FL-RMQ: A Learned Approach to Range Minimum Queries

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 address the problem of designing and implementing a data structure for the Range Minimum Query problem. We show a surprising connection between this classical problem and the geometry of a properly defined set of points in the Cartesian plane. Building on this insight, we hinge upon a well-known result in Computational Geometry to introduce the first RMQ solution that exploits (i.e., *learns*) the distribution of such 2D-points via proper error-bounded linear approximations. Because of these features, we name the resulting data structure: *Fully-Learned* RMQ, shortly `FL-RMQ`.

We prove theoretical bounds for its space usage and query time, covering both worst-case scenarios and average-case performance for uniformly distributed inputs. These bounds compare favorably with the ones achievable by the best-known indexing solutions (i.e., the ones that allow access to the indexed array), especially when the input data follow some geometric regularities that we characterize in the paper, thus providing principled evidence of `FL-RMQ` being a novel data-aware solution to the RMQ problem. We corroborate our theoretical findings with a wide set of experiments showing that `FL-RMQ` offers more robust space-time trade-offs than the other known practical indexing solutions on both artificial and real-world datasets.

We believe that our novel approach to the RMQ problem is noteworthy not only for its interesting space-time trade-offs, but also because it is flexible enough to be applied easily to the encoding variant of RMQ (i.e., the one that does not allow access to the indexed array), and moreover, because it paves the way to research opportunities on possibly other problems.

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

Keywords and phrases Range-Minimum query, Learned data structures, Compact data structures, Experimental results

Digital Object Identifier 10.4230/LIPIcs.CPM.2025.7

Supplementary Material *Software (Source Code)*: `https://github.com/FilippoLari/FL-RMQ` archived at `swh:1:dir:8cc230ba6c318788d428dc4eb92f628d4ac3dde0`

Funding This work was partially supported by the Alfred P. Sloan Foundation with the grant #G-2025-25193 (`sloan.org`); by the NextGenerationEU – 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 spoke "FutureHPC & BigData" of the ICSC – Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing funded by European Union – NextGenerationEU – PNRR.

Acknowledgements We thank Johannes Fischer for his insightful suggestions on an earlier version of this work, the anonymous reviewers for their feedback, and the staff of the Green Data Center at the University of Pisa for providing us with the machines and technical support for executing the numerous experiments detailed in this paper.

[1] Corresponding author

1 Introduction

In today's digital world, an enormous amount of data is generated on a daily basis. This introduces new challenges in designing several advanced software systems. Among others, much attention is devoted to computational problems involving variable-length keys (aka, strings), which undoubtedly constitute a core component of a plethora of big-data applications such as search engines [40, 35, 42], RDF and key-value stores [54, 51], scalable distributed storage systems [13], computational biology tools [3, 52], Large Language Models [7], and n-gram language models [41, 65, 50], just to mention a few.

Recently, motivated by such rapid growth of data, researchers have started to investigate novel methods for designing algorithms and data structures that leverage how those data are distributed to deliver faster or more succinct solutions. The seminal work of Kraska et. al [45] has shaken the foundations of the decades-old field of indexing data structures, showing how replacing well-established design elements, such as B-Tree nodes, with *Machine Learning* (ML) models leads to significant improvements in time efficiency and memory usage. Following this line of work, numerous solutions have been proposed in the indexing scenario, respectively in the one-dimensional case [28, 20, 72, 77, 15, 48] and the multi-dimensional one [21, 64, 58, 63], and their effectiveness have also been evaluated in theory [25, 78]. However, the benefits of such a novel combination are not only limited to the problem of indexing, but several other classical problems have been revisited under this paradigm such as approximate membership queries [66, 19, 73, 68], range filtering [17, 74], frequency estimation [16], compressed *rank* and *select* dictionaries [10, 24], and sorting [46, 47, 12, 69].

Considering such premises, here we focus on the *Range Minimum Query* (RMQ) problem, which finds applications in many string-related tasks such as pattern matching [2, 18], text indexing [31, 67], text compression [61, 14, 49], and string mining [30, 39]. Formally, given an array $A[1..n]$ of n items drawn from a totally ordered universe, the RMQ problem is to design a data structure that returns for any given range $A[i, j]$ the position of the leftmost minimum, indicated hereafter with $RMQ_A(i, j)$. Such a problem is generally framed in two settings: the *indexing setting*, where the data structure can access the underlying array A, and the *encoding setting*, where the array A cannot be accessed.

In the indexing setting, solutions are pretty simple and based on a block decomposition of the input array A or a sparse version of the naive lookup table [5]. It turns out that any indexing solutions using $O(n/C)$ bits must have $\Omega(C)$ query time [11]. An optimal solution matching this lower bound, although not practical, is the one by Fischer & Heun [29].

In the encoding setting, most solutions rely on a sequence of reductions from RMQs on A to *Lowest Common Ancestor* (LCA) queries on a tree representation of A, called *Cartesian tree*. In this setting, the information-theoretic lower bound constraints any encoding solution to use at least $2n - \Theta(\log n)$ bits. Much work has been devoted to efficiently solving LCAs on succinct representations of the Cartesian tree, and the first solution matching such lower bound (up to lower order term) is again due to Fischer & Heun [29]. Recently, more practical encoding approaches have been proposed [36, 23, 4], and also *compressed* RMQ data structures have been investigated [34, 56, 38].

In this article, we focus for the first time in the literature on designing an indexing data structure for the RMQ problem via a learned-based approach. Indeed, we show a surprising connection between RMQs on A and the geometry of a properly defined set of 2D-points in the Cartesian plane. We then hinge on a classical result in computational geometry to design a novel data-aware indexing scheme for RMQs that deploys *piecewise-linear approximations* of the distribution of these points to learn the mapping between properly encoded ranges

Table 1 A summary of the indexing solutions, highlighting the space usage and query time of each data structure. (§) $c(n)$ must be $O(n^\delta)$ for some $0 < \delta < 1$. (†) ℓ is the number of linear models having maximum error ε used by our novel data-aware approach. (‡) Average case complexities on uniformly distributed inputs.

Data Structure	Additional Space (bits)	Query Time
Full Table	$\Theta(n^2 \log n)$	$O(1)$
Block Decomposition	$\Theta(\frac{n}{b} \log b)$	$O(b + \frac{n}{b})$
setting $b = \sqrt{n}$	$\Theta(\sqrt{n} \log n)$	$O(\sqrt{n})$
Bender & Farach-Colton [5]	$\Theta(n \log^2 n)$	$O(1)$
Fischer & Heun [29, Thm. 3.7] (§)	$\frac{2n}{c(n)} - \Theta(\frac{n \log\log n}{c(n) \log n})$	$O(c(n))$
Fischer & Heun [29, Lem. 3.6]	$2n - \frac{6n \log\log n}{\log n} + O(\frac{n}{\log n})$	$O(1)$
Fischer & Heun [29, Thm. 4.1]	$nH_k + O(\frac{n}{\log n}(k \log \sigma + \log\log n))$	$O(1)$
Theorem 1 + Corollary 4 (†)	$4\ell(\log n + \log\log n) + O(\log^2 n)$	$O(\log \ell + \varepsilon)$
Theorem 5 + Corollary 4 (†)	$3\ell(\log \frac{n}{\ell} + \log\log n + 2 + o(1))$ $+2\ell \log(2\varepsilon + 1) + O(\log^2 n)$	$O(\log \ell + \varepsilon)$
Corollary 8 (‡)	$O(\frac{n}{\varepsilon} \log n + \log^2 n)$	$O(\log \frac{n}{\varepsilon} + \varepsilon)$
setting $\varepsilon = \log n$	$O(n)$	$O(\log n)$

of A and the positions of their leftmost minimum (i.e., their RMQs). As a result, the performance of our data structure depends on the number ℓ of such linear functions (aka, models), which is strongly related to the underlying data distribution, and can be several orders of magnitudes smaller than the input size (see Section 4). Because of its algorithmic features, we name our data structure *Fully-Learned RMQ*, hence shortly `FL-RMQ`. We establish theoretical bounds for its space usage and query time, covering both worst-case scenarios and average-case performance for uniformly distributed inputs, which compare favorably with the best-known indexing solutions (see Table 1 and Section 3). Finally, we corroborate our theoretical findings with a wide set of experiments showing that our `FL-RMQ` offers more robust space-time trade-offs than the other known practical indexing solutions on both artificial and real-world datasets (see Section 4).

We believe that our approach is noteworthy not only for the novel space-time trade-offs it introduces for the RMQ problem, but also because it extends the learned-based design paradigm to a new problem (indeed RMQ), it is flexible enough to be applied easily to its encoding variant (as we comment in Section 5), and it paves the way to further research opportunities on possibly other problems too.

2 Related Work

A straightforward solution in the indexing setting (i.e., the one that allows access to the indexed array) is to store the answer for all the $\binom{n}{2}$ valid queries in a table, and then look up the answer in (optimal) constant time. Another technique is to perform a so-called *block decomposition*, where the input array is divided into blocks of size b, and only the answers for those subarrays are stored. Each query is divided into at most three subqueries: one *out-of-block* query that spans several blocks, and two *in-block* queries (completely contained inside a block). In-block queries require at most two linear scans of b elements, while out-of-block queries are answered by comparing $O(n/b)$ block minima. The final answer is obtained by comparing at most three values and overall requires $O(b + n/b)$ time while using $\Theta(n/b \log b)$ bits of space. A value for b that minimizes the query time is $\sqrt{n}$.

Bender & Farach-Colton [5] presented an elegant solution taking $\Theta(n \log n)$ memory words, which uses a sparse version of the naive lookup table, called *sparse table*. The main idea is to compute the answers for all queries whose range size is a power of two, i.e. a table $T[1..n][1..\lceil \log n \rceil]$ with $T[i,j] = RMQ_A(i, i + 2^j - 1)$. An arbitrary query $RMQ_A(i,j)$ is eventually answered by determining first the maximal k such that $2^k \leq j - i + 1$, and then comparing $A[T[i,k]]$ and $A[T[j - 2^k + 1, k]]$ to return their minimum.

In the indexing setting, any data structure using $O(n/C)$ bits in addition to A must have $\Omega(C)$ query time [11]. A solution matching this lower bound is the one introduced by Fischer & Heun [29] using $\frac{2n}{c(n)} - \Theta(\frac{n \log\log n}{c(n) \log n})$ bits on top of A and solving queries in $O(c(n))$ time, where $c(n) = O(n^\delta)$ for an arbitrary constant $0 < \delta < 1$. Such a solution is based on a novel variant of the *Four-Russians-Trick* that only allows storing the answers for blocks with the same Cartesian tree [75] topology. Most notably, an instantiation of their solution with $c(n) = O(1)$ was the first indexing solution to have an optimal query time while only requiring a linear number of bits of space [29]. Interestingly, they also show how to adapt the string compression scheme of Ferragina and Venturini [27] to obtain the first entropy-bounded solution using $nH_k + O(\frac{n}{\log n}(k \log \sigma + \log\log n))$ bits and $O(1)$ query time.

Most encoding approaches (i.e., the ones that are not allowed to access the indexed array) heavily rely on the Cartesian tree, and in particular, on the well-known sequence of reductions from RMQs on A to LCA queries on its corresponding Cartesian tree and back to $\pm$1RMQs [32, 6] (i.e., an RMQ problem on arrays whose adjacent values change just by ± 1). The information-theoretic lower bound for encoding data structures is $2n - \Theta(\log n)$ bits, and again the first result matching such lower bound (up to lower order terms) while delivering $O(1)$ query time is due to Fischer & Heun [29]. It is based on a novel succinct encoding of the Cartesian tree of A and a sublinear size data structure for $\pm$1RMQs.

Based on such an approach, some practical optimizations were introduced [36]. Most notably, Ferrada & Navarro [23] provided a simpler formulation of RMQs based on *balanced parenthesis* [57] and used a *Range-Min-Max tree* [60] to solve the $\pm$1RMQs, hence obtaining a practical solution that matched the space lower bound and fast query times (although not optimal). This paved the way for practical improvements to that solution. In particular, Baumstark et al. [4] provided the most competitive implementation to date.

Another research direction aimed to reduce the space of encoding solutions even further by leveraging data compression techniques. In particular, Gawrychowski et al. [34] proposed two approaches. The first one combines a grammar-compressed representation of A with a modification of the direct access scheme of Bille et al. [9], thus solving RMQs in $O(\log n)$ time while using a number of memory words proportional to the grammar size. The second one augments a top-tree compressed [8] Cartesian tree $\mathcal{T}$ of A with support for $O(\log n)$ time LCA queries (and thus RMQs) using $O(|\mathcal{T}|)$ words of space. Recently, Munro et al. [56] introduced *hypersuccinct trees*, a compression method for ordinal trees achieving optimal space usage for many tree sources. They apply this novel technique to Cartesian trees achieving constant time RMQs while using $1.736n + o(n)$ bits on average. Lastly, Hamada et al. [38] proposed a practical tree compression technique based on *tree covering* [22] and applied it to Cartesian trees. Their implementation is the only available (encoding) solution having constant query time and using less than $2n$ bits on average.

In our paper, we focus on indexing approaches whose key theoretical bounds are summarized in Table 1 alongside our results. For the experimental analysis, we compare our `FL-RMQ` with variants of the block decomposition and with the sparse table approach. A discussion on how to extend `FL-RMQ` to the encoding setting is deferred to conclusions.

3 Learning Range Minimum Queries

We start by framing the RMQ problem as a supervised learning task in which the *training dataset* D_A is formed by taking as input data the range-pairs (i, j), where $0 \leq i \leq j < n$, and the *target data* are the positions $RMQ_A(i, j)$ of their leftmost minima [2]. Hence we aim at learning a model $f : \mathbb{N}^2 \to \mathbb{N}$ that maps valid ranges of A to positions of their leftmost minima. We note that the size of the training dataset $|D_A|$ poses a computational challenge, as the number of valid ranges scales quadratically with the input size. To overcome this issue, we exploit the same idea of the sparse table solution [5], and hence we consider a subset of the original training set whose range-pairs are only the ones whose size is a power of two, and thus $\tilde{D}_A = \{((i, j), m) \in D_A \mid \exists k : \ j - i + 1 = 2^k\}$. This way, we exponentially reduce the size of the training set from $\Theta(n^2)$ to $\Theta(n \log n)$.

Choosing the model f is a fundamental task in designing learned data structures. Indeed, one must balance the ability of the model to mimic the desired mapping (i.e., its accuracy) with the corresponding inference time and storage requirement (i.e., efficiency in time versus efficiency in space). Experimental results on the performance of learned data structures (see e.g. [45, 76, 53, 44, 71, 28]) suggest that linear models are typically preferable over more complex ones, such as quadratic, cubic, or neural networks, and single-dimensional models should be preferred over multi-dimensional ones.

Following these guidelines, since we are dealing with ranges, we first need a method for converting ranges of the form $[i, j]$ with $i \leq j$ into integers. Ideally, we would like an encoding $e()$ that preserves their natural ordering: namely, given the two ranges $[i, j]$ and $[x, y]$, such that $i \leq x$ and $j \leq y$, we want $e(i, j) \leq e(x, y)$.

We notice that if we plot the points corresponding to valid ranges into the Cartesian plane, as shown in Figure 1a, they form an upper triangular grid. Now, to enumerate ranges and get the encoding function $e()$, we decide to take into account their appearance (from left to right) on the diagonals of such a grid in a bottom-up manner. So, given a range $[i, j]$, we are interested in how to compute $e(i, j)$ efficiently. As already mentioned, since we are using the sparse table approach, the length of $[i, j]$ is always a power of two, i.e. $j - i + 1 = 2^k$ for some positive integer k. Therefore, the point in the Cartesian plane corresponding to $[i, j]$ lies on the k-th diagonal (counting from the bottom), and the number of points on such diagonal is equal to $n - 2^k + 1$. Following this observation, $e(i, j)$ can be calculated by first computing the number of points under the k-th diagonal, which is given by $\sum_{d=0}^{k-1} \left(n - 2^d + 1\right) = k(n + 1) - 2^k + 1$, and then adding the relative position of the point corresponding to $[i, j]$ on the k-th diagonal, which is given by the value of i. Combining these quantities, we get the closed formula for our range encoding:

$$e(i, j) = k(n + 1) - 2^k + (i + 1) \tag{1}$$

As for the efficiency of this computation, we remark that $j - i + 1 = 2^k$, hence we do not need to compute powers of two explicitly. However, extracting the exponent k requires computing the most significant bit of a word that can be done in constant time using a precomputed table [3], or when available, through specialized hardware instructions. In both cases, we emphasize that this computation is fast in practice, and thus it can be neglected.

[2] Notice that our concept of a training dataset slightly differs from traditional machine learning scenarios, as our model is not required to generalize to unseen data; instead, it is designed to handle the same data points present in the training set during queries.

[3] Storing the answers for blocks of size $\log n/2$ requires at most two table look-ups and $O(\sqrt{n} \log \log n)$ bits of space.

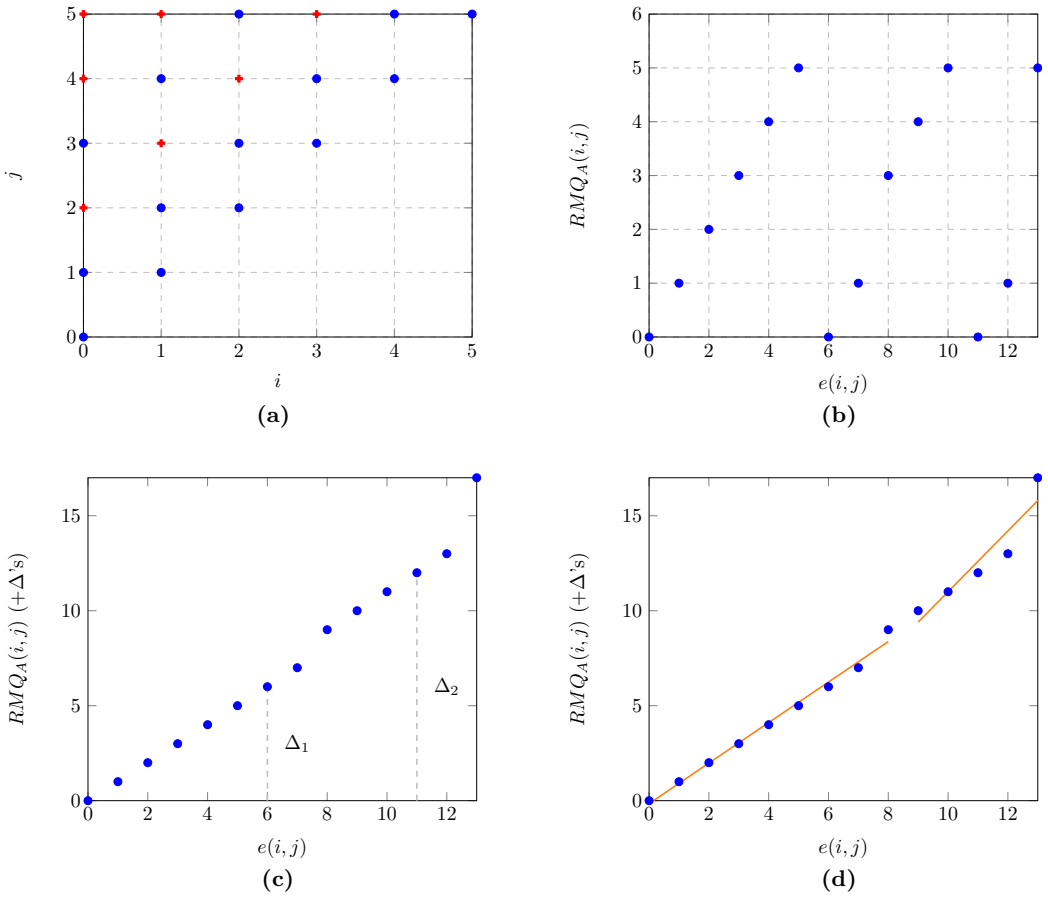

Figure 1 An illustrative example of our approach applied to the array $A = [1, 3, 8, 6, 4, 2]$. Figure 1a shows the grid of 2D-points corresponding to valid ranges of A, red crosses represent ranges whose length is not a power of two, and thus they are not considered in our construction. Figure 1b illustrates the 2D-points corresponding to the encoded ranges of A (x-axis) along with the position of their minimum value (y-axis). Figure 1c depicts the same sequence of 2D-points but, now, with added correction terms to ensure monotonicity. Figure 1d shows a PLA of the monotone sequence with maximum error $\varepsilon = 2$.

The second ingredient of our proposal is an implementation of f based on a *Piecewise-Linear Approximation* (PLA), with maximum error ε, of the sequence of all the encoded ranges and the positions of their minimum value, i.e. $\{(e(i,j), m) \,|\, ((i,j), m) \in \tilde{D}_A\}$. A PLA consists of a sequence of segments each of them represented as a triple $s_z = (r_z, \alpha_z, \beta_z)$, where α_z is the slope, β_z is the intercept, and r_z is the abscissa of the point that started such segment. The segments computed by a PLA are such that every point in the sequence has a vertical distance of at most ε from one of these segments. We highlight that such an ε-approximation of an increasing sequence of 2D-points can always be computed in linear time and optimally, hence using the minimum number of segments [62], as done in several learning-based data structures to date (see e.g. [45, 33, 28] and many others nowadays).

However, deploying the *Piecewise-Linear Approximation* (PLA) in our setting is not immediate. In fact, in general, our sequence of 2D-points is not monotone, see the example of Figure 1b. Although this property is not mandatory for computing a PLA, it is preferable because it makes the data more easily approximated by a smaller number of segments.

Moreover, having an increasing sequence allows us to quickly determine which segment is responsible for a given encoded range (see Section 3.2). Therefore, we make our sequence monotone by following the ordering given by the encoding of Equation 1: this way, ranges corresponding to points lying on the same diagonal have the positions of their minima sorted in increasing order. However, this ordering can be broken when considering the first point lying on the next diagonal, see Figure 1b as an example. To avoid this issue, we add a correction term Δ whenever this ordering is not followed, having care of adding such a correction to all the subsequent positions, as shown in Figure 1c.

In summary, our implementation of f consists of only two components: the optimal number of segments that ε-approximate the position of the minimum of ranges of A whose length is a power of two, properly shifted by corrections that ensure their (increasing) monotonicity; the sequence of those (integer) corrections that allow to scale back a prediction into its original valid position in A. In the subsequent sections, we explore the novel space-time trade-offs given by this representation and evaluate its practical performance. Due to its algorithmic features, we call our solution *Fully-Learned RMQ*, hence shortly `FL-RMQ`.

3.1 Space Usage and Construction

To compute the space taken by the segments and the correction terms, we observe that the Cartesian plane representation of our monotone sequence has both the abscissae and ordinates upper bounded by $n \log n$, because we have at most n points per diagonal and $\log n$ diagonals (see Figure 1c). Therefore, assuming that the PLA outputs ℓ segments of the form $s_z = (r_z, \alpha_z, \beta_z)$, where we recall that α_z and β_z are respectively the slope and intercept of the segment, whereas r_z is the abscissa of the x-axis covered by the segment. Now, the values r_z and β_z are integers that can be represented in $\log(n \log n) = \log n + \log \log n$ bits [10]; while the slope α_z is encoded as a rational number with a numerator and denominator also of $\log n + \log \log n$ bits each [10]. Considering the correction terms, their value is upper bounded by n, and since there are $O(\log n)$ diagonals, they account for $O(\log^2 n)$ bits of extra space. Summing up these quantities gives our first result:

▶ **Theorem 1.** *A plain implementation of our `FL-RMQ` data structure takes* $4\ell(\log n + \log \log n) + O(\log^2 n)$ *bits of space.*

For what concerns the construction, we avoid storing the entire sparse table in memory by slightly modifying the standard dynamic programming approach used to build it incrementally. In particular, we observe that the content of a generic entry is fully determined by two values lying on the previous column. We therefore propose a more space-friendly solution by only keeping two columns, the current one and the previous one, and alternating between them from one iteration to another. This allows us to avoid keeping the whole sequence of points in memory, and by performing constant time streaming insertions inside the PLA [62], the construction space is reduced from $\Theta((n \log n + \ell) \log n)$ to $\Theta((n + \ell) \log n)$ bits, shaving off a logarithmic factor from the input size, while maintaining the same runtime.

▶ **Theorem 2.** *The `FL-RMQ` data structure can be built in* $\Theta(n \log n)$ *time while using* $\Theta((n + \ell) \log n)$ *bits of working space.*

3.2 Supporting Queries

The key novelty of our proposal is to replace the costly sparse table with lightweight linear models approximating the position of the minimum for ranges whose size is a power of two. Following this approach, solving an RMQ on a range $[i, j]$, boils down to solving two other

queries for the ranges $[i, i + 2^k - 1]$ and $[j - 2^k + 1, j]$, where k is the largest power of two such that $2^k \leq j - i + 1$. Let us consider the range $[i, i + 2^k - 1]$, as the same procedure applies to the other. We use a predecessor data structure, say $pred_R$, built on the set $R = \{r_z \mid 0 \leq z < \ell\}$, and proceed in four steps. First, we use Equation 1 to convert the range $[i, i + 2^k - 1]$ into an integer $x = e(i, i + 2^k - 1)$. Second, we use the predecessor data structure on R to retrieve the index $z = pred_R(x)$ of the segment covering the encoding x. Third, we compute the approximate position $p = \lfloor \alpha_z \cdot (x - r_z) + \beta_z \rfloor - \Delta_k$ of the minimum inside the given range (notice that we scale its position into a valid range by subtracting the correction term Δ_k). Fourth, we scan the range $[\max\{p - \varepsilon, i\}, \min\{p + \varepsilon, i + 2^k - 1\}]$ of A that is ensured by construction to contain the answer to $RMQ_A(i, i + 2^k - 1)$. We repeat this four-step process to solve the second query $RMQ_A(j, j - 2^k + 1)$, so that the answer to the original query $RMQ_A(i, j)$ is computed by simply comparing the two results. The last two steps overall take $O(\varepsilon)$ time, and treating the predecessor data structure as a black box yields the following result:

▶ **Theorem 3.** *The `FL-RMQ` data structure supports RMQs in $t + O(\varepsilon)$ time and takes $b + 4\ell(\log n + \log\log n) + O(\log^2 n)$ bits of space, where t and b are respectively the time and space of a predecessor data structure built on a set of ℓ positive integers bounded by $n \log n$.*

For instance, if the predecessor data structure is represented using the modified *Van Emde Boas* tree of Mehlhorn & Näher [55], since the universe of R is bounded by $n \log n$, we achieve $O(\log\log n + \varepsilon)$ query time using $O(\ell)$ additional words of space. To get rid of the dependency from the input size, another possibility is to use the characteristic bit vector of R augmented with a data structure supporting constant time predecessor queries (i.e., *rank* operations) in sublinear extra space [59], then we achieve $O(\varepsilon)$ query time using $n \log n + o(n \log n)$ bits, i.e. a $\Theta(\log n)$ improvement compared to the classical sparse table. Aiming for a more significant space saving, we can drop the predecessor data structure resorting to a classical binary search to find the segment responsible for a given encoded range.

▶ **Corollary 4.** *The `FL-RMQ` data structure supports RMQs in $O(\log \ell + \varepsilon)$ time using $4\ell(\log n + \log\log n) + O(\log^2 n)$ bits of space.*

In practice, we propose the following optimizations to speed up queries. First, since we are searching for the segments covering $e(i, i + 2^k - 1)$ and $e(j - 2^k + 1, j)$, we know that they lie on the k-th diagonal since their length is exactly 2^k. By storing a vector C, such that $C[s]$ is the index of the first segment covering the encoded ranges of the s-th diagonal, we can narrow the two binary searches to the interval $[C[k], C[k + 1]]$ [4], which is hopefully much smaller than $[0, \ell)$. Second, because of our encoding, it holds that $e(i, i + 2^k - 1) < e(j - 2^k + 1, j)$, hence the scope of the second binary search can be further reduced by starting from the position of the first segment found. Moreover, since from $[i, i + 2^k - 1]$ and $[j - 2^k + 1, j]$ there are exactly 2^k other ranges, if k is relatively small, we can adaptively switch to an exponential search for the second range because we can expect to find the corresponding segment in the proximity of the first one.

[4] Notice that the right endpoint is included because the first segment covering the $(k + 1)$-th diagonal may also encompass certain encoded ranges of the k-th diagonal.

3.3 Compressing the Data Structure

Compressing our data structure boils down to providing proper lossless compressors for the segments that constitute the most space-consuming component of our proposal.

Our key algorithmic idea is to modify the segment representation by omitting the slopes and instead include additional information that can be succinctly encoded, allowing us to recover the slopes in constant time. Technically speaking, we encode a generic segment as a tuple $s_z = (r_z, b_z, e_z, \beta_z, \gamma_z)$ where the new values b_z and e_z are respectively the integer-representation of the ordinates of the first and last covered encoded range (i.e., point in the Cartesian plane), while γ_z is the integer-representation of the lastly covered ordinate by s_z. This allows us to recover the slope α_z in constant time by using the equation of a line passing through the points (r_z, β_z) and $(r_{z+1} - 1, \gamma_z)$, namely $\alpha_z = (\gamma_z - \beta_z)/(r_{z+1} - r_z - 1)$ [5].

A potential drawback of this approach is that representing β_z and γ_z from floating-point numbers to integers, as assumed above, could have an unpredictable impact on the approximation error incurred by s_z. However, it is possible to show that this conversion only increases the error ε by an additive constant equal to 3. As a result, the theoretical guarantees on query time discussed in Section 3.2 remain asymptotically unchanged. Due to space constraints, we only sketch the main idea of our proof, deferring its details to the journal version of this paper. Consider a generic range $[i, j]$ whose length is a power of two, let $x = e(i, j)$ be its encoding, and z be the segment's index covering such encoded range. For the sake of presentation, let us denote with $f_z(x)$ the approximate position of the minimum for $[i, j]$ given by the z-th segment and using all integer-based parameters, i.e. $f_z(x) = \lfloor (x - r_z)\alpha_z \rfloor + \beta_z$. The key idea of our proof is to focus on the slope α_z and bound the distance from its original value, let this be $\hat{\alpha_z}$, derived from the floating-point representation of β_z and γ_z. With simple arithmetic manipulations, we can show that $|\alpha_z - \hat{\alpha_z}| \leq 1/(x - r_z)$. Next, we consider the approximation error of the z-th segment when predicting the position of the minimum inside $[i, j]$ by using the integer-based calculation of α_z, instead of the real-based calculation of $\hat{\alpha_z}$, and exploiting the previous bound, we show that $|RMQ_A(i, j) - f_z(x)| \leq \varepsilon + 3$. As anticipated, this additive term does not affect the asymptotic time complexity of RMQs discussed in the previous section.

Now, we notice that the r_zs, b_zs, and e_zs form an increasing sequence of ℓ positive integers bounded by $n \log n$. Therefore, using the Elias-Fano representation [59], we reduce the space of the three sequences to $6\ell + 3\ell \log \frac{n \log n}{\ell} + o(\ell) = 3\ell(\log \frac{n}{\ell} + \log \log n + 2 + o(1))$ bits. Next, we notice that the differences in absolute value between b_z and β_z, and between e_z and γ_z are bounded by $2\varepsilon + 1$. Hence we only encode the gaps between these values, reducing the space for the β_zs and γ_zs to $\ell \log(2\varepsilon + 1)$. Summing the $O(\log^2 n)$ bits to store the corrections (recall Section 3), gives the following space improvement to Theorem 3:

▶ **Theorem 5.** *There exists a compressed version of the* `FL-RMQ` *data structure that takes* $b + 3\ell(\log \frac{n}{\ell} + \log \log n + 2 + o(1)) + 2\ell \log(2\varepsilon + 1) + O(\log^2 n)$ *bits of space.*

A further space reduction can be obtained through a recently introduced optimization for PLAs [1] that modifies the original O'Rourke algorithm so that the last covered abscissa of each segment is the first covered by the next one. This may come at the cost of having a larger number ℓ' of segments that are anyway shown, in practice, to be very close to the optimal one, namely ℓ. The result is twofold: it ensures monotonicity of the β_zs and γ_zs

[5] Notice that the endpoint of the z-th segment is $(r_{z+1} - 1, \gamma_z)$ as the abscissa of the encoded ranges form consecutive integers.

while limiting their absolute difference to $4\varepsilon + 1$. Hence, we can drop the b_zs and e_zs from our previous representation, and store the Elias-Fano encoded sequences r_zs and β_zs, and the gaps between each γ_z and β_z, giving the following space improvement over Theorem 3:

▶ **Corollary 6.** *There exists a compressed version of the* ***FL-RMQ*** *data structure that occupies* $b + 2\ell'(\log \frac{n}{\ell'} + \log\log n + 2 + o(1)) + \ell' \log(4\varepsilon + 1) + O(\log^2 n)$ *bits of space.*

We conclude by mentioning that we could also reduce the space of Theorem 3 via a specialized lossless compressor for the α_zs [28]. Such a compressor is highly beneficial in practice when multiple segments share the same or similar slope, but a theoretical characterization of its space reduction is unknown.

3.4 Bounding the Number of Segments on Uniform Distributions

Expressing the number of segments ℓ produced by the PLA as a function of the input size n and the maximum error ε is crucial for theoretical comparisons with other data structures. Ferragina et al. [25] explored this relationship in the indexing scenario, showing that for independent and identically distributed gaps between consecutive keys, with constant mean and variance, $\ell = \Theta(n/\varepsilon^2)$ with high probability. While this independence assumption does not apply in our framework due to dependent vertical gaps between consecutive encoded ranges, we can still prove a similar bound when the elements of A are uniformly distributed.

Due to space constraints, we provide a brief outline of the main idea and result here, deferring its details to the journal version of this paper. The key idea underlying our proof is to analyze the Cartesian plane representation of our monotone sequence, showing that while the maximum ordinate value is $O(n \log n)$ in the worst-case, it averages to $\Theta(n)$. Intuitively this is shown by computing, for each k between 0 and $\log n$, the average contribution of each range whose length is 2^k to the overall ordinate value. Consequently, the average number of horizontal segments (i.e., having a fixed slope equal to 0) required to ε-approximate the input sequence can be proved to be $\Theta(n/\varepsilon)$. This represents a significant reduction, as the number of such segments in the worst-case is $O((n \log n)/\varepsilon)$, and, most importantly, it provides an obvious upper bound for the number of segments produced by the optimal PLA, which can use segments of arbitrary slope.

▶ **Theorem 7.** *Given an array A of n elements drawn from a totally ordered universe uniformly and independently at random, the number of segments ℓ forming our* ***FL-RMQ*** *data structure is $O(n/\varepsilon)$ on average.*

▶ **Corollary 8.** *On uniformly distributed inputs the* ***FL-RMQ*** *data structure solves queries in both expected $O(\log \frac{n}{\varepsilon} + \varepsilon)$ time and $O(\frac{n}{\varepsilon} \log n + \log^2 n)$ bits of space.*

An immediate consequence of Corollary 8 is that it allows us to compare our `FL-RMQ` data structure with other indexing solutions on uniformly distributed inputs. In particular, fixing the same query time of a block decomposition with blocks of size b (i.e, $\varepsilon = n/b$), `FL-RMQ` is on average more space efficient when $b < \sqrt{n}$. Setting $\varepsilon = \log n$, `FL-RMQ` uses on average less space than a sparse table and asymptotically the same space as the optimal solution of Fischer & Heun, at the cost of a logarithmic slowdown at query time.

4 Experimental Results

We run our experiments on an Ubuntu machine with 3024 GiB of internal memory and an Intel Xeon Platinum 8260M CPU. Our `FL-RMQ` is implemented in `C++`, and RMQs are answered as in Corollary 4 on top of the uncompressed data structure of Theorem 1.

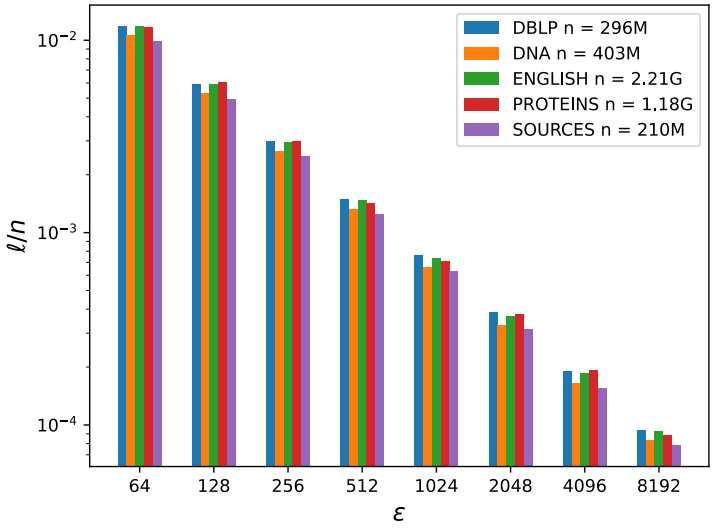

Figure 2 The ratio between the number of segments ℓ and the size n of LCP arrays computed from the Pizza & Chili collection at different values of the error ε.

To assess the performance of `FL-RMQ`, we experiment on both artificial and real-world datasets. Following the methodology of Ferrada & Navarro [23], we generate three artificial datasets of $n = 10^9$ integers. (1) `RAND`: a sequence of uniformly distributed integers in the range $[1, n]$. (2) `INC-`δ: a pseudo-increasing sequence whose i-th element is chosen uniformly at random in the range $[i - \delta, i + \delta]$. (3) `DEC-`δ: analogous to `INC-`δ, but the i-th element is uniformly chosen in the range $[n - i - \delta, n - i + \delta]$. For the latter two datasets, Ferrada & Navarro use $\delta \in \{0, 10^2, 10^4\}$. However, small values of δ make the resulting sequences nearly sorted, giving an undue advantage to our `FL-RMQ`. Therefore we fairly consider the harder instances, namely that with $\delta = 10^4$. For what concerns real-world datasets, we use the *Longest Common Prefix* (LCP) array of texts coming from the *Pizza & Chili* corpus [26].

We start by assessing the effectiveness of our approach by measuring the average number of elements covered by a segment. Using real-world datasets, we compute the ratio between the number of segments ℓ and the size n of the LCP arrays. Figure 2 shows that, even for small values of ε, the number of segments is from 2 to 4 orders of magnitude less than n, proving once again the compression capability of the PLA also in this novel setting of RMQs.

We then focus on evaluating the query time and space usage of our `FL-RMQ` data structure, comparing it with the following baselines: a block decomposition with blocks of progressively larger sizes, specifically $b \in \{\log n, \sqrt[4]{n}, \sqrt[3]{n}, \sqrt{n}\}$, and the classical sparse table approach. We avoided both the entropy compressed and the optimal solution of Fischer & Heun surveyed in Section 2 because there are no available implementations to the best of our knowledge. In this experimental evaluation, we include all artificial datasets but limit the real-world datasets to only the largest one, since Figure 2 shows that their distributions are very similar. Lastly, over all experiments, for each range of size $\{10^1, 10^2, \ldots, 10^7, 10^8\}$, we generate 10^4 queries by randomly choosing the starting point for each of them. In the following, we measure the average query time in *nanoseconds* and the additional space (i.e. the extra space on top of the input array) in *bits per element* (bpe). Because of the page limit, we only report the most significant sample of each experiment in Figure 3, please refer to Appendix 6 for the complete results.

We start by commenting on the performances of the sparse table (orange bullet) and the block decomposition approaches (blue squares), as they do not depend on the underlying data distribution. As expected, the sparse table approach is the fastest on all datasets, as it only requires four memory accesses and a few arithmetic operations independently of the query size. However, we can notice that it is also the most space-consuming, requiring between 960 (`RAND`, `INC-`δ, and `DEC-`δ) and 1024 (`ENGLISH`) bpe. Moving to the block decompositions, we observe that for the smallest range size (10^1) they have the same query time, as they all resort to a simple linear scan when solving queries. The decomposition with block size $\sqrt{n}$, as expected, guarantees the smallest space occupancy but has the slowest query times for mid-sized ranges (10^3 and 10^4). However, for larger ranges ($10^6 - 10^8$), it becomes more efficient among the block decompositions and eventually ranks as the second fastest and even the fastest among them. Switching to blocks of size $\log n$ (lightest blue squares), we observe an almost symmetrical behavior. This configuration is the most space-consuming but delivers the fastest query times for small ranges. However, its performance progressively degrades as the range size increases. Lastly, we observe that blocks of size $\sqrt[4]{n}$ and $\sqrt[3]{n}$ give a trade-off between the previously discussed behaviors. Specifically, $\sqrt[4]{n}$ performs well on mid-size ranges ($10^2 - 10^4$), while $\sqrt[3]{n}$ offers better performance on larger ones ($10^5 - 10^8$).

Turning to our solution, we observe that it behaves similarly on both `ENGLISH` and `RAND`. Hence we only comment on the latter dataset as the same considerations apply to the other. The results are shown in Figure 3 and detailed in Figure 4 and 5 of Appendix 6. For small ranges (10^1 and 10^2), `FL-RMQ` performs comparably to other approaches across all values of ε, with the sparse table being only slightly faster but much more space consuming. For mid-size ranges (i.e., $10^3 - 10^5$), `FL-RMQ` delivers a smooth space-time trade-off that is competitive with the block decompositions. Notably, the one using $\varepsilon = 512$ is very close to the best performances of the $\sqrt[3]{n}$-decomposition. Lastly, for large ranges ($10^6 - 10^8$), our `FL-RMQ` excels by providing the best performance along the Pareto frontier. In particular, the query time is very competitive, varying from hundreds ($\varepsilon = 64$) to thousands ($\varepsilon = 2048$) of nanoseconds, while requiring between 2.06 and 0.06 bpe.

Switching to the results obtained on `INC-`δ shown in Figure 3, and detailed in Figure 6 of Appendix 6, we first observe that for small ranges ($10^1 - 10^4$), our `FL-RMQ` performs similarly to the various block decompositions in terms of query time, across all values of ε. However, for larger ranges ($10^5 - 10^8$), `FL-RMQ` significantly improves on the block decomposition, with the implementation using $\varepsilon = 512$ providing the fastest query time. Considering the space usage, we observe an interesting pattern: for values of ε between 64 and 256, `FL-RMQ` uses more space than any of the block decompositions. This suddenly changes for larger values of ε (between 512 and 2048), where `FL-RMQ` uses 2 orders of magnitude less space than any block decomposition. On `DEC-`δ, we observe a very similar behavior, and the same comments apply (see Figure 7 of Appendix 6). The latter results are particularly interesting, as they emphasize the main strength of `FL-RMQ`: its ability to leverage regularities in the underlying data distribution to achieve novel and competitive space-time trade-offs.

As a final experiment, motivated by the lack of an absolute winner among all the implementations considered, we designed and implemented a `hybrid` approach that combines `FL-RMQ` with a block decomposition. For each dataset, we observe a threshold in the query size where `FL-RMQ` begins to outperform the block decompositions. Based on this, we select the two best-performing configurations and build our `FL-RMQ` only for ranges larger than this threshold, while resorting to the block decomposition for smaller ones. This `hybrid` solution is represented with purple stars in Figure 3, and in the detailed experiments reported in Figures 4 – 7 of Appendix 6. Starting from `RAND` and `ENGLISH`, based on the previous

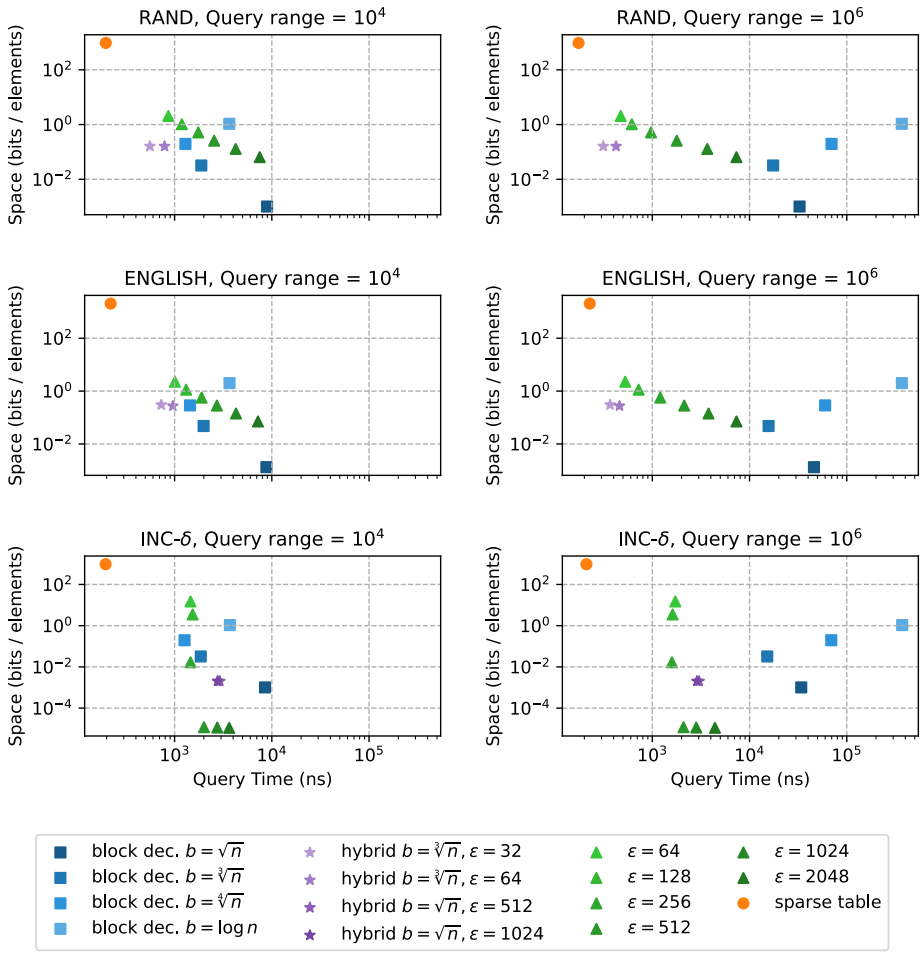

Figure 3 Most significant sample of the RMQs space-time performances on the `RAND`, `ENGLISH`, and `INC-`δ ($\delta = 10^4$) datasets.

experiments, we set the threshold to 10^4 and choose blocks of size $\sqrt[3]{n}$ varying ε between 32 and 64. Figure 3 shows that this combination has good performances on small ranges ($10^1 - 10^3$) while outperforming all the other implementations on larger ones (from 10^4 on). However, for `INC-`δ and `DEC-`δ (Figure 3 and Figures 6 – 7 of Appendix 6), the best implementations of `FL-RMQ` (i.e. the ones using $\varepsilon \in \{512, 1024\}$) are already orders of magnitudes smaller than any block decomposition, with a better or very similar query time, therefore the `hybrid` approach is not convenient in these cases.

5 Conclusions and Future Work

We introduced a novel approach to the RMQ problem by establishing a connection between the input data and the geometry of a properly defined set of points in the Cartesian plane. Building on this insight, we designed a data structure whose performance adapts to the underlying data distribution, provided theoretical guarantees for its efficiency, and experimentally demonstrated its practicality against existing approaches.

While our proposal still requires further development, we suggest several future directions: first, our `FL-RMQ` can be extended to solve the $\pm$1RMQs arising in the encoding setting [29, 36, 23, 4], thus obtaining an encoding data structure with possibly lower order space usage terms and more practical query times (further details in the journal version of this paper); second, extend the use of linear functions for the PLA to non-linear functions [37], which possibly reduce the value of ℓ; third, extend the theoretical bound on the number of segments ℓ produced by the PLA to other input distributions, which could provide deeper insights into the performance of our data structure across various scenarios. Finally, significant opportunities remain for algorithmic engineering, such as utilizing specialized data layouts to enhance cache efficiency in binary searches [43], more space/time efficient solutions for predecessor searches, or applying vectorized instructions to streamline minimum computation over the short ranges $[p - \varepsilon, p + \varepsilon]$ on which the minimum search is restricted [70].

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 Amihood Amir, Gad M. Landau, and Uzi Vishkin. Efficient pattern matching with scaling. *J. Algorithms*, 13(1):2–32, 1992. doi:10.1016/0196-6774(92)90003-U.

3 Alberto Apostolico, Maxime Crochemore, Martin Farach-Colton, Zvi Galil, and S. Muthukrishnan. 40 years of suffix trees. *Commun. ACM*, 59(4):66–73, 2016. doi:10.1145/2810036.

4 Niklas Baumstark, Simon Gog, Tobias Heuer, and Julian Labeit. Practical range minimum queries revisited. In *16th International Symposium on Experimental Algorithms, SEA 2017, June 21-23, 2017, London, UK*, volume 75 of *LIPIcs*, pages 12:1–12:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.SEA.2017.12.

5 Michael A Bender and Martin Farach-Colton. The lca problem revisited. In *LATIN 2000: Theoretical Informatics: 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000 Proceedings 4*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

6 Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM Journal on Computing*, 22(2):221–242, 1993. doi:10.1137/0222017.

7 Michele Bevilacqua, Giuseppe Ottaviano, Patrick S. H. Lewis, Scott Yih, Sebastian Riedel, and Fabio Petroni. Autoregressive search engines: Generating substrings as document identifiers. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL: http://papers.nips.cc/paper_files/paper/2022/hash/cd88d62a2063fdaf7ce6f9068fb15dcd-Abstract-Conference.html.

8 Philip Bille, Inge Li Gørtz, Gad M. Landau, and Oren Weimann. Tree compression with top trees. *Inf. Comput.*, 243:166–177, 2015. doi:10.1016/J.IC.2014.12.012.

9 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comput.*, 44(3):513–539, 2015. doi:10.1137/130936889.

10 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.

11 Gerth Stølting Brodal, Pooya Davoodi, and S. Srinivasa Rao. On space efficient two dimensional range minimum data structures. *Algorithmica*, 63(4):815–830, 2012. doi:10.1007/S00453-011-9499-0.

12 Ivan Carvalho and Ramon Lawrence. Learnedsort as a learning-augmented samplesort: Analysis and parallelization. In *Proceedings of the 35th International Conference on Scientific and*

Statistical Database Management, SSDBM 2023, Los Angeles, CA, USA, July 10-12, 2023, pages 2:1–2:9. ACM, 2023. doi:10.1145/3603719.3603731.

13 Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, 2008. doi:10.1145/1365815.1365816.

14 Gang Chen, Simon J. Puglisi, and William F. Smyth. Lempel-ziv factorization using less time & space. *Math. Comput. Sci.*, 1(4):605–623, 2008. doi:10.1007/S11786-007-0024-4.

15 Supawit Chockchowwat, Wenjie Liu, and Yongjoo Park. Airindex: Versatile index tuning through data and storage. *Proc. ACM Manag. Data*, 1(3):204:1–204:26, 2023. doi:10.1145/3617308.

16 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005. doi:10.1016/J.JALGOR.2003.12.001.

17 Marco Costa, Paolo Ferragina, and Giorgio Vinciguerra. Grafite: Taming adversarial queries with optimal range filters. *Proc. ACM Manag. Data*, 2(1):3:1–3:23, 2024. doi:10.1145/3639258.

18 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, M. Sohel Rahman, German Tischler, and Tomasz Walen. Improved algorithms for the range next value problem and applications. *Theor. Comput. Sci.*, 434:23–34, 2012. doi:10.1016/J.TCS.2012.02.015.

19 Zhenwei Dai and Anshumali Shrivastava. Adaptive learned bloom filter (ada-bf): Efficient utilization of the classifier with application to real-time information filtering on the web. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/86b94dae7c6517ec1ac767fd2c136580-Abstract.html.

20 Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David B. Lomet, and Tim Kraska. ALEX: an updatable adaptive learned index. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 969–984. ACM, 2020. doi:10.1145/3318464.3389711.

21 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.

22 Arash Farzan and J. Ian Munro. A uniform paradigm to succinctly encode various families of trees. *Algorithmica*, 68(1):16–40, 2014. doi:10.1007/S00453-012-9664-0.

23 Héctor Ferrada and Gonzalo Navarro. Improved range minimum queries. *J. Discrete Algorithms*, 43:72–80, 2017. doi:10.1016/J.JDA.2016.09.002.

24 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.

25 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.

26 Paolo Ferragina and Gonzalo Navarro. Pizza and chili, 2005. Accessed on November 6, 2024. URL: https://pizzachili.dcc.uchile.cl/.

27 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.

28 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.

29 Johannes Fischer and Volker Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011. doi:10.1137/090779759.

30 Johannes Fischer, Volker Heun, and Stefan Kramer. Optimal string mining under frequency constraints. In *Knowledge Discovery in Databases: PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4213 of *Lecture Notes in Computer Science*, pages 139–150. Springer, 2006. doi:10.1007/11871637_17.

31 Johannes Fischer, Veli Mäkinen, and Gonzalo Navarro. Faster entropy-bounded compressed suffix trees. *Theor. Comput. Sci.*, 410(51):5354–5364, 2009. doi:10.1016/J.TCS.2009.09.012.

32 Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 135–143, New York, NY, USA, 1984. Association for Computing Machinery. doi:10.1145/800057.808675.

33 Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. Fiting-tree: A data-aware index structure. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1189–1206. ACM, 2019. doi:10.1145/3299869.3319860.

34 Pawel Gawrychowski, Seungbum Jo, Shay Mozes, and Oren Weimann. Compressed range minimum queries. *Theor. Comput. Sci.*, 812:39–48, 2020. doi:10.1016/J.TCS.2019.07.002.

35 Simon Gog, Giulio Ermanno Pibiri, and Rossano Venturini. Efficient and effective query auto-completion. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 2271–2280. ACM, 2020. doi:10.1145/3397271.3401432.

36 Roberto Grossi and Giuseppe Ottaviano. Design of practical succinct data structures for large data collections. In *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*, volume 7933 of *Lecture Notes in Computer Science*, pages 5–17. Springer, 2013. doi:10.1007/978-3-642-38527-8_3.

37 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.

38 Kou Hamada, Sankardeep Chakraborty, Seungbum Jo, Takuto Koriyama, Kunihiko Sadakane, and Srinivasa Rao Satti. A simple representation of tree covering utilizing balanced parentheses and efficient implementation of average-case optimal rmqs. In *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 64:1–64:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.64.

39 Wing-Kai Hon, Rahul Shah, and Jeffrey Scott Vitter. Space-efficient framework for top-k string retrieval problems. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 713–722. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.19.

40 Bo-June Paul Hsu and Giuseppe Ottaviano. Space-efficient data structures for top-k completion. In *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 583–594. International World Wide Web Conferences Steering Committee / ACM, 2013. doi:10.1145/2488388.2488440.

41 Samuel J. Huston, Alistair Moffat, and W. Bruce Croft. Efficient indexing of repeated n-grams. In *Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011*, pages 127–136. ACM, 2011. doi:10.1145/1935826.1935857.

42 Young Mo Kang, Wenhao Liu, and Yingbo Zhou. Queryblazer: Efficient query autocompletion framework. In *WSDM '21, The Fourteenth ACM International Conference on Web Search*

and Data Mining, Virtual Event, Israel, March 8-12, 2021, pages 1020–1028. ACM, 2021. doi:10.1145/3437963.3441725.

43 Paul-Virak Khuong and Pat Morin. Array layouts for comparison-based searching. *ACM J. Exp. Algorithmics*, 22, 2017. doi:10.1145/3053370.

44 Andreas Kipf, Ryan Marcus, Alexander van Renen, Mihail Stoian, Alfons Kemper, Tim Kraska, and Thomas Neumann. SOSD: A benchmark for learned indexes. *CoRR*, abs/1911.13014, 2019. arXiv:1911.13014.

45 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.

46 Ani Kristo, Kapil Vaidya, Ugur Çetintemel, Sanchit Misra, and Tim Kraska. The case for a learned sorting algorithm. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1001–1016. ACM, 2020. doi:10.1145/3318464.3389752.

47 Ani Kristo, Kapil Vaidya, and Tim Kraska. Defeating duplicates: A re-design of the learnedsort algorithm. *CoRR*, abs/2107.03290, 2021. arXiv:2107.03290.

48 Pengfei Li, Hua Lu, Rong Zhu, Bolin Ding, Long Yang, and Gang Pan. DILI: A distribution-driven learned index. *Proc. VLDB Endow.*, 16(9):2212–2224, 2023. doi:10.14778/3598581.3598593.

49 Zsuzsanna Lipták, Francesco Masillo, and Gonzalo Navarro. BAT-LZ out of hell. In *35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024, June 25-27, 2024, Fukuoka, Japan*, volume 296 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CPM.2024.21.

50 Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. *CoRR*, abs/2401.17377, 2024. doi:10.48550/arXiv.2401.17377.

51 Siqiang Luo, Subarna Chatterjee, Rafael Ketsetsidis, Niv Dayan, Wilson Qin, and Stratos Idreos. Rosetta: A robust space-time optimized range filter for key-value stores. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 2071–2086. ACM, 2020. doi:10.1145/3318464.3389731.

52 Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing*. Cambridge University Press, 2015. doi:10.1017/CBO9781139940023.

53 Ryan Marcus, Andreas Kipf, Alexander van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. Benchmarking learned indexes. *Proc. VLDB Endow.*, 14(1):1–13, 2020. doi:10.14778/3421424.3421425.

54 Ruslan Mavlyutov, Marcin Wylot, and Philippe Cudré-Mauroux. A comparison of data structures to manage uris on the web of data. In *The Semantic Web. Latest Advances and New Domains - 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 - June 4, 2015. Proceedings*, volume 9088 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 2015. doi:10.1007/978-3-319-18818-8_9.

55 Kurt Mehlhorn and Stefan Näher. Bounded ordered dictionaries in o(log log N) time and o(n) space. *Inf. Process. Lett.*, 35(4):183–189, 1990. doi:10.1016/0020-0190(90)90022-P.

56 J. Ian Munro, Patrick K. Nicholson, Louisa Seelbach Benkner, and Sebastian Wild. Hypersuccinct trees - new universal tree source codes for optimal compressed tree data structures and range minima. In *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 70:1–70:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.70.

57 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM J. Comput.*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

58 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.

59 Gonzalo Navarro. *Compact Data Structures - A Practical Approach*. Cambridge University Press, 2016. doi:10.1017/CBO9781316588284.

60 Gonzalo Navarro and Kunihiko Sadakane. Fully functional static and dynamic succinct trees. *ACM Trans. Algorithms*, 10(3):16:1–16:39, 2014. doi:10.1145/2601073.

61 Daisuke Okanohara and Kunihiko Sadakane. An online algorithm for finding the longest previous factors. In *Algorithms - ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, September 15-17, 2008. Proceedings*, volume 5193 of *Lecture Notes in Computer Science*, pages 696–707. Springer, 2008. doi:10.1007/978-3-540-87744-8_58.

62 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.

63 Sachith Pai, Michael Mathioudakis, and Yanhao Wang. Wazi: A learned and workload-aware z-index. In *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*, pages 559–571, 2024. doi:10.48786/EDBT.2024.48.

64 Yongxin Peng, Wei Zhou, Lin Zhang, and Hongle Du. A study of learned KD tree based on learned index. In *International Conference on Networking and Network Applications, NaNA 2020, Haikou City, China, December 10-13, 2020*, pages 355–360. IEEE, 2020. doi:10.1109/NANA51271.2020.00067.

65 Giulio Ermanno Pibiri and Rossano Venturini. Efficient data structures for massive N-gram datasets. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 615–624. ACM, 2017. doi:10.1145/3077136.3080798.

66 Jack W. Rae, Sergey Bartunov, and Timothy P. Lillicrap. Meta-learning neural bloom filters. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5271–5280. PMLR, 2019. URL: http://proceedings.mlr.press/v97/rae19a.html.

67 Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theor. Comp. Sys.*, 41(4):589–607, December 2007. doi:10.1007/s00224-006-1198-x.

68 Atsuki Sato and Yusuke Matsui. Fast partitioned learned bloom filter. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/7b2e844c52349134268e819a9b56b9e8-Abstract-Conference.html.

69 Atsuki Sato and Yusuke Matsui. PCF learned sort: a learning augmented sort algorithm with o(n log log n) expected complexity. *CoRR*, abs/2405.07122, 2024. doi:10.48550/arXiv.2405.07122.

70 Sergey Slotin. Algorithms for modern hardware, 2021. Accessed on October 18, 2024. URL: https://en.algorithmica.org/hpc/algorithms/argmin/.

71 Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. Learned index: A comprehensive experimental evaluation. *Proc. VLDB Endow.*, 16(8):1992–2004, 2023. doi:10.14778/3594512.3594528.

72 Chuzhe Tang, Youyun Wang, Zhiyuan Dong, Gansen Hu, Zhaoguo Wang, Minjie Wang, and Haibo Chen. Xindex: a scalable learned index for multicore data storage. In *PPoPP '20: 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, California, USA, February 22-26, 2020*, pages 308–320. ACM, 2020. doi:10.1145/3332466.3374547.

73 Kapil Vaidya, Eric Knorr, Michael Mitzenmacher, and Tim Kraska. Partitioned learned bloom filters. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021. URL: https://openreview.net/forum?id=6BRLOfrMhW.

74 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.

75 Jean Vuillemin. A unifying look at data structures. *Communications of the ACM*, 23(4):229–239, 1980. doi:10.1145/358841.358852.

76 Youyun Wang, Chuzhe Tang, Zhaoguo Wang, and Haibo Chen. Sindex: a scalable learned index for string keys. In *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.

77 Jiacheng Wu, Yong Zhang, Shimin Chen, Yu Chen, Jin Wang, and Chunxiao Xing. Updatable learned index with precise positions. *Proc. VLDB Endow.*, 14(8):1276–1288, 2021. doi:10.14778/3457390.3457393.

78 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. PMLR, 2023. URL: https://proceedings.mlr.press/v202/zeighami23a.html.

6 Complete Experimental Results

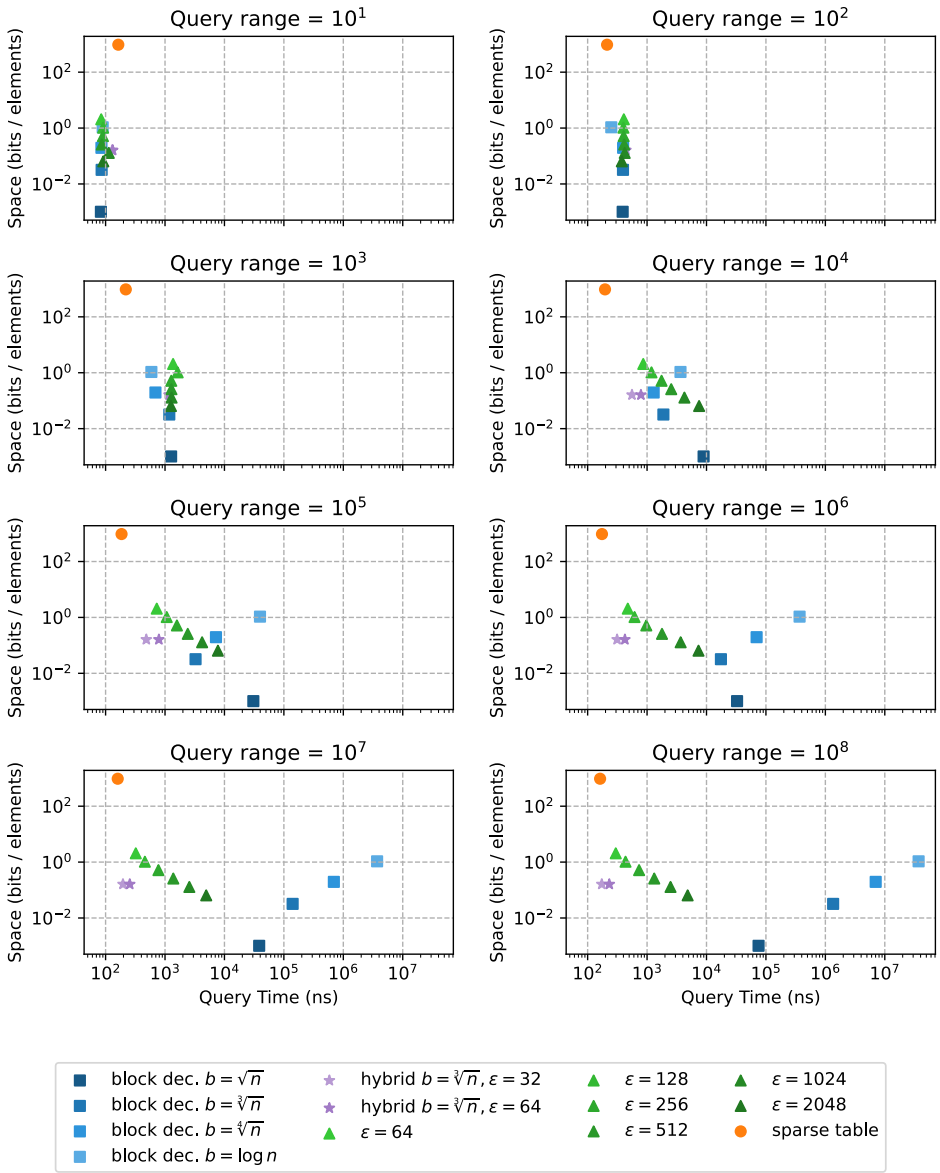

Figure 4 RMQs space-time performances on the `RAND` dataset.

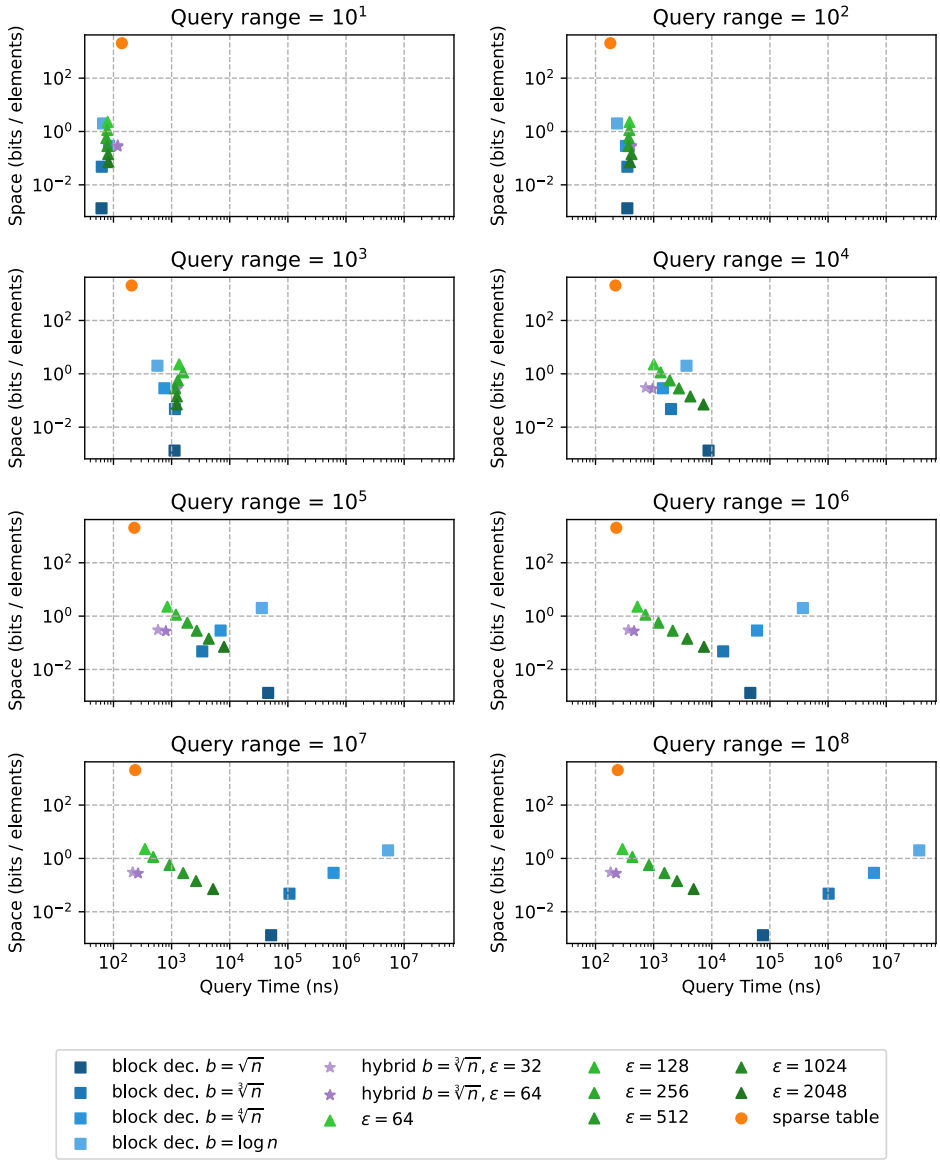

Figure 5 RMQs space-time performances on the `ENGLISH` dataset.

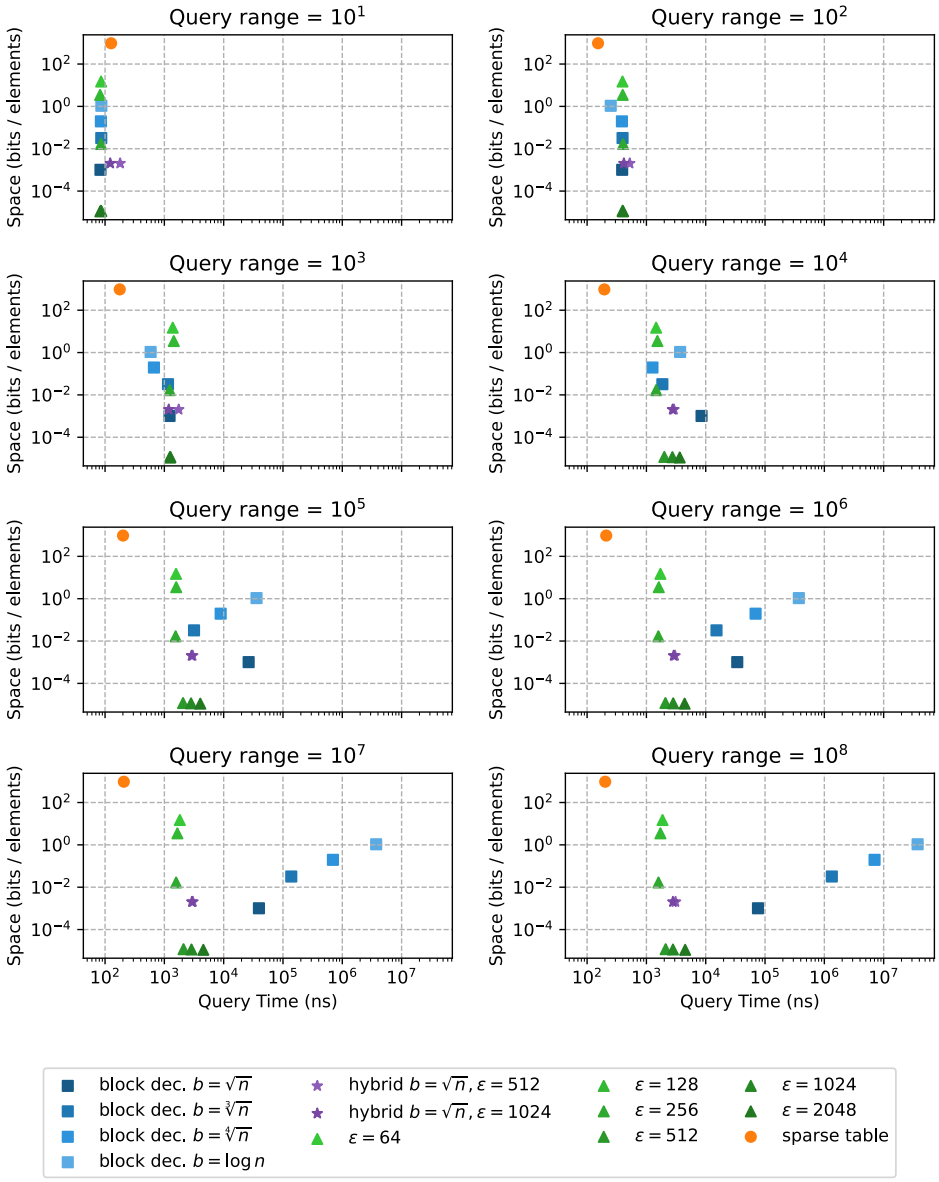

Figure 6 RMQs space-time performances on the `INC-`δ dataset ($\delta = 10^4$).

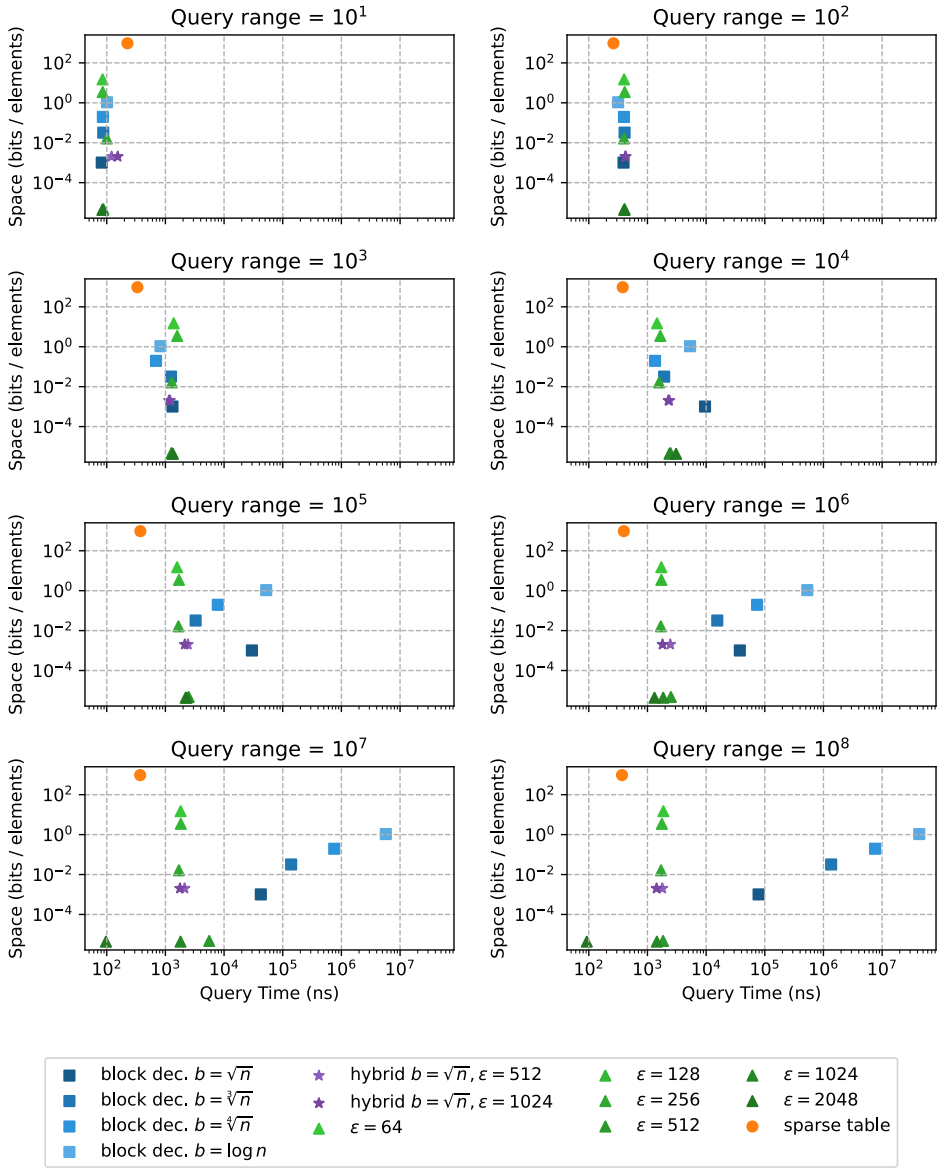

Figure 7 RMQs space-time performances on the DEC-δ dataset ($\delta = 10^4$).