Streaming Algorithms for *k*-Center Clustering with Outliers and with Anonymity

Richard Matthew McCutchen[⋆] and Samir Khuller[⋆⋆]

University of Maryland
{rmccutch,samir}@cs.umd.edu

Abstract. Clustering is a common problem in the analysis of large data sets. *Streaming* algorithms, which make a single pass over the data set using small working memory and produce a clustering comparable in cost to the optimal offline solution, are especially useful. We develop the first streaming algorithms achieving a constant-factor approximation to the cluster radius for two variations of the k-center clustering problem. We give a streaming $(4+\epsilon)$-approximation algorithm using $O(\epsilon^{-1}kz)$ memory for the problem with *outliers*, in which the clustering is allowed to drop up to z of the input points; previous work used a random sampling approach which yields only a bicriteria approximation. We also give a streaming $(6+\epsilon)$-approximation algorithm using $O(\epsilon^{-1}\ln(\epsilon^{-1})k + k^2)$ memory for a variation motivated by anonymity considerations in which each cluster must contain at least a certain number of input points.

Keywords: clustering, k-center, streaming, outliers, anonymity.

1 Introduction

Clustering is a common problem arising in the analysis of large data sets. For many applications in document and image classification [3,9,13,15,16] and data mining, clustering plays a central role [5]. In a typical clustering problem, we have a set of n input points from an arbitrary metric space (with a distance function satisfying the triangle inequality) and wish to partition the points into k clusters. We select a center point for each cluster and consider the distance from each point to the center of the cluster to which it belongs. In the k-center problem, we wish to minimize the maximum of these distances, while in the k-median problem, we wish to minimize their sum. In this paper we focus on k-center clustering, since it is an important problem for which a variety of approaches have been presented. Hochbaum and Shmoys [14] and Gonzalez [10] developed algorithms that achieve a factor 2 approximation in the cluster radius. This is the best possible since one can show by a reduction from the dominating set problem that it is *NP*-hard to approximate k-center with factor $2-\epsilon$ for any $\epsilon > 0$.

[⋆] Supported by an NSF REU Supplement to Award CCF-0430650.
[⋆⋆] Supported by NSF CCF-0430650 and CCF-0728839.

In the analysis of extremely large data sets, it is not possible to hold the entire input in memory at once. Thus, we consider the *streaming* or *incremental* model in which the algorithm reads input points one by one and maintains a valid clustering of the input seen so far using a small amount of working memory. (We contrast such algorithms with *offline* algorithms that use memory polynomial in the size of the input.)

Charikar, Chekuri, Feder and Motwani [5] introduced the incremental model for the k-center problem and gave a very elegant "Doubling Algorithm" that achieves a factor 8 approximation using only $O(k)$ memory. The result is slightly surprising, since it is not obvious at all how to do this incrementally. The key idea is to maintain a lower bound on the radius of an optimal solution. For example, after $k + 1$ input points have been presented, examining the closest pair of points gives us an obvious lower bound on the optimal radius, since at least two of these points must belong to the same cluster.

The key focus of this paper is to deal with outliers, an issue originally raised in [7]. Data is often noisy and a very small number of outliers can dramatically affect the quality of the solution if not taken into account, especially under the k-center objective function, which is extremely sensitive to the existence of points far from cluster centers. The formal definition of the problem is as follows: group *all but z points* into k clusters, minimizing the radius of the largest cluster. An offline factor 3 approximation for the outlier version was developed [7]; it greedily chooses clusters of a certain radius so as to cover as many new input points with each cluster as possible. The factor 3 assumes that we can enumerate all center points in the metric space that the optimal clustering is allowed to use. If not, the algorithm is easily modified to produce a clustering that uses only input points as centers but has radius at most 4 times that of an optimal clustering with unrestricted centers.[1] The same paper also considered the k-median objective function and developed a bicriteria algorithm: if there exists a solution of cost C that drops z outliers, it finds one of cost at most $O(C)$ that drops at most $O(z)$ outliers. More recently, a polynomial time algorithm has been developed for k-medians that delivers a solution of cost $O(C)$ while dropping only z outliers [8].

The offline algorithm for k-center clustering with outliers is not easily adapted to the streaming model because it relies on the ability to count the input points that would be covered by a potential cluster, which is difficult to implement without having the entire data set in memory. In general, dealing with outliers in the streaming model is quite tricky because we have no way to know, as points arrive, which should be clustered and which are outliers. This problem was first considered by Charikar, O'Callaghan and Panigrahy [6], who developed a streaming-model bicriteria approximation for the k-center problem (see also [12]). Their approach is based on taking a random sample of the data set that is small enough to fit in memory and running the offline algorithm [7] on the sample. They then prove that, with high probability, the set of clusters found for the sample is also a good solution for the entire data set. This construction preserves the radius approximation factor of the underlying offline algorithm

[1] We simply expand the disks G_i to radius $2r$ and the disks E_i to radius $4r$.

(3 or 4) but increases the number of outliers to $(1+\epsilon)^2 z$. The sample has size roughly $O(\epsilon^{-2}kn/z)$, where n is the data set size. Therefore, the sampling approach is good when z is linear in n and a slight increase in the number of outliers is acceptable; otherwise, it requires an unreasonable amount of memory.

We present a streaming algorithm for k-center clustering with outliers that is in several ways complementary to that of [6]. Our deterministic algorithm is based on the Doubling Algorithm [5] and also uses the offline algorithm for outliers [7] as a subroutine. It increases the radius approximation factor to $3+\epsilon$ or $4+\epsilon$ but meets the outlier bound z exactly; as far as we are aware, it is the first streaming constant-factor approximation with the latter property. Our algorithm uses $O(\epsilon^{-1}kz)$ memory, so it is suitable when z is *small* and an additional slight increase in the *cluster radius* is acceptable.

Agarwal et al. [1] present an algorithm for shape-fitting with outliers that may be applicable to k-center clustering, and Bădoiu et al. [4] present a sampling-based streaming k-center algorithm that uses coresets. However, both techniques work only in Euclidean spaces $\mathbb{R}^d$; furthermore, the first requires multiple passes over the input and the second has running time exponential in k.

Other recent applications of k-center clustering (with and without outliers) for the purposes of anonymity are considered in [2], but the algorithms given there do not work in the streaming model. We present a streaming $(6+\epsilon)$-approximation algorithm for the k-center clustering problem with a lower bound b on the number of points per cluster. The precise requirement is that it must be possible to allocate each input point to a center within the appropriate radius so that each center gets at least b points, i.e., centers cannot meet the bound by sharing points.

2 Improving Streaming Algorithms by Parallelization

In this section, we develop a parallelization construction that improves the approximation factor of the Doubling Algorithm to $2+\epsilon$ while increasing the running time and memory usage by a factor of $O(\epsilon^{-1}\ln(\epsilon^{-1}))$.[2] We first generalize the Doubling Algorithm to a "Scaling Algorithm" based on a parameter $\alpha > 1$ that maintains a lower bound r on the radius of the optimal cluster and raises it by a factor of exactly α at a time. As it reads points, this algorithm keeps centers separated by at least $2\alpha r$ and ensures that every input point seen so far is within $\eta r = \big(2\alpha^2/(\alpha-1)\big)r$ of a center. Let r^* denote the optimal radius, and let r_0 be half the least distance between two of the first $k+1$ distinct input points, which is used to initialize r.

Naively, the Scaling Algorithm is an η-approximation because it gives us a solution with radius within a factor η of its own lower bound r, and we minimize $\eta = 8$ by choosing $\alpha = 2$. But observe that if $r^* = 1.9r_0$, we get lucky: the algorithm cannot raise r to $2r_0$ because $2r_0$ is not a lower bound on r^*, so it is

[2] Sudipto Guha independently discovered a similar construction [11] based on Gonzalez's algorithm [10]; it also yields a streaming $(2+\epsilon)$-approximation algorithm for k-center clustering.

obliged to return a solution with radius at most $8r_0$, which is only a factor of about 4.2 from optimal.

To ensure that we always get lucky in this way, we run m instances of the Scaling Algorithm in parallel (feeding each input point to each instance) with interleaved sequences of r values. Specifically, we initialize the r value of the ith instance ($i = 1, \ldots, m$) to $\alpha^{(i/m)-1}r_0$ so that the instance takes on the r values $\alpha^{t+(i/m)-1}r_0$, where $t = 0, 1, \ldots$. Consequently, any desired r of the form $\alpha^{(j/m)-1}r_0$ for a positive integer j will eventually be taken on by some instance. Letting j be the smallest integer greater than $m \log_\alpha(r^*/r_0)$, we have $\alpha^{j/m}r_0 > r^*$, so the instance that takes $r = \alpha^{(j/m)-1}r_0$ will be unable to raise r again and thus will return a solution whose radius R is at most $\eta\alpha^{(j/m)-1}r_0$. And by our choice of j, $\alpha^{(j-1)/m}r_0 \le r^*$, so $R \le \eta\alpha^{(1/m)-1}r^*$. Therefore, by taking the best solution produced by any of the m instances, we achieve a factor $(\eta/\alpha)\sqrt[m]{\alpha}$ approximation.

Substituting the expression for η, the approximation factor of the parallelized algorithm becomes $2\big(1+1/(\alpha-1)\big)\sqrt[m]{\alpha}$. Now, we want α large to make $1/(\alpha-1)$ small; intuitively, with larger α, accounting for previous rounds across an increase of r costs less in the Scaling Algorithm's approximation factor. We also want m large to keep $\sqrt[m]{\alpha}$ close to 1. Letting $\alpha = O(\epsilon^{-1})$ and $m = O(\epsilon^{-1}\ln(\epsilon^{-1}))$ gives a factor of $2 + \epsilon$. This approximation factor is essentially the best we can hope for since the best offline algorithms [14,10] are 2-approximations, but there may be a better construction that uses less time and memory. We will apply the same parallelization construction to the streaming-model clustering algorithms described in the following sections.

2.1 Suitability of Parallelized Algorithms

The original model of Charikar et al. [5] requires that a clustering algorithm maintain a single clustering of the input points read so far and modify it only by merging clusters. This model has the advantage that a forest describing the merges can be incrementally written to secondary storage; the forest can later be traversed to enumerate the input points in any desired output cluster without a second pass over the entire input. Parallelized algorithms do not fit this model because they maintain many clusterings and do not choose one until the end. (This is why they do not contradict the lower bound of $1+\sqrt{2}$ on the approximation factor in [5].) Writing out a forest for each of the many partial clusterings under consideration may be impractical. However, parallelized algorithms are still useful when the goal is only to produce statistics for each output cluster (along with the centers themselves) because the statistics can be maintained independently for each partial clustering.

3 Clustering with Outliers

In this section, we develop a streaming algorithm for k-center clustering with z outliers that achieves a constant factor approximation to the cluster radius using $O(kz)$ memory. We then parallelize it to a $(4+\epsilon)$-approximation using $O(\epsilon^{-1}kz)$

memory. The essential difficulty in designing a deterministic streaming algorithm for clustering *with outliers* is that it is dangerous to designate an input point as a cluster center and start forgetting nearby points because they could all be outliers and the center might be needed to cover points elsewhere. Our algorithm overcomes the difficulty by delaying the decision as long as necessary. Specifically, it accumulates input points (remembering all of them) until it sees $z + 1$ points close together. These cannot all be outliers, so it creates a cluster for them and only then can safely forget any later points that fall in that cluster.

The algorithm's state consists of:

- some number $\ell \leq k$ of stored cluster centers, each of which carries a list of $z + 1$ nearby "support points" from which it was originally formed;
- some "free points" that do not fall into existing clusters but cannot yet be made into new clusters because they might be outliers; and
- a lower bound r on the optimal radius, as in the Doubling Algorithm.

The algorithm ensures that clusters of radius ηr at the ℓ stored centers cover all forgotten points, and it checks after processing each input point that it can cover all but at most z of the free points with $k - \ell$ additional clusters of radius ηr. Thus, whenever the algorithm encounters the end of the input, it can produce a solution with radius ηr. The algorithm is based on parameters α, β, and η, which we will choose later to optimize the approximation factor; for the proof of correctness to hold, these parameters must satisfy some constraints that we will state as they arise.

The algorithm is designed so that, whenever its partial solution with radius ηr becomes invalid, it can establish a new lower bound αr on the optimal radius, raise r by a factor of α, and adapt the partial solution to the new value of r; this process is repeated until the validity of the partial solution is restored. Furthermore, we will show that the algorithm will never store more than $O(kz)$ free points at a time, establishing the memory requirement.

As in the Doubling Algorithm [5], we need a certain separation between centers in order to raise r. To this end, we say that two distinct centers *conflict* if some support point of the first is within distance $2\alpha r$ of some support point of the second.

Algorithm 3.1 (Clustering with outliers). Peek at the first $k+z+1$ distinct input points, initialize r to half the least distance between any two of those points, and start with no cluster centers and no free points. Then read the input points in batches. Batches of size kz appear to give the best trade-off between running time and memory usage, but a different size can be used if desired. For each batch, add the points as free points and then perform the following procedure:

1. Drop any free points that are within distance ηr of cluster centers.
2. If some free point p has at least $z+1$ free points within distance βr (including itself), then add p as a cluster center, choosing any $z + 1$ of the free points within distance βr as its support points, and repeat from step 1. If no such p exists, proceed to the next step.

3. Let ℓ be the number of stored cluster centers. Check that $\ell \leq k$ and that at most $(k - \ell)z + z$ free points are stored. Run the 4-approximation offline algorithm for k-center clustering with outliers (see the Introduction) to attempt to cover all but at most z of the free points using $k - \ell$ clusters of radius ηr. If the checks and the offline algorithm both succeed, processing of the current input batch is complete. Otherwise, set $r \leftarrow \alpha r$ and continue to the next step.
4. Unmark all the stored centers and then process them as follows: while there exists an unmarked center c, mark c and drop any other centers that conflict with c with respect to the new value of r. When a center is dropped, its support points are forgotten. (Note that once a center c is marked, it cannot later be dropped on account of another center c' because c' would already have been dropped on account of c.) Repeat from step 1.

When the end of the input is reached, return clusters of radius ηr at the stored centers plus the clusters found by the last run of the offline algorithm. □

Figure 1 shows an intermediate state of the algorithm on a data set with $k = 3$ and $z = 4$. The algorithm is storing $\ell = 2$ cluster centers c_1 and c_2, and each center has $z+1 = 5$ support points (including itself), which are within βr of it. Several other input points within distance ηr of the centers have been forgotten. The algorithm

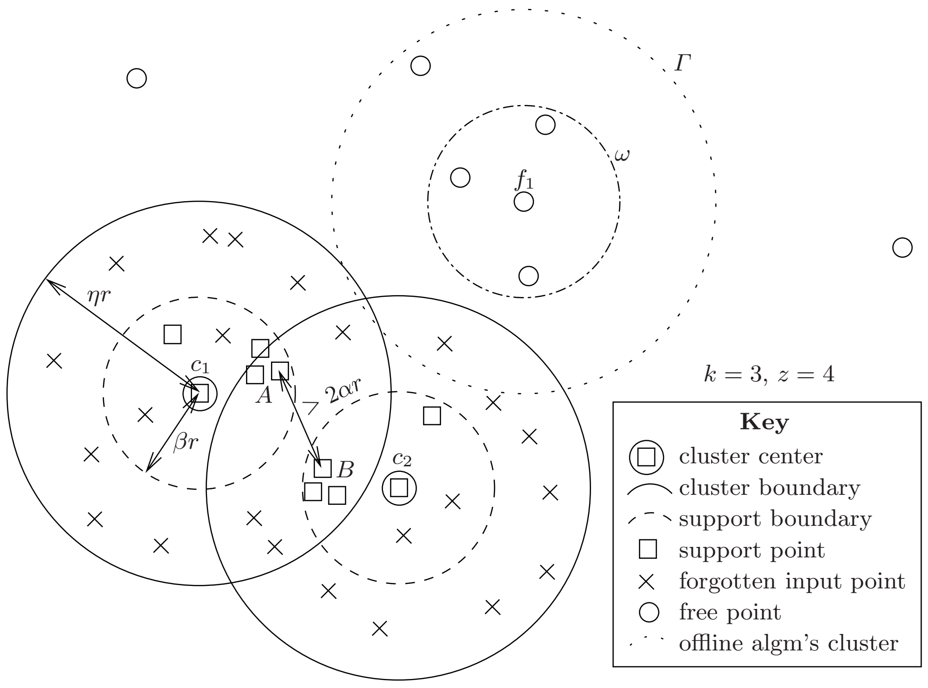

Fig. 1. Example of clustering with outliers

is also storing seven free points, including f_1, which would be converted to a cluster center if there were just one more free point inside circle ω; but as it stands, the algorithm cannot rule out the possibility that all four of the points in ω are outliers. The offline algorithm found the cluster Γ (centered at f_1), which covers all but $2 \leq z$ of the free points; if we combine it with the stored centers, we have a valid clustering of radius ηr for the input points seen so far.

Notice that the three support points A are just far enough from the support points B to avoid a conflict. If they were any closer, then the optimal solution could conceivably cover all six points with a single cluster of radius αr and leave the remaining four support points as outliers, and the proof of correctness of the algorithm's decision to set $r \leftarrow \alpha r$ (see Lemma 3.2(e) below) would fail.

Suppose several free points arrive inside Γ but outside ω. The current clustering covers these points, but if the algorithm allowed them to accumulate indefinitely, it would violate the $O(kz)$ memory bound. Thus, when the number of free points exceeds $(k - \ell)z + z = 8$, the algorithm raises r on the following logic: in the optimal solution, two clusters are busy covering support points of the stored centers, and there is no way a third cluster of radius αr containing at most $z = 4$ points can cover all the free points with at most 4 outliers. (If there were a potential third cluster of more than 4 points, the algorithm would already have recognized it in step 2.) Once r is raised, the support points A and B conflict, so one of the centers c_1, c_2 subsumes the other in step 4.

Lemma 3.2. *The algorithm maintains the following invariants:*

(a) Every time step 1 completes, the remaining free points are at least distance ηr from cluster centers.
(b) Each stored center has $z + 1$ support points within distance βr of it.
(c) No two stored cluster centers conflict.
(d) Every input point the algorithm has read so far either is a free point or is covered by a cluster of radius ηr at a stored center.
(e) The optimal clustering for the input points the algorithm has read so far requires a radius of at least r.

Proof. (a) is obvious. (b) is checked when a center is added and remains true when r increases.

To prove (c), we place the constraint $\eta \geq 2\alpha + \beta$ on our later choice of the parameters. With this constraint, addition of a center c in step 2 preserves the invariant. For if s_1 is a support point of an existing center c_1, then s_1 is within βr of c_1, which is at least ηr from c's support points (since they were previously free points). By the triangle inequality, s_1 is at least distance $\eta r - \beta r \geq 2\alpha r$ from c's support points, so no conflict results. Furthermore, temporary conflicts created by an increase in r are removed in step 4.

Each point the algorithm reads is initially a free point, so the algorithm endangers invariant (d) only when it drops free points or centers. Free points dropped in step 1 are covered by stored centers, so they do not break the invariant. Steps 3 and 4 effectively drop some clusters while expanding the remaining ones to radius $\eta\alpha r$; we must show that any input point that was covered by a dropped cluster before the change is covered by an expanded cluster afterwards. To this end, we constrain $\eta + 2\alpha^2 + 2\beta \leq \eta\alpha$. Let r_0 and $r_1 = \alpha r_0$ denote the old and new values of r, respectively. Consider an input point p that was covered by a dropped center c, meaning that it was within distance ηr_0 of c. c was dropped when a conflicting center c' was marked. The support points causing the conflict were within distance $2\alpha r_1$ of each other and distance βr_0 of their respective centers, so the distance from p to c' is at most

$$\eta r_0 + \beta r_0 + 2\alpha r_1 + \beta r_0 = (\eta + 2\alpha^2 + 2\beta) r_0 \leq \eta\alpha r_0 = \eta r_1.$$

Thus, p is covered by c', and the invariant holds.

Invariant (e) is established by the initial setting of r because one of the k clusters of the optimal solution must cover two of the first $k + z + 1$ distinct input points. To show that increases in r maintain the invariant, we will show that, if step 3 is reached and the optimal clustering C^* for the input read so far has radius *less than* αr, then the algorithm does *not* set $r \leftarrow \alpha r$.

Let c be a stored cluster center. C^* cannot designate all $z + 1$ of c's support points as outliers, so some cluster $c^* \in C^*$ must cover one of c's support points; we say that c^* *bites* c. No two stored cluster centers conflict, so no cluster of C^* (having diameter less than $2\alpha r$) can bite two of them; thus, each stored cluster center is bitten by a different cluster of C^*. In particular, this means $\ell \leq k$. Similarly, by invariant (a) and our assumption that $\eta \geq 2\alpha + \beta$, no cluster of C^* can both bite a stored cluster center and cover a free point. Finally, we constrain $\beta \geq 2\alpha$; then no cluster of C^* can cover $z + 1$ or more free points because, if it did, each of those free points would be within distance βr of all the others and they would have become the support points of a cluster center in step 2.

Now, at least ℓ of the clusters of C^* are devoted to biting stored cluster centers, so at most $k - \ell$ clusters can cover free points; let F^* be the set of these clusters. In order for C^* to be a valid solution for the input read so far, F^* must be a valid clustering of all but at most z of the free points. But we showed that each of the at most $k - \ell$ clusters in F^* covers at most z free points, so there can be at most $(k - \ell)z + z$ free points in total. Finally, the offline algorithm is a 4-approximation, so if we assume that $\eta \geq 4\alpha$, the existence of F^* with radius less than αr guarantees that the offline algorithm will find a clustering of radius ηr. The result is that r is not raised, as desired. □

Theorem 3.3. *The algorithm produces a valid clustering of radius ηr using $O(kz)$ memory and $O(kzn + (kz)^2 \log P)$ time, where P is the ratio of the optimal radius to the shortest distance between any two distinct input points.*

Proof. Validity of the clustering: The first set of ℓ clusters covers all input points except the free points by Lemma 3.2(d), and the second set of $k - \ell$ clusters covers all but at most z of the free points. Thus, together, the k clusters cover all but at most z of the input points.

Memory usage: At any time, the algorithm remembers $\ell(z + 1)$ support points (including the centers), at most $(k - \ell)z + z$ free points from before the current batch, and at most kz free points from the current batch. This is a total of at most $(2k+1)(z+1)$ points, and the working storage needed to carry out the steps is constant per point.

Running time: At the beginning of a batch, we perform step 1 exhaustively in $O(k^2z)$ time. We identify potential centers in step 2 by maintaining a count for each free point p of the free points within distance βr of p. Each time we add or drop a free point, which happens at most $O(kz)$ times per batch, we perform a scan of the other free points to update their counts (this takes $O(kz)$ time). When we convert a free point c to a center, we identify its support points and the free points to be dropped in step 1 on the same scan that drops c itself as a free point. The offline algorithm in step 3 runs in $O((kz)^2)$ time using its own set of distance-$\eta r/2$ counts; we charge a successful run of the offline algorithm, which ends a batch, to that batch and a failed run to the resulting increase in r. In step 4, we have $O(k)$ centers (k from the previous batch and at most one per $z+1$ of the $O(kz)$ free points), so we test each of the $O(k^2)$ pairs of centers for a conflict in $O(z^2)$ time; this takes $O((kz)^2)$ time, which we charge to the increase in r. Now, there are $O(n/kz)$ batches and $O(\log P)$ increases in r, and each batch or increase is charged $O((kz)^2)$ time, giving the desired bound. $\square$

The construction in Section 2 yields an m-instance parallelized algorithm with approximation factor $(\eta/\alpha)\sqrt[m]{\alpha}$. We wish to choose the parameters to minimize this factor. We have the constraints:

$$\eta \geq 2\alpha + \beta \tag{1}$$
$$\eta\alpha \geq \eta + 2\alpha^2 + 2\beta \tag{2}$$
$$\beta \geq 2\alpha \tag{3}$$
$$\eta \geq 4\alpha \tag{4}$$

Setting $\alpha = 4$, $\beta = 8$, and $\eta = 16$ satisfies the constraints and gives an approximation factor of $4^{1+(1/m)}$, so we can achieve a $(4+\epsilon)$-approximation with $m = O(\epsilon^{-1})$. The memory usage and running time of the parallelized algorithm increase by a factor of m to $O(\epsilon^{-1}kz)$ and $O(\epsilon^{-1}(kzn + (kz)^2 \log P))$. Note that two things limit the approximation performance: that of the offline algorithm via (4), and the constraints (3) and (1) that limit what an optimal cluster can do. Thus, an improvement in the approximation factor of the offline algorithm will not carry through to the streaming algorithm unless it comes with a correspondingly better way to analyze optimal clusters.

3.1 Improvement Using a Center-Finding Oracle

There is a $(3+\epsilon)$-approximation version of the streaming algorithm, corresponding to the 3-approximation offline algorithm, when the metric space comes with a *center-finding* oracle. Given a positive integer j, a distance x, and a point set S, the oracle returns a point p having at least j points of S within distance x or announces that no such p exists in the metric space. Such an oracle may be impractical to implement in high-dimensional spaces, but when one is available, we can use it to improve the algorithm.

In step 2, instead of looking for potential centers among the free points, we invoke the oracle with $x = \beta r$, $j = z + 1$, and S being the current set of free points, and we add the resulting point (if any) as a center. Now, when the oracle fails, we know there is no cluster of radius βr centered *anywhere* that covers more than z free points, so we can relax constraint (3) to $\beta \geq \alpha$. In step 3, we substitute the 3-approximation offline algorithm, choosing centers using the oracle, and hence relax constraint (4) to $\eta \geq 3\alpha$. With the modified constraints, we choose $\alpha = \beta = 5$ and $\eta = 15$ to achieve a $(3 + \epsilon)$-approximation with the same $O(\epsilon^{-1}kz)$ memory usage; the running time depends on that of the oracle.

4 Clustering with Anonymity

For the problem of k-center clustering with a lower bound b on the number of points per cluster, we present a construction based on the parallelized Scaling Algorithm of Section 2 that achieves a $(6 + \epsilon)$-approximation. Applications of this problem for anonymity are considered by Aggarwal et al. [2].

Algorithm 4.1 (Clustering with anonymity). Let $\delta = \epsilon/2$. First run the m-instance parallelized Scaling Algorithm with m chosen to achieve a $(2 + \delta)$-approximation, but modify it to keep a count of how many input points "belong" to each center under an assignment of each point to a center within distance $(2 + \delta)r$ of it. (The algorithm does not store this assignment explicitly, but we use it in the proof of correctness.) When an existing center catches a new input point, the center's count is incremented, and when centers are merged, their counts are added. The Scaling Algorithm returns a lower bound r on the radius of the optimal k-center clustering of the input, a list of k *preliminary* centers $c_1, \ldots, c_k$, and the number n_i of input points belonging to each preliminary center c_i.

If $n_i \geq b$ for all i, the preliminary centers c_i constitute a solution within factor $2 + \delta$ of the optimal and we are done. Otherwise, we merge some centers using a scheme resembling the offline algorithm for k-center clustering with anonymity [2]. Given a merging radius R, the scheme works as follows. Initialize all preliminary centers to inactive; then, while there exists a preliminary center c that has no active center within distance $2R$, activate c. Next, attempt to allocate each input point p (belonging to a preliminary center c) to an active center within distance $2R + (2 + \delta)r$ of c in such a way that each active center gets at least b input points. To do this, construct a bipartite graph on the sets P of preliminary centers and A of currently active centers with an edge of infinite capacity connecting a node $x \in P$ to a node $y \in A$ if their distance is at most $2R + (2 + \delta)r$. Add a source s with an edge of capacity n_i to each $c_i \in P$ and a sink t with an edge of capacity b from each $c_i \in A$, and compute a max flow from s to t. If this flow saturates all edges entering t, it represents a valid allocation of the input points, which the merging scheme returns.

We attempt the merging scheme for various values of R in a binary search (which need only consider values of the form $d/2$ and $(d - (2 + \delta)r)/2$ for intercenter distances d) and keep the successful allocation with the smallest value of R. The algorithm returns a clustering consisting of the active centers under this allocation with radius $(4+2\delta)r+2R$. □

Theorem 4.2. *The algorithm produces a clustering with at least b points per cluster whose radius is at most $6+\epsilon = 6+2\delta$ times that of the optimal such clustering.*

Proof. Every input point p belongs to a preliminary center c within distance $(2+\delta)r$ of it and is allocated to an active center c' within distance $2R+(2+\delta)r$ of c, so p is within distance $(4+2\delta)r+2R$ of c'. The algorithm's clustering consists of the active centers, so the clustering covers every input point at radius $(4+2\delta)r+2R$ by virtue of the active center to which the point is allocated. Furthermore, each active center is allocated b points within distance $(4{+}2\delta)r{+}2R$ of it. Therefore, the algorithm's clustering is valid. We must show that it is a $(6+2\delta)$-approximation.

Let r^* be the radius of the optimal clustering, and consider an execution of the merging scheme with $R \geq r^*$. Active centers are separated by more than $2R \geq 2r^*$ by construction, so each lies in a different optimal cluster. We now claim that there exists an allocation of the form sought by the merging scheme, namely the allocation A that gives each input point to the unique active center (if any) lying in its optimal cluster. Let p be an input point; since optimal clusters have diameter $2r^*$, A gives p to an active center c within distance $2r^*$ of it. At the end of the Scaling Algorithm, p belonged to a center c' within distance $(2+\delta)r$, so the distance between c and c' is at most $2r^*+(2+\delta)r \leq 2R+(2+\delta)r$. Thus, the merging scheme could legally allocate p (as counted by c') to c. This is true of every input point p, so the claim is established. Consequently, the merging scheme must succeed whenever $R \geq r^*$.

Thus, when the algorithm takes the smallest R for which the merging scheme succeeds, it will take an $R \leq r^*$. (The algorithm might consider not r^* itself but a slightly smaller value of R for which the merging scheme makes all the same decisions and therefore still must succeed.) The Scaling Algorithm ensures that r is a lower bound on r^*, i.e., $r \leq r^*$. Combining these two inequalities, the radius $(4+2\delta)r+2R$ of the algorithm's clustering is at most $6+2\delta$ times the optimal radius r^*, as desired. □

Theorem 4.3. *The algorithm runs in $O(m(kn+k^2\log P)+k^3\log k)$ time using $O(mk+k^2)$ memory, where $m = O(\epsilon^{-1}\ln(\epsilon^{-1}))$ and P is the ratio of the optimal radius to the shortest distance between any two distinct input points.*

Proof. We use the simple $O(k)$-memory implementation of the Scaling Algorithm that stores only the centers; it performs each of the $O(\log P)$ scalings in $O(k^2)$ time and otherwise processes each point in $O(k)$ time for a total running time of $O(kn+k^2\log P)$. Parallelization multiplies these bounds by m. The running time of the second phase is dominated by the max flow computation, which is done $O(\log k)$ times because there are $O(k^2)$ possible values for R. Using the relabel-to-front algorithm, each max flow invocation takes $O(k^3)$ time and $O(k^2)$ memory. The desired bounds follow. □

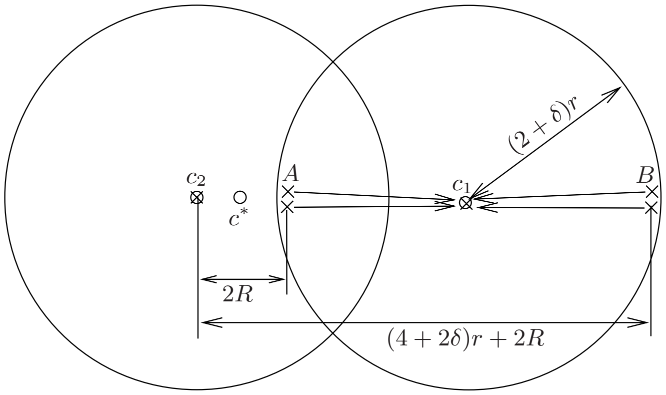

Fig. 2. Example of clustering with anonymity, with $b = 3$

The example in Figure 2 should help clarify the argument and motivate the final radius of $(4 + 2\delta)r + 2R$. We have two preliminary clusters c_1 and c_2 of radius $(2 + \delta)r$ with $n_1 = 5$ and $n_2 = 1$. The Scaling Algorithm decided to make the points A belong to c_1 even though they are actually much closer to c_2, perhaps because c_2 was not created until after they were read. Suppose we activate c_2 in the merging scheme. All we know about it is that it is in some optimal cluster of diameter $2r^*$ that contains b input points. For example, suppose $b = 3$ and there is an optimal cluster centered at c^* (not an input point) that contains c_2 and the points A. In order to guarantee that we can successfully allocate three points to c_2 whenever $R \geq r^*$, we must make all input points within distance $2R$ of c_2 (here the points A) available for allocation to it. But these points could belong to a different center (here c_1) that is another $(2+\delta)r$ away, so to be sure of catching them, we must allow c_2 to take points belonging to centers up to $2R + (2 + \delta)r$ away. However, the algorithm knows only that the five points belonging to c_1 are within $(2 + \delta)r$ of it; it knows nothing else about where they lie. In allowing c_2 to take points from c_1 to ensure that it has access to the points A, we are also opening the possibility of it taking the points B, which are $(4 + 2\delta)r + 2R$ away; there is no obvious way to avoid this. Thus, we set the radius of the final clustering to $(4 + 2\delta)r + 2R$ to make sure the clustering is valid. (For example, if $b = 6$, the algorithm might return a clustering consisting of a single cluster centered at c_2 containing all six points, which would need radius $(4+2\delta)r+2R$.)

The best-known offline algorithm [2], which essentially performs the allocation phase without the perturbation caused by the initial Scaling Algorithm phase, achieves an approximation factor of 2. Whether there is a streaming algorithm with a factor closer to 2 is an open problem.

4.1 Computing Per-cluster Statistics

Suppose that we wish to compute some statistics about the points allocated to each cluster as well as the center itself. In most cases, the algorithm can be extended to compute the statistics without a second pass over the input. For example, consider a medical application in which each input point represents a person who may or may not have a certain disease, and suppose we want to know what percentage of the people in each cluster have the disease. The first phase already maintains a count of people belonging to each center, and we can maintain a second count of people with the disease in the same way. When we allocate the people belonging to a preliminary center in the second phase, we simply allocate the people with the disease in the same proportion. For example, suppose 100 people belong to a preliminary center c_1 and 11 of them have the disease; if we allocate 30 of these 100 people to an active center c_2, we assume that 3.3 of them have the disease. In effect, we are allocating to c_2 30% of each individual who belongs to c_1. The fractionality of the allocation may appear silly but does not really harm the statistics.

In the same way, if we want the average height of the people in each cluster, we can maintain a "total height" value for each center, allocate height values in proportion to people, and then divide the total height allocated to a cluster by the number of people allocated to it. We can even compute several statistics on the same run. In full generality, if each input point comes with a vector of real-number weights, we can compute a total-weight vector for each cluster and divide by the number of points if we desire averages.

5 Conclusions

It is probably possible to combine our techniques for clustering with outliers and with anonymity to obtain an algorithm for the problem with both outliers and anonymity (albeit with a worse approximation factor), but we have not investigated this. One obvious open problem is to find an algorithm for the outlier problem with better running time and memory usage than our approach or the sampling approach of [6], particularly for the case where neither z nor n/z is small, or to prove a lower bound on the amount of memory needed.

If we are allowed multiple passes over the input, we can use a scaling-style algorithm to determine the optimal radius up to a constant factor on the first pass and then bound it more tightly on each subsequent pass by testing multiple guesses in parallel. By spreading the work across passes, we achieve the same approximation factor with a much smaller number of parallel instances. (The basic Hochbaum-Shmoys method [14] works naturally for guess-checking in the streaming model, but the offline algorithm for outliers [7] does not; one could instead use a cut-down guess-checking version of our outlier algorithm.) Developing a better algorithm that fully exploits multiple passes to achieve the same approximation factor using even less memory is another open problem.

Acknowledgment. The authors are grateful to Sudipto Guha for useful discussions.

References

1. Agarwal, P., Har-Peled, S., Yu, H.: Robust shape fitting via peeling and grating coresets. In: Proc. of ACM Symp. on Discrete Algorithms (SODA), pp. 182–191 (2006)
2. Aggarwal, G., Feder, T., Kenthapadi, K., Khuller, S., Panigrahy, R., Thomas, D., Zhu, A.: Achieving anonymity via clustering. In: Proc. of ACM Principles of Database Systems (PODS), pp. 153–162 (2006)
3. Aldenderfer, M.S., Blashfield, R.K.: Cluster Analysis. Sage, Beverly Hills (1984)
4. Bădoiu, M., Har-Peled, S., Indyk, P.: Approximate clustering via core-sets. In: Proc. of ACM Symp. on Theory of Computing (STOC), pp. 250–257 (2002)
5. Charikar, M., Chekuri, C., Feder, T., Motwani, R.: Incremental clustering and dynamic infomation retrieval. In: Proc. of ACM Symp. on Theory of Computing (STOC), pp. 626–635 (1997)
6. Charikar, M., O'Callaghan, L., Panigrahy, R.: Better streaming algorithms for clustering problems. In: Proc. of ACM Symp. on Theory of Computing (STOC), pp. 30–39 (2003)
7. Charikar, M., Khuller, S., Mount, D., Narasimhan, G.: Algorithms for facility location problems with outliers. In: Proc. of ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 642–651 (2001)
8. Chen, K.: A constant factor approximation algorithm for k-median clustering with outliers. In: Proc. of ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 826–835 (2008)
9. Everitt, B.: Cluster Analysis. Heinemann Educational, London (1974)
10. Gonzalez, T.: Clustering to minimize the maximum inter-cluster distance. Theoretical Computer Science 38, 293–306 (1985)
11. Guha, S.: The k-center karma of a data stream (unpublished manuscript) (2007)
12. Guha, S., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams. In: Proc. of IEEE Foundations of Computer Science (FOCS), pp. 359–366 (2000)
13. Hartigan, J.A.: Clustering Algorithms. Wiley, Chichester (1975)
14. Hochbaum, D., Shmoys, D.B.: A best possible approximation algorithm for the k-center problem. Math. of Operations Research 10, 180–184 (1985)
15. Jain, A.K., Dubes, R.C.: Algorithms for clustering data. Prentice Hall, NJ (1988)
16. Rasmussen, E.: Clustering algorithms. In: Frakes, W., Baeza-Yates, R. (eds.) Information Retrieval: Data Structures and Algorithms. Prentice Hall, Englewood Cliffs (1992)