

Streaming-Data Algorithms For High-Quality Clustering

Liadan O'Callaghan*
Stanford University
loc@cs.stanford.edu

Nina Mishra[†]
Hewlett Packard Laboratories
nmishra@hpl.hp.com

Adam Meyerson[‡]
Stanford University
awm@cs.stanford.edu

Sudipto Guha[§]
University of Pennsylvania
sudipto@central.cis.upenn.edu

Rajeev Motwani[¶]
Stanford University
rajeev@cs.stanford.edu

Abstract

Streaming data analysis has recently attracted attention in numerous applications including telephone records, web documents and clickstreams. For such analysis, single-pass algorithms that consume a small amount of memory are critical. We describe such a streaming algorithm that effectively clusters large data streams. We also provide empirical evidence of the algorithm's performance on synthetic and real data streams.

1 Introduction

For many recent applications, the concept of a *data stream* is more appropriate than a data set. By nature, a stored data set is an appropriate model when significant portions of the data are queried again and again, and updates are small and/or relatively infrequent. In contrast, a data stream is an appropriate model when a large volume of data is arriving continuously and it is either unnecessary or impractical to store the data in some form of memory. Some applications naturally generate data *streams* as opposed

to simple data sets. Astronomers, telecommunications companies, banks, stock-market analysts, and news organizations, for example, have vast amounts of data arriving continuously.

Data streams are also appropriate as a model of access to large data sets stored in secondary memory where performance requirements necessitate access via linear scans. For researchers mining medical or marketing data, for example, the volume of data stored on disk is so large that it is only possible to make a small number of passes over the data.

In the data stream model [13], the data points can only be accessed in the order in which they arrive. Random access to the data is not allowed; memory is assumed to be small relative to the number of points, and so only a limited amount of information can be stored. The challenge facing algorithm designers is to perform meaningful computation with these restrictions.

A common form of data analysis in these applications involves *clustering*, i.e., partitioning the data set into subsets (clusters) such that members of the same cluster are similar and members of distinct clusters are dissimilar. This paper is concerned with the problem of clustering data arriving in the form of stream. We provide a streaming algorithm with some guarantees on its performance. We also provide a new clustering algorithm that is used by our streaming method. We give empirical evidence that the clustering algorithm outperforms the commonly-used *k*-Means algorithm. We also experimentally compare our streaming algorithm's performance to Birch.

In what follows, we first describe the clustering problem in greater detail, discuss related work and then give an overview of the paper.

The Clustering Problem In our work, we focus on the following version of the clustering problem: given

*Department of Computer Science, Stanford University, Stanford, CA 94305. Research supported in part by NSF Graduate Fellowship and NSF Grant IIS-9811904.

[†]Hewlett Packard Laboratories, Palo Alto, CA 94304.

[‡]Department of Computer Science, Stanford University, Stanford, CA 94305. Supported by ARO DAAG-55-97-1-0221.

[§]AT&T Research. Part of this work was done while the author was a student at Stanford University and was supported by an IBM Research Fellowship and NSF Grant IIS-9811904.

[¶]Department of Computer Science, Stanford University, Stanford, CA 94305. Research supported by NSF Grant IIS-9811904. A version of this paper with more explanation and experimental results is available at <http://www-db.stanford.edu/loc/icde02.ps>.

an integer k and a collection N of n points in a metric space, find k medians (cluster centers) in the metric space so that each point in N is assigned to the cluster defined by the median point nearest to it. The quality of the clustering is measured by the sum of squared distances (SSQ) of data points to their assigned medians. The goal is to find a set of k medians which minimize the SSQ measure. The generalized optimization problem, in which any distance metric substitutes for the squared Euclidean distance, is known as the k -Median problem¹.

Since the general problem is known to be NP-hard, the goal is to devise algorithms that produce solutions with near-optimal solutions in near-linear running time. The k -Means algorithm provides a heuristic solution to the problem of minimizing SSQ. The algorithm has enjoyed considerable practical success [3], although the solution it produces is only guaranteed to be a local optimum [23]. On the other hand, in the algorithms literature, several approximation algorithms have been proposed for k -Median. A c -approximation algorithm is guaranteed to find a solution whose SSQ is within a factor c of the optimal SSQ, even for a worst-case input. Recently, Jain and Vazirani [16] and Charikar and Guha [5] have provided such approximation algorithms for constants $c \leq 6$. These algorithms typically run in time and space $\Omega(n^2)$ and require random access to the data. Both space requirements and the need for random access render these algorithms inapplicable to data streams. Most heuristics, including k -Means, are also infeasible for data streams because they require random access. As a result, several heuristics have been proposed for scaling clustering algorithms, for example [4, 11]. In the database literature, the BIRCH system [27] is commonly considered to provide a competitive heuristic for this problem. While these heuristics have been tested on real and synthetic datasets, there are no guarantees on their SSQ performance.

Related Work The k -Means algorithm [6] and BIRCH [27] are most relevant to our results. We discuss Birch in more detail in Section 4. Most other previous work on clustering either does not offer the scalability required for a fast streaming algorithm or does not directly optimize SSQ. We briefly review these results — a thorough treatment can be found in Han and Kinber’s book [12].

Partitioning methods subdivide a dataset into k groups. One such example is the k -medoids algorithm [17] which selects k initial centers, repeatedly

¹Although squared Euclidean distance is not a metric, it obeys a relaxed triangle inequality and therefore behaves much like a metric.

chooses a data point randomly, and replaces it with an existing center if there is an improvement in SSQ. k -medoids is related to the CG algorithm given in Section 3.2.1, except that CG solves the facility location variant which is more desirable since in practice one does not know the exact number of clusters k (and facility location allows as input a range of number of centers). Choosing a new medoid among all the remaining points is time-consuming; to address this problem, CLARA [17] used sampling to reduce the number of feasible centers. This technique is similar to what we propose in Theorem 4. A distinguishing feature of our approach is a careful understanding of how sample size affects clustering quality. CLARANS [22] draws a fresh sample of feasible centers before each calculation of SSQ improvement. 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. [4] and its subsequent improvement by Farnstrom et al. [8]. These methods place higher significance on points later in the stream. In contrast, we assume that our data stream is not sorted in any way. Further, these approaches are not known to outperform the popular BIRCH algorithm.

Hierarchical methods decompose a dataset into a tree-like structure. Two common ones are HAC and CURE [11]. Since these methods are designed to discover clusters of arbitrary shape, they do not necessarily optimize SSQ.

Other types of clustering include density-based methods, e.g., DBSCAN [7], OPTICS [2] and DENCLUE [14] and grid-based methods, e.g., STING [26], CLIQUE [1], Wave-Cluster [24], and OPTIGRID [15]. These algorithms are also not designed to directly optimize SSQ.

Research on data stream computation includes work on sampling [25], finding quantiles of a stream of points [18], and calculating the $L1$ -difference of two streams [9].

Overview of Paper The rest of this paper is organized as follows. We begin in Section 2 by formally defining the stream model and the k -Median problem. Our solution for k -Median is obtained via a variant called facility location, which does not specify in advance the number of clusters desired, and instead evaluates an algorithm’s performance by a combination of SSQ and the number of centers used.

Our discussion about streaming can be found in Section 3. Section 3.2 describes our LSEARCH algorithm.

We performed an extensive series of experiments

comparing LSEARCH against the k -Means algorithm, on numerous low- and high-dimensional data. The results presented in Section 4.1 uncover an interesting trade-off between the cluster quality and the running time. We found that SSQ for k -means was worse than that for LSEARCH, and that LSEARCH typically found near-optimum (if not the optimum) solution. However LSEARCH took longer to run.

In Section 4.2 we empirically compare our streaming algorithm to Birch and discover a similar cluster quality versus running time tradeoff.

2 Preliminaries

We begin by defining a stream more formally. A data stream is a finite set N of points $x_1, \dots, x_i, \dots, x_n$ that can only be read in increasing order of the indices i . For example, these points might be vectors in \mathbb{R}^d . A data stream algorithm is not allowed *random access* but can retain a small amount of information about the data it has seen so far. Its performance is measured by the number of linear scans it takes over the data stream, the amount of information it retains, and the usual measures: in the case of a clustering algorithm, for example, these could be SSQ and running time.

We will define the k -Median problem, the derived problem of *SSQ minimization* with which this paper is concerned; and finally the *facility location* problem which will become relevant during the discussion of the algorithms.

Suppose we are given a set N of n objects in a metric space M with distance function d . Then the k -Median problem is to choose k points $c_1, \dots, c_k \in N$ so as to minimize $\sum_{i=1}^k \sum_{x \in N_i} d(x, c_i)$, where $N_i = \{x \in N \mid \forall j : d(c_j, x) \geq d(c_i, x)\}$.

The *SSQ minimization problem* is identical except that $M = \mathbb{R}^b$ for some integer b , d is the Euclidean metric, the medians can be arbitrary points in M , and $d^2(x, c_i)$ replaces $d(x, c_i)$ in the objective function. That is, we are to choose a set $C = \{c_1, \dots, c_k\} \subseteq M$ so as to minimize

$$SSQ(N, C) = \sum_{i=1}^k \sum_{x \in N_i} d^2(x, c_i),$$

where N_i is defined as before. Because of the similarity of these two problems, for the remainder of the paper, we will sometimes refer to the SSQ minimization problem as k -Median.

We now turn to facility location, which is the same as k -Median except that instead of restricting the number of medians to be at most k we simply impose a

cost for each median, or *facility*. The additive cost associated with each facility is called the *facility cost*.² Assume that we are given a facility cost f and a set N of n objects in a metric space M with distance function d . Then the *facility location problem* is to choose a subset $C \subseteq N$ so as to minimize the facility clustering (FC) cost function:

$$FC(N, C) = f|C| + \sum_{i=1}^{|C|} \sum_{x \in N_i} d(x, c_i),$$

($N_i = \{x \in N \mid c_i \text{ is the closest member of } C \text{ to } x\}$).

These problems are known to be NP-hard, and several theoretical approximation algorithms are known [16, 5].

3 Clustering Streaming Data

Our k -Median algorithm for clustering streaming data uses a k -Median subroutine called LSEARCH, which will be explained shortly. We first describe our streaming method.

3.1 Streaming Algorithm

We assume that our data actually arrives in chunks X_1, \dots, X_n , where each X_i is a set of points that can be clustered in main memory. We can turn any stream of individual points into a chunked stream by simply waiting for enough points to arrive.

The k -Median streaming algorithm is as follows:

Algorithm STREAM For each chunk X_i in the stream:

1. Solve k -Median on X_i using LSEARCH
2. $X' \leftarrow ik$ centers obtained from chunks 1 through i iterations of the stream, where each center c obtained by clustering X_i is weighted by the number of points in X_i assigned to c .
3. Output the k centers obtained by clustering X' using LSEARCH

Algorithm STREAM is memory efficient since at the i th point of the stream it retains only $O(ik)$ points. For very long streams, retaining $O(ik)$ points may get prohibitively large. In this case, we can once again cluster the weighted ik centers to retain just k centers.

It has been shown that STREAM produces a solution whose cost is at most a constant times the cost we

²This problem is technically called *uniform-cost facility location*. 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

would get by applying LSEARCH directly to the entire stream:

Theorem 1 [10] *If at each iteration i , a c -approximation algorithm is run in Steps 1 and 3 of Algorithm STREAM, then the centers output are a $5c$ -approximation to the optimum SSQ for $X_1 \cup \dots \cup X_i$, assuming a distance metric on R^d .*³

This theorem shows that STREAM cannot output an arbitrarily bad solution, although in practice we would expect it to get much better solutions; our experimental results are consistent with this expectation.

3.2 LSEARCH Algorithm

STREAM needs a simple, fast, constant-factor-approximation k -Median subroutine. We believe ours is the first such algorithm that also guarantees flexibility in k . 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 FC cost from the facility location definition.

3.2.1 Previous Work on Facility Location

We begin by describing a simple algorithm⁴ CG for solving the facility location problem on a set N of n points in a metric space with metric (relaxed metric) $d(\cdot, \cdot)$, when facility cost is f .

We begin by defining the subroutine *gain*. Assume that we have a feasible solution to facility location on N given $d(\cdot, \cdot)$ and f . That is, we have some set $I \subseteq N$ of currently open facilities, and an assignment for each point in N to some (not necessarily the closest) open facility. For every $x \in N$ we define *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 . $gain(x)$ can be easily computed in $O(|N||I|)$ time.

³Without the Euclidean assumption, Algorithm STREAM is a $10c$ -approximation.

⁴It has been shown by Charikar and Guha [5] that this algorithm will achieve a $(1 + \sqrt{2})$ -approximation to facility location on N with facility cost f .

Algorithm CG(data set N , facility cost f)

1. Obtain an initial solution (I, a) ($I \subseteq N$ of facilities, a an assignment function) that gives a n -approximation to facility location on N with facility cost f .
2. Repeat $\Omega(\log n)$ times:
 - Randomly order N .
 - 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.

Charikar and Guha [5] describe a simple algorithm that obtains an n -approximate initial solution that could be used in step 1.

3.2.2 Our New Algorithm

The above algorithm does not directly solve k -Median but could be used as a subroutine to a k -Median algorithm, as follows. 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 2 *Assume we have a set N of points with a metric $d : N \times N \rightarrow \mathbb{R}^+$. Then given facility costs f and f' with $0 \leq f < f'$, a set C of k centers that is optimal for f , and a set C' of k' centers that is optimal for f' , the following are true:*

1. $k \geq k'$
2. $SSQ(N, C) \leq SSQ(N, C')$
3. $fk + SSQ(N, C) < f'k' + SSQ(N, C')$
4. $SSQ(N, C) = SSQ(N, C')$ iff $k = k'$.

Proof: Let $a = SSQ(N, C)$, $a' = SSQ(N, C')$. Note $a \geq 0$ and $a' \geq 0$. Then $FC(N, C) = fk + a$ and $FC(N, C') = f'k' + a'$, for facility cost f . For facility cost f' , $FC(N, C) = f'k + a$ and $FC(N, C') = f'k' + a'$. By the optimality of C for f we have (I): $fk + a \leq f'k' + a'$, and, by the optimality of C' for f' we have (II): $f'k + a \geq f'k' + a'$. Then we have $f(k - k') \leq f'(k - k')$. Since $f < f'$, it must be that $k \geq k'$. Since $k \geq k'$, we can see by (I) that $a \leq a'$.

Combining (I) and the fact that $f < f'$, we have that $fk + a \leq f'k' + a'$.

If $a = a'$, then if $f = 0$ then $C = N$; that is, $k = n$ and $a = 0$. But then since $a = a'$, we know that C' must also be N so that $k' = n$. If $a = a'$ and $f \neq 0$ then we can combine (I) and (II) to show that $k = k'$. Therefore, $a = a' \Rightarrow k = k'$. If $k = k'$ (note $1 \leq k \leq n$), then we can again combine (I) and (II) to show that $a = a'$. Therefore, $a = a'$ iff $k = k'$. \square

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 dataset is “naturally k -clusterable,” then our algorithm should find k centers.

Theorem 3 *Given a dataset N , let A_i denote the best assignment cost achievable for an instance of FL on N if we allow i medians. If N 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 an FL solution to yield k centers, the solution for $j < k$ centers (and for $l > k$ centers) must have worse total cost than the FL solution for k centers, i.e., $ff + A_j \geq fk + A_k$ 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. \square

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 CG as 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 [5]; 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 the following algorithm for our initial solution:

Algorithm InitialSolution(data set N , facility cost f)

1. Reorder data points randomly
2. Create a cluster center at the first point
3. For every point after the first,
 - Let d be the distance from the current data point to the nearest existing cluster center
 - With probability d/f create a new cluster center at the current data point; otherwise add the current point to the best current cluster

This algorithm runs in time proportional to n times the number of facilities it opens and obtains an expected 8-approximation to optimum [20].

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 c_1, \dots, c_k constitute an optimal solution to the k -Median problem for the dataset N , that C_i is the set of points in N assigned to c_i , and that r_i is the average distance from a point in C_i to c_i for $1 \leq i \leq k$. Assume also that, for $1 \leq i \leq k$, $|C_i|/|N| \geq p$. Let $0 < \delta < 1$ be a constant and let S be a set of $m = \frac{8}{p} \log \frac{2k}{\delta}$ points drawn independently and uniformly at random from N .

Theorem 4 *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 N).*

Proof: If $|S| = m = \frac{8}{p} \log \frac{2k}{\delta}$ then $\forall i$, $Pr\{|S \cap C_i| < mp/2\} < \frac{\delta}{2k}$, by Chernoff bounds. Then $Pr\{\exists i : |S \cap C_i| < mp/2\} < \frac{\delta}{2}$. Given that $|S \cap C_i| \geq mp/2$, the probability that no point from S is within distance $2r_i$ of c_i is at most $\frac{1}{2} \leq \frac{1}{2} \log \frac{2k}{\delta} = \frac{\delta}{2k}$ by Markov’s Inequality. So $Pr\{\exists c_i : \forall x \in S, d(x, c_i) > 2r_i\} \leq \frac{\delta}{2}$. If, for each cluster C_i , our sample contains a point x_i within $2r_i$ of c_i , the cost of the median set $\{x_1, \dots, x_k\}$ is no more than 3 times the cost of the optimal k -Median solution: by the triangle inequality, each assignment distance would at most triple. \square

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., $p = 1/|N|$, then clearly no point can be overlooked as a feasible center. We view such a small subset of points as outliers, not clusters. Hence we 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 but will finish our computation sooner.

Finding a Good Facility Cost Faster If our cost changes very little from one iteration to the next and we have far from k centers, then we stop trying to identify how many centers a facility cost of f will yield, since the answer is unlikely to be k . Further, if a particular value of f yields exactly k centers and our total cost is not decreasing by much then we also stop our binary search, since it is likely that f is close to the true optimum facility cost.

Therefore, we will give a Facility Location subroutine FL 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 N of size n , the metric or relaxed metric $d(\cdot, \cdot)$, the facility cost f , and a current solution (F, g) where $F \subseteq N$ is a set of facilities and $g : N \rightarrow I$ is an assignment function.

Algorithm FL($N, d(\cdot, \cdot), f, \epsilon, (F, g)$)

1. Begin with (F, g) as the current solution
2. Let C be the cost of the current solution on N . Consider the feasible centers in random order; for each feasible center y , if $\text{gain}(y) > 0$, perform all advantageous closures and reassignments (see *gain* description), to obtain a new solution (F', g') [g' should assign each point to its closest center 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

Now we will give our k -Median algorithm for a data set N with distance function d .

Algorithm LSEARCH($N, d(\cdot, \cdot), k, \epsilon, \epsilon', \epsilon''$)

1. $f_{\min} \leftarrow 0$
2. $f_{\max} \leftarrow$ trivial upper bound U
3. $f \leftarrow (f_{\max} + f_{\min})/2$
4. $(I, a) \leftarrow \text{InitialSolution}(N, f)$.
5. Randomly pick $\Theta(\frac{1}{p} \log k)$ points as feasible medians
6. While $\# \text{medians} \neq k$ and $f_{\min} < (1 - \epsilon'')f_{\max}$:
 - Let (F, g) be the current solution
 - Run $FL(N, d, \epsilon, (F, g))$ to obtain a new solution (F', g')
 - If $|F'|$ is “about” k , then $(F', g') \leftarrow FL(N, d, \epsilon', (F', g'))$
 - If $|F'| > k$ then $\{f_{\min} \leftarrow f \text{ and } f \leftarrow (f_{\max} + f_{\min})/2\}$; else if $|F'| < k$ then $\{f_{\max} \leftarrow f \text{ and } f \leftarrow (f_{\max} + f_{\min})/2\}$

7. To simulate a continuous space, move each cluster center to the center-of-mass for its cluster
8. Return our solution (F', g')

The “trivial upper bound” U to which f_{\max} is initially set (in step 2 of LSEARCH) is, for an arbitrarily chosen point $x_0 \in N$, $\sum_{x \in N} d(x, x_0)$. 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 N 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 dataset but is usually small, so this running time is a significant improvement over previous algorithms.

4 Experiments

4.1 Empirical Evaluation of LSEARCH

We present the results of experiments comparing the performance of k -Means and LSEARCH. We conducted all experiments on a Sun Ultra 2 with two, 200MHz processors, 256 MB of RAM, and 1.23 GB of swap space,⁵ running SunOS 5.6.

Low-Dimensional Datasets We generated twenty-eight small datasets and ran LSEARCH and k -Means on each. Each of these consists of between five and sixteen uniform-density, radius-one spheres of real vectors, with five percent random noise. The noise is uniform over the smallest b -dimensional rectangle that is aligned with the axes and contains all the spheres. Here, b denotes the dimension of the dataset, which is at most four. The datasets have between one thousand and seven thousand points each. We generated three types of datasets: grid datasets, in which the spheres are centered at regular or nearly regular intervals; shifted-center datasets, in which the spheres are centered at positions slightly shifted from those of the grid datasets, and random-center datasets, where the centers are random in b -dimensional box. To put the results in perspective relative to earlier work, we also ran both algorithms on a dataset distributed by the authors of BIRCH [27], 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” datasets.

For each dataset, we calculated the SSQ of the k sphere centers on the dataset, and we used this value

⁵Our processes never used more than one processor or went into swap.

as an upper bound on the optimal SSQ of any set of k medians on this dataset. Since each dataset has relatively little noise (none in the case of the Gaussian dataset), 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 dataset, we recorded the best-known SSQ for that dataset; 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.

Because the clusters in the grid datasets (including the Gaussian set) are well-spaced and evenly distributed, k -Means would be expected to perform well on them; the Gaussian set should give particular advantage to k -Means since in a Gaussian the convergence rate is fastest. Indeed, on the grid of Gaussians, 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.

The shifted-center and random-center clusters are less evenly and symmetrically distributed than the grid clusters; k -Means quality was correspondingly worse than on the grid sets, often hitting two different clusters with a single median and losing other medians in the noise. By contrast LSEARCH performed consistently well (i.e., with low variance in SSQ) on all the synthetic datasets. See Figure 1 for a comparison of the best (lowest SSQ) and worst (highest SSQ) clusters on random-center dataset J. The best k -Means medians are drawn as octagons and the worst as tall, narrow diamonds; for LSEARCH the best medians are drawn as squares and the worst as ‘+’ signs. Both LSEARCH solutions are better than the k -Means solutions, but the LSEARCH solutions are also nearly indistinguishable from each other. Figure 2(a) shows the k -Means and LSEARCH SSQ values for the random-center datasets, normalized by the best-known SSQ for each set. The error bars represent the standard deviations for each algorithm on each set (also normalized by best-known SSQ).

Small, High-Dimensional Datasets We also generated ten high-dimensional datasets and ran k -Means and LSEARCH, ten times each, on each of these.

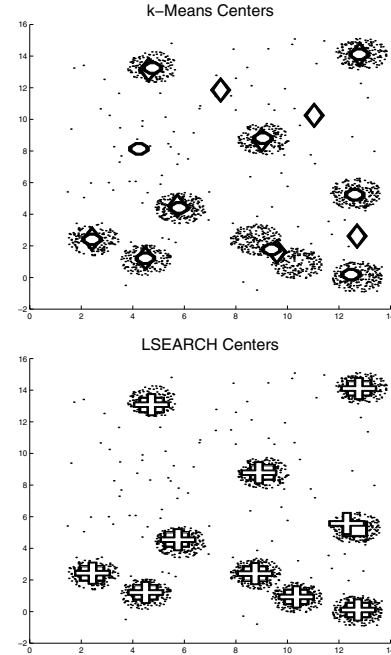


Figure 1. Best and Worst Solutions for Dataset J

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 datasets 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 dataset and averaged the SSQs of the solutions found. Figure 2(b) shows the average SSQ calculated by each algorithm on each dataset, normalized by division by the best-known SSQ. The error bars show normalized standard deviation. For all datasets, 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 dataset experiments. This increased unpredictability may indicate that k -Means is more sensitive than LSEARCH to dimensionality. In terms of running time, LSEARCH is consistently slower, although its running time has very low variance. LSEARCH appears to run approximately 3 times as long as k -Means on these sets (both algorithms used CPU time linear in the dataset dimension). As before, if we count only the amount of time it takes each algorithm to find a good answer, LSEARCH is competitive in running time and excels in solution quality. These results characterize

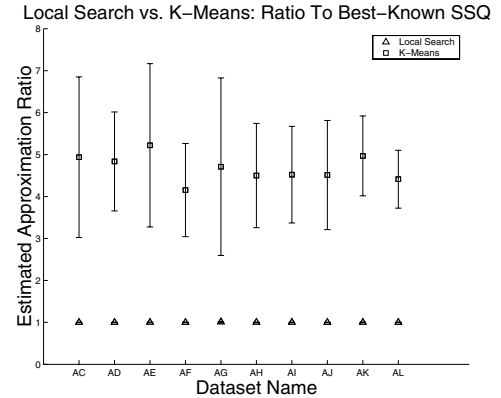
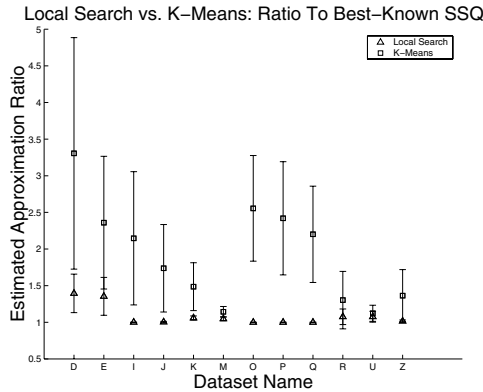


Figure 2. k -Means vs. LSEARCH: (a) Random-Center datasets, (b) High-Dimensional datasets

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.

4.2 Clustering Streams

STREAM k -means Theorem 1 only guarantees that the performance of STREAM is boundable if a constant-factor approximation algorithm is run in steps 1 and 3 of STREAM. Despite the fact that k -means has no such guarantees, due to its popularity we did experiment with running k -means as the clustering algorithm in Steps 1 and 3. Our experiments compare the performance of STREAM LSEARCH and STREAM k -Means with BIRCH.

Birch compresses a large dataset 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 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 [27].

STREAM and BIRCH have a common method of attack: 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 datasets described in Section 4.1, 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” to be 4MB and, to simulate incremental operation on an incoming stream, ran experiments on each of the four “prefixes” of the dataset; the i th 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 chunk⁶. SSQ and running time were both approximately linear in data size for the streaming algorithms. Figure 3 compares 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

⁶We also ran HAC on the CF-Trees, as the original paper advocated, but found that the SSQ was worse than that of k -Means.

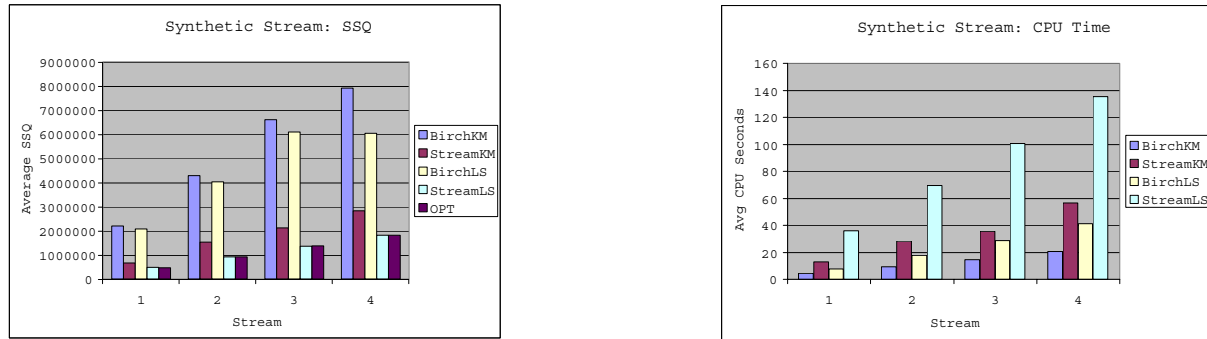


Figure 3. BIRCH vs. STREAM (Synthetic Stream)

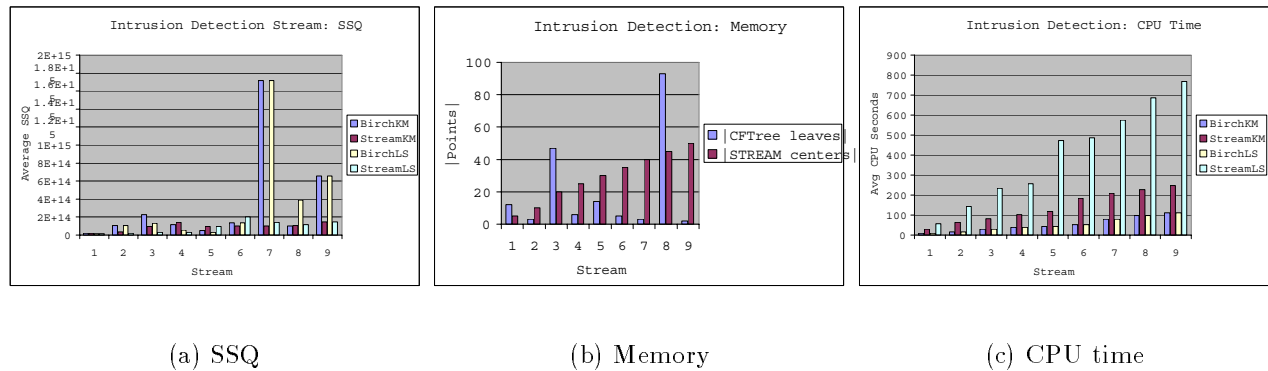


Figure 4. BIRCH vs. STREAM (Network Intrusion)

performance gap.

- STREAM ran 2 to 3 times slower than the corresponding BIRCH implementations.
- STREAM LSEARCH gave nearly optimal quality, just as demonstrated in the previous section. The bottom-up approach in STREAM introduces little extra error, and we have a near-optimal answer.

Network Intrusions Clustering, especially SSQ minimization, is often used for computer network intrusion detection [21, 19]. Since network protection demands the real-time identification of attacks, intrusion detection is a particularly fitting application of streaming.

We used the KDD-CUP'99 intrusion detection dataset which consists of raw TCP-dump data for a local area network, simulating a true Air Force environment with occasional attacks. Features collected

⁷<http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

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 dataset 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 4(a) compares the SSQ of all four methods. BIRCH performed badly on the 7th and 9th chunks of the stream because it generated too small a CF-Tree; the CF-Tree sizes are shown in 4(b). 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, respec-

tively. 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 datasets 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; 4(c) shows CPU time.

Overall, our 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 reasonable quick-and-dirty job. In applications like intrusion detection or target marketing where mistakes can be costly, quality matters more, and our STREAM algorithm is more appropriate.

References

- [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*, pages 94–105, 1998.
- [2] M. Ankerst, M. Breunig, H. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proc. SIGMOD*, 1999.
- [3] P.S. Bradley and U.M. Fayyad. Refining initial points for K-Means clustering. In *Proc. 15th Intl. Conf. on Machine Learning*, pages 91–99, 1998.
- [4] P.S. Bradley, U.M. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. KDD*, pages 9–15, 1998.
- [5] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proc. FOCS*, 1999.
- [6] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1972.
- [7] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases, 1996.
- [8] F. Farnstrom, J. Lewis, and C. Elkan. True scalability for clustering algorithms. In *SIGKDD Explorations*, 2000.
- [9] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate l1-difference algorithm for massive data streams, 1999.
- [10] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc. FOCS*, pages 359–366, 2000.
- [11] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proc. SIGMOD*, pages 73–84, 1998.
- [12] J. Han and M. Kinber, editors. *Data Mining: Concepts and Techniques*. Morgan Kaufman, 2000.
- [13] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams, 1998.
- [14] A. Hinneburg and D. Keim. An efficient approach to clustering large multimedia databases with noise. In *KDD*, 1998.
- [15] A. Hinneburg and D. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *Proc. VLDB*, 1999.
- [16] N. Jain and V.V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *Proc. FOCS*, 1999.
- [17] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data. An Introduction to Cluster Analysis*. Wiley, New York, 1990.
- [18] G. Manku, S. Rajagopalan, and B. Lindsley. Approximate medians and other quantiles in one pass and with limited memory, 1998.
- [19] D. Marchette. A statistical method for profiling network traffic. In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, 1999.
- [20] A. Meyerson. Online facility location. *In preparation*, 2001.
- [21] K. Nauta and F. Lieble. Offline network intrusion detection: Looking for footprints. In *SAS White Paper*, 2000.
- [22] R.T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proc. VLDB*, pages 144–155, 1994.
- [23] S.Z. Selim and M.A. Ismail. K-means type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Trans. PAMI*, 6:81–86, 1984.
- [24] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proc. VLDB*, pages 428–439, 1998.
- [25] J. Vitter. Random sampling with a reservoir, 1985.
- [26] W. Wang, J. Yang, and R. Muntz. Sting: A statistical information grid approach to spatial data mining, 1997.
- [27] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. SIGMOD*, pages 103–114, 1996.