On Distribution Dependent Sub-Logarithmic Query Time of Learned Indexing

Sepanta Zeighami [1] **Cyrus Shahabi** [1]

Abstract

A fundamental problem in data management is to find the elements in an array that match a query. Recently, learned indexes are being extensively used to solve this problem, where they learn a model to predict the location of the items in the array. They are empirically shown to outperform non-learned methods (e.g., B-trees or binary search that answer queries in $O(\log n)$ time) by orders of magnitude. However, success of learned indexes has not been theoretically justified. Only existing attempt shows the same query time of $O(\log n)$, but with a constant factor improvement in space complexity over non-learned methods, under some assumptions on data distribution. In this paper, we significantly strengthen this result, showing that under mild assumptions on data distribution, and the same space complexity as non-learned methods, learned indexes can answer queries in $O(\log\log n)$ expected query time. We also show that allowing for slightly larger but still near-linear space overhead, a learned index can achieve $O(1)$ expected query time. Our results theoretically prove learned indexes are orders of magnitude faster than non-learned methods, theoretically grounding their empirical success.

1. Introduction

It has been experimentally observed, but with little theoretical backing, that the problem of finding an element in an array has very efficient learned solutions (Galakatos et al., 2019; Kraska et al., 2018; Ferragina & Vinciguerra, 2020; Ding et al., 2020). In this fundamental problem in data management, the goal is to find, given a query, the elements in the dataset that match the query (e.g., find the student with grade=q, for a number q, where "grade=q" is the query on a dataset of students). Assuming the query is on a single attribute (e.g., we filter students only based on grade), and that data is sorted based on this attribute, binary search finds the answer in $O(\log n)$ for an ordered dataset with n records. Experimental results, however, show that learning a model (called a *learned index* (Kraska et al., 2018)) that predicts the location of the query in the array can provide accurate estimates of the query answers orders of magnitude faster than binary search (and other non-learned approaches). The goal of this paper is to present a theoretical grounding for such empirical observations.

More specifically, we are interested in answering exact match and range queries over a sorted array A. Exact match queries ask for the elements in A exactly equal to the query q (e.g., grade=q), while range queries ask for elements that match a range $[q, q']$ (e.g., grade is between q and q'). Both queries can be answered by finding the index of the largest element in A that is smaller than or equal to q, which we call the rank of q, *rank*(q). Range queries require the extra step, after obtaining rank(q), of scanning the array sequentially from q up to q' to obtain all results. The efficiency of methods answering range and exact match queries depends on the efficiency of answering rank(q), which is the operation analyzed in the rest of this paper.

In the worst-case, and without further assumption on the data, binary search finds rank(q) optimally, and in $O(\log n)$ operations. Materializing the binary search tree and variations of it, e.g., B-Tree (Bayer & McCreight, 1970) and CSS-trees (Rao & Ross, 1999), utilize caching and hardware properties to improve the performance in practice but theoretical number of operations remains $O(\log n)$ (we consider data in memory and not external storage). On the other hand, *learned indexes* have been empirically shown to outperform non-learned methods by orders of magnitude. Such approaches learn a model that predicts rank(q). At query time, a model inference provides an estimate of rank(q), and a local search is performed around the estimate to find the exact index. An example is shown in Fig. 1, where for the query 13, the model returns index 3, while the correct index is 5. Then, assuming the maximum model error is ϵ, a binary search on ϵ elements of A within the model prediction (i.e., the purple sub-array in Fig. 1) finds the correct answer. The success of learned models is attributed to exploiting patterns in the observed data to learn a small model that accurately estimates the rank of a query in the array.

[1]Univerisity of Southern California. Correspondence to: Sepanta Zeighami <zeighami@usc.edu>, Cyrus Shahabi <shahabi@usc>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

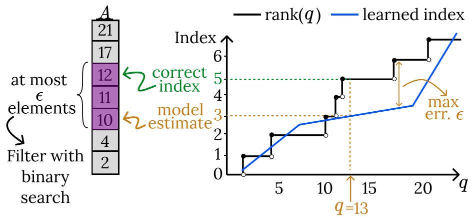

Figure 1. A learned index used to solve the rank problem.

However, to date, no theoretical result has justified their superior practical performance. Ferragina & Vinciguerra (2020) shows a worst-case bound of $O(\log n)$ on query time, the same as traditional methods, but experimentally shows orders of magnitude difference. The only existing result that shows any theoretical benefit to learned indexing is Ferragina et al. (2020), that shows constant factor better space utilization while achieving $O(\log n)$ query time under some assumptions on data distribution. The question remains whether theoretical differences, beyond merely constant factors, exist between learned and traditional approaches.

We answer this question affirmatively. We show that

(i) Using the same space overhead as traditional indexes (e.g., a B-tree), and under mild assumptions on the data distribution, a learned index can answer queries in $O(\log\log n)$ operations on expectation, a significant and asymptotic improvement over the $O(\log n)$ of traditional indexes;

(ii) With the slightly higher but still near-linear space consumption $O(n^{1+\epsilon})$, for any $\epsilon > 0$, a learned index can achieve $O(1)$ expected query time; and

(iii) Under stronger assumptions on data distribution, we show that $O(\log\log n)$ expected query time is also possible with $O(1)$ space overhead ($O(1)$ space overhead is similar to performing binary search without building any auxiliary data structure).

We present experiments showing these asymptotic bounds are achieved in practice.

These results show order of magnitude benefit in terms of expected query time, where the expectation is over the sampling of the data, and not worst-case query time (which, unsurprisingly, is $O(\log n)$ in all cases). Intuitively, this means that although there may exist data instances where a learned index is as slow as binary search, for many data instances (and on expectation), it is fast and sub-logarithmic. Analyzing expected query time allows us to incorporate properties of the data distribution. Our results hold assuming certain distribution properties: query time in (i) and (ii) is achieved assuming bounded p.d.f of data distribution ((i) also assumes non-zero p.d.f), while (iii) assumes the c.d.f of data distribution is efficiently computable. Overall, data distribution had been previously hypothesized to be an important factor on the performance of a learned index (e.g., Kraska et al. (2018)). This paper shows how such properties can be used to analyze the performance of a learned index.

2. Preliminaries and Related Work

2.1. Problem Definition

Setup. We are given an array $A \subseteq \mathcal{D}^n$, consisting of n elements, where $\mathcal{D} \subseteq \mathbb{R}$ is the domain of the elements. Unless otherwise stated, assume $\mathcal{D} = [0, 1]$; we discuss extensions to other bounded or unbounded domains in Sec. 3.4. A is sorted in ascending order, where a_i refers to the i-th element in this sorted order and $A[i : j]$ denotes the sorted subarray containing $\{a_i, ..., a_j\}$. We assume A is created by sampling n i.i.d random variables and then sorting them, where the random variables follow a continuous distribution χ, with p.d.f f_χ and c.d.f F_χ. We use the notation $A \sim \chi$ to describe the above sampling procedure.

Rank Problem. Our goal is to answer the *rank problem*: given the array A and a query q, return the index $i^* = \sum_{i=1}^{n} I_{A[i]\leq q}$, where I is the indicator function. i^* is the index of the largest element no greater than q and is 0 if no such element exists. Furthermore, if $q \in A$, q will be at index $i^* + 1$. We define the *rank function* of an array A as $r_A(q) = \sum_{i=1}^{n} I_{A[i]\leq q}$. The rank function takes a query as an input and outputs the answer to the rank problem. We drop the dependence on A if it is clear from context and simply use $r(q)$.

The rank problem is therefore the problem of designing a computationally efficient method to evaluate the function $r(q)$. Let $\hat{R}_A(q; \theta)$ be a function approximator, with parameters θ that correctly evaluates $r(q)$. The parameters θ of $\hat{R}_A$ are found at a preprocessing step and are used to perform inference at query time. Let $T(\hat{R}_A, q)$ be the number of operations performed by $\hat{R}_A$ to answer the query q and let $S(\hat{R}_A)$ be the space overhead of $\hat{R}_A$, i.e., the number of bits required to store θ (note that $S(\hat{R}_A)$ does not include storage required for the data itself, but only considers the overhead of indexing). We study the *expected query time* of any query q as $\mathbb{E}_{A\sim\chi}[T(\hat{R}_A, q)]$, and the *expected space overhead* as $\mathbb{E}_{A\sim\chi}[S(\hat{R}_A)]$. In our analysis of space overhead, we assume integers are stored using their binary representation so that k integers that are at most M are stored in $O(k \log M)$ bits (i.e., assuming no compression).

Learned indexing. A *learned indexing* approach solves the rank problem as follows. A function approximator (e.g., neural network or a piecewise linear approximator) $\hat{r}_A(q; \theta)$ is first learned that approximates r up to an error ϵ, i.e., $\|\hat{r}_A - r\|_\infty \leq \epsilon$. Then, at the step called *error correction*, another function, $h(i, q)$, takes the estimate $i = \hat{r}_A(q; \theta)$ and corrects the error, typically by performing a binary search (or exponential search when ϵ is not known a priori (Ding et al., 2020)) on the array, A. That is, given that the estimate $\hat{r}_A$ is within ϵ of the true index of q in A, a binary search on the 2ϵ element in A that are within ϵ of $\hat{r}_A(q; \theta)$ finds the correct answer. Letting $\hat{R}_A(q; \theta) = h(\hat{r}_A(q, \theta), q)$, we obtain that for any function approximator, $\hat{r}_A$ with non-zero error ϵ, we can obtain an exact function with expected query time of $\mathbb{E}_{A\sim\chi}[T(\hat{r}_A, q)] + O(\log \epsilon)$ and space overhead of $\mathbb{E}_{A\sim\chi}[S(\hat{r}_A)]$ since binary search requires no additional storage space. In this paper, we show the existence of function approximators, $\hat{R}_A$ that can achieve sub-logarithmic query time with various space overheads.

2.2. Related Work

Learned indexes. The only existing work theoretically studying a learned index is Ferragina et al. (2020). It shows, under assumptions on the gaps between the keys in the array, as $n \to \infty$ and almost surely, one can achieve logarithmic query time with a learned index with a constant factor improvement in space consumption over non-learned indexes. We significantly strengthen this result, showing *sub-logarithmic* expected query time under various space overheads. Our assumptions are on the data distribution itself which is more natural than assumption on the gaps, and our results hold for any n (and not as $n \to \infty$). Though scant in theory, learned indexes have been extensively utilized in practice, and various modeling choices have been proposed under different settings, e.g., Galakatos et al. (2019); Kraska et al. (2018); Ferragina & Vinciguerra (2020); Ding et al. (2020) to name a few. Our results use a hierarchical model architecture, similar to Recursive Model Index (RMI) (Kraska et al., 2018) and piecewise approximation similar to Piecewise Geometric Model index (PGM) (Ferragina & Vinciguerra, 2020) to construct function approximators with sub-logarithmic query time.

Non-Learned Methods. Binary search trees, B-Trees (Bayer & McCreight, 1970) and many other variants (Rao & Ross, 1999; Lehman & Carey, 1985; Bayer, 1972), exist that solve the problem in $O(\log n)$ query time, which is the best possible in the worst case in comparison based model (Navarro & Rojas-Ledesma, 2020). The space overhead for such indexes is $O(n \log n)$ bits, as they have $O(n)$ nodes and each node can be stored in $O(\log n)$ bits. We also note in passing that if we limit the domain of elements to a finite integer universe and do not consider range queries, various other time/space trade-offs are possible (Pătraşcu & Thorup, 2006), e.g., using hashing (Fredman et al., 1984).

3. Asymptotic Behaviour of Learned Indexing

3.1. Constant Time and Near-Linear Space

We first consider the case of constant query time.

Theorem 3.1. *Suppose the p.d.f,* $f_\chi(x)$, *is bounded, i.e.,* $f_\chi(x) \leq \rho$ *for all* $x \in \mathcal{D}$, *where* $\rho < \infty$. *There exists a learned index with space overhead* $O(\rho^{1+\epsilon} n^{1+\epsilon})$, *for any* $\epsilon > 0$, *with expected query time of* $O(\log \frac{1}{\epsilon})$ *operations for any query.* ρ *is a constant independent of* n, *and for any constant* ϵ, *asymptotically in* n, *space overhead is* $O(n^{1+\epsilon})$ *and expected query time is* $O(1)$.

The theorem shows the surprising result that we can in fact achieve constant query time with a learned index of

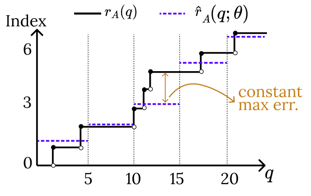

Figure 2. Approximation with a piecewise constant function

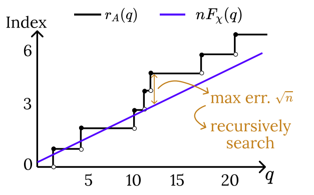

Figure 3. Approximation with c.d.f

size $O(n^{1+\epsilon})$. Although the space overhead is near-linear, this overhead is asymptotically larger than the overhead of traditional indexes (with overhead $O(n \log n)$) and thus the query time complexities are not directly comparable.

Interestingly, the function approximator that achieves the bound in Theorem 3.1 is a simple piecewise constant function approximator, which can be seen as a special case of the PGM model that uses piece-wise linear approximation (Ferragina & Vinciguerra, 2020). Our function approximator is constructed by uniformly dividing the space into k intervals and for each interval finding a constant that best approximates the rank function in that interval. Such a function approximator is shown as $\hat{r}_A(q;\theta)$ in Fig. 2 for $k = 5$. Obtaining constant query time requires such a function approximator to have constant error. It is, however, non-obvious why and when only $O(n^{1+\epsilon})$ pieces will be sufficient on expectation to achieve constant error. In fact, for the worst-case (and not the expected case), for a heavily skewed dataset, achieving constant error would require an arbitrarily large k, as noted by Kraska et al. (2018).

However, Theorem 3.1 shows as long as the p.d.f. of the data distribution is bounded, $O(n^{1+\epsilon})$ pieces will be sufficient for constant query time on expectation. Intuitively, the bound on the p.d.f. is used to argue that the number of data points sampled in a small region is not too large, which is in turn used to bound the error of the function approximation.

Finally, dependence on ρ in Theorem 3.1 is expected, as performance of learned indexes depends on the dataset characteristics. ρ captures such data dependencies, showing that such data dependencies only affect space overhead by a constant factor. From a practical perspective, our experiments in Sec. 5.2 show that for many commonly used real-world benchmarks for learned indexes, trends predicted by Theorem 3.1 hold with $\rho = 1$. However, Sec. 5.2 also shows that for datasets where learned indexes are known to perform poorly, we observe large values of ρ. Thus, ρ can be used to explain why and when learned indexes perform well or poorly in practice.

3.2. Log-Logarithmic Time and Constant Space

Requiring constant query time, as in the previous theorem, can be too restrictive. Allowing for slightly larger query time, we have the following result.

Theorem 3.2. *Suppose c.d.f of data distribution* $F_\chi(x)$ *can*

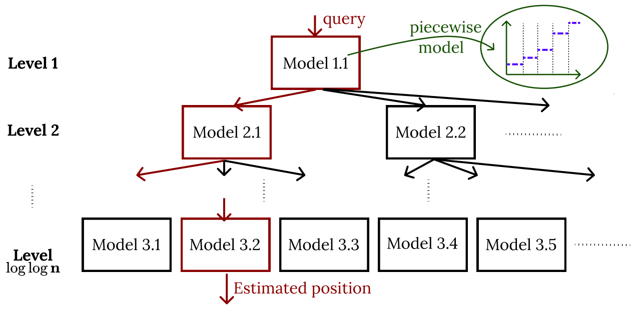

Figure 4. RMI of height $\log \log n$ with piecewise constant models

be evaluated exactly with $O(1)$ operations and $O(1)$ space overhead. There exists a learned index with space overhead $O(1)$, where for any query q, the expected query time is $O(\log \log n)$ operations.

The result shows that we can obtain $O(\log \log n)$ query time if the c.d.f of the data distribution is easy to compute. This is the case for the uniform distribution (whose c.d.f is a straight line), or more generally any distribution with piece-wise polynomial c.d.f. In this regime, we only utilize constant space, and thus our bound is comparable with performing a binary search on the array, which takes $O(\log n)$ operations, showing that the learned approach enjoys an order of magnitude theoretical benefit.

Our model of the rank function is $n \times F_\chi$, where F_χ is the c.d.f of the data distribution. As Fig. 3 shows, our search algorithm proceeds recursively, at each iteration reducing the search space by around $\sqrt{n}$. Intuitively, the $\sqrt{n}$ is due to the Dvoretzky-Kiefer-Wolfowitz (DKW) bound (Massart, 1990), which is used to show that with high probability the answer to a query, q is within $\sqrt{n}$ of $nF_\chi(q)$. Reducing the search space, s, by roughly $\sqrt{s}$ at every level by recursively applying DKW, we obtain the total search time of $O(\log \log n)$ (note that binary search only reduces the search space by a factor of 2 at every iteration).

3.3. Log-Logarithmic Time and Quasi-Linear Space

Finally, we show that the requirement of Theorem 3.2 on the c.d.f. is not necessary to achieve $O(\log \log n)$ query time, provided quasi-linear space overhead is allowed. The following theorem shows that a learned index can achieve $O(\log \log n)$ query time under mild assumptions on the data distribution and utilizing quasi-linear space.

Theorem 3.3. *Suppose p.d.f of data distribution $f_\chi(x)$ is bounded and more than zero, i.e., $\rho_1 \leq f_\chi(x) \leq \rho_2$ for all $x \in \mathcal{D}$, where $\rho_1 > 0$ and $\rho_2 < \infty$. There exists a learned index with expected query time equal to $O(\log \log n)$ operations and space overhead $O(\frac{\rho_2}{\rho_1} n \log n)$, for any query. Specifically, $\frac{\rho_2}{\rho_1}$ is a constant independent of n, so that, asymptotically in n, space overhead is $O(n \log n)$.*

This regime takes space similar to data size, and is where most traditional indexing approaches lie, e.g., binary trees and B-trees, where they need $O(n \log n)$ storage (the $\log n$ is due to the number of bits needed to store each node content) and achieve $O(\log n)$ query time.

The learned index that achieves the bound in Theorem 3.3 is an instance of the Recursive Model Index (RMI) (Kraska et al., 2018). Such a learned index defines a hierarchy of models, as shown in Fig. 4. Each model is used to pick a model in the next level of the tree until a model in the leaf level is reached, whose prediction is the estimated position of the query in the array. Unlike RMI in (Kraska et al., 2018), its height or size of the model within each node is not constant and set based on data size.

Intuitively, the hierarchy of models is a materialization of a search tree based on the recursive search used to prove Theorem 3.2. At any level of the tree, if the search space is s elements (originally, $s = n$) a model is used to reduce the search space to roughly $\sqrt{s}$. It is however non-trivial why and when such a model should exist across all levels and how large the model should be. We use the relationship between the rank function and the c.d.f (through DKW bound), and the properties of the data distribution to show that a model of size around $\sqrt{s}$ is sufficient with high probability. Note that models at lower levels of the hierarchy approximate the rank function only over subsets of the array, but with increasingly higher accuracy. A challenge is to show that such an approximability result holds across all models and all subsets of the array, which is why a lower bound on the p.d.f. is needed in this theorem.

Similar to ρ in Theorem 3.1, ρ_1 and ρ_2 capture data characteristics in Theorem 3.3, showing constant factor dependencies on the model size. Our experiments in Sec. 5.2 show that for most commonly used real-world benchmarks for learned indexes, trends predicted by Theorem 3.3 hold with $\frac{\rho_2}{\rho_1} = 1$. However, Sec. 5.2 also shows that for datasets where learned indexes are known to perform poorly, $\frac{\rho_2}{\rho_1}$ is large, so that $\frac{\rho_2}{\rho_1}$ can be used to explain why and when learned indexes perform well or poorly in practice.

3.4. Distributions with Other Domains

So far, our results assume that the domain of data distribution is [0, 1]. The result can be extended to distributions with other bounded domains, $[r, s]$ for $r, s \in \mathbb{R}$, $r < s$, by standardizing χ as $\frac{\chi - r}{s - r}$. This transformation scales p.d.f of χ by $s - r$. Note that scaling the p.d.f does not affect Theorem 3.3, since both ρ_1 and ρ_2 will be scaled by $s - r$, yielding the same ratio $\frac{\rho_2}{\rho_1}$. On the other hand, ρ in Theorem 3.1 will be scaled by $s - r$. Overall, bounded domain can be true in many scenarios, as the data can be from some phenomenon that is bounded, e.g., age, grade, data over a period of time. Next, we extend our results to distributions with unbounded domains.

Lemma 3.4. *Suppose a learned index, $\hat{R}$, achieves expected* *query time $t(n)$ and space overhead $s(n)$ on distributions with domain $[0,1]$ and bounded (and non-zero) p.d.f. There exists a learned index, $\hat{R}'$ with expected query time $t(n)+1$ and space overhead $O(s(n)\log n)$ on any sub-exponential distribution with bounded (and non-zero) p.d.f.*

Combining Lemma 3.4 with Theorems 3.1 and 3.3, our results cover various well-known distributions, e.g., Gaussian, squared of Gaussian and exponential distributions.

Proof of lemma 3.4 builds the known learned index for bounded domains on $\log n$ different bounded intervals. This achieves the desired outcome due to the tail behavior of sub-exponential distributions (i.e., for distributions with tail at most as heavy as exponential, see Vershynin (2018) for definition). The tail behaviour allows us to, roughly speaking, assume that the domain of the function is $O(\log n)$, because observing points outside this range is unlikely. We note that other distributions with unbounded domain can also be similarly analyzed based on their tail behaviour, with heavier tails leading to higher space consumption.

4. Proofs

Proofs of the theorems are all constructive. PCA Index (Sec. 4.1) proves Theorem 3.1, RDS algorithm proves Theorem 3.2 and RDA Index proves Theorem 3.3. Without loss of generality, we assume the bounded domain $\mathcal{D}$ is $[0,1]$. The proof for the unbounded domain case (i.e., Lemma 3.4) is deferred to Appendix A. Proof of technical lemmas stated throughout this section can also be found in Appendix A.

4.1. Proof of Theorem 3.1: PCA Index

We present and analyze *Piece-wise Constant Approximator (PCA) Index* that proves Theorem 3.1.

4.1.1. Approximating Rank Function

We show how to approximate the rank function r with a function approximator $\hat{r}$. To achieve constant query time, approximation error should be a constant independent of n with high probability, and we also should be able to evaluate $\hat{r}$ in constant time.

Lemma 4.1 shows these properties hold for a piece-wise constant approximation to r. Such a function is presented in Alg. 1 (and an example was shown in Fig. 2). Alg. 1 uniformly divides the function domain into k intervals, so that the i-th constant piece is responsible for the interval $I_i=[i\times\frac{1}{k},(i+1)\times\frac{1}{k}]$. Since $r(q)$ is a non-decreasing function, the constant with the lowest infinity norm error approximating r over I_i is $\frac{1}{2}(r(\frac{i}{k})+r(\frac{i+1}{k}))$ (line 6). Let $\hat{r}_k$ be the function returned by PCF(A, k, 0, 1).

Lemma 4.1. *Under the conditions of Theorem 3.1 and for $k\geq n^{1+\epsilon}\rho^{1+\frac{\epsilon}{2}}$, the error of $\hat{r}_k$ is bounded as*

$$\mathbb{P}(\|\hat{r}_k-r\|_\infty\geq\frac{2}{\epsilon}+1)\leq\frac{1}{n}.$$

Proof of Lemma 4.1. Let $e_i=\sup_{x\in I_i}|\hat{r}(x;\theta)-r(x)|$ be the maximum error in the i-th piece of $\hat{r}$. e_i can be bounded by the number of points sampled in I_i as follows.

Algorithm 1 PCA Index Construction

```
Require: A sorted array $A$, number of pieces $k$, approximation domain lower and upper bounds $l$ and $u$
Ensure: Piecewise constant approximation of $r$ over $[l,u]$
1: procedure PCF($A$, $k$, $l$, $u$)
2:     $P \leftarrow$ array of length $k$ storing the pieces
3:     $\alpha \leftarrow \frac{(u-l)}{k}$
4:     $\delta \leftarrow 0$
5:     for $i \leftarrow 0$ to $k$ do
6:         $P[i] \leftarrow \frac{1}{2}(r_A(l+\alpha i)+r_A(l+\alpha(i+1))$
7:         $\delta_{curr} \leftarrow \lceil\frac{1}{2}(r_A(l+\alpha(i+1))-r_A(l+\alpha i))\rceil$
8:         $\delta \leftarrow \max\{\delta,\delta_{curr}\}$
       return $P$, $\delta$
```

Proposition 4.2. *Let $s_i=|\{j|a_j\in I_i\}|$ be the number of points in A that are in I_i. We have $e_i\leq s_i$*

Using Prop. 4.2, we have $\|\hat{r}-r\|_\infty\leq\max_{i\in\{1,\dots,k\}}s_i$. Prop. 4.2 is a simple fact that relates approximation error to statistical properties of data distribution. Define $s_{max}=\max_{i\in\{1,\dots,k\}}s_i$ and observe that s_{max} is a random variable denoting the maximum number of points sampled per interval, across k equi-length intervals. The following lemma shows that we can bound s_{max} with a constant and with probability $\frac{1}{n}$, as long as k is near-linear in n.

Lemma 4.3. *For any c with $c\geq3$, and if $k\geq n^{1+\frac{2}{c-1}}\rho^{1+\frac{1}{c-1}}$ we have $\mathbb{P}(s_{max}\geq c)\leq\frac{1}{n}$.*

Setting $c=\frac{2}{\epsilon}+1$, we see $k\geq n^{1+\epsilon}\rho^{1+\frac{\epsilon}{2}}$ holds, so that Lemma 4.3 together with Prop. 4.2 prove Lemma 4.1. □

4.1.2. Index Construction and Querying

Let $k=\lceil n^{1+\frac{\epsilon}{2}}\rho^{1+\frac{\epsilon}{4}}\rceil$. We use PCF($A$, k, 0, 1) to obtain $\hat{r}_k$ and δ, where δ is the maximum observed approximation error. As Alg. 1 shows, $\hat{r}_k$ can be stored as an array, P, with k elements. To perform a query, the interval, i, a query falls into is calculated as $i=\lfloor qk\rfloor$ and the constant responsible for that interval, $P[i]$, returns the estimate. Given maximum error δ, we perform a binary search on the subarray $A[l:u]$, for $l=P[i]-\delta$ and $u=P[i]+\delta$ to obtain the answer.

4.1.3. Complexity Analysis

P has $O(n^{1+\frac{\epsilon}{2}})$ entries, and each can be stored in $O(n^{\frac{\epsilon}{2}})$. Thus, total space complexity is $O(n^{1+\epsilon})$. Regarding query time, the number of operations needed to evaluate $\hat{r}_k$ is constant. Thus, the total query time of the learned index is $O(\log\delta)$. Lemma 4.1 bounds δ, so that the query time for any query is at most $\log(\frac{4}{\epsilon}+1)$ with probability at least $1-\frac{1}{n}$ and at most $\log n$ with probability at most $\frac{1}{n}$. Consequently, the expected query time is at most $O(\log(\frac{4}{\epsilon}+1)\times(1-\frac{1}{n})+\log n\times\frac{1}{n})$ which is $O(1)$ for any constant $\epsilon>0$. □

4.2. Proof of Theorem 3.2: RDS Algorithm

We present and analyze *Recursive Distribution Search (RDS)* Algorithm that proves Theorem 3.2.

4.2.1. Approximating Rank Function

We approximate the rank function using the c.d.f of the data distribution, which conditions of Theorem 3.2 imply is easy to compute. As noted by Kraska et al. (2018), rank$(q) = nF_n(q)$, where F_n is the empirical c.d.f. Using this together with DKW bound (Massart, 1990), we can establish that rank(q) is within error $\sqrt{n}$ of nF_χ with high probability. However, error of $\sqrt{n}$ is too large: error correction to find rank(q) would require $O(\log\sqrt{n}) = O(\log n)$ operations.

Instead, we recursively improve our estimate by utilizing information that becomes available from observing elements in the array. After observing two elements, a_i and a_j in A ($i < j$), we update our knowledge of the distribution of elements in $A[i+1 : j-1]$ as follows. Define $F_\chi^{i,j}(x) = \frac{F_\chi(x)-F_\chi(a_i)}{F_\chi(a_j)-F_\chi(a_i)}$. Informally, any element X in $A[i+1 : j-1]$ is a random variable sampled from χ and knowing the value of a_i and a_j implies that $X \in [a_i, a_j]$, so that the conditional c.d.f of X is

$$\mathop{\mathbb{P}}_{X\sim\chi}(X \le x | a_i \le X \le a_j) = F_\chi^{i,j}(x).$$

We then use DKW bound to show $F_\chi^{i,j}$ is a good estimate of the rank function for the subarray $A[i+1 : j-1]$, defining the rank function for the subarray $A[i+1 : j-1]$ as $r^{i,j}(q) = \sum_{z=i+1}^{j-1} I_{a_z \le q}$. Formally, the following lemma shows that given observations $A[i] = a_i$ and $A[j] = a_j$ the elements of $A[i+1 : j-1]$ are i.i.d random variables with the conditional c.d.f $F_\chi^{i,j}(x)$ and uses the DKW bound to bound the approximation error of using the conditional c.d.f to approximate the conditional rank function.

Lemma 4.4. *Consider two indexes i, j, where $1 \le i < j \le n$ and $a_i < a_j$. Let $k = j - i - 1$. For $k \ge 2$, we have*

$$\mathbb{P}(\sup_x |r^{i,j}(x) - kF_\chi^{i,j}(x)| \ge \sqrt{0.5k\log\log k}) \le \frac{1}{\log k}.$$

4.2.2. Querying

We use Lemma 4.4 to recursively search the array. At every iteration, the search is over a subarray $A[i : j]$ (initially, i=1 and $j = n$). We observe the values of a_i and a_j and use Lemma 4.4 to estimate which subarray is likely to contain the answer to the query. This process is shown in Alg. 2. In lines 5-7 the algorithm observes a_i and a_j and attempts to answer the query based on those two observations. If it cannot, lines 8-11 use Lemma 4.4 and the observed values of a_i and a_j to estimate which subarray contains the answer. Line 12 then checks if the estimated subarray is correct, i.e., if the query does fall inside the estimated subarray. If the estimate is correct, the algorithm recursively searches the subarray. Otherwise, the algorithm exits and performs binary search on the current subarray. Finally, line 3 exits when the size of the dataset is too small. The constant 25 is chosen for convenience of analysis (see Sec. 4.2.3).

Algorithm 2 Recursive Distribution Search Algorithm

```
Require: A sorted array $A$ of size $n$ searched from index $i$ to $j$, a query $q$
Ensure: Rank of $q$ in $A[i : j]$
 1: procedure SEARCH($A$, $q$, $i$, $j$)
 2:     $k \leftarrow j - i - 1$
 3:     if $k < 25$ then
 4:         return $i$-1+BINARYSEARCH($A$, $q$, $i$, $j$)
 5:     if $a_i > q$ then return $0$
 6:     if $a_i = q$ then return $1$
 7:     if $a_j \le q$ then return $j - i + 1$
 8:     $\hat{i} \leftarrow i + 1 + k \times F_\chi^{i,j}(q)$
 9:     $r \leftarrow \sqrt{0.5k\log\log k}$
10:     $l \leftarrow \lfloor \hat{i} - r \rfloor$
11:     $u \leftarrow \lceil \hat{i} + r \rceil$
12:     if $a_l > q$ or $a_r < q$ then
13:         return $i - 1$+BINARYSEARCH($A$, $q$, $i$, $j$)
14:     return $l - 1$+ SEARCH($A$, $q$, $l$, $u$)
```

4.2.3. Complexity Analysis

To prove Theorem 3.2, it is sufficient to show that expected query time of Alg. 2 is $O(\log\log n)$ for any query. The algorithm recursively proceeds. At each recursion level, the algorithm performs a constant number of operations unless it exits to perform a binary search. Let the depth of recursion be h and let k_i be the size of the subarray at the i-th level of recursion (so that binary search at i-th level takes $O(\log k_i)$). Let B_i denote the event that the algorithm exits to perform binary search at the i-th iteration. Thus, for any query q, the expected number of operations is

$$\mathbb{E}_{A\sim\chi}[T(\hat{r}, q)] = \sum_{i=1}^{h} c_1 + c_2\mathbb{P}(B_i, \bar{B}_{i-1}, ...\bar{B}_1)\log k_i$$

for constants c_1 and c_2. Note that $\mathbb{P}(B_i, \bar{B}_{i-1}, ...\bar{B}_1) \le \mathbb{P}(B_i|\bar{B}_{i-1}, ...\bar{B}_1)$, where $\mathbb{P}(B_i|\bar{B}_{i-1}, ...\bar{B}_1)$ is the probability that the algorithm reaches i-th level of recursion and exits. By Lemma 4.4, this probability bounded by $\frac{1}{\log k_i}$. Thus $\mathbb{E}_{A\sim\chi}[T(\hat{r}, q)]$ is $O(h)$.

To analyze the depth of recursion, recall that at the last level, the size of the array is at most 25. Furthermore, at every iteration the size of the array is reduced to at most $2\sqrt{0.5n\log\log n}+2$. For $n \ge 25$, $2\sqrt{0.5n\log\log n}+2 \le n^{\frac{3}{4}}$, so that the size of the array at the i-th recursions is at most $n^{(\frac{3}{4})^i}$ and the depth of recursion is $O(\log\log n)$. Thus, the expected total time is $O(\log\log n)$ $\square$.

4.3. Proof of Theorem 3.3: RDA Index

We present and analyze *Recursive Distribution Approximator (RDA)* Index that proves Theorem 3.3.

4.3.1. Approximating Rank Function

We use ideas from Theorems 3.1 and 3.2 to approximate the rank function. We use Alg. 2 as a blueprint, but instead of the c.d.f, we use a piecewise constant approximation to the rank function. If we can efficiently approximate the rank function for subarray $A[i-1:j+1]$, $r^{i,j}$, to within accuracy $O(\sqrt{k\log\log k})$ where $k=j-i-1$, we can merely replace line 8 of Alg. 2 with our function approximator and still enjoy the $O(\log\log n)$ query time. Indeed, the following lemma shows that this is possible using the piecewise approximation of Alg. 1 and under mild assumptions on the data distribution. Let $\hat{r}_t^{i,j}$ be the function returned by PCF($A[i+1:j-1]$, t, a_i, a_j) with t pieces.

Lemma 4.5. *Consider two indexes i, j, where $1 \leq i < j \leq n$ and $a_i < a_j$. Let $k = j-i-1$. For $k \geq 2$, under the conditions of Theorem 3.3 and for $t \geq \frac{\rho_2}{\rho_1}\sqrt{k}$ we have*

$$\mathbb{P}(\|r^{i,j} - \hat{r}_t^{i,j}\|_\infty \geq (\sqrt{0.5\log\log k}+1)\sqrt{k}) \leq \frac{1}{\log k}.$$

Proof of Lemma 4.5. Alg. 1 finds the piecewise constant approximator to $r^{i,j}$ with t pieces with the smallest infinity norm error. Thus, we only need to show the existence of an approximation with t pieces that satisfies conditions of the lemma. To do so, we use the relationship between $r^{i,j}$ and the conditional c.d.f. Intuitively, Lemma 4.4 shows that $r^{i,j}$ and the conditional c.d.f are similar to each other and thus, if we can approximate conditional c.d.f well, we can also approximate $r^{i,j}$. Formally, by triangle inequality and for any function approximator $\hat{r}$ we have

$$\|r^{i,j} - \hat{r}\|_\infty \leq \|r^{i,j} - kF_\chi^{i,j}\|_\infty + \|kF_\chi^{i,j} - \hat{r}\|_\infty. \quad (1)$$

Combining this with Lemma 4.4 we obtain

$$\mathbb{P}(\|r^{i,j} - \hat{r}\|_\infty \geq \sqrt{0.5k\log\log k} + \|kF_\chi^{i,j} - \hat{r}\|_\infty) \leq \frac{1}{\log k}.$$

Finally, Lemma 4.6 stated below shows how we can approximate the conditional c.d.f and completes the proof. □

Lemma 4.6. *Under the conditions of Lemma 4.5, there exists a piecewise constant function approximator, $\hat{r}$, with $\frac{\rho_2}{\rho_1}\sqrt{k}$ pieces such that $\|\hat{r} - kF_\chi^{i,j}\|_\infty \leq \sqrt{k}$.*

4.3.2. Index Construction and Querying

Lemma 4.5 is an analog of Lemma 4.4, showing a function approximator enjoys similar properties as the c.d.f. However, different function approximators are needed for every subarray (for c.d.f.s we merely needed to scale and shift them differently for different subarrays). Given that there are $O(n^2)$ different subarrays, a naive implementation that creates a function approximator for each subarray takes space quadratic in data size. Instead, we only approximate the conditional rank function for certain sub-arrays while still retaining the $O(\sqrt{k\log\log k})$ error bound per subarray.

Construction. Note that $r(q)=0$ only if $q<a_1$, so we can filter this case out and assume $r(q)\in\{1,...,n\}$. RDA is a tree, shown in Fig. 4, where each node is associated with a model. When querying the index, we traverse the tree from the root, and at each node, we use the node's model to choose the next node to traverse. Traversing down the tree narrows down the possible answers to $r(q)$. We say that a node N *covers* a range S_N, if we have $r(q)\in S_N$ for any query, q, that traverses the tree and reaches N. We call $|S_N|$ node N's *coverage size*. Coverage size is the size of search space left to search after reaching a node. The root node, N, covers $\{1,...,n\}$ with coverage size n and the coverage size decreases as we traverse down the tree. Leaf nodes have coverage size independent of n with high probability, so that finding $r(q)$ takes constant time after reaching a leaf. Each leaf node stores the subarray corresponding to the range it covers as its `content`.

Algorithm 3 RDA Index Construction

```
Require: A sorted array $A$ of size $n$ sampled from a distribution $\chi$ with CDF $F_\chi$, a query $q$
Ensure: The root node of the learned index
 1: procedure BUILDTREE($A$, $i$, $j$)
 2:     $k \leftarrow j-i+1$                          ▷ size of $A[i:j]$
 3:     if $k \leq 61$ then
 4:         return Leaf node with content $A[i:j]$
 5:     $\hat{r}, \epsilon \leftarrow$ PCF($A[i:j]$, $\lceil\frac{\rho_2}{\rho_1}\sqrt{k}\rceil$, $a_i$, $a_j$)
 6:     $k' \leftarrow \lceil 2\sqrt{k}(1+\sqrt{0.5\log\log k})+2\rceil$
 7:     if $\epsilon > \frac{k'}{2}$ then
 8:         return Leaf node with content $A[i:j]$
 9:     $C \leftarrow$ array of size $\lceil\frac{k}{k'}\rceil$ containing children
10:     for $z \leftarrow 0$ to $\lceil\frac{k}{k'}\rceil$ do
11:         $C[z] \leftarrow$ BUILDTREE($A$, $zk'$, $(z+2)k'$)
12:     return Non-leaf node with children $C$ and model $\hat{r}$ with max_err $\frac{k'}{2}$
```

RDA is built by calling BUILDTREE(A, 1, n), as presented in Alg. 3. BUILDTREE(A, i, j) returns the root node, N, of a tree, where N covers $\{i,...,j\}$. If the coverage size of N is smaller than some prespecified constant (line 3, analogous to line 3 in Alg. 2), the algorithm turns N into a leaf node. Otherwise, in line 5 it uses Lemma 4.5 to create the model $\hat{r}$ for N, where $\hat{r}$ approximates $r^{i-1,j+1}$ (recall that $r^{i,j}$ is the index function for subarray $A[i+1:j-1]$). If the error of $\hat{r}$ is larger than predicted by Lemma 4.5, the algorithm turns N into a leaf node and discards the model (this is analogous to line 12 in Alg. 2). Finally, for k' as in line 6, the algorithm recursively builds $\lceil\frac{k}{k'}\rceil$ children for N. Each child has a coverage size of $2k'$ and the ranges are spread at k' intervals (line 11). This ensures that the set $\hat{R}=\{\hat{r}-\epsilon,\hat{r}-\epsilon+1,...,\hat{r}+\epsilon\}$, (with $|\hat{R}|\leq k'$ ensured by line 7) is a subset of the range covered by one of N's children. Furthermore, for any query q, ϵ is the maximum error of $\hat{r}$, so $r(q)\in\hat{R}$. Thus, the construction ensures that for any query q that reaches N, $r(q)$ is in the range covered by one of the children of N.

Performing Queries. As Alg. 4 shows, to traverse the tree for a query q from a node N, we find the child of N whose covered range contains $r(q)$. When $\hat{i}$ is N.`model` estimate

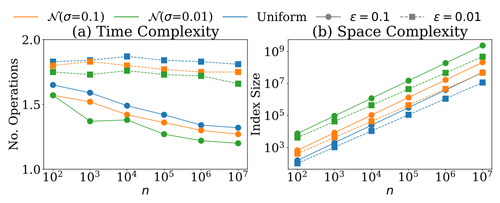

Figure 5. Constant Query and Near-Linear Space

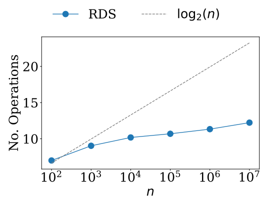

Figure 6. Log-Logarithmic Query and Constant Space

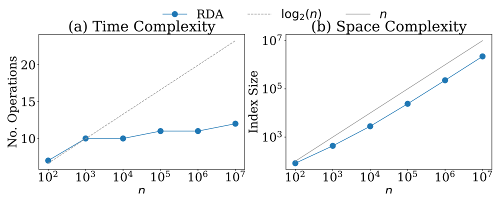

Figure 7. Log-Logarithmic Query and Quasi-Linear Space

Algorithm 4 RDA Index Querying

```
Require: The root node, $N$, of a learned index, a query $q$
Ensure: Rank of query $q$
1: procedure QUERY($N$, $q$)
2:     if $N$ is a leaf node then
3:         return BINARYSEARCH($N$.content)
4:     $\hat{i} \leftarrow N$.model($q$)
5:     $k \leftarrow N$.max_err($q$)
6:     $z \leftarrow \lfloor \frac{\hat{i}-k}{2k} \rfloor$
7:     return QUERY($N$.children[$z$], $q$)
```

with maximum error k, $z = \lfloor \frac{\hat{i}-k}{2k} \rfloor$ gives the index of the child whose range covers $\{\lfloor \frac{\hat{i}-k}{2k} \rfloor 2k, ..., (\lfloor \frac{\hat{i}-k}{2k} \rfloor + 2)2k\}$ and contains $\{\hat{i}-k, ..., \hat{i}+k\}$ as a subset and therefore contains $r(q)$. Thus, the child at index z is recursively searched.

4.3.3. Complexity Analysis

The query time analysis is very similar to the analysis in Sec. 4.2.3 and is thus deferred to Appendix A. Here, we show the space overhead complexity.

All nodes at a given tree level have the same coverage size. If the coverage size of nodes at level i is z_i, then the number of pieces used for approximation per node is $O(\frac{\rho_2}{\rho_1}\sqrt{z_i})$ and the total number of nodes at level i is at most $O(\frac{n}{z_i})$. Thus, total number of pieces used at level i is $c\frac{\rho_2}{\rho_1}\frac{n}{\sqrt{z_i}}$ for some constant c. Note that if the coverage size at level i is k, the coverage size at level $i+1$ is $4\sqrt{k}(1+\sqrt{0.5\log\log k})$ which is more than $k^{\frac{1}{2}}$. Thus, $z_i \geq n^{(\frac{1}{2})^i}$ and $c\frac{\rho_2}{\rho_1}\frac{n}{\sqrt{z_i}} \leq c\frac{\rho_2}{\rho_1}\frac{n}{n^{(\frac{1}{2})^{i+1}}}$. The total number of pieces is therefor at most $cn\frac{\rho_2}{\rho_1}\sum_{i=0}^{c'\log\log n} n^{-(\frac{1}{2})^{i+1}} \leq 3cn\frac{\rho_2}{\rho_1}$ for some constant c'. Each piece has magnitude n and can be written in $O(\log n)$ bits, so total overhead is $O(\frac{\rho_2}{\rho_1}n\log n)$ bits. □

5. Experiments

We empirically validate our theoretical results on synthetic and real datasets (specified in each experiment).

For each experiment, we report *index size* and *number of operations*. Index size is the number of stored integers by each method. Number of operations is the total number of memory operations performed by the algorithm and is used as a proxy for the total number of instructions performed by CPU. The two metrics differ by a constant factors in our algorithm (our methods perform a constant number of operations between memory accesses), but the latter is compiler dependent and difficult to compute. To report the number of operations, we randomly sample a set of 1000 queries Q and a set of $\mathcal{A}$ of 100 different arrays from the distribution χ. Let $n_{q,A}$ be the number of operations for each query in $q \in Q$ on an array $A \in \mathcal{A}$. We report $\max_{q\in Q}\frac{\sum_{A\in\mathcal{A}} n_{q,A}}{|\mathcal{A}|}$, which is the maximum (across queries) of the average (across datasets) number of operations.

5.1. Results on Synthetic Datasets

Constant Query Time and Near-Linear Space. We show that the construction presented in Sec. 4.1 achieves the bound of Theorem 3.1. We consider Uniform and two Gaussian (with $\sigma = 0.1$ and $\sigma = 0.01$) distributions. We vary Gaussian standard deviation to show the impact of the bound on p.d.f (as required by Theorem 3.1). Uniform p.d.f. has bound 1, and bound on Gaussian p.d.f with standard deviation σ is $\frac{1}{\sigma\sqrt{2\pi}}$. We present results for $\epsilon = 0.1$ and $\epsilon = 0.01$, where ϵ is the space complexity parameter in Theorem 3.1.

Fig. 5 shows the results. It corroborates Thoerem 3.1, where Fig. 5 (a) shows constant query time achieved by near-linear space shown in Fig. 5 (b). We also see for larger ϵ, query time actually decreases, suggesting our bound on query time is less tight for larger ϵ. Furthermore, recall that PCA Index scales the number of pieces by $\rho^{1+\frac{\epsilon}{2}}$ to provide the same bound on query time for all distributions (where ρ is the bound on p.d.f). We see an artifact of this in Fig. 5 (b), where when ρ increases index size also increases.

Log-Logarithmic Query Time and Constant Space. We show that the construction presented in Sec. 4.2 achieves the bound of Theorem 3.2. The theorem applies to distributions with efficiently computable c.d.f.s, so we consider distributions over $[0, 1]$ with $F_\chi(x) = x^t$ for $t \in \{1, 4, 16\}$. At $t = 1$, we have the uniform distribution and for larger t the distribution becomes more skewed. Fig. 6 corroborates the log-logarithmic bound of Theorem 3.2. Moreover, the results look identical across distributions (multiple lines are overlaid on top of each other in the figure), showing similar performance for distributions with different skew levels.

Log-Logarithmic Query Time and Quasi-Linear Space. We show that the construction presented in Sec. 4.3.3 achieves the bound of Theorem 3.3. We consider Uniform and two Gaussian ($\sigma = 0.1$ and $\sigma = 0.01$) distributions. The results are presented in Fig. 7. It corroborates

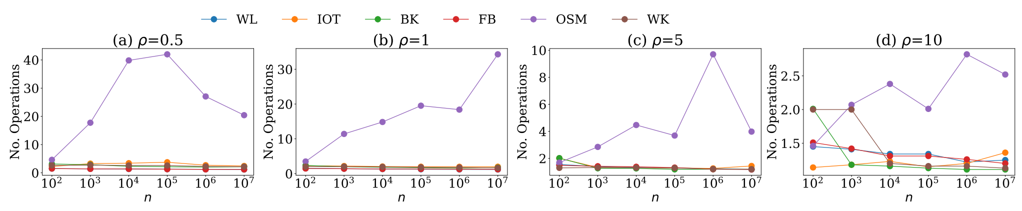

Figure 8. Constant Query Time on Real Datasets

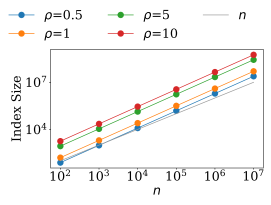

Figure 9. Near-Linear Space on Real Datasets

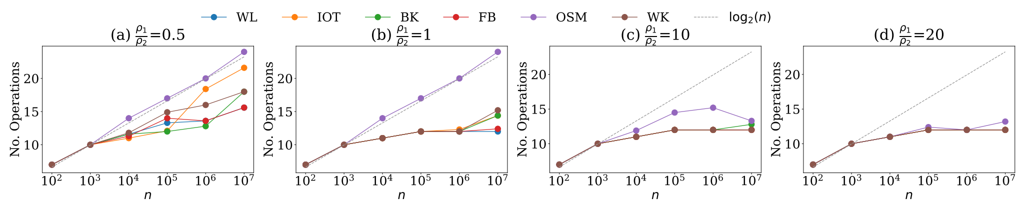

Figure 10. Log-Logarithmic Query on Real Datasets

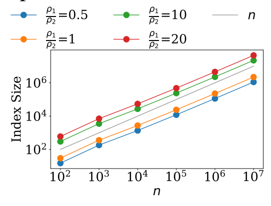

Figure 11. Quasi-Linear Space on Real Datasets

Theorem 3.3, where Fig. 7 (a) shows constant query time achieved by quasi-linear space shown in Fig. 7 (b). Similar to the previous case results look identical across distributions (multiple lines are overlaid on top of each other in the figure). Comparing Fig. 7 (a) and Fig. 6, we observe that using a piecewise function approximator achieves similar results as using c.d.f for rank function approximation.

5.2. Results on Real Datasets

Setup. On real datasets, we do not have access to the data distribution and thus we do not know the value of ρ in Theorem 3.1 or ρ_1 and ρ_2 in Theorem 3.3. Thus, for each dataset, we perform the experiments for multiple values of ρ and ρ_1 or ρ_2 to see at what values the trends predicted by the theorems emerge. Since we do not have access to the c.d.f, Theorem 3.2 is not applicable.

We use 6 real datasets commonly used for benchmarking learned indexes. For each real dataset, we sample n data points uniformly at random for different values of n from the original dataset, and queries are generated uniformly at random from the data range. The datasets are WL and IOT from Ferragina & Vinciguerra (2020); Kraska et al. (2018); Galakatos et al. (2019) and BK, FB, OSM, WK from Marcus et al. (2020) described next. WL: Web Logs dataset containing 714M timestamps of requests to a web server. IOT: timestamps of 26M events recorded by IoT sensors installed throughout an academic building. BK: popularity of 200M books from Amazon, where each key represents the popularity of a particular book. FB: 200M randomly sampled Facebook user IDs, where each key uniquely identifies a user. OSM: cell IDs of 800M locations from Open Street Map, where each key represents an embedded location. WK: timestamps of 200M edits from Wikipedia, where each key represents the time an edit was committed.

Results. Figs. 8 and 9 show time and space complexity of the PCA algorithm (Theorem 3.1) on the real datasets for various values of ρ. Note that value of ρ affects the number of pieces used, as described by Lemma 4.1. Furthermore, Figs. 10 and 11 show time and space complexity of the RDA algorithm (Theorem 3.3) on the real datasets for various ratios of $\frac{\rho_2}{\rho_1}$. Note that value of $\frac{\rho_2}{\rho_1}$ affects the number of pieces used per node, as described by Lemma 4.5.

For all except OSM datasets, trends described by Theorems 3.1 and 3.3 hold for values of ρ and $\frac{\rho_2}{\rho_1}$ as small as 1. This shows our theoretical results hold on real datasets, and the distribution dependent factors, ρ and $\frac{\rho_2}{\rho_1}$, are typically small in practice. However, on OSM dataset value of ρ and $\frac{\rho_2}{\rho_1}$ may be as large as 10 and 20 respectively. In fact, Marcus et al. (2020) shows that non-learned methods outperform learned methods on this dataset. As such, our results provide a possible explanation (large values of ρ) for learned methods not performing as well on this dataset. Indeed, OSM is a one-dimensional projection of a spatial dataset using Hilbert curves (see Marcus et al. (2020)), which distorts the spatial structure of the data and can thus lead to sharp changes to the c.d.f (and therefore large ρ).

6. Conclusion

We theoretically showed and empirically verified that a learned index can achieve sub-logarithmic expected query time under various storage overheads and mild assumptions on data distribution. All our proofs are constructive, using piecewise and hierarchical models that are common in practice. Our results provide evidence why learned indexes perform better than traditional indexes in practice. Future work includes relaxing assumptions on data distribution, finding necessary conditions for sub-logarithmic query time and analyzing the trade-offs of different modeling approaches.

Acknowledgements

This research has been funded in part by NSF grants CNS-2125530 and IIS-2128661, and NIH grant 5R01LM014026. Opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any sponsors, such as NSF.

References

Bayer, R. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta informatica*, 1(4):290–306, 1972.

Bayer, R. and McCreight, E. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, pp. 107–141, 1970.

Ding, J., Minhas, U. F., Yu, J., Wang, C., Do, J., Li, Y., Zhang, H., Chandramouli, B., Gehrke, J., Kossmann, D., et al. Alex: an updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 969–984, 2020.

Ferragina, P. and Vinciguerra, G. The pgm-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proceedings of the VLDB Endowment*, 13 (8):1162–1175, 2020.

Ferragina, P., Lillo, F., and Vinciguerra, G. Why are learned indexes so effective? In *International Conference on Machine Learning*, pp. 3123–3132. PMLR, 2020.

Fredman, M. L., Komlós, J., and Szemerédi, E. Storing a sparse table with 0 (1) worst case access time. *Journal of the ACM (JACM)*, 31(3):538–544, 1984.

Galakatos, A., Markovitch, M., Binnig, C., Fonseca, R., and Kraska, T. Fiting-tree: A data-aware index structure. In *Proceedings of the 2019 International Conference on Management of Data*, pp. 1189–1206, 2019.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pp. 489–504, 2018.

Lehman, T. J. and Carey, M. J. A study of index structures for main memory database management systems. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1985.

Marcus, R., Kipf, A., van Renen, A., Stoian, M., Misra, S., Kemper, A., Neumann, T., and Kraska, T. Benchmarking learned indexes. *Proceedings of the VLDB Endowment*, 14(1):1–13, 2020.

Massart, P. The tight constant in the dvoretzky-kiefer-wolfowitz inequality. *The annals of Probability*, pp. 1269–1283, 1990.

Navarro, G. and Rojas-Ledesma, J. Predecessor search. *ACM Computing Surveys (CSUR)*, 53(5):1–35, 2020.

Pătraşcu, M. and Thorup, M. Time-space trade-offs for predecessor search. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pp. 232–240, 2006.

Rao, J. and Ross, K. A. Cache conscious indexing for decision-support in main memory. *Proceedings of the 25th VLDB Conference*, 1999.

Vershynin, R. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge university press, 2018.

A. Proofs

Proof of Lemma 3.4. Assume w.l.o.g that the sub-exponential distribution is centered. Let Z_l be the event that any point in the array is larger than l or smaller than $-l$ for a positive number l. Since the distribution is sub-exponential and using union bound, $\mathbb{P}(Z_l) \leq 2ne^{-lK}$ for some constant K. To have $2ne^{-lK} \leq \frac{1}{\log n}$ we get that $2n\log n \leq e^{lK}$ and $l \geq \frac{1}{K}\log(2n\log n)$. So let $l = \lceil \frac{2}{k}\log 2n \rceil$ and we have $\mathbb{P}(Z_l) \leq \frac{1}{\log n}$. Now, to construct $\hat{R}'$ we first check if any point in A is larger than l or smaller than $-l$. If so, we don't build the index and only do binary search. Otherwise, create $2l$ instances of $\hat{R}$ index, with the i-th index covering the range $[-l+i, -l+i+1]$. Note that interval $[-l+i, -l+i+1]$ has length 1, so that scaling and translating the distribution to interval $[0, 1]$ does not impact the p.d.f of the distribution. Queries use one of the learned models to find the answer. Thus, the query time is $O(\log n)$ with probability $\frac{1}{\log n}$, and it is $t(n)$ with probability $1 - \frac{1}{\log n}$, which is $O(t(n))$. The space overhead is now $O(s(n)\log n)$. □

Proof of Lemma 4.4. Recall that X_1,..., X_n are i.i.d random variables sampled from χ. Furthermore, the array A is a reordering of the random variables so that $A[i] = X_{s_i}$ for some index s_i. That is, each element $A[i]$ is itself a random variable and equal to one of X_1, ..., X_n. $A[i] = a_i$ is a random event.

For $k = j - i - 1 \geq 2$, let $\{X_{r_1}, ..., X_{r_k}\} \subseteq \{X_1, ..., X_n\}$ be the elements of the sub-array $A[i+1 : j-1]$, which we denote by $X_{r_z} \in A[i+1 : j-1]$ for $1 \leq z \leq k$. Note that $X_{r_1}, ..., X_{r_k}$ is not sorted, but the subarray $A[i+1 : j-1]$ is the random variables $X_{r_1}, ..., X_{r_k}$ in a sorted order. For any random variable X_{r_z} for some $z \in \{1, ..., k\}$, we first obtain it's conditional c.f.d given the observations $A[i] = a_i$ and $A[j] = a_j$. The conditional c.d.f can be written as

$$\mathbb{P}(X_{r_z} < x | A[i] = a_i, A[j] = a_j). \qquad (2)$$

Let $\bar{X} = \{X_1, ...X_n\} \setminus \{X_{r_z}\}$. Given that $X_{r_z} \in A[i+1 : j-1]$, the event $A[i] = a_i, A[j] = a_j$, is equivalent to the conjunction of the following events: (i) at most $i-1$ of r.v.s in $\bar{X}$ are less than a_i, (ii) at least i of r.v.s in $\bar{X}$ are less than or equal to a_i, (iii) at most $j-2$ of r.v.s in $\bar{X}$ are less than a_j, (iv) at least $j-1$ of r.v.s in $\bar{X}$ are less than or equal to a_j, and (v) $a_i \leq X_{r_z} \leq a_j$. This is because (i) and (ii) imply $A[i]=a_i$, while (iii)-(v) imply $A[j]=a_j$. Conversely, $A[i]=a_i$ and $X_{r_z} \in A[i+1:j-1]$ imply (i) and (ii), $A[j]=a_j$ and $X_{r_z} \in A[i+1:j-1]$ imply (iii) and (iv), and $A[i]=a_i, A[j]=a_j, X_{r_z} \in A[i+1:j-1]$ imply (v).

Now, denote by $\phi(\bar{X})$ the event described by (i)-(iv), so that Eq. 2 is

$$\mathbb{P}(X_{r_z} < x|\phi(\bar{X}), a_i \leq X_{r_z} \leq a_j).$$

X_{r_z} is independent from all r.v.s in $\bar{X}$, so that Eq. 2 simplifies to

$$\mathbb{P}(X_{r_z} < x|a_i \leq X_{r_z} \leq a_j) = F_{\chi}^{i,j}(x),$$

For all $X_{r_z} \in \{X_{r_1}, ..., X_{r_k}\}$. Thus, the r.v.s in $\{X_{r_1}, ..., X_{r_k}\}$ have the conditional c.d.f $F_{\chi}^{i,j}(x)$.

A similar argument to the above shows that r.v.s in any subset of $X_{r_1}, ..., X_{r_k}$ are independent, given $A[i]=a_i, A[j]=a_j$. Specifically, let $\tilde{X} \subseteq \{X_{r_1}, ..., X_{r_k}\}$ for $|\tilde{X}| \geq 2$. Then, the joint c.d.f of r.v.s in $\tilde{X}$ can be written as

$$P(\forall_{X\in\tilde{X}} X < x|A[i]=a_i, A[j]=a_j). \tag{3}$$

Similar to before, define $\bar{X} = \{X_1, ..., X_n\} \setminus \tilde{X}$. Given that $\forall_{X\in\tilde{X}} X \in A[i+1:i-1]$, the event $A[i]=a_i, A[j]=a_j$ is equivalent to the conjunction of the following events: (i) at most $i-1$ of r.v.s in $\bar{X}$ are less than a_i, (ii) at least i of r.v.s in $\bar{X}$ are less than or equal to a_i, (iii) at most $j-1-|\tilde{X}|$ of r.v.s in $\bar{X}$ are less than a_j, (iv) at least $j-|\tilde{X}|$ of r.v.s in $\bar{X}$ are less than or equal to a_j, and (v) $\forall X \in \tilde{X}$, we have $a_i \leq X \leq a_j$. Now let $\phi(\bar{X})$ be the event described by (i)-(iv), so that Eq. 3 can be written as

$$P(\forall_{X\in\tilde{X}} X < x|\phi(\bar{X}), \forall_{X\in\tilde{X}} a_i \leq X \leq a_j).$$

All r.v.s in $\tilde{X}$ are independent from all r.v.s in $\bar{X}$, so that Eq. 3 simplifies to

$$P(\forall_{X\in\tilde{X}} X < x|\forall_{X\in\tilde{X}} a_i \leq X \leq a_j).$$

Finally, all r.v.s in $\tilde{X}$ are also independent from each other, so we obtain that Eq. 3 is equal to

$$\Pi_{X\in\tilde{X}} P(X < x|a_i \leq X \leq a_j), \tag{4}$$

Proving the independence of r.v.s in $\tilde{X}$ conditioned on $A[i]=a_i, A[j]=a_j$.

To summarize, we have shown that $X_{r_1}, ..., X_{r_k}$ r.v.s conditioned on $A[i]=a_i, A[j]=a_j$ are k i.i.d random variables with the c.d.f $F_{\chi}^{i,j}(x)$. Moreover, $\frac{1}{k} r_A^{i,j}(x)$ is the empirical c.d.f of the k r.v.s. By DKW bound (Massart, 1990) and for $t \geq 0$, we have

$$\mathbb{P}(\sup_x |\frac{1}{k} r_A^{i,j}(x) - F_{\chi}^{i,j}(x)| \geq \frac{t}{\sqrt{k}}) \leq 2e^{-2t^2}.$$

Rearranging and substituting $t = \sqrt{0.5 \log\log k}$ proves the lemma. □

Proof of Lemma 4.6. $k = j-i-1$. Divide the range $[a_i, a_j]$ into t uniformly spaced pieces, so that the z-th piece approximates $kF_{\chi}^{i,j}$ over $I_z = [a_i + z\frac{a_j-a_i}{t}, a_i + (z+1)\frac{a_j-a_i}{t}]$, which is an interval of length $\frac{a_j-a_i}{t}$. Let $P(x)$ be the constant in the Taylor expansion of $kF_{\chi}^{i,j}$ around some point in I_z. By Taylor's remainder theorem,

$$\sup_{x\in I_z} |P(x) - kF_{\chi}^{i,j}(x)| \leq k \times \frac{f_{\chi}(c)}{F_{\chi}(a_j) - F_{\chi}(a_i)} \times \frac{a_j - a_i}{t} \tag{5}$$

for some $c \in I_z$, where we have used the fact that the derivative of the c.d.f is the p.d.f, and that any two point in I_z are at most $\frac{a_j-a_i}{t}$ apart.

By mean value theorem, there exist a $c' \in I_z$ so that $\frac{F_{\chi}(a_j)-F_{\chi}(a_i)}{a_j-a_i} = f_{\chi}(c')$. This together with Eq. 5 yields

$$\sup_{x\in R} |P(x) - kF_{\chi}^{i,j}(x)| \leq k \times \frac{f_{\chi}(c)}{f_{\chi}(c')} \times \frac{1}{t} \leq \frac{\rho_2}{\rho_1}\frac{k}{t}.$$

Setting $t \geq \frac{\rho_2}{\rho_1}\sqrt{k}$ ensures $\frac{\rho_2}{\rho_k}\frac{k}{t} \leq \sqrt{k \log\log k}$ so that

$$\|P(x) - kF_{\chi}^{i,j}(x)\|_{\infty} \leq \sqrt{k \log\log k}$$

□

Proof of Query Time Complexity of Theorem 3.3. The algorithm traverses the tree recursively proceeds. At each recursion level, the algorithm performs a constant number of operations unless it perform a binary search. Let the depth of recursion be h and let k_i be the coverage size of the node at the i-th level of recursion (so that binary search at i-th level takes $O(\log k_i)$). Let B_i denote the event that the algorithm performs binary search at the i-th iteration. Thus, for any query q, the expected number of operations is

$$\mathbb{E}_{A\sim\chi}[T(\hat{r}, q)] = \sum_{i=1}^{h} c_1 + c_2\mathbb{P}(B_i, \bar{B}_{i-1}, ...\bar{B}_1)\log k_i$$

for constants c_1 and c_2. Note that $\mathbb{P}(B_i, \bar{B}_{i-1}, ...\bar{B}_1) \leq \mathbb{P}(B_i|\bar{B}_{i-1}, ...\bar{B}_1)$, where $\mathbb{P}(B_i|\bar{B}_{i-1}, ...\bar{B}_1)$ is the probability that the algorithm reaches i-th level of tree and performs binary search. By Lemma 4.5, this probability bounded by $\frac{1}{\log k_i}$. Thus $\mathbb{E}_{A\sim\chi}[T(\hat{r}, q)]$ is $O(h)$.

To analyze the depth of recursion, recall that at the last level, the size of the array is at most 61. Furthermore, at every iteration the size of the array is reduced by $4(\sqrt{n})(1+\sqrt{0.5\log\log n})$. For $n \geq 61$, $4(\sqrt{n})(1+\sqrt{0.5\log\log n}) \leq n^c$ for some constant $c < 1$, so that the size of the array at the i-th recursions is at most n^{c^i} and the depth of recursion is $O(\log\log n)$. Thus, the expected total time is $O(\log\log n)$ □.

Proof of Prop. 4.2. Note that $r_A(x)$ is a non-decreasing step function, where each step has size 1. Let s_i be the number of steps of $r_A(x)$ in the interval I_i. Therefore,

$$|r_A(x) - r_A(x')| \leq s_i, \tag{6}$$

for any $x, x' \in I_i$. Therefore, for $x \in I_i$, substituting

$\hat{r}(x;\theta) = r_A(i \times \frac{1}{k})$ into Eq. 6 we get

$$e_i \leq s_i. \tag{7}$$

Furthermore, points of discontinuity (i.e., steps) of $r_A(x)$ occur when $x = A[j]$ for $j \in [n]$. Therefore, $s_i = |\{j|A[j] \in I_i\}|$. That is, s_i is equal to the number of points in A that are sampled in the range I_i. □

Proof of Lemma 4.3. Specifically, we bound the probability that $s_{max} \geq c$, for some constant c. In other words, we bound the probability of the event, E, that any interval has more than c points sampled in it, for any $c \geq 3$. Let $\delta_i = F_\chi(\frac{i+1}{k}) - F_\chi(\frac{i}{k})$ be the probability that a point falls inside I_i, so that δ_i^c is the probability that a set of c sampled points fall inside I_i. Taking the union bound over all possible subsets of size c, we get that the probability, p_i, that the i-th interval has c points or more is at most

$$p_i \leq \binom{n}{c}\delta_i^c \leq \frac{(en)^c}{c^c}\delta_i^c,$$

Where the second inequality follows from Sterling's approximation. By mean value theorem, there exists $c \in [\frac{i}{k}, \frac{i+1}{k}]$ such that $F_\chi(\frac{i+1}{k}) - F_\chi(\frac{i}{k}) = f_\chi(c)(\frac{1}{k})$. Therefore, $\delta_i \leq \frac{\rho}{k}$. Thus, by union bound

$$P(E) \leq \sum_{i=1}^{k} \frac{(en)^c}{c^c}(\frac{\rho}{k})^c = k\frac{(en)^c}{c^c}(\frac{\rho}{k})^c. \tag{8}$$

Now set $k \geq n^{1+\frac{2}{c-1}}\rho^{1+\frac{1}{c-1}}$ and substitute into Eq. 8, we obtain that $P(E) \leq \frac{1}{n}$. □