

Abstract

Pass-Efficient Algorithms for Clustering

Kevin Li-Chiang Chang

2006

The proliferation of computational problems involving massive data sets has necessitated the design of computational paradigms that model the extra constraints placed on systems processing very large inputs. Among these algorithmic paradigms, the *pass-efficient* model captures the constraints that the input may be much too large to fit in main memory for processing, and that system performance is optimized by sequential access to the data in storage. Thus, in the pass-efficient model of computation, an algorithm may make a constant number of sequential passes over read-only input while using a small amount of random access memory. The resources to be optimized are memory, number of passes, and per element processing time.

We give pass-efficient algorithms for clustering and finding structure in large amounts of data. Our algorithms have the property that the number of passes allotted is an input parameter to the algorithm. We answer questions regarding the intrinsic tradeoffs between the number of passes used by a pass-efficient algorithm and the amount of random access memory required. Our algorithms use adaptive sampling techniques that are quite general and can be used to solve many massive data set problems.

The first family of clustering problems that we consider is *learning mixtures of distributions*. In these problems, we are given samples drawn according to a probability distribution known as a mixture of distributions, and must reconstruct the density function of the original mixture. Our algorithms show tradeoffs between the number of passes and the amount of memory required: if the algorithm makes a few

extra passes, the amount of memory required drops off sharply. We also prove lower bounds on the amount of memory needed by any ℓ -pass randomized algorithm, thus showing that our tradeoff is nearly tight. The second family of clustering problems that we consider is the combinatorial optimization problem of *facility location* and related problems. Our pass-efficient algorithms for this problem exhibit the same sharp tradeoffs as our algorithm for learning mixtures of distributions.

We also give clustering algorithms that are not in the streaming model for partitioning a graph to approximately minimize certain natural objective functions.

Pass-Efficient Algorithms for Clustering

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Kevin Li-Chiang Chang

Dissertation Director: Ravi Kannan

December 2006

Copyright © 2006 by Kevin Li-Chiang Chang
All rights reserved.

Contents

Acknowledgements	iv
1 Introduction	1
1.1 Pass-Efficient and Streaming Algorithms	4
1.1.1 Definition of the Models	4
1.1.2 A brief survey of streaming and pass-efficient algorithms	5
1.2 Our Contribution	8
1.2.1 Generative Clustering: Learning mixture models in multiple passes	9
1.2.2 Clustering by Combinatorial Optimization: Approximating facility location in multiple passes	13
1.2.3 Clustering by Combinatorial Optimization: Graph Partitioning	17
1.3 Notes to the Reader	19
2 Preliminaries	21
2.1 General Purpose Algorithmic Tools	22
2.1.1 Bounds on Tails of Distributions	22
2.2 Pass-Efficient and Streaming Algorithms	26
2.2.1 Approximating the ℓ_1 norm of a stream of dynamic updates	26
2.3 Communication Complexity	28

2.3.1	r -round communication complexity	30
2.3.2	Connections to streaming and pass-efficient algorithms	31
3	Pass-Efficient Algorithms for Generative Clustering	33
3.1	Preliminaries	34
3.2	A Single Pass Algorithm	36
3.3	Multiple Passes	43
3.3.1	The Algorithm	44
3.3.2	Testing intervals for uniformity: Proof of Theorem 10.	52
3.4	Matching bounds on memory usage of randomized algorithms	57
3.4.1	A Lower Bound	58
3.4.2	An Upper Bound	60
3.5	Learning mixtures of linear distributions	63
3.5.1	Testing intervals for linearity.	63
3.5.2	The algorithm	67
3.6	Learning mixtures of uniform distributions in two dimensions	68
3.6.1	Testing rectangles for verticality	68
3.6.2	The Algorithm	72
4	Pass-Efficient Algorithms for the Facility Location Problem	75
4.1	Review of Indyk’s sublinear k -median algorithm	76
4.2	The Algorithm: Pass-Efficient FL	78
4.2.1	Overview	78
4.2.2	The Algorithm and proof of correctness	80
4.2.3	Approximate selection in one pass	87
4.3	k -Facility Location	89
4.4	Lower bounds for multiple pass clustering	91

4.5	Remaining Questions	93
5	Graph Partitioning Algorithms	94
5.1	Motivation: A network security problem	95
5.2	Proof of Theorem 42	98
5.2.1	Preliminaries: Sparse Cuts	98
5.2.2	The Algorithm	102
5.3	Hardness of Approximation	109
6	Conclusion and Future Work	112
6.1	Summary of Results	113
6.2	Massive Data Set Algorithms: Future Directions	114
	Bibliography	116

Acknowledgements

Foremost, I must thank my adviser, Ravi Kannan, for his invaluable mentoring. His vision and technical assistance inspired this dissertation and guided my growth as a scholar. I am extremely grateful not only for his role in my intellectual development, but also for his empathy and encouragement: he was patient and supportive during the hard times, and was happy for me during the good times.

The theory professors have all been wonderfully helpful during my time at Yale; they were all more than happy to speak with me and offer their advice. In addition, Jim Aspnes helped shape Chapter 5. Dana Angluin and Joan Feigenbaum patiently served as readers for my thesis. Dan Spielman volunteered much help in my professional development.

I am indebted to Sanjeev Arora for inspiring me to pursue research in theoretical computer science and for his research collaboration, to Petros Drineas for reading my thesis and for kindly taking an active interest in my professional development, and to my co-author Alex Yampolskiy who helped shape Chapter 5.

I thoroughly enjoyed my time at Yale primarily because of the friends and classmates that I met. Over four years, they have helped me grow immensely as a person; I am especially thankful to Fred Shic, for listening to me whenever I needed to talk, without fail.

Lastly, nothing would have been possible without the loving support of my sister,

mother, and father. I dedicate this thesis to my mother, who always thought about me before herself.

Kevin Chang

May 2006

Chapter 1

Introduction

The modern phenomenon of our growing ability to store extremely large amounts of information on computer systems has been accompanied by the proliferation of such massive data sets for many different problem domains. These data sets occur in such disparate areas as the analysis of communications networks, scientific computing, analysis of commercial transactions, and bioinformatics. Massive data sets present new and fascinating challenges for the design of algorithms that can efficiently cope with the added stresses of computing on inputs of sizes that are often measured in terabytes.

The memory space of a typical computer system may be abstractly viewed as organized into two broad categories: random access memory, which can be accessed very efficiently but has a limited capacity, and storage, which relatively has a very large capacity but is very slow to access. Storage devices usually consist of secondary storage in the form of disks, or sometimes tertiary storage, in the form of tapes and optical devices. The main difficulties of massive data set computation arise from the constraint that sufficiently large data sets cannot fit into the main memory of a computer and thus must be placed into storage. Computing thus cannot occur with

the data *entirely in memory*, but rather must be *performed with the data on disk*.

It is well known that reading and writing to secondary and tertiary storage is the performance limiting factor for computations on data in storage, and that the throughput of the I/O subsystem is optimized for sequential access of the data in storage devices. Tapes are obvious examples of storage devices that exhibit this phenomenon, but disks also exhibit this behavior: random access requires that the disk head physically move and then wait for the disk to rotate to the appropriate position before reading can occur. These penalties, known respectively as seek times and rotational latency, are very expensive and can be avoided by reading the data sequentially without seeking. For massive data set calculations, frequent random access will severely lower performance.

The role of theoretical computer science is to provide rigorous studies of the intrinsic possibilities and impossibilities of computability and resource usage. Algorithms designed in the traditional theoretical models of computation are often not suitable for practical massive data set computation, because many features of traditional computation cannot be reconciled with the stringent constraints of massive data set computation. For example, the traditional models of computation do not make any distinction between storage and memory, effectively assuming that all computation occurs in memory and that an arithmetic operation will cost as much as access to any data element. The blurring of storage and memory can yield algorithms that require large amounts of random access memory and pay no premium for frequent random access of the data. Lastly, polynomial running time is often not a good criterion for efficiency. From a practical perspective, if the size of the input is very large, even quadratic running time can be prohibitively expensive.

Thus, in order to reason rigorously about the performance of algorithms that adhere to the strict design principles of massive data set computation, it is necessary

to abstract the notions of storage and memory, and to precisely define the meaning of “very little random access.” A theoretical model of computation that addresses the concerns outlined above is the *pass-efficient* model of computation. In this model, the data is presented as a read-only array that must be accessed in a small number of sequential passes using a small amount of memory. Related theoretical models for massive data sets include streaming algorithms, sliding window algorithms, and sublinear algorithms that rely on sampling the data.

In the pass-efficient model for massive data sets, the two resources that we are most concerned with are the number of passes over the input array and the amount of memory required by the algorithm. A natural question to ask is: *what is the intrinsic power of each pass?* For example, can a two pass algorithm solve a problem using much less space than a one pass algorithm? Is this *tradeoff* between the number of passes and the amount of memory intrinsic to the problem or is it just an artifact of an inefficient algorithmic use of passes? In this dissertation, we will consider these types of questions for problems with natural massive data set applications.

Some of the most important massive data set problems involve *clustering* and *learning*. Clustering is a vaguely defined class of problems in which data elements must be partitioned into a number of groups such that “similar” elements are placed in the same group and “dissimilar” elements are placed in different groups. The choice of a precise meaning for “similar” and the choice of a clustering objective is often very specific to the given application domain, and can be a difficult judgment to make. Massive data set applications of clustering are of great interest to practitioners of data mining and information retrieval, who attempt to extract structure and learn relations from large amounts of data; clustering is a major subfield of such studies. Motivated by these practical concerns, much work on massive data set algorithms for clustering has appeared in the theoretical computer science literature.

Clustering problems in computer science can be roughly classified as either *generative clustering* or *worst-case clustering by combinatorial optimization*. In generative clustering, we assume that the data are samples drawn according to k different random processes, and our task is to learn attributes of the different processes. In worst-case clustering by combinatorial optimization, we do not assume anything about the input; instead we partition the data points in order to optimize some pre-defined objective function of the partition. We will consider pass-efficient algorithms for generative clustering and both pass-efficient and conventional combinatorial optimization algorithms for worst-case clustering.

1.1 Pass-Efficient and Streaming Algorithms

1.1.1 Definition of the Models

In this dissertation, we are concerned with the *pass-efficient model* of computation. In this model, the input is given as a read-only array of n elements. The order of the elements is arbitrary, and is assumed to be adversarial; in particular, the elements cannot be assumed to be in sorted order. Algorithms may only access the data through a constant number of complete, sequential passes over the entire array. The algorithm is ideally allowed $O(1)$ bits of random access memory (although $o(n)$ might be necessary) in which to perform and store intermediate calculations and sketches of the data. While processing each element of the input array, the algorithm may only use computing time that is ideally independent of n , but after each pass, it is allowed more computing time (typically $o(n)$). We are most concerned with three resources: the number of passes, the amount of random access memory required, and the per-element-compute time. Multiple pass algorithms were first considered in [59], and were first called pass-efficient in [23].

The class of pass-efficient algorithms that perform their calculations in a single pass are known as *streaming* algorithms (these algorithms are typically allowed polylogarithmic per-element-compute time). If the data arrive online at such a high rate that one cannot (or does not care to) store all of it, then one pass algorithms are especially appealing because they only need to read each input element in the data stream once; an input element may be immediately deleted after being processed.

1.1.2 A brief survey of streaming and pass-efficient algorithms

Streaming and pass-efficient algorithms have been considered in the theoretical computer science literature for many different problems with natural applications to massive data sets. The first work in the streaming model was due to Munro and Paterson [59], who developed single and multiple pass algorithms for median finding, selection, and sorting. Henzinger, Raghavan, and Rajagopalan [38] considered computing basic graph properties related to query processing, in one or multiple passes, and proved lower bounds on the amount of memory needed.

Computing statistics is a very appealing massive data set problem. Calculating frequency moments and related problems in the streaming model is a fundamental and well-studied area. In the frequency moment problem, we are given a data stream consisting of pairs $\langle i, a \rangle$, where $i \in [n]$ is an index of a bucket and $a \in [m]$ is a number to place in the bucket. Here, $[n]$ denotes the set $\{1, \dots, n\}$. If $a_i = \sum_{\langle i, a \rangle} a$ is the sum of the numbers placed in bucket i , then the k th frequency moment is given by $\sum a_i^k$. Alon, Matias, and Szegedy [3] developed small space streaming algorithms for approximating the frequency moments for $k = 0, 2$, and were the first to use communication complexity to prove lower bounds of $\Omega(n^{1-5/k})$ for $k \geq 5$ on the

amount of memory required by streaming algorithms. Indyk [43] then presented small space streaming algorithms for the case where $0 \leq k \leq 2$. The lower bounds on memory were improved by Bar-Yossef, Jayram, Kumar, and Sivakumar [9] and Charkrabarti, Khot and Sun [12] to $\tilde{\Omega}(n^{1-2/k})$ ¹ for $k \geq 2$. Indyk and Woodruff [45] then presented algorithms with matching upper bounds. Algorithms for the related problems of finding the L^1 distance between two vectors given in adversarial order in a stream and finding the length of a vector given as a stream of dynamic updates were given by Feigenbaum, Kannan, Strauss, and Viswanathan [30] and Indyk [43], respectively.

A number of clustering algorithms have appeared in the streaming model. We will examine this body of work more carefully in Section 1.2.2. Streaming algorithms exist for k -center [14], k -median [16, 36], and clustering to minimize the sum of cluster diameters [17].

Histogram construction and maintenance is an important problem in the database literature; the problem has also been studied in the context of streaming algorithms. Guha, Koudas, and Shim [35] considered the following problem: given a stream of n real numbers a_1, \dots, a_n in sorted order, construct a piecewise constant function F with k pieces such that $\sum_i |F(i) - a_i|^2$ is minimized. They gave a $1 + \epsilon$ factor approximation algorithm using space $O(k^2/\epsilon \log n)$. A variant of the problem is that of histogram maintenance [32, 33], in which the a_i s are presented as a stream of updates of the form “add 7 to a_3 ”, and the algorithm must always be able to output a histogram for the current state of the a_i s. Gilbert *et al.* [33] provide a one pass $1 + \epsilon$ approximation algorithm that uses time per update and total space polynomial in $(k, 1/\epsilon, \log \|a\|_1$ or $\log \|a\|_2, \log n)$.

Frieze, Kannan, and Vempala [31] first proposed sampling the rows and columns

¹The notation $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ is asymptotic notation that suppresses polylog factors

of an $m \times n$ matrix to find the best low-rank approximation of a matrix to within a small additive error in time independent of m and n . Deshpande, Rademacher, Vempala, and Wang [20] showed that successive rounds of adaptive sampling will reduce the error exponentially in the number of rounds. Since samples from the matrix can be drawn in a single pass, these algorithms can be adapted to the pass-efficient model in a straightforward fashion. Drineas *et al.* [22] provided a sampling algorithm, which can be implemented in the pass-efficient model, to compute an approximate singular value decomposition of a matrix. They showed that the algorithm can then be applied to compute a 2-approximation to the **NP**-Hard ℓ_2^2 clustering problem. The sampling paradigm was improved and applied to pass-efficient algorithms for approximate matrix multiplication, for computing a succinct approximate representation of a matrix, and for approximate linear programming [24, 25, 26].

Feigenbaum *et al.* [29] considered graphs presented in read-only arrays. They gave streaming algorithms for such problems as finding approximate spanners, diameter and girth. They also considered multiple pass algorithms and showed that one cannot find vertices at distance d from a given vertex using fewer than d passes and $O(n)$ space.

A variant of the model has been introduced by Aggarwal, Datar, Rajagopalan, and Ruhl [2], who proposed augmenting the multiple pass model by adding a sorting primitive. In their new model, the elements in the input array may now be sorted based on a chosen feature calculated from the input elements, in addition to being accessed through a small number of passes. The authors' justification for this addition is based on their claim that sorting on disk is in fact one of the few non-sequential primitives that can be implemented to maximize the throughput of the I/O system. They proved that certain problems can be solved more efficiently in terms of passes and space if sorting were allowed.

1.2 Our Contribution

In this dissertation, we present pass-efficient algorithms for two families of clustering problems and conventional algorithms for a third clustering problem.

An interesting feature of our pass-efficient algorithms is that the number of passes over the input array that the algorithms may make is an input parameter. Our algorithms will exhibit sharp tradeoffs between the number of passes they make and the amount of memory they require. Furthermore, we will prove for some of these algorithms that the tradeoff is nearly tight and is thus an intrinsic property of the power of the pass-efficient model for this problem.

Although the two clustering problems that we consider in the pass-efficient model are quite different, our general approach of adaptive sampling implemented in multiple passes is surprisingly the same. In a single pass, we draw a small sample of size m from the input array and store it in memory. After the pass, we compute a solution on the sample. The crucial observation is that this solution will be a good solution for a large proportion of the input. In a subsequent pass we determine the small set of points of the input array that are not clustered well by this solution, and recursively solve the problem on these points. Thus, we draw a sample of size m from these points, effectively sampling them at a much higher rate than in previous iterations. This will provide us with many more samples from the input elements that are “difficult” to approximate.

We note that this general framework is similar to the machine learning technique of boosting (for a detailed description of boosting and its context of machine learning, see the book by Kearns and Vazirani [49]). A weak classification algorithm is an algorithm that will correctly classify a set of points with an error that is only slightly better than random guessing, whereas a strong classification algorithm is an

algorithm that will take an input parameter $\epsilon > 0$ and classify with error at most ϵ . Boosting is a technique that uses a weak learning algorithm as a subroutine to derive a strong learning algorithm. As an analog to the weak classifier, we use an algorithm to solve the problem coarsely on a sample. We then use this routine repeatedly to derive an accurate solution.

1.2.1 Generative Clustering: Learning mixture models in multiple passes

We present algorithms for the unsupervised learning of mixtures of distributions in the pass-efficient model of computation. This research appeared in [13].

Suppose we are given k probability distributions F_1, \dots, F_k over a universe Ω , each with a *mixing weight* $w_i > 0$, such that $\sum_i w_i = 1$. A point drawn according to the *mixture* of these k distributions is defined by choosing the i th distribution with probability w_i and then picking a point according to F_i . If F is the density function of the mixture, the problem we consider is: can we approximately reconstruct F from samples placed in a read-only input array? More precisely, our goal is to find a function G such that the L^1 distance between F and G is at most ϵ : $\int_{\Omega} |F - G| \leq \epsilon$. We first consider the case where $\Omega = \mathbb{R}$ and each F_i is uniform over some contiguous interval $(a, b) \subset \mathbb{R}$. We call the resulting mixture a *mixture of k uniform distributions*.

Suppose that X is a set of n points randomly chosen from \mathbb{R} according to a mixture of k -uniform distributions with density function F . X can be ordered (possibly by the adversary) to produce a sequence that constitutes the read-only input array. Our problem is then: design a pass-efficient algorithm that approximates F from the input array X . Our pass-efficient learning algorithm for this problem has the property that its memory requirement drops off sharply as a function of the number

of passes allotted.

Our Results and Techniques

We present the following results. Here ℓ is any positive integer chosen by the user.

1. We give a 2ℓ -pass algorithm for learning a mixture of k uniform distributions with error at most ϵ using $\tilde{O}(k^3/\epsilon^{2/\ell})$ bits of RAM, for any integer $\ell > 0$, provided that the number of samples in the input array satisfies $n = \Omega^*(1.25^\ell k^6/\epsilon^6)$. Alternatively, we can view this as an algorithm with error at most ϵ^ℓ that uses $\tilde{O}(k^3/\epsilon^2 + k\ell/\epsilon)$ bits of memory, provided that $n = \tilde{\Omega}(1.25^\ell k^6/\epsilon^{6\ell})$.
2. We slightly generalize our learning problem and prove a lower bound of $\Omega\left(\left(\frac{1}{2c}\right)^{1/(2\ell-1)} c^{-2\ell+1}\right)$ for the bits of RAM needed by any ℓ -pass randomized algorithm that solves the general problem, where c is a fixed constant. We strengthen our pass-efficient algorithm in order to provide an upper bound of $\tilde{O}(1/\epsilon^{4/\ell})$ on bits needed by an ℓ -pass algorithm.
3. We generalize our multiple pass algorithm for uniform distributions to algorithms for *learning a mixture of k linear distributions* and *learning a mixture of k two dimensional uniform distributions*. The former is a mixture of distributions over the domain \mathbb{R} such that each F_i has a density function that is linear over some contiguous interval. The latter is a mixture of distributions over the domain \mathbb{R}^2 such that each F_i is uniform over some axis-aligned rectangle.

The error of our algorithm falls exponentially with the number of passes while holding the amount of memory required constant. An alternative interpretation is that memory falls sharply with the number of passes, while holding the error constant.

Making more passes over the data allows our algorithms to use less space, thus trading one resource in the pass-efficient model for the other. This feature demonstrates the potential for multiple passes by providing flexible performance guarantees on space needed and passes used; the algorithm can be tailored to the specific needs and limitations of a given application and system configuration.

The lower bound proves that this tradeoff between the number of passes and the amount of memory required is nearly tight for a slightly more general problem. Thus, this exponential tradeoff is an intrinsic property of the problem.

We now describe the general technique for our multiple pass algorithm. In a single pass, we partition the domain into a set of intervals based on samples of the input array. Our algorithm will then estimate F on each of these intervals separately. In a second pass, we count the number of points in X that lie in each of these intervals and run a subroutine called **Constant?**, which determines whether or not F is (approximately) constant on each interval. If F is constant on an interval I , then the density of F can be estimated easily by the number of points of X that fall in the interval. If F is not constant on I , the algorithm recursively estimates the density in the interval during the subsequent passes, in effect “zooming in” on these more troublesome intervals until we are satisfied with our approximation.

The crucial component of the above algorithm is a one-pass subroutine that uses a small amount of space to determine if F is constant on a given interval. We prove that Indyk’s one-pass algorithm for computing the ℓ_1 length of a vector given as a stream of dynamic updates [43] can be used to solve this problem using extra space at most $O((\ell \log(1/\epsilon) + \log k) \log(\ell k/\delta\epsilon))$ (we will discuss this algorithm in Section 2.2.1).

In order to prove lower bounds for multiple pass algorithms, we appeal to known results for the communication complexity of the GT (Greater Than) function, which

determines for two inputs a, b whether $a > b$. Specifically, we show that any ℓ -pass algorithm that solves our problem will induce a $(2\ell - 1)$ -round protocol for the GT function. Known lower bounds [56] on the $(2\ell - 1)$ -round communication complexity of GT will then provide lower bounds for the memory usage of ℓ -pass algorithms.

Related Work

Our problem of learning mixtures of uniform distributions is similar to histogram construction and maintenance, which we mentioned in Section 1.1.2. The streaming algorithm for histogram maintenance [32] will compute the best piecewise constant representation of the density of a set of points, without assuming any sort of generative model for the input. Thus, this problem is in some ways more general than learning mixtures of uniform distributions (however, it is subtly different), but very different from the two dimensional case, or the linear case. We note that this algorithm for histogram maintenance is single pass and does not have the interesting tradeoff properties.

Generative mixture models is one of the major clustering domains in data mining. Much work has been done on the unsupervised learning of mixtures of k Gaussian distributions in high dimension, which involves estimation of the parameters of the constituent Gaussians, (i.e. their centers, covariance matrices, and mixing weights). Assumptions on the separation of the Gaussians need to be made in order to prove rigorous results about the accuracy of these algorithms; the problems are ill-defined if the Gaussians overlap too much. The first paper in the theoretical computer science literature on this problem was due to Dasgupta [19], who gave an algorithm for learning the centers of the constituent Gaussians. The result was later improved in a series of papers by Arora and Kannan [4], Vempala and Wang [69], and Kannan, Salmasian, and Vempala [47].

1.2.2 Clustering by Combinatorial Optimization: Approximating facility location in multiple passes

Facility Location is a well-studied problem in combinatorial optimization. In this problem, we are given a metric space (X, d) consisting of a set X of n points, a distance metric d on X , and a facility cost f_i for each $x_i \in X$. The problem is to find a set $F \subseteq X$ that minimizes the objective function $\sum_{x_i \in F} f_i + \sum_{x_i \in X} d(x_i, F)$, where $d(x_i, C)$ denotes the distance from x_i to the closest point in the set C .

A primary obstacle for designing pass-efficient algorithms with small space requirements is that there are easy-to-construct instances of the facility location problem that cannot be approximated to within any constant factor with fewer than $\Omega(n)$ facilities. Intuitively, this indicates that one cannot hope to design an algorithm that provides a solution to the problem using $o(n)$ space. Indeed, we can prove that any ℓ -pass randomized algorithm that approximates even the *cost* of the optimum solution will need at least $\Omega(n/\ell)$ space, using a simple application of the communication complexity of the set disjointness function.

We address these difficulties by parameterizing the amount of space used by our algorithm by the number of facilities opened by the approximate solution. Thus, our algorithm will use $3(\ell - 1)$ passes to compute a solution with approximation ratio $O(\ell)$ using at most $\tilde{O}(k^* n^{2/\ell})$ bits of memory, where k^* is the number of facilities opened by the approximate solution. Note that for many instances k^* may in fact be much smaller (or possibly much larger) than the number of facilities opened in an optimum solution.

A compelling feature of this algorithm is that the amount of memory decreases sharply as the number of passes increases. Thus, making a few more passes will substantially reduce the amount of space used by the algorithm at the cost of slightly

increasing the approximation ratio. The similarity of this tradeoff to the tradeoff for our learning algorithm is not a coincidence: the general framework of adaptive sampling implemented in multiple passes that can be used to learn mixtures of distributions can also be adapted to facility location.

We also propose an alternative way of addressing the difficulty presented by the $\Omega(n/\ell)$ lower bound on the amount of space needed by an ℓ -pass algorithm for facility location. We define the *k-facility location problem* to be a hybrid between k -median and facility location, where we must find the best facility location solution that uses at most k facilities. A situation where this might arise is if we are constrained to building at most k facilities (perhaps we only have a limited amount of labor, time, or equipment), but each potential facility will incur a different cost. k -median is a special case of the problem when all facilities costs are 0. We adapt our multiple pass algorithm to solve this problem using a small amount of extra space.

Our Results and Techniques

We present the following results:

1. For any integer $\ell \geq 2$, our main algorithm for facility location is a $3(\ell - 1)$ -pass algorithm with approximation ratio $O(\ell)$ that uses at most $\tilde{O}(k^* n^{2/\ell})$ bits of memory, where k^* is the number of facilities opened by the approximate solution.
2. We adapt our main algorithm to an algorithm for k -facility location with approximation ratio $O(\ell)$ using 3ℓ passes and at most $\tilde{O}(k^{(\ell-1)/\ell} n^{1/\ell})$ bits of memory.
3. We prove a lower bound of $\Omega(n/\ell)$ for the number of bits needed by any ℓ -pass randomized algorithm that computes the *cost* of the optimum facility location

solution to within any polynomial factor.

For facility location, our technique is a combination of the general approach for our learning algorithm and a generalization of a sublinear bicriterion approximation algorithm for k -median developed by Indyk [42]. The high level description is that we take a sample S from the input array X and cluster the sample with a known facility location algorithm. The resulting facilities can then be shown to be a good clustering for X , except for a small set of outliers. In subsequent passes, we “zoom in” on the outliers by recursively calling our algorithm on those points and sampling them at a higher frequency.

Related Work

Many algorithms with constant factor approximation ratios have been designed for facility location in the traditional models of computation, including [15, 54, 55, 62], as well as approximation schemes for instances that lie in the Euclidean plane [5]. It is also known that the problem cannot be approximated to within a factor of 1.46, unless $\mathbf{NP} \subseteq \mathbf{DTIME}(n^{\log \log n})$ [34]. In this paper, we consider pass-efficient algorithms for facility location.

The three most basic clustering problems in combinatorial optimization are k -center, k -median, and facility location. k -center and k -median are the problems of finding a set of centers $C \subseteq X$, $|C| \leq k$, that minimizes the objective function $\max_i d(x_i, C)$ and $\sum_i d(x_i, C)$, respectively. Charikar, Chekuri, Feder, and Motwani [14] designed a single pass constant factor approximation algorithm for k -center that uses at most $O(k)$ extra memory; this algorithm has the added appeal that it follows a bottom-up approach to clustering, known as hierarchical agglomerative clustering, which is known to be very efficient in practice. Guha, Mishra, Motwani, and O’Callaghan [36] designed a single pass algorithm for k -median with approximation

ratio $2^{O(1/\epsilon)}$ using at most $O(n^\epsilon)$ extra space, for $\epsilon > 0$. This was later improved by Charikar, O’Callaghan, and Panigrahy [16] to a constant factor approximation ratio using at most $O(k \log^2 n)$ extra space. Charikar and Panigrahy [17] also gave a streaming algorithm for the objective function of finding k clusters that minimize the sum of cluster diameters, using $O(k)$ space, but also using ck centers, for some constant c . These algorithms achieve nearly optimum space usage and constant factor approximation ratios and thus cannot be substantially improved. However, prior to this work, no small space pass-efficient or streaming algorithm existed for the general case of facility location.

Indyk [44] designed a one pass algorithm for highly restricted geometric instances of facility location. If the input is contained in the lattice $\{1, \dots, \delta\}^d \subseteq \mathbb{R}^d$, his algorithm will find an $O(d \log^2 \delta)$ approximation to the optimum *cost* of a facility location solution using space at most $O(\log^{O(1)}(\delta + n))$, for the case when the facility costs are restricted to be uniform. Note that if the diameter of the input is diam , then $\text{diam}/\sqrt{d} \leq \delta \leq \text{diam}$.

Meyerson [55] gave an online algorithm with an approximation ratio of $O(\log n)$ if the demand points in the data stream are ordered arbitrarily, and an approximation ratio of $O(1)$ if they are ordered randomly. Note that these approximation ratios are stronger than competitive ratios: the algorithm will output a solution with cost at most $O(\log n)$ times the offline optimum, rather than the best possible online solution. Note that, in general, online algorithms do not optimize for space. Meyerson’s algorithm thus uses $\Omega(n)$ space, but satisfies the rigid online requirement that it must always maintain a solution with each element it processes. This online algorithm was a crucial subroutine in the one pass k -median algorithm given in [16].

Badoiu, Czumaj, Indyk, and Sohler [8] consider massive data set algorithms for facility location. They developed an algorithm that runs in time $O(n \log^2 n)$ for

approximating the *cost* of the optimum solution for the restricted case where all nodes have uniform facility cost. Since the input to the problem consists of $\Theta(n^2)$ pairwise distances between n points, the running time of the algorithm is sublinear in the size of the input. They also prove a lower bound of $\Omega(n^2)$ on the running time of any algorithm that approximates the cost of the optimum facility location solution, for the general case where facility costs are unrestricted.

1.2.3 Clustering by Combinatorial Optimization: Graph Partitioning

In a graph partitioning problem, we are asked to find a set of edges or nodes to remove from a graph in order to partition the nodes into a set of disjoint, connected components. Usually we optimize an objective function that depends on the number of removed edges and properties of the partition of the graph output by the algorithm. Each component of the partition can be thought of as a cluster, and thus graph partitioning algorithms are a form of top-down clustering.

We consider an **NP**-Hard graph partitioning problem that we call the *sum-of-squares partition* problem. In the *node cut* version, we are given an integer m and want to remove m nodes (and their adjacent edges) that partition the graph into components $\{C_i\}$ such that $\sum |C_i|^2$ is minimized. This is a natural problem, since intuitively a partition of the graph that minimizes this objective function will have its large components all be of roughly equal size. We present a greedy bicriterion-approximation algorithm that will remove a set of $O(m \log^{1.5} n)$ nodes that partitions the graph into components $\{\tilde{C}_i\}$ such that $\sum |\tilde{C}_i|^2 \leq O(1)\text{OPT}$, where OPT is the optimum value of the objective function when at most m nodes are removed. The algorithm is simple to describe: it repeatedly applies known algorithms to find and

remove sparse node cuts from the graph. This greedy partitioning continues until $\Theta(m \log^{1.5} n)$ nodes have been removed. Our algorithm generalizes to the *edge cut* version, which has a much simpler analysis. We also prove complementary hardness of approximation results, reducing from minimum vertex cover, which cannot be approximated to within a constant factor of 1.36 [21].

The algorithm appeared in a paper with James Aspnes and Aleksandr Yampolskiy [7] on a problem of network security. Chapter 5 of this thesis will focus on our algorithm and its analysis, but will also discuss at a high level its context in [7]. In this study, we designed a game theoretic model for the propagation of a virus in a network represented as a graph, where the nodes are the users and the edges are the connections. Each node must choose to either install anti-virus software at some known cost C , or risk infection with a potential cost L . The virus starts at a random initial node in the graph and will spread to other nodes depending upon the network topology and the topology of the nodes that chose to install anti-virus software. Our study proved many game theoretic properties of this system, showing that selfish behavior will incur a cost to the entire system that is much higher than the best cost that can be achieved. As an alternative to anarchy, we considered a centralized solution to the problem of computing the set of nodes on which to install anti-virus software that will minimize the expected cost to the entire network. This algorithmic problem reduces to the sum-of-squares partition problem; our algorithm thus induces an approximation algorithm for finding the best configuration of anti-virus software installation.

Related Work

The seminal work of Leighton and Rao [53] on multicommodity flows and sparse cuts inspired much work on many different graph partitioning problems. Roughly, a

sparse cut is a cut with few edges that partitions the graph into large components (we will define the sparsity of a cut precisely in Chapter 5). Leighton and Rao designed an $O(\log n)$ approximation algorithm for finding the sparsest cut. Their work was improved by Arora, Rao, and Vazirani [6] who gave an algorithm with approximation ratio $O(\sqrt{\log n})$. Problems related to finding the sparsest cut include finding balanced cuts, k -balanced partitions, ρ -separators, and graph bisections. A ρ -separator is a set of edges whose removal partitions the graph into connected components such that each component has at most ρn nodes. A k -balanced partition is an $O(k)$ -separator that creates at most k components. A c -balanced cut is a cut that will partition the nodes into two components, each of size at least cn . An approximation algorithm for minimum c balanced cuts can be found in [6]. Even, Naor, Rao, and Schieber [27] designed approximation algorithms for finding minimum balanced separators and k -balanced partitions. Feige and Krauthgammer [28] designed approximation algorithms for the minimum graph bisection problem of finding the smallest cut that will divide the graph into two components of size exactly $n/2$.

Kannan, Vempala, and Vetta [48] designed a graph partitioning algorithm for clustering that involved recursively removing sparse cuts. They proved that their algorithm computed a graph partitioning that approximately optimized any balance of the following criteria: each component has a high conductance (a measure of “how well-knit” a graph is, with connections to sparsity), and the proportion of all the edges that cross the cut is small.

1.3 Notes to the Reader

In this thesis, we will assume that the reader is familiar with basic concepts and notation from calculus, probability theory [10], linear algebra [41], randomized algorithms

[58], and basic complexity classes [64].

In Chapter 2, we review the background that we will use in our algorithms and proofs: bounds on the weight of tails of probability distributions, small space streaming algorithms for computing the norms of vectors, and computational complexity. In Chapter 3, we present our results for learning mixtures of distributions. In Chapter 4, we present our results for facility location and related problems. In Chapter 5, we present an approximation algorithm for the sum-of-squares-partition problem. In Chapter 6, we will make some concluding remarks as well as discuss exciting open problems that arise from this research.

Chapter 2

Preliminaries

In this dissertation we study the theory of algorithms for both traditional computation and massive data set computation. We first review some of the algorithmic techniques that can be used for both traditional and pass-efficient algorithms; we then review background material specific to streaming and pass-efficient algorithms, and lastly review communication complexity, which is often applied to proving lower bounds on the space usage of streaming and pass-efficient algorithms.

The study of the intrinsic computational resources required to solve a problem are approached from two different directions. The *theory of algorithms* attempts to determine how little time or space is sufficient for an abstract computer to solve a computational problem. *Computational complexity theory* attempts to determine how much time or space is necessary for an algorithm to solve a problem. These two fields complement each other: lower bounds on the amount of time or space required to solve such problems as facility location are very helpful to algorithm designers, who can narrow their search for efficient algorithms to those that respect the lower bounds. Similarly, algorithmic results providing upper bounds on the resource usage required for solving a problem will focus a complexity theorist's search for lower

bounds.

2.1 General Purpose Algorithmic Tools

Many fundamental optimization problems, including facility location and many of the graph partitioning problems (including the sum-of-squares partition problem), can be proved to be computationally intractable under some long-standing conjectures from complexity theory. However, if we are satisfied with an algorithm that is slightly suboptimum, algorithms for finding such solutions are often very practical. The field of approximation algorithms addresses such issues. For a minimization problem, we say that an algorithm has approximation ratio $\alpha \geq 1$ if it will always output a solution with cost at most αOPT , where OPT is the cost of the optimum solution to the instance. For a maximization problem, we say that an algorithm has approximation ratio α , for $0 \leq \alpha \leq 1$, if it will always output a solution with cost at least αOPT . For a comprehensive survey of approximation algorithms, see the book by Vazirani [67] or the book edited by Hochbaum [39].

2.1.1 Bounds on Tails of Distributions

A major paradigm in algorithm design and analysis is the use of randomness. Algorithms that can utilize the results of random coin tosses can often perform better than their deterministic counterparts. This is perhaps counter-intuitive, since a random coin toss will provide no insight into the structure of the problem under consideration. Whether or not the perceived power of randomness can be attributed to either the intrinsic power of coin tosses or to a lack of some crucial insight on the part of algorithm designers is an open question. (See Section 2.3 for a discussion of results in a different area of theoretical computer science where randomness can be proven to

be helpful.) For surveys of topics in randomized algorithms, see the books [57, 58].

We will review some of the bounds on the probability of random variables deviating far from their means. These theorems are fundamental to the analysis of randomized algorithms and computational learning theory. We will operate in both discrete and continuous probability spaces. For a review of probability theory, see the textbooks by Billingsley or Sinai: [10, 63].

Markov and Chebyshev Inequalities

The *Markov inequality* is the most basic, and the weakest, of these bounds. It has the appeal that it is the strongest inequality that requires almost no knowledge of the random variable. The only restriction is that the random variable must be non-negative.

Let X be a random variable with finite mean, such that $X \geq 0$. Then:

$$\Pr[X \geq tE[X]] \leq \frac{1}{t}.$$

The proof of the Markov inequality is a particularly simple proof by contradiction. Suppose that the Markov inequality does not hold. Then $\Pr[X \geq tE[X]] > \frac{1}{t}$, which implies that $E[X] > 1/t \cdot tE[X] > E[X]$.

A stronger inequality that requires knowing the second moment of the random variable is the *Chebyshev Inequality*. If X is a random variable (that can possibly take negative values) with finite mean and variance, then

$$\Pr[|X - E[X]| \geq t\sqrt{\text{Var}[X]}] \leq \frac{1}{t^2}$$

The Chebyshev inequality can be derived by applying the Markov inequality to the random variable $(X - E[X])^2$. The inequality generalizes to higher order moments.

Chernoff Bounds

A more involved application of the Markov inequality will yield sharper concentration properties for sums of independent random variables, known as *Chernoff Bounds* [67].

Let X_i be a sequence of n independent, identically distributed random variables taking values in $\{0, 1\}$, with $E[X_i] = \mu$. Let $\bar{X} = \frac{\sum_{i=1}^n X_i}{n}$ be an estimate of μ . For $0 < \epsilon < 1$, we then have the Chernoff bound for the upper tail:

$$\Pr [\bar{X} \geq (1 + \epsilon)\mu] \leq e^{-n\mu\epsilon^2/4}$$

and the lower tail:

$$\Pr [\bar{X} \leq (1 - \epsilon)\mu] \leq e^{-n\mu\epsilon^2/2}.$$

We have stated these inequalities for sequences of independent, identically distributed random variables that take values of either 0 or 1; they can be generalized to the case where the X_i s take values in $[0, 1]$ and are not identically distributed [40].

VC Bounds

Our proofs make use of a powerful statistical tool known as the *Vapnik-Chervonenkis (VC) Dimension* and its associated VC Bound. They were introduced to the theoretical computer science community by the seminal paper of Blumer, Ehrenfeucht, Haussler, and Warmuth [11], who used VC bounds to prove results in computational learning theory.

Consider the following scenario. Let X_1, \dots, X_m be independent identically distributed random variables taking values in universe Ω and drawn according to a probability measure μ ($\mu : \Omega \rightarrow [0, 1]$ is a function such that $\mu(A)$ gives the probability that $X_i \in A$) and let \mathcal{C} be a family of sets (we require a few technical conditions

such as the measurability of μ and \mathcal{C} with respect to some σ -algebra \mathcal{F} over Ω , which will always be true for our well-behaved applications). A sample of m points drawn according to μ is then an instantiation of these X_i s; call the sample X . Given a set $U \in \mathcal{C}$, we can estimate $\mu(U)$ by the number of points of the sample that lie in U : $|X \cap U|/m$. A natural question to ask is: How close is this empirical estimate likely to be to the true value of $\mu(U)$? Chernoff bounds give sharp bounds on the relative error of the estimate for a *fixed* $U \in \mathcal{C}$. VC bounds, on the other hand, give sharp bounds on the absolute error of the estimate for *all* $U \in \mathcal{C}$ *simultaneously*.

VC Bounds are given in terms of the *VC dimension* of \mathcal{C} , which is a combinatorial measure of the complexity of the constituent sets of \mathcal{C} . We say that \mathcal{C} *shatters* the set of k points $X = \{x_1, \dots, x_k\}$ if for all subsets $U \subseteq X$, there exists a set $C \in \mathcal{C}$ such that $X \cap C = U$. The VC dimension of \mathcal{C} is the largest k such that there exists a set of k points that can be shattered by \mathcal{C} . For the purposes of this dissertation, we only need to know that the VC dimension of the set of all open intervals in \mathbb{R} , which is given by $\{(a, b) | a < b, a, b \in \mathbb{R}\}$, is $d = 2$ and the VC dimension of the set of all axis-aligned rectangles in \mathbb{R}^2 , which is given by $\{(a, b) \times (c, d) | a < b, c < d, a, b, c, d \in \mathbb{R}\}$, is $d = 4$. We then have the following tail inequality.

Fact 1 (Vapnik and Chervonenkis, Talagrand bound) (*Theorem 1.2 from [65]*)

Let \mathcal{C} be a family of measurable sets with VC dimension d , and let $\epsilon > 0$, $\delta > 0$. Then there exists a constant c_0 such that if X is a set of m samples drawn according to μ , and

$$m \geq c_0 \frac{1}{\epsilon^2} \left(d \log \left(\frac{1}{\epsilon} \right) + \log \left(\frac{1}{\delta} \right) \right), \quad (2.1)$$

then

$$Pr \left[\sup_{U \in \mathcal{C}} \left| \frac{|X \cap U|}{m} - \mu(U) \right| \geq \epsilon \right] \leq \delta.$$

The power of the VC bound lies in the fact that the error is bounded *simultaneously*

for all sets in \mathcal{C} .

Vapnik and Chervonenkis [66] were the first to use VC dimension in order to prove bounds similar to Fact 1. Talagrand [65] later derived the optimum bound, but this does not change the asymptotic sample complexity for our purposes.

Union Bound

The so-called union bound is not a tail inequality, but is a simple inequality that is quite useful for the analysis of algorithms. Suppose we are given a probability space $(\Omega, \mathcal{F}, \mu)$. For any sequence of events $A_i \in \mathcal{F}$, $\mu(\cup A_i) \leq \sum \mu(A_i)$.

2.2 Pass-Efficient and Streaming Algorithms

In this section we review algorithmic techniques and results that apply primarily to streaming and pass-efficient computation.

2.2.1 Approximating the ℓ_1 norm of a stream of dynamic updates

Consider a data stream X consisting of pairs of the form $\langle i, a \rangle$, $i \in [n]$ and $a \in \{-M, \dots, M\}$. This stream represents vector $v \in \mathbb{R}^n$, where each component of the vector is given by $v_i = \sum_{\langle i, a \rangle \in X} a$. Thus, each time we see a pair $\langle i, a \rangle$ from the data stream, we add the integer a to the i th component of v . We define the quantity $L_1(X) = \left(\sum_{i \in [n]} \left| \sum_{\langle i, a \rangle \in X} a \right| \right)$. The ℓ_p norm of a vector $v \in \mathbb{R}^n$ is defined as $\|v\|_p = \left(\sum_{i=1}^n |v_i|^p \right)^{1/p}$ (thus the 2-norm is the standard Euclidean norm: $\|v\|_2 = |v| = \sqrt{\sum_{i=1}^n v_i^2}$). Note that $L_1(X) = \|v\|_1$ is the ℓ_1 length of v . Indyk designed a one pass algorithm for this problem. We state a weaker version of his result that is suitable for our purposes:

Fact 2 [43] *There is an algorithm that will find an estimate of $L_1(S)$ that is within an additive error of $\pm \frac{L_1(S)}{2}$ of $L_1(S)$ with probability $1-\delta$ and that uses $O(\log M \log(1/\delta))$ bits of memory, $O(\log M \log(n/\delta))$ random bits, and $O(\log(n/\delta))$ arithmetic operations per pair $\langle i, a \rangle$.*

Indyk's algorithm works roughly as follows. In order to avoid storing all components of the vector v at a cost of $\Omega(n)$ bits of memory, the algorithm projects the vector into a randomly chosen subspace of low dimension d . Suppose the random projection is defined by a random $d \times n$ matrix A . The algorithm maintains the d dimensional vector \tilde{v} , which it initializes to 0. Upon reading the element $\langle i, a \rangle$, it will update \tilde{v} by setting $\tilde{v} \leftarrow \tilde{v} + A(a\vec{e}_i)$, where \vec{e}_i is the i th basis vector of \mathbb{R}^n with 1 in the i th component and 0 in all other components. After all elements of the data stream have been processed, we have $\tilde{v} = Av$. Random projection is a major algorithmic tool for speeding up algorithms for problems with instances in high dimensional vector spaces. For a treatment of random projection and some of its applications in theoretical computer science, see the book by Vempala [68].

The random projection will approximately preserve the norm of the vector, but vastly reduce its dimension to d , and hence reduce the number of components that need to be stored to d . At first glance, it may seem necessary to store the projection matrix of size at least dn ; however, pseudorandom generators for space bounded computations allow us to instead store a small, truly random seed and generate the dn (pseudorandom) entries of the projection matrix. Thus, instead of storing the matrix, we run the pseudorandom generator on the same random seed to generate each matrix element as we need it, and then delete it from memory immediately after using it. These pseudorandom generators for space bounded computation were designed by Nisan [60] and do not use hard core bits and thus do not require cryptographic assumptions such as the existence of one way functions.

2.3 Communication Complexity

Communication complexity is an important model in complexity theory that has found applications in proving lower bounds for the space complexity of streaming and pass-efficient algorithms. Alon, Matias, and Szegedy [3] were the first to make this connection. We will discuss this application of communication complexity in Section 2.3.2. Other techniques for proving lower bounds on streaming and pass-efficient algorithms include using information theory [9]. Communication complexity was a model first introduced by Yao [70]. See the book by Kushilevitz and Nisan [52] for a survey of results and applications of communication complexity.

We now give a brief description of the most basic model of communication complexity. Two players, Alice and Bob, get input parameters $a \in \{0,1\}^n$ and $b \in \{0,1\}^n$, respectively; neither knows anything about the other's input. The players' goal is to cooperate and compute $f(a,b)$, for some boolean function $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}$. Since neither player knows the entire input to f , neither player can compute the solution outright without first “consulting” with the other player. Each player thus alternately sends and receives messages (in the form of bit strings) until one of them can determine $f(a,b)$. We assume that each player has an arbitrary amount of computing power, in terms of time and space, and that the players are cooperating in order to minimize the amount of communication (i.e. neither player will “lie”). Thus, we are most interested in the *number of bits of communication* that are intrinsically needed in order to compute the function.

Alice and Bob's communication is not haphazard, but rather governed by a *protocol* $\Pi(a,b)$ that determines when each player sends a message, and the contents of that message. The contents of the message sent by a given player is a function of the player's private input and the transcript of the messages communicated thus

far. At the end of the protocol, one of the two players will have the value of $f(a, b)$. The communication cost of the protocol, $\text{cost}(\Pi(a, b))$, is defined as the aggregate number of bits in all messages sent.

Example: A trivial protocol Π to compute $f(a, b)$ is for Bob to send a message containing b , and for Alice to then compute $f(a, b)$. In this case, $\text{cost}(\Pi(a, b)) = n$ for all a, b . In general, we hope to do better than sending one player's entire input, although this is not always possible.

$D(f) = \min_{\Pi} \max_{a, b} \text{cost}(\Pi(a, b))$, which is the cost of the best protocol on its worst case input pair, is called the *deterministic communication complexity* of the function f .

If the players are given access to private random coin tosses, then we can have *randomized protocols*. A randomized protocol must determine the correct value of $f(a, b)$ with probability at least $2/3$. The randomized communication complexity of the function f is then the cost of the best randomized protocol over its worst case input pair, and is denoted by $R(f)$. Clearly, $R(f) \leq D(f)$. For certain problems, the communication complexity of a randomized protocol can be proved to be much smaller than its deterministic counterpart.

We will be interested in the communication problems associated with the following functions:

Equality : $\text{EQ}_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$, such that $\text{EQ}_n(a, b) = 1$ if and only if $a = b$.

Greater Than : $\text{GT}_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$, such that $\text{GT}_n(a, b) = 1$ if and only if $a > b$.

Disjointness : $\text{DISJ}_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$, such that $\text{DISJ}_n(a, b) = 1$ if and only if $\forall i \in [n], a_i \neq b_i$.

Fact 3 *The following classic results hold:*

1. $D(\text{EQ}_n) = n, R(\text{EQ}_n) = \Theta(\log n)$

2. $D(\text{GT}_n) = n, R(\text{GT}_n) = \Theta(\log n)$

3. $D(\text{DISJ}_n) = n, R(\text{DISJ}_n) = \Theta(n)$.

Note that the randomized protocols for EQ and GT are provably much better than the deterministic protocols that must communicate all n bits of one player's input.

The communication complexity of the disjointness function was proved by Kalyanasundaram and Schnitger [46] and Razborov [61] and is often used for proving lower bounds for streaming algorithms. Indeed, we will use it in a straightforward manner to prove complexity results for pass-efficient algorithms for facility location.

2.3.1 r -round communication complexity

The basic communication complexity model has been generalized in many ways and has also been restricted in many ways. Our interest in pass-efficient algorithms for generative clustering leads us to consider the restriction of the model to *r -round communication complexity*.

In this model, we only allow protocols in which the players may send a total of at most r messages; for the sake of simplicity, we require that Alice send the first message. The cost of an r -round protocol Π on input a, b is defined slightly differently than in the unrestricted case. We define $\text{cost}^r(\Pi(a, b))$ as the length of the longest message sent by one of the players. The randomized r -round communication complexity of a function f is then the cost of the best randomized r -round protocol on its worst case instance, and is denoted by $R^r(f)$.

The following fact was proved by Miltersen, Nisan, Safra, and Wigderson [56], who used a more general technique to prove results about asymmetric communication,

which is a communication problem where Alice and Bob get inputs of very different size.

Fact 4 [56] $R^r(\text{GT}_n) = \Omega(n^{1/r} c^{-r})$ for some constant c .

The lower bound is nearly tight, since there exists a randomized r -round protocol for GT_n with complexity at most $O(n^{1/r} \log n)$ [56]. Recall that $R(\text{GT}_n) = \Theta(\log n)$; the optimum protocol requires $O(\log n)$ rounds of communication, which is consistent with Fact 4.

2.3.2 Connections to streaming and pass-efficient algorithms

The technique presented in this section for reducing communication complexity to the space complexity of pass-efficient and streaming algorithms is standard, but we state it more precisely. We first define a *streaming-compatible reduction* from a communication problem f to a target function g . We then show how the communication complexity of f can be used to prove lower bounds on the memory necessary for a streaming or pass-efficient algorithm for g .

Suppose we are interested in the communication problem associated with the function $f : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$. A *streaming-compatible reduction* from f to a target function $g : \{0, 1\}^m \rightarrow \{0, 1\}$ is an integer t and a pair of functions $\phi_A : \{0, 1\}^n \rightarrow \{0, 1\}^t$ and $\phi_B : \{0, 1\}^n \rightarrow \{0, 1\}^{m-t}$ such that $g(\phi_A(a) \circ \phi_B(b)) = 1$ if and only if $f(a, b) = 1$, where the \circ operator denotes concatenation of two strings of bits. There is no restriction on the computational complexity of ϕ_A and ϕ_B ; they can be arbitrary functions.

Streaming-compatible reductions are suitable for proving lower bounds for the memory usage of pass-efficient algorithms for the target function g . The main insight is that if we have a randomized ℓ -pass algorithm P for g that uses at most $M(P)$

bits of memory in the worst case, this can induce a $2\ell - 1$ round protocol with worst case cost $M(P)$.

Theorem 5 *Suppose there exists an ℓ pass algorithm P that will compute a function g correctly with probability at least $2/3$ that uses at most $M(P)$ bits of memory in the worst case. If there exists a streaming-compatible reduction from f to g , then*

1. $M(P) \geq R^{2\ell-1}(f)$
2. $M(P) \geq \frac{R(f)}{2\ell-1}$.

Proof: Let ϕ_A and ϕ_B be the functions associated with the streaming-compatible reduction. The ℓ -pass algorithm for g will induce a $2\ell - 1$ round protocol for the communication problem f in the following manner. Suppose Alice and Bob are given inputs a and b , respectively. They will simulate P on the read-only input array with entries given by $\phi_A(a) \circ \phi_B(b)$.

Alice simulates the first pass of P on $\phi_A(a)$ and sends a message to Bob containing the contents of the algorithm's working memory. Bob then simulates the remainder of the first pass of P on $\phi_B(b)$, and sends a message to Alice containing the contents of the algorithm's working memory. Alice then simulates the second pass of P . They continue in this manner until they have simulated all ℓ passes, exchanging a total of $2\ell - 1$ messages. At this point, with probability at least $2/3$, Bob will be able to compute $g(\phi_A(a) \circ \phi_B(b))$ and can thus compute $f(a, b)$.

This protocol for f exchanged $2\ell - 1$ messages, each of size at most $M(A)$. Thus, $M(A) \geq R^{2\ell-1}(f)$. The total number of bits communicated is at most $(2\ell - 1)M(A)$. Thus, $M(A) \geq R(f)/(2\ell - 1)$. \square

Chapter 3

Pass-Efficient Algorithms for Generative Clustering

In this Chapter, we will consider pass-efficient algorithms for the problem of learning mixtures of distributions.

Mixture Learning Problem: *Given $\epsilon > 0$ and a set of independent samples drawn according to a mixture of k uniform distributions in \mathbb{R} with density function F , find a function G such that $\int_{\mathbb{R}} |F - G| \leq \epsilon$.*

This algorithmic problem has not been previously studied in any model of computation. As we mentioned in Section 1.2.1, this problem is related to the previously studied problems of histogram maintenance in the streaming model and learning mixtures of Gaussians in the conventional model of computation.

Our main result is a 2ℓ pass algorithm that uses at most $\tilde{O}(k^3/\epsilon^{2/\ell})$ bits of memory and a nearly matching lower bound for a slightly stronger problem. In Section 3.2, we give a one pass algorithm for learning mixtures of k uniform distributions. In Sections 3.3 and 3.4, we present our main algorithm and its complementary lower bound on the memory requirements, respectively. In Sections 3.5 and 3.6, we generalize our

algorithm to learn mixtures of linear distributions in \mathbb{R} and to learn mixtures of uniform distributions in \mathbb{R}^2 , respectively.

3.1 Preliminaries

We first prove that the probability density function induced by a mixture of uniform distributions can be described as a piecewise constant function.

Lemma 6 *Given a mixture of k uniform distributions $(a_i, b_i), w_i$, let F be the induced probability density function on \mathbb{R} . F (except possibly at at most $2k$ points) can be represented as a collection of at most $2k - 1$ disjoint open intervals, with a constant value on each interval.*

Proof: We show how to construct the partition of the real line and the constant values that F holds on the partition.

First, consider the sequence defined by combining the a_i s and b_i s into a single sorted sequence of $2k$ points and eliminating all duplicate points. Call this new sequence $c_i, i = 1, \dots, m \leq 2k$.

Then, consider the following $m - 1$ intervals: $I_i = (c_i, c_{i+1})$ for $i = 1, \dots, m - 1$. These intervals cover the entire support of F (the set $\{x : F(x) \neq 0\}$), except possibly at the c_i s (of which there are at most $2k$.)

Note that F is constant on each interval (c_i, c_{i+1}) because, by definition of the c_i s, no endpoint of the original intervals falls in the open interval (c_i, c_{i+1}) . Let d_i be the value of F on I_i .

If χ_{I_i} is the characteristic function of interval I_i (i.e. $\chi_{I_i}(x) = 1$ if $x \in I_i$ and 0 if $x \notin I_i$) then

$$F(x) = \sum_i d_i \chi_{I_i}(x) \tag{3.1}$$

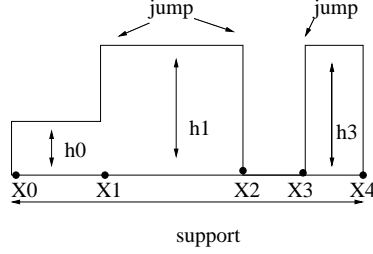


Figure 3.1: A mixture of 3 uniform distributions, labeled as a step function with 4 steps. $h_2 = 0$ is not labeled; it is the density of the third step.

except possibly at the points c_i . \square

Since the above characterization of F holds except on a set of measure zero, it will suffice for our algorithm to learn F with this representation. We will be somewhat lax in our language, and say that a set of intervals partitions some larger interval I , even if there are a finite number of points of I that are not included in any interval of the partition.

Definition 1 We define the step function representation of a mixture of k uniform distributions to be the set of at most $2k - 1$ pairs, $(x_i, x_{i+1}), h_i$, for $i = 0, \dots, m - 1 \leq 2k$ such that the density of the mixture is a constant h_i on the interval (x_i, x_{i+1}) .

Implicit in this definition is the fact that the m steps form a partition of the interval (x_0, x_m) .

Definition 2 The support of a mixture of uniform distributions given by step function representation $(x_i, x_{i+1}), h_i$ consisting of m steps is the interval (x_0, x_m) . If F is the density function of the mixture, we denote its support by $\text{Support}(F)$.

Definition 3 A jump of a step function is a point where the density changes (i.e. an endpoint shared by two different steps of the mixture).

We will work exclusively with the step function representation of the mixture.

3.2 A Single Pass Algorithm

In this section, we present a single pass algorithm that with probability $1 - \delta$ will learn the density function of a mixture of k uniform distributions within L^1 distance ϵ using at most $\tilde{O}(k^2/\epsilon^2)$ bits of memory.

OnePass is a divide and conquer algorithm, reminiscent of merge sort. Given a uniform random sample of $m = \tilde{O}(k^2/\epsilon^2)$ points drawn from the datastream, **OnePass** first initializes the computation, then sorts the m points. It calls subroutine **Cluster**, which divides the sample into two halves and recursively approximates the density function F in each half. **Cluster** then applies a *merging* procedure that decides whether or not the last interval of one half can be merged with the first interval of the other half.

Before describing the algorithm, we make the following definition.

Definition 4 *Suppose we have a set of points S and two intervals $J \subset J'$. We say that the empirical density of J is consistent with J' on S , or simply J is consistent with J' on S , if*

$$\left| \frac{|J \cap S|}{|S| \text{length}(J)} - \frac{|J' \cap S|}{|S| \text{length}(J')} \right| \leq \frac{\epsilon}{97k \log m} \cdot \frac{1}{\text{length}(J)}.$$

The algorithm is described in Figure 3.2.

We define the *base intervals* to be the intervals of size 2 created in the base case of **Cluster**. We define an *intermediate interval* to be an interval that was output by one of the calls to **Cluster** in the recursion, but not necessarily one of the final output intervals. We say that **Cluster** *merges* two intermediate intervals I_{k_1} and J_1 if in Step 5 it chooses to output the set of intervals containing $K = I_{k_1} \cup J_1$.

Input: datastream X , such that $|X| \geq m$ (see below).

1. In one pass, draw a sample of size

$$m = O\left(\frac{k^2 \log^2(k/\epsilon) + \log^2 \log(1/\delta)}{\epsilon^2} \left(\log \frac{k}{\epsilon} + \log \frac{1}{\delta}\right)\right) \quad (3.2)$$

from X , where m is a power of 2. Sort the samples, and call the set S .

2. Call **Cluster** on S , and let the resulting intervals be $\tilde{J}_1, \dots, \tilde{J}_v$.
3. Output the approximate density function, G , as the step function given by the intervals $\{\tilde{J}_i\}$ with densities $|S \cap \tilde{J}_i| / (m \text{length}(\tilde{J}_i))$.

Subroutine Cluster

Input: A set of t sorted points, $T = \{s_1, \dots, s_t\}$.

Base Case: If $t < 3$, then output interval (s_1, s_{t+1}) , where s_{t+1} is the point following s_t in S . Note that $T \subseteq (s_1, s_{t+1})$.

Main Recursion:

1. Set $T_1 = \{s_1, \dots, s_{t/2}\}$ and $T_2 = \{s_{t/2+1}, \dots, s_t\}$
2. Call **Cluster** on T_1 , and let the resulting intervals be I_1, \dots, I_{k_1} .
3. Call **Cluster** on T_2 and let the resulting intervals be J_1, \dots, J_{k_2} .
4. Set $K = I_{k_1} \cup J_1$.
5. If the empirical densities of both I_{k_1} and J_1 are consistent with K on T , then output the set of intervals $\{I_1, \dots, I_{k_1-1}, K, J_2, \dots, J_{k_2}\}$. Otherwise, output the set of intervals $\{I_1, \dots, I_{k_1}, J_1, \dots, J_{k_2}\}$

Figure 3.2: Algorithm **OnePass**

If viewed from a bottom-up perspective, the basic idea of **OnePass** is that the base case consists of a partition of $\text{Support}(F)$ into base intervals. Adjacent intermediate intervals are then repeatedly merged together if the empirical densities of the two intervals are similar enough. After all mergers, the resulting intervals define the steps of the output function, G . Care must be taken in choosing which adjacent intervals to consider for merging; each time an interval is merged, a small error is induced. If we consider merging indiscriminately, then we can potentially induce a large aggregate error if an interval participates in too many mergers. The recursion defined by **Cluster** gives a structure for choosing the mergers, such that no interval is involved in more than $\log m$ mergers. This fact is explicitly used for the proof of Lemma 8.

A simple calculation will show that we have chosen m in Equation 3.2 such that, with the appropriate constants, the VC bound guarantees that with probability $1 - \delta$,

$$\left| \frac{|S \cap I|}{m} - \int_I F \right| \leq \frac{\epsilon}{194k \log m}$$

for all intervals $I \subseteq \text{Support}(F)$. It follows that with probability $1 - \delta$,

$$\left| \frac{\int_I F}{\text{length}(I)} - \frac{|S \cap I|}{m \text{length}(I)} \right| \leq \frac{\epsilon}{194k \log m} \cdot \frac{1}{\text{length}(I)}, \quad (3.3)$$

for all intervals $I \subseteq \text{Support}(F)$. The results of this section are conditioned on this event occurring.

Lemma 7 *The output of **OnePass** contains at most $4k$ steps.*

Proof: Suppose the number of base intervals is N , which is some power of 2. Note that **Cluster** is called exactly $N - 1$ times. If we can show that all but $4k - 1$ of those calls to **Cluster** result in a merger, then we have shown the total number of

intervals in the output of **OnePass** is $4k$, since each merger reduces the number of intervals by one.

Thus, we seek to prove that all but $4k - 2$ of the calls to **Cluster** result in a merger, using the following claim.

Claim: Suppose two adjacent intermediate intervals I_1 and I_2 are contained in the same step of F , which has density h . Then both I_1 and I_2 are consistent with $I_1 \cup I_2$ on S .

Proof of Claim: By Inequality (3.3), we know that both I_1 and I_2 satisfy:

$$\left| h - \frac{|I_i \cap S|}{\text{mlength}(I_i)} \right| \leq \frac{\epsilon}{194k \log m \cdot \text{length}(I_i)}$$

and that $I_1 \cup I_2$ satisfies:

$$\left| h - \frac{|(I_1 \cup I_2) \cap S|}{\text{mlength}(I_1 \cup I_2)} \right| \leq \frac{\epsilon}{194k \log m \cdot \text{length}(I_1 \cup I_2)}.$$

Combining these two inequalities yields

$$\begin{aligned} \left| \frac{|(I_i) \cap S|}{\text{mlength}(I_i)} - \frac{|(I_1 \cup I_2) \cap S|}{\text{mlength}(I_1 \cup I_2)} \right| &\leq \frac{\epsilon}{194k \log m} \left(\frac{1}{\text{length}(I_i)} + \frac{1}{\text{length}(I_1 \cup I_2)} \right) \\ &\leq \frac{\epsilon}{97k \log m \cdot \text{length}(I_i)}. \end{aligned}$$

This proves the claim.

Thus, two adjacent intervals will fail to merge in Step 5 of **Cluster** only if one or both contains a jump in F . Since there are at most $2k - 1$ jumps in F , and each can induce at most two failed mergers (one, which fails to merge an intermediate interval containing the jump with the intermediate interval to its right, and the other with the intermediate interval to its left). The algorithm may fail to merge two intermediate

intervals at most $4k - 2$ times and the lemma follows. \square

Lemma 8 *Suppose J is an intermediate interval output by some call to **Cluster**.*

Then

$$\left| \frac{\int_J G}{\text{length}(J)} - \frac{\int_J F}{\text{length}(J)} \right| \leq \frac{\epsilon}{96k \text{length}(J)}.$$

Proof: Since all intervals output by some (possibly intermediate) call to **Cluster** are contained in one of the steps output by **OnePass**, we know that $J \subseteq \tilde{J}_i$, where \tilde{J}_i is some step of the output distribution G with density $|S \cap \tilde{J}_i| / (m \text{length}(\tilde{J}_i))$.

If J were produced by some intermediate call to **Cluster** (i.e. J is not part of the output, but rather J is strictly contained in \tilde{J}_i), then J must have been merged into some larger intermediate interval J_1 in Step 5 of some call to **Cluster**. If J_1 is a strict subset of \tilde{J}_i , then in turn J_1 must have been merged into some larger interval J_2 , and so forth until we finally reach \tilde{J}_i . If we set $J_0 = J$ and $J_t = \tilde{J}_i$, we have the following chain of inclusions: $J = J_0 \subset J_1 \subset \dots \subset J_{t-1} \subset J_t = \tilde{J}_i$. Since the depth of the recursion of **OnePass** is $\log m$, the number of mergers J is involved in is at most $\log m$. Hence, $t \leq \log m$.

By the design of the algorithm, J_j is merged into J_{j+1} only if J_j is consistent with J_{j+1} on S ; thus, we must have for all $0 \leq j \leq t - 1$

$$\left| \frac{|S \cap J_j|}{m \text{length}(J_j)} - \frac{|S \cap J_{j+1}|}{m \text{length}(J_{j+1})} \right| \leq \frac{\epsilon}{97k \log m \text{length}(J_j)}.$$

Thus, we can estimate the error:

$$\begin{aligned} \left| \frac{|S \cap J|}{m \text{length}(J)} - \frac{|S \cap \tilde{J}|}{m \text{length}(\tilde{J})} \right| &\leq \sum_{j=0}^{t-1} \left| \frac{|S \cap J_j|}{m \text{length}(J_j)} - \frac{|S \cap J_{j+1}|}{m \text{length}(J_{j+1})} \right| \\ &\leq \sum_{j=0}^{t-1} \frac{\epsilon}{97k \log m \cdot \text{length}(J_j)} \leq \frac{\epsilon}{97k \text{length}(J)}, \end{aligned}$$

where the last inequality follows from the fact that $t \leq \log m$ and $\text{length}(J) \leq \text{length}(J_j)$, for all j .

The Lemma then follows from applying Inequality (3.3) and the fact that $|S \cap \tilde{J}_i|/m\text{length}(\tilde{J}_i)$ is precisely the value of G on interval J :

$$\begin{aligned} \left| \frac{\int_J G}{\text{length}(J)} - \frac{\int_J F}{\text{length}(J)} \right| &\leq \left| \frac{|S \cap J|}{m\text{length}(J)} - \frac{|S \cap \tilde{J}_i|}{m\text{length}(\tilde{J}_i)} \right| + \left| \frac{|S \cap J|}{m\text{length}(J)} - \frac{\int_J F}{\text{length}(J)} \right| \\ &\leq \frac{\epsilon}{97k\text{length}(J)} + \frac{\epsilon}{194k \log m \cdot \text{length}(J)} \leq \frac{\epsilon}{96k\text{length}(J)}, \end{aligned}$$

for m sufficiently large. \square

Theorem 9 *One pass algorithm **OnePass** will learn a mixture of k uniform distributions to within L^1 distance ϵ with probability at least $1 - \delta$, using at most $\tilde{O}\left(\frac{k^2}{\epsilon^2}\right)$ bits of memory.*

Proof: Consider a step \tilde{J}_i of the approximate distribution G , and let t_i be the number of jumps of F that occur in \tilde{J}_i . We seek to prove the following bound on the error induced by G on \tilde{J}_i :

$$\int_{\tilde{J}_i} |F - G| \leq (t_i + 1) \frac{\epsilon}{6k}. \quad (3.4)$$

We know that there are at most $4k$ of the \tilde{J}_i s from Lemma 7, and $\sum t_i \leq 2k - 1$, since there are at most $2k - 1$ jumps in F . Inequality (3.4) implies that

$$\int |F - G| = \sum_i \int_{\tilde{J}_i} |F - G| \leq \sum_i (t_i + 1) \frac{\epsilon}{6k} \leq \epsilon,$$

from which the theorem follows.

We now prove the veracity of Inequality (3.4). Consider one of the intervals that comprises a step of the output distribution, \tilde{J}_i . Recall that the intervals created by

Cluster in the base case of the recursion formed a partition of $\text{Support}(F)$, and that for such an interval C , $|C \cap S| < 3$. Thus, the step \tilde{J}_i can be partitioned into a set of these base intervals. Suppose the jumps occur at points $x_1, \dots, x_{t_i} \in \tilde{J}_i$. Let C_j be the base interval such that $x_j \in C_j$.

Now consider the set $U = \tilde{J}_i \setminus (\cup_j C_j)$. Since \tilde{J}_i is an interval and the C_j s are intervals, of which there are at most t_i , U is a union of at most $t_i + 1$ intervals. Let these intervals be D_1, \dots, D_l , where $l \leq t_i + 1$. Note that since the C_j s are all the intervals that contain jumps, it follows that the D_j s do not contain jumps and hence are each contained in a step of F .

We first bound the error induced by the C_j s. Fix a C_t . From the VC-bound,

$$\left| \frac{|S \cap C_t|}{m} - \int_{C_t} F \right| \leq \frac{\epsilon}{194k \log m}.$$

Since $|S \cap C_t|/m < 3/m$, it follows that $\int_{C_t} F \leq \epsilon/(97k \log m)$. Lemma 8 thus implies that $\int_{C_t} G < 2\epsilon/(96k)$. It then follows that

$$\int_{C_t} |F - G| \leq \left| \int_{C_t} F + \int_{C_t} G \right| \leq 3 \frac{\epsilon}{96k \log m} \leq \frac{\epsilon}{32k}.$$

We now bound the error induced by the D_j s. Fix such a D_t . We may assume that $|S \cap D_t|/m \geq \epsilon/24k$; otherwise, we can show that the induced error on D_j is small using the same argument as for the C_t s.

Claim: If $|S \cap D_t| \geq 16$, there exists an intermediate interval I such that $I \subseteq D_t$ and $|S \cap I| \geq 1/4|S \cap D_t|$.

Proof of claim: Let $q = |S \cap D_t|$. Recall that $S = \{s_1, \dots, s_m\}$ was the set of sample points, in sorted order. Then $S \cap D_t = \{s_p, s_{p+1}, \dots, s_{p+q}\}$, for some integer p . Note that $2^{\lfloor \log(q/2) \rfloor} > q/4$. Thus, there must exist an integer j such that $p \leq j \cdot 2^{\lfloor \log(q/2) \rfloor} \leq$

$(j + 1) \cdot 2^{\lfloor \log(q/2) \rfloor} \leq p + q$. The set $\{s_r | j \cdot 2^{\lfloor \log(q/2) \rfloor} \leq r \leq (j + 1) \cdot 2^{\lfloor \log(q/2) \rfloor}\} \subset D_t$ will be the input set T to one of the calls to **Cluster**, and thus will comprise an entire intermediate interval I . This proves the claim.

Since F and G are both constant on D_t and hence are both constant on $I \subseteq D_t$, $|\int_I F - \int_I G| = \int_I |F - G|$. By Lemma 8 it follows that $|\int_I F - \int_I G| \leq \epsilon/96k$. Since the value of F and G are constant on all of D_t , we know that

$$\int_{D_t} |F - G| = \frac{\text{length}(D_t)}{\text{length}(I)} \int_I |F - G| \leq \frac{\text{length}(D_t)}{\text{length}(I)} \frac{\epsilon}{96k}.$$

By the VC bound, we can show that $\int_I F \geq 1/8 \int_{D_t} F$, which implies that $\text{length}(I) \geq 1/8 \text{length}(D_t)$. Combining this fact with the above inequality yields

$$\int_{D_t} |F - G| \leq \frac{\epsilon}{12k}.$$

This implies that the total error induced by \tilde{J}_i is

$$\int_{\tilde{J}_i} |F - G| = \sum_j \int_{C_j} |F - G| + \sum_j \int_{D_j} |F - G| \leq t_i \frac{\epsilon}{12k} + (t_i + 1) \frac{\epsilon}{12k} \leq (t_i + 1) \frac{\epsilon}{6k},$$

which proves that Inequality (3.4) is correct. \square

3.3 Multiple Passes

In this section, we show that additional passes can significantly reduce the amount of memory needed to achieve a fixed level of accuracy. Given a mixture of k uniform distributions with density function F and an integer ℓ , algorithm **SmallRam** can approximate F to within L^1 distance ϵ with 2ℓ passes, using at most $\tilde{O}(k^3/\epsilon^{2/\ell})$ memory, provided that $n = \tilde{\Omega}((1.25^\ell k^6)/\epsilon^{6\ell})$. We first give an algorithm with accuracy ϵ^ℓ

that uses space at most $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$.

SmallRam makes use of a subroutine called **Constant?**. **Constant?** is a single pass algorithm that will determine whether or not a datastream contains points drawn from a distribution with a constant density function. In Section 3.3.2, we describe the algorithm and prove the following theorem about its performance:

Theorem 10 *Let H be a mixture of at most k uniform distributions. Given a datastream X consisting of $|X| = \tilde{O}(\frac{k^5}{\beta^5} \log(1/\delta))$ samples drawn according to H , with probability $1 - \delta$ single pass algorithm **Constant?** will **accept** if H is uniform, and will **reject** if H is not within L^1 distance β of uniform. **Constant?** uses at most $O((\log(1/\beta) + \log k) \log(1/\delta))$ bits of memory.*

3.3.1 The Algorithm

Subroutine **Estimate** draws a random sample S of size $m = \tilde{O}(k^2/\epsilon^2)$ from the input stream. It sorts S and partitions it into $O(k/\epsilon)$ intervals. **Estimate** then calls **Constant?** on each of the intervals; **Constant?** uses the entire datastream X to determine if the distribution is constant on each interval. If F is close to constant on an interval I , we can output the interval as a step of the final output G with a constant density estimate derived from the quantity $|X \cap I|$, which we can determine in one pass over the datastream. Since $|X|$ is large, this estimate will be very accurate. However, if F is far from constant on I , then estimating its density by a constant is too coarse; thus we repeat the process by recursively calling **Estimate** on I . This recursive call can be thought of as “zooming in on the jumps.” See Figure 3.4.

Given a sequence of points X and an interval J , we define $X|_J$ to be the subsequence of X that consists of points that fall in J . If X is presented as a data stream,

a pass over X can simulate a pass over the data stream $X|_J$.

Our main algorithm is given in Figure 3.3.

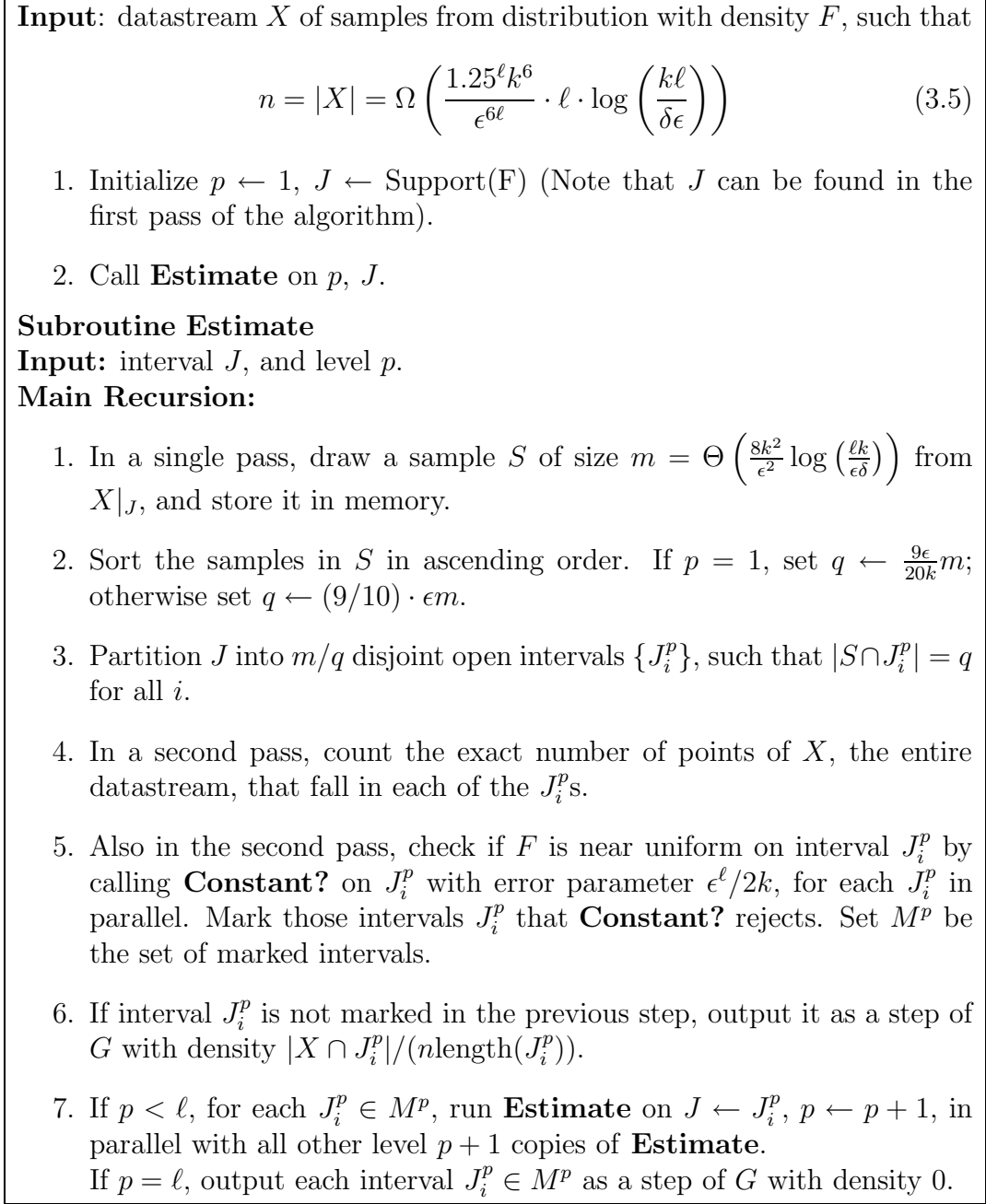


Figure 3.3: Algorithm **SmallRam**

Remark The order in which this recursive algorithm computes each call to **Estimate** has a natural structure defined by the level of the call (defined by its parameter

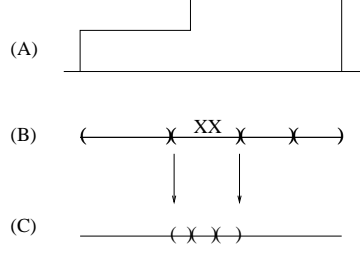


Figure 3.4: Zooming. (A) A step distribution. (B) Intervals created in Step 3 of some call to **Estimate**. One of them contains a jump and has been marked. (C) **Estimate** is recursively called on the marked interval.

p): All level p copies of **Estimate** are run in parallel, then all level $p + 1$ copies are run in parallel, and so forth. All copies of **Estimate** running in parallel may read the datastream simultaneously, thus limiting the total number of passes per level to two. The total number of passes is therefore 2ℓ .

We say that an interval J_i^p created in Step 3 of a level p call is a level p interval.

The following lemma states that the weight of F is roughly the same in all level p intervals.

Lemma 11 *Let J_i^p be an interval created in Step 3 of a level p call to **Estimate**. With probability at least $1 - \delta\epsilon/(25k^2\ell)$, J_i^p contains at most $\epsilon^p/2k$ proportion of the total weight of F , and at least $(8/10)^p\epsilon^p/2k$ proportion of the total weight. That is,*

$$\left(\frac{8}{10}\right)^p \cdot \frac{\epsilon^p}{2k} \leq \int_{J_i^p} F \leq \frac{\epsilon^p}{2k}.$$

Proof: We use a simple inductive argument to prove that an interval J_i^p created at level p contains at least $(8/10)^p\epsilon^p/2k$ proportion of the total weight of F . Assume that all level $p - 1$ intervals have at least $(8/10)^{p-1}\epsilon^{p-1}/2k$ proportion of the total weight of F , with probability at least $1 - (p - 1)\delta\epsilon/(50k^2\ell^2)$. Suppose interval J_i^p is marked in a level p call to **Estimate**. Note that J_i^p was contained in some level $p - 1$ interval $J_{i'}^{p-1}$ that was marked in level $p - 1$.

Claim: If $J_{i'}^{p-1}$ contains at least $(8/10)^{p-1}\epsilon^{p-1}/2k$ proportion of the weight of F , then J_i^p contains at least $(8/10)\epsilon$ fraction of the weight of $J_{i'}^{p-1}$ with probability at least $1 - \delta\epsilon/(50k^2\ell^2)$.

Proof of Claim: Consider the level p call to **Estimate** with $J_{i'}^{p-1}$ as input. It was this call that created J_i^p . Since we chose $m = \Omega(k^2/\epsilon^2 \log(\ell k/\delta\epsilon))$, applying the VC bound will show that with probability at least $1 - \delta\epsilon/(50k^2\ell^2)$,

$$\left| \frac{|S \cap J_i^p|}{m} - \int_{J_i^p} F \right| \leq \frac{1}{10}\epsilon.$$

Since J_i^p was chosen so that $|X \cap J_i^p| = (9/10)\epsilon|X \cap J_{i'}^p|$, J_i^p contains at least $(8/10)\epsilon$ fraction of the total weight of $J_{i'}^{p-1}$. This proves the claim.

By the inductive hypothesis, $J_{i'}^{p-1}$ contains at least $(8/10)^{p-1}\epsilon^{p-1}/2k$ fraction of the weight of F with probability $1 - (p-1)\delta\epsilon/(50k^2\ell^2)$; thus J_i^p contains at least $(8/10)^p\epsilon^p/2k$ fraction of the weight of F with probability $1 - p\delta\epsilon/(50k^2\ell^2)$, which follows from the union bound from Section 2.1.1.

The base case of the induction follows from the same argument as the proof of the claim, except we note that at the first level, in Step 2 we had set $q = \frac{9}{10} \cdot \frac{\epsilon}{2k} \cdot m$. It follows from the VC bound that

$$\left| \int_{J_i^1} F - \frac{|S \cap J_i^1|}{m} \right| \leq \frac{1}{10} \cdot \frac{\epsilon}{2k},$$

from which it follows that J_i^1 will contain at least $\frac{8}{10} \cdot \frac{\epsilon}{2k}$ of the weight of F .

We can prove that J_i^p contains at most $\epsilon^p/2k$ of the weight of F with probability at least $1 - p\delta\epsilon/(50k^2\ell^2)$ with an analogous argument. \square

Corollary 12 *Let J_i^p be some interval created in step 3 of **Estimate**, and suppose*

$\epsilon < 8/10$. With probability at least $1 - \delta\epsilon/(20k^2\ell)$, the datastream X satisfies

$$|X \cap J_i^p| = \Omega\left(\frac{k^5}{\epsilon^{5\ell}} \cdot \ell \cdot \log\left(\frac{k\ell}{\delta\epsilon}\right)\right). \quad (3.6)$$

Proof: By Lemma 11, we know that any given interval contains at least $(8/10)^\ell(\epsilon^\ell/2k)$ proportion of the weight of F with probability at least $1 - \delta\epsilon/(25k^2\ell)$. Since there are sufficiently many points in X , (this quantity is given in Equation (3.5)), the lemma follows from an application of the Chernoff bound. \square

Lemma 13 *With probability at least $1 - \delta/2$, the aggregate number of intervals marked in level p calls to **Estimate** is at most $2k - 1$, for all p .*

Proof: We prove this Lemma by induction on the level. Assume that at most $2k - 1$ intervals are marked in the $(p - 1)$ th iteration with probability at least $1 - (p - 1)\delta/2\ell$.

Since level p calls to **Estimate** are only executed on intervals marked in the $p - 1$ th iteration, by the inductive hypothesis, there are at most $2k - 1$ level p calls to **Estimate**, with probability $1 - (p - 1)\delta/2\ell$. Thus, the total number of level p intervals is at most $(10/9)2k/\epsilon \cdot (2k - 1)$. With probability $1 - \delta/4\ell$, Corollary 12 holds for all level p intervals.

Because Corollary 12 ensures that we have an adequate number of sample points fall in all level p intervals, we may apply Theorem 10. Thus, with probability $1 - \delta/2\ell$, **Constant?** will work as guaranteed on all level p intervals. **Constant?** will reject interval J_i^p only if F is not uniform on J_i^p (i.e. J_i^p contains a jump). Since there are at most $2k - 1$ jumps in F , at most $2k - 1$ intervals can be rejected, and thus marked in the p th iteration, with probability at least $1 - p\delta/2\ell$. \square

We summarize all the events from the above lemmas and corollary:

Corollary 14 *With probability at least $1 - \delta/2$, the following are true:*

1. Any level p interval contains at most $\epsilon^p/2k$ proportion of the weight.
2. Equation (3.6) holds for all intervals created in Step 3 of **Estimate**.
3. There are at most $2k - 1$ calls to **Estimate** at any level.
4. No call to **Constant?** fails. (Recall that **Constant?** is a Monte Carlo algorithm.)

We now prove the main theorem of this section.

Theorem 15 *With probability at least $1 - \delta$, **SmallRam** will compute an approximation to F within L^1 distance ϵ^ℓ of F , using at most 2ℓ passes and $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$ bits of memory.*

Proof: We condition this proof on all four items of Corollary 14 being true, which has probability at least $1 - \delta/2$.

We first bound the amount of memory used. At any level of the computation, we are running at most $2k - 1$ parallel copies of **Estimate**. Since each copy uses at most $O(m)$ bits of memory, the total amount of memory used by **Estimate** is at most $2k \cdot O(m)$.

Each call to **Constant?** requires at most $O((\ell \log(1/\epsilon) + \log k) \log(\ell k/\epsilon\delta))$ bits of memory. Since we run at most $O(k/\epsilon)$ copies of **Constant?** in parallel, the total memory used by our algorithm is at most:

$$2k \cdot O(m) + O\left(\frac{\ell k}{\epsilon} \left(\log \frac{1}{\epsilon} + \log k\right) \log\left(\frac{1}{\epsilon\delta}\right)\right) = O\left(\left(\frac{k^3}{\epsilon^2} + \frac{\ell k}{\epsilon} \log \frac{1}{\epsilon}\right) \log\left(\frac{\ell k}{\epsilon\delta}\right)\right).$$

We next prove that the algorithm outputs a good approximation to F . Note that after all ℓ levels of the algorithm, our approximation G consists of a partition of $\text{Support}(F)$ into disjoint intervals, with a constant density on each interval.

We classify the intervals that define the steps of G into two types.

1. Unmarked intervals that had their densities estimated in Step 6 of some call to **Estimate**. A type (1) interval J_i^p can further be classified into one of two cases:

(a) F is constant on J_i^p .

(b) F has a jump in J_i^p , but this was not detected by **Constant?**.

2. Intervals marked in level ℓ that had their density estimated by 0 in Step 7 of a level ℓ call to **Estimate**.

A bound for the error from type (1), case (a) intervals

Suppose J_i^p is a type (1), case (a) interval. It follows from the Chernoff bound and Corollary 14, item 2, that with probability at least $1 - \delta\epsilon/(32k^2\ell)$,

$$\left| \frac{|X \cap J_i^p|}{n} - \int_{J_i^p} F \right| \leq \frac{\epsilon^\ell}{4k} \int_{J_i^p} F.$$

This inequality then implies that

$$\begin{aligned} \frac{\epsilon^\ell}{4k} \int_{J_i^p} F &\geq \left| \frac{|X \cap J_i^p|}{n} - \int_{J_i^p} F \right| = \left| \int_{J_i^p} G - \int_{J_i^p} F \right| \\ &= \int_{J_i^p} |F - G|, \end{aligned}$$

where the last equality follows from the fact that F and G are both constant on J_i^p .

Let Γ be the set of all type (1) case (a) intervals. Since there are at most $((10/9)4\ell k^2/\epsilon)$ intervals in Γ , the probability that the above inequality holds for all of them is at least $1 - \delta/4$. The total error induced by all such intervals is at

most:

$$\sum_{J_i^p \in \Gamma} \int_{J_i^p} |F - G| \leq \frac{\epsilon^\ell}{4k} \sum_{J_i^p \in \Gamma} \int_{J_i^p} F \leq \frac{\epsilon^\ell}{4k}.$$

A bound for the error from type (1), case (b) intervals

Suppose that J_i^p belongs to case (b). It follows from the Chernoff bound and item 2 of Corollary 14 that with probability $1 - \delta\epsilon/(32k^2\ell)$,

$$\begin{aligned} \left| \frac{|X \cap J_i^p|}{n \text{length}(J_i^p)} - \frac{\int_{J_i^p} F}{\text{length}(J_i^p)} \right| &\leq \frac{\epsilon^\ell}{8k^2 \text{length}(J_i^p)} \int_{J_i^p} F \\ &\leq \frac{\epsilon^\ell}{8k^2 \text{length}(J_i^p)}. \end{aligned}$$

Define $\bar{F}|_{J_i^p} = \frac{1}{\text{length}(J_i^p)} \int_{J_i^p} F$ to be the *average density* of F on J_i^p . Since J_i^p was not marked, we know that subroutine **Constant?** accepted when run on J_i^p . This means that F is at most $\epsilon^\ell/2k$ in L^1 distance from constant on J_i^p :

$$\int_{J_i^p} |F - \bar{F}|_{J_i^p}| \leq \frac{\epsilon^\ell}{2k}.$$

Combining the two inequalities above gives us

$$\frac{\epsilon^\ell}{2k} + \frac{\epsilon^\ell}{8k^2} \geq \int_{J_i^p} \left| F - \frac{|X \cap J_i^p|}{n \text{length}(J_i^p)} \right| = \int_{J_i^p} |F - G|.$$

If t_1 is the number of case (b) intervals, the total error induced by all case (b) intervals is at most $(\epsilon^\ell/2k + \epsilon^\ell/8k^2) \cdot t_1$, with probability at least $1 - \delta/4$.

A bound for the error from type (2) intervals

Each interval of type (2) was created at a level ℓ call to **Constant?**. Thus by Corollary 14, item 1, each contains at most $\epsilon^\ell/2k$ proportion of the weight of F . Estimating F on these intervals by 0 will induce an error of at most $\epsilon^\ell/2k$. The total error induced by type (2) intervals is at most $\epsilon^\ell/2k \cdot t_2$, where t_2 is the number of

type 2 intervals.

A bound for the total error

Since type (1) case (b) and type (2) intervals only occur at jumps in steps of F , $t_1 + t_2 \leq 2k - 1$. Thus, with probability at least $1 - \delta$, the total error is at most

$$\begin{aligned} \frac{\epsilon^\ell}{4k} + (t_1 + t_2) \left(\frac{\epsilon^\ell}{2k} + \frac{\epsilon^\ell}{8k^2} \right) &\leq \frac{\epsilon^\ell}{4k} + (2k - 1) \left(\frac{\epsilon^\ell}{2k} + \frac{\epsilon^\ell}{8k^2} \right) \\ &\leq \frac{\epsilon^\ell}{4k} + \frac{(2k - 1)\epsilon^\ell}{2k} + \frac{\epsilon^\ell}{4k} = \epsilon^\ell. \end{aligned}$$

□

If we analyze the algorithm with $\epsilon^{1/\ell}$ in place of ϵ , we get an ϵ approximation requiring $\tilde{O}(k^3/\epsilon^{2/\ell})$ bits of memory. Note that the $k\ell/\epsilon$ term disappears, because we assume that $\ell = O(\log(1/\epsilon))$ (memory usage does not decrease beyond a constant factor if $\ell \geq \log(1/\epsilon)$).

Corollary 16 *With probability at least $1 - \delta$, there exists an algorithm that will compute an approximation to F within L^1 distance ϵ of F , using 2ℓ passes and $\tilde{O}(k^3/\epsilon^{2/\ell})$ bits of memory.*

3.3.2 Testing intervals for uniformity: Proof of Theorem 10.

We now describe the single pass subroutine **Constant?** called by **SmallRam**. Given a datastream of n samples from a mixture of k uniform distributions with density function H , **Constant?** will accept H if it is constant, and reject if it is not within β in L^1 distance of constant.

By suitably scaling and translating the input, we may assume without loss of generality that the support of H is exactly the interval $[0, 1]$. Thus, our subroutine will determine whether $\int_0^1 |H - 1| \leq \beta$ or not.

Preliminaries

The η -smoothed distribution of H . For ease of exposition of the proof of Theorem 10, we introduce the η -smoothed distribution of H_η , which is a smoothed version of H .

Definition 5 *Given a mixture of k uniform distributions that defines a probability density H and a number $0 < \eta \leq 1$, define the η -smoothed distribution to be the following distribution, with density H_η . Let $i \leq 1/\eta - 1$ be the integer such that $x \in (i\eta, (i+1)\eta]$. Then*

$$H_\eta(x) = \frac{1}{\eta} \int_{i\eta}^{(i+1)\eta} H(y) dy.$$

It is immediate that H_η is a step distribution consisting of at most $1/\eta$ steps. Thus, let h_i be the constant density of H_η on each of the intervals $(i\eta, (i+1)\eta)$. Let $\alpha_i = |h_i - 1|$ be the difference between the density of H_η and 1 on the i th interval. Our interest in H_η lies in the following useful properties.

Lemma 17 *If H is uniform, then H_η is also uniform.*

Lemma 18 *Let $\eta < \beta/5k$ and $I = (x_i, x_{i+1})$, w_i be a step of H . Suppose that*

$$|w_i \text{length}(I) - \text{length}(I)| > \frac{\beta}{2k}.$$

If H_η is the density function of the η -smoothed distribution of H , then for some step I' of H_η , we have $|H_\eta(x) - 1| > \beta/2k$ for $x \in I'$.

Proof: There are two cases.

Case 1: $w_i \text{length}(I) > \text{length}(I) + \beta/2k$. Case A can be broken down into two subcases.

Subcase A: $\text{length}(I) \geq 2\eta$. Immediately, we know that $w_i > 1 + \beta/2k$. Since $\text{length}(I) \geq 2\eta$, there must exist an integer j such that $(j\eta, (j+1)\eta) \subset I$. On the domain $(j\eta, (j+1)\eta)$, H_η will have the same density as H . Thus, the density of H_η on $(j\eta, (j+1)\eta)$ will be at least $1 + \beta/2k$.

Subcase B: $\text{length}(I) < 2\eta$. Then we know that at least half of I is contained in a single interval $(j\eta, (j+1)\eta)$ of the η partition. Also, we know that $w_i \text{length}(I) \geq \beta/2k$. By definition, for $x \in (j\eta, (j+1)\eta)$,

$$H_\eta(x) = \frac{1}{\eta} \int_{j\eta}^{(j+1)\eta} H dy \geq \frac{1}{\eta} \int_{I \cap (j\eta, (j+1)\eta)} H dy \geq \frac{1}{2\eta} \int_I H = \frac{1}{2\eta} w_i \text{length}(I) \geq \frac{5}{4} > 1 + \frac{\beta}{2k},$$

for β/k small enough.

Case 2: $w_i \text{length}(I) \leq \text{length}(I) - \beta/2k$. In this case, we must have $\text{length}(I) \geq 2\eta$.

The proof follows the treatment in **Subcase A** above. \square

Corollary 19 *If $\eta \leq \beta/5k$ and H is not within L^1 distance β of uniform, there exists a step of H_η such that $\alpha_i \geq \beta/2k$.*

The two lemmas and the corollary establish the connection between H and H_η . We will prove that **Constant?** will accept if H_η is uniform, and will reject if there exists a step with density h_i of H_η such that $|h_i - 1| > \beta/2k$. Lemma 17 and Corollary 19 establish that this is sufficient for the guarantee of Theorem 10.

The algorithm and its proof of correctness

First, we define two random variables based on the datastream that can be used to estimate α_i .

1. Let $N_i = |X \cap (i\eta, (i+1)\eta)|$ be the number of points of X that fall into the interval $(i\eta, (i+1)\eta)$. Note that $\mathbf{E}[N_i/n\eta] = h_i$.
2. Let $\hat{\alpha}_i = \left| \frac{1}{\eta} \frac{N_i}{n} - 1 \right|$ be an estimate of α_i based on the datastream. Note that $\mathbf{E}[\hat{\alpha}_i] = \alpha_i$.

The random variable $\hat{\alpha}_i$ gives an estimate of α_i , the quantity we are interested in. However, in a single pass we do not have enough space to compute and store $\hat{\alpha}_i$ for all $1/\eta$ values of i . The proof of the next lemma uses Indyk's algorithm for the small space computation of the length of a vector given as a dynamic stream of updates (see Section 2.2.1) to estimate $\|\hat{\alpha}\|_1$. From the value of $\|\hat{\alpha}\|_1$, we can glean our desired information: a small value of $\|\hat{\alpha}\|_1$ implies that all $\hat{\alpha}_i$ are small, whereas a large value implies that at least one $\hat{\alpha}_i$ is large.

Lemma 20 *In a single pass over X , we can compute an estimate ζ such that $(1/2)\|\hat{\alpha}\|_1 \leq \zeta \leq (3/2)\|\hat{\alpha}\|_1$ with probability at least $1 - \delta$, using space at most $O(\log(\eta n) \log(1/\delta))$.*

Proof: First consider the construction of a datastream S_X derived from X :

1. For each element $x \in X$, let $j_x \in [1/\eta]$ be the integer such that $x \in (j_x\eta, (j_x + 1)\eta)$. We append $\langle j_x, 1 \rangle$ to S_X .
2. We append to S_X the $1/\eta$ elements $\langle i, -\eta n \rangle$, for all $i \in [1/\eta]$.

Clearly Indyk's algorithm on input S_X can be simulated in a single pass over X . We have:

$$L_1(S_X) = \sum_{i=0}^{1/\eta} |N_i - \eta n| = \eta n \sum_i \left| \frac{1}{\eta} \frac{N_i}{n} - 1 \right| = \eta n \|\hat{\alpha}\|_1.$$

Thus, in a single pass we can derive an estimate ζ of $\|\hat{\alpha}\|_1$ such that $(1/2)\|\hat{\alpha}\|_1 \leq \zeta \leq (3/2)\|\hat{\alpha}\|_1$ with Indyk's algorithm using space at most $O(\log(\eta n) \log(1/\delta))$. \square

See Figure 3.5 for the formal description of the algorithm.

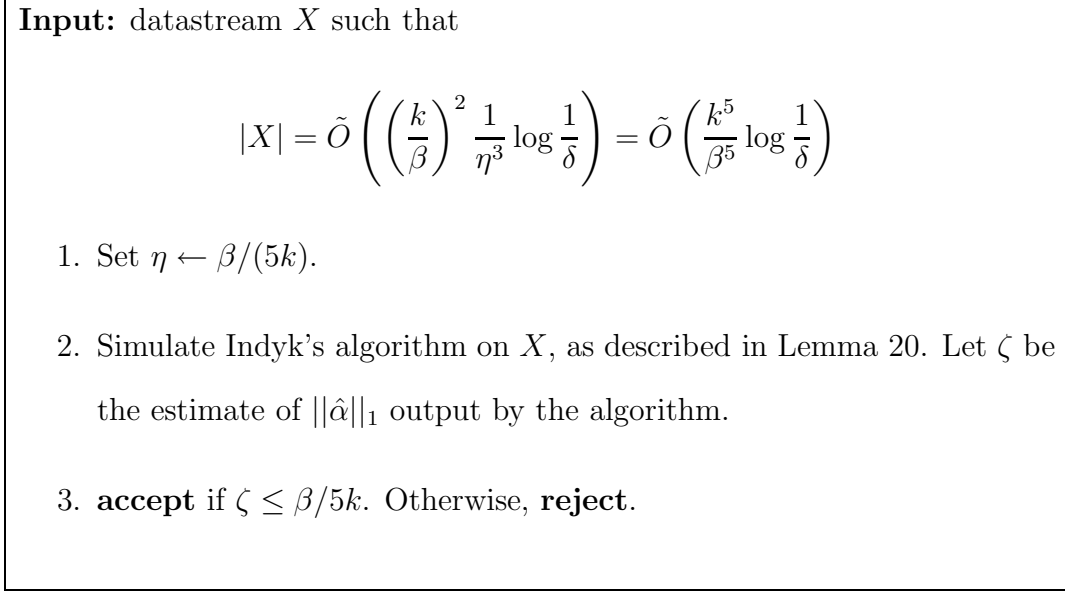


Figure 3.5: Algorithm **Constant?**

Proof of Theorem 10: Since $n = \tilde{O}(k^2/(\beta^2\eta^3) \log(1/\delta))$, an application of the Chernoff bound implies that with probability at least $1 - \delta$, for all $1/\eta = O(k/\beta)$ choices of i simultaneously: if $h_i \leq 2$ then $|N_i - h_i\eta n| \leq \frac{\beta\eta}{40k}\eta n$, which implies that

$$|\hat{\alpha}_i - \alpha_i| \leq \frac{\beta\eta}{20k}.$$

If $h_i \geq 2$ then the Chernoff bound yields $|N_i - h_i\eta n| \leq \frac{\beta\eta}{20k}h_i\eta n$, which implies that

$$\hat{\alpha}_i \geq \frac{1}{2} \geq \frac{\beta}{k}.$$

We prove the correctness of **Constant?** by considering two cases:

Case 1: H_η is uniform. Then $h_i = 1$ for all i , and $\alpha_i = 0$ for all i . Thus, $\hat{\alpha}_i \leq \frac{1}{20k}\beta\eta$ for all i . By Lemma 20, this implies that $\zeta \leq \frac{3}{2}\|\hat{\alpha}\|_1 \leq \beta/5k$; therefore

Constant? will accept H_η . Appealing to Lemma 17 it follows that **Constant?** will accept if H is uniform.

Case 2: There exists at least one h_i such that $|h_i - 1| \geq \beta/2k$. Then $\|\hat{\alpha}\|_1 \geq \beta/2k - \frac{1}{20k}\beta\eta > \beta/(2.5k)$. By Lemma 20, this implies that $\zeta \geq \frac{1}{2}\|\hat{\alpha}\|_1 > \beta/5k$; therefore **Constant?** will reject H_η . Appealing to Corollary 19, it follows that **Constant?** will reject if H is not within L^1 distance β of uniform. \square

3.4 Matching bounds on memory usage of randomized algorithms

In this section, we slightly generalize our learning problem and consider lower bounds on the amount of memory needed by any ℓ pass randomized algorithm for this new problem. We then present nearly matching upper bounds for this general problem by adapting our algorithm from the previous section.

These lower bounds will confirm that a compelling performance feature of our algorithm, its exponential tradeoff between the amount of memory required and the number of passes taken, is close to the best possible. We first define the following generalized learning problem:

Generalized Learning Problem: *Let F be the density function of a mixture of at most $1/\epsilon$ uniform distributions over the domain $[0, 1] \subset \mathbb{R}$. Let $t \in [0, 1]$ be the largest number such that F is a step distribution with at most k steps on $[0, t]$. Given a datastream X consisting of sufficiently many independent samples from F , with probability at least $1 - \delta$, find a function G and a number $t' > t$ such that $\int_0^{t'} |F - G| < \epsilon$.*

Intuitively, the problem is to learn the density function of the first k steps of a

step distribution that contains at most $1/\epsilon$ steps.

We use known lower bounds for the communication complexity of r -round protocols to prove that any ℓ pass algorithm needs at least $\Omega\left(\left(\frac{1}{2\epsilon}\right)^{1/(2\ell-1)} c^{-2\ell+1}\right)$ bits of memory to solve this problem for the $k = 3$ case. We then generalize our algorithms for learning mixtures of k -uniform distributions to algorithms for solving the generalized learning problem. These algorithms will provide an upper bound of $\tilde{O}(k^3/\epsilon^{4/\ell})$ on the number of bits of memory needed by an ℓ pass algorithms, if ℓ is an even integer.

3.4.1 A Lower Bound

We will use the communication complexity framework and the result from [56] that $R^r(\text{GT}_n) = \Omega(n^{1/r} c^{-r})$ (see Section 2.3) to prove the following Theorem:

Theorem 21 *Any ℓ -pass randomized algorithm that solves the Generalized Learning Problem for $k = 3$ with probability at least $2/3$ and error ϵ requires at least $\Omega\left(\left(\frac{1}{2\epsilon}\right)^{1/(2\ell-1)} c^{-2\ell+1}\right)$ bits of memory, for some fixed constant c .*

Proof: Fix $k = 3, \epsilon > 0$ and $m = 1/(2\epsilon)$. We will give a streaming compatible reduction from the communication problem GT_m to the Generalized Learning Problem with input parameters k and ϵ . Thus, suppose there exists an ℓ pass algorithm A that solves the Generalized Learning Problem with probability $2/3$ and that uses at most $M(A)$ bits of memory. We will prove that A induces a $2\ell - 1$ -round protocol for GT_m that uses at most $M(A)$ bits of communication per message. This will prove that $M(A) \geq R^{2\ell-1}(\text{GT}_m) = \Omega(1/(2\epsilon)^{2\ell-1})$.

Suppose that Alice and Bob are given vectors $a, b \in \{0, 1\}^m$. Alice will construct a probability density function μ_a^A . μ_a^A will be a piecewise constant function on each of the $2m$ intervals $(i/m, (i + 1/2)/m]$ and $((i + 1/2)/m, (i + 1)/m]$ as follows: set

$\mu_a^A(x) = 2a_{m-i}$ for $x \in (i/m, (i + 1/2)/m]$ and $\mu_a^A(x) = 2(1 - a_{m-i})$ for $x \in ((i + 1/2)/m, (i + 1)/m]$.

Bob will construct a similar function μ_b^B , but “reversed”: $\mu_b^B(x) = 2(1 - b_{m-i})$ for $x \in (i/m, (i + 1/2)/m]$ and $\mu_b^B(x) = 2b_{m-i}$ for $x \in ((i + 1/2)/m, (i + 1)/m]$.

Alice (Bob) generates an arbitrarily large set of independent samples drawn according to μ_a (μ_b) of arbitrary precision and places them in a datastream X_a (X_b). We require that $|X_a| = |X_b|$. The datastream formed by appending X_b to X_a , denoted by $X_a \circ X_b$, can be viewed as having been drawn from the distribution defined by density function $\mu = \frac{\mu_a^A}{2} + \frac{\mu_b^B}{2}$. Note that μ is a step function with at most $1/\epsilon$ steps.

Claim: If Bob is given the solution to the Generalized Learning Problem on datastream $X = X_a \circ X_b$ with parameter $k = 3$, he can compute $\text{GT}_m(a, b)$.

Proof of Claim: Bob has in his possession the vector b and the output of the algorithm: a number t' such that on $(0, t')$, μ is comprised of at least 3 steps, and a function μ_{approx} such that

$$\int_0^{t'} |\mu - \mu_{\text{approx}}| < \epsilon. \quad (3.7)$$

In order to compute $\text{GT}_m(a, b)$, all Bob needs to do is learn the bits a_i for $i \geq j^*$, where j^* is the highest order bit for which the vectors a and b differ. Note that μ is a step function consisting of 3 steps on the interval $(0, (j^* + 1)/m)$: it has a constant value of 1 on $(0, j^*/m)$ since $a_i = b_i$ for bits $i > j^*$. If $a_{j^*} = 1$ (and correspondingly $b_{j^*} = 0$) then it has a constant value of 2 on $(j^*/m, (j^* + 1/2)/m)$ and a value of 0 on $((j^* + 1/2)/m, (j^* + 1)/m)$. If $a_{j^*} = 0$ (and correspondingly $b_{j^*} = 1$), the situation is reversed. Note that $t' \geq (j^* + 1)/m$.

Bob can compute $\text{GT}_m(a, b)$ in the following manner. He finds (by enumeration) a vector $\tilde{a} \in \{0, 1\}^m$ that induces the probability density $\tilde{\mu} = \frac{\mu_a^A}{2} + \frac{\mu_b^B}{2}$, such that $\int_0^t |\tilde{\mu} - \mu_{\text{approx}}| \leq \epsilon$. Bob then outputs 1 iff $\tilde{a} \geq b$.

The correctness of this scheme follows from the observation that $\tilde{a}_i = a_i$ for all bits $i \geq j^*$. Suppose this were not the case, that \tilde{a} differed from a by a bit a_i for $i \geq j^*$. Then $\int_0^t |\tilde{\mu} - \mu| \geq 2\epsilon$, but since Bob is guaranteed inequality (3.7), it follows from the triangle inequality that $\int_0^t |\tilde{\mu} - \mu_{\text{approx}}| \geq \epsilon$, which is a contradiction. Thus, $\tilde{a}_{j^*} = a_{j^*}$. This proves the claim.

We now describe the $2\ell - 1$ -round protocol for computing $\text{GT}_m(a, b)$. Alice simulates the first pass of algorithm A on X_a to find the contents of memory at this intermediate stage of the pass. She sends these $M(A)$ bits of memory to Bob. Bob continues the simulation of the first pass of A on X_b and sends his $M(A)$ bits of memory to Alice. They simulate each pass of A in this fashion until all ℓ passes have been completed, sending a total of $2\ell - 1$ messages of size at most $M(A)$.

After the completion of the communication, Bob will have the output of A run on datastream $X_a \circ X_b$, from which he can determine $\text{GT}_m(a, b)$. \square

3.4.2 An Upper Bound

In this section we generalize our algorithm for learning a mixture of k -uniform distributions to one that will solve our generalized learning problem. Our algorithm will provide the following upper bound on the amount of memory needed by a 2ℓ pass algorithm:

Theorem 22 *There exists a 2ℓ pass algorithm that will solve the Generalized Learning Problem with probability at least $1 - \delta$ and error ϵ^ℓ , using at most $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$ bits of memory.*

By transforming the parameters ϵ and ℓ , we arrive at the following corollary, which is an upper bound comparable to the lower bound of Theorem 21.

Corollary 23 *For even ℓ , there exists an ℓ -pass algorithm that can solve the Generalized Learning problem with probability $2/3$ and error ϵ using at most $\tilde{O}(k^3/\epsilon^{4/\ell})$ bits of memory.*

We show how to modify algorithm **SmallRam** from the previous section to solve our generalized problem. Recall the recursive organization of **SmallRam**. The 2ℓ passes were organized into ℓ successive calls to **Estimate**. The p th call to **Estimate** was assigned level p .

The input to a call to **Estimate** is an interval J . In the first pass of this call, we drew a sample from $X|_J$ and partitioned J into level p subintervals with weight $\Theta(\epsilon)$ (or $\Theta(\epsilon/k)$ if $p = 0$) of the weight that fell in J . In the second pass, we tested each subinterval for uniformity by calling subroutine **Constant?**; if F was close to constant on a subinterval then we estimated F 's density on J with high accuracy using the entire datastream. If F was not close to constant on J , we marked the subinterval and recursively estimated F on the subinterval in subsequent passes. In the final pair of passes, marked intervals were estimated by a constant 0.

In order to adapt **SmallRam** to solving the generalized learning problem, we make the following three modifications:

1. In Step 5, call **Constant?** with error parameter $\epsilon^{\ell+1}/2$ and with input parameter $1/\epsilon$ in lieu of k (i.e. **Constant?** will assume that its input is a mixture of at most $1/\epsilon$ uniform distributions, rather than k). By Theorem 10, **Constant?** will need $O(\ell \log(1/\epsilon) \log(\ell k/\epsilon\delta))$ bits of memory.
2. After all level p calls to **Estimate** terminate, we may potentially have a large number of level $p+1$ recursive calls to **Estimate**. We do not execute them all,

but rather only the calls on the k intervals that are closest to 0. We “drop” the recursive calls on all other intervals, and thus do not have estimates of F on the entire domain.

3. In addition to outputting the densities of intervals, output $t \in (0, 1)$, which is the largest number such that we have estimates of $F(x)$ for all $x \in (0, t)$.

Proof sketch of Theorem 22: We first bound the amount of memory needed by the modified algorithm. Due to our second modification, we know that there are at most k recursive calls at any level. Each of these parallel calls takes a sample of size $\tilde{O}(k^2/\epsilon^2)$ in its first pass, and in its second pass makes $O(k/\epsilon)$ calls to **Constant?**, each of which uses at most $O(\ell \log(1/\epsilon) \log(\ell k/\epsilon\delta))$ bits of memory. Thus, the total memory usage of the algorithm is $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$.

We next bound the error of our algorithm for the generalized learning problem. The crux of the analysis of the error of **SmallRam** relies on four properties. We present their analogs for the generalized algorithm. The first three properties are proved in a similar manner to those of **SmallRam** by applying VC and Chernoff bounds; we will not repeat these arguments here. The fourth follows immediately from the modifications given above.

1. The weight of F that lies in each level ℓ interval is at most $\epsilon^\ell/2k$.
2. Estimating F on unmarked intervals on which F is constant incurs a negligible error, due to the large datastream.
3. Estimating F by a constant on unmarked intervals on which F contains a jump incurs an error of at most $\epsilon^{\ell+1}/2$, due to our guarantees for **Constant?**.
4. At most k recursive calls were made at each level p .

Let G be the approximation to the first k steps of F produced by our modified algorithm, given by: $\int_0^t |F - G|$. The algorithm outputs G as a constant on a set of intervals that partitions $(0, t)$. There are three types of intervals: (a) unmarked intervals in which F does not contain a jump, (b) unmarked intervals in which F does contain a jump (there are at most $1/\epsilon$ since F is a mixture of at most $1/\epsilon$ distributions), (c) marked level ℓ intervals estimated as 0 (there are at most k).

The total error is then:

$$\int_0^t |F - G| = \text{error from (a)} + \text{error from (b)} + \text{error from (c)} \leq 1/\epsilon \cdot \epsilon^{\ell+1}/2 + k \cdot \epsilon^\ell/2k \leq \epsilon^\ell.$$

□

3.5 Learning mixtures of linear distributions

A linear distribution has a density function that is a linear function defined over a continuous interval in \mathbb{R} . In this section we consider the problem of learning a mixture of k linear distributions in \mathbb{R} . Let F be the density function of the mixture. We will need to make the assumption that the algorithm is given an upperbound on F , call it w : $F(x) \leq w$ for all x in the domain. As with mixtures of uniform distributions, we can describe F as a piecewise linear density function with at most $O(k)$ different pieces: $a_i x + b_i$ for $x \in (x_i, x_{i+1})$. Note that a mixture of uniform distributions is a special case of a mixture of linear distributions.

3.5.1 Testing intervals for linearity.

Constant? can be strengthened to test whether or not a datastream of samples is drawn from a distribution that is within L^1 distance β of a single linear distribution.

For the rest of this section, we will write “linear” in lieu of “single linear distribution.” We will prove the following analog to Theorem 10.

Theorem 24 *Let H be the density function of a probability distribution that is piecewise linear, with at most k . Let $w \geq 0$ be an upperbound on H (i.e. $H(x) \leq w$ for all $x \in \mathbb{R}$). Given a datastream X consisting of $\tilde{O}(k^5 w^5 / \beta^5)$ samples drawn according to H , there exists an algorithm **Linear?** that with probability $1 - \delta$ will **accept** if H is linear and will **reject** if H is not within L^1 distance β of linear. The algorithm uses $O((\log(1/\beta) + \log k + \log w) \log(1/\delta))$ bits of memory.*

By suitably scaling, we may assume that the domain of H is the interval $(0, 1)$. As before, for ease of exposition of the proof of correctness of **Linear?**, we will work with the η -smoothed distribution of H . Note that the η -smoothed distribution of H will be constant on each of the $1/\eta$ intervals $(j\eta, (j+1)\eta)$ that define the partition for the η -smoothed distribution. Let h_i be the density of the distribution on the interval $(i\eta, (i+1)\eta)$.

Denote by $\text{interp}_H(x)$ the line that passes through the two points $(\eta/2, H_\eta(\eta/2))$ and $(1 - \eta/2, H_\eta(1 - \eta/2))$. More explicitly, it is given by the function $\text{interp}_H(x) = ax + b$ where

$$a = \left(\frac{h_{\frac{1}{\eta}-1} - h_0}{1 - \eta} \right) \quad (3.8)$$

and $b = h_0 - a\eta/2$. It is immediate that if H is linear, then $H(x) = \text{interp}_H(x)$.

Set $\alpha_j = \left| h_j - \frac{1}{\eta} \int_{j\eta}^{(j+1)\eta} \text{interp}_H(x) \right|$ to be the difference between h_i and the average value of $\text{interp}_H(x)$ on the interval $(j\eta, (j+1)\eta)$.

Lemma 25 *If H is linear, then $\alpha_j = 0$ for all j .*

Lemma 26 *Let $\eta \leq \beta/(6kw)$. Suppose that there exists some component of H , $a_i x + b_i$ for $x \in (x_i, x_{i+1})$, such that $\int_{x_i}^{x_{i+1}} |H(x) - \text{interp}_H(x)| dx \geq \beta/k$. Then there exists a j such that $\alpha_j \geq \beta/2k$.*

Proof: Define $J = \{j \in \mathbb{Z} | (j\eta, (j+1)\eta) \cap (x_i, x_{i+1}) \neq \emptyset\}$ to be the set of indices that correspond to intervals that intersect (x_i, x_{i+1}) . Let γ_j be the L_1 distance between $H(x)$ and $\text{interp}_H(x)$ on the interval $(j\eta, (j+1)\eta)$, given by $\gamma_j = \int_{j\eta}^{(j+1)\eta} |H(x) - \text{interp}_H(x)| dx$.

Claim: There are at most 3 values of $j \in J$ such that $\frac{1}{\eta}\gamma_j \neq \alpha_j$.

Proof of Claim: There are only three ways this can occur:

1. When $(j+1)\eta > x_{i+1}$. This only occurs for one value of j , $\max_{j \in J} j$.
2. When $j\eta \leq x_i$. Similarly, this only occurs for one value of j , $\min_{j \in J} j$.
3. The definitions of α_j and H_η imply that

$$\alpha_j = \left| h_j - \frac{1}{\eta} \int_{j\eta}^{(j+1)\eta} \text{interp}_H(x) dx \right| = \frac{1}{\eta} \left| \int_{j\eta}^{(j+1)\eta} (H(x) - \text{interp}_H(x)) dx \right|.$$

It is apparent that $\alpha_j \neq \frac{1}{\eta}\gamma_j$ only when $H(x) - \text{interp}_H(x)$ changes sign in $(j\eta, (j+1)\eta)$. For j such that $(j\eta, (j+1)\eta) \subset (x_i, x_{i+1})$, this occurs for at most one j , since $H(x)$ and $\text{interp}_H(x)$ are both linear in (x_i, x_{i+1}) .

This proves the claim.

The assumption that $a_i x + b_i \leq w$ for all $x \in (x_i, x_{i+1})$ implies that

$$\gamma_j = \int_{j\eta}^{(j+1)\eta} |a_i x + b_i - \text{interp}_H(x)| \leq w\eta, \quad (3.9)$$

for all j such that $(j\eta, (j+1)\eta) \subseteq (x_i, x_{i+1})$.

From Inequality (3.9) with $\eta \leq \beta/(6kw)$ and $\sum_{j \in J} \gamma_j \geq \beta/k$, it follows that $4 \leq |J| \leq 1/\eta$. Again from Inequality (3.9) and then averaging, it follows that for at least four of $j \in J$, $\gamma_j \geq \beta\eta/2k$.

From our claim, we know that $\alpha_j = \gamma_j/\eta \geq \beta/2k$ for at least one of these four j .

□

Corollary 27 *If H is not within L^1 distance β of linear, there exists a j such that $\alpha_j \geq \beta/2k$.*

As in the uniform case, we estimate the value of α_i from the datastream.

1. Let $\text{interp}_H^{\hat{}}$ be some estimate of interp_H , such that

$$\left| \int_{j\eta}^{(j+1)\eta} \left(\text{interp}_H^{\hat{}}(x) - \text{interp}_H(x) \right) dx \right| = O\left(\frac{\beta\eta}{k}\right).$$

(we will show how to compute $\text{interp}_H^{\hat{}}$ later).

2. Let $N_i = |X \cap (j\eta, (j+1)\eta)|$.

3. Let $\hat{\alpha}_i = \left| \frac{N_i}{n\eta} - \frac{1}{\eta} \int_{i\eta}^{(i+1)\eta} \text{interp}_H^{\hat{}}(x) dx \right|$.

If the datastream contains $n = \tilde{O}(k^5 w^2 / \beta^5)$ points, then we can decide if any α_i satisfies $\alpha_i \geq \beta/2k$ by approximating $\|\hat{\alpha}\|_1$ using Indyk's algorithm as we did for **Constant?**. With Lemma 25 and Corollary 3.5.1 in lieu of their counterparts from Section 3.3.2, the proof of correctness is the same as for Theorem 10, except for the added (but straightforward) accounting of the error caused by estimating $\text{interp}_H(x)$.

All that remains to be shown is how to find the estimate $\text{interp}_H^{\hat{}}$. In order to do so, we estimate the values of h_0 and $h_{\frac{1}{\eta}-1}$. Our estimates are $\hat{h}_0 = N_0/(n\eta)$ and $\hat{h}_{1/\eta-1} = N_{1/\eta-1}$, respectively, which can be computed in the pass simultaneously with Indyk's algorithm in small space. (Note that when we use Indyk's algorithm to approximate α_i , we do not need to know $\text{interp}_H^{\hat{}}$ until after the datastream has been examined. See the proof of Lemma 20.) Since $n = \tilde{O}(k^2 w^2 / (\beta^2 \eta^3 \log(1/\delta)))$, it follows from the Chernoff bound that $|N_i - n\eta h_i| = O(\beta\eta/k)h_i\eta n$. Thus, $|\hat{h}_i - h_i| = O(\beta\eta/k)$, under the assumption that $h_i \leq \max_x H(x) \leq w$. By Equation (3.8), our error in calculating the slope and intercept of $\text{interp}_H(x)$ will be at most $O(\beta\eta/k)$.

As desired, for all j we have

$$\left| \int_{j\eta}^{(j+1)\eta} \left(\text{interp}_H(x) - \text{interp}_{\hat{H}}(x) \right) dx \right| = 2O(\beta\eta/k)\eta = O(\beta\eta^2/k).$$

Remark When we use **Linear?** as a subroutine for our algorithm, we will need to determine if some subinterval I of the domain of H is close to linear. Call the length of this subinterval l . Testing I for linearity can be done with the same algorithm as above, except we only analyze the subinterval (without scaling it to be $(0, 1)$). In this case, the η smoothed distribution won't consist of $1/\eta$ intervals, but rather only l/η intervals. We compute the line $\text{interp}_{H|_I}$ from the first and last of the l/η intervals.

3.5.2 The algorithm

The 2ℓ pass algorithm for finding a function G that approximates F to within L^1 distance ϵ^ℓ is the same as **SmallRam**, except we use subroutine **Linear?** in lieu of **Constant?**, and for those intervals J that we do not mark (i.e. they are within ϵ^ℓ of linear) we approximate F by $\text{interp}_{F|_J}$, which we showed how to find accurately in the previous section.

Theorem 28 *There exists an algorithm that, with probability at least $1 - \delta$, can learn a mixture of k linear distributions within L^1 distance ϵ^ℓ using at most $\tilde{O}(k^3/\epsilon^2)$ bits of memory. The algorithm requires the datastream to have size $\tilde{O}(k^5 w^5 / \epsilon^5)$.*

The proof of the above theorem follows the proof of Theorem 15.

3.6 Learning mixtures of uniform distributions in two dimensions

We now consider the problem where F is the density function of a mixture of k uniform distributions on axis-aligned rectangles in $(0, 1) \times (0, 1) \subset \mathbb{R}^2$. We call such a distribution a *mixture of k uniform(2) distributions*. Similarly, we will call a mixture of uniform distributions over intervals in \mathbb{R} a *mixture of uniform(1) distributions*.

F can be described by k axis aligned rectangles, $R_i \subseteq (0, 1) \times (0, 1)$, $i = 1, \dots, k$, each with weight w_i . Our algorithms will handle the case where the rectangles intersect. Note that each of these rectangles is defined by four boundary lines. We say that F is *vertical* on a rectangle $R = (0, a) \times (0, b)$ if R does not contain any of the horizontal boundary lines of the R_i s that define F .

3.6.1 Testing rectangles for verticality

Our algorithm will require an analog to Theorem 10. In this section we will prove the following Theorem, using the same technique as the proof of Theorem 10. In \mathbb{R}^2 , the L^1 distance between two functions $F, G : \mathbb{R}^2 \rightarrow \mathbb{R}$ is given by $\int_{\mathbb{R}^2} |F - G|$.

Theorem 29 *Let H be the density function of a mixture of k uniform(2) distributions over universe $R \subseteq \mathbb{R}^2$, and let w be an upper bound on H (i.e. $H(x) \leq w$ for all $x \in \mathbb{R}^2$) and let X be a datastream of samples drawn according to H , such that $|X| = \tilde{\Omega}(w^2 k^{12} / \epsilon^{12})$. There exists a single pass algorithm that with probability $1 - \delta$ will **accept** if H is vertical on R and will **reject** if H is not within L^1 distance β of a vertical mixture of $8k$ -uniform(2) distributions, using at most $O((\log 1/\beta + \log k + \log w) \log(1/\delta))$ bits of memory.*

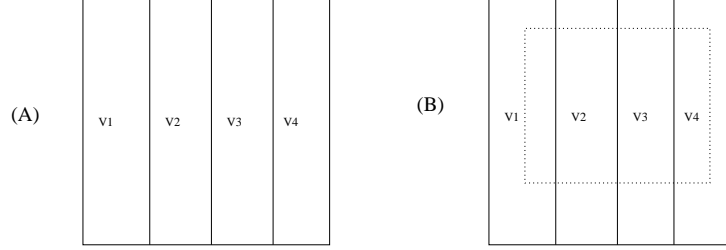


Figure 3.6: (A) The ζ -area partition of a rectangle. (B) A rectangle of the mixture given by dotted lines, superimposed on the partition. Note that since V_2 and V_3 are adjacent rectangles that do not contain corners, their densities are the same.

We may assume that $R = (0, a) \times (0, b)$ is a rectangle. We first define the ζ -area partition of R , which is a two dimensional analog to the partition associated with the η smoothed distribution.

Definition 6 *Given a rectangle $R = (0, a) \times (0, b)$, a ζ -area partition of R is a partition of R into a set of $1/\zeta$ vertical rectangles of equal area, $\{V_i\}$, such that $V_i = (i\zeta a, (i + 1)\zeta a) \times (0, b)$.*

See Figure 3.6a. Let the $\beta/8kw$ -area partition of R be given by the set $\{V_i\}$. Given a rectangle $S = (a_1, a_2) \times (b_1, b_2) \subseteq R$, define the function $\tilde{H}_S : (b_1, b_2) \rightarrow \mathbb{R}$ by $\tilde{H}_S(x) = \int_{a_1}^{a_2} H(x, y) dx$.

Lemma 30 *If H is vertical on R , then each \tilde{H}_{V_i} is a constant function, for all V_i in the ζ -area partition of R .*

Lemma 31 *If $\zeta < \beta/(8kw)$ and each \tilde{H}_{V_i} is within L^1 distance $\beta^2/16wk$ of a constant function for all V_i , then H is within L^1 distance β of a vertical mixture of at most $8k + 1$ uniform(2) distributions on R .*

Proof: Assume the condition of the lemma holds. We will show the existence of a vertical function G that is within L^1 distance β of H .

Recall that H is a mixture of uniform distributions on k different axis aligned rectangles. Each of these constituent rectangles has four corners. Thus, at most $2k$ of the V_i s contain corners of rectangles.

Claim: Assuming the condition of the lemma, if V_i and V_{i+1} are adjacent rectangles of the ζ -area partition, that do not contain corners, then H on $V_i \cup V_{i+1}$ can be approximated by a constant function with error in L^1 distance at most $2\beta^2/16wk$.

Proof of claim: Since V_i does not contain any corners of mixture rectangles, it does not contain any vertical boundary edges (since V_i itself takes up an entire vertical strip of R). Furthermore, any horizontal boundary edge must cut through all of V_i . Thus, $H(x_1, y) = H(x_2, y) = \tilde{H}_{V_i}(y)$ for all $(x_1, y), (x_2, y) \in V_i$. Thus, if $\tilde{G}_{V_i}(y) = c$ is a constant approximation to the function \tilde{H}_{V_i} with L^1 error at most $\beta^2/16wk$, then the constant function $G_{V_i}(x, y) = c/\zeta$ is a good approximation to H in V_i : $\int_{V_i} |H - G_{V_i}| \leq \beta^2/(16wk)$.

If V_i and V_{i+1} are adjacent strips such that neither contain corners of endpoints, then it follows that $H(x_1, y) = H(x_2, y) = \tilde{H}_{V_i}(y) = \tilde{H}_{V_{i+1}}$, for all $(x_1, y), (x_2, y) \in V_i \cup V_{i+1}$. Thus, the function $G_{V_i \cup V_{i+1}} = c/\zeta$ will be within L^1 distance $2\beta^2/16wk$ from H on $V_i \cup V_{i+1}$. See Figure 3.6b.

Using induction, it can be shown that H can be approximated by a constant function on m adjacent partition rectangles that do not contain corners with error at most $m\beta^2/16kw$.

We now show how we can partition R into $8k + 1$ vertical rectangles and assign a constant value on each rectangle, such that the induced function on R is within β in L^1 distance of H : **(a)** H on each of the at most $2k$ V_i s that contain a corner can be estimated arbitrarily by 0. The total weight of H in each V_i is at most $w\zeta \leq \beta/8k$. Since there are at most $2k$ of these V_i s, the total error is at most $2k \cdot \beta/8k = \beta/2$.

(b) The remaining kw/β V_i s can be partitioned into a set of at most $4k + 1$ larger rectangles, such that H on each large rectangle T that contains t of the V_i s can be estimated by a constant with error at most $t\beta^2/16wk$. Since there are at most $8kw/\beta$ of the V_i s, the total error is at most $\beta/2$. \square

Thus, we will design our algorithm to accept if all \tilde{H}_{V_i} are constant, and reject if at least one \tilde{H}_{V_i} is not within L^1 distance $\beta^2/16kw$ of constant. If H is vertical on R , then our algorithm will accept since all \tilde{H}_{V_i} will be constant; if H is not within β of a vertical mixture of $8k$ rectangles, then our algorithm will reject.

We now show how to determine whether or not all of the distributions defined by the \tilde{H}_{V_i} s are close to constant. The crucial observation is that since \tilde{H}_{V_i} is a mixture of k uniform(1) distributions over \mathbb{R} , we can determine if it is close to uniform in a manner similar to **Constant?**. We outline the details below.

Set $\eta = \beta^2/80k^2w$ and let the η -smoothed distribution of \tilde{H}_{V_i} be $\tilde{H}_{V_{i\eta}}$. Let the intervals defined by the η -smoothed distribution be $(x_0^i, x_1^i), (x_1^i, x_2^i), \dots, (x_{1/\eta-1}^i, x_{1/\eta}^i)$ and h_j^i be the density of $\tilde{H}_{V_{i\eta}}$ on the interval (x_j^i, x_{j+1}^i) . Note that $x_j^i = x_0 + j\eta$.

As in Section 3.3.2, it can be shown that if $\alpha_j^i = \left| h_j^i - \zeta \int_{(x_j^i, x_{j+1}^i) \times (b_1, b_2)} H \right| \leq \beta^2/32wk^2$ for all j , then \tilde{H}_{V_i} is within $\beta^2/16wk$ of uniform. Thus, our algorithm determines whether or not $\alpha_j^i \leq \beta^2/32wk^2$ for all i and j . In order to estimate each α_j^i , we define the following random variables:

1. $N_j^i = |X \cap (V^i \cap [(x_j^i, x_{j+1}^i) \times (0, b)])|$.
2. $\hat{\alpha}_j^i = |N_j^i/n\eta - \sum_j N_j^i/n|$. Note that $\mathbf{E}[\hat{\alpha}_j^i] = \alpha_j^i$.

If we require that $|X| = \tilde{\Omega}(w^2k^4/(\beta^4\eta^4\zeta^2)) = \tilde{\Omega}(w^2k^{12}/\beta^{12})$, then from the VC bound, for all i, j it follows that $|N_j^i/n\eta - h_j^i| \leq \beta^2\eta\zeta/200wk^2$ and that $|N_j^i/n - \int_{V_i} H| \leq \beta^2\eta\zeta/200wk^2$.

Thus, $|\alpha_j^i - \hat{\alpha}_j^i| \leq \beta^2 \eta \zeta / 100wk^2$. Let α be the vector of dimension $1/\eta\zeta$ whose entries consist of all the α_j^i s. As we argued before, if all $1/\eta\zeta$ of the $\alpha_j^i = 0$, then $|\hat{\alpha}_j^i| \leq \beta^2 \eta \zeta / 100wk^2$ for all i, j . This implies that $\|\hat{\alpha}\|_1$ is less than $\beta^2 / 100wk^2$. If some $|\alpha_j^i| \geq \beta^2 / 32wk^2$, then this implies that some $|\hat{\alpha}_j^i| \geq \beta^2 / 50wk^2$. This implies that $\|\hat{\alpha}\|_1 \geq \beta^2 / 50wk^2$. Thus, if our algorithm accepts if $\|\hat{\alpha}\|_1 \leq \beta^2 / 100wk^2$, rejects if $\|\hat{\alpha}\|_1 \geq \beta^2 / 50wk^2$, and otherwise acts arbitrarily, then it will accept if H is vertical and will reject if it is not within β of vertical.

We use Indyk's algorithm to approximate $\|\hat{\alpha}\|_1$ to within a factor of $1/4$ in a single pass using space $O((\log 1/\beta + \log k + \log w) \log 1/\delta)$. With this solution, we can solve the above problem.

3.6.2 The Algorithm

The algorithm for learning a mixture of k -uniform(2) distributions is similar to the algorithm in the one dimensional case. We show how to modify **SmallRam** in order to learn a mixture of k uniform(2) distributions:

Theorem 32 *Let F be the density function of a mixture of k uniform(2) distributions, such that $F(x) \leq w$ for all $x \in \mathbb{R}^2$. Suppose X is a datastream drawn according to F , with a sufficient number of elements. There exists a $4\ell + 1$ -pass algorithm that approximates F to within L^1 distance ϵ^ℓ using at most $\tilde{O}(\ell k^4 / \epsilon^3 + \ell^2 k / \epsilon)$ bits of memory.*

Proof sketch: The main algorithm can be organized into ℓ recursive calls, each of which requires two passes. At a p th level call, the input is a rectangle $R^p = (0, a) \times (0, b) \subseteq R$. We give an outline of the recursive algorithm:

1. In the first pass, we draw a sample of size m from $X|_{R^p}$. If $p = 1$, we set $m = \Theta(\epsilon^2 / k^2)$; for subsequent p , we set $m = \Theta(\epsilon^2)$. We use this sample to

partition R^p into a set of rectangles $\{R_i\}$, such that

- (a) Each rectangle R_i can be written as $(0, a) \times (x_i^1, x_i^2)$ (i.e. they are “horizontal”).
- (b) Each rectangle has at least 0.8ϵ (or $0.8\epsilon/k$ if $p = 1$) proportion of the weight of F in R^p and at most 0.9ϵ (or $0.9\epsilon/k$ if $p = 1$). It can be shown that $0.8^p \epsilon^p / k \leq \int_{R_i} F \leq 0.9^p \epsilon^p / k$.

2. In the second pass of each pair of passes, we check each of the $\Theta(k/\epsilon)$ subrectangles R_i to see if they are within $\epsilon^{\ell+1}/8k$ of vertical using the one pass algorithm from Section 3.6.1. We set the failure probability to be at most $1 - O(\delta \epsilon^{c\ell} k^c)$ for some constant c . With these input parameters, each call to the subroutine will require $O((\log 1/\epsilon + \log k + \log w)\ell^2 \log(k/\epsilon\delta))$ bits of memory. We mark the R_i s that are rejected. Note that since the R_i s are horizontal rectangles, at most $2k$ of them can contain horizontal boundary lines; thus, at most $2k$ of them are marked.
3. If F is within $\epsilon^{\ell+1}$ of a mixture of $8k$ vertical uniform(2) distributions in R , then projecting points onto the x axis will form a mixture of at most $4k$ uniform(1) distributions. Call this distribution \tilde{F}_{R_i} . We can learn \tilde{F}_{R_i} to within L^1 distance $\epsilon^{\ell+1}/2$ by calling our 2ℓ pass algorithm **SmallRam** from the previous section. Call this approximation \tilde{F}'_{R_i} and set $G_{R_i}(x, y) = \tilde{F}'_{R_i}(x)$ for all $(x, y) \in R_i$, to be our approximation to F in R_i . We claim that

$$\int_{(x,y) \in R_i} |G_{R_i}(x, y) - F(x, y)| \leq \frac{\epsilon^{\ell+1}}{4k}.$$

It is straightforward to prove that \tilde{F}'_{R_i} is within L^1 distance $\epsilon^{\ell+1}/8k$ of the distribution constructed in the proof of Lemma 31. Since this distribution is

within L^1 distance $\epsilon^{\ell+1}/8k$ of F in R , by the triangle inequality we know that F is within L^1 distance $\epsilon^\ell/4k$ of our approximation \tilde{F}'_{R_i} .

4. In parallel, recursively approximate the density of F in each of the rectangles R_i that were marked in Step 2.

The above algorithm can be implemented in $4\ell + 1$ passes; the 2ℓ passes used by each call to **SmallRam** can be run in parallel with the passes of the main algorithm. The algorithm needs $\tilde{O}(\ell k^4/\epsilon^3 + \ell^2 k^2/\epsilon^2)$ bits of memory, since each call to **SmallRam** uses $\tilde{O}(k^3/\epsilon^2 + \ell k/\epsilon)$ bits of memory, and there are at most $O(\ell k/\epsilon)$ parallel calls to **SmallRam**.

The proof that the total error of the function output by the algorithm does not exceed ϵ^ℓ is similar to the analysis of the error of **SmallRam**. \square

If we transform ϵ , we arrive at the following corollary.

Corollary 33 *There exists a $4\ell + 1$ pass algorithm that will learn F to within L^1 distance ϵ using at most $\tilde{O}(k^4/\epsilon^{3/\ell})$ bits of memory.*

Chapter 4

Pass-Efficient Algorithms for the Facility Location Problem

In this chapter, we consider pass-efficient algorithms for the fundamental worst-case clustering problem of facility location.

Facility Location Problem: *Given a metric space (X, d) and a facility cost f_i for each $x_i \in X$, find a set of facilities F that minimizes the objective function: $\sum_{x_i \in F} f_i + \sum_{x_i \in X} d(x_i, F)$.*

We present a $3\ell - 1$ pass algorithm that uses at most $\tilde{O}(k^* n^{2/\ell})$ bits of memory, where k^* is the number of facilities opened by our approximate solution. The algorithm will achieve an approximation ratio of $O(\ell)$. Although the facility location problem and the problem of learning mixtures of linear distributions seem completely unrelated, the paradigm of adaptive sampling implemented in a small number of passes will apply to both situations. Indeed, our facility location algorithm exhibits the same sharp tradeoff between the number of passes taken by the algorithm and the amount of memory required: the amount of memory will decrease with $1/\ell$ in the exponent.

In Section 4.1 we will review a known sampling-based k -median algorithm that will provide some of the ideas that we will use in Section 4.2 for our main facility location algorithm. In Section 4.3, we will consider algorithms for k -facility location, a problem that we define to be a hybrid between k -median and facility location. Lastly, in Section 4.4 we prove lower bounds on the space usage of ℓ -pass facility location algorithms; although the lower bounds will be far from tight, they will give us much insight into the intrinsic space complexities of the problem.

4.1 Review of Indyk’s sublinear k -median algorithm

Recall that the k -median problem is to find a set $C \subset X$, such that $|C| = k$, that minimizes the objective function $\sum_{x_i \in X} d(x_i, C)$. In this section, we give a sketch of a sampling-based algorithm for k -median by Indyk [42] that provides some of the intuition for our facility location algorithm in Section 4.2. The algorithm runs in time $\tilde{O}(kn)$; since the input contains the representation of a metric space consisting of $\Theta(n^2)$ pairwise distances, this is a sublinear algorithm for massive data sets. Let OPT be the value of the optimum objective function of the k -median instance. We call a k -median algorithm an (α, β) -bicriterion approximation algorithm if it produces a solution with cost at most αOPT using at most βk centers. We assume that such an algorithm exists, and we treat it as a black box. A possible candidate for the black box includes the $(1 + \gamma, 1 + 2/\gamma)$ algorithm (for any $\gamma > 0$) by Charikar and Guha [15] that runs in time $\tilde{O}(n^2)$.

Indyk’s algorithm can be roughly stated as follows:

1. Take a sample of size $s = O(\sqrt{kn} \log k)$ from the nodes.

2. Run the black box algorithm on the sample to get a set C of βk centers.
3. Sort the points in X based on their distance from C (i.e. $d(x_i, C)$), and identify the set R that contains the $n - \tilde{O}(\sqrt{nk})$ points closest to the centers in C .
4. Run the black box algorithm on $X \setminus R$ to get an additional βk centers, and output all $2\beta k$ centers.

Indyk proved that the above algorithm would output a solution with $2\beta k$ centers that costs at most $O(1)\text{OPT}$, where OPT is the optimum objective function value for a solution with exactly k centers.

We sketch the proof of correctness of the algorithm. Fix an optimum k -median solution with centers c_1, \dots, c_k and let C_i denote the set of points serviced by center c_i . Let $I = \left\{ i : |C_i| \geq \tilde{\Omega}\left(\sqrt{\frac{n}{k}}\right) \right\}$ be the set of indices that correspond to “large” clusters in the optimum solution. Since these large clusters should be well-represented in the sample S , we expect that the centers in C are a “good approximation” for all points that lie in the set $\cup_{i \in I} C_i$, that is $\sum_{x_i \in \{\cup_{i \in I} C_i\}} d(x_i, C) \leq O(1)\text{OPT}$. (This intuition can be formalized by applying the Markov inequality and Chernoff bounds.) Since the number of points that lie outside of $\cup_{i \in I} C_i$ is at most $k \cdot \tilde{O}\left(\sqrt{\frac{n}{k}}\right) = \tilde{O}(\sqrt{kn})$, it follows that $\sum_{x_i \in R} d(x_i, C) \leq \sum_{x_i \in \{\cup_{i \in I} C_i\}} d(x_i, C) \leq O(1)\text{OPT}$.

Using the k -median black box, the algorithm then clusters the set $X \setminus R$ consisting of outlying points that are not close to the centers in C ; since the optimum centers can induce a clustering of cost at most 2OPT on the nodes in $X \setminus R$, the cost of clustering the outliers with βk centers will be at most $O(1)\text{OPT}$. The algorithm runs in time $\tilde{O}(kn)$, since it runs an $\tilde{O}(n^2)$ time bicriterion black box algorithm for k -median on instances of size $\tilde{O}(\sqrt{kn})$.

4.2 The Algorithm: Pass-Efficient FL

In this section, we prove the main result of the chapter:

Theorem 34 *Let $\ell > 0$ be any integer. With probability at least $1 - 1/n$, $3(\ell - 1)$ pass algorithm **Pass-Efficient FL** will output a solution with cost at most $36(\ell - 1)\alpha\text{OPT}$, where α is the approximation ratio of a suitable facility location algorithm. If the solution opens k^* facilities, then the extra memory used is at most $O(k^*n^{2/\ell} \log^2 n \log(\max_i f_i))$.*

Note that the space complexity has a factor of $\log(\max_i f_i)$. We note that any algorithm that reads each facility cost must use at least $\log(\max_i f_i)$ bits of memory.

4.2.1 Overview

We first give a high-level overview of our iterative algorithm **Pass-Efficient FL**. At each iteration it takes a sample from the input array and computes a facility location solution on the sample. The algorithm identifies and removes from consideration those points of the *entire input array* that are serviced well by the solution on the sample, and iterates on points not serviced well. In subsequent iterations, the algorithm increases the rate at which points are sampled (i.e. elements in the i th iteration are picked with probability roughly $n^{(i+1)/\ell}/n$) until the last iteration, when we sample each point with probability 1.

More specifically, in Step 1 of iteration i , **Pass-Efficient FL** takes a sample S_i of the input array X_i . In Steps 2 and 3 it computes an approximate facility location solution on S_i that opens a set of facilities F_i . In Step 4, the algorithm tests to see if the number of facilities in F_i is small. If so, then each cluster of S_i on average must be large; intuitively this condition will be enough to show that the facilities F_i opened by clustering the sample S_i are in fact a good solution for a very large

subset, $R_i \subset X_i$, of the points from X_i . In the next iteration we can recurse on the much smaller set of points $X_{i+1} = X_i \setminus R_i$ for which F_i is a poor solution. In this case, the number of samples that we take (and hence the space complexity of our algorithm) will not change from this iteration to the next: although the next iteration will sample at a higher frequency, this will be balanced by its having many fewer points in X_{i+1} to consider. If the number of clusters in F_i is large, then we cannot make any guarantees about how well F_i will cluster X_i , and we must recurse on almost the entire input array. In this case, the number of samples taken in the next iteration will increase by a factor of $n^{1/\ell}$.

It is worth noting that our algorithm uses ideas from previously designed sampling based algorithms for k -median presented in Section 4.1, but its adaptation to multiple passes and to facility location contains many new insights into these problems. Informally, facility location is often considered an “easier” problem than k -median; the number of facilities is not fixed and thus an approximation algorithm has more flexibility in finding a low-cost solution. However, this flexibility enjoyed by algorithms in the conventional model of computation is the root of the difficulties faced by small-space pass-efficient algorithms: One must know how many potential facilities to store in memory prior to calculating the solution. For k -median, this is easy since we know that the solution must contain exactly k medians, but for facility location a solution may use anywhere from 1 to n facilities. Thus, central to any small-space, pass-efficient algorithm for facility location is an accurate estimate of the number of facilities required. Storing in memory many fewer than the optimum number would lead to a solution with a poor approximation ratio, while storing many more than the requisite number will result in a potentially large waste of storage. Furthermore, estimating the number of facilities required is a non-trivial problem that cannot be gleaned from a single sample from the input. Thus, in essence our

algorithm performs an iterative, small-space search for the correct number of facilities while simultaneously finding a good solution. This feature is the core of the algorithm that makes it possible to solve facility location when the naive sampling that is adequate for k -median will fail.

The algorithm uses a black box facility location algorithm with approximation ratio α that requires only a linear amount of space (for example, [15]). We require that the black box solve a slightly stronger facility location problem with *non-uniform demands*. Each point in X is assigned an integer demand e_i . The objective is to find a set of facilities F that minimizes $\sum_{x_i \in F} f_i + \sum_{x_i \in X} e_i \cdot d(x_i, F)$.

The algorithm also calls a subroutine **One – PassSelect**(Y, k), where Y is a datastream of numbers and k an integer. **One – PassSelect**(Y, k) will return an element of Y between the $|Y| - 2k$ and $|Y| - k$ largest elements of Y . Since we will require that $k = n^{(\ell-1)/\ell}$, it will require too much memory to store the k largest elements of Y . Thus, **One – PassSelect**(Y, k) is a sampling based algorithm that will use at most $O((|Y|/k) \log n)$ bits of memory to solve the problem. We discuss this algorithm in Section 4.2.3.

4.2.2 The Algorithm and proof of correctness

A technical issue that we must address is how the metric space is presented in the input array. For example, distances can be given implicitly by a function of the nodes (for instance, Euclidean or Hamming distance). A general approach would be that each point is represented by an $(n - 1)$ -dimensional vector giving its distance to every other point in X . We will first assume that distances are given by a function, and then later show how to relax this assumption and achieve the same performance guarantees for the more general case.

As a preprocessing step to the algorithm, we round the facility cost of each

node down to the nearest power of 2 and solve this modified instance, as in [55]. By rounding each facility cost, we have ensured that there are at most $\log \max_i f_i$ distinct values of facility costs; if we run an algorithm with approximation ratio β with these facility costs, the resulting solution will have an approximation ratio of at most 2β for the original facility costs.

Algorithm **Pass-Efficient FL** is given in Figure 4.1. In order to assist the reader in managing the proliferation of variable names in this section, we provide the following table:

name	meaning
X	original input
d	distance metric on X
n	$ X $
ℓ	total number of iterations
X_p	points still to be clustered after p iterations
n_p	$ X_p $
S_p	sample drawn in p th iteration
s_p	$ S_p $
F_p	set of facilities found in p th iteration
R_p	set of points in X_p “clustered well” by F_p
F'_p	set of <i>potential</i> facilities found in Step 2
F_{approx}	final set of facilities
k^*	$ F_{\text{approx}} $
c	the constant from Lemma 36
α	approximation ratio of black box algorithm
d'	metric d with distances scaled up by n_p/s_p
F^*	set of optimum facilities

Input: Metric space (X, d) and facility costs $\{f_i\}$.
Initialize: $X_1 \leftarrow X$, $n_1 \leftarrow n$, $F_{\text{approx}} \leftarrow \emptyset$.
Main Loop: For $p = 1, \dots, \ell - 1$, do:

1. In one pass, draw a sample S_p from X_p uniformly at random with replacement of size
$$s_p = c \frac{n^{(p+1)/\ell}}{n} \cdot n_p \log n,$$
where c is the constant from Lemma 36. If $p = \ell - 1$, set $S_{\ell-1} \leftarrow X_{\ell-1}$.
2. In a second pass over X , for each node $x \in S_p$ and each distinct facility cost f , store the facility closest to x with cost f . Call this set of nodes F'_p ; note that $|F'_p| \leq s_p \log \max_i f_i$.
3. Construct the following instance of facility location on a subset of X : metric space $(S_p \cup F'_p, d')$ where $d'(x, y) = (n_p/s_p) \cdot d(x, y)$, with facility costs the same. Let the demand for each point of S_p be 1, and the demand of each point in $F'_p \setminus S_p$ be 0.

Solve the instance using a black box algorithm for facility location with approximation ratio α . Let F_p be the set of facilities.
4. (a) If $|F_p| < s_p/(cn^{1/\ell} \log n)$ and $p \neq \ell - 1$, run algorithm **One-Pass Select** to find an element $e \in X_p$ such that $d(e, F_p)$ is between the $n_p - n_p \cdot n^{-1/\ell}$ and $n_p - \frac{1}{2}n_p \cdot n^{-1/\ell}$ elements of X_p with largest service cost. Set $R_p \leftarrow \{x : x \in X_p, d(x, F_p) \leq d(e, F_p)\}$.
(b) If $|F_p| \geq s_p/(cn^{1/\ell} \log n)$ and $p \neq \ell - 1$, set $R_p \leftarrow S_p$.
(c) If $p = \ell - 1$, set $R_{\ell-1} \leftarrow X_{\ell-1}$.
5. Set $X_{p+1} \leftarrow X_p \setminus R_p$, $n_{p+1} \leftarrow |X_{p+1}|$, $F_{\text{approx}} \leftarrow F_{\text{approx}} \cup F_p$, and $p \leftarrow p + 1$.

Output the facilities F_{approx} .

Figure 4.1: Algorithm **Pass-Efficient FL**

Remarks

1. In our proofs we will assume that the points in R_p are serviced by the facilities in F_p , even though there may be closer facilities found in other iterations.
2. At the $\ell - 1$ th iteration of the algorithm, we are at the special case where our “sample” contains all remaining points that have yet to be clustered.

3. Note that in Step 1, we assume that we know the value of n_p . We don't know the value exactly, but know it to within a factor of 2, which is enough to implement Step 1.

In order to prove Theorem 34, we first bound the cost of F_{approx} on X in Lemma 37, then bound the amount of memory used by **Pass-Efficient FL** in Lemma 38.

Definition 7 Let $\text{service}(S, F) = \sum_{x_i \in S} d(x_i, F)$ denote the service cost of servicing points in S with facilities F . Define $\text{cost}(S, F, \eta) = \sum_{x_i \in F} f_i + \eta \text{service}(S, F)$ to be the total cost of servicing the set of points S with the facilities F , with distance scaled by a factor of η .

In order to bound the cost of the final solution, $\text{cost}(X, F_{\text{approx}}, 1)$, we first present the following lemma, which is partially adapted from an argument from [42]:

Lemma 35 For fixed p , with probability at least $1 - 1/(10\ell n^2)$, $\text{cost}(S_p, F_p, n_p/s_p) \leq 9\alpha \text{OPT}$.

Proof: Fix an optimum solution on X_p that opens a set of facilities $F^* \subseteq X$, with $\text{cost}(X_p, F^*, 1) = \text{OPT}_p \leq \text{OPT}$. The cost of these facilities on the sample S_p with distances scaled by n_p/s_p as in Step 3 is $\text{cost}(S_p, F^*, n_p/s_p) = \sum_{x_i \in F^*} f_i + \frac{n_p}{s_p} \sum_{x_i \in S_p} d(x_i, F^*)$. By the Markov inequality, we know that with probability at most $1/3$,

$$\frac{n_p}{s_p} \sum_{x_i \in S_p} d(x_i, F^*) \geq 3 \sum_{x_i \in X} d(x_i, F^*).$$

Thus, with probability at least $2/3$, we have $\text{cost}(S_p, F^*, n_p/s_p) \leq \sum_{x_i \in F^*} f_i + 3 \sum_{x_i \in X} d(x_i, F^*) \leq 3\text{OPT}$.

Note that possibly F^* contains points that are not in $S_p \cup F'_p$. Thus, we construct a set of facilities $\tilde{F}^* \subset S_p \cup F'_p$. For each node $x_i \in F^*$, find the closest node in S_p ,

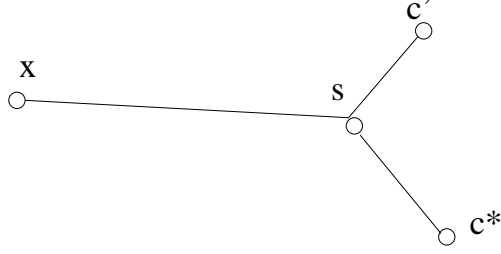


Figure 4.2: Consider the cost of servicing a point $x \in S_p$. c^* , with facility cost f , is the facility used by the optimum solution, $s \in S_p$ is the closest point in S_p to c^* , and $c' \in F'_p$ is the closest point to s_p with facility cost f . Then $d(x, c') \leq d(x, s) + d(s, c') \leq d(x, s) + d(s, c^*) \leq d(x, c^*) + 2d(s, c^*) \leq 3d(x, c^*)$.

and add the closest facility of cost $f_i \in F'_p$ to \tilde{F}^* . A standard argument will show that $\text{service}(S_p, \tilde{F}^*) \leq 3 \cdot \text{service}(S_p, F^*)$. See Figure 4.2. Thus, $\text{cost}(S_p, \tilde{F}^*, n_p/s_p) \leq 3\text{cost}(S_p, F^*, n_p/s_p) \leq 9\text{OPT}$. The black box algorithm will find a solution with cost at most $9\alpha\text{OPT}$.

We can boost the probability to $1 - 1/(10\ell n^2)$ by repeating Steps 1, 2 and 3 in parallel $O(\log n)$ times (note that one would never have the situation $\ell > n$), and choosing the solution with the smallest value of $\text{cost}(S_p, F, n_p/s_p)$. \square

We will also use the following Lemma that Charikar, O'Callaghan, and Panigrahy used for proving the correctness of a one pass algorithm for k -median with outliers:

Lemma 36 [16] *Let S be a sample of s points taken uniformly at random from X , where $s \geq ck \log n/\epsilon$, for some constant c . With probability at least $1 - \frac{1}{n^2}$, for all subsets $C \subseteq S$, $|C| = k$, we can service at least a $1 - \epsilon/2$ fraction of the points of X with centers at C with cost at most $2\frac{|X|}{|S|}\text{service}(S, C)$.*

Lemma 37 *With probability at least $1 - 1/n$, $\text{cost}(X, F_{\text{approx}}, 1) \leq 36\alpha(\ell - 1)\text{OPT}$.*

Proof: Since the R_p s form a partition of X , and $F_{\text{approx}} = \cup_p F_p$, we know $\text{cost}(X, F_{\text{approx}}, 1) \leq \sum_p \text{cost}(R_p, F_p, 1)$.

Claim: $\text{cost}(R_p, F_p, 1) \leq 18\alpha\text{OPT}$.

Proof of claim: Recall that S_p is a sample of size s_p from data stream X_p of size n_p . We have two cases. If in Step 4 of **Pass-Efficient FL**, we found that $|F_p| \geq s_p/(cn^{1/\ell} \log n)$, then the claim follows immediately from Lemma 35, since $n_p/s_p \geq 1$ and $R_p = S_p$.

Otherwise, in Step 4 we found that $|F_p| < s_p/(cn^{1/\ell} \log n)$. In this case, we can view F_p as a set of $k = s_p/(cn^{1/\ell} \log n)$ centers. Since S_p is a sample of X_p of size at least $ckn^{1/\ell} \log n$ points, it follows from Lemma 36, that at least a $1 - 1/(2n^{1/\ell})$ fraction of X can be serviced with cost at most $2(n_p/s_p) \cdot \text{service}(S_p, F_p)$. Since R_p is a set containing at most a $1 - 1/(2n^{1/\ell})$ fraction of the points of X_p closest to facilities in F_p , we have that $\text{service}(R_p, F_p, 1) \leq 2(n_p/s_p) \cdot \text{service}(S_p, F_p)$. Thus,

$$\text{cost}(R_p, F_p, 1) \leq 2 \left(\sum_{x_i \in F_p} f_i + \frac{n_p}{s_p} \text{service}(S_p, F) \right) \leq 18\alpha \text{OPT},$$

where the last inequality follows from Lemma 35. This proves the claim.

With the claim, we have:

$$\text{cost}(X, F_{\text{approx}}, 1) \leq \sum_{p=1}^{\ell-1} \text{cost}(R_p, F_p, 1) \leq 18\alpha(\ell - 1)\text{OPT}.$$

Recall that we rounded the facility cost down to the nearest power of 2. This will increase the cost of the solution by at most a factor of two. Thus, the approximation ratio of the algorithm is $36\alpha(\ell - 1)$. \square

Lemma 38 *The amount of memory used by **Pass-Efficient FL** is at most $O(k^* n^{2/\ell} \log^2 n \log(\max_i f_i))$, where k^* is the number of facilities output by the algorithm.*

Proof: By induction, we will prove that $s_p \leq c(\max_{i < p} |F_i|) \cdot n^{2/\ell} \log n$ for $p \geq 1$.

We will artificially set $|F_0| = 1$ (this is just for the base case analysis; there is no F_0 constructed by our algorithm). Since the final set of facilities output by the algorithm, F_{approx} , satisfies $|F_{\text{approx}}| \geq \max_{p \geq 0} |F_p|$, the lemma will follow (note the extra $(\log n \log(\max_i f))$ factor in the lemma, which is derived from the boosting in Lemma 35, and the fact that we must store the set F'_p from Step 2).

Assume the inductive hypothesis for all level $p \leq m$: $s_m \leq c(\max_{i < m} |F_i|) \cdot n^{2/\ell} \log n$.

If at Step 5 of the algorithm, we had found that $|F_m| < s_m/(cn^{1/\ell} \log n)$, then $n_{m+1} \leq n_m/n^{1/\ell}$. This implies that

$$s_{m+1} \leq c \frac{n^{(p+1)/\ell}}{n} \cdot n_{m+1} \log n \leq c \frac{n^{p/\ell}}{n} \cdot n_m \log n = s_m \leq c \left(\max_{i \leq m} |F_i| \right) \cdot n^{2/\ell} \log n.$$

If on the other hand at Step 5, we had found that $|F_m| \geq s_m/(cn^{1/\ell} \log n)$ then

$$s_{m+1} \leq n^{1/\ell} s_m \leq c |F_m| n^{2/\ell} \log n.$$

Note that the maximum memory requirement of the algorithm is just the size of the sample, since we assume that the black box algorithm will use only an amount of space that is linear in the size of the input and because the call to **One-Pass Select** will use at most $O(n^{1/\ell})$ bits of memory, by Lemma 39. \square

Combining Lemmas 37 and 38 proves Theorem 34.

Remark Thus far, we have assumed that distances are presented to the algorithm as an easily computed function of the input points, but this will not be possible for general metric spaces. As we mentioned before, a different approach would be that each input point is given by an $(n - 1)$ dimensional vector giving its distances to all other points. In this case, when we sample the points in Step 1, we do not want to

store the entire $(n - 1)$ dimensional vector in memory. Instead, in the first pass we would draw our sample and in the second pass we could then store an $(|S_p| - 1)$ -dimensional vector for each node that contains the distance between the node and the $|S_p|$ nodes in the sample. The algorithm may then proceed as stated. Note this would then require on the order of $(k^*)^2 n^{4/\ell}$ integers of working memory, but correspondingly X would contain n^2 integers.

4.2.3 Approximate selection in one pass

In this section we describe the one-pass subroutine **One-Pass Select** called by **Pass-Efficient FL**. The algorithm is given in Figure 4.3. Its input is a datastream of numbers Y and an integer k , and it outputs an element of Y that is between the $|Y| - 2k$ and $|Y| - k$ th largest elements of Y .

Of related interest is the paper of Munro and Paterson [59], who give a P pass algorithm for selecting the k th largest element of a datastream with n elements, using extra space at most $O(n^{1/P} \log^{2-2/P} n)$. Munro and Paterson prove a nearly matching lower bound on the space usage of deterministic P -pass selection algorithms. Our small-space algorithm **One-Pass Select** avoids making more than one pass by random sampling and guaranteeing only approximate selection.

Input: Datastream Y of m numbers, integer k .

1. In one pass, take a sample S of size $s = c'(m/k) \log(1/\delta)$ uniformly at random with replacement from Y for some constant $c' > 0$.
2. Sort S and output e , the $(s - \frac{3}{2} \frac{k}{m} s)$ th largest element of S .

Figure 4.3: Algorithm **One-Pass Select**

Lemma 39 One-Pass Select *will find an element between the $m - k$ and $m - 2k$ th largest elements of Y with probability at least $1 - \delta$ using at most $c'(m/k) \log(1/\delta)$ bits of extra memory, for appropriately chosen constant $c' > 0$.*

Proof: Let y_1, \dots, y_m be the elements of Y in ascending order, with ties broken arbitrarily. (We consider all elements of the array to be distinct, but some may take the same value.) Define $\text{index} : Y \rightarrow [m]$, to be the function that takes an element $z \in Y$ and maps it to the unique index such that $z = y_{\text{index}(z)}$.

We first prove an upper bound on $\Pr [\text{index}(e) \geq m - k]$. Let s_i be the i th element chosen uniformly at random for the sample. Let X_i be a random variable that is 1 if $\text{index}(s_i) \geq m - k$ and 0 otherwise. Note that $\mathbb{E}[X_i] = k/m$. If constant c' is chosen appropriately, an application of the Chernoff bound yields:

$$\Pr \left[\sum_i X_i \geq \left(1 + \frac{1}{2}\right) \frac{k}{m} s \right] \leq \frac{\delta}{2}.$$

Note that $\text{index}(e) \geq m - k$ implies that at least $\frac{3}{2}(k/m)s$ of the elements of S have indices that are greater than $m - k$; this corresponds to the event that $\sum X_i \geq \frac{3}{2} \frac{k}{m} s$. Thus, $\Pr [\text{index}(e) \geq m - k] \leq \delta/2$.

We next prove an upper bound on $\Pr [\text{index}(e) \leq m - 2k]$. Let Y_i be a random variable that is 1 if $\text{index}(s_i) \leq m - 2k$ and 0 otherwise. Since $\mathbb{E}[Y_i] = 2k/m$, an application of the Chernoff bound yields (again, for appropriately chosen c'):

$$\Pr \left[\sum_i Y_i \leq \left(1 - \frac{1}{10}\right) \frac{2k}{m} s \right] \leq \frac{\delta}{2}.$$

Note that $\text{index}(e) \leq m - 2k$ implies that $\sum Y_i \leq \frac{9}{10} \frac{2k}{m} s$. Thus, $\Pr [\text{index}(e) \leq m - 2k] \leq \delta/2$.

It follows that $\Pr [m - 2k \leq \text{index}(e) \leq m - k] \geq 1 - \delta$. \square

4.3 k -Facility Location

In Figure 4.4, we give algorithm k -FL, which is an adaptation of our facility location algorithm to solving the following hybrid problem between k -median and facility location:

k -Facility Location Problem: *Given a metric space (X, d) and a facility cost f_i for each point $x \in X$, find a set $F \subset X$, such that $|F| \leq k$, that minimizes the objective function: $\sum_{x_i \in F} f_i + \sum_{x_i \in X} d(x_i, F)$.*

Note that k -median is a special case of k -facility location where $f_i = 0$ for all i and that facility location is the special case where $k = n$.

Theorem 40 *There exists a 3ℓ pass algorithm with approximation ratio $O(\ell)$ for k -facility location using extra space at most $O(k^{(\ell-1)/\ell} n^{1/\ell} \log^2 n \log(\max_i f_i))$.*

The algorithm is simpler than **Pass-Efficient FL**, and can be considered a generalization of Indyk's algorithm that we outlined in Section 4.1 to multiple passes for k facility location.

Proof: If the black box k -median and facility location algorithms use only a linear amount of space, then it is clear that the above algorithm will use at most $\tilde{O}(k^{(\ell-1)/\ell} n^{1/\ell})$ extra space. Thus, in order to prove Theorem 40, we just need to bound $\text{cost}(X, F_{\text{approx}}, 1)$. We first bound the cost of the set of facilities F prior to running the k -median approximation algorithm as we did in Section 4.2:

$$\text{cost}(X, F, 1) \leq \sum_p \text{cost}(R_p, F_p, 1). \quad (4.1)$$

Claim: $\text{cost}(R_p, F_p, 1) \leq O(\alpha)\text{OPT}$.

Input: Metric space (X, d) , facility costs $\{f_i\}$, integer k .

Initialize: $X_1 \leftarrow X$, $p \leftarrow 1$, $F \leftarrow \emptyset$.

Main Loop: For $p = 1, \dots, \ell$, do:

1. In a single pass, draw a sample S_p of size $s = ck^{(\ell-1)/\ell}n^{1/\ell} \log n$ from the datastream X_p . For $p = \ell$, just set $S_\ell \leftarrow X_\ell$.
2. In a second pass, run a black box approximation algorithm for facility location on S_p taking into consideration all facilities in X exactly as in **Pass-Efficient FL**. Call the output F_p .
3. Call **One-Pass Select** to find a point $e \in X_p$ such that e is within the $k^{p/\ell}n^{1-p/\ell}$ and $(1/2)k^{p/\ell}n^{1-p/\ell}$ points of X_p that are farthest away from F_p . Set $R_p \leftarrow \{x : x \in X_p, d(x, F_p) < d(e, F_p)\}$.
4. Set $X_{p+1} \leftarrow X_p \setminus R_p$, $F \leftarrow F \cup F_p$, and $p \leftarrow p + 1$.

Run a k -median algorithm on the set F , with each $x_i \in F$ weighted by the number of points in X that it services. Output the resulting set of k facilities as F_{approx} .

Figure 4.4: Algorithm k -FL

Proof of claim: Since Steps 1 and 2 of k -FL are identical to Steps 1, 2, and 3 in **Pass-Efficient FL**, we can apply Lemma 35 to deduce that $\text{cost}(S_p, F_p, n_p/s) \leq 9\alpha\text{OPT}$.

Fix an optimum k -facility location solution on X_p with facilities $C = \{c_1, \dots, c_k\}$. Let C_i be the set of points in X_p that are serviced by c_i and let the set of indices corresponding to large clusters in C be $I = \{i : |C_i| \geq \frac{1}{2}k^{-(\ell-p)/\ell}n^{1-p/\ell}\}$. Note that $|X_p| \leq k^{(p-1)/\ell}n^{1-(p-1)/\ell}$. Since we are drawing a sample of size $s = ck^{(\ell-1)/\ell}n^{1/\ell} \log n$

points from X_p , we will have $|X_p|/s < (1/c)k^{-(\ell-p)/\ell}n^{1-p/\ell} \log n$; the sample S_p will contain many points from C_i for $i \in I$ with high probability. Intuitively if we cluster S_p , these large C_i s will be serviced well. Indeed, we can prove with an argument that is exactly the same as the argument in [42] (but with the addition of facility costs) that if constant c is chosen appropriately,

$$\text{cost}(\cup_{i \in I} C_i, F_p, 1) \leq O(\alpha)\text{OPT}.$$

Note that the number of points of X_p that do not fall in $\cup_{i \in I} C_i$ is at most $k \cdot \frac{1}{2}k^{-(\ell-p)/\ell}n^{1-p/\ell} = \frac{1}{2}k^{p/\ell}n^{1-p/\ell}$. Since the set R_p excludes at least the $\frac{1}{2}k^{p/\ell}n^{1-p/\ell}$ points of X_p that are farthest away from facilities in F_p , it follows that $\text{cost}(R_p, F_p, 1) \leq \text{cost}(\cup_{i \in I} C_i, F_p, 1) \leq O(1)\text{OPT}$. This proves the claim.

The claim, along with Equation (4.1), implies that $\text{cost}(X, F, 1) \leq O(\ell)\text{OPT}$. However, F may not be a feasible solution because it may contain more than k facilities. Thus, the final step in the algorithm is to run a constant factor approximation algorithm for k -median on F , with each point in F weighted by the number of points in X that it services. The result will be a subset of k facilities. Reasoning similar to the proof of Theorem 2.3 from [36] will prove that $\text{service}(X, F_{\text{approx}}) = O(\ell)\text{OPT}$. Lastly, since $F_{\text{approx}} \subseteq F$, it follows that $\sum_{x_i \in F_{\text{approx}}} f_i \leq \sum_{x_i \in F} f_i \leq O(\ell)\text{OPT}$.

Thus, $\text{cost}(X, F_{\text{approx}}, 1) = O(\ell)\text{OPT}$. \square

4.4 Lower bounds for multiple pass clustering

Using the communication complexity of the set disjointness problem, we can prove the following lower bound:

Theorem 41 *Any ℓ pass randomized algorithm that, with probability at least $2/3$,*

computes an $O(n^m)$ approximation to the cost of the optimum solution to facility location, where $m > 0$ is any constant, must use at least $\Omega(n/\ell)$ bits of memory.

Proof: We show there is a streaming compatible reduction from the disjointness problem to the facility location problem.

Suppose Alice and Bob have private subsets $A, B \subseteq [n]$, respectively, and want to communicate to determine whether $|A \cap B| \geq 1$ or $|A \cap B| = 0$. Let $a, b \in \{0, 1\}^n$ be the characteristic vectors of the sets A and B (i.e. $a_i = 1$ iff $i \in A$, $b_i = 1$ iff $i \in B$).

Given sets A and B , construct the following instance of facility location with $2n$ nodes x_i^a, x_i^b for $i = 1, \dots, n$: $d(x_i^a, x_i^b) = 1$ for all i , and all other distances are arbitrarily large. If $a_i = 0$, set $f_{x_i^a} = 1$. If $a_i = 1$, set $f_{x_i^a} = n^{m+1}$. Similarly, If $b_i = 0$, set $f_{x_i^b} = 1$. If $b_i = 1$, set $f_{x_i^b} = n^{m+1}$. It follows that if A and B are disjoint, then the instance has cost $2n$. Otherwise, the instance has cost at least n^m .

We then set the functions $\phi_A(A)$ to represent the set of points x_i^a and integers $f_x^{a_i}$ for $i = 1, \dots, n$, and $\phi_B(B)$ to represent the set of points x_i^b and the integers $f_x^{b_i}$. Alice and Bob can thus simulate a $2\ell - 1$ round protocol for the disjointness problem by simulating an ℓ pass facility location algorithm on the instance $\phi_A(A) \circ \phi_B(B)$ (see the proof of Theorem 5.)

Since the randomized communication complexity of the disjointness problem is known to be $R(\text{DISJ}) = \Theta(n)$, it follows that any pass-efficient algorithm that computes facility location will use at least $\Omega(n/(2\ell - 1))$ bits of memory by applying Theorem 5. \square

The lower bound proves that it is not possible to design a pass-efficient algorithm that uses less than $o(n)$ space. Thus, our small-space bound on the amount of memory required must necessarily be parameterized by k^* .

4.5 Remaining Questions

Two major questions about the complexity of pass efficient algorithms for facility location remain after this study. The space usage of **Pass-Efficient FL** is given by $\tilde{O}(k^*n^{2/\ell})$. Can the algorithm be adapted in order to change the dependence from k^* to k_{OPT} , where k_{OPT} is the number of facilities opened by an optimum solution. (Since k_{OPT} need not be smaller than k^* , this is not necessarily better in terms of resource usage, but it is in some sense more elegant.) The proof of our lower bound on the space required by pass-efficient algorithms for facility location used instances of facility location that required $\omega(n)$ facilities for any approximation. Is it possible to prove lower bounds on instances that require only k facilities in the optimum solution but that require $\omega(k)$ space to solve in a small number of passes? Would it then be possible to verify that the tradeoff between passes and memory required is tight?

Chapter 5

Graph Partitioning Algorithms

In this chapter, we consider approximation algorithms in the usual Turing model of computation for the clustering of an arbitrary, undirected graph. This is a new and interesting **NP**-hard variant of classical graph partitioning problems that we call the *Sum-of-squares Partition Problem*.

Sum-of-Squares Partition Problem: *Given an undirected graph $G = (V, E)$ and an integer m , remove a set $F \subseteq V$ of at most m nodes in order to partition the graph into disconnected components H_1, \dots, H_l , such that $\sum_i |H_i|^2$ is minimized.*

The *edge cut version* of the sum-of-squares-partition problem is similar, but asks for the removal of m edges, rather than nodes, to disconnect the graph.

We call an algorithm for the sum-of-squares partition problem an (α, β) -bicriterion approximation algorithm, for $\alpha, \beta \geq 1$, if it outputs a node cut consisting of at most αm nodes that partitions the graph into connected components $\{H_i\}$ such that $\sum |H_i|^2 \leq \beta \cdot \text{OPT}$, where OPT is the objective function value of the optimum solution that removes at most m nodes.

In Section 5.2, we present an algorithm for this problem and in Section 5.3 we

prove complementary hardness of approximation results. Our main algorithmic result is:

Theorem 42 *There exists a polynomial time $(O(\log^{1.5} n), O(1))$ -bicriterion approximation algorithm for the sum-of-squares partition problem.*

5.1 Motivation: A network security problem

This combinatorial optimization problem arose in the context of a study of a model of virus propagation and network security in collaboration with James Aspnes and Aleksandr Yampolskiy. We give a brief summary of the main results of this study and the role of the sum-of-squares partition problem. We refer the reader to the journal version of the paper for more details [7].

The Model

The network topology is modeled as an undirected graph $G = (V, E)$, with V representing the set of hosts and E representing the links between them. Each host is faced with the decision to either purchase and install anti-virus software at a fixed cost C , or remain vulnerable and risk the possibility of infection, with a potential cost L . After the hosts have made their decisions, the virus initially attempts to infect a single, randomly chosen node; call it v_0 . If the node is vulnerable, then the virus will infect the node v_0 and eventually spread to all of its vulnerable neighbors. The virus continues spreading in this fashion; ultimately, a vulnerable node v in the graph will become infected if there exists a path from v_0 to v in G that is not blocked by a secure node.

Consider the following, equivalent interpretation of the virus propagation model. Suppose that the nodes in subset $R \subseteq V$ choose to purchase the anti-virus software

at a cost C . Removing the subset R from V will partition the graph G into disjoint, connected components H_1, \dots, H_k of vulnerable nodes. Initially, the virus attempts to infect a randomly chosen node v_0 . If v_0 is vulnerable, then for the single H_i such that $v_0 \in H_i$, all nodes in H_i will be infected, while all other nodes will remain unaffected.

In our model, the *individual cost* to each host is defined as its expected cost, which is C if the host installs anti-virus software, and $L \cdot (\text{Probability of infection})$ otherwise. The *social cost* to the entire system is defined as the sum of the individual costs. Using the symbols from the previous paragraph, the cost of the entire system can easily be shown to be

$$C \cdot |R| + \frac{L}{n} \sum_i |H_i|^2. \quad (5.1)$$

We will refer to the social cost of the optimum choice of R as the *social optimum*.

A Game Theoretic Approach

A natural question that arises from this model is: *How should hosts decide whether or not to purchase anti-virus software?* If hosts are free to make their own decisions, a standard game-theoretic approach to this problem is to assume that they are rational and selfish. With these assumptions, a host will install anti-virus software if the cost C is lower than the expected cost of infection.

A configuration is in *Nash Equilibrium* if no vulnerable host would otherwise want to install anti-virus software, and if no secure host would rather be vulnerable than pay C . In order to study this type of selfish behavior, we analyzed the social cost of the worst-case configuration in Nash Equilibrium. The ratio of the cost of the worst-case Nash Equilibrium to the social optimum is often called the *price of anarchy*, because it measures the degree to which selfish behavior will hurt the system as a

whole. In some network topologies the price of anarchy is very high, $\Theta(n)$, which implies that selfish behavior can be grossly suboptimum in terms of social cost. In particular, the optimum solution on the star graph topology with $C = Ln/(n - 1)$ will have the property that the optimum solution is to install anti-virus software on the center node, which will achieve a social cost of $C + L(n - 1)/n$. It can be shown that installing anti-virus software on any single node will be in Nash Equilibrium; if this node is not the center node, then the social cost will be $C + L(n - 1)^2/n$. The price of anarchy will then be:

$$\frac{C + L(n - 1)^2/n}{C + L(n - 1)/n} = \frac{L(n - 1)}{2L(n - 1)/n} = n/2.$$

A Centralized Solution

In essence, selfish behavior results in a locally optimized solution, which we proved can be a factor of $\Theta(n)$ away from the social optimum. As an alternative to anarchy, the hosts may instead relinquish their autonomy and have their decision imposed upon them by a globally optimized solution. It can be shown that finding an optimum central solution is **NP**-hard. However, the bicriterion approximation algorithm for sum-of-squares partition provides an $O(\log^{1.5} n)$ approximation to the social optimum, which is much more desirable than the $\Theta(n)$ approximation derived from the worst-case Nash Equilibrium. Recall Equation (5.1), the expression for the social cost of installing anti-virus software on a set R . In order to approximately optimize this social cost function, we can run our algorithm for sum-of-squares partition for all choices of m , the number of nodes to remove, and pick the best solution in terms of social cost.

5.2 Proof of Theorem 42

In this section, we will describe the algorithm **PartitionGraph** that will achieve a $(O(\log^{1.5} n), O(1))$ bicriterion approximation ratio for the sum-of-squares partition algorithm. We will work with black box approximation algorithms for the fundamental problem of finding sparse cuts in graphs. Before presenting our main algorithm, we will review parts of the sparse cut literature and will show the connection between finding sparse cuts and approximating the sum-of-squares partition problem.

5.2.1 Preliminaries: Sparse Cuts

Let $G = (V, E)$ be an undirected graph. We denote the *edge cut* defined by a vertex set $S \subseteq V$ and \bar{S} by $E(S, \bar{S})$, which consists of the edges between the vertex set S and \bar{S} . If $G = (V, E)$ is a directed graph, the cut between S and \bar{S} is the set of edges from S directed to \bar{S} , which we also denote by $E(S, \bar{S})$.

Definition 8 (sparsity: *edge cut version*) *Let $G = (V, E)$ be an undirected or directed graph. The sparsity of cut $E(S, \bar{S})$ is given by*

$$\frac{|S| \cdot |\bar{S}|}{|E(S, \bar{S})|}. \quad (5.2)$$

We can also consider removing nodes to disconnect the graph.

Definition 9 (sparsity: *node cut version*) *Let $G = (V, E)$ be an undirected graph. The sparsity of node cut R that partitions the residual vertices $V \setminus R$ into disconnected components V_1 and V_2 is given by*

$$\frac{\left(|V_1| + \frac{|R|}{2}\right) \left(|V_2| + \frac{|R|}{2}\right)}{|R|}. \quad (5.3)$$

Important Note: In the literature, sparsity is usually defined as the inverse of expression (5.2), and finding the sparsest cut is a minimization problem. We have presented it as a maximization problem, since this is more natural for our application.

Known algorithms for sparse edge cuts

Leighton and Rao [53] first posed the question of finding the sparsest cut in their study of multicommodity flows. They gave an $\Omega(1/\log n)$ approximation algorithm for the sparse edge cut problem (when sparsest cut is defined as a maximization problem). Later, Arora, Rao, and Vazirani [6] presented an algorithm with approximation ratio $\Omega(1/\sqrt{\log n})$; their algorithm embeds the graph into a metric over \mathbb{R}^d , calculated by a semidefinite programming relaxation of sparsest cut, and then computes a randomized rounding that relies on the geometric structure of the embedding. Agarwal, Charikar, Makarychev, and Makarychev [1] extended this algorithm to find sparse edge cuts in directed graphs. All three of these sparse cut algorithms extend to the case where the edges of the graph are weighted.

Finding a sparse node cut

Known sparse cut algorithms usually solve edge cut problems. For our purposes, cuts that involve removing nodes in order to disconnect the graph are more relevant. The algorithms mentioned above find edge cuts in graphs, and are not directly applicable to our problem. However, there is a well-known procedure for reducing a node cut problem on an undirected graph to an edge cut problem on a weighted, directed graph. For completeness, we outline the procedure. A similar discussion in [53] considers the related problem of finding a balanced node cut in a graph.

Fact 43 *If β^* is the optimum sparsity of any node cut in an undirected graph $G = (V, E)$, the Agarwal, Charikar, Makarychev, and Makarychev algorithm can be used*

to find a node cut with sparsity at least $\beta^*/c\sqrt{\log n}$, for some constant c .

Proof: We now state a well-known procedure for reducing a node cut problem in an undirected graph to an edge cut problem in a weighted, directed graph.

The reduction is as follows: form directed graph G^* with vertex set $V^* = \{v|v \in V\} \cup \{v'|v \in V\}$ and edge set $E^* = \{(v, v')|v \in V\} \cup \{(v', v)|v \in V\} \cup \{(v', u)|(u, v) \in E\} \cup \{(u', v)|(u, v) \in E\}$. The weight of the $\{(v, v')|v \in V\}$ edges are 1, and weight of all other edges is infinity.

Suppose we have a node cut in G that removes a set R to partition $V \setminus R$ into components V_1 and V_2 ; this cut has node cut sparsity

$$\frac{(|V_1| + |R|/2) + (|V_2| + |R|/2)}{|R|}.$$

A corresponding directed edge cut in G^* is the cut (S^*, \bar{S}^*) , where $S^* = \{v'|v \in V_1\} \cup \{v|v \in V_1\} \cup \{v|v \in R\}$ and $\bar{S}^* = \{v'|v \in V_2\} \cup \{v|v \in V_2\} \cup \{v'|v \in R\}$. The edges crossing the cut are precisely $\{(v, v')|v \in R\}$. Thus, the edge sparsity of the cut in G^* is

$$\frac{(2|V_1| + |R|) \cdot (2|V_2| + |R|)}{|R|}.$$

The sparsities of the node and edge cuts are the same, up to the scaling factor of 4.

Suppose we have a directed edge cut in G^* defined by (S^*, \bar{S}^*) with non-zero sparsity. Only edges of the form (v, v') will cross the cut, since all other edges have weight infinity. In a similar manner as above, it can be shown that the set of vertices $\{v|(v, v') \text{ crosses the cut}\}$ corresponds to a set of removed vertices in G that will define a node cut with the exact same sparsity as the cut in G^* (again up to the scaling factor of 4). \square

Connections between sparse cuts and sum-of-squares partitioning

Our algorithm for solving the sum-of-squares partition problem will take a general approach that is similar to the greedy log n -approximation algorithm for set cover. A high-level description is that we repeatedly remove the node cut that gives us the best per-removed-node-benefit, quantified as its *cost-effectiveness*.

Suppose we have a connected subgraph H with k nodes. If node cut R creates disconnected components with node sets V_1 and V_2 , this cut has decreased the objective function value (\sum size of component²) by $k^2 - |V_1|^2 - |V_2|^2$. We thus define the *cost-effectiveness* of node cut R by $(k^2 - |V_1|^2 - |V_2|^2)/|R|$. The cost-effectiveness of R is equal to

$$\begin{aligned} \frac{k^2 - |V_1|^2 - |V_2|^2}{|R|} &= \frac{(|V_1| + |V_2| + |R|)^2 - |V_1|^2 - |V_2|^2}{|R|} \\ &= \frac{|R|^2 + 2|V_1||V_2| + 2|R|(|V_1| + |V_2|)}{|R|} \\ &= \frac{2|V_1||V_2|}{|R|} + |R| + 2(k - |R|) \\ &= \frac{2|V_1||V_2|}{|R|} + 2k - |R|. \end{aligned}$$

We then have the following relationship between finding sparse cuts and cost-effective cuts.

Lemma 44 *Let H be a graph with k nodes. If α^* is the maximum cost-effectiveness of all node cuts of H , the Arora-Rao-Vazirani sparse cut algorithm will find a cut with cost-effectiveness at least $\alpha^*/(c\sqrt{\log k})$, for some constant c .*

Proof: Consider a node cut that removes node set R and partitions the remaining nodes of H into disconnected components with node sets V_1 and V_2 . We can manipulate the expression for the cut's sparsity in a similar manner to our algebraic

manipulation of the cost-effectiveness to determine that

$$\frac{\left(|V_1| + \frac{|R|}{2}\right) \left(|V_2| + \frac{|R|}{2}\right)}{|R|} = \frac{|V_1||V_2|}{|R|} + \frac{k}{2} - \frac{|R|}{4};$$

We then have the following relations between the cost-effectiveness of a cut, α , and its sparsity, β .

$$\beta = \frac{|V_1||V_2|}{|R|} + \frac{k}{2} - \frac{|R|}{4} = \frac{\alpha}{2} - \frac{k}{2} + \frac{|R|}{4} \geq \frac{\alpha}{4}.$$

and

$$\alpha > 2\beta.$$

Thus, we know there exists a node cut with sparsity at least $\alpha^*/4$ (i.e. the cut with the highest cost-effectiveness). The sparse cut algorithm on H will find a node cut with sparsity at least $\alpha^*/(c\sqrt{\log k})$, for some constant c . This node cut will have cost-effectiveness at least $2\alpha^*/(c\sqrt{\log k})$. \square

5.2.2 The Algorithm

We are now ready to describe our algorithm **PartitionGraph** (see Figure 5.1), which is a $(O(\log^{1.5} n), O(1))$ bicriterion approximation algorithm for the sum-of-squares partition problem.

We now give some lemmas that characterize the behavior of the **PartitionGraph** algorithm.

Lemma 45 *PartitionGraph* outputs a node cut with at most $O(\log^{1.5} n)m$ removed nodes.

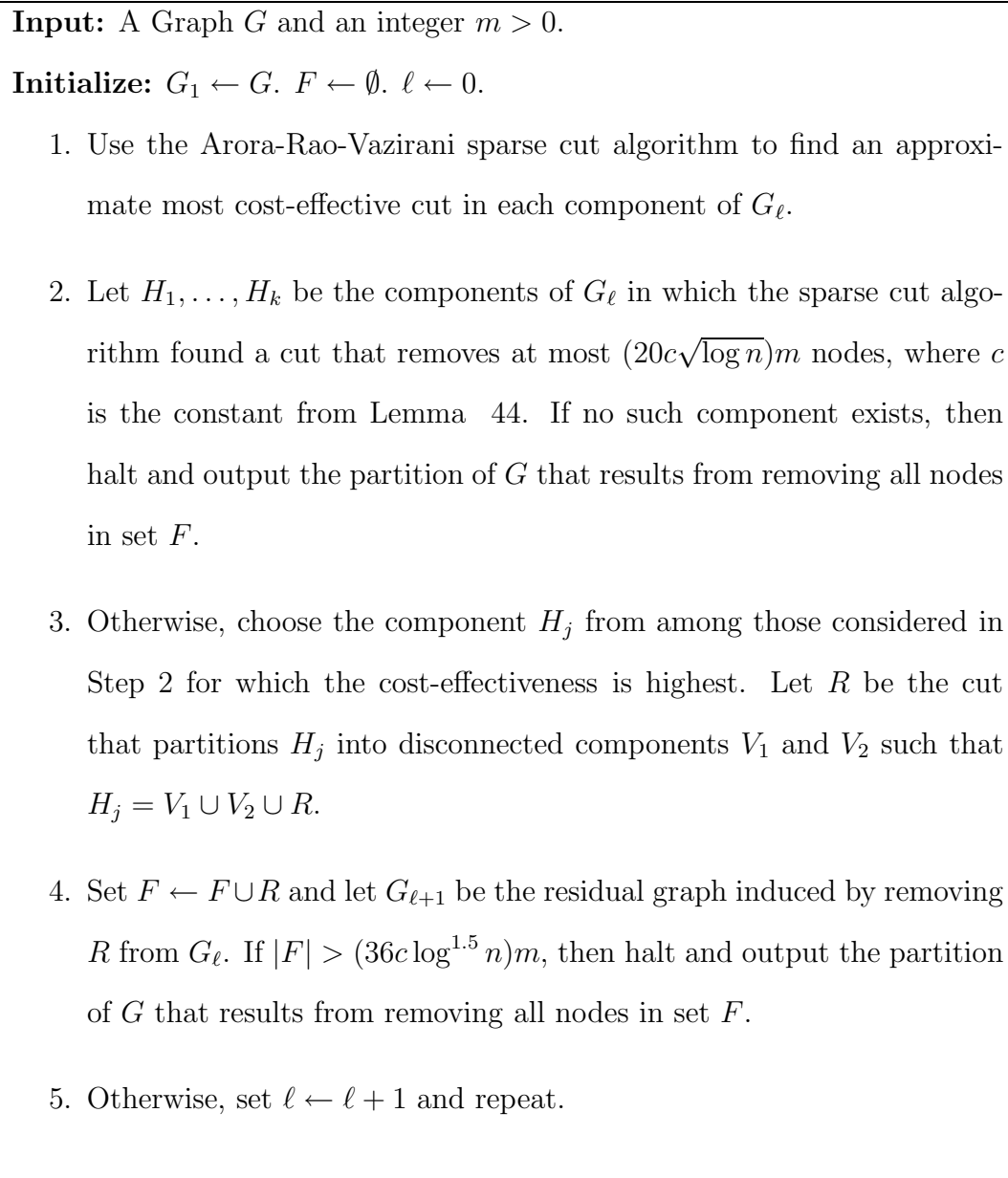


Figure 5.1: Algorithm **PartitionGraph**

Proof: Since the algorithm halts as soon as we augment the set of marked nodes such that $|F| > (36c \log^{1.5} n)m$, we know that at the beginning of each iteration, F contains at most $(36c \log^{1.5} n)m$ marked nodes. Since we add at most $(20c\sqrt{\log n})m$ marked nodes in the final iteration, the total number of marked nodes is at most $O(\log^{1.5} n)m$. \square

Fix an optimum solution for the sum-of-squares partition problem with objective function value OPT and let F^* be the optimum set of m removed nodes. In the next few proofs, we will denote the number of nodes in graph G by $|G|$. We will also denote an “intersection” of a graph G and a node set U by $G \cap U$, which is the set of nodes that G and U share.

Lemma 46 *Suppose after a number of iterations, the graph G_ℓ consists of k connected components H_1, \dots, H_k , and let $S = \sum |H_i|^2$.*

Either $S \leq 72 \cdot \text{OPT}$ or there exists a component H_i such that the Arora-Rao-Vazirani algorithm will find a node cut in H_i with at most $(20c\sqrt{\log n})m$ removed nodes and cost-effectiveness at least $S/(18cm\sqrt{\log n})$ (or possibly both).

Proof: Assume that $S > 72 \cdot \text{OPT}$. Note that the node cut defined by the set $F^* \cap G_\ell$ divides G_ℓ into a graph with objective function value at most OPT . This node cut thus induces a cost decrease of at least $S - S/72 > S/2$.

Define $F_i^* = F^* \cap H_i$ and $m_i = |F_i^*|$. Also, let the subgraph induced by removing vertices in $F_i^* \cap H_i$ from H_i be composed of connected components H_i^j for $j = 1, \dots, r_i$ (i.e, the optimum set of marked nodes partitions H_i into these components). Note that $\sum_i \sum_j |H_i^j|^2 \leq \text{OPT}$.

Since the total reduction in our objective function value from removing $\cup_i F_i^*$ from G_ℓ is at least $S/2$ due to our assumption that $S > 72 \cdot \text{OPT} > 2 \cdot \text{OPT}$, we have:

$$\sum_i \left(|H_i|^2 - \sum_j |H_i^j|^2 \right) \geq \frac{S}{2}, \quad (5.4)$$

because the outer summand on the left hand side of the inequality is the amount the objective function is reduced in each component.

Let I be the set of indices i for which

$$\frac{\left(|H_i|^2 - \sum_{j=1}^{r_i} |H_i^j|^2\right)}{m_i} \geq \frac{S}{4m} \quad (5.5)$$

(i.e. the per-node-benefit is at least $S/(4m)$). We have two cases. We show that the first case is consistent with the statement of the lemma, whereas the second case is impossible.

1. There exists an $i \in I$ such that for all $j = 1, \dots, r_i$, $|H_i^j| \leq 1/3|H_i|$.

First consider the subcase where $m_i \geq |H_i|/50$. Removing all of the at most $50m_i$ nodes in H_i will give us a trivial node cut with cost-effectiveness at least

$$\frac{|H_i|^2}{50m_i} \geq \frac{S}{200m} > \frac{S}{18cm\sqrt{\log n}}$$

for sufficiently large n . The first inequality follows from the fact that H_i and m_i satisfy Inequality (5.5). The second inequality follows from the existence of $\sqrt{\log n}$ in the denominator of the last term, which will dominate the constant terms when n is sufficiently large.

Now consider the subcase where $m_i < |H_i|/50$. We know that removing the m_i nodes in the set F_i^* partitions H_i into disconnected components $H_i^1, \dots, H_i^{r_i}$, such that $|H_i^j| \leq 1/3|H_i|$ for all j .

Claim: $\{H_i^j\}_j$ can be partitioned into two sets of indices J_1 and J_2 such that $\sum_{j \in J_1} |H_i^j| \geq (1/3 - 1/50)|H_i|$ and $\sum_{j \in J_2} |H_i^j| \geq (1/3 - 1/50)|H_i|$.

Proof of claim: Construct J_1 and J_2 by successively considering each H_i^j and adding it to the J_i that contains the fewest aggregate nodes in its components. Since each H_i^j has size at most $1/3|H_i|$, the construction guarantees that

$\left| \sum_{H_i^j \in J_1} |H_i^j| - \sum_{H_i^j \in J_2} |H_i^j| \right| \leq 1/3|H_i|$. Since $\sum_j |H_i^j| \geq 49/50|H_i|$ (because we assumed that $m_i < |H_i|/50$), the claim follows.

The claim implies that the set F_i^* will define a node cut of H_i that creates two disconnected components, $V_1 = \cup_{j \in J_1} H_i^j$ and $V_2 = \cup_{j \in J_2} H_i^j$, such that $(1/3 - 1/50)|H_i| \leq |V_1|, |V_2|$ and $|V_1| + |V_2| \geq 49/50|H_i|$. These conditions imply that $|V_1||V_2| \geq |H_i|^2/9$. Thus, the cost-effectiveness of the cut will be

$$2 \frac{|V_1||V_2|}{|R|} + 2|H_i| - |R| \geq \frac{2|H_i|^2}{9m_i} \geq \frac{2 \left(|H_i|^2 - \sum_{j=1}^{r_i} |H_i^j|^2 \right)}{9m_i} \geq \frac{S}{18m},$$

where the last inequality follows from Inequality (5.5). Lemma 44 guarantees that the sparse cut algorithm will find a cut in H_i with cost-effectiveness at least $S/(18cm\sqrt{\log n})$. The node cut output by the algorithm cannot contain more than $20cm\sqrt{\log n}$ nodes. Such a node cut would have cost-effectiveness at most $S/(20cm\sqrt{\log n})$, since any cut in G_ℓ can decrease the objective function value by at most S , which is less than the guaranteed cost-effectiveness of $S/(18cm\sqrt{\log n})$.

2. For each $i \in I$, there exists a j^* such that $|H_i^{j^*}| > 1/3|H_i|$. Also, note that $\text{OPT} > \sum_{i \in I} |H_i^{j^*}|^2$. We prove, by contradiction, that this case cannot occur. Thus, assume the case does occur.

Claim: $\sum_{i \in I} \left(|H_i|^2 - \sum_j |H_i^j|^2 \right) \geq S/8$.

Proof of claim: Let \bar{I} be the set of intervals such that $\left(|H_i|^2 - \sum_{j=1}^{r_i} |H_i^j|^2 \right) / m_i \leq S/(4m)$. Recalling equation (5.4), we have

$$S/2 \leq \sum_{i \in I} \left(|H_i|^2 - \sum_j |H_i^j|^2 \right) + \sum_{i \in \bar{I}} \left(|H_i|^2 - \sum_j |H_i^j|^2 \right).$$

Also, we have

$$\begin{aligned}
\sum_{i \in \bar{I}} \left(|H_i|^2 - \sum_j |H_i^j|^2 \right) &= \sum_{i \in \bar{I}} m_i \frac{(|H_i|^2 - \sum_j |H_i^j|^2)}{m_i} \\
&\leq \sum_{i \in \bar{I}} m_i \frac{S}{4m} \\
&\leq \frac{S}{4m} \sum_{i \in \bar{I}} m_i \leq \frac{S}{4}.
\end{aligned}$$

Combining these two inequalities proves the claim.

We have the inequalities:

$$\text{OPT} > \sum_{i \in I} |H_i^{j^*}|^2 \geq \sum_{i \in I} \frac{1}{9} |H_i|^2 \geq \frac{1}{9} \cdot \frac{S}{8},$$

where we used our claim for the last inequality. Thus, $\text{OPT} \geq S/72$. This is a contradiction to the assumption we made at the first line of the proof.

□

We now present the proof of Theorem 42.

Let a_j be the number of connected components that comprise the graph G_j at the beginning of the j th iteration, and let those connected components be $H_1^j, \dots, H_{a_j}^j$. Let $S_j = \sum_{i=1}^{a_j} |H_i^j|^2$ be the value of the objective function at the beginning of the j th iteration; thus $S_0 \leq n^2$ is its initial value. Let l be the number of iterations the algorithm needs to terminate, and S_{l+1} be the objective function's final value.

We wish to show that after the algorithm terminates, we have reduced the objective function value to $S_{l+1} = O(1) \cdot \text{OPT}$. Let F be the final set of nodes removed from G . If the algorithm terminates at Step 2 of the l th iteration because the sparse cut algorithm only found node cuts with more than $(20c\sqrt{\log n})m$ removed nodes,

then from Lemma 46 we know that $S_{l+1} \leq 72 \cdot \text{OPT}$. Thus, we assume this does not occur, so that $S_{l+1} > 72 \cdot \text{OPT}$ (in order to apply the “either” part of Lemma 46 to all iterations).

In order to reason about the decrease in the objective function value after each iteration, we impute to each node in F a per-node-decrease in the objective function value, given by the cost-effectiveness of its node cut. We then show that the total imputed decrease will decrease the objective function by a factor of $O(1)/n^2$, from which the theorem will follow.

More formally, suppose the set of marked nodes is given by the sequence $F = \{f_1, \dots, f_k\}$, where the nodes are in the order in which they were removed from the graph: nodes removed at an earlier iteration occur earlier in the sequence. From Lemma 45, we know that $k = |F| = \Theta(\log^{1.5} n)m$.

Let b_j be the iteration in which f_j was removed. We impute to f_j the value $\delta_j =$ cost-effectiveness of cut removed in iteration b_j . From Lemma 46, we know that $\delta_j \geq S_{b_j}/(18cm\sqrt{\log n})$.

Set $T_0 = S_0$ and $T_i = T_{i-1} - \delta_i$ to be the value of the objective function after node f_i 's per-node-decrease contribution has been accounted for. Note $T_k = S_{l+1}$.

Claim: For all i , $T_i \leq T_{i-1} - T_{i-1}/(18cm\sqrt{\log n})$

Proof of claim: Proving the claim reduces to proving that $\delta_i = T_{i-1} - T_i \geq T_{i-1}/(18cm\sqrt{\log n})$. Fix an i . We have two cases.

1. $b_i = b_{i-1}$ (i.e. f_i and f_{i-1} were removed in the same iteration). Then $\delta_i \geq S_{b_i}/(18cm\sqrt{\log n})$, but $S_{b_i} > T_i$, since S_{b_i} is the objective function value at the beginning of iteration b_i , whereas T_i is the objective function value “during” iteration b_i .
2. $b_i = b_{i-1} + 1$ (i.e. f_i was removed in the iteration after f_{i-1} was removed).

Then $\delta_i \geq S_{b_i}/(18cm\sqrt{\log n}) = T_i/(18cm\sqrt{\log n})$, since in this case T_i is the objective function value at the start of iteration b_i .

This proves the claim.

We therefore have $T_k \leq T_0(1 - 1/(18cm\sqrt{\log n}))^k \leq n^2(1 - 1/(18cm\sqrt{\log n}))^k$. Since $k > 36cm \log^{1.5} n$, it follows that $S_{l+1} = T_k = O(1) \leq O(1) \cdot \text{OPT}$, concluding the proof of Theorem 42.

The algorithm given above can be adapted in a straightforward way to yield an algorithm for the edge cut version of the sum-of-squares partition problem (instead of taking sparse node cuts, take sparse edge cuts), from which an analog to Theorem 42 may be derived. The above analysis of the node cut algorithm is more complicated than the corresponding analysis of the edge cut algorithm, since node cuts modify the node set, causing many difficulties.

5.3 Hardness of Approximation

In this section, we prove that it is hard to achieve a bicriterion approximation of $(\alpha, 1)$, for some constant $\alpha > 1$, by reduction from vertex cover. Hastad [37] proved that it is **NP**-hard to approximate vertex cover to within a constant factor of $8/7 - \epsilon$, for any $\epsilon > 0$. Dinur and Safra [21] improved the lower bound to $10\sqrt{5} - 21 \geq 1.36$. We show that if we have a graph G with a vertex cover of size m , then a $(1.18 - \epsilon, 1)$ algorithm for the sum-of-squares partition problem can be used to find a vertex cover in G of size at most $(1.36 - 2\epsilon)m$.

Theorem 47 *It is **NP**-hard under Cook reduction to approximate the sum-of-squares partition problem to within a bicriterion factor $(1.18 - \epsilon, 1)$, for any $\epsilon > 0$.*

Proof: Suppose graph $G = (V, E)$, $|V| = n$, contains a vertex cover C consisting of m nodes. Removing the m nodes of C and their incident edges will remove all edges from the graph. This will partition the graph into $n - m$ disconnected components consisting of 1 node each.

If we consider C as a solution to the sum-of-squares partition problem for removing m nodes, the solution will have an objective function value of $n - m$. Thus, an $(\alpha, 1)$ approximation algorithm for sum-of-squares partition will remove a set $R \subseteq V$ of nodes, such that $|R| \leq \alpha m$, in order to achieve an objective function of at most $n - m$. Let $V' = V \setminus R$ be the remaining nodes, and $\{H_i\}$ be the connected components in the residual graph.

Let S be the nodes of V' that are contained in connected components of size greater than 1 in the residual graph. It follows that $R \cup S$ is a vertex cover of G . We seek to bound the cardinality of $R \cup S$.

We first observe that the number of nodes that are contained in connected components of size 1 is $|V' \setminus S| = n - |R| - |S|$. Using the fact that if $|H_i| \geq 2$, then $|H_i|^2 \geq 2|H_i|$, we note that

$$\begin{aligned}
 n - m &\geq \sum_i |H_i|^2 \\
 &\geq \sum_{i:|H_i|=1} |H_i|^2 + \sum_{i:|H_i|\geq 2} |H_i|^2 \\
 &\geq n - |R| - |S| + 2|S| \\
 &\geq n - |R| + |S|.
 \end{aligned}$$

This implies that $|S| \leq |R| - m \leq \alpha m - m$, which implies that $|R \cup S| \leq (2\alpha - 1)m$.

Thus, a $(1.18 - \epsilon, 1)$ -bicriterion approximation algorithm for sum-of-squares partition will find a vertex cover of size $(1.36 - 2\epsilon)m$ in G . If OPT_{vc} is the cardinality of the optimum vertex cover, then we can search for an approximately minimum vertex

cover by running the algorithm described above for all $m = 1, \dots, n$ and outputting the best vertex cover, which will have size at most $(1.36 - 2\epsilon) \cdot \text{OPT}_{\text{VC}}$. \square

As mentioned before, sum-of-squares partition is intimately related to the problems of sparsest cut, balanced cut, and ρ -separator, all with uniform demands (i.e. the nodes all have weight 1). There are no known hardness of approximation results for any of these problem; we speculate that techniques for proving hardness of approximation for both α and β would yield hardness of approximation for some of these fundamental cut problems, which have proved elusive thus far. We note that Chawla *et al.* [18] and Khot and Vishnoi [51] have proved super-constant hardness of approximation results for stronger versions of these problems, specifically sparsest cut and balanced cut with general demands, assuming the unique games conjecture of Khot [50]. At the time of the submission of this thesis, this is a stronger assumption than $\mathbf{P} \neq \mathbf{NP}$.

Chapter 6

Conclusion and Future Work

In this dissertation we considered clustering algorithms in traditional and massive data set models of computation. We presented algorithms in the pass-efficient model, in which the algorithm may only access the input by making a small number of sequential passes over the input array. In this model, we studied the tradeoff between passes and space, and found that, for natural massive data set problems, allowing a few more passes over the input array can dramatically improve the space complexity of the algorithms. Furthermore, our pass-efficient algorithms for clustering adhered to a general paradigm of adaptive sampling implemented in multiple passes: in one pass, we sampled the input and computed a solution on the sample using small space; in another pass we identified the small set of points that were not “clustered well” by this solution and recursively called the algorithm on the “outliers.” We demonstrated that this sampling paradigm is quite general by adapting it to two very different problem domains with seemingly few structural similarities: learning density functions of generative mixture models and combinatorial optimization over discrete metric spaces.

6.1 Summary of Results

In Chapter 3, we presented pass-efficient algorithms for *learning mixtures of distributions*. For any positive integer ℓ , 2ℓ pass algorithm **SmallRam** will learn a mixture of k -uniform distributions within error ϵ using space at most $\tilde{O}(k^3/\epsilon^{2/\ell})$. Thus, if the algorithm makes a few more passes over the data, the amount of memory required decreases sharply. We then slightly generalized the problem to the *generalized learning problem*; using the r round communication complexity of the GT problem, we proved that any ℓ pass randomized algorithm that solves the generalized learning problem requires at least $\Omega(1/\epsilon^{1/2\ell})$ bits of memory, and strengthened our algorithm **SmallRam** to prove a nearly matching upperbound of $\tilde{O}(1/\epsilon^{4/\ell})$ for the same problem. We adapted our algorithms for learning mixtures of uniform distributions in \mathbb{R} to the more general problems of learning mixtures of uniform distributions over axis aligned rectangles in \mathbb{R}^2 and learning mixtures of linear distributions in \mathbb{R} .

In Chapter 4, we presented pass-efficient algorithms for the *facility location* problem, which is one of the most well-studied clustering problems in combinatorial optimization. In 3ℓ passes, our algorithm computes an $O(\ell)$ approximation using at most $\tilde{O}(k^*n^{2/\ell})$ bits of memory, where k^* is the number of facilities opened by a near-optimum solution on X . The algorithm shows the same tradeoff of passes and memory as the algorithm for learning mixtures of distributions. We proved a lower bound of $\Omega(n/\ell)$ on the amount of memory needed by an ℓ pass algorithm for facility location using the communication complexity of the DISJ function.

In Chapter 5, we presented algorithms for clustering graphs in the traditional model of computation. The problem that we considered, the *sum-of-squares partition* problem, is motivated by a game theoretic model of network security [7]. Our algorithm for the sum-of-squares partition problem uses known algorithms for finding

sparse cuts to repeatedly remove sparse node cuts in order to disconnect the graph. We showed that after this recursive cut procedure removed a total of $\Theta(\log^{1.5} n)m$ nodes from the graph, the sum of squares of the sizes of the residual components would be at most $O(1)\text{OPT}$. We proved complementary hardness of approximation results for the sum-of-squares partition problem by reduction from vertex cover.

6.2 Massive Data Set Algorithms: Future Directions

This dissertation consists in large part of pass-efficient algorithms for clustering large data sets. There is much potential for the analysis of algorithmic problems for natural massive data set problems considered in the data mining and machine learning communities, from the perspective of theoretical computer science.

For example, a direct extension of this dissertation work would be to design pass-efficient algorithms for other types of distributions. In particular, learning mixtures of Gaussian distributions in high dimensions would be particularly compelling; algorithms used in practice tend to be heuristic in nature, while algorithms with theoretical guarantees are not suitable for large data sets. An interesting variant of this work would be to design algorithms that will provably find the *best fit* mixture of k distributions (uniform, linear, or Gaussian), while making no assumptions about the distribution of the input.

Another extension would consist of adapting the pass-efficient algorithm for facility location, which is quite general, to solve other interesting clustering problems from combinatorial optimization. Of particular interest would be an algorithm for a generalization of the famous k -means problem to an arbitrary metric, in which we must choose k centers C that minimize $\sum_{i \in X} d(i, C)^2$. Good pass-efficient algorithms

exist for the important Euclidean case [22], but not for the generalized metric.

These are just two important practical problems that may be abstracted and studied rigorously in the context of the pass-efficient or other massive data set models of computation. The potential for theoretical computer science to provide new perspectives and interesting insights into practical massive data set problems is enormous; this dissertation is just a part of the beginning of this exploration.

Bibliography

- [1] A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 573–581, New York, NY, USA, 2005. ACM Press.
- [2] G. Aggarwal, M. Datar, S. Rajagopalan, and M. Ruhl. On the streaming model augmented with a sorting primitive. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 540–549, 2004.
- [3] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.
- [4] S. Arora and R. Kannan. Learning mixtures of separated nonspherical Gaussians. *Annals of Applied Probability*, 15(1A):69–92, 2005.
- [5] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k -medians and related problems. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 106–113, 1998.
- [6] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, pages 222–231, 2004.
- [7] J. Aspnes, K. Chang, and A. Yampolskiy. Inoculation strategies for victims of viruses and the sum-of-squares partition problem. In *To appear in Journal of Computer and Systems Sciences. A preliminary version appeared in the 16th Annual ACM-SIAM Symposium on Discrete Algorithms.*, pages 43–52, 2005.
- [8] M. Badoiu, A. Czumaj, P. Indyk, and C. Sohler. Facility location in sublinear time. In *32nd International Colloquium on Automata, Languages, and Programming*, pages 866–877, 2005.
- [9] Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004.

- [10] P. Billingsley. *Probability and Measure*. Wiley-Interscience, 3rd edition, 1995.
- [11] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(1):929–965, 1989.
- [12] A. Chakrabarti, S. Khot, and X. Sun. Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In *IEEE Conference on Computational Complexity*, pages 107–117, 2003.
- [13] K. Chang and R. Kannan. The space complexity of pass-efficient algorithms for clustering. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1157–1166, 2006.
- [14] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.
- [15] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. *SIAM Journal on Computing*, 34(4):803–824, 2004.
- [16] M. Charikar, L. O’Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the 35th Annual ACM Symposium on the Theory of Computing*, pages 30–39, 2003.
- [17] M. Charikar and R. Panigrahy. Clustering to minimize the sum of cluster diameters. *Journal of Computer and System Sciences*, 68(2):417–441, 2004.
- [18] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. In *IEEE Conference on Computational Complexity*, pages 144–153, 2005.
- [19] S. Dasgupta. Learning mixtures of Gaussians. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 634–644, 1999.
- [20] A. Deshpande, L. Rademacher, S. Vempala, and G. Wang. Matrix approximation and projective clustering via volume sampling. *17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1117–1126, 2006.
- [21] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1), 2005.
- [22] P. Drineas, A. M. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Machine Learning*, 56(1-3):9–33, 2004.

- [23] P. Drineas and R. Kannan. Pass efficient algorithm for large matrices. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 2003.
- [24] P. Drineas, M. Mahoney, and R. Kannan. Fast Monte Carlo algorithms for matrices I: approximating matrix multiplication. *To appear in SIAM Journal on Computing*. Available as *YALEU/DCS/TR-1269*, 2004.
- [25] P. Drineas, M. Mahoney, and R. Kannan. Fast Monte Carlo algorithms for matrices II: computing low-rank approximations to a matrix. *To appear in SIAM Journal on Computing*. Available as *YALEU/DCS/TR-1270.*, 2004.
- [26] P. Drineas, M. Mahoney, and R. Kannan. Fast Monte Carlo algorithms for matrices III: computing an efficient approximate decomposition of a matrix. *To appear in SIAM Journal on Computing*. Available as *YALEU/DCS/TR-1271.*, 2004.
- [27] G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. *SIAM Journal on Computing*, 28(6):2187–2214, 1999.
- [28] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal on Computing*, 31(4):1090–1118, 2002.
- [29] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the streaming model: The value of space. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 745–754, 2005.
- [30] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L^1 -difference algorithm for massive data streams. *SIAM Journal on Computing*, 32(1):131–151, 2002.
- [31] A. M. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the Association for Computing Machinery*, 51(6):1025–1041, 2004.
- [32] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*, pages 389–398, 2002.
- [33] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 79–88, 2001.
- [34] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 649–657, 1998.

- [35] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 471–475, 2001.
- [36] S. Guha, N. Mishra, R. Motwani, and L. O’Callaghan. Clustering data streams. In *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing*, pages 359–366, 2000.
- [37] J. Hastad. Some optimal inapproximability results. *Journal of the Association for Computing Machinery*, 48(4):798–859, 2001.
- [38] M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: External Memory Algorithms*, 50:107–118, 1999.
- [39] D. Hochbaum, editor. *Approximation Algorithms for NP-hard problems*. PWS Publishing, Boston, 1996.
- [40] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.
- [41] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.
- [42] P. Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, pages 428–434, 1999.
- [43] P. Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. In *Proceedings of the 41th IEEE Symposium on Foundations of Computer Science*, pages 189–197, 2000.
- [44] P. Indyk. Algorithms for dynamic geometric problems over data streams. In *Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, pages 373–380, 2004.
- [45] P. Indyk and D. Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the 37th Annual ACM Symposium on the Theory of Computing*, pages 202–208, 2005.
- [46] B. Kalyanasundaram and G. Schintger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.
- [47] R. Kannan, H. Salmasian, and S. Vempala. The spectral method for general mixture models. In *Proceedings of the 18th Annual Conference on Learning Theory*, pages 444–457, 2005.

- [48] R. Kannan, S. Vempala, and A. Vetta. On clusterings: Good, bad and spectral. *Journal of the Association for Computing Machinery*, 51(3):497–515, 2004.
- [49] M. Kearns and U. Vazirani. *An introduction to computational learning theory*. MIT Press, 1994.
- [50] S. Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*, pages 767–775, 2002.
- [51] S. Khot and N. Vishnoi. The unique games conjecture, integrality gap for cut problems and the embeddability of negative type metrics into l_1 . In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 53–62, 2005.
- [52] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1996.
- [53] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the Association for Computing Machinery*, 46(6):787–832, 1999.
- [54] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 229–242, 2002.
- [55] A. Meyerson. Online facility location. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 426–431, 2001.
- [56] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.
- [57] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.
- [58] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [59] J. I. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
- [60] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.
- [61] A. A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992.

- [62] D. B. Shmoys, É. Tardos, and K. Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 265–274, 1997.
- [63] Y. Sinai. *Probability Theory: An Introductory Course*. Springer, 1992.
- [64] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, 1996.
- [65] M. Talagrand. Sharper bounds for Gaussian and empirical processes. *Annals of Probability*, 22(1):28–76, 1994.
- [66] V. Vapnik and A. Červonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.
- [67] V. Vazirani. *Approximation Algorithms*. Springer, 2001.
- [68] S. Vempala. *The Random Projection Method*. American Mathematical Society, 2004.
- [69] S. Vempala and G. Wang. A spectral algorithm for learning mixtures of distributions. *Journal of Computer and System Sciences*, 68(4):841–860, 2004.
- [70] A. Yao. Some complexity questions related to distributed computing. In *Proceedings of the 11th Annual ACM Symposium on the Theory of Computing*, pages 290–213, 1979.