APPROXIMATION ALGORITHMS FOR CLUSTERING STREAMS AND LARGE DATA SETS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Liadan O'Callaghan
June 2003

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Rajeev Motwani
(Principal Adviser)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mihalis Yannakakis

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Nina Mishra

Approved for the University Committee on Graduate Studies:

To my parents.

Acknowledgments

I owe a great debt of gratitude to my advisor Rajeev Motwani. Rajeev is better than the advisor I would have ended up with at a design-your-own-advisor workshop. He always introduces me to exactly the right people and research problems; he is around whenever I have a question, but encourages me to think on my own and work on any problems that interest me; his knowledge of the field is such that no amount of time I ever spent searching the literature for related work ever yielded as much information as a question to him during one of our weekly meetings; and his liberal praise and amusing anecdotes are always appreciated.

Nina Mishra of HP Labs has been an invaluable mentor, advocate, colleague, and friend. She listens to my half-formed thoughts, whether they are ill-conceived or clever, and gives me the benefit of her ideas, experience, and knowledge. Her mental catalogue of problems is always overflowing, so we are never bored. I also thank the other researchers who made my time at HP Labs very fruitful and exciting.

I extend most appreciative thanks to Mihalis Yannakakis, who served on my Reading Committee, and to Ashish Goel and Jennifer Widom, who served on my Orals Committee. Many other professors have taught and encouraged me, including Andrew Appel, Sanjeev Arora, Dan Boneh, Doug Clark, David Dill, Hector Garcia-Molina, Leo Guibas, Don Knuth, Daphne Koller, Daniel Lehmann, Kai Li, John Mitchell, Serge Plotkin, Vaughan Pratt, Dana Ron, Yoav Shoham, Ken Steiglitz, and Jeff Ullman.

I thank my co-authors not named above: Brian Babcock, Moses Charikar, Mayur Datar, Tomás Feder, Sudipto Guha, Adam Meyerson, Chris Olston, Rina Panigrahy, and Dilys Thomas. I hope that my graduation will not mean the end of my collaboration with them.

I am grateful to the students and staff of the Computer Science Department, who have gotten me out of many a tough scrape, and whom I will always remember fondly. I thank the administrative staff, who led me through the maze of Stanford bureaucracy and maintained the well-oiled machine that is the Gates computer network. My fellow students in the Computer Science department have proved an invaluable source of knowledge, inspiration, and ideas, as well as a wellspring of advice, much of it solicited. I particularly thank Zoe Abrams, Gagan Aggarwal, Urszula Chajewska, Steve Freund, Aris Gionis, Greg Humphreys, Pedrito Maynard-Zhang, Mark Mitchell, Kamesh Munagala, Aarati Parmar, Kelly Shaw, Craig Silverstein, Suresh Venkatasubramanian, An Zhu, the past and present occupants of the theory wing, and all those students I have been privileged to meet.

Finally, I thank my husband Xavier, a source of wisdom, encouragement, strength, and especially entertainment.

Abstract

As data collection and storage become easier and cheaper, corporate and research data warehouses have proliferated and grown. In exploring the ways in which these vast resources can be used, datamining algorithm designers face new problems that arise due to the size and nature of many data sets. The amount of data that can be stored on a set of disks has far outstripped the quantity that can be processed in the main memory of a computer, and in many cases it is desirable to process information as it is generated, and then discard all but a short synopsis. New data models that reflect the importance of processing a large amount of data using a small amount of space and time have been developed which allow the designers of new algorithms to evaluate the relative benefits of various approaches.

We will discuss three such models, each with different algorithmic requirements: the *Very Large Data Set* model, which applies to static data sets stored on distributed systems but larger than main memory; the *Data Stream* model, which applies to static data sets larger than main memory that are stored on slow, linearly accessible devices; and the *Sliding Window* model, which assumes a data set that may be in the process of being generated, without any foreseen end, of which, at any moment, only a recent segment (which is much larger than main memory) is of interest. Clustering, or grouping data into representative subsets, is a tool commonly used in data analysis, and can often make the manipulation of large data sets simpler. For example, a search engine could cluster the pages returned by a query, and then return a summary of the categories of pages found, enabling a user to narrow or change the search without reading through a long list of results. We will present algorithms for variants of the clustering problems known as k–Median and k–Center, both NP-Hard problems

that admit good approximation algorithms. For each of the above-mentioned data models, we will present constant-factor approximation algorithms for various versions of k–Median and k–Center.

Contents

List of Figures

Chapter 1

Introduction

As data collection and storage become easier and cheaper, corporate and research data warehouses have proliferated and grown. Nationwide retailers, telecommunications companies, medical researchers, website operators, and arbitrageurs have access to huge databases full of sales, telephone call record, internet, genome, molecule configuration, clickstream, and stock ticker data. The ways in which these vast resources can be used are only beginning to be explored. Credit card companies may wish to classify their customers' purchases to detect fraud or to identify groups who would be likely to take advantage of particular promotions. Pharmaceutical companies would benefit from algorithms that could group enzymes, pathogens, and cell-membrane receptors according to which types of molecular substructures they contain, and match them with drugs that would bind with the appropriate sites. Internet search engines would like to summarize the results of a search query as just a few categories of web pages, so that the user can quickly find a webpage of interest or else narrow or change the search.

The size and nature of these data warehouses pose specific problems to datamining algorithm designers, however. The amount of data that can be stored on a set of disks has far outstripped the quantity that can be processed in the main memory of a computer, and in many cases it is desirable to process information as it is generated, and then discard all but a short synopsis. New data models that reflect the importance of processing a large amount of data using a small amount of space and time have been developed which allow the designers of new algorithms to evaluate the relative benefits of various approaches.

We will discuss three data models, each of which demands a different kind of algorithm. In the first model, which we will call the *Very Large Data Set* model, we assume a static, stored data set that admits random access, but which is so large that we can only use algorithms whose running time and space requirements are very efficient and independent of the problem-instance size. In the second, we assume that the data set is bounded but too large to be processed all at once in main memory. In this, the *Data Stream* model [53], we will assume that the algorithm has very little memory compared with the size of the data, that random access to the data is not available, and that the data can only be accessed via linear scans. In other words, the algorithm cannot choose the order in which it receives the data, and it can only store a small amount of information about the data it has seen. The goal of data stream algorithms will be to process the input in a single linear scan (or in a small number of scans), and in very small space. Finally, the third model we will describe is the *Sliding Window* model. This is a different data stream model that assumes that the data set may not be stored at all, and may be in the process of being generated, without any foreseen end. In this model, as the stream is arriving for processing, the processing algorithm may receive queries about the nature of the data, but only the most recent part of the stream, at any given moment, is considered "relevant." A "window" that always represents some final segment of the stream, and that "slides" forward along the stream, dropping old elements as it advances, defines the part of the data on which queries should be answered. The complication in this model is that the amount of data defined by this stream may, in general, be much too large to be held explicitly in main memory. The goal of sliding-window algorithms is to maintain enough information about the data in the current window, which it updates as the window slides, so that it can give approximately accurate answers to queries. Next we will elaborate more on these three models.

1.1 The Very Large Data Set Model

In Chapter 3, we will present a sampling-based algorithm that assumes the *Very Large Data Set Model*, in which random access is available, but it is desirable to be able to carry out computations in such a way that the running time and space requirements are always low and do not grow with the size of the data set. For example, the data set could be the set of all World Wide Web pages, which are distributed over many computers, but which admit random access. In this case, it is impossible for the entire data set to be stored in the memory of a single machine. Furthermore, it is best if the number of webpage requests made by each invocation of the algorithm, as well as the number of webpages each invocation stores, are guaranteed to be small. These numbers should not depend on the size of the instance, even if the number of webpages grows exponentially over time, so that the algorithm is invoked on larger and larger instances. This model is quite different from the data stream model, which does not allow random access but admits running times that grow (slowly) with input size.

1.2 The Data Stream Model

A data stream is a finite sequence of points $x_1, \ldots, x_i, \ldots, x_n$ read in increasing order of the indices i. The performance of an algorithm that operates on data streams is measured by the number of passes it must make over the stream, when constrained in terms of available memory, and also by the more conventional measures such as running time. For example, astronomical data, telephone records, and sets of transactions at retail chains can be modeled as data streams. These data sets are far too large to fit in main memory and are typically stored in secondary storage devices, making access, particularly random access, very expensive. Data stream algorithms access the input only via linear scans and not random access, and only require one or, at most, a few such scans over the data. Furthermore, since the amount of data far exceeds the amount of space (main memory) available to the algorithm, it is not possible for the algorithm to "remember" too much of the data scanned in the past.

This scarcity of space necessitates the design of a novel kind of algorithm that stores only a *summary* of past data, leaving enough memory for the processing of future data. Because the linear order in which the data are presented may be adversarial, a data stream algorithm whose output is insensitive to data order is generally to be preferred, unless the problem specifically calls for a solution that takes data order into account.

We remark that the stream model is not the same as the model of online algorithms. Online algorithms are required to react to each input before receiving the next. For example, when a caching algorithm receives a request for a memory element that is not currently in the cache, the algorithm must pick a cache element to evict, and put the new element in its place, before receiving the next request. Furthermore, the performance metric normally applied to online algorithms is the competitive ratio, the worst-case ratio of the cost of the behavior of the online algorithm, to that of an optimal offline algorithm, which employs the strategy that is the best for the particular set (and order) of inputs given to the online algorithm. Online algorithms, then, are evaluated in terms of the (irreversible) decisions they make after each new input arrives. By contrast, a data stream algorithm need not necessarily output any answer until it reaches the end of the stream. It must decide, upon each new input, what information (if any) to store in response, but it may, for example, buffer some portion of the stream, record information about all the buffered items at once, and then discard all the buffered items, rather than making a final decision about each input individually. Whether to buffer the elements in the scarce memory is a decision that must be made when each input enters, but the algorithm is not evaluated directly on this decision. Another difference between the online and data stream models is the space requirement; an online algorithm, for example, has access to information about all the input it has seen and all the decisions it has made, whereas a data stream algorithm is typically assumed to have only small storage capabilities relative to the total number of input elements.

In Chapters 4, 5, 6, and 7, the data stream model above will be assumed. In Chapter 8, we will switch to the sliding window model, which we describe next.

1.3 The Sliding Window Model

Under the sliding window model, a data stream is a *possibly unbounded* list of points that may be generated in real time and never stored, and so only one-pass algorithms are considered. Because memory will not be expected to grow during the execution of an algorithm processing such a stream, and because the number of points that have passed at any given moment may not even be known, the space requirements of such algorithm will not depend on the length of the stream or the amount of the stream that has been processed. The goal in this case will be to maintain estimate of statistics that need only be valid for some final segment of the stream. A "window" that always contains a contiguous set of data elements in the stream, ending with the most recent point, and whose boundaries are defined by a time interval, a number of data elements, or an initiating event, is imagined to slide over the stream, so that only the points within the window are considered "relevant." In Chapter 8 we present a sliding-window algorithm in which the window is defined by a window-size parameter W: it always consists of the most recent W elements in the stream. In general, the size of the window is much too large for the algorithm to store all relevant points, and so the goal of a sliding-window algorithm is to maintain space-efficient data structures that always reflect the current contents of the window, and that enable the algorithm to answer queries posed regarding the points in the current window.

This model is more applicable than the data stream model presented earlier if the data set is being generated in real time and may not be stored (so that more than one linear scan may be impossible), and if *concept drift*, meaning a change over time in the character of the data, may occur. In particular, because it only allows queries that apply to a final segment of the stream, this model applies almost exclusively to data sets whose elements are in some time-related order, and in which concept drift is *expected*.

1.4 Clustering Data Streams and Large Data Sets

Clustering, or grouping data into representative subsets, is a tool commonly used in data analysis, and can often make the manipulation of large data sets simpler. For example, in a database maintained by a direct-marketing firm, a cluster might represent a set of customers with similar characteristics. In the context of the web, a cluster could be a set of web pages with similar content. In census data a cluster might represent a demographic group that is disproportionately likely to have contaminated drinking water. In each of these applications, computing a clustering makes the data easier to manipulate, and easier for a human to parse. For example, finding clusters in a customer database might make targeted advertising easier; clustering web pages could help identify large groups of pages on similar topics; and finding census data clusters might help identify trends that would otherwise be lost in the sea of census data. Because of their vast potential applicability, clustering algorithms have been the subject of a great deal of research.

All clustering problems involve similarity-based associations among elements of a data set, but there are many measures of similarity, many ways of measuring how well the associations reflect the character of the data, and even various concepts of association. Many clustering algorithms assume that the points exist in some metric space, often real space under the Euclidean (or another $\mathcal{L}_p$) norm; the similarity is then defined simply by the distance metric [18, 80]. In the field of document clustering, the "cosine distance," which is nearly a metric (it obeys a "relaxed triangle inequality" although not the triangle inequality itself), is often used[99]. For some clustering problems, the distance function need not obey the triangle inequality, and need not be symmetric, real-valued, or even defined on all pairs of points [13, 72, 105].

Just as there are different notions of similarity, there are different notions of what clusterings are legal, and what constitutes a good clustering. Under some definitions of clustering, a point can belong fractionally to each of several groups in the subdivision; this type of clustering is sometimes called "soft," as distinct from a "hard" clustering, in which a partition imposed on the data assigns each point to exactly one group. For example, if the data are considered to be drawn from a distribution, such as a mixture of k Gaussians, we may use an EM algorithm to learn the distribution [27, 94]; in this case, for every data point x and each of the learned Gaussians $F_i = N(\mu_i, \sigma_i)$, there will be some probability (possibly very small) that F_i generated x. On the other hand, k–Means [29, 96] is an example of an algorithm that makes a "hard" assignment of points to clusters. k–Means aims to find k centroids such that, if every point in the data set is "assigned" to its closest centroid, the sum of assignment distances is minimized, where the assignment distance of a point is typically defined as the squared Euclidean distance from the point to its nearest centroid.

In most clustering problems, the quality of a particular solution is given by some (usually NP-Hard) objective function defined over the set of possible solutions. For example, given a set of n points in $\Re^d$, one may wish to find the k Gaussians that were most likely to have generated this set [25]. A solution to the *k–Center* problem is a choice of k members of the data set, called "centers"; the k–Center objective function, which is to be minimized, is the maximum assignment distance (where assignment distance is, as before, the distance from a data point to its nearest center)[30, 57]. The *k–Median* problem[67] is the same as the k–Center problem except that the goal is to minimize the *sum*, rather than the largest, of the assignment distances. An instance of the *Facility Location* problem is a set of "demand" points and a set of "feasible facilities," with a distance function defined on all pairs of points, such that each feasible facility has an associated "facility cost." The goal is to pick a subset of the feasible facilities so as to minimize the sum of the facility costs of the chosen facilities, plus the sum of assignment distances of the demand points, where the assignment distance of a demand point is its distance to the nearest facility. This problem is a Lagrangian relaxation of k–Median in which the number of clusters (facilities) is not fixed, but an additive charge is associated with each new cluster. The *Correlation Clustering* problem is, given a complete graph on n vertices (which can be thought of as data points), each of whose edges is labeled "+" or "-," find a partition of the vertices into subsets, to minimize the number of "+" edges between subsets, plus the number of "-" edges within subsets [13]. An exception to the general rule that clustering is NP-Hard is the following problem: given n points with a symmetric but possibly non-metric distance function, divide the points into clusters so as to maximize

$D - d$, where D is the minimum inter-cluster distance and d is the maximum cluster diameter. Hierarchical agglomerative clustering (HAC) [93] finds the optimal solution to this problem in polynomial time. In Section 1.6, we will give formal definitions for the k–Median and k–Center problems, the two clustering problems on which we will focus for the remainder of this work.

Data sets in the applications mentioned are growing at an exponential rate. The web is the most striking example of this expansion, but exponential growth has been observed in many other domains as well. As the data sets grow, the need for fast clustering algorithms becomes more and more apparent. Most of the theoretical work in clustering has focused on producing polynomial-time approximation algorithms for NP-Hard clustering problems. While these algorithms work very well for moderate-size data sets, they are not useful if the data cannot be clustered in main memory. At the same time, most work in the applied community on algorithms for large data sets has resulted in the development of heuristics whose performance is unproven and may vary widely depending on the data set, the objectives of the data owner, and the tuning of a large number of complicated parameters. Here, we will consider approximation algorithms for clustering large data sets and data streams. We will analyze the performance guarantees of these approximation algorithms, and also see how principles from approximation-algorithm design can be used to build better heuristics.

1.5 Notation

First we will introduce some notation that will be arise in the algorithmic discussions.

Notation 1 *A metric space* $\mathcal{X}$ *will be denoted* $(N, dist)$*, where* N *is a (finite or infinite) ground set, and* $dist\colon N \times N \to \mathbb{R}^+ \cup \{0\}$ *is a distance function such that:*

- $\forall x,\ y \in X,\ dist(x,y) = 0 \leftrightarrow x = y$
- $\forall x,\ y \in X,\ dist(x,y) = dist(y,x)$
- $\forall x,\ y,\ z \in X,\ dist(x,y) + dist(y,z) \geq dist(x,z)$

Notation 2 *All the clustering algorithms we will present will operate on* data points, *which will be points in a metric space; in some cases, data points will have associated integer weights.*

Notation 3 *Unless otherwise specified, $\mathcal{X}$ will denote a metric space with ground set N and metric $dist(\cdot,\cdot)$. Y will specify a static, stored set of data points. X will denote a stream of data points, where stream is defined according to either the bounded-length data stream definition or the sliding-window definition.*

Notation 4 *Sometimes we will give asymptotic running times and space requirements using "$\tilde{O}$" notation for compactness. A quantity said to be $\tilde{O}(f(n)f(k))$, where n and k are input parameters variables related to the size of a problem instance, will be asymptotically $O(f(n)f(k)\log^p n \log^q k)$, for some constants p, q. That is, the $\tilde{O}$ will hide polylogarithmic factors in the variables that appear inside the parenthesis; this notation will not hide factors that are polylogarithmic in other parameters. For example, if n is the problem size and k is an input parameter, an algorithm with $\tilde{O}(k^2)$ running time may have $\Omega(k^2 \log k)$ running time but not $\Omega(k^2 \log n)$ running time.*

1.6 Clustering Definitions

We will now give the precise definitions of the clustering problems mentioned in Section 1.4 that we will study in later chapters. In the following definitions, let $\mathcal{X} = (N, \mathrm{dist})$ be a metric space, and let k be a positive integer no greater than the cardinality of N.

Except when otherwise specified, we will always assume that the space required to store any point from the metric space, and also the time required to compute the distance between any pair of points is $O(1)$. In general, we will not assume that we achieve this distance-computation upper bound by storing an $n \times n$ distance matrix, but rather that the points are given in a format that allows constant-time computation of inter-point distances.

Definition 1 *Let M and Y be finite subsets of N with $|M| = k$. Then for all $x \in Y$, define $\textsc{asg}(x, M) = \min_{m \in M} dist(x, m)$ to be the closest member of M to x.*

Let $\text{AD}(x, M) = dist(x, \text{ASG}(x, M))$ *be called the* assignment distance *of* x*, under the median set* M*.*

Definition 2 *Let* M *and* Y *be finite subsets of* N*, with* $|M| = k$*. Then define* $\Phi^Y(M) = \sum_{x \in Y} \text{AD}(x, M)$*.* $\Phi^Y(M)$*, the sum of assignment distances under the medians* M*, of the members of* Y*, is called the* k*–Median cost of the medians* M *on the set* Y*, under the metric* $dist(\cdot, \cdot)$*.*

Definition 3 *If* Y_1 *(the* demand set*) and* Y_2 *(the* feasible set*, or set of feasible medians) are subsets of* N*,* Y_1 *is finite, and* $|Y_2| \geq k$*, the* k–Median problem *is to find the set* $M \subseteq Y_2$ *of size at most* k *that minimizes* $\Phi^{Y_1}(M)$*. Thus an instance of* k*–Median is defined by the demand set* Y_1*, the set* Y_2 *of feasible medians, and* k*, the number of medians desired. Define* $\text{OPT-KM}(Y_1, Y_2, k) = \min_{M \subseteq Y_2; |M| = k} \Phi^{Y_1}(M)$*. That is,* $\text{OPT-KM}(Y_1, Y_2, k)$ *is the cost of the best* k*–Median solution when* Y_1 *is the demand set and* Y_2 *is the feasible set. If* Y_2 *is not specified, it is assumed to be* Y_1 *(this version of the problem is sometimes called* discrete k–Median*). In the* continuous k–Median problem*,* Y_2 *is assumed to be* N*.*

We will also consider *weighted* demand sets, meaning sets of elements from a metric space, each associated with a positive, integral weight. If the demand set in a k–Median instance is weighted, each assignment distance in the k–Median objective function will be multiplied by the weight of the assigned point. That is, if Y_w is a weighted demand set, such that each $x \in Y_w$ is associated with a weight $w(x)$, then the k–Median cost of a set M of medians for Y_w is $\Phi^{Y_w}(M) = \sum_{x \in Y_w} (w(x) \cdot \text{AD}(x, M))$. A demand set that is not weighted is equivalent to a weighted set in which all points are of unit weight, although the latter may require extra storage for the weights.

Definition 4 *Let* C *and* Y *be finite subsets of* N*, with* $|C| = k$*. Then define* $\Psi^Y(C) = \max_{x \in Y} \text{AD}(x, C)$*.* $\Psi^Y(C)$*, the greatest assignment distance under the centers* C*, of the members of* Y*, is called the* k*–Center cost of the centers* C *on the set* Y*.*

Definition 5 *If* Y_1 *(the* demand set*) and* Y_2 *(the* feasible set*) are subsets of* N*,* Y_1 *is finite, and* $|Y_2| \geq k$*, the* k–Center problem *is to find the set* $C \subseteq Y_2$ *of size at most*

k that minimizes $\Psi^{Y_1}(C)$. Define OPT-KC$(Y_1, Y_2, k) = \min_{C \subseteq Y_2; |C|=k} \Psi^{Y_1}(C)$. *That is,* OPT-KC$(Y_1, Y_2, k)$ *is the cost of the best k–Center solution when Y_1 is the demand set and Y_2 is the set of feasible centers. If Y_2 is not specified, it is assumed to be Y_1 (this version of the problem is sometimes called* discrete k–Center*). In the* continuous k–Center *problem, Y_2 is assumed to be N.*

In the preceding definitions, Y is actually an abbreviation for $Y \times I$, where I is a set of unique identifiers, so that if there are two or more instances of the same point in the metric space $\mathcal{X}$, they will be given separate identifiers and treated separately. For example, if the demand set consists of n copies of the same point, the size of the demand set will be n rather than 1 (the number of distinct values), and sums over the demand set will count each demand point, not just one of them. Appending an index set I to every demand set would be very cumbersome, however, and so we will omit the reference to I but treat identical members of Y as distinct entities. The sets of feasible medians and feasible centers will always be sets rather than multisets. Likewise, sets of medians or centers will always be assumed to be sets (although some of the algorithms discussed here may arrive at solutions with two or more identical medians or centers); given the k–Median and k–Center objective functions, there is no advantage to having multiple copies of a single median or center, since the presence of extra copies does not lower the objective function.

There is an important difference (in the context of datamining algorithms) between a weighted set and a multiset. In some sense, both represent the same thing: having c copies of a point is equivalent to having a single, weight-c copy of that point in a weighted set, in terms of its impact on the k–Median objective. Either way, the distance of the point to its nearest median is multiplied by c in the sum of assignment distances. In the context of datamining algorithms, however, these two types of sets are very different. A multiset consisting of c, unit-weight copies of a single point will in general take up much more space than a weighted set consisting of just one point of weight c. Some of the algorithms we will discuss can tolerate weighted input, and others cannot. In general, unless otherwise indicated, all input sets will be assumed to be multisets. Whether a set is weighted should be clear from context, but almost all input sets will be weighted.

1.7 Approximation Algorithm Definitions

Both k–Center and k–Median problems are NP-Hard [67, 38], meaning that, unless P = NP, there is no polynomial-time algorithm for either problem, that is guaranteed to get the optimal answer (i.e., the set of medians minimizing OPT-KM or the set of centers minimizing OPT-KC). However, *approximation algorithms* are known for both problems. We will discuss some of them in Chapter 2, but for now we will define approximation algorithms and some related terms.

Definition 6 *Let P be an NP-Hard optimization problem, and let* OBJ *be the objective function associated with P. If A is a polynomial-time algorithm for P, and $\mathcal{I}$ is an instance of P, let $A(\mathcal{I})$ denote the solution found by A for $\mathcal{I}$, and let* $\text{OPT}_\mathcal{I}$ *denote the cost (according to* OBJ*) of the optimal solution for $\mathcal{I}$.*

Let $\alpha > 1$ be a constant. Suppose P is a minimization problem (i.e., the optimal solution to each instance is the one that minimizes OBJ*). Then A is an α-*approximation algorithm *for P just if for every instance $\mathcal{I}$ of P,* $\text{OBJ}(A(\mathcal{I})) \leq \alpha\text{OPT}_\mathcal{I}$*. If P is a maximization problem, then A is an α-approximation algorithm for P just if for every instance $\mathcal{I}$ of P,* $\text{OBJ}(A(\mathcal{I})) \geq \text{OPT}_\mathcal{I}/\alpha$*. The value α is called the* approximation ratio *of A, and solutions found by A are said to be α-*optimal*, or* optimal to an α factor.

Super-polynomial-time computation is generally regarded as infeasible, and so constructing an approximately accurate algorithm that is useless for reasons of running time is not a popular branch of inquiry. Therefore, in general, approximation algorithms must be polynomial-time. The constant α could be replaced with a function of the problem instance, for example of the size of the input, such as $\log n$, where n is the length of the input. There are many NP-Hard problems for which the best known approximation ratio (or the best possible approximation ratio, meaning the best approximation ratio that can be achieved in polynomial time) is $\log n$ or another increasing function of the input size or input parameters.

Some algorithms for NP-Hard problems relax not only the optimality of the solution they find, but also the conformity with the criteria that determine the feasibility of the solution. For example, consider the bi-clique problem. Given a bipartite graph, it is NP-Hard to find the bi-clique (complete bipartite subgraph) with the most edges, but there is a polynomial-time algorithm [83] that finds a nearly-complete bipartite subgraph with nearly as many edges as in the largest biclique. An approximation algorithm would look for actual bicliques with nearly as many edges as in the optimal biclique, but for this problem, the best approximation ratios achievable are poor, and bicriterion algorithms do better. Relaxing the feasibility criteria, so that the solution is allowed to be a very dense and need not be complete, allows better approximation ratios, meaning subgraphs with more edges. We will define bicriterion approximations in the context of k–Median.

Definition 7 *A* bicriterion (a, b)-approximation algorithm *for the k–Median problem is a polynomial-time algorithm that, given any instance (Y_1, Y_2, k) of k–Median, produces a set M of at most ak medians with $\Phi^{Y_1}(M) \leq b\textsc{opt-km}(Y_1, Y_2, k)$.*

In the algorithms we will discuss in this paper, a and b may be constants or increasing functions of the input size or characteristics of the input set. We will also discuss bicriterion approximation algorithms for variants of k–Median and k–Center.

1.8 Overview

In Chapter 2, we review some results in the data streams and clustering literature. In Chapter 3, we assume the very large data set model. In Chapter 4 we move to the data stream model, which is the input model assumed through Chapter 7. Finally, in Chapter 8, we discuss an algorithm for the sliding window model.

In Chapter 3 we detail a k–Median algorithm whose running time is independent of the number of points in the instance. This algorithm is appropriate if the data set to be clustered is very large but still permits random access. The algorithm can be applied if there is an optimal solution in which all clusters account for at least a constant fraction of the total point set; the larger this constant is, the lower the running time, and the higher the accuracy, the algorithm. We also give an algorithm that can be used if we are willing to accept a solution that clusters *most* of the data set and treats the remaining points as *outliers*. We give lower bounds that show the relationship between running time and smallest cluster size or number of points excluded if a constant-factor approximation is to be achieved; these lower bounds show that our algorithm has near-minimal running time given its requirements.

In Chapter 4, we consider a different model of computation, namely the data stream model. In this chapter we assume a data set that is large but bounded, and which can only be accessed via (slow, expensive) linear scans. We assume that the set is too large to be processed in main memory, but that the size of memory is a small polynomial (for example, the square- or cube root) in the number of data points. We give a randomized algorithm that uses only a single linear scan and runs in $O(nk \log n)$ time, where n is the number of points and k the desired number of clusters. All these algorithm are based on a divide-and-conquer framework, by which the data stream is read in a piecemeal fashion rather than all at once. The divide-and-conquer technique and its analysis will be used in the algorithms and proofs in Chapters 5, 6, and 8. We also give a deterministic algorithm that achieves a polylogarithmic approximation factor, and lower bounds on the running time of deterministic and randomized approximation algorithms. The lower bound for randomized algorithms is due to Mettu and Plaxton [79] but is reported here for completeness.

In Chapters 5, 6, and 7, we continue to assume the data stream model of computation. In Chapter 5 we examine a one-pass algorithm for approximately solving k–Median on data streams, that only requires $k \log^2(n)$ space, representing an improvement over the space bounds given in Chapter 4. Chapter 6 contains a description of a k–Median algorithm for data streams that uses the divide-and-conquer framework and calls an efficient heuristic clustering subroutine. In that chapter we prove several theorems about the quality of the solutions that this heuristic will find, and we also present the results of experiments analyzing the performance of the divide-and-conquer application of the heuristic, and comparing this performance to the performance of some algorithms in popular use. In Chapter 7 we introduce algorithms for some data-stream variants of k–Median and k–Center. We consider the *asymmetric k–Median problem*, in which distances are not symmetric; we also consider variants of k–Median and k–Center in which the solution can exclude outliers (the former problem having been discussed, although not under the data stream model, in Chapter 3).

In Chapter 8 we move on to the sliding window model, and discuss an algorithm that maintains an approximately optimal k–Median solution for the most recent W elements of a data stream, where W is a parameter. The algorithm runs in $\tilde{O}(\frac{k^2}{\tau^3}W^{2\tau})$ space, where) $< \tau < 1$ is a constant, and the data structure requires $O(k \log W)$ amortized maintenance time whenever a new point arrives from the stream.

1.9 Original Publications

The results presented in Chapter 3 represent joint work with Adam Meyerson and Serge Plotkin [81]. The material from Chapter 4 is from a paper with Sudipto Guha, Nina Mishra, and Rajeev Motwani that was originally published in FOCS 2000[48], although the presentation is somewhat changed. Most of the results presented in Chapter 6 originally appeared in an ICDE 2002 paper with Sudipto Guha, Adam Meyerson, Nina Mishra, and Rajeev Motwani [90], although some new experiments performed since publication, as well as some results omitted from the original paper for lack of space, appear in that chapter. A combined journal version of both papers was published in a special issue of TKDE in 2003[47]. The results from Chapters 5 and 7 appear as one paper with Moses Charikar and Rina Panigrahy in STOC 2003 [21]. Finally, the material in Chapter 8 appear as part of a paper at PODS 2003 [12] with Brian Babcock, Mayur Datar, and Rajeev Motwani, although here the presentation is somewhat different, and details are included that were omitted from the original publication for lack of space.

Chapter 2

Related Work

In this chapter, we review the clustering literature and the existing work on various data models. We will focus on results from the theoretical computer science community, but we will also survey some more applied work.

2.1 Work on Related Data Models

There is abundant work on input models that capture various aspects of the growing commercial and research data warehouses. Perhaps the most common tool in algorithms for large data sets that admit random access is sampling. If drawing a random sample, with or without replacement, is possible, it often allows analysis in much less time and considerably less space, without much loss in accuracy. It is often possible to prove that a sample from a data set will share many of the essential characteristics of the original data set, so that analysis of the sample will provide almost as accurate information as could be obtained from the original data. We will mention many sampling-based results in Section2.2.1.

An input model related to, although distinct from, the data stream model is that of online algorithms. Mettu and Plaxton [78] considered a version of the k–Median problem they called the "Online Median Problem;" here the goal was to order the set of points in such a way that for any k, the first k points when treated as a k–Median solution were a good approximation to the best k–Median solution.

Meyerson gave an algorithm for the following online version of the Facility Location problem: given a set of demand points, all of which are feasible facilities, that arrive in some possibly adversarial order, upon the arrival of each demand point, either make it a facility or assign it to the nearest open facility (given that demand points can never be reassigned even if a closer facility opens). The algorithm gives an expected $O(\log n)$ approximation, and if inputs arrive in random rather than possibly adversarial order, it gives an expected $O(1)$-approximation. Earlier, Charikar *et al.* [17] gave an algorithm for a similar online version of the k-Center problem, that gives a constant factor approximation using $O(k)$ space. The algorithm maintains at most k clusters at all times and modifies the clustering solution using a very restricted set of operations that do not allow an existing cluster to be broken up and its points distributed to different clusters.

Henzinger, Raghavan, and Rajagopalan[53] formalized the data stream model and gave several algorithms and complexity results related to graph-theoretic problems and their applications. One of the first results in data streams was the result of Munro and Paterson [85], who studied the space requirements of selection and sorting as a function of the number of passes made over the data. They showed that $O(n^{\frac{1}{p}})$ memory was both necessary and sufficient (up to logarithmic terms) for an algorithm allowed at most p passes for selection. A fundamental and widely applicable data streams result was Vitter's reservoir sampling algorithm [102], which, while reading a data stream of elements, maintains at every step a set of k elements chosen uniformly at random without replacement from the stream. The sample chosen from the first m elements can be computed very quickly given the mth stream element and the sample chosen from the first $m-1$ elements.

The literature on data stream computation is quite developed. It includes results on measuring the sizes of joins and self-joins, by Alon *et al.* [4]; counting the number of inversions (pairs of elements (x, y) with $x < y$, such that x appears after y) in a stream, by Ajtai*et al.* [2]; counting the number of triangles in a graph, by Bar-Yossef *et al.* [14]; approximating the $\mathcal{L}_p$ difference between two streams (when the streams are treated as vectors), by Feigenbaum *et al.* [34] and Fong and Strauss [36]; maintenance of histograms, by Guha *et al.* [46] and Gilbert*et al.* [41]; approximate selection in a single pass, by Manku *et al.* [73, 74]; frequency-moment approximation, by Alon *et al.* [5]; a small-space approximation algorithm for counting the number of distinct elements in a data stream, by Flajolet and Martin [35]. Data streams are also discussed by researchers in systems fields such as computer graphics [58, 84, 89].

There is also some work on data stream algorithms that discount older information. Often, older information may be unnecessary; for example, Boyen [15] showed that a probability distribution or transition function can be approximately inferred or learned from a stream of observations by an algorithm that only considers observations within a narrow window around the moment of interest. Streams that are being generated over time exhibit often concept drift, a gradual shift in the character of the data. In this case, it may be desirable to maintain at all times a characterization of the *recent* behavior of the stream, rather than of the entire history of the stream. The amount of information considered "recent" may be quite small, but it is often much larger than can be stored and analyzed in main memory. Algorithms for maintaining such a characterization therefore require some mechanism for discounting the contributions of older elements to statistics calculated for the recent part of the stream, without explicitly storing the part of the stream that is still relevant. Aging is one possible solution to this problem that typically involves giving data elements less and less weight as they become older and older. An example is the exponential-decay mechanism used by Gilbert *et al.* [40] in the maintenance of approximate answers to aggregate queries over databases. Aging is generally computationally simple, but it does not allow a hard distinction between "recent" and "old" elements; instead the contribution of each element decreases gradually as it ages.

The sliding window model, which has also received attention, treats all elements younger than a certain age as recent, and treats all others as having expired. Babcock *et al.* [10] studied sampling under the sliding window model, showing how to maintain a sample of roughly fixed size from the most recent segment of a stream. Datar *et al.* [26] considered maintaining certain simple statistics such as the number of '1's among the most recent W bits in a bitstream, the sum of the most recent W real numbers in a stream, or the $\mathcal{L}_1$ or $\mathcal{L}_2$ norm of the vector given by the most recent W stream elements. They introduced a novel structure called an Exponential

Histogram (EH) that could maintain these and a class of other statistics for the most recent W elements of a stream, using storage only logarithmic in W. Gibbons and Tirthapura [39] improved on these results using the "waves" data structure, reducing the update time, meaning the time required for updating the structure upon the insertion of a new stream element (and the expiration of an old one) from $O(1)$ amortized time and $O(\log n)$ worst-case time to $O(1)$ in the worst case.

2.2 Related Work on Clustering

Next we will review some of the existing theoretical work on clustering algorithms. First, we will list some results on k–Median and k–Center, the two problems we will be concerned with. Next, we will mention work on other clustering problems of interest, including literature in the theory and applied literature.

2.2.1 k–Median

Guha showed that it is NP-Hard to approximate k–Median within a factor of $1 + 1/e$ [43]. That is, unless P = NP, there is no polynomial-time α-approximation algorithm for k–Median with $\alpha < 1 + 1/e$. Nevertheless, the abundance of work on the k–Median problem has provided a number of constant-factor approximation algorithms [18, 19, 63]. In the geometric setting, Arora, Raghavan, and Rao [7] gave a polynomial-time approximation scheme (PTAS) for the k–Median problem in low dimensions; a PTAS is a family of algorithms that includes, for each fixed $\epsilon > 0$, a polynomial-time $(1 + \epsilon)$-approximation algorithm. (In general, the running time may be exponential in $1/\epsilon$.) Charikar, Guha, Tardos, and Shmoys [19] gave the first constant-factor approximation for general metric spaces. Jain and Vazirani [63], gave a Primal-Dual approximation; like many optimization problems, k–Median can be expressed as a linear program, and, although this algorithm does not solve an LP, it uses LP duality, applied to the k–Median primal program, to find lower bounds on the cost of the optimal k–Median solution. Charikar and Guha subsequently improved the approximation ratio [18]. The best approximation known to the general problem is $3+\epsilon$, given by Arya *et al.* [8]. All these algorithms require $\Omega(n^2)$ running time to solve k–Median approximately on n points.

Thorup [101] gave a constant-factor approximation algorithm for k–Median on sparse graph metrics (a sparse graph metric is a metric space describable by a sparse graph that represents points as vertices, and inter-point distances as the shortest-path distances between vertices). The k–Median algorithm runs in $\tilde{O}(m)$ time, where m is the number of edges in the graph. Algorithms for k–Center and Facility Location are also given. Indyk [59] gave a sampling-based bicriterion algorithm which produces a solution of βk medians for which the cost is at most α times the cost of the optimal solution with exactly k medians, where α and β are both constants greater than 1. This algorithms runs in $O(nk)$ time and will be given explicitly in Chapter 4. Mishra *et al.* [82] gave a sampling-based algorithm that, with probability at least $1-\delta$, produces k medians for a set Y whose cost on Y is at most $2\alpha\textsc{opt-km}(Y,Y,k)+\epsilon$, where α is the approximation ratio guaranteed by the k–Median approximation algorithm used as a subroutine and ϵ is an input parameter. The running time depends on $(M/\epsilon)^2$, where M is the diameter of the data set, and on $\log(n/\delta)$. In the special case of Euclidean space, the dependence of the running time on $\log(n/\delta)$ is avoided, and the clustering cost is at most $\alpha\textsc{opt}+\epsilon$. Mettu and Plaxton [79] gave a sampling-based algorithm that, with high probability, produces a constant approximation; this algorithm assumes that the diameter of the point set does not exceed the smallest distance between distinct points by a factor of more than $2^{O(n/\log(n/k))}$. The running time of this algorithm is $O(n(k+\log n)+k^2\log^2 n)$, which is $O(nk)$ when k is $\Omega(\log n)$ and $O(n/log^2 n)$. They also showed that *any* randomized k–Median algorithm guaranteeing an α-approximation with non-negligible probability (say, at least 1/100) requires $\Omega(nk)$ running time, under one assumption: that R_d, the ratio between the diameter of the point set and the smallest distance between two distinct points in the set, is at least $\alpha\beta n/k$, where β is a sufficiently large constant.

2.2.2 k–Center

If $\epsilon > 0$ there is no polynomial-time $(2-\epsilon)$-approximation for the k–Center problem unless P = NP [56], and a polynomial-time 2-approximation exists [57, 30]. Charikar *et al.* [17] gave a streaming algorithm for the k–Center problem that gives a constant-factor approximation using $O(k)$ space. In fact, that algorithm maintains at most k clusters at all times and modifies the clustering solution using a very restricted set of operations that do not allow an existing cluster to be broken up and its points distributed to different clusters. Alon *et al.* [3] presented various sampling-based property-testing algorithms for k–Center and related problems. For example, one algorithm distinguishes (with high probability) between the case that a set of points can be clustered with k centers within radius b, and the case that it cannot be clustered with k centers within radius $b' > b$ without the prior removal of an ϵ fraction of the points. In the first case, the algorithm can also produce a clustering. The sample sizes (and running times) are independent of the size of the set, and are polynomial in k and $1/\epsilon$). Charikar *et al.* [20] introduced the "robust" k–Center problem, in which a small number of outliers can be excluded. They presented a bicriterion algorithm which produces k centers whose objective function cost, if the worst ϵ fraction of the points are excluded, is within a constant factor of the cost of the best clustering that excludes only an $\epsilon/2$ fraction. They also presented algorithms for similar versions of the k–Median and Facility Location problems.

2.2.3 Other Clustering Objectives

We next give a very brief survey of the work on other clustering objectives. There are many problems with objective functions similar to those of k–Median and k–Center. The Facility Location problem, as mentioned, is a Lagrangian relaxation of k–Median in which each median has an associated additive charge. Several approximation algorithms have been proposed for this problem [98, 44, 70, 23, 18], including local search variants, [70, 18, 100], which start with an initial solution and improve it by making local changes. Indyk [60] gave a PTAS for the metric 2-median problem, in which the goal is to find a cut of a point set so as to minimize the sum of all interpoint distances that do not cross the cut. Charikar and Panigrahy [22] gave a bicriterion streaming algorithm for the objective of finding k clusters so as to minimize the sum of cluster diameters. Their algorithm requires $O(k)$ space and achieves a constant factor approximation but also increases the number of centers by a constant factor; it works under the same set of restricted cluster update operations as used by the k–Center algorithm of Charikar *et al.* [17]. The problem of maximizing $D - d$, where D is the minimum inter-cluster distance and d is the maximum cluster diameter, is solved optimally by the polynomial-time Hierarchical Agglomerative Clustering (HAC) algorithm. This is another local search algorithm, which begins with each point as a singleton cluster, and then repeatedly merges the nearest pair of clusters until there is only one cluster left. $D-d$ is calculated for each intermediate clustering, beginning with the initial singleton clustering, and including every clustering that results from a merge operation, and the clustering for which this objective is maximum is also the optimal solution over all possible clusterings.

Spectral clustering algorithms compute matrices from the data (for example, the adjacency matrix or an "affinity" matrix, which represents a similarity function based on the interpoint distances), and then use the eigenvectors of the matrix to find a good clustering. Ng *et al.* [87] compute the eigenvectors of an affinity matrix for a set of points, and then run k–Means on the normalized eigenvectors to find k clusters. Kannan *et al.* [66] give an approximation algorithm for the problem of finding a partition for the vertices of a complete, edge-weighted graph so that each part of the partition has some minimum conductance α, and the total weight of inter-cluster edges is minimum. Here, the clusters are just the parts of the partition, so that inter-cluster edges are edges that cross the partition. The conductance of a cluster C is the minimum, over all cuts of the vertex set C, of the conductance of the cut. The conductance of a cut $(S, C \setminus S)$, in turn, is the total weight of edges crossing the cut, divided by the lesser of $a(S)$ and $a(C \setminus S)$, where $a(T)$, for a vertex set T, total weight of edges with at least one endpoint in T (edges with both endpoints in T are counted twice). Vempala and Wang [103], compute the singular value decomposition of a sample matrix from a mixture of spherical Gaussians; if the samples are projected into the subspace spanned by the top k right singular vectors, they become well-separated and it is easier to classify the samples according to the Gaussians from which they came.

There are also graph clustering problems. For example, Bansal *et al.* [13] show that the Correlation Clustering problem mentioned in Chapter 1 is NP-Hard. This problem is, given a complete graph, with each edge labeled either "+" or "-", find a partition of the vertices so as to minimize the number of "+" edges between clusters plus the number of "-" edges within clusters. They present a constant-factor approximation algorithm for the problem of minimizing the number of "disagreements," *i.e.*, "+" edges between clusters and "-" edges within clusters. They also present a PTAS for maximizing the number of "agreements," *i.e.*, "+" edges within clusters and "-" edges between clusters. They note that a 2-approximation for the latter problem is trivial, because a set of singleton clusters if "-" edges outnumber "+" edges, or a single cluster of all vertices if "-" edges are outnumbered, has at least half as many agreements as possible.

2.2.4 Clustering Work in the Applied Community

The k-Means algorithm [29] and BIRCH [106] are two heuristics in common use which we will discuss at length in Section 6.3. Most other clustering heuristics either do not offer the scalability required for a fast streaming algorithm, or else have objective functions very different from the k–Median and k–Center objectives, and are therefore somewhat incomparable with the algorithms we will present and analyze. We briefly review some heuristics here; a thorough treatment can be found in Han and Kamber's book [50].

Partitioning methods subdivide a data set into k groups. One such example is the k-medoids algorithm [68] which selects k initial centers, repeatedly chooses a data point randomly, and replaces it with an existing center if doing so improves the (k–Median) objective function value. k-medoids is related to the CG algorithm mentioned in Chapter 4 and detailed in Section 6.2, except that CG solves the Facility Location problem rather than k–Median. Recall that Facility Location has the same objective function as k–Median, except that in Facility Location the restriction on the number of medians is replaced with an additive cost that discourages solutions with many medians. In practice one often does not know the exact number of clusters k, and so this algorithm is more practical in such cases. Choosing a new medoid among all the remaining points is time-consuming; to address this problem, CLARA [68] used sampling to reduce the number of feasible centers. This technique is similar to what we propose in Theorem 19. A distinguishing feature of our approach is a careful understanding of how sample size affects clustering quality. CLARANS [88] draws a fresh sample of feasible centers before each calculation of the improvement in the objective function value. All of the k-medoid types of approaches, including PAM, CLARA, and CLARANS, are known not to be scalable and thus are not appropriate in a streaming context.

Other examples of partitioning methods include Bradley *et al.* [16] and its subsequent improvement by Farnstrom *et al.* [32], which repeatedly takes k weighted centers (initially chosen randomly with weight 1) and as much data as can fit in main memory and computes a k-clustering. The new k centers so obtained are then weighted by the number of points assigned, the data in memory is discarded, and the process repeats on the remaining data. A key difference between this approach and the divide-and-conquer technique is that the Bradley-Farnstrom method places higher significance on points later in the data set; by contrast, the divide-and-conquer method does not assume anything about the order of the points in the data stream. Furthermore, these approaches are not known to outperform the popular BIRCH algorithm.

Hierarchical methods, density-based methods, and grid-based methods are common classes of clustering algorithms that are designed to optimize objective functions that are different from (and, in many cases, in opposition to) the k–Median objective. Hierarchical methods decompose a data set into a tree-like structure. Hierarchical Agglomerative Clustering, or HAC, the most common, treats each point as a cluster and then repeatedly merges the closest pair. If a given number of clusters k is desired, the merging stops when only k separate clusters remain. Hierarchical algorithms are known to suffer from the irrevocability of hierarchical merge and split operations [50]. Another hierarchical technique is CURE [49], which represents a cluster by multiple points that are initially well-scattered in the cluster and then shrunk toward the cluster center by a certain fraction. HAC and CURE are designed to discover clusters of arbitrary shape and thus do not necessarily optimize the k–Median objective. Density-based methods define clusters as dense regions of space separated by less dense regions. Such methods typically continue to grow clusters as long as their density exceeds a specified threshold. Density-based algorithms include DBSCAN [31], OPTICS [6], and DENCLUE [54]. Grid-based methods typically quantize the space into a fixed number of cells that form a grid-like structure. Clustering is then performed on this structure. Examples of such algorithms include STING [104], CLIQUE [1], Wave-Cluster [97], and OPTIGRID [55]. Cluster boundaries in these methods are axis-aligned hyperplanes, (no diagonal separations are allowed) and so grid-based approaches often find solutions that are quite poor according to the k–Median objective.

Chapter 3

k–Median on Very Large Data Sets

In this chapter, we give a sampling-based algorithm for the k–Median problem, running in $O(k(\frac{k^2}{\epsilon}\log k)^{p+1}\log(\frac{k}{\epsilon}\log k))$ time and $O(\frac{k}{\epsilon}\log\frac{k}{\epsilon})$ space, where ϵ is a confidence parameter. This is the first k–Median algorithm with fully polynomial running time that is independent of n, the number of points. It gives a solution that is, with high probability, a constant-factor approximation, if each cluster in the optimal solution has $\Omega(\frac{n\epsilon}{k})$ points. We give near-matching lower bounds showing that this cluster-size assumption is necessary. We also present a related algorithm for finding a clustering that excludes a small number of outliers.

3.1 Very Large Data Set Model

Most previous work on approximation algorithms for the k–Median problem assumed that main memory was quadratic or super-quadratic in the size of the data. If the data set under consideration is the set of pages in the World Wide Web, for example, while random access to the points may be permitted, it is certainly impossible to operate on all pages at once. In fact, it is desirable for the clustering time to be fully independent of the number of data points. One technique that therefore suggests itself is random sampling. If we can learn about the properties of the entire data set by examining a small sample whose size can be independent of the size of the entire set, then we may be able to develop a scalable clustering algorithm. Sampling is a tool common to many clustering algorithms; examples of the use of random sampling in clustering algorithms can be found in Chapter 2.

We immediately recognize several challenges that will arise if sampling is to be used. First, we must be able to sample the data uniformly at random. If the selection of the sample were adversarial, we could "miss" significant portions of the data set and therefore lose the chance of producing a good clustering. Second, it is possible for a small number of data points to have a very significant effect on the clustering. For example, suppose there is a very small (say $o(\sqrt{n})$) set S of points that are extremely far from all the other points. If any of these points are assigned to groups containing points *outside* of S, the clustering quality will be poor. Unfortunately, a small sample will be very unlikely to include *even one* point from this set, and so it will be impossible to produce a good solution for this instance unless we examine more points. [1] Third, this approach precludes outputting an explicit assignment of each point to a group, since performing this assignment will require at least linear time.

In order to deal with these problems, we will make several assumptions. First, we will assume that we can select a data point chosen uniformly at random in constant time. We can think of the data as being stored in some random-access memory device[2], so that we can select a random location and access that data point in effectively constant time. We observe that if we do not have random access, and the data are ordered adversarially on a device that must be read in-order, then the data are effectively streaming and the best we can hope for is linear time (the time required for a scan of the point set, even if no other operations are performed). Second, we assume that each of the clusters in the *optimum* solution, against which we compare our clustering, is large. The "average" size of a cluster would be n/k, and we will assume each cluster to have $\Omega(n/k)$ size. Equally well, we can allow our solution to disregard a small fraction of "outlying" points. Third, we will output k group

[1] The $\Omega(nk)$ running-time lower bound given by Mettu and Plaxton [79] reflects this problem, as well.

[2] This device is not necessarily main memory; for example, the data set could be a set of web pages, distributed over machines accessible by the one performing the clustering.

medians, which are "representative" of the data set. We consider this output sufficient without assigning each point explicitly to its nearest median (indeed, if the data set is large, it is probably better to output the median set rather than an explicit assignment of every point).

The assumption that some optimal solution has relatively "large" clusters, or else that some data can be treated as outlying noise, has been seen earlier in clustering literature. For example, Dasgupta [25] assumes data generated from k Gaussians each with weight $\Omega(1/k)$, and there are various papers on clustering with outliers [3, 20]. The assumption is motivated by the fact that the goal of our algorithms is to produce several broad classes that are representative of most data points, and that clustering a small number of outliers is not as important. Often the data set itself has "noise" due to observational or recording errors; this noise may lead to small numbers of extreme outliers which should in fact be disregarded by the clustering solution.

Under the above assumptions, the algorithm presented here will produce a set of medians in $O(k(\frac{k^2}{\epsilon}\log k)^{p+1}\log(\frac{k}{\epsilon}\log k))$ time and $O(\frac{k}{\epsilon}\log\frac{k}{\epsilon})$ space. Here, ϵ is such that in some optimal solution each cluster contains at least $\frac{\epsilon n}{k}$ data points (or equivalently that we can disregard ϵn points). Observe that this running time is *independent* of n, the number of data points. Our algorithm guarantees that, with high probability, the cost medians we produce is within $O(1)$ of the cost of the optimum medians. Our methods also provide a technique for *evaluating* candidate clusterings in sublinear time, under the same assumptions about the nature of the data set. We give near-matching lower bounds relating the size of the sample to the smallest minimum cluster size that is allowable if a constant approximation is to be produced; in particular, we show that our sample size is no more than $(\log(k/\epsilon))$ times as large as it must be to guarantee a constant approximation with non-negligible probability. We remark that our results are similar to those of Mishra *et al.* [82], mentioned in Chapter 2 in the Euclidean case, although that algorithm has running time that depends on the diameter of the point set, but makes no assumptions about the sizes of individual clusters.

3.2 Overview

The algorithm consists of two steps. In the first step, described in section 3.4, we generate several candidate median sets. To generate each set, we draw a sample uniformly with replacement from the data and compute an approximately-best clustering for the sample using the algorithm of Arya *et al.* [8]. We show that, with constant probability, the cost of a candidate set is within a constant factor of the cost of the best possible set of medians. By increasing the constants in the approximation factor and sample size, we could produce an algorithm that would with high probability produce a good clustering; the required increases would be unacceptably large, however. We can obtain better results by repeatedly sampling and clustering, and taking the best of our candidate median sets.

Determining the best set of candidate medians is a nontrivial problem. The straightforward approach to testing would take linear time. Thus, the second part of our algorithm is a method for determining the best of several candidate clusterings. We show, given a number of candidates, that we are able to select one whose cost is within $O(1)$ of the cost of the best candidate. This process can be performed, again using sampling, in time independent of n. The basic technique is to draw a number of random samples and measure the quality of each set of candidate medians on each random sample. We will return the set of medians that have the best observed quality (lowest minimum value over all samples). This evaluation of candidates is discussed in section 3.5. As mentioned earlier, both steps of our algorithm require that the clusters in some optimal solution be large, and that we can select a data point uniformly from the input in constant time.

Combining the steps outlined above gives us an algorithm that produces constant approximations to the best k–Median solution, in running time independent of n, the size of the data set (under certain assumptions about the nature of the input). We present this overall result in section 3.6, and near-matching lower bounds in section 3.8.

Some data sets contain "noise" or "outliers" that make the data inherently difficult to cluster, in the following sense: the best possible k–Median solution is "much more expensive” than the best solution that excludes a small number of points (i.e., does not group them and does not pay their distance costs). Using a natural extension of the algorithms already described, we can produce a constant approximation to the best clustering that excludes ϵn outliers, provided our algorithm is allowed to exclude a slightly higher ($O(1)$ times ϵn) number of outliers. The approach used is identical, except that we need to use the algorithm of Charikar *et al.* [20] instead of that of Arya *et al.* [8] to cluster the sample. We discuss this extension in section 3.7.

3.3 Notation

Before giving the full description and analysis of the algorithm, we first introduce some notation.

Definition 8 *Y is the data set, with $|Y| = n$. Y is a subset of a metric space with metric dist$(\cdot, \cdot)$.*

Definition 9 *S and T are samples, with $|S| = s$ and $|T| = t$.*

Definition 10 *k is the desired number of medians*

Definition 11 *ϵ is such that every optimum cluster contains at least $\frac{\epsilon n}{k}$ points.*

Definition 12 *Recall that $\Phi^Y(M)$ represents the k–Median cost of medians M as a clustering of the data set Y. That is, $\Phi^Y(M) = \sum_{x \in Y} \text{AD}(x, M)$. $\Phi^S(M)$ is the cost of the same medians M when used to cluster the sample S.*

Definition 13 *M^* represents a median set that minimizes $\Phi^Y(\cdot)$, and M will represent a candidate set of medians. Here $|M| = |M^*| = k$.*

3.4 Generation of a Good Candidate Set

The candidate-generation algorithm will use as a subroutine a k–Median approximation algorithm SWAPSdue to Arya *et al.* [8], who showed that the medians M ultimately returned by SWAPS for a data set Z have $\Phi^Z(M) \leq (3+2/p)\text{OPT-KM}(Z, Z, k)$.

We will discuss SWAPS before giving the description of the candidate-generation algorithm GEN. SWAPS is a local-search algorithm, meaning it begins with an initial solution consisting of k medians, and makes local improvements to its solution until the current solution cannot be improved substantially. If the input data set is Z with $|Z| = n$, and the current solution is a set M of k medians, the only local changes to M that will be considered are "swaps," meaning exchanges of up to p members of Z that are not currently medians, for the same number of members of M. (Here, p will be a parameter no greater than k.) That is, a swap replaces M by $(M \setminus M') \cup M''$, where $M' \subseteq M$, $M'' \subseteq Z$, and $|M'| = |M''|$. Let M', the set of medians to be removed, be called the "out-set," and let M'', the set to be added, be called the "in-set." If the current solution is M, we will say a swap with out-set M' and in-set M'' is "admissible" just if $\Phi^Z((M \setminus M') \cup M'') \leq (1 - \frac{\gamma}{k})\Phi^Z(M)$, where $0 < \gamma < 1$ is a constant. That is, the swap is admissible if it lowers the objective function cost by a factor of $1 - \frac{\gamma}{k}$. A description of SWAPS is given in Figure 3.1.

Algorithm SWAPS(data set Z, integer k, integer $p \leq k$)[8]

1. $M \leftarrow$ FP(Z, k).
2. While there exists an admissible swap with out-set M' and in-set M'', where $|M'| = |M''| = p$, reset $M \leftarrow (M \setminus M') \cup M''$.
3. Return M.

Figure 3.1: Algorithm SWAPS

In Step 1 of SWAPS, we find an initial solution using the $O(nk)$-time farthest-point k–Center algorithm FP due to due to Gonzalez [42]; this algorithm finds a k–Center solution, but the k centers it finds can just as well be treated as a k–Median solution. We can prove that this k–Median solution M has $\Phi^Z(M) \leq 2n \cdot$ OPT-KM(Z, Z, k). Gonzalez showed that FP is a 2-approximation for k–Center. Recall that the k–Center objective function is the maximum assignment distance. Suppose that the k points M returned by FP have $\Psi^Z(M) = r$, and let $r^* =$ OPT-KC(Z, Z, k). Then $r \leq 2r^*$. As a k–Median solution, M has $\Phi^Z(M) \leq rn$, since no point has an assignment distance greater than r under M. However, since $r^* = \text{OPT-KC}(Z, Z, k)$, we know that $\text{OPT-KM}(Z, Z, k) \geq r^* \geq r/2$, since there is no k–Median solution in which all points have assignment distances less than r^*. It follows that $\Phi^Z(M) \leq rn \leq 2n \cdot \text{OPT-KM}(Z, Z, k)$.

We will now prove bounds on the running time SWAPS requires and the space it uses. The running time will be the running time of FP, which is $O(nk)$, plus the time spent in the improvement steps. We can bound the total time used in improvement by bounding the number of improvement steps, the number of possible swaps that must be checked in each improvement step, and the amount of time it takes to check whether a swap is admissible. The admissibility of a single swap can be determined in $O(nk)$ time: for each point in Z, the distance to the nearest member of $(M \setminus M') \cup M''$ must be calculated. Define $P(n, k, p) = C(n, p)C(k, p)$, which is $O((nk)^p)$; note that there are never more than $P(n, k, p)$ possible swaps of up to p elements, because a swap of fewer than p elements can be expressed as a p-element swap with an in-set and out-set that overlap. Therefore, each improvement step requires $O((nk)^{p+1})$ time. The number of improvement steps remains to be bounded. Note that each improvement step lowers the k–Median objective function cost by a factor of $(1 - \gamma/k)$ or more.

Since the initial solution found by FP is a $2n$-approximation, and SWAPS improves the solution by a factor of $(1 - \gamma/k)$ with each swap, there cannot be more than $\log(2n)/(-\log(1 - \gamma/k))$ improvement steps. Using the approximation that for small x, $-\log(1 - x)$ is approximately x, we find that the number of improvement steps is $O(\frac{k}{\gamma} \log n)$, which, given that γ is a constant, is $O(k \log n)$. Therefore, the amount of time required for the improvement steps is $O(k(nk)^{p+1} \log n)$. The asymptotic running time of the initial-solution algorithm FP is $O(nk)$, which is dominated by the time spent on improvements, and so the total running time of SWAPS, using FP for initialization, is $O(k(nk)^{p+1} \log n)$.

The space usage of FP is just linear in the input size; as we will see presently, the remainder of SWAPS also uses space linear in the size of the input set Z. After initialization, SWAPS only performs improvement steps, each of which entails considering up to $P(n, k, p)$ possible swaps, deciding whether each swap considered is admissible, and carrying out the first admissible swap (or returning if none exists).

Deciding whether a single swap is admissible requires no more space asymptotically than the storage of Z, and since all $P(n, k, p)$ potential swaps can be considered in a lexicographic order and need not be represented explicitly, finding an admissible swap (or finding that none exists) is a linear-space operation. Carrying out a swap also does not require more than $O(|Z|)$ space. Therefore, the algorithm SWAPS, on input set Z and integers k and p, requires $O(n)$ space and $O(k(nk)^{p+1} \log n)$.

We can now give the description of the algorithm GEN that produces a set of candidate medians M; the algorithm is given in Figure 3.2.

Algorithm GEN(data set Y, integer k)

1. Draw a sample S of s points uniformly at random, with replacement, from Y, where s is $\Omega(\frac{k}{\epsilon} \log k)$.
2. $M \leftarrow$ SWAPS(S, k). Let $M = \{m_1, \ldots, m_k\}$.
3. Return M.

Figure 3.2: Algorithm GEN

We will execute the clustering algorithm SWAPS in Step 2 using S as the demand and feasible sets, but we will treat the medians M as a *continuous k–Median* solution. That is, we will regard M as a solution to the k–Median instance with demand set S and feasible set N, where N is the set of all points in the metric space (so N at least contains all of Y). As Theorem 5 shows, if a $(g/2)$-approximation algorithm for the discrete k–Median problem is automatically a g–approximation to the continuous problem. In the case of the algorithm SWAPS, g can be chosen from the interval $[6 + 4/k, 10]$: for a given value p (the number of medians to be swapped at a time), the approximation is $6 + 4/p$. [3] Since the running time of SWAPS is $O(k(nk)^{p+1} \log n)$ on an input of n points, the running time of GEN, which invokes SWAPS on an input of $O(\frac{k}{\epsilon} \log k)$ points, is $O(k(\frac{k^2}{\epsilon} \log k)^{p+1} \log(\frac{k}{\epsilon} \log k))$. GEN requires just $O(\frac{k}{\epsilon} \log k)$ space.

[3] A lower value of g can be guaranteed (at the cost of a higher running time) if a different algorithm, for example SMALLER-SPACE from Chapter 4, is used instead of SWAPS.

Theorem 1 *With constant probability the algorithm* GEN *produces medians M such that $\Phi^Y(M) \leq O(1)\Phi^Y(M^*)$.*

Proof: The main idea of the proof is to show that each of the optimum medians has a nearby median among those produced by GEN. This claim depends upon various probabilistic events, and requires the assumption that the optimum clusters are not very small. First we make some useful definitions.

Consider an optimal median set $M^* = \{m_1^*, \ldots, m_k^*\}$. For $1 \leq i \leq k$, let C_i^* be the cluster of points about M_i^*. Let $n_i = |C_i^*|$. Let n_i^S be the number of points in the ith optimal cluster that were also picked by the sample. Let $Q_i = \sum_{x \in C_i^*} \text{AD}(x, M*)$ and $R_i = \sum_{x \in C_i^*} \text{AD}(x, M)$. That is, Q_i is the total assignment cost under the optimal centers M^* for the points in the ith optimal cluster, and R_i is the total assignment cost of the same points to our centers M. We immediately observe that $\sum_{1 \leq i \leq k} Q_i = \Phi^Y(M^*)$ and that $\sum_{1 \leq i \leq k} R_i = \Phi^Y(M)$. Let Q_i^S and R_i^S represent the same summations as Q_i and R_i, but restricted to cover only the points in the sample: $Q_i^S = \sum_{x \in C_i^* \cap S} \text{AD}(x, M*)$ and $R_i^S = \sum_{x \in C_i^* \cap S} \text{AD}(x, M)$

We can bound the cost of using our medians to cluster the entire data set Y as follows. The distance from a point $x \in Y$ to the nearest median in M is bounded by the sum of the distance to its optimum median and the distance from this optimum median to the nearest median in M. We observe $\sum_{1 \leq i \leq k} Q_i^S = \Phi^S(M^*)$ and $\sum_{1 \leq i \leq k} R_i^S = \Phi^S(M)$. The triangle inequality ensures a bound of

$$\min_{x \in C_i^* \cap S} (d(M_i^*, x) + \text{AD}(x, M))$$

on the distance from optimum median M_i^* to the nearest M_j. This quantity is at most

$$\frac{1}{n_i^S}\left(\sum_{x \in C_i^* \cap S} \text{AD}(x, M^*) + \text{AD}(x, M)\right) = \frac{1}{n_i^S}(Q_i^S + R_i^S)$$

$\Phi^Y(M)$ is therefore at most

$$\sum_{x \in Y} \text{AD}(x, M^*) + \sum_{i=1}^{k} \sum_{x \in C_i^*} \frac{1}{n_i^S}(Q_i^S + R_i^S)$$

$$= \sum_{x \in Y} \text{AD}(x, M^*) + \sum_{i=1}^{k} \frac{n_i}{n_i^S}(Q_i^S + R_i^S).$$

For each i, the expected value of $\frac{n_i}{n_i^S}$ is $\frac{n}{s}$, but the actual value may differ. Linearity of expectations does not apply, because the values of Q_i^S and R_i^S depend upon the choice of sample and are not independent of the value of n_i^S. We therefore apply Chernoff Bounds and make use of the assumption that the clusters are not small, to show that with constant probability all the n_i/n_i^S are close to their expected values. By Chernoff Bounds, $\forall i$ $\mathbf{Pr}[n_i^S < Bn_i(s/n)] < exp(-sn_i(1-B)^2/(2n))$, where $0 < B < 1$ is a chosen parameter that trades the sample size against the approximation factor. We would like this probability to be smaller than $\frac{1}{k}$ so that there is a constant probability that *all* clusters are well represented in the sample. After some algebra, and making use of our assumption that $n_i \geq \frac{\epsilon n}{k}$, we can determine that $s = \frac{2}{(1-B)^2}\frac{k}{\epsilon}\log(32k)$ is sufficient to guarantee that $\mathbf{Pr}[\exists i \; n_i^s < Bn_i(s/n)] < \frac{1}{32}$. The important point is that, with constant probability, *all* of the optimum clusters will be well represented in the sample. Provided all optimum clusters are well represented, we can simplify the earlier expression given for the quality of clustering using the medians derived from the sample, obtaining

$$\Phi^Y(M) \leq \Phi^Y(M^*) + \frac{n}{sB}\sum_{i=1}^{k}(Q_i^S + R_i^S) = \Phi^Y(M^*) + \frac{n}{sB}(\Phi^S(M) + \Phi^S(M^*)).$$

The medians M were chosen to minimize $\Phi^S(M)$. Since the subroutine SWAPS guarantees a g-approximation, we conclude that no other medians M' can have $\Phi^S(M) > g\Phi^S(M')$. In particular, $\Phi^S(M) \leq g\Phi^S(M^*)$. It follows that $\Phi^Y(M) \leq \Phi^Y(M^*) + \frac{n}{sB}(1+g)\Phi^S(M^*)$. The final step is estimating $\Phi^S(M^*)$, the cost of M^* on the sample. The expected value is $E[\Phi^S(M^*)] = \frac{s}{n}\Phi^Y(M^*)$. Of course, there is some possibility that our sample has a much higher cost, but we can apply Markov's Inequality to bound the probability of this event. We observe that $\mathbf{Pr}[\Phi^S(M^*) \leq \frac{16}{15}\frac{s}{n}\Phi^Y(M^*)] \geq \frac{1}{16}$. We conclude that $\mathbf{Pr}[\Phi^Y(M) \leq 1 + (1+g)\frac{16}{15B}\Phi^Y(M^*)] \geq 1 - \frac{1}{32} - \frac{15}{16} = \frac{1}{32}$. The required sample size was $s = \frac{2}{(1-B)^2}\frac{k}{\epsilon}\log(32k)$. If we consider $B = \frac{4}{5}$ we will obtain an approximation ratio of $1 + (1+g)\frac{4}{3}$ with probability at least $\frac{1}{32}$, and a sample size of $50\frac{k}{\epsilon}\log(32k)$. By modifying the constants involved, we can make our approximation ratio arbitrarily close to $2 + g$. ■

We have shown that a given candidate set produced by GEN has constant probability of being a good approximation to the optimal k–Median solution. One approach to clustering would be to change the constants so as to guarantee a high probability of success. We observe, however, that our proof depends upon the claim that $\Phi^S(M^*) = \frac{s}{n}\Phi^Y(M^*)$. This claim holds in an expected sense but not with high probability. It might be the case that a small number of points contribute significantly to the cost of clustering; imagine a set of $\frac{n}{s}$ points which account for nearly all the cost of the optimum clusters on the set Y. Our sample expects to see only one of these points, but there is reasonably good probability we will see two, thus doubling the value of $\Phi^S(M^*)$ relative to $\Phi^Y(M^*)$. Guaranteeing a good result with high probability would therefore cost at least a factor of two in the approximation ratio.

Instead, we will independently generate r candidate sets, where r is a constant. The probability that at least one of them produces a good approximation is $1 - (\frac{31}{32})^r$, which approaches 1 as r grows. This process allows us to guarantee with arbitrarily high probability that we can generate a set of candidates such that *one of them* is a good approximation to k–Median. That is, if the candidate sets are $M_1, \ldots, M_r$, and M_{BEST} is the candidate set with the lowest Φ^Y cost, then with probability at least $1-(31/32)^r$, $\Phi^Y(M_{BEST}) \le (1+\frac{4}{3}(1+g))\Phi^B(M^*)$. It is clear that if we are willing to tolerate a bicriterion algorithm that produces $O(k)$ (in this case, rk) medians whose cost on Y is within a constant factor of OPT-KM(Y, Y, k), we can simply return the union of all candidate median sets $\bigcup\{M_1, \ldots, M_r\}$. However, if we want just k medians, we can identify a good candidate set from $M_1, \ldots, M_r$ using the procedure given in the next section.

Next we give a procedure that will identify a good median set from $M_1, \ldots, M_r$.

3.5 Close Estimation of Candidate Quality

Suppose we are given r sets of candidate medians M_1 through M_r. We would like to determine which of these sets minimizes $\Phi^Y(M_i)$, in sublinear time. Clearly it is possible to measure $\Phi^Y(M_i)$ exactly for each i, by calculating, for each M_i, the assignment distances of *all* n points in Y to the medians M_i. This process would take $\Omega(nkr)$ time, however, defeating the purpose of using a sampling-based algorithm. Instead, we will *estimate* each $\Phi^Y(M_i)$.

The estimation procedure will be as follows. l independent estimation samples $T_1, \ldots, T_l$, with $|T_j| = t$ for all $1 \leq j \leq l$, will be drawn uniformly at random from Y. Each of these samples will consist of t points drawn uniformly at random, with replacement. The number l and the size of these samples will be discussed shortly. On each of these estimation samples T_j, for each M_i, $\Phi^{T_j}(M_i)$ will be computed *exactly* in $O(tk)$ time. For each M_i, the estimated cost $\Phi^{EST}(M_i)$ will be calculated as $\min_{1\leq j \leq l}(\frac{n}{t}\Phi^{T_j}(M_i))$. The candidate set with minimum Φ^{EST} will be returned as the final k–Median solution.

First we will show that M_{BEST}, the candidate set with lowest Φ^Y cost, will with high probability have $\Phi^{EST} \leq \alpha\Phi^Y(M_{BEST})$, for some small constant α. It will follow that with high probability there will be some M_i with a small $\Phi^{EST}(M_i)$, and therefore that the median set returned by the algorithm will at least have a low *estimated* cost. Finally we will show the soundness of the estimation procedure: that for every candidate set M_i it will be unlikely that $\Phi^{EST}(M_i)$ is much *lower* than $\Phi^Y(M_i)$. It will follow that the median set returned has not only a low *estimated* cost but also a low *actual* (Φ^Y) cost.

We introduce two new parameters α and c. The parameter α trades off the number of samples against the accuracy of the estimation, while c trades off the number of samples against the probability of failure. We independently draw $\frac{c\alpha}{\alpha-1}$ samples T_j from Y, each of size $t = \Theta(\frac{k}{\epsilon}\log\frac{k}{\epsilon})$. For each median set M_i, we will compute $\Phi^{EST}(M_i) = \min_{1\leq j\leq l}(\frac{n}{t}\Phi^{T_j}(M_i))$. We will show that the medians that minimize Φ^{EST} produce a good approximation to the minimum Φ^Y.

3.5.1 Low-Estimate Property of Candidate Sets

Lemma 1 *For any candidate set M, if we compute $\Phi^{EST}(M)$ using $\frac{c\alpha}{\alpha-1}$ samples ($\alpha \geq 2$), then with probability at least $1 - e^{-c}$, $\Phi^{EST}(M) \leq \alpha\Phi^Y(M)$.*

Proof: The proof follows directly from Markov's Inequality, and the fact that for any random sample T, $E[\Phi^T(M)] = (t/n)\Phi^Y(M)$, where $t = |T|$. ■

As $\Phi^Y(M_{BEST})$ is with high probability close to $\Phi^Y(M^*)$, with high probability $\Phi^{EST}(M_{BEST})$ is close to $\Phi^Y(M^*)$. Therefore, with high probability some candidate set will have an Φ^{EST} close to $\Phi^Y(M^*)$.

3.5.2 Soundness of the Estimation Method

For each candidate median set M we can consider the ordering of all the points $x \in Y$ by $\text{AD}(x, M)$, their assignment distances under M[4]. Further, imagine dividing all the points of Y into $b = \frac{kD}{\epsilon}$ "bins" each containing $\frac{\epsilon n}{kD}$ points, where $D \geq 2$ (see Figure 3.3). The first bin will have the "cheapest" points (i.e., those with the smallest assignment distances) and the last bin will have the most expensive points. We will denote by $\text{COST}(i)$ the cost of the ith bin, that is, the sum of assignment distances of points in this bin.

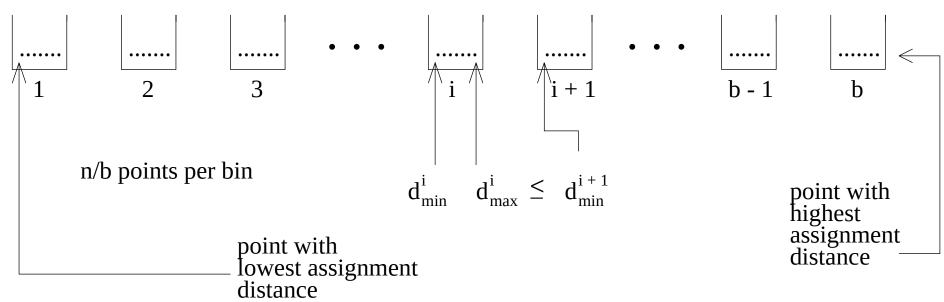

Figure 3.3: Ordering of points by assignment distances under a candidate median set M.

By selecting our sample size to be sufficiently large, we will guarantee that with high probability, for each median set, each bin is "well-represented" in each estimation sample. We will then show how to bound the cost $\Phi^Y(M)$ in terms of $\Phi^{EST}(M)$ under the assumption that all bins are well-represented.

We will set $t = \frac{4b}{(1-\beta)^2} \log\left(rb\frac{c\alpha}{\alpha-1}\right)$, which is $\Theta(\frac{k}{\epsilon} \log \frac{k}{\epsilon})$. Here β is a parameter trading off between the sample size and the accuracy of the estimation method, r is

[4]This ordering and division into bins are merely analytical tools in this proof, and are not computed by the algorithm.

the number of candidate sets, $b = \frac{kD}{\epsilon}$ is the number of bins, and α and c are the parameters defining the number of samples.

Lemma 2 *With high probability, for every triple of (bin, median set, sample), we have at least $\frac{\beta t}{b}$ points from this bin in the sample.*

Proof: The expected number of points is $\frac{t}{b}$ since points from Y are equally divided among the bins. Using Chernoff Bounds, the probability that a particular bin for a particular median set has too few points in a particular sample is at most $e^{\frac{-t(1-\beta)^2}{2b}}$. Substituting the value of t yields a probability of at most $(rb\frac{c\alpha}{\alpha-1})^{-2}$. There are r median sets, b bins, and $\frac{c\alpha}{\alpha-1}$ samples, so the probability that there exists a bad triple remains very small (and can be reduced by increasing the constants). ■

Lemma 3 *With high probability, for all candidate median sets M, $\Phi^{EST}(M) \geq \beta \sum_{i=1}^{b-1} \textsc{cost}(i)$.*

Proof: Let us denote by d_i^{max} the assignment distance of the last point, i.e., the point with largest assignment distance, in bin i. Let d_i^{min} be the assignment distance of the first point in bin i. (See Figure 3.3.)

Recall that

$$\Phi^{EST}(M) = \min_{1 \leq j \leq l} (\frac{n}{t} \Phi^{T_j}(M)),$$

where $l = c\alpha/(\alpha-1)$ is the number of estimation samples, and T_j is the jth estimation sample. By Lemma 2, with high probability, all T_j have at least $\beta t/b$ points that fall into bin i for median set M, and that will therefore contribute at least d_i^{min} apiece to $\Phi^{T_j}(M)$. Therefore, with high probability,

$$\frac{n}{t} \min_{1 \leq j \leq l} \Phi^{T_j}(M) = \Phi^{EST}(M) \geq \frac{n}{t} \sum_{i=1}^{b} \frac{\beta t}{b} d_i^{min}.$$

$\frac{n}{t} \sum_{i=1}^{b} \frac{\beta t}{b} d_i^{min} \geq \frac{\beta n}{b} \sum_{i=1}^{b-1} d_i^{max}$, since, for all $1 \leq i \leq b$, $d_i^{min} \geq d_{i-1}^{max}$. $\frac{\beta n}{b} \sum_{i=1}^{b-1} d_i^{max} \geq \beta \sum_{i=1}^{b-1} \textsc{cost}(i)$, because $\textsc{cost}(i)$ is the sum of assignment costs of the n/b members of bin i, for medians M, and these costs are each at most d_i^{max}. ■

Lemma 4 *For all candidate median sets M, the cost of the most expensive bin,* $\textsc{cost}(b)$*, is at most* $\Phi^Y(M^*) + \frac{1}{D-1}\sum_{i=1}^{b-1}\textsc{cost}(i)$*.*

Proof: We will show an upper bound on the cost of the most expensive bin, by showing a way of "moving" points in this bin to their centers[5]. We will call a point "bad" if it is in the most expensive bin, and we will call it "good" if it is not in any of the $D-1$ most expensive bins. Figure 3.4 shows the bins re-labeled as to whether they contain good, bad, or non-good points.

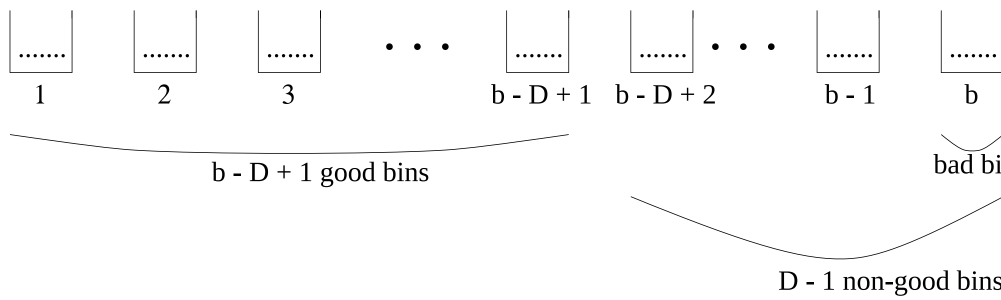

Figure 3.4: Good, bad, and non-good bins.

Consider the optimal clusters $C_1^*, C_2^*, \ldots, C_k^*$ (those induced by the optimal medians M^*). Some of these contain bad points. We will imagine moving each bad point first to its optimal median and then to the location of a good point in the same optimal cluster, in such a way that each bad point has a unique new (good) location and so that the sum of new assignment distances of the (formerly) bad points is bounded. For each optimal cluster containing one or more bad points, we can find for each bad point a unique good point in the same cluster (because each optimal cluster contains at least $\frac{\epsilon n}{k}$ points, and there are at most $(D-1)\frac{\epsilon n}{kD} = \frac{D-1}{D}\frac{\epsilon n}{k}$ points in Y which are not good). If we move each bad point to the location of a good point in the same optimal cluster, then we know that the sum of the movement distances is at most $\Phi^Y(M^*)$. This bound holds because each bad point is moved at most $d_1 + d_2$ where d_1 is its own assignment distance under the optimal centers, and d_2 is the assignment distance of a different point in that optimal cluster; adding up a d_1 and a d_2 for each

[5]The idea of "moving" points is only a tool for analyzing the assignment distances: if there is a path over which a point x can travel to its center $C(x)$, then the length of this path is certainly an upper bound on the assignment distance of x.

bad point, we get a sum of only some of the optimal-median assignment distances, which is then at most $\Phi^Y(M^*)$. Figure 3.5 illustrates this process.

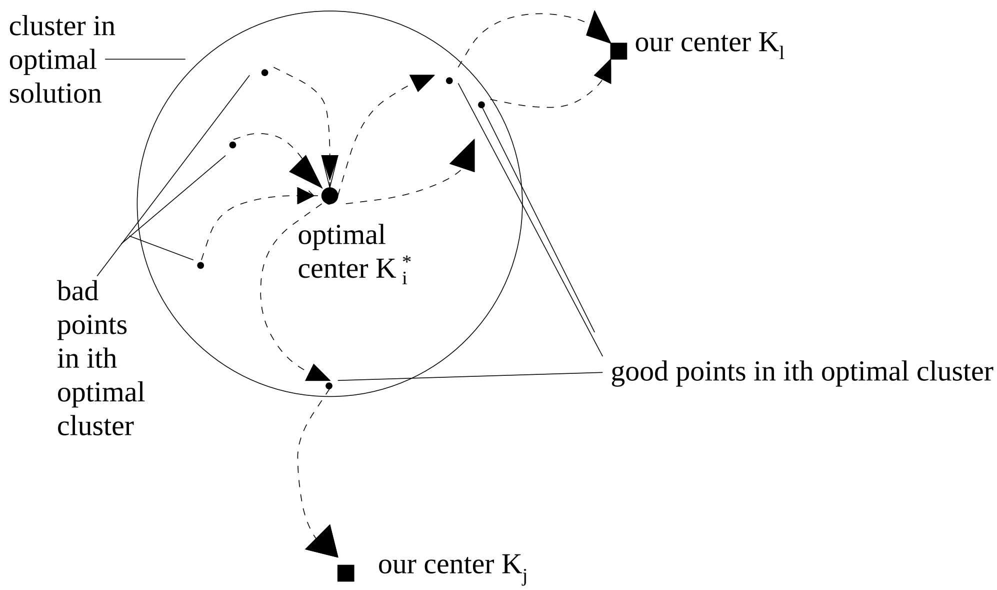

Figure 3.5: Assignment distances of bad points.

Now that the bad points are all at good locations, we need to show that the points at these good locations can be assigned to medians in M for a total assignment cost of at most $\frac{1}{D-1}\sum_{i=1}^{b}(\textsc{cost}(i))$; the lemma will follow. Recall that each bad point was assigned to a unique good point, and so, in the worst case, the bad points have been moved so that they occupy exactly the locations of the points from bin $b-D+1$ (the "worst" good bin). In other words, the total remaining distance that the bad points need to travel from their new locations to medians in M is at most $\textsc{cost}(b-D+1)$. Since the points were arranged in bins by non-decreasing assignment distances, bin $b-D+1$ accounts for at most a $1/(D-1)$ fraction of the costs of the $D-1$ bins from $b-D+1$ through $b-1$. Therefore, $\textsc{cost}(b-D+1) \leq \frac{1}{D-1}\sum_{i=b-D+1}^{b-1}\textsc{cost}(i) \leq \frac{1}{D-1}\sum_{i=1}^{b-1}\textsc{cost}(i)$. The lemma follows. ∎

Theorem 2 *With high probability, $\Phi^Y(M)$ is at most $\Phi^Y(M^*) + \frac{1}{\beta}\frac{D}{D-1}\Phi^{EST}(M)$.*

Proof: This follows directly from the two lemmas. ∎

By adjusting the constants, we can cause the above expression to converge so that $\Phi^Y(M)$ is bounded by an expression arbitrarily close to $\Phi^Y(M^*) + \Phi^{EST}(M)$. Combining this result with lemma 1, we have $\Phi^Y(M) - \Phi^Y(M^*) \leq \Phi^{EST}(M) \leq \alpha\Phi^Y(M)$.

3.6 Combining Generation and Estimation

We can now prove the following theorem.

Theorem 3 *Given an instance of k–Median, a data set Y of n points with metric dist, and an integer k, with optimal solution M^* having all clusters of size at least $\frac{\epsilon n}{k}$, we can with high probability produce a $(9 + O(\frac{1}{p}))$-approximation (where $p \in \{1, \ldots, k\}$), in $O(k(\frac{k^2}{\epsilon}\log k)^{p+1}\log(\frac{k}{\epsilon}\log k))$ time and $O(\frac{k}{\epsilon}\log\frac{k}{\epsilon})$ space.*

Proof: We first generate a constant (r) number of candidate median sets. By Theorem 1 the approximation ratio of the best candidate set is with high probability bounded above by a quantity arbitrarily close to $2 + g$. We then perform the testing step, selecting a candidate M whose estimated cost $\Phi^{EST}(M)$ is (by Theorem 2) with high probability between $\Phi^Y(M) - \Phi^Y(M^*)$ and $\alpha\Phi^Y(M)$. Therefore, the true approximation factor of the final medians is 3+g, with high probability. g is the approximation ratio of the continuous k–Median algorithm used to find the candidates; the algorithm SWAPS yields $g = 6 + 4/p$ in $O(k(\frac{k^2}{\epsilon}\log k)^{p+1}\log(\frac{k}{\epsilon}\log k))$ time and $O(\frac{k}{\epsilon}\log k)$ space. The estimation takes space linear in the estimation sample size (which is $O(\frac{k}{\epsilon}\log\frac{k}{\epsilon})$) and $O(\frac{k^2}{\epsilon}\log\frac{k}{\epsilon})\frac{cr\alpha}{\alpha-1}$ time where r is the $(O(1))$ number of candidate sets and $\frac{c\alpha}{\alpha-1}$ is the $(O(1))$ number of estimation samples. The theorem follows. ∎

3.7 Clustering with Outliers

Instead of assuming all the optimum clusters are large, we can instead allow a clustering that discards some fraction δ of the demand points. These points are considered outliers (or noise) and their distances are not included in the cost of the solution. If

Algorithm **SmallSpaceOKM**(Y)

1. For $i = 1$ to a do:
   (a) Draw $s = \frac{32k}{\delta} \ln\left(\frac{20k}{\delta}\right)$ points uniformly at random, with replacement, from Y. Call the sample S_i.
   (b) Using the algorithm of Charikar *et al.* , find medians $M_{i,1}, \ldots, M_{i,k}$ whose $(1 - 2.25\kappa_1\delta)$–cost on S_i is at most κ_2 times the cost of the best $(1 - 2.25\delta)$–solution for S_i. Store the set $\{M_{i,1}, \ldots, M_{i,k}\}$ as M_i.
2. For $j = 1$ to b, draw $t = \frac{32}{\lambda} \ln\left(\frac{2a^2b^2}{\lambda^2}\right)$ points uniformly at random, with replacement, from Y, and call the sample T_j. Here, $\lambda = 3\kappa_1 + 4$.
3. For $i = 1$ to a, calculate l_i as $\max_{1 \le j \le b} \{\Phi^{T_j}_{1-3\lambda}(M_i)\}$.
4. Return the median set M_i whose value of l_i is lowest.

Figure 3.6: Algorithm **SmallSpaceOKM**

M is a set of k medians for a set Y of n points, we will call the cost of M on Y, not counting the δn highest assignment distances, the $(1-\delta)$–cost of the solution M; this cost will be denoted $\Phi^Y_{1-\delta}(M)$. For a point set Y, a set of medians that are considered in terms of their $(1-\delta)$–cost on Y will be called a $(1-\delta)$–solution for Y; for simplicity, the $(1-\delta)$–cost of a solution already designated as a $(1-\delta)$–solution will simply be called the cost of the solution (unless this term is ambiguous). Finally, a median set M^* is an optimal $(1-\delta)$–solution if it is the set of medians with the lowest $(1-\delta)$–cost on Y.

Given a set Y of n points for which the medians $M^* = \{M^*_1, \ldots, M^*_k\}$ are an optimal $(1-\delta)$–solution of cost C, the algorithm **SmallSpaceOKM** given in Figure 3.6 will, with high probability, find a $(1-c\delta)$–solution of cost $O(1)$ times C, for constant c. It uses as a subroutine an algorithm of Charikar *et al.* [20] that, given a point set X admitting a $(1-\delta)$–solution of cost D, finds a solution whose $(1-\kappa_1\delta)$–cost on X is at most $\kappa_2 D$. For example, κ_1 could be 3 and κ_2 could be 6. (a and b in the description of **SmallSpaceOKM** are constants.)

In order to prove that **SmallSpaceOKM** finds a $(1-c\delta)$–solution of cost $O(1)$

times C, for constant c, we will first show that for every S_i, M^*, the optimal $(1-\delta)$–solution for Y, is, with constant probability, a good solution. We will infer that the solutions M_i are also good solutions for their respective samples S_i, by the approximation guarantee of the algorithm of Charikar *et al.* Next, we will show that each of these solutions is, with constant probability, good for Y. It will follow that the probability is high that at least one of the solutions M_i is a good solution for Y. Later, we will show that steps 2 through 4 of **SmallSpaceOKM** identify a candidate M_i with a good cost on Y.

Lemma 5 *For all $1 \leq i \leq a$, with probability greater than $9/10$, $\Phi^{S_i}_{1-2.25\kappa_1\delta}(M_i) \leq 1.5\kappa_2 \frac{s}{n} C$.*

Proof: Suppose, as before, that the points in Y are ordered by increasing assignment distance under M^*, and divided into $2/\delta$ bins of $\delta n/2$ points each. Fix a particular sample S_i. Recall that $|S_i| = s$. For every bin, the expected number of sample points that come from this bin is $(\delta/2)s$; since $s \geq \frac{32k}{\delta} \ln\left(\frac{20k}{\delta}\right)$, the probability that S_i contains fewer than $0.75(\delta/2)s = 0.375\delta s$ or more than $1.5(\delta/2)s = 0.75\delta s$ points from a particular bin (say, the first), is less than $\delta/(40k) \leq \delta/20$. Let E_1 denote the event that there exists a bin that contributes fewer than $0.375\delta s$ or more than $0.75\delta s$ points to S_i. An application of the union bound shows that the probability of E_1 is less than $1/10$. Therefore, assume that E_1 does not occur.

Then each of the three most expensive bins contributes at most $0.75\delta s$ points to S_i, and so there are no more than $2.25\delta s$ points in S_i that originated in these bins. We can now show that M^* is a good $(1-2.25\delta)$–solution for S_i, and it will follow that the algorithm of Charikar *et al.* finds a good $(1-2.25\kappa_1\delta)$–solution for S_i.

As before, for each l from 1 to $2/\delta$, let d_l^{min} denote the cost under M^* of the cheapest member of Y in bin l, and let d_l^{max} denote the cost of the most expensive member of Y in that bin. Then $\Phi^{S_i}_{1-2.25\delta}(M^*)$, the $(1-2.25\delta)$–cost of M^* on S_i, is at most

$$\sum_{l=1}^{2/\delta-3} (0.75\delta s) d_l^{max},$$

since excluding the top $2.25\delta s$ members of S_i guarantees that only points from the lowest $2/\delta - 3$ bins are counted, and since d_l^{max} is an upper bound on the cost, under M^*, of the points in bin l. This sum is at most

$$\sum_{l=2}^{2/\delta-2} (0.75\delta s) d_l^{min} \leq \frac{s}{n} \sum_{l=1}^{2/\delta-2} (1.5(\delta/2)n) d_l^{min} \leq 1.5\frac{s}{n} \sum_{l=1}^{2/\delta-2} \textsc{cost}(l)$$

$$\leq 1.5\frac{s}{n}\Phi_{1-\delta}^{Y}(M^*) = 1.5\frac{s}{n}C$$

where $\textsc{cost}(l)$ is the cost of the lth bin of Y, with respect to M^*. It follows that $\Phi_{1-2.25\kappa_1\delta}^{S_i}(M_i) \leq 1.5\kappa_2\frac{s}{n}C$. (Recall that M_i is the set of medians generated from S_i by the algorithm of Charikar *et al.*) As E_1 has a probability less than $1/10$, the lemma follows. ■

Next, we will show that, for every $1 \leq i \leq a$, M_i has good probability of being a good solution for Y. In particular, we will show an assignment of $(1 - 6\kappa_1 + 8)\delta n$ members of Y to the medians M_i, that costs at most $4(1 + \kappa_2)C$; it will follow that the *best* assignment of a $(1 - 6\kappa_1 + 8)\delta$ fraction of n will cost at most $4(1 + \kappa_2)C$. The following definitions will be useful in the next three proofs.

Definition 14 *Let D denote the δn members of Y that are excluded by the $(1 - \delta)$–solution M^* on Y.*

Definition 15 *For $1 \leq l \leq k$, let C_l^* represent the set of members of Y that are assigned to M_l^* under the $(1 - \delta)$–solution M^* on Y. Let $n_l = |C_l^*|$.*

Definition 16 *For $1 \leq l \leq k$, call C_l^** large *if $|C_l^*| \geq \delta n/k$; otherwise call C_l^** small. *Without loss of generality, let $C_1^*, C_2^*, \ldots, C_{k_1}^*$ be the large clusters under the $(1 - \delta)$–solution M^* for the set Y, and let $C_{k_1+1}^*, \ldots, C_k^*$ be the small clusters. Note that the small clusters together comprise at most δn points.*

Definition 17 *For all $1 \leq i \leq a$, let A_i denote the subset of S_i that is assigned under both the following solutions: M^* as a $(1 - 2.25\delta)$–solution for S_i, and M_i as a $(1 - 2.25\kappa_1\delta)$–solution for S_i. That is, $x \in A_i$ just if $x \in S_i$, x is among the cheapest $(1 - 2.25\delta)s$ members of S_i under M^*, and x is among the cheapest $(1 - 2.25\kappa_1\delta)s$ members of S_i under M_i. $|A|$ is therefore at least $(1 - 2.25(1 + \kappa_1)\delta)s$.*

Definition 18 *Let $1 \leq i \leq a$. If a large cluster C_l^* under the solution M^* for Y is such that $|C_l^* \cap A_i| \geq \frac{1}{2}|C_l^* \cap S_i|$, then C_l^* is* covered *for sample S_i. That is, C_l^* is covered for sample S_i just if at least half of its representatives in S_i are assigned under the solutions M^* and M_i for S_i. If a large cluster C_l^* is not covered for S_i, it is said to be* lost *for S_i.*

Lemma 6 *For all $1 \leq i \leq a$, if $\Phi^{S_i}_{1-2.25\delta}(M^*) \leq 1.5(s/n)C$, then with probability greater than $1 - \delta/10$, $\Phi^{Y}_{1-(6\kappa_1+8)\delta}(M_i) \leq 4(1+\kappa_2)C$.*

Proof: Fix an i from 1 to a. Let E_2 denote the event that some large cluster of n_l points had fewer than $0.75(s/n)n_l$, or more than $1.5(s/n)n_l$, members in S_i. Note that because of the size of S_i, the probability that a particular large cluster had too many or too few representatives in S_i is less than $\delta/(10k)$. An application of the union bound shows that the probability that E_2 occurred is less than $\delta/10$. Therefore, assume E_2 does not occur.

Recall that $C_1^*, \ldots, C_{k_1}^*$ are the "large" clusters; without loss of generality, let $C_1^*, \ldots, C_{k_2}^*$ (where $k_2 \leq k_1$) be the large clusters that are covered for S_i, and let $C_{k_2+1}^*, \ldots, C_{k_1}^*$ be the large clusters that are lost for S_i. For the remainder of the proof, we will simply call the former the "covered" clusters, and the latter the "lost" clusters, since in this context these terms are unambiguous. Let X be the set of all representatives of the lost clusters that were picked to be in the sample S_i. Recall that at least $(1 - 2.25(\kappa_1 + 1)\delta)s$ members of S_i were assigned under *both* M^* and M_i; then the number of points excluded by at least one of these solutions is at most $2.25(\kappa_1 + 1)\delta s$. Then $|X| \leq 4.5(\kappa_1 + 1)\delta s$, since more than half of X was excluded. By assumption that E_2 does not occur, every cluster of n_l members of Y must have at least $0.75(s/n)n_l$ members in S_i; therefore, if the lost clusters had altogether only $4.5(\kappa_1+1)\delta s$ representatives in S_i, the number of members of Y in lost clusters cannot exceed $(4.5/0.75)(\kappa_1 + 1)\delta n = 6(\kappa_1 + 1)\delta n$.

Let $Y' = \bigcup_{l=1}^{k_2} C_l^*$, the union of the covered clusters. Note that Y' is the subset of Y that excludes D (the δn points left unassigned by the $(1-\delta)$–solution M^* for Y), the small clusters, and the lost clusters; therefore $|Y'| \geq (1 - (6\kappa_1 + 8)\delta)Y$. An assignment of Y' to the medians M_i then constitutes a $(1 - (6\kappa_1 + 8)\delta)$–solution on

Y, and so $\Phi^{Y}_{1-(6\kappa_1+8)\delta}(M_i) \leq \Phi^{Y'}_{1}(M_i) = \Phi^{Y'}(M_i)$; it suffices to show an upper bound for $\Phi^{Y'}(M_i)$, and the lemma will follow.

As in section 3.4, it is possible to give an upper bound on the distance from the median M_l^* of each covered cluster C_l^* to the nearest median in M_i. For each covered cluster C_l^*, define $Q_l = \sum_{x \in C_l^* \cap A_i} d(x, C_l^*)$ and $R_l = \sum_{x \in C_l^* \cap A_i} d(x, C(x))$, where $C(x)$ is the member of M_i closest to x. That is, Q_l is the sum of assignment distances, under M^*, for the points in C_l^* that were in the sample S_i *and* assigned to medians (i.e., not excluded) under both M^* and M_i. R_l is the sum, over the same points, of assignment distances under M_i. As before, the distance between M_l^* and its nearest median in the set M_i is at most the minimum, over all $x \in C_l^* \cap A_i$, of $d(x, M_l^*) + d(x, C(x))$. Therefore, the distance from M_l^* to the nearest member of M_i is at most

$$\frac{1}{|C_l^* \cap A_i|} \sum_{x \in C_l^* \cap A_i} (d(x, M_l^*) + d(x, C(x))),$$

which, since C_l^* is covered, is at most

$$\frac{2}{|C_l^* \cap S_i|} \sum_{x \in C_l^* \cap A_i} (d(x, M_l^*) + d(x, C(x))) = \frac{2}{|C_l^* \cap S_i|}(Q_l + R_l).$$

The distance from each member of Y' to the nearest member of M_i is at most the distance to the closest member of M^*, plus the distance from there to the closest member of M_i, and so we can now put an upper bound on $\Phi^{Y'}(M_i)$.

$$\Phi^{Y'}(M_i) \leq \Phi^{Y}_{1-\delta}(M^*) + \sum_{l=1}^{k_2} \sum_{x \in C_l^*} \frac{2}{|C_l^* \cap S_i|}(Q_l + R_l)$$

$$= \Phi^{Y}_{1-\delta}(M^*) + \sum_{l=1}^{k_2} n_l \frac{2}{|C_l^* \cap S_i|}(Q_l + R_l).$$

(Recall that k_2 is the index of the highest-indexed covered cluster.) This bound holds because $\Phi^{Y}_{1-\delta}(M^*)$ is certainly an upper bound on the sum of assignment costs for the members of Y', and because the remaining distance for each point in C_l^* to the nearest member of M_i is at most $\frac{2}{|C_l^* \cap S_i|}(Q_l + R_l)$. By assumption that E_2 has not occurred, for all $1 \leq l \leq k_2$, $|C_l^* \cap S_i| \geq 0.75(s/n)n_l$, and so

$$\Phi^{Y'}(M_i) \leq \Phi^{Y}_{1-\delta}(M^*) + \sum_{l=1}^{k_2} n_l(\frac{2}{0.75(s/n)n_l})(Q_l + R_l)$$

$$= \Phi^{Y}_{1-\delta}(M^*) + \frac{8}{3}\frac{n}{s}\sum_{l=1}^{k_2}(Q_l + R_l)$$

$$\leq \Phi^{Y}_{1-\delta}(M^*) + \frac{8}{3}\frac{n}{s}(\Phi^{S_i}_{1-2.25\delta}(M^*) + \Phi^{S_i}_{1-2.25\kappa_1\delta}(M_i)).$$

The second inequality holds because $\Phi^{S_i}_{1-2.25\delta}(M^*)$ counts all the assignment distances counted in $\sum_{l=1}^{k_2} Q_l$, and because $\Phi^{S_i}_{1-2.25\kappa_1\delta}(M_i)$ counts all the assignment distances counted in $\sum_{l=1}^{k_2} R_l$. By assumption, $\Phi^{S_i}_{1-2.25\delta}(M^*) \leq 1.5(s/n)C$, and so $\Phi^{S_i}_{1-2.25\kappa_1\delta}(M_i) \leq 1.5\kappa_2(s/n)C$. It follows that

$$\Phi^{Y'}(M_i) \leq \Phi^{Y}_{1-\delta}(M^*) + \frac{8}{3}\frac{n}{s}(1+\kappa_2)\frac{s}{n}1.5C$$

$$= \Phi^{Y}_{1-\delta}(M^*) + 4(1+\kappa_2)C.$$

As $\Phi^{Y}_{1-(6\kappa_1+8)\delta}(M_i) \leq \Phi^{Y'}(M_i)$, the lemma follows. ■

Lemma 7 *With probability more than* $1-(1/5)^a$, *at least one of the candidate median sets* $M_1, M_2, \ldots, M_a$ *is a* $(1-\beta\delta)$*–solution for* Y *of cost at most* $4(1+\kappa_2)C$, *where* $\beta = 6\kappa_1 + 8$.

Proof: The proof follows from the previous two lemmas. The probability is less than $1/10+\delta/10 < 1/5$ that the $(1-\beta\delta)$–cost of a particular M_i is greater than $4(1+\kappa_2)C$. The probability that *all* of $M_1, \ldots, M_a$ cost more than $4(1+\kappa_2)C$ is therefore less than $1-(1/5)^a$. ■

Once the candidate median sets $M_1, \ldots, M_a$ are generated, it is necessary to pick one that is likely to represent a good approximation to the optimal $(1-\delta)$–solution on Y, but, as before, we want to find such a candidate set quickly (i.e., in time and space independent of n). We will again save time by quickly estimating the costs of

$M_i, \ldots, M_a$ rather than calculating them exactly. In the next part of the algorithm, b samples $T_1, \ldots, T_b$ are taken from Y and used to estimate the $(1-2\beta\delta)$-cost for each M_i on Y. It remains to be shown that the M_i with the lowest estimate is, with high probability, a good median set.

Lemma 8 *Given that one of $M_1, \ldots M_a$ has $(1-\beta\delta)$–cost at most $4(1+\kappa_2)C$, with probability more than $1-\frac{\lambda}{ab}$* **SmallSpaceOKM** *returns a set M_{i^*} with $\Phi^Y_{1-5\lambda}(M_{i^*}) \leq 8(1+\kappa_2)C$, where $\lambda = \beta\delta/2 = (3\kappa_1+4)\delta$.*

Proof: As before, for each M_i consider the ranking of Y according to assignment distances under M_i, and the division of Y into bins of size $\lambda n = \beta\delta n/2$. For all $1 \leq j \leq b$, $|T_j| = t = \frac{32}{\lambda}\ln\left(\frac{2a^2b^2}{\lambda^2}\right)$. The probability is less than $\lambda^2/(a^2b^2)$ that for a particular test sample T_j, a particular median set M_i, and a particular bin l of Y with respect to M_i, bin l contributes more than $1.5\lambda t$ or fewer than $0.75\lambda t$ representatives to T_j. Let E_3 represent the event that there *exists* a combination of test sample, median set, and bin such that the bin contributes too many or too few points to the sample. By the union bound, it follows that the probability of E_3 is less than $\lambda/(ab)$. This probability can of course be made smaller if the sample is larger. We will therefore assume for the moment that E_3 does not occur.

For all T_j, given that no bin of Y contributes more than $1.5\lambda t$ points to T_j, and therefore that the highest two bins contribute at most $3\lambda t$ members to T_j, it follows that for all M_i and T_j, $\Phi^{T_j}_{1-3\lambda}(M_i) \leq 1.5\frac{t}{n}\Phi^Y_{1-2\lambda}(M_i)$. Recall that for each S_i, **SmallSpaceOKM** calculates l_i as $\max_{1\leq j\leq b}\{\Phi^{T_j}_{1-3\lambda}(M_i)\}$. Therefore, for all M_i, $\frac{n}{t}l_i \leq 1.5\Phi^Y_{1-2\lambda}(M_i)$. By assumption, one of $M_1, \ldots M_a$ has $(1-\beta\delta)$–cost, i.e., $(1-2\lambda)$–cost, at most $4(1+\kappa_2)C$. Then, in particular, if we pick i^* so that $l_{i^*} = \min_{1\leq i\leq a}\{l_i\}$, it follows that $\frac{n}{t}l_{i^*} \leq 1.5(4(1+\kappa_2)C) = 6(1+\kappa_2)C$.

For all T_j and S_i, given that every bin of Y contributes at least $0.75\lambda t$ points to T_j, it follows that if we exclude the most expensive $3\lambda t$ members of T_j (according to their assignment distances under M_i), we are excluding, at most, the four highest bins of Y. Therefore, for all S_i and T_j, $\Phi^{T_j}_{1-3\lambda}(M_i) \geq 0.75(t/n)\Phi^Y_{1-5\lambda}(M_i)$. Then for all i, $\frac{n}{t}l_i \geq 0.75\Phi^Y_{1-5\lambda}(M_i)$, and so $0.75\Phi^Y_{1-5\lambda}(M_{i^*}) \leq \frac{n}{t}l_{i^*} \leq 6(1+\kappa_2)C$. Therefore, $\Phi^Y_{1-5\lambda}(M_{i^*}) \leq 8(1+\kappa_2)C$, and so the theorem follows. ■

The following is immediate:

Theorem 4 *With probability more than* $1-(1/5)^a-\lambda/(ab)$ *the algorithm* **SmallSpaceOKM** *returns a median set whose* $(1-c\delta)$*–cost on* Y *is* $O(1)$ *times* C*, where* c *is* $O(1)$ *and* C *is the cost of the best* $(1-\delta)$*–solution for* Y*.*

3.8 Lower Bounds

The following example illustrates the trade-off between sample size and the number of points that must be excluded from the clustering (or the minimum cluster size). Assume we are trying to cluster using a sample of size s, and that we want a $(1-\delta)$–cost competitive with the best $(1-\delta')$–cost. (We will see how δ is a function of s, but it certainly must be at least δ'.)

Consider the class of data sets consisting of:

- $k-1$ "small" clusters, each consisting of n/s co-located points.
- $\delta' n$ singleton points.
- One "big" cluster of $n-\frac{(k-1)n}{s}-\delta' n$ points that are all co-located.

Assume all inter-point distances are infinite (or very large), except distances between pairs of co-located points, which are zero.

Then the optimal $(1-\delta')$–cost is 0. We will say our sample "misses" a cluster if the cluster has no members in the sample. For each small cluster that is missed, our solution will have to exclude an additional n/s points in order to have cost competitive with the best $(1-\delta')$–cost.

We expect each small cluster to contribute one point to the sample. Therefore, each cluster has constant probability of being missed, and so we expect the sample to miss $\Omega(k)$ small clusters. Even if we repeat our sampling a constant number of times, it is still likely that every sample will miss $\Omega(k)$ small clusters. We conclude that either δ must be at least $\delta'+\Omega(\frac{k}{s})$, or our optimum solution must require that all clusters contain $\Omega(\frac{n}{s})$ points.

Our upper bounds match these lower bounds to within a factor of $\Theta(\log \frac{k}{\epsilon})$. This factor ensures that clusters are represented in (roughly) equal proportion; this condition is required for our analysis and is naturally somewhat stronger than the lower bound condition that each cluster have at least one representative in the sample. It is straightforward to extend our results to consider samples whose size depends on n; for example if we use samples of size $\Theta(\sqrt{n}\log n)$, we can cluster a data set in which the optimum solution has no cluster of size smaller than $\sqrt{n}$.

Chapter 4

k–Median on Data Streams

In this chapter we will present approximation algorithms for the k–Median clustering problem, that is suitable for data streams. We first give a randomized, constant-factor approximation algorithm for k–Median that makes one pass over the data using n^ρ memory (for $\rho < 1$) and requires only $\tilde{O}(nk)$ time, where n is the number of points in the stream. We also prove a lower bound of $\Omega(nk)$ on the running time of deterministic, constant-factor approximation algorithms for k–Median. (As mentioned in Chapter 2, Mettu and Plaxton [79] have since proved the same lower bound for the running time of all randomized, constant-factor approximation algorithms for k–Median, under reasonable assumptions.) Finally, we give a deterministic $\tilde{O}(nk)$-time, single-pass algorithm that guarantees a $poly \log(n)$-approximation to k–Medians, that uses n^ρ space, for $\rho < 1$.

The algorithms here use a "divide-and-conquer" paradigm that will be used in the next three chapters. The central technique is very simple: a data stream that cannot be clustered in main memory is instead clustered in "chunks." We take the stream a bit at a time, taking as many points as can be clustered in main memory, applying a constant-factor approximation subroutine to the memory-clusterable "chunk," and retaining the medians found by the subroutine, each with an associated weight representing the number of points from the chunk that were assigned to this median by the clustering subroutine. The original points are discarded. When all the data points have been processed, the medians retained from all the clusterings, which are merely weighted versions of points in the same metric space as that containing the original points, are themselves clustered; the medians found by this clustering are returned as the final medians. We will prove that this process still guarantees a constant-factor approximation to the optimal k–Median solution.

We begin by showing how to find an approximate k–Median solution without using quadratic space; we will later address the issue of clustering in one pass. In Section 4.1 we give a simple algorithm based on divide-and-conquer that achieves a constant-factor approximation in small space. Elements of the algorithm and its analysis form the basis for the constant-factor algorithm given in Section 4.1.3. This algorithm runs in $O(n^{1+\rho})$ time, uses $O(n^{\rho})$ memory, and makes just a single pass over the data. Next, in Section 4.3, we show how, using randomization, to reduce the running time to $\tilde{O}(nk)$ without requiring more than a single pass. In Section 4.4 we show it is not possible to obtain any bounded approximation ratio in deterministic $o(nk)$ time; we note that after this lower bound was originally published, Mettu and Plaxton [79] proved that the same lower bound applies to randomized algorithms under most reasonable assumptions. The result is stated in Chapter 2. Finally, we show how to achieve a $poly \log n$ approximation ratio in a single pass in deterministic $\tilde{O}(nk)$ time.

Throughout this chapter, we will assume the existence of a metric space $\mathcal{X} = (N, \mathrm{dist})$ (a metric space with domain N and distance function $\mathrm{dist}(\cdot, \cdot)$).

4.1 Divide-and-Conquer Framework Introduced

One of the first requisites of a data-stream clustering algorithm is that it be able to carry out its computation in small space. We consider algorithms that reduce their space requirements by examining the data in a piecemeal fashion. In particular, we study the performance of a divide-and-conquer algorithm, called SMALL-SPACE, that divides the data into segments, clusters each of these segments, and then again clusters the medians obtained (where each median is associated with a weight equal to the number of points closer to it than to any other median). We show that given a constant-factor approximation algorithm, running it in divide-and-conquer fashion still yields a constant approximation ratio (albeit a slightly higher one). We then propose another algorithm (SMALLER-SPACE) that is similar to the piecemeal approach except that instead of reclustering only once, it repeatedly reclusters weighted medians. For this algorithm, we prove that reclustering a constant number of times still produces a constant-factor approximation, although, as expected, the constant factor worsens with each successive reclustering. In describing the reclustering process, we precisely define the trade-off between memory requirements and the approximation-ratio guarantee.

4.1.1 The Algorithm SMALL-SPACE

We start with the version of the algorithm, SMALL-SPACE, that reclusters only once; this algorithm is given in Figure 4.1. Elements of the algorithm and its analysis will be used in a black-box manner in the algorithms in the rest of the paper. The algorithm requires constant-factor k–Median approximation algorithms CFA-KM-I and CFA-KM-II. (These may be the same algorithm.) Assume both of these algorithms produce weighted medians, of the form (p, w), where p is a point giving the location of the median, and w is a weight giving the number of points assigned to the median by the algorithm. Let n^ρ, where $0 < \rho < 1$, be the amount of memory available when the algorithm SMALL-SPACE is invoked, given as the number of weighted points (p, w) that can be stored simultaneously. Let $\sigma > 1$ be such that CFA-KM-I and CFA-KM-II can both cluster n_0 points in $O(n_0^\sigma)$ space.

In Step 4, let $M_j = \{m_{j,1}, \ldots, m_{j,k}\}$; note that each $m_{j,i}$ is of the form (p, w) where w is the number of members of χ_j assigned to $m_{j,i}$ by CFA-KM-I. CFA-KM-II must therefore be able to operate on a *weighted* point set. (Converting χ' into an unweighted set by making w unit-weight copies of every point with weight w would undo the compression achieved by clustering χ_j, so CFA-KM-II must be able to cluster χ' itself without expanding the set in this way.) Later, we will give an example of an algorithm that can cluster weighted sets, and show how to round the weights so that a different algorithm can be used instead.

It may not be possible to set the number and sizes of the segments χ_1 through

Algorithm SMALL-SPACE(Y)

1. $\chi' \leftarrow \emptyset$.
2. Set l so that $|Y|/l$ and lk are both at most $n^{\rho/\sigma}$.
3. Divide Y into l disjoint segments $\chi_1, \ldots, \chi_l$.
4. For each j: $M_j \leftarrow$ CFA-KM-I(χ_i); $\chi' \leftarrow \chi' \cup M_j$.
5. $M \leftarrow$ CFA-KM-II(χ').
6. Return M.

Figure 4.1: Algorithm SMALL-SPACE

χ_{i-1} so as to guarantee that each of the segments χ_i can be clustered in the available memory, that is, the memory *not* being used to store the $(i-1)k$ weighted medians from segments χ_1 through χ_{i-1}, and that the lk members of χ' can be clustered in main memory as well. Therefore, this algorithm may not be able to carry out its computation in main memory and so is not truly a streaming algorithm. We will soon see an extension of this algorithm that, given memory that is polynomial in the length of the point stream, will not overrun main memory.

4.1.2 Analysis of SMALL-SPACE

We now prove that, because at both levels of clustering the algorithm SMALL-SPACE uses constant-factor approximation algorithms, the final medians also represent a constant approximation to the k–Median problem. First, we describe the relationship between the discrete and continuous clustering problem. The following is folklore and is included for completeness.

Theorem 5 *Consider an instance of k–Median with a demand set Y. Also let M be a subset of the metric space domain N but not necessarily of Y. Suppose M is a solution of cost C, that is, $\Phi^Y(M) = C$. Then there exists $M' \subseteq Y$ with $\Phi^Y(M') \leq 2C$. Furthermore, for all $Y' \subseteq N$ with $Y \subseteq Y'$,* OPT-KM$(Y, Y, k) \leq 2$OPT-KM(Y, Y', k).

Proof: For each $m_i \in M$, pick $m'_i \in Y$ as $\text{argmin}_{s \in Y}\text{dist}(s, m_i)$. That is, for each m_i in M, let m'_i be the closest member of Y. Then the solution $M' = \{m'_1, \ldots, m'_k\}$ has $\Phi^Y(M') \leq 2C$, because, for each $s \in Y$, $\textsc{ad}(x, M') = \text{dist}(x,\ \textsc{asg}(x, M')) \leq \text{dist}(x,\ \textsc{asg}(x, M)) + \text{dist}(\textsc{asg}(x, M'),\ \textsc{asg}(x, M)) = \textsc{ad}(x, M) + \text{dist}(\textsc{asg}(x, M'),\ \textsc{asg}(x, M)) \leq 2\textsc{ad}(x, M)$. That is, because each m'_i was chosen as the closest member of Y to m_i, no point assigned to m_i is closer, and so each assignment distance at most doubles when the medians M' are used in place of M. Furthermore, if $Y \subseteq Y' \subseteq N$, a solution $M^* \subseteq Y'$ with $\Phi^Y(M^*) = \textsc{opt-km}(Y, Y', k)$ has a corresponding solution $M^{*\prime} \subseteq Y$ with $\Phi^Y(M^{*\prime}) \leq 2\textsc{opt-km}(Y, Y', k)$, and so the theorem follows. ∎

Note that Theorem 5 implies that any algorithm providing an α-approximation to the discrete k–Median problem automatically guarantees a 2α-approximation to continuous k–Median, because, for a fixed demand set, the cost of the best continuous k–Median solution must be at least half the cost of the best discrete solution.

The following separability theorem sets the stage for the divide-and-conquer algorithm.

Theorem 6 *Let Y be a set of n points, and consider an arbitrary partition of Y into sets $\chi_1, \ldots, \chi_l$. Then*

$$\sum_{j=1}^{l} \textsc{opt-km}(\chi_j, Y, k) \leq \textsc{opt-km}(Y, Y, k),$$

and

$$\sum_{j=1}^{l} \textsc{opt-km}(\chi_j, \chi_j, k) \leq 2\textsc{opt-km}(Y, Y, k).$$

If the metric space is the set of real-valued vectors under the Euclidean metric, then $\sum_{j=1}^{l} \textsc{opt-km}(\chi_j, N, k) \leq \textsc{opt-km}(Y, N, k)$, that is, the factor 2 is avoided in the case of continuous k–Median in Euclidean space.

Proof: Consider the medians $M^* \subseteq Y$ with $\Phi^Y(M^*) = \textsc{opt-km}(Y, Y, k)$. Then $\Phi^Y(M^*) = \sum_{j=1}^{l} \Phi^{\chi_j}(M^*) = \textsc{opt-km}(Y, Y, k)$. But $\sum_{j=1}^{l} \Phi^{\chi_j}(M_j^*) \leq \sum_{j=1}^{l} \Phi^{\chi_j}(M^*)$, where, for all j, M_j^* is a median set that realizes $\Phi^{\chi_j}(M_j^*) = \textsc{opt-km}(\chi_j, Y, j)$. Furthermore, Theorem 5 implies $\sum_{j=1}^{l} \textsc{opt-km}(\chi_j, \chi_j, k) \leq 2\sum_{j=1}^{l} \textsc{opt-km}(\chi_j, Y, k)$

$\leq 2\textsc{opt-km}(Y, Y, k)$. If the metric space $\mathcal{X}$ with domain N is the set of real-valued vectors under the L_2 norm, then $\sum_{j=1}^{l} \textsc{opt-km}(\chi_j, N, k) = \sum_{j=1}^{l} \Phi^{\chi_j}(M_j^*)$ $\leq \sum_{j=1}^{l} \Phi^{\chi_j}(M^*) = \textsc{opt-km}(Y, N, k)$. ■

Next we show that the new instance solved in Step 5 with demand set χ' has a good feasible clustering solution. Note that if M^* is a solution with $\Phi^Y(M^*) = \textsc{opt-km}(Y, Y, k)$, $|\chi'|$ may not overlap with M^* at all; that is, it may be the case that *none* of the medians from M^* are feasible medians for the new instance (in which we can only find medians from the set χ' itself).

Theorem 7 $\textsc{opt-km}(\chi', \chi', k) \leq 2\left(\textsc{opt-km}(Y, Y, k) + \sum_{j=1}^{l} \Phi^{\chi_j}(M_j)\right)$, *where for* $1 \leq j \leq l$ M_j *is the median set found for* χ_j *in Step 4 of the algorithm* SMALL-SPACE.[1]

Proof: We will show that the medians M^* for which $\Phi^Y(M^*) = \textsc{opt-km}(Y, Y, k)$ would have $\Phi^{\chi'}(M^*) \leq \textsc{opt-km}(Y, Y, k) + \sum_{j=1}^{l} \textsc{opt-km}(\chi_j, \chi_j, k)$. Since M^* may not be a subset of χ', it may not be a feasible solution, but Theorem 5 guarantees that there is a solution $M' \subseteq \chi'$ (hence, a *feasible* solution) with $\Phi^{\chi'}(M') \leq 2\Phi^{\chi'}(M^*)$, and the theorem will follow. Therefore, we need only show that $\Phi^{\chi'}(M^*) \leq \textsc{opt-km}(Y, Y, k) + \sum_{j=1}^{l} \textsc{opt-km}(\chi_j, \chi_j, k)$.

Recall that for $1 \leq j \leq l$, $M_j = \{m_{j,1}, \ldots, m_{j,k}\}$. For every j from 1 to l, for each each $m_{j,i} \in M_j$, identify the point $x \in Y$ assigned to $m_{j,i}$ (i.e., with $\textsc{asg}(x, M_j) = m_{j,i}$), for which the sum $\textsc{ad}(x, M_j) + \textsc{ad}(x, M^*)$ is minimal; let $x_{i,j}$ denote this point x.

Therefore, each median $m_{i,j}$ can be mapped to a member of M^* via $x_{i,j}$. The cost, say F, of assigning members of χ' to M^* according to this mapping is an upper bound on $\Phi^{\chi'}(M^*)$, since it may not assign every weighted median to its nearest member of M^*.

$$F \leq \sum_{j=1}^{l} \sum_{i=1}^{k} w_{i,j}(\text{dist}(x_{i,j}, m_{i,j}) + \textsc{ad}(x_{i,j}, M^*)),$$

[1] Again, the factor 2 is avoided if we use the Euclidean distance and allow medians to be arbitrary points.

which, by choice of the points $x_{i,j}$, is at most

$$\sum_{j=1}^{l}\sum_{i=1}^{k}\sum_{\mathrm{ASG}(x,M_j)=m_{i,j}}(\textsc{ad}(x,M_j)+\textsc{ad}(x,M^*)) \leq \Phi^Y(M^*)+\sum_{j=1}^{l}\Phi^{\chi_j}(M_j).$$

This last sum is equal to $\textsc{opt-km}(Y,Y,k)+\sum_{j=1}^{l}\textsc{opt-km}(\chi_j,\chi_j,k)$, and the theorem follows. ■

We have shown limits on the costs of the *optimal* solutions to the instances solved in Steps 4 and 5 of the algorithm SMALL-SPACE. Note that SMALL-SPACE uses constant-factor approximation algorithms CFA-KM-I and CFA-KM-II in those steps; if these algorithms guarantee approximation factors of b and c, respectively, the following theorem holds.

Theorem 8 *The algorithm* SMALL-SPACE *guarantees a constant approximation factor of* $2c(1+2b)+2b$*. That is,* $\Phi^Y(M) \leq (2c(1+2b)+2b)\,\textsc{opt-km}(Y,Y,k)$*, where* M *is the set of the final medians returned by* SMALL-SPACE*.*

Proof: Let $C^* = \textsc{opt-km}(Y,Y,k)$. Then $\Phi^Y(\chi') \leq \sum_{j=1}^{l}\Phi^{\chi_j}(M_j) \leq 2bC^*$, from Theorem 6. Let $C=\Phi^Y(\chi')$. By Theorem 7, $\textsc{opt-km}(\chi',\chi',k) \leq 2(C+C^*)$, and since Step 5 uses an algorithm with approximation ratio c, the solution found in that step costs at most $2c(C+C^*) \leq 2c(2bC^*+C^*) = 2c(1+2b)C^*$. $\Phi^Y(M)$ is at most the sum of $\Phi^Y(\chi')$ and $\Phi^{\chi'}(M)$, since for all $x \in Y$, $\textsc{ad}(x,M) \leq \textsc{ad}(x,\chi') + \mathrm{dist}(\textsc{asg}(x,\chi'),\textsc{asg}(\textsc{asg}(x,\chi'),M))$. The theorem follows. ■

If CFA-KM-I is a *bicriterion* approximation algorithm for k–Median, meaning that it produces at most ak medians whose cost on Y is at most b times $\textsc{opt-km}(Y,Y,k)$, the algorithm SMALL-SPACE will still produce a $(2c(1+2b)+2b)$-approximation, although χ' will consist of alk weighted points, so $(alk)^\sigma$ must be at most n^ρ.

4.1.3 True Streaming Algorithm

Next, we will return to a problem described earlier: how to ensure that each of the data segments χ_j, and the weighted medians χ', can be clustered in the memory that is available. First, assuming that $|\chi'| = lk$ (i.e., that CFA-KM-I is not a bicriterion algorithm), we must have $(lk)^{\sigma_2} \leq n^\rho$, so $l \leq \frac{1}{k} n^{\rho/\sigma_2}$. Next, we observe that when a segment χ_j is clustered using the algorithm CFA-KM-I, the space available (given as a number of points) is $n^\rho - (j-1)k$, and so $|\chi_j| \leq (n^\rho - (j-1)k)^{1/\sigma}$. Clearly, $|Y|$ may be so large that there is no setting of l and $|\chi_1|, \ldots, |\chi_l|$ for which these inequalities all hold.

In this case, a natural extension of SMALL-SPACE suggests itself; we will call this new algorithm SMALLER-SPACE. We shall assume from now on that there is memory available for the storage of $2n^\rho$ (rather than just n^ρ) points; this new assumption makes the algorithm easier to state and the analysis much simpler. We still assume that both CFA-KM-I and CFA-KM-II can cluster n_0 points in $O(n_0^\sigma)$ space. SMALLER-SPACE will divide memory in half, designating the first half for median storage and the second half for clustering. Define $\mu = \rho/\sigma$ and $L = \lceil \frac{\sigma - \rho}{\rho - \kappa\sigma} \rceil$, where $\kappa = \log_n k$ is guaranteed to be less than μ/α for some $\alpha > 0$. Note that L is a constant. At any given time, SMALLER-SPACE will have up to n^μ "level-i" medians, for all i from 0 to L. Level-0 medians will simply be points from Y, level-1 medians will be analogous to those found by CFA-KM-I in the algorithm SMALL-SPACE, that were ultimately clustered by CFA-KM-II to produce final medians. SMALLER-SPACE will cluster level-i medians whenever there are n^μ of them, producing k level-$(i+1)$ medians. For each i from 1 to L, M_i will be the current set of level-i medians. Since no more than n^μ points will ever be clustered at once, no more than n^ρ space will ever be occupied with clustering, and so the clustering will always fit, as intended, in the n^ρ space designated for clustering. Likewise, as long as there are never more than n^μ level-L medians, no more than Ln^μ medians will ever be stored in the n^ρ space designated for median storage. As long as $L \leq n^{\frac{\rho(\sigma-1)}{\sigma}}$, the number of medians will not overrun the n^ρ space allocated for them. We will see shortly why the n^μ limit on the number of level-L medians is guaranteed to hold. The algorithm SMALLER-SPACE is presented formally in Figure 4.2.

We can now prove that there are never more than n^μ medians in the top level. For more than n^μ level-L medians to accumulate, the algorithm would need to process strictly more than $\frac{n^{2\mu}}{k}$ level-$(L-1)$ medians, since every n^μ level-$(L-1)$ clustered would produce k level-L medians. The existence of more than $\frac{n^{2\mu}}{k}$ level-$(L-1)$

Algorithm SMALLER-SPACE(Y)

1. $U \leftarrow Y$.
2. For $1 \leq i \leq L$ $M_i \leftarrow \emptyset$.
3. While $U \neq \emptyset$, do:
   (a) Set R to be the first n^μ members of U (or all of U if $|U| < n^\mu$).
   (b) $M_1 \leftarrow M_1 \cup$ CFA-KM-I(R).
   (c) $i \leftarrow 1$.
   (d) While $i \leq L$ and $|M_i| = n^\mu$:
      i. $M_{i+1} \leftarrow M_{i+1} \cup$ CFA-KM-II(M_i).
      ii. $M_i \leftarrow \emptyset$.
      iii. $i \leftarrow i + 1$.
4. If $|M_i| > 0$ for some $0 \leq i < L$:
   (a) $i \leftarrow 0$.
   (b) While $i < L - 1$:
      i. $M_{i+1} \leftarrow M_{i+1} \cup$ CFA-KM-II(M_i).
      ii. $i \leftarrow i + 1$.
5. Return CFA-KM-II(M_L) as the final medians.

Figure 4.2: Algorithm SMALLER-SPACE

medians would imply that more than $\frac{n^{3\mu}}{k^2}$ level-$(L-2)$ medians had been clustered, and, ultimately, that Y had contained more than $\frac{n^{L\mu}}{k^{L-1}}$ points. By the definitions of L and μ, and the restriction that $k = n^\kappa < n^{\mu/\alpha}$ for some $\alpha > 0$, this number is at least n, and so for there to be more than n^μ level-L medians there would have to be more than n members of Y, which is a contradiction.

What we have proven, in effect, is that, unlike SMALL-SPACE, SMALLER-SPACE can cluster data streams, because it can read through Y just once and perform its computation within the memory allocated to it. Next, we will analyze the quality of the solution produced by SMALLER-SPACE, which we will show to be $2^{O(L)}$ times

OPT-KM(Y, Y, k), or a constant times optimal. This approximation factor will follow from the fact that for every level i, obtaining level-$(i+1)$ medians from level-i medians only causes the approximation factor to increase by a constant factor. Since there is a constant number L of levels, the total approximation factor will also be a constant.

Theorem 9 *The algorithm* SMALLER-SPACE*, given a data stream Y, produces a set M of k medians from Y such that $\Phi^Y(M)$ is at most a constant times* OPT-KM(Y, Y, k).

Proof: For i from 1 to L, let $\widehat{M_i}$ denote the set of all level-i medians ever produced by SMALLER-SPACE, and for simplicity, let OPT = OPT-KM(Y, Y, k). For all i from 1 to L, let A_i denote $\Phi^Y(\widehat{M_i})/\text{OPT}$, and let $A_{L+1} = \Phi^Y(M)/\text{OPT}$, where M is the final set of medians output by the algorithm; our goal is to find A_{L+1}. First, from Theorem 6 and the fact that CFA-KM-I is a b-approximation algorithm, $A_1 = \Phi^Y(\widehat{M_1})/\text{OPT} \leq 2b$. Next, from Theorems 7 and 8, we can see that, for all i from 2 to $L+1$, since the medians $\widehat{M_i}$ are obtained from the medians $\widehat{M_{i-1}}$ by way of CFA-KM-II, a c-approximation algorithm, $A_i \leq A_{i-1} + 2c(A_{i-1} + 1) = (2c+1)A_{i-1} + 2c$. Therefore, $A_{L+1} \leq (2c+1)^L(2b) + 2c\sum_{j=0}^{L-1}(2c+1)^j = (2c+1)^L(2b) + 2c\frac{(2c+1)^L-1}{2c} = (2c+1)^L(2b) + (2c+1)^L - 1 = (2c+1)^L(2b+1) - 1$. Therefore, $\Phi^Y(M) \leq ((2c+1)^L(2b+1)-1)\text{OPT}$. As L is a constant, the approximation factor is a constant, and the theorem follows. ■

CFA-KM-Iand CFA-KM-IIcan be bicriterion algorithms, if the algorithm SMALLER-SPACE is modified slightly. The clustering subroutine applied to M_L (the highest set of medians stored in the median-storage half of memory) must produce exactly k medians, since these will be the answer ultimately returned by SMALLER-SPACE. Suppose we use an (a, b)-approximation for all levels but the last, and a d-approximation algorithm at the end to cluster M_L for k final medians. Let $\kappa' = \log_n ak$; a must still be small enough that the compression from n^μ to ak points at each level is significant, and so κ must still be less than μ/α for some $\alpha > 0$. Then L will become $\frac{\sigma-\rho}{\rho-\kappa'\sigma}$, which is still constant. A_L, the approximation factor of M_L, will be $(2b+1)^L - l$, and the approximation factor of the final medians will be $A_L + 2d(A_L+1) = (2d+1)(2b+1)^L - 1$.

4.2 Reducing the Number of Levels

The provable approximation quality of the algorithm will depend (exponentially) on the number of levels of clustering. Even though this number is a constant given the assumptions on memory size and the number of clusters, it is profitable to reduce the number of levels. Recall that the number of levels is $\frac{\sigma-\rho}{\rho-\kappa\sigma}$. Here, ρ and κ, which relate the number of points to the size of available memory and the value k, are unchangeable, but σ, the constant giving the space requirements of the clustering subroutines, depends only on the subroutines used. Choosing a clustering subroutine with lower space complexity will therefore reduce the number of levels of clustering. Recall that the approximation ratio of the overall divide-and-conquer algorithm is $((2c+1)^L(2b+1)-1)$, so reducing the number L of levels will lower the final approximation ratio, as long as the approximation ratios b and c of the clustering subroutines used do not grow so high that they outweigh the effect of the reduction of L.

We can use the local search algorithm of Charikar and Guha[18] to provide a bicriterion approximation in space that is linear in the number of points clustered at a time. The advantage of this algorithm is that it maintains only an assignment and therefore does not require large amounts of memory for complicated data structures. The complication introduced by this algorithm is that it actually solves a lagrangian relaxation of k–Median known as "Facility Location." There are only two differences between the k–Median and Facility Location objective functions: First, only k–Median specifies the number k of medians. Second, in the Facility Location objective, there is an additive term of yz, where z is the "facility," or per-median, cost, and y is the number of medians in the solution.

Therefore, in order to use this local search algorithm, it is necessary to construct a "wrapper" algorithm around it that defines a facility cost (which in effect penalizes a solution containing many more than k medians), passes this to the local search algorithm, and reads the solution produced. The facility cost that will cause the local search algorithm to return a solution with between, say, k and $2k$ medians will depend on the structure of the inter-point distances, and will not be known up front, but the wrapper algorithm can guess this value in powers of $(1+\gamma)$ for some $0 < \gamma < 1/6$, invoking the local search algorithm with each cost that it guesses. The wrapper will determine and return the lowest-cost solution with at most $2k$ medians. At every level of clustering except the last, this combination of the wrapper algorithm and the local search algorithm can be used. For the last step, the primal-dual approximation algorithm of Jain and Vazirani[63], which is guaranteed to produce only k medians, can be employed, so that SMALLER-SPACE ultimately returns k medians.

The Running Time To cluster n_0 points, the local search algorithm of Charikar and Guha requires $O(n_0^2 \log n_0)$ time. Therefore, whenever n^μ points are clustered, the algorithm takes $O(n^{2\mu}(\mu \log n))$ time. At each level, the number of points (and, hence, the number of clusterings) goes down by a factor of $n^\mu/(2k) = n^{\mu-\kappa'}$, where $\kappa' = \log_n 2k$. If $\kappa' < \mu$ (note that both are constants), the number of points (and therefore the running time) decreases geometrically at each level, and so the total running time of SMALLER-SPACE is asymptotically the same as the time spent on the first level of clustering, namely $O(n^{1+\mu} \log n)$. μ is just ρ/σ, and, since the algorithm of Charikar and Guha uses only linear space, σ is 1. Therefore, the total running time is $O(n^{1+\rho} \log n)$.

We therefore claim the following theorem,

Theorem 10 SMALLER-SPACE *solves k–Median on a data stream of n points, in $O(\rho n^{1+\rho} \log n)$ time and $\Theta(n^\rho)$ space, up to an approximation factor of $2^{O(\frac{1}{\rho})}$.*

Proof: Assuming that κ' (and therefore κ as well) is at most $\mu/4 = \rho/3$, $L = \frac{\sigma-\rho}{\rho-\kappa'\sigma} = \frac{1-\rho}{\rho-\kappa'}$ is at most $\frac{3}{2}\frac{1-\rho}{\rho}$, which is $O(1/\rho)$, and the theorem follows. ■

We have shown how to reduce the number of levels by using a more space-efficient clustering subroutine. Next, we will show how to reduce the running time to $\tilde{O}(nk)$. This running time nearly matches the lower bounds for the running time of all constant-factor approximation algorithms (even non-streaming algorithms) for this problem.

4.3 Clustering Data Streams in $\tilde{O}(nk)$ Time

In this section we will show how to reduce the running time of SMALLER-SPACE from $\tilde{O}(n^{1+\rho})$ to $\tilde{O}(nk)$. We will make use of the bicriterion, $\tilde{O}(nk)$-time algorithm of Indyk[59] as a subroutine. We first give a description and analysis of that algorithm, and then explain how to incorporate it into the divide-and-conquer framework.

4.3.1 Indyk's Algorithm

The algorithm of Indyk, which we will call SAMPLE-OUTLIER, is randomized, and employs sampling in such a way that it cannot operate on a weighted point set. Figure 4.3 gives the algorithm. Later, we will show how SAMPLE-OUTLIER can be modified to accept weighted input, but first we will state the basic algorithm, which takes as input a point set Y of n_0 points. It makes use of the primal-dual algorithm of Jain and Vazirani [63] mentioned earlier.[2]

Algorithm SAMPLE-OUTLIER(Y)[59]

1. For i from 1 to $2\log n_0$, set $M_i \leftarrow$ S-O-SUBROUTINE(Y).
2. For each M_i, calculate $\Phi^Y(M_i)$.
3. Return the M_i whose value of Φ^Y is lowest.

Figure 4.3: Algorithm SAMPLE-OUTLIER

The algorithm S-O-SUBROUTINE, given in Figure 4.4, returns $2k$ medians.
A single iteration of S-O-SUBROUTINE first finds k medians for a random sample from Y, then identifies the worst outliers (the points of highest assignment distance) with respect to these medians, and finds k medians for them. SAMPLE-OUTLIER runs this algorithm $2\log n_0$ times and takes the best solution returned. Indyk proved that S-O-SUBROUTINE gives a constant-probability, bicriterion approximation:

[2]In fact, in the paper published by Indyk, which preceded the publication of the primal-dual algorithm[63], the clustering subroutine used was a bicriterion, $O(n^2k^2)$-time local search algorithm [70].

Algorithm S-O-SUBROUTINE(Y)

1. Draw a sample of size $s = \sqrt{n_0 k}$.
2. Find k medians from these s points using the primal dual algorithm. Call this set of medians L.
3. Assign each of the n_0 original points to its closest median.
4. Collect the n_0/s points with the largest assignment distance.
5. Find k medians from among these n_0/s points. Call this median set R.
6. Return $L \cup R$.

Figure 4.4: Algorithm S-O-SUBROUTINE

Theorem 11 *[59] With constant probability, the medians $L \cup R$ have $\Phi^Y(L \cup R) \leq \beta\text{OPT-KM}(Y, Y, k)$, where β is a constant.*

It follows that with high probability SAMPLE-OUTLIER produces a bicriterion approximation. The running time of the primal-dual algorithm is quadratic in the input size, so Step 2 of S-O-SUBROUTINE takes $O(n_0 k)$ time, and Step 5 takes $O((n_0/s)^2) = O(n/k)$ time. The total running time of S-O-SUBROUTINE is $O(n_0 k)$, and so SAMPLE-OUTLIER, which calls it $2 \log n_0$ times runs in $\tilde{O}(n_0 k)$ time. The space complexity of the primal-dual algorithm is also quadratic in its input size; SAMPLE-OUTLIER invokes the primal-dual algorithm on samples of $O(\sqrt{n_0 k})$ points, and can therefore cluster n_0 points in $O(n_0 k)$ space.

4.3.2 Extension to the Weighted Case

As mentioned earlier, SAMPLE-OUTLIER, as stated, cannot accept weighted input. It samples from the point set and solves k–Median (to an approximation) on the sample; if it does not sample according to the weights (i.e., pick each point with probability proportional to its weight), the samples may have very different optimal solutions than the original data set, since high-weight points may get excluded, and the approximation guarantees of Indyk's algorithm will fail. If the algorithm *does* sample according to weights in this way, it faces a different problem, however. The second phase, in which the $\sqrt{n_0/k}$ points of highest assignment distance are identified and clustered, can only work if the first phase, which clustered a sample of size $\sqrt{n_0 k}$, covered all but approximately $n_0/\sqrt{n_0 k}$ points at low assignment cost. Since the sample size is based on n_0 and not on the total weight of the point set, there may be many more than $n_0/\sqrt{n_0 k}$ outliers after the first phase. If the sample size were based on weight, so that the first clustering could cluster a high fraction (weight-wise) of the points at low cost, the space and running time requirements would no longer be guaranteed to be linear (or even polynomial) in the data size.

Some simple changes to SAMPLE-OUTLIER make it usable at higher levels of clustering, when the inputs are weighted. First, for some constant $\gamma > 0$, we round the weights of all data points up to the nearest power of $(1 + \gamma)$. For every power of $(1 + \gamma)$, the set of points whose weights have all been rounded to this power will be called a "weight class." We identify all the non-empty weight classes (note that there can be at most $\log_{1+\gamma} n$ since all the original points have unit weights and since the total weight of all points does not change from level to level). For each weight class, we run SAMPLE-OUTLIER, finding $2k$ medians for the class; then we run the primal-dual algorithm on the set of at most $2k \log_{1+\gamma} n$ resulting medians and save the k weighted medians returned by this algorithm. The running time increases only by an $O(\log n)$ factor.

With the modification so that it can take weighted input sets, SAMPLE-OUTLIER can be used at every level of clustering in SMALLER-SPACE except the last (which still requires the primal-dual algorithm or another algorithm that guarantees exactly k medians). At the first level of clustering, we will cluster n^ρ/k points at a time, so that when the quadratic-space primal-dual algorithm is applied to samples of size $\sqrt{(n^\rho/k)k} = \sqrt{n^\rho}$ it can operate in the n^ρ available space. The first level of clustering will therefore require $\frac{n}{n^\rho/k} = kn^{1-\rho}$ calls to SAMPLE-OUTLIER, each of which will require $O(n^\rho \log n)$ time, and so the total time needed to produce level-1 medians will be $O(kn^{1-\rho}n^\rho \log n) = \tilde{O}(nk)$. At each level, the total number of points clustered will be $2k/n^\rho$ times as many as in the previous level, and the rounding technique mentioned above will only increase the running time by an $O(\log n)$ factor, so the running time of each level will still be just $\tilde{O}(nk)$.

If we substitute the local-search algorithm of Charikar and Guha for all levels of clustering except the first (which can still be accomplished using SAMPLE-OUTLIER) and the last (for which the primal-dual algorithm can be used), SMALLER-SPACE still runs in $\tilde{O}(nk)$ total time. In this case, the number of points clustered in the second level of clustering is $2n\frac{k}{n^\mu}$, and so the second level of clustering consists of $2kn^{1-2\mu}$ clusterings of n^μ points each, each requiring $O(\mu n^{2\mu}\log n)$ time. The total time spent in clustering level-1 medians is therefore $O(nk\log n)$; since the running times decrease geometrically from level to level, the total running time of this implementation of SMALLER-SPACE is again asymptotically equal to that of the first two levels: $\tilde{O}(nk)$.

Both the above implementations have small values of σ and therefore a small, constant number of levels. Therefore, we have the following theorem:

Theorem 12 *The k–Median problem has a randomized, one-pass, constant-factor approximation algorithm running in $O(nk\log n)$ time, using n^ρ memory, if $\log_n k < \rho$.*

4.4 Lower Bounds

In this section we show that the running times of the algorithms presented in this chapter cannot be much further reduced; we also explore the question of derandomization. We will show that no deterministic algorithm could have a running time that is lower than $\tilde{O}(nk)$, the running time of the randomized algorithms presented here, by more than a polylogarithmic factor. We conclude that the running time of the randomized divide-and-conquer algorithm is nearly optimal among constant-factor approximation algorithms. We will also derandomize this algorithm and produce a single pass, small-memory k–Median algorithm that guarantees a *poly* $\log n$ approximation; it will follow that derandomization can be achieved at the cost of a higher approximation ratio.

We now show that any constant-factor deterministic approximation algorithm must calculate $\Omega(nk)$ interpoint distances (for example, by querying a distance matrix or other distance oracle, or computing a distance from the representations of points, if distances are implicit, as in the case of Euclidean distances among real-valued vectors). Each distance calculation will be assumed to take unit time, and so an $\Omega(nk)$ running time lower bound will follow.

We consider the family of point sets consisting of the sets that admit partitions into k groups such that every pair of points has distance 0 if both points are from the same group, and 1 otherwise. For all such point sets, the optimum k-Median solution has cost 0, and so, to guarantee a constant-factor approximation to k–Median, an algorithm must be guaranteed to find the optimum solution.

Solving k–Median on this family of instances is equivalent to the following Graph k-Partition Problem: Given an integer k and a complete k-partite graph G, find the k-partition of the vertices of G into independent sets. The equivalence can be easily realized as follows: The set of points $\{s_1, \ldots, s_n\}$ to be clustered naturally translates to the set of vertices $\{v_1, \ldots, v_n\}$ and there is an edge between v_i and v_j if and only if $\text{dist}(s_i, s_j) > 0$. Observe that a constant-factor-approximate solution can be computed with t queries to the distance function if and only if a graph k-partition can be computed with t queries to the adjacency matrix of G. Kavraki*et al.* [69] show that any deterministic algorithm that finds a Graph k-Partition requires $\Omega(nk)$ queries to the adjacency matrix of G. This result establishes a deterministic lower bound for k–Median.

Theorem 13 *A deterministic k–Median algorithm must make $\Omega(nk)$ queries to the distance function to achieve a constant-factor approximation.*

4.5 $\tilde{O}(nk)$-Time Deterministic Algorithms

One natural question is what can be achieved without randomization. We have already seen how to get an $O(n^{1+\rho})$-time clustering algorithm that uses n^ρ space and gives a constant-factor approximation. In order to reduce the running time of this algorithm to $\tilde{O}(nk)$, we would be restricted to cluster so few $((O(k))$ points at a time that the number of levels would increase to $\log n$ and the approximation factor to some polynomial in n. Therefore, a different approach is needed if we are to find a fast, deterministic algorithm.

Theorem 14 *There is a deterministic, one-pass algorithm, running in $\tilde{O}(nk)$ time and n^ρ space, that produces a $(poly \log n)$-approximation for the k–Median problem.*

Proof: First we will have to construct an algorithm that runs in time $\tilde{O}(nk)$. Then we can reduce the space required in the same way as for the previously described randomized algorithm.

Consider the primal-dual algorithm that gives a constant-factor (say c) approximation for the k–Median problem. This algorithm takes time (and space) an^2 for some constant a. Consider the following algorithm, which we will call A_1: partition the n original points into p_1 equal-size subsets, apply the primal-dual algorithm to each of these subsets, and then apply it to the p_1k weighted points so obtained, to get k final medians. If we choose $p_1 = (n/k)^{\frac{2}{3}}$, the running time of A_1 is $2an^{\frac{4}{3}}k^{\frac{2}{3}}$, and the space required is $2an^{\frac{4}{3}}k^{\frac{2}{3}}$ also. By Theorem 8 we have an approximation of $4c^2 + 4c$.

Now define A_2 to split the dataset into p_2 subsets and apply A_1 on each of them and on the resulting intermediate medians (notice we can easily ensure an implementation to get a one-pass algorithm). Solving to minimize the running time will yield $p_2 = (n/k)^{4/5}$. Therefore the running time and space required both become $4an^{\frac{16}{15}}k^{\frac{14}{15}}$.

If we continue this process so that A_i calls A_{i-1} on p_i subsets, we can prove without much difficulty that the running time and the space required by the algorithm will both be $a2^in^{\left(1+\frac{1}{2^{2^{i-1}}-1}\right)}k^{\left(1-\frac{1}{2^{2^{i-1}}-1}\right)}$. The approximation factor c_i is $4c_{i-1}^2 + 4c_{i-1}$.

To bring the exponent of n in the running time down to 1, it is sufficient to have $i = \Theta(\log\log\log n)$. This makes the running time nk (hiding poly $\log\log n$ factors) and gives approximation $O(\log^p n)$ since the approximation factor is 4^{2^i}. Thus we have a poly-$\log n$ approximation in $\tilde{O}(nk)$ space and time. Now we can use this in our previous algorithm to get an $O(\log^p n)$ approximation in n^ρ space and $\tilde{O}(nk)$ time, without using randomization. ■

The above actually shows that we have an $O(n^{1+\rho})$-time clustering with approximation guarantee polynomial in $\frac{1}{\rho}$. Combining this with Theorem 10 we get the following,

Theorem 15 *There is a $\tilde{O}(n^{1+\rho\delta})$-time, $\Theta(n^{\delta})$-space deterministic algorithm guaranteeing an approximation factor that is $O(poly(\frac{1}{\rho})2^{\frac{1}{\delta}})$.*

Chapter 5

Polylogarithmic-Space k–Median Algorithm

In Chapter 4 a randomized, constant-factor approximation algorithm for k–Median on data streams was presented, that required space polynomial in n, the number of points in the stream. In this section, we present a randomized algorithm for the same problem, that produces a constant-factor approximation in one pass using $O(k \log^2 n)$ storage space and $O(k^2 n \log^2 n)$ running time.

An algorithm for solving k–Median on data streams must maintain a small amount of state; for example, as it reads more points in the stream, the algorithm may maintain a succinct representation of a clustering of the section of the stream that has already been read. In Chapter 4, this succinct representation consisted of the existing medians, each accompanied by an integer weight giving the number of data stream points assigned to it. It also seems necessary for the algorithm to maintain an implicit *lower bound* on the optimal solution for the sequence seen so far. Thus it appears that any clustering objective function that admits such a clustering algorithm must have succinct lower bound certificates on the cost of the optimal solution. For the k–Center objective (minimize maximum cluster radius), such a succinct certificate follows directly from the analysis of the offline k–Center problem. The existence of such a certificate was the basis for the streaming algorithm for clustering with the k–Center objective in [17]. In the case of the sum of diameters objective [22], the succinct lower bound certificate is non-trivial; however, a lower bound certificate consisting of $O(k)$ carefully chosen points can be obtained from a careful analysis of the primal and dual LP formulations for this problem. Here we will show how to derive a succinct lower bound certificate for the k–Median problem, and ultimately how to find a constant-factor-approximate k–Median solution for an n-point stream, with high probability.

5.1 Summary of Algorithmic Method

The main algorithm, POLYLOG-SPACE, will use a subroutine ONLINE-FL (to be described shortly) that, given a point stream, finds a set of medians for an initial segment of that stream. POLYLOG-SPACE operates in phases, during each of which a further segment of the input stream is consumed. Assume that the instance on which POLYLOG-SPACE is invoked consists of the point stream X and the integer k. For convenience, each member of X will be thought of as having an associated weight equal to 1. In each phase i, POLYLOG-SPACE invokes ONLINE-FL on a modified version X_i of the original input stream X; for each median m found by ONLINE-FL for X_i, POLYLOG-SPACE calculates the number $w(m)$ of members of X_i assigned to m. $w(m)$ will be called the "weight" of median m. Each median found during phase i is then associated with its weight, and the weighted set of medians is stored as the "solution from phase i." For every i, the stream X_{i+1} input to phase $i+1$ is the solution from phase i, concatenated with the remaining unread points from X; in other words, X_{i+1} is just X, with the medians found in phase i replacing the initial segment of X that has been read before phase $i+1$. The first segment of X_{i+1} (the solution from phase i) consists of points with possibly non-unit weights; the second part has unit weights.

5.2 ONLINE-FL Subroutine

The subroutine ONLINE-FL invoked in each phase of POLYLOG-SPACE is Meyerson's online Facility Location algorithm [80]. The Facility Location problem, which was mentioned in Chapter 4 and which will be discussed more fully in Chapter 6, is a

Lagrangian relaxation of the k–Median problem. In Facility Location, in place of the restriction on the number of medians, the objective function charges an additive cost for each median, or "facility," that the solution contains. We first review Meyerson's algorithm (Figure 5.1) and recast the guarantee in a form that will be convenient for us.

Algorithm ONLINE-FL (point stream X, facility cost f)[80]

1. Make one pass over X performing the following steps for each $x \in X$:
   (a) Let δ be the distance of the current point x to the closest already-open facility.
   (b) With probability δ/f (or probability 1 if $\delta/f > 1$), open a new facility at x. Otherwise, the new point is assigned to the closest already-open facility.
2. Return the set of facilities.

Figure 5.1: Algorithm ONLINE-FL[80]

Recall that we will apply ONLINE-FL during each phase to a weighted point set. When a point x of weight w is processed, the probability of building a facility at x in Step 1b of the algorithm is $\min(w\delta/f, 1)$.

Consider an instance of k–Median (a stream X and integer k), and let $OPT =$ OPT-KM(X, X, k); recall that OPT-KM(S_1, S_2, k) is the minimum, over all $T \subseteq S_2$ with $|T| = k$, of $\sum_{x \in S_1} \min_{t \in T} \mathrm{dist}(x, t)$. Let L be a lower bound on OPT. (We will later explain how this lower bound is obtained). We will run the facility location algorithm with facility cost $f = \frac{L}{k(1+\log n)}$. The proof of the following lemma follows from the proof technique in [80].

Lemma 9 *The number of medians* ONLINE-FL*is expected to produce is at most* $k(1+\log n)(1 + 4OPT/L)$. *The expected cost is at most* $L + 4OPT$.

Proof: Let $c_1^*, c_2^*, \ldots, c_k^*$ be the medians in the optimal solution. Let d_p^* denote the distance from point p to the nearest median in the optimum solution. Let C_i^* denote the set of points assigned to median c_i^*, and let $A_i^* = \sum_{p \in C_i^*} \mathrm{dist}(p, c_i^*)$. Define a_i^* to be $A_i^*/|C_i^*|$. Focus on some optimum cluster C_i^*. For $j = 1, 2, \ldots, \log n$, let S_j be the set of points p in C_i^* for which $2^{j-1}a_i^* < d_p^* \leq 2^j a_i^*$ from the median c_i^*. (Note that S_j is empty for $j > \log n$). Now, consider the points in S_j. The expected service cost (sum of distances of points to their medians) paid before a median is opened at some point in S_j is at most f. Any subsequent point $p \in S_j$ has distance at most $3d_p^*$ to this median. The service cost paid for this subsequent point is bounded by $3d_p^*$. On the other hand, the probability that a median is opened at p is at most $\frac{3d_p^*}{f}$.

Now consider the set of points within a_i^* of the optimum median c_i^*. The expected (service) cost paid before a median is opened in this set is at most f. After a median is opened within this set, the distance of a subsequent point p from its nearest median is at most $a_i^* + d_p^*$. The service cost paid for this subsequent point is bounded by $a_i^* + d_p^*$. On the other hand, the probability that a median is opened at p is at most $(a_i^* + d_p^*)/f$.

We will bound separately the expected number of medians opened by the algorithm, and the expected (service) cost paid by the algorithm. The expected number of medians opened by the algorithm (amongst points in C_i^*) is bounded by $1 + \log n + \frac{1}{f}\left(\sum_{p \in C_i^*}(a_i^* + 3d_p^*)\right) \leq 1 + \log n + \frac{4A_i^*}{f}$. Summing this over all clusters $C_i^*, i = 1, \ldots, k$, we find that the total number of medians opened by the algorithm is bounded by $k(1 + \log n)(1 + 4OPT/L)$.

Now let us bound the expected service cost paid by the algorithm. The expected service cost paid for points in C_i^* is bounded by $f(1 + \log n) + 3\sum_{p \in C_i^*}(d_p^* + a_i^*) \leq \frac{L}{k}$ $+4A_i^*$. Summing up over all clusters shows that the expected service cost paid by the algorithm is at most $L + 4OPT$. ■

The following corollary of Lemma 9 follows by a simple application of the Markov inequality and the union bound.

Corollary 1 *With probability at least* $1/2$*, algorithm* ONLINE-FL *produces a solution of cost at most* $4(L + 4OPT)$ *and using at most* $4k(1 + \log n)(1 + \frac{4OPT}{L})$ *medians.*

5.3 Phases and phase transitions

We now present the details of the algorithm POLYLOG-SPACE. As mentioned earlier, the algorithm operates in a series of phases and marks points in the stream as "read" as they are processed. When we change phases, the algorithm might see a point, but not mark it as "read". We use the notion of "read" points to obtain a bound on the number of phases by ensuring that every phase makes progress, i.e., marks at least one new point as "read". For ease of exposition, we initially describe a simpler (although slightly incorrect) algorithm and analyze it assuming that there are at most n phases. In the next section, we will modify the algorithm to ensure that the number of phases is at most n. For now, we will simply denote the number of phases as p. The algorithm POLYLOG-SPACE is given in Figure 5.2; β is a constant to be explained later, and k is assumed to be less than $n = |X|$.

Algorithm POLYLOG-SPACE(point stream X, integers k and n)

1. $L_1 \leftarrow$ SET-LB$(X)/\beta$.
2. $i \leftarrow 1$; $X_1 \leftarrow X$.
3. While there are unread points in X:
   (a) $M_i \leftarrow$ PARALLEL-CLUSTER(L_i, X_i, k, n);
   (b) $X_{i+1} \leftarrow M_i||(X - R_i)$;
   (c) $L_{i+1} = \beta L_i$;
   (d) $i \leftarrow i + 1$;
4. Return M_{i-1}, the medians given by the most recent invocation of PARALLEL-CLUSTER.

Figure 5.2: Algorithm POLYLOG-SPACE

Algorithm SET-LB(point stream X, integer k)

1. Without marking any points of X as "read," let d equal the distance between the closest pair of the first $k + 1$ members of X in order;
2. return d;

Figure 5.3: Algorithm SET-LB

The first call made by POLYLOG-SPACE sets a lower bound using the following procedure SET-LB, given in Figure 5.3. In step 3a of POLYLOG-SPACE, PARALLEL-CLUSTER returns M_i, a set of (weighted) medians found for the part of X_i that it read. In step 3b, $X - R_i$ (to be defined more formally later) is the portion of X that is still unread at the end of phase i, and $M_i||(X - R_i)$ is the concatenation of M_i and $(X - R_i)$. The description of PARALLEL-CLUSTER is given in Figure 5.4. In this description, β and γ are constants that will be chosen to satisfy the following condition:

$$\gamma + 4(1 + 4(\beta + \gamma)) \leq \gamma\beta \tag{5.1}$$

Algorithm PARALLEL-CLUSTER(point stream S, lower bound L_i, integers k and n)

1. Run $2 \log n$ parallel invocations of ONLINE-FL with $f = L_i/(k(1+\log n))$ on the input S.
2. Each invocation is run as long as the number of medians does not exceed $4k(1 + \log n)(1 + 4(\gamma + \beta))$ and the cost of the solution (on the modified input) does not exceed $4L_i(1 + 4(\gamma + \beta))$. If either of these conditions is violated by a particular invocation, we stop running this invocation.
3. Continue until all invocations have been stopped.
4. When the last invocation finishes, mark as "read" all points seen by this invocation, except for the last (i.e., the point that would cause the median limit or the cost limit to be exceeded); return the medians found by this invocation.

Figure 5.4: Algorithm PARALLEL-CLUSTER

The ith phase of POLYLOG-SPACE consists of the ith call to the routine PARALLEL-CLUSTER, which consumes an initial segment of the modified stream X_i passed to it as a parameter. The result of this phase is that the members of X_i that are seen during this ith call to PARALLEL-CLUSTER (except for the last point to be seen) are marked as "read" so that the unread points can be appended untouched to the set of medians found by PARALLEL-CLUSTER for X_i.

We will next prove that the cost of the medians returned by POLYLOG-SPACE is at most a constant times the cost of the best k–Median solution for X, and that there are at most n phases. The following notation (some of which has already been used in the algorithm description) will be useful in these proofs.

Definition 19 *For $1 \leq i \leq p$, let R_i denote the subset of X that is marked as "read" during phases 1 through i. Let S_i denote the subset of X that is seen by the algorithm during phases 1 through i. Note that $R_i \subseteq S_i$, that in phases 1 through $p-1$, S_i contains only one more point than R_i (the point seen, but returned unread, by the latest-finishing invocation of* ONLINE-FL*), and that in phase p $R_i = S_i$.*

Definition 20 *If $1 \leq i < p$ let $OPT_i =$ OPT-KM(S_i, X, k). (Recall that S_i is the subset of X that is seen in phases 1 through i.) Note that the solution that attains the cost OPT_i on S_i may include medians not in S_i.*

Definition 21 *Let A_i denote the cost of the medians M_{i-1} on R_{i-1}, the points read (i.e., marked as "read") before phase i. Define $A_1 = 0$.*

Fact 1 *Each new point marked "read" by* POLYLOG-SPACE *adds at most f (the current value of facility cost when the point is read) to the solution cost.*

Proof: If for a given point the distance to the nearest existing median is greater than f, this point is made into a median at a cost of f. ■

Fact 2 *The value returned by* SET-LB *to* POLYLOG-SPACE *is a lower bound on* OPT-KM(X, X, k).

Proof: Among the first $k+1$ members of X, two must be assigned to the same cluster, and that cluster will contribute at least their interpoint distance to the overall k–Median cost. Therefore OPT_1 is at least the distance between the closest pair among these first $k+1$ points, as long as the first $k+1$ points are all read in phase 1, which they will be unless the cost threshold of $4(1+4(\gamma+\beta))L_1$ is exceeded before the $(k+1)$th point is read. No point ever contributes more than f to the total cost; therefore, the total cost before the $(k+1)$th point is read cannot exceed

$(k+1)f = (k+1)L_1/(k(1+\log n)) \leq L_1$, so the phase cannot end before the first $k+1$ points are read. ■

Lemma 10 *Let $P(i)$ denote the event that $A_i \leq \gamma L_i$, and let $Q(i)$ denote the event that $L_{i+1} = \beta L_i \leq OPT_i$. For appropriate choice of β and γ, with probability* $1 - \frac{1}{poly(n)}$*, for all* $1 \leq i < p$*, $P(i)$ and $Q(i)$ hold.*

Proof: First we will show $P(1)$, $P(2)$, and $Q(1)$. We will then show that if $P(i')$ holds for all $i' \leq i$ and $Q(i')$ holds for all $i' < i$, then $P(i+1)$ holds, and with probability at least $1 - 1/(n^2)$ $Q(i)$ holds as well. Since p, the number of phases of POLYLOG-SPACE, is at most n, it will follow that with probability at least $1 - 1/n$ $P(p-1)$ and $Q(p-1)$ both hold.

$P(1)$ holds trivially as long as $\gamma \geq 0$, because $A_1 = 0$. $P(2)$ holds since $4(1 + 4(\gamma+\beta)) \leq \gamma\beta$ (this follows from the condition (5.1)), because A_2 is guaranteed not to exceed $4(1+4(\gamma+\beta))L_1$, and $L_2 = \beta L_1$. Finally, $Q(1)$ is also true, because L_1 is set to $1/\beta$ times a lower bound on OPT-KM(S_1, X, k).

We will next prove that, with high probability, P and Q hold for all phases except the last. Let $i < p$ be such that $\forall i' \leq i$, $P(i')$, and $\forall i' < i$, $Q(i')$. We will show that with high probability $Q(i)$ holds, and that, if $i+1 \leq p-1$, $P(i+1)$ holds. Suppose for contradiction that $OPT_i < \beta L_i$. Let $OPT'_i =$ OPT-KM$(M_{i-1} \cup (S_i - S_{i-1}), X, k)$, that is, the cost of the k members of X whose cost on the modified input seen during phase i is optimal. Note also that the solution that attains a cost of OPT_i on $M_{i-1} \cup (S_i - S_{i-1})$ could use medians outside the set of points seen in phases 1 through i. Then $OPT'_i \leq A_i + OPT_i$ (the proof is similar to the proof of Theorem7 in 4), and so $OPT'_i \leq (\beta+\gamma)L_i$. Therefore, the modified instance seen in phase i must have a feasible solution costing $(\beta+\gamma)L_i$, and so, by Corollary 1, each invocation has a probability of at least 1/2 of *reading* the last member of S_i before overrunning the limits on cost or number of medians. However, since all invocations stop without reading this last point, either we have a contradiction (and so $OPT_i \geq \beta L_i$) or else a probability-$1/n^2$ event has occurred.

We also need to ensure that $A_{i+1} \leq \gamma L_{i+1}$ if $i+1 \leq p-1$. Let C_i denote the cost of M_i on $M_{i-1} \cup (R_i - R_{i-1})$, the modified input read during phase i. By definition of the algorithm, $C_i \leq 4(1+4(\beta+\gamma))L_i$. Note that the cost of M_{i-1} with respect to R_{i-1}, the points marked as read before phase i, is $A_i \leq \gamma L_i$. Then A_{i+1}, the cost of the solution that the algorithm has at the *end* of phase i (measured with respect to the original input points R_i), is at most $A_i + C_i$. However, since $\gamma + 4(1+4(\beta+\gamma)) \leq \gamma\beta$ (condition (5.1)), it follows that $A_i + C_i \leq \gamma L_{i+1}$, i.e., , $P(i+1)$ holds.

$Q(p-1)$ holds unless for some phase $i < p$ $Q(i)$ fails. We have seen that for each phase i before the last, if $Q(i-1)$ holds, this failure has probability at most $1/n^2$. In the next section, we will show there are at most n phases (by modifying the algorithm slightly). Since there are at most n phases, the probability that $Q(p-1)$ fails is at most $1/n$. ■

If $Q(p-1)$ holds, then $L_{p-1} \leq OPT_{p-1}/\beta$. OPT_{p-1} is certainly no more than OPT_p, and so $L_p = \beta L_{p-1} \leq OPT_p$. Since the last phase finds a solution costing at most $4(1+4(\gamma+\beta))L_p$, the medians produced by phase p cost at most $4(1+4(\gamma+\beta))OPT_p$.

5.4 Bounding the number of phases

The analysis in the previous section, used the fact that there are n phases, but this may not hold for the algorithm as presented since several phase changes may occur without the consumption of new input. To remedy this problem, at each phase change, we compute an improved lower bound on the optimal by taking the solution from the previous phase and the next new point. This will ensure that in every phase, at least one new point is marked "read".

Let x denote the point in the data stream that is seen, but not marked "read" in phase i. Recall that OPT_i is the value of the optimal solution on $R_i \cup \{x\}$. We will compute an improved lower bound L_{i+1} so as to ensure that $OPT_i \leq \beta L_{i+1}$. This in turn will ensure that, unless a failure occurs, phase $i+1$ marks x as "read" before it terminates. This is because at the termination of phase $i+1$, we have that (with very high probability) the value of the optimal solution on the points seen so far exceeds βL_{i+1}.

We compute an approximate k–Median solution to the modified instance consisting of M_i (the weighted medians from phase i) and x. In order to do this, we use a c-approximation algorithm for k–Median. Let APX_i denote the value of the c-approximate solution obtained. We set L_{i+1} as follows:

$$\begin{aligned} L'_{i+1} &= \frac{APX_i}{2c(1+\gamma)} \\ L_{i+1} &= \max(\beta L_i, L'_{i+1}) \end{aligned}$$

In the previous section, we proved that the invariant $A_{i+1} \leq \gamma L_{i+1}$ is maintained by the algorithm. Note that the new procedure for setting L_{i+1} does not violate this invariant. Furthermore, examining the proof in the previous section, it is easy to see that the algorithm maintains the stronger invariant $A_{i+1} \leq \beta\gamma L_i$. We will also need the following condition on β and γ:

$$\beta \geq 2c(1+\gamma) + \gamma \tag{5.2}$$

Note that this does not contradict the condition (5.1) we have already imposed on β and γ. We first prove that L_{i+1} is a valid lower bound on the optimal cost.

Lemma 11 $L_{i+1} \leq OPT_i$.

Proof: We know that $OPT_i \geq \beta L_i$. We need to prove that $OPT_i \geq L'_{i+1}$. Recall that OPT_i is the value of the optimal k–Median solution on $R_i \cup \{x\}$. (This could use medians outside $R_i \cup \{x\}$). Also recall that A_{i+1} is the cost of assigning the points in R_i to the medians in M_i. This implies that there is a k–Median solution for the weighted points in M_i together with x, with cost at most $OPT_i + A_{i+1} \leq OPT_i + \beta\gamma L_i$. (Again, this could use medians outside $M_i \cup \{x\}$). Further, we can transform this solution to a solution that uses medians only in $M_i \cup \{x\}$, with cost at most $2(OPT_i + \beta\gamma L_i)$. Since we use a c-approximation algorithm to compute a k–Median solution on this instance, $APX_i \leq 2c(OPT_i + \beta\gamma L_i)$.

$$\begin{aligned} L'_{i+1} &= \frac{APX_i}{2c(1+\gamma)} \leq \frac{2c(OPT_i + \beta\gamma L_i)}{2c(1+\gamma)} \\ &= \frac{1}{1+\gamma} OPT_i + \frac{\gamma}{1+\gamma} \beta L_i \\ &\leq OPT_i \end{aligned}$$

The last inequality follows since $\beta L_i \leq OPT_i$. ■

Next, we prove that that OPT_i is upper-bounded by βL_{i+1} (which ensures progress in each phase). First, we state and prove a bound we will need in the proof.

Claim 1 $APX_i \geq OPT_i - A_{i+1}$.

Proof: The statement of the claim is equivalent to $APX_i + A_{i+1} \geq OPT_i$. Note that APX_i is the cost of a k–Median solution on the weighted medians in M_i and the point x. A_{i+1} is the cost of assigning the points in R_i to the medians in M_i. By combining these two solutions, we get a feasible k–Median solution for the points in $R_i \cup \{x\}$ with cost at most $APX_i + A_{i+1}$. Since OPT_i is the cost of the optimal solution for this instance, it follows that $APX_i + A_{i+1} \geq OPT_i$. ■

Lemma 12 $OPT_i \leq \beta L_{i+1}$.

Proof:

$$\begin{aligned} L'_{i+1} &= \frac{APX_i}{2c(1+\gamma)} \geq \frac{OPT_i - A_{i+1}}{2c(1+\gamma)} \\ 2c(1+\gamma)L'_{i+1} + A_{i+1} &\geq OPT_i \\ (2c(1+\gamma)+\gamma)L_{i+1} &\geq OPT_i \\ \beta L_{i+1} &\geq OPT_i \end{aligned}$$

The last step follows because β and γ are chosen so that (5.2) holds. ■

5.5 The final medians

In order to produce the final medians, we simply solve a k–Median problem on the set of $O(k \log n)$ medians found by the algorithm POLYLOG-SPACE, treating the medians as points weighted by the number of points assigned to them. Suppose POLYLOG-SPACE produces $k \log n$ medians M with $\Phi^X(M) \leq \alpha$OPT-KM(X, X, k). (α is a constant). Then, by Theorem 7 in Chapter4, OPT-KM$(M, M, k) \leq 2(1+\alpha)OPT$ (here M is the weighted demand set and also the feasible set). If we use a c-approximation algorithm for this k–Median instance, we get a set M_f of k final medians with $\Phi^M(M_f) \leq 2c(1+\alpha)$OPT-KM$(M, M, k)$ and $\Phi^X(M_f) \leq [(2c+1)(1+\alpha)-1]$OPT-KM$(X, X, k)$, by Theorem8 from Chapter4.

The running time of the entire algorithm is dominated by the time taken by POLYLOG-SPACE, since the final medians are computed from a small set of medians, namely the $k \log n$ medians produced by POLYLOG-SPACE. In POLYLOG-SPACE, each phase (except possibly the last) begins with a set of operations performed on the solution from the previous phase, and then includes operations on unread points. We will bound the total running time of POLYLOG-SPACE by treating separately the operations on existing solutions and the operations on unread points.

Consider any member x of X; suppose x is read during phase i. Then for the phases up to and including $i-2$, there are no distances computed to x. In phase $i-1$ there are $O(k \log^2 n)$ distance computations involving x, since each of the parallel invocations may compute the distance of x to each of their existing $O(k \log n)$ medians; then, again, in phase i, x will participate in $O(k \log^2 n)$ distance computations. Therefore, the processing of unread points takes $O(kn \log^2 n)$ time. In each phase, the processing of the solution from the previous phase takes $O(k^2 \log^2 n)$ time, and since there are at most n phases, the running time of the algorithm POLYLOG-SPACE is $O(k^2 n \log^2 n)$.

Theorem 16 *The streaming algorithm for k–Median produces a constant approximation, in one pass, with probability $1 - \frac{1}{poly(n)}$; the space required is at most the space occupied by $O(k \log^2 n)$ points from the stream, and the running time is $O(k^2 n \log^2 n)$.*

Chapter 6

A Fast Heuristic for k–Median on Streams

In this chapter we adapt some of the algorithms discussed in earlier chapters and produce a fast, easily implemented k–Median heuristic, giving some theoretical guarantees on its performance. We give empirical evidence of the its superiority over the commonly-used k–Means algorithm. We then use this new algorithm, LSEARCH, as a subroutine in the divide-and-conquer framework defined in Chapter 4, with two levels of clustering. and experimentally demonstrate the superior performance of this combination as a k–Median algorithm for streaming data.

The algorithm LSEARCH uses the idea, introduced by Guha *et al.* [48][1], of obtaining the k–Median solution via a Facility Location subroutine, in that case, the Charikar-Guha local search algorithm. In Section 6.2 we will recapitulate the Facility Location objective, as we will be using a slightly different type of distance function that is commonly used in practice. Section 6.2 also describes new and critical conceptualizations that characterize which instances of Facility Location we will solve to obtain a k–Median solution, and also what subset of feasible facilities is sufficient to obtain an approximate solution. LSEARCH, which is itself based on the Charikar-Guha algorithm, is also detailed in this section; we will first review the Charikar-Guha algorithm and then give the details of (and the motivation for) the ways in which

[1]This technique is used in Chapter 4.

LSEARCH adapts that algorithm.

In Section 6.3, we show the results of some experiments that measure the empirical performance of the LSEARCH heuristic on a variety of synthetic and real data. The results lend support to the heuristic improvements discussed in Section 6.2. In Section 6.3.1, we show the results of an extensive series of tests comparing LSEARCH against the popularly used k–Means algorithm; LSEARCH outperformed k–Means on nearly all the data sets, which included both low- and high-dimensional data. The results presented in this section uncover an interesting trade-off between the cluster quality and the running time. Since both algorithms are randomized, we ran each one several times on each data set; over the course of multiple runs, there was a large variance in the performance of k–Means, whereas LSEARCH was consistently good. LSEARCH took longer to run for each trial but for most data sets found a near-optimal answer before k–Means found an equally good solution. On many data sets k–Means never found a good solution.

Finally, in Section 6.3.2 we empirically compare the divide-and-conquer method to the popular BIRCH [106] streaming algorithm. BIRCH is similar to the divide-and-conquer framework in that it summarizes a stream of points as a set of "pre-clusters," so that a clustering algorithm (such as k–Means, LSEARCH, or hierarchical agglomerative clustering, or HAC) can be applied for a final set of clusters. We show the results of running BIRCH with HAC, BIRCH with k–Means, and BIRCH with LSEARCH, as well as divide-and-conquer applications of k–Means and of LSEARCH. We discover a tradeoff between cluster quality and running time similar to that revealed by the earlier set of experiments. On both the synthetic and the real streams, running k–Means or LSEARCH in the divide-and-conquer manner produced higher-quality results than running the same algorithm with BIRCH, and the divide-and-conquer LSEARCH technique did better than divide-and-conquer k–Means, but BIRCH ran faster than the divide-and-conquer methods.

6.1 Preliminaries

Among heuristics for k–Median-like problems, k–Means is by far the most popularly used. In its most popular incarnation, the k–Means algorithm accepts input from d-dimensional real space, and uses the Euclidean distance function, which we will denote $\textsc{euc}(\cdot,\cdot)$. The objective function that k–Means aims to minimize is the sum of squared assignment distances, although it is guaranteed only to find a local optimum, not a global optimum of this objective. We will call the k–Means objective function SSQ, and we define it presently.

Definition 22 *Given a finite set Y of vectors from $\mathbb{R}^d$, the* SSQ *of a finite set $M \subseteq \mathbb{R}^d$ on Y is $\textsc{ssq}^Y(M) = \sum_{x\in Y} \textsc{euc}^2(x, \textsc{asg}(x, M))$.*

The *SSQ minimization problem*, for which k–Means is a heuristic, is to find a set $M \subseteq \mathbb{R}^d$, with $|M| = k$, for which $\textsc{ssq}^Y(M)$ is minimal. This problem is simply the continuous k–Median problem in Euclidean space, except that instead of a distance metric we use the squared Euclidean distance. The squared Euclidean distance, while not a metric, is very similar to a metric, in that it obeys a "relaxed triangle inequality," in the following sense: For all points a, b, $c \in \mathbb{R}^d$, $\textsc{euc}^2(a,c) \leq 2(\textsc{euc}^2(a,b) + \textsc{euc}^2(b,c))$. In other words, while the squared distance from a to c is not bounded by the sum of the squared distances from a to b and from b to c, it does not exceed their sum by more than a factor of 2. For this reason, many k–Median approximation algorithms (including the algorithm LSEARCH which we will present shortly) are also approximation algorithms for SSQ minimization. While LSEARCH is a general k–Median algorithm that would operate in a general metric space, the experiments will compare it against algorithms that assume real-valued vectors, one of which is designed to minimize SSQ, and so for the experiments we will evaluate solution quality using SSQ.

We now turn to Facility Location, which is the same problem as k–Median except that instead of restricting the number of medians to be at most k we simply impose an additive cost for each median, or *facility*. The cost associated with adding a facility is called the *facility cost*. Assume that we are given a facility cost f and a set Y of n

points with a metric (or relaxed metric) $\mathrm{dist}(\cdot,\cdot)$. Then the Facility Location cost of a set $M \subseteq Y$ of medians (or facilities) is given by $\Lambda^Y(M,f) = f|M| + \sum_{x\in Y} \textsc{ad}(x,M)$, where $\textsc{ad}(x,M)$, as defined before, is the distance, under the metric dist, from x to the nearest member of M, the facility to which x is assigned. We will define $\textsc{opt-fl}(Y,f) = \min_{M\subseteq Y} \Lambda^Y(M,f)$. The Facility Location problem is to find $M \subseteq N$ (where N is the domain of the metric space $\mathcal{X}$ with distance dist) that realizes $\textsc{opt-fl}(Y,f)$ and hence minimizes $\Lambda^Y(M,f)$. Note that as in the case of k–Median there may be more than one optimal solution, and in this case two optimal solutions may have different numbers of facilities. Facility Location is known to be NP-hard, and several theoretical approximation algorithms are known [63, 18].[2]

6.2 LSEARCH Algorithm

Here we will describe our new algorithm for k–Median, which is based on the concept of local search, in which we start with an initial solution and then refine it by making local improvements. Throughout this section, we will refer to the "cost" of a set of medians, meaning the cost according to Λ.

Previous Work on Facility Location

We begin by describing the Charikar-Guha Facility Location algorithm [18] [3]. We will refer to this algorithm as CG; it accepts as input a set Y of n points with distance function $\mathrm{dist}(\cdot,\cdot)$ that is a metric or relaxed metric, and a facility cost f.

We begin by defining the subroutine GAIN. Assume that we have a feasible solution to Facility Location on Y given $\mathrm{dist}(\cdot,\cdot)$ and f. That is, we have some set $I \subseteq Y$

[2]The variant of Facility Location described here is technically called *uniform-cost Facility Location*; in general, each potential facility could have its own facility cost. In the general Facility Location problem, each point could have a different facility cost, but we will only consider the uniform version, in which the facility cost is the same for all points. Furthermore, we have only considered instances in which the feasible set of facilities is the same as the demand set Y; in general these could be different, but for our purposes, we will only consider solutions in which the facility set is a subset of the set of demand points.

[3]Charikar and Guha showed that this algorithm will achieve a $(1+\sqrt{2})$-approximation to Facility Location.

of currently open facilities, and an assignment for each point in Y to some (not necessarily the closest) open facility. For every $x \in Y$ we define the *gain* of x, GAIN(x), to be the cost we would save (or further expend) if we were to open a facility at x, if one does not already exist, and then perform all advantageous reassignments and facility closings, subject to the following two constraints: first, that points cannot be reassigned except to x, and second, that a facility with member points can be closed only if its members are first reassigned to x. If there is already a facility at x, the computation is the same except that it does not include the cost of opening a facility at x. GAIN(x) can be computed in $O(|Y|)$ time. A $\Theta(|Y|)$-time procedure that calculates the cost that would be saved or expended by the opening of x, and that also performs the advantageous openings, closings, and reassignments if opening x as a facility is found to be advantageous, can be found in Appendix A. The algorithm CG is is given in Figure 6.1. The input set is Y with $|Y| = n$.

Algorithm CG(data set Y, facility cost f)

1. Obtain an initial solution (I, a) ($I \subseteq Y$ of facilities, a an assignment function) that gives an n-approximation to Facility Location on Y with facility cost f.
2. Repeat $\Omega(\log n)$ times:
   - Randomly order Y.
   - For each x in this random order: calculate GAIN(x), and if GAIN(x) > 0, add a facility there and perform the allowed reassignments and closures.

Figure 6.1: Algorithm CG

Charikar and Guha [18] describe a simple algorithm that obtains an n-approximate initial solution that could be used in Step 1. The algorithm CG will run in $O(n^2 \log n + T(n))$ time, where $T(n)$ is the running time of the initial-solution algorithm.

The algorithm CG does not directly solve k–Median but could be used as a subroutine to a k–Median algorithm, as follows. (This method was mentioned briefly in Chapter 4. We first set an initial range for the facility cost f (between 0 and an easy-to-calculate upper bound); we then perform a binary search within this range to find a value of f that gives us the desired number k of facilities; for each value of f that we try, we call Algorithm CG to get a solution.

Binary Search Two questions spring to mind, however: first, will such a binary search technique work, and second, will this algorithm, which uses Algorithm CG as a subroutine, find good solutions quickly? The following theorem justifies a binary search.

Theorem 17 *Assume we have a set Y of points with a metric dist : $Y \times Y \to \mathbb{R}^+ \cup \{0\}$. Then given facility costs f and f' with $0 \le f < f'$, a set M_1 of k facilities (or medians) that is optimal for f (i.e., $\Lambda^Y(M,f) = \text{OPT-FL}(Y,f)$), and a set M' of k' facilities that is optimal for f', the following are true:*

1. $k \ge k'$
2. $\Phi^Y(M) \le \Phi^Y(M')$
3. $fk + \Phi^Y(M) < f'k' + \Phi^Y(M')$
4. *If* $f \ne 0$, $\Phi^Y(M) = \Phi^Y(M')$ *if and only if* $k = k'$.

Recall that $\Phi^Y(M)$ is the sum of assignment distances, under the metric dist, from the demand set Y to the median (or facility) set Y. If dist were be replaced with EUC^2, the squared Euclidean distance, Φ would replace SSQ.

Proof: Let $a = \Phi^Y(M)$, $a' = \Phi^Y(M')$. Note $a \ge 0$ and and $a' \ge 0$. Then $\Lambda^Y(M,f) = fk + a$, $\Lambda^Y(M') = fk' + a'$, $\Lambda^Y(M,f') = f'k + a$, and $\Lambda^Y(M',f') = f'k' + a'$. By the optimality of M for f we have (I): $fk + a \le fk' + a'$, and, by the optimality of M' for f' we have (II): $f'k + a \ge f'k' + a'$. Then we have $f(k-k') \le a' - a \le f'(k-k')$. Since $f < f'$, it must be that $k \ge k'$, proving the first assertion. Since $k \ge k'$, we can see by (I) that $a \le a'$, proving the second. Combining (I) and the fact that $f < f'$, we have that $fk + a \le f'k' + a'$, proving the third.

Suppose $f \ne 0$. If $a = a'$ then we can combine (I) and (II) to show that $k = k'$. If $k = k'$ (note $1 \le k \le n$), then we can again combine (I) and (II) to show that $a = a'$. Therefore, the fourth assertion follows: given $f \ne 0$, $a = a'$ if and only if $k = k'$. ■

For a given Facility Location instance, there may exist a k such that there is no facility cost for which an optimal solution has exactly k medians, but if the data set is "naturally k-clusterable," then our algorithm should find k centers. In particular, if having k rather than $k-1$ medians allows the optimal assignment cost to decrease by a factor of k, then an optimum solution with exactly k medians must exist.

Theorem 18 *Given a data set Y, let A_i denote the best assignment cost achievable for an instance of Facility Location on Y if we allow i medians. If Y has the property that $A_k \leq \frac{A_j}{k-j+1}$ for all $j < k$ then there is a Facility Location solution with k centers.*

Proof: For a Facility Location solution to yield k centers, the solution for $j < k$ centers (and for $l > k$ centers) must have worse total cost than the Facility Location solution for k centers, i.e., $fj + A_j \geq fk + A_k$ and $fl + A_l \geq fk + A_k$. In other words, when $\frac{A_j - A_k}{k-j} \leq f \leq \frac{A_k - A_l}{l-k}$, $\frac{A_k - A_l}{l-k}$ is at most A_k. So in order for some facility cost to exist we need $A_k \leq \frac{A_j - A_k}{k-j}$ for all $j > k$. The result follows. ■

For example, if we realize an improvement going from $k-1$ to k centers of the form $A_k \leq \frac{A_{k-1}}{k}$ then an optimal solution with exactly k centers exists.

The next question to answer is whether this method of calling CGas a subroutine of a binary search is fast enough. The answer is that the algorithm's $\Theta(n^2 \log n)$ running time is prohibitive for large data streams. Therefore, we describe a new local search algorithm that relies on the correctness of the above algorithm but avoids the super-quadratic running time by taking advantage of the structure of local search in certain ways.

Finding a Good Initial Solution On each iteration of step 2 of Algorithm CG, we expect the total solution cost to decrease by some constant fraction of the way to the best achievable cost [18]; if our initial solution is a constant-factor approximation rather than an n-approximation (as used by Charikar and Guha), we can reduce our number of iterations from $\Theta(\log n)$ to $\Theta(1)$. We will therefore use Meyerson's online Facility Location algorithm, given in Section 5.2, for our initial solution. Here we will call this algorithm INITIALSOLUTION; recall that it takes a data set Y and a facility cost f as arguments, and that it takes each point of Y in an online fashion, making each point a new facility with probability δ/f, where δ is the distance to the nearest open facility.This algorithm runs in time proportional to n times the number of facilities it opens and obtains an expected 8-approximation to optimum [80].

Sampling to Obtain Feasible Centers Next we present a theorem that will motivate a new way of looking at local search. It is stated and proved in terms of the actual k–Median problem, but holds, with slightly different constants, for SSQ minimization. Assume the points $M^* = \{M_1^*, \ldots, M_k^*\}$ constitute an optimal solution to the k–Median problem for the data set Y (so $\Phi^Y(\{M_1^*, \ldots, M_k^*\}) = \text{OPT-KM}(Y, Y, k)$), that C_i is the set of points in Y assigned to M_i^*, and that r_i is the average distance from a point in C_i to M_i^* for $1 \le i \le k$. Assume also that, for $1 \le i \le k$, $|C_i|/|Y| \ge p$. Let $0 < \delta < 1$ be a constant and let S be a set of $s = \frac{8}{p} \log \frac{2k}{\delta}$ points drawn independently and uniformly at random from Y.

Theorem 19 *There is a constant α such that with high probability, the optimum k–Median solution in which medians are constrained to be from S has cost at most α times the cost of the optimum unconstrained k–Median solution (where medians can be arbitrary points in Y).*

Proof: If $|S| = s = \frac{8}{p} \log \frac{2k}{\delta}$ then $\forall i$, $Pr\{|S \cap C_i| < sp/2\} < \frac{\delta}{2k}$, by Chernoff bounds. Then $Pr\{\exists i : |S \cap C_i| < sp/2\} < \frac{\delta}{2}$. Given that $|S \cap C_i| \ge sp/2$, the the probability that no point from S is within a distance $2r_i$ of M_i^* is at most $(\frac{1}{2})^{sp/2} < (\frac{1}{2})^{\log \frac{2k}{\delta}} = \frac{\delta}{2k}$ by Markov's Inequality. So $Pr\{\exists M_i^* : \forall x \in S, \text{dist}(x, M_i^*) > 2r_i\} \le \frac{\delta}{2}$. Suppose that for each cluster C_i, our sample contains a point x_i within $2r_i$ of M_i^*. Then consider an arbitrary $y\ inS$; if $\text{ASG}(y, M^*) = M_i^*$, $\text{dist}(y, x_i) \le \text{dist}(y, M_i^*) + \text{dist}(M_i^*, x_i) = \text{AD}(y, M^*) + \text{dist}(M_i^*, x_i) \le \text{AD}(y, M^*) + 2r_i$. Then

$$\begin{aligned}
\sum_{y \in S} \text{AD}(y, \{x_1, \ldots, x_k\}) &\le \sum_{y \in S} (\text{AD}(y, M^*) + \text{dist}(M_i^*, x_i)) \\
&\le \sum_{y \in S} (\text{AD}(y, M^*) + \sum_{i=1}^{k} 2r_i |C_i| \\
&= 3\Phi^S(M^*).
\end{aligned}$$

■

In some sense we are assuming that the smallest cluster is not too small. If, for example, the smallest cluster contains just one point, i.e., $p = 1/|Y|$, then it is clear that no point can be overlooked as a feasible center. We view such a small subset of points as a set of outliers, not a clusters, and we will assume that outliers have been removed. Therefore if instead of evaluating GAIN() for every point x we only evaluate it on a randomly chosen set of $\Theta(\frac{1}{p}\log k)$ points, we are still likely to choose good medians and will finish our computation sooner.

Finding a Good Facility Cost Faster Third, we recall that each successive set of GAIN() operations tends to bring us closer to the best achievable cost by a constant fraction. The total gain achieved during one set of GAIN() computations (and the associated additions and deletions of medians) should then be smaller than the previous total gain by a constant fraction, in an absolute sense. Assume that, for the current value of f, the cost is changing very little, but the current solution has far from k medians; we can abort and change the value of f, knowing that it is very unlikely that the best possible solution for this f would have k medians. We can also stop trying to improve the solution when we get to a solution with k medians and find that the solution cost is improving by only a small amount after each new set of GAIN() operations. If the current number of medians is far from k, the definition of "small improvement" will probably be much broader than if we already have exactly k medians. In the first case we are simply observing that the current value of f is probably quite far from the one we will eventually want to arrive at, and we are narrowing the interval of possible f values in which we will search. In the second case, we have arrived at a good solution, but, because the number of medians is unlikely to change for this value of f, we want to get as good a solution as possible without wasting many iterations improving by an insignificant fraction.

Therefore, we will give the Facility Location subroutine FL, given in Figure 6.2, that our k–Median algorithm will call; FL will take a parameter $\epsilon \in \mathbb{R}^+$ that controls how soon it stops trying to improve its solution. The other parameters will be the data set Y of size n, a subset L of Y listing the feasible medians, *i.e.*, the members of

Y that can be made facilities, the metric or relaxed metric dist$(\cdot,\cdot)$, the facility cost f, and a current solution (F, g) where $F \subseteq Y$ is a set of facilities and $g : Y \to I$ is an assignment function.

Algorithm FL(Y, L, dist$(\cdot,\cdot)$, f, ϵ, (F,g))

1. Begin with (F.g) as the current solution
2. Let C be the cost of the current solution on Y. Consider the feasible medians (members of L) in random order; for each feasible median y, if GAIN$(y) > 0$, perform all advantageous closures and reassignments (see GAIN description), to obtain a new solution (F', g') [g' should assign each point to its closest median in F']
3. Let C' be the cost of the new solution; if $C' \leq (1-\epsilon)C$, return to step 2; else output the solution

Figure 6.2: Algorithm FL

The k–Median algorithm LSEARCHis given in Figure 6.3. The "trivial upper bound" U to which f_{max} is initially set (in step 2 of LSEARCH) is, for an arbitrarily chosen point $x_0 \in Y$, $\sum_{x \in Y} \mathrm{dist}(x, x_0)$. If $f = U$ then there is a feasible solution of cost exactly $2f$; no solution with more medians could possibly be better. Instances of Facility Location on Y with $f > U$ will not admit solutions with more than two medians, and so the binary search need not consider higher values of f. The running time of LSEARCH is $O(n|I| + nk \log k)$. I depends on the properties of the data set but is usually small, so this running time is a significant improvement over previous algorithms.

6.3 Experiments

6.3.1 Empirical Evaluation of LSEARCH

We present the results of experiments comparing the performance of k–Means and LSEARCH. All experiments were conducted on a Sun Ultra 2 with two, 200MHz

Algorithm LSEARCH $(Y, \mathrm{dist}(\cdot,\cdot), k, \epsilon, \epsilon', \epsilon'')$

1. $f_{min} \leftarrow 0$.
2. $f_{max} \leftarrow$ trivial upper bound U.
3. $f \leftarrow (f_{max} + f_{min})/2$.
4. $(I, a) \leftarrow \mathrm{InitialSolution}(Y,\ f)$.
5. Randomly pick $\Theta(\frac{1}{p}\log k)$ points as feasible medians, and L be the set of these feasible medians.
6. While the number of medians is not k and $f_{min} < (1-\epsilon'')f_{max}$:
   (a) Let (F, g) be the current solution.
   (b) Run $\textsc{fl}(Y, L, d, \epsilon, (F, g))$ to obtain a new solution (F', g').
   (c) If $k-1 \leq |F'| \leq k+1$ or $0.8k \leq |F'| \leq 1.2k$, then $(F', g') \leftarrow \textsc{fl}(Y, L, d, \epsilon', (F', g'))$.
   (d) If $|F'| > k$
      - then $f_{min} \leftarrow f$ and $f \leftarrow (f_{max} + f_{min})/2$;
      - else if $|F'| > k$ then $f_{max} \leftarrow f$ and $f \leftarrow (f_{max} + f_{min})/2$.
7. To simulate a continuous space, move each cluster median to the center of mass for its cluster.
8. Return our solution (F', g').

Figure 6.3: Algorithm LSEARCH

processors, 256 MB of RAM, and 1.23 GB of swap space,[4] running SunOS 5.6. First we will describe the precise implementations of k–Means used in these experiments. Then we will compare the behavior of k–Means and LSEARCH on a variety of synthetic datasets.

Figure 6.4 gives the k–Means algorithm. The k initial medians are usually chosen from some distribution over either $\mathbb{R}^d$ or Y. In each of the experiments presented here, one of the two following methods was used: According to the first method,

[4]Our processes never used more than one processor or went into swap.

Algorithm k–Means(data set $Y \subseteq \mathbb{R}^d$, integer k)

1. Pick k points $m_1, \ldots, m_k$ from $\mathbb{R}^d$
2. "Assign" each point in Y to the closest of the m_i. We say a point x is a "member of cluster j" if it is assigned to m_j.
3. Repeat until no point changes membership, or until some maximum number of iterations:
    - Recalculate each m_i to be the Euclidean mean of the points assigned to it
    - Reassign each point in Y to the closest of the (updated) m_i.

Figure 6.4: Algorithm k–Means

for every i from 1 to d, the value of the ith coordinate of each median was chosen uniformly at random from the interval $[\text{left}_i, \text{right}_i]$. Here left_i and right_i are the minimum and maximum values, respectively, of the ith coordinate of any point in Y (that is, $[\text{left}_i, \text{right}_i]$ gives the range of the data set within dimension i). According to the second initialization method, each initial median was given the value of a point chosen uniformly at random from Y. The first method may work better in datasets in which many of the points are in one cluster, but a significant number of points are in less populous clusters. If medians are chosen to be data points, the algorithm is very likely to pick more than one median from the populous cluster and then settle into a local optimum in which each median represents part of the populous cluster and one or more small clusters, whereas the first method has a better chance of assigning one median to each cluster. The second method may work better in datasets that are spread out and span a region in $\mathbb{R}^d$ that is mostly empty. The description of each set of experimental results specifies the initialization method used.

Low-Dimensional Data Sets We generated twenty-eight small data sets and ran LSEARCH and k–Means on each. Each one consists of five to sixteen uniform-density, radius-one spheres of real vectors, with five percent random noise. The noise is uniform over the smallest d-dimensional rectangle that is aligned with the axes and contains all the spheres. Here, d denotes the dimension of the data set, which is at most four. The data sets have between one thousand and seven thousand points each. We generated three types of data sets: grid data sets, in which the spheres are centered at regular or nearly regular intervals; shifted-center data sets, in which the spheres are centered at positions slightly shifted from those of the grid data sets, and random-center data sets, where the centers are random in a d-dimensional box. We first ran k–Means using the first method of choosing initial medians, in which each coordinate of every medians is chosen uniformly at random from the range occupied by the data in that dimension. (For comparison, we later ran k–Means using the second method, in which points are chosen uniformly at random from Y; we will present the results of these experiments shortly.) To put the results in perspective relative to earlier work, we also ran both algorithms on a data set distributed by the authors of BIRCH [106], which consists of one hundred, two-dimensional Gaussians in a ten-by-ten grid, with a thousand points each. We considered this among the "grid" data sets.

For each data set, we calculated the SSQ of the k sphere centers on the data set, and we used this value as an upper bound on the optimal SSQ of any set of k medians on this data set. Since each data set has relatively little noise (none in the case of the Gaussian data set), the vast majority of points are in these relatively tight spheres, so the sphere centers should be very close to the optimal centers. In a few cases, LSEARCH or k–Means found better solutions than the calculated "optimal." Therefore, for each data set, we recorded the best known SSQ for that data set; this is simply the minimum of the best SSQ found experimentally, and the pre-calculated upper bound. Because both k–Means and LSEARCH are randomized, we ran k–Means and LSEARCH ten times, to allow for different random initializations, and recorded SSQ and CPU running time for each run of each algorithm.

Figure 6.5 compares the clustering quality achieved by k–Means on the grid data sets to that achieved by LSEARCH; the graph gives the average (represented as a triangle in the case of LSEARCHand as a square in the case of k–Means) and standard deviation (represented as an error bar above and below the corresponding mean) of the estimated approximation ratios of the solutions found by the two algorithms; the estimated approximation ratios were calculated as the ratio of the ssq of the k–Means or LSEARCH solution to the ssq of the best known solution (in many cases, this solution was one found by LSEARCH). It is evident that LSEARCH gave very stable answers in the sense that from solution to solution there was almost no variation in ssq; furthermore, LSEARCH nearly always found an essentially optimal solution. k–Means did well on these sets, but was much less consistent and did not provide as good solutions on average. Its estimated approximation ratios were often around 1.4 and sometimes reached 2.5 or higher. k–Means converged particularly quickly on the Gaussian data set, as would be expected given the concentration around the cluster means that would characterize such a data set. On that data set, k–Means ran, on average, for 298 s, with a standard deviation of 113 s. LSEARCH ran for 1887 s on average, with a standard deviation of 836 s. The average SSQ of k–Means solutions was 209077, with standard deviation 5711, whereas for LSEARCH the SSQ was 176136 with standard deviation 337. On the other grid data sets, k–Means was usually faster than LSEARCH; Figure 6.6 summarizes the averages and standard deviations of the execution times of the two algorithms when run on the grid data sets.

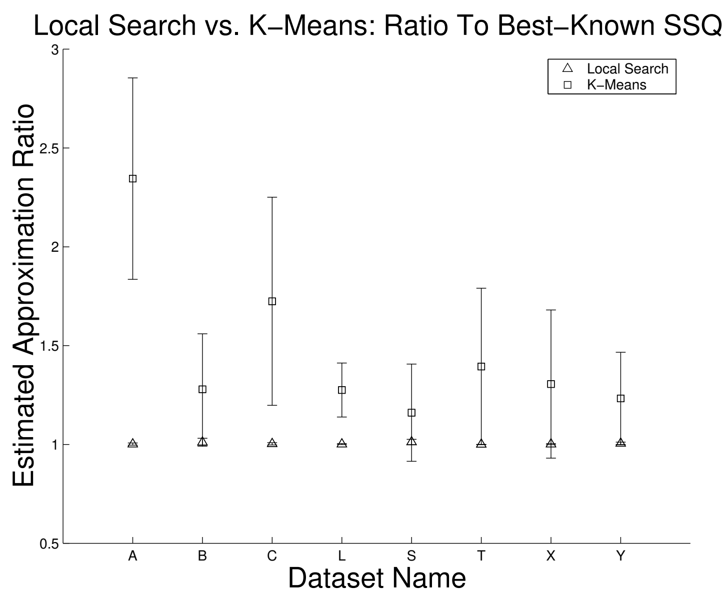

Figure 6.5: k–Means (Init. Method 1) vs. LSEARCH: SSQ on Grid Data Sets

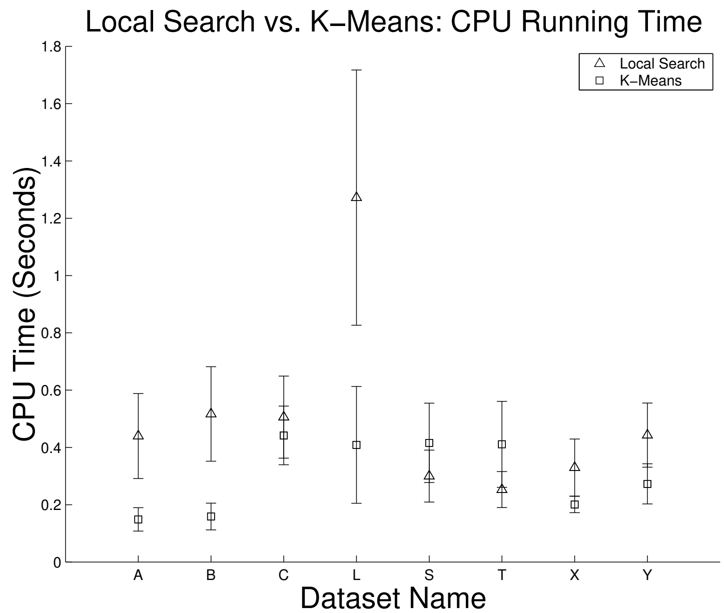

Figure 6.6: k–Means (Init. Method 1) vs. LSEARCH: CPU Time on Grid Data Sets

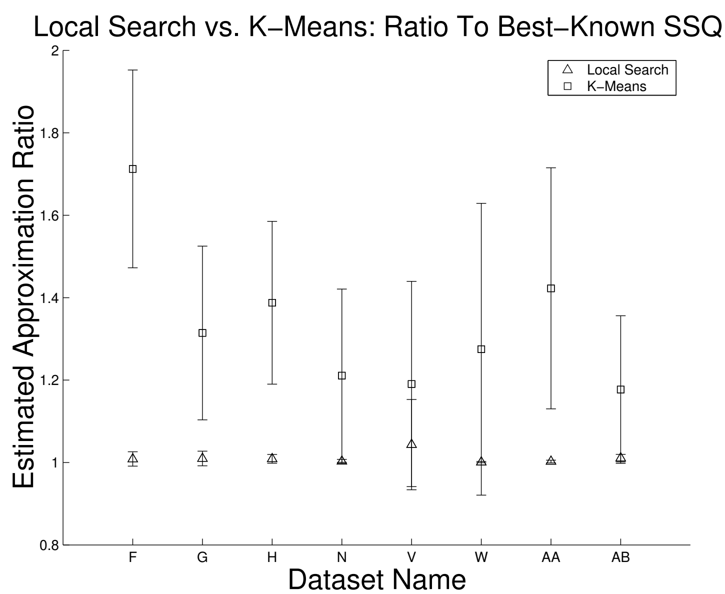

Figure 6.7: k–Means (Init. Method 1) vs. LSEARCH: SSQ on Shifted-Center Data Sets

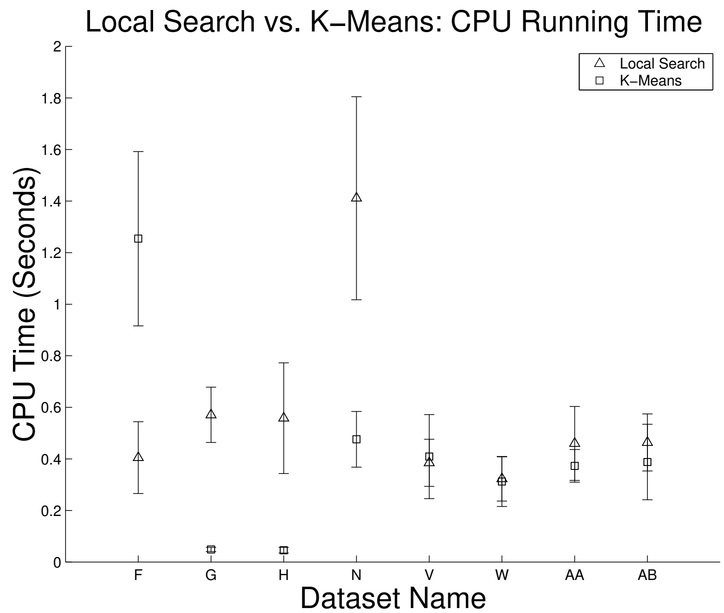

Figure 6.8: k–Means (Init. Method 1) vs. LSEARCH: CPU Time on Shifted-Center Data Sets

On the shifted-center data sets, LSEARCH again produced consistent results and always achieved SSQs that were very close to the best known, and k–Means produced solutions of varying quality that were rarely as good as the LSEARCH results. On these sets, as Figure 6.7 indicates, k–Means fared slightly better than it had on the grid sets, usually achieving approximation ratios around 1.3. Figure 6.8 summarizes the execution times for these data sets, which were often very similar.

On the random-center data sets, LSEARCH still generally got answers close to the best known, whereas k–Means produced SSQs in a much wider range and generally achieved estimated approximation ratios between 2 and 2.5. Figure6.9 summarizes the random-center data set SSQs. Figures 6.22 through 6.29 on pages 114 through 117 give examples of why k–Means performance was consistently worse and often much worse on these data sets. Figures 6.22 and 6.23 show, respectively, the best (lowest SSQ) and worst (highest SSQ) medians found by k-Means for the random-center data set J. The best medians still included one median that covered two clusters, and another that sat amidst noise, but the worst contained two instances of each type of

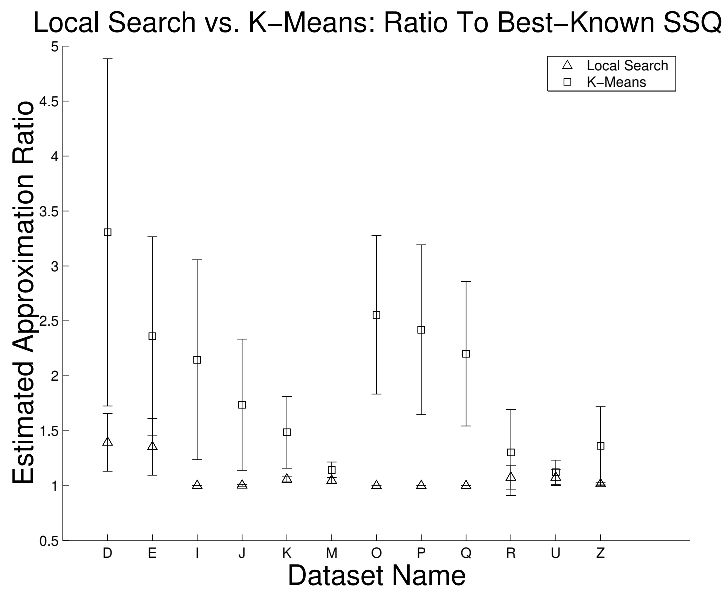

Figure 6.9: k–Means (Init. Method 1) vs. LSEARCH: SSQ on Random-Center Data Sets

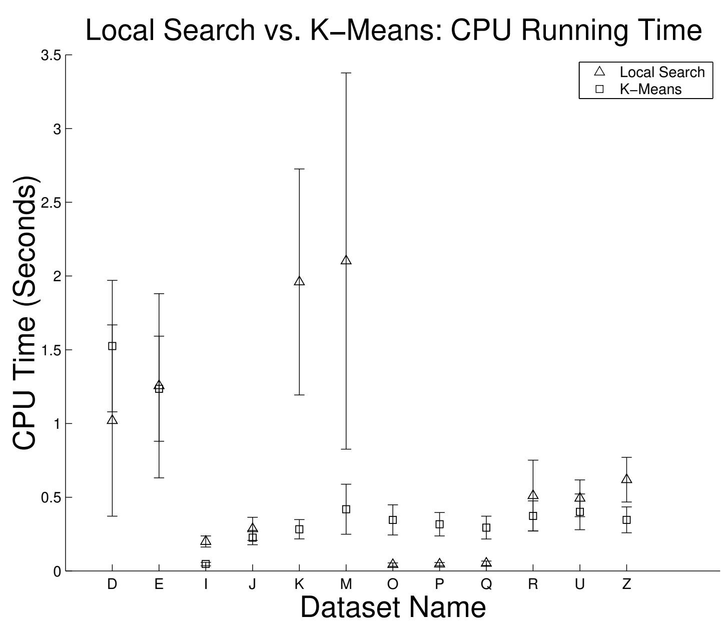

Figure 6.10: k–Means (Init. Method 1) vs. LSEARCH: CPU Time on Random-Center Data Sets

incorrect median placement. By contrast, as shown in Figures 6.24 and 6.25, the best and worst LSEARCH solutions are nearly indistinguishable and both very close to optimal. Figures 6.26 and 6.27 show the best and worst medians found by k–Means for the data set I; the best medians are quite good, but the worst include two medians covering one cluster, two medians each spanning two clusters, and one median in the noise. As Figures 6.28 and 6.29 demonstrate, the best and worst LSEARCH solutions for the same data set are both very close to optimal and nearly identical, down to the median position within each cluster. Figure 6.30 shows that, counter to intuition, k–Means can fail even in the one-dimensional setting. These figures show the data set O, a one-dimensional, random-center set, which is indicated by a histogram where the height of each bar gives the number of data points in the corresponding (x-axis) range. This data set has five well-separated clusters, and yet the worst k–Means solution consisted of one median for a true cluster, two medians each covering two clusters, and two medians with essentially no members. On these sets, the dominance of k–Means with respect to execution time was less certain; LSEARCH actually ran decidedly faster (and got much better SSQ) on the one-dimensional sets O, P, and Q, and was comparable or faster on some others. In a few cases, such as K and M, LSEARCH took much longer to finish than k–Means (and in the case of the data set M, the extra time did not buy a much better solution than that returned by k–Means). The execution times are summarized in Figure 6.10.

Second Initialization Method for k–means For comparison, we tried the second initialization method for k–Means on the twenty-eight synthetic data sets described above and on the grid of Gaussians distributed by the BIRCH authors. The SSQ results for the twenty-eight smaller data sets are presented in Figures 6.11, 6.12, and 6.13. In these figures as before, the means and standard deviations of the SSQ achieved by each algorithm on each data set are indicated. For k–Means using the first initialization method, the means are drawn as squares, and for k–Means using the second method, the means are drawn as diamonds; for both algorithms, the standard deviation is represented as an error bar drawn above and below the mean.

For the grid of Gaussians, the average SSQ using the second initialization method

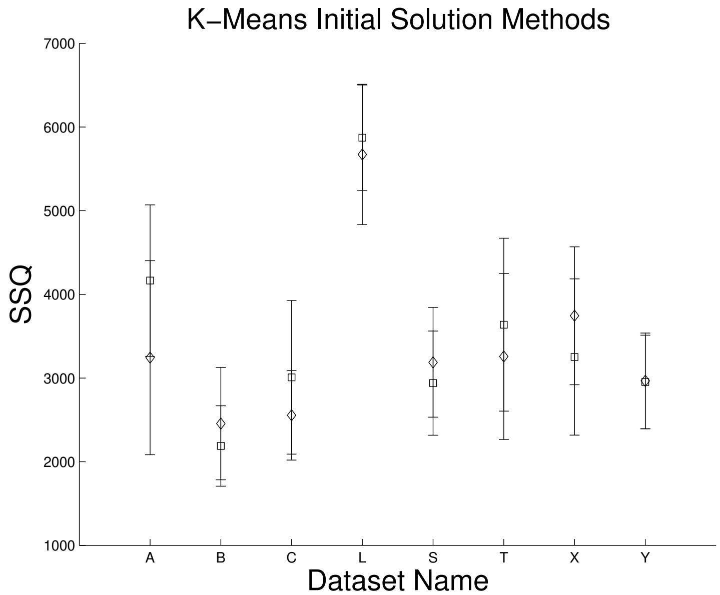

Figure 6.11: k–Means Init. Method 2 vs. Method 1: Grid Data Sets

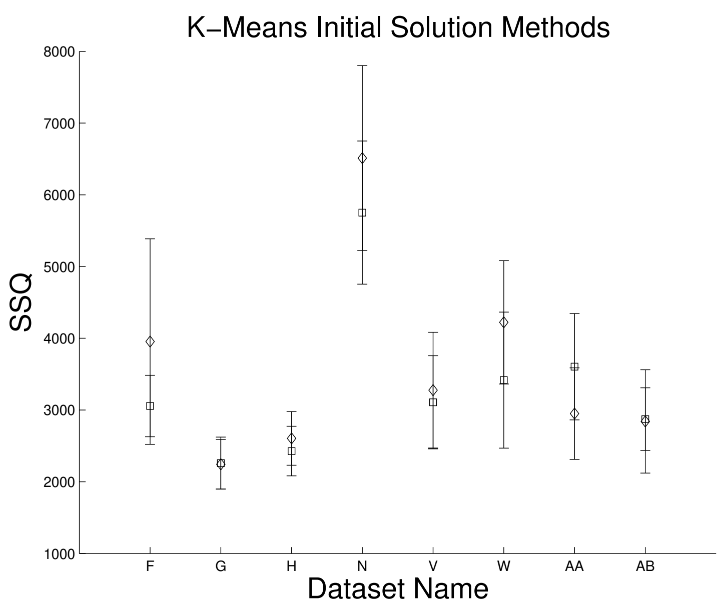

Figure 6.12: k–Means Init. Method 2 vs. Method 1: Shifted-Center Data Sets

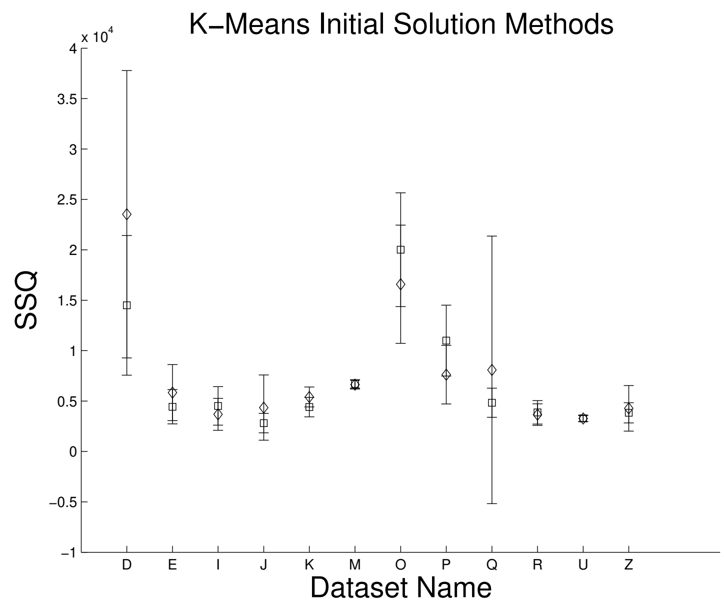

Figure 6.13: k–Means Init. Method 2 vs. Method 1: Random-Center Data Sets

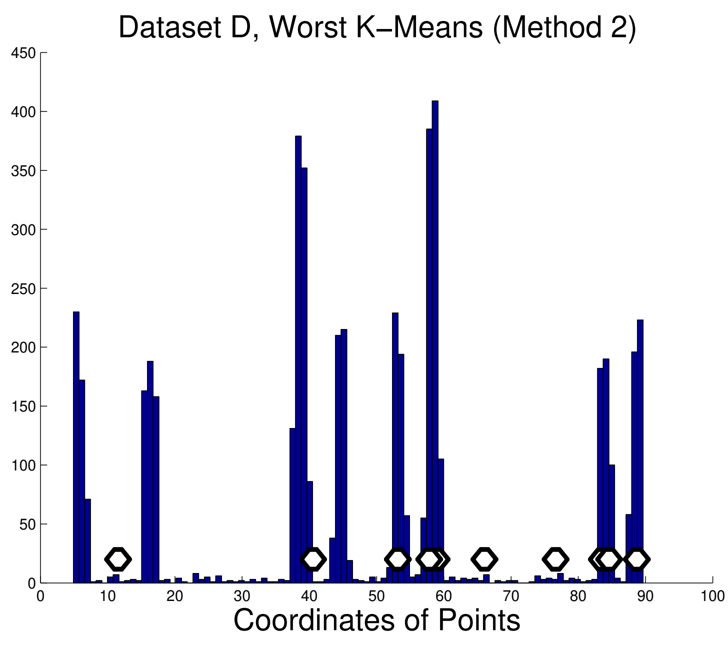

Figure 6.14: Worst k–Means Medians (Init. Method 2) for Data Set D

was 209920, with a standard deviation of 9457. The mean running time was 164 s, with a standard deviation of 32. This performance was worse in terms of SSQ and better in terms of running time than the first method, which gave a mean SSQ of 209077 and used 298 s of CPU time on average. Given how concentrated the data points were around the optimal medians in the grid of Gaussians, any set of initial medians was likely to consist only of points very close to the optimal medians; however, it is also extremely unlikely that every cluster was well-represented in the initial solution, and so, the initial medians probably formed a well-separated but incomplete grid, speeding convergence to a sub-optimal solution. The second method seemed to do worse overall than the first on the other twenty-eight low-dimensional sets, and this implementation of k–Means performed particularly badly on data set D, a ten-cluster, one-dimensional, random-center set in which two pairs of clusters overlapped heavily. Figure 6.14 shows this data set, also in histogram form, and the worst medians found by k–Means using the second initialization method. The pairs of overlapping clusters are around 38 and 58 on the x-axis, as indicated by the doubly-high histogram bars in those regions. k–Means grouped the cluster (or clusters) near 38 with the cluster near 45, gave two medians to the other double-cluster, coalesced two clusters, and wasted two medians on noise.

Small, High-Dimensional Data Sets We also generated ten high-dimensional data sets and ran k–Means and LSEARCH, ten times each, on each of these. All ten have one thousand points and consist of ten uniform-density, randomly-centered, d-dimensional hypercubes with edge length two, and five percent noise. The dimensionalities d of the data sets are: 100 in AC and AD, 125 in AE and AF, 150 in AG and AH, 175 in AI and AJ, and 200 in AK and AL. As before, we ran both algorithms ten times on each data set and averaged the SSQs of the solutions found. For these experiments, the second initialization method for k–Means was used; since nearly all of the 100- to 200-dimensional space was noise, picking initial medians uniformly at random dramatically raised the probability that medians would be chosen from the high-density regions. Nevertheless, the estimated approximation ratios were much higher for k–Means in this set of experiments. Figure 6.15 shows the average SSQ

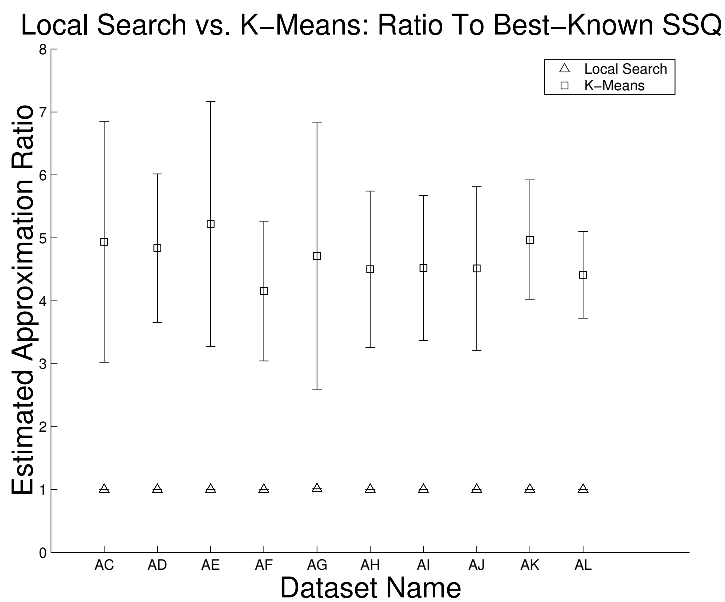

Figure 6.15: k–Means (Init. Method 2) vs. LSEARCH: SSQ on High-Dimensional Data Sets

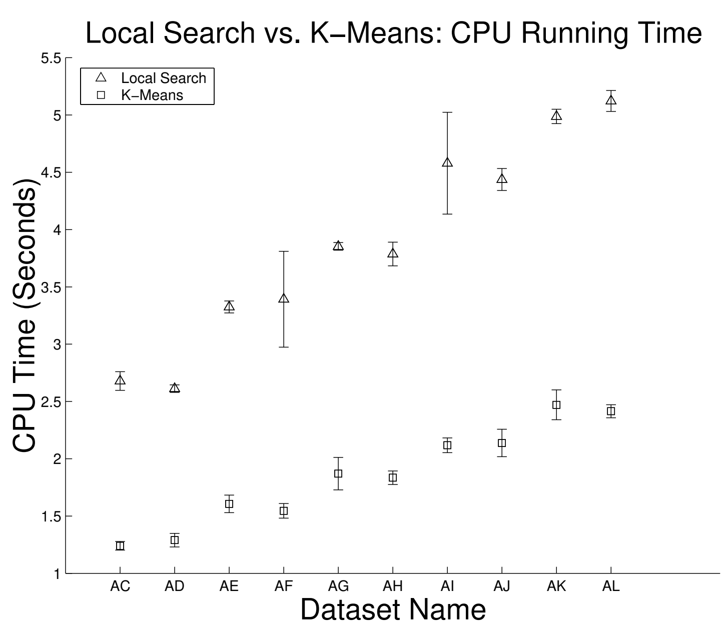

Figure 6.16: k–Means (Init. Method 2) vs. LSEARCH: CPU Time on High-Dimensional Data Sets

calculated by each algorithm on each data set, normalized by the best known SSQ. The error bars show normalized standard deviation. For all data sets, the answer found by k–Means has, on average, four to five times the average cost of the answer found by LSEARCH, which is always very close to the best known SSQ.

The standard deviations of the k–Means costs are typically orders of magnitude larger than those of LSEARCH, and they are even higher, relative to the best known cost, than in the low-dimensional data set experiments. This increased unpredictability may indicate that k–Means is more sensitive than LSEARCHto dimensionality. In terms of running time, LSEARCH is consistently slower, although its running time has very low variance. As shown in Figure 6.16, LSEARCH ran approximately 3 times as long as k–Means on these sets, and both algorithms used CPU time linear in the data set dimension. If instead of counting the running time of each individual execution of each algorithm, we count the cumulative time spent by each algorithm until an execution for which the algorithm finds a high-quality answer, LSEARCH is competitive in running time and excels in solution quality, because in one execution it is likely to find a solution superior to those found by three (or even more) executions of k–Means. These results characterize very well the differences between LSEARCH and k–Means. Both algorithms make decisions based on local information, but LSEARCH uses more global information as well. Because it allows itself to "trade" one or more medians for another median at a different location, it does not tend to get stuck in the local minima that plague k–Means. For example, a solution containing one median with very few members that are mostly outliers, and one median that is shared between two large clusters, may be a local optimum for k–Means. LSEARCH, however, will prefer to exchange the "noise" median in return for an extra median for the pair of clusters that currently share a median, and it is therefore likely to converge to a much better solution.

6.3.2 Clustering Streams

Next we present the results of experiments comparing the divide-and-conquer method from Chapter 4 with the popular BIRCH [106] algorithm. In this section, we will refer to the divide-and-conquer method as STREAM, so that STREAM LSEARCH indicates the use of the LSEARCH algorithm as the clustering subroutine in a two-level divide-and-conquer clustering, and STREAM k–Means indicates the use of k–Means as the clustering subroutine in the same divide-and-conquer scheme.

STREAM k–Means Recall that clustering using the divide-and-conquer method is guaranteed to produce a constant-factor approximation *only* if the clustering subroutine is a constant-factor approximation algorithm. k–Means makes no such guarantees, and so a clustering solution obtained by STREAM k–Means may be arbitrarily expensive with respect to an optimal solution. Due to the popularity of the k–Means algorithm, however, we performed experiments using STREAM k–Means and compared the results with those achieved by STREAM LSEARCH and BIRCH.

BIRCH compresses a large data set into a smaller one via a CF-tree (clustering feature tree). Each leaf of this tree captures sufficient statistics (namely the first and second moments) of a subset of points. Internal nodes capture sufficient statistics of the leaves below them. The algorithm for computing a CF-tree tree repeatedly inserts points into the leaves while the radius of the set of points associated with a leaf does not exceed a certain threshold. If the threshold is exceeded, a new leaf is created and the tree is appropriately balanced. If the tree does not fit in main memory, a new threshold is used to create a smaller tree. Numerous heuristics are employed to make such decisions [106]. Once the data set is compressed in this manner into a CF-tree, another clustering algorithm, such as k–Means or LSEARCH is run on the leaves of the CF-tree, which are analogous to the level-1 medians produced during the execution of a divide-and-conquer clustering.

In the sense that they both produce intermediate clusters, STREAM and BIRCH have a common method of attack, namely repeated preclustering of the data. The preclustering of STREAM is bottom-up, where every substep is a clustering process, whereas the preclustering in BIRCH is top-down partitioning. To put the results on equal footing, we gave both algorithms the same amount of space for retaining information about the stream.

Synthetic Data Stream We generated a 16MB stream of 50,000 points in 40-dimensional Euclidean space. The stream is similar to the data sets described in Section 6.3.1, except that the cluster diameters vary by a factor of 9, and the number of points by a factor of 6.33. We set the "chunk size" (the amount of data to be clustered at once) to be 4MB and, to simulate incremental operation on an incoming stream, ran experiments on each of the four "prefixes" of the data set; the ith prefix is simply the first $(4 \times i)$MB of data, or, equivalently, the first i chunks.

We ran BIRCH on each prefix to generate a CF-Tree, and then ran k–Means and LSEARCH ten times each on each CF-Tree. We also ran STREAM k–Means and STREAM LSEARCH ten times each, measuring quality and running time after each new chunk[5]. Because the data were high-dimensional, k–Means was always run using the second initialization method, according to which initial medians are chosen uniformly at random from the point set. SSQ and running time were both approximately linear in data size for the streaming algorithms. Figures 6.17 and 6.18 compare the performance in SSQ and running time of the four clustering methods. In summary

- STREAM LSEARCH and STREAM k–Means gave SSQs 2 to 3 times better than those achieved by the corresponding BIRCH implementations (i.e., BIRCH LSEARCH and BIRCH k–Means, respectively). The BIRCH CF-trees usually had one "mega-center," a point of very high weight, along with a few points of very low weight; this missummarization of the data stream accounted for the performance gap.

- STREAM ran 2 to 3 times slower than the corresponding BIRCH implementations.

- STREAM LSEARCH found solutions nearly as good as the best known, just as demonstrated in the previous section. The bottom-up approach in STREAM introduces little extra error, and we have a near-optimal answer.

[5]We also ran HAC on the CF-Trees, as the original BIRCH paper advocated, but found that the SSQ was worse than that of the clusters found by k–Means from the CF-tree.

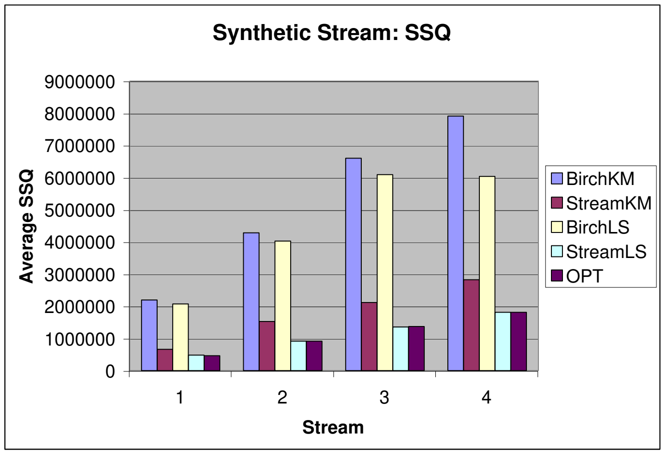

Figure 6.17: BIRCH vs. STREAM on Synthetic Stream: SSQ

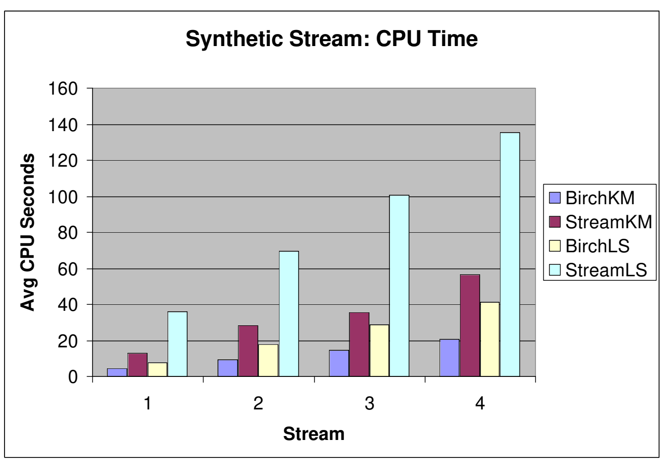

Figure 6.18: BIRCH vs. STREAM on Synthetic Stream: CPU time

Network Intrusions Clustering, and in particular SSQ minimization, is often used for computer network intrusion detection [86, 76]. Since network protection demands the real-time identification of attacks, it is essential to be able to perform the intrusion-detection data analysis on streaming data. We used the KDD-CUP'99[6] intrusion detection data set which consists of raw TCP-dump data for a local area network, simulating a true Air Force computing environment with occassional attacks. Features collected for each connection include the duration of the connection, the number of bytes transmitted from source to destination (and vice versa), and the number of failed login attempts. All 34 continuous attributes out of the total 42 attributes available were selected for clustering. One outlier point was removed. The data set was treated as a stream of nine, 16-Mbyte chunks. The data was clustered with $k = 5$ since each data point represented either normal usage or one of four attack types: denial of service, unauthorized access from a remote machine (e.g., guessing password), unauthorized access to root, and probing (e.g., port scanning).

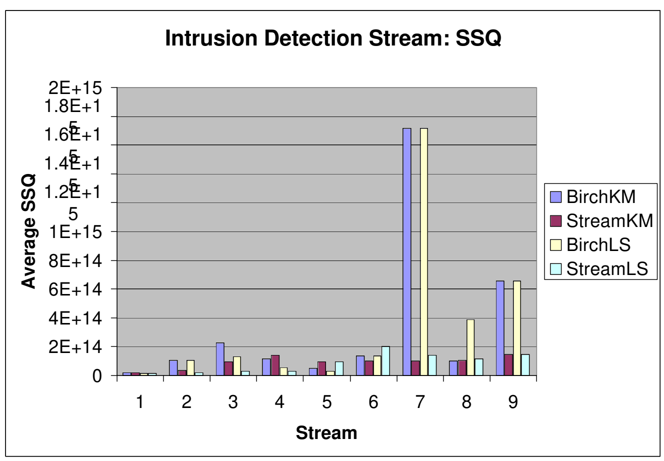

Figure 6.19: BIRCH vs. STREAM on Network Intrusion Stream: SSQ

[6]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

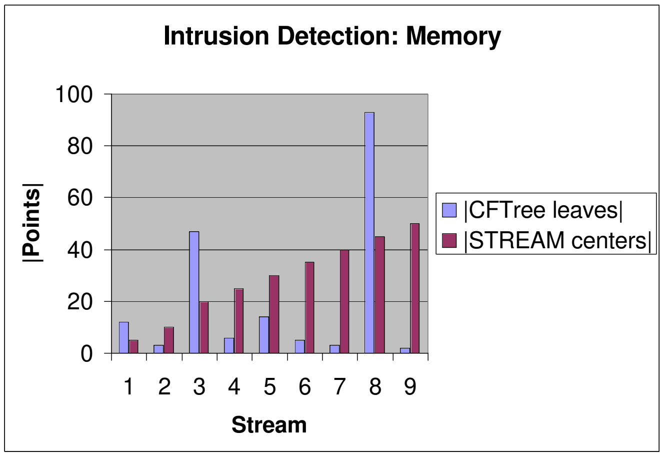

Figure 6.20: BIRCH vs. STREAM on Network Intrusion Stream: Memory Usage

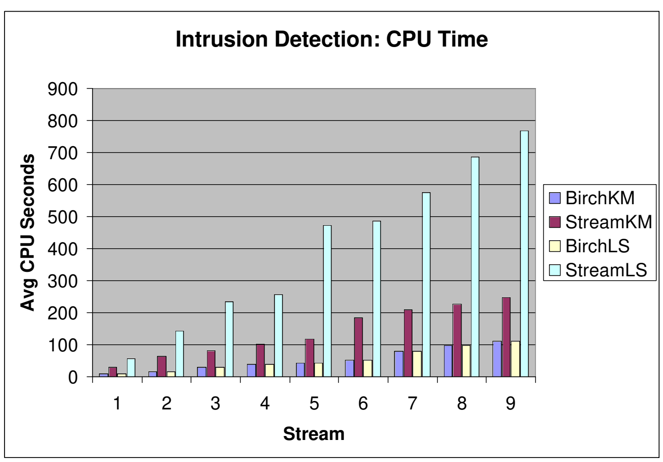

Figure 6.21: BIRCH vs. STREAM on Network Intrusion Stream: CPU Time

Figure 6.19 compares the SSQ of all four methods. BIRCH performed badly on the 7th an 9th chunks of the stream because it generated too small a CF-Tree; the CF-Tree sizes are shown in Figure 6.20. Even though STREAM and BIRCH were given the same amount of memory, BIRCH did not take full advantage of it. BIRCH's CF-Trees had 3 and 2 leaves respectively even though it was allowed 40 and 50 leaves, respectively. We believe that the source of the problem lies in BIRCH's global decision to increase the leaf radius when the CF-Tree size exceeds constraints. For many data sets BIRCH's decision to increase the radius is probably a good one - it certainly reduces the size of the tree. However, this global decision can fuse points from separate clusters into the same CF leaf. Running any clustering algorithm on the fused leaves will yield poor clustering quality; here, the effects are dramatic.

Consistent with expectation, BIRCH ran faster; 6.21 shows CPU time. Overall, these results point to a tradeoff between cluster quality and speed. In applications like clustering web search results, where speed is of the essence, BIRCH is likely to do a quick, reasonably good job. In applications like intrusion detection or target marketing where mistakes can be costly, quality matters more, and the STREAM algorithm is more appropriate.

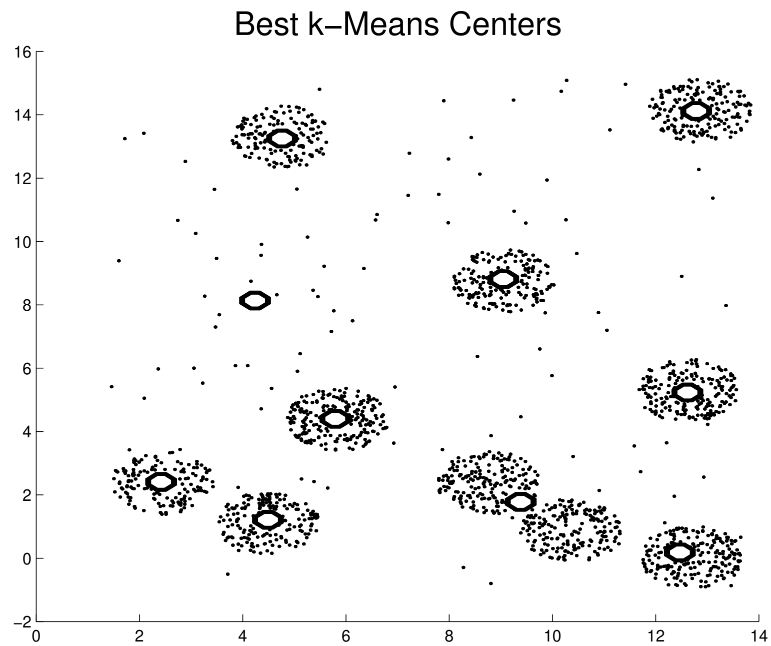

Figure 6.22: Best k–Means Medians (Init. Method 1) for Data Set J

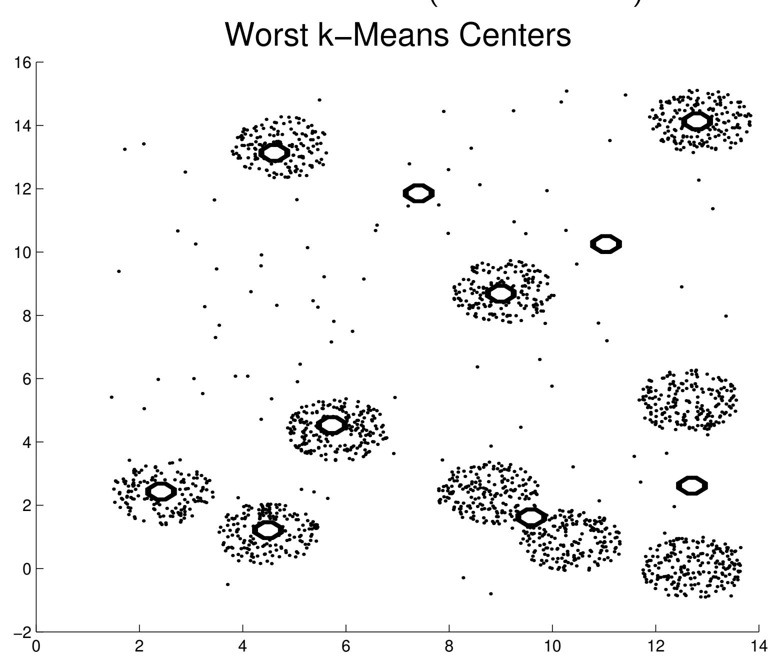

Figure 6.23: Worst k–Means Medians (Init. Method 1) for Data Set J

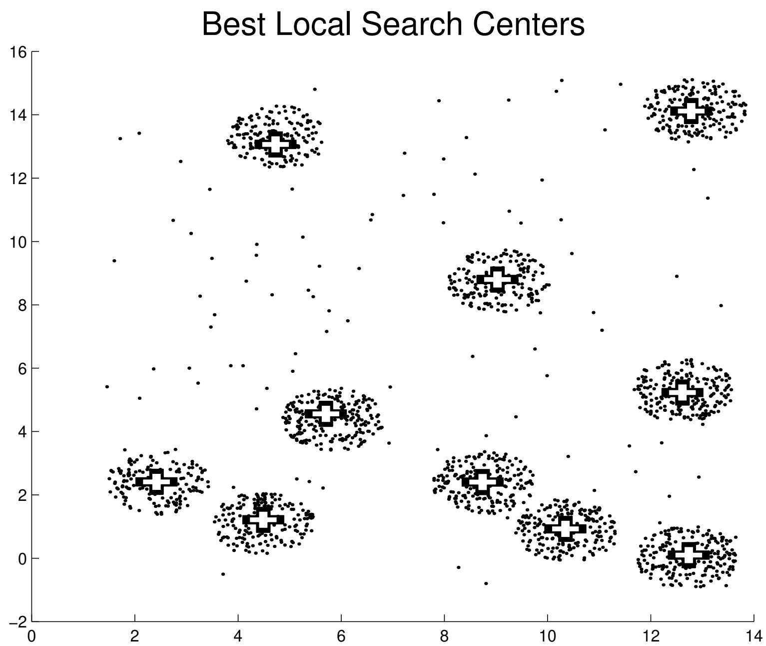

Figure 6.24: Best LSEARCH Medians for Data Set J

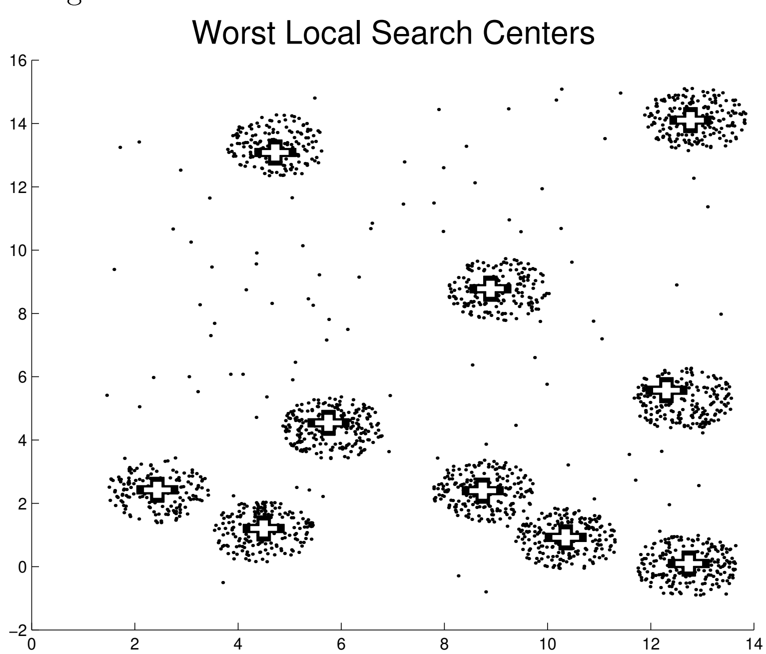

Figure 6.25: Worst LSEARCH Medians for Data Set J

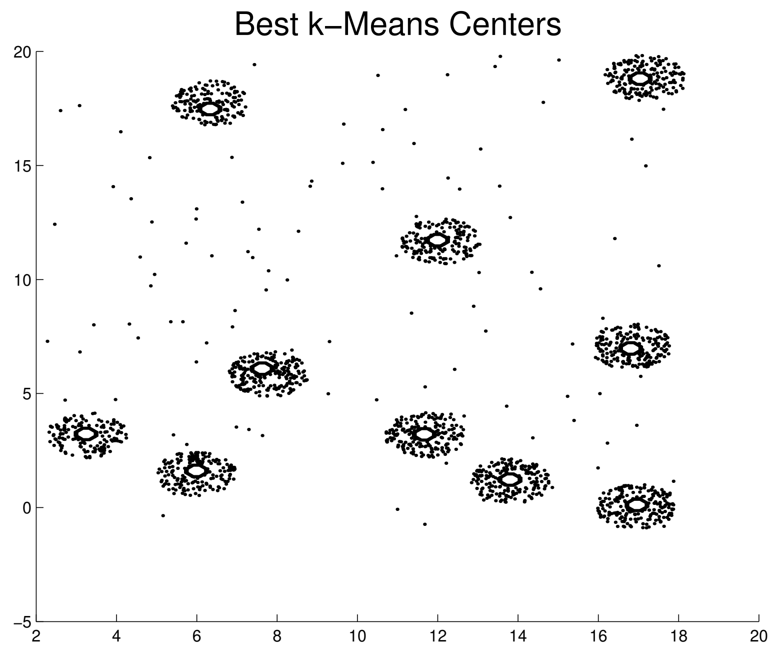

Figure 6.26: Best k–Means Medians (Init. Method 1) for Data Set I

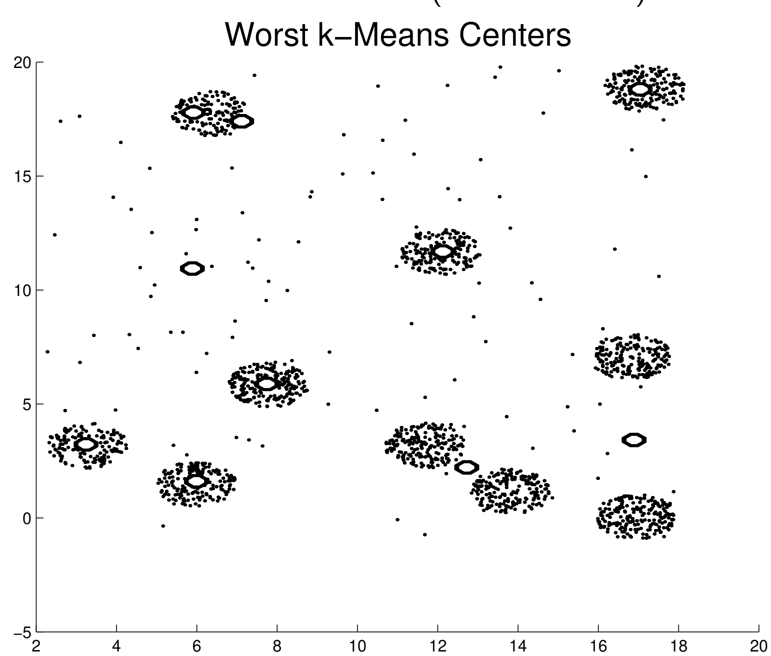

Figure 6.27: Worst k–Means Medians (Init. Method 1) for Data Set I

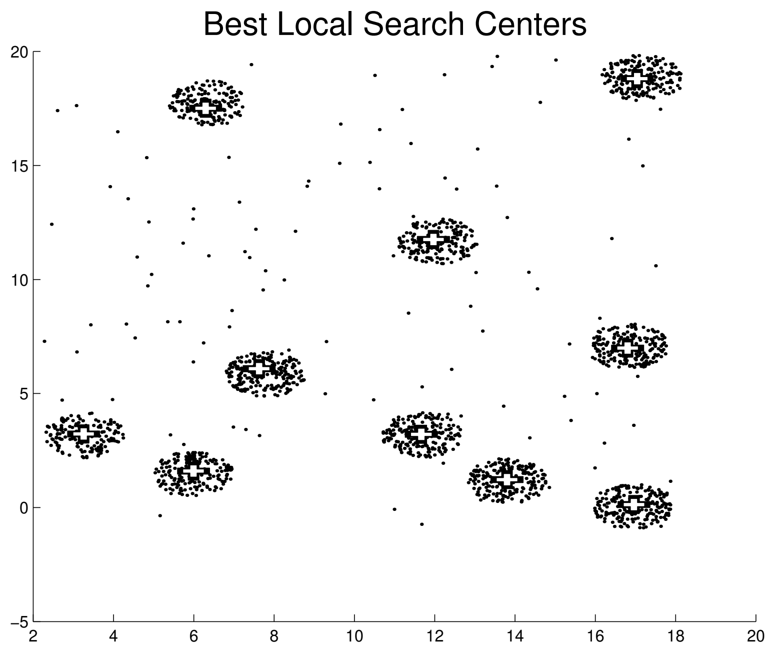

Figure 6.28: Best LSEARCH Medians for Data Set I

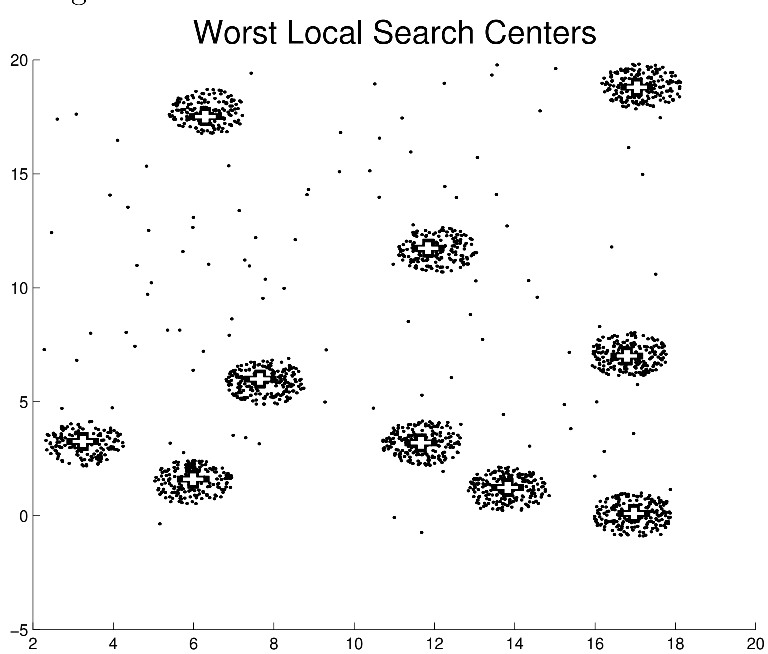

Figure 6.29: Worst LSEARCH Medians for Data Set I

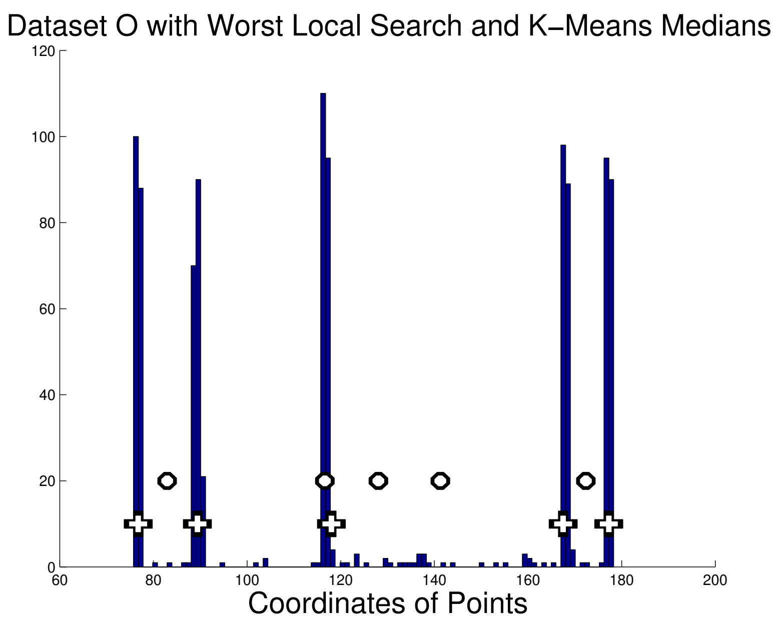

Figure 6.30: Worst k–Means and LSEARCH Medians for Data Set O

Chapter 7

Other Data Stream Clustering Problems

In this chapter, we will present algorithms for other variants of some clustering problems we have already discussed. In section 7.1 we will give an algorithm for the asymmetric k–Median problem on data streams. An instance of this problem is an integer k and a data stream of points; the points are assumed to exist in a space with an asymmetric distance function, and the points are represented according to some format that allows inter-point distances to be easily computed. The meaning of "asymmetric distance function" will be defined more precisely in section 7.1. Earlier algorithms for this problem did not work under the data stream model. Lin and Vitter [72] gave LP-based algorithms for the k–Median problem with arbitrary distances. Their filtering technique gets a $(1+\epsilon)$ approximation by using $(1+\frac{1}{\epsilon})(\ln n+1)k$ centers. They also showed that an $O(\log n)$ increase in the number of centers is inevitable for this problem. Young [105] improved these results by giving a greedy algorithm which produces a $(1+\epsilon)$ approximation using $k\ln(n+n/\epsilon)$ centers.

In some cases, it may be possible to find clusters for a data stream, so that almost all points are very close to their cluster centers. If only a small fraction of the points are far from their closest centers, these few points can be called "outliers" and excluded, that is, not counted in the cost of the clustering. In section 7.2, we present an algorithm for solving k–Center on a stream, allowing a small fraction of the set to be treated as "outliers." For example, if we have n data points and allow δn to be treated as outliers, the cost of a solution is the largest assignment distance, not counting the δn largest assignment distances. In section 7.3, we give an algorithm for the corresponding version of k–Median; here, allowing a δ fraction of the points to be excluded corresponds to excluding the δn highest assignment distances from the k–Median objective function. Charikar *et al.* [20] introduced these outlier formulations for static, stored data sets (rather than streams) and gave constant-factor algorithms for the outlier versions of the k–Center and k–Median problems.

7.1 Algorithms for Asymmetric k–Median

In this section we present algorithms for solving k–Median on sets with asymmetric distance functions. Here, distances are directed, so that $\text{dist}(x, y)$ may not equal $\text{dist}(y, x)$. However, if X is the point set, $\forall\, x,\, y,\, z \in X$ $\text{dist}(x, x) = 0$, and $\text{dist}(x, z) \leq \text{dist}(x, y) + \text{dist}(y, z)$ (that is, the distances obey a directed triangle inequality). The goal will be to select a set M of k data points $m_1, \ldots, m_k$ minimizing $\sum_{x \in X} \text{AD}(x, M)$. (Recall that, because of the way $\text{AD}(x, M)$ is defined, the assignment distances here are directed from assigned points to medians, although the algorithm can easily be changed to solve the reverse problem, in which distances *from* medians are minimized.) Initially, we will assume that all inter-point distances are finite and positive; later we will show how to modify the algorithm in case some inter-point distances are zero. The algorithms are based on the idea of growing ever-larger clusters around medians until all points have been assigned. Initially, a small coverage radius r_1 will be set, and all points will be regarded as unassigned. If for some point x at least some minimum fraction of still-unassigned points can be assigned to x for a cost of at most r_1 apiece, x will be made a median, and its neighbors within r_1 will be marked as assigned, or "covered." We will use the terms "assigned" and "covered" interchangeably. Once there are no points with enough neighbors within r_1, the radius will be increased (doubled, say) to r_2, and we repeat the same procedure. We can show that every time we increase the radius, the number of points assigned by our algorithm is almost as many as the number assigned within the old coverage radius by the optimum

k–Median solution.

Non-streaming Algorithm Given n points with a possibly asymmetric distance function such that the ratio Δ of the largest distance d_{max} to the smallest inter-point (i.e., positive) distance d_{min} is known, and input parameters ϵ and k, the algorithm will produce a solution of at most $2k \log \Delta/\epsilon$ medians, whose cost is at most a constant times the optimum cost. The algorithm proceeds in at most $2 \log \Delta$ phases, with radius r_i for phase i, where $r_i = (1+\epsilon)r_{i-1}$. We will discuss later how to find a good value for r_1, but for now we will assume we have an $r_1 > 0$.

Definition 23 *A point x is said to i–cover a point y if dist$(y, x) \leq r_i$.*

Initially, all points will be unassigned, and in each phase we will "cover" more points, that is, assign them to new medians. For every phase i (call the first phase 1), let U_i denote the number of points that are still unassigned after this phase; define U_0 = n. In phase i we will consider each $x \in X$ in turn; if a point x i-covers at least $\epsilon U_{i-1}/k$ of the unassigned points, we will mark them as "covered" and make x a median. [1] After phase i, we will set $r_{i+1} = (1+\epsilon)r_i$, and then begin phase $i+1$. We will continue until all points are assigned.

Lemma 13 *Let $M^* = \{M_1^*, \ldots, M_k^*\}$ denote the medians of some optimal solution. Let $U_i^* = |\{x|\text{AD}(x, M^*) > r_i\}|$, that is, the number of points that cost more than r_i in the optimal solution. For every phase i, $U_i \leq U_i^* + \epsilon U_{i-1}$.*

Proof: In phase i, each point that i-covers at least $\epsilon U_{i-1}/k$ unassigned points becomes a median (and the points it i-covers become "covered"). Therefore, just after phase i, when there are U_i points that are not yet covered, there is no point that has at least $\epsilon U_{i-1}/k$ unassigned points within r_i of it. In particular, none of the optimum medians has more than this many unassigned points within a distance of r_i. Therefore, the number of still-unassigned points that have at least one optimum median within r_i is at most $k\epsilon U_{i-1}/k = \epsilon U_{i-1}$. The lemma follows. ■

[1] Note that each point need only be considered once per phase as a potential median.

Set $r_1 = d/\Delta$ for an arbitrary inter-point distance d. Observe that $r_1 \leq d_{min}$, and so $nr_1 \leq nd_{min}$. Suppose that $k \leq n/2$. (In fact $k \leq \min\{n/\log n, n/\log\Delta\} \leq n/2$, or else the trivial solution with n medians is feasible in our framework.)

Theorem 20 *The above algorithm finds $O(k\log\Delta)$ medians, whose cost is at most a constant times the optimum k–Median cost.*

Proof: Let OPT denote the cost of the optimum medians, and let COST denote the cost of the medians produced by the above algorithm. Then

$$\text{OPT} \geq \sum_{i=1}^{m-1}(r_i - r_{i-1})U_i^* = \frac{1}{1+\epsilon}\sum_{i=1}^{m-1}(r_{i+1}-r_i)U_i^*,$$

where m is the number of phases, and r_0 is 0. Then

$$\begin{aligned}
\text{OPT} &\geq \frac{1}{1+\epsilon}\sum_{i=1}^{m-1}(r_{i+1}-r_i)(U_i - \epsilon U_{i-1}) \\
&= \frac{1}{1+\epsilon}\sum_{i=1}^{m-1}(r_{i+1}-r_i)U_i - \frac{\epsilon}{1+\epsilon}\sum_{i=1}^{m-1}(r_{i+1}-r_i)U_{i-1} \\
&\geq \frac{1}{1+\epsilon}(\text{COST} - nr_1) - \frac{\epsilon}{1+\epsilon}\sum_{i=0}^{m-2}(r_{i+2}-r_{i+1})U_i \\
&= \frac{\text{COST}}{1+\epsilon} - \epsilon\sum_{i=1}^{m-2}(r_{i+1}-r_i)U_i - \frac{nr_1}{1+\epsilon} - \frac{\epsilon}{1+\epsilon}(\epsilon r_1)U_0 \\
&\geq \frac{\text{COST}}{1+\epsilon} - \epsilon\sum_{i=0}^{m-1}(r_{i+1}-r_i)U_i + \frac{\epsilon-1}{1+\epsilon}(nr_1) \\
&\geq \frac{\text{COST}}{1+\epsilon} - \epsilon[(1+\epsilon)\text{COST} + nr_1 - U_1 r_1] + \frac{\epsilon-1}{1+\epsilon}(nr_1) \\
&\geq \frac{\text{COST}}{1+\epsilon} - \epsilon[(1+\epsilon)\text{COST}] - \epsilon[nr_1] + \frac{\epsilon-1}{1+\epsilon}(nr_1) \\
&\geq \text{COST}\frac{1-\epsilon(1+\epsilon)^2}{1+\epsilon} - \frac{\epsilon^2+1}{1+\epsilon}(nr_1) \\
&\geq \text{COST}\frac{1-\epsilon(1+\epsilon)^2}{1+\epsilon} - \frac{\epsilon^2+1}{1+\epsilon}2\text{OPT}
\end{aligned}$$

The last inequality holds because $k \leq n/2$ implies $\textsc{opt} \geq (n-k)d_{min} \geq nd_{min}/2 \geq nr_1/2$. Therefore $\textsc{cost} \leq \frac{3+\epsilon+2\epsilon^2}{1-\epsilon(1+\epsilon)^2}\textsc{opt}$.

As r_1 is a lower bound on the smallest inter-point distance in X, the algorithm will terminate in at most $2\log_{1+\epsilon}\Delta$ rounds, since if r_i reaches $d\Delta$ it need not be expanded further. Therefore, $O(k \log \Delta)$ medians are produced. ■

Streaming Algorithm Next we will describe how to apply the above algorithm using less memory, in a small number ($O(\log \Delta)$) of passes. The new algorithm uses $O(k^2 \log \Delta)$ space. If memory is scarce, counting the number of new points i-covered by some $x \in X$ requires a pass over X. Therefore, when deciding whether to make x a median, we will *estimate* the true i-coverage of x using a pre-drawn sample. We cannot guarantee that the condition from Lemma 13 holds, but we show that with high probability each round covers a large enough proportion of the points covered by the optimal algorithm.

We will set r_1 and U_0 as before, and choose medians in phases. Every phase i will use three passes over X. First, we will in one pass choose k/ϵ independent random samples from the set of points that are still not covered. We will make a second pass over the data set, examining each point and making it a median if its estimated coverage is high enough. Finally, in the third pass, we will find U_i, the number of points still not covered.[2] (In phase $i+1$, the k/ϵ samples will be taken from these U_i.) We have the algorithm STREAM-AKM, given in Figure 7.1 (the linear passes over X are noted).

The sample size and the condition for becoming a median are chosen so that Lemma 13 above holds with high probability. The condition for becoming a median is that a point have high enough coverage on the still-unassigned points from the current (pth) sample. Because of the size of the samples and the fact that they are drawn independently (so that earlier medians are not picked on the basis of how many points they cover from *later* samples), a large coverage on a sample implies (with high probability) a large coverage on the set of all unassigned points. Consider the

[2] This counting can be done in a single pass, but note that, later, determining whether a specific point is covered can only be accomplished by a computation of the distance to each existing median.

STREAM-AKM(data stream X, integer k, $0 \leq \epsilon \leq 1$)

1. Set r_1.
2. $i \leftarrow 1$.
3. While $i \leq 2 \log \Delta$, do:
   (a) Set $p \leftarrow 1$.
   (b) (First pass): Draw k/ϵ independent samples $S_{i,1}$, …, $S_{i,k/\epsilon}$, each of $\frac{8k \log(nk \log \Delta)}{\epsilon}$ points drawn uniformly at random (with replacement) from the U_{i-1} unassigned points.
   (c) (Second pass): Exit this pass and go to Step 3d when the last $x \in X$ is examined, or $p > k/\epsilon$. For each x, if x i-covers at least $\log n$ (remaining) points in the pth sample ($S_{i,p}$), do:
      i. Make x the next median from the current ith phase. Call this median $m_{i,p}$.
      ii. $p \leftarrow p + 1$.
      iii. Remove from $S_{i,p}$ all points that are covered by medians $m_{i,1}$, …, $m_{i,p}$. If $S_{i,p}$ becomes empty after this removal, continue increasing p by 1 and removing covered points from $S_{i,p}$ until an $S_{i,p}$ is found that is non-empty after the removal step, or until $p > k/\epsilon$.
   (d) (Third pass): Either the point stream or the set of samples was exhausted. Perform another pass to identify points in X that are not covered by the medians $m_{i,1}$, …, $m_{i,k/\epsilon}$.
   (e) $r_{i+1} \leftarrow (1 + \epsilon) r_i$; $i \leftarrow i + 1$.

Figure 7.1: Algorithm STREAM-AKM

optimum medians $M^* = \{m_1^*, \ldots, m_k^*\}$, and let C_j^* denote $\{x \in X | m_j^* = \text{ASG}(x, M^*)\}$, the set of points assigned to m_j^*. Suppose phase i is in progress. For all $x \in X$, let $N(x)$ denote the set of points in X that are i-covered by x. For the purposes of analysis, $S_{i,p}$ can be thought of as being drawn immediately after the $(p-1)$th median is chosen in phase i, as it is independent of existing medians and other samples with respect to the U_{i-1} points that were unassigned at the end of phase $i - 1$.

Suppose that, at some moment during phase i, the optimal median m_j^* is under consideration as a candidate median. Suppose that $m_{i,1}, \ldots, m_{i,p}$ have already been chosen as medians in phase i. Then S_{p+1} is the sample currently under consideration. The portion of $N(c_j)$ that remains to be covered is $R = \bigcap_{l=1}^{p} \{\overline{N(m_{i,l})}\} \cap N(m_j^*)$; suppose $|R| > \epsilon U_{i-1}/k$. By Chernoff bounds, the probability that fewer than $4 \log n$ members of R appeared in S_{p+1} when it was initially drawn is less than $(nk \log \Delta)^{-1}$. Of course, since the only points that could have been deleted from S_{p+1} were members of $\bigcap_{l=1}^{p} \overline{N(m_{i,l})}$, no members of R can have been deleted from S_{p+1}. Thus, if a large portion of the neighborhood in X of m_j^* remains unassigned even after these p medians have been chosen, with high probability m_j^* will cover a large portion of $S_{i,p}$ and will be chosen. Since the same fact is true of every optimal median, there is probability less than $k(nk \log \Delta)^{-1} = (n \log \Delta)^{-1}$ that when phase i terminates there will exist an optimal cluster C_j^* with more than $\epsilon U_{i-1}/k$ remaining unassigned points. Therefore when phase i terminates, the probability is less than $(n \log \Delta)^{-1}$ that there will be more than ϵU_{i-1} unassigned points whose assignment distances under the optimum solution are r_i or less. As there are at most $\log \Delta$ phases, with probability at least $1-1/n$ Lemma 13 holds throughout the execution of the algorithm. At most $\frac{k}{\epsilon} \log \Delta/$ medians are produced.

Zero Distances If many points have zero assignment cost under the optimal solution, the above algorithm may make assignments costing as much as $r_1 > 0$ even if there is a zero-cost solution. Running ZP, a preprocessing algorithm amounting to an execution of STREAM-AKM with a few modifications, solves the problem. ZP adds another $O(\log n)$ passes over X, and finds $O(\log n)$ additional medians.

The changes are as follows. Instead of setting an initial radius r_1 and then having $O(\log \Delta)$ phases, each with a new radius $r_i = (1+\epsilon)^{i-1} r_1$, we will use the same coverage radius $r_0 = 0$ in each phase, and we will have $\log_{1/\epsilon} n$ phases. The sample size will change to $8k \log (nk \log_{1/\epsilon} n)/\epsilon$, and i-covering will be changed to "0-covering": x 0-covers y if $\mathrm{dist}(y, x) = 0$. We can dispense with Step 3e. ZP will conclude with the following clean-up step: if there is exactly one member of X that is still unassigned after all phases, make this point a median. ZP will make at most $(k/\epsilon)(\log_{1/\epsilon} n) + 1$ medians.

Suppose that in the optimal solution with k medians, exactly αn members of

X have an assignment distance of 0; call these the "cheap" points, and the rest "expensive." Let $U_0 = \alpha n$ (a quantity not known to ZP). We will start off with all members of X unassigned, and U_i will denote the number of cheap points that are still unassigned after phase i of ZP. For each phase i, the probability is at least $1-1/(n \log_{1/\epsilon} n)$ that $U_i \leq \epsilon U_{i-1}$. Therefore, after all $\log_{1/\epsilon} n$ phases, with probability at least $1-1/n$, the number of unassigned cheap points is at most α. Note that if α is 1 (the optimal solution has zero cost), a single unassigned point may remain, but the clean-up step removes it. Hence, no cheap points remain after ZP finishes.

Let n' denote the number of members of X that remain unassigned after ZP. Note that all these points are expensive (i.e., have positive assignment distance under the optimal solution). If $n' = \delta n$, then $\alpha \leq 1-\delta$. If $\delta = 1$, we have found a zero-cost solution (albeit with $O(k \log n)$ medians), and we can return this solution without running the rest of the algorithm. Otherwise, we run the algorithm STREAM-AKM using all n members of X as feasible medians, but only trying to cover the n' unassigned points. We will return the $O(k \log n)$ medians found by ZP and the $O(k \log \Delta)$ medians found by STREAM-AKM. If $\delta \geq 1/2$, then at least half the points in the optimal solution have assignment cost at least d_{min}, and so $r_1 n \leq 2\text{OPT}$. In this case, the analysis, and the cost approximation factor, from the previous section still hold. Suppose instead that $\delta < 1/2$. Let OPT' denote the sum of assignment distances of the remaining n' points, under the optimal solution. Then $\text{OPT} \geq \text{OPT}' \geq n' d_{min} \geq r_1 n'$. In this case, the analysis from the previous section shows that the cost of the solution we find is at most $\frac{2+\epsilon+\epsilon^2}{1-\epsilon(1+\epsilon)^2}\text{OPT}$.

7.2 k–Center with Outliers

In this section, we will present an algorithm for the metric k–Center problem on data streams. In general, the object of the k–Center problem is to find cluster centers, and associate points with these centers, so as to minimize the largest cluster radius. The radius of the jth cluster is the distance from the jth center to the farthest point assigned to it. If the problem requirements are relaxed so that a small fraction of the points can be *excluded*, that is, not assigned to clusters, much cheaper solutions may become available. Clearly, as the number that may be excluded increases, the lowest-achievable cost cannot rise.

Suppose that for some set X, r is the smallest maximum radius that can be achieved if at most δ fraction of X can be excluded. We present a one-pass algorithm with space complexity $O(k \log n)$ that will (with high probability) find a solution that excludes only a bit more than a δ fraction and achieves a maximum radius that is at most a constant times r. The algorithm is based on the principle that a random sample taken from X is roughly as "clusterable" as X itself. That is, if a particular set of centers covers a $(1-\delta)$ fraction of X within radius r, these same centers should cover approximately the same fraction of a sample of X within the same radius. We now describe the model and algorithm more formally.

Let a solution to k–Center be denoted by a set $\{c_1, \ldots c_k;\ r\}$ where the c_i are centers and $r \geq 0$ is a radius. If $T = \{c_1, \ldots, c_k; r\}$ is a k–Center solution for a set X, say that $x \in X$ is *assigned* under T if $\exists c_i$ such that $\text{dist}(x, c_i) \leq r$. If a point x is not assigned under T, we say T *excludes* x. Call T an (r, δ)-solution if at least a $(1-\delta)$ fraction of the points in X are assigned under T.

If T is an (r, δ)-solution for X, and S is a subset of X, then T can also be viewed as a solution for S. The radius will not change when T is reinterpreted as a solution for S, but, given this radius, T may exclude a different fraction of S than of X. Note that T can be a valid solution even if it contains centers that are not in S. A solution T' for S can similarly be interpreted as a solution on X.

Assume the existence of a k–Center algorithm **A** that takes a data set X, an integer k, and a $0 \leq \delta \leq 1$, and returns k centers. Assume that there is a constant $\alpha > 1$ such that, if r^* is the smallest radius for which X has an (r^*, δ)-solution, **A** returns an (r', δ)-solution for X, where $r' \leq \alpha r^*$.[3] Then we can construct the algorithm OUT-KC, given in Figure 7.2.

The easiest way to produce the sample is simply to generate s random indices from 1 to n, sort them, and then for each index i, put a copy of the ith member of the stream in the sample (if an index i is picked j times, the sample will contain j copies of the ith stream element). T may exclude a different fraction of X than of

[3] Such an algorithm is given, with $\alpha = 3$, in [20].

OUT-KC (data set X, integer k, $0 < \epsilon < 1$, $0 < \delta < 1$)

1. Take a sample S of s points, uniformly at random with replacement, from X
2. Let $T = \{c_1, \ldots, c_k; r\}$ be the output of $\mathbf{A}(S, (1+\epsilon)\delta)$.
3. Return T, interpreted as a solution on X.

Figure 7.2: Algorithm OUT-KC

S. As we shall see, however, with probability at least $1 - 2/n^2$, T will not exclude more than $(1+\epsilon)^2\delta$ fraction of X, and r will be at most α times the smallest possible radius for a solution that excludes at most δ fraction of X.

Lemma 14 *Let S be a sample of $s \geq 8\frac{\ln n}{(\delta\epsilon^2)}$ points taken uniformly at random from X. Let T be an (r, δ)-solution on X. Then, with probability at least $1 - 1/n^2$, T excludes at most a $(1+\epsilon)\delta$ fraction of S.*

Proof: By Chernoff bounds, the chance that more than a $(1+\epsilon)\delta$ fraction of S is from the δ fraction of X excluded by T, is at most $1 - 1/n^2$. ■

The following lemma shows that, independent of S, it is unlikely that there exists an expensive solution on X that is cheap on S. Then regardless of S, given a solution that is cheap for S, there is only a small chance that this solution is expensive on X.

Lemma 15 *Let S be a sample of $s \geq \frac{2(1+\epsilon)(4+k)\ln n}{\epsilon^2\delta_s}$ points taken uniformly at random with replacement from X. Consider an (r, δ_s)-solution T on S that is an (r, δ')-solution on X. The probability that $\delta' \leq (1+\epsilon)\delta_s$ is at least $1 - 1/n^2$.*

Proof: Consider all solutions that have radius r and that exclude strictly more than $(1+\epsilon)\delta_s$ fraction of X. There are fewer than $n^{2+k}(e/k)^k$ such solutions (as there are at most n^2 possible radii and at most $C(n, k)$ possible choices of centers). Consider such a solution, say T', and let δ'_s denote the fraction T' excludes on S. With probability at most $e^{-s\delta_s\epsilon^2/(2(1+\epsilon))}$, $\delta'_s \leq \delta_s$, by Chernoff bounds. By choice of the value of s, this probability is less than $e^{-(4+k)\ln n}$. Therefore, the probability that there exists an

(r, δ')-solution T'' for X with $\delta' > (1+\epsilon)\delta_s$, that is also an (r, δ'_s)-solution on S with $\delta'_s \leq \delta_s$, is at most $e^{(2+k)\ln n + k - k\ln k} \times e^{-(4+k)\ln n} < n^{-2}$. The lemma follows. ■

Theorem 21 *Let $T = \{c_1, \ldots, c_k; r\}$ be the solution output by* OUT-KC(X, k, ϵ, δ). *Suppose T is an (r', δ')-solution on X. With probability more than $1 - 2/n^2$, the following are both true:*

- $\delta' \leq (1+\epsilon)^2\delta$
- *If the optimal solution that excludes at most a δ fraction of X has radius r^*, then $r' \leq \alpha r^*$*

Proof: Suppose there is an (r, δ)-solution on X. By Lemma 14, with probability at least $1 - 1/n^2$ there will be a solution on S with radius r that excludes at most $(1+\epsilon)\delta$. In this case, the solution returned by $\mathbf{A}(S, (1+\epsilon)\delta)$ will be an $(\alpha r, (1+\epsilon)\delta)$-solution on S. By Lemma 15, with probability at least $1 - 1/n^2$ this solution will exclude at most $(1+\epsilon)^2\delta$ on X. The theorem follows. ■

7.3 k–Median with Outliers

We give a similar algorithm for "k–Median with Outliers." Recall that in k–Median the objective is the sum (or average) of assignment distances. If a solution is allowed to exclude some outliers, the cost will be the average cost (distance to the closest median) of the points that are *not* excluded. We present a one-pass, $O(k \log n)$-space, bicriterion algorithm: if the best solution that excludes at most a δ fraction of the points has some average assignment distance r, this algorithm finds a solution that excludes only slightly more than a δ fraction and achieves an average assignment distance that is $O(r)$.

We will diverge slightly from the usual terminology in this section, because we will want to discuss the distance that would ordinarily be called the "assignment distance," namely, the distance from a point to the nearest median, but, because not all points will be assigned, we will want a different name for this distance. A solution T to k–Median for a data set X of n points will consist of a set $M = \{m_1, \ldots, m_k\}$

of medians. For all $x \in X$, we will call $\textsc{ad}(x, \textsc{asg}(x, M))$ the *radius* of x under T, rather than the *assignment distance* of x. If T can exclude a δ fraction of X, it will of course exclude the δn members of highest radii. Every $x \in X$ that is not excluded will be *assigned*, and in this case the radius of x will also be called the *assignment distance* of x. T will be called an (r, δ)-solution for X if it has average assignment distance r (not *radius*, since only the radii of the assigned points are counted) when it assigns a $(1 - \delta)$-fraction of the points in X.

As before, for $S \subseteq X$, any solution on X can be interpreted as a solution on S, and vice versa. In this case, when we reinterpret a solution for X as a solution for S (or vice versa), we will only keep the medians (not necessarily the radius or fraction excluded) the same. Assume the existence of a k–Median algorithm $\mathbf{B}$ that takes a data set X, an integer k, and a $0 \leq \delta \leq 1$, and returns k medians. Assume constants $\gamma > 1$ and $\beta > 0$ such that if r^* is the smallest radius for which X has an (r^*, δ)-solution, $\mathbf{B}$ returns an (r', δ')-solution for X, where $r' \leq \gamma r^*$ and $\delta' \leq (1 + \beta)\delta$. [4] Given such an algorithm $\mathbf{B}$, we can construct an algorithm essentially identical to OUT-KC for the k–Median problem with outliers; the algorithm OUT-KM is given in Figure 7.3.

OUT-KM (data set X, integer k, $0 < \epsilon < 1$, $0 < \delta < 1$)

1. Take a sample S of s points, uniformly at random with replacement, from X
2. Let $T = \{m_1, \ldots, m_k\}$ be the output of $\mathbf{B}(S, (1 + \epsilon)\delta)$.
3. Return T, interpreted as a solution on X.

Figure 7.3: Algorithm OUT-KM

Definition 24 *Let y be a member of a data set Y, and let T be a solution for Y. Let the members of Y be sorted in increasing order of radius under T (break ties arbitrarily). Define* ***r-rank***$_Y(y)$ *to be the fraction of points of Y that come after y in*

[4] Such an algorithm is given in [20]. For example, β could be 2 and γ could be 6.

this sorted order.[5]

Notation 5 *If P is a set of data points with median set M, let $\overline{P}^M$ denote the average radius under M, among points in P. When it is obvious which median set is intended, we will simply denote this average as $\overline{P}$.*

Fact 3 *Let $A = \sum_{i=1}^{p} x_i$, where all the x_i are non-negative and $\forall 1 \leq i < p$ $x_i \leq x_{i+1}$. Also let $\mu = A/p$. Then $x_m \leq \frac{p}{p-m}\mu = A/(p-m)$.*

Proof: Otherwise, $\sum_{i=m}^{p} x_i \geq \frac{A}{p-m}(p-m+1) > A$, which is a contradiction. ■

Lemma 16 *Let S be a sample of s points drawn uniformly at random with replacement from X.*

Let $s \geq \max\{\frac{4\ln n}{(1+\rho)\ln(1+\rho)-\rho} \times \frac{(1-\delta)/\delta\epsilon}{1-(1+\epsilon/2)\delta}, \frac{2\ln n}{\delta}\}$. Let T be an (r, δ)-solution on X. Then, with probability at least $1 - 1/n^2$, there exist $r' \leq (1+\rho)r$ and $\delta' \leq (1+\epsilon)\delta$ so that T is an (r', δ') solution on S.

Proof: As $s \geq \frac{2\ln n}{\delta((2+\epsilon)\ln 2-\epsilon/2)}$, with probability at least $1 - 1/n^2$, the member y of S with $\textbf{r-rank}_S(y) = (1+\epsilon)\delta$ has lower radius than the member (say, x) of X with $\textbf{r-rank}_X(x) = (1+\epsilon/2)\delta$. Define μ as the average radius of members of X with $\textbf{r-rank}_X$ at least $(1+\epsilon/2)\delta$. Then the radius of x is at most $\tilde{r} = 2\mu(1-\delta)/(\delta\epsilon)$, by Fact 3. Define the scaled radius of each member of X to be its radius divided by $\tilde{r}$. Then the sum of the lowest $1-(1+\epsilon/2)\delta$ scaled radii is $B = [1-(1+\epsilon/2)\delta]n\mu/\tilde{r}$.

Let S' be the set consisting of the first $(1-(1+\epsilon)\delta)s$ sample points (where points are sorted by increasing scaled radii). Recall that with high probability these points all have scaled radii less than 1 (the scaled radius of the distinguished point x). The expected value of the sum of the scaled radii of points in S' is then at most Bs/n. By Chernoff bounds, the probability that this sum actually exceeds $(1+\rho)Bs/n$ is at most $1/n^2$, since $s \geq \frac{4\ln n}{(1+\rho)\ln(1+\rho)-\rho} \times \frac{(1-\delta)/\delta\epsilon}{1-(1+\epsilon/2)\delta}$. The lemma follows. ■

[5] The **r-rank** of an element x is with respect to a solution as well as to the data set to which x belongs. Since the solution in question will usually be obvious in what follows, it will not usually be necessary to mention the solution from which the radii (and hence **r-rank**s) are calculated.

Lemma 17 *Let S be a sample of s points taken uniformly at random with replacement from X. Assume that $s \geq \max\{\frac{2(k+4)(2+\epsilon)\ln n}{\epsilon\delta_s}, \frac{16((k+4)\ln n+\ln 2)}{C\delta_s\epsilon}, \frac{8(k+4)\ln 2n}{\delta_s\epsilon}(\frac{1+\alpha}{\alpha})^2\}$, where $C = \frac{2(1-(1+\epsilon/2)\delta_s}{\epsilon\delta_s}$. Consider an (r_s, δ_s)-solution T on S. Then, with probability at least $1 - 1/n^2$, there exist $r' \leq (1+\alpha)r_s$ and $\delta' \leq (1+\epsilon)\delta_s$ such that T is an (r', δ')-solution on X.*

Proof: We will first fix a specific T' that is an (r, δ)-solution on X, where $r > (1+\alpha)r_s$ and $\delta > (1+\epsilon)\delta_s$. We will show that, if this solution is only allowed to exclude a δ_s fraction of S, then with high probability it has an average assignment cost greater than r_s.

Consider the element $y \in S$ with $\textbf{r-rank}_S(y) = \delta_s$ and the element $x \in X$ with $\textbf{r-rank}_X(x) = (1+\epsilon/2)\delta_s$. By Chernoff bounds, since $s \geq \frac{2(k+4)(2+\epsilon)\ln n}{\epsilon\delta_s}$, with probability at least $1 - n^{-(k+4)}$, the radius of y is greater than or equal to that of x. Therefore, with high probability, the cheapest $(1-\delta_s)$ fraction of S includes a point whose radius is at least the radius of x.

Let $S' = \{y \in S | \textbf{r-rank}_S(y) \geq \delta_s\}$, $X'' = \{x \in X | \textbf{r-rank}_X(x) \geq (1+\epsilon/2)\delta_s\}$, and $S'' = \{x \in S \mid \textbf{r-rank}_X(x) \geq (1+\epsilon/2)\delta_s\}$. So $S'' = S \cap X''$. Note that $\overline{S'}$ is exactly the average assignment distance of T' on the best $(1-\delta_s)$ fraction of S. Thus, the goal is to show that $\overline{S'} > r_s$. Earlier we saw that, with high probability $S'' \subseteq S'$ and so $\overline{S''} \leq \overline{S'}$. Therefore, any lower bound on $\overline{S''}$ will also (with high probability) be a lower bound on $\overline{S'}$. We will find a lower bound on $\overline{S''}$ (and hence on $\overline{S'}$) as follows.

Let μ denote the average radius of the members of X with $\textbf{r-rank}_X$ at least $(1+\epsilon)\delta_s$. Note that $\mu \geq r > (1+\alpha)r_s$, since μ is the radius when we only exclude a $(1+\epsilon)\delta_s < \delta$ fraction of X. Also, let z be the member of X with $\textbf{r-rank}_X$ equal to $(1+\epsilon)\delta_s$, and let $\tilde{r}$ denote the radius of z. Recall that S'' is the set of sample points with $\textbf{r-rank}_X$ no less than $(1+\epsilon/2)\delta_s$. Therefore, S'' may have members with $\textbf{r-rank}_X$ less than $(1+\epsilon)\delta_s$ (whose radius must therefore be at least $\tilde{r}$). For the remainder of the proof, we will assume that X has been modified in the following way: for all members of X whose $\textbf{r-rank}_X$ is at most $(1+\epsilon)\delta_s$, their radius is set to $\tilde{r}$. We will further assume that S has been drawn from this modified version of X. Note that this modification does not increase the value of any radius, and that it therefore does not raise $\overline{S''}$. Therefore, any lower bound on $\overline{S''}$ under this new assumption is a lower bound on the true value of $\overline{S''}$ and therefore on the true value of $\overline{S'}$. Let $\mu' = \overline{X''}$ under the new assumption. Note that by definition of z and $\tilde{r}$, we know that $\mu \leq \mu'$.

Recall that $C = \frac{2(1-(1+\epsilon/2)\delta_s)}{\delta_s \epsilon}$; then $\tilde{r} \leq C\mu'$, by Fact 3. For each member w of X'', we calculate a scaled radius, which is the radius of w, divided by $C\mu'$. Then the maximum of the scaled radii is at most 1, and the average scaled radius, over members in X'', is exactly $\mu'/(C\mu') = 1/C$. Because $s \geq \frac{16((k+4)\ln n + \ln 2)}{C\delta_s \epsilon}$, with probability at least $1-(2n)^{-(k+4)}$, S contains at least $\frac{1}{2}|X''|\frac{s}{n}$ members of X'' (i.e., at least half as many as expected). The expected average of the scaled radii for S'' is exactly the average scaled radius of X'', namely $1/C$. Assuming S'' does contain at least $\frac{1}{2}|X''|s/n$ members of X'', the probability that the average of the scaled radii of S'' is at least $1/(1+\alpha)$ times its expected value is also at least $1-(2n)^{-(k+4)}$. This fact follows from the fact that $s \geq \frac{8(k+4)\ln 2n}{\delta_s \epsilon}(\frac{1+\alpha}{\alpha})^2$.

Therefore, with high probability, $\overline{S''}$, the average of members of S'', before scaling, is at least $\frac{\mu'}{(1+\alpha)}$. Recall that $\mu' \geq \mu \geq (1+\alpha)r_s$; therefore, with high probability, $\overline{S''} \geq r_s$.

We have shown that if a particular T' is an (r, δ)-solution on X with $r > (1+\alpha)r_s$ and $\delta > (1+\epsilon)\delta_s$, then with probability at least $1 - 2n^{-(k+4)}$, T' must either exclude more than a δ_s fraction of S, or else have an average radius greater than r_s.

There are at most $n^2 C(n,k) \leq n^{(2+k)} e^k / k^k$ such solutions. Therefore, the chance that all (r_s, δ_s)-solutions for S correspond to solutions for X that exclude at most a $(1+\epsilon)\delta_s$ fraction and have an average radius at most $((1+\alpha)r_s$, is more than $1 - e^{(2+k)\ln n + k - k\ln k} \times e^{\ln(2) - (4+k)\ln n} < n^{-2}$. The lemma follows. ■

The following theorem, which follows from the two lemmas, shows that if the exclusion fraction is allowed to be slightly more than δ, the average radius in the solution found by OUT-KM$(X, k, \epsilon, delta)$ is not much more than the best possible for a solution that excludes at most a δ fraction of points.

Theorem 22 *Let T be the solution found by* OUT-KM *for input (X, k, ϵ, δ). Suppose that, over all solutions for X that exclude at most a δ fraction of the points, r^* is the lowest possible average assignment distance. Then, with probability more than $1 - 2/n^2$, T is a $(\gamma(1+\alpha)(1+\rho)r^*, (1+\beta)(1+\epsilon)^2\delta)$-solution for X.*

Proof: If there is an (r, δ)-solution on X, then, by Lemma 16, with probability at least $1 - 1/n^2$ there is a $((1 + \rho)r, (1 + \epsilon)\delta)$-solution on S. Then $\mathbf{B}(S, (1 + \epsilon)\delta)$ will return a $(\gamma(1 + \rho)r, (1 + \beta)(1 + \epsilon)\delta)$-solution for S, which, by Lemma 17, will with probability at least $1 - 1/n^2$ be a $(\gamma(1 + \alpha)(1 + \rho)r, (1 + \beta)(1 + \epsilon)^2\delta)$-solution on X. ■

Proof: Suppose there is an (r, δ) solution on X. By Lemma 16, with probability at least $1 - 1/n^2$ there will be a solution on S with radius at most $(1 + \rho)r$ that excludes at most $(1 + \epsilon)\delta$. Then the solution returned by $\mathbf{B}(S, (1 + \epsilon)\delta)$ will have radius at most $\gamma(1 + \rho)r$ and will exclude at most $(1 + \beta)(1 + \epsilon)\delta$ on S. By Lemma 17, with probability at least $1 - 1/n^2$ this solution will have average radius at most $\gamma(1 + \alpha)(1 + \rho)r$ if it is allowed to exclude $(1 + \beta)(1 + \epsilon)^2\delta$ on X. The theorem follows. ■

Chapter 8

k–Median under the Sliding Window Model

Many data sets, such as telephone call records, clicks on a set of web pages, or events recorded by sensors in a network, consist of sets of points that each have an associated time: the time at which a call was placed, at which a user clicked on a path, or at which an event occurred. Often, it is more advantageous to learn about the properties of the most recent points, than to analyze the entire data set together. In such cases, the *sliding window* model may be a useful abstraction.

In this model, data elements arrive continually, and only the most most recent W elements are used when answering queries, where W is a user-specified parameter. In general, it is assumed that W is much larger than the number of points that could be stored at once in main memory; that is, it is not possible simply to maintain a list of the most recent W points, updating this list when a new point arrives. In this chapter, we will examine an algorithm that can, at any moment, produce k medians whose k–Median cost on the most recent W data points is approximately optimal with respect to the cost of the best k–Median clustering. The algorithm maintains an approximate k–Median solution for the last W data points using $\tilde{O}(\frac{k^2}{\tau^3}W^{2\tau})$ memory, where $\tau < 1/2$ is a parameter which trades off the space bound with the approximation factor of $2^{O(1/\tau)}$.

8.1 The Sliding-Window Model

In previous chapters we examined clustering under the data stream model, which assumes that the data to be analyzed cannot fit in main memory. However, up until now, we considered only bounded data streams. There are many instances of data sets that are bounded and stored, but which cannot be processed simultaneously in main memory. For example, a set of high-dimensional points representing many different measurements taken for a large set of stars might be permanently stored on a slow device, but it might be too large to be processed in main memory, so random access would be costly.

However, in cases in which data are being generated continually, the problem of *concept drift* – a change over time in the characteristics of a data set according to the properties analyzed by datamining – may arise. In such cases, it is often preferable to maintain information about the most recent points. For example, suppose a telephone company wishes to determine whether telephones that are being billed at low residential rates are actually being used for telemarketing or other commercial purposes. It is likely that calls placed or received by a line more than six months ago are useless (and possibly misleading) in the determination of the purpose of this phone line, since the line is likely to have had the same user (and calling patterns) for the past six months, but may have had a different user in the past. In this case, we can imagine the stream as a linear list of points, and consider a "window" that slides along the list, always holding exactly the points representing phone calls within the most recent six months.

Instead of a six-month-long window, we could have a window of constant width, *i.e.*, that holds exactly W points, for some given integer $W > 0$. This type of window would be more fitting if, for example, we are monitoring the traffic through a particular router and, for the packets destined for a certain range A of IP addresses, we want to maintain a running estimate of the fraction of these packets that come from a certain other range B of IP addresses. In this case, traffic destined for range B may be very "bursty," *i.e.*, unpredictably variable, and so it may be impossible to determine a time interval length T such that windows of length T are be very likely to have at least one packet headed for range A, and at most some maximum number C of such packets. It makes more sense to keep the window width constant, so that we are not calculating meaningless statistics (the fraction of zero packets that come from range B) or dealing with unmanageably large (more than C) numbers of packets from range A.

Thus we speak of maintaining a statistic σ "over a sliding window," meaning that we keep track of the value of σ as a function of the part of the data stream that is recent enough to be within the window of relevance. In this chapter we will only consider fixed-width windows, that is, windows containing a fixed, pre-specified number W of data points. In general, W may be much larger than the number of data points that could be stored explicitly in main memory, and so we cannot simply maintain a list of the most recent W elements, adding a new element and subtracting the oldest whenever a new point arrives in the stream. We will present an algorithm for maintaining a continuous k–Median solution over a sliding window of length W, that uses $\tilde{O}(\frac{k^2}{\tau^3}W^{2\tau})$ memory, for any $\tau < 1/2$, using amortized $O(k \log W)$ insertion time per data point (hiding factors that are polylogarithmic in W). If W_i represents the window of active points at time i, the algorithm produces k medians $m_1, \ldots, m_k$ such that $\Phi^{W_i}(\{m_1, \ldots, m_k\})$ is at most $2^{O(1/\tau)}$ times $\textsc{opt-km}(W_i, N, k)$, where N is the (possibly uncountable) set of all points in the metric space.

The main problem that will arise in sliding-window problems, setting them apart from other stream analysis problems, will be to update memory to reflect the expiration of old data points, given that the window size far exceeds the amount of memory. While it may be easy to update memory to record the arrival of a new point, it is generally harder to make changes based on the exodus of a point whose identity has not been recorded explicitly. Next, we will briefly discuss a class of data structures, called *exponential histograms*, that solve this problem for a broad class of sliding-window problems.

8.2 Exponential Histograms

Datar *et al.* [26] introduced the exponential histogram, or EH, as a tool for maintaining a count of the number of '1's among the W most recent bits in a bitstream. They also presented a general framework, making use of an EH, for maintaining an estimate of any of a class of aggregate functions over sliding windows. Given a stream of data points, an exponential histogram groups data elements into buckets covering contiguous time intervals (so that each of the most recent W points falls into exactly one bucket), and maintains summary statistics for each bucket. The most recent W points (*i.e.*, those within the window) are termed "active" and the rest are called "expired." A bucket is called "active" if it contains at least one active point, and when the last active point in a bucket expires, the bucket is said to "expire" and is dropped from memory. To curb memory usage, the EH consolidates pairs of adjacent buckets whenever possible. Combining two buckets entails computing summary statistics from the statistics of the buckets to be combined, and then creating a new bucket in their place, with these new summary statistics, that represents the set of data points formerly represented by the combined buckets (note that this set is still contiguous). The EH also guarantees that no bucket contributes too much to the value of the function that is being estimated over the sliding window; as we shall see, this guarantee enables the EH to deal with the problem of eliminating expired data.

At most times, there is one bucket containing some expired and some unexpired data. The statistics for this oldest bucket, which straddles the sliding-window boundary, are inaccurate because there is no way of distinguishing the relative contributions of the expired and unexpired data to the statistics of this bucket. This inaccuracy is precisely the problem of eliminating expired data, which particularly characterizes the sliding window model. The EH algorithm, however, guarantees that this inaccuracy is small, because, as mentioned, the EH bounds the contribution of each bucket (and therefore of the oldest bucket) to the value of the function being maintained over the window.

Datar *et al.* proved that, if f is a function satisfying the following properties for all multisets X and Y in its domain, the EH structure can be used to maintain an estimate of the value of f over a sliding window:

Property 1. $f(X) \geq 0$.

Property 2. $f(X) \leq poly(|X|)$.

Property 3. $f(X \cup Y) \geq f(X) + f(Y)$.

Property 4. $f(X \cup Y) \leq C_f(f(X) + f(Y))$, where $C_f \geq 1$ is a constant.

As we will see shortly, the "statistic" that will be maintained by the k–Median algorithm will actually be a set of k medians, and an estimate of $\textsc{opt-km}(W_t, N, k)$, where N is the set of all points in the metric space, and W_t is the set of the last W points in X_t (*i.e.*, the window points that are active at time t). Note that this is a *continuous* k–Median problem, whereas most of the algorithms in the previous chapters gave solutions to discrete k–Median problems. The reason that a common set of feasible medians must be assumed (in this case, the ground set of the metric space) is that the function $\textsc{opt-km}(S, S, k)$ fails to satisfy property 3, as follows. $\textsc{opt-km}(S_1 \cup S_2, S_1 \cup S_2, k)$ can be less than $\textsc{opt-km}(S_1, S_1, k) + \textsc{opt-km}(S_2, S_2, k)$, since in the latter each demand set has fewer feasible medians than in the former. Because all points in the metric space will be feasible medians in the k–Median instances in this chapter, it is most likely that the feasible set will be specified implicitly. For example, the metric space is likely to be some vector space with a metric distance function. We will show how to maintain a close estimate of $\textsc{opt-km}(W_t, N, k)$, by extending the EH technique, which cannot be applied directly to the k–Median problem for reasons that will be described presently.

$\textsc{opt-km}$ (regardless of the feasible and demand sets) also fails to satisfy Property 4, above, a property that is central to the ability of the EH to limit the inaccuracy due to the expired data in the oldest active bucket. The following example shows why Property 4 fails for $\textsc{opt-km}$: Consider two sets S_1 and S_2 of data points, each tightly clustered about its median. (In fact, S_1 and S_2 could each be multisets consisting of many copies of the same point.) $\textsc{opt-km}(S_1, N, 1)$ and $\textsc{opt-km}(S_2, N, 1)$ will both be small. However, if the median of S_1 is far from the median of S_2, $\textsc{opt-km}(S_1 \cup S_2, N, 1)$ can be arbitrarily large. Therefore, naïvely applying the EH technique might lead to a configuration of a large number of buckets, none of which could be combined without the production of a bucket with a large value of OPT-KM. Such a configuration would force the algorithm to choose between overrunning memory and producing a solution of possibly high cost. In this chapter, we will see how to extend the EH technique so as to guarantee an approximate solution to the k–Median problem over sliding windows.

8.3 Clustering Over Sliding Windows

In this section we present an approximation algorithm for the Sliding-Window k–Median (SWKM) problem which is to be able to produce[1], at any time t, a set M_t of k medians with $\Phi^{W_t}(M_t) = \text{OPT-KM}(W_t, N, k)$; recall that W_t is the set of active points at time t, and N is the set of all points in the metric space. The approximation algorithm will be able to produce a set of medians whose cost is at most a constant times the optimal. SWKM is a discrete k–Median problem, but the set of feasible medians is not the same as the demand set; the feasible set has been chosen in this way for convenience.

The first k–Median algorithm uses $\tilde{O}(\frac{k^2}{\tau^3} W^{2\tau})$ memory and produces a bicriterion approximation: a set of $2k$ medians whose k–Median cost is within a constant factor $(2^{O(1/\tau)})$ of optimal, where $\tau < 1/2$ is a parameter which captures the trade-off between the space bound and the approximation ratio. We will show how to improve this algorithm and find *exactly* k medians whose approximation guarantee is also $(2^{O(1/\tau)})$, without asymptotically increasing the amount of memory required.

[1]Later, we shall see the importance of the distinction between *producing* a set of medians and *being able* to produce them; actually producing them will take time, and so it is faster to produce them only when necessary, but to have enough information stored at all times in the data structure so that it is possible to produce them on demand.

8.3.1 Bicriterion Approximate Clustering

We will present the bicriterion algorithm first. We begin by defining the terminology we will use to describe data streams in a way that will be convenient for this discussion. Then we describe the EH data structure used by the SWKM algorithm, the combination step for combining two buckets, and the method for associating a cost with each bucket. Next, we discuss the estimation procedure of the final clustering of the current active window after each data point arrives, or whenever a clustering is desired. Finally, we present the overall algorithm that maintains the data structure as new points arrive.

We will refer to the points in the stream as entering at the right and moving towards the left, so that the window of active points slides to the right with respect to the stream, picking up one new point at the right and dropping one out at the left whenever it advances. We employ the notion of a *timestamp*, which corresponds to the position of an active data point in the current window. We timestamp the active data points from most recent (rightmost) to oldest (leftmost) with the most recently arrived data point having a timestamp of 1 and the oldest having timestamp W. x_i will denote the data point with timestamp i. Clearly, the timestamps change with every new arrival, and we do not wish to make explicit updates. A simple solution is to record the arrival times in a wraparound counter of $\log W$ bits; then the timestamp can be extracted by comparison with the counter value of the current arrival.

The exponential histogram is far more general than the traditional histogram used in the database literature. In particular, every bucket in an EH stores some summary or synopsis structure for a contiguous set of data points. The bucket represents this segment of the stream, and we will sometimes say that these points are "in" the bucket, or that the bucket "contains" the points, although the bucket stores only the synopsis of these points, and not the points themselves. In this case, the synopsis will be a set of medians computed for the contiguous segment of the stream represented by the bucket; these medians stored by a given bucket will have been obtained by levels of divide-and-conquer clustering of the points represented by the bucket. Along with this synopsis, every bucket stores the timestamp of the most recent data point in that bucket (the *bucket timestamp*). When the timestamp of a bucket reaches $W+1$, all data points in the bucket have expired, so that bucket can be dropped and its memory reclaimed. The buckets are numbered $B_1, B_2, \ldots, B_m$, starting from most recent (B_1) to oldest (B_m); $t_1, t_2, \ldots, t_m$ denote the bucket timestamps. All buckets except the oldest contain only active points; the oldest always contains at least one active point, and usually contains one or more expired points.

Each median stored in a bucket B is the result of one or more levels of clustering of points in B. As in Chapter 4, points from the data stream will be called "level-0" medians, and medians obtained by i levels of clustering will be called "level-i" medians. When a set of level-0 medians from bucket B are clustered, each level-0 median has a closest level-1 median in the resulting k–Median solution, to which it is "assigned." These assignments are not stored by the histogram, but for each level-1 median, a sum of assignment costs (where an assignment cost is the assignment distance times the weight of the assigned point) is computed. Eventually, the level-1 medians (along with others) may be clustered to produce level-2 medians, and each level-1 median will be assigned to the nearest level-2 median. All the level-0 medians that were assigned to a particular level-1 median will now be assigned to its level-2 median. The sum of assignment costs associated with the level-1 median will now be treated as its estimated cost; this cost, plus the cost of its assignment to the level-2 median to which it is assigned, will be added into the cost of that level-2 median. In this way, the assignments can be thought of as a set of trees, one for each final median, with every level-0 median represented as a leaf, and, through a chain of assignments traveling up the tree containing this leaf, ultimately assigned to the root. The path from the leaf to the root goes first to the level-1 median to which the leaf was assigned, then to the level-2 median to which that level-1 median was assigned, and so on.

Each median is represented by the triple $(p(x), w(x), c(x))$. The value $p(x)$ is the identifier of x; in Euclidean space, for example, this could be the coordinates of x. The value $w(x)$ is the weight of a median x, *i.e.*, the number of points that x represents. As in Chapter 4, if x is of level 0, $w(x) = 1$, and if x is of level i, then $w(x)$ is the sum of the weights of the level-$(i-1)$ points that were assigned to x when the level-i clustering was performed. Finally, $c(x)$ is the estimated cost of x alluded to just above. If x is of level 0, $c(x) = 0$; if x is of level 1, $c(x)$ is the sum of assignment distances of the members of x; and, if x is of level $i > 1$, $c(x)$ is the sum over all members y of x, of $c(y) + w(y) \cdot \mathrm{dist}(x, y)$. Thus, $c(x)$ is an overestimate of the "true" cost of x, meaning the true value of $\sum_{y \in T(x)} \mathrm{dist}(y, x)$, where $T(x)$ is the subtree of assignments rooted at x.

For each bucket, as in the divide-and-conquer framework, we maintain medians at intermediate levels, and whenever there are W^τ medians at the same level in a bucket, we cluster them into $2k$ medians at the next higher level. Let L denote the number of levels of medians that will be allowed (so that $L - 1$ is the highest level a median can attain). How large must L be to guarantee that it is not necessary to have more than $L - 1$ levels of clustering? An equivalent question is how large L must be so that it would not be possible for the algorithm to produce more than W^τ level-$(L-1)$ medians. Given that each level of clustering reduces W^τ medians to just $2k$, if $L \geq 2/\tau$ and $\log_W(2k) \leq \tau/2$, it is not possible for at most W level-0 medians to produce more than W^τ level-$(L - 1)$ medians, and so buckets that are restricted to contain L levels of medians with W^τ per level can represent up to W original points (level-0 medians). As we will see, we can prevent buckets from representing more than W points, and so it is sufficient for each bucket to have the space to store L levels of N^τ medians each, where $L = 2/\tau$. Thus, each bucket B_i can be split into L groups $R_i^0, \ldots, R_i^{L-1}$, where each R_i^j contains medians at level j. Each group contains at most W^τ medians. Along with a collection of medians, the algorithm also maintains for each bucket B_i a *bucket cost*, which is a conservative overestimate of the assignment cost of clustering all the points represented by the bucket.

Combination Let B_i and B_j be two (typically adjacent) buckets that need to be combined to form the merged bucket $B_{i,j}$, and let $R_i^0, \ldots, R_i^{L-1}$ and $R_j^0, \ldots, R_j^{L-1}$ be the groups of medians from the two buckets, where R_i^l (resp., R_j^l) represents the group of medians at level l in bucket B_i(resp., B_j). We set $R_{i,j}^0 = R_i^0 \bigcup R_j^0$. If $|R_{i,j}^0| > W^\tau$, then we cluster the points from $R_{i,j}^0$ and set $R_{i,j}^0$ to be empty. Let C_0 denote the set of $O(k)$ medians resulting from the clustering of $R_{i,j}^0$, or the empty set if $R_{i,j}^0$ did not need to be clustered. We carry over these level-1 medians to the next group $R_{i,j}^1$. Set $R_{i,j}^1 = R_i^1 \bigcup R_j^1 \bigcup C_0$. As before, if there are more than W^τ medians, we cluster them to get a carry-over set C_1, and so on. In general, after at most L unions, each possibly followed by clustering, we get the combined bucket $B_{i,j}$. Finally, we compute bucket cost as described above by computing a clustering of all medians in the bucket, at all levels. (The original medians are not destroyed by this operation, although in many cases the result of this combination operation will replace the two buckets B_i and B_j, in which case the original buckets will be destroyed and $B_{i,j}$ will substitute for it.)

The randomized algorithm of Indyk [59], implemented as described in Chapter 4, will always be used to cluster the level-0 medians. This algorithm requires $\tilde{O}(nk)$ time and space to cluster n points, and with high probability gives a bicriterion approximation: $2k$ medians whose cost is at most a constant times the cost of the optimal solution (for the points clustered). When clustering higher-level medians, the algorithm will use the local search algorithm of Charikar and Guha [18]. This algorithm requires $O(n)$ space and $O(n^2 \log n)$ time to cluster n points.

Cost Function Consider a bucket B_i. Let $Z_i = \bigcup_{j=0}^{L} R_i^j$ be the set of medians in B_i, and let Y_i be the set of points (level-0 medians) represented by B_i. The bucket cost function is an estimate of the assignment cost of clustering the points represented by the bucket, that is, of $\textsc{opt-km}(Y_i, N, k)$. Whenever a bucket cost is computed, it is stored with the bucket. We compute a clustering of Z_i, first using the algorithm of Indyk mentioned above to find $2k$ medians, then using the primal-dual algorithm of Jain and Vazirani [63] to produce a set M of k medians $m_1, \ldots, m_k$. The cost function for bucket B_i is given by: $f(B_i) = \sum_{x \in Z_i} c(x) + w(x) \cdot \textsc{ad}(x, M)$. Because the medians Z_i themselves were also computed using constant-factor approximation algorithms, we have $\textsc{opt-km}(Y_i, N, k) \leq f(B_i) \leq 2^{O(L)}\textsc{opt-km}(Y_i, M, k)$. Therefore, $f(B_i)$ is an overestimate (by at most a constant factor) of $\textsc{opt-km}(Y_i, N, k)$.

No median (either a "real" median in a bucket, or a median computed only for purposes of computing the cost of a bucket or the combination of two buckets) will ever be produced by more than L levels of clustering (up to $L-1$ for any stored medians, and up to one more level for medians calculated from input containing level-$(L-1)$ medians). It follows that every time this SWKM algorithm calculates a set of medians, their approximation factor with respect to the original points they represent (that is, the points that would form the leaves of the clustering trees of which these medians are the roots) will be bounded above by some constant, say β, of the form $2^{O(L)}$. We can then say that $\text{OPT-KM}(Y_i, N, k) \leq f(B_i) \leq \beta\text{OPT-KM}(Y_i, N, k)$.

Note that if B_i contains only level-0 medians the clustering algorithm used to compute the bucket cost is randomized, and so two different computations of $f(B_i)$ might arrive at two different values. Note, however, that because the clustering algorithm produces a constant-factor approximation to the clustering instance (which, in the case of bucket-cost computations, may include level-$(L-1)$ medians), the cost returned for B_i will always be at least $\text{OPT-KM}(Y_i, N, k)$ and at most $\beta\text{OPT-KM}(Y_i, N, k)$.

In addition to its own bucket cost $f(B_i)$, the bucket B_i also maintains two other costs: $f(B_{i-1,i-2})$, and estimate of the cost of the bucket that would result if B_{i-1} and B_{i-2} were merged and $F(B_i^*)$, an estimate of the cost of the bucket that would result if buckets B_1 through B_{i-1} were merged. These values are updated whenever a new point enters the window (and another point expires).

Estimation Let $B_1, B_2, \ldots, B_m$ denote the buckets at any time instant t. B_m is the oldest bucket and contains medians that represent some data points that have already expired. If a query is posed at this moment that asks for a clustering of the active points we do the following:

- If the bucket B_m consists only of level-0 medians, discard those that have expired, and cluster the rest, with all medians from buckets B_1 through B_{m-1}, using the algorithm of Charikar and Guha[18], to produce $2k$ medians; use the primal-dual algorithm of Jain and Vazirani[63] to produce k final medians. Return these k medians.

- Otherwise, if B_m does not still retain all its level-0 medians:

  - Cluster the medians in bucket B_m, using the Charikar-Guha algorithm, to produce $2k$ medians, and use the Jain-Vazirani algorithm to reduce these to k final medians.

  – Using the Charikar-Guha algorithm, cluster all medians in buckets B_1 through B_{m-1} together to produce $2k$ medians, and cluster these using the Jain-Vazirani algorithm to produce k final medians. Return the k final medians from B_m and the k final medians from B_1 through B_{m-1} as the answer.

If required, the procedure can also provide an estimate for the assignment cost using the same technique as that used for computing the cost function over buckets.

We now calculate the running time of this procedure. We will show that m is $O(\log W)$, but for now, let us assume this bound holds. If B_m has retained all its level-0 medians, the number of medians clustered using the Charikar-Guha algorithm is W^τ for B_m, plus $(m-1)LW^\tau$ for the rest. This is fewer than LmW^τ altogether, which can be clustered in $O(L^2m^2W^{2\tau}\log(LmW^\tau))$ time; this total is $\tilde{O}(L^2W^{2\tau})$. Otherwise, at most LW^τ medians are clustered from B_m, requiring $O(L^2W^{2\tau}(\log(LW^\tau))$, or $\tilde{O}(L^2W^{2\tau})$, time, and at most $(m-1)LW^\tau$ medians are clustered from the remaining buckets, requiring $O((m-1)^2L^2W^{2\tau}\log(mLW^\tau))$, or $\tilde{O}(L^2W^{2\tau})$, time. Therefore, the asymptotic running time of the estimation algorithm is $\tilde{O}(L^2W^{2\tau})$. (In either case, the time accounted for by the Jain-Vazirani algorithm is dominated by the time accounted for by the Charikar-Guha algorithm.) As the Charikar-Guha algorithm uses only linear space, the space used by the estimation procedure is asymptotically the same as the amount of space required for median storage.

Algorithm In Figure 8.1 we give pseudocode for the k–Median maintenance algorithm. This algorithm maintains the EH data structure which we can use to calculate a k–Median solution and cost when queried. The basic maintenance step is the insertion of the most recent point x_t from the data stream: this procedure specifies what updates to perform on the data structure whenever a new point arrives (and an old one leaves the window). The pseudocode includes references to the *suffix bucket* B_i^* (where $1 \leq i \leq m$) which represents the combination of all buckets that are later (farther to the right) than B_i. These suffixes are not maintained at all times but instead are computed when required. In the following algorithm, they are computed incrementally as follows: B_2^* is B_1; B_3^* is computed as the combination of buckets 1 and 2; B_4^* is computed as the combination of bucket 3 and B_3^*, and so on. The crux of the insertion algorithm is a right-to-left sweep that merges pairs of buckets whose estimated *combined* costs are low.

Algorithm INSERT

1. If there are fewer than k distinct level-0 medians (i.e., points from k different locations in the metric space, so that no k medians can assign all these points for a total cost of 0) in B_1 add the point x_t as a level-0 median in bucket B_1. Otherwise, create a new bucket B_1 to contain x_t and renumber the existing buckets accordingly.
2. If bucket B_m has timestamp more than W, delete it.
3. Calculate $f(B_1)$. Also, calculate $f(B_{2,1})$ and record this value.
4. $i \leftarrow 2$; while $i < m$ do:
   (a) Calculate $f(B_{i+1,i})$ and record this value.
   (b) Calculate $f(B_i^*)$, and record $F(B_i^*)$ as $\max\left(f(B_i^*), f(B_{i-1,i-2})\right)$. (If $i = 2$, simply let $F(B_2^*) = f(B_1)$.)
   (c) Compare $f(B_{i+1,i})$ to the value $F(B_i^*)$ just calculated.
      - If $i = m - 1$, and buckets B_m and B_{m-1} together represent more than W level-0 medians, exit the algorithm now.
      - If $f(B_{i+1,i}) \leq 2F(B_i^*)$, merge buckets B_{i+1} and B_i, calling the joint bucket B_i; record the value calculated for $f(B_{i+1,i})$ as $f(B_i)$ for this new B_i; and renumber the higher-indexed buckets accordingly. Note that now $f(B_i) \leq 2F(B_i^*)$.
      - If the buckets were not merged, leave $f(B_i)$ unchanged, and $i \leftarrow i + 1$.

Figure 8.1: Algorithm INSERT.

We note that this algorithm prevents the creation of buckets that represent more than W level-0 medians apiece. It would not be possible, in any case, for more than one bucket to represent W or more points, since the older of the two would have expired. In fact, a bucket representing W or more points would have to be the oldest, B_m. Therefore, it suffices to prevent the creation of an oldest bucket with more than

W points. We will see that this prevention step does not harm the correctness or efficiency of the algorithm.

Definition 25 *We will say a bucket B_i is* created *if a median is added to B_i or if B_i results from the merging of two buckets. We will call a bucket* safe *if when it is created, one of the following is true:*

- $i = 1$
- $i > 1$ *and* $f(B_i) \leq 2F(B_i^*)$

A bucket continues to be called "safe" as long as it exists, but it ceases to exist when it is merged with another bucket, or dropped from memory because it has expired.

Fact 4 *For $1 \leq i < j \leq 3$, let Y_{ij} denote the portion of the stream from x_{t_i} to x_{t_j}, inclusive, with $t_1 < t_2 < t_3$, and let $C_{12} = \text{OPT-KM}(Y_{12}, N, k)$, $C_{13} = \text{OPT-KM}(Y_{13}, N, k)$, and $C_{23} = \text{OPT-KM}(Y_{23}, N, k)$. Then $C_{12} \leq C_{13}$, and $C_{23} \leq C_{13}$.*

Proof: In $\text{OPT-KM}(Y_{12}, N, k)$ and $\text{OPT-KM}(Y_{13}, N, k)$, the feasible set is the same, but $Y_{13} \supseteq Y_{12}$, and so C' cannot be smaller than C. ■

Lemma 18 *At the moment when Step 4b of the algorithm* INSERT *is executed, the inequality $\text{OPT-KM}(\bigcup_{j=1}^{i-1} Y_j, N, k) \leq F(B_i^*) \leq \beta\text{OPT-KM}(\bigcup_{j=1}^{i-1} Y_j, N, k)$ holds. In this expression, for all $1 \leq j < i$, Y_j is the set of medians represented by bucket B_j at the moment this step is executed.*

Proof: If $i = 2$ when the step is executed, $F(B_2^*)$ is set to $f(B_1)$. Since this value $f(B_1)$ is calculated using an approximation algorithm, $F(B_2^*)$ is guaranteed not to exceed $\beta\text{OPT-KM}(Y_1, N, k)$ and not to fall below $\text{OPT-KM}(Y_1, N, k)$.

If $i > 2$, $F(B_2^*)$ is set to be the maximum of the values $f(B_i^*)$ and $f(B_{i-1,i-2})$. The first of these values was calculated in this step. The second was calculated on an earlier iteration of this loop during the same right-to-left sweep, when the algorithm examined the buckets whose labels were (and are) B_{i-1} and B_{i-2} (with respect to the current value of i) and chose not to merge them. In this case, because

$\text{OPT-KM}(\bigcup_{j=1}^{i-1} Y_j, N, k) \leq f(B_i^*)$, we can infer that $\text{OPT-KM}(\bigcup_{j=1}^{i-1} Y_j, N, k) \leq F(B_i^*)$. The inequality $\text{OPT-KM}(Y_{i-1} \cup Y_{i-2}, N, k) \leq f(B_{i-1,i-2}) \leq \beta \text{OPT-KM}(Y_{i-1} \cup Y_{i-2}, N, k)$ also holds, and by Fact 4, $\text{OPT-KM}(Y_{i-1} \cup Y_{i-2}, N, k) \leq \text{OPT-KM}(\bigcup_{j=1}^{i-1} Y_j, N, k)$. Since $F(B_i^*)$ is set to be the larger of two quantities that are both bounded above by $\text{OPT-KM}(\bigcup_{j=1}^{i-1} Y_j, N, k)$, $F(B_i^*)$ is itself bounded by this quantity, and the lemma follows. ■

Lemma 19 *Consider one execution of the algorithm* INSERT*, corresponding to the insertion of one point. If all buckets are safe just before this execution of the algorithm, then all buckets are safe just after the execution.*

Proof: Recall that if a bucket is safe just after it has been created, it remains safe as long as it exists. Therefore, it is only necessary to show that all buckets created by the algorithm are safe. A bucket is created when a new point is added to it or when two buckets are merged by a sweep operation to create it.

In the first step, a new point is added to B_1, and so bucket B_1 is created in our terminology; because the bucket that has been created is B_1, it is by definition safe. (Note that it makes no difference whether B_1 is receiving its first point.) The second and third steps do not create new buckets.

The fourth step is the right-to-left sweep. The execution of the algorithm ends just after the sweep finishes, and so if all buckets are safe after this sweep, we are done. By definition, B_1 is still safe after the sweep (but note no bucket is merged with B_1 in the sweep). For all other buckets B_i that exist after the sweep, if B_i was not created during the sweep, it is still safe. If it was created during this sweep, $f(B_i) \leq 2F(B_i^*)$, as noted in the description of INSERT, because the former B_{i+1} and B_i were only merged if $f(B_{i+1,i})$ (now stored as $f(B_i)$ per the algorithm) was at most $2F(B_i^*)$. The lemma follows. ■

Lemma 20 *If all buckets are safe at the time t when the algorithm is asked for a clustering solution for the current window, the algorithm produces a set M of $2k$ medians with $\Phi^{W_t}(M) \leq 2^{O(L)}\text{OPT-KM}(W_t, N, k)$.*

Proof: The algorithm produces two sets of k medians: k for the oldest bucket that has not yet expired, B_m, and k for the newer $m-1$ buckets. Let M_1 denote the set of k medians found for B_m and M_2 denote the set of k medians found for B_1 through B_{m-1}. Also, let $Y_{old} \cup Y_{act}$ denote the part of the stream represented by B_m, where Y_{old} is the set of expired points in B_m and Y_{act} is the set of active points in that bucket; let Y_{new} denote the more recent part of the stream, that represented by buckets B_1 through B_m. Note that W_t, the set of points in the current window, is the same as $Y_{act} \cup Y_{new}$. Then we wish to show that $\Phi^{Y_{act} \cup Y_{new}}(M_1 \cup M_2)$ is at most some constant times $\textsc{opt-km}(Y_{act} \cup Y_{new}, N, k)$.

Let us first compare $\Phi^{Y_{new}}(M_2)$ with $\textsc{opt-km}(Y_{new}, N, k)$. Recall that β is an upper bound on the approximation ratio represented by any set of medians calculated by this SWKM algorithm, as guaranteed by the fact that no medians calculated during the algorithm are removed from the original data points by more than L levels of clustering. Because of the way the medians M_2 are produced, we can conclude that $\textsc{opt-km}(Y_{new}, N, k) \leq \Phi^{Y_{new}}(M_2) \leq \beta\textsc{opt-km}(Y_{new}, N, k)$. For the same reason, $\textsc{opt-km}(Y_{old} \cup Y_{act}, N, k) \leq \Phi^{Y_{old} \cup Y_{act}}(M_1) \leq \beta\textsc{opt-km}(Y_{old} \cup Y_{act}, N, k)$.

If all buckets are safe, then when B_m was created, one of the following was true:

1. It was B_1.

2. It was B_i, for some $i > 1$, and its estimated cost $f(B_i)$ at the time was no more than $2F(B_i^*)$ as calculated then.

In the first case, B_m still consists of only level-0 medians; therefore, the expired points Y_{old} are explicitly stored and can be deleted before the k medians for B_m are calculated. In this case, $\Phi^{Y_{act}}(M_1)$ is bounded by a constant factor times $\textsc{opt-km}(Y_{act}, N, k)$, where the constant factor is even smaller than β (since it represents only a single level of clustering). Also, $\Phi^{Y_{new}}(M_2) \leq \beta\textsc{opt-km}(Y_{new}, N, k)$, and so $\Phi^{Y_{act} \cup Y_{new}}(M_1 \cup M_2) \leq \beta\textsc{opt-km}(Y_{act} \cup Y_{new}, Nkk)$.

In the second case, we will show that $\textsc{opt-km}(Y_{old} \cup Y_{act}, N, k)$ is no more than $2\beta\textsc{opt-km}(Y_{new}, N, k)$. First, let $\widehat{Y}$ denote the set of level-0 medians that were represented by B_m at the time it was created; note that the set has not changed since then, so B_m still represents exactly these points, so $\widehat{Y} = Y_{old} \cup Y_{act}$. Also, let $\overline{Y}$ represent the points that were active and more recent than the points in B_m, at the time B_m was created. Note that Fact 4 implies that $\text{OPT-KM}(\overline{Y}, N, k) \le \text{OPT-KM}(Y_{new}, N, k)$, because Y_{new} is a superset of $\overline{Y}$. Therefore, it will suffice to show that $\text{OPT-KM}(\widehat{Y}, N, k) \le 2\beta\text{OPT-KM}(\overline{Y}, N, k)$.

If B_m was initially created as B_2, it had $\text{OPT-KM}(\widehat{Y}, N, k) \le f(B_2) \le 2F(B_2^*) = 2f(B_1) \le 2\beta\text{OPT-KM}(\overline{Y}, N, k)$. Otherwise, if B_m was created as B_i for some $i > 1$, the inequality $\text{OPT-KM}(\widehat{Y}, N, k) \le f(B_i) \le 2F(B_i^*)$ held when B_m was created. Lemma 18 implies that $F(B_i^*) \le \beta\text{OPT-KM}(\overline{Y}, N, k)$, and so $\text{OPT-KM}(\widehat{Y}, N, k) \le 2\beta\text{OPT-KM}(\overline{Y}, N, k)$.

We have shown that because B_m is safe, we know that $\text{OPT-KM}(Y_{old} \cup Y_{act}, N, k) \le 2\beta\text{OPT-KM}(Y_{new}, N, k)$. We want an upper bound for $\Phi^{Y_{act} \cup Y_{new}}(M_1 \cup M_2)$. This quantity is certainly at most $\Phi^{Y_{act}}(M_1) + \Phi^{Y_{new}}(M_2)$, since restricting a median set cannot lower assignment distances. Furthermore, this second quantity is at most $\Phi^{Y_{new}}(M_1) + \Phi^{Y_{old} \cup Y_{act}}(M_2)$. Then:

$$\begin{aligned}
\Phi^{Y_{act} \cup Y_{new}}(M_1 \cup M_2) &\le \Phi^{Y_{new}}(M_1) + \Phi^{Y_{old} \cup Y_{act}}(M_2) \\
&\le \Phi^{Y_{new}}(M_1) + \beta\text{OPT-KM}(Y_{old} \cup Y_{act}, N, k) \\
&\le \Phi^{Y_{new}}(M_1) + 2\beta^2\text{OPT-KM}(Y_{new}, N, k) \\
&\le (2\beta^2 + \beta)\text{OPT-KM}(Y_{new}, N, k)
\end{aligned}$$

The lemma follows. ■

Because the algorithm preserves the invariant that all buckets are safe, the hypothesis of the previous lemma is true; therefore, the answer produced in response to a query for a k–Median solution is indeed a constant-factor bicriterion approximation.

Lemma 21 *Suppose all non-zero inter-point distances in the stream X are at most $P(W)$, and at least $1/Q(W)$, where P and Q are non-zero polynomials in W. Then the number of buckets never exceeds $O(\log W)$.*

Proof: Without loss of generality, assume $Q(W)$ is identically 1 so that the smallest non-zero distance is 1 (otherwise, multiply all distances by $Q(W)$ and replace P with $P \cdot Q$). For all buckets B_i, let Y_i denote the set of level-0 medians represented by B_i.

Note that the only time during the algorithm when the number of buckets increases is when the bucket B_1 has just been "closed" because it has accumulated $k+1$ distinct points, and a new point enters, so that a bucket is created for the new point. Therefore, if we can claim that just before the new bucket was created there were at most $\alpha(\log W)$ buckets for some constant α, we can also claim that the number of buckets never exceeds $(\alpha+1)(\log W)$, and the theorem will follow. Since there is a right-to-left sweep just before every new point enters the window, it will suffice to show that just after a sweep the number of buckets is at most $\alpha(\log W)$ for some constant $\alpha > 0$.

In the last sweep, for every i from 2 to $m-1$, values of $f(B_{i+1,i})$ and $F(B_i^*)$ were calculated and recorded; in what follows, we will use the expressions $f(B_{i+1,i})$ and $F(B_i^*)$ to denote the most recently recorded versions of these quantities. Recall that if buckets B_2 and B_3, say, were merged, they became bucket B_2, so that a new value $f(B_{3,2})$ was calculated for B_3 (formerly B_4) and B_2 (the bucket that resulted from the merge of the former B_3 and B_2). The value of $f(B_{3,2})$ would get continually overwritten until the bucket resulting from the last merge was left separate from the newest B_3.

Consider the bucket that is currently B_2. Since it is no longer B_1, it has been "closed," and points have been added to B_1; it must be that B_2 contains more than k distinct points; then $\text{OPT-KM}(Y_2, N, k) \geq 1$, as all non-zero interpoint distances are at least 1. It is also true, then, that $f(B_{2,1}) \geq \text{OPT-KM}(Y_2 \cup Y_1, N, k) \geq 1$.

Now consider the bucket currently labeled B_3. It may have been created during the most recent sweep, but it was ultimately left, and not merged with B_4, when the sweep passed it. Therefore, $f(B_{4,3}) > 2F(B_3^*)$, as guaranteed by the sweep. Recall that $F(B_3^*) \geq f(B_{2,1})$ by the way $F(B_3^*)$ was calculated during the sweep. In fact, for all buckets B_{2j}, B_{2j-1} with $4 \leq 2j < m-1$, because the sweep left B_{2j-1} without merging it with B_{2j}, we know $f(B_{2j,2j-1}) > 2F(B_{2j-1}^*) \geq 2f(B_{2j-2,2j-3})$. Note that this property does not necessarily hold for B_m or for B_{m-1}: these buckets may have been left unmerged not because they represented high enough clustering costs, but because they collectively represented more than W level-0 medians. By induction, $f(B_{2j,2j-1}) > 2^{j-1}f(B_{2,1})$, and so $f(B_{m-2,m-3})$ (or, if $m-2$ is odd, $f(B_{m-3,m-4})$)

exceeds $2^{m/2-2} f(B_{2,1})$ (or $2^{(m-3)/2-1} f(B_{2,1})$). For simplicity, assume $m-2$ is even.

By assumption, all distances are bounded by $P(W)$, so $\textsc{opt-km}(W_t, N, k) \leq WP(W)$, where W_t is the set of points currently in the window just before the next new point is read. Since $\textsc{opt-km}(Y_m \cup Y_{m-1}, N, k) \leq \textsc{opt-km}(W_t, N, k)$, and since $f(B_{m-2,m-3})$ cannot exceed $\beta\textsc{opt-km}(Y_{m-2} \cup Y_{m-3}, N, k)$, it follows that $2^{m/2-2} f(B_{2,1}) < f(B_{m-2,m-3}) \leq \beta\textsc{opt-km}(W_t, N, k) \leq \beta WP(W)$. As $f(B_{2,1}) \geq 1$, we have $2^{m/2-2} \leq \beta WP(W)$ and so $m-2 \leq 2(1+\lg(\beta WP(W)))$, which is $O(\log W)$. Then m is also $O(\log W)$, and the lemma follows. ■

Zero-Cost Solutions In fact, there is an important issue that we have not addressed: what happens if more than W^τ level-0 medians accumulate in bucket B_1 before there are $k+1$ distinct points (points from $k+1$ distinct locations in the metric space) in that bucket. The bucket cannot be "closed," *i.e.*, it cannot become B_2, until it has points from $k+1$ different locations; otherwise, we could create a very large number of such buckets. However, if whenever W^τ medians accumulate at level i we replace them with k level-$(i+1)$ medians (one for each of the distinct locations), we may make W^τ level-$(L-1)$ medians in this bucket before any medians expire, and this bucket will overflow.

The answer is not to make any medians higher than level 1 until a point from a $(k+1)$th location arrives. If the first W^τ level-0 medians to arrive in B_1 are from only k distinct locations (or fewer), we will make one level-1 median for each value, and set the weights and timestamps of these medians accordingly (their costs, of course, will be 0). Until the arrival of a point from a $(k+1)$th location, every time W^τ level-0 medians accumulate, we will make two updates to the existing level-1 medians and expunge the level-0 medians. The first update will be to the timestamps of the level-1 medians: the timestamp of each level-1 median will be set to that of the most recent level-0 median at that location. The second update will be to the weights of the level-1 medians: for each level-1 median, we will raise its weight by the number of new level-0 medians at its location, unless doing so would raise its weight above W, in which case we will set its weight to W. Every time a new level-0 median enters from one of the existing locations, we will also check to see if any level-1 median has expired, and, if so, expunge it from the bucket. (Note that at most one median can expire at any given time.) In this way, the bucket will always consist of at most W^{τ} level-0 medians and k level-1 medians each of weight at most W.

Clearly this method will work as long as the bucket never contains points from more than $(k+1)$ distinct locations: Until a point from a $(k+1)$th location arrives, the k–Median cost will always be zero, and so it will not matter if the weights of the level-1 medians are inaccurate. When a point from a new location arrives, the outcome will depend on these weights. If the weights in B_1 sum to W or more, the bucket will be closed, and the new point will start its own bucket. Because the algorithm will always prevent a pair of buckets from merging if their combined weight exceeds W, the bucket of only k distinct values (now B_2) will never be merged. The medians in this bucket will therefore never be clustered with other points, and so their weights will be irrelevant. If, instead, the weights in B_1 sum to less than W, they are still accurate (in fact the solution represented by any level-1 medians in this bucket is optimal), and so B_1 will admit the new point (from the $(k+1)$th location) and immediately close.

Improving the Running Time The above k–median algorithm is efficient in terms of memory, but not in running time. After each point arrives, the algorithm checks every bucket to see how many buckets to the left can merge with it. In order to reduce the per-item processing time, we can use a batch processing technique to reduce the amortized processing time to $O(k \log W)$ per element insertion. To this end, we draw a distinction between data structure maintenance and output production. We assume that the algorithm will not be called upon to *produce* an updated clustering as each new point is added. Instead, it may maintain sufficient statistics so as to be able, upon request, to generate a valid clustering quickly. Requests for the current clusters may come at arbitrary points in time, but we assume that they will not come too frequently. In short, we distinguish between update time and query time for the data structure.

We can modify the insertion Algorithm so that it does not execute the bucket combination procedure until B_1 has accumulated $\frac{k}{\tau^3} W^{2\tau} \log W$ points that together come from at least $k+1$ distinct locations. When B_1 has reached this stage, it is said to have "filled up"; it closes, and its points are clustered (using the algorithm of Indyk[59]) to produce $2k$ level-1 medians. Even as new buckets are created, so that B_1 may be renumbered B_2, B_3, and so on, it retains its original data points until it expires, is combined with another bucket, (when it is numbered B_i) satisfies $f(B_i) \leq 2F(B_i^*)$. (Note that if B_i is combined with another bucket B_{i-1}, the resulting combined bucket B_{i-1} will necessarily satisfy $f(B_{i-1}) \leq 2F(B_{i-1}^*)$). After clustering the points in B_1, we execute the remaining steps in the algorithm (discarding expired buckets and performing the right-to-left sweep).

Using the same analysis as for the algorithm SMALLER-SPACE in Chapter 4, we see that the time required for the maintenance of the data structure (rather than for producing medians on demand) is dominated by the time required for the first level of clustering, since at each level of clustering the number of points decreases geometrically. Therefore, the amount of time required for the maintenance of $\frac{k}{\tau^3}W^{2\tau}\log W$ points is the amount of time required for their first clustering, which is $\tilde{O}\frac{k^2}{\tau^3}W^{2\tau}\log W$ if we use Indyk's $\tilde{O}(nk)$-time k–Median algorithm. Per point, this amount is $O(k\log W)$.

This modification to the algorithm does not affect its correctness or efficiency. Lemma 18 still holds, since a sweep is still only performed after B_1 closes. Lemma 19 still holds, because the bucket creation and sweep steps still do not create any unsafe buckets. Lemma 20 still holds because it depends only on the safety of all buckets, which has the same meaning as before. Finally, Lemma 21 still holds because the sweep only occurs after B_1 is closed, and so values of f and F are not recalculated until B_1 is complete. In essence, just as before, the algorithm performs a sweep and combines as many buckets as possible before more than one new bucket is created.

Now we can compute the space and time used by the estimation algorithm. Whenever a current clustering is desired, if B_m still retains all its level-0 medians, we first discard the expired ones and cluster the remaining level-0 medians using the algorithm of Indyk [59], producing $2k$ medians. Since there are at most $(\frac{k}{\tau^3}W^{2\tau}\log W)$ active points in B_m, this step takes $\tilde{O}(\frac{k^2}{\tau^3}W^{2\tau}\log W)$ time and space. Now we cluster these $2k$ medians with all medians from buckets B_1 through B_{m-1}, using the Charikar-Guha algorithm, producing $2k$ medians. (Again, we can use the Jain-Vazirani algorithm to cluster these medians to produce just k, using only a negligible amount of time and space). In this step, if a bucket B_i with $i < m$ has retained its level-0 medians, note that none of them has expired, so that we can cluster the level-1 medians which were computed when this bucket "closed." This second clustering is of at most mLW^τ medians and requires $O(m^2L^2W^{2\tau}\log(mLW^\tau))$, or $\tilde{O}(L^2W^{2\tau})$, time. It runs in space linear in the number of medians, and so does not require more space, asymptotically, than required already for median storage. Therefore, in the case that B_m retained its original level-0 medians, estimation can be completed using $\tilde{O}((m^2L + \frac{k^2}{\tau^3})W^{2\tau}\log W)$ $= \tilde{O}(\frac{k^2}{\tau^3})W^{2\tau})$ time and space.

If B_m does not retain its level-0 medians, we first cluster the medians from B_m using the Charikar-Guha algorithm, and cluster the resulting $2k$ medians down to k using the Jain-Vazirani algorithm. There are at most LW^τ medians in this bucket, and so this part of the clustering takes $O(L^2W^{2\tau}\log(LW^\tau))$ time. We then do the same with the medians from buckets B_1 through B_m, of which there are fewer than mLW^τ. Clustering these takes $O(m^2L^2W^{2\tau}\log(mLW^\tau))$ time. For the final medians, we present the k medians found for B_m and the k medians found for the other buckets. The total running time in this case is $\tilde{O}(L^2W^{2\tau})$. Because only the Charikar-Guha algorithm is used in this case, the space required for estimation is no larger than that used already for median storage.

Therefore, the running time and space are asymptotically higher if B_m has retained its level-0 medians, but, in either case, both are bounded by $\tilde{O}(\frac{k^2}{\tau^3})W^{2\tau})$.

Putting everything together we have Theorem 23. The memory bound follows from Lemma 21. The space used to maintain the original data points for buckets whose f value is more than twice the F value of their summary bucket, dominates the space used to maintain the EH itself. The approximation guarantee follows from Lemma 20, while the amortized maintenance time and query time follow from the discussion in the last several paragraphs.

Theorem 23 *The revised algorithm provides a* $(2, 2^{O(1/\tau)})$ *bicriterion approximation to the* SWKM *problem for any* $\tau < 1/2$. *It uses* $\tilde{O}(\frac{k^2}{\tau^3}W^{2\tau})$ *memory, and requires* $O(k\log W)$ *amortized maintenance time per data point. The query time for the data* *structure that we maintain is* $\tilde{O}(\frac{k^2}{\tau^3}W^{2\tau})$.

We have described a bicriterion approximation algorithm that produces $2k$ medians as opposed to k. Next, we build upon this algorithm to get an algorithm that produces exactly k medians, while preserving the approximation guarantee of $2^{O(1/\tau)}$.

8.3.2 Producing Exactly k Clusters

Suppose that a query is made for a clustering of the current window, and that the bucket B_m still retains its level-0 medians; the algorithm already described produces k medians in this case. In the case that the original level-0 medians from B_m have been discarded, in order to produce exactly k medians, we change the algorithm in two ways: First, with each median, we maintain additional information that lets us estimate to within a constant factor the number of active data points that are assigned to it. Second, we change the estimation procedure that produces a solution whenever a request is made for the current clusters. Rather than separately clustering the oldest bucket B_m and the suffix bucket B_m^*, we cluster the medians from all the buckets together. However, the weights of the medians from bucket B_m are adjusted so that they only reflect the contribution of active data points, discounting the contribution of the expired points.

Let $c > 1$ be a constant. Recall that each median z in the last bucket B_m has an associated cost $c(z)$ and weight $w(z)$, where $w(z)$ is the number of original points that have been assigned to z, some of which may have expired. Suppose that for each z we know a revised weight $w'(z) \leq \min\{w(z), c \cdot a(z)\}$, where $a(z)$ is the true number of active level-0 medians assigned to z. Let $f(B_m)$ be as calculated during the last sweep (with the original, not the revised, weights). Also let $F(B_m^*)$ be as calculated during the last sweep, and let $f(B_m) \leq F(B_m^*)$. Note that this inequality is guaranteed to hold if the level-0 medians from B_m have not been retained. Let Z'_m denote the set of medians from B_m, with the revised weights replacing the original weights, and let Z_m^* denote the set of medians from buckets B_1 through B_{m-1}. Then consider the k–Median instance consisting of $Z_m^* \cup Z'_m$. We will show this instance has a solution whose cost is bounded in terms of $\text{OPT-KM}(W_t, X, k)$, the optimal solution for all currently active level-0 medians.

Lemma 22 $\text{OPT-KM}(Z'_m \cup Z^*_m, X, k) \le F(B^*_m) + c(f(B_m) + \text{OPT-KM}(W_t, X, k))$.

Proof: We must find a feasible k–Median solution for $Z'_m \cup Z^*_m$. If M is the set of medians realizing $\text{OPT-KM}(W_t, X, k))$, then M is clearly a feasible solution for $Z'_m \cup Z^*_m$.

First we will bound $\Phi^{Z^*_m}(M)$. Let Y^*_m denote the set of original points represented by buckets B_1 through B_{m-1}. For every median z in $B_1 \cup \ldots \cup B_m$, let $\text{CLUSTER}(z)$ denote the set of active level-0 medians assigned to z. Also, for every point y in the current active window W_t, let $\text{MED}(y)$ denote the highest-level median to which it has been assigned.

Let each median z in $B_1 \cup \ldots \cup B_{m-1}$ "nominate" the $y' \in \text{CLUSTER}(z)$ that minimizes $\text{dist}(z, y') + \text{AD}(y', M)$, the sum of the distance to z and the distance to the optimal medians M. What is the cost of assigning z to $\text{ASG}(y', M)$, the median in M to which y' (z's nominated point) is assigned? This cost is at most the weight of z, times the distance from z to $\text{ASG}(y', M)$, which, by the triangle inequality, is at most $w(z) \cdot (\text{dist}(z, y') + \text{AD}(y', M))$. Because $w(z)$ is exactly $|\text{CLUSTER}(z)|$, this quantity is exactly $\sum_{y \in \text{CLUSTER}(z)} \text{dist}(z, y') + \text{AD}(y', M)$, which, by choice of y', is at most $\sum_{y \in \text{CLUSTER}(z)} \text{dist}(z, y) + \text{AD}(y, M)$. Therefore, we have shown that the cost of assigning z to a median in M is at most $\sum_{y \in \text{CLUSTER}(z)} \text{dist}(z, y) + \text{AD}(y, M)$. The cost of assigning *all* members of Z^*_m to medians in M is therefore at most

$$\sum_{y \in Y^*_m} (\text{dist}(y, \text{MED}(y)) + \text{AD}(y, M)).$$

We will return to this quantity shortly, but first we will bound the cost of assigning the medians in B_m to M.

The cost of assigning some z in Z'_m to M will be calculated slightly differently, for two reasons. First, the weight we will use for z will be the revised weight $w'(z)$ which does not exceed $w(z)$ but may be less, and second, we will not be able to "nominate" expired points.

Let each median $z \in Z'_m$ nominate the member y' of $\textsc{cluster}(z)$ minimizing $\text{dist}(z, y') + \textsc{ad}(y', M)$. We will bound the cost of assigning z to M by $w'(z) \cdot (\text{dist}(z, y') + \textsc{ad}(y', M))$, which, because $w'(z)$ is at most c times the number of active level-0 medians assigned to z, is at most $\sum_{y \in \text{CLUSTER}(z)} c \cdot (\text{dist}(z, y') + \textsc{ad}(y', M))$. By choice of y', this sum is at most $\sum_{y \in \text{CLUSTER}(z)} c \cdot (\text{dist}(z, y) + \textsc{ad}(y, M))$. Therefore, the cost of assigning all z in Z'_m to the members of M is at most

$$\sum_{y \in Y'_m} c \cdot (\text{dist}(y, \textsc{Med}(y)) + \textsc{ad}(y, M)),$$

where Y'_m is the set of active level-0 medians represented by B_m.

The cost of assigning $Z^*_m \cup Z'_m$ to M is bounded by the sum of the two upper bounds we have calculated, which is

$$\begin{aligned}
\sum_{y \in Y^*_m} (\text{dist}(y, \textsc{Med}(y)) + \textsc{ad}(y, M)) + \sum_{y \in Y'_m} c \cdot (\text{dist}(y, \textsc{Med}(y)) + \textsc{ad}(y, M)) \\
\le F(B^*_m) + \sum_{y \in Y^*_m} \textsc{ad}(y, M) + \sum_{y \in Y'_m} (c \cdot \textsc{ad}(y, M)) + cf(B_m) \\
\le F(B^*_m) + cf(B_m) + \sum_{y \in Y^*_m \cup Y'_m} (c \cdot \textsc{ad}(y, M)) \\
= F(B^*_m) + cf(B_m) + \sum_{y \in W_t} (c \cdot \textsc{ad}(y, M)) \\
= F(B^*_m) + cf(B_m) + c \cdot \Phi^{W_t}(M)
\end{aligned}$$

■

Lemma 22 shows that, if we had a way of calculating a revised weight $w'(z)$ for every median z in buckets B_1 through B_m, as described above, then the k–Median instance consisting of all medians in buckets B_1 through B_{m-1}, and all medians in B_m with the revised weights replacing the original weights, would have a solution costing at most $c(f(B_m) + \textsc{opt-km}(W_t, X, k)) + F(B^*_m)$. Note that $F(B^*_m) \le \beta\textsc{opt-km}(Y^*_m, X, k) \le \beta\textsc{opt-km}(W_t, X, k)$, where Y^*_m is the set of active points in buckets B_1 through B_{m-1}. The first inequality follows from the way $F(B^*_m)$ is calculated; the second follows from Fact 4, since W_t contains Y^*_m. Also, by hypothesis,

$f(B_m) \leq 2F(B_m^*)$, and so we have a bound of $((2c+1)\beta+1)$OPT-KM(W_t, X, k) on the cost of this new instance. If we solve the instance using a constant-factor approximation k–Median algorithm, we will find a solution whose cost is still $O(\beta)$ OPT-KM(W_t, X, k). Therefore, we need only show how to calculate the revised weights.

We show how to estimate the number of active data points assigned to each median, to within a factor of 2, using at most $k^L \log W$ space per median. Suppose that all medians in buckets B_1 through B_m are assigned indices, so that the medians are labeled m_1 through m_p, where p is the number of medians. For any median m_j, consider the following imaginary stream $A^{(j)}$ of $0-1$ values: For each actual data point x_i, $A_i^{(j)} = 1$ if x_i was assigned to median m_j, and $A_i^{(j)} = 0$ otherwise. Note that if the median m_j belongs to a particular bucket, $A_i^{(j)}$ will be zero outside the interval covered by that bucket. Estimating the number of active data points assigned to m_j is the same as counting the number of ones from the last W elements of $A^{(j)}$. If the stream $A^{(j)}$ were presented explicitly, this problem could be solved using an EH data structure that uses $O(\frac{1}{\epsilon'} \log W)$ memory and provides an estimate at all times that has error at most ϵ', as shown by Datar *et al.* [26]. We set ϵ' equal to 1/2 for our estimation. Unfortunately, the stream A_j is not presented explicitly. Medians at level 2 and above are produced by the combination of lower-level medians. All points previously assigned to lower-level medians are now assigned to a higher-level median, but there is no way to reconstruct the exact arrival order (timestamps) of those points and simulate the stream $A^{(j)}$ since the original points have been discarded. In order to estimate the number of active points for such higher-level medians we use the following two observations about EHs over $0-1$ data streams:

1. Consider two Exponential Histograms (EH1 and EH2) corresponding to two $0-1$ data streams for the same error parameter ϵ' and the same window size W. If all the 1's in EH1 have arrival times strictly greater than the arrival times for 1's in EH2 then we can combine the two EHs to get an EH for the union of the two streams in time $O(\log W)$. We say such EHs are *non-overlapping*. This "combination" can be easily achieved by placing the buckets of the two EHs one after the other and then making a sweep from right to left to combine the buckets. This process takes $O(\log W)$ time, and the combined EH uses

$O(\log W)$ buckets or words of memory.

2. If the two data streams (respectively two EHs) are overlapping, *i.e.*, they do not satisfy the property that all the 1's in one of them arrive before the 1's in the other, then we can maintain two separate EHs corresponding to them to answer count queries over the union of the two streams. The EHs corresponding to these overlapping streams are called *overlapping* EHs.

When a higher-level median is formed by the clustering of lower-level medians we assign all the EHs of the lower-level medians to the higher-level medians. We combine as many non-overlapping EHs as possible, but those that are overlapping are maintained separately. We claim that a median at level l needs to maintain at most k^{l-1} EHs, where each EH requires $O(\log W)$ memory. Since L is the maximum level of any median, the amount of extra memory required per median is $O(k^L \log W)$.

We prove the claim by induction on the level of the medians. Original points (level-0 medians) do not maintain an EH; they simply maintain their timestamps. The $A^{(j)}$'s corresponding to them have 1 in exactly one position and zero everywhere else. When level-0 medians are clustered to form k level-1 medians, for each level-1 median thus formed we can insert the level-0 medians assigned to it in the sorted (decreasing) order of timestamps so that we get a single EH for a level-1 median. Note that the k level-1 medians obtained in such a clustering may have overlapping EHs. A pair of level-i medians are called *conflicting* if they are created at the same time and have overlapping EHs. A level-i median can conflict with at most $k-1$ other level-i medians—the ones created during the same clustering of level-$(i-1)$ medians.

Consider what happens when we cluster level-1 medians to form a level-2 median. Of all the level-1 medians assigned to a single level-2 median, any given level-1 median can conflict with at most $k-1$ other level-1 medians. Consequently, we can organize the assigned medians into k groups such that within a group there are no conflicting medians. For each of these groups the corresponding EHs will be non-overlapping and can be combined to form a single EH. Thus a level-2 median will have at most k EHs. By the inductive hypothesis, assume each level-l median has at most k^{l-1}

overlapping EHs. By the same arguments as those given above, when a level-$l+1$ median is formed, the level-l medians assigned to it can be organized into k groups so that within each group there are no conflicting medians. Moreover, since every level-l median within a group can contribute at most k^{l-1} overlapping EHs, we can combine all of them to form k^{l-1} overlapping EHs. Thus, across the k groups the level-$(l+1)$ median has a total of k^l EHs. Th claim follows.

The combination procedure for EHs described above is executed during the clustering of higher-level medians, *i.e.*, medians from level 1 and above. It is done in the last phase when points are assigned to the cluster centers. The time taken to assign every point is now $O(k^L \log W)$ instead of $O(1)$. Since we use a quadratic running time algorithm to cluster the higher-level medians this extra assignment time does not increase the asymptotic running time for clustering provided $k^L \log W < W^\tau$. Moreover, the asymptotic space requirement of the algorithm also remains unchanged. Consider the two different types of bucket: Buckets that retain their original points have $\frac{k}{\tau^3} W^{2\tau} \log W$ original points and at most k level-1 medians. All other buckets are formed by combination of other buckets and do not retain their original points. They contain at most LW^τ higher-level medians and, as explained above, each such median requires at most $O(k^L \log W)$ memory for estimating number of active points assigned to it. If $k^L \log W < W^\tau$ then the memory requirement of such a bucket is also $O(LW^{2\tau})$. It follows that the total memory requirement is $\tilde{O}(\frac{k^2}{\tau^3} W^{2\tau})$. The approximation factor for this algorithm has been established in Lemma 22, and so the final result, summarized in the theorem below, follows.

Theorem 24 *For $\tau < 1/2$, the SKWM algorithm provides a $2^{O(1/\tau)}$-approximation. It uses $\tilde{O}(\frac{k^2}{\tau^3} W^{2\tau})$ memory and requires $O(k \log W)$ amortized maintenance time per data point, provided $k^L \log W < W^\tau$.*

Chapter 9

Conclusion and Open Problems

We have seen algorithms for variants of the k–Median and k–Center problems under different data models that capture some important aspects of emerging and growing data sets. In Chapter 3 we saw a constant-factor approximation algorithm for k–Median that runs in $O(k(\frac{k^2}{\epsilon}\log k)^2 \log(\frac{k}{\epsilon}\log k))$ time and $O(\frac{k}{\epsilon}\log\frac{k}{\epsilon})$ space, independent of the number of data points. In Chapter 4 we examined constant-factor approximation algorithms for k–Median under the data stream model that required space polynomial in the number of points, and that ran in $\tilde{O}(nk)$ or $O(n^{1+\rho})$ time, where n is the number of data points, k the number of clusters, and $\rho > 1$ a constant such that the algorithm has access to enough memory to store at least n^ρ points. In Chapter 5 an improved constant-factor approximation algorithm requiring just $O(k\log^2 n)$ space. A k–Median heuristic was introduced in Chapter 6, which also included an experimental comparison of the results of this heuristic with those of the popular k–Means heuristic, and a comparison of the performance of the divide-and-conquer approach introduced in Chapter 4 with that of the BIRCH [106] heuristic. Chapter 7 contained constant-factor bicriterion approximation algorithms for outlier-robust formulations of k–Median and k–Center, and for the asymmetric k–Median problem, on streams. The algorithm for asymmetric k–Median relaxes the number of centers by a factor of $\log\Delta$, where Δ is the ratio of the diameter of the point set to the smallest inter-point distance, while the algorithms for k–Median and k–Center with outliers relax the fraction of outliers. Finally, in Chapter 8, we introduced a constant-factor algorithm for a sliding-window k–Median problem, running in $\tilde{O}(\frac{k^2}{\tau^3}W^{2\tau})$ space and using just $O(k \log W)$ amortized update time, where $\tau < 1$ is a constant.

A number of open problems remain in this area. One problem is to narrow the gap between the lower and upper bounds on the best possible approximation ratio achievable in polynomial time for the k–Median problem; currently the lower bound is $1 + 1/e$ and the upper bound is $3 + 2/k$. An interesting subject of further experimentation would be to explore further the "practical" approximation ratios of various heuristics and approximation algorithms, that is, the approximation ratios achieved in practice by these algorithms.

Another open question is whether it is possible to provide a constant-factor approximation algorithm for the sliding-window k–Median algorithm in polylogarithmic, rather than polynomial, space. Whereas the divide-and-conquer algorithm from Chapter 4 was usable in the Exponential Histogram framework as described in Chapter 8, the polylogarithmic-space algorithm from Chapter 5 may not be easily extensible to the sliding-window domain.

The relationships among space, running time, approximation factor, and, in the case of the data stream model, the number of passes, is still not well understood. For example, it may be provably impossible to obtain a constant-factor approximation to the k–Median problem in less than $\log n$ space using only a single pass, but a second pass may enable a $\sqrt{\log n}$-space algorithm to obtain a constant-factor approximation. The discovery of more lower bounds on the amount of space and the number of passes required for constant-factor approximations would shed more light on the currently-known upper bounds, and might well lead to the discovery of better algorithms.

Appendix A

$\Theta(n)$-Time Implementation of GAIN(x)

We present the $\Theta(n)$-running time implementation of the GAIN procedure from the Charikar-Guha [18] local-search algorithm (where n is the number of points in the input). We include the steps that calculate how much the savings would be if x were opened as a facility, and the steps that open x, close other facilities, and perform reassignments if opening x is found to be advantageous. The procedure assumes the existence of n-element arrays NUM-ASSIGNED and ASSIGNED-TO as follows. For every $y \in Y$, NUM-ASSIGNED $[y]$ gives the number of members of Y that have been assigned to y (if y is not a facility, this number is 0, and otherwise it is at least 1, since y must be assigned to itself), and ASSIGNED-TO $[y]$ gives the facility to which y has been assigned. The algorithm works in five phases, each taking $O(n)$ time. In the first phase (Step 4), the algorithm sees which points would lower their assignment distance by being assigned to x. Every point whose assignment cost would decrease if it became assigned to x records the amount by which its cost would decrease, and adds this to a running total of the savings (or extra cost) that would result if x became a new facility. Every point whose assignment cost would *increase* (or remain the same) if it became assigned to x (including all current facilities, whose current assignment cost is 0) record the amount by which their assignment distance would increase, adding it to a value stored by their current facility. In the next phase

(Step 5), every facility checks this stored value and compares it to f; if the former is smaller, this facility would do better to close, and assign itself and all its members to x, and so it adds the savings to the running total. If the running total of savings now indicates that opening x as a facility (or, in the case that x was already open, performing the advantageous closings and reassignments) would save cost overall, the algorithm executes three phases (Steps 7 through 9) that perform all the specified closings and reassignments. (Otherwise, these phases are skipped.)

Algorithm GAIN(x)

1. Set SAVE $= 0$ if x is already a facility; otherwise set SAVE $= -f$.
2. Make a copy of ASSIGNED-TO called TEMP-ASSIGNED-TO.
3. Make a new, n-element array called MEMBER-DISTANCE-CHANGE, with, for all $y \in Y$, MEMBER-DISTANCE-CHANGE[y] set to 0.
4. For each element y: let $d_1 = \text{dist}(y, x)$, and let $d_2 = \text{dist}(y, \text{ASSIGNED-TO}[y])$, and
   - if $d_1 < d_2$:
     (a) Change TEMP-ASSIGNED-TO[y] to x.
     (b) Add $(d_2 - d_1)$ to SAVE.
   - Otherwise add $d_1 - d_2$ to MEMBER-DISTANCE-CHANGE[ASSIGNED-TO[y]].
5. For all $y \in Y$, if y is a facility and MEMBER-DISTANCE-CHANGE[y] $< f$, add ($f-$MEMBER-DISTANCE-CHANGE[y]) to SAVE.
6. If SAVE is 0 or less, quit; otherwise continue.
7. For all $y \in Y$, if TEMP-ASSIGNED-TO[y] $= x$ but ASSIGNED-TO[y] $\neq x$:
   (a) Subtract 1 from NUM-ASSIGNED[ASSIGNED-TO[y]].
   (b) Add 1 to NUM-ASSIGNED[x].
   (c) Change ASSIGNED-TO[y] to x.
8. For all $y \in Y$, if y is a facility with MEMBER-DISTANCE-CHANGE[y] $< f$:
   (a) Add 1 to NUM-ASSIGNED[x].
   (b) Change ASSIGNED-TO[y] to x.
   (c) Set NUM-ASSIGNED[y] $= 0$. This is the signal that y is closing.
9. For all $y \in Y$ if NUM-ASSIGNED[ASSIGNED-TO[y]] $= 0$, the facility to which y was assigned has now closed, so:
   (a) Add 1 to NUM-ASSIGNED[x].
   (b) Change ASSIGNED-TO[y] to x.

Figure A.1: Algorithm GAIN

Bibliography

[1] R. Agrawal, J.E. Gehrke, D. Gunopulos, and P. Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In *Proc. SIGMOD*, 1998.

[2] M. Ajtai, T. S. Jayram, R. Kumar, and D. Sivakumar Approximate counting of inversions in a data stream. *Proc. 34th STOC*, 2002.

[3] N. Alon, S. Dar, M. Parnas and D. Ron. Testing of clustering. *Proc. 41st FOCS*, 2000.

[4] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking Join and Self-Join Sizes in Limited Storage. *Proc. 18th PODS* pp. 10-20, 1999.

[5] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *JCSS* 58(1): 1999.

[6] M. Ankerst, M. Breunig, H. Kriegel, and J. Sander. OPTICS: Ordering Points To Identify the Clustering Structure. In *Proc. SIGMOD*, 1999.

[7] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k–median and related problems. *Proc. STOC*, 1998.

[8] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k–median and facility location problems. *Proc. STOC*, 2001.

[9] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. "Models and Issues in Data Stream Systems." In *Proceedings of the 21st ACM Symposium on Principles of Databases Systems (PODS)*, 2002, pages 1–16.

[10] B. Babcock, M. Datar, and R. Motwani. "Sampling from a Moving Window over Streaming Data." In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002, pages 633–634.

[11] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. "Sliding Window Computations over Data Streams." Technical Report. Available online at `http://dbpubs.stanford.edu:8090/pub/2002-25`.

[12] B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. "Sliding Window Computations over Data Streams." In *Proc. PODS*, 2003.

[13] N. Bansal, A. Blum, S. Chawla. Correlation clustering. *Proc. FOCS*, 2002.

[14] Z. Bar-Yossef, S. R. Kumar, and D. Sivakumar. Reductions in streaming algorithms, with an application to counting triangles in graphs. *Proc. SODA*, 2002.

[15] X. Boyen. Inference and Learning in Complex Stochastic Processes. Ph.D. Dissertation, Stanford University, 2002.

[16] P. S. Bradley, U. M. Fayyad, and C. Reina. "Scaling clustering algorithms to large databases." In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD)*, 1998, pages 9–15.

[17] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *Proc. STOC*, 1997.

[18] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k–Median problems. *Proc. FOCS*, 1999.

[19] M. Charikar, S. Guha, É. Tardos, and D. Shmoys. A constant factor approximation algorithm for the k–Median problem. *Proc. STOC*, 1999.

[20] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. *Proc. SODA* 2001.

[21] M. Charikar, L. O'Callaghan, and R. Panigrahy. "Better Streaming Algorithms for Clustering Problems." In *Proc. of 35th ACM Symposium on Theory of Computing (STOC)*, 2003.

[22] M. Charikar and R. Panigrahy. Clustering to minimize the sum of cluster diameters. *Proc. STOC*, 2001.

[23] F. A. Chudak. Improved approximation algorithms for uncapacitated facility location. In R. E. Bixby, E. A. Boyd, and R. Z. Ríos-Mercado, editors, *Integer Programming and Combinatorial Optimization*, volume 1412 of *Lecture Notes in Computer Science*, pages 180–194, Berlin, 1998. Springer.

[24] C. Cortes, K. Fisher, D. Pregibon, and A. Rogers. Hancock: A language for extracting signatures from data streams. In *Proc. ACM SIGKDD Intl. Conf. on Knowledge and Data Mining*, 2000.

[25] S. Dasgupta. Learning mixtures of gaussians. *Proc. FOCS*, 1999.

[26] M. Datar, A. Gionis, P. Indyk, and R. Motwani. "Maintaining Stream Statistics over Sliding Windows." In *Proceedings of Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002, pages 635–644.

[27] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum-likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc. Ser. B*, 1984, 39.

[28] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering in large graphs and matrices. *Proc. SODA*, 1999.

[29] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1972.

[30] M. Dyer and A. M. Frieze. A simple heuristic for the p-center problem. *Operations Research Letters*, v. 3, 1985.

[31] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *Proc. KDD*, 1996.

[32] F. Farnstrom and J. Lewis and C. Elkan. True Scalability for Clustering Algorithms. In *SIGKDD Explorations*, 2000.

[33] T. Feder and D. Greene. Optimal Algorithms for Approximate Clustering. In *Proceedings of 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 434-444, 1988.

[34] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate L1-difference algorithm for massive data streams. *Proc. FOCS*, 2000.

[35] P. Flajolet and G. N. Martin. Probabilistic Counting In *Proceedings of 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 76-82, 1983.

[36] J. Fong and M. Strauss. An Approximate Lp-Difference Algorithm for Massive Data Streams. *Proc. 17th STACS* pp. 193–204, 2000.

[37] A. Frieze, R. Kannan, and S. Vempala. Fast Monte Carlo algorithms for finding low rank approximation. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages ,1998.

[38] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W.H. Freeman, 1979.

[39] P. Gibbons, and S. Tirthapura. “Distributed Streams Algorithms for Sliding Windows” In *Proc. of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2002.

[40] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries *Proc. 27th VLDB* pp. 79–88, 2001.

[41] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. *Proc. STOC*, 2002.

[42] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, vol 38, pages 293–306, 1985.

[43] S. Guha. Approximation algorithms for facility location problems. Ph.D. Dissertation, Stanford University, 2000.

[44] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 649–657, 1998.

[45] S. Guha and N. Koudas. Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation. *Proc. 18th ICDE*, 2002.

[46] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. *Proc. STOC*, 2001.

[47] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams: Theory and Practice. In *IEEE Transactions on Knowledge and Data Engineering*, May/June 2003, 15(3).

[48] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. *Proc. FOCS*, 2000.

[49] S. Guha, R. Rastogi, and K. Shim. "CURE: An efficient clustering algorithm for large databases." In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD)* , 1998, pages 73–84.

[50] J. Han and M. Kamber, eds. Data Mining: Concepts and Techniques. Morgan Kaufman, 2000.

[51] J. A. Hartigan. *Clustering Algorithms.* John Wiley and Sons, 1975.

[52] M.R. Henzinger, P. Raghavan, and S. Rajagopalan "Computing on Data Streams.", Technical Report 1998-011, Compaq Systems Research Center, Palo Alto, CA, May, 1998

[53] M.R. Henzinger, P. Raghavan, and S. Rajagopalan "Computing on Data Streams.", In DIMACS series in Discrete Mathematics and Theoretical Computer Science, v. 50, 1999.

[54] A. Hinneburg and D. Keim. An Efficient Approach to Clustering Large Multimedia Databases with Noise. In *Proc. KDD*, 1998.

[55] A. Hinneburg and D. Keim. Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. In *Proc. VLDB*, 1999.

[56] D. S. Hochbaum, ed. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, 1996.

[57] D. S. Hochbaum and D. B. Shmoys. A best possible approximation algorithm for the k–Center problem. *Mathematics of Operations Research*, v. 10, 1985.

[58] G. Humphreys. A Stream Processing Approach to Interactive Graphics on Clusters of Workstations Ph.D. Dissertation, Stanford University, 2002.

[59] P. Indyk. Sublinear time algorithms for metric space problems. *Proc. STOC*, 1999.

[60] P. Indyk. A sublinear time approximation scheme for clustering in metric spaces. *Proc. FOCS*, 1999.

[61] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. *Proc. FOCS*, 2000.

[62] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys* 31, 1999.

[63] K. Jain and V. Vazirani. "Primal-Dual Approximation Algorithms for Metric Facility Location and k–median Problems." In *Proc. 40th IEEE Symp. on Foundations of Computer Science (FOCS)*, 1999, pages 1–10.

[64] K. Jain and V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual scheme and Lagrangian relaxation. *Journal of the ACM*, v. 48, 2001.

[65] N. Joze-Hkajavi and K. Salem. Two-phase clustering of large datasets. Tech. rpt. CS-98-27, Univ. of Waterloo, 1998.

[66] R. Kannan, S. Vempala, and A. Vetta. On Clusterings – Good, Bad, and Spectral. *Proc. FOCS*, 2000.

[67] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems, part ii: p-medians. *SIAM Journal of Appl. Math*, v. 37, 1979.

[68] L. Kaufman and P. J. Rousseeuw. Finding Groups in Data. An Introduction to Cluster Analysis. New York: Wiley, 1990.

[69] L. E. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. In *Journal of Computer and System Sciences* Special issue, vol 57, pages 50-60, 1998.

[70] M. Korupolu, C. G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. *Proc. SODA*, 1998.

[71] J.-H. Lin and J. S. Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, v. 44, 1992.

[72] J.-H. Lin and J. S. Vitter. ϵ-approximations with minimum packing constraint violation. *Proc. STOC*, 1992.

[73] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate Medians and other Quantiles in One Pass and with Limited Memory. *Proc. SIGMOD Conference*, pp. 426–435, 1998.

[74] G. S. Manku, S. Rajagopalan, B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. *Proc. SIGMOD Conference*, 1999.

[75] J. Marroquin and F. Girosi. Some extensions of the k–Means algorithm for image segmentation and pattern recognition. *AI Memo 1390, MIT AI Lab.*, 1993.

[76] D. Marchette. A Statistical Method for Profiling Network Traffic. In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, 1999.

[77] J. Matousek", On approximate geometric k-clustering. In *Discrete Computational Geometry*, 2000.

[78] R. Mettu and C. G. Plaxton. The online median problem. *Proc. FOCS*, 2000.

[79] R. Mettu and C. G. Plaxton. Optimal time bounds for approximate clustering. *Proc. UAI*, 2002.

[80] A. Meyerson. Online facility location. *Proc. FOCS*, 2001.

[81] A. Meyerson, L. O'Callaghan and S. Plotkin. A Fast, Small-Space Clustering Algorithm, Independent of Data Size. Unpublished manuscript, 2001.

[82] N. Mishra, D. Oblinger and L. Pitt. Sublinear time approximate clustering. *Proc. SODA*, 2001.

[83] N. Mishra, D. Ron, and R. Swaminathan. Large Clusters of Web Pages. In *Workshop on Algorithms and Models for the Web Graph (WAW)*, 2002.

[84] A. Mohr and M. Gleicher. Reconstructing from Streams for Stylized Rendering. In *Proc. ACM Symposium on Interactive 3D Graphics*, 2002.

[85] J. I. Munro and M. S. Paterson. Selection and Sorting with Limited Storage. *Theoretical Computer Science*, vol 12, pages 315-323, 1980.

[86] K. Nauta and F. Lieble. Offline Network Intrusion Detection: Looking for Footprints", SAS White Paper, 2000.

[87] A. Ng, M. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an Algorithm. In *Proc. NIPS*, 2002.

[88] R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proc. VLDB*, 1996.

[89] C. Niederauer, M. Houston, M. Agrawala, and G. Humphreys. Non-Invasive Interactive Visualization of Dynamic Architectural Environments. In *Proc. ACM Symposium on Interactive 3d Graphics*, 2003.

[90] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-Data Algorithms For High-Quality Clustering. *Proc. ICDE*, 2002.

[91] C. R. Palmer and C. Faloutsos. "Density biased sampling: an improved method for data mining and clustering." In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD)*, 2000, pages 82–92.

[92] D. Pelleg and A. W. Moore. "Accelerating exact k–means algorithms with geometric reasoning." In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (KDD)*, 1999.

[93] L. Pitt and R. E. Reinke. Criteria for polynomial-time (conceptual) clustering. *Machine Learning*, 1988, 2(4).

[94] R. A. Redner and H. F. Walker. Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review*, 1984, 26(2).

[95] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection.* Wiley, New York, 1987.

[96] S. Z. Selim and M. A. Ismail. K–Means type algorithms: A generalized convergence theorem and characterization of local optimality. In *IEEE Trans. PAMI*, vol. 6, pp 81–86, 1984.

[97] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In *Proc. VLDB*, 1998.

[98] D. B. Shmoys, É. Tardos, and K. I. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.

[99] M. Steinbach, G. Karypis, and V. Kumar. A Comparison of document clustering techniques. Tech rept. #00-034, Univ. of Minn.

[100] M. Pal, É. Tardos, and T. Wexler. Facility Location with Nonuniform Hard Capacities. In *Proc. FOCS*, 2001.

[101] M. Thorup. Quick k-median, k-center, and facility location for sparse graphs. *Proc. ICALP*, 2001.

[102] J. Vitter. Random Sampling with a Reservoir. In *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[103] S. Vempala and G. Wang. A spectral algorithm for learning mixtures of distributions. *Proc. FOCS*, 2002.

[104] W. Wang, J. Yang, and R. Muntz. STING: A Statistical Information Grid Approach to Spatial Data Mining. In *Proc. VLDB*, 1997.

[105] N. E. Young. K-medians, facility location, and the Chernoff-Wald bound. In *Proc. SODA*, 2000.

[106] T. Zhang, R. Ramakrishnan, and M. Livny. "BIRCH: an efficient data clustering method for very large databases." In *Proc. of the ACM SIGMOD Intl. Conference on Management of Data (SIGMOD)*, 1996, pages 103–114.