Coresets for Decision Trees of Signals

Ibrahim Jubran
Department of Computer Science
University of Haifa, Israel
ibrahim.jub@gmail.com

Ernesto Evgeniy Sanches Shayda
Department of Computer Science
University of Haifa, Israel
ernestosanches@gmail.com

Ilan Newman
Department of Computer Science
University of Haifa, Israel
ilan@cs.haifa.ac.il

Dan Feldman
Department of Computer Science
University of Haifa, Israel
dannyf.post@gmail.com

Abstract

A $k$-decision tree $t$ (or $k$-tree) is a recursive partition of a matrix (2D-signal) into $k \geq 1$ block matrices (axis-parallel rectangles, leaves) where each rectangle is assigned a real label. Its regression or classification loss to a given matrix $D$ of $N$ entries (labels) is the sum of squared differences over every label in $D$ and its assigned label by $t$. Given an error parameter $\varepsilon \in (0, 1)$, a $(k, \varepsilon)$-coreset $C$ of $D$ is a small summarization that provably approximates this loss to every such tree, up to a multiplicative factor of $1 \pm \varepsilon$. In particular, the optimal $k$-tree of $C$ is a $(1 + \varepsilon)$-approximation to the optimal $k$-tree of $D$.

We provide the first algorithm that outputs such a $(k, \varepsilon)$-coreset for every such matrix $D$. The size $|C|$ of the coreset is polynomial in $k \log(N)/\varepsilon$, and its construction takes $O(Nk)$ time. This is by forging a link between decision trees from machine learning – to partition trees in computational geometry.

Experimental results on sklearn and lightGBM show that applying our coresets on real-world data-sets boosts the computation time of random forests and their parameter tuning by up to x10, while keeping similar accuracy. Full open source code is provided.

1 Introduction

Decision trees are one of the most common algorithms used in machine learning today, both in the academy and industry, for classification and regression problems [52]. Informally, a decision tree is a recursive binary partition of the input feature space into hyper-rectangles, where each such hyper-rectangle is assigned a label. If the labels are given from a discrete set, the trees are usually called classification trees, and otherwise they are usually called regression trees. Variants include non-binary partitions and forests [29].

Why decision trees? Advantages of decision trees, especially compared to deep learning, include: (i) Interpretability. They are among the most popular algorithms for interpretable (transparent) machine learning [51]. (ii) Usually require small memory space, which also implies fast classification time. (iii) Accuracy. Decision trees are considered as one of the few competitors of deep networks. In competitions, such as the ones in Kaggle [36], they are one of the favorite classifiers [10], especially on small or traditional tabular data. (iv) May learn from small training data.

The goal is usually to compute the optimal $k$-tree $t^*$ for a given dataset $D$ and a given number $k$ of leaves, according to some given loss function. In practice, researchers usually use ensemble of trees called forests, e.g., a Random Forest [12], which are usually learned from different subsets of the

Coresets are one of the central methods to facilitate the analysis of large data sets.\cite{47}. Informally, for an input dataset \( D \), a set \( T \) of models, an approximation error \( \varepsilon \in (0,1) \), and a loss function \( \ell \), a coreset \( C \) is a data structure that approximates the loss \( \ell(D,t) \) for every model \( t \in T \), up to

**Challenges.** The motivation for this paper originated from the following challenges:

(i) **Sub-optimality.** Hardness of decision tree optimization is both a theoretical and practical obstacle\cite{31}. It is NP-hard to compute the optimal \( k \)-tree, or its approximation, when the number \( k \) is not fixed.\cite{14,39}. There were several attempts to improve the optimality of decision tree algorithms, from binary-split decision trees as in\cite{6,8}, in a line of work of e.g.,\cite{2,61}. Nevertheless, greedy implementations e.g., CART\cite{40} and C4.5\cite{51} have remained the dominant methods in practice.

(ii) **Computation time.** Due to this lack of optimality, finding a decision tree that provides a good accuracy usually requires many runs, since each of them returns only a local minimum that might be arbitrarily far from the global optimum. The final model usually consists of a forest containing many trees. Popular forest implementations include the famous Sklearn, XGBoost, LightGBM, and CatBoost libraries\cite{49,15,37,19}, which all utilize (as default) an ensemble of at least 100 trees. Moreover, there is a list of dozen parameters to calibrate including: number of trees, depth of each tree, pruning/splitting strategies on each tree and between them, and many others. To this end, the running time for obtaining reasonable results even on moderate size datasets might be impractical.

(iii) **Scalability.** Existing techniques tend not to scale to realistically-sized problems unless simplified to trees of a specific form as stated in\cite{31}.

(iv) **Streaming, parallel, and dynamic updates.** The common algorithms mentioned above do not support continuous learning or updating of the modeled tree when an input sample is either added or removed from the dataset, e.g., when the dataset does not fit into memory or arrives on-the-fly. Similarly, we do not know techniques to train a single tree in parallel on multiple machines.

1.1 Coresets

"Coresets are one of the central methods to facilitate the analysis of large data sets."\cite{47}. Informally, for an input dataset \( D \), a set \( T \) of models, an approximation error \( \varepsilon \in (0,1) \), and a loss function \( \ell \), a coreset \( C \) is a data structure that approximates the loss \( \ell(D,t) \) for every model \( t \in T \), up to
a multiplicative factor of $1 \pm \varepsilon$, in time that depends only on $|C|$. Hence, ideally, $C$ is also much smaller than the original input $D$.

**Why coresets?** The main motivation for constructing a coreset is to compute the optimal model or its approximation, much faster, while sacrificing little accuracy. Furthermore, a coreset for a family of classifiers is many times a “silver bullet” that provides a unified solution to all Challenges (i)-(iv) above. Combining the two main coreset properties: merge and reduce [22, 26, 11], which are usually satisfied, with the fact that a coreset approximates every model, and not just the optimal model, enables it to support streaming and distributed data [11, 11, parallel computation [21], handle constrained versions of the problem [25], model compression [20], parameter tuning [43] and more.

**Coreset construction techniques.** There are many different techniques for coreset construction, ranging from loss-less to lossy, from deterministic to randomized, and from greedy to non-greedy constructions. Examples include accurate coresets [33] and non-accurate coresets [23] via computational geometry, random sampling-based coresets [22, 17, 44], and greedy deterministic coresets via the Frank-Wolfe algorithm [16]. Recently, many works focus on developing frameworks for general families of loss functions, e.g., [22, 57]. We refer the interested reader to the surveys [2, 11, 50, 11, 4, 21] with references therein.

**Practical usage.** Since a coreset is not just another solver that competes with existing solutions, but a data structure for approximating any given model in the family, we can apply existing approximation algorithms or heuristics on the coreset to obtain similar results compared to the original (full) data. Let the coreset be small, we may run these heuristics multiple times, or apply the hyperparameter tuning using the coreset [43], thus reducing the computational burden by orders of magnitude.

**Main challenges:** (i) No coreset. Unfortunately, it is not clear at all that a small coreset exists for a given family of models. In fact, we can conclude from [54] that a coreset for decision trees does not exist in general; see details below. In this case, we can either give up on the coreset paradigm and develop a new solver, or add assumption on the input dataset, instead of targeting every possible dataset that may be very artificial and unrealistic, as the counter example in [54]. In this paper, we choose the latter option. (ii) Unlike, say, uniform sampling, every problem formulation requires a different coreset construction, which may take years of research to design.

### 1.2 First coreset for decision trees and their generalization

In this paper, we tackle a generalized and more complex set of models than decision trees, where, rather than a recursive binary partition, we allow the input feature space $\mathbb{R}^d$ to be partitioned into any $k$ disjoint axis-parallel hyper-rectangles; see Fig. 1. This generalization is essential in order to support future non-recursive and not necessarily binary classification models, e.g., ID3 and C4.5 [30, 51]. To our knowledge, this is the first coreset with provable guarantees for constructing decision trees.

**Definition 1 (k-segmentation).** For an integer $d \geq 1$ that denotes the dimension of the feature space $A = \mathbb{R}^d$, and an integer $k \geq 1$ that denotes the size of the partition (number of leaves), a function $s : A \to \mathbb{R}$ is a $k$-segmentation if there is a partition $B = \{B_1, \ldots, B_k\}$ of $A$ into $k$ disjoint axis-parallel hyper-rectangles (blocks), such that $|\{s(b) : b \in B\}| = 1$ for every block $B \in B$, i.e., $s$ assigns a unique value for all the entries in each of its $k$ rectangles; see Fig. 7. We define the union over all possible such $k$-segmentations by $\text{SEG}_{(k,d)}$.

We now define our loss function, and an optimal $k$-segmentation model over some set $A \subseteq \mathbb{R}^d$.

**Definition 2 (Loss function).** For a dataset $D = \{(x_1, y_1), \ldots, (x_N, y_N)\} \subseteq A \times \mathbb{R}$, an integer $k \geq 1$, and a $k$-segmentation $s \in \text{SEG}_{(k,d)}$, we define the sum of squared error (SSE) loss

$$\ell(D, s) := \sum_{(x,y) \in D} (s(x) - y)^2$$

as the loss of fitting $s$ to $D$. A $k$-segmentation $s^*$ is an optimal $k$-segmentation of $D$ if it minimizes $\ell(D, s)$ over every $k$-segmentation $s \in \text{SEG}_{(k,d)}$, i.e., $s^* \in \text{arg min}_{s \in \text{SEG}_{(k,d)}} \ell(D, s)$. The optimal SSE loss is denoted by $\text{opt}_k(D) := \ell(D, s^*)$.

For example, the optimal 1-segmentation $s^*$ of $D$ is the constant function $s^* \equiv \frac{1}{|D|} \sum_{(x,y) \in D} y$ since the mean of a set of numbers minimizes the sum of squared distances to the elements of the set. Also, $\text{opt}_{|D|}(D) = 0$ for every dataset $D$. 


We are now ready to formally define a coreset for the $k$-segmentation problem (and $k$-decision trees of at most $k$ leaves, in particular).

**Definition 3 (Coreset).** Let $D = \{(x_1,y_1), \cdots , (x_n,y_n)\} \subseteq A \times \mathbb{R}$ be an input dataset. Let $k \geq 1$ be an integer and $\varepsilon \in (0,1)$ be the desired approximation error. A $(k, \varepsilon)$-coreset for $D$ is a data structure $(C,u)$ where $C \subseteq A \times \mathbb{R}$ is an ordered set, and $u : C \rightarrow [0,\infty)$ is called a weight function, such that $(C,u)$ suffices to approximate the loss $\ell(D,s)$ of the original dataset $D$, up to a multiplicative factor of $1 \pm \varepsilon$, in time that depends only on $|C|$ and $k$, for any $k$-segmentation $s$.

**Practical usage.** As defined above and discussed in Section 1.1 a coreset approximates every model in our set of models SEG($k,d$). Hence, a coreset for decision trees is clearly also a coreset for forests with an appropriate tuning for $k$, since every tree in the forest is approximated independently by the coreset. We expect that applying existing heuristics (not necessarily with provable guarantees) such as sklearn [49] or LightGBM [37] on the coreset, would yield similar results compared to the original data. Indeed, our experimental results in Section 6 validate those claims.

**No coreset for general datasets.** Unfortunately, even for the case of $k = 4$ and $A \subseteq \mathbb{R}$, i.e., when the input is simply a one dimensional dataset $D = \{(x_1,y_1), \cdots , (x_n,y_n)\}$ where $x_1, \cdots , x_n$ are real numbers, and the labels $y_1, \cdots , y_n \in \{0,1\}$ have only binary values, it is easy to construct datasets which have provably no $k$-segmentation (or even $k$-tree) coreset of size smaller than $n$; see e.g. [54]. Hence, there is no non-trivial decision tree coreset for general datasets of $n$ vectors in any dimension. However, as we prove in the rest of the paper, a coreset does exist for datasets where the input is a matrix, i.e., a discrete signal where every coordinate in the domain is assigned a label (value), rather than a random set of $n$ vectors.

**The first coreset for $n \times m$-signals.** To overcome the above problem, while still obtaining a small coreset, we assume a discretization of the dataset so that every coordinate has a label. We also assume, mainly for simplicity and lack of space, that the input feature space is $A = [n] \times [m] \subseteq \mathbb{R}^2$. That is, the input can be represented by an $n \times m$ matrix. The output coreset may contain fraction of entries, as in Fig. 3 which is called an $n \times m$ signal; see Section 1.5. Our assumption on the input data seems to be the weakest assumption that can enable us to have a provably small coreset for any input. Furthermore, it seems natural for e.g. images, matrices, or any input data from sensors (such as GPS) that has a value in every cell or continuous in some other sense.

**Previous work.** The prior works [54, 24, 62], which only handle the case of segmenting a 1-dimensional signal, use relaxations similar to our relaxation above to obtain a coreset of size $O(k/\varepsilon^2)$. However, our results apply easily for the case of vectors (1-dimensional signals) as in [54] and generalize for tensors if $d \geq 3$. We also give further applications, and provide extensive experiments with popular state of the art software.

A special case for $d = 2$ includes image compression, where quadtrees are usually used in e.g. MPEG4 to replace the image by smooth blocks of different sizes [55], or for completion of missing values [58]. Using dynamic programming, it is easy to compute the optimal tree of a 2D-signal $D$ in $O(k^2 n^5)$ time [5], which is impractical even for small datasets, unless applied on a small coreset of $D$. However we do not know of any such coreset construction, for $d \geq 2$, with provable guarantees on its size. To this end, the following questions are the motivation for this paper: (i): Is there a small coreset for any $n \times m$ signal (e.g. of sub-linear size)? (ii): If so, can it be computed efficiently? (iii): Can it be used on real-world datasets to boost the performance of existing random forest implementations?

**Extensions.** For simplicity, we focus on the classic sum of squared distances (SSE) or the risk minimization model [59]. However, our suggested techniques mainly assume that a coreset for the case $k = 1$ is known, which is trivial for SSE, but exists for many other loss functions e.g., non-squared distances; see Section 6.

### 1.3 Our Contribution

For any given error parameter $\varepsilon \in (0,1)$, and an integer $k \geq 1$, this paper answers affirmatively the above three questions. More formally, in this paper we provide:

(i): A proof that every $n \times m$ signal $D$ has a $(k,\varepsilon)$-coreset $(C,u)$ of size $|C|$ polynomial in $k \log(nm)/\varepsilon$. To our knowledge, this is the first coreset for decision trees whose size is smaller than the input; see Theorem 8. Due to lack of space, our full proofs are given in the appendix.

(ii): A novel coreset construction algorithm that outputs such a coreset $(C,u)$ with the above guarantees, for every given input signal $D$. Its running time is $O(nmk)$, i.e., linear in the input size
[D]. Unlike common coreset constructions, our algorithm is deterministic; see Algorithm [3].

(iii): AutoML for decision trees: since the suggested coreset approximates every tree of at most \( k \) leaves, we may use the same coreset for hyperparameter tuning. We demonstrate this in Section [5] by calibrating the parameter \( k \) using only the coreset, as compared to using the original (big) data.

(iv): Experimental results that apply modern solvers, such as the sklearn and LightGBM libraries, on this coreset for real-world public datasets. We show that our coreset can help boost the computation time of the above forest implementations and their parameter tuning by up to \( 10 \), while keeping similar accuracy; see Section [5].

(v): Open source code for our algorithms [3]. We expect that it will be used by both the academic and industry communities, not only to improve the running time of existing projects, but also to extend the algorithms and experimental results to other libraries and cost functions; see Section [6].

1.4 Novel technique: partition trees meet decision trees

In a seminal paper [28] during the 80’s of the previous century, Haussler and Welz introduced the importance of VC-dimension by Vapnik–Chervonenkis [60]. Their main application was partition trees for answering range queries.

Informally, a partition tree of a given set of points on the plane is the result of computing recursively a simplicial partition which is defined as follows. For a set \( D \) of \( N \) points on the plane, a \((1/\varepsilon)\)-simplicial partition is the partition of \( D \) into \( O(1/\varepsilon) \) subsets, such that: (i) each subset has at most \( 2\varepsilon N \) points, and (ii) Every line in the plane intersects the convex hull of at most \( \sqrt{1/\varepsilon} \) sets. Answering range queries of the form “how many points in \( D \) are in a given rectangular” in sub-linear time, using partition trees, is straightforward: We can sum in \( O(1/\varepsilon) \) time the number of points in the subsets of the above simplicial partition that are not intersected by the query rectangular. We then continue recursively to count the points on each of the \( \sqrt{1/\varepsilon} \) intersected sets.

In other words, the main idea behind the above work is to partition the input into a (relatively small) number of subsets, each containing a fraction of the input, such that each query (in this case, a rectangular shape) might intersect only a small fraction of those subsets. Such a partition is termed a simplicial partition. The number of points contained in non-intersected subsets can be easily computed, while the sum of points in intersected subsets require a more involved solution. The novelty in that work is how to achieve such a partition of the input.

Our paper closes a loop in the sense that it forges links between decision trees in machine learning – to partition trees from computational geometry. We aim to generalize the above technique from covering problems to regression and classification problems. This is by devising an algorithm which achieves the above requirements, but where the query is a decision tree (and not a rectangular shape), and the cost function is the sum of squared distances to the query and not the number of points.

More precisely, we partition the input dataset \( D \) into a relatively small number of subsets, such that every possible decision tree (query) intersects at most few of these subsets. We then independently compress every subset via another novel algorithm such that the cost (sum of squared distances, in this case) of points contained in non-intersected subsets can be easily and accurately estimated, while the cost of points in intersected subsets can be provably approximated via a more involved calculation. This is very different from existing coreset techniques that are sampling-based [38], or utilize convex optimization greedy algorithms [16]. Our main challenge was to define and design such a “simplicial partition for sum of squared distances”, and the coreset to be computed for each subset in this partition.

1.5 Preliminaries

In this section we define the notation that will be used in the next sections.

Let \( n \geq 1 \) and denote \( [n] = \{1, \ldots, n\} \). An \( n \)-signal is a set \( \{(x, f(x)) \mid x \in [n]\} \) that is defined by a function \( f : [n] \to \mathbb{R} \) (known as the graph of \( f \)). For an additional integer \( m \geq 1 \), an \( n \times m \) signal \( D = \{(x, g(x)) \mid x \in [n] \times [m]\} \) is the set that corresponds to a function \( g : [n] \times [m] \to \mathbb{R} \). That is, \( D \) represents an \( n \times m \) real matrix whose size is \( |D| = N = nm \). For integers \( i_1, i_2, j_1, j_2 \) such that \( 1 \leq i_1 \leq i_2 \leq n \) and \( 1 \leq j_1 \leq j_2 \leq m \), an \( n \times m \) sub-signal is the set \( B = \{(x, g(x)) \mid x \in \{1, \ldots, i_2\} \times \{j_1, \ldots, j_2\}\} \subseteq D \); see Fig. 1. A sub-signal \( B \) is called a row (respectively, column) if \( i_1 = i_2 \) (respectively, \( j_1 = j_2 \)). For a sub-signal \( B \), we
denote by $B^T = \{(i,j), y) | ((i,j), y) \in B\}$ the transposed sub-signal. A $k$-segmentation $s$ is said to intersect an $n \times m$ sub-signal $B$ if $s$ assigns at least two distinct values to the entries of $B$, i.e., $|\{(x,y) | (x,y) \in B\}| \geq 2$. Furthermore, by definition, a $k$-segmentation $s$ induces a partition of an $n \times m$ sub-signal $B$ into at most $k$ $n \times m$ sub-signals. For two functions $f, g : \mathbb{R} \to \mathbb{R}$ we use the big $O$ notation $f(x) \in O(g(x))$, thinking of $O(g(x))$ as the class of all functions $h(x)$ such that $|h(x)| \leq c|g(x)|$ for every $x > x_0$, for some constants $c$ and $x_0$. Lastly, we denote $SEG_k := SEG(k,2)$ for brevity.

**Paper organization.** Section 2 provides a rough approximation to the $k$-segmentation problem. Section 3 provides an algorithm for computing a simplicial partition for the $k$-segmentation problem. Each region in this partition will be then compressed individually in Section 4 to obtain our desired coreset. Experimental results and discussions are given in Section 5, and a conclusion in Section 6.

## 2 Bi-criteria Approximation

A coreset construction usually requires some rough approximation to the optimal solution as its input. Unfortunately, we do not know how to efficiently compute even a constant factor approximation to the optimal $k$-segmentation problem in Definition 2 as explained in Section 1. Instead, we provide an $(\alpha, \beta)_k$ or bi-criteria approximation [22], where the approximation is with respect to a pair of parameters: the number of regions in the partition may be up to $\beta k$ instead of $k$, and the loss may be up to $\alpha \cdot opt_k(D)$ instead of $opt_k(D)$.

**Definition 4** $(\alpha, \beta)_k$-approximation. Let $D$ be an $n \times m$ sub-signal, $k \geq 1$ be an integer and let $\alpha, \beta > 1$. A function $s : [n] \times [m] \to \mathbb{R}$ is an $(\alpha, \beta)_k$-approximation of $D$, if $s$ is a $\beta k$-segmentation whose fitting loss to $D$ is at most $\alpha$ times the loss of the optimal $k$-segmentation of $D$, i.e., $s \in SEG(\beta k)$ and $\ell(D, s) = \sum_{(x,y) \in D} (s(x) - y)^2 \leq \alpha \cdot opt_k(D)$.

We now describe an algorithm that computes such an approximation, in time only linear in the input’s size $|D| = nm$. The following lemma gives the formal statement. A suggested implementation for the algorithm is given in the appendix, as well as the full proof of the lemma; see Section 5.

**Lemma 5.** Let $D = \{(x_1, y_1), \ldots, (x_{nm}, y_{nm})\}$ be an $n \times m$ sub-signal and $k \geq 1$ be an integer. Then, there is an algorithm that can compute, in $O(k \log(nm))$ time, an $(\alpha, \beta)_k$-approximation for $D$, where $\alpha \in O(k \log(nm))$ and $\beta \in O(k \log^2(nm))$.

**Overview of the bicriteria algorithm from Lemma 5.** The algorithm is iterative and works as follows. At the $i$th iteration, we find a collection $B_i$ of at most $t$ disjoint sub-signals in $D_i$ (where $D_0 = D$ is the input), for which: (i) $\sum_{B \in B_i} opt_1(B) \leq opt_k(D_i) \leq opt_k(D)$, and (ii) $\cup_{B \in B_i} B$ has size $|\cup_{B \in B_i} B| \leq |D_i|/c$ for some parameter $c$ that depends on $k$, i.e., those sub-signals contain at least a $1/c$ fraction of $D_i$. We then define $D_{i+1} = D_i \setminus \cup_{B \in B_i} B$. After repeating this for at most $\psi \in O(\log(nm))$ iterations, we end up covering all entries of $D$ with sub-signals where the overall loss of the sub-signals in each iteration is at most $opt_k(D)$. This defines a partition of $D$ into a collection of at most $t \psi^2$ disjoint sets $B'$, which, in turn, define a set of at most $(t \psi)^2$ distinct sub-signals. The output is now simply the function $s$ that assigns, for every $B \in B'$ and $b \in B$, the mean value $s(b) = \frac{1}{|B|} \sum_{(i,j) \in B} y$ of $B$. See Pseudo-code in Algorithm 4 at the appendix.

## 3 Balanced Partition

In this section we present Algorithm 2 which computes a partition similar to the simplicial partition described in Section 1.4. It computes, in $O(|D|)$ time, a partition $B$ of the input $D$ that satisfies the following properties: (i) $|B|$ depends on $k/e$ but independent of $|D|$, (ii) the loss $opt_1(B)$ of every $B \in B$ is small, and (iii) every $k$-segmentation $s$ intersects only few sub-signals $B \in B$; see Definition 6, Fig. 2 and Lemma 7. A full proof is given at the appendix; see Section 6.

**Definition 6** (Balanced Partition). Let $D$ be an $n \times m$ signal, $k \geq 1$ be an integer, and $c_1, c_2, c_3 > 0$. A $(c_1, c_2, c_3)_k$-balanced partition of $D$ is a partition $B$ of $D$ such that: (i) $B$ contains $|B| \leq c_1 n \times m$ sub-signals, (ii) $opt_1(B) \leq c_2$ for every $B \in B$, and (iii) every $k$-segmentation $s$ intersects at most $c_3$ sub-signals $B \in B$ (i.e., assigns more than one unique number to those sub-signals).

**Lemma 7.** Let $D$ be an $n \times m$ signal, $k \geq 1$ be an integer, $\epsilon \in (0,1/4)$ be an error parameter, and $s : [n] \times [m] \to \mathbb{R}$ be an $(\alpha, \beta)_k$-approximation of $D$, where $\alpha, \beta > 1$. Define $\sigma := \frac{\ell(D, s)}{\alpha}$
Figure 2: (Left): A $6 \times 5$ signal $D$ consisting of 6 rows $R_1, \ldots, R_6$. (Middle): A step by step illustration of the call to $B := \text{PARTITION}(D, 1/4, 64)$ which partitions $D$ into $|B| = 13$ sub-signals (sub-matrices) as follows. (1) $B' := \text{SLICEPARTITION}([R_1], 4)$ (top green row). (2) $B' := \text{SLICEPARTITION}(R_1 \cup R_2, 4)$ (middle green matrix), and so on as long as the output contains at most $|B'| \leq 1/\gamma = 4$ sub-signals (as in the bottom green matrix (3)). We then append $B'$ to the output $B$, and repeat with the remaining $\{R_4, R_5, R_6\}$. (4) $B' := \text{SLICEPARTITION}([R_1], 4)$ which already returns $|B'| = 5 > 1/\gamma$ signals (yellow matrix). We append them to $B$ and repeat. (Right): The final partition $B$, where $\text{opt}_1(B) \leq \gamma^2 \sigma = 1/4^2 \cdot 64 = 4$ for every $B \in B$.

and $\gamma := \frac{\varepsilon^2}{3k^5}$. Let $B$ be an output of a call to $\text{PARTITION}(D, \gamma, \sigma)$; see Algorithm 2. Then $B$ an $(O\left(\frac{\varepsilon^2}{\gamma^2}\right), \gamma^2 \sigma, O\left(\frac{k\varepsilon^2}{\gamma^2}\right))$-balanced partition of $D$. Moreover, $B$ can be computed in $O(nm)$ time.

**Overview of Algorithms** Algorithm 2 gets as input an $n \times m$-signal $D$ and two parameters $\sigma, \gamma$. Algorithm 2 aims to compute a balanced partition of $D$; see Fig. 2. In turn, it calls Algorithm 1 which takes as input an $n \times m$ sub-signal $R$ that is defined by several contiguous rows of the original dataset $D$, and a parameter $\sigma > 0$, and aims to compute a partition $B$ of $R$. To do so, Algorithm 1 partitions $R$ along the vertical dimension (e.g., into vertical slices), in a greedy fashion, such that for every $B \in B$, $\text{opt}_1(B)$ is as large as possible, while still upper bounded by $\sigma$. This will ensure that the partition is into a relatively small number of slices. In the case where one of the sub-signals $B$ in this vertical partition of $R$ contains only one column, and already exceeds the maximum tolerance $\text{opt}_1(B) > \sigma$, we recursively apply Algorithm 1 to $B'$ in order to partition $B$ horizontally. As long as the total number of slices returned by Algorithm 1 is smaller than $1/\gamma$, Algorithm 2 adds yet another row to the previous set of rows, and repeats the above process. At this point, the partition of the current horizontal slice (collection of rows) $R$ is final, and is added to the output partition of Algorithm 2. In turn, a new horizontal slice $R$ of just one row, the first row of $D$ that is not included in the previous $R$, is initiated on which we again call Algorithm 1.

**Algorithm 1: SLICEPARTITION($D, \sigma$)**

**Input**: A parameter $\sigma > 0$ and an $n \times m$ signal $D = \{(x_i, y_i)\}_{i=1}^N$.

**Output**: A partition $B$ of $D$.

1. $B := \emptyset$ and $c_{\text{begin}} := 1$
2. while $c_{\text{begin}} \leq m$ do
3.     $B := \{(i, j), y \in D \mid j = c_{\text{begin}}\}$ // extract first column
4.     if $\text{opt}_1(B) > \sigma$ then
5.         $B' := \text{SLICEPARTITION}(B, \sigma)$
6.         $B := B \cup \{B' \mid |B' | \leq \gamma^2\sigma\}$
7.         $c_{\text{begin}} := c_{\text{begin}} + 1$
8.     else
9.         $c_{\text{end}} := c_{\text{begin}}$
10.        while $\text{opt}_1(B) \leq \sigma$ and $c_{\text{end}} < m$ do
11.            $c_{\text{end}} := c_{\text{end}} + 1$ and $\text{lastB} := B$
12.            $B := \{(i, j), y \in D \mid i \in [c_{\text{begin}}, c_{\text{end}}]\}$
13.        end
14.     end
15. return $B$

**Algorithm 2: PARTITION($D, \gamma, \sigma$)**

**Input**: An $n \times m$ signal $D$, a parameter $\gamma \in (0, 1)$, and a lower bound $\sigma \in [0, \text{opt}_1(D)]$.

**Output**: A partition $B$ of $D$; see Lemma 7

1. $B := \emptyset$ and $r_{\text{begin}} := 1$
2. while $r_{\text{begin}} \leq n$ do
3.     $R := \{(i, j), y \in D \mid i = r_{\text{begin}}\}$ // extract first row
4.     $B' := \text{SLICEPARTITION}(R, \gamma^2 \sigma)$
5.     $r_{\text{begin}} := r_{\text{begin}} + 1$
6.     last$B' := B'$
7.     while $|B'| \leq 1/\gamma$ and $r_{\text{end}} < n$ do
8.         $r_{\text{end}} := r_{\text{end}} + 1$
9.         last$B' := B'$
10.        $S := \{(i, j), y \in D \mid i \in [r_{\text{begin}}, r_{\text{end}}]\}$ // extract a slice
11.        $B' := \text{SLICEPARTITION}(S, \gamma^2 \sigma)$
12.        $B := B \cup \text{lastB}$
13.     end
14. return $B$
4 Coreset Construction

In this section, we present our main algorithm (Algorithm 3), which outputs a \((k, \varepsilon)\)-coreset for a given \(n \times m\) signal \(D\), the number of leaves \(k \geq 1\), and an approximation error \(\varepsilon \in (0, 1)\).

**Overview of Algorithm 3**. The algorithm first utilizes the \((\alpha, \beta)\)-approximation from Section 2 to obtain a lower bound \(\sigma \leq \text{opt}_k(D)\) for the optimal \(k\)-segmentation. It then computes, as described in Section 3, a balanced partition \(B\) of \(D\), where \(\text{opt}_k(B)\) is small and depends on \(\sigma\), for every \(B \subseteq D\). Finally, it computes a small representation \((C_B, u_B)\) for every \(B \in B\), and returns the union of those representations. Each such pair \((C_B, u_B)\) satisfies: (i) \(|C_B| = 4\), and (ii) has the same weighted sum of values, weighted sum of squared values, and sum of weights, as \(B\), i.e., \(\sum_{(a,b) \in C_B} u_B((a,b)) \cdot (b \cdot |b^2| + 1) = \sum_{(x,y) \in B} (y \cdot |y^2| + 1)\); see Fig. 3. Such a representation can be computed using Caratheodory’s theorem, as explained in Section 5 of the supplementary material.

![Figure 3](image)

**Some intuition behind Algorithm 3**. Consider some \(k\)-segmentation \(s\). By the properties of the balanced partition \(B\) of \(D\), only a small number of sub-signals \(B \subset D\) are intersected by \(s\), i.e., assigned at least \(2\) distinct values. For every non-intersected sub-signal \(B \subset D\), the loss \(\ell(B, s)\) is accurately estimated by the (coreset) pair \((C_B, u_B)\). On the other hand, for every sub-signal \(B \subset D\) which is intersected by \(s\), by the guarantees of the representation \((C_B, u_B)\), the loss \(\ell(B, s)\) will be approximated, using only \((C_B, u_B)\), up to some small error that depends on \(\text{opt}_k(B)\). However, again by the properties of \(B\), we have that \(\text{opt}_k(B)\) is small. Hence, using the union \((C, u)\) of the representations we can approximate \(\ell(D, s)\) as required. Furthermore, combining that \(|C_B| \in O(1)\) for every \(B \subset D\) with the fact that \(|B| = n\), \(|C| = |B| = n\), and \(|C_B| = 4\), is small yields that \(|C|\) is indeed small; see Theorem 8.

**Algorithm 3**: SIGNAL-CORESET\((D, k, \varepsilon)\); see Theorem 8

**Input**: An \(n \times m\) signal \(D\), an integer \(k \geq 1\), and an error parameter \(\varepsilon \in (0, 1/4)\).

**Output**: \((k, \varepsilon)\)-coreset \((C, u)\) for \(D\).

\[ s := \alpha \approx \beta \approx O(k \log(nm)) \] and \(\beta \approx k^{O(1)} \log^2(nm) \); see Lemma 5 for suggested implementation.

\[ \gamma := \varepsilon^2 / (\beta k), \quad \sigma := \ell(D, s) / \alpha, \quad \text{and } C := \emptyset \]

\( B := \text{PARTITION}(D, \gamma, \sigma) \) // see Algorithm 2.

for every set \(B \subset D\) do

\( (C_B, u_B) := \text{a\,(1,0)-coreset for } B, \text{ (a zero error coreset for } k = 1), \text{ such that } C_B \subseteq B\).

\(|C_B| = 4, \quad \text{and } \sum_{(a,b) \in C_B} u_B((a,b)) \cdot (b \cdot |b^2| + 1) = \sum_{(x,y) \in B} (y \cdot |y^2| + 1)\) this is done using Caratheodory’s theorem; see Corollary 17 in the appendix.

Replace each of the coordinates \(a\) of the 4 pairs \((a,b) \in C\) with one of the 4 corner coordinates of the pairs in \(B\); see detailed explanation if the proof of Theorem 8.

\( C := C \cup C_B \text{ and } u((a,b)) := u_B((a,b)) \text{ for every } (a,b) \in C_B \).

return \((C, u)\)

**Theorem 8** (Coreset). Let \(D = \{(x_1, y_1), \ldots, (x_N, y_N)\}\) be an \(n \times m\) signal i.e., \(N := nm\). Let \(k \geq 1\) be an integer (that corresponds to the number of leaves/rectangles), and \(\varepsilon \in (0, 1/4)\) be an error parameter. Let \((C, u)\) be the output of a call to SIGNAL-CORESET\((D, k, \varepsilon/\Delta)\) for a
We implemented our coreset construction from Algorithm 3 in Python 3. - contains (ii) Gesture Phase Segmentation Dataset - returns a uniform random sample of size \( \tau \) from \( D \). In all tests \( \tau \) was set to the size of the coreset \( \Delta \)-coreset for \( (D, k, \varepsilon) \) for fair comparison.

Datasets. We used the following common implementations: (i) the function RandomForestRegressor from the sklearn.ensemble package, and (ii) the function LGBMRegressor from the lightGBM package that implements a forest of gradient boosted trees. Both functions were used with their default hyperparameters, unless states otherwise.

Data summarizations. We consider the following compression schemes:

(i): \( \Delta \)-coreset \((D, k, \varepsilon)\) - The implementation based on Algorithm 3. In all experiments we used a constant \( k = 2000 \) for computing the coreset, regardless of the (larger) actual \( k \) value in each test, since \( k = 2000 \) was sufficient to obtain a sufficiently small empirical approximation error. Hence, the parameter \( \varepsilon \) controls the trade-off between size and accuracy.

(ii): RandomSample \((D, \tau)\) - returns a uniform random sample of size \( \tau \) from \( D \). In all tests \( \tau \) was set to the size of the coreset \( \Delta \)-coreset \((D, k, \varepsilon)\) for fair comparison.

5 Experimental Results

We implemented our coreset construction from Algorithm 3 in Python 3.7, and in this section we evaluate its empirical results, both on synthetic and real-world datasets. More results are placed in the supplementary material; see Section A. Open-source code can be found in [53]. The hardware used was a standard MSI Prestige 14 laptop with an Intel Core i7-10710U and 16GB of RAM. Since our coreset construction algorithm does not compete with existing solvers, but improves them by reducing their input as a pre-processing step, we apply existing solvers as a black box on the small coreset returned by Algorithm 3. The results show that our coreset can boost, by up to x10 times, the running time and storage cost of common random forest implementations.

Implementations for forests. We used the following common implementations: (i) the function RandomForestRegressor from the sklearn.ensemble package, and (ii) the function LGBMRegressor from the lightGBM package that implements a forest of gradient boosted trees. Both functions were used with their default hyperparameters, unless states otherwise.

Data summarizations. We consider the following compression schemes:

(i): \( \Delta \)-coreset \((D, k, \varepsilon)\) - The implementation based on Algorithm 3. In all experiments we used a constant \( k = 2000 \) for computing the coreset, regardless of the (larger) actual \( k \) value in each test, since \( k = 2000 \) was sufficient to obtain a sufficiently small empirical approximation error. Hence, the parameter \( \varepsilon \) controls the trade-off between size and accuracy.

(ii): RandomSample \((D, \tau)\) - returns a uniform random sample of size \( \tau \) from \( D \). In all tests \( \tau \) was set to the size of the coreset \( \Delta \)-coreset \((D, k, \varepsilon)\) for fair comparison.

Datasets. We used the following pair of datasets from the public UCI Machine Learning Repository [50], each of which was normalized to have zero mean and unit variance for every feature:

(i): Air Quality Dataset [18] - contains \( n = 9358 \) instances and \( m = 15 \) features.

(ii) Gesture Phase Segmentation Dataset [43] - contains \( n = 9900 \) instances and \( m = 18 \) features.

The experiment. The goal was to predict missing entries in every given dataset, by training random forests on the available data. The test set (missing values) consists of 30% of the dataset, and was extracted from the input dataset matrix by randomly and uniformly choosing a sufficient number of \( 5 \times 5 \) patches in the input dataset, and defining them as missing values. The final loss of a trained forest is the sum of squared distances between the forest predictions for the missing values, and the ground truth values. To tune the hyperparameter \( k \), we randomly generate a set \( K \) of possible values for \( k \) on a logarithmic scale. Then, we either: (i) apply the standard tuning (train the forest on the full data, for each value in \( K \), and pick the one with the smallest test set error), or (ii) compress the input (only once) into a small representative set, and then apply the standard tuning on the small, rather than the full, data. The experiment was repeated 10 times. All the results are averaged over all 10 tests; see Fig. 4.

Discussion. While the size and accuracy of our coreset are independent of our exact implementation of Algorithm 3, the running time is heavily based on our naive implementation, as compared to the very efficient professional Python libraries. This explains why most of the running time is still devoted to the coreset construction rather than the forest training. Nevertheless, even our simple implementation yielded improvements of up to x10 in both computational time and storage, for a relatively small accuracy drop of 0.03 in the SSE. Tuning more than one hyperparameter will result in a bigger improvement. Furthermore, Fig. 5 empirically shows that tuning a hyperparameter on the
Figure 4: Experimental results. (Top): The X-axis is the compression size. For every compression size $\gamma$, hyperparameter tuning is applied on both the coreset and the uniform sample (which are both of size $\gamma$). A random forest is then trained, on the full data, using those tuned parameters. The Y-axis presents the test set SSE loss of the trained forests. (Bottom left): Hyperparameter tuning. For every different value of $k$ (X-axis), a forest is trained using this parameter value either on the compression (of two different sizes) or on the full data. The Y-axis presents $\ell + k/10^5$, where $\ell$ is the normal SSE loss of the trained forest on the test set. (Bottom right): Time comparison. The Y-axis presents the total running time of both to compute the compression and to tune the parameter $k$ on the compression (out of 50 different values). Note that the bottom right figures measure the total time to tune the parameter $k$ in the bottom left figures, but using many more compression sizes. The optimal obtained parameter was then used to train the random forest in the top figures.

coreset yields a loss curve very similar to the loss curve of tuning on the full data. Lastly, we observe that, in practice, our coresets have size much smaller than predicted in the pessimistic theory.

6 Conclusions and Future Work

While coresets for $k$-trees do not exist in general, we provided an algorithm that computes such a coreset for every input $n \times m$ signal. The coreset size depends polynomially on $k \log(nm)/\varepsilon$ and can be computed in $O(nmk)$ time. Our experimental results on real and synthetic datasets demonstrates how to apply existing forest implementations and tune their hyperparameters on our coreset to boost their running time and storage cost by up to $x10$. In practice our coreset works very well also on non-signal datasets, probably since they have “real-world” properties that do not exist in the artificial worst-case example from Section 1.2. An open problem is to define these properties. Moreover, while this paper focuses on the sum of squares distances loss, we expect that the results can be generalized to support other loss functions; see Section 1.2. Lastly, supporting high-dimensional data (tensors), instead of matrices, is a straightforward generalization that can be achieved via minor modifications to our algorithms. We also leave this to future work.

Acknowledgments and Disclosure of Funding

This research was supported by The ISRAEL SCIENCE FOUNDATION, grant number 379/21.

References


Figure 5: The blobs dataset. The dataset $D$ was generated using the function `sklearn.datasets.make_blobs`, and contains $n = 17,000$ points clustered into 3 clusters (containing 8500, 5800, and 2700 points), each with a different label. A coreset $(C, u)$ was constructed using Algorithm 3. From top down, the rows illustrate: (i) The input dataset $D$, (ii) The balanced partition of $D$, including the number of sets in the partition (iii) The weighted coreset points. Each point $(x, y) \in C$ is plotted at location $x$, colored according to its label $y$, and its radius is proportional to its weight $u(x, y)$. The percentage of the coreset size relative to the full data is presented. (iv) The unweighted coreset points. Each point $(x, y) \in C$ is plotted at location $x$, colored according to $y$, and has a fixed radius. The percentage of the coreset size relative to the full data is presented. (v) The partition of the space via a decision tree computed using a call to `DecisionTreeRegressor` from the `sklearn.tree` package, where the input was the weighted coreset points only. Each region is assigned a color according to the label assigned to it by the computed tree. (vi) Similar to Row (v), but where the decision tree is trained on the full data. Algorithm 5 was used during training of the decision tree to evaluate the loss of each model.

A Additional Experiments

In this section, we present some additional experiments conducted using our algorithm from Sections 3-4. We give visual illustration both of our coreset itself and of the result of applying a very common decision tree implementation on the coreset, as compared to running the same function on the original (full) data; see Fig. 5-6.

Discussion. Visual representation. As seen in Fig. 5-6 the balanced partition in the second row partitions the input data into multiple subsets, where, as expected, flat and relatively smooth regions are partitioned into a smaller number of large cells, while more complex regions are partitioned into a
Figure 6: The moons dataset. The dataset was generated using the function `sklearn.datasets.make_moons`. The dataset contains $n = 24,000$ points spread across two interleaving half circles (12,000 points for each half circle), each with a different label. See caption of Fig. 5 for a detailed explanation about the rows.

A larger number of finer cells. This is expected since the balanced partition insures a small variance inside each cell.

Furthermore, as seen in the third row of the above figures, the weighted coreset contains a small number of “large” circles (points with large weight) in the flat and relatively smooth regions, while it contains large number of “small” circles (points with small weight) in the more complex regions of the input.

**Accuracy.** As seen in the last two rows of Fig. 5-6, the decision tree trained only on the coreset points resembles the decision tree trained on the full data, even for coresets of size only 6%, 8%, and 14% of the input data, as seen in Fig. 5-6 and 7 respectively. This implies a $\times 10$ faster training time of a decision tree (or, similarly, a forest) on a given coreset, compared to training it on the full data, with almost no compromises to the accuracy.

The difference in the coreset size required in order to accurately represent the full data depends on the complexity of the input dataset. Indeed, the dataset in Fig. 5 is a much simpler dataset for a decision tree to classify, compared to the dataset in Fig. 7. Hence, the coreset sizes required in Fig. 5 are smaller than the ones in Fig. 7.
Figure 7: The circles dataset. The dataset was generated using the function `sklearn.datasets.make_circles`. The dataset contains \( n = 26,000 \) points spread across a big circle (14,000 points) and a small circle (12,000 points), each with a different label. See caption of Fig. 5 for a detailed explanation about the rows.

B Bi-criteria Approximation

Notations. A sub-signal \( B \) is said to be horizontally intersected by a \( k \)-segmentation function \( s \) if there are \( ((i_1, j_1), y_1), ((i_2, j_2), y_2) \in B \) where \( i_1 \neq i_2 \) such that \( s(i_1, j_1) \neq s(i_2, j_2) \). Similarly, a block \( B \) of \( D \) is said to be vertically intersected by \( s \) if there are \( ((i_1, j_1), y_1), ((i_2, j_2), y_2) \in B \) where \( j_1 \neq j_2 \) such that \( s(i_1, j_1) \neq s(i_2, j_2) \). \( B \) is said to be intersected by \( s \) if \( B \) is either horizontally or vertically intersected, i.e., \( \left| \{ s(x) \mid (x, y) \in B \} \right| > 1 \). A set of sub-signals \( \mathcal{B} \) is said to be horizontally (vertically) intersected by \( s \) if it contains a sub-signal \( B \in \mathcal{B} \) that is horizontally (vertically) intersected by \( s \).

Also, we might abuse notation and denote by signal (sub-signal) an \( n \times m \) signal (sub-signal) and by \( k \)-segmentation an \( n \times m \) \( k \)-segmentation.

In this section we give a constructive proof for Lemma 5. A suggested implementation for this constructive proof is given in Algorithm 4.

We first prove a small technical observation (see Observation 9), and then we prove Lemma 10 which will be used throughout the proof of Lemma 5.

**Observation 9.** Let \( A \) and \( B \) be two \( n \times m \) sub-signals. Then it holds that
\[
\text{opt}_1(A \cup B) \geq \text{opt}_1(A) + \text{opt}_1(B).
\]
Algorithm 4: Bicriteria(\(D, k\)); Lemma \(\text{[5]}\)

**Input**: An \(n \times m\) sub-signal \(D = \{(x_i, y_i)\}_{i=1}^N\) and an integer \(k \geq 1\).

**Output**: An \((\alpha, \beta)_k\)-approximation for \(D\).

1. \(B := \emptyset\)
2. \(\nu, \gamma := \text{sufficiently large constants} \quad // \quad \text{see proof of Lemma \([5]\)}\)
3. \(\text{while } |D| > k \log N \text{ do}
4. \quad \text{if } D \text{ contains a row } R \text{ with } |R| \geq \frac{|D|}{2\nu k} \text{ then}
5. \quad \quad \text{Partition } [m] \text{ into } t' = \gamma k \text{ intervals } [m] = \bigcup_{j=1}^{t'} I_j \text{ such that every } j \in [t'], \text{the size of each corresponding sub-signal } R_j = \{((x_1, x_2), y) \in R \mid x_2 \in I_j\} \text{ is } |R_j| \in \left\{\frac{|R|}{t'}, \frac{|R|}{t'} + 1\right\}. // \text{e.g., by a greedy pass over } [m].
6. \quad \quad B := \text{the set of } t' - 2k \text{ signals } R_j \text{ with the smallest } \text{opt}_1(R_j).
7. \quad \text{else}
8. \quad \quad \text{Partition } [n] \text{ into } \psi \text{ intervals } [n] = \bigcup_{j=1}^{\psi} I_j \text{ such that for every } j \in [\psi], \text{the size of each corresponding sub-signal } D_j = \{((x_1, x_2), y) \in D \mid x_1 \in I_j\} \text{ is } |D_j| = \frac{|D|}{\nu k}, \quad // \text{e.g., by a greedy pass over } [n].
9. \quad \quad \text{if at least } \psi/2 \text{ of the sub-signals } D_j \text{ do not contain a column } \text{col} \text{ of size } |\text{col}| \geq \frac{|D|}{2(\nu k)^2} \text{ then}
10. \quad \quad \quad \text{Vertically partition each of the (at least } \psi/2 \text{) sub-signals } D_j \text{ into } \psi \text{ sub-signals, each such sub-signal } B \text{ of size } \frac{|D_j|}{2(\nu k)^2} \leq |B| \leq \frac{|D_j|}{(\nu k)^2}, \text{ and let } B' \text{ contain the union of all those sub-signals. e.g., via a greedy algorithm.}
11. \quad \quad \quad B := \text{the set of } |B'| - 4\psi^2k^3 - 2k\psi \text{ signals } B \in |B'| \text{ with the smallest } \text{opt}_1(B).
12. \quad \text{else}
13. \quad \quad B := \left\{C \mid C \text{ is a column of } D_j, |C| \geq \frac{|D_j|}{2(\nu k)^2} \text{ and } j \in [\psi]\right\}
14. \quad D := D \setminus \bigcup_{B \in B'} B \text{ and } \mathcal{B} := \mathcal{B} \cup B
15. \(s(b) := 1/|B| \sum_{(x,y) \in B} y \text{ for every } b \in B \text{ and } B \in \mathcal{B}\)
16. \text{return } s

**Proof.** Let \(C = A \cup B\). Let \(\mu = \frac{1}{|C|} \sum_{(x,y) \in C} y\) be the weighted mean of \(A \cup B\). By Definition of \text{opt} we have that

\[
\text{opt}_1(A \cup B) = \sum_{(x,y) \in C} (y - \mu)^2 = \sum_{(x,y) \in A} (y - \mu)^2 + \sum_{(x,y) \in B} (y - \mu)^2 \geq \text{opt}_1(A) + \text{opt}_1(B),
\]

where the first derivation holds since the mean of a points minimizes the sum of squared distances to those points, and the last derivation is by the definition of \text{opt}_1.

**Lemma 10.** Let \(D = \{(x_1, y_1), \ldots, (x_N, y_N)\}\) be an \(n \times m\) sub-signal and let \(k \geq 1\) be an integer. Then, in \(O(N)\) time we can find a set \(B\) of \(|B| = t \in O(k^3)\) mutually disjoint blocks with respect to \(D\), for which

\begin{enumerate}[(i)]
\item \(\sum_{B \in B} \text{opt}_1(B) \leq \text{opt}_k(D)\).
\item \(|\bigcup_{B \in B} B| \in \Omega \left(\frac{N}{k^3}\right)\).
\end{enumerate}

**Proof.** Let \(\nu > 50\) be an arbitrary parameter and let \(\gamma \geq 8\) be a parameter that will be defined later. We will prove Lemma \([10]\) for \(t \leq 2\nu^3k^3\) and for \(|\bigcup_{B \in B} B| \geq \frac{N}{k^3}\).

We start with the simple 1-dimensional case, namely – we assume that \(m = 1\). In this case, we just partition \([n]\) into \(t' = \gamma k\) consecutive intervals \([n] = \bigcup_{j=1}^{t'} E_j\), such that each corresponding sub-signal \(D_j = \{(a, b) \in D \mid a \in E_j\}, j \in [t]\) of \(D\) has equal share of elements in \(D\) (up to \(\pm 1\)), i.e.,
\[ |D_i| \in \{ |D|/t' - 1, |D|/t' + 1 \} \]. This can be done by moving point by point along the elements of \(D\), which are assumed to be sorted in ascending order according to \(a\) for every \(((a, b), y) \in D\), and defining a new interval at the first moment the current interval contains more than \(|D|/t'\) elements of \(D\) or just at the prior point. We then compute for each \(D_j, j \in [t']\) the optimal \(\text{opt}_t(D_j)\), and return as output the set \(B\) containing the \(t = t' - 2k\) sub-signals \(D_j\) with the smallest loss \(\text{opt}_t(D_j)\), among all the \(t'\) sub-signals \(\{D_1, \ldots, D_{t'}\}\).

To see what guarantees we get, note that the computation takes \(O(nm) = O(N)\) time to sort the elements \(((a, b), y)\) of \(D\) according to \(a\), since \(a \in [nm]\) is a bounded integer. Afterwards, the above greedy partition also takes \(O(nm) = O(N)\) time. Furthermore,

\[ |\bigcup_{B \in \mathcal{B}} B| \geq (2 - \gamma)k : \frac{|D|}{\gamma k} \geq \frac{\gamma - 2}{\gamma} |D| \geq \frac{N}{8^\nu k^\gamma} . \]

For \(\gamma \geq 8\) this proves Property (ii) above.

Finally, any \(n \times 1\) \(k\)-segmentation function intersects at most \(2k\) sub-signals from \(\{D_j\}_{j=1}^{t'}\) (i.e., at most \(2k\) sub-signals are assigned more than \(1\) distinct value via the \(k\)-segmentation function). Hence, the optimal \(k\)-segmentation \(s^*\) of \(D\) intersects at most \(2k\) of those sub-signals as well. This implies that at least \(t' - 2k \geq (\gamma - 2)k\) of the intervals \(\{D_j\}_{j=1}^{t'}\) which are assigned \(1\) distinct value \(|\{s(x) \mid (x, y) \in D_j\}| = 1\). Hence, by Observation\(9\) we conclude that \(\text{opt}_k(D) \geq \sum_{B \in \mathcal{B}} \text{opt}_t(D)\) which verifies the 1st item above.

We remark that for the 1-dimensional case we can do much better (there is an overall \((1 + \varepsilon)\)-approximation of the \(k\)-segmentation using logarithmic number of blocks), but this will not be used here. Another remark is that we have ignored the \(\pm 1\) slack in the sizes above, making the actual part of \(D\) that is removed at least \(\frac{\gamma - 2}{\gamma} |D| - t'\). This is insignificant in the 1-dimensional case above, as for \(t = O(1)\) and for \(|D| \geq \log n\) this would be an insignificant fraction, while for smaller \(D\), we can just use single point sub-signals.

**The 2-dim case**: Consider a row \(R\) of \(D\), say the \(i\)'th row \(R = \{((a, b), y) \in D \mid a = i\}\). We call a row \(R\) of \(D\) \(R\)-heavy if \(|R| \geq |D|/r\), namely \(-R\) contains at least \(|D|/r\) elements from \(D\). Analogously, we define a column to be \(r\)-heavy.

Assume first that our \(D\) contains a \(\nu k\)-heavy row \(R\). We choose \(R\) and use it as in the 1-dimensional case. As explained above we can find in \(R\) a set of disjoint sub-signals \(B\) containing \(\gamma k\) blocks and for which the first item above holds for \(\text{opt}_k(R)\) and in particular for \(\text{opt}_k(D)\), i.e., \(\sum_{B \in \mathcal{B}} \text{opt}_t(B) \leq \text{opt}_k(R) \leq \text{opt}_k(D)\). Further, using the above guarantees for the 1-dim case, \(|\bigcup_{B \in \mathcal{B}} B| \geq \frac{\gamma - 2}{\gamma} |R| \geq \frac{\gamma - 2}{\gamma} \cdot \frac{|D|}{\nu k}\). This proves 2nd case (with \(\gamma \geq 3\)).

Otherwise, let \(e_i = |R_i|\) where \(R_i = \{((a, b), y) \in D \mid a = i\}\). By our assumption \(e_i \leq |D|/\nu k\) for every \(i \in [n]\). Our algorithm is essentially identical to the 1-dimensional case, on the 1-dim array \(L = (e_1, \ldots, e_n)\) where we weight the \(ith\) element by \(e_i\). Namely, we find a partition of \(L\) into \(\psi\) contiguous subintervals \(\mathcal{E} = \{E_1, \ldots, E_\psi\}\) such that the corresponding sub-signals \(D_j = \{(a, b, y) \in D \mid a \in E_j\}\) of \(D\) are as equal as possible. By our assumption this could be done so that for any \(j \in \psi\), the number of elements in \(D_j\) is between \(\frac{|D|}{\nu k}\) and at most \(\frac{2|D|}{\nu k}\). This is since adding a new ‘point’ from the list to an existing interval may increase the sum by at most \(|D|/\nu k\). This implies that \(\nu k/2 \leq \psi \leq \nu k\).

Next we perform the above algorithm again on \(D_1, \ldots, D_\psi\) with the intention to “vertically partition” each such \(D_j\), i.e., split each \(D_j\) into sets according to the value \(b\) for every \(((a, b), y) \in D_j\) (rather than considering the value \(a\) above). Let \(r = 2\nu k^2\), we continue with the following case analysis:

(i) at least a \(\frac{1}{2}\)-fraction of \(\{D_1, \ldots, D_\psi\}\) contain no a \(r\)-heavy column, and (ii) at most a \(\psi/2\) of \(\{D_1, \ldots, D_\psi\}\) contain no a \(r\)-heavy column.

**Case (i)**: At least a \(\frac{1}{2}\)-fraction of \(\{D_1, \ldots, D_\psi\}\) contain no a \(r\)-heavy column. Then we partition each set \(D_j\) with no \(r\)-heavy column into \(\psi_i\) sub-signals \(\{D_j^{(1)}, \ldots, D_j^{(\psi_i)}\}\) of nearly equal number of points, where the partition is applied onto the values \(b\) of every \(((a, b), y) \in D_j\). By a reasoning similar to that above, each \(r/2 \leq \psi_i \leq r\), and each such block \(B \in \{D_j^{(1)}, \ldots, D_j^{(\psi_i)}\}\) contains \(|D_i|/r \leq |B| \leq 2|D_i|/r\). Using the bounds on \(|D_i|\) and \(r\) we get \(\frac{|D|}{2\nu k^2} \leq |B| \leq \frac{2|D|}{\nu k^2}\). In particular
We conclude that the total number of such sub-signals is at most \( t' \), and \( t' \leq 2\nu^3k^3 \). On the other hand, \( t' \geq \frac{\psi |D|}{2r_k} \cdot \frac{k^2}{2r_k} \geq \nu^3k^3/8 \). Let \( B_2 \) be the collection of these sub-signals.

We choose for our output collection, \( B \), the set of \( t' - z \) sub-signals \( B \in B_2 \) with the smallest \( \text{opt}_1(B) \), for \( z = 2k(r + \psi) \leq 6k^3/\nu^3 \). We then ‘delete’ from \( B \) all \( \psi \)-heavy column. In this case we take the heavy column from each \( B \in B_2 \) with the smallest loss, we obtain that the loss \( \ell(D,s) \) is at least \( \sum_{B \in B} \ell(B) \) proving the 1st item of the lemma for this case.

**Case (ii):** At most \( \psi/2 \) of \( \{D_1, \ldots, D_\psi\} \) contain no a \( r \)-heavy column, namely – at least \( \psi/2 \) of the \( D_i \) have a \( r \)-heavy column. In this case we take the heavy column from each \( D_i \) as its own sub-signal. We get a collection \( B_1 \) of \( \psi/2 \) blocks. We now return as output the set \( B \) of the \( \psi/2 \)-sub-signals \( B \in B_1 \) with the smallest \( \text{opt}_1(B) \). We note that number of blocks we output in this case is at most \( \psi \leq \nu k \).

Note also that \( B \) contains at least \( \psi/2 \) blocks, it follows that \( |\cup_{B \in B} B| \geq \frac{\psi}{2} \cdot \frac{|D|}{2r_k^2} \geq \frac{|D|}{8\nu k} \) which proves the 2nd item in the lemma.

Finally, note that any \( k \)-segmentation \( s \) can intersect at most \( 2k \) intervals from \( B_1 \) (similarly to the 1-dim case). Hence, there are at least \( |B_1| - 2k = \psi_1 - 2k \) sub-signals in \( B_1 \) that are not-intersected by \( s \), which implies that its loss \( \ell(D,s) \) is at least the sum \( \sum_{B \in B} \text{opt}_1(B) \), which proves the 1st item of this lemma.

**Remark:** we did not optimise the parameter. A slightly better partition can be obtained (less blocks), but this is good enough for our purposes.

**Computational time:** Note that the elements of the input \( D \) can be sorted in lexicographic order in \( O(nm) = O(N) \) time since the coordinates \( a \) and \( b \) for every \((a,b),y) \in D \) are bounded integers. Then, a linear-time preprocessing can be applied to the input \( D \) to store some statistics, e.g., the number of elements in each non-empty row or column, and the index of the next non-empty row or column for every element in \( D \). Afterwards, the above greedy partition also takes \( O(nm) = O(N) \) time.

We now restate and prove Lemma 5 from Section 2.

**Lemma 11 (Lemma 5).** Let \( D = \{(x_1,y_1),\ldots,(x_N,y_N)\} \) be an \( n \times m \) signal and \( k \geq 1 \) be an integer. Then, in \( O(kN) \) time we can compute an \((\alpha, \beta)_k\)-approximation for \( D \), where \( \alpha \in O(k \log N) \) and \( \beta \in O(kO(1) \log^2 N) \).

The proof of the above claim is a constructive proof. A suggested implementation is provided in Algorithm 4.

**Proof.** The top level idea of the algorithm is as follows. We suggest an iterative algorithm. We start the first iteration with \( D_1 = D \) and, using Lemma 10, find a collection of disjoint sub-signals \( B_1 = \{ B_1, \ldots, B_t \} \) in \( D_1 \) (which will not necessarily cover the entire signal \( D_1 \)), such that: (i) the sum of their 1-segmentation loss satisfies, \( \sum_{i=1}^t \text{opt}_1(B_i) \leq \text{opt}_k(D_1) \), and (ii) \( \cup_{B \in B_1} B \) has size \( |\cup_{B \in B_1} B| \geq |D_1|/c \) for some \( c \) (e.g., \( c \in O(k) \) in the lemma). Let \( B_1 \) be such a collection. We then ‘delete’ from \( D_1 \) the elements of \( \cup_{B \in B_1} B \), and set \( D_2 = D_1 \setminus \cup_{B \in B_1} B \).

In the \( i \)th iteration, we repeat the same process with respect to the current set \( D_i \). Namely, we find a collection \( B_i \) of at most \( t \) disjoint sub-signals in \( D \), for which: (i) \( \sum_{B \in B_i} \text{opt}_1(B) \leq \text{opt}_k(D_i) \), and (ii) \( \cup_{B \in B_i} B \) has size \( |\cup_{B \in B_i} B| \geq |D_i|/c \), i.e., those blocks cover at least a constant fraction of \( D_i \).

Repeating these iterative procedure for at most \( \psi = O(c \log(nm)) \) times, we end up covering all entries of \( D \) with sub-signals where the overall loss of the sub-signals in each iteration is at most
opt_k(D). This defines a collection of at most tψ sub-signals B that cover the entire original set D. Hence, the total overall loss over those sub-signals is \( \sum_{B \in B} OPT_1(B) \leq \psi opt_k(D) \). By defining the output function \( s \) such that for every \( B \in B \) and \( b \in B \), \( s \) assigns to \( b \) the mean value of \( B \), i.e., \( s(b) = 1/|B| \sum_{(i,j),y} e_B y \), we obtain that \( \ell(B, s) = opt_1(B) \) for every \( B \in B \), and that the

\[ t(D, s) \leq \sum_{B \in B} \ell(B, s) \leq \psi opt_k(D). \]

While for every \( B \in B \) \( s \) assigns the same value for every element \( b \in B \), there is not necessarily a partition of \( n \times m \) into \( |B| \in O(t \psi) \) distinct axis-parallel blocks that correspond to the sub-signals of \( B \). Therefore, \( s \) is not necessarily a \( |B| \)-segmentation function. However, looking at all possible intersections of the sub-signals in \( B \), it is easy to realize that the \( t \psi \) sub-signals in \( B \) define a partition of \( D \) into at most \( O(t^2 \psi^2) \) sub-signals \( B' \) that indeed correspond to a distinct partition of \( n \times m \) into \(|B'| \) distinct axis-parallel blocks. Hence, \( s \) is guaranteed to be a \( |B'| \)-segmentation function.

The parameters that are guaranteed by the lemma are \( c \in O(k), t \in O(k^3) \). This implies that \( \beta = |B'| \in O(t^2 \psi^2) = O(k^8 \log^2 nm) \) and \( \alpha = \psi \in O(k \log nm) \).

**Computational time:** By Lemma[10] each of the \( \psi \) iterations above takes time linear in the input size. The input size in the \( i \)th iteration is \( O(N((k-1)/k)^i) \) since at each iteration we remove at least a \( 1/k \) fraction of the input. Hence, the total running time is the sum of the geometric series \( N \cdot \sum_{i \in \psi} ((k-1)/k)^i \in O(kN) \).

\[ \square \]

\[ \Box \]

\[ \text{C Balanced Partition} \]

In this section we give our full proof for Lemma[7]. We first prove the following lemma regarding the output of Algorithm[1].

**Lemma 12.** Let \( D \) be an \( n \times m \) sub-signal, and \( \sigma > 0 \) be a parameter. Let \( B = \{B_1, \ldots, B_{|B|}\} \) be the output of a call to \( \text{SLICEPARTITION}(D, \sigma) \), where the sub-signals in \( B \) are numbered according to the order in which each of them was added to \( B \); see Algorithm[4]. Then the following properties hold:

\[(i) \quad B \text{ is a partition of } D. \]

\[(ii) \quad \text{opt}_1(B) \leq \sigma \text{ for every sub-signal } B \in B. \]

\[(iii) \quad \text{If } |B| > 8k \text{ then for any } k \text{-segmentation } s \text{ that does not horizontally intersect } D \text{ we have } \ell(D, s) \geq \left( \frac{|B|}{4} - 2k \right) \sigma. \]

\[(iv) \quad B \text{ can be computed in } O(|D|) \text{ time.} \]

**Proof.** We consider the variables defined in Algorithm[1].

**Proof of (i):** By construction it immediately follows that \( B \) is a partition of \( D \).

**Proof of (ii):** Consider a sub-signal \( B \in B \). We prove (ii) for each of the following cases: **Case (a):** \( B \) was added to \( B \) at Line[12] and **Case (b):** \( B \) was either added to \( B \) at Line[6].

**Case (a):** In this case, by the condition at Line[9] \( B \) must satisfy that \( \text{opt}_1(B) \leq \sigma \).

**Case (b):** In this case, \( B \) was returned via a recursive call. Hence, this case holds trivially by Case (a) above.

Therefore, (ii) above holds by combining Cases (a)–(b).

**Proof of (iii):** Let \( t = |B| \) and assume for simplicity that \( t \) is an even number. Recall that the index of each sub-signal in \( B \) indicates its order of insertion to \( B \), i.e., \( B_1 \) is the first sub-signal that was inserted to \( B \) and \( B_k \) was the last such sub-signal to be inserted to \( B \).

Observe that each recursive call \( B' := \text{SLICEPARTITION}(B^T, \sigma) \) at Line[5] returns at least \( |B'| \geq 2 \) sub-signals. This is because the recursive call happens only when \( \text{opt}_1(B^T) = \text{opt}_1(B) > \sigma \), which can only happen if \(|\{(i,j) \mid (i,j), y \in B\}| > 1\), i.e., \( B^T \) exceeds the maximum tolerance, and can indeed be partitioned into sub-signals. Hence, there are at least \( t/4 \) distinct pairs of consecutive sub-signals \( B_i \) and \( B_{i+1} \) that were both either computed via the recursive call or both were not computed via the recursive call. We now show that each such pair satisfies \( \text{opt}_1(B_i \cup B_{i+1}) > \sigma \).
Consider a pair of consecutive sub-signals \( B_i \) and \( B_{i+1} \) that were both not computed via the recursive call at Line 5. Let \( B' \subseteq B_{i+1} \) contain the elements \( ((i, j), y) \in B_{i+1} \) with the smallest value of \( i \) over all elements of \( B_{i+1} \). By the greedy partition loop at Line 9 we obtain that \( \text{opt}_1(B_i \cup B') > \sigma \). We now have that
\[
\text{opt}_1(B_i \cup B_{i+1}) \geq \text{opt}_1(B_i \cup B') > \sigma,
\]
where the first inequality is by Claim 9.

Consider a pair of consecutive sub-signals \( B_i \) and \( B_{i+1} \) that were both computed via the recursive call at Line 5. Then, similarly to the previous argument, we obtain that \( \text{opt}_1(B_i \cup B_{i+1}) > \sigma \).

Now, let \( s \) be a \( k \)-segmentation that does not horizontally intersect \( B \), i.e., it does not horizontally intersect any \( B \in \mathcal{B} \). By the definition of \( s \), there might be at most \( 2k \) sub-signals in \( B \) which are vertically intersected by \( s \). Hence, among the \( t/4 \) distinct consecutive pairs of sub-signals discussed above there are at least \( t/4 - 2k \) such pairs that are not intersected by \( s \).

Since \( \mathcal{B} \) is a partition of \( D \), we have that \( \ell(D, s) \) is at least the sum of \( \text{opt}_1(B_i \cup B_{i+1}) \geq \sigma \), over the above \( t/4 - 2k \) non-intersected pairs of sub-signals. Hence,
\[
\ell(D, s) \geq \left( \frac{t}{4} - 2k \right) \sigma = \left( \frac{|\mathcal{B}|}{4} - 2k \right) \sigma.
\]

**Proof of (iv):** The greedy Algorithm 1 can be implemented so that it computes only \( O(|D|) \) operations. The most costly operation is the computation of \( \text{opt}_1(B) \) for some sub-signal \( B \). We now argue that this can be computed in \( O(1) \) time. Let \( B \) be a sub-signal and let \( \mu_B = 1/|B| \sum_{(x,y) \in B} y \) be its mean value. Observe that
\[
\text{opt}_1(B) = \sum_{(x,y) \in B} (y - \mu_B)^2 = \sum_{(x,y) \in B} y^2 + |B| \cdot \mu_B - 2\mu_B \sum_{(x,y) \in B} y.
\]

By precomputing and storing some statistics at each of the signal’s elements, then the three terms on the right hand side of (1) can all be evaluated in \( O(1) \) time for any sub-signal \( B \). Hence, the total running time of Algorithm 1 is linear in the input size.

\[ \square \]

We now restate and prove Lemma 7 from Section 3.

**Lemma 13.** Let \( D \) be an \( n \times m \) signal, \( k \geq 1 \) be an integer, \( \epsilon \in (0, 1/4) \) be an error parameter, and \( s : [n] \times [m] \to \mathbb{R} \) be an \( (\alpha, \beta) \)-approximation of \( D \). Define \( \sigma := \frac{\ell(D, s)}{\alpha} \) and \( \gamma := \frac{\epsilon^2}{4 \beta^2} \). Then algorithm \textsc{Partition}(\( D, \gamma, \sigma \)) outputs a partition \( \mathcal{B} \) of \( D \) that is an \( \left( O \left( \frac{\alpha}{\gamma} \right), \gamma^2 \sigma, O \left( \frac{k\alpha}{\gamma} \right) \right)_k \)-balanced partition in \( O(|D|) \) time.

**Proof.** To prove that \( \mathcal{B} \) is a \( \left( O \left( \frac{\alpha}{\gamma} \right), \gamma^2 \sigma, O \left( \frac{k\alpha}{\gamma} \right) \right)_k \)-balanced partition as in Definition 6 we need to prove the following properties:

(i) \( \text{opt}_1(B) \leq \gamma^2 \sigma \) for every \( B \in \mathcal{B} \).

(ii) \( \mathcal{B} \) is a partition of \( D \) whose size is \( |\mathcal{B}| \in O \left( \frac{\alpha}{\gamma} \right) \).

(iii) For every \( k \)-segmentation \( \hat{s} \) there are \( O \left( \frac{k\alpha}{\gamma} \right) \) sub-signals \( B \in \mathcal{B} \) for which \( \hat{s} \) assigns at least 2 distinct values, i.e., \(|\{\hat{s}(x) \mid (x,y) \in B\}| \geq 2\).

**Proof of (i):** Observe that the output set \( \mathcal{B} \) contains the union of multiple output sets \( B' := \textsc{SlicePartition}(\cdot, \gamma^2 \sigma) \) computed via calls to Algorithm 1. By Property (ii) of Lemma 12 every sub-signal \( B \in B' \) in such output set \( B' \) satisfies that \( \text{opt}_1(B) \leq \gamma^2 \sigma \). Hence, Property (i) of Lemma 7 immediately holds.

\[ \square \]
We now prove that the number of output cells is at most $O(\alpha/\gamma)$. Let $B_1, \cdots, B_t$ denote the set of sub-signals $lastB'$ in each of the $t$ executions of Line 12 i.e., $B_1 := lastB'$ at the first time Line 12 was executed. Each such set is called a horizontal set.

Recall that $s \in SEG_{3\beta k}$ is a $\beta k$-segmentation. First of all, by the definition of a $\beta k$-segmentation function there are at most $2\beta k$ sets among the horizontal sets $B_1, \cdots, B_t$ which can be horizontally intersected by $s$.

Consider two consecutive horizontal sets $B_i, B_{i+1}$ that are not horizontally intersected by $s$, and let $H = \bigcup_{B \in B_i \cup B_{i+1}} B$ be the union of all the sub-signals in $B_i \cup B_{i+1}$. We now argue that the loss $\ell(H, s)$ is at least $O(\gamma \sigma)$. Hence, the number of blocks in horizontal sets that can be intersected by $s$ is at most $2\beta k$. Hence, the total number of horizontal sets is at most

$$m \in O \left( \frac{\alpha}{\gamma} + 2\beta k \right) \in O \left( \frac{\alpha}{\gamma} \right).$$

We now prove that the number of output cells is at most $|B| \in O \left( \frac{\alpha}{\gamma} \right)$ in two steps. In step (i) we consider the horizontal sets that contain more than one row of $D$ and show that they contain a total of $O \left( \frac{\alpha}{\gamma} \right)$ sub-signals. In step (ii) we consider the horizontal sets that contain exactly one row of $D$ and prove that they also contain a total of $O \left( \frac{\alpha}{\gamma} \right)$ sub-signals.

**Step (i):** By [2], the total number of horizontal sets is at most $m \in O(\frac{\alpha}{\gamma})$. Therefore, the total number of horizontal slices that contain more than one row of $A$ is also at most $O \left( \frac{\alpha}{\gamma} \right)$. By the construction in Algorithm [2], each such horizontal set $B_i$ with more than 1 row of $A$ is partitioned into at most $\frac{1}{\gamma}$ sub-signals. Hence, the total number of blocks in horizontal sets that contain more than one row of $D$ is at most $O \left( \frac{\alpha}{\gamma} \cdot \frac{2}{\gamma} \right) = O \left( \frac{\alpha}{\gamma^2} \right)$.

**Step (ii):** Consider all the horizontal sets $B_i$ which contain one row of $D$, and which have been partitioned into $|B_i| \leq 2\beta k \leq 1/\gamma$ blocks. The total number of blocks in such horizontal slices is thus bounded by the maximum number of horizontal slices $m \in O(\alpha/\gamma)$ times $1/\gamma$ for a total of at most $O(\alpha/\gamma^2)$ blocks.

For the rest of this step, we assume that all horizontal sets $B_i$ have $|B_i| \geq 2\beta k$. Let $G \subset [t]$ denote the indices of the horizontal sets which contain exactly one row of $D$, and let $i \in G$. Observe that $B_i$ was computed, at some point, via a call $B_i := SLICEPARTITION(\bigcup_{B \in B_i} B, \gamma^2 \sigma)$. Also, since the points $\bigcup_{B \in B_i} B$ of $B_i$ all have the same row index, observe that $B_i$ cannot be horizontally intersected by $s$. Therefore, by substituting $D = \bigcup_{B \in B_i} B, k = \beta k$ and $\sigma = \gamma^2 \sigma$ in Property (iii) of Lemma 12.
we obtain that
\[
\ell(B_i, s) \geq \left( \frac{|B_i|}{4} - 2\beta k \right) \cdot \gamma^2 \sigma.
\] (3)

Furthermore, we have that
\[
\alpha \cdot \sigma = \ell(D, s) \geq \sum_{i \in G} \ell(B_i, s) \geq \sum_{i \in G} \left( \frac{|B_i|}{4} - 2\beta k \right) \cdot \gamma^2 \sigma,
\] (4)

where the first derivation is by the definition of \( \sigma \), the second derivation holds since \( \{(x, y) \in B \mid B \in B_i, i \in G\} \subseteq D \), and the third derivation is by \( \Box \). Rearranging terms in (4) concludes Step (ii) as
\[
\sum_{i \in G} |B_i| \leq \frac{4\alpha}{\gamma^2} + 8\sum_{i \in G} \beta k \leq \frac{4\alpha}{\gamma^2} + 8t\beta k
\]
\[
\leq \frac{4\alpha}{\gamma^2} + \frac{8\alpha \beta k}{\gamma} \in O\left( \frac{\alpha}{\gamma^2} \right).
\]

Therefore, the total number of blocks in horizontal sets that contain exactly one row of \( D \) is at most \( O\left( \frac{\alpha}{\gamma^2} \right) \).

Proof of (iii): By the properties above we have that: (i) there are at most \( O(\alpha/\gamma) \) horizontal sets, and (ii) each horizontal set \( B_i \) either contains at most \( O(1/\gamma) \) sub-signals, or all the points \( ((i, j), y) \in B \) of all the blocks \( B \in B_i \) have the same row index \( i \). Let \( s \) be a \( k \)-segmentation. \( s \) can horizontally intersect all the (at most) \( 1/\gamma \) sub-signals of at most \( k \) horizontal sets, and can vertically intersect at most \( 1 \) block from each of the \( O(\alpha/\gamma) \) horizontal sets. Hence, the total number of intersected sub-signals is \( O(k\alpha/\gamma) \).

Computational time: We now prove that \( B \) can be computed in \( O(|D|) \) time. The computational time of Algorithm 2 is dominated by the computational time of Line 11 where we partition a slice \( S \). Using Algorithm 7 we can partition each such slice \( S \) in linear \( O(|S|) \) time; see Lemma 12.

Therefore, the naive implementation, i.e. by calling Algorithm 7 for every slice \( S \), will result in \( O(|D|^2) \) overall time, since many rows of \( D \) participate many times in such a call to Algorithm 7.

However, we can implement Line 11 in \( O(m) \) time, rather than \( O(|S|) \) time, by preprocessing the input signal \( D \), in linear \( O(|D|) \), and storing some statistics for every element \( ((i, j), y) \in D \). For example, one can store the sum of values and squared values over all elements \( ((i', j'), y') \) where \( i' < i \) or \( j' < j \). Using those values we can compute \( \text{opt}(B) \) in \( O(1) \) time for every sub-signal \( B \) of \( D \).

Now, using such statistics (and possibly more statistics), Line 11 can be implemented in \( O(m) \) time via a greedy algorithm that iterates over the points of the last row \( R = \{(i, j), y) \in D \mid i = r_{\text{end}}\} \) added to \( S \) (i.e. with no need to iterate over other elements of \( S \)). We leave the small details to the reader.

\[ \Box \]

## D Coreset Construction

In this section, we provide the proof of correctness for our main coreset construction algorithm presented in Algorithm 3, see Theorem 15. Furthermore, we provide an algorithm than gets as input a \( k \)-segmentation \( s \), as well as a \( (k, \varepsilon) \)-coreset for some input dataset \( D \), which was computed using Algorithm 5. The algorithm returns a \( (1 + \varepsilon) \)-approximation to the loss \( \ell(D, s) \), in \( O(k|C|) \) time; see Algorithm 6 and full details in Lemma 14.

In what follows, for an \( n \times m \) sub-signal \( B \) and a weight function \( u : B \to [0, \infty) \), we abuse notation and denote \( u((a, b)) \) by simply \( u(a, b) \) for \( (a, b) \in B \).

Some intuition behind Algorithm 5: Given a \( (k, \varepsilon) \)-coreset \((C, u)\) for an input dataset \( D = \{(x_1, y_1), \ldots, (x_N, y_N)\} \), and a \( k \)-segmentation \( s \), the algorithm outputs a \( (1 + \varepsilon) \)-approximation to \( \ell(D, s) \) in time that depends only on \( k \) and \( |C| \).

During the computation of \((C, u)\) in Algorithm 5 a partition \( B \) of \( D \) was computed. Then, for every set \( B \) in the partition \( B \), a representative pair \((C_B, u_B)\) for \( B \) was computed and added to \( C \).
To approximate the loss $\ell(D, s)$, we will approximate individually $\ell(B, s)$ for every $B \in \mathcal{B}$, and return the sum of those losses. Therefore, we now consider a single set $B \in \mathcal{B}$, and consider the following two cases.

**Case (i):** $s$ assigns the same value for all the elements of $B$. Then, by construction, it is guaranteed that $\ell(B, s) = \sum_{(x, y) \in B} u_B(x, y)(s(y) - y)^2$. Therefore, in this case, $\ell(B, s)$ will be accurately estimated using $(C_B, u_B)$.

**Case (ii):** $s$ assigns more than one unique value to the elements of $B$. In this case, if we ignore the computational time, we would ideally want to compute a “smoothed version” $(S, w)$ of $(C_B, u_B)$, as shown in Fig. 1 (see (9)-(11) below for formal details). Then, we would return the loss $\sum_{(x, y) \in S} w(x, y)(s(y) - y)^2$. However, computing $(S, w)$ is not necessary, since there are many subsets of $B$ in which all the elements $x \in B$ have simultaneously the same label in $S$ and are assigned the same value by $s$. Combining this with the fact that those subsets are of rectangular (simple) shape, we obtain that the loss over those subsets can be evaluated efficiently, as computed in Algorithm 5.

---

Algorithm 5: *Fitting-loss* $(C, u, s)$; see Lemma 14

**Input:** A $(k, \varepsilon)$-coreset $(C, u)$ which was returned from a call to $\text{SIGNAL-CORESET}(D, k, \varepsilon/\Delta)$ in Algorithm 3 for some $n \times m$-signal $D$, $k \geq 1$, $\varepsilon \in (0, 1)$ and a sufficiently large $\Delta \geq 1$. A $k$-segmentation (or $k$-tree) $s$.

**Output:** A $(1 + \varepsilon)$-approximation to the loss $\ell(D, s)$.

1. $\text{loss} := 0$
2. **for** every 4 consecutive elements $\hat{C} = \{(a_i, b_i)\}_{i=1}^4$ in $C'$ **do**
   3. Denote by $B$ the sub-signal that corresponds to $C'$. // By construction in Algorithm 3, the coordinates $a$ of the 4 elements $(a, b) \in \hat{C}$ are the corners of $B$.
   4. $z := |\{s(x) \mid (x, y) \in B\}|$
   5. **if** $z = 1$ // i.e., $s$ does not intersect $B$
   6. **then**
      7. $\text{loss}_C := \sum_{(x, y) \in C_B} u_B(x, y)(s(x) - y)^2$. // note that $s(x_1) = s(x_2)$ for every $x_1, x_2 \in B$
   8. **else**
       9. // In this case, $s$ intersects $B$
   10. Denote by $S$ the partition that $s$ induces onto $[n] \times [m]$. // $S$ contains $|S| \leq k$ subsets of $[n] \times [m]$.
   11. $i := 1$
   12. **for** every $S' \in S$ **do**
   13. Denote by $\ell$ the label that $s$ assigns to the elements of $S'$ i.e., $s(x, y) = \ell$ for every $(x, y) \in S'$
   14. $z := |B \cap S'|$ // The number of elements in the intersection of the $S'$ and the subset of $[n] \times [m]$ that is represented by $C'$.
   15. $\text{loss}_C := 0$
   16. **while** $z \geq 1$ **do**
   17. **if** $u(a_i, b_i) \leq z$ **then**
       18. $\text{loss}_C := \text{loss}_C + u(a_i, b_i) \cdot (\ell - b_i)^2$
       19. $u(a_i, b_i) := 0$
       20. $z := z - u(a_i, b_i)$
       21. $i := i + 1$
   22. **else**
       23. $\text{loss}_C := \text{loss}_C + z \cdot (\ell - b_i)^2$
       24. $u(a_i, b_i) := u(a_i, b_i) - z$
       25. $z := 0$
   26. **return** $\text{loss} := \text{loss} + \text{loss}_C$. 

---

24
Lemma 14. Let $D = \{(x_1, y_1), \cdots, (x_N, y_N)\}$ be an $n \times m$ signal i.e., $N := nm$. Let $k \geq 1$ be an integer, $\varepsilon \in (0, 1/4)$ be an error parameter, and $(C, u)$ be the output of a call to \textsc{Signal-Coreset}$(D, k, \varepsilon)$ (see Algorithm 3). Let $s$ be a $k$-segmentation (in particular, a $k$-tree). Finally, let $\text{loss}$ be the output of a call to \textsc{Fitting-Loss}($((C, u), s)$; see Algorithm 5. Then there is a sufficiently large constant $\Delta \geq 1$ such that

$$|\ell(D, s) - \text{loss}| \leq \Delta \varepsilon \cdot \ell(D, s).$$

Moreover, $\text{loss}$ can be computed in $O(k|C|)$ time.

Proof. We consider the variables defined in Algorithm 5.

First, consider a subset $\hat{C}$ of $C$ from some iteration of the For loop at Line 2 of Algorithm 5, and let $B$ be the sub-signal that corresponds to $\hat{C}$, as in Line 3. We now prove that the loss $\text{loss}_{\hat{C}}$, computed in the same iteration of the For loop i.e., at Lines 4-25 satisfies the following claim.

Claim 14.1. Let $z = |\{s(x) \mid (x, y) \in B\}|$ be the number of distinct values $s$ assigns to the coordinates of $B$ (as computed in Line 4). Then, $\text{loss}_{\hat{C}}$ satisfies that

\[
\begin{cases}
\ell(B, s) = \text{loss}_{\hat{C}} & \text{if } z = 1 \\
\ell(B, s) - \text{loss}_{\hat{C}} \leq \varepsilon \cdot \ell(B, s) + O\left(\frac{\text{opt}(B) - \varepsilon}{\varepsilon}\right) & \text{otherwise}
\end{cases}
\]

Proof. We prove Claim 14.1 using the following case analysis: (i) $z = 1$ and (ii) $z \geq 2$.

Case (i): $z = 1$. We prove that $\ell(B, s) = \text{loss}_{\hat{C}}$.

Since the input coreset $(C, u)$ was computed using Algorithm 3, we know that the set $\hat{C}$ was computed at Line 3 of Algorithm 3, along with a weight function $\hat{u}$. Hence, the pair $(\hat{C}, \hat{u})$ satisfy, by construction, the following property:

\[
\sum_{(a, b) \in \hat{C}} \hat{u}((a, b)) \cdot (b - \hat{s}) = \frac{1}{2} \sum_{(x, y) \in B} (y^2 - 1).
\]

(5)

Now, for any constant $\hat{s} \in \mathbb{R}$, we have that

\[
\sum_{(a, b) \in \hat{C}} \hat{u}(a, b)(b - \hat{s})^2
\]

\[
= \sum_{(a, b) \in \hat{C}} \hat{u}(a, b) \cdot b^2 + \hat{s}^2 \sum_{(a, b) \in \hat{C}} \hat{u}(a, b) - 2\hat{s} \sum_{(a, b) \in \hat{C}} \hat{u}(a, b)
\]

\[
= \sum_{(x, y) \in B} y^2 + \frac{1}{2} \sum_{(x, y) \in B} (y - \hat{s})^2 - 2\hat{s} \sum_{(x, y) \in B} y
\]

\[
= \sum_{(x, y) \in B} (y - \hat{s})^2,
\]

where the second equality is by (5).

Since $z = |\{s(x) \mid (x, y) \in B\}| = 1$, there is a constant $\hat{s} \in \mathbb{R}$ such that

\[
\ell(B, s) = \sum_{(x, y) \in B} (y - \hat{s})^2.
\]

(7)

Hence, we have that

\[
\ell(B, s) = \sum_{(x, y) \in B} (y - \hat{s})^2 = \sum_{(a, b) \in \hat{C}} \hat{u}(a, b)(b - \hat{s})^2 = \text{loss}_{\hat{C}},
\]

where the first derivation is by (7), the second derivation is by (6), and the last derivation is by the definition of $\text{loss}_{\hat{C}}$ at Line 7 of Algorithm 5.
Case (ii): \( z \geq 2 \). We prove that
\[
|\ell(B, s) - \text{loss}_C| \leq \varepsilon \cdot \ell(B, s) + O\left( \frac{\text{opt}_1(B)}{\varepsilon} \right).
\]

We first observe that, by the triangle inequality, for any \( a, b, c \in \mathbb{R} \) we have that
\[
||a - c||^2 - ||b - c||^2 = ||a - c||^2 - ||b - c||^2 = (a - b) \cdot (a - c) + (b - c)
\]
\[
\leq |a - b| \cdot (2|a - c| + |a - b|)
\]
\[
= |a - b|^2 + 2|a - c| \cdot |a - b|
\]
\[
= |a - b|^2 + 2\sqrt{\varepsilon}|a - c| \cdot \frac{|a - b|}{\sqrt{\varepsilon}}
\]
\[
\leq |a - b|^2 + \varepsilon \cdot |a - c|^2 + \frac{|a - b|^2}{\varepsilon}
\]
\[
= \varepsilon \cdot |a - c|^2 + \left( 1 + \frac{1}{\varepsilon} \right) \cdot (a - b)^2,
\]
where the second inequality holds since \( 2xy \leq x^2 + y^2 \) for every \( x, y \in \mathbb{R} \).

**Smoothed coreset.** In Algorithm 3 we computed some small compression \( C_B \), along with a weights function \( u_B \), for every subset \( B \) in the partition of the input. The size \( |C_B| \) of this compression is a small constant, independent of the (potentially large) size of \( B \). The pair \((C_B, u_B)\) satisfy a set of properties, which we visually demonstrate via this “smoothed coreset” notion; see Fig. 3. Informally, the “smoothed version” of \((C_B, u_B)\) is another pair \((C', u'_B)\), such that \( C' \) contains a duplication of the elements of \( C_B \). The number of duplications of every element \( c \) from \( C_B \) is according to its weight \( u_B(c) \).

We now formally define a “smoothed version” of a pair \((\hat{C}, \hat{u})\). A pair \((S, w)\) is said to be a smoothed version of the pair \((\hat{C}, \hat{u})\) if it satisfies the following properties: (i) \((S, w)\) has the same sum of weights, sum of labels, and sum of squared labels as \((\hat{C}, \hat{u})\), (ii) The set of coordinates \( \{a\,(a, b) \in S\} \) in \( S \) covers the entire set of coordinates \( \{x\,(x, y) \in B\} \) of the original set \( B \), with possible duplicates, and (iii) The sum of weights over all elements in \( S \) with the same coordinate is 1. Formally,
\[
\sum_{(a,b) \in S} w((a,b)) \cdot (b \mid b^2 \mid 1) = \sum_{(a,b) \in \hat{C}} \hat{u}((a,b))(b \mid b^2 \mid 1),
\]
\[
\{x\,(x, y) \in B\} = \{a\,(a, b) \in S\},
\]
and
\[
\sum_{(a,b) \in S, a = x} w((a,b)) = 1 \text{ for every } (x, y) \in B.
\]

In what follows, for every pair \((S, w)\) which is a smoothed version of \((\hat{C}, \hat{u})\), we prove the following two properties: We now prove the following two properties: first, that
\[
|\ell(B, s) - \ell((S, w), s)| \leq \varepsilon \cdot \ell(B, s) + O\left( \frac{\text{opt}_1(B)}{\varepsilon} \right),
\]
for every pair \((S, w)\) which is a smoothed version of \((\hat{C}, \hat{u})\). Second, we need to prove there is some pair \((\hat{S}, \hat{w})\) which is a smoothed version of \((\hat{C}, \hat{u})\) that satisfies
\[
\text{loss}_{\hat{C}} = \ell((\hat{S}, \hat{w}), s),
\]
where \( \text{loss}_{\hat{C}} \) is the loss computed at Lines 3-25 using only the pair \((\hat{C}, \hat{u})\) (i.e., at the current iteration of the outer-most For loop of Algorithm 5), without actually computing \((\hat{S}, \hat{w})\)). Case (ii) then immediately holds by combining (12) and (13) above.

**A proof of (12).** Let \((S, w)\) be a pair which is a smoothed version of \((\hat{C}, \hat{u})\). By definition of \((S, w)\), we have that \( w \) sums to 1 over all \((a, b) \in S\) with the same \( a \), as in (11). Therefore, for every
(x, y) ∈ B we can rewrite the term \((y - s(x))^2\) as \(\sum_{(a,b) \in S: a=x} w(a,b)(y - s(x))^2\). Now, define \(y_B(x) = y\) for every \((x, y) \in B\). We therefore have that

\[
\ell(B, s) = \sum_{(x, y) \in B} (y - s(x))^2
\]

\[
= \sum_{(x, y) \in B} \left( \sum_{(a,b) \in S: a=x} w(a,b) \right) \cdot (y - s(x))^2
\]

\[
= \sum_{(x, y) \in S} w(x, y) (y_B(x) - s(x))^2,
\]

where the last equality holds by (10) and by simply combining the two sums.

We now have that

\[
|\ell(B, s) - \ell((S, u), s)|
\]

\[
= \left| \sum_{(x, y) \in S} w(x, y)(y_B(x) - s(x))^2 - \sum_{(x, y) \in S} w(x, y)(y - s(x))^2 \right|
\]

\[
= \sum_{(x, y) \in S} w(x, y) \cdot ((y_B(x) - s(x))^2 - (y - s(x))^2)
\]

\[
\leq \sum_{(x, y) \in S} w(x, y) \cdot |(y_B(x) - s(x))^2 - (y - s(x))^2|
\]

\[
\leq \sum_{(x, y) \in S} w(x, y) \left( \varepsilon \cdot (y_B(x) - s(x))^2 + \left(1 + \frac{1}{\varepsilon}\right)(y_B(x) - y)^2 \right)
\]

\[
= \varepsilon \cdot \sum_{(x, y) \in S} u(x, y) \cdot (y_B(x) - s(x))^2 + \left(1 + \frac{1}{\varepsilon}\right) \sum_{(x, y) \in S} w(x, y) \cdot (y_B(x) - y)^2
\]

\[
= \varepsilon \cdot \ell(B, s) + \left(1 + \frac{1}{\varepsilon}\right) \sum_{(x, y) \in S} u(x, y) \cdot (y_B(x) - y)^2,
\]

where (15) is by combining (14) and the definition of \(\ell\), (16) holds since the sum of absolute values is greater or equal than the absolute value of a sum, (17) holds by substituting in (8) every term in the sum, and (18) is by (14).

We now bound the rightmost term of (18). Let \(\hat{s} \equiv 1/|B| \sum_{(x, y) \in B} y\) be a 1-segmentation function that returns the label mean of \(B\). We have that

\[
\sum_{(x, y) \in S} w(x, y) \cdot (y_B(x) - y)^2
\]

\[
\leq 2 \cdot \sum_{(x, y) \in S} w(x, y) \cdot ((y_B(x) - \hat{s}(x))^2 + (y - \hat{s}(x))^2)
\]

\[
= 2 \sum_{(x, y) \in S} w(x, y) \cdot (y_B(x) - \hat{s}(x))^2
\]

\[
+ 2 \sum_{(x, y) \in S} w(x, y) \cdot (y - \hat{s}(x))^2
\]

\[
= 2 \cdot (\ell(B, \hat{s}) + \ell((S, u), \hat{s}))
\]

\[
= 2 \cdot (\ell(B, \hat{s}) + \ell(B, \hat{s}))
\]

\[
= 4 \cdot \ell(B, \hat{s})
\]

\[
= 4 \cdot \text{opt}_1(B)
\]

where (19) is by the weak triangle inequality, (20) is by combining the definition of \(\ell\) with (14), (21) holds by Case (i) above, and (22) holds since the label means minimizes its sum of squared differences to the labels.
We now prove Lemma 14. Consider the construction of \((C, u)\) in Algorithm 5. By definition and Lemma 5, the function \(s'\) computed at Line 5 of Algorithm 5 is an \(O(k^8 \log nm)\)-segmentation and satisfies that
\[
\ell(D, s') \in O(k \log n \cdot \text{opt}_k(D)) .
\]
By the last derivation, let \(c_\alpha\) be the smallest constant such that
\[
\ell(D, s') \leq c_\alpha k \log n \cdot \text{opt}_k(D).
\]
By the last inequality and definitions of \(\sigma\) and \(\alpha\) we obtain that
\[
\sigma := \frac{\ell(D, s')}{\alpha} \leq \frac{c_\alpha k \log nm \cdot \text{opt}_k(D)}{\alpha} = \frac{c_\alpha k \log nm \cdot \text{opt}_k(D)}{c_\alpha k \log nm} = \text{opt}_k(D).
\]
Now consider the partition \(\mathcal{B}\) computed at Line 3 of Algorithm 5 via a call to \(\text{PARTITION}(D, \gamma, \sigma)\). By Lemma 7, \(\mathcal{B}\) satisfies that
\[
\begin{align*}
(i) \quad & \text{opt}_1(B) \leq \gamma^2 \sigma \text{ for every } B \in \mathcal{B}. \\
(ii) \quad & \mathcal{B} \text{ is a partition of } D \text{ whose size is } |\mathcal{B}| = O\left(\frac{\alpha}{\gamma^2} \right).
\end{align*}
\]

(iii) There are \( O \left( \frac{k \alpha}{\gamma} \right) \) sub-signals \( B \in \mathcal{B} \) where \( |\{ s(x) \mid (x, y) \in B \}| > 1 \).

(iv) \( \mathcal{B} \) can be computed in \( O(|\mathcal{D}|) \) time.

Consider the pair \( (C_B, u_B) \) computed at Line 5 of Algorithm 3 for some \( B \in \mathcal{B} \). Now, consider the For loop at Line 2 of Algorithm 3. For every pair \( (C_B, u_B) \), there is an iteration of this For loop for which \( C' = C_B \). In this iteration, Algorithm 5 computes a loss \( \text{loss}_{C_B} \) that corresponds to \( (C_B, u_B) \).

We can plug \( B, (C_B, u_B), \) and \( \text{loss}_{C_B} \) in Claim 14.1 to obtain that:

\[
\begin{align*}
\ell(B, s) &= \text{loss}_{C_B} \\
\ell(B, s) - \text{loss}_{C_B} &\leq \varepsilon \cdot \ell(B, s) + O \left( \frac{\text{opt}_1(B)}{\varepsilon} \right) \quad \text{if } z = 1 \\
\ell(B, s) - \text{loss}_{C_B} &\leq \varepsilon \cdot \ell(B, s) + O \left( \frac{\gamma^2 \sigma}{\varepsilon} \right) \quad \text{otherwise}
\end{align*}
\]

Let \( B_1 \subseteq B \) contain the set of sub-signals in \( B \) that are not intersected by \( s \), i.e., \( B_1 = \{ B \in \mathcal{B} \mid |s(B)| = 1 \} \), and let \( B_2 = B \setminus B_1 \) be the set of sub-signals which are partially intersected by \( s \).

By Property (iii) above,

\[ |B_2| \leq O \left( \frac{k \alpha}{\gamma} \right). \tag{25} \]

Furthermore, by combining (24) with Property (i) above, for every \( B \in B_2 \) we have that

\[ |\ell(B, s) - \text{loss}_{C_B}| \leq \varepsilon \cdot \ell(B, s) + O \left( \frac{\text{opt}_1(B)}{\varepsilon} \right) \leq \varepsilon \cdot \ell(B, s) + O \left( \frac{\gamma^2 \sigma}{\varepsilon} \right). \tag{26} \]

In other words, the loss \( \ell(B, s) \) of every sub-signal \( B \in B_2 \) is approximated by \( \text{loss}_{C_B} \) up to some small error. Hence, by summing over all \( B \in B_2 \) we obtain that

\[
\sum_{B \in B_2} |\ell(B, s) - \text{loss}_{C_B}| \\
\leq \sum_{B \in B_2} \left( \varepsilon \cdot \ell(B, s) + O \left( \frac{\text{opt}_1(B)}{\varepsilon} \right) \right) \\
\leq \varepsilon \cdot \ell(D, s) + O \left( |B_2| \cdot \frac{\gamma^2 \sigma}{\varepsilon} \right) \tag{27} \\
\leq \varepsilon \cdot \ell(D, s) + O \left( \frac{k \alpha}{\gamma} \cdot \frac{\gamma^2 \sigma}{\varepsilon} \right) \\
\leq \varepsilon \cdot \ell(D, s) + O \left( \frac{\text{opt}_k(D)}{\varepsilon} \cdot \text{opt}_k(D) \right) \tag{28} \\
\leq \varepsilon \cdot \ell(D, s) + O(\varepsilon \cdot \text{opt}_k(D)) \tag{29} \\
\leq \ell(D, s). \tag{30}
\]

where (27) follows from (26), (28) is by (25), (29) is by (23), and (30) holds since \( k \alpha \gamma \leq \varepsilon^2 \), and (31) holds since \( \text{opt}_k(D) \leq \ell(D, s) \) for every \( k \)-segmentation \( s \).

Furthermore, for every \( B \in B_1 \), by (24) we have that \( \ell(B, s) = \text{loss}_{C_B} \). Hence, by summing over every \( B \in B_1 \) we obtain that

\[ \sum_{B \in B_1} \ell(B, s) = \sum_{B \in B_1} \text{loss}_{C_B}. \tag{32} \]

In other words, the loss \( \ell(B, s) \) of every sub-signal \( B \in B_1 \) is accurately estimated by \( \text{loss}_{C_B} \).

Algorithm 5 then outputs the sum of losses

\[ \text{loss} := \sum B \in \mathcal{B} \text{loss}_{C_B}. \tag{33} \]

We hence obtain that

\[
|\ell(D, s) - \text{loss}| = \left| \sum_{B \in B_1} (\ell(B, s) - \text{loss}_{C_B}) + \sum_{B \in B_2} (\ell(B, s) - \text{loss}_{C_B}) \right| \\
\leq \left| \sum_{B \in B_1} (\ell(B, s) - \text{loss}_{C_B}) \right| + \left| \sum_{B \in B_2} (\ell(B, s) - \text{loss}_{C_B}) \right| \in O(\varepsilon \cdot \ell(D, s)), \tag{34}
\]
where the first derivation is by \[^{33}\] second derivation is by the triangle inequality, and the last is by combining \(^{32}\) and \(^{31}\).

By \(^{34}\), there is a constant \(\Delta \geq 1\) such that
\[|\ell(D, s) - \text{loss}| \leq \Delta \epsilon t(D, s).\]

This concludes the proof of the claim in Lemma \(^{14}\).

**Computational time:** Line \(^2\) of Algorithm \(^5\) is a loop with \(|C|/|P|\) iterations. Inside this loop: if \(z > 1\) line \(^7\) is computed in \(O(|P|)\), else line \(^11\) is another loop with \(O(k)\) iterations, inside which the line \(^20\) is executed at most \(|C|\) times and line \(^24\) can be executed only once because it results in \(z = 0\) and in exiting from the while loop.

In total the complexity of line \(^15\) is \(O(|C|)\), of line \(^11\) is \(O(k|C|)\) and of line \(^2\) and the whole algorithm: \(O(k|C|)\)

**Space complexity:** Algorithm \(^5\) uses only constant amount of additional storage space because in each line of the algorithm only numeric variables are created and variables are reused inside the loops.

\[\square\]

**Theorem 15** (Coreset). Let \(D = \{(x_1, y_1), \ldots, (x_N, y_N)\}\) be an \(n \times m\) signal i.e., \(N := nm\). Let \(k \geq 1\) be an integer (that corresponds to the number of leaves/rectangles), and \(\epsilon \in (0, 1/4)\) be an error parameter. Let \((C, u)\) be the output of a call to SIGNAL-CORESET \((D, k, \epsilon, \Delta)\) for a sufficiently large constant \(\Delta \geq 1\); see Algorithm \(^3\). Then, \((C, u)\) is a \((k, \epsilon)\)-coreset for \(D\) of size \(|C| \in \left(\frac{\log(N)^{O(1)}}{\epsilon^4}\right)\); see Definition \(^3\). Moreover, \((C, u)\) can be computed in \(O(kN)\) time.

**Proof.** To prove that \((C, u)\) is a \((k, \epsilon)\)-coreset for \(D\), we need to prove that for every \(k\)-segmentation \(s\), \((C, u)\) suffices to approximate the loss \(\ell(D, s)\), up to a multiplicative factor of \(1 + \epsilon\), in time that depends only on \(|C|\) and \(k\).

Let \(s\) be a \(k\)-segmentation and let \(\text{loss} \geq 0\) be an output of a call to FITTING-LOSS \(((C, u), s)\); see Algorithm \(^5\). Then, by Lemma \(^{14}\), \(\text{loss}\) can be computed in \(O(|kC|)\) time (i.e., in time that depends only on \(|C|\) and \(k\)), and provides, as required, a \((1 + \epsilon)\)-approximation to \(\ell(D, s)\) as
\[|\ell(D, s) - \text{loss}| \leq \Delta \cdot \frac{\epsilon}{\Delta} \cdot \ell(D, s) = \epsilon \cdot \ell(D, s).\]

Hence, \((C, u)\) is a \((k, \epsilon)\)-coreset for \(D\).

Line \(^1\) of Algorithm \(^3\) can be computed in \(O(k \cdot |D|)\) time by Lemma \(^5\). Line \(^2\) of Algorithm \(^3\) can be computed in \(O(|D|)\) time by Lemma \(^7\). The loop at Line \(^3\) can be computed in \(\sum_{B \in B} O(|B|) = O(|D|)\) time by Section \(^3\). Hence, the call SIGNAL-CORESET \((D, k, \epsilon)\) can be implemented in \(O(kmn)\) time.

**Proof behind Line \(^6\)**. We now prove that the replacements of the coordinates applied at Line \(^6\) does not violate the correctness of the algorithm. Observe that replacing the coordinates of entries inside each cell, while keeping the same labels, does not affect the variance of this subset. Therefore, the cost of this cell, which is computed in Algorithm \(^5\) and depends only on the labels, remains exactly the same.

**Space complexity:** By construction, each pair \((C_B, u_B)\) computed at Line \(^5\) can be stored using only \(O(1)\) space. Hence, the concatenation \((C, u)\) of the \(|B|\) pairs \(|\{(C_B, u_B) | B \in B\}| = O(\log(N)^{O(1)}) \times (\epsilon^4) = O\left(\frac{k^{O(1)} \log(N)^{O(1)}}{\epsilon^4}\right)\) space.

\[\square\]

**E The Caratheodory Theorem**

Given a point \(p \in \mathbb{R}^d\) inside the convex hull of a set of points \(P \subseteq \mathbb{R}^d\), Caratheodory’s Theorem proves that there is a subset of at most \(d + 1\) points in \(P\) whose convex hull also contains \(p\).

**Theorem 16** (Caratheodory’s Theorem \([13, 48]\)). Let \(P \subseteq \mathbb{R}^d\) be a (multi)set of \(n\) points. Then in \(O(n^d)\) time we can compute a subset \(Q \subseteq P\) and a weights functions \(u : Q \rightarrow [0, \infty)\) such that: (i) \(Q \subseteq P\), (ii) \(|Q| = d + 1\), (iii) \(\sum_{q \in Q} u(q) \cdot q = \frac{1}{n} \sum_{p \in P} p\), and (iv) \(\sum_{q \in Q} u(q) = n\).
Corollary 17. Let $D$ be an $n \times m$ sub-signal. Then, in $O(|D|)$ time we can compute a weighted $n \times m$ sub-signal $(A, w)$ such that: (i) $A \subseteq D$, (ii) $|A| = 4$, (iii) $\sum_{(a,b) \in A} w(a,b) \cdot (b \mid b^2 \mid 1) = \sum_{(x,y) \in D} (y \mid y^2 \mid 1)$, and (iv) $\sum_{(a,b) \in A} w(a,b) = |D|$.

Proof. Define the multi-set $P = \{(y \mid y^2 \mid 1) \in \mathbb{R}^3 \mid (x,y) \in D\}$. Now, substituting $P$ in Theorem 16 yields that in $O(n)$ time we can compute a subset $Q \subseteq P$ and a weights functions $u : Q \to [0, \infty)$ such that: (i) $Q \subseteq P$, (ii) $|Q| = 4$, (iii) $\sum_{q \in Q} u(q) \cdot q = \frac{1}{|D|} \sum_{p \in P} p$, and (iv) $\sum_{q \in Q} u(q) = |D|$.

Now, add to $A$ a single element $(x,y) \in D$ for every $(y \mid y^2 \mid 1) \in Q$. In other words, for every element chosen for the set $Q$ by the Caratheodory theorem, add its corresponding element from $D$ to $A$. Furthermore, define $w(x,y) = |D| \cdot u((y \mid y^2 \mid 1))$ for every $(x,y) \in A$. Corollary 17 trivially holds for $(A, w)$.

\qed