post asphyctic newborns (2007), In: *Proceedings of the 29th IEEE Engineering in Medicine and Biology Society Annual International Conference EMBC 2007*

[28] W. Maass. Efficient agnostic PAC-learning with simple hypothesis, In: *Proceedings of the 7th Annual ACM Conference on Computational Learning Theory COLT 1994*, pp. 67–75

[29] T. Mielikäinen, E. Terzi, P. Tsaparas. Aggregating time partitions, In: *Proceedings of the 12th ACM SIGKDD Conference on Knowledge Discovery and Data Mining KDD 2006*, pp. 347–356.

[30] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Oxford Lecture Series in Mathematics and its Applications, Oxford Univ. Press 2006

[31] E. Terzi, P. Tsaparas. Efficient algorithms for sequence segmentation, In: *Proceedings of the 6th SIAM Conference on Data Mining SDM 2006*