Streaming-Data Algorithms For High-Quality Clustering

Liadan O'Callaghan *   Nina Mishra   Adam Meyerson   Sudipto Guha

Rajeev Motwani

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 [17], 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 challenging problem of clustering data arriving in the form of stream. We provide a streaming algorithm with theoretical 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 demonstrate our streaming algorithm's superior 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 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 from 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 problem[1].

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 k–Median problem. The algorithm has enjoyed considerable practical success [3], although the

---

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

solution it produces is only guaranteed to be a local optimum [28]. 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 [20] 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, 14]. In the database literature, the BIRCH system [32] 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 and BIRCH [32] are most relevant to our results. We discuss these 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 [15].

Partitioning methods subdivide a dataset into k groups. One such example is the k-medoids algorithm [21] which selects k initial centers, repeatedly 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 [21] 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 [26] 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 Farnstorm et al. [9]. 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 [14]. 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 [8], OPTICS [2] and DENCLUE [18] and grid-based methods, e.g., STING [31], CLIQUE [1], Wave-Cluster [29], and OPTIGRID [19]. These algorithms are not designed to directly optimize SSQ.

Research on data stream computation includes work on sampling [30], finding quantiles of a stream of points [22], and calculating the $L1$-difference of two streams [11].

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. The streaming algorithm given in this section is shown to enjoy theoretical quality guarantees. 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. Since both algorithms are randomized, we ran each one several times on each dataset; 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 datasets found a near-optimal answer before k–Means found an equally good solution. On many datasets k–Means never found a good solution.

2 Preliminaries

We begin by defining a stream more formally. A data stream is a finite set N of points $x_1, \ldots, x_i, \ldots, x_n$ that can only be read in increasing order of the indices i. For example, these points might be vectors in $\Re^d$. 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 the problem of choosing k points $c_1, \ldots, c_k \in N$ so as to minimize

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

where $N_i = \{x \in N | \forall j : d(c_j, x) \geq d(c_i, x)\}$.

The SSQ minimization problem is identical except that $M = \Re^b$ 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 minimize the "sum of squares" (SSQ) rather than the sum of distances. 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 once again that we are given a facility cost z 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:

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

---

[2]This 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

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

These problems are known to be NP-hard, and several theoretical approximation algorithms are known [20, 5].

3 Clustering Streaming Data

Our algorithm for clustering streaming data uses a subroutine called LSEARCH, which will be explained shortly.

3.1 Streaming Algorithm

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

The streaming algorithm, STREAM, is as follows: We cluster the ith chunk X_i using LSEARCH, and assign each resulting median a weight equal to the sum of the weights of its members from X_i. We then purge memory, retaining only the k weighted cluster centers, and apply LSEARCH to the weighted centers we have retained from $X_1, \ldots, X_i$, to obtain a set of (weighted) centers for the entire stream $X_1 \cup \cdots \cup X_i$.

Algorithm STREAM is memory efficient since at the ith 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 would get by applying LSEARCH directly to the entire stream (supposing it fit in main memory):

Theorem 1 *[13] If at each iteration i, a c-approximation algorithm is run in Steps 2 and 4 of Algorithm STREAM, then the centers output are a $5c$-approximation to the optimum SSQ for $X_1 \cup \cdots \cup X_i$, 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

---

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

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 algorithm[4] 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 z. First, we will make one definition.

Assume that we have a feasible solution to facility location on N given $d(\cdot, \cdot)$ and z. 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 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 possible 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. The *gain* of x can be easily computed in $O(|NI|)$ time.

Algorithm CG(data set N, facility cost z)

1. Obtain an initial solution (I, f) ($I \subseteq N$ of facilities, f an assignment function) that gives a n-approximation to facility location on N with facility cost z.

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.

---

[4]It 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 z.

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 z (between 0 and an easy-to-calculate upper bound); we then perform a binary search within this range to find a value of z that gives us the desired number k of facilities; for each value of z 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?

As to the first, consider a dataset N ($|N| = n$), with distance $d(\cdot, \cdot)$, on which we wish to solve facility location. If the facility cost increases, the number of centers in the optimal solution does not increase, the SSQ does not decrease, and the total (FC) solution cost increases. As opening a new facility becomes more expensive, we are forced to close centers and give solutions with higher assignment distances and FC cost. These cost relationships justify a binary search on z (proof omitted).

Theorem 2 *Assume we have a set N of points with a metric $d : N \times N \to \Re^+$. 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 $k = k'$.*

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. In particular, if having k rather than $k-1$ medians allows the optimal assignment cost to decrease by a factor of k, then an optimal solution with exactly k medians must exist (proof omitted).

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.*

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 $\Theta(n^2 \log n)$ 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 above, 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 z)

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/z 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 [24].

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 $c_1, \ldots, c_k$ 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 $1 \le i \le k$. Assume also that, for $1 \le i \le k$, $|C_i|/|N| \ge p$. 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 $|S \cap C_i| \ge mp/2$, the the probability that no point from S is within distance $2r_i$ of the optimum center c_i is at most $\frac{1}{2}^{mp/2} \le \frac{1}{2}^{\log \frac{2k}{\delta}} = \frac{\delta}{2k}$ by Markov's Inequality. So $Pr\{\exists c_j : \forall x \in S \, d(x, c_i) > 2r_i\} \le \frac{\delta}{2}$. 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 $\{x_1, \ldots, x_k\}$ 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). $\qquad\square$

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 z will yield, since the answer is unlikely to be k. Further, if a particular value of z 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 z is close to the true optimum facility cost.

Therefore, we will give a Facility Location subroutine that our k–Median algorithm will call; it will take a parameter $\epsilon \in \Re$ 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 z, and an initial solution (I, a) where $I \subseteq N$ is a set of facilities and $a : N \to I$ is an assignment function.

Algorithm FL(N, $d(\cdot, \cdot)$, z, ϵ, (I, a))

1. Begin with (I, a) as the current solution

2. Let C be the cost of the current solution on N. Consider the feasible centers in random order, and for each feasible center y, if $gain(y) > 0$, perform all advantageous closures and reassignments (as per $gain$ description), to obtain a new solution (I', a') [a' should assign each point to its closest center in I']

3. Let C' be the cost of the new solution; if $C' \leq (1 - \epsilon)C$, return to step 2

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. $z_{min} \leftarrow 0$
2. $z_{max} \leftarrow \sum_{x \in N} d(x, x_0)$ (for x_0 an arbitrary point in N)
3. $z \leftarrow (z_{max} + z_{min})/2$
4. $(I, a) \leftarrow$ InitialSolution(N, z).
5. Randomly pick $\Theta(\frac{1}{p} \log k)$ points as feasible medians
6. While #medians $\neq k$ and $z_{min} < (1 - \epsilon'')z_{max}$:

   - 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 $\{z_{min} \leftarrow z$ and $z \leftarrow (z_{max} + z_{min})/2\}$; else if $|F'| > k$ then $\{z_{max} \leftarrow z$ and $z \leftarrow (z_{max} + z_{min})/2\}$

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 initial value of z_{max} is chosen as a trivial upper bound on the value of z we will be trying to find.[5] The running time of LSEARCH is $O(nm + nk \log k)$ where m is the number of facilities opened by InitialSolution. m 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,[6] 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. To put the results of both algorithms in perspective relative to earlier work, we also ran both algorithms on a dataset distributed by the authors of BIRCH [32], which consists of one hundred, two-dimensional Gaussians in a ten-by-ten grid, with a 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.

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

---

[5]If the facility cost f is the sum of assignment costs when we open only some particular facility, then the solution that opens only this facility will have cost $2f$. There may be an equally cheap solution that opens exactly two facilities and has zero assignment cost (if every point is located exactly at a facility), and there may be cheaper solutions that open only one facility (if there is a different facility for which the sum of assignment costs would be less than f), but there cannot exist an optimal solution with more than two facilities. This value f is therefore an upper bound for z since k–Median is trivial when $k = 1$.

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

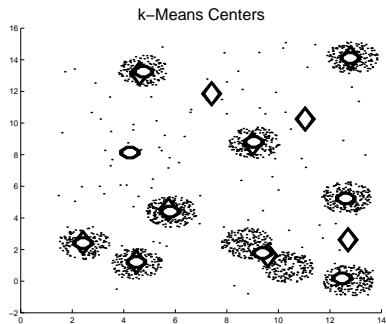deviation 5711, whereas for LSEARCH the SSQ was 176136 with standard deviation 337.

Figure 1. k–Means Centers for Dataset J

The shifted-center and random-center datasets exhibit much less symmetry than the grid datasets, and so k–Means clustering quality was correspondingly worse than on the grid sets. Because it uses less global information, k–Means can get distracted by the random noise and can inappropriately group two clusters together, especially when clusters are not evenly spaced or distributed; 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) k–Means clusters on one of the random-center sets. Although both solutions are poor, the high-SSQ solution is considerably worse than the low-SSQ solution. The best and worst LSEARCH clusters for this dataset were nearly indistinguishable and had almost the same SSQ. Figure 2 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 ran k–Means and LSEARCH, ten times each, on each of ten high-dimensional datasets. 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 3 shows the average SSQ calculated by each algorithm on each dataset, normalized by division by the best-known SSQ. The error bars represent the normalized standard deviations. For all datasets, the answer
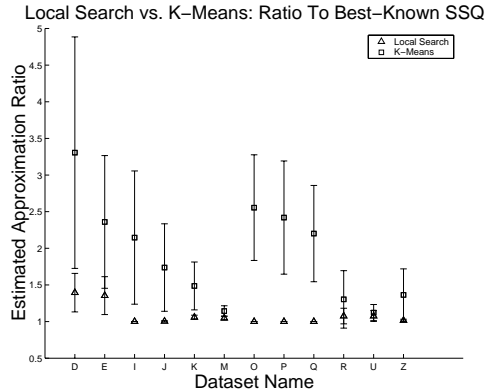

Figure 2. k–Means vs. LSEARCH: Random-Center Datasets

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.
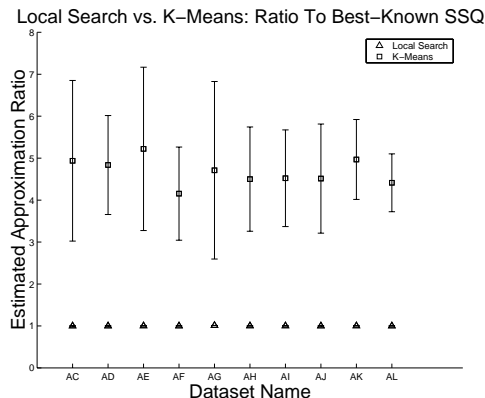

Figure 3. k–Means vs. LSEARCH: High-Dimensional Datasets

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. 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 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 2 and 4 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 2 and 4. 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 CFtree (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 CFtree tree repeatedly inserts points into the leaves provided that the radius of the set of points associated with a leaf does not exceed a certain threshold. If the threshold is exceeded then a new leaf is created and the tree is appropriately balanced. If the tree does not fit in main memory then a new threshold is used to create a smaller tree. Numerous heuristics are employed to make such decisions, refer to [32] for details.

STREAM and BIRCH have a common method of attack: repeated preclustering of the data. However 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. Hence the results compare SSQ and running time.

Synthetic Data Stream We generated a stream approximately 16MB in size, consisting of 50,000 points in 40-dimensional Euclidean space. The stream was generated similarly to the datasets described in 4.1, except that the diameter of clusters varied by a factor of 9, and the number of points by a factor of 6.33. We divided the point set into four consecutive chunks, each of size 4MB, and calculated an upper bound on the SSQ for each, as in previous experiments, by finding the SSQ for the centers used to generate the set. We

ran experiments on each of the four "prefixes" induced by this segmentation into chunks: the prefix consisting of the first chunk alone, that consisting of the second chunk appended after the first, the prefix consisting of the concatenation of the first three chunks, and the prefix consisting of the entire stream.

As in previous experiments, we ran LSEARCH and k–Means ten times each on any dataset (or CF-tree) on which we tested them. Since BIRCH and HAC are not randomized, this repetition was not necessary when we generated CF-trees or ran HAC. Thus we ended up with four choices depending on the clustering algorithm chosen and the method of preclustering.
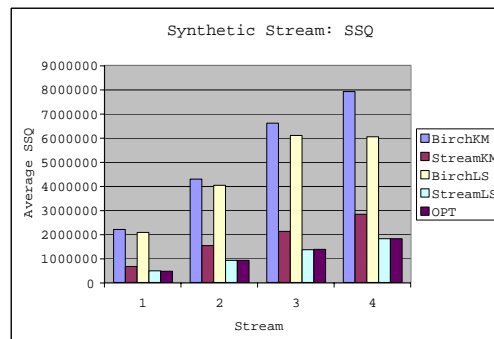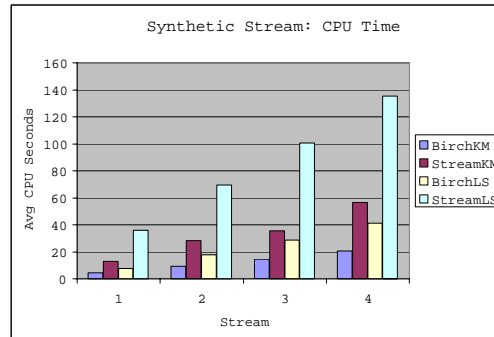

Figure 4. BIRCH vs. STREAM: SSQ

Figure 5. BIRCH vs. STREAM: CPU Time

The performance of each ialgorithm was linear both in error and running time as expected. In summary

- STREAMgave an SSQ of factor 2 to 3 times lower compared to the corresponding implementation

with BIRCH that used the same clustering algorithm. The reason for this performance gap was that the BIRCH CF-trees usually had one "megacenter," a point of very high weight, along with a few points of very small weight; in other words, BIRCH missummarized the data stream by grouping a large number of points together at their mean.

- STREAMran a factor 2 to 3 times slower than the corresponding implementation using BIRCH. BIRCH uses a top down partitioning as preclustering process, while STREAM uses a bottom up clutering as a preclustering step. The results demonstrate the effect of more accurate decisions in STREAM regarding storing the summary staistics.

- STREAMLSEARCH gave nearly optimal quality. This is natural since in the previous section we demonstrated that LSEARCH gave near optimal clustering. The bottom up approach in STREAM introduces little extra error in the process, and we have a near optimal answer.

Network Intrusions Clustering, and in particular algorithms that minimize SSQ, are popular techniques for detecting intrusions [25, 23]. Since detecting intrusions the moment they happen is tantamount to protecting a network from attack, intrusions are a particularly fitting application of streaming. Offline algorithms simply do not offer the immediacy required for successful network protection.

In our experiments we used the KDD-CUP'99[7] intrusion detection dataset which consists of two weeks of raw TCP dump data for a local area network simulating a true Air Force environment with occasional attacks. Features collected for each connection include the duration of the connection, the number of bytes transmitted from source to destination (and vice versa), the number of failed login attempts, etc. 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 sized chunks. The data was clustered into five clusters since there were four types of possible attacks (plus no attack). The four attacks included denial of service, unauthorized access from a remote machine (e.g., guessing password), unauthorized access to root, and probing (e.g., port scanning).

The leftmost chart in Figure 6 compares the SSQ of BIRCH-LS with that of STREAMLS; the middle chart makes the same comparison for BIRCH k–Means and

STREAM k–Means. BIRCH's performance on the 7th an 9th chunks can be explained by the number of leaves in BIRCH's CFTree, which appears in the third chart.

Even though STREAM and BIRCH are given the same amount of memory, BIRCH does not fully take advantage of it. BIRCH's CFTree has 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 radius of points allowed in a leaf when the CFTree 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.

In terms of cumulative average running time, shown in Figure 6, BIRCH is faster. STREAM LS varies in its running time due to the creation of a weighted dataset (Step 1).

Overall, our results point to a cluster quality vs. running time tradeoff. In applications where speed is of the essence, e.g., clustering web search results, BIRCH appears to do a reasonable quick and dirty job. In applications like intrusion detection or target marketing where mistakes can be costly our STREAM algorithm exhibits superior SSQ performance.

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] C. Cortes, K. Fisher, D. Pregibon, and A. Rogers. Hancock: A language for extracting signatures from data streams. In *Proc. of the 2000 ACM SIGKDD Intl. Conf. on Knowledge and Data Mining*, pages 9–17, August 2000.

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

---

[7]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
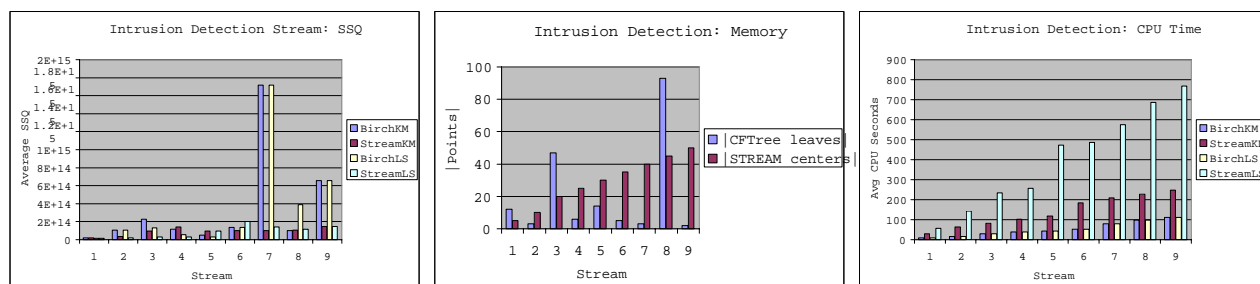
SSQ   Memory   CPU time

Figure 6. Network Intrusion Data : BIRCH vs. STREAM

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

[9] F. Farnstrom, J. Lewis, and C. Elkan. True scalability for clustering algorithms. In *SIGKDD Explorations*, 2000.

[10] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. MII Press, Mento Park, 1996.

[11] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan. An approximate l1-difference algorithm for massive data streams, 1999.

[12] A. Gersho and R.M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Norwell, MA, 1992.

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc. FOCS*, pages 359–366, 2000.

[14] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proc. SIGMOD*, pages 73–84, 1998.

[15] J. Han and M. Kamber, editors. *Data Mining: Concepts and Techniques*. Morgan Kaufman, 200.

[16] J. A. Hartigan. *Clustering Algorithms*. John Wiley and Sons, New York, 1975.

[17] M. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams, 1998.

[18] A. Hinneburg and D. Keim. An efficient approach to clustering large multimedia databases with noise. In *KDD*, 1998.

[19] A. Hinneburg and D. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *Proc. VLDB*, 1999.

[20] N. Jain and V.V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *Proc. FOCS*, 1999.

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

[22] G. Manku, S. Rajagopalan, and B. Lindsley. Approximate medians and other quantiles in one pass and with limited memory, 1998.

[23] D. Marchette. A statistical method for profiling network traffic. In *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, 1999.

[24] A. Meyerson. Online facility location. *In preparation*, 2001.

[25] K. Nauta and F. Lieble. Offline network intrusion detection: Looking for footprints. In *SAS White Paper*, 2000.

[26] R.T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proc. VLDB*, pages 144–155, 1994.

[27] L. Pitt and R.E. Reinke. Criteria for polynomial-time (conceptual) clustering. *Machine Learning*, 2:371, 1987.

[28] 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.

[29] 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.

[30] J. Vitter. Random sampling with a reservoir, 1985.

[31] W. Wang, J. Yang, and R. Muntz. Sting: A statistical information grid approach to spatial data mining, 1997.

[32] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. SIGMOD*, pages 103–114, 1996.