# Clustering Data Streams

Sudipto Guha * Nina Mishra † Rajeev Motwani ‡ Liadan O'Callaghan §

August 29, 2000

## Abstract

We study clustering under the data stream model of computation where given a sequence of points, the objective is to maintain a consistently good clustering of the sequence observed so far, using small amount of memory and time. The data stream model is relevant to new classes of applications such as web click stream analysis and multimedia data analysis. We give constant-factor approximation algorithms for the $k$–Median problem in the data stream model of computation in a single pass.

## 1 Introduction

A data stream is an ordered sequence of points that can be read only once or a small number of times. Formally, a data stream is a 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 the algorithm must make over the stream, when constrained in terms of available memory, in addition to the more conventional measures. In particular, customer click streams, telephone records, large sets of web pages, multimedia data, and sets of retail chain transactions can be modeled as data streams. These data sets are generally far too large to fit in main memory and are typically stored in secondary storage devices, making access, particularly random access, very expensive. Any algorithms to be run on such data sets must therefore run in a mode where they merely scan the data from start to finish, and the number of such linear scans is required to be kept small (preferably only a single scan). 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 necessitates the design of a novel kind of algorithm which must extract summaries of past data for storing into memory, so as to enable processing of the data to be seen in the future. We remark that this is not the same as the model of online algorithms.

Clustering has recently been widely studied across several disciplines, but only a few of the techniques developed scale to support clustering of very large data sets. In particular, most algorithms for the $k$-Median clustering problem, which is to find $k$ centers in a set of $n$ points so as to minimize the sum of radii, require large space and random access to the points. We give constant-factor approximation algorithms for the $k$-Median problem that naturally fit into this data stream setting. Our algorithms make a single pass over the data and use small space. We first give a randomized constant-factor approximation algorithm for $k$-Median, which makes one pass over the data using $n^\epsilon$ memory and requiring only $\tilde{O}(nk)$ time. We also prove that any deterministic $k$-Median algo-

*Department of Computer Science, Stanford University, CA 94305. Email: sudipto@cs.stanford.edu. Research supported by IBM Research Fellowship and NSF Grant IIS-9811904.

†Hewlett Packard Laboratories, Palo Alto, CA 94304, Email: nmishra@hpl.hp.com

‡Department of Computer Science, Stanford University, CA 94305. Email: rajeev@cs.stanford.edu. Research supported in part by NSF Grant IIS-9811904.

§Department of Computer Science, Stanford University, CA 94305. Email: loc@cs.stanford.edu. Research supported in part by an NSF Graduate Fellowship, ARO MURI Grant DAAH04-96-1-0007, and NSF Grant IIS-9811904.

rithm that achieves a constant-factor approximation cannot run in time less than $\Omega(nk)$. Finally, we give a deterministic $\tilde{O}(nk)$-time, polylog($n$)-approximation single-pass algorithm that uses $n^\epsilon$ space.

**Related Work on Data Streams** One of the first results in data streams was the result of Munro and Paterson [**?**] where they studied the space requirement of selection and sorting as a function of the number of passes over the data. The model was formalized by Henzinger, Raghavan, and Rajagopalan [**?**], who gave several algorithms and complexity results related to graph theoretic problems and their applications. We omit detailed discussion of recent results [**?**, **?**, **?**, **?**] on data streams, which while interesting are not relevant to the clustering problems considered here.

**Related Work on Clustering** In this paper we shall consider models in which clusters have a distinguished point, or "center." In the $k$–Median problem, in which the objective is to minimize the average radius or the sum of the distances from data points to their closest cluster centers. The 1–median problem was first posed by Weber [**?**]. In the $k$–Center problem the objective function is to minimize the maximum radius of a cluster. The above problems are all NP-hard, so we will be concerned with approximation algorithms. We will assume the domain space of points to be discrete, i.e., the cluster centers must be among the input points. The continuous case is related to the discrete problem by small factors (see Theorem **??**). Throughout the paper we also assume that the points are drawn from a metric space.

In the recent past several approximation algorithms have been proposed for the $k$–Median problem [**?**, **?**, **?**]. These algorithms require $O(n^2)$ space to compute the dual variables or primal constraints. We will be interested in algorithms which use more than $k$ medians but run in linear space [**?**, **?**, **?**].

Charikar, Chekuri, Feder, and Motwani [**?**] gave a constant-factor algorithm for the incremental $k$–Center problem, which is also a single-pass algorithm in $O(nk \log k)$ time and $O(k)$ space. There is a large difference, however, between the $k$–Center and the $k$–Median problem since a set of $k+1$ suitably separate points provides a lower bound for the $k$–Center problem. These points can be thought of as a proof of the goodness of the clustering. For the $k$–Median problem, allowing weighted points, no such succinct proof exist. The optimization problem takes on a more global character.

**Our Results** We begin by giving an algorithm that requires small space, and then later address the issue of clustering in one pass. In Section **??** 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 **??**. This algorithm runs in time $O(n^{1+\epsilon})$ time, uses $O(n^\epsilon)$ memory and makes a single pass over the data. Next, in Section **??**, using randomization, we show how to reduce the running time to $O(nk)$ without requiring more than a single pass. In Section **??** we show it is not possible to obtain any bounded approximation ratio in deterministic $o(nk)$ time. In Section **??**, we also show how to achieve a poly-$\log n$ approximation ratio in a single pass in deterministic $\tilde{O}(nk)$ time.

## 2 Clustering in Small Space

One of the first requisites of clustering a data stream is that the computation be carried out in small space. Our goal will be first to show that clustering can be carried out in small ($n^\epsilon$ for $n$ data points) space, without being concerned with the number of passes. Subsequently we will see how to implement the algorithm in one pass.

In order to cluster in a small space, we investigate algorithms that examine 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 pieces, clusters each of these pieces, and then again clusters the centers obtained (where each center is weighted by the number of points closest to it). We show that this piecemeal approach doesn't hurt too much in that: if we had a constant-factor approximation algorithm, running it in divide-and-conquer fashion would still yield a (slightly worse) constant approximation. 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 centers. For this algorithm, we prove that if we recluster a constant number of times, a constant factor is still obtained, although, as expected, the constant factor worsens with each successive reclustering. The advantage of Small(er)-Space will be clear: we will sacrifice somewhat the quality of the clustering approximation to obtain an algorithm that can cluster with much less memory.

## 2.1 A Simple Divide-and-Conquer Strategy with Separability Theorems

We start with the version of the algorithm that reclusters only once. Elements of the algorithm and its analysis will be used in a black-box manner in the algorithms in the rest of the paper.

**Algorithm Small-Space(S)**

1. Divide $S$ into $l$ disjoint pieces $\chi_1, \dots, \chi_l$.
2. For each $i$, find $O(k)$ centers in $\chi_i$.
3. Let $\chi'$ be the $O(lk)$ centers obtained in (2) where each center $c$ in $\chi_i$ is weighted by the number of points in $\chi_i$ closest to center $c$.
4. Cluster $\chi'$ to find $k$ centers.

Since we are interested in clustering in small space, $l$ will be set so that both $S$ and $\chi'$ fit in main memory, if possible. If $S$ is very large, no such $l$ may exist – we'll address this issue later.

Before analyzing algorithm Small-Space, we describe the relationship between the discrete and continuous clustering problem. The following is folklore and is included for completeness.

**Theorem 2.1** *Given an instance of the $k$-median problem with a solution of cost $C$, where the medians may not belong to the set of input points, there exists a solution of cost $2C$ where all the medians belong to the set of input points.*

**Proof:** Consider the solution of cost $C$, and let the points $j_1, \dots, j_q$ be assigned to median $i$. Since median $i$ may not be in the input, consider the point $j_l$ which is closest to $i$ as the median (instead of $i$). Thus the assignment distance of every point $j_r$ doubles, since $c_{j_r j_l}$ can be bound by $c_{j_l i} + c_{j_r i}$ (where $c_{xy}$ denotes the distance from $x$ to $y$). Thus over all the $n$ points in the original set, the assignment distance can at most double, summing to at most $2C$. $\Box$

The following separability theorem sets the stage for a divide-and-conquer algorithm. This theorem carries over to other clustering metrics such as the sum of squared distances and so on.

**Theorem 2.2** *Given a set of points, and an arbitrary partitioning of it in disjoint sets $\chi_1, \dots, \chi_\ell$, the sum of the optimum solution values for the $k$-median problem on these $\ell$ sets of points is at most twice the cost of the optimum $k$-median problem solution for the entire $n$ points.* [1]

**Proof:** Consider the medians used for the optimum $k$-median solution. If each partition uses these medians, the cost of the solution will be exactly the cost of the optimal solution. This follows since the objective function for $k$-median is the sum of distances to the nearest median for every point. However the set of medians chosen by the optimum solution need not be present in a partition. But in the case where the medians points can be arbitrary points in the space, the above theorem is proved.

In case we have to chose the medians from the given set of points, the $k$-medians used by the optimum solution will not be available to every partition. In this case use Theorem **??** to construct a solution which is at most 2 times the cost of the optimum solution. $\Box$

Next we show that the new instance, where all the points $i$ that have median $i'$ shift their weight to the point $i'$ (i.e., the weighted $O(lk)$ centers $S'$ in Step 2 of Algorithm Small-Space), has a good feasible clustering solution. Notice that the set of points in the new instance is much smaller and may not even contain the medians of the optimum solution.

[1] The factor 2 is avoided in the Euclidean case, if we allow that medians can be arbitrary points in space, rather than requiring that they be points from the original data set.

**Theorem 2.3** *If the cost of the solution we got by partitioning the point set is $C$, and if $C^*$ is the cost of the optimum solution to the $k$-median problem for the entire set, then there exists a solution of cost $2(C + C^*)$ to this new instance.* [2]

**Proof:** As in the proof of the previous theorem, we will consider the $k$ medians in the optimum solution.

Each point $i'$ in the new instance can be viewed as a collection of points. Consider a point $i$ whose weight $d_i$ was shifted to point $p'$ in the new instance. In that case, $i'$ was the median to which $i$ was assigned to and we can consider the point $i$ to be located at $i'$ in the new instance. Thus the objective function can be bound by summing over $i$, the assignment distance of $i'$ weighted by $d_i$.

Let the median to which $i$ was assigned to in the optimal solution be $\sigma(i)$. Thus $i'$ has a median at distance $c_{i'\sigma(i)}$. Thus the contribution of $i$ in the new instance is $d_i c_{i'\sigma(i)}$. This is bound by $d_i c_{ii'} + d_i c_{i\sigma(i)}$. The first term, summed over all points $i$, evaluates to $C$ and the second term evaluates to $C^*$.

Thus we showed an assignment to the medians of the optimal solution at cost $C + C^*$. Using Theorem **??**, the theorem follows. $\square$

We now show that if we run a bicriteria $(a, b)$ approximation algorithm (where at most $ak$ medians are output with cost at most $b$ times the optimum $k$–Median solution) in Step 2 of Algorithm Small-Space and we run a $c$-approximation algorithm in Step 4, then the resulting approximation of the $k$ centers output by Small-Space is bounded.

**Theorem 2.4** *The algorithm Small-Space has an approximation factor of $2c(1 + b) + b$.*

**Proof:** Let the optimal $k$-median solution be of cost $C^*$. Then the cost of the solution $C$ at the end of the first stage is at most $bC^*$. This is true due to Theorem **??**, since we are adding the cost of the solutions to each partition, each of which is a $b$-approximation for that partition. Now by Theorem **??**, there exists a solution to the $k$-median problem on the modified instance of cost $2(C + C^*)$. Since we have a $c$-approximation, we have a solution of cost $2c(1+b)C^*$ to the modified instance.

Consider the following thought experiment: if point $i$ was assigned to median $i'$ in the first solution, assume the point $i$ is located $i'$ in a new instance. Thus instead of summing over $i'$, we can account the cost of the solution by summing over $i$ the assignment distance of $i'$ weighted by $d_i$. If the assignment distance of $i'$ is $c_{i'\sigma(i')}$ in our solution, $\sum_i d_i c_{i'\sigma(i')}$ is $2c(1 + b)C^*$.

If we consider a solution to the original instance where $i$ is assigned to $\sigma(i')$, the contribution of $i$ is at most $d_i$ times $c_{i\sigma(i')}$. The distance $c_{i\sigma(i')}$ is bounded by $c_{ii'} + c_{i'\sigma(i')}$. Thus the total contribution can be bounded by $\sum_i d_i(c_{ii'} + c_{i'\sigma(i')})$. Now $\sum_i d_i c_{ii'}$ is $C$, and $\sum_i d_i c_{i'\sigma(i')}$ is $2c(1 + b)C^*$, which proves the theorem. $\square$

The black-box nature of the algorithm will now allow us to devise a divide-and -conquer strategy based algorithm.

## 2.2 Divide-and-Conquer Strategy

We now generalize Small-Space so that the algorithm recursively calls itself on a successively smaller set of weighted centers.

**Algorithm Smaller-Space(S,i)**

1. Divide $S$ into $l$ disjoint pieces $\chi_1, \ldots, \chi_l$.
2. For each $i$, find $O(k)$ centers in $\chi_i$.
3. Let $S'$ be the $O(lk)$ centers obtained in (2) where each median $c$ in $\chi_i$ is weighted by the number of points in $\chi_i$ closest to center $c$.
4. Call algorithm Smaller-Space$(S', i - 1)$.

We can claim the following,

**Theorem 2.5** *Algorithm Smaller-Space$(S, i)$ for constant $i$, gives a constant factor approximation for the $k$–Median problem.*

**Proof:** Assume that the approximation factor for the $j$th level is $A_j$. From Theorem **??** we know that the cost of the solution of the first level is $2b$ times

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

optimal. From Theorem **??** we get that the approximation factor $A_j$ would satisfy a simple recurrence,

$$A_j = 2A_{j-1}(2b+1) + 2b$$

The solution of the recurrence is $c \cdot (2(2b+1))^j$. This is $O(1)$ given $j$ is a constant. □

We need to store the intermediate medians, though, using up memory, and this seems to limit the number of subsets into which we partition the original data set. If we partition it into more than $\sqrt{n/k}$ subsets, then since each generates at least $k$ points we will overrun memory.

In the next section we will see a way to get around this problem. In fact we will be able to implement the hierarchical scheme more cleverly and obtain a clustering algorithm for an interesting model of computation. We have two themes to develop this idea. The first is to do away with the storage of the intermediate medians, and the second is to design a more interesting recursive algorithm. We take up the former and relegate the second to a later section.

# 3 The Data Stream Model

Under the Data Stream Model computation takes place within bounded space $M$, and the data can only be seen via linear scans from the first record onwards (i.e., a data point can be seen only once in a scan, and points must be viewed in order).

In this section we will modify the multi-level algorithm to operate on data streams. We will present a one-pass, $O(1)$-approximation in this model assuming that the bounded memory $M$ is not too small, more specifically $n^\epsilon$ if $n$ denotes the number of records in the stream.

This model and the line of analysis have similarities to incremental clustering and online models. However our approach will be a bit different. We will maintain a forest of assignments. We will complete this to $k$ trees, and all the nodes in a tree will be assigned to the median denoted by the root of the tree. First we will show how solve the problem of storing intermediate medians. Next we will inspect the space requirements and running time.

**Data Stream Algorithm** To achieve this, we will modify our multi-level algorithm slightly. The algorithm will be the following:

1. Input the first $m$ points; use a bicriterion algorithm to reduce these to $O(k)$ (say $2k$) points. As usual, the weight of each intermediate median is the number of points assigned to it in the bicriterion clustering. (Assume $m$ is a multiple of $2k$.) This requires $O(f(m))$ space, which for a primal dual algorithm can be $O(m^2)$. We will see a $O(mk)$-space algorithm later.
2. Repeat the above till we have seen $m^2/(2k)$ of the original data points. At this point we have $m$ intermediate medians.
3. Cluster these $m$ first-level medians into $2k$ second-level medians and proceed.
4. In general, maintain at most $m$ level-$i$ medians, and, on seeing $m$, generate $2k$ level-$i+1$ medians, with the weight of a new median as the sum of the weights of the intermediate medians assigned to it.
5. When we have seen all the original data points (or we want to have a clustering of the points we have seen so far) we cluster all the intermediate medians into $k$ final medians.

Note that this algorithm is identical to the multi-level algorithm described before.

The number of levels required by this algorithm is at most $O(\log(n/m)/\log(m/k))$. If we have $k \ll m$ and $m = O(n^\epsilon)$ for some constant $\epsilon < 1$, we have an $O(1)$-approximation. Using linear programming or primal dual algorithms we will have $m = \sqrt{M}$ where $M$ is the memory size (ignoring factors due to maintaining intermediate medians of different levels). We argued that the number of levels would be a constant when $m = n^\epsilon$ and hence when $M = n^{2\epsilon}$ for some $\epsilon < 1/2$.

**Linear Space Clustering** The approximation quality which we can prove (and intuitively the actual quality of clustering obtained on an instance) will depend heavily on the number of levels we have. From this perspective it is profitable to use a space-efficient algorithm.

We can use the local search algorithm in [?] to provide a bicriterion approximation in *space linear in* $m$, the number of points clustered at a time. The advantage of this algorithm is that it maintains only an assignment and therefore uses linear space. However the complication is that for this algorithm to achieve a bounded bicriterion approximation, we need to set a "cost" to each median used, so that we penalize if many more than $k$ medians are used. The algorithm solves a facility location problem after setting the cost of each median to be used. However this can be done by guessing this cost in powers of $(1+\gamma)$ for some $0 < \gamma < 1/6$ and choosing the best solution with at most $2k$ medians. In the last step, to get $k$ medians we would have to use a two step process of reducing the number of medians to $2k$ and then using [?, ?] to reduce to $k$. This would allow us to cluster with $m = M$ points at a time provided $k^2 \leq M$.

**The Running Time** The running time of this clustering is dominated by the contribution from the first level. The local search algorithm is quadratic and the total running time is $O(n^{1+\epsilon})$ where $M = n^{\epsilon}$. We argued before, however, that $\epsilon$ will not be very small and hence the approximation quality which we can prove will remain small.

We therefore claim the following theorem,

**Theorem 3.1** *We can solve the $k$–Median problem on a data stream with time $O(n^{1+\epsilon})$ and space $\theta(n^{\epsilon})$ up to a factor $2^{O(\frac{1}{\epsilon})}$.*

We have two avenues to pursue. The running time will be lower-bounded by the space we require, and we improve this bottleneck to get linear space clustering, but first, to achieve scalability, our goal will be to get clustering in time $\tilde{O}(nk)$. This will mean an amortized update of $O(k\ polylog(n))$. In the next section we will motivate how to achieve this, and provide evidence that ours is a hard bound for the running time of a clustering algorithm.

The second issue is to present an algorithm with approximation guarantee which is polynomial in $\frac{1}{\epsilon}$. We will show how to achieve this in Section **??**.

# 4 $\tilde{O}(nk)$ Time Clustering of Data Streams

Let us recall the algorithm we have developed so far. We have $k^2 \ll M$, and we are applying an alternate implementation of a multi-level algorithm.

We are clustering $m = O(M)$ (assuming $M = O(n^{\epsilon})$ for constant $\epsilon > 0$) points and storing $2k$ medians to "compress" the description of these data points. We use the local search-based algorithm in [?]. We keep repeating this procedure till we see $m$ of these descriptors or intermediate medians and compress them further into $2k$. Finally, when we are required to output a clustering, we compress all the intermediate medians (over all the levels there will be at most $O(M)$ of them) and get $O(k)$ penultimate medians which we cluster into exactly $k$ using the primal dual algorithm as in [?, ?].

## 4.1 Previous results on $\tilde{O}(nk)$ time clustering

We will use the results in [?] on metric space algorithms that are subquadratic. The algorithm as defined will consist of two passes and will have constant probability of success. For high probability results, the algorithm will make $O(\log n)$ passes. As stated, the algorithm will only work if the original data points are *unweighted.* Consider the following algorithm:

1. Draw a sample of size $s = \sqrt{nk}$.
2. Find $k$ medians from these $s$ points using the primal dual algorithm in [?].
3. Assign each of the $n$ original points to its closest median.
4. Collect the $n/s$ points with the largest assignment distance.
5. Find $k$ medians from among these $n/s$ points.
6. We have at this point $2k$ medians.

**Theorem 4.1** *[?] The above algorithm gives an $O(1)$ approximation with $2k$ medians with constant probability.*

The above algorithm [3] provides a constant-factor approximation for the $k$–Median problem (using $2k$ medians) with constant probability. Repeat the above experiment $O(\log n)$ times for high probability. We will not run this algorithm by itself, but as a substep in our algorithm. The algorithm requires $\tilde{O}(nk)$ time and space. Using this algorithm with the local search tradeoff results in [?] reduces the space requirement to $O(\sqrt{nk})$.

Alternate sampling-based results exist even in the weighted case under various assumptions like Euclidean space [?].

### 4.2 Extension to weighted case

We need this sampling-based algorithm to work on weighted input. It is necessary to draw a random sample based on the weights of the points; otherwise the medians with respect to the sample do not convey much information. The simple idea of sampling points with respect to their weights does not help. The philosophy of the above method is that a random sample will be reasonable for most points, that there will not be many outliers (at most $n$ divided by the sample size, up to constants), and that in the second phase it is sufficient to account for these outliers.

If the points have weights, however, in the first step we may only eliminate $k$ points. Therefore sampling according to weights does not carry through. Contrast this with the algorithm in [?] where the points were in Euclidean space and the measure was sum of squares of distances. Both these facts were crucial for their algorithm.

We suggest the following modification. The basic idea is scaling. We can round the weights to the nearest power of $(1+\epsilon)$ for $\epsilon > 0$. In each group we can ignore the weight and lose a $(1+\epsilon)$ factor. Since we have an $\tilde{O}(nk)$ algorithm, summing over all groups, the running time is still $\tilde{O}(nk)$. The correct way to implement this is to compute the exponent values of the weights and use only those groups which exist, otherwise the running time will depend on the largest weight.

### 4.3 The Full Algorithm

We will use this sampling-based scheme to develop a one-pass and $O(nk)$-time algorithm that requires only $O(n^\epsilon)$ space.

- Input the first $O(M/k)$ points, and use the randomized algorithm above to cluster this to $2k$ intermediate median points.
- Use a local search algorithm to cluster $O(M)$ intermediate medians of level $i$ to $2k$ medians of level $i+1$.
- Use the primal dual algorithm of Jain and Vazirani [?] to cluster the final $O(k)$ medians to $k$ medians.

Notice that the algorithm remains one pass, since the $O(\log n)$ iterations of the randomized subalgorithm just add to the running time. Thus, over the first phase, the contribution to the running time is $\tilde{O}(nk)$. Over the next level, we have $\frac{nk}{M}$ points, and if we cluster $O(M)$ of these at a time taking $O(M^2)$ time, the total time for the second phase is $O(nk)$ again. The contribution from the rest of the levels decreases geometrically, so the running time is $\tilde{O}(nk)$. As shown in the previous sections, the number of levels in this algorithm is $O(\log_{\frac{M}{k}} n)$, and so we have a constant factor approximation for $k \ll M = \theta(n^\epsilon)$ for some small $\epsilon$. [4]

Thus we claim the following theorem,

**Theorem 4.2** *The $k$–Median problem has a constant-factor approximation algorithm running in time $O(nk\log n)$, in one pass over the data set, using $n^\epsilon$ memory, for small $k$.*

[3] The algorithm presented here, without the last step, is essentially the same as in [?], however the primal dual algorithm which requires $O(n^2)$ time to solve $k$–Median problem was not known when the result was published. The result proved therein was using $O(n^2k^2)$ local search algorithm in [?] which was a bicriterion approximation.

[4] We could have used the sampling-based algorithm in the intermediate steps as well, however such a recursive, sampling-based algorithm will have greater errors, in theory and very likely in practice.

# 5 Lower Bounds and Deterministic Algorithms

In this section we explore whether our algorithms could be speeded up further and whether randomization is needed. For the former, note that we have a clustering algorithm that requires time $\tilde{O}(nk)$ and a natural question is could we have done better? We'll show that we couldn't have done much better since a deterministic lower bound for $k$–Median is $\Omega(nk)$. Thus, modulo randomization, our time bounds pretty much match the lower bound. For the latter, we show one way to get rid of randomization that yields a single pass, small memory $k$–Median algorithm that is a poly-$\log n$ approximation. Thus we do also have a deterministic algorithm, but with more loss of clustering quality.

## 5.1 Lower Bounds

We now show that any constant-factor deterministic approximation algorithm requires $\Omega(nk)$ time. We measure the running time by the number of times the algorithm queries the distance function.

We consider a restricted family of sets of points where there exists a k-clustering with the property that the distance between any pair of points in the same cluster is 0 and the distance between any pair of points in different clusters is 1. Since the optimum $k$-clustering has value 0 (where the *value* is the distance from points to nearest centers), any algorithm that doesn't discover the optimum k-clustering does not find a constant factor approximation.

Note that the above problem is equivalent to the following Graph $k$-Partition Problem: Given a graph $G$ which is a complete $k$-partite graph for some $k$, 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 $v_i, v_j$ iff $\mathrm{dist}(s_i, s_j) > 0$. Observe that a constant-factor $k$-clustering can be computed with $t$ queries to the distance function iff a graph $k$-partition can be computed with $t$ queries to the adjacency matrix of $G$.

Kavraki, Latombe, Motwani, and Raghavan [?] 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 5.1** *A deterministic $k$–Median algorithm must make $\Omega(nk)$ queries to the distance function to achieve a constant factor approximation.*

## 5.2 Deterministic Algorithms in $\tilde{O}(nk)$ time

One natural question we can ask ourselves is what we can achieve if we do not have randomization. We already have seen how to get an $O(n^{1+\epsilon})$-time clustering algorithm in $n^\epsilon$ space which gives a constant factor approximation. However this constant factor grows as $2^{\frac{1}{\epsilon}}$ and if we were to ask for an $\tilde{O}(nk)$-time algorithm we would have an approximation factor polynomial in $(n/k)$. Modifying our approach slightly, we can show the following,

**Theorem 5.2** *In $\tilde{O}(nk)$ deterministic time, we have a poly-$\log n$ approximation for the $k$–Median problem in $n^\epsilon$ space and a single pass.*

**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 which gives a constant factor (say $c$) approximation for the $k$–Median problem. This algorithm takes time (and space) $an^2$ some constant $a$. (We will have to evaluate the constants in the running time, as we will shortly see that these will blow up.) Now consider the algorithm that chooses $p_1$ partitions of equal size and applies the algorithm to these partitions repeatedly, and then to the $nk/p_1$ points so obtained, to get $k$ points. If we choose $p_1 = (n/k)^{\frac{2}{3}}$, the running time of the algorithm is $2an^{\frac{4}{3}}k^{\frac{2}{3}}$. Call this algorithm $A_1$. The space required by this algorithm is $2an^{\frac{4}{3}}k^{\frac{2}{3}}$ also. However by Theorem **??** we have an approximation of $2c^2 + 3c$.

Now define $A_2$ to partition the dataset in $p_2$ partitions and apply $A_1$ on all of them and 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 $4an^{\frac{16}{15}}k^{\frac{14}{16}}$.

If we carry this process on that $A_i$ calls $A_{i-1}$ on $p_i$ partitions, without much difficulty we can prove that the running time and the space required by the algorithm will be $a2^i n^{\left(1+\frac{1}{2^{2^{i-1}}-1}\right)} k^{\left(1-\frac{1}{2^{2^{i-1}}-1}\right)}$. However the approximation factor $c_j$ grows as $c_j = 2c_{j-1}^2 + c_{j-1}$.

To get the exponent of $n$ in the running time to be 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 $O(\log^p n)$ approximation in $n^\epsilon$ space and $\tilde{O}(nk)$ time, without using randomization. □

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

**Theorem 5.3** *The $k$–Median problem can be approximated in time $\tilde{O}(n^{1+\epsilon\delta})$ and space $\theta(n^\delta)$ up to a factor of $O(poly(\frac{1}{\epsilon})2^{\frac{1}{\delta}})$.*

# Acknowledgments

We thank Umesh Dayal, Aris Gionis, Meichun Hsu, Piotr Indyk, Dan Oblinger, and Bin Zhang for numerous fruitful discussions.

# References

[1] M. Charikar, C. Chekuri, T. Feder and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997.

[2] M. Charikar, and S. Guha. Improved Combinatorial Algorithms for the Facility Location and and $k$–Median Problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 378-388, 1999.

[3] M. Charikar, S. Guha, É. Tardos and D. B. Shmoys. A constant factor approximation algorithm for the $k$–Median problem. in *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 1-10, 1999.

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

[5] 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.

[6] J. Feigenbaum, S. Kannan, M. Strauss, and M. Vishwanathan. An approximate $L^1$–difference algorithm for massive data sets. In *Proceedings of 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 501-511, 1999.

[7] M. R. Henzinger, P .Raghavan, and S. Rajagopalan. Computing on Data Streams Technical Report 1998-011, Digital Equipment Corporation, Systems Research Center, May 1998.

[8] 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.

[9] P. Indyk, Sublinear time algorithms for metric space problems. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 428-434, 1999.

[10] K. Jain and V. Vazirani, Primal-Dual Approximation Algorithms for Metric Facility Location and $k$–Median Problems. In *Proceedings of the*

*40th Annual IEEE Symposium on Foundations of Computer Science*, pages 1-10, 1999.

[11] V. Kann, S. Khanna, J. Lagergren, and A. Panconesi. On the hardness of approximating MAX k–cut and its dual.

[12] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–10, 1998.

[13] G .S .Manku, S. Rajagopalan, and B. Lindsay. Approximate medians and other quantiles in one pass with limited memory. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 426-435, 1998.

[14] G .S .Manku, S. Rajagopalan, and B. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large databases. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 251-262, 1999.

[15] N. Mishra, D. Oblinger, and L. Pitt. Sublinear Time Approximate (PAC) Clustering. Manuscript, 2000.

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

[17] A. Weber. Ueber den Standort der Industrien. Erster Teil. Reine Theorie der Standorte. Mit einem mathematischen Anhang von G.PICK. (in German). Verlag, J. C. B. Mohr, Tbingen, Germany, 1909.