Fast and Near–Optimal Algorithms for Approximating Distributions by Histograms

Jayadev Acharya
MIT
jayadev@csail.mit.edu

Ilias Diakonikolas
University of Edinburgh
ilias.d@ed.ac.uk

Chinmay Hegde
MIT
chinmay@csail.mit.edu

Jerry Li
MIT
jerryzli@csail.mit.edu

Ludwig Schmidt
MIT
ludwigs@mit.edu

ABSTRACT

Histograms are among the most popular structures for the succinct summarization of data in a variety of database applications. In this work, we provide fast and near-optimal algorithms for approximating arbitrary one dimensional data distributions by histograms.

A k-histogram is a piecewise constant function with k pieces. We consider the following natural problem, previously studied by Indyk, Levi, and Rubinfeld [ILR12] in PODS 2012: Given samples from a distribution p over $\{1, \ldots, n\}$, compute a k-histogram that minimizes the ℓ_2-distance from p, up to an additive ε. We design an algorithm for this problem that uses the information–theoretically minimal sample size of $m = O(1/\varepsilon^2)$, runs in sample–linear time $O(m)$, and outputs an $O(k)$– histogram whose ℓ_2-distance from p is at most $O(\mathrm{opt}_k) + \epsilon$, where opt_k is the minimum ℓ_2-distance between p and any k-histogram. Perhaps surprisingly, the sample size and running time of our algorithm are independent of the universe size n.

We generalize our approach to obtain fast algorithms for multi-scale histogram construction, as well as approximation by piecewise polynomial distributions. We experimentally demonstrate one to two orders of magnitude improvement in terms of empirical running times over previous state-of-the-art algorithms.

1. Introduction

In recent years, we have witnessed the proliferation of massive datasets in a variety of scientific and technological domains. Moreover, our inferential goals on the gathered data have become increasingly ambitious. A classical approach to deal with this phenomenon involves constructing *succinct synopses* of the data. In most settings, the use of synopses is essential when exploring massive datasets; see the recent survey [CGHJ12] for an extensive treatment. As expected, a compact representation of a very large dataset may be *lossy* in general. For a given synopsis structure, we are interested in the *space* it requires, the *error* it introduces (quantifying how well it preserves the desired properties of the data), and the *time* needed for its construction and operation. Understanding the precise tradeoff between these criteria has been an important goal in database research and beyond for the last two decades.

In this work, we design efficient algorithms for the compact representation of data distributions by *histograms*. Formally, a k-histogram over the universe $[n]$ is a piecewise constant function with k interval pieces. The parameter k is a natural measure of the space requirement, since a k-histogram can be represented with $O(k)$ numbers. Informally speaking, for a data distribution p over $[n]$, a k-histogram h is a "good" representation of p if $p(i)$ – i.e., the relative frequency of item i in the data – is "close" to $h(i)$. Histograms constitute the oldest and most widely used method for the succinct approximation of data. In the context of databases, they were first proposed in [Koo80], and have since been extensively studied in a variety of applications [GMP97, JKM+98, CMN98, TGIK02, GGI+02, GSW04, GKS06, ILR12].

Our approach can be extended, more generally, to approximating distributions by *piecewise polynomials*. Piecewise polynomial functions are a natural generalization of histograms, where the data distribution in each interval is represented by a degree-d polynomial (i.e., histograms correspond to the special case $d = 0$). A piecewise polynomial with k intervals and degree d is represented by $O(k(d+1))$ numbers, and hence the product $k(d+1)$ is a natural measure of its space requirements. Due to their flexibility, piecewise polynomials can provide an even more succinct approximation to data than histograms and have been previously considered as a synopsis structure in this context (see, e.g., [GKS06]). We note that piecewise polyno-

PODS'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

ACM 978-1-4503-2757-2/15/05 ...$15.00.

mials have received significant attention in other scientific disciplines, including approximation theory [Che66] and statistics (see, e.g., [Sil86, WW83, SHKT97, WN07, CDSS14a, CDSS14b] and references therein).

A common error metric used for fitting a histogram to the data is the "sum squared error", also known as the *V–optimal* measure [IP95]. Here, the goal is to construct a synopsis structure that minimizes the ℓ_2–norm from the underlying data distribution. Most existing works in this area assume that the data distribution p is provided explicitly in the input, and hence the corresponding algorithms inherently incur at least a *linear* dependence on the universe size n. In an early work on this topic, Jagadish *et al.* [JKM$^+$98] provided a dynamic programming algorithm that runs in time $O(kn^2)$ and outputs the best V–optimal histogram of a one-dimensional data distribution over $[n]$. Subsequent results [TGIK02, GGI$^+$02, GKS06] achieved a series of improvements to the running time by focusing on near-optimal solutions. The culmination of this research is the work of Guha *et al.* [GKS06], who gave a $(1+\delta)$ multiplicative approximation to the best V–optimal histogram running in time $O(n + k^3 \log^2 n/\delta^2)$.

When handling massive datasets that range from petabytes to exabytes in size, the requirement that our algorithms read the entire input is unrealistic. A standard way to obtain a small representative subset of the input involves *random sampling* from the data (see [CD14] for a recent tutorial). Ideally, we would like to draw a small number of samples from our data set and efficiently post–process the samples to obtain an accurate representation of the underlying data distribution. Observe that choosing a uniformly random element from a multi-set D over $[n]$ is equivalent to drawing i.i.d. samples from the underlying distribution p of relative frequencies. Hence, the problem of constructing an optimal histogram (or piecewise polynomial) representation under a given metric is essentially equivalent to the problem of *learning* (i.e., performing density estimation) an arbitrary discrete distribution over $[n]$ with respect to this metric.

In recent work, Indyk *et al.* [ILR12] studied the histogram construction problem in the aforementioned random sampling framework. Their main result is an additive ε-approximation algorithm for this problem with running time $O((k^5/\varepsilon^8)\cdot\log^2 n)$. The sample complexity of their algorithm is logarithmic in the domain size n, and quadratic in the number k of histogram pieces. The authors of [ILR12] posed as an open question whether the running time of their algorithm can be improved and whether the logarithmic dependence on n in the sample complexity is necessary.

In this work, we design simple, highly efficient algorithms for approximating data distributions by histograms (and piecewise polynomials) in the ℓ_2-norm using random samples. Our algorithms have information–theoretically optimal sample complexity (independent of the domain size n), run in sample–linear time, and output a histogram (or piecewise polynomial) whose number of intervals and accuracy are within a small constant factor of the best possible. In particular, for the case of histograms, our algorithm runs in time $O(1/\varepsilon^2)$, *independent of both n and k*. In the following section we formally state our results and provide a detailed comparison with prior work.

2. Our Results and Techniques

2.1 Basic Definitions.

We consider distributions over $[n] := \{1, \ldots, n\}$, which are functions $p : [n] \to [0,1]$ such that $\sum_{i=1}^n p(i) = 1$, where $p(i)$ is the probability of element i under p. For convenience, we will use p to denote the distribution with mass function $p(i)$. The ℓ_2-norm of a function $f : [n] \to \mathbb{R}$ is $\|f\|_2 := \sqrt{\sum_{i=1}^n f(i)^2}$. The ℓ_2-distance between functions $f, g : [n] \to \mathbb{R}$ is defined as $\|f - g\|_2$. Let $\mathcal{D}_n$ denote the class of all probability distributions over $[n]$. A function (or distribution) $f : [n] \to \mathbb{R}$ is s-sparse if f is nonzero for at most s points, i.e., $|\{i \in [n] \,|\, f(i) \neq 0\}| \leq s$. For a function f and a set $I \subseteq [n]$, we define f_I as the restriction of f to I; namely, for $i \in I$, we have $f_I(i) := f(i)$, and for $i \notin I$, $f_I(i) := 0$.

Interval Partitions and k-Histograms. Fix a partition of $[n]$ into a set of disjoint intervals $\mathcal{I} = \{I_1, \ldots, I_\ell\}$. For such a partition $\mathcal{I}$ we denote the number of intervals by $|\mathcal{I}|$, i.e., $|\mathcal{I}| = \ell$. For an interval $J \subseteq [n]$, we denote its cardinality or length by $|J|$, i.e., if $J = [a, b]$, with $a \leq b \in [n]$, then $|J| = b - a + 1$. A *k-histogram* is a piecewise constant function $h : [n] \to \mathbb{R}$ with at most k interval pieces, i.e., there exists a partition $\mathcal{I} = \{I_1, \ldots, I_k\}$ of $[n]$ into k intervals I_j, $j \in [k]$, with corresponding values v_j such that $h(i) = v_j$ for all $i \in I_j$. In addition, if h is a probability distribution, then it is referred to as a k-histogram distribution. We use $\mathcal{H}_k^n$ to denote the class of all k-histogram distributions over $[n]$.

Given m independent samples $\{s_j\}_{j=1}^m$ drawn from a distribution $p \in \mathcal{D}_n$, the *empirical distribution* $\widehat{p}_m$ over $[n]$ is the discrete distribution defined as follows: for all $i \in [n]$, $\widehat{p}_m(i) := |\{j \in [m] \mid s_j = i\}|/m$.

Agnostic Histogram ℓ_2–Learning. We cast our histogram construction problem within the framework of distribution estimation. More specifically, we study the following learning problem: Fix parameters $\alpha, \beta \geq 1$. Given m i.i.d. draws from a distribution $p \in \mathcal{D}_n$, and parameters $k \in \mathbb{Z}_+$, $\delta, \varepsilon > 0$, our goal is to output a hypothesis $h \in \mathcal{H}_t^n$ with $t \leq \alpha \cdot k$ such that with probability at least $1 - \delta$ our hypothesis satisfies $\|h - p\|_2 \leq \beta \cdot \mathrm{opt}_k + \varepsilon$, where $\mathrm{opt}_k := \min_{h' \in \mathcal{H}_k^n} \|h' - p\|_2$. That is, the algorithm outputs a t-histogram distribution h whose ℓ_2–error from p is almost as small as the minimum error attainable by any k-histogram.

The approximation factors α and β quantify the performance of the learning algorithm; the former quantifies the number of interval pieces of the constructed histogram (space), while the latter quantifies the achieved accuracy. To measure the complexity, we are interested in the number of samples drawn from p (sample complexity), and the total running time of the algorithm (computational complexity). The "gold standard" in

this context is an algorithm achieving an optimal performance guarantee (i.e., $\alpha = 1$, $\beta = 1$) that uses an information–theoretically minimum sample size and runs in sample–linear time.

2.2 Main Results.

As our main contribution, we design a fast constant–factor approximation algorithm for the aforementioned histogram construction problem. Formally, we obtain:

THEOREM 2.1 (MAIN). *There is an algorithm that, given k, $0 < \varepsilon, \delta < 1$ draws $m = O((1/\varepsilon^2) \cdot \log(1/\delta))$ samples from an arbitrary $p \in \mathcal{D}_n$, runs in time $O(m)$, and with probability at least $1-\delta$ outputs a $5k$-histogram h such that $\|h - p\|_2 \leq 2 \cdot \mathrm{opt}_k + \varepsilon$. Moreover, any algorithm for this problem requires $\Omega((1/\varepsilon^2) \cdot \log(1/\delta))$ samples from p, independent of its running time.*

That is, the algorithm uses an information theoretically minimal sample size (up to a constant), runs in sample–linear time (independent of n), and outputs a histogram representation whose number of intervals and accuracy are within a small constant factor of the best possible.

Our algorithm is simple, easy to implement, and works well in practice. Our experimental evaluation (Section 5) shows that it outperforms previous approaches by at least one order of magnitude. Moreover, our algorithmic approach is quite robust, generalizing effortlessly to *multi-scale* histogram construction, as well as approximation by piecewise polynomial functions. Before we elaborate on these extensions, let us provide a brief explanation of our techniques.

At a high-level, our algorithm can be "decoupled" into two independent stages. In the first stage, we draw $m = O((1/\varepsilon^2) \cdot \log(1/\delta))$ samples from p and construct the empirical distribution $\widehat{p}_m$. In the second stage, we post-process $\widehat{p}_m$ to obtain an $O(k)$-histogram distribution h that approximately minimizes the ℓ_2–distance from $\widehat{p}_m$. While this decoupling approach seems intuitive, its correctness relies crucially on the structure of the ℓ_2–norm.[1] The second stage exploits the $O(m)$–sparsity of the empirical distribution to remove the dependence on n in the running time. We remark that a black-box application of previous algorithms to post-process $\widehat{p}_m$ would lead to super-linear running times (see Section 2.3). We stress that it is by no means obvious how to perform the second stage in time $O(m)$ – i.e., linear in the sparsity of $\widehat{p}_m$ – and this is the main algorithmic contribution of our work.

Our algorithm for implementing the second stage (Algorithm 1 in Section 3.2) is based on an iterative greedy approach. Starting from $\widehat{p}_m$ (itself an $O(m)$-histogram), in each iteration the algorithm merges pairs of consecutive intervals according to a natural notion of error that the merging operation would incur. In particular, it merges all pairs *except* the ones with largest error. The algorithm terminates when the number of remaining intervals is $O(k)$, and the constructed histogram is obtained by "flattening" the empirical distribution with respect to the final set of intervals.

In most applications, the desired value of k (number of intervals in the output histogram) is *a priori* unknown. Instead, the underlying goal is merely to efficiently compute a succinct representation of the data with a few pieces (i.e., small enough value of k) and small error (i.e., small enough value of opt_k). There is a *trade-off* between these two criteria and it is important to design algorithms that capture this tradeoff. A straightforward adaptation of Algorithm 1 can be used to approximately capture the *entire* Pareto curve between k and opt_k, while still running in linear time. In particular, we show (see Section 3.4):

THEOREM 2.2 (MULTI-SCALE HISTOGRAM). *There is an algorithm that, given $0 < \varepsilon, \delta < 1$ draws $m = O((1/\varepsilon^2) \cdot \log(1/\delta))$ samples from an arbitrary $p \in \mathcal{D}_n$, runs in time $O(m)$, and has the following performance guarantee: With probability at least $1-\delta$,* for every k, $1 \leq k \leq m$, *it outputs a t-histogram h_t with $t \leq 8k$ and an error estimate e_t such that (i) $\|h_t - p\|_2 \leq 2 \cdot \mathrm{opt}_k + \varepsilon$, and (ii) $\|h_t - p\|_2 - \varepsilon \leq e_t \leq \|h_t - p\|_2 + \varepsilon$.*

Note that for all $1 \leq k \leq m$ the above algorithm gives us an accurate estimate e_t of the true error $\|h_t - p\|_2$. This is useful for selecting the value of k such that the desired tradeoff between number of intervals and accuracy is achieved.

Finally, we remark that our iterative greedy approach can be generalized to efficiently fit more general classes of functions to the data, such as piecewise polynomials. A (k, d)-piecewise polynomial is a piecewise polynomial function f with k interval pieces $\{I_1, \ldots, I_k\}$ such that f agrees with a degree-d polynomial within each I_j. The piecewise polynomial approximation algorithm is very similar to Algorithm 1. The new technical ingredient is an efficient routine to "project" the data (in a given interval) on the class of degree-d polynomials. We design a fast iterative routine for this subproblem by exploiting properties of discrete Chebyshev polynomials. Specifically, we prove the following:

THEOREM 2.3 (PIECEWISE POLYNOMIALS). *There is an algorithm that, given k, d, and $0 < \varepsilon, \delta < 1$, draws $m = O((1/\varepsilon^2) \cdot \log(1/\delta))$ samples from an arbitrary $p \in \mathcal{D}_n$, runs in time $O(m \cdot (d+1)^2)$, and with probability at least $1 - \delta$ outputs a $(5k, d)$-piecewise polynomial f such that $\|f - p\|_2 \leq 2 \cdot \mathrm{opt}_{k,d} + \varepsilon$, where $\mathrm{opt}_{k,d}$ is the minimum ℓ_2–distance between p and any (k, d)–piecewise polynomial.*

2.3 Comparison with Previous Work.

As mentioned in the introduction, the majority of prior work on histogram construction assumes that the data distribution p is explicitly provided as the input. Jagadish *et al.* [JKM+98] used dynamic programming to compute the best V–Optimal k-histogram of a distribution over $[n]$ in time $O(n^2 k)$. Regarding approximation algorithms, the most relevant reference is the work of Guha *et al.* [GKS06], who gave a $(1+\delta)$-multiplicative approximation that runs in time $O(n + k^3 \log^2 n/\delta^2)$. (See [GKS06] for references on several other approxima-

[1] Notably, such a decoupling into sampling and optimization fails for other metrics, e.g., the ℓ_1–norm [CDSS14a, CDSS14b].

tion algorithms based on alternative approaches, such as wavelet–based techniques.)

[JKM+98] also give a greedy algorithm for the *dual* version of the histogram construction problem. That is, given a bound b on the ℓ_2-error, output a histogram whose error is at most b with the minimum number of pieces k^*. Their algorithm runs in $O(n)$ time and outputs a $3k^*$-histogram with error at most $3b$. We remark that a black-box application of this algorithm to solve the primal problem will necessarily lead to super-linear running times, as it requires an appropriate binary search procedure on the error parameter.

Our sampling stage can be combined with the above approaches to obtain qualitatively better results. In particular, by applying the dynamic programming approach [JKM+98] to the empirical distribution $\widehat{p}_m$, we obtain a k-histogram with error at most $\mathrm{opt}_k + \varepsilon$. This guarantee corresponds to $\alpha = \beta = 1$, which is optimal. The major disadvantage is the resulting running time of $\Theta(m^2 k)$, which is prohibitively large for most applications. Moreover, our experiments indicate that this approach may sometimes result in over-fitting. Similarly, post-processing the empirical distribution using the approximation algorithm of [GKS06] gives a k-histogram with error at most $(1+\delta)\cdot\mathrm{opt}_k + \varepsilon$, i.e., $\alpha = 1$ and $\beta = 1 + \delta$. The running time of this approach is $\Omega(m + k^3 \log^2 m/\delta^2)$.[2] The second term in this expression makes this approach sub-optimal; this is reflected in the experimental comparison in Section 5. Finally, we point out that an adaptation of the dual greedy algorithm in [JKM+98] for m-sparse signals would also lead to a running time super-linear in m. In addition, this approach does not generalize to piecewise polynomial approximation and our experiments show that it empirically performs worse both in terms of approximation ratio and in terms of running time.

[ILR12] studied the agnostic histogram ℓ_2–learning problem in the same framework as ours. Their main result is an algorithm that, given sample access to an arbitrary data distribution p over $[n]$, computes an $O(k\log(1/\varepsilon))$-histogram whose ℓ_2–distance from p is opt_k. That is, they achieve an (α, β) approximation guarantee, where $\alpha = O(\log(1/\varepsilon))$ and $\beta = 1$. Their algorithm has sample complexity $\widetilde{O}((k^2/\varepsilon^4)\log n)$ and running time $\widetilde{O}((k^5/\varepsilon^8)\cdot\log^2 n)$. In comparison, our algorithm in Theorem 2.1 achieves $\alpha = 5$, $\beta = 2$, and has both sample complexity and running time of $O(1/\varepsilon^2)$.

Finally, we mention a different set of histogram approximation algorithms developed in [GSW04] that run in linear time. However, these algorithms provide guarantees for so-called *relative error* measures. As far as we know, none of these approaches imply results for the ℓ_2-learning problem that we consider.

Paper Structure. In Section 3.1 we provide tight upper and lower bounds on the sample complexity of our histogram learning problem. In Section 3.2 we present our main histogram construction algorithm establishing Theorems 2.1 and 2.2. In Section 3.4 we present the generalization of our algorithm to multi-scale histogram construction. In Section 4 we present the generalization of our algorithm to more general functions, including piecewise polynomials. Finally, in Section 5 we describe our experimental evaluation and compare with previous approaches. For the sake of readability, some proofs are deferred to an appendix.

[2]It is unclear whether an adaptation of [GKS06] to m-sparse signals over $[n]$ can lead to running time independent of n.

3. Near-optimal Histogram Approximation

In this section we describe our algorithms for histogram approximation establishing Theorems 2.1 and 2.2.

3.1 Tight Bounds on Sample Complexity.

The following simple theorem establishes the sample upper bound of $O((1/\varepsilon^2)\cdot\log(1/\delta))$ and justifies our two stage learning approach:

THEOREM 3.1. *Let $p \in \mathcal{D}_n$. For any constants $\alpha, \beta \geq 1$ there is an algorithm that, given k and $0 < \varepsilon, \delta < 1$, draws $m = O((1/\varepsilon^2)\cdot\log(1/\delta))$ samples from p and with probability at least $1-\delta$ outputs a hypothesis $h \in \mathcal{H}_t^n$ with $t \leq \alpha \cdot k$ such that $\|h - p\|_2 \leq \beta \cdot \mathrm{opt}_k(p) + \beta\varepsilon$.*

We require the following simple lemma:

LEMMA 3.1. *Fix $0 < \varepsilon, \delta < 1$. Let $p \in \mathcal{D}_n$ and let $\widehat{p}_m$ be the empirical distribution formed by considering $m = \Omega\left((1/\varepsilon^2)\cdot\log(1/\delta)\right)$ samples. Then, with probability at least $1-\delta$, we have that $\|\widehat{p}_m - p\|_2 \leq \varepsilon$.*

PROOF. For $i \in [n]$, let n_i be the number of occurrences of element i among the m samples. Note that $n_i \sim \mathrm{Binom}(m, p(i))$, hence $\mathrm{Var}[n_i] = mp(i)\,(1-p(i))$. Since $\widehat{p}_m(i) = n_i/m$ we can write

$$\begin{aligned}
\mathbb{E}[\|\widehat{p}_m - p\|_2^2] &= \sum_{i=1}^{n} \mathbb{E}\left[(\widehat{p}_m(i) - p(i))^2\right] \\
&= (1/m^2)\cdot\sum_{i=1}^{n} \mathbb{E}\left[(n_i - mp(i))^2\right] \\
&= (1/m^2)\cdot\sum_{i=1}^{n} \mathrm{Var}[n_i] \\
&= (1/m)\cdot\sum_{i=1}^{n} p(i)\cdot(1-p(i)) \\
&< 1/m.
\end{aligned}$$

Consider the random variable $Y = \|\widehat{p}_m - p\|_2$. Jensen's inequality implies that

$$\mathbb{E}[Y] \leq \sqrt{\mathbb{E}[Y^2]} < 1/\sqrt{m} \leq \varepsilon/4.$$

We can write $Y = g(s_1, \ldots, s_m)$, where $s_j \sim p$ and the samples s_j, $j \in [m]$, are mutually independent. Note that the function g satisfies the following Lipschitz property:

$$|\, g(s_1, \ldots, s_j, \ldots, s_m) - g(s_1, \ldots, s_j', \ldots, s_m)\,| \;\leq\; \frac{2}{m}$$

for any $j \in [m]$ and $s_1, \ldots, s_m, s_j' \in [n]$ because changing a single sample moves only $1/m$ of the empirical

probability mass. Hence, McDiarmid's inequality [McD89] implies that

$$\Pr\left[Y > \mathbb{E}[Y] + \eta\right] \leq \exp(-\eta^2 m/2).$$

The lemma follows from our choice of parameters by setting $\eta = \varepsilon/4$. $\square$

PROOF OF THEOREM 3.1. The algorithm proceeds in two stages:

(i) Construct the empirical distribution $\widehat{p}_m$, and

(ii) Compute and output $h \in \mathcal{H}_t^n$ such that

$$\|h - \widehat{p}_m\|_2 \leq \beta \cdot \mathrm{opt}_k(\widehat{p}_m) \ .$$

We now sketch correctness. By Lemma 3.1, with probability at least $1-\delta$ over the samples, we will have that $\|p - \widehat{p}_m\|_2 \leq \varepsilon$. We henceforth condition on this event. Since $\|p - \widehat{p}_m\|_2 \leq \varepsilon$, it follows that $|\mathrm{opt}_k(p) - \mathrm{opt}_k(\widehat{p}_m)| \leq \varepsilon$. The proposition follows by an application of the triangle inequality. $\square$

Theorem 3.1 reduces the histogram construction problem for the distribution p to essentially the same problem for the empirical $\widehat{p}_m$ with an additive loss in the error guarantee. The main algorithmic ingredient lies in efficiently implementing Step (ii). In the following subsection, we design an algorithm (Algorithm 1) that implements the second step in time $O(m)$, while achieving approximation guarantees of $\alpha = 5$ and $\beta = 2$. We remark that an entirely analogous proposition (with an identical proof) holds for constructing piecewise polynomial approximations.

We also prove a matching information-theoretic lower bound of $\Omega((1/\varepsilon^2)\cdot\log(1/\delta))$ for the sample complexity of our learning problem. Our lower bound applies even for the very special case of $k = 2$ and $\mathrm{opt}_k = 0$.

THEOREM 3.2. *Fix $0 < \varepsilon, \delta < 1/2$. Any algorithm with sample access to an arbitrary $p \in \mathcal{D}_n$ that agnostically learns p to ℓ_2-distance ε with probability $1-\delta$ must use $m = \Omega\left((1/\varepsilon^2)\cdot\log(1/\delta)\right)$ samples.*

PROOF. Let p_1 and p_2 be 2-histogram distributions on $[n]$, such that $p_1(1) = 1/2+\varepsilon = p_2(2)$, $p_1(2) = 1/2-\varepsilon = p_2(1)$, and for all $i \in \{3,\ldots,n\}$, $p_1(i) = p_2(i) = 0$. Then, $\|p_1 - p_2\|_2 = 2\sqrt{2}\varepsilon$. We consider the following special case of our agnostic learning problem: We are given sample access to a distribution p over $[n]$ and we are promised that the underlying distribution is either p_1 or p_2. Our goal is to learn the distribution p with probability at least $1-\delta$. We will prove a sample lower bound of $\Omega\left((1/\varepsilon^2)\cdot\log(1/\delta)\right)$ for this problem. Note that, since both p_1 and p_2 are 2-histograms (i.e., $\mathrm{opt}_2 = 0$) with effective support the set $\{1,2\}$, we can assume without loss of generality that our hypothesis is a 2-histogram supported on $\{1,2\}$. Hence, the parameters α and β in the definition of agnostic learning are irrelevant for this special case.

To prove our result, we consider the following (essentially equivalent) hypothesis testing problem. Given m independent samples from $p \in \{p_1, p_2\}$ decide whether the underlying distribution is p_1 or p_2 with error probability at most δ. We call this the problem of *distinguishing* between p_1 and p_2.

Formally, we use this hypothesis testing problem to prove our lower bound. We show:

(a) If p_1 and p_2 cannot be distinguished with m samples with probability at least $1-\delta$, then the ℓ_2 error of any learning algorithm that succeeds with probability $1-\delta$ is at least ε.

(b) p_1 and p_2 cannot be distinguished with probability $1-\delta$ with fewer than $\Omega((1/\varepsilon^2)\cdot\log(1/\delta))$ samples.

Proof of (a)**:** We show the contrapositive. Suppose there is an algorithm $\mathcal{A}$ that with probability at least $1-\delta$ outputs a hypothesis q with ℓ_2 error at most ε. We claim that the following tester distinguishes p_1 and p_2 with probability at least $1-\delta$:

$$\begin{aligned} &p_1, \quad \text{if } \|p_1 - q\|_2 < \|p_2 - q\|_2 \\ &p_2, \quad \text{otherwise.} \end{aligned}$$

Indeed, by the triangle inequality we have that

$$\max\{\|p_1 - q\|_2, \|p_2 - q\|_2\} > \sqrt{2}\varepsilon.$$

Therefore, if the underlying distribution is p_i and q satisfies $\|p_i - q\|_2 < \varepsilon$, the tester correctly distinguishes p_1 and p_2.

Proof of (b)**:** It is a well-known lemma in hypothesis testing (see, e.g., Theorem 4.7, Chapter 4 of [BY02]) that any hypothesis tester that distinguishes between p_1 and p_2 with probability of error δ must use at least

$$\Omega\left(\frac{1}{h^2(p_1,p_2)}\cdot\log(1/\delta)\right)$$

samples, where $h(p_1,p_2)$ denotes the Hellinger distance between p_1 and p_2. Recall that the Hellinger distance between distributions p and q over a finite set $\mathcal{X}$ is defined as [LCY00]

$$h(p,q) \stackrel{\text{def}}{=} \sqrt{\frac{1}{2}\sum_{x\in\mathcal{X}}\left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2}.$$

Therefore, for p_1 and p_2, we have

$$\begin{aligned} h^2(p_1,p_2) &= \frac{1}{2}\sum_{1\leq x\leq n}\left(\sqrt{p_1(x)} - \sqrt{p_2(x)}\right)^2 \\ &= 1 - \sqrt{1-4\varepsilon^2} \\ &= \frac{4\varepsilon^2}{1+\sqrt{1-4\varepsilon^2}} \\ &\leq 2\varepsilon^2, \end{aligned}$$

which completes the proof. $\square$

3.2 Near-optimal Histograms in Input Sparsity Time.

In this subsection, we describe our main algorithm for approximating an arbitrary discrete function with a histogram under the ℓ_2-norm. Our algorithm runs in input sparsity time and achieves approximation error and number of pieces (of the output histogram) that

are both optimal up to constant factors. Formally, we study the following problem: Given an s-sparse function $q : [n] \to \mathbb{R}$ and a parameter k, compute a k-histogram h' such that $\|q - h'\|_2 = \mathrm{opt}_k$. To achieve $O(s)$ running time, we allow the output to consist of up to $O(k)$ pieces and are willing to accept an approximation guarantee of $O(\mathrm{opt}_k)$. Our algorithm will offer a trade-off between the constants hidden in the $O(\cdot)$-notation.

Before introducing the algorithm, let us concern ourselves with the following simple problem: for a single fixed interval I and a function q supported on I, find the value c which minimizes $\sum_{i \in I}(c - q(i))^2$. This corresponds to finding the best 1-histogram approximation to q over I. An easy calculation shows that $c = \frac{1}{|I|}\sum_{i \in I} q(i)$ is the minimizing value. This motivates the following definition.

DEFINITION 3.1 (FLATTENING). *For any interval I, let $\mu_q(I) = \frac{1}{|I|}\sum_{i \in I} q(i)$ be the value of the best 1-histogram approximation to q on I. Furthermore, let $err_q(I) = \sum_{i \in I}(q(i) - \mu_q(I))^2$ denote the ℓ_2-squared error incurred by this approximation. If $\mathcal{I} = \{I_1, \ldots, I_s\}$ is a partition of $[n]$, we let $\bar{q}_{\mathcal{I}} : [n] \to \mathbb{R}$ be the function given by $\bar{q}_{\mathcal{I}}(x) = \mu_q(I_i)$ if $x \in I_i$. We call this function the* flattening *of q over $\mathcal{I}$. By the discussion above, $\bar{q}_{\mathcal{I}}(x)$ is the best fit to q among all functions that are constant on each of the intervals $I_1, \ldots, I_s$.*

Our algorithm works as follows: We start by representing the s-sparse input function q as an $O(s)$ histogram (by assigning a separate interval to each nonzero and each contiguous block of zeros). We call the corresponding set of $O(s)$ intervals $\mathcal{I}^0$. Note that this representation is exact, i.e., $\bar{q}_{\mathcal{I}^0} = q$. Then, we repeatedly perform the following steps: given the current set of intervals $\mathcal{I}^j = \{I_1, \ldots, I_{s_j}\}$, we pair up consecutive intervals and form $I'_u = I_{2u-1} \cup I_{2u}$ for all $1 \le u \le \frac{s_j}{2}$. For each such pair of intervals, we compute the error incurred by I'_u with respect to the input function, i.e., $\mathrm{err}_q(I'_u)$. Using these error values, we find the new intervals I'_u with the $O(k)$ largest $\mathrm{err}_q(I'_u)$. For each interval with large error, we include its two components I_{2u-1} and I_{2u} in $\mathcal{I}^{j+1}$, i.e., we do *not* merge these two intervals in the current iteration. For the remaining intervals (i.e., the ones with smaller error), we include I'_u in $\mathcal{I}^{j+1}$, i.e., we merge the two subintervals. We repeat this process as long as $\mathcal{I}^j$ has more than $O(k)$ intervals. Our final $O(k)$-histogram is the flattening of q over the final set of intervals produced.

At a high level, our algorithm succeeds for the following reason: Let q^* be the optimal k-histogram approximation to q. Intuitively, we accumulate error beyond opt_k when we flatten an interval where q^* has a jump, since on these intervals we diverge from the optimal solution. However, while we may accidentally flatten an interval containing a jump of q^*, this mistake cannot contribute a large error because we only flatten intervals when the resulting error is small compared to the other merging candidates.

For a formal description of our algorithm, see the pseudocode given in Algorithm 1. In addition to the parameter k, our algorithm has two additional parameters that quantify trade-offs between the different objectives: (i) The parameter δ controls the trade-off between the approximation ratio achieved by our algorithm and the number of pieces in the output histogram. (ii) The parameter γ controls the trade-off between the running time and the number of pieces in the output histogram. For details regarding the parameters, see the analysis in the following subsection.

3.3 Analysis of Algorithm 1.

We begin our analysis by establishing the desired approximation guarantee.

THEOREM 3.3. *Let $q : [n] \to \mathbb{R}$ be an s-sparse function. Moreover, let $\mathcal{I}$ be the partition of $[n]$ returned by* CONSTRUCTHISTOGRAM(q, k, δ, γ). *Then*

$$|\mathcal{I}| \le \left(2 + \frac{2}{\delta}\right)k + \gamma \quad \text{and} \quad \|\bar{q}_{\mathcal{I}} - q\|_2 \le \sqrt{1+\delta}\cdot\mathrm{opt}_k ,$$

where $\mathrm{opt}_k = min\|q' - q\|_2$ and q' ranges over all k-histograms.

PROOF. Let $\mathcal{I} = \{I_1, \ldots, I_{k'}\}$ be the partition of $[n]$ returned by CONSTRUCTHISTOGRAM. By construction, we have that $k' \le (2 + \frac{2}{\delta})k + \gamma$. Furthermore, let q^* be a k-histogram with optimal error, i.e., $\|q^* - q\|_2 = \mathrm{opt}_k$. We denote the corresponding partition of $[n]$ with $\mathcal{I}^* = \{I^*_1, \ldots, I^*_k\}$.

Note that we can decompose the error incurred by $\bar{q}_{\mathcal{I}}$ into the error on the individual intervals:

$$\|\bar{q}_{\mathcal{I}} - q\|_2^2 = \sum_{i=1}^{k'} \|(\bar{q}_{\mathcal{I}})_{I_i} - q_{I_i}\|_2^2 = \sum_{i=1}^{k'} \mathrm{err}_q(I_i) .$$

This enables us to analyze the error $\|\bar{q}_{\mathcal{I}} - q\|_2$ in two parts: (i) the intervals I_i which are fully contained in an interval of the optimal k-histogram I^*_j, and (ii) the intervals I_i that intersect at least two intervals I^*_j and I^*_{j+1}. The intervals in case (i) do not contain a "jump" of the optimal histogram q^* and hence our approximation on those intervals is at least as good as q^*. On the other hand, the intervals in case (ii) contain a jump of q^*, but we can bound the error from these intervals because our algorithm does not merge the intervals with large errors. We separately analyze the two cases.

Case (i). Let F be the set of intervals output by our algorithm that do not contain a "jump" of the best k-histogram q^*, i.e., $F = \{i \in [k'] \mid I_i \subseteq I^*_j \text{ for some } j \in [k]\}$. For each $i \in F$, we have defined the value of $\bar{q}_{\mathcal{I}}$ on I_i so that $\|(\bar{q}_{\mathcal{I}})_{I_i} - q_{I_i}\|_2$ is minimized among 1-histograms on I_i. Since q^* is also a 1-histogram on I_i, we have $\|(\bar{q}_{\mathcal{I}})_{I_i} - q_{I_i}\|_2 \le \|q^*_{I_i} - q_{I_i}\|_2$. Summing over all $i \in F$ gives

$$\begin{aligned}\sum_{i \in F} \|(\bar{q}_{\mathcal{I}})_{I_i} - q_{I_i}\|_2^2 &\le \sum_{i \in F} \|q^*_{I_i} - q_{I_i}\|_2^2 \\ &\le \|q^* - q\|_2^2 \\ &\le \mathrm{opt}_k^2 . \end{aligned} \tag{1}$$

Case (ii). We now consider the set of intervals containing a jump of the k-histogram q^*, i.e., $J = \{i \in [k'] \mid I_i \not\subseteq I^*_j \text{ for all } j \in [k]\}$. Since q^* is a k-histogram,

Algorithm 1 Approximating with histograms by merging.

1: **function** ConstructHistogram(q, k, δ, γ)
2: ▷ *We assume that q is given as the sorted set of nonzeros $\{(i_1, y_1), \ldots, (i_s, y_s)\}$ such that $y_j = q_{i_j}$.*
3: $J \leftarrow \bigcup_{j=1}^{j \le s} \{i_j - 1, i_j, i_j + 1\}$
4: ▷ Set of relevant indices.
5: ▷ *Precompute partial sums for fast mean and error computation.*
6: $r_j \leftarrow \sum_{i_u \le j} y_u$ for $j \in J$
7: $t_j \leftarrow \sum_{i_u \le j} y_u^2$ for $j \in J$
8: ▷ *Initial histogram.*
9: Let $\mathcal{I}^0 \leftarrow \{I_1, \ldots, I_{s_0}\}$ be the initial partition of $[n]$ defined by the set J.
10: ▷ *Iterative greedy merging (we start with $j = 0$).*
11: **while** $|\mathcal{I}^j| > (2 + \frac{2}{\delta})k + \gamma$ **do**
12: Let s_j be the current number of intervals.
13: ▷ *Compute the errors for merging neighboring pairs of intervals.*
14: **for** $u \in \{1, 2, \ldots, \frac{s_j}{2}\}$ **do**
15: $e_u \leftarrow \mathrm{err}_q(I_{2u-1} \cup I_{2u})$
16: Let L be the set of u with the $(1 + \frac{1}{\delta})k$ largest e_u and M be the set of the remaining u.
17: $\mathcal{I}^{j+1} \leftarrow \bigcup_{u \in L} \{I_{2u-1}, I_{2u}\}$
18: ▷ Keep the intervals with large merging errors.
19: $\mathcal{I}^{j+1} \leftarrow \mathcal{I}^{j+1} \cup \{I_{2u-1} \cup I_{2u} \,|\, u \in M\}$
20: ▷ Merge the remaining intervals.
21: $j \leftarrow j + 1$
22: **return** $\mathcal{I}^j$

we have $|J| \le k$. For each I_i with $i \in J$, there are two sub-cases: (a) I_i is one of the initial intervals in $\mathcal{I}^0$, (b) I_i was created in some iteration of the algorithm. In case (a), we clearly have $\|(\bar{q}_{\mathcal{I}})_{I_i} - q_{I_i}\|_2 = 0$, so we focus our attention on case (b).

Consider the iteration in which I_i was created. Since I_i is the result of merging two smaller intervals, the error $\mathrm{err}_q(I_i)$ was not among the $(1 + \frac{1}{\delta})k$ largest errors. Let L be the set of indices with the largest errors, so $\mathrm{err}_q(I_i) \le \mathrm{err}_q(I_j)$ for all $j \in L$ (see line 16 of Alg. 1). At most k of the intervals in L can contain a jump of q^*, so at least $\frac{k}{\delta}$ intervals in L are contained in an interval in $\mathcal{I}^*$. Let L' be this set of intervals, i.e., $L' = \{j \in L \,|\, I_j \subseteq I_u^* \text{ for some } u \in [k]\}$. Using a similar optimality argument as in Case (i), we have that

$$\sum_{j \in L'} \mathrm{err}_q(I_j) \;=\; \sum_{j \in L'} \|(\bar{q}_{\mathcal{I}})_{I_j} - q_{I_j}\|_2^2 \;\le\; \mathrm{opt}_k^2 \,.$$

Since $|L'| \ge \frac{k}{\delta}$, this implies that

$$\min_{j \in L'} \mathrm{err}_q(I_j) \le \frac{\delta}{k} \mathrm{opt}_k^2 \,.$$

Therefore, we have that $\mathrm{err}_q(I_i) \le \frac{\delta}{k}\mathrm{opt}_k^2$ because $\mathrm{err}_q(I_i)$ was not among the largest errors. Hence we have:

$$\sum_{i \in J} \|(\bar{q}_{\mathcal{I}})_{I_i} - q_{I_i}\|_2^2 \;\le\; |J| \cdot \frac{\delta}{k}\mathrm{opt}_k^2 \;\le\; \delta \cdot \mathrm{opt}_k^2 \,. \quad (2)$$

Since $F \cup J = [k']$, we can now combine Eq. (1) and Eq. (2) to get our final approximation guarantee:

$$\begin{aligned} \|\bar{q}_{\mathcal{I}} - q\|_2^2 \;&=\; \sum_{i \in F} \|(\bar{q}_{\mathcal{I}})_{I_i} - q_{I_i}\|_2^2 + \sum_{i \in J} \|(\bar{q}_{\mathcal{I}})_{I_i} - q_{I_i}\|_2^2 \\ &\le\; \mathrm{opt}_k^2 + \delta \cdot \mathrm{opt}_k^2 \;=\; (1 + \delta) \cdot \mathrm{opt}_k^2 \,. \end{aligned}$$

□

Next, we consider the running time of our algorithm. Intuitively, each iteration of the merging algorithm takes time linear in the current number of intervals by using the precomputed partial sums (see line 7 of Alg. 1) and a linear-time selection algorithm [CSRL01, p. 189]. Moreover, every iteration of the merging algorithm reduces the number of intervals by a constant factor as long as the number of intervals is larger than $(1 + \frac{1}{\delta})k$. Therefore, the total running time of our algorithm is bounded by the time complexity of the first iteration, which is $O(s)$. We show this more formally below.

Theorem 3.4. *Let $q : [n] \to \mathbb{R}$ be an s-sparse function. Then* ConstructHistogram(q, k, δ, γ) *runs in time*

$$O\left(s \;+\; k\left(1 + \frac{1}{\delta}\right)\log\left(\frac{(1 + 1/\delta)k}{\gamma}\right)\right) \,.$$

Proof. Consider J, r_j, and t_j as defined in ConstructHistogram. Clearly, all three quantities can be computed in $O(s)$ time. Moreover, precomputing the partial sums allows us to compute $\mathrm{err}_q(I)$ in constant time if the endpoints of I are contained in J. In particular, let $I = \{a, a + 1, \ldots, b\}$ with $a, b \in J$. Then

$$\begin{aligned} \mathrm{err}_q(I) &= \sum_{i \in I} q(i)^2 - \frac{1}{|I|}\left(\sum_{i \in I} q(i)\right)^2 \\ &= t_b - t_a + y_a^2 - \frac{1}{b - a + 1}(r_b - r_a + y_a)^2 \,, \end{aligned}$$

which we can evaluate in constant time. Since our algorithm only constructs intervals with endpoints in the

set J, this allows us to quickly evaluate all instances of err_q over the course of the algorithm.

Next, we show that for all j, the $(j+1)$-th iteration of the loop in ConstructHistogram can be performed in time $O(s_j)$ (recall $s_j = |\mathcal{I}^j|$ is the number of intervals active in the j-th iteration). As outlined above, the e_u can be computed in $O(s_j)$ total time. Moreover, we can find the set L containing the $(1+1/\delta)k$-th largest e_u in linear time: first, we find the $(1+1/\delta)k$-th largest single e_u using a linear time selection algorithm [CSRL01, p. 189]. Then, we can find all elements in L in one further pass over the e_u, checking whether the current e_u is larger than the threshold value of the $(1+1/\delta)k$-th largest single e_u. After this, we can easily create the set $\mathcal{I}^{j+1}$ in linear time and hence the entire iteration can be performed in time $O(s_j)$.

Moreover, for all $j \geq 1$ until we terminate, we have that

$$s_{j+1} \;=\; \frac{s_j - (2+2/\delta)k}{2} + \left(2+\frac{2}{\delta}\right)k \;=\; \frac{s_j + (2+2/\delta)k}{2}\,. \tag{3}$$

The first equality follows by construction, since in each iteration of the loop, we keep $(2+\frac{2}{\delta})k$ intervals from the previous iteration and merge the remaining pairs into one new interval per pair. Starting with $s_0 = s$, and applying Equation 3 j times, we obtain that for all j until the algorithm terminates,

$$s_j = 2^{-j}s + (1-2^{-j})\left(2+\frac{2}{\delta}\right)k\,. \tag{4}$$

In particular, this implies that the algorithm terminates after at most $\lambda = \log(s/\gamma)$ iterations. Let $\lambda' = \log((2+2/\delta)k+s) - \log((2+2/\delta)k)$. By a straightforward manipulation, we see that for all $j \leq \lambda'$, we have $2^{-j}s \geq (1-2^{-j})(2+2/\delta)k$, and for all $j > \lambda'$, the opposite direction holds. By the arguments above, it follows that the total runtime of ConstructHistogram(q,k,δ,γ) is $O(\sum_{j=1}^{\lambda} s_j)$. The proof follows from the following sequence of inequalities:

$$\begin{aligned}
\sum_{j=0}^{\lambda} s_j \;&=\; \sum_{j=0}^{\lambda'} s_j + \sum_{j=\lambda'+1}^{\lambda} s_j \\
&\leq\; 2\sum_{j=0}^{\lambda'} 2^{-j}s \;+\; 2\sum_{j=\lambda'+1}^{\lambda}(1-2^{-j})\left(2+\frac{2}{\delta}\right)k \\
&\leq 4s + (\lambda-\lambda')\left(2+\frac{2}{\delta}\right)k \\
&\leq 4s + \log\left(\frac{(2+2/\delta)k}{\gamma}\right)\left(2+\frac{2}{\delta}\right)k\,, \qquad (5)
\end{aligned}$$

where the last line is obtained by expanding the expression for $\lambda - \lambda'$ and using the fact $(2+2/\delta)k \geq 0$. □

Thus, we have:

Corollary 3.1. *For any constant $c > 0$,* ConstructHistogram$(q,k,\delta,c(2+2/\delta)k)$ *runs in time $O(s)$ and returns a $(1+c)(2+2/\delta)k$-histogram.*

Proof. Following the analysis of Theorem 3.4, when we plug in $\gamma = c(2+2/\delta)k$ into Equation 5, we get that

$$\sum_{j=0}^{\lambda} s_j \;\leq\; 4s + \log\left(\frac{1}{c}\right)\left(2+\frac{2}{\delta}\right)k.$$

We can assume that $(2+2/\delta)k \leq s$ since otherwise the input $\mathcal{I}^0$ is already $(2+2/\delta)k$-flat. Hence, we have $\sum_{j=0}^{\lambda} s_j = O(s)$ which completes the proof. □

With the parameterization in Corollary 3.1, the algorithm runs in linear time for *all* values of k. The reason for this parameterization is that the merging algorithm makes increasingly slower progress as the number of intervals decreases (and hence the fraction of intervals which are not merged increases). In fact, for the regime $k = O(\frac{s}{\log s})$, the algorithm runs in time $O(s)$ for any $\gamma \geq 1$.

3.4 Multi-scale Histogram Construction.

Our previously described algorithm (Algorithm 1) requires a priori knowledge of the parameter k, i.e., the desired number of intervals in the output histogram approximation. We show that a variant of Algorithm 1, which we call ConstructHierarchicalHistogram, works *without* knowledge of k and produces good histogram approximations for all values of $k \geq 1$. The guarantees for the running time, number of histogram intervals, and approximation error are similar to those of Algorithm 1.

As before, our algorithm works for arbitrary s-sparse functions. We start with a histogram consisting of $O(s)$ intervals and reduce the number of intervals by a fourth in each iteration of the algorithm. At any stage, suppose the intervals are $I_1, \ldots, I_{s'}$. As before, we pair the intervals as I_{2i+1}, I_{2i} and compute the approximation errors of the merged intervals. We keep *half* of the pairs corresponding to the *largest* errors and merge the remaining $\frac{s'}{2}$ pairs of intervals. This reduces the number of intervals from s' to $\frac{3s'}{4}$. A detailed pseudocode for this variant of the merging algorithm is given in Alg. 2. We have:

Theorem 3.5. *Let $q : [n] \to \mathbb{R}$ be an s-sparse function. Moreover, let $\mathcal{I}^0, \mathcal{I}^1, \ldots, \mathcal{I}^L$ be the partitions of $[n]$ returned by* ConstructHierarchicalHistogram(q). *Then given any $1 \leq k \leq s$, there is an $\mathcal{I}^j$ with $|\mathcal{I}^j| \leq 8k$ and*

$$\|\bar{q}_{\mathcal{I}^j} - q\|_2 \;\leq\; 2 \cdot \mathrm{opt}_k\,,$$

where $\mathrm{opt}_k = \min \|q' - q\|_2$ and q' ranges over all k-histograms. Moreover, the algorithm runs in $O(s)$ time.

We remark that a *single* run of Algorithm 2 produces a hierarchical histogram that achieves a constant-factor approximation guarantee for *any* target number of histogram pieces.

Proof of Theorem 3.5. We first prove the claimed running time of our algorithm. In each iteration, the number of intervals reduces by a fourth, and the number of initial intervals is $O(m)$. Moreover, we can output $\mathcal{I}^j$

Algorithm 2 Learning histograms by hierarchical merging.

function ConstructHierarchicalHistogram(q)
  ▷ *We assume that q is given as the sorted set of nonzeros $\{(i_1, y_1), \ldots, (i_s, y_s)\}$ such that $y_j = q_{i_j}$.*
  ▷ *Set of relevant indices.*
  $J \leftarrow \bigcup_{j=1}^{j \le s} \{i_j - 1, i_j, i_j + 1\}$
  ▷ *Initial histogram.*
  Let $\mathcal{I}^0 \leftarrow \{I_1, \ldots, I_{s_0}\}$ be the initial partition of $[n]$ defined by the set J.
  ▷ *Iterative greedy merging.*
  $j \leftarrow 0$
  while $|\mathcal{I}^j| \geq 8$ **do**
    Let s_j be the current number of intervals.
    ▷ *Compute the errors for merging neighboring pairs of intervals.*
    for $u \in \{1, 2, \ldots, \frac{s_j}{2}\}$ **do**
      $e_u \leftarrow \mathrm{err}_q(I_{2u-1} \cup I_{2u})$
    Let L be the set of u with the $\frac{s_j}{4}$ largest e_u and M be the set of the remaining u.
    ▷ *Keep the intervals with the large merging errors.*
    $\mathcal{I}^{j+1} \leftarrow \bigcup_{u \in L} \{I_{2u-1}, I_{2u}\}$
    ▷ *Merge the remaining intervals.*
    $\mathcal{I}^{j+1} \leftarrow \mathcal{I}^{j+1} \cup \{I_{2u-1} \cup I_{2u} \mid u \in M\}$
    $j \leftarrow j + 1$
  return $\mathcal{I}^0, \mathcal{I}^1, \ldots, \mathcal{I}^j$

in each iteration in time linear in the current number of intervals. Therefore, the total running time satisfies,

$$T(s) = T\left(\frac{3s}{4}\right) + O(s).$$

We start with m steps, and therefore the algorithm runs in time $O(m)$.

We now turn our attention to the approximation guarantee. Suppose we want to match the approximation error of a k-histogram. Consider the iteration of our algorithm in which the number of intervals drops below $8k$ for the first time. We output the corresponding partition $\mathcal{I}^j$, which by definition satisifies $|I_j| \leq 8k$. For the approximation error, we follow the proof of Theorem 3.3: we consider the set of intervals that contain a "jump" of an optimal k-histogram approximation q^*. Since there were at least $8k$ intervals before this iteration, we form $4k$ pairs of intervals and do not merge at least $2k$ pairs of intervals with the largest errors. At least k of those intervals do not contain a jump and thus contribute an error of at most opt_k. This also holds for any previous iteration of the algorithm. Therefore, each interval containing a jump contributes at most $\frac{\mathrm{opt}_k}{k}$ error, and the total contribution of all intervals containing a jump is at most opt_k. Combined with the contribution from intervals containing no jump (also at most opt_k in total), the final approximation error is $2 \cdot \mathrm{opt}_k$. □

4. A Generalized Merging Algorithm and Fitting Piecewise Polynomials

In this section we adapt Algorithm 1 to work for more general classes of functions, in particular, piecewise polynomials. These results establish Theorem 2.3.

4.1 An Oracle Version of Algorithm 1.

Note that Algorithm 1 relies only on the following property of piecewise constant functions: given an interval I and a function f, we can efficiently find the constant function with minimal ℓ_2 error to f on I. We can extend this idea to fit more general types of functions (for example, the class of piecewise polynomials.) We formalize this intuition in the following definitions.

Definition 4.1 (Projection Oracle). *Let $\mathcal{F}$ be any set of functions from $[n]$ to $\mathbb{R}$. $\mathcal{O}$ is a* projection oracle *for $\mathcal{F}$ if it takes as input an interval I and a function $f : I \to \mathbb{R}$, and outputs a function $g^* \in \mathcal{F}$ and a value v such that*

$$v = \|g_I^* - f\|_2 \leq \inf_{g \in \mathcal{F}_I} \|g_I - f\|_2 .$$

Definition 4.2. *A function $f : [n] \to \mathbb{R}$ is a k-piecewise $\mathcal{F}$-function if there exists a partition of $[n]$ into k disjoint intervals $\{I_1, \ldots, I_k\}$ so that for each i, $f_{I_i} = (g_i)_{I_i}$, for some $g_i \in \mathcal{F}$.*

For example, suppose $\mathcal{F}$ is the set of all constant functions on $[n]$. Then there is a trivial projection oracle $\mathcal{O}$ for $\mathcal{F}$: for any function $f : I \to \mathbb{R}$, the optimal flat approximator to f is the function which is constantly $\mu_f(I)$ (as we argued in Section 3.2). The error of this approximation is $\mathrm{err}_f(I)$ and is also easy to calculate. Thus, one way of describing Algorithm 1 is as follows:

given input function q, in each iteration, we compute the best estimator for q from $\mathcal{F}$ using $\mathcal{O}$ on each merged interval and its associated error. Subsequently, we merge all intervals except those with top $O(k)$ errors, and repeat the process.

The idea of the generalized merging algorithm is the following. For general classes of functions $\mathcal{F}$, at each iteration, we first call the corresponding projection oracle $\mathcal{O}$ over consecutive pairs of intervals to find the best fit in $\mathcal{F}$ over this larger interval. We also calculate the ℓ_2-error of the approximation using $\mathcal{O}$. As in Algorithm 1, we merge all intervals except those with large errors. This general algorithm, which we call ConstructGeneralHistogram$(q, k, \delta, \gamma, \mathcal{O})$, is conceptually similar to Algorithm 1 except that we use a projection oracle instead of a flattening step. Assuming the existence of such an oracle, we can formally state our main result for approximation with k-piecewise $\mathcal{F}$-functions.

Theorem 4.1. *Let $q : [n] \to \mathbb{R}$ be s-sparse. Let $\mathcal{F}$ be any set of functions on $[n]$ to $\mathbb{R}$ as above, and let $\mathcal{O}$ be a projection oracle for $\mathcal{F}$ which, given any s'-sparse function q' on a subinterval of I, runs in time $O(\alpha s')$. Then* ConstructGeneralHistogram$(q, k, \delta, \gamma, \mathcal{O})$ *runs in time*

$$O\left(\alpha\left(s \;+\; k\left(1+\frac{1}{\delta}\right)\log\left(\frac{(1+1/\delta)k}{\gamma}\right)\right)\right)$$

and outputs a k'-piecewise $\mathcal{F}$-function $f : [n] \to \mathbb{R}$ where

$$k' \le \left(2+\frac{2}{\delta}\right)k + \gamma$$

and which satisfies $\|f - q\|_2 \le (1+\delta)\cdot\|f' - q\|_2$ where f' ranges over all k-piecewise $\mathcal{F}$-functions.

The proof of Theorem 4.1 is a simple adaptation of the proof of Theorem 3.3.

4.2 Finding the best fit polynomial on a subinterval.

We now specialize Theorem 4.1 to piecewise polynomial approximation. Let us first restate this problem in the terminology introduced above. Fix an interval $I = [a, b] \subseteq [n]$. It will be convenient to represent functions $w : I \to \mathbb{R}$ as vectors $v = (v_1, \ldots, v_{b-a+1})$, where $v_i = w(a + i - 1)$. For a fixed interval I, the vector representation of a function is uniquely defined. Hence, throughout this section we will blur the distinction between a function and its vector representation.

We define $\mathcal{P}_d(I)$ to be the set of $v \in \mathbb{R}^{|I|}$ such that there exists a degree-d polynomial $p : \mathbb{R} \to \mathbb{R}$ such that for all $i \in I$, $v_i = p(a + i - 1)$, and $\mathcal{P}_d = \mathcal{P}_d([n])$. We construct an efficient procedure FitPoly$_d(I, q)$ (see Algorithm 3 in the appendix) that takes as input an interval $I \subseteq [n]$ and a function $q : I \to \mathbb{R}$, and finds the best approximation of the function within the interval with a degree-d polynomial. In particular, we will prove:

Theorem 4.2. *There exists a projection oracle* FitPoly$_d$ *for $\mathcal{P}_d$, which, given an interval I and an s-sparse function $q : I \to \mathbb{R}$, runs in time $O(d^2 s)$.*

Theorems 4.1 and 4.2 give our desired result, which we now state for completeness:

Corollary 4.1. *Let $q : [n] \to \mathbb{R}$ be s-sparse. Then,* ConstructGeneralHistogram$(q, k, \delta, \gamma,$ FitPoly$_d)$ *runs in time*

$$O\left(d^2\left(s \;+\; k\left(1+\frac{1}{\delta}\right)\log\left(\frac{(1+1/\delta)k}{\gamma}\right)\right)\right),$$

and outputs a t-piecewise degree-d polynomial function $f : I \to R$, such that $t \le \left(2+\frac{2}{\delta}\right)k + \gamma$ and $\|f - q\|_2 \le (1+\delta)\cdot\|f' - q\|_2$ where f' ranges over all k-piecewise degree-d polynomials.

Note that, as in Corollary 3.1, for a natural choice of parameters δ and γ, the running time becomes $O(d^2 s)$. It remains to prove Theorem 4.2, which we defer to the appendix. The core of algorithm is a fast projection onto the space of degree-d polynomials via the orthonormal basis of discrete Chebyshev polynomials [Sze89]. The key technical ingredient is a subroutine which evaluates these basis polynomials at s points in time $O(d^2 s)$. This compares favorably to the $O(d^\omega s)$ time algorithm given in [GKS06] (where ω is the matrix multiplication constant).

5. Experimental Evaluation

In order to evaluate the empirical performance of our algorithm, we conduct several experiments on real and synthetic data. As with our analysis, we split the empirical evaluation into two parts: "offline" histogram approximation (where we assume that the full data distribution is explicitly present), and histogram approximation from sampled data. All experiments in this section were conducted on a laptop computer from 2010, using an Intel Core i7 CPU with 2.66 GHz clock frequency, 4 MB of cache, and 8 GB of RAM. We used the Debian GNU/Linux distribution and `g++` 4.8 as compiler with the `-O3` flag (all algorithms were implemented in C++). All reported running times are averaged over at least 10 trials (and up to 10^4 trials for algorithms with a fast running time).

5.1 Histogram approximation.

Due to the large amount of prior work on histogram approximation, we focus on the most relevant algorithms in our comparison. Moreover, we study the performance in the case of *dense* input, i.e., we make no assumptions about the number of nonzeros, since most prior algorithms work in this setting. We denote the size of the dense input with n. Note that our merging algorithm (Alg. 1) is directly applicable to dense inputs, which can be represented as an n-sparse function. We run the following histogram approximation schemes:

`exactdp` An exact dynamic program with running time $O(n^2 k)$ [JKM+98].

`dual` A variant of the linear time algorithm for the dual problem given in [JKM+98]. Since the target error is not known in the primal version of the problem, this variant incurs an extra logarithmic factor due to a binary search over opt.

`merging` Our algorithm as outlined in Alg. 1. We use parameters $\delta = 1{,}000$ and $\gamma = 1.0$ so that our algorithm produces a histogram with $2k+1$ pieces.

`merging2` The same algorithm as `merging`, but using $k' = \frac{k}{2}$ as input. Hence the resulting histogram has $k+1$ pieces.

`fastmerging` A variant of Alg. 1 with a more aggressive merging scheme that merges larger groups of intervals in the early iterations.[3] We also use $\delta = 1{,}000$ and $\gamma = 1.0$ for fastmerging.

`fastmerging2` The same algorithm as `fastmerging`, but using $k' = \frac{k}{2}$ as input. Similar to `merging2`, the resulting histogram has $k+1$ pieces.

We run the algorithms on two synthetic and one real-world data set with input sizes ranging from $n = 1{,}000$ to 16,384 (see Fig. 1). We have chosen the same real-world data set as in [GKS06] so that our results are comparable to the results given there. For the `hist` and `poly` data sets, we run the algorithms with $k = 10$. For the `dow'` dataset, we use $k = 50$.

Table 1 contains detailed results for our experiments. Our algorithms are several orders of magnitude faster than the exact dynamic program. Interestingly, our algorithms also achieve a very good approximation ratio. Note that for `merging` and `fastmerging`, the approximation ratio is *better* than 1.0 on the `poly` and `dow` data sets. The reason is that the algorithms produce histograms with $2k+1$ pieces, which makes a better approximation than that achieved by `exactdp` possible. In particular, the empirical results are significantly better than those predicted by our theoretical analysis, which only guarantees a very large constant factor for $\delta = 1{,}000$. Further, both the `merging2` and `fastmerging2` variants still achieve a good approximation ratio while using only $k+1$ pieces in the output histogram.

Compared to `dual`, all variants of our algorithm achieve a better approximation ratio (by a factor close to 1.75 on the `dow` data set). Moreover, the `fastmerging` variants are roughly ten times faster than the `dual` algorithm on the two larger data sets.

The fastest algorithm in [GKS06] is AHIST-L-Δ, which achieves an approximation ratio of about 1.003 for the `dow` data set with $n = 16{,}384$ and $k = 50$. This approximation ratio is better than `merging2` and `fastmerging2`, but worse than `merging` and `fastmerging` (note however that AHIST-L-Δ uses exactly k pieces in the output histogram). All of our algorithms are significantly faster: the reported running time of AHIST-L-Δ on `dow` is larger than one second, which is more than 1,000 times larger than the running time of our algorithms. While the running times are not directly comparable due to different CPUs and compilers, a comparison of running times of `exactdp` in our experiments with those in [GKS06] shows that our speedup due to CPU and compiler is around 3×. Even assuming a generous 10×, our algorithms are still more than two orders of magnitude faster than AHIST-L-Δ.

[3]One can show that with a more aggressive merging, the `fastmerging` algorithm performs only $O(\log \log n)$ rounds of merging, as opposed to $O(\log n)$ as in the "binary" merging algorithm given in Alg. 1. However, the total running time is determined by the first round of merging and remains $O(n)$.

5.2 Histogram approximation from samples.

In addition to offline histogram approximation, we also study the performance of our algorithms for the histogram learning problem. We use the same data sets as in Figure 1, but normalize them to form a probability distribution. Due to the slow running time of `exactdp`, we also subsample the `poly` and `dow` data sets by a factor of 4 and 16 respectively (using uniformly spaced samples) so that all data sets have a support of size roughly 1,000. We call the resulting data sets `hist'`, `poly'`, and `dow'`, and use the same values of k as in the offline histogram approximation experiments.

Figure 2 shows the results of the histogram learning experiments. In order to reduce clutter in the presentation of our results, we only perform sampling experiments with the `exactdp`, `merging`, and `merging2` algorithms. Our histogram approximation algorithms demonstrate very good empirical performance and often achieve a better approximation error than `exactdp`. The results indicate that the additional time spent on constructing an exact histogram fit to the empirical distribution does not lead to a better approximation to the true underlying distribution.

Acknowledgements.

The authors would like to thank Piotr Indyk for helpful discussions.

I.D. was supported by a Marie Curie CIG, EPSRC grant EP/L021749/1, and a SICSA grant. J.A., C.H., and L.S. were supported by an MIT-Shell energy initiative. J.L. was supported by NSF grant CCF-1217921 and DOE grant DE-SC0008923.

6. References

[BY02] Z. Bar-Yossef. *The Complexity of Massive Data Set Computations*. PhD thesis, University of California, Berkeley, 2002.

[CD14] G. Cormode and N. Duffield. Sampling for big data: A tutorial. In *ACM KDD*, 2014.

[CDSS14a] S. Chan, I. Diakonikolas, R. Servedio, and X. Sun. Efficient density estimation via piecewise polynomial approximation. In *STOC*, pages 604–613, 2014.

[CDSS14b] S. Chan, I. Diakonikolas, R. Servedio, and X. Sun. Near-optimal density estimation in near-linear time using variable-width histograms. In *NIPS*, pages 1844–1852, 2014.

[CGHJ12] G. Cormode, M. Garofalakis, P. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 2012.

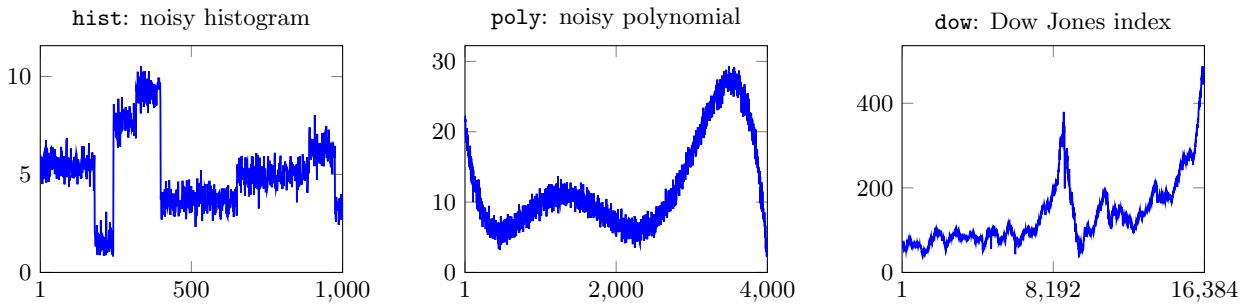

Figure 1: Data sets for the offline histogram approximation experiments. (left) `hist` is a histogram consisting of 10 pieces contaminated with Gaussian noise ($n = 1000$). (middle) `poly` is a degree 5 polynomial, also contaminated with Gaussian noise ($n = 4000$). (right) `dow` is a time series of Dow-Jones Industrial Average (DJIA) daily closing values ($n = 16384$).

		exactdp	merging	merging2	fastmerging	fastmerging2	dual
hist	Error (ℓ_2)	16.1	16.4	16.6	17.0	21.5	25.8
	Error (relative)	**1.00**	**1.02**	**1.03**	**1.06**	**1.34**	**1.60**
	Time (milliseconds)	55.391	0.038	0.037	0.020	0.014	0.108
	Time (relative)	**3,910**	**2.7**	**2.6**	**1.4**	**1.0**	**7.6**
poly	Error (ℓ_2)	105.1	85.9	111.6	85.6	111.7	124.0
	Error (relative)	**1.00**	**0.82**	**1.06**	**0.81**	**1.06**	**1.18**
	Time (milliseconds)	858.064	0.112	0.112	0.048	0.041	0.446
	Time (relative)	**20,924**	**2.7**	**2.7**	**1.2**	**1.0**	**10.9**
dow	Error (ℓ_2)	904.0	733.1	1,046.1	727.5	1,079.1	1,838.1
	Error (relative)	**1.00**	**0.81**	**1.16**	**0.80**	**1.19**	**2.03**
	Time (milliseconds)	73576.921	0.510	0.478	0.205	0.173	1.849
	Time (relative)	**425,540**	**3.0**	**2.8**	**1.2**	**1.0**	**10.7**

Table 1: Results of the algorithms on the three data sets in Fig. 1. The relative errors are reported as ratios compared to the error achieved by `exactdp`. For the relative running times, the baseline is `fastmerging2`.

[Che66] E. Cheney. *Introduction to Approximation Theory*. McGraw-Hill, New York, New York, 1966.

[CMN98] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? In *ACM SIGMOD*, 1998.

[CSRL01] T. Cormen, C. Stein, R. Rivest, and C. Leiserson. *Introduction to Algorithms*. 2nd edition, 2001.

[GGI+02] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *STOC*, 2002.

[GKS06] S. Guha, N. Koudas, and K. Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. Database Syst.*, 2006.

[GMP97] P. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *VLDB*, 1997.

[GSW04] S. Guha, K. Shim, and J. Woo. REHIST: Relative error histogram construction algorithms. In *VLDB*, 2004.

[ILR12] P. Indyk, R. Levi, and R. Rubinfeld. Approximating and Testing k-Histogram Distributions in Sub-linear Time. In *PODS*, 2012. Corrected version available as ECCC TR11-171.

[IP95] Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *ACM SIGMOD*, 1995.

[JKM+98] H. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, 1998.

[Koo80] R. Kooi. *The Optimization of Queries in Relational Databases*. PhD thesis, 1980.

[LCY00] L. Le Cam and G. L. Yang. *Asymptotics in statistics: some basic concepts*. Springer, 2000.

[McD89] C. McDiarmid. On the method of bounded differences. In *Surveys in Combinatorics 1989*, pages 148–188. London Mathematical Society Lecture Notes, 1989.

[SHKT97] C. J. Stone, M. H. Hansen, C. Kooperberg, and Y. K. Truong. Polynomial splines and

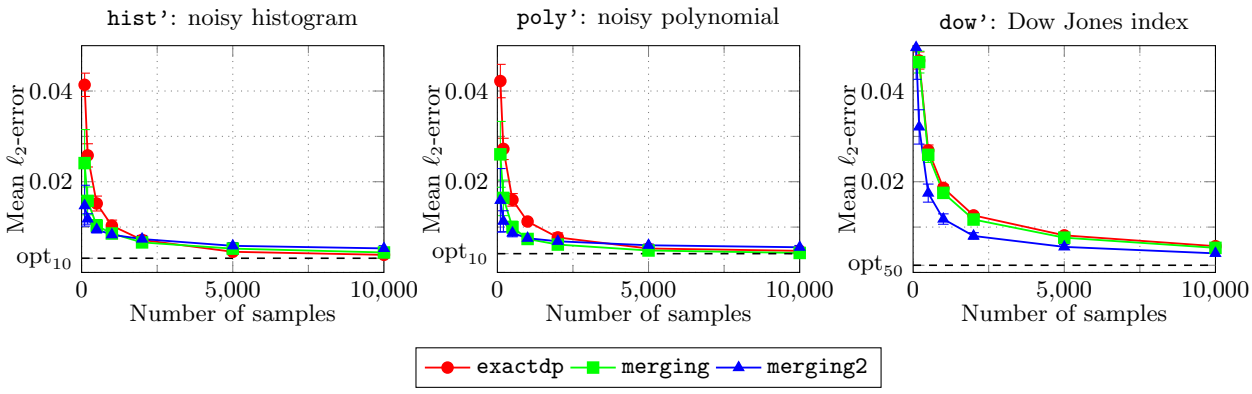

Figure 2: Resuls of the histogram learning experiments for three different data sets (see Figure 1). Every data point is the mean error over 20 trials and the error bars indicate one standard deviation. The opt_k values are the approximation error of the best k-histogram fit to the underlying distribution.

their tensor products in extended linear modeling: 1994 wald memorial lecture. *Ann. Statist.*, 25(4):1371–1470, 1997.

[Sil86] B. Silverman. *Density Estimation.* Chapman and Hall, London, 1986.

[Sze89] G. Szegö. *Orthogonal Polynomials*, volume XXIII of *American Mathematical Society Colloquium Publications*. A.M.S, 1989.

[TGIK02] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *ACM SIGMOD*, 2002.

[WN07] R. Willett and R. Nowak. Multiscale poisson intensity and density estimation. *IEEE Trans. Info. Theory*, 2007.

[WW83] E. J. Wegman and I. W. Wright. Splines in statistics. *Journal of the American Statistical Association*, 78(382):pp. 351–365, 1983.

APPENDIX

A. Proofs and Description of FitPoly$_d$

We now describe FitPoly$_d$, and prove Theorem 4.2. In particular, we show how to perform the projection of q onto $\mathcal{P}_d(I)$ in time $O(d^2 s)$, where s is the sparsity of q.

Fix an interval $[a,b] = I \subseteq [n]$. Then, $\mathcal{P}_d(I)$, the class of all polynomials of degree at most d, is a $(d+1)$-dimensional linear subspace spanned by the vectors $x_0, \ldots, x_d$ where x_j is the vector associated with the function $f_j(x) = x^j$, that is, $x_j = (a^j, (a+1)^j, \ldots, b^j)$. By the linear transform $x \to x - a$, each x_j can be written as a linear combination of the vectors $y_j = (0^j, 1^j, 2^j, \ldots, (|I|-1)^j)$. We henceforth assume that the interval of interest is $I = [0, b-1]$.

Now the problem of finding the best-fit polynomial to q becomes equivalent to finding $v \in \mathcal{P}_d(I)$ that minimizes $\|v - q\|_2^2$. This is exactly the problem of finding the projection of q onto the vector space $\mathcal{P}_d(I)$. We do so by appealing to the theory of discrete orthogonal polynomials.

Definition A.1. *The* discrete Chebyshev, *or* Gram *polynomial of degree r for the interval $[0, \ldots, b-1]$ is the polynomial given by*

$$t_r(y) = \frac{1}{W_r} r! \cdot \Delta^r \left(\binom{y}{r} \binom{y-b}{r} \right) \tag{6}$$

where Δ is the forward difference operator on functions given by $\Delta f(x) \stackrel{def}{=} f(x+1) - f(x)$, $\binom{x}{r}$ is the generalized binomial coefficient, and

$$W_r = \frac{b \cdot (b^2-1)^{r^2}}{(2r+1)}$$

where $y^{\underline{r}} \stackrel{def}{=} y(y-1)\ldots(y-r+1)$ denotes the rth falling power.

It is well known [Sze89] that $\{t_r\}_{r=0}^d$ forms an orthonormal basis for $\mathcal{P}$. Thus, for any function q, the projection of q onto $\mathcal{P}_d(I)$ is given by $q(x) = \sum_{r=0}^d a_r t_r(x)$ where

$$a_r = \sum_{i=0}^{b-1} q(i) \cdot t_r(i), \tag{7}$$

and by Parseval's theorem, the error of the projection is given by $\|q\|_2^2 - \sum_{r=0}^d a_r^2$. FitPoly$_d$ computes a_r and the error using these formulas. The formal pseudocode is given in Algorithm 3.

The crucial subroutine is EvaluateGram(x,d,b), which compute the values $t_r(x)$ for $r = 0, \ldots, d$, for any fixed $x \in [0, b-1]$, in time $O(d^2)$. The formal pseudocode for EvaluateGram(x,d,b) is given in Algorithm 4.

Lemma A.1. EvaluateGram(x,d,b) *computes $t_0(x), \ldots, t_d(x)$ in time $O(d^2)$.*

Proof. EvaluateGram(x,d,b) computes the quantities $t_0(x), \ldots, t_d(x)$ as follows. For fixed r, we can evaluate $\binom{y}{r}$ for $y = x, x+1, \ldots, x+r$ in time $O(d)$ since $\binom{y}{r} = y^{\underline{r}}/r!$. We can evaluate $r!$ and $x^{\underline{r}}$, for all $r \le d$, in time $O(d)$. Moreover, given $y^{\underline{r}}$, in constant time we can compute $(y+1)^{\underline{r}} = (y+1) \cdot y^{\underline{r}}/(y-r+1)$.

Algorithm 3 Projecting an s-sparse vector on to $\mathcal{P}_d([0, b-1])$.

function FITPOLY$_d(q, b)$
  ▷ *We assume that q is given as the sorted set of nonzeros $\{(i_1, y_1), \ldots, (i_s, y_s)\}$ such that $y_j = q_{i_j}$.*
  ▷ *Set of relevant indices.*
  ▷ *Compute values of $t_r(i)$ for $i = i_1, \ldots, i_s$*
  $v_i \leftarrow$ EVALUATEGRAM(i, d, b) for $i = i_1, \ldots, i_s$
  ▷ *Compute inner product of q with t_r for all $r = 0, \ldots, d$*
  for $r = 0, \ldots, d$ **do**
    $a_r \leftarrow \sum_{j=1}^{s} y_j v_{i_j}(r)$
  ▷ *Compute the error of the projection*
  $\text{err} = \sum_{j=1}^{s} y_j^2 - \sum_{r=0}^{d} a_r^2$
  return $([a_0, \ldots, a_r], \text{err})$

Thus we can evaluate $r!$ and $y^{\underline{r}}$ in time $O(d)$ for $y = x, x+1, \ldots, x+r$, and $\binom{y}{r}$ for $y = x, x+1, \ldots, x+r$ and for all r in time $O(d^2)$. Similarly, we can find $\binom{y-b}{r}$ for $y = x, x+1, \ldots, x+r$ and for all r in time $O(d^2)$. It is straightforward to compute all the $(b^2-1)^{\underline{r^2}}$ in time $O(d^2)$ from the formula, and thus we can compute all the W_r in time $O(d^2)$ as well.

Therefore, we can compute the values of $\nu_r(y) := \binom{y}{r}\binom{y-b}{r}$ at $y = x, \ldots, x+r$, $r!$, and W_r for all r simultaneously in total time $O(d^2)$. Through similar methods as those described above, we can also compute $\binom{r}{j}$ for all $0 \le r \le d$ and for all $0 \le j \le r$ in time $O(d^2)$. Given that we know these values, we conclude that we can evaluate $t_r(x)$ in time $O(d^2)$, since for any x, we have

$$\begin{aligned} t_r(x) &= \frac{1}{W_r} r! \cdot \Delta^r\left(\nu_r(x)\right) \\ &= \frac{1}{W_r} r! \sum_{j=0}^{r} (-1)^j \binom{r}{j} \nu_r(x + r - i). \end{aligned}$$

This concludes the proof. □

We are now in a position to prove Theorem 4.2.

THEOREM 4.2. *There exists a projection oracle* FITPOLY$_d$ *for $\mathcal{P}_d$, which, given an interval I and an s-sparse function $q : I \to \mathbb{R}$, runs in time $O(d^2 s)$.*

PROOF. We first run EVALUATE-GRAM(i_j, d, b) for $j = 1, \ldots, s$. We now have access to the values $t_r(i_j)$ for all $r = 0, \ldots, d$ and all $j = 1, \ldots, d$. Given these values, computing the a_r for the original function q is simple: if q is supported on points $i_1, \ldots, i_s$, we evaluate $t_r(i_j)$, for all $0 \le r \le d$ and for all $j = 1, \ldots, s$. We then apply Equation 7, $a_r = \sum_{j=1}^{s} q(i_j) t_r(i_j)$, and compute the error using Parseval's theorem as described above. This procedure takes time $O(d^2 s)$. □

Algorithm 4 EVALUATEGRAM(x, d, b) returns v_x, a function on $0, \ldots, d$ so that $v_x(r) = t_r(x)$, where t_r is the Gram polynomial of degree r on the interval $[0, b-1]$ as defined in Equation 6. Throughout the description of the algorithm, since we have to evaluate certain mathematical expressions that depend on b and r, we will denote by $\bar{a}$ any value a which we have computed.

function EVALUATEGRAM(x, d, b)
  ▷ *Evaluate* W_r *for* $r = 0, \ldots, d$
  $\overline{(b^2-1)^{\underline{0}}} \leftarrow 1$
  for $r = 1, \ldots, d$ **do**
    $\overline{(b^2-1)^{\underline{r^2}}} \leftarrow \overline{(b^2-1)^{\underline{(r-1)^2}}} \cdot \prod_{j=(r-1)^2+1}^{r} (b2 - 1 - j)$
  for $r = 0, \ldots, d$ **do**
    $W_r \leftarrow \frac{b \cdot \overline{(b^2-1)^{\underline{r^2}}}}{(2r+1)}$
  ▷ *Evaluate* $r!$ *for* $r = 0, \ldots, d$
  $\overline{0!} \leftarrow 1$
  for $r = 1, \ldots, d$ **do**
    $\overline{r!} \leftarrow r \cdot \overline{(r-1)!}$
  ▷ *Evaluate* $\binom{r}{j}$ *for* $r = 0, \ldots, d$ *and* $j \leq r$
  for $r = 0, \ldots, d$ **do**
    $\overline{\binom{r}{0}} \leftarrow 1$
    $\overline{\binom{r}{r}} \leftarrow 1$
    for $j = 1, \ldots, r-1$ **do**
      $\overline{\binom{r}{j}} \leftarrow \overline{\binom{r-1}{j}} + \overline{\binom{r-1}{j-1}}$
  ▷ *Evaluate* $\nu_r(i)$ *for* $r = 0, \ldots, d$ *and for all* $i = 0, \ldots, r$
  for $r = 0, \ldots, d$ **do**
    ▷ *Evaluate* $\binom{y}{r}$ *for all* $y = x, \ldots, x + r$
    $\overline{\binom{x}{r}} \leftarrow x^{\underline{r}}/r!$
    for $i = 1, \ldots, r$ **do**
      $\overline{\binom{x+i}{r}} \leftarrow (x + i + 1) \cdot \overline{\binom{x+i}{r}}/(x + i - r + 1)$
    ▷ *Evaluate* $\binom{y-b}{r}$ *for all* $y = x, \ldots, x + r$
    $\overline{\binom{x-b}{r}} \leftarrow (x-b)^{\underline{r}}/r!$
    for $i = 1, \ldots, r$ **do**
      $\overline{\binom{x-b+i+1}{r}} \leftarrow (x - b + i + 1) \cdot \overline{\binom{x-b+i}{r}}/(x - b + i - r + 1)$
    for $i = 1, \ldots, r$ **do**
      $\overline{t_r(i)} \leftarrow \overline{\binom{x+i}{r}} \cdot \overline{\binom{x-b}{r}}$
  ▷ *Evaluate* $\nu_x(r) = t_r(x)$ *for* $r = 0, \ldots, d$
  for $r = 0, \ldots, d$ **do**
    $\nu_x(r) \leftarrow \frac{1}{W_r} r! \sum_{j=0}^{r} (-1)^j \overline{\binom{r}{j}} \cdot \overline{t_r(x + r - i)}$
  return ν_x