International Journal of Computational Geometry
& Applications
Vol. 24, No. 2 (2014) 107–123

DOI: 10.1142/S0218195914500058

COMPUTING k CENTERS OVER STREAMING DATA FOR SMALL k*

HEE-KAP AHN†, HYO-SIL KIM‡, SANG-SUB KIM§ and WANBIN SON¶
Department of Computer Science and Engineering, POSTECH
Pohang, Korea
†*heekap@postech.ac.kr*
‡*hyosil.kim@gmail.com*
§*helmet1981@postech.ac.kr*
¶*mnbiny@postech.ac.kr*

Received 9 May 2013
Revised 10 March 2014
Communicated by Timothy M. Chan

ABSTRACT

In this paper, we consider the k-center problem for streaming points in $\mathbb{R}^d$. More precisely, we consider the single-pass streaming model, where each point in the stream is allowed to be examined only once and a small amount of information can be stored in a device. Since the size of memory is much smaller than the size of the data in the streaming model, it is important to develop an algorithm whose space complexity does not depend on the number of input data. We present an approximation algorithm for $k = 2$ that guarantees a $(2 + \varepsilon)$-factor using $O(d/\varepsilon)$ space and update time in arbitrary dimensions for any metric. We show that our algorithm can be extended to approximate an optimal k-center within factor $(2 + \varepsilon)$ for $k > 2$.

Keywords: Streaming algorithms; k-center; approximation algorithms.

1. Introduction

Clustering is the task of partitioning a given set into subsets, called *clusters*, subject to various objective functions. As one of the fundamental problems raised in facility location, clustering also plays an important role in many applications such as data mining,[14,19] image processing,[26] and astrophysics.[7,13]

Most of previous work on clustering in the literature assumes the static setting (off-line), that is, data is known in advance. It is not so easy, however, to keep all data in a memory as the amount of data has significantly increased over the last

*This research is supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

decades. This massive data set made the *streaming model* extremely popular. In this model, it is important to design an algorithm whose space complexity does not depend on the size of input, since the memory size is typically much smaller than the input size. We consider the *single-pass streaming model*,[5] where elements in data stream are allowed to be examined only once and a limited amount of information can be stored in a device.

Among various clustering problems, we consider the *minmax radius clustering*, also known as the *k-center problem*: Given a set P of n points in d-dimensional metric space, find k points called *centers* such that the maximum distance between a point in P and its nearest center is minimized. (Here, the k-center problem asks to return such a minmax distance (radius) as well as the k centers.) The k-center problem can also be formulated to find k congruent balls such that their union covers P and their common radius is minimized. The *discrete* k-center problem requires that obtained k centers be a subset of P.

In this paper, we study the k-center problem in the single-pass streaming model for small constant k.

Previous work in the static setting. If k is a part of input, the k-center problem is NP-hard,[16] even in the plane.[24] In fact, it is known to be NP-hard to approximate the k-center problem with a factor smaller than 2 for arbitrary metric spaces, and with a factor smaller than 1.822 for the Euclidean space.[6] Gonzalez[17] gave a 2-approximation algorithm that runs in $O(kn)$-time for any metric space, and Feder and Greene[15] improved the running time to $O(n \log k)$ for any L_p-metric. For fixed k and d, Agarwal and Procopiuc[3] gave an exact algorithm that runs in $n^{O(k^{1-(1/d)})}$ time for any L_p-metric. Note that if d is not fixed, even the Euclidean (L_2-metric) k-center problem and the rectilinear (L_∞-metric) k-center problem are NP-hard for fixed $k \geqslant 2$ and $k \geqslant 3$, respectively.[23]

If k and d are small constants, there are more efficient algorithms. For the Euclidean 1-center problem (computing the minimum enclosing ball), there is a linear time algorithm for any fixed dimension.[12] For the Euclidean 2-center problem in the plane, the best known algorithm is given by Chan,[8] which runs deterministically in $O(n \log^2 n \log^2 \log n)$ time using $O(n)$ space. In three dimensions, Agarwal *et al.*[1] gave two randomized algorithms; one runs in $O(n^3 \log^8 n)$ expected time and the other runs in $O(n^2 \log^8 n/(1 - r^*/r_0)^3)$ expected time, where r^* is the radius of an optimal 2-center and r_0 is the radius of the optimal 1-center.

Previous work on data streams. The *coreset* framework is one of the fundamental tools for designing streaming algorithms since it captures an approximate "shape" of input in small size.[2,9,27] For the k-center problem, Zarrabi-Zadeh[21] showed a method of maintaining an ε-coreset using $O(k/\varepsilon^d)$ space under any L_p-metric, which is, to our best knowledge, the only known result for obtaining an ε-coreset for the k-center problem whose space complexity does not depend on n.

While the coreset framework guarantees an ε-approximate solution to the problem, its exponential dependency on d is not so attractive in high dimensions. Sacrificing the factor, some work using space polynomial in d has been presented; Charikar *et al.*[11] gave an 8-approximation algorithm using $O(dk)$ space, and Guha,[18] in parallel with McCutchen and Khuller,[22] gave a $(2+\varepsilon)$-approximation algorithm using $O((dk/\varepsilon)\log(1/\varepsilon))$ space to the k-center problem for any metric space. (Note that Charikar *et al.*,[11] Guha[18] and McCutchen and Khuller[22] did not explicitly mention d in the complexity analysis since they assume that there is an oracle that computes the distance between two points in $O(1)$ time.)

For small k, especially for the Euclidean 1-center, the problem has been studied extensively. In fixed dimensions, one can devise a $(1+\varepsilon)$-approximation algorithm using $O(1/\varepsilon^{(d-1)/2})$ space by maintaining extreme points along a number of different directions, which can be considered as a generalization of the algorithm given by Hershberger and Suri.[20] For arbitrary dimensions, Zarrabi-Zadeh and Chan[28] gave a 1.5-approximation algorithm that maintains only one center and one radius, which is the minimum amount of storage to maintain a single ball. Agarwal and Sharathkumar[4] developed a $(1+\sqrt{3})/2+\varepsilon \approx 1.3661$-approximation algorithm using $O((d/\varepsilon^3)\log(1/\varepsilon))$ space and showed that any algorithm in the single-pass stream with space polynomially bounded in d cannot approximate the optimal 1-center within factor $(1+\sqrt{2})/2 > 1.207$. Chan and Pathak[10] improved the approximation factor to 1.22 by analyzing the algorithm of Agarwal and Sharathkumar[4] more carefully.

For the Euclidean 2-center problem, Poon and Zhu[25] proposed an algorithm that guarantees a 2-approximation for $d = 1$ and a 5.708-approximation for $d > 1$ using the minimum space, namely two centers and one radius.

Our results. We first present an approximation algorithm MELP for computing an optimal 2-center in the single-pass streaming model. MELP guarantees a $(2+\varepsilon)$-approximation using $O(d/\varepsilon)$ space and $O(d/\varepsilon)$ update time for arbitrary dimensions d under any metric space, with the assumption that it takes $O(d)$ time for computing the distance between any two points. This algorithm works in the discrete setting as well. Our algorithm improves by a $\log(1/\varepsilon)$ factor the space and time complexity of Guha's and McCutchen and Khuller's $(2+\varepsilon)$-approximation algorithm for $k = 2$.

Using the algorithms for the 1-center and the 2-center problems as base cases, we develop an algorithm that approximates an optimal k-center within factor $(2+\varepsilon)$ using $O((k+3)! \cdot 2^k d/\varepsilon)$ space and $O((k+3)!d/\varepsilon)$ update time. The complexities of our algorithm are extremely high for large k, but we believe that it is reasonable for small k and ε. For a small constant k, our algorithm spends $O(d/\varepsilon)$ space and $O(dn/\varepsilon)$ time in total, with a reasonably small hidden constant in the complexity, while Guha's algorithm and McCutchen and Khuller's algorithm use $O((dn/\varepsilon)\log(1/\varepsilon) + (d/\varepsilon)\log r^*)$ time and $O((dn/\varepsilon)\log(1/\varepsilon))$ time in total, respectively, with $O((d/\varepsilon)\log(1/\varepsilon))$ space (here, r^* is the optimal radius of the two clusters). We remark the details in Sec. 5.

Table 1 shows a summary of results of the 2-center problem and the k-center problem for $k \geqslant 3$. Note that Guha[18] only mentioned total time complexity of his algorithm, so we use an amortized analysis (by dividing the total time complexity by n) for update time of Guha's algorithm in Table 1.

Table 1. Results for the 2-center problem and the k-center problem over the single-pass streaming model, with the assumption that it takes $O(d)$ time for computing the distance between any two points. $O^*(x)$ denotes $O(x)$ amortized time.

	Algorithm	Factor	Space	Update time
2-center	Our result	$2+\varepsilon$	$O(d/\varepsilon)$	$O(d/\varepsilon)$
	Charikar *et al.*[11]	8	$O(d)$	$O(d)$
	Poon and Zhu[25]	5.708	two centers, one radius	$O(d)$
	Guha[18]	$2+\varepsilon$	$O((d/\varepsilon)\log(1/\varepsilon))$	$O^*((d/\varepsilon)\log(1/\varepsilon) + (d/n\varepsilon)\log r^*)$
	McCutchen and Khuller[22]	$2+\varepsilon$	$O((d/\varepsilon)\log(1/\varepsilon))$	$O((d/\varepsilon)\log(1/\varepsilon))$
	Zarrabi-Zadeh[21]	$1+\varepsilon$	$O(1/\varepsilon^d)$	$O^*(1)$
k-center	Our result	$2+\varepsilon$	$O((k+3)!\cdot 2^k d/\varepsilon)$	$O((k+3)!d/\varepsilon)$
	Charikar *et al.*[11]	8	$O(dk)$	$O(dk^2)$
	Guha[18]	$2+\varepsilon$	$O((dk/\varepsilon)\log(1/\varepsilon))$	$O^*((dk/\varepsilon)\log(1/\varepsilon) + (dk/n\varepsilon)\log r^*)$
	McCutchen and Khuller[22]	$2+\varepsilon$	$O((dk/\varepsilon)\log(1/\varepsilon))$	$O((dk^2/\varepsilon)\log(1/\varepsilon))$
	Zarrabi-Zadeh[21]	$1+\varepsilon$	$O(k/\varepsilon^d)$	$O^*(1)$

2. Preliminaries

Let P be a set of n points in d-dimensional metric space. In the single-pass streaming model, the points in P are arriving one by one, and are allowed to be examined only once. The points in P are labeled in order of their arrivals. That is, p_i is a point in P that arrives at the i-th step. We denote by P_i a subset of points in P that have arrived until the i-th step, that is, $P_i = \{p_1, p_2, \ldots, p_i\}$.

Let $B(c, r)$ denote a ball of radius r centered at c, and let $r(B)$ and $c(B)$ denote the radius and the center of a ball B, respectively. The distance between any two points p, q in the space is denoted by $|pq|$.

An *optimal solution* to the 2-center problem or an *optimal* 2*-center* is a solution of the 2-center problem in $\mathbb{R}^d$, when all points in P are assumed to be known in advance. Let B_1^* and B_2^* denote the two congruent balls in an optimal solution, c_1^* and c_2^* denote their centers, r^* denote their radius, and δ^* denote the distance between B_1^* and B_2^*, that is, $\delta^* = |c_1^* c_2^*| - 2r^*$. Let P_1^* and P_2^* be subsets of P such that $P_1^* = P \cap B_1^*$ and $P_2^* = P \cap B_2^*$. If B_1^* and B_2^* are disjoint, so are P_1^* and P_2^*. If the notations are used without *, they denote an approximate solution generated

by our algorithms. The notations for the k-center problem are naturally extended from above.

Overview. The rest of the paper is organized as follows. In the following two sections, we consider the 2-center problem. In Sec. 3, we assume that the points in P are well separated ($\delta^* > 2r^*$) and devise a procedure MergeExpand, which partitions P optimally (Corollary 1). Note that even though this subroutine partitions P optimally into P_1^* and P_2^*, we are not able to obtain an exact solution because of the limited storage. By approximating the points in each partition using a ball, MergeExpand guarantees a 2-approximation to the 2-center problem using $O(d)$ space and $O(d)$ update time for any metric (Lemma 1).

In Sec. 4 we assume that the points in P are not well separated ($\delta^* \leqslant 2r^*$) and devise a subroutine LayerPartition which partitions the space into $O(1/\varepsilon)$ *layers*. These layers are concentric balls centered at p_1. We prove that one of the layers guarantees a $(2+\varepsilon)$-approximation to the 2-center problem for any metric (Lemma 3). Table 2 compares the two subroutines that are used in Secs. 3 and 4.

Table 2. Subroutines that are used to approximate a 2-center.

Subroutines	Condition	Factor	Space	Update time
MergeExpand	$\delta^* > 2r^*$	2	$O(d)$	$O(d)$
LayerPartition	$\delta^* \leqslant 2r^*$	$2+\varepsilon$	$O(d/\varepsilon)$	$O(d/\varepsilon)$

In Sec. 5, we present a $(2+\varepsilon)$-approximation algorithm to the k-center problem. We again consider two cases: (i) the distance from p_1 to its farthest point in P is at most $4kr^*$, and (ii) the distance is greater than $4kr^*$. The first case can be solved using a technique similar to LayerPartition. For the second case, assuming that we have data structures that return a $(2+\varepsilon)$-approximation to the t-center problem for $t = 1, \ldots, k-1$, we can solve it for the k-center problem. Using the data structures for the 1-center and the 2-center problems as base cases, we can recursively build a data structure for the k-center problem that uses $O((k+3)! \cdot 2^k d/\varepsilon)$ space in total and $O((k+3)!d/\varepsilon)$ update time.

We conclude the paper with a summary and open questions in Sec. 6.

3. The 2-Center Problem: The Case $\delta^* > 2r^*$

We introduce a subroutine MergeExpand that can be used for $\delta^* > 2r^*$. MergeExpand always maintains a ball B_U centered at p_1 whose radius is the distance between p_1 and its farthest point among the points of P that have arrived so far. It also maintains two congruent balls, B_1 and B_2, whose centers and common radius constitute a solution to the 2-center problem for the points of P that have arrived so far. The basic procedure of MergeExpand works as follows: It initializes B_1 and B_2 to the first two points p_1 and p_2, respectively. The ball B_U is initialized to the ball centered at p_1 with radius $|p_1p_2|$. When a new point p_i arrives, if p_i is not contained in $B_1 \cup B_2$, MergeExpand updates the current two balls B_1 and B_2

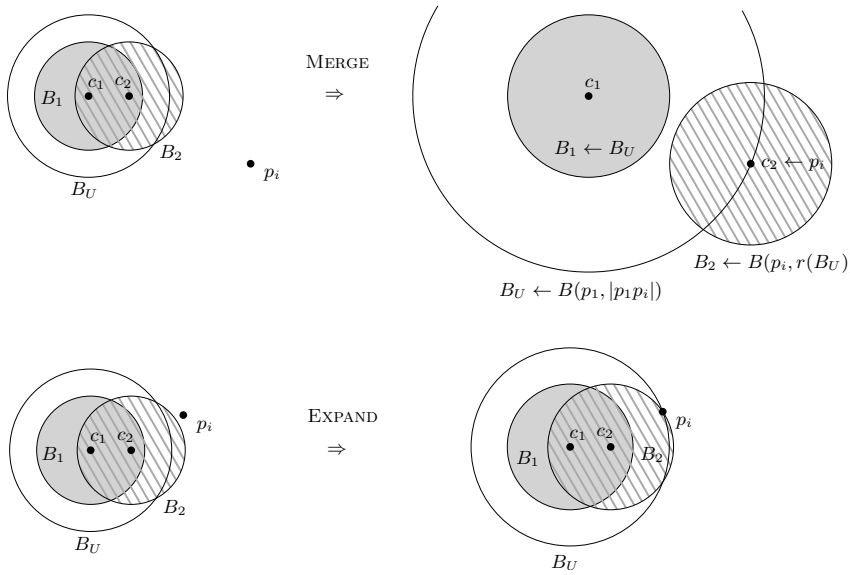

Fig. 1. When p_i arrives, MERGE occurs if $r(B_U) \leqslant \min\{|c_1p_i|, |c_2p_i|\}$, and EXPAND occurs otherwise.

using either MERGE- or EXPAND-operation. MERGE replaces the current two balls B_1 and B_2 with B_U and $B(p_i, r(B_U))$, respectively. EXPAND replaces B_1 and B_2 with two new balls such that the centers remain the same but the common radius becomes $\min\{|c_1p_i|, |c_2, p_i|\}$. MERGEEXPAND always chooses the operation which makes the updated common radius smaller (see Fig. 1). Finally, MERGEEXPAND updates B_U if p_i is not contained in B_U.

The precise description of the subroutine is given as follows:

Algorithm MergeExpand
Input. A sequence of points $\{p_1, p_2, \ldots, p_n\}$
Output. Two congruent balls B_1 and B_2

1. $B_1 \leftarrow B(p_1, 0)$
2. $B_2 \leftarrow B(p_2, 0)$
3. $B_U \leftarrow B(p_1, |p_1p_2|)$
4. **for** $i \leftarrow 3$ **to** n
5. mindist $\leftarrow \min\{|c_1p_i|, |c_2p_i|\}$
6. **if** $r(B_1) <$ mindist /* if p_i is not contained in $B_1 \cup B_2$ */
7. **then if** $r(B_U) \leqslant$ mindist
8. **then** $B_1 \leftarrow B_U, B_2 \leftarrow B(p_i, r(B_U))$. /* MERGE */
9. **else** $B_1 \leftarrow B(c_1, \text{mindist})$, $B_2 \leftarrow B(c_2, \text{mindist})$ /* EXPAND */
10. **if** $p_i \notin B_U$
11. **then** $B_U \leftarrow B(p_1, |p_1p_i|)$
12. **return** B_1 and B_2

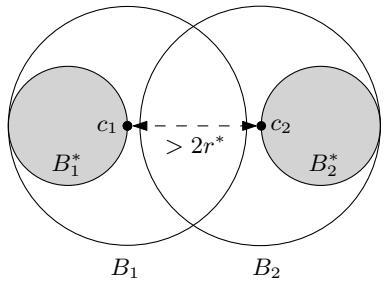

Fig. 2. MergeExpand returns an optimal partition of P for $\delta^* > 2r^*$.

Note that MergeExpand returns *discrete* two centers, which means that the two centers are two points of P. Especially, B_1 and B_U always take p_1 as their centers. The space that the subroutine maintains is thus the coordinates of the centers of B_1 and B_2, and the radii of B_1 and B_U.

Lemma 1. *For $\delta^* > 2r^*$,* MergeExpand *computes a 2-approximation to the 2-center problem using $O(d)$ space and update time in arbitrary dimensions d under any metric.*

Proof. Assume that $P_{i-1} \subset B_1^*$ and p_i is the first point that lies in B_2^*. We claim that this is the last moment when a Merge-operation occurs. Indeed, at the end of the $(i-1)$-th step, we have $r(B_U) \leqslant 2r^*$ and $c_1, c_2 \in B_1^*$. When p_i arrives at the i-th step, we have $\min\{|c_1p_i|, |c_2p_i|\} \geqslant \delta^* > 2r^*$ and therefore a Merge-operation occurs. Then the algorithm changes the center c_2 of B_2 to $p_i \in B_2^*$, while c_1 remains the same. The radius of B_U gets larger to $|p_1p_i| \geqslant \delta^*$. For any forthcoming point p_j with $j > i$, we have either $p_j \in B_1^*$ or $p_j \in B_2^*$, and therefore $\min\{|c_1p_j|, |c_2p_j|\} \leqslant 2r^* < \delta^*$. This implies that $r(B_U) > \min\{|c_1p_j|, |c_2p_j|\}$ and the claim follows.

Now we claim that $r(B_1) = r(B_2) \leqslant 2r^*$. Until the $(i-1)$-th step, the claim holds because $P_{i-1} \in B_1^*$ and $c_1, c_2 \in B_1^*$. At the i-th step, the last Merge-operation occurs and $c_2 = p_i \in B_2^*$. We have $r(B_1) = r(B_2) \leqslant 2r^*$. From the $(i+1)$-th step, no Merge-operation occurs and therefore two centers remain the same, that is, $c_1 = p_1 \in B_1^*$ and $c_2 = p_i \in B_2^*$. This implies that $|c_1p| \leqslant 2r^*$ for any $p \in P \cap B_1^*$ and $|c_2q| \leqslant 2r^*$ for any $q \in P \cap B_2^*$.

For update time, line 5 of MergeExpand takes $O(d)$ time for computing distances between a pair of points in d dimensional space while the remaining lines 6–11 of the **for** loop take only $O(1)$ time. □

It is not difficult to see that for $\delta^* > 2r^*$, MergeExpand guarantees an *optimal partition* of P into two disjoint subsets, which means $P_1^* \subset B_1$, $P_2^* \subset B_2$, $P_1^* \cap B_2 = \emptyset$, and $P_2^* \cap B_1 = \emptyset$ (see Fig. 2).

Corollary 1. *For $\delta^* > 2r^*$,* MergeExpand *returns an optimal partition of P.*

4. The 2-Center Problem: The Case $\delta^* \leqslant 2r^*$

For the case $\delta^* \leqslant 2r^*$, it is not easy to partition the point set optimally, since the two balls of an optimal solution are too close or overlap and we cannot keep enough information in the single-pass streaming setting.

To overcome the difficulty, Guha's algorithm[18] maintains a number of candidate solutions such that they have different thresholds for r^* from each other. Once a candidate solution s becomes invalid, the algorithm constructs a new candidate solution by using the k centers of s and an increased threshold for r^*. To guarantee an approximation factor they maintain $O((1/\varepsilon)\log(1/\varepsilon))$ candidate solutions for $k = 2$.

The idea we use in this section is similar to that of Guha's algorithm: maintain a small number of candidates and choose the best one among them. But the difference is that each candidate solution of our algorithm is associated with a partition of points. Once a new candidate solution is needed, our algorithm simply constructs it by defining a new partition. Because we consider only the case $\delta^* \leqslant 2r^*$, we maintain $O(1/\varepsilon)$ candidates that are enough to guarantee the approximation factor. We discuss the details in the following.

Data structures. The subroutine LayerPartition always maintains $m = \lceil 12/\varepsilon \rceil$ *concentric* balls centered at p_1. Let $\mathcal{B}_i = \{b_1, b_2, \ldots, b_m\}$ denote the set of such m balls for P_i. To make it clear, we call b_j the j-th *layer* of $\mathcal{B}_i$. For each layer b_j, the point set P_i is partitioned into two disjoint subsets: $P_{ij} := P_i \cap b_j$ and $P_{ij}^\circ := P_i \setminus P_{ij}$. For each layer b_j, we maintain its associated ball b_j° with radius $r(b_j)$ that encloses all points in P_{ij}°. The first point that lies outside b_j becomes the center of b_j°. If there is any point lying outside $b_j \cup b_j^\circ$, we set the layer $b_j :=$ invalid and do not consider it any further.

Initialization. The subroutine starts from two input points p_1 and p_2. Initially, it creates m concentric balls (layers) centered at p_1. The radius of each layer b_j is $(j/m) \cdot |p_1 p_2|$ for $j = 1, 2, \ldots, m$. We also initialize the associated ball of each layer to be $\emptyset$.

Update. When a point $p_i \in P$ arrives ($i \geqslant 2$), we have two cases: p_i is contained in b_m of $\mathcal{B}_{i-1}$ or not. When p_i is contained in b_m, let b_j be the smallest layer in $\mathcal{B}_{i-1}$ that contains p_i. For each "valid" layer b_k with $1 \leqslant k < j$, we update the information of b_k° as follows. If $b_k^\circ = \emptyset$, we let $b_k^\circ := B(p_i, r(b_k))$. If $b_k^\circ \neq \emptyset$ but $p_i \notin b_k^\circ$, we let $b_k :=$ invalid (see Figs. 3(a)–3(c)).

If p_i is not contained in b_m, we update the m layers one by one in increasing order of radius. For ease of explanation, let b_j denote the j-th layer before the update and let $\overline{b_j}$ denote the j-th layer after the update. Let x be the integer

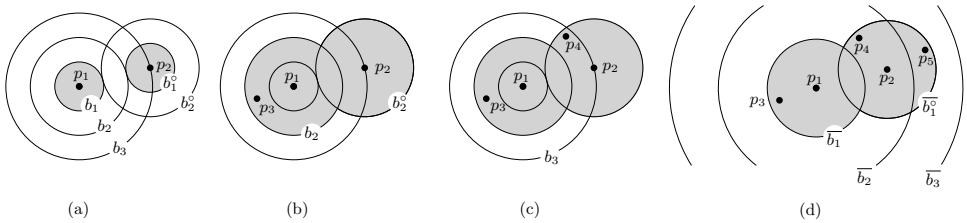

Fig. 3. Three layers b_1, b_2, b_3 and their associated balls. Initially, all $b_1^\circ, b_2^\circ, b_3^\circ$ are set to $\emptyset$. (a) When p_2 arrives, we let $b_i^\circ := B(p_2, r(b_i))$ for $i = 1, 2$. (b) When p_3 arrives, we let $b_1 :=$ invalid. (c) When p_4 arrives, we do nothing. (d) When p_5 arrives, we create new three layers $\overline{b_1}, \overline{b_2}$ and $\overline{b_3}$. Since layer $\overline{b_1}$ coincides with b_2, we let $\overline{b_1^\circ} := b_2^\circ$. $\overline{b_2^\circ} = B(p_5, r(\overline{b_2}))$, and $\overline{b_3^\circ} = \emptyset$.

satisfying $2^x r(b_m) \geqslant |p_1 p_i| > 2^{x-1} r(b_m)$. Then $\overline{b_j}$ is a ball centered at p_1 with radius $(j/m) \cdot 2^x r(b_m)$ for $j = 1, \ldots, m$.

For j satisfying $2^x \cdot j \leqslant m$, layer $\overline{b_j}$ coincides with layer b_k with $k = 2^x \cdot j$. If b_k is invalid or b_k° is not empty but does not contain p_i, we let $\overline{b_j} :=$ invalid. Otherwise, we let

$$\overline{b_j^\circ} := \begin{cases} B(p_i, r(\overline{b_j})) & \text{if } b_k^\circ \text{ is } \emptyset \\ b_k^\circ & \text{if } b_k^\circ \text{ contains } p_i. \end{cases}$$

For j satisfying $2^x \cdot j > m$, we let $\overline{b_j^\circ} := B(p_i, r(\overline{b_j}))$ if $p_i \notin \overline{b_j}$, and let $\overline{b_j^\circ} := \emptyset$ otherwise (see Fig. 3(d)).

We get the following algorithm:

Algorithm LayerPartition
Input. A sequence of points $\{p_1, p_2, \ldots, p_n\}$ and a positive real value ε
Output. A layer (ball) b and its associated ball b°

```
1.  m ← ⌈12/ε⌉
2.  for j ←1 to m
3.      Create a layer b_j = B(p_1, (j/m) · |p_1p_2|)
4.      Initialize b_j° ←∅
5.  for i ←2 to n
6.      if p_i ∈ b_m
7.        then k ←1
8.             while p_i ∉ b_k
9.                   if b_k ≠ invalid and b_k° = ∅
10.                    then b_k° ←B(p_i, r(b_k))
11.                    else  if b_k ≠ invalid and p_i ∉ b_k°
12.                               then b_k ←invalid
13.                  k ←k + 1
14.       else  Compute an integer x satisfying 2^x r(b_m) ⩾ |p_1p_i| > 2^{x−1} r(b_m)
15.             for j←1 to m
16.                k←2^x · j
```

```
17.                 b_j ← B(p_1, (k/m) · r(b_m))
18.                 if k ⩽ m
19.                   then if b_k ≠ invalid and b_k° = ∅
20.                             then b_j° = B(p_i, r(b_j))
21.                             else if b_k ≠ invalid and p_i ∈ b_k°
22.                                       then b_j° ← b_k°
23.                                       else b_j ← invalid
24.                   else if p_i ∉ b_j
25.                             then b_j° := B(p_i, r(b_j))
26.                             else b_j° := ∅
27. return the smallest valid layer b and its associated ball b°
```

Before we analyze the complexity and approximation factor of algorithm LayerPartition, we need the following lemma.

Lemma 2. *Any two balls with centers on P and radius at least $2r^*$ contain all points of P if no ball contains the center of the other ball.*

Proof. Let B_1 and B_2 be two such balls with centers p and p' of P, respectively. Since $|pp'| > 2r^*$, every optimal pair of balls B_1^* and B_2^* with radius r^* contains p and p' in two different balls. Without loss of generality, assume that $p \in B_1^*$ and $p' \in B_2^*$. Then $B_1^* \subset B_1$ and $B_2^* \subset B_2$, which proves the lemma. □

Lemma 3. *For $\delta^* \leqslant 2r^*$,* LayerPartition *guarantees a $(2+\varepsilon)$-approximation to the 2-center problem using $O(d/\varepsilon)$ space and update time in arbitrary dimensions d under any metric.*

Proof. When the algorithm finishes, we are left with the final m layers $\mathcal{B}_n$. To bound the radius of $\overline{b_j} \in \mathcal{B}_n$ for $j = 1, \ldots, m$, we consider the last point p_i that changes the m layers, that is, we have $\mathcal{B}_{i-1} \neq \mathcal{B}_i$ and $\mathcal{B}_i = \mathcal{B}_{i+1} = \cdots = \mathcal{B}_n$.

Let us denote the j-th layer in $\mathcal{B}_{i-1}$ by b_j for $j = 1, \ldots, m$. By definition $r(\overline{b_m}) = 2^x r(b_m)$, where x satisfies $2^{x-1} r(b_m) < |p_1 p_i| \leqslant 2^x r(b_m)$. Since $\delta^* \leqslant 2r^*$, no pair of points in P has distance larger than $6r^*$, and thus $|p_1 p_i| \leqslant 6r^*$ and $r(\overline{b_m}) = 2^x r(b_m) < 2|p_1 p_i| \leqslant 12r^*$. Since $m = \lceil 12/\varepsilon \rceil$, we have $r(\overline{b_j}) = (j/m) \cdot r(\overline{b_m}) \leqslant j \cdot \varepsilon r^*$ for $j = 1, \ldots, m$.

Let $\overline{b_j}$ be the smallest valid layer. Then the union of $\overline{b_j}$ and its associated ball $\overline{b_j^\circ}$ contains the points of P. If $j = 1$, we have $r(\overline{b_1}) \leqslant \varepsilon r^*$, and $\overline{b_1}$, together with $\overline{b_1^\circ}$, gives a $(2+\varepsilon)$-approximation. Otherwise, we have $\overline{b_{j-1}} = \mathsf{invalid}$ and Lemma 2 implies that $((j-1)/m) \cdot r(\overline{b_m}) < 2r^*$. Therefore,

$$r(\overline{b_j}) = (j/m) \cdot r(\overline{b_m}) = ((j-1)/m) \cdot r(\overline{b_m}) + (1/m) \cdot r(\overline{b_m}) < 2r^* + \varepsilon r^* = (2+\varepsilon) r^*.$$

□

Algorithm MELP executes both MERGEEXPAND and LAYERPARTITION, and then it returns the pair of balls with smaller radius between the solutions obtained by these subroutines.

Theorem 1. *Given $\varepsilon > 0$,* MELP *guarantees a $(2+\varepsilon)$-approximation to the 2-center problem for streaming points using $O(d/\varepsilon)$ space and update time in arbitrary dimensions d under any metric.*

5. Extension to the *k*-Center Problem

Let $\eta(p, P)$ denote a point in P that is the farthest from p. We consider two cases $|p_1\,\eta(p_1, P)| \leqslant 4kr^*$ and $|p_1\,\eta(p_1, P)| > 4kr^*$. In this section, we show a $(2+\varepsilon)$-approximation algorithm to the k-center problem.

5.1. *The case $|p_1\,\eta(p_1, P)| \leqslant 4kr^*$*

In this case, we use a technique similar to LAYERPARTITION. We always maintain $m = \lceil 8k/\varepsilon \rceil$ concentric balls (layers) $b_1, b_2, \ldots, b_m$ centered at p_1, and keep a partitioning information for each layer.

For each layer b_j we maintain $k-1$ associated balls $b^\circ_{j1}, b^\circ_{j2}, \ldots, b^\circ_{j(k-1)}$ with radius $r(b_j)$ as follows. Initially, all associated balls are set to $\emptyset$. The first point lying outside b_j becomes the center of b°_{j1} with radius $r(b_j)$. Similarly, the first point lying outside $b_j \cup (\bigcup_{1 \leqslant t \leqslant \ell-1} b^\circ_{jt})$ becomes the center of $b^\circ_{j\ell}$ with radius $r(b_j)$. If there is a point lying outside $b_j \cup (\bigcup_{1 \leqslant t < k} b^\circ_{jt})$, we let $b_j :=$ invalid and do not consider it any further.

The subroutine starts from the two input points p_1 and p_2, and sets each layer b_j (centered at p_1) to have radius $(j/m) \cdot |p_1 p_2|$ for $j = 1, \ldots, m$. The new layers are created upon arrival of a point lying outside b_m and they are defined similarly to LAYERPARTITION. Let $\mathbf{B}^* = \{B^*_1, B^*_2, \ldots, B^*_k\}$ be an optimal solution to the k-center problem and recall that $\mathcal{B}_n = \{b_1, \ldots, b_m\}$ is the final layers. We now show that there is a valid layer $b_j \in \mathcal{B}_n$ whose radius is at most $(2+\varepsilon)r^*$.

Lemma 4. *For $|p_1\,\eta(p_1, P)| \leqslant 4kr^*$, we can compute a $(2+\varepsilon)$-approximation to the k-center problem using $O(k^2 d/\varepsilon)$ space and update time in arbitrary dimensions d under any metric.*

Proof. We first claim that any k congruent balls $B_1, B_2, \ldots, B_k$ with centers $c_1, c_2, \ldots, c_k$ on P and radius at least $2r^*$ contain all points in P in their union if no ball contains any center of the other balls. (This claim is a generalization of Lemma 2.) Since any pair of centers is distance greater than $2r^*$, no optimal solution contains more than one center in one of its k balls. By the pigeonhole principle, each ball B^*_i of an optimal solution contains exactly one distinct center, say c_j. This implies $B^*_i \subset B_j$, which proves the claim.

Consider the final layers $\mathcal{B}_n = \{b_1, \ldots, b_m\}$. Using the proof in Lemma 3, we can show that for $m = \lceil 8k/\varepsilon \rceil$, each valid layer b_j has radius $r(b_j) = (j/m) \cdot$

$r(b_m) \leqslant j \cdot \varepsilon r^*$ for $j = 1, \ldots, m$. Let b_j be the smallest valid layer. By construction, the union of b_j and its $k-1$ associated balls $b^\circ_{j1}, \ldots, b^\circ_{j(k-1)}$ contains all points of P. We are going to show that $r(b_j) < (2+\varepsilon)r^*$. If $j = 1$, we have $r(b_1) \leqslant \varepsilon r^*$, and b_1, together with its $k-1$ associated balls, gives a $(2+\varepsilon)$-approximation. Otherwise, layer b_{j-1} was set to invalid upon arrival of a point lying outside b_{j-1} and its $k-1$ associated balls, and the claim above implies that b_{j-1} had radius $((j-1)/m) \cdot r(b_m) < 2r^*$. Therefore,

$$r(b_j) = (j/m) \cdot r(b_m) = ((j-1)/m) \cdot r(b_m) + (1/m) \cdot r(b_m) < 2r^* + \varepsilon r^* = (2+\varepsilon)r^*.$$

Since we maintain $O(k/\varepsilon)$ layers and for each layer we maintain k balls, we need $O(k^2 d/\varepsilon)$ space and update time. □

5.2. *The case* $|p_1\, \eta(p_1, P)| > 4kr^*$

In this case, we maintain $m = 2k$ concentric balls (layers) $b_1, \ldots, b_m$ centered at p_1 as we do in LayerPartition. The main idea we use here is as follows. For any optimal solution $\mathbf{B}^*$, there is a layer in $\mathcal{B}_n$ that does not intersect any ball in $\mathbf{B}^*$ and partitions $\mathbf{B}^*$ into two nonempty subsets. We prove this formally in the following lemma.

Lemma 5. *For $|p_1\, \eta(p_1, P)| > 4kr^*$, there is some layer $b_j \in \{b_1, \ldots, b_k\} \subset \mathcal{B}_n$ such that b_j does not intersect any ball in $\mathbf{B}^*$ and partitions $\mathbf{B}^*$ into two nonempty subsets; b_j contains at least one ball in $\mathbf{B}^*$.*

Proof. Let us define the *layer-distance* to be the value $r(b_j) - r(b_{j-1})$ (which is the same for every $j = 2, \ldots, m$). Since $|p_1\, \eta(p_1, P)| > 4kr^*$, the layer-distance is greater than $4kr^*/2k = 2r^*$ by construction. Since every ball in $\mathbf{B}^*$ has diameter $2r^*$ and b_1 has radius greater than $2r^*$, b_1 contains every ball in $\mathbf{B}^*$ that contains p_1. Note that there is at least one ball of $\mathbf{B}^*$ that contains p_1 and therefore at least one ball of $\mathbf{B}^*$ is contained in b_1.

Now we claim that $\eta(p_1, P) \notin b_k$. To see this, consider the last point p_i that changes the m layers, that is, we have $\mathcal{B}_{i-1} \neq \mathcal{B}_i$ and $\mathcal{B}_i = \mathcal{B}_{i+1} = \cdots = \mathcal{B}_n$. This change occurs because the largest layer b_m of $\mathcal{B}_{i-1}$ does not contain p_i. Therefore $\mathcal{B}_i$ consists of new m layers the largest of which has radius $2^x r(b_m) \geqslant |p_1 p_i| > 2^{x-1} r(b_m)$ for some integer x for $b_m \in \mathcal{B}_{i-1}$. For two layers $b_k, b_{2k} \in \mathcal{B}_i$, we have $r(b_{2k}) = 2r(b_k)$ so $p_i \notin b_k$ and therefore $\eta(p_1, P) \notin b_k$. This implies that every ball in $\mathbf{B}^*$ that contains $\eta(p_1, P)$ either intersects the boundary of b_k or lies outside of b_k.

Since the layer-distance is greater than $2r^*$, no ball in $\mathbf{B}^*$ can intersect more than one layer boundary. By the pigeonhole principle, there must be at least one layer b_j for $j = 1, \ldots, k$ whose boundary does not intersect any ball in $\mathbf{B}^*$ and partitions $\mathbf{B}^*$ into two nonempty subsets. □

Let b_ℓ denote the largest layer of $\mathcal{B}_n$ that does not contain $\eta(p_1, P)$. Since $\ell \geqslant k$ we have the following corollary.

Corollary 2. *Any k layers chosen from $b_1, b_2, \ldots, b_\ell$ contain a layer that does not intersect any ball in $\mathbf{B}^*$ and partitions $\mathbf{B}^*$ into two nonempty subsets.*

Data structures. We abuse the definition of b_ℓ such that from now on it denotes the largest layer of the current set $\mathcal{B}_i$ of m layers that does not contain $\eta(p_1, P_i)$. Based on Corollary 2, we choose k layers from $b_1, b_2, \ldots, b_\ell$ and choose $b_{\ell+1}$. More specifically, we choose all even layers $b_2, b_4, \ldots, b_{2(\ell-k)}$ from layers with index at most $2(\ell - k)$ and choose all layers with index from $2(\ell - k) + 1$ to $\ell + 1$. We call each chosen layer a *canonical layer* of $\mathcal{B}_i$. For each canonical layer b, we maintain $k - 1$ pairs of data structures each pair of which consists of a data structure for a t-center of the points lying inside b and a data structure for a $(k - t)$-center of the points lying outside of b for $t = 1, \ldots, k - 1$. For layers that are not canonical, we do not maintain any other information, except the layer itself.

Update. If the next point p_{i+1} is contained in $b_{\ell+1}$, we update the data structures of each canonical layer as follows: we update only the data structures for the points lying inside the layer by adding p_{i+1} if p_{i+1} is contained in the canonical layer, and update only the data structures for the points lying outside the layer by adding p_{i+1} otherwise.

If p_{i+1} lies outside of $b_{\ell+1}$, we get a set of new canonical layers either because $p_{i+1} \in b_m$ and the index ℓ gets increased or because $p_{i+1} \notin b_m$ and we get a new set $\mathcal{B}_{i+1}$ of m layers different from $\mathcal{B}_i$. Let C denote the set of current canonical layers and $\overline{C}$ denote the set of new canonical layers. Note that every canonical layer of $\overline{C}$ that is smaller than or equal to $b_{\ell+1}$ of C coincides with a canonical layer of C. We reuse the data structures of t- and $(k - t)$-centers, for $t = 1, \ldots, k - 1$, maintained in a canonical layer b of C if it coincides with a canonical layer b' of $\overline{C}$ as follows: b' simply reuses the data structures for t-centers of the points lying inside of b. It also takes the data structures for $(k - t)$-centers of the points lying outside of b, but updates them by adding p_{i+1}. For each canonical layer b' of $\overline{C}$ that does not coincide with a canonical layer of C, we take the data structures for t-centers of the points lying inside of layer $b_{\ell+1}$ of C. We update them by adding p_{i+1} if it is contained in b', or we construct new data structures for $(k - t)$-centers of p_{i+1} otherwise.

Computing *k*-centers. To compute the 1-center, we use a simple algorithm that takes p_1 as its center and the distance between the farthest point from p_1 as its radius. This gives a 2-approximation to the 1-center problem, and obviously, needs only $O(d)$ space. For the 2-center problem, algorithm MELP guarantees a $(2 + \varepsilon)$-approximation. Now, we can recursively build data structures that compute a $(2+\varepsilon)$-approximation to the t-center problem for $t = 3, \ldots, k$.

Now we analyze the space and update time complexity for our algorithm to the k-center problem.

Theorem 2. *Given $\varepsilon > 0$, we can compute a $(2+\varepsilon)$-approximation to the k-center problem for streaming points using $O((k+3)! \cdot 2^k d/\varepsilon)$ space and $O((k+3)! \cdot d/\varepsilon)$ update time in arbitrary dimensions d under any metric, with the assumption that it takes $O(d)$ time for computing the distance between any two points.*

Proof. Let $L(k)$ denote the size of the data structure for a k-center and let $M(k)$ denote the size of the data structure for a k-center for the case $|p_1\eta(p_1, P)| > 4kr^*$. Note that even if $|p_1\eta(p_1, P)| > 4kr^*$, we can have $|p_1\eta(p_1, P')| \leqslant 4kr^*$ for some proper subsets P' of P. We have

$$L(k) = M(k) + O(k^2 d/\varepsilon) = M(k) + ck^2 L(2) \tag{1}$$

$$M(k) = 2(k+1)\big(L(k-1) + L(k-2) + \cdots + L(2) + L(1)\big) \tag{2}$$

for some constant $c > 0$. The base cases are $L(1) = L(2) = O(d/\varepsilon)$. Now we have,

$$\begin{aligned} L(k) &= M(k) + ck^2 L(2) \\ &\leqslant 2(k+1)\big(L(k-1) + L(k-2) + \cdots + L(2) + L(1) + ckL(2)\big). \end{aligned}$$

By letting $\mathcal{L}(k) = \sum_{i=1}^{k} L(i)$, we have

$$\begin{aligned} L(k) &\leqslant 2(k+1)\big(\mathcal{L}(k-1) + ckL(2)\big) \\ &= 2(k+1)\big(L(k-1) + \mathcal{L}(k-2) + ckL(2)\big) \\ &\leqslant 2(k+1)\big(2k(\mathcal{L}(k-2) + ckL(2)) + \mathcal{L}(k-2) + ckL(2)\big) \\ &= 2(k+1)(2k+1)\big(\mathcal{L}(k-2) + ckL(2)\big) \\ &\leqslant 2(k+1)(2k+1)\big(2(k-1)(\mathcal{L}(k-3) + ckL(2)) + \mathcal{L}(k-3) + ckL(2)\big) \\ &= 2(k+1)(2k+1)(2k)\big(\mathcal{L}(k-3) + ckL(2)\big) \\ &= \cdots \\ &\leqslant 2(k+1)\left(\prod_{i=4}^{k}(2i+1)\right)\big((ck+1)L(2) + L(1)\big) \\ &\leqslant 2(k+1)\left(\prod_{i=4}^{k}2(i+1)\right)\big((ck+1)L(2) + L(1)\big) \\ &= 2(k+1)\left(2^{k-3}\frac{(k+1)!}{4!}\right)\big((ck+1)L(2) + L(1)\big) = O\big((k+3)! \cdot 2^k d/\varepsilon\big). \end{aligned}$$

Each canonical layer b maintains two sets of data structures: one set consisting of data structures for points lying inside b and another set consisting of data structures for points lying outside of b. When a new point p arrives, only one of two sets is updated depending on whether $p \in b$ or not. Since the update time is linear to the size of the data structure being updated, it takes $O((k+3)! \cdot d/\varepsilon)$. □

Remarks. The complexities of our algorithm are extremely high for large k, but it is reasonable for small k. Let us consider when $k = 3$. Our algorithm maintains at most eight data structures for the 2-center problem, eight data structures for the 1-center problem, and one data structure to solve the 3-center problem for the case $|p_1\,\eta(p_1, P)| \leqslant 4kr^*$ by Eqs. (1) and (2). On the arrival of a new point, it updates at most four data structures for the 2-center problem, four data structures for the 1-center problem, and one data structure to solve the 3-center problem for the case $|p_1\,\eta(p_1, P)| \leqslant 4kr^*$.

Our algorithm spends $O(d/\varepsilon)$ space and update time with a reasonable hidden constant in the complexity to compute the k-center problem for small constant k, while Guha's algorithm and McCutchen and Khuller's algorithm still spend $O((d/\varepsilon)\log(d/\varepsilon))$ space and update time. Compare to Guha's algorithm[18] and McCutchen and Khuller's algorithm,[22] we believe that our algorithm outperforms for small k and ε.

6. Conclusions

In this paper, we first considered the 2-center problem over a single-pass data stream. Because of the constraint that we are not allowed to see the streaming data more than once, it is not so easy to devise algorithms that guarantee a good approximation factor using as small space as possible. Nevertheless, we presented an algorithm that guarantees a $(2+\varepsilon)$-approximation using $O(d/\varepsilon)$ space and update time for arbitrary dimensions d under any metric. Then we showed that our algorithm can be extended to approximate an optimal k-center within factor $(2+\varepsilon)$ for $k > 2$.

We do not know any better lower bound than $(1+\sqrt{2})/2 \approx 1.207$ for the worst-case approximation ratio of streaming k-center algorithms using space polynomially bounded in d. We suspect that for the streaming k-center problem for $k > 1$, the lower bound can be improved. Another interesting question is whether it is possible to devise a c-approximation algorithm for $c \leqslant 2$ using space polynomially bounded by d for some constant k.

Acknowledgment

We are grateful to an anonymous referee for a careful reading of an earlier version of this paper and helping to improve the structure of the paper.

References

1. P. K. Agarwal, R. B. Avraham and M. Sharir, The 2-center problem in three dimensions, *Proc. 26th ACM Symp. Computational Geometry* (2010), pp. 87–96.
2. P. K. Agarwal, S. Har-Peled and K. R. Varadarajan, Approximating extent measures of points, *J. ACM* **51**(4) (2004) 606–635.
3. P. K. Agarwal and C. M. Procopiuc, Exact and approximation algorithms for clustering, *Algorithmica* **33** (2002) 201–226.

4. P. K. Agarwal and R. Sharathkumar, Streaming algorithms for extent problems in high dimensions, *Proc.* 21*st ACM-SIAM Symp. Discrete Algorithms* (2010), pp. 1481–1489.
5. C. C. Aggarwal, *Data Streams: Models and Algorithms* (Springer-Verlag, 2007).
6. M. Bern and D. Eppstein, Approximation algorithms for geometric problems, *Approximation Algorithms for NP-Hard Problems* (PWS Publishing Co., 1996).
7. I. A. Bonnell, M. R. Bate and S. G. Vine, The hierarchical formation of a stellar cluster, *Monthly Notices of the Royal Astronomical Society* **343**(2) (2003) 413–418.
8. T. M. Chan, More planar two-center algorithms, *Comput. Geom.* **13**(3) (1999) 189–198.
9. T. M. Chan, Faster core-set constructions and data-stream algorithms in fixed dimensions, *Comput. Geom.* **35** (2006) 20–35.
10. T. M. Chan and V. Pathak, Streaming and dynamic algorithms for minimum enclosing balls in high dimensions, *Proc.* 12*th Int. Conf. Algorithms and Data Structures* (2011), pp. 195–206.
11. M. Charikar, C. Chekuri, T. Feder and R. Motwani, Incremental clustering and dynamic information retrieval, *SIAM J. Comp.* **33**(6) (2004) 1417–1440.
12. B. Chazelle and J. Matoušek, On linear-time deterministic algorithms for optimization problems in fixed dimension, *J. Algorithms* **21** (1996) 579–597.
13. C. J. Clarke, I. A. Bonnell and L. A. Hillenbrand, The formation of stellar clusters, *Protostars and Planets IV*, eds. V. Mannings, A. P. Boss and S. S. Russell (University of Arizona Press, Tucson, 2000), pp. 151–177.
14. U. Fayyad, G. Piatetsky-Shapiro and P. Smyth, From data mining to knowledge discovery in databases, *AI Magazine* **17** (1996) 37–54.
15. D. Feder and D. H. Greene, Optimal algorithms for approximate clustering, *Proc.* 20*th Annu. ACM Symp. Theory of Computing* (1988), pp. 434–444.
16. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W.H. Freeman, New York, 1979).
17. T. F. Gonzalez, Clustering to minimize the maximum intercluster distance, *Theor. Comput. Sci.* **38** (1985) 293–306.
18. S. Guha, Tight results for clustering and summarizing data streams, *Proc.* 12*th Int. Conf. Database Theory* (ACM, 2009), pp. 268–275.
19. J. Han and M. Kamber, *Data Mining: Concepts and Techniques* (Morgan Kaufmann, 2006).
20. J. Hershberger and S. Suri, Adaptive sampling for geometric problems over data streams, *Comput. Geom.* **39**(3) (2008) 191–208.
21. H. Zarrabi-Zadeh, Core-preserving algorithms, *Proc.* 20*th Canadian Conf. Computational Geometry* (2008), pp. 159–162.
22. R. M. McCutchen and S. Khuller, Streaming algorithms for k-center clustering with outliers and with anonymity, *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, LNCS 5171 (2008), pp. 165–178.
23. M. Megiddo, On the complexity of some geometric problems in unbounded dimension, *J. Symbolic Comput.* **10** (1990) 327–334.
24. M. Megiddo and K. J. Supowit, On the complexity of some common geometric location problems, *SIAM J. Comp.* **13**(1) (1984) 182–196.
25. C. K. Poon and B. Zhu, Streaming with minimum space: An algorithm for covering by two congruent balls, *Proc.* 6*th Int. Conf. Combinatorial Optimization and Applications* (2012), pp. 269–280.
26. M. Sonka, V. Hlavac and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 3rd edn. (Thomson Learning, 2007).

27. H. Zarrabi-Zadeh, An almost space-optimal streaming algorithm for coresets in fixed dimensions, *Algorithmica* **60** (2011) 46–59.
28. H. Zarrabi-Zadeh and T. M. Chan, A simple streaming algorithm for minimum enclosing balls, *Proc.* 18*th Canadian Conf. Computational Geometry* (2006), pp. 139–142.