Compressibility Measures and Succinct Data Structures for Piecewise Linear Approximations

Paolo Ferragina
Department L'EMbeDS, Sant'Anna School of Advanced Studies, Pisa, Italy
Department of Computer Science, University of Pisa, Italy

Filippo Lari[1]
Department of Computer Science, University of Pisa, Italy

Abstract

We study the problem of deriving compressibility measures for *Piecewise Linear Approximations* (PLAs), i.e., error-bounded approximations of a set of two-dimensional *increasing* data points using a sequence of segments. Such approximations are widely used tools in implementing many *learned data structures*, which mix learning models with traditional algorithmic design blocks to exploit regularities in the underlying data distribution, providing novel and effective space-time trade-offs.

We introduce the first lower bounds to the cost of storing PLAs in two settings, namely *compression* and *indexing*. We then compare these compressibility measures to known data structures, and show that they are asymptotically optimal up to a constant factor from the space lower bounds. Finally, we design the first data structures for the aforementioned settings that achieve the space lower bounds plus small additive terms, which turn out to be *succinct* in most practical cases. Our data structures support the efficient retrieval and evaluation of a segment in the (compressed) PLA for a given x-value, which is a core operation in any learned data structure relying on PLAs.

As a result, our paper offers the first theoretical analysis of the maximum compressibility achievable by PLA-based learned data structures, and provides novel storage schemes for PLAs offering strong theoretical guarantees while also suggesting simple and efficient practical implementations.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Data compression; Information systems → Information retrieval

Keywords and phrases Piecewise Linear Approximations, Succinct Data Structures, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.31

Related Version *Full Version*: `https://arxiv.org/abs/2509.07827` [8]

Funding The work of P.F. was partially supported by the Alfred P. Sloan Foundation with the grant #G-2025-25193 (`sloan.org`). The work of both authors has been supported by the Next-GenerationEU – National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) – Project: "SoBigData.it - Strengthening the Italian RI for Social Mining and Big Data Analytics" – Prot. IR0000013 – Avviso n. 3264 del 28/12/2021; and by the European Union – Horizon 2020 Program under the scheme "INFRAIA-01-2018-2019 – Integrating Activities for Advanced Communities", Grant Agreement n. 871042, "SoBigData++: European Integrated Infrastructure for Social Mining and Big Data Analytics" (`http://www.sobigdata.eu`).

1 Introduction

The rapid growth of data has led to the development of novel techniques for designing data structures that leverage regularities in the underlying data distribution to deliver faster or more succinct solutions to proper query operations. This novel line of research, known as *learned data structures* [15], integrates machine learning models with traditional algorithmic techniques to achieve improved space occupancy and query time performance.

[1] Corresponding author

A widely used tool in the field of learned data structures is the so-called *Piecewise Linear Approximation* (PLA), an error-bounded approximation of a two-dimensional set of points using a sequence of segments [20]. The common use of PLAs is due to their ability to capture linear trends in the input data, which frequently occur in many practical applications, while using little space and providing fast inference time. In this work, we focus on PLAs arising in two settings: (1) the *compression* setting, where the input sequence of points is increasing in both coordinates but the x-values are also consecutive; and (2) the *indexing* setting, where instead the y-values of the input sequence are consecutive. These two settings are general enough to cover a broad range of applications, including *rank* and *select* dictionaries [7, 2], time series compression [14], string indexing [24], one-dimensional indexing [15, 13], multi-dimensional indexing [17, 4], range minimum queries [9], monotone minimal perfect hashing [10], and range filtering [23], to mention a few.

Prior work concerning the study of theoretical properties of PLAs has primarily focused on providing *upper bounds* on the optimal number of segments ℓ as a function of the sequence length n and the maximum allowed error ε, that is the maximum vertical distance between the y-coordinate of any input point and its *covering* segment. Such a relationship is fundamental from a theoretical perspective since it can explain the surprising performance of learned data structures and also allow a theoretical comparison with their classical counterparts. In this context, it is worth mentioning the pioneering study of Ferragina et al. [11] showing that in the indexing case under the assumption that the gaps between consecutive keys are independently and identically distributed random variables, with constant mean and variance, the optimal number of segments ℓ is $\Theta(n/\varepsilon^2)$ with high probability. Later works [25, 3] provided better upper bounds, but they only considered PLAs formed by *horizontal segments*.

As far as *lower bounds* are concerned, we mention the work of Zeighami and Shahabi [26] which was the first to limit from below the model size (e.g., the number of segments ℓ in a PLA or the number of neurons in a neural network) of any learned data structure achieving a given desired accuracy ε for a wide range of problems arising in database systems such as one- and multi-dimensional indexing, cardinality estimation, and range-sum estimation.

In our work, we assume that the segments have *arbitrary slopes* and the number of segments ℓ forming an optimal PLA is known (e.g., computed by [20]), and thus provide the first information-theoretic lower bound to the storage complexity of that PLA in both the compression and the indexing settings (see Theorems 6 and 8). We then compare two representative data structures against these lower bounds, namely the LA-vector by Boffa et al. [2] (for the compression setting) and the PGM-index by Ferragina & Vinciguerra [13] (for the indexing setting), showing that both are optimal up to a constant factor. Finally, we conclude our work by designing two novel data structures for the aforementioned compression and indexing settings, that provide the same asymptotic query time as the LA-vector and the PGM-index, but with small additive terms to the introduced lower bounds.

2 Background

We assume the transdichotomous word RAM model where the word size w matches the logarithm of the problem size and arithmetic, logic, and bitwise operations, as well as access to w-bit memory cells, take $O(1)$ time. To simplify, we denote with $\log x$ the logarithm in base 2 of x, and we assume that logarithms hide their ceiling and thus return integers.

Given a static sorted sequence $S = 1 \leq x_1 < x_2 < \cdots < x_n \leq u$ of n elements drawn from a finite integer universe U of size u, we consider the following fundamental operations:

- $rank(S, x)$, which for $x \in U$ returns the number of S's elements that are smaller or equal to x, i.e. $|\{x_i \in S \,|\, x_i \leq x\}|$.
- $select(S, i)$, which for $i \in \{1, 2, \ldots, n\}$, returns the element in S of position i, i.e. $rank(S, x) = i$. We assume $select(S, 0) = 0$.

If S is represented with its characteristic bit vector B_S, namely, a bit vector of length u such that $B_S[i] = 1$ if $i \in S$, and $B_S[i] = 0$ otherwise, then supporting such operations is equivalent to support the $rank_1(B_S, x)$, which yields the number of 1s before position x, and $select_1(B_S, i)$, which gives the position of the i-th 1 in B_S. These operations have been studied extensively with many practical and theoretical results, and are at the core of any succinct or compressed data structure (see [18] for a comprehensive treatment). For our purposes, we briefly mention that they can be supported in $O(1)$ time while using only $o(u)$ additional bits on top of the underlying bit vector.

The information-theoretic lower bound to store S is $\mathcal{B}(u, n) = \log \binom{u}{n}$. Such value is related to the zero-order empirical entropy of its characteristic bit vector B_S, defined as $u\mathcal{H}_0(B_S) = n \log \frac{u}{n} + (u - n) \log \frac{u}{u-n}$. In fact, with simple arithmetic manipulations one obtains $\mathcal{B}(u, n) = u\mathcal{H}_0(B_S) - O(\log u)$. The best upper bound to represent S with the minimum redundancy with respect to $\mathcal{B}(u, n)$, and while still being able to perform *rank* and *select* in optimal constant time, is given by the following data structure due to Pătraşcu [21]:

▶ **Theorem 1** (Succincter). *For any $c > 0$, a monotone sequence S of n positive integers drawn from a universe of size u can be stored using $\mathcal{B}(u, n) + O(u/\log^c(u/c)) + O(u^{3/4} \, poly \log u)$ bits and support rank and select in $O(c)$ time.*

Theorem 1 implies constant time *rank* and *select* operations in $\mathcal{B}(u, n) + O(u/\log^c u)$ space, which essentially matches the lower bound of Pătraşcu & Viola [22]. To further reduce the space redundancy on top of $\mathcal{B}(u, n)$, one must allow for slower *rank* or *select* operations. A prominent example is given by the Elias-Fano encoding scheme [6, 5], which attains the following space-time trade-off for the aforementioned operations [18]:

▶ **Theorem 2** (Elias-Fano encoding). *A monotone sequence S of n positive integers drawn from a universe of size u can be stored using $2n + n \log \frac{u}{n} + o(n)$ bits while supporting select in $O(1)$ time and rank in $O(\min\{\log \frac{u}{n}, \log n\})$ time.*

Indeed, the space usage of Theorem 2 can be expressed in terms of $\mathcal{B}(u, n)$ as follows:

$$2n + n \log \frac{u}{n} + o(n) = u\mathcal{H}_0(B_S) + O(n) = \mathcal{B}(u, n) + O(n + \log u) \tag{1}$$

Thus obtaining a lower redundancy term with respect to Theorem 1 whenever S is very sparse, which, however, requires giving up the constant time *rank* operation. The literature offers other trade-offs (e.g., [18, 19]) that could be adopted in our solution, leading to corresponding time-space bounds. However, for the sake of presentation, in the remainder of the paper we focus on the results provided in Theorems 1 and 2.

A new line of research has recently started to investigate novel methods for designing algorithms and data structures that leverage regularities in the underlying data to deliver faster or more succinct solutions to a wide variety of problems. Most of these methods rely on a technique known as *Piecewise Linear Approximation* (PLA), which involves approximating a sequence of two-dimensional points using a set of segments.

▶ **Definition 3.** *Given a set of n two-dimensional data points $P \subseteq \mathbb{R}^2$, a Piecewise Linear Approximation of P with maximum error $\varepsilon \geq 1$ is the smallest set of non-overlapping segments $\{s_1, s_2, \ldots, s_\ell\}$ that entirely covers P, ensuring that every point in P lies within a maximum vertical distance ε from its corresponding segment.*

A PLA can always be computed optimally, hence producing the minimum number of segments ℓ in time proportional to the sequence length using an algorithm by O'Rourke [20].

Here, we focus on PLAs computed from monotone sequences of two-dimensional data points, in which a point (x_i, y_i) is smaller than (x_{i+1}, y_{i+1}) because it has a strictly smaller abscissa $x_i < x_{i+1}$ and a smaller ordinate $y_i \leq y_{i+1}$. In particular, we consider two specific scenarios: (1) the compression setting, where PLAs are computed on sequences having consecutive x-values (i.e., $x_{i+1} = x_i + 1$, with $x_1 = 1$), and (2) the indexing setting, in which PLAs are computed on sequences having consecutive y-values (i.e. $y_{i+1} = y_i + 1$, with $y_1 = 1$). As already mentioned, despite the names, such scenarios are quite general and encompass a wide range of applications. To get a practical sense of the usage of PLAs in the design of learned data structures, we briefly outline two prominent examples.

In the design of learned and compressed *rank* and *select* dictionaries [2], the input sequence $S = 1 \leq x_1 < x_2 < \cdots < x_n \leq u$, is first mapped to a Cartesian plane representation where the x-axis corresponds to the sequence of positions $1, 2, \ldots, n$, while the y-axis represents the sequence values, namely x_is. This transformation converts S into a set of monotonically increasing two-dimensional points: $\{(i, x_i) \,|\, x_i \in S\}$. The key innovation of this approach is that the Cartesian plane representation reveals inherent regularities in the underlying sequence S. A PLA is then computed from such a sequence of points, and compression is achieved by storing for each point (i, x_i) the absolute difference between x_i and the approximation of the segment for position i using just $\log(2\varepsilon + 1)$ bits. Performing a $select(S, i)$ operation then reduces to first finding the segment covering the abscissa value i, computing the approximate sequence value from such segment in i, and finally adding the correction term stored in position i to retrieve the exact value. A $rank(S, x)$ operation can be naively solved by binary searching on the *select* values, even if faster alternatives do exist [2].

In the design of learned one-dimensional indexes [13], given the input sequence $S = 1 \leq x_1 < x_2 < \cdots < x_n \leq u$, one performs the same mapping above onto a Cartesian plane representation. In this case, the x-axis contains the sequence values, while the y-axis corresponds to their respective positions $1, 2, \ldots, n$. Again, this transformation converts S into a set of monotonically increasing two-dimensional data points: $\{(x_i, i) \,|\, x_i \in S\}$. The data structure then consists of a PLA computed from such a set of points and a search for a given key x proceeds in three steps. First, the segment covering x is identified through a binary search on the first keys covered by each segment. Second, the approximate position i of x is obtained from such a segment. Third, since the underlying sequence is monotonically increasing and the approximate position i of x is precise up to an error ε, it is *corrected* via a (binary) search in the interval $[i - \varepsilon, i + \varepsilon]$. Such queries can be sped up by applying the same construction recursively to the sequence of first keys covered by each segment until a single one remains (see e.g. [13, 15]). This approach implements the index as a hierarchical structure, forming a tree of linear models. Consequently, searches involve a top-down traversal of the tree, trading faster queries for an increased space usage.

3 Related Work

In the following, we review existing methods for the lossless compression of PLAs, focusing on the $predict(x)$ operation, which, given a value lying on the x-axis, identifies the segment covering x and computes the corresponding y-value *predicted* by that segment.

Ferragina & Vinciguerra introduced the PGM-index [13], the first solution in the indexing case based on the optimal PLA computed by the O'Rourke algorithm [20], i.e., the one consisting of the minimum number of segments ℓ providing an ε-approximation of the input points. In particular, they designed two compression methods for the segments. The first one hinges on the observation that, in the indexing setting, the intercepts are monotonically

increasing. Denoting with β_i the intercept of the i-th segment and with y_i its first covered y-value, this easily follows since $\beta_i \in [y_i - \varepsilon, y_i + \varepsilon]$ and $\beta_{i+1} \in [y_{i+1} - \varepsilon, y_{i+1} + \varepsilon]$, but $y_i + \varepsilon \leq y_{i+1} - \varepsilon$ as in this particular case a segment covers at least 2ε keys [13, Lemma 2]. Exploiting this, they convert the monotone sequence of floating-point intercepts into integers and store them with the data structure of Okanohara & Sadakane [19]. But, they stored uncompressed the first covered keys and slopes of each segment. For the sake of comparison, we provide the space usage for the PLA composing just the first layer of the PGM index. Since the universe of the underlying sequence is u, and each slope can be expressed as a rational number with a numerator and denominator respectively of $\log n$ and $\log u$ bits, they obtained the following result which we specialize via two approaches, one based on binary search (as proposed in [13]), and one using the data structure of Pătraşcu (see Theorem 1):

▶ **Lemma 4** (PGM-index [13]). *The PLA composing the first layer of the PGM-index can be stored using:*

- $\ell\,(1.92 + \log \frac{n^2}{\ell} + 2\,\log u + o(1))$ *bits, while supporting the predict operation in* $O(\log \ell)$ *time via binary search.*
- $\ell\,(1.92 + \log \frac{n^2}{\ell} + \log u + o(1)) + \log \binom{u}{\ell} + O(u/\log^c u)$ *bits for any constant* $c > 0$*, while supporting the predict operation in* $O(c)$ *time.*

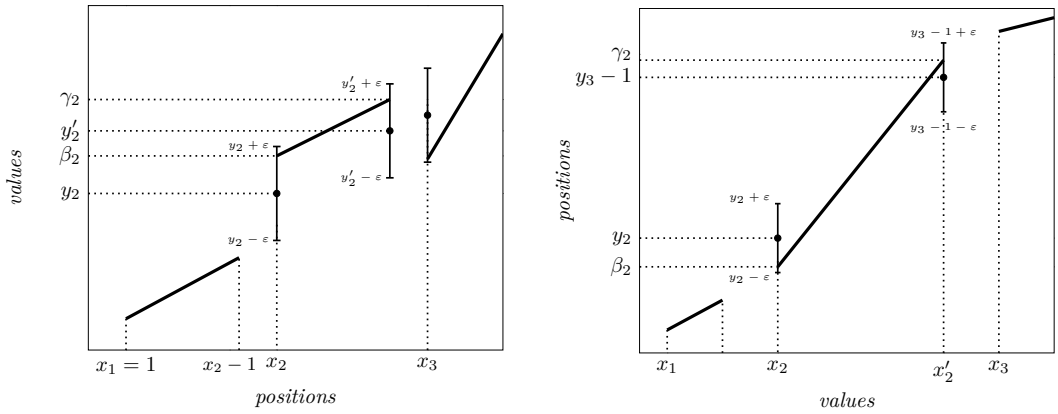
The second compression method targets the segment's slopes, and is based on the observation that the O'Rourke algorithm does not compute a single segment but a whole family of segments whose slopes identify an interval of reals. The compressor then works greedily (and optimally) by trying to assign the same slope to most segments of the PLA, thus reducing the number of slopes to be fully represented. This is very effective on some real-world datasets, however, the authors did not provide an analysis of its space occupancy.

Moving to the compression setting, Boffa et al. introduced the LA-vector [2], the first learned and compressed *rank* and *select* dictionary based on the optimal PLA. Their compression scheme for segments hinges on the assumption that the sequence of intercepts is monotonically increasing, which is not necessarily the case in general. This is because, in the compression setting, segments may cover fewer than 2ε points, as the y-value can increase by more than one between consecutive x-values. Ferragina & Lari [9] recently showed how to waive that assumption with a negligible impact on space occupancy, obtaining the following bound (we drop the cost of cn bits for storing the correction vector C in Theorem 3.2 of [2], and we add the solution based on the data structure of Pătraşcu):

▶ **Lemma 5** (LA vector, Theorem 2.3 in [2] and Theorem 5 in [9]). *The PLA of the LA-vector can be stored using*

- $\ell\,(2\log \frac{u}{\ell} + \log \frac{n}{\ell} + 6 + 2\log(2\varepsilon + 1) + o(1))$ *bits, while supporting the predict operation in* $O(\log \ell)$ *time via binary search.*
- $\ell\,(2\log \frac{u}{\ell} + 4 + 2\log(2\varepsilon + 1) + o(1)) + \log \binom{n}{\ell} + O(n/\log^c n)$ *bits for any constant* $c > 0$*, while supporting the predict operation in* $O(c)$*.*

Lastly, Abrav and Medvedev proposed the PLA-index [1] in a Bioinformatics application where the x- and y-values of the underlying sequence are increasing but not necessarily contiguous. They took an orthogonal approach to PLA compression by modifying the O'Rourke algorithm, enforcing each segment to start from the ending point of the previous one. This allows them to derive a more compact encoding, which may produce a suboptimal number of segments $\ell' \geq \ell$. As their approach does not guarantee optimality and targets different sequence types, we defer comparison to the journal version.



(a) **(b)**

Figure 1 The key parameters of PLAs arising in the two considered settings. Figure 1a shows an example in the compression case. The values on the x-axis are consecutive, the first segment always begins at position 1, and each segment starts at an x-value immediately after the ending of the previous one. Intercepts and the last y-values of each segment stay within a range of size $2\varepsilon + 1$ from the first and last covered values of the sequence. Notice how the intercepts do not necessarily form an increasing sequence. Figure 1b considers the indexing case, which is similar to the compression setting except that the values on the y-axis are consecutive. Hence, the last covered y-value of a segment is implicitly determined by the first covered y-value of the next one, and the intercepts β_is form a monotone sequence as $y_i + \varepsilon \leq y_{i+1} - \varepsilon$ since a segment covers at least 2ε keys [13, Lemma 2].

We conclude by noticing that none of these results focused on studying lower bounds to the cost of storing PLAs (possibly in compressed form). Therefore, the first novel contribution of our work lies in deriving the first compressibility measures for PLAs in the two aforementioned settings. It will turn out that the storage schemes for PLAs of the LA-vector and the PGM-index are asymptotically optimal up to a constant factor. Hence, our second contribution is to design the first data structures for the compression and indexing settings that achieve the space lower bounds plus small additive terms, which are *succinct* in most practical cases.

4 Compressibility Measures for PLAs

We begin by observing that, in general, the segments of a PLA, whether in the compression or indexing setting, have floating-point components. As discussed in Section 3, some existing compression schemes assume that segments either start or end at an integer y-coordinate to exploit integer compression techniques. In our context, we work under the same assumption for two reasons: first, when establishing a lower bound on the space required to represent a PLA, it is important to note that the class of PLAs with floating-point components is larger than that of PLAs with integer components. Consequently, any lower bound proven for the latter also applies to the former. Second, as proved by Ferragina & Lari [9] this reduction has a negligible impact on the approximation error of the resulting PLA, and in fact, it only increases that error by a constant 3. Therefore, without loss of generality, we can safely restrict our analysis to PLAs having segments with integer starting and ending ordinates.

Under this assumption, given the optimal number of segments ℓ, the approximation error ε, the universe size u, and the length of the sequence n, we consider the problem of counting the number of possible PLAs that can be constructed using these parameters, in both the compression and indexing cases. Then, by a simple information-theoretic argument, taking

the logarithm of such quantities gives a lower bound on the number of bits required to represent any PLA in the considered settings, and this provides a compressibility measure that any compression scheme should strive to achieve. Starting from PLAs arising in the compression setting, we prove the following theorem:

▶ **Theorem 6.** *Let $\mathcal{C}(\ell, \varepsilon, u, n, \mathbf{y})$ be the set of PLAs of ℓ segments having maximum error ε, covering a monotone sequence of n elements drawn from a universe of size u in the compression setting, and having $\mathbf{y} = y_1 \leq y_2 \leq \cdots \leq y_\ell$ as the first covered y-values. Then:*

$$|\mathcal{C}(\ell, \varepsilon, u, n, \mathbf{y})| = \binom{n-\ell-1}{\ell-1}\binom{u+\ell-1}{\ell}\left(\prod_{i=1}^{\ell-1}(y_{i+1}-y_i+1)\right)(2\varepsilon+1)^{2\ell}$$

Proof. Let us denote with x_i the first position covered by the i-th segment, and let us start by constraining each x_i in such a way as to obtain a valid PLA with error ε. Since the first segment always starts from the first position, it is $x_1 = 1$. Each segment covers at least two elements because there is always a segment connecting two points without incurring any error (i.e., the one connecting them), hence $x_{i+1} - x_i \geq 2$ (recall that in the compression setting the x-values are strictly increasing and contiguous).

To ease our counting, we introduce the following change of variables $\bar{x}_i = x_i - (i-1)$. In this way, $\bar{x}_2 = x_2 - 1 \geq 2$ and $\bar{x}_\ell = x_\ell - (\ell - 1) \leq n - \ell$, since $x_2 \geq 3$ and $x_\ell \leq n - 1$. Additionally, $\bar{x}_{i+1} - \bar{x}_i \geq 1$ (since the $\bar{x}_i$s are distinct) and thus $x_{i+1} - x_i = \bar{x}_{i+1} - \bar{x}_i + 1 \geq 2$. Therefore, choosing the first position of each segment while simultaneously satisfying the previous constraint on the x_is, is equivalent to choosing $\ell - 1$ distinct values from $\{2, 3, \ldots, n - \ell\}$. The total number of ways to perform this choice is then $\binom{n-\ell-1}{\ell-1}$.

Let β_i and y_i denote, respectively, the intercept of the i-th segment and its first covered y-value (see Figure 1a). Choosing β_i is equivalent to first selecting y_i and then choosing a value within the interval $[y_i - \varepsilon,\, y_i + \varepsilon]$, which contains $2\varepsilon + 1$ elements. In the compression setting, the input sequence may contain repeated values, so consecutive segments can share the same first covered y-value (i.e., $y_{i+1} = y_i$). Thus, choosing the first y-value of each of the ℓ segments amounts to selecting an increasing sequence of ℓ elements from $\{1, 2, \ldots, u\}$. The number of such choices is $\binom{u+\ell-1}{\ell}$. Since each β_i can then be any value of the interval $[y_i - \varepsilon,\, y_i + \varepsilon]$, the total number of possible selections for the ℓ intercepts is $\binom{u+\ell-1}{\ell}(2\varepsilon+1)^\ell$.

Lastly, let γ_i be the y-value given by the i-th segment evaluated on its last covered position (see Figure 1a). We perform the same reasoning as for the intercepts. Let y'_i be the value of the sequence on the last position covered by the i-th segment. Choosing γ_i is equivalent to first selecting y'_i and then choosing any of the values inside $[y'_i - \varepsilon, y'_i + \varepsilon]$. Now, since we are dealing with monotonically increasing sequences, to obtain a valid PLA, we need that $y'_i \geq y_i$ and $y'_i \leq y_{i+1}$ for every $1 \leq i < \ell$, notice that $y'_\ell = u$. Consequently, each y'_i can be independently chosen from the interval $[y_i, y_{i+1}]$, and each γ_i is then selected inside the interval $[y'_i - \varepsilon, y'_i + \varepsilon]$, therefore, the total number of ways to choose the ℓ last y-values of each segment is $\left(\prod_{i=1}^{\ell-1}(y_{i+1} - y_i + 1)\right)(2\varepsilon+1)^\ell$.
Combining these quantities, our claim follows. ◀

Theorem 6 suggests the following lower bound on the number of bits that any lossless compressor for PLAs in the compression setting must output.

▶ **Corollary 7.** *The minimum number of bits to represent any PLA from* $\mathcal{C}(\ell, \varepsilon, u, n, \mathbf{y})$ *is:*

$$\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) = \log \binom{n-\ell-1}{\ell-1} + \log \binom{u+\ell-1}{\ell} + \sum_{i=1}^{\ell-1} \log\left(y_{i+1} - y_i + 1\right) + 2\ell \log\left(2\varepsilon + 1\right)$$

Next, we focus on PLAs arising in the indexing setting. Due to space limitations, we present only the main result here and defer the proof to the full version of this paper [8]. We only mention that the proof follows the same strategy as that of Theorem 6, with the only difference that in the indexing setting, each segment (except possibly the last one) covers at least 2ε keys (see [13, Lemma 2]), thus reducing the possible choices for the first covered x- and y-values.

▶ **Theorem 8.** *Let* $\mathcal{I}(\ell, \varepsilon, u, n, \mathbf{x})$ *be the set of PLAs of* ℓ *segments having maximum error* ε*, covering a strictly increasing sequence of* n *elements drawn from a universe of size* u *in the indexing setting, and having* $\mathbf{x} = x_1 < x_2 < \cdots < x_\ell$ *as the first covered* x*-values. Then:*

$$|\mathcal{I}(\ell, \varepsilon, u, n, \mathbf{x})| = \binom{u-\ell(2\varepsilon-1)}{\ell} \binom{n-\ell(2\varepsilon-1)-1}{\ell-1} \left(\prod_{i=1}^{\ell-1} (x_{i+1} - x_i - 1)\right) (2\varepsilon+1)^{2\ell}$$

Theorem 8 gives the following lower bound on the number of bits that any lossless compressor has to produce when applied to PLAs in the indexing setting.

▶ **Corollary 9.** *The minimum number of bits to represent any PLA from* $\mathcal{I}(\ell, \varepsilon, u, n, \mathbf{x})$ *is:*

$$\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x}) = \log \binom{u-\ell(2\varepsilon-1)}{\ell} + \log \binom{n-\ell(2\varepsilon-1)-1}{\ell-1} + \sum_{i=1}^{\ell-1} \log\left(x_{i+1} - x_i - 1\right) + 2\ell \log\left(2\varepsilon + 1\right)$$

As already mentioned, unlike in the compression setting, segments always cover at least 2ε keys (except possibly the last one). Therefore, the lower bound of Corollary 9 can be made independent from the choice of the first covered x-values, by noting that $x_{i+1} - x_i \geq 2\varepsilon$. Nevertheless, this more general form of the lower bound comes at the cost of underestimating the minimum number of bits required to store any PLA in the indexing case when segments span more than 2ε elements.

Having established lower bounds on the space required to represent lossless any PLA in both the compression and indexing settings, we now shift our focus to designing data structures that use space close to $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$ and $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x})$ while efficiently supporting the core operation of any learned data structure hinging on PLAs (e.g., [13, 2, 23, 7, 14, 9]), namely, returning the y-value *predicted* by the segment covering a given x-value.

5 Succinct Data Structures for PLAs

We begin by establishing constraints on the parameters of a PLA that are commonly observed in many practical applications. Specifically, we assume that $\epsilon = O(1)$, $\ell = o(n)$ and $n = o(u)$. Next, we proceed by formalizing the core operation performed on any PLA in both the compression and indexing settings. In particular, we define the *predict*(x) operation, which, given a value lying on the x-axis, identifies the segment covering x and computes the

corresponding y-value *predicted* by that segment. Designing data structures that efficiently support this operation while achieving space usage close to the lower bounds of Section 4 is of both practical and theoretical interest since it can help to understand what the limits of PLA-based learned data structures are in terms of the redundancy needed to store and query the PLA itself, and possibly improve the performances of such data structures.

At the end of Section 2, we commented on the relation between the current literature and our lower bounds, highlighting that the storage schemes for PLAs of the LA-vector and the PGM-index are *compact* [18]. Due to space constraints, we sketch the main result here; please refer to the full version of this paper for the complete proofs [8]. Starting with the LA-vector (see Section 5.1), it turns out that the storage space of Lemma 5 can be expressed as $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$ with an additive term that is either $O(\ell \log \frac{u}{\ell})$ or $O(\ell \log \frac{u}{\ell} + \frac{n}{\log^c n})$ depending if queries are answered with a binary search or with the data structure of Pătraşcu [21]. In the first case, the overhead is always $O(\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}))$, while in the second case, that bound holds only when $\ell = \omega(n/(\log u \log^c n))$. As a result, the storage space of the LA-vector is asymptotically optimal up to a constant factor, and thus is compact. Moving to the PGM-index (see the full version of this paper [8]), the additive term on top of the lower bound $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x})$ is either $\Theta(\ell \log u)$ or $O(\ell \log u + \frac{u}{\log^c u})$, depending on how queries are answered. In the first case, the overhead is always $\Theta(\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x}))$, while in the second case that bound holds only when $\ell = \Omega(u/\log^{c+1} u)$. Therefore, the storage space of the PGM-index is also compact.

Motivated by this, the following sections will describe the design of the first data structures for the compression and indexing settings, achieving small additive terms to the lower bounds of Section 4, which turn out to be succinct in most practical cases, and most importantly provide the same asymptotic query time as the LA-vector and the PGM-index.

5.1 Using space close to $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$

Given a PLA $\{s_1, s_2, \ldots, s_\ell\} \in \mathcal{C}(\ell, \varepsilon, u, n, \mathbf{y})$, we represent a generic segment s_i as a tuple $s_i = (x_i, \beta_i, \gamma_i, y_i, y'_i)$ where x_i is the first covered x-value of the input sequence, β_i is the intercept, γ_i is the last y-value given by the segment, y_i and y'_i are respectively the first and last y-values of the input sequence covered by the i-th segment (see Figure 1a). To encode each segment, we begin by storing a bit sequence X containing the first x-value covered by each segment. Specifically, X encodes in unary the deltas between the first x-values of consecutive segments (i.e., between x_{i+1} and x_i), shifted by a constant to further reduce space usage. Such a sequence is defined as follows:

- $X = 0^{x_2 - x_1 - 2} 1 0^{x_3 - x_2 - 2} 1 \ldots 0^{x_\ell - x_{\ell-1} - 2} 1$. The sequence is well-defined since $x_{i+1} - x_i \geq 2$. The length of X is $\sum_{i=1}^{\ell-1}(x_{i+1} - x_i - 1) = x_\ell - x_1 - (\ell - 1)$, which is upper bounded by $n - \ell - 1$ since $x_1 = 1$ and $x_\ell \leq n - 1$, and contains $\ell - 1$ ones. The i-th element of X can be accessed by noticing that $x_1 = 1$ and for any $i > 1$, $x_i = \mathit{select}_1(X, i - 1) + i$.

Next, we store an integer sequence Y encoding the first covered y-values of each segment. Given our assumption that $\ell = o(n)$, both the X and Y sequences are sparse. Since they are also monotonically increasing by construction, we represent them using the Elias–Fano encoding scheme[2]. By Theorem 2, the space in bits required to store X and Y is:

$$(\ell - 1) \log \left(\frac{n - \ell - 1}{\ell - 1} \right) + \ell \log \left(\frac{u}{\ell} \right) + 4\ell + o(\ell) \tag{2}$$

[2] We turn X into an integer sequence by interpreting each unary code as an integer.

Noting that $\log\binom{u+\ell-1}{\ell} = \log\binom{u}{\ell} + O(\ell)$ (since $\binom{u+\ell-1}{\ell} = \binom{u}{\ell} \cdot \left(\prod_{i=1}^{\ell-1}(u+i)/\prod_{i=1}^{\ell-1}(u-i)\right)$ and the second factor can be bounded by $c^{\ell-1}$ for some constant $c > 1$) and applying the Elias-Fano space bound from Equation 1, we can rewrite Equation 2 in the following form:

$$\log\binom{n-\ell-1}{\ell-1} + \log\binom{u+\ell-1}{\ell} + O(\ell + \log u) \tag{3}$$

Next we consider the sequence of last covered y-values of each segment, i.e. $y'_1, y'_2, \ldots, y'_{\ell-1}$, omitting y'_ℓ as it is always equal to u. Since each $y'_i \in [y_i, y_{i+1}]$, we store the relative values using just $\log(y_{i+1} - y_i + 1)$ bits and concatenate their representation in a single bit vector B, of size $|B| = \sum_{i=1}^{\ell-1} \log(y_{i+1} - y_i + 1)$ bits. To access each y'_i, we maintain an auxiliary array P, where each p_i is the starting index of the binary representation of y'_i inside B. Notice that this technique resembles the *dense pointers* method of Ferragina & Venturini [12]. Array P is both sparse and monotonically increasing, hence we store it with the Elias-Fano encoding scheme using the following number of bits (recall Theorem 2):

$$2(\ell-1) + (\ell-1)\log\left(\frac{\sum_{i=1}^{\ell-1}\log(y_{i+1} - y_i + 1)}{\ell-1}\right) + o(\ell) \tag{4}$$

Using Jensen's inequality, and noticing that $\sum_{i=1}^{\ell-1}(y_{i+1} - y_i + 1) = y_\ell + \ell - 1$ which is at most $u + \ell - 1$ since $y_\ell \le u$, we obtain that $|P| = O(\ell \log\log \frac{u}{\ell})$ bits. Therefore, the sequence of the last covered y-values is encoded as the bit vector B and the Elias-Fano encoding of the array P, using the following number of bits:

$$\left(\sum_{i=1}^{\ell-1}\log(y_{i+1} - y_i + 1)\right) + O(\ell \log\log \frac{u}{\ell}) \tag{5}$$

Lastly, since each segment is an ε-approximation for the points it covers, it holds that $|\beta_i - y_i| \le 2\varepsilon$ and $|\gamma_i - y'_i| \le 2\varepsilon$. Therefore we store each intercept β_i and each last y-value γ_i as deltas with respect to y_i and y'_i inside two arrays Δ_β and Δ_γ using $2\ell\log(2\varepsilon+1)$ bits. Combining this with Equations 3 and 5, and recalling Corollary 7, the space usage of our storage scheme is $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) + O(\ell \log\log \frac{u}{\ell} + \log u)$ bits. That is succinct space, since $O(\ell \log\log \frac{u}{\ell} + \log u) \subseteq o(\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}))$ because $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$ includes a term that is $\Theta(\ell \log \frac{u}{\ell})$, and we consider the case that $\ell = \Omega(\log u / \log\log u)$.

Without introducing any extra space on top of this representation, performing a *predict*(x) query proceeds as detailed in Algorithm 1. For simplicity, we assume that the segment covering x is never the first one nor the last one. Corner cases can be handled in constant time without altering the query time.

Because of Theorem 2, all the *select* operations on the Elias-Fano encoded sequences X, Y, and P take $O(1)$ time. Reading the binary representation of numbers from B, as well as accessing the deltas inside Δ_β and Δ_γ, requires a $O(1)$ number of bitwise and bit-shifting operations (see [18, Chapter 3] for the technical details), hence $O(1)$ time in the transdichotomous word RAM model we are assuming. The most expensive part of Algorithm 1 is then the predecessor search on X, which dominates the cost of a *predict* query, making the overall time $O(\log \ell)$.

To lower the cost of the predecessor search, and thus achieve faster query times, one must allow for a larger additive term to the lower bound $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$. To address this, we directly store the sequence of first covered x-values as $X = x_2, x_3, \ldots, x_\ell$, whose universe is bounded by $n - 1$ and contains $\ell - 1$ values (recall $x_1 = 1$ in the compression setting).

Algorithm 1 Performs a *predict* operation on our compressed storage scheme for PLAs.

Input : The arrays of first covered x-values and y-values X and Y, the bit-vector B, the array of offsets P, the array of deltas Δ_β and Δ_γ, and a value x.
Output : A value y such that $predict(x) = y$.

$i = pred(X, x)$; `// predecessor search on` X `using a binary search`
$x_i = select_1(X, i-1) + i$;
$x_{i+1} = select_1(X, i) + (i+1)$;
$y_i = select_1(Y, i)$;
$s_i = select_1(P, i)$;
$e_i = select_1(P, i+1) - 1$;
$y'_i = read(B, s_i, e_i)$; `// read the bit sequence` $B[s_i, e_i]$ `representing` y'_i
$\beta_i = y_i + \Delta_\beta[i]$;
$\gamma_i = y'_i + \Delta_\gamma[i]$;
return $\left\lfloor (x - x_i)\frac{(\gamma_i - \beta_i)}{(x_{i+1} - x_i)} \right\rfloor + \beta_i$;

This sequence allows the predecessor of any given position (and, consequently, the index of the segment covering that position) to be computed via a simple *rank* operation. X is then stored using Theorem 1, which guarantees that for any constant $c > 0$, both *rank* and *select* operations take $O(c)$ time while requiring $\log \binom{n-1}{\ell-1} + O\left(\frac{n}{\log^c n}\right)$ bits of space.

Since we assumed that $\ell = o(n)$, we refactor the term $\log \binom{n-1}{\ell-1}$ as follows:

$$\begin{aligned}
\log \binom{n-1}{\ell-1} &= \log \binom{n-\ell-1}{\ell-1} + \log \left(\frac{\prod_{i=1}^{\ell}(n-i)}{\prod_{i=\ell}^{2\ell-1}(n-i)} \right) \\
&\leq \log \binom{n-\ell-1}{\ell-1} + \ell \log \left(\frac{n-1}{n-2\ell+1} \right) \\
&= \log \binom{n-\ell-1}{\ell-1} + O(\ell)
\end{aligned}$$

Where the inequality is obtained by upper-bounding each term in the numerator by $n-1$ and lower-bounding each term in the denominator by $n - 2\ell + 1$. The space usage is then:

$$\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) + O(\ell \log\log \frac{u}{\ell} + \frac{n}{\log^c n}) \tag{6}$$

We now show that the space usage of this second data structure is succinct. To this end, we use the well-known upper and lower bounds on the logarithm of the binomial (i.e., $k \log \frac{m}{k} \leq \log \binom{m}{k} \leq k \log (\frac{em}{k})$, where $e \approx 2.718$ is the Euler's number, and thus $\log \binom{m}{k} = \Theta(k \log \frac{m}{k})$), to rewrite the lower bound of Corollary 7 as follows:

$$\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) = \Theta(\ell(\log \frac{u}{\ell} + \log \frac{n}{\ell})) = \Theta(\ell \log \frac{u}{\ell}) \tag{7}$$

Where the Ω-term comes from the first two additive terms in the formula of $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y})$ and the approximation of the logarithm of the binomial coefficient written above. The $O()$-term comes similarly and from upper bounding the summation in Corollary 7 using Jensen's inequality, as already done in Section 5.1.

Focusing on the additive term of Equation 6, it is easy to see that $O(\ell \log\log \frac{u}{\ell} + \frac{n}{\log^c n}) \subseteq o(\ell \log \frac{u}{\ell} + \frac{n}{\log^c n})$, by considering $\ell = \omega(n/(\log u \log^c n))$ which implies that $\ell \log \frac{u}{\ell} = \omega(\frac{n}{\log^c n})$. Therefore, from Equation 7, it is $O(\ell \log\log \frac{u}{\ell} + \frac{n}{\log^c n}) \subseteq o(\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}))$ for those values of ℓ that are $\omega(n/(\log u \log^c n))$ and do not violate our initial assumption that $\ell = o(n)$.

As a result, we proved the following theorem, which gives data structures able to store any PLA in the compression setting using space close to the lower bound of Corollary 7 while supporting efficient *predict* queries.

▶ **Theorem 10.** *There exist data structures storing any PLA from* $\mathcal{C}(\ell, \varepsilon, u, n, \mathbf{y})$ *as follows:*

- *Using* $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) + O(\ell \log\log \frac{u}{\ell} + \log u)$ *bits of space, while supporting predict in* $O(\log \ell)$ *time.*
- *Using* $\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}) + O(\ell \log\log \frac{u}{\ell} + n/\log^c n)$ *bits of space for any constant* $c > 0$*, while supporting predict in* $O(c)$ *time.*

We conclude by comparing Theorem 10 with the storage scheme for PLAs of the LA-vector (see Lemma 5). We start by considering the one solving the *predict* queries with a binary search on the first covered positions, reporting its space usage in bits here to facilitate comparison:

$$\ell(2\log\frac{u}{\ell} + \log\frac{n}{\ell} + 6 + 2\log(2\varepsilon+1) + o(1)) \tag{8}$$

We notice that the term $2\ell\log(2\varepsilon+1)$ is shared by both Corollary 7 and Equation 8, hence we only focus on the remaining parts. Since we assumed that $\ell = o(n)$, the terms $\ell\log\frac{u}{\ell}$ and $\ell\log\frac{n}{\ell}$ appearing in Equation 8 can be respectively bounded as $\log\binom{u+\ell-1}{\ell} + O(\ell + \log u)$ and $\log\binom{n-\ell-1}{\ell-1} + O(\ell + \log n)$ (see Section 2 and 5.1). Consequently, Equation 8 can be rewritten in the following form, which is closer to the one of Corollary 7.

$$\log\binom{n-\ell-1}{\ell-1} + \log\binom{u+\ell-1}{\ell} + \ell\log\frac{u}{\ell} + 2\ell\log(2\varepsilon+1) + O(\ell + \log u) \tag{9}$$

Lastly, the summation $\sum_{i=1}^{\ell-1}\log(y_{i+1} - y_i + 1)$ appears in the lower bound of Corollary 7, but not in Equation 9 which in turns contain also the additive terms $\ell\log\frac{u}{\ell} + O(\ell + \log u)$.

Now, the minimum of the summation is $\ell - 1$, and this occurs when each segment covers exactly two positions. We rule out the case in which it is zero, as this would force $y_{i+1} = y_i$, thus only considering PLAs formed by a single segment. In this way, the gap between the lower bound and the storage space of the LA-vector is thus $\Theta(\ell\log\frac{u}{\ell}) = \Theta(\mathcal{B}_{\mathcal{C}}(\ell, \varepsilon, u, n, \mathbf{y}))$. Hence, the storage scheme for PLAs of the LA-vector with the correction of Ferragina & Lari is *compact*, because it is up to a constant factor from the optimal. Therefore, our first data structure in Theorem 10 offers an even smaller additive term to the optimal space usage, while having the same asymptotic query time. Specifically, assuming $\ell = \Omega(\log u/\log\log u)$, it exponentially reduces the overhead, from $O(\ell\log\frac{u}{\ell})$ to $O(\ell\log\log\frac{u}{\ell})$.

Similarly, an analogous result holds for the two data structures described in Lemma 5 and Theorem 10, which answer queries in $O(c)$ time for any constant $c > 0$. In this case, it is sufficient to notice that term due to the data structure of Pătraşcu is shared by both, and the additive term of the LA-vector becomes $O(\ell\log\frac{u}{\ell} + \frac{n}{\log^c n})$. Therefore, using the same argument we used for proving the succinctness of the second data structure in Theorem 10, it is possible to show that for $\ell = \omega(n/\log u\log^c n)$ the storage scheme of the LA-vector is again compact, while ours is succinct, and thus offering a lower overhead to the optimal space usage without altering its asymptotic query time.

In conclusion, we remark that, beyond its theoretical appeal, the first data structure of Theorem 10 is also practical, as it leverages techniques with highly engineered implementations, such as the Elias-Fano encoding of monotone sequences [16] and arrays of fixed- and variable-size elements [18]. Indeed, under the same assumption that $\ell = \Omega(\log u/\log\log u)$, our storage scheme asymptotically requires just $O(\log\log\frac{u}{\ell})$ bits per segment over the theoretical

minimum, while still providing practically fast *predict* queries. Conversely, the second data structure is mostly theoretical, and it aims at showing how close we can approach the lower bound of Corollary 7 while still supporting (optimal) constant time *predict* queries.

5.2 Using space close to $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x})$

The compression and indexing settings are closely related, the only difference lies in which axis is constrained to have consecutive values. Indeed, this similarity can also be observed in the two lower bounds of Corollary 7 and 9 sharing the same form. As a result, the design of a data structure that achieves space usage close to the lower bound for the indexing setting, while supporting efficient *predict* queries, builds upon the same ideas and algorithmic techniques introduced in Section 5.1. Due to space constraints, we present only the main result in this section, deferring the complete to the full version of this paper [8].

▶ **Theorem 11.** *There exist data structures storing any PLA from $\mathcal{I}(\ell, \varepsilon, u, n, \mathbf{x})$ as follows:*

- *Using $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x}) + O(\ell \log\log \frac{u}{\ell} + \log u)$ bits of space, while supporting the predict operation in $O(\log \ell)$ time.*
- *Using $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x}) + O(\ell \log\log \frac{u}{\ell} + u/\log^c u)$ bits of space for any constant $c > 0$, while supporting the predict operation in $O(c)$ time.*

The storage scheme of the PGM-index is optimal up to a constant factor. Considering the first data structure of Lemma 4, it can be expressed as $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x}) + \Theta(\ell \log u)$. As a result, Theorem 11 introduces the first data structure storing any PLA in the indexing setting within lower-order terms to $\mathcal{B}_{\mathcal{I}}(\ell, \varepsilon, u, n, \mathbf{x})$, under the assumption that $\ell = \Omega(\log u / \log\log u)$, with the same asymptotic query time (see the full version of this paper [8]).

The same observations of Section 5.1 regarding the practicality of our data structures hold. The first one is the most practical, while the second is just theoretical and shows how close the lower bound can be approached while answering queries in (optimal) constant time.

6 Conclusions

In this work, we addressed the problem of deriving the first compressibility measures for PLAs of monotone sequences in two distinct settings: *compression* and *indexing*, which are general enough to cover a broad range of applications. The measures we obtained provide a theoretical foundation for understanding the space requirements of any learned data structure relying on PLAs in any of its components.

Based on our compressibility measures, we analyzed two representative data structures for storing PLAs, namely the one of the LA-vector (compression setting) and the one of the PGM-index (indexing setting), showing that they are both optimal up to a constant factor.

Motivated by this study, we proposed two novel data structures for the above settings, which turn out to be succinct (thus optimal up to lower-order terms) in most practical cases while retaining the same asymptotic query time of the LA-vector and PGM-index. Overall, they offer strong theoretical guarantees and suggest efficient practical implementations, providing valuable tools for practitioners and researchers in learned data structures.

Future works could investigate an extension of our analysis to *Piecewise Non-Linear Approximations*, for which successful applications already exist [14]. Another interesting direction, complementary to our work, is the implementation of the proposed data structures with a thorough experimental evaluation of their space-time trade-offs.

References

1 Md. Hasin Abrar and Paul Medvedev. Pla-index: A k-mer index exploiting rank curve linearity. In *24th International Workshop on Algorithms in Bioinformatics, WABI 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 312 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.WABI.2024.13.

2 Antonio Boffa, Paolo Ferragina, and Giorgio Vinciguerra. A learned approach to design compressed rank/select data structures. *ACM Trans. Algorithms*, 18(3):24:1–24:28, 2022. doi:10.1145/3524060.

3 Luis Alberto Croquevielle, Guang Yang, Liang Liang, Ali Hadian, and Thomas Heinis. Beyond logarithmic bounds: Querying in constant expected time with learned indexes. In *28th International Conference on Database Theory, ICDT 2025, March 25-28, 2025, Barcelona, Spain*, volume 328 of *LIPIcs*, pages 19:1–19:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.ICDT.2025.19.

4 Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. *Proc. VLDB Endow.*, 14(2):74–86, 2020. doi:10.14778/3425879.3425880.

5 Peter Elias. Efficient storage and retrieval by content and address of static files. *J. ACM*, 21(2):246–260, 1974. doi:10.1145/321812.321820.

6 Robert M. Fano. On the number of bits required to implement an associative memory. Technical report, Massachusetts Institute of Technology, Project MAC, Computer Structures Group, 1971.

7 Paolo Ferragina, Marco Frasca, Giosuè Cataldo Marinò, and Giorgio Vinciguerra. On nonlinear learned string indexing. *IEEE Access*, 11:74021–74034, 2023. doi:10.1109/ACCESS.2023.3295434.

8 Paolo Ferragina and Filippo Lari. Compressibility Measures and Succinct Data Structures for Piecewise Linear Approximations, 2025. arXiv:2509.07827.

9 Paolo Ferragina and Filippo Lari. FL-RMQ: A learned approach to range minimum queries. In *36th Annual Symposium on Combinatorial Pattern Matching, CPM 2025, June 17-19, 2025, Milan, Italy*, volume 331 of *LIPIcs*, pages 7:1–7:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.CPM.2025.7.

10 Paolo Ferragina, Hans-Peter Lehmann, Peter Sanders, and Giorgio Vinciguerra. Learned monotone minimal perfect hashing. In *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 46:1–46:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.46.

11 Paolo Ferragina, Fabrizio Lillo, and Giorgio Vinciguerra. On the performance of learned data structures. *Theor. Comput. Sci.*, 871:107–120, 2021. doi:10.1016/J.TCS.2021.04.015.

12 Paolo Ferragina and Rossano Venturini. A simple storage scheme for strings achieving entropy bounds. *Theor. Comput. Sci.*, 372(1):115–121, 2007. doi:10.1016/J.TCS.2006.12.012.

13 Paolo Ferragina and Giorgio Vinciguerra. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proc. VLDB Endow.*, 13(8):1162–1175, 2020. doi:10.14778/3389133.3389135.

14 Andrea Guerra, Giorgio Vinciguerra, Antonio Boffa, and Paolo Ferragina. Learned compression of nonlinear time series with random access. In *Proc. 41st IEEE International Conference on Data Engineering (ICDE)*, 2025.

15 Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 489–504. ACM, 2018. doi:10.1145/3183713.3196909.

16 Danyang Ma, Simon J. Puglisi, Rajeev Raman, and Bella Zhukova. On elias-fano for rank queries in fm-indexes. In *31st Data Compression Conference, DCC 2021, Snowbird, UT, USA, March 23-26, 2021*, pages 223–232. IEEE, 2021. doi:10.1109/DCC50243.2021.00030.

17 Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. Learning multi-dimensional indexes. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 985–1000. ACM, 2020. doi:10.1145/3318464.3380579.

18 Gonzalo Navarro. *Compact Data Structures – A Practical Approach*. Cambridge University Press, 2016.

19 Daisuke Okanohara and Kunihiko Sadakane. Practical entropy-compressed rank/select dictionary. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM, 2007. doi:10.1137/1.9781611972870.6.

20 Joseph O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Commun. ACM*, 24(9):574–578, 1981. doi:10.1145/358746.358758.

21 Mihai Pătraşcu. Succincter. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 305–313. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.83.

22 Mihai Pătraşcu and Emanuele Viola. Cell-probe lower bounds for succinct partial sums. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 117–122. SIAM, 2010. doi:10.1137/1.9781611973075.11.

23 Kapil Vaidya, Tim Kraska, Subarna Chatterjee, Eric R. Knorr, Michael Mitzenmacher, and Stratos Idreos. SNARF: A learning-enhanced range filter. *Proc. VLDB Endow.*, 15(8):1632–1644, 2022. doi:10.14778/3529337.3529347.

24 Youyun Wang, Chuzhe Tang, Zhaoguo Wang, and Haibo Chen. Sindex: a scalable learned index for string keys. In Taesoo Kim and Patrick P. C. Lee, editors, *APSys '20: 11th ACM SIGOPS Asia-Pacific Workshop on Systems, Tsukuba, Japan, August 24-25, 2020*, pages 17–24. ACM, 2020. doi:10.1145/3409963.3410496.

25 Sepanta Zeighami and Cyrus Shahabi. On distribution dependent sub-logarithmic query time of learned indexing. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 40669–40680, 2023. URL: https://proceedings.mlr.press/v202/zeighami23a.html.

26 Sepanta Zeighami and Cyrus Shahabi. Towards establishing guaranteed error for learned database operations. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL: https://openreview.net/forum?id=6tqgL8VluV.