



Range Entropy Queries and Partitioning

Sanjay Krishnan  

Department of Computer Science, University of Chicago, IL, USA

Stavros Sintos  

Department of Computer Science, University of Illinois at Chicago, IL, USA

Abstract

Data partitioning that maximizes or minimizes Shannon entropy is a crucial subroutine in data compression, columnar storage, and cardinality estimation algorithms. These partition algorithms can be accelerated if we have a data structure to find the entropy in different subsets of data when the algorithm needs to decide what block to construct. While it is generally known how to compute the entropy of a discrete distribution efficiently, we want to efficiently derive the entropy among the data items that lie in a specific area. We solve this problem in a typical setting when we deal with real data, where data items are geometric points and each requested area is a query (hyper)rectangle. More specifically, we consider a set P of n weighted and colored points in \mathbb{R}^d . The goal is to construct a low space data structure, such that given a query (hyper)rectangle R , it computes the entropy based on the colors of the points in $P \cap R$, in sublinear time. We show a conditional lower bound for this problem proving that we cannot hope for data structures with near-linear space and near-constant query time. Then, we propose exact data structures for $d = 1$ and $d > 1$ with $o(n^{2d})$ space and $o(n)$ query time. We also provide a tune parameter t that the user can choose to bound the asymptotic space and query time of the new data structures. Next, we propose near linear space data structures for returning either an additive or a multiplicative approximation of the entropy. Finally, we show how we can use the new data structures to efficiently partition time series and histograms with respect to entropy.

2012 ACM Subject Classification Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases Shannon entropy, range query, data structure, data partitioning

Digital Object Identifier 10.4230/LIPIcs.ICDT.2024.6

Related Version *Full Version:* <https://arxiv.org/abs/2312.15959> [27]

1 Introduction

Discrete entropy is defined as the expected amount of information needed to represent an event drawn from a probability distribution. That is, given a probability distribution \mathcal{D} over the set \mathcal{X} , the entropy is defined as $H(\mathcal{D}) = -\sum_{x \in \mathcal{X}} \mathcal{D}(x) \cdot \log \mathcal{D}(x)$. The entropy has a few different interpretations in information theory and statistics, such as:

- (Compression) Entropy is a lower-bound on data compressibility for datasets generated from the probability distribution via the Shannon source coding theorem.
- (Probability) Entropy measures a probability distribution's similarity to a uniform distribution over the set \mathcal{X} on a scale of $[0, \log |\mathcal{X}|]$.

Because of these numerous interpretations, entropy is a highly useful optimization objective. Various algorithms, ranging from columnar compression algorithm to histogram construction and data cleaning, maximize or minimize (conditional) entropy as a subroutine. These algorithms try to find high or low entropy data subsets. Such algorithms can be accelerated if we have a data structure to efficiently calculate the entropy of different subsets of data. However, while it is known how to compute the entropy of an entire distribution efficiently, there is a little work on such “range entropy queries”, where we want to derive efficiently the entropy among the data items that lie in a specific area. To make this problem more concrete, let us consider a few examples.



© Sanjay Krishnan and Stavros Sintos;

licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 6; pp. 6:1–6:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

► **Example 1** (Columnar Compression). An Apache Parquet file is a columnar storage format that first horizontally partitions a table into row groups, and then applies columnar compression along each column within the row group. A horizontal partitioning that **minimizes the entropy** within each partition can allow for more effective columnar compression.

► **Example 2** (Histogram Construction). Histogram estimation often uses a uniformity assumption, where the density within a bucket is modeled as roughly uniform. A partitioning that **maximizes the entropy** within each partition can allow for more accurate estimation under uniformity assumptions.

► **Example 3** (Data Cleaning). As part of data exploration, a data analyst explores different subsets of data to find areas with high entropy/uncertainty. Usually, subsets of data or items in a particular area of the data with high entropy contain dirty data so they are good candidates for applying data cleaning methods. For example, Chu et al. [16] used an entropy-based scheduling algorithm to maximize the uncertainty reduction of candidate table patterns. Table patterns are used to identify errors in data.

The first two problems above have a similar structure, where an outer-algorithm leverages a subroutine that identifies data partitions that minimize or maximize entropy. In the third problem we aim to explore areas with high entropy by running arbitrary range entropy queries. We formulate the problem of range entropy query problem in a typical and realistic setting when we deal with real data: we assume that each item is represented as a point in the Euclidean space. More specifically, we consider a set P of n weighted and colored points in \mathbb{R}^d . The goal is to construct a data structure, such that given a query (hyper)rectangle R , compute the entropy of the points in $P \cap R$ (denoted by $H(P \cap R)$). The entropy of $P \cap R$ is defined as the entropy of a discrete distribution \mathcal{D}_R over the colors in $P \cap R$: Let U_R be the set of all colors of the points in $P \cap R$. For each color $u_j \in U_R$, we define a value (we can also refer to it as an independent event or outcome) α_j with probability w_j equal to the sum of weights of points with color u_j in $P \cap R$ divided by the sum of the weights of all points in $P \cap R$. Notice that $\sum_{u_j \in U_R} w_j = 1$. Unfortunately, we do not have direct access to this distribution; we would need $\Omega(n)$ time to construct the entire distribution \mathcal{D}_R in the query phase. Using the geometry of the points along with key properties from information theory we propose data structures to find the entropy of \mathcal{D}_R without constructing \mathcal{D}_R explicitly.

► **Definition 4** (Range entropy query problem). *Given a set P of n weighted and colored points in \mathbb{R}^d , the goal is to construct a data structure with low space such that given any query rectangle R , it returns $H(P \cap R)$ in sub-linear time $o(|P|)$.*

If the number of colors in P is bounded by a constant then the range entropy query problem can be easily solved. However, in the worse case the number of different colors is $O(n)$. Our goal is to construct data structures whose query time is always sublinear with respect to n .

Summary of Results. One of the main challenges with range entropy queries is that entropy is not a *decomposable* quantity. Let P_1, P_2 be two sets of points such that $P_1 \cup P_2 = P$ and $P_1 \cap P_2 = \emptyset$. If we know $H(P_1), H(P_2)$ there is no straightforward way to compute $H(P_1 \cup P_2)$. In this paper, we build low space data structures such that given a rectangle R , we visit points or subsets of points in $P \cap R$ in a particular order and carefully update the overall entropy. All our results for the range entropy problem can be seen in Table 1.

- In Section 2 we introduce some useful notation and we revisit a way to update the entropy of the union of two sets with no color in common in $O(1)$ time.

■ **Table 1** New results (lower bound in the first row and data structures with their complexities in the next rows). $t \in [0, 1]$ is a tune parameter. $\tilde{O}(\cdot)$ notation hides a $\log^{O(1)} n$ factor, where the $O(1)$ exponent is at most linear on d . $Q(n)$ is any function of n that represents the query time of a data structure storing n items.

Type	Space	Query Time	Preprocessing
Lower bound	$\tilde{\Omega}\left(\left(\frac{n}{Q(n)}\right)^2\right)$	$\tilde{O}(Q(n))$	–
$d = 1$, exact	$O(n^{2(1-t)})$	$\tilde{O}(n^t)$	$O(n^{2-t})$
$d > 1$, exact	$\tilde{O}(n^{(2d-1)t+1})$	$\tilde{O}(n^{1-t})$	$\tilde{O}(n^{(2d-1)t+1})$
$d \geq 1$, Δ -additive approx.	$\tilde{O}(n)$	$\tilde{O}\left(\frac{1}{\Delta^2}\right)$	$\tilde{O}(n)$
$d \geq 1$, $(1 + \varepsilon)$ -multiplicative approx.	$\tilde{O}(n)$	$\tilde{O}\left(\frac{1}{\varepsilon^2}\right)$	$\tilde{O}(n)$
$d = 1$, ε -additive and $(1 + \varepsilon)$ -multiplicative approx.	$\tilde{O}\left(\frac{n}{\varepsilon}\right)$	$\tilde{O}(1)$	$\tilde{O}\left(\frac{n}{\varepsilon}\right)$

- In Section 3, we reduce the set intersection problem to the range entropy query problem in \mathbb{R}^2 . We prove a conditional lower bound showing that we cannot hope for $O(n \text{ polylog } n)$ space and $O(\text{polylog } n)$ query time data structures for the range entropy queries.
- Exact data structure for $d = 1$. In Section 4.1, we efficiently partition the input points with respect to their x coordinates into buckets, where each bucket contains a bounded number of points. Given a query interval R , we visit the bounded number of points in buckets that are partially intersected by R and we update the overall entropy of the buckets that lie completely inside R . For any parameter t chosen by the user, we construct a data structure in $O(n^{2-t})$ time, with $O(n^{2(1-t)})$ space and $O(n^t \log n)$ query time.
- In Section 4.2, instead of partitioning the points with respect to their geometric location, we partition the input points with respect to their colors. We construct $O(n^{1-t})$ blocks where two sequential blocks contain at most one color in common. Given a query rectangle we visit all blocks and we carefully update the overall entropy. For any tune parameter t chosen by the user, we construct a data structure in $O(n \log^{2d} n + n^{(2d-1)t+1} \log^{d+1} n)$ time with $O(n \log^{2d-1} n + n^{(2d-1)t+1})$ space and $O(n^{1-t} \log^{2d} n)$ query time.
- Additive approximation. In Subsection 5.1 we use known results for estimating the entropy of an unknown distribution by sampling in the *dual access model*. We propose efficient data structures that apply sampling in a query range in the dual access model. We construct a data structure in $O(n \log^d n)$ time, with $O(n \log^{d-1} n)$ space and $O\left(\frac{\log^{d+3} n}{\Delta^2}\right)$ query time. The data structure returns an additive Δ -approximation of the entropy with high probability. It also supports dynamic updates in $O(\log^d n)$ time.
- Multiplicative approximation. In Subsection 5.2 we propose a multiplicative approximation of the entropy using the results for estimating the entropy in a streaming setting. One significant difference with the previous result is that in information theory at least $\Omega\left(\frac{\log n}{\varepsilon^2 \cdot H'}\right)$ sampling operations are needed to find get an $(1 + \varepsilon)$ -multiplicative approximation, where H' is a lower bound of the entropy. Even if we have efficient data structures for sampling (as we have in additive approximation) we still do not have an efficient query time if the real entropy H is extremely small. We overcome this technical issue by considering two cases: i) there is no color with total weight more than $2/3$, and ii) there exists a color with total weight at most $2/3$. While in the latter case the entropy can be extremely small, an additive approximation is sufficient in order to get a multiplicative approximation. In the former one, the entropy is large so we apply the standard sampling method to get a multiplicative approximation. We construct a data structure in

$O(n \log^d n)$ time, with $O(n \log^d n)$ space and $O\left(\frac{\log^{d+3}}{\varepsilon^2}\right)$ query time. The data structure returns a multiplicative $(1 + \varepsilon)$ -approximation of the entropy. It also supports dynamic updates in $O(\log^d n)$ time.

- Additive and multiplicative approximation. In Subsection 5.3, we propose a new data structure for approximating the entropy in the query range for $d = 1$. We get the intuition from data structures counting the number of colors in a query interval. Such a data structure finds a geometric mapping to a different geometric space, such that if at least a point with color u_i exists in the original $P \cap R$, then there is a unique point with color u_i in the corresponding query range in the new geometric space. Unfortunately, this property is not sufficient for finding the entropy. Instead, we need to know more information about the weights of the points and the entropy in canonical subsets of the new geometric space, which is challenging to do. We construct a data structure in $O\left(\frac{n}{\varepsilon} \log^5 n\right)$ time, with $O\left(\frac{n}{\varepsilon} \log^2 n\right)$ space and $O\left(\log^2 n \log \frac{\log n}{\varepsilon}\right)$ query time. The data structure returns an $(1 + \varepsilon)$ -multiplicative and ε -additive approximation of the entropy.
- Partitioning using entropy. In Section 6 we show how our new data structures can be used to run partitioning algorithms over time series, histograms, and points efficiently.

Related work. Entropy has been used a lot for partitioning to create histograms in databases. For example, To et al. [38] used entropy to design histograms for selectivity estimation queries. In particular, they aim to find a partitioning of k buckets in 1d such that the cumulative entropy is maximized. They consider a special case where they already have a histogram (so all items of the same color are accumulated to the same location) and the goal is to partition the histogram into k buckets. They propose a greedy algorithm that finds a local optimum solution. However there is no guarantee on the overall optimum partitioning. Using our new data structures, we can find the entropy in arbitrary range queries, which is not supported in [38]. Our data structures can also be used to accelerate partitioning algorithms with theoretical guarantees (see Subsection 6) in a more general setting, where points of the same color have different locations.

In addition, there is a number of papers that use entropy to find a clustering of items. Cruz et al. [19] used entropy for the community detection problem in augmented social networks. They describe a greedy algorithm that exchanges two random nodes between two random clusters if the entropy of the new instance is lower. Barbará et al. [6] used the *expected entropy* for categorical clustering. They describe a greedy algorithm that starts with a set of initial clusters, and for each new item decides to place it in the cluster that has the lowest entropy. Li et al. [29] also used the expected entropy for categorical clustering but they extend it to probabilistic clustering models. Finally, Ben-Gal et al. [8] used the expected entropy to develop an entropy-based clustering measure that measures the homogeneity of mobility patterns within clusters of users. All these methods do not study the problem of finding the entropy in a query range efficiently. While these methods perform well in practice, it is challenging to derive theoretical guarantees. In spatial databases items are represented as points in \mathbb{R}^d , so our new data structures could be used to find faster and better entropy-based clustering techniques. For example, we could run range entropy queries with different radii around a center until we find a cluster with small radius and small (or large) expected entropy.

There is a lot of work on computing an approximation of the entropy in the streaming setting [11, 15, 23, 28]. For a stream of m distinct values (m colors in our setting) Chakrabarti et al. [14] compute an $(1 + \varepsilon)$ -multiplicative approximation of the entropy in a single pass using $O(\varepsilon^{-2} \log(\delta^{-1}) \log m)$ words of space, with probability at least $1 - \delta$. For a stream

of size n (n points in our setting) Clifford and Cosma [17] propose a single-pass ε -additive algorithm using $O(\varepsilon^{-2} \log n \log(n\varepsilon^{-1}))$ bits with bounded probability. Harvey et al. [25] allow deletions in the streaming setting and they propose a single-pass $(1 + \varepsilon)$ -multiplicative algorithm using $\tilde{O}(\varepsilon^{-2} \log^2 m)$ words of space with bounded probability. Furthermore, they propose a single-pass ε -additive approximation using $\tilde{O}(\varepsilon^{-2} \log m)$ words of space. While some techniques from the streaming setting are useful in our query setting, the two problems are fundamentally different. In the streaming setting, preprocessing is not allowed, all data are processed one by one and an estimation of the entropy is maintained. In our setting, the goal is to construct a data structure such that given any query range, the entropy of the items in the range should be computed in sublinear time, i.e., without processing all items in the query range during the query phase.

Let \mathcal{D} be an unknown discrete distribution over n values. There is an interesting line of work on approximating the entropy of \mathcal{D} by sampling in the *dual access model*. Batu et al. [7] give an $(1 + \varepsilon)$ -multiplicative approximation of the entropy of \mathcal{D} with sample complexity $O(\frac{(1+\varepsilon)^2 \log^2 n}{\varepsilon^2 \cdot H'})$, where H' is a lower bound of the actual entropy $H(\mathcal{D})$. Guha et al. [23] improved the sample complexity to $O(\frac{\log n}{\varepsilon^2 \cdot H'})$, matching the lower bound $\Omega(\frac{\log n}{(2+\varepsilon)\varepsilon^2 \cdot H'})$ found in [7]. Canonne and Rubinfeld [13] describe a Δ -additive approximation of the entropy with sample complexity $O(\frac{\log^2 n}{\Delta^2})$. Caferov et al. [12] show that $\Omega(\frac{\log^2 n}{\Delta^2})$ sample queries are necessary to get Δ -additive approximation. All these algorithms return the correct approximations with constant probability. If we want to guarantee the result with high probability then the sample complexity is multiplied by a $\log n$ factor.

A related query to estimating the entropy is the range color query. Given a set of colored points in \mathbb{R}^d , the goal is to construct a data structure such that given a query rectangle, it returns the number of colors in the query range.

2 Preliminaries

Let P be a set of n points in \mathbb{R}^d and let U be a set of m colors $U = \{u_1, \dots, u_m\}$. Each point $p \in P$ is associated with a color from U , i.e., $u(p) = u_i$ for $u_i \in U$. Furthermore, each point $p \in P$ is associated with a non-negative weight $w(p) \geq 0$. For a subset of points $P' \subseteq P$, let $P'(u_i) = \{p \in P' \mid u(p) = u_i\}$, for $i \leq m$, be the set of points having color u_i . Let $u(P') = \{u_i \mid \exists p \in P', u(p) = u_i\}$ be the set of colors of the points in P' . Finally, let $w(P') = \sum_{p \in P'} w(p)$. The entropy of set P' is defined as $H(P') = \sum_{i=1}^m \frac{w(P'(u_i))}{w(P')} \log \left(\frac{w(P')}{w(P'(u_i))} \right)$.

For simplicity, and without loss of generality, we can consider throughout the paper that $w(p) = 1$ for each point $p \in P$. All the results, proofs, and properties we show hold for the weighted case almost verbatim. Hence, from now on, we assume $w(p) = 1$ and the definition of entropy becomes

$$H(P') = \sum_{i=1}^m \frac{|P'(u_i)|}{|P'|} \log \left(\frac{|P'|}{|P'(u_i)|} \right) = \sum_{u_i \in u(P')} \frac{|P'(u_i)|}{|P'|} \log \left(\frac{|P'|}{|P'(u_i)|} \right). \quad (1)$$

If $|P'(u_i)| = 0$, then we consider that $\frac{|P'(u_i)|}{|P'|} \log \left(\frac{|P'|}{|P'(u_i)|} \right) = 0$.

Updating the entropy. Let $P_1, P_2 \subset P$ be two subsets of P such that $u(P_1) \cap u(P_2) = \emptyset$. The next formula for the entropy of $P_1 \cup P_2$ is known (see [38])

$$H(P_1 \cup P_2) = \frac{|P_1|H(P_1) + |P_2|H(P_2) + |P_1| \log \left(\frac{|P_1| + |P_2|}{|P_1|} \right) + |P_2| \log \left(\frac{|P_1| + |P_2|}{|P_2|} \right)}{|P_1| + |P_2|}. \quad (2)$$

6:6 Range Entropy Queries and Partitioning

If $|u(P_2)| = 1$ then

$$H(P_1 \cup P_2) = \frac{|P_1|H(P_1)}{|P_1| + |P_2|} + \frac{|P_1|}{|P_1| + |P_2|} \log \left(\frac{|P_1| + |P_2|}{|P_1|} \right) + \frac{|P_2|}{|P_1| + |P_2|} \log \left(\frac{|P_1| + |P_2|}{|P_2|} \right). \quad (3)$$

Finally, if $P_3 \subset P_1$ with $|u(P_3)| = 1$ and $u(P_1 \setminus P_3) \cap u(P_3) = \emptyset$ then

$$H(P_1 \setminus P_3) = \frac{|P_1|}{|P_1| - |P_3|} \left(H(P_1) - \frac{|P_3|}{|P_1|} \log \frac{|P_1|}{|P_3|} - \frac{|P_1| - |P_3|}{|P_1|} \log \frac{|P_1|}{|P_1| - |P_3|} \right). \quad (4)$$

We notice that in all cases, if we know $H(P_1), H(P_2)$ and the cardinality of each subset we can update the entropy in $O(1)$ time.

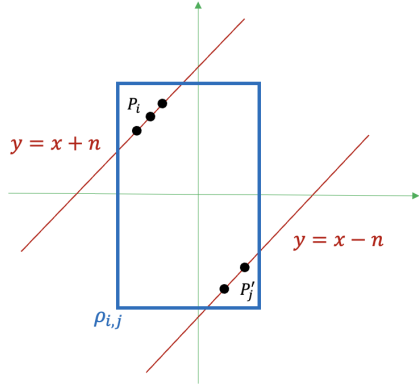
Range queries. In some data structures we need to handle range reporting or range counting problems. Given P , we need to construct a data structure such that given a query rectangle R , the goal is to return $|R \cap P|$, or report $R \cap P$. We use range trees [10]. A range tree can be constructed in $O(n \log^d n)$ time, it has $O(n \log^{d-1} n)$ space and can answer an aggregation query (such as count, sum, max etc.) in $O(\log^d n)$ time. A range tree can be used to report $R \cap P$ in $O(\log^d n + |R \cap P|)$ time. Using *fractional cascading* the $\log^d n$ term can be improved to $\log^{d-1} n$ in the query time. However, for simplicity, we consider the simple version of a range tree without using fractional cascading. Furthermore, a range tree can be used to return a uniform sample point from $R \cap P$ in $O(\log^d n)$ time. We give more details about range trees and sampling in the full version of the paper [27]. There is also lot of work on designing data structures for returning k independent samples in a query range efficiently [1, 2, 26, 32, 37, 39, 40]. For example, if the input is a set of points in \mathbb{R}^d and the query range is a query hyper-rectangle, then there exists a data structure [32] with space $O(n \log^{d-1} n)$ and query time $O(\log^d n + k \log n)$. For our purposes, it is sufficient to run k independent sampling queries in a (modified) range tree with total query time $O(k \log^d n)$.

Expected entropy and monotonicity. Entropy is not monotone because if $P_1 \subseteq P_2$, it does not always hold that $H(P_1) \leq H(P_2)$. Using the results in [29], we can show that $H(P_1) \geq \frac{|P_1|-1}{|P_1|} H(P_1 \setminus \{p\})$, for a point $p \in P_1 \subset P$. If we multiply with $|P_1|/n$ we have $\frac{|P_1|}{n} H(P_1) \geq \frac{|P_1|-1}{n} H(P_1 \setminus \{p\})$. Hence, we show that, for $P_1 \subseteq P_2 \subseteq P$, $\frac{|P_1|}{n} H(P_1) \leq \frac{|P_2|}{n} H(P_2)$. The quantity $\frac{|P_1|}{|P_1|} H(P_1)$ is called expected entropy. This monotonicity property helps us to design efficient partitioning algorithms with respect to expected entropy, for example, find a partitioning that minimizes the cumulative or maximum expected entropy. $\quad \text{t}$

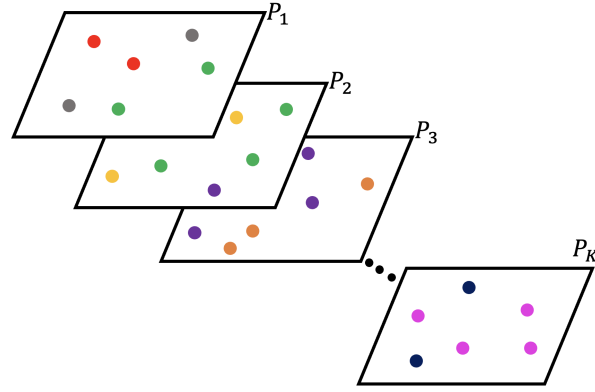
3 Lower Bound

In this section, we give a lower bound for range entropy queries in the real-RAM model. We show a reduction from the set intersection problem that suggests that data structures with near-linear space and polylogarithmic query time are unlikely to exist even for $d = 2$.

The set intersection problem is defined as follows. Given a family of sets S_1, \dots, S_g , with $\sum_{i=1}^g |S_i| = n$, the goal is to construct a data structure such that given a query pair of indices i, j , the goal is to decide if $S_i \cap S_j = \emptyset$. It is widely believed that for any positive value $Q \in \mathbb{R}$, any data structure for the set intersection problem with $O(Q)$ query time needs $\Omega \left(\left(\frac{n}{Q} \right)^2 \right)$ space [20, 35, 36], skipping $\log^{O(1)} n$ factors. Next, we show that any data structure for solving the range entropy query can be used to solve the set intersection problem.



■ **Figure 1** Lower bound construction.



■ **Figure 2** Partition P into K buckets in \mathbb{R}^2 . Two consecutive buckets have at most one color in common.

Let S_1, \dots, S_g be an instance of the set intersection problem as we defined above. We design an instance of the range entropy query constructing a set P of $2n$ points in \mathbb{R}^2 and $|U| = |\bigcup_i S_i|$. Let $n_0 = 0$ and $n_i = n_{i-1} + |S_i|$ for $i = 1, \dots, g$. Let $s_{i,k}$ be the value of the k -th item in S_i (we consider any arbitrary order of the items in each S_i). Let $S = \bigcup_i S_i$, and $q = |S|$. Let $\sigma_1, \dots, \sigma_q$ be an arbitrary ordering of S . We set $U = \{1, \dots, q\}$. Next, we create a geometric instance of P in \mathbb{R}^2 : All points lie on two parallel lines $L = x + n$, and $L' = x - n$. For each $s_{i,k}$ we add in P two points, $p_{i,k} = (-(k + n_{i-1}), -(k + n_{i-1}) + n)$ on L , and $p'_{i,k} = ((k + n_{i-1}), k + n_{i-1} - n)$ on L' . If $s_{i,k} = \sigma_j$ for some $j \leq q$, we set the color/category of both points $p_{i,k}, p'_{i,k}$ to be j . Let P_i be the set of points corresponding to S_i that lie on L , and P'_i the set of points corresponding to S_i that lie on L' . We set $P = \bigcup_i (P_i \cup P'_i)$. We note that for any pair i, j , points $P_i \cup P'_j$ have distinct categories if and only if $S_i \cap S_j = \emptyset$. P uses $O(n)$ space and can be constructed in $O(n)$ time.

Let \mathcal{D} be a data structure for range entropy queries with space $S(n)$ and query time $Q(n)$ constructed on n points. Given an instance of the set intersection problem, we construct P as described above. Then we build \mathcal{D} on P and we construct a range tree \mathcal{T} on P for range counting queries. Given a pair of indexes i, j the question is if $S_i \cap S_j = \emptyset$. We answer this question using \mathcal{D} and \mathcal{T} on P . Geometrically, it is known we can find a rectangle $\rho_{i,j}$ in $O(1)$ time such that $\rho_{i,j} \cap P = P_i \cup P'_j$ (see Figure 1). We run the range entropy query $\mathcal{D}(\rho_{i,j})$ and the range counting query $\mathcal{T}(\rho_{i,j})$. Let $H_{i,j}$ be the entropy of $P_i \cup P'_j$ and $n_{i,j} = |P_i \cup P'_j|$. If $H_{i,j} = \log n_{i,j}$ we return that $S_i \cap S_j = \emptyset$. Otherwise, we return $S_i \cap S_j \neq \emptyset$.

The data structure we construct for answering the set intersection problem has $O(S(2n) + n \log n) = \tilde{O}(S(2n))$ space. The query time is $(Q(2n) + \log n)$ or just $O(Q(n))$ assuming that $Q(n) \geq \log n$.

► **Lemma 5.** *In the preceding reduction, $S_i \cap S_j = \emptyset$ if and only if $H_{i,j} = \log n_{i,j}$.*

Proof. If $S_i \cap S_j = \emptyset$ then from the construction of P we have that all colors in $P_i \cup P'_j$ are distinct, so $n_{i,j} = |u(P_i \cup P'_j)|$. Hence, the entropy $H(P_i \cup P'_j)$ takes the maximum possible value which is $H(P_i \cup P'_j) = \sum_{v \in u(P_i \cup P'_j)} \frac{1}{n_{i,j}} \log n_{i,j} = \log n_{i,j}$.

If $H_{i,j} \neq \log n_{i,j}$ we show that $S_i \cap S_j \neq \emptyset$. The maximum value that $H_{i,j}$ can take is $\log n_{i,j}$ so we have $H_{i,j} < \log n_{i,j}$. The entropy is a measure of uncertainty of a distribution. It is known that the discrete distribution with the maximum entropy is unique and it is the uniform distribution. Any other discrete distribution has entropy less than $\log n_{i,j}$. Hence the result follows. ◀

We conclude with the next theorem.

► **Theorem 6.** *If there is a data structure for range entropy queries with $S(n)$ space and $Q(n)$ query time, then for the set intersection problem there exists a data structure with $O(S(2n))$ space and $O(Q(2n))$ query time, skipping $\log n$ factors.*

4 Exact Data Structures

In this section we describe data structures that return the entropy in a query range, exactly. First, we provide a data structure for $d = 1$ and we extend it to any constant dimension d . Next, we provide a second data structure for any constant dimension d . The first data structure is better for $d = 1$, while the second data structure is better for any constant $d > 1$.

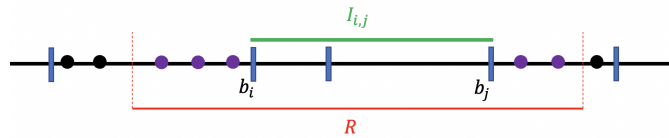
4.1 Efficient data structure for $d = 1$

Let P be a set of n points in \mathbb{R}^1 . Since the range entropy query problem is not decomposable, the main idea is to precompute the entropy in some carefully chosen canonical subsets of P . When we get a query interval R , we find the maximal precomputed canonical subset in R , and then for each color among the colors of points in R not included in the canonical subset, we update the overall entropy using Equations 2, 3, and 4. We also describe how we can precompute the entropy of all canonical subsets efficiently.

Data Structure. Let $t \in [0, 1]$ be a parameter. Let $B_t = \{b_1, \dots, b_k\}$ be $k = n^{1-t}$ points in \mathbb{R}^1 such that $|P \cap [b_j, b_{j+1}]| = n^t$, for any $j < n^{1-t}$. For any pair $b_i, b_j \in B_t$ let $I_{i,j} = [b_i, b_j]$ be the interval with endpoints b_i, b_j and let I be the set of all intervals. For any pair b_i, b_j we store the interval $I_{i,j}$ and we precompute $H_{i,j} = H(P \cap I_{i,j})$, and $n_{i,j} = |P \cap I_{i,j}|$. Next, we construct an interval tree \mathcal{T} on I . Finally, for each color $u \in u(P)$ we construct a search binary tree \mathcal{T}_u over $P(u)$.

We have $|B_t| = O(n^{1-t})$ so $|I| = O(n^{2(1-t)})$. The interval tree along with all the search binary trees have $O(n)$ space in total. Hence we need $O(n^{2(1-t)})$ space for our data structure. In the full version [27], we show how we can construct the data structure in $O(n^{2-t})$ time.

Query procedure. Given a query interval R , we find the maximal interval $I_{i,j} \in I$ such that $I \subseteq R$ using the interval tree. Recall that we have precomputed the entropy $H_{i,j}$. Let $H = H_{i,j}$ be a variable that we will update throughout the algorithm storing the current entropy. Let also $N = n_{i,j}$ be the variable that stores the number of items we currently consider to compute H . Let $P_R = P \cap (R \setminus I_{i,j})$ be the points in $P \cap R$ that are not included in the maximal interval $I_{i,j}$. See also Figure 3.



■ **Figure 3** Instance of the query algorithm given query interval R . Purple points are points in P_R .

We visit each point in P_R and we identify $u(P_R)$. For each $\mathbf{u} \in u(P_R)$, we run a query in $\mathcal{T}_{\mathbf{u}}$ with range $I_{i,j}$ finding the number of points in $P \cap I_{i,j}$ with color \mathbf{u} . Let $n_{\mathbf{u}}$ be this count.

If $n_{\mathbf{u}} = 0$ then there is no point in $P \cap I_{i,j}$ with color \mathbf{u} so we insert $|P_R(\mathbf{u})|$ items of color \mathbf{u} in the current entropy using Equation 3. In that formula, $|P_1| = N$, $H(P_1) = H$ and $|P_2| = |u(P_R)|$. We update $N = N + |u(P_R)|$, and H with the updated entropy $H(P_1 \cup P_2)$.

If $n_{\mathbf{u}} > 0$ then there is at least one point in $P \cap I_{i,j}$ with color \mathbf{u} . Hence, we update the entropy H , by first removing the $n_{\mathbf{u}}$ points of color \mathbf{u} in $P \cap I_{i,j}$ and then re-inserting $n_{\mathbf{u}} + |u(P_R)|$ points of color \mathbf{u} . We use Equation 4 for removing the points with color \mathbf{u} with $|P_1| = N$, $H(P_1) = H$, and $|P_3| = n_{\mathbf{u}}$. We update $N = N - n_{\mathbf{u}}$ and H with the updated entropy $H(P_1 \setminus P_3)$. Then we use Equation 3 for re-inserting the points with color u , with $|P_1| = N$, $H(P_1) = H$, and $|P_2| = n_{\mathbf{u}} + |u(P_R)|$. We update $N = N + n_{\mathbf{u}} + |u(P_R)|$ and H with the updated entropy $H(P_1 \cup P_2)$. After visiting all colors in $u(P_R)$, we return the updated entropy H . The correctness of the algorithm follows from Equations 3, 4. For each color $u \in u(P_R)$ we update the entropy including all points of color u .

For a query interval R we run a query in the interval tree to find $I_{i,j}$ in $O(\log n)$ time. The endpoints of R intersect two intervals $[b_h, b_{h+1}]$ and $[b_v, b_{v+1}]$. Recall that by definition, such interval contains $O(n^t)$ points from P . Hence, $|P_R| = O(n^t)$ and $|u(P_R)| = O(n^t)$. For each $\mathbf{u} \in u(P_R)$, we spend $O(\log n)$ time to search $\mathcal{T}_{\mathbf{u}}$ and find $n_{\mathbf{u}}$. Then we update the entropy in $O(1)$ time. Overall, the query procedure takes $O(n^t \log n)$ time.

► **Theorem 7.** *Let P be a set of n points in \mathbb{R}^1 , where each point is associated with a color, and let $t \in [0, 1]$ be a parameter. A data structure of $O(n^{2(1-t)})$ size can be computed in $O(n^{2-t})$ time, such that given a query interval R , $H(P \cap R)$ can be computed in $O(n^t \log n)$ time.*

In the full version [27], we extend this data structure to any constant $d > 1$.

4.2 Efficient data structure for $d > 1$

While the previous data structure can be extended to higher dimensions, here we propose a more efficient data structure for $d > 1$. In this data structure we split the points with respect to their colors. The data structure has some similarities with the data structure presented in [3, 4] for the max query under uncertainty, however the two problems are different and there are key differences on the way we construct the data structure and the way we compute the result of the query.

Data Structure. We first consider an arbitrary permutation of the colors in U , i.e. u_1, \dots, u_m . The order used to partition the items is induced from the permutation over the colors. Without loss of generality we set $u_j = j$ for each $j \leq m$. We split P into $K = O(n^{1-t})$ buckets P_1, \dots, P_K such that i) each bucket contains $O(n^t)$ points, and ii) for every point $p \in P_i$ and $q \in P_{i+1}$, $u(p) \geq u(q)$. We notice that for any pair of buckets P_i, P_{i+1} it holds $|u(P_i) \cap u(P_{i+1})| \leq 1$, see Figure 2. We slightly abuse the notation and we use P_i to represent both the i -th bucket and the set of points in the i -th bucket.

For each bucket P_i , we take all combinatorially different (hyper)rectangles R_i defined by the points P_i . For each such rectangle r , we precompute and store the entropy $H(P_i \cap r)$ along with the number of points $n(P_i \cap r) = |P_i \cap r|$. In addition, we store $u^+(r)$, the color with the maximum value (with respect to the permutation of the colors) in $r \cap P_i$. Furthermore, we store $u^-(r)$, the color with the minimum value in $r \cap P_i$. Let $n^+(r) = |\{p \in r \cap P_i \mid u(p) = u^+(r)\}|$ and $n^-(r) = |\{p \in r \cap P_i \mid u(p) = u^-(r)\}|$. Finally, for each bucket P_i we construct a modified range tree \mathcal{T}'_i over all R_i , such that given a query rectangle R it returns the maximal rectangle $r \in R_i$ that lies completely inside R . We note that $r \cap P_i = R \cap P_i$. This can be done by representing the d -dimensional hyper-rectangles as $2d$ -dimensional points merging the coordinates of two of their corners.

6:10 Range Entropy Queries and Partitioning

Overall, we need $O(n \log^{2d-1} n)$ space for the modified range trees \mathcal{T}'_i , and $O(n^{1-t} \cdot n^{2dt}) = O(n^{(2d-1)t+1})$ space to store all additional information (entropy, counts, max/min color) in each rectangle. This is because there are $O(n^{1-t})$ buckets, and in each bucket there are $O(n^{2dt})$ combinatorially different rectangles. Overall, our data structure has $O\left(n \log^{2d-1} n + n^{(2d-1)t+1}\right)$ space.

Query Procedure. We are given a query (hyper)rectangle R . We visit the buckets P_1, \dots, P_K in order and compute the entropy for $R \cap (P_1 \cup \dots \cup P_i)$. Let H be the overall entropy we have computed so far. For each bucket P_i we do the following: First we run a query using \mathcal{T}'_i to find $r_i \in R_i$ that lies completely inside R . Then we update the entropy H considering the items in $P_i \cap r_i$. If $u^-(r_{i-1}) = u^+(r_i)$ then we update the entropy H by removing $n^-(r_{i-1})$ points with color $u^-(r_{i-1})$ using Equation 4. Then we insert $n^-(r_{i-1}) + n^+(r_i)$ points of color $u^+(r_i)$ in H using Equation 3. Finally, we remove $n^+(r_i)$ points of color $u^+(r_i)$ from the precomputed $H(P_i \cap r_i)$ using Equation 4 and we merge the updated H with $H(P_i \cap r_i)$ using Equation 2. We note that in the last step we can merge the updated H with the updated $H(P_i \cap r_i)$ because no color from the points used to compute the current H is appeared in the points used to compute the current $H(P_i \cap r_i)$. On the other hand, if $u^-(r_{i-1}) \neq u^+(r_i)$, then we merge the entropies H and $H(P_i \cap r_i)$ using directly Equation 2.

In each bucket P_i we need $O(\log^{2d} n)$ to identify the maximal rectangle r_i inside R . Then we need $O(1)$ time to update the current entropy H . Overall, we need $O(n^{1-t} \log^{2d} n)$ time.

Fast Construction. All range trees can be computed in $O(n \log^{2d} n)$ time. Next, we focus on computing $H(P_i \cap r)$ for all rectangles $r \in R_i$. We compute the other quantities $n(P_i \cap r)$, $u^-(r)$, and $u^+(r)$ with a similar way. A straightforward way is to consider every possible rectangle r and compute independently the entropy in linear time. There are $O(n^{2dt})$ rectangles so the running time is $O(n^{2dt+1})$. We propose a faster construction algorithm.

The main idea is to compute the entropy for rectangles in a specific order. In particular, we compute the entropy of rectangles that contain c points after we compute the entropies for rectangles that contain $c - 1$ points. Then we use Equations 3, 4 to update the entropy of the new rectangle without computing it from scratch. Overall, we construct the data structure in $O(n^{(2d-1)t+1} \log^{d+1} n)$ time. We describe the missing details in the full version of the paper [27].

► **Theorem 8.** *Let P be a set of n points in \mathbb{R}^d , where each point is associated with a color, and let $t \in [0, 1]$ be a parameter. A data structure of $O(n \log^{2d-1} n + n^{(2d-1)t+1})$ size can be computed in $O(n \log^{2d} n + n^{(2d-1)t+1} \log^{d+1} n)$ time, such that given a query hyper-rectangle R , $H(P \cap R)$ can be computed in $O(n^{1-t} \log^{2d} n)$ time.*

5 Approximate Data Structures

In this section we describe data structures that return the entropy in a query range, approximately. First, we provide a data structure that returns an additive approximation of the entropy and next we provide a data structure that returns a multiplicative approximation efficiently. Finally, for $d = 1$, we design a deterministic and more efficient data structure that returns an additive and multiplicative approximation of the entropy.

5.1 Additive approximation

In this Subsection, we construct a data structure on P such that given a query rectangle R and a parameter Δ , it returns a value h such that $H(P \cap R) - \Delta \leq h \leq H(P \cap R) + \Delta$. The intuition comes from the area of finding an additive approximation of the the entropy of an unknown distribution in the dual access model [13].

Let D be a fixed distribution over a set of values $\alpha_1, \dots, \alpha_N$. Each value α_i has a probability $D(\alpha_i)$ which is not known, such that $\sum_{i=1}^N D(\alpha_i) = 1$. The authors in [13] show that if we ask $O\left(\frac{\log^2 \frac{N}{\Delta} \log N}{\Delta^2}\right)$ sample queries in the dual access model, then we can get a Δ additive-approximation of the entropy of D with high probability in $O\left(\frac{\log^2 \frac{N}{\Delta} \log N}{\Delta^2} \mathcal{S}\right)$ time, where \mathcal{S} is the running time to get a sample. In the dual access model, we consider that we have a dual oracle for D which is a pair of oracles ($\text{SAMP}_D, \text{EVAL}_D$). When required, the sampling oracle SAMP_D returns a value α_i with probability $D(\alpha_i)$, independently of all previous calls to any oracle. Furthermore, the evaluation oracle EVAL_D takes as input a query element α_i and returns the probability weight $D(\alpha_i)$.

Next, we describe how the result above can be used in our setting. The goal in our setting is to find the entropy $H(P')$, where $P' = P \cap R$, for a query rectangle R . The colors in $u(P')$ define the distinct values in distribution D . By definition, the number of colors is bounded by $|P'| = O(n)$. The probability weight is defined as $\frac{|P'(u_i)|}{|P'|}$. We note that in [13] they assume that they know N , i.e., the number of values in distribution D . In our case, we cannot compute the number of colors $|u(P')|$ efficiently. Even though we can easily compute an $O(\log^d n)$ approximation of $|u(P')|$, it is sufficient to use the loose upper bound $|u(P')| \leq n$. This is because, without loss of generality, we can assume that there exist $n - |u(P')|$ values/colors with probability (arbitrarily close to) 0. All the results still hold. Next we present our data structure to simulate the dual oracle.

Data structure. For each color $u_i \in U$ we construct a range tree \mathcal{T}_i on $P(u_i)$ for range counting queries. We also construct another range tree \mathcal{T} on P for range counting queries, which is independent of the color. Next, we construct a range tree \mathcal{S} on P for range sampling queries. In particular, by pre-computing the number of points stored in the subtree of each node of the range tree, we can return a sample in a query region efficiently. For more details the reader can check the full version of the paper [27], and [1, 2, 26, 32, 37, 39, 40] where the authors propose a data structure for finding k samples in a query region efficiently¹. We need $O(n \log^d n)$ time to construct all the range trees, while the overall space is $O(n \log^{d-1} n)$.

Query procedure. The query procedure involves the algorithm for estimating the entropy of an unknown distribution in the dual access model [13]. Here, we only need to describe how to execute the oracles SAMP_D and EVAL_D in $P' = P \cap R$ using the data structure.

- **SAMP_D:** Recall that SAMP_D returns α_i with probability $D(\alpha_i)$. In our setting, values $\alpha_1, \dots, \alpha_n$ correspond to colors. So, the goal is to return a color u_i with probability proportional to the number of points with color u_i in P' . Indeed, \mathcal{S} returns a point p uniformly at random in P' . Hence, the probability that a point with color u_i is found is $\frac{|P'(u_i)|}{|P'|}$.

¹ While it is known how to get k independent weighted samples in a query hyper-rectangle in $O(\log^d n + k \log n)$ time [32], the overall asymptotic query time of our problem remains the same if we use a range tree as described in [27] with $O(k \log^d n)$ query time.

■ EVAL_D : Recall that given a value α_i , EVAL_D returns the probability weight $D(\alpha_i)$. Equivalently, in our setting, given a color u_i , the goal is to return $\frac{|P'(u_i)|}{|P'|}$. Using \mathcal{T}_i we run a counting query in the query rectangle R and find $|P'(u_i)|$. Then using \mathcal{T} , we run a counting query in R and we get $|P'|$. We divide the two quantities and return the result. In each iteration, every oracle call SAMP_D and EVAL_D executes a constant number of range tree queries, so the running time is $O(\log^d n)$. The algorithm presented in [13] calls the oracles $O\left(\frac{\log^2 \frac{n}{\Delta} \log n}{\Delta^2}\right)$ times to guarantee the result with probability at least $1 - 1/n$, so the overall query time is $O\left(\frac{\log^{d+1} n \cdot \log^2 \frac{n}{\Delta}}{\Delta^2}\right)$. We note that if $\Delta < \frac{1}{\sqrt{n}}$ then the query time is $\Omega(n \log n)$. However, it is trivial to compute the entropy in $P \cap R$ in $O(n \log n)$ time by traversing all points in $P \cap R$. Hence, the additive approximation is non-trivial when $\Delta \geq \frac{1}{\sqrt{n}}$. In this case, $\log^2 \frac{n}{\Delta^2} = O(\log^2 n)$. We conclude that the query time is bounded by $O\left(\frac{\log^{d+3} n}{\Delta^2}\right)$. We conclude with the next theorem.

► **Theorem 9.** *Let P be a set of n points in \mathbb{R}^d , where each point is associated with a color. A data structure of $O(n \log^{d-1} n)$ size can be computed in $O(n \log^d n)$ time, such that given a query hyper-rectangle R and a real parameter Δ , a value h can be computed in $O\left(\frac{\log^{d+3} n}{\Delta^2}\right)$ time, such that $H(P \cap R) - \Delta \leq h \leq H(P \cap R) + \Delta$, with high probability.*

This data structure can be made dynamic under arbitrary insertions and deletions of points using well known techniques [9, 21, 33, 34]. The update time is $O(\log^d n)$.

5.2 Multiplicative approximation

In this Subsection, we construct a data structure such that given a query rectangle R and a parameter ε , it returns a value h such that $\frac{1}{1+\varepsilon}H(P \cap R) \leq h \leq (1+\varepsilon)H(P \cap R)$. The intuition comes from the area of finding a multiplicative approximation of the the entropy of an unknown distribution in the dual access model [23] and the streaming algorithms for finding a multiplicative approximation of the the entropy [14]. In particular, in this section we extend the streaming algorithm proposed in [14] to work in the query setting.

We use the notation from the previous Subsection where D is an unknown distribution over a set of values $\alpha_1, \dots, \alpha_N$. It is known [23] that if we ask $O\left(\frac{\log N}{\varepsilon^2 \cdot H'}\right)$ queries in the dual access model, where H' is a lower bound of the actual entropy of D , i.e., $H(D) \geq H'$, then we can get an $(1+\varepsilon)$ -multiplicative approximation of the entropy of D with high probability, in $O\left(\frac{\log N}{\varepsilon^2 \cdot H'}\mathcal{S}\right)$ time, where \mathcal{S} is the time to get a sample. We consider that we have a dual oracle for D which is a pair of oracles $(\text{SAMP}_D, \text{EVAL}_D)$, as we had in additive approximation. Similarly to the additive approximation, in our setting we do not know the number of colors in $P' = P \cap R$ or equivalently the number of values N in distribution D . However it is sufficient to use the upper bound $|u(P')| \leq n$ considering $n - |u(P')|$ colors with probability (arbitrarily close to) 0. If we use the same data structure constructed for the additive approximation, we could solve the multiplicative-approximation, as well. While this is partially true, there is a big difference between the two problems. What if the actual entropy is very small so H' is also extremely small? In this case, the factor $\frac{1}{H'}$ will be very large making the query procedure slow.

We overcome this technical difficulty by considering two cases. If H' is large, say $H' \geq 0.9$, then we can compute a multiplicative approximation of the entropy efficiently applying [23]. On the other hand, if H' is small, say $H' < 0.9$, then we use the ideas from [14] to design an efficient data structure. In particular, we check if there exists a value a_M with $D(a_M) > 2/3$.

If it does not exist then H' is large so it is easy to handle. If a_M exists, we write $H(D)$ as a function of $H(D \setminus \{a_M\})$ using Equation 4. In the end, if we get an additive approximation of $H(D \setminus \{a_M\})$ we argue that this is sufficient to get a multiplicative approximation of H' .

Data Structure. For each color c_i we construct a range tree \mathcal{T}_i over $P(u_i)$ as in the previous Subsection. Similarly, we construct a range tree \mathcal{T} over P for counting queries. We also construct the range tree \mathcal{S} for returning uniform samples in a query rectangle. In addition to \mathcal{S} , we also construct a variation of this range tree, denoted by $\bar{\mathcal{S}}$. Given a query rectangle R and a color c_i , $\bar{\mathcal{S}}$ returns a point from $\{p \in R \cap P \mid u(p) \neq c_i\}$ uniformly at random. In other words, $\bar{\mathcal{S}}$ is a data structure over P that is used to return a point in a query rectangle uniformly at random excluding points of color c_i . While $\bar{\mathcal{S}}$ is an extension of \mathcal{S} , the low level details are more tedious. We describe $\bar{\mathcal{S}}$ in the full version of the paper [27].

The complexity of the proposed data structure is dominated by the complexity of $\bar{\mathcal{S}}$. Overall, it can be computed in $O(n \log^d n)$ time and it has $O(n \log^d n)$ space.

Query procedure. First, using \mathcal{T} we get $N = |P \cap R|$. Using \mathcal{S} we get $\frac{\log(2n)}{\log 3}$ independent random samples from $P \cap R$. Let P_S be the set of returned samples. For each $p \in P_S$ with $u(p) = u_i$, we run a counting query in \mathcal{T}_i to get $N_i = |P(u_i) \cap R|$. Finally, we check whether $\frac{N_i}{N} > 2/3$. If we do not find a point $p \in P_S$ (assuming $u(p) = u_i$) with $\frac{N_i}{N} > 2/3$ then we run the algorithm from [23]. In particular, we set $H' = 0.9$ and we run $O\left(\frac{\log n}{\varepsilon^2 \cdot H'}\right)$ oracle queries SAMP_D or EVAL_D , as described in [23]. In the end we return the estimate h . Next, we assume that the algorithm found a point with color u_i satisfying $\frac{N_i}{N} > 2/3$. Using $\bar{\mathcal{S}}$ (instead of \mathcal{S}) we run the query procedure of the previous Subsection and we get an ε -additive approximation of $H((P \setminus P(u_i)) \cap R)$, i.e., the entropy of the points in $P \cap R$ excluding points of color c_i . Let h' be the ε -additive approximation we get. In the end, we return the estimate $h = \frac{N-N_i}{N} \cdot h' + \frac{N_i}{N} \log \frac{N}{N_i} + \frac{N-N_i}{N} \log \frac{N}{N-N_i}$.

Correctness. It is straightforward to see that if there exists a color u_i containing more than $2/3$'s of all points in $P \cap R$ then $u_i \in u(P_S)$ with high probability. For completeness, in [27] we prove that this is the case with probability at least $1 - 1/(2n)$. Hence, with high probability, we make the correct decision.

If there is not such color, then in the full version [27] we show that the entropy in this case should be $H(P \cap R) > 0.9$. Hence, $O\left(\frac{\log n}{\varepsilon^2}\right)$ oracle queries are sufficient to derive an $(1 + \varepsilon)$ -multiplicative approximation of the correct entropy.

The interesting case is when we find a color u_i such that $\frac{N_i}{N} > 2/3$ and $\frac{N_i}{N} < 1$ (if $\frac{N_i}{N} = 1$ then $H(P \cap R) = 0$). Using the results of the previous Subsection along with the new data structure $\bar{\mathcal{S}}$, we get $h' \in [H((P \setminus P(u_i)) \cap R) - \varepsilon, H((P \setminus P(u_i)) \cap R) + \varepsilon]$ with probability at least $1 - 1/(2n)$. We finally show that the estimate h we return is a multiplicative approximation of $H(P \cap R)$. From Equation 4, we have $H(P \cap R) = \frac{N-N_i}{N} H((P \setminus P(u_i)) \cap R) + \frac{N_i}{N} \log \frac{N}{N_i} + \frac{N-N_i}{N} \log \frac{N}{N-N_i}$. Since $h' \in [H((P \setminus P(u_i)) \cap R) - \varepsilon, H((P \setminus P(u_i)) \cap R) + \varepsilon]$, we get $h \in [H(P \cap R) - \varepsilon \frac{N-N_i}{N_i}, H(P \cap R) + \varepsilon \frac{N-N_i}{N_i}]$. If we show that $\frac{N-N_i}{N_i} \leq H(P \cap R)$ then the result follows. By the definition of entropy we observe that $H(P \cap R) \geq \frac{N_i}{N} \log \frac{N}{N_i} + \frac{N-N_i}{N} \log \frac{N}{N-N_i}$. In the full version of the paper [27] we show that $\frac{N-N_i}{N_i} \leq \frac{N_i}{N} \log \frac{N}{N_i} + \frac{N-N_i}{N} \log \frac{N}{N-N_i}$, if $1 > \frac{N_i}{N} > 2/3$. We conclude that $h \in [(1 - \varepsilon)H(P \cap R), (1 + \varepsilon)H(P \cap R)]$.

Analysis. We first run a counting query on \mathcal{T} in $O(\log^d n)$ time. Then the set P_S is constructed in $O(\log^{d+1} n)$ time, running $O(\log n)$ queries in $\bar{\mathcal{S}}$. In the first case of the query procedure (no point p with $\frac{N_i}{N} > 2/3$) we run $O\left(\frac{\log n}{\varepsilon^2}\right)$ oracle queries so in total it runs in

$O(\frac{\log^{d+1}}{\varepsilon^2})$ time. In the second case of the query procedure (point p with $\frac{N_i}{N} > 2/3$) we run the query procedure of the previous Subsection using $\bar{\mathcal{S}}$ instead of \mathcal{S} , so it takes $O(\frac{\log^{d+3}}{\varepsilon^2})$ time. Overall, the query procedure takes $O(\frac{\log^{d+3}}{\varepsilon^2})$ time.

► **Theorem 10.** *Let P be a set of n points in \mathbb{R}^d , where each point is associated with a color. A data structure of $O(n \log^d n)$ size can be computed in $O(n \log^d n)$ time, such that given a query hyper-rectangle R and a parameter $\varepsilon \in (0, 1)$, a value h can be computed in $O(\frac{\log^{d+3} n}{\varepsilon^2})$ time, such that $\frac{1}{1+\varepsilon}H(P \cap R) \leq h \leq (1+\varepsilon)H(P \cap R)$, with high probability.*

This structure can be made dynamic under arbitrary insertions and deletions of points using well known techniques [9, 21, 33, 34]. The update time is $O(\log^d n)$.

5.3 Efficient additive and multiplicative approximation for $d = 1$

Next, for $d = 1$, we propose a deterministic, faster approximate data structure with query time $O(\text{polylog } n)$ that returns an additive and multiplicative approximation of the entropy $H(P \cap R)$, given a query rectangle R .

Instead of using the machinery for entropy estimation on unknown distributions, we get the intuition from data structures that count the number of colors in a query region R . In [24], the authors presented a data structure to count/report colors in a query interval for $d = 1$. In particular, they map the range color counting/reporting problem for $d = 1$ to the standard range counting/reporting problem in \mathbb{R}^2 . Let P be the set of n colored points in \mathbb{R}^1 . Let $\bar{P} = \emptyset$ be the corresponding points in \mathbb{R}^2 they construct. For every color $u_i \in U$, without loss of generality, let $P(u_i) = \{p_1, p_2, \dots, p_k\}$ such that if $j < \ell$ then the x -coordinate of point p_j is smaller than the x -coordinate of point p_ℓ . For each point $p_j \in P(u_i)$, they construct the 2-d point $\bar{p}_j = (p_j, p_{j-1})$ and they add it in \bar{P} . If $p_j = p_1$, then $\bar{p}_1 = (p_1, -\infty)$. Given a query interval $R = [l, r]$ in 1-d, they map it to the query rectangle $\bar{R} = [l, r] \times (-\infty, l)$. It is straightforward to see that a point of color u_i exists in R if and only if \bar{R} contains exactly one transformed point of color u_i . Hence, using a range tree $\bar{\mathcal{T}}$ on \bar{P} they can count (or report) the number of colors in $P \cap R$ efficiently. While this is more than enough to count or report the colors in $P \cap R$, for the entropy we also need to know (in fact precompute) the number of points of each color u_i in P' , along with the actual entropy in each canonical subset. Notice that a canonical subset/node in $\bar{\mathcal{T}}$ might belong to many different query rectangles \bar{R} that correspond to different query intervals R . Even though a point of color u_i appears only once in $\bar{R} \cap \bar{P}$, there can be multiple points with color u_i in $R \cap P$. Hence, there is no way to know in the preprocessing phase the exact number of points of each color presented in a canonical node of $\bar{\mathcal{T}}$. We overcome this technical difficulty by pre-computing for each canonical node v in $\bar{\mathcal{T}}$, monotone pairs with approximate values of (interval, number of points), and (interval, entropy) over a sufficiently large number of intervals. Another issue is that entropy is not monotone, so we split it into two monotone functions and we handle each of them separately until we merge them in the end to get the final estimation.

Before we start describing the data structure we prove some useful properties that we need later. For a set of colored points $P' \subseteq P$, with $N = |P'|$, let $F(P') = N \cdot H(P') = \sum_{u_i \in u(P')} N_i \cdot \log \frac{N}{N_i}$, where N_i is the number of points in P' with color u_i . We prove the next lemma in the full version [27].

► **Lemma 11.** *The function $F(\cdot)$ is monotonically increasing. Furthermore, $F(P') = O(N \log N)$, and the smallest non-zero value that $F(\cdot)$ can take is at least $\log N$.*

Data structure. We apply the same mapping from P to \bar{P} as described above [24] and construct a range tree $\bar{\mathcal{T}}$ on \bar{P} . Then we visit each canonical node v of $\bar{\mathcal{T}}$. If node v contains two points with the same color then we can skip it because this node will not be returned as a canonical node for any query \bar{R} . Let v be a node such that \bar{P}_v does not contain two points with the same color. Let also x_v be the smallest x -coordinate of a point in \bar{P}_v . Finally, let $U_v = u(\bar{P}_v)$, and $P(U_v) = \{p \in P \mid u(p) \in U_v\}$. Notice that $P(U_v)$ is a subset of P and not of \bar{P} . We initialize an empty array S_v of size $O(\frac{\log n}{\varepsilon})$. Each element $S_v[i]$ stores the maximum x coordinate such that $(1 + \varepsilon)^i \geq |P(U_v) \cap [x_v, x]|$. Furthermore, we initialize an empty array H_v of size $O(\frac{\log n}{\varepsilon})$. Each element $H_v[i]$ stores the maximum x coordinate such that $(1 + \varepsilon)^i \geq F(P(U_v) \cap [x_v, x])$. We notice that both functions $F(\cdot)$, and cardinality of points are monotonically increasing. For every node of $\bar{\mathcal{T}}$ we use $O(\frac{\log n}{\varepsilon})$ space, so in total, the space of our data structure is $O(\frac{n}{\varepsilon} \log^2 n)$. In the full version of the paper [27] we show how we can construct the data structure $\bar{\mathcal{T}}$ in $O(\frac{n}{\varepsilon} \log^5 n)$ time.

Query procedure. Given a query interval $R = [a, b]$, we run a query in $\bar{\mathcal{T}}$ using the query range \bar{R} . Let $V = \{v_1, \dots, v_k\}$ be the set of $k = O(\log^2 n)$ returned canonical nodes. For each node $v \in V$ we run a binary search in array S_v and a binary search in H_v with key b . Let ℓ_v^S be the minimum index such that $b \leq S_v[\ell_v^S]$ and ℓ_v^H be the minimum index such that $b \leq H_v[\ell_v^H]$. From their definitions, it holds that $|P(U_v) \cap R| \leq (1 + \varepsilon)^{\ell_v^S} \leq (1 + \varepsilon)|P(U_v) \cap R|$, and $F(P(U_v) \cap R) \leq (1 + \varepsilon)^{\ell_v^H} \leq (1 + \varepsilon)F(P(U_v) \cap R)$. Hence, we can approximate the entropy of $P(U_v) \cap R$, defining $\mathcal{H}_v = \frac{(1 + \varepsilon)^{\ell_v^H}}{(1 + \varepsilon)^{\ell_v^S - 1}}$. The next Lemma shows that \mathcal{H}_v is a good approximation of $H(P(U_v) \cap R)$.

► **Lemma 12.** *It holds that $H(P(U_v) \cap R) \leq \mathcal{H}_v \leq (1 + \varepsilon)^2 H(P(U_v) \cap R)$.*

Proof. We have $\mathcal{H}_v = \frac{(1 + \varepsilon)^{\ell_v^H}}{(1 + \varepsilon)^{\ell_v^S - 1}}$. From their definitions, we have that $|P(U_v) \cap R| \leq (1 + \varepsilon)^{\ell_v^S} \leq (1 + \varepsilon)|P(U_v) \cap R|$, and $F(P(U_v) \cap R) \leq (1 + \varepsilon)^{\ell_v^H} \leq (1 + \varepsilon)F(P(U_v) \cap R)$. It also holds that $(1 + \varepsilon)^{\ell_v^S - 1} \leq |P(U_v) \cap R|$ and $(1 + \varepsilon)^{\ell_v^S - 1} \geq \frac{|P(U_v) \cap R|}{(1 + \varepsilon)}$. Hence $\mathcal{H}_v \leq \frac{(1 + \varepsilon)F(P(U_v) \cap R)}{|P(U_v) \cap R|/(1 + \varepsilon)} \leq (1 + \varepsilon)^2 H(P(U_v) \cap R)$. Furthermore, $\mathcal{H}_v \geq \frac{F(P(U_v) \cap R)}{|P(U_v) \cap R|} = H(P(U_v) \cap R)$. ◀

We find the overall entropy by merging together pairs of canonical nodes. Notice that we can do it easily using Equation 2 because all colors are different between any pair of nodes in V . For example, we apply Equation 2 for two nodes $v, w \in V$ as follows:

$$\frac{(1 + \varepsilon)^{\ell_v^S} \mathcal{H}_v + (1 + \varepsilon)^{\ell_w^S} \mathcal{H}_w + (1 + \varepsilon)^{\ell_v^S} \log \left(\frac{(1 + \varepsilon)^{\ell_v^S} + (1 + \varepsilon)^{\ell_w^S}}{(1 + \varepsilon)^{\ell_v^S - 1}} \right) + (1 + \varepsilon)^{\ell_w^S} \log \left(\frac{(1 + \varepsilon)^{\ell_v^S} + (1 + \varepsilon)^{\ell_w^S}}{(1 + \varepsilon)^{\ell_w^S - 1}} \right)}{(1 + \varepsilon)^{\ell_v^S - 1} + (1 + \varepsilon)^{\ell_w^S - 1}}.$$

In the end we compute the overall entropy \mathcal{H} . The next Lemma shows the correctness of our procedure.

► **Lemma 13.** *If we set $\varepsilon \leftarrow \frac{\varepsilon}{4 \cdot c \cdot \log n}$, it holds that $H(P \cap R) \leq \mathcal{H} \leq (1 + \varepsilon)H(P \cap R) + \varepsilon$, for a constant $c > 0$.*

Proof. We assume that we take the union of two nodes $v, w \in V$ using Equation 2. We can use this equation because nodes v, w do not contain points with similar colors. Let $H_1 = H(P(U_v) \cap R)$, $H_2 = H(P(U_w) \cap R)$, $N_1 = |P(U_v) \cap R|$, and $N_2 = |P(U_w) \cap R|$. We have

$$\mathcal{H}_{v,w} = \frac{(1 + \varepsilon)^{\ell_v^S} \mathcal{H}_v + (1 + \varepsilon)^{\ell_w^S} \mathcal{H}_w + (1 + \varepsilon)^{\ell_v^S} \log \left(\frac{(1 + \varepsilon)^{\ell_v^S} + (1 + \varepsilon)^{\ell_w^S}}{(1 + \varepsilon)^{\ell_v^S - 1}} \right) + (1 + \varepsilon)^{\ell_w^S} \log \left(\frac{(1 + \varepsilon)^{\ell_v^S} + (1 + \varepsilon)^{\ell_w^S}}{(1 + \varepsilon)^{\ell_w^S - 1}} \right)}{(1 + \varepsilon)^{\ell_v^S - 1} + (1 + \varepsilon)^{\ell_w^S - 1}}.$$

6:16 Range Entropy Queries and Partitioning

Using Lemma 12, we get

$$\mathcal{H}_{v,w} \leq \frac{(1+\varepsilon)^4 N_1 H_1 + (1+\varepsilon)^4 N_2 H_2 + (1+\varepsilon)^2 N_1 \log\left((1+\varepsilon)^2 \frac{N_1+N_2}{N_1}\right) + (1+\varepsilon)^2 N_2 \log\left((1+\varepsilon)^2 \frac{N_1+N_2}{N_2}\right)}{N_1+N_2}$$

and we conclude that

$$\mathcal{H}_{v,w} \leq (1+\varepsilon)^4 H((P(U_v) \cup P(U_w)) \cap R) + (1+\varepsilon)^2 \log(1+\varepsilon)^2.$$

Similarly if we have computed $\mathcal{H}_{x,y}$ for two other nodes $x, y \in V$, then

$$\mathcal{H}_{x,y} \leq (1+\varepsilon)^4 H((P(U_x) \cup P(U_y)) \cap R) + (1+\varepsilon)^2 \log(1+\varepsilon)^2.$$

If we compute their union, we get

$$\mathcal{H}_{v,w,x,y} \leq (1+\varepsilon)^6 H((P(U_v) \cup P(U_w) \cup P(U_x) \cup P(U_y)) \cap R) + [(1+\varepsilon)^4 + (1+\varepsilon)^2] \log(1+\varepsilon)^2.$$

In the end of this process we have

$$\mathcal{H} \geq H(P \cap R)$$

because all intermediate estimations of entropy are larger than the actual entropy. For a constant c , it also holds that

$$\mathcal{H} \leq (1+\varepsilon)^{c \log(\log n)} H(P \cap R) + \sum_{j=1}^{c \log(\log n)/2} (1+\varepsilon)^{2j} \log(1+\varepsilon)^2.$$

This quantity can be bounded by

$$\mathcal{H} \leq (1+\varepsilon)^{c \log(\log n)} H(P \cap R) + c \log(\log n) (1+\varepsilon)^{c \log(\log n)} \log(1+\varepsilon).$$

We have the factor $\log(\log n)$ because $|V| = O(\log^2 n)$ so the number of levels of recurrence is $O(\log(\log n))$.

Next, we show that if we set $\varepsilon \leftarrow \frac{\varepsilon}{4 \cdot c \log(\log n)}$, then $\mathcal{H} \leq (1+\varepsilon)H(P \cap R) + \varepsilon$.

We have

$$\left(1 + \frac{\varepsilon/4}{c \log(\log n)}\right)^{c \log(\log n)} \leq e^{\varepsilon/4} \leq 1 + \varepsilon.$$

The first inequality holds because of the well known inequality $(1+x/n)^n \leq e^x$. The second inequality is always true for $\varepsilon \in (0, 1)$. Then we have

$$(1+\varepsilon)c \log(\log n) \log\left(1 + \frac{\varepsilon}{4 \cdot c \log(\log n)}\right) \leq 2c \log(\log n) \log\left(1 + \frac{\varepsilon}{4 \cdot c \log(\log n)}\right).$$

Next, we show that this quantity is at most ε . Let $L = c \log(\log n)$ and let

$$f(x) = x - 2L \log\left(1 + \frac{x}{4L}\right)$$

be a real function for $x \in [0, 1]$. We have

$$f'(x) = 1 - \frac{2L}{L \ln(16) + x \ln(2)}.$$

We observe that $\ln(16) \approx 2.77$ and $x \ln(2) \geq 0$ so $f'(x) \geq 0$ and f is monotonically increasing. So $f(x) \geq f(0) = 0$. Hence, for any $\varepsilon \in [0, 1]$ we have

$$\varepsilon - 2L \log\left(1 + \frac{\varepsilon}{4L}\right) \geq 0.$$

We conclude with

$$\mathcal{H} \leq (1+\varepsilon)H(P \cap R) + \varepsilon. \quad \blacktriangleleft$$

We need $O(\log^2 n)$ time to get V from $\bar{\mathcal{T}}$. Then, we run binary search for each node $v \in V$ so we spend $O(\log^2 n \log \frac{\log n \log \log n}{\varepsilon}) = O(\log^2 n \log \frac{\log n}{\varepsilon})$ time. We merge and update the overall entropy in time $O(|V|)$, so in total the query time is $O(\log^2 n \log \frac{\log n}{\varepsilon})$.

► **Theorem 14.** *Let P be a set of n points in \mathbb{R}^1 , where each point is associated with a color, and let $\varepsilon \in (0, 1)$ be a parameter. A data structure of $O(\frac{n}{\varepsilon} \log^2 n)$ size can be computed in $O(\frac{n}{\varepsilon} \log^5 n)$ time, such that given a query hyper-rectangle R , a value h can be computed in $O(\log^2 n \log \frac{\log n}{\varepsilon})$ time, such that $H(P \cap R) \leq h \leq (1 + \varepsilon)H(P \cap R) + \varepsilon$.*

6 Partitioning

The new data structures can be used to accelerate some partitioning algorithms with respect to the (expected) entropy. Let DS be one of our new data structures over n items that can be constructed in $O(P(n))$ time, has $O(S(n))$ space, and given a query range R , returns a value h in $O(Q(n))$ time such that $\frac{1}{\alpha}H - \beta \leq h \leq \alpha \cdot H + \beta$, where H is the entropy of the items in R , and $\alpha \geq 1$, $\beta \geq 0$ two error thresholds. On the other hand, the straightforward way to compute the (expected) entropy without using any data structure has preprocessing time $O(1)$, query time $O(n)$ and it returns the exact entropy in a query range.

In most cases we consider the expected entropy to partition the dataset as this is mostly the case in entropy-based partitioning and clustering algorithms. Except of being a useful quantity bounding both the uncertainty and the size of a bucket, it is also monotone. All our data structures can work for both the entropy and expected entropy quantity almost verbatim. We define two optimization problems. Let **MaxPart** be the problem of constructing a partitioning with k buckets that maximizes/minimizes the maximum (expected) entropy in a bucket. Let **SumPart** be the problem of constructing a partitioning with k buckets that maximizes/minimizes the sum of (expected) entropies over the buckets. For simplicity, in order to compare the running times, we skip the $\log(n)$ factors from the running times.

Partitioning for $d = 1$. We can easily solve **MaxPart** using dynamic programming: $DP[i, j] = \min_{\ell < i} \max\{DP[i - \ell, j - 1], \text{Error}[i - \ell + 1, i]\}$, where $DP[i, j]$ is the minimum max entropy of the first i items using j buckets, and $\text{Error}[i, j]$ is the expected entropy among the items i and j . Since Error is monotone, we can find the optimum $DP[i, j]$ running a binary search on ℓ , i.e., we do not need to visit all indexes $\ell < i$ one by one to find the optimum. Without using any data structure the running time to find $DP[n, k]$ is $O(kn^2)$. Using DS, the running time for partitioning is $O(P(n) + knQ(n))$. If we use the data structure from Section 4.1 for $t = 0.5$, then the running time is $O(kn\sqrt{n}) = o(kn^2)$.

Next we consider approximation algorithms for the **MaxPart** and **SumPart** problems.

It is easy to observe that the maximum value and the minimum non-zero value of the optimum solution of **MaxPart** are bounded polynomially on n . Let $[l_M, r_M]$ be the range of the optimum values. We discretize the range $[l_M, r_M]$ by a multiplicative factor $(1 + \varepsilon)$. We run a binary search on the discrete values. For each value $e \in [l_M, r_M]$ we consider, we construct a new bucket by running another binary search on the input items, trying to expand the bucket until its expected entropy is at most e . We repeat the same for all buckets and we decide if we should increase or decrease the error e in the next iteration. In the end the solution we find is within an $(1 + \varepsilon)$ factor far from the max expected entropy in the optimum partitioning. Without using any data structure, we need $O(n \log \frac{1}{\varepsilon})$ time to construct the partitioning. If we use DS we need time $O(P(n) + kQ(n) \log \frac{1}{\varepsilon})$. If we use the data structure

in Subsection 5.2 we have partition time $O\left(n + \frac{k}{\varepsilon^2} \log \frac{1}{\varepsilon}\right) = o\left(n \log \frac{1}{\varepsilon}\right)$. If we allow a Δ additive approximation in addition to the $(1 + \varepsilon)$ multiplicative approximation, we can use the data structure in Subsection 5.1 having partition time $O\left(n + \frac{k}{\Delta^2} \log \frac{1}{\varepsilon}\right) = o\left(n \log \frac{1}{\varepsilon}\right)$.

Next, we focus on the SumPart problem. It is known from [22] (Theorems 5, 6) that if the error function is monotone (such as the expected entropy) then we can get a partitioning with $(1 + \varepsilon)$ -multiplicative approximation in $O\left(P(n) + \frac{k^3}{\varepsilon^2} Q(n)\right)$ time. Hence, the straightforward solution without using a data structure returns an $(1 + \varepsilon)$ -approximation of the optimum partitioning in $O\left(\frac{k^3}{\varepsilon^2} n\right)$ time. If we use the data structure from Subsection 5.2 we have running time $O\left(n + \frac{k^3}{\varepsilon^4}\right)$, which is $o\left(\frac{k^3}{\varepsilon^2} n\right)$ and multiplicative error $(1 + \varepsilon)^2$. If we set $\varepsilon \leftarrow \varepsilon/3$ then in the same asymptotic running time we have error $(1 + \varepsilon)$. If we also allow $\Delta \cdot n$ additive approximation, we can use the additive approximation DS from Subsection 5.1. The running time will be $O\left(n + \frac{k^3}{\varepsilon^2 \Delta^2}\right) = o\left(\frac{k^3}{\varepsilon^2} n\right)$.

Partitioning for $d > 1$. Partitioning and constructing histograms in high dimensions is usually a very challenging task, since most of the known algorithms with theoretical guarantees are very expensive [18]. However, there is a practical method with some conditional error guarantees, that works very well in any constant dimension d and it has been used in a few papers [5,30,31]. The idea is to construct a tree having a rectangle containing all points in the root. In each iteration of the algorithm, we choose to split (on the median in each coordinate or find the best split) the (leaf) node with the minimum/maximum (expected) entropy. As stated in previous papers, let make the assumption that an optimum algorithm for either MaxPart or SumPart is an algorithm that always chooses to split the leaf node with the smallest/largest expected entropy. Using the straightforward solution without data structures, we can construct an “optimum” partitioning in $O(kn)$ time by visiting all points in every new generated rectangle. Using DS, the running time of the algorithm is $O(P(n) + kQ(n))$. In order to get an optimum solution we use DS from Subsection 4.2. The overall running time is $O(n^{(2d-1)t+1} + kn^{1-t})$. This is minimized for $n^{(2d-1)t+1} = kn^{1-t} \Leftrightarrow t = t^* = \frac{\log k}{2d \log n}$, so the overall running time is $O(kn^{1-t^*}) = o(kn)$. If we allow $(1 + \varepsilon)$ -multiplicative approximation we can use the DS from Subsection 5.2. The running time will be $O\left(n + \frac{k}{\varepsilon^2}\right) = o(kn)$. If we allow a Δ -additive approximation, then we can use the DS from Subsection 5.1 with running time $O\left(n + \frac{k}{\Delta^2}\right) = o(kn)$.

7 Conclusion

In this work, we presented efficient data structures for computing (exactly and approximately) the entropy of the points in a rectangular query in sub-linear time. Using our new data structures we can accelerate partitioning algorithms for columnar compression (Example 1) and histogram construction (Example 2). Furthermore, we can accelerate the exploration of high uncertainty regions for data cleaning (Example 3).

There are multiple interesting open problems derived from this work. i) Our approximate data structures are dynamic but our exact data structures are static. Is it possible to have dynamic data structure for returning the exact entropy? ii) We showed a lower bound for designing exact data structures when $P \in \mathbb{R}^d$ for $d \geq 2$. Does the lower bound extend for $d = 1$? iii) There is still a gap between the proposed lower bound and upper bound. An interesting problem is to close that gap. iv) Can we extend the faster deterministic approximation data structure from Subsection 5.3 in higher dimensions?

References

- 1 Peyman Afshani and Jeff M Phillips. Independent range sampling, revisited again. In *35th International Symposium on Computational Geometry (SoCG 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.SoCG.2019.4.
- 2 Peyman Afshani and Zhewei Wei. Independent range sampling, revisited. In *25th Annual European Symposium on Algorithms (ESA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.3.
- 3 Pankaj K Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. Range-max queries on uncertain data. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 465–476, 2016. doi:10.1145/2902251.2902281.
- 4 Pankaj K Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. Range-max queries on uncertain data. *Journal of Computer and System Sciences*, 94:118–134, 2018. doi:10.1016/j.jcss.2017.09.006.
- 5 Linas Baltrunas, Arturas Mazeika, and Michael Bohlen. Multi-dimensional histograms with tight bounds for the error. In *2006 10th International Database Engineering and Applications Symposium (IDEAS'06)*, pages 105–112. IEEE, 2006. doi:10.1109/IDEAS.2006.31.
- 6 Daniel Barbará, Yi Li, and Julia Couto. Coolcat: an entropy-based algorithm for categorical clustering. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 582–589, 2002. doi:10.1145/584792.584888.
- 7 Tuğkan Batu, Sanjoy Dasgupta, Ravi Kumar, and Ronitt Rubinfeld. The complexity of approximating entropy. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 678–687, 2002. doi:10.1145/509907.510005.
- 8 Irad Ben-Gal, Shahar Weinstock, Gonen Singer, and Nicholas Bambos. Clustering users by their mobility behavioral patterns. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(4):1–28, 2019. doi:10.1145/3322126.
- 9 Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980. doi:10.1016/0196-6774(80)90015-2.
- 10 Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 1997.
- 11 Lakshminath Bhuvanagiri and Sumit Ganguly. Estimating entropy over data streams. In *Algorithms-ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings 14*, pages 148–159. Springer, 2006. doi:10.1007/11841036_16.
- 12 Cafer Caferov, Barış Kaya, Ryan O'Donnell, and AC Say. Optimal bounds for estimating entropy with pmf queries. In *International Symposium on Mathematical Foundations of Computer Science*, pages 187–198. Springer, 2015. doi:10.1007/978-3-662-48054-0_16.
- 13 Clément Canonne and Ronitt Rubinfeld. Testing probability distributions underlying aggregated data. In *International Colloquium on Automata, Languages, and Programming*, pages 283–295. Springer, 2014. doi:10.1007/978-3-662-43948-7_24.
- 14 Amit Chakrabarti, Graham Cormode, and Andrew McGregor. A near-optimal algorithm for computing the entropy of a stream. In *SODA*, volume 7, pages 328–335. Citeseer, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283418>.
- 15 Amit Chakrabarti, Khanh Do Ba, and S Muthukrishnan. Estimating entropy and entropy norm on data streams. *Internet Mathematics*, 3(1):63–78, 2006. doi:10.1080/15427951.2006.10129117.
- 16 Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1247–1261, 2015. doi:10.1145/2723372.2749431.
- 17 Peter Clifford and Ioana Cosma. A simple sketching algorithm for entropy estimation over streaming data. In *Artificial Intelligence and Statistics*, pages 196–206. PMLR, 2013. URL: <http://proceedings.mlr.press/v31/clifford13a.html>.

- 18 Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011. doi:10.1561/19000000004.
- 19 Juan David Cruz, Cécile Bothorel, and François Poulet. Entropy based community detection in augmented social networks. In *2011 International Conference on computational aspects of social networks (CASoN)*, pages 163–168. IEEE, 2011. doi:10.1109/CASON.2011.6085937.
- 20 Pooya Davoodi, Michiel Smid, and Freek van Walderveen. Two-dimensional range diameter queries. In *Latin American Symposium on Theoretical Informatics*, pages 219–230. Springer, 2012. doi:10.1007/978-3-642-29344-3_19.
- 21 J. Erickson. Static-to-dynamic transformations. <http://jeffe.cs.illinois.edu/teaching/datastructures/notes/01-statictodynamic.pdf>.
- 22 Sudipto Guha, Nick Koudas, and Kyuseok Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Transactions on Database Systems (TODS)*, 31(1):396–438, 2006. doi:10.1145/1132863.1132873.
- 23 Sudipto Guha, Andrew McGregor, and Suresh Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 733–742, 2006. URL: <http://dl.acm.org/citation.cfm?id=1109557.1109637>.
- 24 Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995. doi:10.1006/jagm.1995.1038.
- 25 Nicholas JA Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy via approximation theory. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 489–498. IEEE, 2008. doi:10.1109/FOCS.2008.76.
- 26 Xiaocheng Hu, Miao Qiao, and Yufei Tao. Independent range sampling. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 246–255, 2014. doi:10.1145/2594538.2594545.
- 27 Sanjay Krishnan and Stavros Sintos. Range entropy queries and partitioning. *CoRR*, abs/2312.15959, 2023. doi:10.48550/arXiv.2312.15959.
- 28 Ping Li and Cun-Hui Zhang. A new algorithm for compressed counting with applications in shannon entropy estimation in dynamic data. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 477–496. JMLR Workshop and Conference Proceedings, 2011. URL: <http://proceedings.mlr.press/v19/li11a/li11a.pdf>.
- 29 Tao Li, Sheng Ma, and Mitsunori Ogihara. Entropy-based criterion in categorical clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 68, 2004. doi:10.1145/1015330.1015404.
- 30 Xi Liang, Stavros Sintos, and Sanjay Krishnan. JanusAQP: Efficient partition tree maintenance for dynamic approximate query processing. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 572–584. IEEE, 2023. doi:10.1109/ICDE55515.2023.00050.
- 31 Xi Liang, Stavros Sintos, Zechao Shang, and Sanjay Krishnan. Combining aggregation and sampling (nearly) optimally for approximate query processing. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1129–1141, 2021. doi:10.1145/3448016.3457277.
- 32 Andres Lopez Martinez. Parallel minimum cuts: An improved crew pram algorithm. *Master's thesis. KTH, School of Electrical Engineering and Computer Science (EECS)*, 2020.
- 33 Mark H Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1983. doi:10.1007/BFb0014927.
- 34 Mark H Overmars and Jan van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Information Processing Letters*, 12(4):168–173, 1981. doi:10.1016/0020-0190(81)90093-4.

- 35 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 815–823. IEEE, 2010. doi:10.1109/FOCS.2010.83.
- 36 Saladi Rahul and Ravi Janardan. Algorithms for range-skyline queries. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 526–529, 2012. doi:10.1145/2424321.2424406.
- 37 Yufei Tao. Algorithmic techniques for independent query sampling. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 129–138, 2022. doi:10.1145/3517804.3526068.
- 38 Hien To, Kuorong Chiang, and Cyrus Shahabi. Entropy-based histograms for selectivity estimation. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 1939–1948, 2013. doi:10.1145/2505515.2505756.
- 39 Lu Wang, Robert Christensen, Feifei Li, and Ke Yi. Spatial online sampling and aggregation. *Proceedings of the VLDB Endowment*, 9(3):84–95, 2015. doi:10.14778/2850583.2850584.
- 40 Dong Xie, Jeff M Phillips, Michael Matheny, and Feifei Li. Spatial independent range sampling. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2023–2035, 2021. doi:10.1145/3448016.3452806.