Topics in Massive Data Summarization

by

Xuan Zheng

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2008

Doctoral Committee:

Assistant Professor Martin J Strauss, Chair
Professor Hosagrahar V Jagadish
Associate Professor Kevin J Compton
Associate Professor Anna Catherine Gilbert
Associate Professor Jignesh M Patel

TABLE OF CONTENTS

LIST OF FIGURES

Figure

CHAPTER I

Introduction

The goal of this thesis is to find time- and space- efficient algorithms for massive data sets. In recent years, input sizes in many problems have grown to the point where "polynomial computational overhead" is too coarse a measure.

For streaming data, we need to build data summaries in sub-linear time and space, with constant update time, and support queries with guaranteed quality in constant time. In this thesis, we focus on building a specific class of data summary—histograms. In a streaming data model, the underlying data distribution is too large to be stored, and we do not have access to previous data entries. We discuss two different models. In the first one, there is a corresponding weight vector associated with the input streaming data vector, which is motivated by the fact that people can be more interested in some data entries than the others. In the second one, each data entry is a density function instead of a deterministic value, which is motivated by the uncertainty in the real life.

For data summaries built on data from more than one party, we need to minimize both computation and communication. For example, we may require a bound on communication that is polylogarithmic in input length in some application with large inputs. In this thesis, privacy and result quality are also addressed. In this

model, every party p can communicate via a private channel with any other in the system directly. All information p gets from other parties during the computation, including p's coin flips, is called p's view. We want p's view to be uncorrelated with others' inputs for every party p in the system to enforce privacy.

1.1 Motivation

In this thesis, we focus on summarization of massive data sets, whose underlying distribution are often too large to be stored precisely.

A histogram is one kind of summary, used as a small-space, approximate synopsis. A histogram is a piecewise-constant approximation of an observed data distribution. Histograms have found many applications in database management systems, perhaps most commonly for query selectivity estimation in query optimizers [28]; that is, given a query P, we need to estimate the fraction of records in the database that satisfy P [40]. Many commercial database systems maintain histograms to summarize the contents of relations and permit efficient estimation of query result sizes and access plan costs, especially for multiple-relation queries [47], [45]. In [19], authors discussed the histogram application about selectivity estimation in probabilistic graphical models.

It also found applications in approximate query answering [2], load balancing in parallel join execution [44], mining time-series data [33], partition-based temporal join execution, query profiling for user feedback, etc. Ioannidis has a nice overview of the history of histograms, their applications, and their use in commercial DBMSs [27]. Also, Poosala's thesis provides a systematic treatment of different types of histograms [44]. Formally:

Definition 1.1. A B-bucket *histogram* h of length N is a partition of $[0, N)$ into

intervals $[b_0, b_1) \cup [b_1, b_2) \cup \cdots \cup [b_{B-1}, b_B)$, where $b_0 = 0$ and $b_B = N$, together with a collection of B heights h_j, for $0 \leq j < B$, one for each bucket. A point query c_i to h returns the estimate h_j where $b_j \leq i < b_{j+1}$.

In building a B-bucket histogram, we want to choose $B-1$ boundaries b_j and B heights h_j, dependent on the data vector c. A number of different criteria are known [44] for choosing b_j's and h_j's; a popular and effective one is the *V-Opt histogram* [29], where b_j's and h_j's are chosen to minimize the total square error, taken *uniformly over the set of all point queries*, or, equivalently, $\|c-h\|^2 = \sum_i (c_i - h_{j(i)})^2$. (Once we have chosen the boundaries, the best bucket height on an interval I is the average of c over I.)

Section 4.2 provides an example of our motivation. In a network, one server gets continuous web log data from its clients. It needs to keep track of the field "number of bytes" in each log data to detect abnormal behavior, allocate resources, or simply understand its clients' requirements better. In this specific application, the underlaying log data is too large to be stored, so we need to store a summary like a histogram. Also, the log data comes continuously and no backtracking is allowed, so the server can only read each item once and should act quickly. In the above application, we say that the log data is streamed. Formally, we consider a vector c given either as a stream $c_0, c_1, \cdots$ of aggregate values, or in the more general dynamic maintenance model, as a stream $(i_0, x_0), (i_1, x_1), \cdots$ of updates, where (i_j, x_j) is interpreted as a request to update c_{i_j} to $c_{i_j} + x_j$. When processing c_j or (i_j, x_j), the algorithm has no access to the data seen before. We want to build histogram over c.

Our motivation for Chapter III lies in the applications, where the *workload* of queries should be take into account for which the V-Opt histogram is optimized. In

particular, when some of the point queries are more frequent than the others, the histogram needs to be better at approximating answers to the frequent queries rather than the infrequent ones. In other words, the metric to minimize is not the sum of squared errors *uniformly* over all point queries, but that obtained by weighting the error on each point query by the workload of how frequently each point query is posed. Formally:

Definition 1.2. Given an input signal $c_0 \cdots c_{N-1}$ and *workload* $w_0 \cdots w_{N-1}$, $0 \leq w_i$, the *workload-optimal* B-bucket histogram h_{opt} is the choice of b_j's and h_j's that minimize $\|c - h\|_w^2 = \sum_i \ w_i(c_i - h_{j(i)})^2$.

We consider the problem of finding h_{opt} with respect to workload on streamed signals, as well as stored or streamed workloads. Formally, we consider a vector c and the corresponding workload w in one of the following two forms:

- c comes as a stream $c_0, c_1, \cdots$, or in the dynamic maintenance model, and the algorithm has access only to the current data. The weight vector w is stored and the algorithm has access to every w_i at any time. In practice, many different c's may follow the same workload w. Storing only w is reasonable in such applications.

- c and w come as a stream $(c_0, w_0), (c_1, w_1), \cdots$, and the algorithm has access only to the current data. We call this lockstep model.

Heavy hitters is another kind of summarization. For example, the most searched words is a useful summary of daily search words for search engines. Finding heavy hitters or most frequent items is a basic statistic on a database relationship. It provides a useful measurement of the data distribution and can help improve performance of selectivity estimation. It finds application in other areas like data mining

and networking management as well. For example, keeping track of the heaviest traffic can help a server better know about its clients' demand and improve its service. It can also be used in anomaly detection by keeping track of the most dramatically changed workload in a network.

In many of the motivating applications above, the underlying data are distributed over two or more parties, each of which is reluctant to reveal its own data to other parts unless necessary. In such cases, we require secure and private multiparty computation. This has been studied for several decades, starting with [50, 7]. Any protocol for computing a function of several inputs can be converted, gate-by-gate, to a *private* protocol, in which no party learns anything from the protocol messages other than what can be deduced from the function's input/output relation. The computational overhead is at most polynomial in the size of the inputs.

As input sizes grow, however, we need to minimize both communication and computation overhead. For example, absent privacy concerns, applications may require that a protocol uses at most polylogarithmic communication. General-purpose secure multiparty computation may blow up communication exponentially, so additional techniques are needed. In one theoretical approach, individual protocols are designed for functions of interest such as database lookup (the *private information retrieval* problem [12, 37, 8]) and building decision trees [38]. Another approach, the breakthrough [42], converts any protocol into a private one with little communication blowup. But unfortunately, approach [42] imposes a computational blowup that may be exponential.

The approach we follow, which was introduced in [15], is to substitute an approximate function for the desired function. Many functions of interest have good approximations that can be computed efficiently both in terms of computation and

communication. A caveat is that the traditional definition of privacy is no longer appropriate. Instead, a protocol π computing an approximation $\widetilde{f}$ to a function f is a private approximation protocol [15] for f if

- π is a private protocol for $\widetilde{f}$ in the traditional sense that the messages of π leak nothing beyond what is implied by inputs and $\widetilde{f}$, *and*,
- the output $\widetilde{f}$ leaks nothing beyond what is implied by inputs and f.

Several examples were given in [15] and more details will be presented in Chapter II. In Chapter V, we build a private protocol, finding heavy hitters on two parties.

1.2 Related Work

1.2.1 Workload-aware algorithms

The database community has proposed methods to synopsize data distributions, taking workload into account. Query feedback from the execution engine of a DBMS was used in [11] to modify the synopsis. Histogram boundaries are refined adaptively in [34, 1, 46] based on a dynamically evolving workload that is continuously updated based on feedback from the query engine; they differ in how they approximate values within buckets, how they weight the workload etc. Still, these methods do not give any provable results on approximating h_{opt}. There has been some work on *other* synopses that are workload-aware. For example, [18] proposed sampling methods that adapt to recent workload. IBM's LEO optimizer [49] uses workload information for a variety of synopses. In [41], a $O(N^2B/\log B)$ time algorithm is presented for determining the optimal choice of B Haar wavelet terms; this has recently been improved to $O(N^2)$ time [22]. The Haar basis is modified in [39] with the knowledge of the workload and algorithms for obtaining B-term synopses are designed for this new basis; while this algorithm works in linear time, it does not provide a near-optimal

B-term Haar wavelet synopsis. For special workloads, [41] presented a near-linear algorithm for finding the optimal B-term Haar wavelet synopsis. All of these results for Haar and related bases [41, 22, 39] work only when *both* the signal and workload are available in a stored form without any loss of information, and not streamed with polylogarithmic space. However, when both the signal and workload are stored explicitly without loss of information, the dynamic programming from [30] immediately gives an $O(N^2B)$ time algorithm for finding the optimal h_{opt}, so the challenge in [41, 22] arises from working with the Haar wavelet basis and does not reflect on the difficulty in constructing h_{opt}. In [24], there are many results of the same flavor as our result—indeed, the expanded version of [24] contains many generalizations not considered here—but the results of [24] do not address directly our time- and space-bounded, workload-aware problem with the bounds we give. To summarize, the *significant open problem with finding h_{opt} is when either the signal or the workload is streamed or both are streamed, with space polylogarithmic in N.*

The result given in [23] is crucial to our work in Chapter III. It approximates h_{opt} in sub-linear computation time and space, with constant update time per item. Details will be presented in Chapter III.

There are many papers that address the Heavy Hitters problem and sketching in general, in a variety of contexts. Many of the needed ideas can be seen in [35] and other important papers include [5, 4, 20, 14]. We will talk about this in more detail in section 2.1.

1.2.2 Private Approximations

As to private communication-efficient protocols, there is work for specific functions including the Private Information Retrieval problem [12, 37, 8], building decision trees [38], set intersection and matching [16], and k'th-ranked element [3].

The breakthrough [42] gives a general technique for converting any protocol into a private protocol with little communication overhead. It is not the end of the story, however, because the computation may increase exponentially.

Work in private approximations include [15] that introduced the notion as a conference paper in 2001 and gave several protocols. Some negative results were given in [25] for approximations to NP-Hard functions; more on NP-hard search problems appears in [6]. Recently, [26] gives a private approximation to the Euclidean norm that is central to our paper, which will be introduced in Chapter V.

Statistical work such as [10] also addresses the privacy of algorithms on massive data sets, but the goals are significantly different than ours.

1.3 Specific Contributions

1.3.1 Workload-aware histograms

Our contributions on histograms are as follows. Suppose the data items are integers, and the weights are positive integers between the minimum weight, $w_{\min}$, and the maximum weight, $w_{\max}$. Let $M = \max\{||A||^2, \frac{w_{\max}}{w_{\min}}\}$ be a bound on the range of data and weights. $U = \sum_{i=0}^{N-1} w_i$ and let c_1 and c_2 denote constants.

Workload w is stored without loss of information. We present an $O(N+\text{poly}(B, \log N, \log M, 1/\epsilon))$-time algorithm to compute a B-bucket histogram h with $\|c - h\|_w^2 \leq (1+\epsilon)\|c - h_{\text{opt}}\|_w^2$ where h_{opt} is the workload-optimal B-bucket histogram, with respect to arbitrary w. This is the first near-linear[1] time algorithm for approximating h_{opt} under non-uniform workloads. The above algorithm can be run in the time series model taking only $O(1)$ time per new item and using $\text{poly}(B, \log N, \log M, 1/\epsilon)$ space and post-processing time to construct the $(1+\epsilon)$-approximate histogram. Under the

[1] Note that, for moderate values of the parameters other than N, the run time is dominated by $O(N)$. In this dissertation, we use the term "near-linear" for this type of cost.

dynamic maintenance model, the above algorithm can be modified using previously known techniques so that the time per update, total space used, and postprocessing time are all poly$(B, \log N, \log M, 1/\epsilon)$. This is the first known set of algorithms that use sublinear—polynomial in B, $1/\epsilon$ and polylogarithmic in N, M—space for dealing with data stream signals and yet yields $(1+\epsilon)$ approximate h_{opt} histograms for any w. It matches the previously known bounds for the special case when the workload is uniform [20].

Workload w is streamed. In the lockstep model, we are given a stream $(c_0, w_0), (c_1, w_1), \ldots$ of data items, together with their associated workload weights, in order. An algorithm is run in the lockstep model taking $c_1 N \log U + \left(\frac{B \log U \log M}{\epsilon}\right)^{c_2}$ time, and using $\left(\frac{B \log U \log M}{\epsilon}\right)^{O(1)}$ space to construct the $(1+\epsilon)$-approximate histogram, where $U = \sum_{i=0}^{N-1} w_i$.

1.3.2 Implementation

Our contributions on implementation are as follows.

- We built the first non-trivial statistical tool for declarative data description language PADS . Basically, PADS takes structured or semi-structured streamed data as input, and first converts them into PADS types. Then, users can get statistics over interesting fields by using our tools. More details can be found in Chapter IV.

- We built a uniform interface, so that further statistical tools can be incorporated into the system by specifying only a high-level algorithm.

1.3.3 Private Protocols

Our contributions to private protocols are as follows. Suppose there are two parties, each holding a vector, a and b. They want a summary for the vector sum $c = a + b$.

Euclidean approximate heavy hitters problem. First, we consider the problem in which there is a parameter, B, and the players ideally want c_{opt}, the B largest terms in c, *i.e.*, the B biggest values together with the corresponding indices. Unfortunately, finding c_{opt} exactly requires linear communication. Instead, the players use polylogarithmic communication (and polynomial work and $O(1)$ rounds) to output a vector $\widetilde{c}$ with $\|\widetilde{c} - c\|_2 \leq (1+\epsilon)\|c_{\text{opt}} - c\|_2$. In our protocol, the players learn nothing more than what can be deduced from c_{opt} and $\|c\|_2$.

Leaking the Euclidean norm represents a weaker result than not leaking the Euclidean norm, but (i) leaking $\|c\|_2$ is necessary in some circumstances and (ii) computing or approximating $\|c\|_2$ is desirable in some circumstances. First, we give a straightforward lower bound showing that, for some (reasonable) values of parameters $M, N, \ldots$, computing $\widetilde{c}$ leaking only c_{opt} requires $\Omega(N)$ communication. In fact, for some (artificial) classes of inputs, $\Omega(N)$ communication is needed unless $\|c\|_2$ itself is not only potentially leaked, but actually computed exactly. On the other hand, one can regard the Euclidean norm as semantically interesting, so that we can regard the top B terms *together with the Euclidean norm* as a compound, extended summary. In particular, since $\widetilde{c}$ is computed, leaking $\|c\|_2$ is equivalent to leaking $\|c\|_2^2 - \|\widetilde{c}\|_2^2 = \|\widetilde{c} - c\|_2^2$, *i.e.*, the error in our representation, which is a useful and common thing to compute. Our protocol indeed can be modified to output an approximation $\|\widetilde{c} - c\|_\sim$ with $\|\widetilde{c} - c\|_2 \leq \|\widetilde{c} - c\|_\sim \leq (1+\epsilon)\|\widetilde{c} - c\|_2$, so we can regard

the protocol as solving two cascaded approximation problems: find a near-best representation $\widetilde{c}$, then find an approximation $\|\widetilde{c}-c\|_{\sim}$ to $\|\widetilde{c}-c\|_2$. It is natural to expect a protocol for $\widetilde{c}$ to leak c_{opt} and a protocol for $\|\widetilde{c}-c\|_{\sim}$ to leak $\|\widetilde{c}-c\|_2$; while lower bounds prevent that, we can compute $\widetilde{c}$ and $\|\widetilde{c}-c\|_{\sim}$ *simultaneously* and guarantee that, *overall*, we leak only c_{opt} and $\|\widetilde{c}-c\|_2$.

Extension of basic result. We can immediately use our basic result as black box for approximate sparse representations over any orthonormal basis such as wavelet or Fourier, with little additional algorithmic or privacy work. The result says that we provide an at-most-B term Fourier representation that is almost as good (in the Euclidean sense) as the best B-term Fourier representation and leaks no more than the best B-term representation and the Euclidean norm. The Fourier basis may be substituted by any orthonormal basis, such as Hadamard or Wavelet. It demonstrates that the basic result can be applied in a variety of interesting applications.

We can also use the result as a black box for taxicab approximate heavy hitters, *i.e.*, finding $\widetilde{c}$ with $\|\widetilde{c}-c\|_1 \leq (1+\epsilon)\|c_{\text{opt}}-c\|_1$, leaking c_{opt} and $\|c\|_2$. Thus we have shown that the private Euclidean norm approximation can be used for non-Euclidean problems.

1.4 Overview

In Chapter II, we review sketch algorithms. The idea is to compress the original data into a much smaller data structure, while preserving some important properties. The compressed data structure can be used to evaluate functions over the original data approximately (with quality guarantee). We use this idea in Chapters III–V. We also review several different private protocols: We will review fundamental theories in protocol privacy, which are used widely in Chapter V. Our contribution

is mainly in providing a protocol that supports functional privacy, of which we will review the definitions.

In Chapter III, we describe the problem of building an optimal histogram with respect to a workload, and present two different algorithms—both finding near-optimal histogram with quality guarantees. The first one is less time efficient, but works with streamed workload; the second one requires to store all the weights, but only uses near-linear time. The chapter ends with proof of lower bounds and some comments on compressed workload.

In Chapter IV, we implement some algorithms working on streaming data. We first do experiments to compare histograms optimized for the uniform workload and for the true non-uniform workload, as well as compare our results to optimal results in real environment. Then we incorporate our histogram result into PADS system. We also build an interface for the system, so that more "streaming algorithms" can be added easily.

In Chapter V, we describe the problem of finding Euclidean heavy hitters privately. We first present a protocol, analyze its efficiency, correctness and privacy. Then we extend it to some other cases—with different measurement and/or different orthogonal bases. At the end of Chapter V, we will give some lower bounds which are met by our protocol.

In Chapter VI, we first introduce the definition of probabilistic data streams. Then we present an efficient algorithm to compute near-optimal histograms with a quality guarantee under L_2 norm. For histograms under L_1 norm, we give a heuristic algorithm and show some properties of it. We also give a direction to give guarantees for our heuristic.

CHAPTER II

Background

In this chapter, we introduce some background knowledge. We first review the idea of "sketching", which we present as maintaining projections of a vector on various randomized subspaces. Sketch-based methods have been studied in many environments, for example in [9, 14, 20], and are used to build statistics for massive datasets in sublinear time and space. Section 2.1 covers the main ideas of these methods, which will be used in following chapters. We then review several private protocol definitions in Section 2.2. Protocol privacy is fundamental and has been studied for decades; our result in Chapter V is a protocol supporting functional privacy. We will close this section by introducing the definition of k-anonymity in database privacy as well as comparing previous approaches.

2.1 Sketch-Based Methods

A dyadic interval (formal definition in Chapter III) is of the form $[i2^j, \cdots, (i+1)2^j)$, for integers i and j. Let $\pi(c, I)$ be the projection of c on a dyadic interval I. Sketch based methods maintain projections of the vector on various randomized sets. The authors of [20] present a sketch technique supporting several properties, including the following two which will be used in our work.

- Quick update of the sketch on processing a new data.

- Pre-Identification on c with parameter θ: find a compact list that contains all dyadic intervals I for which $\|\pi(c, I)\| \geq \theta\|c\|$.

The sketch given in [20] takes space $\text{poly}(\log N, 1/\theta, 1/\epsilon)$. The idea is that for l_p norm, it chose a random vector V according to a symmetric p-stable distribution. Then it projected c onto random sets decided by V.

In Chapter V we need a sketch which can identify all heavy hitters (and probably some other items) and takes similar space. The idea is to use a 0/1 random sketch matrix to project the vector c onto random subspace as follows.

Definition 2.1. (Sketch of a vector.) Given a vector c, a *linear sketch* of c is Rc, where R is a random matrix generated from a prescribed distribution, called the *measurement matrix.*

In our case, as is typical, the matrix R will be a pseudorandom matrix, that can be generated from a short pseudorandom seed.

As in [20], one can except with small probability, estimate c_i by $\widetilde{c}_i$ from Rc, where R is a ± 1-valued matrix with $\text{poly}(\log(N), B, 1/\epsilon)$ independent rows, each of which is a pairwise independent family. By repeating $O(k)$ times and taking a median, one can drive down the failure probability to 2^{-k}. By adjusting parameters, we can estimate such c_i well enough as $\widetilde{c}_i$ so that $|\widetilde{c}_i - c_i|^2 \leq (\epsilon/B)\|c\|_2^2$. As in [20], we can use R to estimate the sum of items in a specific group and bound the oracle in time $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$.

Theorem 2.2. *([20]) Fix parameters N, M, B, k, ϵ as above. Fix $\theta \geq \text{poly}(\log(N), \log(M), B, k, 1/\epsilon)^{-1}$. There is a distribution on sketch matrices R and a corresponding algorithm that, from R and sketch Rc of a vector c, outputs a set that includes all terms with magnitude at least $\theta\|c\|_2$ (and possible other terms).*

2.2 Privacy

2.2.1 Protocol Privacy

Secure and private computation on two or more parties have been studied for several decades, starting with [50, 7]. In a private protocol, no party learns anything from the protocol messages other than what can be deduced from the function's input/output relation. Intuitively, there is a polynomial time algorithm, which takes an instance of the function's input/output relation as input, and outputs all the messages each party has learned during the computation in protocol. Formally,

A two-party computation task is specified by a (possibly randomized) mapping g from a pair of inputs $(a, b) \in \{0,1\}^* \times \{0,1\}^*$ to a pair of outputs $(c, d) \in \{0,1\}^* \times \{0,1\}^*$. Let $\pi = (\pi_A, \pi_B)$ be a two-party protocol computing g. Consider the probability space induced by the execution of π on input $\mathbf{x} = (a, b)$ (induced by the independent choices of random inputs r_A, r_B). Let $\mathsf{view}_A^\pi(\mathbf{x})$ (resp., $\mathsf{view}_B^\pi(\mathbf{x})$) denote the entire view of Alice (resp., Bob) in this execution, including her input, random input, and all messages she has received. Let $\mathsf{output}_A^\pi(\mathbf{x})$ (resp., $\mathsf{output}_B^\pi(\mathbf{x})$) denote Alice's (resp., Bob's) output. Note that the above four random variables are defined over the same probability space. Two distributions (or ensembles) $\mathcal{D}_1$ and $\mathcal{D}_2$ are said to be *computationally indistinguishable* with security parameter k, $\mathcal{D}_1 \stackrel{c}{\equiv} \mathcal{D}_2$, if, whenever $X_1 \sim \mathcal{D}_1$ and $X_2 \sim \mathcal{D}_2$ and for any function C having a circuit of size at most 2^k, we have then $|\Pr(C(X_1) = 1) - \Pr(C(X_2) = 1)| \leq 2^{-k}$.

Definition 2.3. Let X be the set of all valid inputs $\mathbf{x} = (a, b)$. A protocol π is a *private protocol computing* g if the following properties hold:

Correctness. The joint outputs of the protocol are distributed according to $g(a, b)$.

Formally,

$$\{(\mathsf{output}_A^\pi(\mathbf{x}), \mathsf{output}_B^\pi(\mathbf{x}))\}_{\mathbf{x}\in X} \equiv \{(g_A(\mathbf{x}), g_B(\mathbf{x}))\}_{\mathbf{x}\in X},$$

where $(g_A(\mathbf{x}), g_B(\mathbf{x}))$ is the joint distribution of the outputs of $g(\mathbf{x})$.

Privacy. There exist probabilistic polynomial-time algorithms S_A, S_B, called *simulators*, such that:

$$\{(S_A(a, g_A(\mathbf{x})), g_B(\mathbf{x}))\}_{\mathbf{x}=(a,b)\in X} \stackrel{c}{\equiv} \{(\mathsf{view}_A^\pi(\mathbf{x}), \mathsf{output}_B^\pi(\mathbf{x}))\}_{\mathbf{x}\in X}$$
$$\{(g_A(\mathbf{x}), S_B(b, g_B(\mathbf{x}))\}_{\mathbf{x}=(a,b)\in X} \stackrel{c}{\equiv} \{(\mathsf{output}_A^\pi(\mathbf{x}), \mathsf{view}_B^\pi(\mathbf{x}))\}_{\mathbf{x}\in X}$$

Any protocol for computing a function of several inputs can be converted, gate-by-gate, to a private protocol. An efficient general result is as follows.

Proposition 2.4. (General-Purpose Secure Multiparty Computation (SMC) [50]) Two parties holding inputs x and y can privately compute any circuit C with communication and computation $O(k(|C| + |x| + |y|))$, where k is a security parameter, in $O(1)$ rounds.

This secure multiparty computation result is equivalent to the oblivious transfer result, which both rely on the feasibility of secret key exchange and in turn the existence of one-way function and finally the most fundamental assumption that NP is not equal to P.

2.2.2 Functional Privacy

As the input sizes grow in some multiparty computation tasks, even linear time communication and/or computation overhead is too much to afford. In such cases, approximate computation instead of exact computation is necessary to achieve sub-linear overhead. The motivation of specifying functional privacy is to define privacy

of an approximate result in terms of the exact result. Intuitively, an approximate result is private with respect to the exact result, if there is a probabilistic polynomial time algorithm, which takes the exact result as input and outputs the final approximate result. Formally,

Definition 2.5 (Private Approximation Protocol [15]). A two-party protocol π is a private approximation protocol for a deterministic, common-output function g on inputs a and b if π computes a (possibly randomized) approximation $\widetilde{g}$ to g such that

- $\widetilde{g}$ is a good approximation to g (in the appropriate sense)
- π is a private protocol for $\widetilde{g}$ in the traditional sense.
- (Functional Privacy.) There exists a probabilistic polynomial-time simulator S such that:

$$\{S(g(\mathbf{x}))\}_{\mathbf{x}=(a,b)\in X} \stackrel{c}{\equiv} \widetilde{g}(\mathbf{x}).$$

In our case of a deterministic function to be output to both Alice and Bob, a (weakly) equivalent definition is as follows, known as the "liberal" definition in [15]:

Definition 2.6. A two-party protocol π is a private approximation protocol for a deterministic, common-output function g on inputs a and b in the *liberal sense* if π computes a (possibly randomized) approximation $\widetilde{g_w}$ to g such that

- $\widetilde{g_w}$ is a good approximation to g (in the appropriate sense)
- There exists a probabilistic polynomial-time simulators S_A and S_B such that:

$$\{S_A(a, g(\mathbf{x}))\}_{\mathbf{x}=(a,b)\in X} \stackrel{c}{\equiv} \{\mathsf{view}_A^\pi(\mathbf{x})\}_{\mathbf{x}\in X}$$

$$\{S_B(b, g(\mathbf{x}))\}_{\mathbf{x}=(a,b)\in X} \stackrel{c}{\equiv} \{\mathsf{view}_B^\pi(\mathbf{x})\}_{\mathbf{x}\in X}$$

Roughly speaking, the equivalence is as follows. Suppose there are simulators in the standard definition. Then, putting $\widetilde{g_w} = \widetilde{g}$, a simulator for the liberal defintion can be constructed by simulating $\widetilde{g_w}(a, b) = \widetilde{g}(a, b)$ from $g(a, b)$ using the hypothesized simulator for functional privacy, then simulating Alice's view from $\widetilde{g_w}(a, b)$ and a using the hypothesized traditional simulator for the protocol that computes $\widetilde{g}$. In the other direction, suppose there is a simulator in the liberal definition. Let τ be a transcript of Alice's view except for input a. (As it turns out, it is not necessary to include a in τ. If a is much longer than τ—as in our situation—we want to avoid including a in τ in order to keep τ short.) Define $\widetilde{g} = \widetilde{g_w}.\tau$ to be $\widetilde{g_w}$ with τ encoded into its low-order bits. We assume that this kind of encoding into approximations can be accomplished without significantly affecting the goodness of approximation; in fact, we will assume that the value represented does not change at all, even if the "approximate" value is zero—that is, τ is auxiliary data rather than an actual part of the value of $\widetilde{g}$. Note that a protocol for $\widetilde{g_w}$ also serves as a protocol for $\widetilde{g}$. It is trivial to simulate the messages of the protocol given a and $\widetilde{g}$. Use the hypothesized simulator in the liberal definition to show functional privacy.

CHAPTER III

Workload-aware Optimal Histograms

3.1 Problem Statement

In this section, we address the problem of computing optimal histograms on data streams. Our primary question is, do the powerful theoretical results known for uniform histogram construction on data streams [20, 23, 24] hold for the workload-aware case as well?

Definition 3.1. Given an input signal $c_0 \cdots c_{N-1}$ and *workload* $w_0 \cdots w_{N-1}$, $0 \leq w_i$, the *workload-optimal* B-bucket histogram h_{opt} is the choice of b_j's and h_j's that minimize $\|c - h\|_w^2 = \sum_i \ w_i(c_i - h_{j(i)})^2$.

The problem of finding h_{opt} is interesting on streamed signals and both stored and streamed workloads. We will present a near-linear time algorithm for stored workloads and an $O(N \log U)$ time algorithm for streamed workloads.

3.2 Preliminaries

Our algorithm under non-uniform workload uses and extends the following definitions and lemmas from [23], which presents a linear time algorithm under uniform workload:

3.2.1 Definitions from previous work

Definition 3.2. Inner Product with Weight: For any two signals $\mathbf{A}$ and $\mathbf{B}$ with length N respectively and the same length N weight vector w, define $\langle \mathbf{A}, \mathbf{B} \rangle_w = \sum_{i=1}^{N} \mathbf{A}_i \mathbf{B}_i w_i$ and $\|\mathbf{A}\|_w^2 = \langle \mathbf{A}, \mathbf{A} \rangle_w$ where w_i is a non-negative weight at index i. We continue to write $\langle \mathbf{A}, \mathbf{B} \rangle$ and $\|\mathbf{A}\|^2$ for the dot product and norm under uniform workload.

Definition 3.3. Bucket Robust Representation [21, 23]. Fix a signal c. A representation h_{r} is called a (B, ϵ)-bucket-robust approximation to c if (i) $\|h_{\mathrm{r}} - c\| \leq \epsilon \|h_{\mathrm{opt}} - c\|$, or, (ii) for any representation h on the boundaries of h_{r} and any other $B - 1$ boundaries, with optimal parameters, we have $(1 - \epsilon)\|c - h_{\mathrm{r}}\|^2 \leq \|c - h\|^2$.

Intuitively, a bucket-robust representation is a representation that either is very close to h_{opt}, or can't be improved much by refinement by B more buckets.

We consider signals indexed on $\{0, 1, \cdots, N - 1\}$, where N is a power of 2. A dyadic interval is an interval of the form $[k2^j, (k + 1)2^j)$, where j and k are integers. The function that equals 1 on set S and zero elsewhere is denoted χ_S. We define Haar wavelets as following:

Definition 3.4. Wavelet: A wavelet is a function ψ on $[0, N)$ of one of the following forms:

$\frac{1}{\sqrt{N}} \chi_{[0,N)}$

$2^{-j/2}(-\chi_{[k2^{j-1},(k+1)2^{j-1})} + \chi_{[(k+2)2^{j-1},(k+3)2^{j-1})})$.

Definition 3.5. Support of Wavelet: The support of a vector v is defined as $\mathrm{supp}(v) = \{t : v(t) \neq 0\}$.

The support of a wavelet vector of the first type is the entire interval $[0, N)$. Each

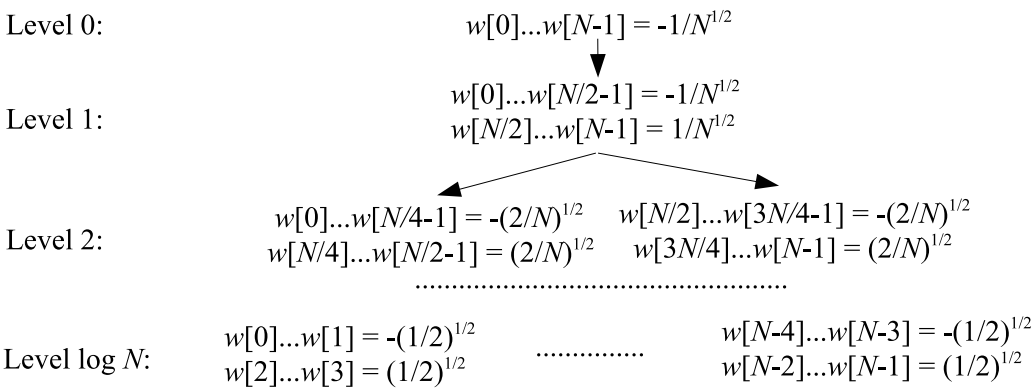

Figure 3.1: Illustration of Haar wavelets

wavelet of the second type is constant on the left half and right half of its support, and takes values on its left and right halves that are negatives of each other.

Definition 3.6. Level of Wavelet: Wavelets with length of support 2^i are defined to be on the $i^{th.}$ level. A wavelet of the first type is defined to be on the $0^{th.}$ level.

Dynamic intervals and Haar wavelets can be represented in a tree. Figure 3.1 is an illustration. Each node in this tree represents one wavelet vector, and there are N nodes in total. For $i = 1 \cdots \log N$, level i contains all wavelet vectors with support length $2N/2^i$. The supports of wavelet vectors in the same level are disjoint. The support of each wavelet vector is the union of the supports of its children.

3.2.2 A Previous Uniform-Workload Algorithm

The algorithm in [23] proceeds as follows.

(i). (Selection of large wavelet terms.) Read in a length-N stream c of time-series data and output a list L of the $B' \leq \text{poly}(B, \log N, 1/\epsilon)$ wavelet terms with largest coefficients.

(ii). (Construction of bucket-robust representation.) Select the largest $B'' = \text{poly}(B, \log N, \log M, 1/\epsilon)$ terms from L, greedily, using a particular 2-part stopping rule

(described in detail below). Call the result h_{r}, a *bucket-robust* histogram of $O(B'')$ buckets.

(iii). (Construction of output.) Find a best B-bucket histogram h to h_{r}, and output h as a $(1+\epsilon)$-approximate histogram to c.

Note that the first step is performed on the stream, but the last two steps are full-space, polynomial-time post-processing algorithms on small input, that is, input of polylogarithmic size. In [23], the authors showed that the computational cost of each step meets the claim.

We now consider in more detail the relevant parts of the [23] algorithm. In Step (ii), we need an additional parameter ϵ_{r}. We take terms from L, from biggest to smallest, $4B\log(N)$ at a time, and add them to h_{r}, which is initially the zero histogram. Let h'_{r} denote the *next* value of h_{r}, *i.e.*, h_{r} plus the next $4B\log(N)$ terms to be taken. We stop when either of the following conditions is met:

- (No Progress.) $(1-\epsilon_{\mathrm{r}})\|c-h_{\mathrm{r}}\|_2^2 \geq \|c-h'_{\mathrm{r}}\|_2^2$.
- (Many Terms.) We have accumulated T terms, for some T which is at most $O(\epsilon_{\mathrm{r}}^2\log(1/\epsilon_{\mathrm{r}})B\log(N))$.

Using a case analysis, the output h is shown to be correct whichever stopping rule is used. The conditions in bucket-robustness (Definition 3.3) correspond to the Many Terms and No Progress stopping rules, respectively.

In Step (iii), dynamic programming similar to that in [30] is used. In particular, the dynamic programming algorithm accesses h_{r} only by making the following query. Given interval $[\ell, r)$, what is the best height a of a 1-bucket histogram $a\chi_{[\ell,r)}$ and what is the resulting error $\sum_{\ell\leq i<r}(h_{\mathrm{r}i}-a)^2$ on that interval? This query must be answered in time to meet the post-processing bound.

The following is the main result from [23]:

Lemma 3.7. *Given B, N, and ϵ, there exists $\epsilon_r \geq (\epsilon/B)^{O(1)}$ such that a nearly-optimal representation h to a (B, ϵ_r)-bucket-robust representation h_r is also $(1+\epsilon)$-nearly optimal to c.*

3.3 Algorithm in lockstep model

At a high level, our algorithm proceeds as following. Firstly, we define an expanded-workload domain from the original domain, and an equivalent transformation of vectors between these two domains, which preserves norms and dot products. Then, we compute the near-optimal representation under uniform workload for the equivalent signal in the expanded-workload domain as in [23]. Finally, we lift the near-optimal representation back to the original domain, which is also a near-optimal representation in the original domain with respect to the specified workload.

3.3.1 Notation and Basics

Definition 3.8. Original Domain: Consider a signal c indexed on $\{0, 1, \cdots, N-1\}$. The domain $\{0, 1, \cdots, N-1\}$ is called the original domain of c.

Definition 3.9. Expanded-workload Domain: For each signal c with weight vector w, define $U = \sum_{i=0}^{N-1} w_i$. The domain $\{0, 1, \cdots, U-1\}$ is called the expanded-workload domain of c.

For any signal c in the original domain, we define an equivalent signal c' in the expanded-workload domain as following: $\forall i \in \{0, 1, \cdots, N-1\}$, let $s = \sum_{j=0}^{i-1} w_i$. We define $c'_s, c'_{s+1} \ldots c'_{s+w_i-1}$ to be c_i.

Figure 3.2 is an illustration. Each entry c_i in the original domain has w_i copies in the expanded-workload domain. An example of c and the equivalent c' is:

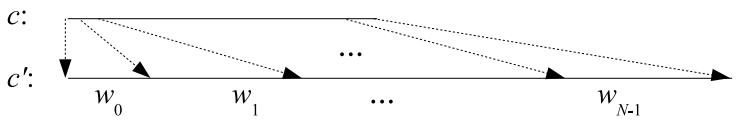

Figure 3.2: Transformation between original and expanded domain

c: $\langle 1, 2, 3\rangle$ in the original domain with weight $\langle 5, 2, 1\rangle$, where $N = 3$.

c': $\langle 1, 1, 1, 1, 1, 2, 2, 3\rangle$ in the expanded-workload domain, where $U = 8$.

Lemma 3.10. *For any N-dimensional vectors $\mathbf{A}$ and $\mathbf{B}$ in the original domain, and so-defined equivalent U-dimensional vectors $\mathbf{A}'$ and $\mathbf{B}'$ in the expanded-workload domain, norms and dot products are preserved by transformation,* i.e.

(i). $\|\mathbf{A}\|_w^2 = \|\mathbf{A}'\|^2$

(ii). $\langle \mathbf{A}, \mathbf{B}\rangle_w = \langle \mathbf{A}', \mathbf{B}'\rangle$

Proof.

$$\langle \mathbf{A}, \mathbf{B}\rangle_w = \sum_{i=0}^{N-1} \mathbf{A}_i \mathbf{B}_i w_i = \sum_{i=0}^{N-1} \sum_{j=1}^{w_i} \mathbf{A}_i \mathbf{B}_i = \sum_{k=0}^{U-1} \mathbf{A}'_k \mathbf{B}'_k = \langle \mathbf{A}', \mathbf{B}'\rangle.$$

Set $\mathbf{B} = \mathbf{A}$, we can get 1 from 2. □

Definition 3.11. Natural Boundary: For all i in $\{0, 1, \cdots, N-1\}$, $s = \sum_{j=0}^{i-1} w_i$ is defined to be a natural boundary in the expanded-workload domain.

From the above definition, there are altogether N natural boundaries in the expanded-workload domain. We say that, an index x in the expanded-workload domain meets a wavelet `w`, if and only if x falls in the support of `w`.

3.3.2 Representation in Expanded-Workload Domain

According to the definition of wavelet vectors, it is easy to see that given an index $x \in [0, U)$ and a level $i \in [0, \log U]$, one can compute in constant time the specific

wavelet vector $\mathtt{w}$ on the $i^{th\cdot}$ level that meets x. In addition let $\mathrm{supp}(\mathtt{w}) = [\mathtt{w_1}, \mathtt{w_2})$, one can compute the local index $x' = x - \mathtt{w_1} + 1$ with respect to $\mathtt{w}$ in constant time.

Note that the coefficient of a wavelet $\mathtt{w}$ in the expanded-workload domain can be non-zero only if there is at least one natural boundary that meets $\mathtt{w}$. If not, all the data at indices meeting $\mathtt{w}$ will be the same. The left half and right half of $\mathtt{w}$ will then compensate each other, so the coefficient of $\mathtt{w}$ will vanish. Since there are N natural boundaries in total, and the expanded-workload domain is of dimension U, there are at most $N \log U$ non-zero coefficients in total.

We find these $N \log U$ non-zero coefficients in the expanded-workload domain in a way similar to that described in [23] and we set the following lemma:

Lemma 3.12. *There is an algorithm that reads in a stream* $(c_0, w_0), (c_1, w_1) \cdots$ *and outputs the* $N \log U$ *non-zero wavelet coefficients (in arbitrary order), using per-item time* $O(\log U)$ *and space* $O(\log U)$.

Proof. Given wavelet $\mathtt{w}$ and natural boundary $x \in [0, U)$, if x meets $\mathtt{w}$, we say $\mathtt{w}$ is active with respect to x. For each x, there are $\log U + 1$ active wavelets, one for each level. We use a $(\log U + 1) \times 4$ table R to keep information about all these active wavelets. After processing an index $x \in [0, U)$, cell $R[i][0]$ specifies the wavelet $\mathtt{w}$ on the $i^{th\cdot}$ level that is currently active; cells $R[i][1]$ and $R[i][2]$ store current accumulative sums for the left and right halves of $\mathtt{w}$ respectively; cell $R[i][3]$ stores the local index x' with respect to the current active $\mathtt{w}$.

For coming (c_j, w_j), we first get the index x of its last copy in expanded-workload domain. This can be done in constant time by accumulating over all past items. Then for each $i \in [0, \log U]$, suppose $\mathtt{w^i}$ is the current active wavelet on the $i^{th\cdot}$ level, we check:

- If $\mathtt{w^i}$ is NOT the wavelet stored in $R[i][0]$, we need to clean up and report the coefficient of last active wavelet on $i^{th.}$ level. This can be done in constant time, using information in R and c_j. Also we need to update all cells on the $i^{th.}$ row of R.

- If $\mathtt{w^i}$ is the wavelet stored in $R[i][0]$ and x meets its left half, we need to update (accumulate) $R[i][1]$ and $R[i][3]$. This can be done in constant time.

- If $\mathtt{w^i}$ is the wavelet stored in $R[i][0]$ and x meets its right half, we need to update (accumulate) $R[i][2]$ and $R[i][3]$, and probably $R[i][1]$. This can be done in constant time.

For each coming (c_j, w_j) we need to update $O(\log U)$ cells, each in constant time. So the algorithm uses per-item time $O(\log U)$ and $O(\log U)$ space.

□

We use the method mentioned in [23] to get the B' largest coefficients. Thus:

Lemma 3.13. *There is an algorithm that takes B' as input, reads in a stream $(c_0, w_0), (c_1, w_1) \cdots$ and outputs the top B' wavelet coefficients, using per-item time $O(\log U)$ and using space $O(\max(B', \log U))$.*

Note that we have at most $N \log U$ non-zero wavelet coefficients, so the part of computing top B' coefficients will run in $O(N \log U)$ time in total.

As in [23], we have:

Lemma 3.14. *There exists an algorithm that, given B, U, and ϵ, on input the N values of an integer-valued signal c and a non-negative integer-valued weight vector $\langle w_0, w_1, \cdots w_{N-1} \rangle$, with $\|c\|_w \leq M$, outputs a B-bucket histogram h'^* with*

$$\|c' - h'^*\|^2 \leq (1 + O(\epsilon))\|c' - h'_{\text{opt}}\|^2$$

where c' is the expanded domain transform of c and h'_{opt} is the optimal B-bucket histogram representation to c', both in the expanded-workload domain. The algorithm uses space $B(\log(U)\log(M)/\epsilon)^{O(1)}$ and time $c_1 N \log U + (B\log(M)\log(U)/\epsilon)^{c_2}$, where c_1 and c_2 are constants.

3.3.3 Histogram in Original Domain

Lemma 3.15. *There is a B bucket representation h'_{b} satisfying: all of its boundaries are natural boundaries in the expanded-workload domain, and*

$$\|c' - h'_{\text{b}}\|^2 \leq \|c' - h'^*\|^2$$

Proof. We adjust h'^* to get h'_{b} in the following way: For every boundary c of h'^*, falling between two adjacent natural boundaries a and b, suppose the signal c' has value s between a and b, and the representation h'^* has value s_1 between a and c, and value s_2 between c and b. If $|s - s_1| \leq |s - s_2|$, move c to b, so that h'_b has value s_1 between a and b. Otherwise, move c to a. [see figure 3.3].

Suppose $|s - s_1| \leq |s - s_2|$, we have:

$$\begin{aligned} \|c' - h'_{\text{b}}\|^2 &= \|c' - h'^*\|^2 - [(s - s_2)^2 - (s - s_1)^2] * (b - c) \\ &\leq \|c' - h'^*\|^2 \end{aligned}$$

The proof is similar when $|s - s_1| > |s - s_2|$.

□

Lemma 3.16. *$\|c - h\|_w^2 \leq (1 + O(\epsilon))\|c - h_{\text{opt}}\|_w^2$, where h is the original-domain transform of h'_{b}.*

Proof. Define $h'_{\text{b\_opt}}$ to be the expanded domain transform of h_{opt}. Clearly, it has B

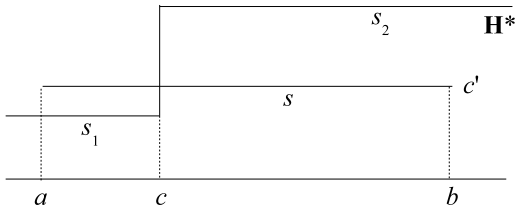

Figure 3.3: Illustration of histograms in Lemma 3.15

buckets. We have:

$$\begin{aligned}
\|c-h\|_w^2 &= \|c'-h'_{\mathrm{b}}\|^2, && \text{By equivalence}\\
&\le \|c'-h'^*\|^2, && \text{By lemma 3.15}\\
&\le (1+O(\epsilon))\|c'-h'_{\mathrm{opt}}\|^2, && \text{By lemma 16}\\
&\le (1+O(\epsilon))\|c'-h'_{\mathrm{b_opt}}\|^2, && \text{By definition}\\
&= (1+O(\epsilon))\|c-h_{\mathrm{opt}}\|_w^2, && \text{By equivalence.}
\end{aligned}$$

□

Note that to adjust h'^* to h'_{b}, we only need to store all the natural boundaries that are adjacent to boundaries of wavelet vectors with top B' coefficients. So this step can be run in $O(B)$ time and $O(B')$ space.

3.3.4 Main Results, Streamed Weights

Combining the above results, we have:

Theorem 3.17. *There is an algorithm that, given parameters B, N, M, ϵ, weight vector w, and data c in time series with $\|c\|_w^2 \le M$, outputs a B-bucket histogram h with*

$$\|c-h\|_w^2 \le (1+O(\epsilon))\|c-h_{\mathrm{opt}}\|_w^2$$

where h_{opt} is the best possible B-bucket histogram representation to c under weight

w. The algorithm uses space $B(\log(U)\log(M)/\epsilon)^{O(1)}$, and runs in time $c_1 N \log U + \left(\frac{B \log U \log M}{\epsilon}\right)^{c_2}$, where c_1 and c_2 are two constants.

3.4 Near-Linear Time Algorithm

In Section 3.3, we present an algorithm running in time $O(N \log U)$, where $U = \sum_{i=0}^{N-1} w_i$. However, U may go up to $O(2^N)$ or even worse in practice and even when $U = O(N)$, running time $O(N \log N)$ could be too expensive for some application. In this section, we will remove the factor $\log U$ and present a near-linear time algorithm in a model where w is stored.

At a high level, our algorithm proceeds as follows. We will regard the weights as rounded to a power of $(1+\epsilon)$; there is a small number $p = \log_{1+\epsilon}(M)$ of these classes. We split the incoming streaming data into p new substreams of data, according to the associated weight. For each sub-stream, we create a bucket-robust representation, as in [23]. Combining the bucket-robust representations gives a linearly-robust representation h_{r}, that we define below. Lemma 3.25-3.26 and Lemma 3.27 guarantee that h_{r} is correct and can be computed in desired time. Finally, a near-best B-bucket representation h to h_{r} will be constructed. Lemma 3.28- 3.30 show how to compute h efficiently by dynamic programming similar to that in [30], and Lemma 3.31 proves its approximation quality to complete the proof.

3.4.1 Notation and Basics

We consider signals of length N, with weights $w_1, \cdots, w_N$, and such that $\|c^2\| \leq M$. We will assume that data items are integers, and that weights are positive integers in the range $w_{\min} = 1$ to some $w_{\max} \leq M$.

Definition 3.18. Rounded weights w.r.t. original weights: Let $p = (\log_{1+\epsilon} M) + 1$. Define p numbers $w^1, w^2, \cdots, w^p$, where $w^i = (1+\epsilon)^{i-1}$. Round all the original

weights $w_1, \cdots, w_N$ down to $w^1, w^2, \cdots, w^p$, denoted as $w'_1, \cdots, w'_N$ respectively, i.e., $\forall i \in \{1, \cdots, N\}, \exists j \in \{1, \cdots, p\}$ such that $w^j = w'_i$ and $w_i \leq (1+\epsilon)w'_i \leq (1+\epsilon)w_i$. We use $w' = (w'_1 \cdots w'_N)$ to represent the length-N rounded weight vector.

Lemma 3.19. *(Close relationship between original weights and rounded weights:) Fix a single c of dimension N. Then $\|c - h'_{\text{opt}}\|_w^2 \leq (1+\epsilon)\|c - h_{\text{opt}}\|_w^2$, where h_{opt} is the optimal B bucket representation to c under weight w, and h'_{opt} is the optimal B-bucket representation to c under weight w'.*

Proof.

$$\begin{aligned}
\|c - h'_{\text{opt}}\|_w^2 &= \sum_{i=1}^{N} [c_i - h'_{\text{opt}}(i)]^2 w_i \\
&\leq (1+\epsilon) \sum_{i=1}^{N} [c_i - h'_{\text{opt}}(i)]^2 w'_i \\
&\leq (1+\epsilon) \sum_{i=1}^{N} [c_i - h_{\text{opt}}(i)]^2 w'_i \\
&\leq (1+\epsilon) \sum_{i=1}^{N} [c_i - h_{\text{opt}}(i)]^2 w_i \\
&= (1+\epsilon)\|c - h_{\text{opt}}\|_w^2
\end{aligned}$$

□

Next we describe how to build a data structure R in time $O(N)$ and one pass, that answers all the following queries in specified time.

- For all $j < N$, recover w'_j in constant time.
- For all $j < N$, recover the number of $k < j$ such that $w'_k = w'_j$ in constant time.
- For all $j < N$ and $i \leq p$, get the index k, if any, such that $j \leq k < N$, $w'_j = w^i$, and $\forall j \leq t < k, w'_t \neq w^i$ in $O(\log N)$ time. In other words, we want to find the smallest index k greater than j and with the same rounded weight as j.

- For all $j < N$ and $i \leq p$, get the index k, if any, such that $0 \leq k \leq j$, $w'_j = w^i$, and $\forall k < t \leq j, w'_t \neq w^i$ in $O(\log N)$ time. In other words, we want to find the greatest index k smaller than j and with the same rounded weight as j.

One of such data structures works as follows: We have an array R_1 of length N, and a list R_2 of p queues, which takes in total $O(N)$ space. For every index i, if $w'_i = w^j$, we set $R_{1i} = j$, push i to the end of queue R_{2j}, and accumulate the total number of elements in that queue. The first two queries can be answered directly from R. For the last two queries, we can use binary search to locate w_j in its queue in $O(\log N)$ time and answer the query directly from that queue.

3.4.2 Weakly Robust Representations

Definition 3.20. (Partition and combination of representations:) Given a stream c and a partition $\mathcal{P}$ of $[0, N)$, $c_i^{\mathcal{P}}$ is a sub-stream under partition $\mathcal{P}$, for $i \in \{1, \cdots m\}$. Define $\#_{i\in\{1\cdots m\}} c_i^{\mathcal{P}}$ as the combination of all the sub-streams. We use $\#c_i^{\mathcal{P}}$ for short in what follows. So $c = \#c_i^{\mathcal{P}}$.

Definition 3.21. Given a weight vector w, define the partition $\mathcal{P}$ as following: For each $i \in \{1, \cdots N\}$, if $w'_i = w^j$, put i into the $j^{th.}$ group. Let h^i be the (B_r, ϵ_r)-bucket-robust representation for $c_i^{\mathcal{P}}$, where B_r and ϵ_r are in the order of $(1/\epsilon, B, \log M, \log N)$. Define h_r to be $\#h^i$.

Reference [23] gives the definition of bucket-robust representation under uniform workload. Unfortunately, that definition fails in our framework of separate sub-streams based on weight classes. We give a brief illustration why. Suppose there are just two weight classes, that partition $[0, N)$ precisely into the even and odd indices. Suppose h' and h'' are two bucket-robust histograms, defined on the even and odd indices, respectively, and each has a small number of buckets. Suppose that

all heights represented in h' and h'' are distinct. Suppose that h' and h'' have about equal shares of the error. Finally, suppose that the reason for bucket-robustness is *only* that they are not much improved by refinement by B more buckets; that is, suppose $\|c - h'\|^2$ and $\|c - h''\|^2$ are each approximately $\frac{1}{2}\|c - h_{\text{opt}}\|^2 > 0$, which can happen if $c - h_{\text{opt}}$ is noisy. We use the following example to show that all of these assumptions are consistent.

Example 3.22. We simulate a signal c of dimension 300, where each $c_i \in (0, 1)$ is generated randomly by RAND() with seed 10000 in C. The optimal one-bucket histogram of c has square error 24.9414 and the optimal five-bucket histogram has square error 23.1963, which means for any $\epsilon \in [0.07, 1)$, the optimal one-bucket histogram itself is a $(5, \epsilon)$-bucket-robust representation only that it is not move improved by refinement of 4 more buckets.

Now, let h be the combination of h' and h''; we will show that h is not bucket-robust. Note that, because of the even/odd partition induced by the given weights, h has N buckets, each of size 1. Then one of the bucket-robust conditions says that if we further refine h and then *optimize the heights*, we do not get much improvement, multiplicatively. Clearly, by optimizing the heights, we can get error zero! The other condition says that $\|c - h\|^2 \approx \epsilon^2\|c - h_{\text{opt}}\|^2$. But $\|c - h\|^2 = \|c - h'\|^2 + \|c - h''\|^2 \approx \|c - h_{\text{opt}}\|^2 \gg \epsilon^2\|c - h_{\text{opt}}\|^2$. It follows that neither condition of bucket-robustness is satisfied. Therefore, we need to weaken the definition of bucket-robustness somewhat. We also need to expand it to non-uniform workload.

Definition 3.23. Fix a signal c and rounded weight vector w'. Given parameters B and ϵ, let h_{opt} denote an optimal B-bucket histogram for c under w'. A representation h_{r} is called a (B, ϵ)-linearly-robust (or just (B, ϵ)-robust henceforth) approximation

to c under weight w', if, for any B-bucket histogram h_B and any scalars a and b, either $\|c - h_{\mathrm{r}}\|^2 \leq \epsilon^2 \|c - h_{\mathrm{opt}}\|^2$ or $(1-\epsilon)\|c - h_{\mathrm{r}}\|_w^2 \leq \|c - (ah_{\mathrm{r}} + bh_B)\|_w^2$.

The first condition is similar to the corresponding condition in bucket-robustness: the error $\|c - h_{\mathrm{r}}\|^2$ is already tiny compared with $\|c - h_{\mathrm{opt}}\|^2$. The second condition is implied by the corresponding condition in bucket-robustness. Hence, linearly-robustness is a weaker notion than bucket-robustness. We also need to modify Definition 3.21 as follows.

Definition 3.24. Given a weight vector w, define the partition $\mathcal{P}$ as following: For each $i \in \{1, \cdots N\}$, if $w'_i = w^j$, put i into the $j^{th.}$ group. Let h^i be the (B_r, ϵ_r)-bucket-robust representation for $c_i^{\mathcal{P}}$, and I be the set of all index i's satisfying h^i is linear-robust *only* because it is much better than the optimal histogram. Define h_{r_1} to be $\#_{i \in I} h^i$, and h_{r_2} to be $\#_{i \in \{1,2,\cdots,p\}-I} h^i$. Define h_c to be the combination of h_{r_1} and h_{r_2}.

Next we show that both I and $\{1, 2, \cdots, p\} - I$ can be non-empty. Namely, there is a h^i, which is linear-robust:

- *only* because it is much better than the optimal. Consider a signal with exponentially distributed data. Given a histogram, adding one more bucket can improve a lot since the data changes dramatically. So it is possible to find some histogram which improves the optimal a lot but still has room to improve by combining with some B_r-bucket histogram.

- *only* because it can not be much improved by combination of any B_r-bucket histogram. Example 3.22 works here too, since the optimal five bucket histogram can be viewed as a combination of one bucket histogram and some five bucket histogram.

Lemma 3.25. *Histograms h_{r_1} and h_{r_2} are (B_r, ϵ_r)-robust representations to the corresponding parts $\widetilde{c}_1$ and $\widetilde{c}_2$ of c, under weight w'.*

Proof. As in Definition 3.24, for each $i \in 1, \cdots, m$, h^i is a (B_r, ϵ_r)-robust representation for $c_i^{\mathcal{P}}$ because it is much better than the optimal or it is not much improved by combination of any B_r-bucket histogram. Since h_{r_1} and h_{r_2} can be viewed as linear combinations of all h^i, taking corresponding rounded weight w^i as coefficient, they are (B_r, ϵ_r)-robust representations to $\widetilde{c}_1$ and $\widetilde{c}_2$, under weight w'. □

Now we claim that the combination of h_{r_1} and h_{r_2} is (B_r, ϵ_r)-robust to c under machine precision.

Fact 3.26. Under weight w', histogram h_c is not necessarily a (B_r, ϵ_r)-robust representation to c literally, but is a (B_r, ϵ_r)-robust representation to c under machine precision in practice.

Proof. In the case that both I and $\{1, 2, \cdots, p\} - I$ are non-empty, it is possible to make up a h_c that fails in both conditions of linear-robustness by setting proper error tolerance ϵ and workload w.

Next we show that $h_{r_2} = \widetilde{c}_2$ to machine precision. As reviewed in Section 3.2.2, h_{r_2} is the combination of robust histograms which are computed by accumulating $(1/\epsilon, \log M)^{O(1)}$ terms. Since each accumulation reduces the error by a factor of $O(1/\epsilon, \log M)$, the maximal error M will be reduced by a factor of $\Omega((\log M)^{\log M}) = \Omega(M^{\log \log M})$ and becomes $o(\frac{1}{M^{\log \log M}})$. The assumption that $o(\frac{1}{M^{\log \log M}})$ is equal to 0 under machine precision is reasonable. □

Fact 3.26 shows that h_c is robust under reasonable assumption. We will use h_r instead of h_c as the combination of h_{r_1} and h_{r_2} from now on.

Lemma 3.27. *There are two constants c_1 and c_2 such that h_r can be computed in $c_1 N + \left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$ time.*

Proof. Note that we need constant time per item to split each data in the original stream into the corresponding substream, according to $\mathcal{P}$. We then run an algorithm on each substream that takes time linear plus $\left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$ (see [23]). Since the substreams' total length is N, the result follows. □

We use an array to store each h^i in the form $(w^i : h^i_1, l^i_1, h^i_2, l^i_2 \cdots, h^i_{B'}, l^i_{B'})$, where h^i_i is the height and l^i_i is the left boundary. Using a result from [23], we have:

Lemma 3.28. *For $j = 1, 2, \cdots, N$, each $h_{r,j}$ can be computed in $O(\log B')$ time, with $B' \leq \left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$, given $\|c\| \leq M$, where $h_{r,j}$ is the value at the $j^{th.}$ index in h_r.*

Proof. Using the data structure R as specified before, for every j we can recover w'_j and the number of $k < j$ such that $w'_k = w'_j$ defined as Pre_j in constant time. Suppose $w'_j = w^i$. Using binary search, we can find $\text{Pre}_j \in [l^i_t, l^i_{t+1})$ in time $O(\log B')$, and $h_{r,j} = h^i_t$. □

Lemma 3.29. *Given h_r, there is an algorithm that takes as input parameters B, N, M, and ϵ and histogram h_r with $\|h_r\| \leq M$, and computes the best height h to $h_{r,i} \cdots h_{r,j}$ along with the associated error in time $O(pB' \log B')$, where $B' \leq \left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$, and $p = O(\log M)$, $\forall i, j \in \{1, \cdots N\}$.*

Proof. Let $w_{\text{sum}} = \sum_{k=i}^{j} w'_k$. Define a random variable X such that $X = h_{r,k}$ with probability $\frac{w'_k}{w_{\text{sum}}}$, for $i \leq k \leq j$. Then $E[X] = h$ is the height of the optimal one-bucket representation to h_r between i and j. Therefore, $h = \frac{\sum_{k=i}^{j} w'_k h_{r,k}}{\sum_{k=i}^{j}(w'_k)} = \frac{\sum_{k=i}^{j} w'_k h_{r,k}}{w_{\text{sum}}}$.

Note that all the data in one bucket of a robust representation in each stream have the same rounded weight and the same height in h_r. So it will take constant

time to compute the sum of weights and the sum of the product of data and weights for each bucket. Since there are B' buckets in each robust representation, and p robust representations in total, and we need $O(\log B')$ time to decide the index of each $h_{r,i}$ in its corresponding substream, the height and error can be computed in $O(pB' \log B')$ time. □

3.4.3 Histogram extracted from robust representation

We use dynamic programming to get a B-bucket representation h to h_r. First assume that we know an approximation E to the optimal error E_{opt}, satisfying $E_{\text{opt}} \leq E \leq 2E_{\text{opt}}$, where $E_{\text{opt}} = \|h_r - \widehat{h}\|_{w'}^2$, and $\widehat{h}$ is the optimal representation to h_r under weight w'. Define $Far[j,l]$ to be a position x such that some j-bucket histogram on $[0,x)$ has error at most $(\ell + j + 1)\frac{\epsilon}{2B}E$ but no j-bucket histogram on $[0,x+1)$ has error at most $\frac{\ell\epsilon}{2B}E$. We build a $O(B^2/\epsilon)$-sized table T to store the information, for each $j \leq B$ and each $l \leq O(B/\epsilon)$. For each entry of T, we compute:

$$T[j][l] = Far[j,l] = \max_{l_1+l_2=l+1} x : cost(Far[j-1,l_1],x) \leq l_2$$

where $cost(l,r)$ is the difference of h_r and the optimal one-bucket representation between index l and index r to it under weight w'.

Lemma 3.30. *Given B, w', ϵ, and robust representations $h^1, \cdots, h^p$, with $\|c\| \leq M$, by using $O(B^2)$ additional space, we can output a B-bucket representation h in $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ time, with*

$$\|h_r - h\|_{w'}^2 \leq (1+\epsilon)\|h_r - \widehat{h}\|_{w'}^2$$

where $\widehat{h}$ is the best B-bucket representation to h_r.

Proof. Assume we know the proper E as specified. For some optimal histogram, let e_j be the error in the $j^{th.}$ bucket, and let $m_j = \lceil e_j \times 2B/(\epsilon E) \rceil$. Then, inductively,

h will do at least as well as using error bound $m_j \frac{\epsilon}{2B} E$ for the $j^{th.}$ bucket, which means the boundaries it finds will all be equal to or to the right of the corresponding boundaries in the optimal histogram. The overall error is sub-optimal by $\frac{\epsilon}{2B} E$ per bucket, which is at most ϵE_{opt} overall. Thus the overall error is $(1+\epsilon)E_{\text{opt}}$, as desired. □

We can find a proper E as follows. Since the zero histogram has error at most M, we have $1 \leq E_{\text{opt}} \leq M$. So there is a proper $E \in S = \{M, \frac{M}{2}, \cdots, 1\}$. We use binary search to find an i such that, for $E = \frac{M}{2^i}$, $T[B][B/\epsilon] = N$, and for $E = \frac{M}{2^{i+1}}$, $T[B][B/\epsilon] < N$. Then $\frac{M}{2^i}$ is a proper E satisfying $E_{\text{opt}} \leq E \leq 2E_{\text{opt}}$. Since $|S| = O(\log M)$, this can be done in time $O(\log \log M)$.

3.4.4 Main Results, Stored Weights

Lemma 3.31. *Fix a signal c with rounded weight w', and let h_r be a (B, ϵ)-robust approximation to c under weight w'. Let h be a B-bucket approximation got from h_r by the algorithm specified above. Then*

$$\|c - h\|_{w'}^2 \leq (1 + O(\epsilon))\|c - h'_{\text{opt}}\|_{w'}^2$$

where h'_{opt} is the optimal B-bucket representation to $\widetilde{c}_1$ under weight w'.

Proof. Extend the proof from [23] under uniform weight to weight w'. Notice that, the proof under uniform weight uses robustness, the triangle inequality and the Pythagorean theorem. We will check that all these properties are hold under weight w'.

1. Robustness is preserved under weight w' by Lemma 3.25.

2. Triangle inequality: For any two N-dimensional signals X and Y with weight w, define another two N-dimensional signals S and T, satisfying $S_i = \sqrt{w_i} X_i$ and

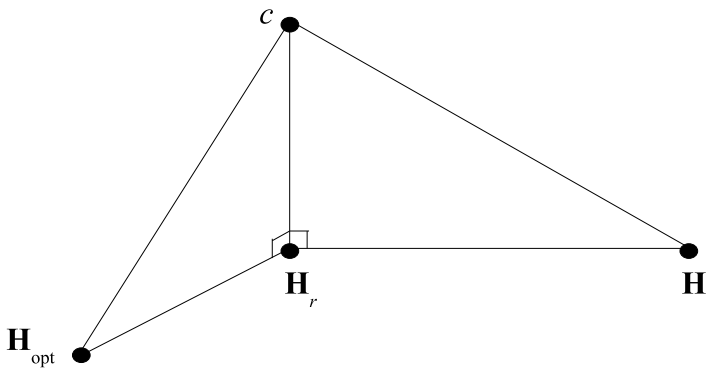

Figure 3.4: Illustration of histograms in Lemma 3.31. By optimality of h_r, there are near right angles as indicated.

$T_i = \sqrt{w_i} Y_i$ for all i. We have:

$$\begin{aligned}
\|X - Y\|_w &= \left[\sum_{i=1}^{N} w_i |X_i + Y_i|^2\right]^{\frac{1}{2}} \\
&= \left[\sum_{i=1}^{N} |S_i + T_i|^2\right]^{\frac{1}{2}} \\
&\leq \left[\sum_{i=1}^{N} |S_i|^2\right]^{\frac{1}{2}} + \left[\sum_{i=1}^{N} |T_i|^2\right]^{\frac{1}{2}} \\
&= \left[\sum_{i=1}^{N} w_i |X_i|^2\right]^{\frac{1}{2}} + \left[\sum_{i=1}^{N} w_i |Y_i|^2\right]^{\frac{1}{2}} \\
&= \|X\|_w + \|Y\|_w
\end{aligned}$$

3. Pythagorean theorem: For any three N-dimensional signals A, B and C with weight w, if $\langle A - C, B - C \rangle_w = 0$, we have

$$\begin{aligned}
\|A - B\|_w^2 &= \|A - C + C - B\|_w^2 \\
&= \langle A - C + C - B, A - C + C - B \rangle_w \\
&= \|A - C\|_w^2 + \|C - B\|_w^2 + 2\langle A - C, C - B \rangle_w \\
&= \|A - C\|_w^2 + \|C - B\|_w^2
\end{aligned}$$

Roughly speaking, the weak robustness property insures that there are near-right

angles at c-h_r-h_{opt} and c-h_r-h. Since the leg c-h_r is the same in the two triangles and since h-h_r is shorter than h_{opt}-h_r, it follows that h-c is not much longer than h_{opt}-c. See Figure 3.4. □

Combining Lemma 3.19 and Lemma 3.31, we can go from the rounded weights to the original weights, and have:

Theorem 3.32. *There is an algorithm that, given parameters B, N, M, ϵ, weight vector w, and data c in time series with $\|c\|_w^2 \leq M$, outputs a B-bucket histogram h with*

$$\|c - h\|_w^2 \leq (1 + O(\epsilon))\|c - h_{\text{opt}}\|_w^2$$

where h_{opt} is the best possible B-bucket histogram representation to c under weight w. The algorithm uses space $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ in addition to the space associated with w (independent of the input). The algorithm uses time in $c_1 N + \left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$, where c_1 and c_2 are two constants.

3.5 Lower Bounds

It is easy to see that a histogram algorithm that first reads the data and then is given a workload must store all the data, since the choice of workload and histogram approximation criterion can force the algorithm to recover any data item exactly. This immediately gives

Theorem 3.33. *Suppose data and workload values are interleaved arbitrarily. For any $B \geq 3$, any algorithm that outputs a nearly optimal B-bucket histogram uses space $\Omega(N \log(M))$ bits (enough to store all the data) up to a constant factor.*

While Theorem 3.33 is the strongest possible statement about interleaving data and workload values, it says nothing about compressing the workload. Above we

showed that, to get a $(1+\epsilon)$-factor approximation, one can round weights to a power of $(1+\epsilon)$ (*i.e.*, discard low-order bits). We now show that, in a sense, this is the only kind of lossy compression that is possible.

Lemma 3.34. *Suppose an algorithm reads and processes a workload of length N and bound M into an object s of size $|s|$, then discards everything about the workload except s, then reads time series data. If, for any workload, any data, and any sufficiently small $\epsilon > 0$, the algorithm produces, with probability $\gg 1/2$, a $(1+\epsilon)$-approximation to the best 3-bucket histogram, then the algorithm can be used as a subroutine to store all values from a vector of positive integer entries bounded by $M/4$, of length $\geq N - O(\log(M)/\epsilon)$, up to the factor $(1+O(\epsilon))$.*

Proof. Suppose we are given a vector v of entries bounded by $M/4$, of length $N - \log(M)/\epsilon$. Construct a workload as follows. The first $O(\log(M)/\epsilon)$ values are all the powers of $(1+\epsilon)$, in order, called "reference values." The next value is equal to M. Finally, the last $N - \log(M)/\epsilon$ values are the original vector. Run the first part of the algorithm on this workload, producing s.

Our goal now is to recover any v_j from s and j. To do this, consider the data that is all zeros except for a 1 at a position corresponding to a reference workload value of $(1+\epsilon)^k$ and another 1 at position corresponding to v_j. If $v_j > (1+\epsilon)^{k+1}$, then the best 3-bucket histogram gets v_j right and is zero everywhere else (getting the 1 at reference position $(1+\epsilon)^k$ wrong). Similarly, if $v_j < (1+\epsilon)^{k-1}$ the best 3-bucket histogram gets v_j wrong and $(1+\epsilon)^k$ right. It follows that we can learn v_j up to the factor $(1+\epsilon)^2 = (1+O(\epsilon))$. □

In particular, the lemma above implies:

Theorem 3.35. *For any $B \geq 3$ and any sufficiently small $\epsilon > 0$, if an algorithm*

represents in space $|s|$ *a workload* w *of length* N *and bound* M *and finds* $(1+\epsilon)$*-near-best* B*-bucket histograms with respect to* w*, then* $|s|$ *is at least the space needed to store* $(N - O(\log(M)/\epsilon))$ *counters of* $\Omega(\log(M)/\epsilon)$ *states.*

CHAPTER IV

Experiment and Application

In section 4.1, we will show how the workload affects the final result, and compare the result of our algorithm in section 3.4 to optimal. In section 4.2, we will briefly introduce a tool, "PADS", which processes ad-hoc data streams, and show, in detail, how to build and customize histograms (under uniform workload only) using our tool. We also unify the interface between generated library on system level (specified below) and user code on application level (specified below), so that future statistical profiling tools can be added into the system more easily.

4.1 Experiment Results

4.1.1 Environment

The experiment data is based on a trace containing one day's worth of all HTTP requests to the ClarkNet WWW server, which is a full Internet access provider for the Metro Baltimore - Washington DC area. Each tuple has five attributes as follows:

host name — time — request — return code — bytes transfered,

so {*haus.efn.org – [05/Sep/1995:00:00:00 -0400] "GET /pub/atomicbk/catalog/erotica .html HTTP/1.0" 200 11362*} is an example tuple. All the tuples come in a stream.

We consider each second as an index in the stream, and aggregate the total number

of bytes transferred in that second as the data. We carry out two experiments as follows.

- In the first experiment, we design some typical non-uniform workloads. We will compare the errors of optimal histograms oblivious to and aware of these workloads, to show the importance of taking workloads into account.
- In the second experiment, we aggregate the number of successful requests (with return code beginning with 2) in each second as the weight. This is the workload we would get for our data indexed by second if queries are initially generated uniformly by 5-tuples. Thus we derive a workload from the data rather than synthesize a workload. We will compare the error under our algorithm and the error under optimal representations, both oblivious to and aware of the workload. We test the near-linear algorithm only.

We define the following parameters in both experiments:

parameter	description	value
DIMEN	Length of the input stream.	1024
BLOCKNUM	Number of buckets allowed in the final representation.	Varies
ERROR	Error ϵ allowed in the final representation.	Varies

The program uses Visual C++ 6.0 as compiler, and runs under Windows XP.

4.1.2 Results from first experiment

In each table, the first row shows the errors of optimal histograms aware of workload and the second row shows the error ratios of optimal histograms oblivious to workload compared to the errors in first row and in the same column.

- $w_i = \frac{1}{i+1}$:

BLOCKNUM	1	2	4	8	16	32
Error of h_{opt} aware of w	41.2	35.4	33.2	29.3	24.0	17.7
Error ratio of h_{opt} oblivious to w	1.01	2.01	2.12	2.20	3.04	3.57

- $w_i = r^i$, with $w_0 = 1$ and $w_{N-1} = 500$:

BLOCKNUM	1	2	4	8	16	32
Error of h_{opt} aware of w ($\times 10^5$)	7.08	7.01	5.81	4.99	4.09	3.06
Error ratio of h_{opt} oblivious to w	1.00	1.07	1.21	1.36	1.94	2.72

- $w_i = r^i$, with $w_0 = 1$ and $w_{N-1} = 100$:

BLOCKNUM	1	2	4	8	16	32
Error of h_{opt} aware of w ($\times 10^5$)	1.95	1.93	1.67	1.44	1.21	0.94
Error ratio of h_{opt} oblivious to w	1.00	1.10	1.19	1.29	2.01	2.44

- $w_i = i$:

BLOCKNUM	1	2	4	8	16	32
Error of h_{opt} aware of w ($\times 10^6$)	4.92	4.89	4.34	3.88	3.32	2.65
Error ratio of h_{opt} oblivious to w	1.00	1.22	1.28	1.22	2.20	2.07

In general, the error ratio of the workload-oblivious optimal representation to the workload-aware optimal representation will first increase as the number of buckets goes up, then decrease sharply since both optimal histograms can get zero error when the number of buckets is as large as the dimension. From the above experiment results, we notice that: 1. The error ratio can go up to 3 or even higher under these typical workloads, while the BLOCKNUM $\ll N$. 2. The more dramatically the workload fluctuates, the bigger error ratio we can get. Considering that the real world workload can be much more irregular than the distributions we used in

these experiments, an algorithm aimed at finding representation aware of workload is necessary.

4.1.3 Result from second experiment

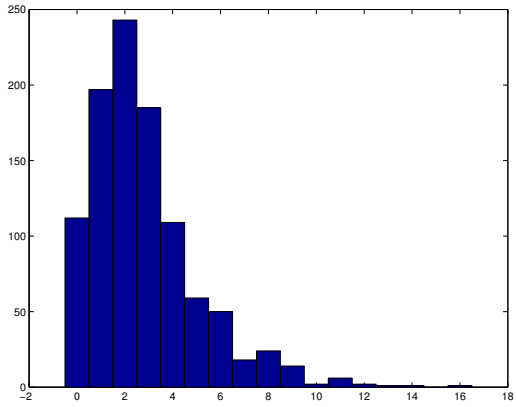

Figure 4.1: Distribution of Weight

In this experiment, we aggregate the number of successful requests in each second as the weight. We have the weight distribution shown in figure 4.1. The x-axis represents the weight, and the y-axis represents the number of indices having this weight.

Here are the results from our experiment:

BLOCKNUM	1	2	4	8	16
Error of h_{opt} a.o. w ($\times 10^4$)	3.82	3.78	3.01	2.49	2.06
Ratio of h_{opt} o.t. w	1.13	1.14	1.17	1.40	2.31
Ratio of h a.o. w ($\epsilon = 2$)	1.00 (1)	1.01 (1)	1.27 (1)	1.53 (1)	1.85 (1)
Ratio of h a.o. w ($\epsilon = 1$)	1.00 (1)	1.01 (1)	1.27 (1)	1.46 (3)	1.73 (3)
Ratio of h a.o. w ($\epsilon = 0.1$)	1.00 (1)	1.01 (1)	1.27 (1)	1.45 (3)	1.53 (6)
Ratio of h a.o. w ($\epsilon = 0.01$)	1.00 (1)	1.00 (2)	1.21 (3)	1.42 (3)	1.52 (8)

For each number B of buckets, the number in the first row is the actual error of B-

bucket optimal workload-aware histogram. Numbers from the second row to the last are error ratios of optimal workload-oblivious histogram and approximate workload-aware histograms with user defined error tolerance ϵ compared to the error of optimal workload-aware histogram. In our experiment, some approximate histograms use fewer than B buckets. Numbers in brackets show the number of buckets actually used in the result representation.

From the above result, we have some observations as follows.

- When we cut the parameter ERROR from 2 to 1, there are some relatively big improvements in some columns, while the improvements made by cutting ERROR from 0.1 to 0.01 are relatively small in those columns. So we may say, 0.1 is a good value for the parameter ERROR, and cutting it further may make little contribution to the final result representation.

- In many cases, the number of buckets actually used in the result representation is much smaller than the parameter BLOCKNUM. We may use the following technique to improve our algorithm in practice: Let $B_{\max}$ be the dimension, and $B_{\min}$ be the parameter BLOCKNUM set by the user. There is some B between $B_{\max}$ and $B_{\min}$, which leads to a result representation having BLOCKNUM buckets. We do binary search over all $B_{\min} \leq B \leq B_{\max} = N$ to find this B, which will put additional $O(\log N)$ iterations.

  For example. In our second experiment, when users ask for a 8-bucket histogram with error tolerance $\epsilon = 0.01$, they will actually get a 3-bucket histogram. To improve this, we set $B_{\min} = 8$, and search in the range $B \in [8, N)$. Finally, we can get a 8-bucket histogram when we set BLOCKNUM = 16 as the table shows.

4.2 Histogram Application in PADS

4.2.1 PADS Introduction

For the complete information about PADS, including documentation and downloads, see [43].

PADS is a declarative data description language that allows data analysts to describe both the physical layout of ad-hoc data sources and semantic properties of that data. Figure 4.2 is an illustration of its architecture. From such descriptions, the PADS compiler processes input data and produces basic libraries (Generated Library) automatically. High level applications for manipulating the data are provided in the PADS library, to which user has access. A possible application is to build statistical profiling tools over streaming data. As each data item comes, the tool accumulates it with previous result and gets updated statistics result like minimal item, maximal item, histogram, heavy hitters, etc. We call such tools as accumulators. The PADS system initially had only trivial accumulators, such as for computing the minimal or maximal item.

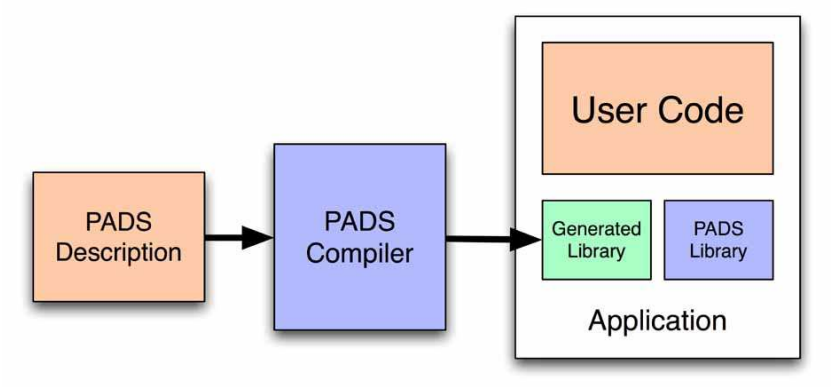

Figure 4.2: PADS Architecture

4.2.2 Example Histogram Result

Figure 4.3 is an example report for the length field of a web server log data.

```
*** Histogram Result ***
From 0 to 397, with height 4016.
From 398 to 398, with height 36122.
From 399 to 423, with height 5584.
From 424 to 424, with height 33250.
From 425 to 499, with height 3126.
*** Histogram Result ***
From 0 to 196, with height 3286.
From 197 to 197, with height 30430.
From 198 to 313, with height 3233.
From 314 to 314, with height 30430.
From 315 to 499, with height 3655.
*** Histogram Result ***
From 0 to 242, with height 3686.
From 243 to 243, with height 36122.
From 244 to 441, with height 3720.
From 442 to 442, with height 26074.
From 443 to 499, with height 7035.
*** Histogram Result ***
From 0 to 93, with height 4204.
From 94 to 94, with height 36122.
From 95 to 206, with height 3000.
From 207 to 210, with height 21496.
From 211 to 499, with height 3890.
. . . . . .
```

Figure 4.3: Portion of histogram report for length field of web server log data

In this particular run, an optimal 5-bucket histogram is built for every 500 values seen in the data source. From the result, we have some interesting observations. In the first three histograms, two spikes are detected for each. It means that there are at least two transactions with *bytes-transferred* ten times more than the average. In the last histogram, though there is only one spike detected, we also have four consecutive transactions with notably larger number of bytes transferred compared to the average. By adjusting number of buckets used, a network monitor may find this tool useful.

4.2.3 Customization

The histogram accumulator is the first nontrivial accumulator built for PADS. We now describe various customizations we provide to PADS users. Users are allowed to customize various aspects of histogram by setting the appropriate field in the histogram data structure. There are two kinds of customization. The fields `INIT_N` to `INIT_e` are related to building histograms itself. Users can choose to build optimal, approximate or equal-width histograms under L^1 or L^2 norm. The fields `INIT_scale` to `entry_t_fromFloat` are designed to help the algorithm fit the system better. Specifically,

- `INIT_N` is an unsigned integer denoting the number of values for histogram to summarize. If the number of values in the data source exceeds `INIT_N`, histograms will be built on each `INIT_N` data values respectively, until the end of data source is reached.

- `INIT_B` is an unsigned integer denoting the number of buckets in final histogram. As `INIT_B` increases, accuracy of final histogram approximation is increased, while more time and space is consumed.

- `INIT_M` is an unsigned integer denoting an upper bound of data values in data source. Time consumed increases polylogarithmically in `INIT_M`, so `INIT_M` can be set very large if little about data values in data source is known.

- `INIT_ISE` denotes whether buckets in the final histogram are required to be of the same width. If `INIT_ISE` is set to be non-zero, all buckets have equal width. In this case, the time needed is linear in `INIT_N`, and only constant space will be used. However, the resulting histogram will have less accuracy than histogram with near-optimal bucket boundaries.

- `INIT_ISO` denotes whether final histogram is required to be optimal or not. This parameter is valid only when `INIT_ISE` is zero. If `INIT_ISO` is set to be non-zero, the resulting histogram will be the most accurate one among all `INIT_B` bucket histograms. However, the time needed is cubic in `INIT_N`, which could be extremely large, and the space needed is linear in `INIT_N`, since all data values in each `INIT_N` section are required to be stored.

- `INIT_n` denotes whether the L^1 or the L^2 is used to measure the accuracy of final histogram. Currently, the L^1 measurement is supported only when all the data values are stored. In other words, it is supported only when `INIT_ISE` is zero, and `INIT_ISO` is non-zero.

- `INIT_e` denotes error tolerance of the final histogram. This parameter is valid only when non-optimal result is allowed, namely both `INIT_ISE` and `INIT_ISO` are zeroes. The final histogram will be guaranteed to be no worse than (1+`INIT_e`) times of the optimal one, but the time and space needed increase as the error tolerance decreases.

- `INIT_scale` specifies a number of least significant bits to drop in integer-valued data. The algorithm uses `INIT_scale` to avoid overflow. On reading data, it divides each data item by `INIT_scale` and rounds it up. On outputting histogram, it multiplies the height of each histogram bucket by `INIT_scale`.

- `INIT_maxPortion` specifies the size of an internal buffer. While the algorithm is running, there is a limited number of histograms that can be stored in the memory. Users are expected to either clear and print those histograms or store them into hard disk during the process. If a new histogram is required to be built while the number of stored histograms is already `INIT_maxPortion`, all

histograms in the memory will be cleared and a warning will be reported.

- `entry_t_toFloat` is a function pointer, converting any data type into a real number, which can be handled by the algorithm. Only types with well-defined conversion functions to real number are considered as meaningful types, and can be summarized correctly by histograms. Users are allowed to overload their own conversion function for each field.

- `entry_t_fromFloat` is a function pointer, converting real number back into its original data type. Only types with well-defined conversion functions from real number can be printed correctly. Users are allowed to overload their own conversion functions for each field.

Follows is an example of the use of `entry_t_toFloat` and `entry_t_fromFloat`. PADS provides a basic data type `PIP` as a structure of four real numbers $I_0, \cdots, I_3$ to store *"IP address"*. Users need to overload the above two converting functions to get statistics over `PIP` correctly. A possible way to convert `PIP` type p into a real number r is:

$$r = p.I_0 \times 255^3 + p.I_1 \times 255^2 + p.I_2 \times 255 + p.I_3$$

To report, users can recover every field of p exactly from r. In some application, users may care about network ID only and overload the converting function as $r = p.I_0 \times 255 + p.I_1$. The last two hosts ID will be hidden in the reported result in this case.

4.2.4 Operations

There are three kinds of histogram functions. Functions ENTRY_T_HIST_INIT to ENTRY_T_HIST_CLEANUP are related to setting up and cleaning up the running environment – including allocating space, setting proper parameters at the beginning

and deallocating space at the end. Function ENTRY_T_HIST_ADD updates current statistics each time a new item arrives. Functions ENTRY_T_HIST_REPORTFULL2IO to ENTRY_T_HIST_REPORTALL are used to print the resulting histograms either to screen or to an IO stream. All functions return a success indicator of type PERROR_T. Their behaviors declared for a PADS type are as follows:

- PERROR_T ENTRY_T_HIST_INIT (P_t *pads,entry_t_hist *h): Initializes a histogram data structure. This function must be called before any data can be added to a histogram.

- PERROR_T ENTRY_T_HIST_SETPARA (P_t *pads,entry_t_hist *h,P_hist *d_hist): Customizes histogram data structure. This function must be called to make any customization effective.

- PERROR_T ENTRY_T_HIST_RESET (P_t *pads,entry_t_hist *h): Reinitializes the histogram data structure. This function can be used to set any point of the data source as the start point of a new run. But it can not be used to reset any previously-defined parameters.

- PERROR_T ENTRY_T_HIST_CLEANUP (P_t *pads,entry_t_hist *h): Deallocates all memory associated with histogram.

- PERROR_T ENTRY_T_HIST_ADD (P_t *pads,entry_t_hist *h,Pbase_pd *pd,entry_t *rep,Puint32 *isFull): Inserts a data value. This function is called once for each new record. Any data type with an associated mapping function to Pfloat64 is considered as a meaningful type. This function tracks fields with meaningful type and legal values only. The output parameter isFull will be set nonzero, if the current data is the last one of a portion.

- Perror_t entry_t_hist_reportFull2io (P_t *pads,Sfio_t *outstr,const char *prefix, const char *what, int nst,entry_t_hist *h): Writes summary report for finished histograms to *outstr. In most cases, when this function is called, all stored histograms will be reported and the space will be released, while the current one won't.

- Perror_t entry_t_hist_reportAll2io (P_t *pads,Sfio_t *outstr,const char *prefix, const char *what, int nst,entry_t_hist *h): Writes summary report for all histograms to *outstr. When this function is called, all histograms will be reported and the space will be released.

- Perror_t entry_t_hist_reportFull (P_t *pads,const char *prefix,const char *what,int nst,entry_t_hist *h): Writes summary report for finished histograms to screen.

- Perror_t entry_t_hist_reportAll (P_t *pads,const char *prefix,const char *what,int nst,entry_t_hist *h): Writes summary report for all histograms to screen.

4.2.5 Unified interface for future accumulators

PADS provides a unified interface for all statistical profiling tools. Users are people who will use the following functions. For them, once they get familiar with one tool, they know how to use all others. Coders are people who provide the following functions. They can implement future tools more easily, focusing on the high-level algorithm only.

The statistical profiling tool function interface declared for a PADS type is as follows. Coders need to supply the body of each function, and replace SPT (statistical profiling tool) by meaningful tool names.

- ENTRY_T_SPT_INIT (P_t *pads,entry_t_SPT *h): Initializes.
- ENTRY_T_SPT_SETPARA (P_t *pads,entry_t_SPT *h,P_SPT *d_SPT): Customizes.
- ENTRY_T_SPT_RESET (P_t *pads,entry_t_SPT *h): Reinitializes.
- ENTRY_T_SPT_CLEANUP (P_t *pads,entry_t_SPT *h): Deallocates all memory associated.
- ENTRY_T_SPT_ADD (P_t *pads,entry_t_SPT *h,Pbase_pd *pd,entry_t *rep,Puint32 *isFull): Inserts a data value.
- ENTRY_T_SPT_REPORT2IO (P_t *pads,Sfio_t *outstr,const char *prefix, const char *what, int nst,entry_t_SPT *h): Writes summary report to *outstr.
- ENTRY_T_SPT_REPORT (P_t *pads,const char *prefix,const char *what,int nst, entry_t_SPT *h): Writes summary report to screen.

Some of the above input parameters are defined and supplied by PADS system. Coders need to supply the following two:

- ENTRY_T_SPT *H: H is a structure containing all inherent parameters needed to build the tool. Users don't have access to H.
- P_SPT *D_SPT: D_SPT is a structure containing all parameters, that are allowed to be modified by users. As in the histogram example, users can customize various aspects of the tool by setting the appropriate field of D_SPT, and calling ENTRY_T_SPT_SETPARA.

Figure 4.4 illustrates a sample use of the above functions to print a summary. This template consists of four parts. In declaration part, users are responsible for supplying a structure that contains all customization. The initialization part prepares the

system for running accumulators and remains the same for all accumulators. The high-level algorithm part is the core of this template program. Users first initialize its accumulator and set all customization, then use a loop to read and process each data item in order, and finally print out the result. In the clean up part, users should clean up all the memory used by both accumulator and PADS system.

We now show how to use the above interface to build a clustering tool. All functions specified above are similar to those fulfilled for building histogram, except the update function ENTRY_T_SPT_ADD which carries the main part of the algorithm. We provide a naive way to supply its body as follows since our purpose is to test the interface. Users are allowed to set two customizations. Parameter `INIT_CTYPE` is the underlying distribution of each cluster, which can be fully characterized by distribution function. Parameter `INIT_K` is the maximal number of clusters allowed. These two parameters decide the underlying model of the data source. When processing a new data item:

- If the total number of processed data items is less than a user-specified threshold `INIT_MAX`, we use traditional *K-means Algorithm* to get `INIT_K` clusters. For each cluster, we also compute its mean and variance based on user-specified distribution function.
- If not we compute, for each cluster, the probability of this new data item based on its own distribution function. We will add this item to the cluster where it has the highest probability.

Besides reporting the mean and variance of each cluster, we also report: 1. All elements in a cluster, if the number of elements in that cluster is less than a user-defined threshold. 2. An element, if the probability it falls in its own cluster based

```
#include "wsl.h"
#define DEF_INPUT_FILE"data/wsl"
int main(int argc, char** argv) {
-------------------------------------------------------------------
% Declarations
-------------------------------------------------------------------
  P_t *pads;
  Pio_disc_t *io_disc;
  P_SPT default_SPT;
  entry_t rep;
  entry_t_pd pd;
  entry_t_m mask;
  entry_t_SPT c;
  Puint32 isFull;
-------------------------------------------------------------------
  % Initialization
-------------------------------------------------------------------
  char *fname = DEF_INPUT_FILE;
  io_disc = P_nlrec_noseek_make(0);
  P_open(&pads, 0, io_disc);
  entry_t_init(pads, &rep);
  entry_t_pd_init(pads, &pd);
  entry_t_m_init(pads, &mask, P_CheckAndSet);
  if (P_ERR == P_io_fopen(pads, fname)) {
   error(2, "*** P_io_fopen failed ***");
   return -1;
  }
-------------------------------------------------------------------
  % Coders supplied high-level algorithm
-------------------------------------------------------------------
  entry_t_SPT_init(pads, &h);
  entry_t_SPT_setPara(pads, h, default_SPT);
  while (!P_io_at_eof(pads)) {
   entry_t_read(pads, &mask, &pd, &rep);
   entry_t_SPT_add(pads, &h, &pd, &rep), &isFull);
  }
  entry_t_SPT_report(pads, "", 0, 0, &h);
-------------------------------------------------------------------
  % Clean up
-------------------------------------------------------------------
  P_io_close(pads);
  entry_t_cleanup(pads, &rep);
  entry_t_pd_cleanup(pads, &pd);
  entry_t_SPT_cleanup(pads, &h);
  P_close(pads);
  return 0;
}
```

Figure 4.4: Sample use of a statistical profiling tool

on the distribution function is less than a user-specified threshold. We call any item satisfying either of the above two measurements a possible abnormality, and others normal items. Note that, to minimize the use of memory, after processing `INIT_MAX` data items we will delete all normal items and keep the mean and variance for each cluster only.

Figure 4.5 is an example report for a web server log data. In this particular run, a maximal number of three clusters are built for all the data values seen in the data source. This particular run finds some possible abnormalities. Users can modify the underlying data model easily by customization, so this tool is good for research purpose.

[Describing each tag arm of <top>.host]
==========
<top>.host.resolved : array nIP of Puint8
==========
Array lengths:
Clustering based distribution: User defined distribution.
mean 4, and variance 0, containing 4 elements.
==========
Possible abnormality based on probability 0.010000:
Possible abnormality based on clustering elements number 0.100000:
---
allArrayElts : uint8

Clustering based distribution: User defined distribution.
mean 128, and variance 77, containing 8 elements.
mean 136, and variance 0, containing 4 elements.
mean 97, and variance 0, containing 4 elements.
==========
Possible abnormality based on probability 0.010000:
Data (around): 49
Data (around): 207
Data (around): 49
Data (around): 207
Data (around): 50
Data (around): 207
Data (around): 50
Possible abnormality based on clustering elements number 0.100000:
==========
<top>.host.symbolic : array sIP of Pstring_SE
==========
Array lengths:
Clustering based distribution: User defined distribution.
mean 4, and variance 0, containing 7 elements.
==========
Possible abnormality based on probability 0.010000:
Possible abnormality based on clustering elements number 0.100000:
---
allArrayElts : string

Clustering based distribution: User defined distribution.
mean non defined., and variance non defined., containing 28 elements.
==========
Possible abnormality based on probability 0.010000:
Possible abnormality based on clustering elements number 0.100000:
. . . . . .

Figure 4.5: Portion of clustering report for web server log data

CHAPTER V

Private Approximate Heavy Hitters

5.1 Problem Statement

In this section, we address certain privacy issues that arise in processing massive data sets. Alice and Bob have two vectors a and b, and they want a summary for the vector sum $c = a + b$ privately in a precise sense we give below. First, we consider the Euclidean approximate heavy hitters problem, in which there is a parameter, B, and the players ideally want c_{opt}, the B largest terms in c. Approximate results with quality guarantees are allowable, if the techniques satisfy the privacy requirements in section 2.2.2. Then we extend our results to the problem of finding heavy hitters under L_1 norm and under L_2 after a transformation to another orthogonal basis.

5.2 Preliminaries

5.2.1 Parameters and Notation

Fix parameters N, M, B, k, ϵ. We will consider two players, Alice and Bob, who will have inputs, a and b respectively, that are vectors of length N taking integer values in the range $-M$ to $+M$. Throughout, we will be interested in summaries of size B for the vector $c = a + b$. For example, in the main result, we are interested ideally in the largest B terms of c. A vector c is written $c = (c_0, c_1, c_2, \ldots, c_{N-1}) = \sum c_j \delta_j$, where j is an *index*, c_j is a *value*, δ_j is the vector that is 1 at index j and 0

elsewhere, and $c_j\delta_j$, which can be implemented compactly and equivalently written as the pair (j, c_j), is a *term*, in which c_j is the *coefficient*.

We compare terms by the *magnitudes* of their coefficients, braking ties by the indices. That is, we will say that $(j, c_j) < (k, c_k)$ if $|c_j| < |c_k|$ or both $|c_j| = |c_k|$ and $j < k$. Thus all terms are strictly comparable. A heavy hitter summary is an expression of the form $\sum_{i\in\Lambda} \eta_i\delta_i$. If $|\Lambda|$ must be at most B, then the best heavy hitter summary c_{opt} for a vector c occurs where $\{(i, \eta_i) : i \in \Lambda\}$ consists of the B largest terms.

The Euclidean norm of c is $\|a\|_2 = \sqrt{\sum_i a_i^2}$ and the taxicab norm is $\|a\|_1 = \sum_i |a_i|$. The *support* $\text{supp}(a)$ of a vector a is the set of indices where a is non-zero, $\{i : a_i \neq 0\}$.

The parameter ϵ is a distortion parameter. We will guarantee summaries whose error is at most the factor $(1 + \epsilon)$ times the error of the best possible summary.

The parameter k is a security and failure probability parameter. Algorithms will be expected to succeed except with probability 2^{-k} and 2^{-k} will serve as an upper bound for the allowable statistical distance between indistinguishable distributions.

We will be interested in protocols that use communication $\text{poly}(B, \log(N), k, \log(M), 1/\epsilon)$, local computation $\text{poly}(B, N, k, \log(M), 1/\epsilon)$, and number of rounds that is constant.

5.2.2 Approximate Data Summaries

In the heavy hitters problem, we are given parameters B and N and the goal is to find the B largest terms in a vector c of length N. We will be interested in two approximate versions, parametrized also by ϵ. In the approximate heavy hitters problem, we want a summary $\widetilde{c} = \sum_{i\in\Lambda} \eta_i\delta_i$ such that $\|\widetilde{c} - c\| \leq (1 + \epsilon)\|c_{\text{opt}} - c\|$, where the norms are, respectively, 2-norms (in the EUCLIDEAN APPROXIMATE HEAVY HITTERS problem) and 1-norms (in the TAXICAB APPROXIMATE HEAVY HITTERS problem).

In order to describe previous algorithms that are relevant to us, we first need some definitions. Fix a vector $c = (c_0, c_1, c_2, \ldots, c_{N-1}) = \sum_{0 \le i < N} c_i \delta_i$, whose terms are $t_0 = (0, c_0), t_1 = (1, c_1), \ldots, t_{N-1} = (N-1, c_{N-1})$. Suppose the sequence $i'_0, i'_1, \ldots$ is a decreasing rearrangement of c, *i.e.*, $t_{i'_0} > t_{i'_1} > \cdots > t_{i'_{N-1}}$.

Definition 5.1. (Significant index.) Let $I \subseteq [0, N)$ be a set of indices. Then i is a *(I, θ)-significant index* for c if and only if $c_i^2 \ge \theta \sum_{j \in I} |c_j|^2$.

That is, an index is significant if the corresponding value is large compared with all the values in some set. In some of the algorithms below, we will find the largest term (if it is sufficiently large), subtract it off, then recurse on the residual signal. This motivates the following definitions.

Definition 5.2. (Qualified index set.) Fix parameters ℓ and θ. The set $Q = \{i'_0, i'_1, \ldots, i'_{m-1}\}$ is a *(ℓ, θ)-qualified index set* for c if and only if

- $m \le \ell$,
- $\forall j \in [0, m-1]$, i'_j is a $(\{i'_j, i'_{j+1}, \ldots, i'_{N-1}\}, \theta)$-significant index, and
- i'_m is NOT a $(\{i'_m, i'_{m+1}, \ldots, i'_{N-1}\}, \theta)$-significant index.

That is, a qualified index set consists of the largest possible length m for a prefix of $i'_0, i'_1, \ldots, i'_{m-1}$ such that, for each $j < m$, we have $c_{i'_j}^2 \ge \theta(c_{i'_j}^2 + c_{i'_{j+1}}^2 + c_{i'_{j+2}}^2 + \cdots + c_{i'_{N-1}}^2)$. In particular, if the terms happen to be in decreasing order to begin with, *i.e.*, if $|c_0| > |c_1| > \cdots$, then a qualified index set is $\{0, 1, 2, \ldots, m-1\}$ for the largest m such that, for each $j < m$, we have $c_j^2 \ge \theta(c_j^2 + c_{j+1}^2 + c_{j+2}^2 + \cdots c_{N-1}^2)$.

Note that for each ℓ, θ, and vector c, there is only one (ℓ, θ)-qualified index set for c. We use $Q_{c,\ell,\theta}$ to denote it. We sometimes write $Q_{\ell,\theta}$ when c is understood.

The following are straightforward.

Proposition 5.3. For any $\theta_1 < \theta_2$, Q_{ℓ,θ_2} set is a subset of Q_{ℓ,θ_1}.

Proposition 5.4. Fix parameters N, M, B, k, ϵ and vector c as above. If $\widetilde{c} = \sum_{i \in Q_{c,B,\frac{\epsilon}{B(1+\epsilon)}}} c_i \delta_i$, then $\|\widetilde{c} - c\|_2^2 \leq (1+\epsilon)\|c_{\text{opt}} - c\|_2^2$.

Proof. Assume without loss of generality that $|c_0| > |c_1| > \cdots$ and let $q = |Q_{c,B,\frac{\epsilon}{B(1+\epsilon)}}|$. If $q = B$, then $\widetilde{c} = c_{\text{opt}}$ and we are done. Otherwise we have

$$\begin{aligned} \|\widetilde{c} - c\|_2^2 &= \sum_{q \leq i < B} |c_i|^2 + \|c_{\text{opt}} - c\|_2^2 \\ &\leq B|c_q|^2 + \|c_{\text{opt}} - c\|_2^2 \\ &\leq \frac{\epsilon}{1+\epsilon}\|\widetilde{c} - c\|_2^2 + \|c_{\text{opt}} - c\|_2^2, \end{aligned}$$

whence

$$\left(1 - \frac{\epsilon}{1+\epsilon}\right)\|\widetilde{c} - c\|_2^2 \leq \|c_{\text{opt}} - c\|_2^2.$$

The result follows. □

We also need a modified version of Theorem 2.2 (sketch):

Theorem 5.5. *Fix parameters N, M, B, k, ϵ as above. Fix $\theta \geq \text{poly}(\log(N), \log(M), B, k, 1/\epsilon)^{-1}$. There is a pseudorandom matrix R with description of size* $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$, *and a corresponding algorithm that, from R and sketch Rc of a vector c, outputs a superset of $Q_{c,B,\theta}$, in time* $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$.

In particular, the number or rows in R and the size of the output is bounded by the expression $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$ in accordance with the time bound on the algorithm. The algorithm admits efficient SFE protocols from inputs a and b, as described in section 2.2.2, and can be modified to run privately in $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$ time.

Note that the algorithm returns a superset of $Q_{c,B,\theta}$ but that even $Q_{c,B,\theta}$ itself suffices for a good approximation.

Proof. As in theorem 2.2, we can get a set D that includes all terms with magnitude at least $\theta\|c\|_2^2$ in $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$ time. To get a superset of a qualified set, we subtract off D and repeat as long as new c_i are found that are large compared with the residual vector (*i.e.*, as long as D is not empty). At most $O(\log(MN))$ repetitions are needed since, after $O(\log(MN))$ repetitions, we have reduced $\|c\|_2^2$ from its initial value of at most M^2N to its least possible positive value of 1.

`non_private_Quelified_Set`

- Inputs: N, M, B, k, ϵ, sketch matrix R and Rc.
- Output: A superset of $Q_{c,B,\theta}$, containing $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$ items.

(i). $\widetilde{c} = c$.

(ii). Repeat

- $D =$`finding_heavy_hitters`$(\widetilde{c}, \theta)$.
- $\widetilde{c} = \widetilde{c} - \sum_{\widetilde{c}_i \in D} c_i \delta_i$
- Output D.

Until D is empty.

Figure 5.1: Non-private algorithm finding a Qualified Set

□

5.2.3 Privacy

In this section, we show that our result admits privacy requirement specified in section 2.2.2.

Consider definition 2.5. In our case, $g(a, b)$ will formally be the pair $(c_{\text{opt}}, \|c\|_2)$ and $\widetilde{g}(a, b)$ will be $\widetilde{c}$. We will informally say that we "approximate c_{opt} leaking only c_{opt} and $\|c\|_2$," since there is a simulator that takes c_{opt} and $\|c\|_2$ as input and simulates the approximate output $\widetilde{c}$ and the protocol messages. In Section 5.5, we will show that leaking $\|c\|_2$ is in some sense unavoidable for this problem. Equivalently, one could define $g(a, b)$ to be the pair $(c_{\text{opt}}, \|c_{\text{opt}} - c\|)$ and define $\widetilde{g}(a, b)$ to be the pair

$(\widetilde{c}, \|\widetilde{c} - c\|_{\sim})$, where $\|\cdot\|_{\sim}$ is an approximation to the Euclidean norm (see below).

For this result, we need the following standard definitions and previous results.

Definition 5.6 (Additive Secret Sharing). An intermediate value x of a joint computation is said to be *secret shared* between Alice and Bob if Alice holds r and Bob holds $x - r$, modulo some fixed large prime, where r is a random number independent of all inputs and outputs.

The Private Sample Sum problem is as follows.

Definition 5.7 (Private Sample Sum). At the start, Alice holds a vector a of length N and Bob holds a vector b. Alice and Bob also hold a secret sharing of an index i. At the end, Alice and Bob hold a secret sharing of $a_i + b_i$.

That is, neither the index i nor the value $a_i + b_i$ becomes known to the parties. Efficient protocols for this can be found (or can be constructed immediately from related results) in [42, 15], under various assumptions about the existence of Private Information Retrieval, such as in [8].

Proposition 5.8. There is a protocol `private-sample-sum` for the PRIVATE SAMPLE SUM problem that requires $\text{poly}(N, k)$ computation, $\text{poly}(\log(N), k)$ communication, and $O(1)$ rounds.

Our results also rely on the following protocol from [26], that privately approximates the Euclidean norm of the vector sum.

Proposition 5.9. (Private l_2 approximation, [26]) Suppose Alice and Bob have integer-valued vectors a and b in $[-M, M]^N$ and let $c = a + b$. Fix distortion ϵ and security parameter k. There is a protocol `private_norm_estimation` that computes an approximation $\|c\|_{\sim}$ to $\|c\|_2$ such that

- $\frac{1}{1+\epsilon}\|a+b\|_2 \leq \|a+b\|_\sim \leq \|a+b\|_2$.
- The protocol requires $\text{poly}(k\log(M)N/\epsilon)$ local computation, $\text{poly}(k\log(M)\log(N)/\epsilon)$ communication, and $O(1)$ rounds.
- The protocol is a private approximation protocol for $\|c\|$ in the sense of Definition 2.6.

Furthermore, the protocol's only access to a and b is through the matrix-vector products Ra and Rb, where R is a pseudorandom matrix known to both players.

5.3 Private Euclidean Heavy Hitters

5.3.1 Algorithm

We consider the setting in which Alice has signal a of dimension N, and Bob has signal b of the same dimension. Let $c = a + b$. Both parties want to learn a representation $\widetilde{c} = \sum_{t\in T_{\text{out}}} t$ such that $\|c-\widetilde{c}\|_2^2 \leq (1+\epsilon)\|c-c_{\text{opt}}\|_2^2$ and such that at most c_{opt} and $\|c\|_2$ is revealed. A protocol is given in Figure 5.2.

First, to gain intuition, we consider some easy special cases of the protocol's operation. For our analysis, assume that the terms in c are already positive and in decreasing order, $c_0 > c_1 > \cdots > c_{N-1} > 0$. We will be able to find the coefficient value of any desired term, so we focus on the set of indices. Let $I_{\text{opt}} = \{0,1,2,\ldots,B-1\}$ denote the set of indices for the optimal B terms, and I denote the set of indices the algorithm has recovered. Thus $Q_{c,B,\theta} \subseteq Q_{c,B,\frac{\theta}{1+\epsilon}} \subseteq I_{\text{opt}}$ and $Q_{c,B,\frac{\theta}{1+\epsilon}} \subseteq I$.

The ideal output is I_{opt}, though any superset of $Q_{c,B,\theta}$ suffices to get an approximation with error at most $(1+\epsilon)$ times optimal. This includes the set $I \supseteq Q_{c,B,\theta}$. The set I_B of the largest B terms indexed by I contains $Q_{c,B,\theta}$, so I_B is a set of at most B terms with error at most $(1+\epsilon)$ times optimal. If $|Q_{c,B,\theta}| = B$, then $I_B = Q_{c,B,\theta} = I_{\text{opt}}$, and I_B is a private and correct output.

`private_Euclidean_heavy_hitters`

- Known structural parameters: N, M, B, ϵ, k, which determine $\theta = \frac{\epsilon}{B(1+\epsilon)}$ and B'
- Individual inputs: vectors a and b, of length N, with integer values in the range $[-M, M]$.
- Output: With probability at least $1 - 2^{-k}$, a set T_{out} of at most B terms, such that $\left\|c - \sum_{t \in T_{\text{out}}} t\right\|_2^2 \leq (1+\epsilon)\left\|c - \sum_{t \in T_{\text{opt}}} t\right\|_2^2$.

(i). Exchange pseudorandom seeds (in the clear). Generate measurement matrices R_1 and R_2. Alice locally constructs sketches $R_1 a$ and $R_2 a = (R_2^0 a, R_2^1 a, \ldots R_2^{B-1} a)$, where the measurement matrix R_1 is used for a non-private Euclidean Heavy Hitters and the measurement matrix $R_2 = (R_2^0, R_2^1, \ldots, R_2^{B-1})$ is used for B independent repetitions of `private_norm_estimation`. Bob similarly constructs $R_1 b$ and $R_2 b$.

(ii). Using general-purpose SFE, do

- Use an existing (non-private) Euclidean Heavy Hitters protocol to get, from $R_1 a$ and $R_1 b$, a secret-sharing of a superset I of $Q_{c,B,\frac{\theta}{1+\epsilon}}$, in which I has exactly $B' \leq \text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$ indices. (Pad with arbitrary indices, if necessary.)

(iii). Use `private-sample-sum` to compute, from I, a, and b, secret-shared values for each index in I. Let T denote the corresponding set of secret-shared terms. (Both the index and value of each term in T is secret shared.) Enumerate I as $I = \{i_0, i_1, \ldots\}$ with $t_{i_0} > t_{i_1} > \cdots$.

(iv). Using SFE, do

- for $j = 0$ to $B - 1$
  - (a) From $R_2^j, R_2^j a, R_2^j b, t_0, t_1, \ldots, t_{i_{j-1}}$, sketch $r_j = c - (t_{i_0} + t_{i_1} + \cdots + t_{i_{j-1}})$ as $R_2^j r_j = (R_2^j a + R_2^j b - R_2^j(t_{i_0} + t_{i_1} + \cdots + t_{i_{j-1}}))$.
  - (b) use `private_norm_estimation` to estimate $\|r_j\|_2^2$ as $\|r_j\|_\sim^2$, satisfying $\frac{1}{1+\epsilon}\|r_j\|_2^2 \leq \|r_j\|_\sim^2 \leq \|r_j\|_2^2$.
  - (c) If $|c_{i_j}|^2 < \theta \|r_j\|_\sim^2$, break (out of for-loop)
  - (d) Output t_j

(v). For technical reasons, encode the pseudorandom seeds for R_1 and R_2 into the low-order bits of the output or (as we assume here) provide R_1 and R_2 as auxiliary output.

Figure 5.2: Protocol for the Euclidean Heavy Hitters problem

The difficulty arises when $|Q_{c,B,\theta}| < B$, in which case some of I_B may be arbitrary and should not be allowed to leak. So the algorithm needs to find a private set I_{out} of the largest $|I_{\text{out}}|$ terms with $Q_{c,B,\theta} \subseteq I_{\text{out}} \subseteq I_B$. The challenge is subtle. Let s denote $|Q_{c,B,\theta}|$. If the algorithm knew s, the algorithm could easily output $Q_{c,B,\theta}$, which is the indices of the top s terms, a correct and private output. Unfortunately, determining $Q_{c,B,\theta}$ or $s = |Q_{c,B,\theta}|$ requires $\Omega(N)$ communication (see Section 5.5), so we cannot hope to find $Q_{c,B,\theta}$ exactly. Non-private norm estimation can be used to find a subset I_{out} with $Q_{c,B,\theta} \subseteq I_{\text{out}} \subseteq Q_{c,B,\frac{\theta}{1+\epsilon}} \subseteq I_{\text{opt}}$, which is correct, but not quite private. Given $|I_{\text{out}}|$, the contents of $I_{\text{out}} \subseteq I_{\text{opt}}$ are indeed private, but the *size* of I_{out} is, generally, non-private. Fortunately, if we use a *private* protocol for norm estimation, $|I_{\text{out}}|$ remains private. We now proceed to a formal analysis.

5.3.2 Correctness and Cost

Theorem 5.10. *Protocol* `private_Euclidean_heavy_hitters` *requires* $\text{poly}(N, \log(M), B, k, 1/\epsilon)$ *local computation,* $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$ *communication, and* $O(1)$ *rounds.*

Proof. By existing work, all costs of Steps (i) to (iii) are as claimed. Now consider Step (iv). Observe that the function being computed in Step (iv) has inputs and outputs of size bounded by $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$ and takes time polynomial in the size of its inputs. In particular, the instances of `private_norm_estimation` do *not* start from scratch with a reference to a or b; rather, they pick up from the precomputed short sketches $R_2 a$ and $R_2 b$. It follows that this function can be wrapped with SFE, preserving the computation and communication up to polynomial blowup in the size of the input and keeping the round complexity to $O(1)$. □

We now turn to correctness and privacy. Let I_{out} denote the set of indices corre-

sponding to the set T_{out} of output terms.

Theorem 5.11. *Protocol* `private_Euclidean_heavy_hitters` *is correct.*

Proof. The correctness of Steps (ii) and (iii) follows from previous work. In Step (iv), we first show that $Q_{B,\frac{\epsilon}{B(1+\epsilon)}} \subseteq I_{\text{out}}$.

We assume that $\frac{1}{1+\epsilon}\|r_j\|_2^2 \leq \|r_j\|_\sim^2 \leq \|r_j\|_2^2$ always holds; by Proposition 5.9, this happens with high probability. Thus, if $|c_{i_j}|^2 \geq \frac{\epsilon}{B(1+\epsilon)}\|r_j\|_2^2$, then $|c_{i_j}|^2 \geq \frac{\epsilon}{B(1+\epsilon)}\|r_j\|_2^2 \geq \frac{\epsilon}{B(1+\epsilon)}\|r_i\|_\sim^2$.

By construction, $Q_{B,\frac{\epsilon}{B(1+\epsilon)}} \subseteq I$. A straightforward induction shows that, if $j \in Q_{B,\frac{\epsilon}{B(1+\epsilon)}}$, then iteration j outputs t_{i_j} and the previous iterations output exactly the set of the j larger terms in I.

By Proposition 5.4, since I_{out} is a superset of $Q_{B,\frac{\epsilon}{B(1+\epsilon)}}$, if $\widetilde{c} = \sum_{j\in I_{\text{out}}} c_{i_j}\delta_{i_j}$, then $\|\widetilde{c} - c\|_2^2 \leq (1+\epsilon)\|c_{\text{opt}} - c\|_2^2$, as desired. □

Before giving the complete privacy argument, we give a lemma, similar to the above. Suppose a set P of indices is a subset of another set Q of indices. We will say that P is a *prefix* of Q if $i \in P, t_j > t_i$, and $j \in Q$ imply $j \in P$.

5.3.3 Privacy

Lemma 5.12. *The output set* I_{out} *is a prefix of* $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$ *except with probability* 2^{-k}.

Proof. Note that $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$ is a subset of I and $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$ is a prefix of the universe, so $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$ is a prefix of I. The set I_{out} is also a prefix of I. It follows that, of the sets I_{out} and $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$, one is a prefix of the other (or they are equal). The relations among these sets are illustrated in figure 5.3.

So suppose, toward a contradiction, that $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$ is a proper prefix of I_{out}. Let $q = \left|Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}\right|$, so q is the least number such that i_q is *not* in $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$. If the

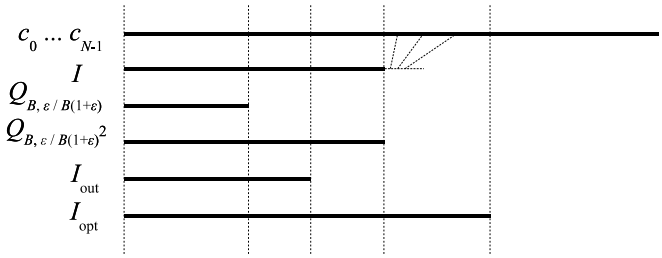

Figure 5.3: Illustration of the relations among sets

protocol halts before considering q, then $I_{\text{out}} \subseteq Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$, a contradiction. So, in particular, we may assume that $q < B$ (so the for-loop doesn't terminate early). Then, by definition of $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$, we have $|c_{i_q}|^2 < \frac{\epsilon}{B(1+\epsilon)^2}\sum_{j\geq q}|c_{i_j}|^2$. It follows that

$$\begin{aligned} |c_{i_q}|^2 &< \frac{\epsilon}{B(1+\epsilon)^2}\sum_{i\geq q}|c_i|^2 \\ &= \frac{\epsilon}{B(1+\epsilon)^2}\|r_q\|_2^2 \\ &\leq \frac{\epsilon}{B(1+\epsilon)}\|r_q\|_{\sim}^2. \end{aligned}$$

Thus the protocol halts without outputting t_q, after outputting exactly the elements in $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$. □

Finally, we turn to privacy.

Theorem 5.13. *Protocol* `private_Euclidean_heavy_hitters` *leaks no more than* $\|c\|_2^2$ *and* c_{opt}.

Proof. With the random inputs R_1 and R_2 encoded into the output, it is straightforward to show that Protocol `private_Euclidean_heavy_hitters` is a private protocol in the traditional sense that the protocol messages leak no more than the inputs and outputs. This is done by composing simulators for `private-sample-sum` and SFE. It remains only to show that we can simulate the joint distribution on $(\widetilde{c}, R_1, R_2)$

given as simulator-input c_{opt} and $\|c\|$. We will show that R_1 is indistinguishable from independent of the joint distribution of $(\widetilde{c}, R_2)$, which we will simulate directly.

First, we show that R_1 is independent. Except with probability $2^{-\Omega(k)}$, the intermediate set I is a superset of $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$ and the norm estimation is correct. In that case, the protocol outputs a prefix of $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$ and we get identical output if I is replaced by $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$. Also, $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$ can be constructed from c_{opt} and $\|c\|_2$. Since the protocol proceeds without further reference to R_1, we have shown that the pair $(\widetilde{c}, R_2)$ is indistinguishable from being independent of R_1. It remains only to simulate $(\widetilde{c}, R_2)$.

Note that the output $\widetilde{c}$ does depend non-negligibly on R_2. If $|c_{i_j}|^2$ is very close to $\theta\|r_j\|_2^2$, then the test $|c_{i_j}|^2 < \theta\|r_j\|_{\sim}^2$ in the protocol may succeed with probability non-negligibly far from 0 and from 1, depending on R_2, since the distortion guarantee on $\|r_j\|_{\sim}^2$ is only the factor $(1 \pm \epsilon)$.

The simulator is as follows. Assume that the terms in c_{opt} are $t_0, t_1, \ldots, t_{B-1}$ in decreasing order, $t_0 > t_1 > \cdots > t_{B-1}$. For each $j \leq B$, compute $E_j = \|c - (t_0 + \cdots + t_{j-1})\|_2^2 = \|c\|_2^2 - \|t_0 + \cdots + t_{j-1}\|_2^2$ and then run the `private_norm_estimation` simulator on input E_j and ϵ to get a sample from the joint distribution $(\widetilde{E}_j, \widetilde{R}_2)$, where $\widetilde{E}_j$ is a good estimate to E_j and $\widetilde{R}_2$ is distributed indistinguishably from R_2. Note that $\widetilde{R}_2$ is used in the `private_norm_estimation` protocol and is part of the view in that. Our simulator then outputs t_{i_j} if $|c_{i_j}|^2 \geq \frac{\epsilon}{B(1+\epsilon)}\widetilde{E}_j$, and halts, otherwise, following the final for-loop of the protocol. It also outputs $\widetilde{R}_2$. Call the output of the simulator $\widetilde{s} = \sum_j t_{i_j}\delta_{i_j}$.

Again using the fact that a prefix of $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$ is output, if $j \in Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$, then $i_j = j$; *i.e.*, the j'th largest output term is the j'th largest overall, so that, if j is output, we have $E_j = \|r_j\|_2^2$. Thus $(\widetilde{E}_j, \widetilde{R}_2)$ is distributed indistinguishably

from $(\|r_j\|_{\sim}^2, R_2)$. The protocol finishes deterministically using I and $(\|r_j\|_{\sim}^2)$'s and the simulator finishes deterministically using $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$ and $(\widetilde{E}_j)$'s, but, since the protocol output is identical if I is replaced by $Q_{B,\frac{\epsilon}{B(1+\epsilon)^2}}$, the distributions on output $(\widetilde{c}, R_2)$ of the protocol and $(\widetilde{s}, \widetilde{R}_2)$ of the simulator are indistinguishable. □

5.3.4 Conclusion

In summary,

Theorem 5.14. *Suppose Alice and Bob hold integer-valued vectors a and b in $[-M, M]^N$, respectively. Let B, k and ϵ be user-defined parameters. Let $c = a + b$. Let T_{opt} be the set of the largest B terms in c. There is an protocol, taking a, b, B k and ϵ as input, that computes a representation $\widetilde{c}$ of at most B terms such that:*

- $\|\widetilde{c} - c\|_2 \leq (1+\epsilon)\|c_{\text{opt}} - c\|_2$.
- *The algorithm uses* $\text{poly}(N, \log(M), B, k, 1/\epsilon)$ *time,* $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$ *communication, and $O(1)$ rounds.*
- *The protocol succeeds with probability $1 - 2^{-k}$ and leaks only c_{opt} and $\|c\|_2$ with security parameter k.*

Corollary 5.15. With the same hyptotheses and resource bounds, there is a protocol that computes $\widetilde{c}$ and an approximation $\|\widetilde{c} - c\|_{\sim}$ to $\|\widetilde{c} - c\|_2$ such that $\frac{1}{1+\epsilon}\|\widetilde{c} - c\|_2 \leq \|\widetilde{c} - c\|_{\sim} \leq \|\widetilde{c} - c\|_2$ and the protocol leaks only c_{opt} and $\|\widetilde{c} - c\|_2$.

Proof. Run the main protocol and output also $\|\widetilde{c} - c\|_{\sim}$, which is computed in the course of the main protocol. The main simulator can compute the joint distribution on $(\widetilde{c}, R_1, R_2)$ given c_{opt} and $\|c\|$, so it remains only to show that we can simulate $\|c\|$ given c_{opt} and the distribution on $\|\widetilde{c} - c\|_2$.

Note that $\|c\|_2^2 = \|\widetilde{c} - c\|_2^2 + \|\widetilde{c}\|_2^2$. Our simulator outputs $\|c\|_{\sim}^2 = \|\widetilde{c} - c\|_2^2 + \|\widetilde{s}\|_2^2$ as an estimation of $\|c\|_2^2$, where $\widetilde{s}$ is the output of the main simulator. Since $\widetilde{c}$ is

distributed indistinguishably from $\widetilde{s}$, $\|c\|_2^2$ and $\|c\|_{\sim}^2$ are computationally indistinguishable. □

5.4 Extensions

5.4.1 Extension to Manhattan Heavy Hitters

In this section, we show that our result of Euclidean approximation can be extended to approximate manhattan heavy hitters.

Lemma 5.16. *Let $\widetilde{c}$ be the output of* `private_Euclidean_heavy_hitters`*. If $\|c - \widetilde{c}\|_2 \leq (1+\epsilon)\|c - c_{\text{opt}}\|_2$, then $\|c - \widetilde{c}\|_1 \leq (1+\sqrt{B\epsilon})\|c - c_{\text{opt}}\|_1$.*

Proof. Let (i, c_i) be the largest term which is not in $Q_{B,\frac{\epsilon}{B(1+\epsilon)}}$. From Theorem 5.14 we know $(\sum_{i\leq j<B} c_j^2)^{\frac{1}{2}} < \sqrt{\epsilon}(\sum_{B\leq j<N} c_j^2)^{\frac{1}{2}}$. Using the fact that $\frac{1}{\sqrt{|\operatorname{supp}(x)|}}\|x\|_1 \leq \|x\|_2 \leq \|x\|_1$ for any signal x, we get

$$\frac{1}{\sqrt{B}}\sum_{i\leq j<B}|c_j| \leq \left(\sum_{i\leq j<B} c_j^2\right)^{\frac{1}{2}} \leq \sqrt{\epsilon}\left(\sum_{B\leq j<N} c_j^2\right)^{\frac{1}{2}} \leq \sqrt{\epsilon}\sum_{B\leq j<N}|c_j|.$$

Thus we have

$$\begin{aligned}
\|c - \widetilde{c}\|_1 &\leq \sum_{i\leq j<N}|c_j| \\
&= \sum_{i\leq j<B} c_j + \sum_{B\leq j<N}|c_j| \\
&= (\sqrt{\epsilon B}+1)\sum_{B\leq j<N}|c_j| \\
&= (\sqrt{\epsilon B}+1)\|c - c_{\text{opt}}\|_1.
\end{aligned}$$

□

Theorem 5.17 follows directly:

Theorem 5.17. *Suppose Alice and Bob hold integer-valued vectors a and b in $[-M, M]^N$, respectively. Let B, k and ϵ be user-defined parameters. Let $c = a+b$. Let T_{opt} be the set of the largest B terms in c. There is an protocol, taking a, b, M, N, B, k and ϵ as input, and computes a representation $\widetilde{c}$ of at most B terms such that:*

- $\|\widetilde{c} - c\|_1 \leq (1+\epsilon)\|c_{\text{opt}} - c\|_1$.
- *The algorithm uses* $\text{poly}(N, \log(M), B, k, 1/\epsilon)$ *time,* $\text{poly}(\log(N), \log(M), B, k, 1/\epsilon)$ *communication, and* $O(1)$ *rounds.*
- *The protocol succeeds with probability* $1 - 2^{-k}$ *and leaks only* c_{opt} *and* $\|c\|_2$ *with security parameter* k.

5.4.2 Extension to other Orthonormal Bases

In this section, we consider other orthonormal bases, such as the Fourier basis. Alice and Bob hold vectors a and b as before, and want the B largest Fourier terms—frequencies and corresponding coefficient values. The exact problem requires $\Omega(N)$ communication, so they settle for an approximation, namely, they want a B-term Fourier representation $\widetilde{c}$ such that $\|\widetilde{c} - c\|_2 \leq (1+\epsilon)\|c_{\text{opt}} - c\|_2$, where c_{opt} is the best possible B-term Fourier representation.

We note that a straightforward generalization of our main result solves this problem privately and efficiently. Alice and Bob locally compute the inverse Fourier transform $F^{-1}a$ and $F^{-1}b$ of their vectors a and b. Because the Fourier transform and its inverse are linear, $x = F^{-1}c = F^{-1}a + F^{-1}b$. Alice and Bob now want to compute an approximation to the ordinary heavy hitters for the vector x. Suppose the result is $\widetilde{x}$. Then $\widetilde{x}$ is the compact collection of Fourier terms and $\widetilde{c} = F\widetilde{x}$ is the corresponding approximate representation of c. By the Parseval equality, since the Fourier basis is orthogonal, for any y, we have $\|y\|_2 = \|Fy\|_2 = \|F^{-1}y\|_2$. It follows

that $\|\widetilde{c} - c\|_2 \leq (1+\epsilon)\|c_{\text{opt}} - c\|_2$ if and only if $\|\widetilde{x} - x\|_2 \leq (1+\epsilon)\|x_{\text{opt}} - x\|_2$, so the algorithm is correct when transformed to the Fourier domain. It also follows that leaking $\|c\|_2$ is equivalent to leaking $\|Fc\|_2$, so the algorithm is private when transformed to the Fourier domain. Alice and Bob require the additional overhead of computing a Fourier transform locally, which fits within the overall budget.

5.5 Lower Bounds

In this Section, we show some lower bounds for problems related to our main problem, such as computing an approximation to c_{opt} without leaking $\|c\|_2$. The results are straightforward, but we include them to motivate the approximation and leakage of the protocols we present.

Theorem 5.18. *There is an infinite family of settings of parameters M, N, B, k such that any protocol that computes the Euclidean norm exactly on the sum c of individually-held inputs a and b uses communication $\Omega(N)$. Similarly, any protocol that computes the exact Heavy Hitters or computes the qualified set $Q_{c,1,1}$ exactly uses communication $\Omega(N)$.*

Proof. Consider the set disjointness problem, which requires $\Omega(N)$ communication [36]. Alice and Bob hold $\{0,1\}$-valued vectors a and b of length N such that each of a and b has exactly $(N/4)$ 1's and the supports are either disjoint or intersect in exactly one index. The task is to determine the intersection size. Then, if $c = a + b$, we have $\|c\|_2^2 = N/2$ or $\|c\|_2^2 = N/2 + 3$, depending on the size of the intersection, so a protocol for $\|c\|_2$ can be used to solve the set disjointness problem. Similarly, finding the one largest heavy hitter solves the set disjointness problem.

Now consider vectors of length $N+1$ in which indices 0 to $N-1$ directly code an instance of set disjointness as above and index N has a value that is always

$\sqrt{N/2+2}$. Then $|Q_{c,1,1}| = 1$ or $|Q_{c,1,1}| = 0$ depending on the norm over indices 0 to $N-1$, which requires communication $\Omega(N)$ to determine. □

The above theorem motivates our study of *approximate* heavy hitters, for which there are protocols with exponentially better communication cost than the exact heavy hitters problem. The next theorem motivates leaking the Euclidean norm, by showing that *any* efficient protocol for the approximate heavy hitters problem leaks the Euclidean norm on all instances within a class.

Theorem 5.19. *There is an infinite family of settings of parameters M, N, B, k, ϵ such that any protocol that solves the Euclidean Heavy Hitters problem on the sum c of individually-held inputs a and b, leaking only c_{opt}, uses communication $\Omega(N)$. Furthermore, for an infinite class of inputs in which $\|c\|_2$ is not constant, any such protocol either computes $\|c\|_2$ or uses communication $\Omega(N)$.*

Proof. Consider vectors c of one of two cases, given by random permutations of the following vectors:

$$\begin{cases} (2N, \overbrace{1, 1, \ldots, 1}^{N/2-1}, 0, 0, \ldots, 0), & \text{(case 1)} \\ (2N, \overbrace{N, N, \ldots, N}^{N/2-1}, 0, 0, \ldots, 0), & \text{(case 2).} \end{cases}$$

Fix $B = 1$ and $\epsilon \gg 1/N$. A correct protocol finds the top term in case 1. In case 2, it turns out that the correctness requirement is vacuous, but, fortunately, the privacy requirement is useful. A protocol leaking only c_{opt} must behave indistinguishably in cases 1 and 2 since c_{opt} is the same, so a private protocol reliably finds the the top coefficient in case 2. Since a protocol for case 2 can be used to solve the set disjointness problem, such a protocol uses $\Omega(N)$ bits of communication. In particular, any protocol either behaves differently on the two cases—thereby computing $\|c\|_2$ for inputs in the union of the two cases—or uses communication $\Omega(N)$. □

Note that a correct protocol also finds the top term under Manhattan measurement in case 1. So the above theorem also shows that it is impossible in some cases to solve the approximate *Manhattan* heavy hitters problem efficiently without leaking the *Euclidean* norm.

Although the class of inputs above is contrived, the (implied) parameter settings are natural, *i.e.*, $\log(M), \log(N), B, k, 1/\epsilon$ can be made to be polynomially related, etc.

CHAPTER VI

Optimal Histograms on Probabilistic Data Streams

6.1 Statement of Problems

In this section, we introduce problems regarding probabilistic data streams [31], [48]. We focus on the primary question: can the algorithms used for deterministic data streams be applied to build optimal histograms in the probabilistic case? The authors in [32] gave algorithms over probabilistic data streams to estimate aggregations such as MEDIAN, DISTINCT and REPEAT-RATE by instantiation. In [13], by using modified streaming sketch synopses, the authors managed to estimate DISTINCT and extend it to approximate "join sizes" over a static probabilistic database.

Definition 6.1. (Uncertain Domain. [31]) Let $\mathcal{B}$ denote a discrete base domain, and let $\perp$ denote a special symbol that does not belong to $\mathcal{B}$. An uncertain domain $\mathcal{U}$ over $\mathcal{B}$ is the set of all probability distribution functions, or pdfs, over $\mathcal{B} \cup \{\perp\}$.

In contrast with traditional data streams, in which each value v is a real number, each v in a probabilistic data stream is a pdf encoding the belief of each b in the domain $\mathcal{B}$. In this section, let $\mathcal{B} = [1, R]$ so that each pdf can be described in the following three ways:

- (Enumeration [31]) Each v is described as a series of pairs $\{\langle i_1, p_1\rangle, \cdots, \langle i_l, p_l\rangle\}$, where $\Pr_v[i_s] = p_s$ for $s = 1, \cdots, l$, and $\Pr_v[\perp] = 1 - \sum_s p_s$. This model is an

extension of the probabilistic database model. The authors in [31] give some motivation.

- (Function) Each v is described as a probability density function $p_v(x)$ for $x \in \mathcal{B}$. We define $\Pr_v[x] = p_v(x)$. This model can find applications as well. For example, a marketer may know that each week's sales follow the Gaussian distribution with certain parameters. He is interested in computing aggregations over these distributions like: What are the average sales for Tuesday?

- (Series of samples) Each v is described as a series of samples $\{i_1^v, \cdots, i_l^v\}$, where $i_s^v \in \mathcal{B}$ for $s = 1, \cdots, l$. We define $\Pr_v[x] = \frac{|s:i_s^v=x|}{l}$ for $s = 1, \cdots, l$. The motivation comes from cascaded queries. For example, computing the median exactly requires linear space, but space-efficient algorithms may output an approximate result. Running these algorithms repeatedly will give a series of sample approximate medians. We may want to compute aggregations over these samples like: What is the median of these sample medians?

In this section, we modify the existing algorithms which are applied to traditional data streams to the above three models of probabilistic data stream, and get some preliminary results.

6.2 Summary Structure

Authors in [31] use possible streams to define aggregation query semantics over probabilistic data streams. Following their definition, we define the frequency of an item $x \in \mathcal{B}$ for a probabilistic data stream $v_1, \cdots, v_n$ as $Q(x) = \sum_{s=1}^{n} \Pr_{v_s}[x]$. For each x, we are interested in $Q(x)$ and we call this a point query.

There are many structures and corresponding algorithms to efficiently approximate point queries over traditional data streams. Authors in [17] summarized some

of them and introduced a CR-PRECIS structure in detail. They also described how to use the result of point queries to approximate other statistics, like heavy hitters.

Definition 6.2. CR-PRECIS: In [17], the authors proposed the structure as follows: the structure is parameterized by height k and width t. They chose t consecutive prime numbers $k \leq p_1 < p_2 < \cdots < p_t$ and keep a collection of t tables T_j, for $j = 1, \cdots, t$ where T_j has p_j integer counters. To answer point query, for each $x \in \mathcal{B}$, they updated for each table:

$$\text{for } j = 1 \text{ to } t \text{ do } \{T_j[x \bmod p_j] = T_j[x \bmod p_j] + 1\}$$

and computed $\min_{j\in[1,N]}\{T_j[x \bmod p_j]\}$ as the approximation $\tilde{f}_x$. For any $x \neq y$, if $T_j[x \bmod p_j] = T_j[y \bmod p_j]$, we say that x and y are conflicting items with respect to j.

We will modify their CR-PRECIS structure in this section. Let n be length of the stream.

Proposition 6.3. *Using a* CR-PRECIS *structure with height* $k \geq 12$ *and width* $t \geq 1$*, there is an algorithm estimating the frequency of* $x \in \mathcal{B}$ *as* $\hat{f}_x$ *in* $O(t(t + \frac{k}{\ln k}) \log(t + \frac{k}{\ln k})(\log n))$ *bits, satisfying:*

$$0 \leq \hat{f}_x - f_x \leq \frac{(\log_k |\mathcal{B}| - 1)}{t}(n - f_x)$$

Proof. In a probabilistic data stream, each item v contains information about more than one $x \in \mathcal{B}$. For each cell in the j^{th} table, if v is in the form of "sample", we update it with the sum of all frequencies. Otherwise, we update it with the maximal frequency of all conflicting items with respect to j. We still output the minimal value of all cells containing x as the approximation $\hat{f}_x$. We analyze the update time for each v in all three forms as follows.

- Enumeration: We maintain a temporary structure T' of the same size as T. For each $\langle i_s, p_s \rangle$ pair, for $j = 1$ to $t : \{T'_j[i_s \bmod \text{prime}_j] = \max(p_s, T'_j[i_s \bmod \text{prime}_j])\}$. We keep track of all cells in T' that have been updated by an array. After reading the entire v, we add all updated cells of T' into T, and clear T' to 0. The update time is $O(t|v|)$.

- Function: Let $T_j[c]$ be the c^{th} cell in the j^{th} table. Let $x = \arg \max_{p_v(s)}(s \bmod \text{prime}_j = c)$, we update $T_j[c] = +x$. The update time for each v depends on the nature of the function, but, for widely-used probability density functions, each x can be computed in constant time. Therefore, the update time for each v is bounded by the size of the table, which is $O(t(t + \frac{k}{\ln k}) \log(t + \frac{k}{\ln k}))$.

- Series of samples: We maintain the same structure T' as in "enumeration" case, but update for each i_s^v for $j = 1$ to t: $\{T'_j[i_s^v \bmod \text{prime}_j] = +1\}$. We keep track of $|v|$ by an integer and all updated cells by an array. After reading the entire v, we divide all updated cells in T' by $|v|$ and add them back to T. Then we clear T' to 0. The update time is $O(t|v|)$.

We use $\tilde{f}_x$ as the the approximation given by Definition 6.2. From the construction, it is easy to see that $\hat{f}_x = \tilde{f}_x$ in the "sample" case and $f_x \leq \hat{f}_x \leq \tilde{f}_x$ in the other two cases. Note that the length of the stream $n = \sum_{i \in \mathcal{B}} f_i$. Our result follows the result given in [17] directly.

□

6.3 Histogram

In this section, we consider building histograms $h = \langle h_1, \cdots, h_n \rangle$ over probabilistic data streams.

Definition 6.4. (Error Of A Histogram [31]) The error of a deterministic histogram h over a probabilistic data stream $\mathcal{P} = \langle v_1, \cdots, v_n \rangle$ is defined to be the expected value of difference between h and the stream with respect to distribution on the possible streams.

We rewrite the above definition concretely:

Definition 6.5. The error of h over $\mathcal{P}$ is $\sum_{i=1}^{n} \sum_{x \in \mathcal{B}} \{\Pr_{v_i}[x](h_i - x)^2\}$ for L_2 norm and $\sum_{i=1}^{n} \sum_{x \in \mathcal{B}} \{\Pr_{v_i}[x]|h_i - x|\}$ for L_1 norm.

Notice that, for every given $x \in \mathcal{B}$ in v_i, the sum of probabilities over all possible streams containing that x is $\Pr_{v_i}[x]$. Simple calculation shows that definition 6.4 and 6.5 are equivalent.

6.3.1 Histograms Under L_2 Norm

According to definition 6.5, let h_{opt} with minimal $\sum_{i=1}^{n} \sum_{x \in \mathcal{B}} \{\Pr_{v_i}[x](h_i - x)^2\}$ be the optimal B-bucket histogram over $\mathcal{P}$ under L_2 norm. To show that h_{opt} is also an optimal histogram over $\mathcal{P}'$ under L_2 norm, where $\mathcal{P}' = \langle \text{mean}(v_1) \cdots \text{mean}(v_n) \rangle$, we need the following lemma, which describes the relationship between the errors of mean and a specific height over certain distribution:

Lemma 6.6. *Let $e = mean(v_i) = \sum_{s=1}^{l} i_s p_s$ and h be a height, we have:*

$$(h - e)^2 + \sum_{s=1}^{l} \{p_s(e - i_s)^2\} = \sum_{s=1}^{l} \{p_s(h - i_s)^2\}$$

Proof. Note that $(h - e)^2 = (h - e)^2 \sum_{s=1}^{l} p_s$, so the left side can be reduced to:

$$\sum_{s=1}^{l} p_s[(e - i_s)^2 + (h - e)^2] = \sum_{s=1}^{l} p_s[2e^2 + i_s^2 + h^2 - 2ei_s - 2he]$$

The right side is:

$$\sum_{s=1}^{l} p_s[i_s^2 + h^2 - 2hi_s]$$

Comparing these, it remains to show $\sum_{s=1}^{l} p_s[2e^2+i_s^2+h^2-2ei_s-2he] = \sum_{s=1}^{l} p_s[i_s^2+h^2-2hi_s]$, which is equivalent to showing $\sum_{s=1}^{l} p_s[e^2+hi_s-ei_s-he] = 0$. We have: $\sum_{s=1}^{l} p_s[e^2] = e^2$; $\sum_{s=1}^{l} p_s[hi_s] = he$; $\sum_{s=1}^{l} p_s[ei_s] = e^2$ and $\sum_{s=1}^{l} p_s[he] = he$. So the original equation holds. □

Though we only show that the property holds under enumeration form, it can be shown under function and sample forms using similar proofs. Lemma 6.6 tells us to find the optimal h_{opt}, we need to minimize $\sum_{i=1}^{N}[h_i - \text{mean}(v_i)]^2$. So the optimal b-bucket histogram over $\mathcal{P}'$ is also an optimal b-bucket histogram over $\mathcal{P}$.

Lemma 6.7. *A near-optimal B-bucket histogram h' over $\mathcal{P}'$ is also a near optimal B-bucket histogram over $\mathcal{P}$.*

By using "near-optimal", we mean the error under this histogram is within $(1+\epsilon)$ factor of the error under the corresponding optimal histogram.

Proof. Let $A = \sum_{i=1}^{N} \sum_{x \in \mathcal{B}} \{\Pr_{v_i}[x](\text{mean}(v_i) - x)^2\}$. Let h and h_{opt} be the near optimal and optimal B-bucket histograms over $\mathcal{P}'$. Let e and e_{opt} be the errors of h and h_{opt} over $\mathcal{P}$ respectively. From lemma 6.6, we have:

$$\begin{aligned} e &= \sum_{i=1}^{N}[h_i - \text{mean}(v_i)]^2 + A \\ &\leq (1+\epsilon)\sum_{i=1}^{N}[h_{\text{opt},i} - \text{mean}(v_i)]^2 + A \\ &< (1+\epsilon)\sum_{i=1}^{N}[h_{\text{opt},i} - \text{mean}(v_i)]^2 + (1+\epsilon)A \\ &= (1+\epsilon)e_{\text{opt}}. \end{aligned}$$

□

We can now extend the algorithm described in [23] to compute near-optimal B-bucket histograms over $\mathcal{P}$. The only additional work is to compute the mean for each

data item online. It requires $O(|v|)$ time per item in the forms of "Enumeration" and "Sample". Update time for each item in the form of "Function" depends on the property of the density function. But for widely-used functions, constant time is enough.

Proposition 6.8. *There is an algorithm that, given parameters B, N, M, ϵ, and probabilistic data stream $\mathcal{P}$ with $\|\mathcal{P}\|_w^2 \leq M$, outputs a B-bucket histogram h with*

$$\|\mathcal{P} - h\|^2 \leq (1 + O(\epsilon))\|\mathcal{P} - h_{\text{opt}}\|^2,$$

where h_{opt} is the best possible B-bucket histogram representation to $\mathcal{P}$. The algorithm uses space $\left(\frac{B \log N \log M}{\epsilon}\right)^{O(1)}$ and time $c_1 N|v| + \left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$, where c_1 and c_2 are two constants.

6.3.2 Histograms Under L_1 Norm

In this section, let h_{opt} with minimal $\sum_{i=1}^{N} \sum_{x \in \mathcal{B}} \{\Pr_{v_i}[x]|h_i - x|\}$ be the optimal B-bucket histogram for $\mathcal{P}$ under L_1 norm. A straightforward heuristic to get a near-optimal h is to summarize $\mathcal{P}'' = \langle \text{median}(v_1) \cdots \text{median}(v_n) \rangle$ and get an approximate B-bucket histogram over P'' as we did in last section under L_2 norm. However, this heuristic does not work under L_1 norm. We will give a counterexample as follows.

Example 6.9. We want to build a 2-bucket histogram over the following probabilistic stream: $\mathcal{P} = \langle \{\langle 5, \frac{1}{5}\rangle, \langle 25, \frac{1}{5}\rangle, \langle 45, \frac{1}{5}\rangle, \langle 65, \frac{1}{5}\rangle, \langle 85, \frac{1}{5}\rangle\}, \{\langle 155, 1\rangle\}, \{\langle 47, 1\rangle\}\rangle$. As a summarization, $\mathcal{P}'' = \langle 45, 155, 47\rangle$. The optimal histogram over $\mathcal{P}''$ has its boundary between the first two items (e.g. $\langle 45, 101, 101\rangle$) while the optimal histogram over $\mathcal{P}$ has its boundary between the last two items (e.g. $\langle 100, 100, 47\rangle$). By modifying the item which is misclassified, the difference between the above two histograms can be arbitrarily large. Any algorithm based on $\mathcal{P}''$ only can not get a quality guaranteed approximation over $\mathcal{P}$.

To solve the problem, we need, for each distribution, more information than the median. Let $x \in \mathcal{B}$ be a value in the domain of the i^{th} item, and $f_i(x) = \sum_{x \in \mathcal{B}} \{\Pr_{v_i}[x]|h_i - x|\}$ be the L_1 difference between the i^{th} distribution and the height x, i.e., if we use x as the estimation of the i^{th} distribution, $f(x)$ will be the error under L_1 norm. Obviously, $f_i(x)$ has its minimal value when $x = \text{median}(v_i)$.

Lemma 6.10. *There is an algorithm, which takes a distribution v_i and a value $x \in \mathcal{B}$ as input, first outputs a structure in linear time and $\left(\frac{1}{\epsilon}\right)^{O(1)}$ space, and then estimates $f_i'(x)$ from the structure in $\left(\frac{1}{\epsilon}\right)^{O(1)}$ time with $(1-\epsilon)f_i(x) \leq f_i'(x) \leq f_i(x)$.*

Proof. Figure 6.1 shows the shape of a possible f_i function. We observe the following properties of f_i:

- The function f_i has its minimal value when $x = \text{median}(v_i)$. That is, $f_i(\text{median}(v_i))$ $= \min_{h_i} \sum_{x \in \mathcal{B}} \{\Pr_{v_i}[x]\,|h_i - x|\}$.
- Suppose distribution v_i contains m values with non-zero probability, then its corresponding $f_i(x)$ consists of $m+1$ segments, while each value with non-zero probability is a connector of two adjacent segments.
- Suppose x falls in the segment with slope k. Then we have $|k| = \left|-\sum_{y=-\infty}^{x} \Pr_{v_i}[v_i](y) + \sum_{y=x}^{\infty} \Pr_{v_i}[v_i](y)\right|$. Notice that $|k| \in [0,1]$.
- The function f_i is convex.

We build the structure in the following way: Let $f_i'(x) = k(x - x_{\min}) + \min$ as shown in figure 6.1. For the segments on the right of $x_{\min}$, we:

- Record $x_{\min} = \text{median}(v_i)$ and record corresponding $f_i(x_{\min})$.
- Record the slope k' and the leftmost point of the rightmost segment.

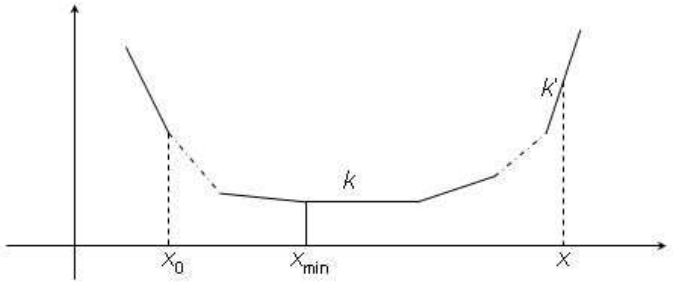

Figure 6.1: The L_1 difference between v_i and a height, as function of the height

- Check segments from right to left until we reach $x_{\min}$. If the current slope $k < (1-\epsilon)k'$, record k, the leftmost point of this segment and set $k' = k$.

The above process can be done in linear time. Since $|k| \in [0,1]$ for all segments, we record information for $(\frac{1}{\epsilon})^{O(1)}$ segments.

To estimate x, suppose the slope of the segment containing x is k and $x_0 \leq x$ with slope k_0, error y_0 and $x_1 > x$ are two adjacent points recorded in the structure. We estimate $f_i(x)$ by $f_i'(x) = k_0(x - x_0) + y_0$. We have:

$$\begin{aligned} f_i'(x) &= k_0(x-x_0) + y_0 \\ &\geq (1-\epsilon)k(x-x_0) + y_0 \\ &\geq (1-\epsilon)k_0(x-x_0) + (1-\epsilon)y_0 \\ &\geq (1-\epsilon)f_i(x) \end{aligned}$$

Similar for segments on the the left of $x_{\min}$. □

Suppose the structure for each distribution v_i is s_i, we have the following property:

Lemma 6.11. *Let $e_{\min}$ be the distance between optimal height and $v_j \cdots v_k$ under L_1 norm. There is an algorithm, which takes $s_j \cdots s_k$ as input, and outputs an approximation height h in amortized $O(s_i)$ time per item, satisfying $(1-\epsilon)e_{\min} \leq e' \leq e_{\min}$, where e' is the distance between h and $v_j \cdots v_k$ under L_1 norm.*

Proof. We compute h as the height minimizing $\sum_{i=j}^{k} f_i'(h)$. Notice that, the minimum can only appear at positions that are recorded in $s_j \cdots s_k$, since each f_i' is a convex function. Our complexity result follows.

Let $h_{\min}$ be the optimal height, $e'_{\min} = \sum_{i=j}^{k} f_i'(h_{\min})$, and $e = \sum_{i=j}^{k} f_i(h)$. From lemma 6.10, we have:

$$(1-\epsilon)e_{\min} \leq (1-\epsilon)e \leq e' \leq e'_{\min} \leq e_{\min}$$

□

Now we give a heuristic algorithm as follows:

(i). We build a bucket robust histogram h_r as defined in Definition 3.3 over $\mathcal{P}''$, i.e., the median of each item.

(ii). In the same run, we also record the structure s_i for each distribution.

(iii). Use any polynomial-time algorithm to compute the optimal histogram to h_r.

Building structures requires pre-processing time only. Since computing the optimal 1-bucket histogram requires amortized $O(s_i)$ time instead of constant time, we will have an extra $(\frac{1}{\epsilon})^{O(1)}$ factor to the original $\left(\frac{B \log N \log M}{\epsilon}\right)^{c_2}$ time. Also, to record each structure, we again have an extra $(\frac{1}{\epsilon})^{O(1)}$ factor to the space we used.

To make this algorithm from a heuristic one to a $(1+\epsilon)$- approximation, the only thing left is to show a robust histogram over $\mathcal{P}''$ is also a robust histogram over $\mathcal{P}$. We will leave it as an open question for now.

CHAPTER VII

Conclusion

In this thesis, we gave results for three problems and explained an implementation we have done on one of the results.

We built a near-optimal B-bucket histogram, admitting non-uniform workload, in polylogarithmic post-processing time and space on streaming data. We gave deterministic algorithms to build a B-bucket histogram, in polylogarithmic post-processing time and space on probabilistic data streams. We also showed how to compute quality-guaranteed heavy hitters of the sum of two vectors in a private way and in polylogarithmic communication. We applied our histogram results in the implementation part.

There is one open question: how to give guarantees to our heuristic histogram result on probabilistic data streams under L_1 measurement.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *Proc. ACM SIGMOD*, 1999.

[2] S. Acharya, P. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *SIGMOD Conference*, pages 574–576, 1999.

[3] Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the k th-ranked element. In *eurocrypt04*, pages 40–55, 2004.

[4] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. *J. Comput. Syst. Sci.*, 64(3):719–747, 2002. Earlier version in PODS '99.

[5] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. Earlier version in STOC '96.

[6] A. Beimel, P. Carmi, K. Nissim, and E. Weinreb. Private approximation of search problems. In *Proc. 38th Annual ACM Symposium on the Theory of Computing*, pages 119–128, 2006.

[7] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th Annual ACM Symposium on the Theory of Computing*, pages 1–10. ACM Press, 1988.

[8] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology — EUROCRYPT '99*, LNCS 1592, pages 404–414. Springer-Verlag, 1999.

[9] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *ICALP02*, pages 693–703, 2002.

[10] S. Chawla, C. Dwork, F. McSherry, A. Smith, and H. Wee. Toward privacy in public databases. In *Proc. Second Theory of Cryptography Conference*, pages 363–385, 2005.

[11] C. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proc. ACM SIGMOD*, 1994.

[12] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45:965–981, 1998. Earlier version in FOCS '95.

[13] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *Proc. ACM SIGMOD*, 2007.

[14] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. In *Proc. ACM Principles of Database Systems*, pages 296–306, 2003.

[15] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. N. Wright. Secure multiparty computation of approximations. *Transactions on Algorithms*, 2006. To appear. Earliar version in ICALP 2001.

[16] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology — EUROCRYPT '04*, LNCS 3027, pages 1–19. Springer-Verlag, 2004.

[17] S. Ganguly and A. Majumder. Cr-precis: A deterministic summary structure for update data streams. Technical Report arXiv:cs/0609032v3, IIT Kanpur, October 2006.

[18] V. Ganti, M. Lee, and R. Ramakrishnan. Icicles–self-tuning samples for approximate query answering. In *Proc. VLDB*, 2000.

[19] L. Getoor, B. Taskar, and D. Koller. Selectivity estimation using probabilistic models. In *Proc. ACM SIGMOD*, 2001.

[20] A. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proc. 34th Annual ACM Symposium on the Theory of Computing*, pages 389–398, 2002.

[21] A. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Near-optimal sparse fourier representations via sampling. In *Proc. 34th Annual ACM Symposium on the Theory of Computing*, pages 152–161, 2002.

[22] S. Guha. A note on wavelet optimization, 2004. `http://www.cis.upenn.edu/~sudipto/notes/wavelet.pdf.gz`.

[23] S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Histogramming data streams with fast per-item processing. In *Proc 29'th ICALP*, pages 681–692, 2002.

[24] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proc. ACM STOC*, pages 471–475, 2001.

[25] S. Halevi, E. Kushilevitz, R. Krauthgamer, and K. Nissim. Private approximations of NP-hard functions. In *Proc. 33th Annual ACM Symposium on the Theory of Computing*, pages 550–559, 2001.

[26] P. Indyk and D. P. Woodruff. Polylogarithmic private approximations and efficient matching. In *Proc. Third Theory of Cryptography Conference*, pages 245–264, 2006.

[27] Y. Ioannidis. The history of histograms (abridged). In *Proc. VLDB*, 2003.

[28] Y. Ioannidis and S. Christodoulakis. Optimal histograms for limiting worst-case error propagation in the size of join results. *ACM Trans. Database Syst.*, 18(4):709–748, 1993.

[29] Y. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *Proc. SIGMOD*, pages 233–244, 1995.

[30] H.Ṽ. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proc. VLDB*, pages 275–286, 1998.

[31] T.S. Jayram, Satyen Kale, and Erik Vee. Efficient aggregation algorithms for probabilistic data. In *SODA07*, 2007.

[32] T.S. Jayram, Andrew McGregor, S. Muthukrishan, and Erik Vee. Estimating statistical aggregates on probabilistic data streams. In *PODS*, 2007.

[33] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. SIGMOD*, 2001.

[34] A. Konig and G. Weikum. Combining histograms and parametric curve fitting for feedback driven query result size estimation. In *Proc. VLDB*, 1999.

[35] E. Kushilevitz and Y. Mansour. Learning decision trees using the fourier sprectrum. In *Proc. 23th Annual ACM Symposium on the Theory of Computing*, pages 455–464, 1991.

[36] E. Kushilevitz and N. Nisan. *Communication complexity*. Cambridge University Press, 1997.

[37] E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *Proc. 38th IEEE Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[38] Y. Lindell and B. Pinkas. Privacy preserving data mining. *J. Cryptology*, 15(3):177–206, 2002. Earlier version in Crypto '00.

[39] Y. Matias and D. Urieli. Optimal workload-based wavelet synopses,. Technical report, TAU, February (revised, July) 2004.

[40] Y. Matias, J. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proc. ACM SIGMOD*, 1998.

[41] S. Muthukrishnan. Nonuniform sparse approximation theory with Haar wavelets. Technical report, DIMACS, 2004.

[42] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *Proc. 33th Annual ACM Symposium on the Theory of Computing*, pages 590–599, 2001.

[43] Complete information about pads system. `http://www.padsproj.org`.

[44] V. Poosala. *Histogram-based estimation techniques in database systems*. PhD thesis, Univ of Wisconsin, 1997.

[45] V. Poosala, P. Haas, Y. Ioannidis, and E. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proc. ACM SIGMOD*, 1996.

[46] L. Qiao, D. Agrawal, and A.Ẽl Abbadi. Rhist: adaptive summarization over continuous data streams. In *Proc. CIKM*, pages 469–476, 2002.

[47] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill Professional Press, 2003.

[48] A. Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *Proc. ICDE*, 2006.

[49] M. Stillger, G. Lohman, V. Markl, and M. Kandil. Leo - db2's learning optimizer. In *Proc. VLDB*, pages 19–28, 2001.

[50] A. Yao. Protocols for secure computation. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.