A Streaming Algorithm for 2-Center With Outliers in High Dimensions

Behnam Hatami* Hamid Zarrabi-Zadeh†

Abstract

We study the 2-center problem with outliers in high-dimensional data streams. Given a stream of points in arbitrary d dimensions, the goal is to find two congruent balls of minimum radius covering all but at most z points. We present a $(1.8 + \varepsilon)$-approximation streaming algorithm, improving over the previous $(4+\varepsilon)$-approximation algorithm available for the problem. The space complexity and update time of our algorithm are $\text{poly}(d, z, 1/\varepsilon)$, independent of the size of the stream.

1 Introduction

The k-center problem—covering a set of points using k congruent balls of minimum radius—is a fundamental problem, arising in many applications such as data mining, machine learning, statistics, and image processing. In real-world applications where input data is often noisy, it is very important to consider outliers, as even a small number of outliers can greatly affect the quality of the solution. In particular, the k-center problem is very sensitive to outliers, and even a constant number of outliers can increase the radius of the k-center unboundedly. Therefore, it is natural to consider the following generalization of the the k-center problem: given a set P of n points in arbitrary d dimensions and a bound z on the number of outliers, find k congruent balls of minimum radius to cover at least $n-z$ points of P. See Figure 1 for an example. In this paper, we focus on the *data stream* model of computation where only a single pass over the input is allowed, and we have only a limited amount of working space available. This model is in particular useful for processing massive data sets, as it does not require the entire data set to be stored in memory.

The Euclidean k-center problem has been extensively studied in the literature. If k is part of the input, the problem in known to be NP-hard in two and more dimensions [10], and is even hard to approximate to within a factor better than 1.82, unless $\text{P} = \text{NP}$ [9]. Factor-2 approximation algorithms are available for the problem in any dimension [9, 11]. For small k and d, better solutions are available. The 1-center problem in fixed dimensions is known to

*Department of Computer Engineering, Sharif University of Technology, Tehran 14588-89694, Iran. Email: `bhatami@ce.sharif.edu`.

†Department of Computer Engineering, Sharif University of Technology, Tehran 14588-89694, Iran. Email: `zarrabi@sharif.edu`.

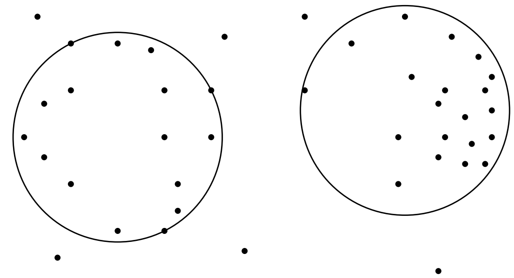

Figure 1: An example of 2-center with 6 outliers.

be LP-type and can be solved in $O(n)$ time [7]. For 2-center in the plane, the current best algorithm runs in $O(n \log^2 n \log^2 \log n)$ time [4].

For k-center with outliers, Charikar *et al.* [6] gave the first algorithm with an approximation factor of 3, which works in any dimension. Better results are known for small k in the plane. The 1-center problem with z outliers in the plane can be solved in $O(n \log n + z^3 n^{\varepsilon})$ time, for any $\varepsilon > 0$, using Matoušek's framework [14]. Agarwal [1] gave a randomized $O(nz^7 \log^3 z)$-time algorithm for 2-center with z outliers in the plane.

In the streaming model, where only a single pass over the input is allowed, McCutchen and Khuller [15] and independently Guha [12] presented algorithms to maintain a $(2+\varepsilon)$-approximation to k-center in any dimension using $O((kd/\varepsilon)\log(1/\varepsilon))$ space. For $k = 1$, Zarrabi-Zadeh and Chan [17] presented a simple algorithm achieving an approximation factor of 3/2 using only $O(d)$ space. Agarwal and Sharathkumar [2] improved the approximation factor to $(1+\sqrt{3})/2 + \varepsilon \approx 1.37$ using $O((d/\varepsilon^3)\log(1/\varepsilon))$ space. The approximation factor of their algorithm was later improved to 1.22 by Chan and Pathak [5]. For $k = 2$, Kim and Ahn [13] have recently obtained a $(1.8+\varepsilon)$-approximation using $O(d/\varepsilon)$ space. Their algorithm extends to any fixed k, with the same approximation factor.

For k-center with z outliers in the streaming model, McCutchen and Khuller [15] gave a $(4+\varepsilon)$-approximation algorithm using $O(\frac{zk}{\varepsilon})$ space. When dimension is fixed, a $(1+\varepsilon)$-approximation to 1-center with outliers can be maintained in $O(z/\varepsilon^{((d-1)/2)})$ space using the notion of robust ε-kernels [3,16]. For 1-center with outliers in high dimensions, Zarrabi-Zadeh and Mukhopadhyay [18] gave a $(\sqrt{2}\alpha)$-approximation, where α is the approximation factor of the underlying algorithm for maintaining 1-center. Combined with the 1.22-approximation algorithm of Chan and Pathak [5], it yields an approximation factor of $(\sqrt{2} \times 1.22) \approx 1.73$ using $O(d^3 z)$ space.

Our result In this paper, we study the 2-center problem with outliers in high dimensional data streams. We present a streaming algorithm that achieves an approximation factor of $1.8+\varepsilon$, for any $\varepsilon > 0$, using $\text{poly}(d, z, \frac{1}{\varepsilon})$ space and update time. This improves over the previous $(4+\varepsilon)$-approximation streaming algorithm available for the problem presented by McCutchen and Khuller [15]. The approximation factor of our algorithm matches that of the best streaming algorithm for the 2-center problem with no outliers. This is somewhat surprising, considering that the current best approximation factors for streaming k-center with and without outliers differ by a multiplicative factor of $\sqrt{2}$ for $k = 1$ [5, 18], and by a factor of 2 for general k [12, 15]. See Table 1 for a comparison.

To obtain our result, we have used a combination of several ideas including parallelization,

Problem	Approximation Factor	
	Without Outliers	With Outliers
1-center	1.22 [5]	1.73 [18]
2-center	$1.8 + \varepsilon$ [13]	$1.8 + \varepsilon$ [Here]
k-center	$2 + \varepsilon$ [12, 15]	$4 + \varepsilon$ [15]

Table 1: Summary of the streaming algorithms for k-center with and without outliers in high dimensions.

far/close ball separation, centerpoint theorem, and keeping lower/upper bounds on the radius and distance of the optimal balls. We have also employed ideas of [13] for the 2-center problem with no outliers. However, our problem is much harder here, as we not only need to find balls of minimum radius, but we also need to decide which subset of points to cluster. This is in particular more challenging in the streaming model, where we only have a single pass over the input, and we must decide on the fly which point is an outlier, and which one can be safely ignored as a non-outlier point, to comply with the working space restriction enforced by the model.

2 Preliminaries

Let $B(c, r)$ denote a ball of radius r centered at c. We use $r(B)$ to denote the radius of a ball B. For two points p and q, the distance between p and q is denoted by $\|pq\|$. Given two balls $B(c, r)$ and $B'(c', r')$, we define $\delta(B, B') \equiv \max\{0, \|cc'\| - r - r'\}$ to be the *distance* between B and B'. Two balls B_1 and B_2 are said to be *α-separated*, if $\delta(B_1, B_2) > \alpha \cdot \max\{r(B_1), r(B_2)\}$.

Given an n-point set P in d-dimensions, a point $c \in \mathbb{R}^d$ is called a *centerpoint* of P, if any halfspace containing c contains at least $\lceil n/(d+1) \rceil$ points of P. It is well-known that any finite set of points in d dimensions has a centerpoint [8]. The following observation is a corollary of this fact.

Observation 1. *Given a set P of $k(d+1)$ points in d dimensions, the centerpoint of P has the property that any convex object not covering the centerpoint, leaves at least k points of P uncovered.*

Given a point set P, the *k-furthest point from $p \in P$* is a point whose distance to p is the k-th largest among all points in P. We assume the standard word-RAM model of computation. Each coordinate value takes a unit of space. Thus, a d-dimensional point takes $O(d)$ space, and basic operations on the points take $O(d)$ time.

3 A Simple Algorithm for 1-Center with Outliers

To warm up, we present a simple 2-approximation streaming algorithm for the 1-center problem with outliers. It utilizes a parallelization technique [15], which will be used extensively during the rest of the paper. The pseudo-code is provided in Algorithm 1. The algorithm

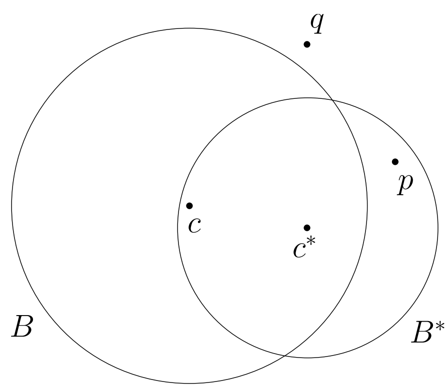

Figure 2: Illustrating the proof of Theorem 1

receives as input a stream of points, P, and the number of outliers, z. It assumes that the first point p_1 of the stream is non-outlier. We will show later how to remove this assumption. The algorithm returns a ball B covering all but at most z points of P.

Algorithm 1 1-CENTER(P, z)

$c \leftarrow$ the first point in P
$B \leftarrow B(c, 0)$
$Q \leftarrow \emptyset$ $\quad \triangleright$ Q represents the buffer
for each p in P **do**
    if $p \notin B$ **then**
        insert p into Q
        if $|Q| = z + 1$ **then**
            $q \leftarrow$ closest point to c in Q
            remove q from Q
            $B \leftarrow B(c, \|cq\|)$
return B

Theorem 1. *Algorithm 1 computes a 2-approximation to the 1-center problem with z outliers, assuming that the first point of the stream is not outlier.*

Proof. Let $B^*(c^*, r^*)$ be the optimal solution, and let c be the first point of the stream, which is assumed to be a non-outlier in the optimal solution. Since c is covered by B^*, for all points $p \in B^*$, we have $\|cp\| \leq \|cc^*\| + \|c^*p\| \leq 2r^*$. Among the $z + 1$ points furthest from c, there is at least one point p which is not outlier, and therefore, it is contained in B^* (see Figure 2). Thus, by our choice of q in the algorithm, we have $\|cq\| \leq \|cp\| \leq 2r^*$, and hence, the ball $B(c, \|cq\|)$ returned by Algorithm 1 is a 2-approximation. $\square$

Algorithm 1 assumes that the first point of the stream is not outlier. To remove this assumption, we run $z+1$ instances of Algorithm 1 in parallel, each of which is given as input one of the first $z + 1$ points of the stream, followed by the rest of the points. Clearly, there exists a point among the first $z + 1$ points of P which is not an outlier in the optimal solution. Therefore, the smallest ball among the $z + 1$ balls computed in parallel is always within factor 2 of the optimal solution. The space complexity of Algorithm 1 for one instance is $O(zd)$, and its update time is $O(d + \log z)$, considering that we can maintain distances of the points in Q to c using a heap. Overall, we get the following.

Theorem 2. *Given a stream of points in d dimensions, we can maintain a 2-approximation to 1-center with z outliers in $O(z^2 d)$ space and $O(zd + z \log z)$ update time.*

4 The 2-Center Problem With Outliers

In this section, we provide a $(1.8 + \varepsilon)$-approximation algorithm for the 2-center problem with outliers. In all algorithms presented in this section, we assume that the first point of the stream, p_1, is non-outlier. This assumption can be easily removed by considering $z + 1$ parallel instances of the algorithm, similar to what we did in Section 3.

Let B_1^* and B_2^* be the balls in an optimal solution to 2-center with z outliers on a point set P. We denote by r^* the optimal radius, and by δ^* the distance between B_1^* and B_2^*. Moreover, we assume, w.l.o.g., that p_1 is in B_1^*.

To prove our main result, we distinguish between two cases. The first case is when $\delta^* > \alpha r^*$, for some constant α to be fixed later. (It will turn out that $\alpha = 12$ is a proper choice.) The second case is when $\delta^* \leq \alpha r^*$, for the same value of α. The geometric insight behind breaking up into these two cases is as follows. When $\delta^* \leq \alpha r^*$, the 1-center with outliers is a good approximation to 2-center with outliers (as will be shown in Lemma 12), and therefore, we can use Algorithm 1 to approximate the optimal solution in this case. On the other hand, when $\delta^* > \alpha r^*$, we know that the optimal balls are well-separated, and hence, we can separate points into distinct areas based on this assumption. We present the details of our algorithms for handling these two cases in the rest of this section.

4.1 The Case $\delta^* > \alpha r^*$

Here, we present a 1.8-approximation algorithm for the case where optimal balls are separated by a distance greater than αr^*. We start with two simple observations.

Observation 2. *Let B_1 and B_2 be two congruent balls of radius r, with distance $\delta > \alpha r$. For any two points $p \in B_1$ and $q \in B_2$, we have $1 \leq \frac{\|pq\|}{\delta} < \frac{\alpha+4}{\alpha}$.*

Proof. The distance between p and q is at most $\delta + 4r$. Hence, $\frac{\|pq\|}{\delta} \leq 1 + \frac{4r}{\delta} < 1 + \frac{4}{\alpha}$. □

Observation 3. *Let B_1 and B_2 be two disjoint balls at distance δ, and let B be an arbitrary ball of radius less than $\frac{\delta}{2}$. Then B intersects at most one of B_1 and B_2.*

We next prove some properties regarding the optimal balls, B_1^* and B_2^*.

Lemma 3. *Let B_1^* and B_2^* be α-separated. Then for any two points $p \in B_1^*$ and $q \in B_2^*$, we have $2r^* < \frac{2}{\alpha}\|pq\|$.*

Proof. By Observation 2, $1 \leq \frac{\|pq\|}{\delta^*} < \frac{\|pq\|}{\alpha r^*}$, and as a result, $2r^* < \frac{2}{\alpha}\|pq\|$. □

The following lemma shows that if B_1^* and B_2^* are α-separated, then a point of B_2^* can be found by only considering $z+1$ points furthest from the first point.

Lemma 4. *Let B_1^* and B_2^* be α-separated, with $\alpha \geq 4$. If p is a point in B_1^*, and S is a $(z+1)$-subset of P furthest from p, then $S \cap B_2^*$ is non-empty.*

Proof. Suppose by way of contradiction that $S \cap B_2^*$ is empty. Since $|S| = z+1$, there is at least one point in S which is not outlier, and hence, it is in B_1^*. Let s be a point in $S \cap B_1^*$ furthest from p. Consider the ball $B = B(p, \|ps\|)$. For any point $q \in P \setminus S$, we have $\|pq\| \leq \|ps\|$, because $s \in S$ and $q \notin S$. Therefore, B covers $P \setminus S$. Since $p, s \in B_1^*$, $\|ps\|$ is at most $2r^*$. Thus, by Observation 3, $B_2^* \cap B = \emptyset$. Therefore, $B_2^* \cap P = \emptyset$, and hence, B_2^* is empty, which contradicts the optimality of the solution. □

Lemma 5. *Let p be a point in B_1^*, and q be the $(z+1)$-furthest point from p. Then, $\delta^* > \frac{\alpha}{\alpha+4}\|pq\|$.*

Proof. By Lemma 4, there exists a point $s \in B_2^*$ such that $\|ps\| \geq \|pq\|$. Thus, by Observation 2, $\frac{\|pq\|}{\delta^*} \leq \frac{\|ps\|}{\delta^*} < \frac{\alpha+4}{\alpha}$. □

Lemma 6. *If $p \in B_1^*(c_1, r^*)$ and $q \in B_2^*(c_2, r^*)$, then $B_1^* \subseteq B(p, 2r^*)$ and $B_2^* \subseteq B(q, 2r^*)$, and hence, at most z points of P lie outside $B(p, 2r^*) \cup B(q, 2r^*)$.*

Proof. For any arbitrary point $s \in B_1^*$, $\|ps\| \leq \|pc_1\| + \|c_1 s\| \leq 2r^*$, and therefore, $B_1^* \subseteq B(p, 2r^*)$. Similarly, we have $B_2^* \subseteq B(q, 2r^*)$. Considering that at most z points of P are outliers, the proof is complete. □

Lemma 7. *Let S be a subset of P of size at least $(d+1)(z+1)$, enclosed by a ball B of radius less than $\delta^*/2$. Then S intersects exactly one of B_1^* and B_2^*, and the centerpoint of S lies inside either B_1^* or B_2^*.*

Proof. Not all points in S are outliers, because $(d+1)(z+1) > z$. Therefore, B intersects at least one of B_1^* and B_2^*. Observation 3 implies that B intersects exactly one of B_1^* and B_2^*. Assume, w.l.o.g., that B intersect B_1^*. Now, by Observation 1, if the centerpoint of S is not in B_1^*, then $z+1$ points of S remain uncovered by B_1^*, contradicting the fact that there are at most z outliers. □

The Algorithm We now describe our algorithm for handling the case $\delta^* > \alpha r^*$. At any point of time, our algorithm maintains a partition of P into three disjoint subsets B_1, B_2, and Buffer. The first point p_1 is assumed, w.l.o.g, to be in B_1^*. (We have already assumed that p_1 is not outlier.) The algorithm tries to partition points in such a way that at the end, B_1 contains the whole B_1^*, and B_2 contains the whole B_2^*, with possibly some outliers being contained in B_1 and B_2. The algorithm sets $c_1 = p_1$ as the fixed center of B_1, and picks c_2 among the points processed so far as a candidate for being the center of B_2. Moreover, the algorithm maintains two values δ and r, where at any time, δ is a lower bound of δ^*, and r is an upper bound of $2r^*$ (under a certain condition).

Our algorithm is presented in Algorithm 2. For each input point $p \in P$, the algorithm first tries to add p to either B_1 or B_2, using functions AddToB_1 and AddToB_2, respectively.

Algorithm 2 2-CENTER-FIRST-CASE(P)

1: $r \leftarrow 0$, $\delta \leftarrow 0$
2: $c_1 \leftarrow p_1$
3: **for each** $p \in P$ **do**
4:     **if** ADDTOB$_1(p)$ = false **and** ADDTOB$_2(p)$ = false **then**
5:         add p to Buffer
6:         **while** $|\text{Buffer}| > z$ **do**
7:             **if** $|B_2| \geq (d+1)(z+1)$ **then**
8:                 $B_1 \leftarrow B_1 \cup B_2$, $B_2 \leftarrow \emptyset$
9:             **else if** c_2 is set **then**
10:                 $B_1 \leftarrow B_1 \cup \{c_2\}$
11:             $T \leftarrow \text{Buffer} \cup B_2 \setminus \{c_2\}$
12:             $B_2 \leftarrow \emptyset$
13:             $c_2 \leftarrow$ (z+1)-furthest point from c_1 in T
14:             $r \leftarrow \frac{2}{\alpha}\|c_1 c_2\|$
15:             Buffer $\leftarrow \emptyset$
16:             **for** $q \in T$ **do**
17:                 **if** ADDTOB$_1(q)$ = false **and** ADDTOB$_2(q)$ = false **then**
18:                     add q to Buffer

If none of them fits, the point is added to Buffer. The function ADDTOB$_1$ adds a point p to B_1 only if it is within distance δ of the center c_1. Similarly, ADDTOB$_2$ adds a point p to B_2 only if it is within r-radius of c_2. The two functions also update the values of δ and r whenever necessary, to maintain the invariants to be defined in Lemma 8.

Algorithm 3 ADDTOB$_1(p)$

if at least $z+1$ points have been processed so far **then**
    $q \leftarrow (z+1)$-furthest point from c_1
else
    $q \leftarrow c_1$
$\delta \leftarrow \frac{\alpha}{\alpha+4}\|c_1 q\|$
if $p \in B(c_1, \delta)$ **then**
    $B_1 \leftarrow B_1 \cup \{p\}$
    return true
return false

Whenever the buffer overflows (in line 6 of Algorithm 2), the algorithm takes one of the following actions depending on the size of B_2. If $|B_2| \geq (d+1)(z+1)$, then the points of B_2 are moved to B_1, and B_2 is reset. Otherwise, the old c_2 (if already set) is moved to B_1, and another point from $T = B_2 \cup \text{Buffer} \setminus \{c_2\}$ is picked as c_2. The while loop iterates at most $O(dz)$ times, because after the first iteration, we are sure that T has at most $(d+1)(z+1)+z$ points, from which one point (i.e., c_2) is removed at each subsequent iteration.

For the sake of analysis, we maintain a "central point", denoted by c_p, which is defined as follows: if $|B_2| < (d+1)(z+1)$, then $c_p = c_2$, otherwise, c_p is the centerpoint of the first

Algorithm 4 ADDTOB$_2$(p)

if c_2 is set **and** $p \in B(c_2, r)$ **then**
  $B_2 \leftarrow B_2 \cup \{p\}$
  if $|B_2| = (d+1)(z+1)$ **then**
    $r \leftarrow (2 + \frac{2}{\alpha}) \times r$
    for q in Buffer **do**
      if $q \in B(c_2, r)$ **then**
        $B_2 \leftarrow B_2 \cup \{q\}$
        remove q from Buffer
  return true
return false

$(d+1)(z+1)$ points currently in B_2. It is clear by our definition that c_p is always inside B_2.

Lemma 8. *The following invariants are maintained during the execution of the algorithm:*

(a) $\delta < \delta^*$

(b) $r \leq \delta/2$

(c) $B_1 \cap B_2^* = \emptyset$

(d) if $c_p \in B_2^*$*, then*

1. $2r^* < r$
2. $B_2 \cap B_1^* = \emptyset$
3. *all points in Buffer are outliers.*

Proof. Invariant (a): At the beginning, $\delta = 0$, which clearly satisfies the invariant. After $z+1$ points of the stream is processed, function ADDTOB$_1$ starts updating δ to $\frac{\alpha}{\alpha+4}\|c_1 q\|$, where q is the $(z+1)$-furthest point from c_1 in the current stream. Now, since $c_1 \in B_1^*$, Lemma 5 implies that $\delta < \delta^*$.

Invariant (b): When c_2 is set by Algorithm 2, it is the $(z+1)$-furthest point from c_1 in a set $T \subseteq P$, and r is set to $\frac{2}{\alpha}\|c_1 c_2\|$. Let q be the $(z+1)$-furthest point from c_1 in the stream at that moment. Then $\|c_1 c_2\| \leq \|c_1 q\|$. For $\alpha \geq 12$, we have $(2 + \frac{2}{\alpha})(\frac{2}{\alpha}) < \frac{\alpha}{2(\alpha+4)}$. Therefore,

$$r \leq \left(2 + \frac{2}{\alpha}\right)\frac{2}{\alpha}\|c_1 c_2\| < \frac{\alpha}{2(\alpha+4)}\|c_1 q\| = \delta/2,$$

which implies that the invariant holds, even after increasing r by function ADDTOB$_2$.

Invariant (c): We first claim that if c_2 is set, then $B(c_p, \frac{2}{\alpha}\|c_1 c_p\|) \subseteq B_2(c_2, r)$. If $|B_2| < (d+1)(z+1)$, then $c_p = c_2$ and $r = \frac{2}{\alpha}\|c_1 c_2\|$, and therefore, $B_2 = B(c_p, \frac{2}{\alpha}\|c_1 c_p\|)$. When the size of B_2 reaches $(d+1)(z+1)$, the central point c_p moves to the centerpoint of B_2, and r is increased by a factor of $(2 + \frac{2}{\alpha})$. Because the centerpoint of B_2 lies in B_2, we have $c_p \in B(c_2, \frac{2}{\alpha}\|c_1 c_2\|)$. Therefore, $\|c_2 c_p\| \leq \frac{2}{\alpha}\|c_1 c_2\|$, and hence

$$\|c_1 c_p\| \leq \|c_1 c_2\| + \|c_2 c_p\| \leq (1 + \frac{2}{\alpha})\|c_1 c_2\|. \tag{1}$$

Now, we have

$$\begin{aligned}
B(c_p, \frac{2}{\alpha}\|c_1c_p\|) &\subseteq B(c_p, (1+\frac{2}{\alpha})\frac{2}{\alpha}\|c_1c_2\|) \\
&\subseteq B(c_2, \|c_2c_p\| + (1+\frac{2}{\alpha})\frac{2}{\alpha}\|c_1c_2\|) \\
&\subseteq B(c_2, (2+\frac{2}{\alpha})\frac{2}{\alpha}\|c_1c_2\|) = B_2(c_2, r),
\end{aligned}$$

which completes the proof of the claim.

Now, we prove invariant (c). A point p can be added to B_1 in two cases. The first case is in function AddToB_1, where the point is added to B_1 only if it is within distance δ of the center c_1, which by invariant (a), guaranties $\|pc_1\| < \delta^*$. Therefore, $p \notin B_2^*$ in this case.

The second case occurs in Algorithm 2, when the buffer overflows and B_2 is non-empty. The algorithm takes one of the following actions depending on the size of B_2. If $|B_2| < (d+1)(z+1)$, then $c_p = c_2$, and the algorithm adds c_2 to B_1. Suppose by way of contradiction that $c_2 \in B_2^*$. By Lemma 3 and invariant (b), $2r^* < \frac{2}{\alpha}\|c_1c_2\| = r \le \delta/2 \le \delta$. Therefore, by Lemma 6, there must be at most z points outside $B_1(c_1, \delta)$ and $B_2(c_2, r)$, which contradicts the overflow of the buffer. If $|B_2| \ge (d+1)(z+1)$, then c_p is the centerpoint of the first $(d+1)(z+1)$ points currently in B_2. In this case, we add all points of B_2 to B_1. By invariant (b), $r \le \delta/2 < \delta^*/2$. Therefore, by Lemma 7, B_2 intersects exactly one of B_1^* and B_2^*, and therefore we have either $c_p \in B_1^*$ or $c_p \in B_2^*$. Suppose by way of contradiction that $c_p \in B_2^*$. In this case, by the claim that we proved earlier, B_2 covers $B(c_p, \frac{2}{\alpha}\|c_1c_p\|)$. Moreover, by Lemma 3 and invariant (b), $2r^* < r \le \delta/2 \le \delta$. Therefore, by Lemma 6, there must be at most z points outside $B_1(c_1, \delta)$ and $B(c_p, \frac{2}{\alpha}\|c_1c_p\|) \subseteq B_2(c_2, r)$, which contradicts the overflow of the buffer.

Invariant (d-1): By Observation 2, if $c_1 \in B_1^*$ and $c_p \in B_2^*$, then $1 \le \frac{\|c_1c_p\|}{\delta^*} < \frac{\|c_1c_p\|}{\alpha r^*}$, and as a result, $2r^* < \frac{2}{\alpha}\|c_1c_p\|$. If $|B_2| < (d+1)(z+1)$, then $c_p = c_2$, and by Algorithm 2, $r = \frac{2}{\alpha}\|c_1c_2\|$, and therefore, $2r^* < r$. If $|B_2| \ge (d+1)(z+1)$, then by inequality (1),

$$2r^* < \frac{2}{\alpha}\|c_1c_p\| \le (1+\frac{2}{\alpha})\frac{2}{\alpha}\|c_1c_2\| < (2+\frac{2}{\alpha})\frac{2}{\alpha}\|c_1c_2\| = r.$$

Invariant (d-2): We know that $c_p \in B_2$. If $c_p \in B_2^*$, then by invariants (a) and (d-1), $2r^* < r \le \delta/2 < \delta^*/2$. Now, by Observation 3, B_2 intersects only B_2^*, and hence, $B_2 \cap B_1^* = \emptyset$.

Invariant (d-3): By invariants (b) and (d-1), $2r^* < r \le \delta/2 \le \delta$. Therefore, by Lemma 6, all points outside $B_1(c_1, \delta)$ and $B_2(c_2, r)$ are outliers. $\square$

Answering Queries We now describe how the information maintained by Algorithm 2 can be used to answer queries of the following form: find two congruent balls of minimum radius to cover all but at most z points of the stream processed so far.

We call a partition of P into subsets B_1, B_2, and Buffer a *proper partition*, if B_1 completely contains B_1^*, B_2 completely contains B_2^*, and all points in Buffer are outliers. The key point here is that if we have an algorithm for the 1-center problem with outliers, then given a proper partition of P into B_1, B_2, and Buffer, we can find an optimal solution to 2-center with z outliers on P. Algorithm 6 describes how to find such a solution. We know that all points in Buffer are outliers. Therefore, there are $z - |\text{Buffer}|$ outliers in $B_1 \cup B_2$. However,

Algorithm 5 QUERY

1: solutions $\leftarrow \{\text{MinCover}(B_1, B_2, \text{Buffer})\}$
2: candidates $\leftarrow B_2 \cup$ Buffer
3: **if** $|B_2| \geq (d+1)(z+1)$ **then**
4: candidates $\leftarrow$ Buffer
5: $B_1 \leftarrow B_1 \cup B_2$
6: $B_2 \leftarrow \emptyset$
7: **for** $c \in$ candidates **do**
8: B_1', B_2', Buffer$'$ = PARTITION(c, candidates, B_1)
9: add MINCOVER(B_1', B_2', Buffer$'$) to solutions
10: **return** min {solutions}

Algorithm 6 MINCOVER$(B_1, B_2, \text{Buffer})$

solutions $\leftarrow \{\}$
for $k \leftarrow 0, \ldots, (z - |\text{Buffer}|)$ **do**
$r_1 \leftarrow$ 1-CENTER(B_1, k)
$r_2 \leftarrow$ 1-CENTER$(B_2, z - |\text{Buffer}| - k)$
add $\max\{r_1, r_2\}$ to solutions
return min {solutions}

we do not know how many outliers are exactly in each of B_1 and B_2. To overcome this issue, we try all possible combinations of k outliers in B_1 and $z - |\text{Buffer}| - k$ outliers in B_1, for $0 \leq k \leq z - |\text{Buffer}|$, and return the one with the minimum radius. Clearly, one of the combinations explored corresponds to an optimal solution, and therefore, the output of Algorithm 6 is optimal.

Now, we describe our query algorithm presented in Algorithm 5. There are two cases in the algorithm. If $c_p \in B_2^*$, then by invariants (b) and (d), the current sets B_1, B_2, and Buffer maintained by Algorithm 2 form a proper partition, and hence, the computed solution in line 1 is optimal. Otherwise, if $c_p \notin B_2^*$, then invariant (d) cannot be used. However, a crucial fact here is that if we know a point $p \in B_1^*$ and a point $q \in B_2^*$, then a proper partition can be computed (using function PARTITION to be described in Algorithm 7). We already know that $c_1 \in B_1^*$. Therefore, it only remains to find a point $c \in B_2^*$. To find such a point, we simply check all possible candidate points. By invariant (c), $B_1 \cap B_2^* = \emptyset$. Therefore, there exists a point in $(B_2 \cup \text{Buffer}) \cap B_2^*$, and hence, we only need to consider the points in $B_2 \cup$ Buffer as candidates for c. However, the size of B_2 may be very large. The next lemma shows that if $|B_2| \geq (d+1)(z+1)$, then Buffer $\cap B_2^* \neq \emptyset$, and therefore, we can only consider the points in Buffer as candidates for c in this case.

Lemma 9. *At any time, if* $|B_2| \geq (d+1)(z+1)$ *and* $c_p \notin B_2^*$*, then* $B_2 \cap B_2^* = \emptyset$.

Proof. By invariants (a) and (b), we know that $r \leq \delta/2 < \delta^*/2$. By Lemma 7, $c_p \in B_1^* \cup B_2^*$. Since $c_p \notin B_2^*$, we have $c_p \in B_1^*$. On the other hand, by Observation 3, B_2 intersects at most one of B_1^* and B_2^*. Therefore, $B_2 \cap B_2^* = \emptyset$. □

Our partitioning algorithm (Algorithm 7) works as follows. For the current candidate point c,

Algorithm 7 PARTITION(c, S, B_1)

$r \leftarrow \frac{2}{\alpha}\|c_1c\|$
$\delta \leftarrow \max\{\delta, r\}$
$B'_1 \leftarrow B_1$, $B'_2 \leftarrow \emptyset$, Buffer$' \leftarrow \emptyset$
for $p \in S$ **do**
  if ADDTOB$'_1(p)$ = false **and** ADDTOB$'_2(p)$ = false **then**
    add p to Buffer$'$
return B'_1, B'_2, Buffer$'$

the algorithm constructs $B'_1(c_1, \max\{\delta, \frac{2}{\alpha}\|c_1c\|\})$ and $B'_2(c, \frac{2}{\alpha}\|c_1c\|)$. We know that $B_1 \subseteq B'_1$, and hence, we only need to see which points in Buffer $\cap B_2$ are inside B'_1. Algorithm 7 uses functions ADDTOB$'_1$ and ADDTOB$'_2$ for adding points to B'_1 and B'_2, respectively. These functions are the same as ADDTOB$_1$ and ADDTOB$_2$, with the only exception that they add points to B'_i instead of B_i, for $i = 1, 2$. Note that all variables in Algorithms 5 and 7, including B_1, B_2, r, and δ are local variables, and changing them will not effect their value in the main algorithm.

If $c \in B^*_2$, then by Lemma 3, $2r^* \leq \frac{2}{\alpha}\|c_1c\|$. Therefore, by Lemma 6, $B^*_1 \subseteq B'_1$ and $B^*_2 \subseteq B'_2$. On the other hand, since the distance of the new points added to B'_1 is less than $\|c_1p\| \leq \max\{\delta, \frac{2}{\alpha}\|c_1c^*\|\}$, we have by invariant (c) that $B'_1 \cap B^*_2 = \emptyset$. As a result, B'_1 (resp., B'_2) completely covers B^*_1 (resp., B^*_2), and the points in Buffer are all outliers. Therefore, if $c \in B^*_2$, the algorithm finds a proper partition. The following theorem summarizes the result of this section.

Theorem 10. *If $\delta^* > \alpha r^*$, a 1.8-approximation to 2-center with z outliers can be maintained in $O(d^3z^2)$ space and* poly(d, z) *update/query time.*

Proof. Our algorithm for answering queries (Algorithm 5) considers all valid candidates for c, and therefore, for at least one of them the partition obtained in proper. In the streaming model, we cannot afford keeping all the points of B_1 and B_2. Therefore, in both Algorithms 2 and 5, we maintain the sets B_1 and B_2 in a data structure that supports adding points, and gives a β-approximation to 1-center with k outliers, for $k = 0, \ldots, z$. Moreover, we maintain a set $B_u = B_1 \cup B_2$ in a similar data structure. Note that these data structures do not need to maintain all the points. They only need to have a buffer of size $(d+1)(z+1)$ to keep the most recently added points, because we access points in B_2 only if its size is less than $(d+1)(z+1)$.

To maintain B_1, B_2, and B_u, we use the streaming algorithm of [5,18], which provides an approximation factor of $1.22 \times \sqrt{2} < 1.8$. The algorithm uses $O(d^3z)$ space and has poly(d, z) update time. Since we need to run $z + 1$ instances of Algorithm 2 in parallel, the space and update time are multiplied by a factor of z. □

4.2 The Case $\delta^* \leq \alpha r^*$

Our idea in this section is to carefully adapt the algorithm of Kim and Ahn [13], originally designed for maintaining an approximate 2-center. To avoid duplication, we just sketch the main steps of their algorithm, and explain our modifications to it. Kim and Ahn's algorithm,

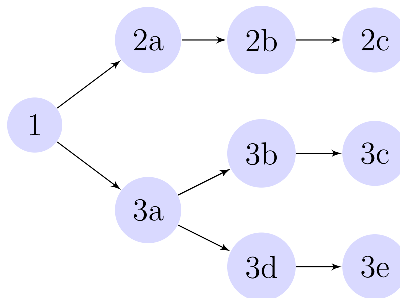

Figure 3: State diagram of the KA algorithm. Labels are taken from [13].

Algorithm 8 2-Center-Second-Case(P, z, r)

solutions $\leftarrow \{\}$
for each (n_1, n_2, n_3, n_4) such that $\sum n_i = z$ **do**
    for each $\pi \in \{1, 2, 3\}$ **do**
        counter$_i \leftarrow 0$, for $i = 1, \dots, 4$
        $B_1 \leftarrow B(p_1, r)$, $B_2 \leftarrow \emptyset$
        $j \leftarrow 1$ $\triangleright$ j represents current level
        for each $p \in P$ **do**
            if $p \notin B_1 \cup B_2$ **then**
                counter$_j \leftarrow$ counter$_j + 1$
                if counter$_j > n_j$ **then**
                    $j \leftarrow j + 1$
                    if $j > 4$ **then** exit the inner for loop
                    $(B_1, B_2) \leftarrow$ KA.insert(p, π)
        if $j \leq 4$ **then**
            add $\max \{r(B_1), r(B_2)\}$ to solutions
return $\min \{$solutions$\}$

which we refer to as the KA algorithm, has 9 different states, shown in Figure 3. Depending on the points arrived so far, the algorithm is in one of these states. In each state, the algorithm keeps at most two balls as a candidate solution. A transition between the states occurs whenever a point not covered by any of the two balls arrives.

The algorithm starts at node 1, and proceeds through the transition graph as points arrive. In some states, there is more than one state to follow, and the algorithm has no prior information which one is the correct choice. However, there are only three different paths to follow in the transition graph. Hence, we can easily run three instances of the algorithm in parallel, each of which follows one of the paths deterministically, to make sure that at any time, at least one of the instances is in a correct state.

Our modification is on the transition part. Points that are covered by the current solution can be safely ignored, as they do not cause any change in the current solution, and hence, they cause no transition. Only those points that lie outside the current solution are candidates for being outliers. Since the number of outliers in each state is unknown, we try all possible choices. The observation here is that the transition graph is a DAG of depth four. If n_i

($1 \leq i \leq 4$) represents the number of outliers in depth i, then it suffices to consider all tuples $(n_1, \ldots, n_4)$ such that $\sum_{i=1}^{4} n_i = z$. It is easy to verify that there are $O(z^3)$ such tuples.

The pseudocode of our algorithm is presented in Algorithm 8. For each possible choices of n_1 to n_4, and each of the three paths in the transition graph, numbered from 1 to 3, the algorithm keeps a candidate solution (B_1, B_2) to the 2-center of non-outlier points, a parameter j representing the current level in the transition graph, and four counters to keep track of the number of outliers seen so far at each level.

The algorithm starts with $B_1 = B(p_1, r)$ and $B_2 = \emptyset$, which corresponds to Case 1 of the KA algorithm. (The value r is given as input to the algorithm, satisfying $r \geq 1.2r^*$.) For each new point p, we first check if it is contained in the current solution. If so, then B_1 and B_2 are valid solutions so far, and we proceed to the next point. Otherwise, if the number of outliers seen in the current level has not yet reached n_j, we consider p as an outlier and proceed. Otherwise, we go to the next level, and update the current candidate solution, (B_1, B_2), using the KA algorithm. We give the transition path π along with the point p to the KA algorithm to help it deterministically decide which state to choose as the next one.

After all points in P are processed, if we are in one of the four states in the current path, then the obtained solution is added to the feasible solutions. Otherwise, the solution is not feasible, and is abandoned as in the KA algorithm. Finally, we return the best solution among all computed feasible solutions. Kim and Ahn [13] proved that in all feasible solutions computed this way, the larger ball among B_1 and B_2 has radius at most $3/2r$, provided $\delta^* \leq \alpha r^*$. (Their proof is stated for $\alpha = 2$, but can be extended to any $\alpha \geq 2$.) Assuming that we have a good estimate r satisfying $1.2r^* \leq r < (1.2 + 2\varepsilon/3)r^*$, we get the following.

Theorem 11. *For $\delta^* \leq \alpha r^*$, Algorithm 8 maintains a $(1.8 + \varepsilon)$-approximation to 2-center with z outliers in $O(dz^3)$ space and $O(dz^3)$ update time, assuming that the first point of the stream is not outlier, and that an estimate $1.2r^* \leq r < (1.2 + \frac{2}{3}\varepsilon)r^*$ is provided to the algorithm.*

Proof. Since our algorithm considers all possible solutions, the best solution obtained has larger radius at most $3r/2$ [13]. Combined by our assumption of $r \leq (1.2 + \frac{2}{3}\varepsilon)r^*$, the approximation factor of $3r/2 \leq (1.8 + \varepsilon)r^*$ follows. Our algorithm maintains at most two balls in each case, and therefore it uses $O(dz^3)$ space. Whenever a new point is inserted, the algorithm updates the solution for each subcase in $O(d)$ time. Therefore, the update time of the algorithm is $O(dz^3)$. Answering a query consists of choosing the minimum radius among all the candidate solutions, which amounts to $O(dz^3)$ total time. □

Maintaining an Estimate We assumed in Theorem 11 that at any time, an estimate $1.2r^* \leq r < (1.2 + 2\varepsilon/3)r^*$ is available to the algorithm. In the following, we show how such an estimate can be maintained upon processing the stream. The next lemma provides a key ingredient of our result.

Lemma 12. *Given a point set P in $\mathbb{R}^d$, an optimal solution to 1-center with z outliers on P yields a $(2 + \frac{\alpha}{2})$-approximation for 2-center with z outliers, provided that $\delta^* \leq \alpha r^*$.*

Proof. Let r_1^* and r^* be the optimal radii for the 1-center and 2-center problems with z outliers on P, respectively. It is clear that $r^* \leq r_1^*$, because any feasible solution B^* for

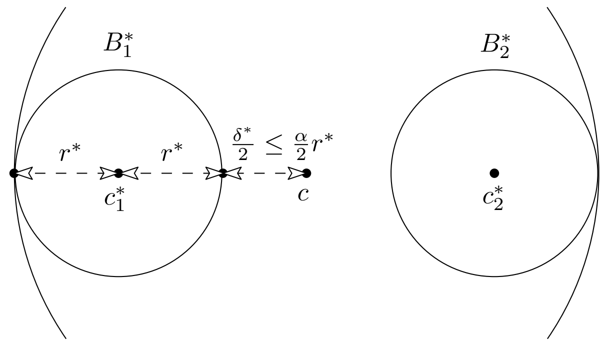

Figure 4: Illustrating the proof of Lemma 12

1-center with z outliers yields a feasible solution (B^*, B^*) for 2-center with z outliers. Now, suppose that $B_1^*(c_1^*, r^*)$ and $B_2^*(c_2^*, r^*)$ are the balls in an optimal solution for the 2-center problem with z outliers. Let c be the midpoint of the segment connecting c_1^* to c_2^* (see Figure 4). Clearly, $B\left(c, \frac{\delta^*}{2} + 2r^*\right)$ covers both B_1^* and B_2^*. Therefore, it is a feasible solution for the 1-center problem with z outliers. Hence, $r_1^* \leq \left(2 + \frac{\alpha}{2}\right) r^*$. □

Corollary 13. *If $\delta^* \leq \alpha r^*$, Algorithm 1 computes a $(4+\alpha)$-approximation to r^*.*

Proof. This is a direct corollary of Theorem 1 and Lemma 12. □

Lemma 14. *At any time over the stream, an estimate $1.2r^* \leq r < (1.2 + 2\varepsilon/3)r^*$ can be maintained in $O(dz^3/\varepsilon)$ space and $O(dz^3/\varepsilon)$ update time, assuming that the first point of the stream is not outlier.*

Proof. We use Algorithm 1 to find a $(4+\alpha)$-approximation to r^* by Corollary 13. Let r_i be the radius calculated by Algorithm 1 after receiving the i-th point, p_i. Clearly, the sequence of r_i's is increasing. Let k be an integer such that $2^{k-1} \leq r_i \leq 2^k$, and set $\ell_i = 2^k$. (If $r_i = 0$, we set $\ell_i = 0$.) Obviously, $\ell_i \leq 2r_i$, and hence, by Corollary 13, ℓ_i is a $(8+2\alpha)$-approximation to r^*. We divide the interval $(0, 1.2\ell_i]$ into $m = \lceil 1.2(3\alpha + 12)/\varepsilon \rceil$ equal segments, each of length $t_i = 1.2\ell_i/m$. Clearly, $t_i \leq (2\varepsilon/3)r^*$. Therefore, in the set $R_i = \{j \cdot t_i \,|\, j = 1, \ldots, m\}$, there is at least one value r for which the inequality $1.2r^* \leq r \leq (1.2 + \frac{2\varepsilon}{3})r^*$ holds.

We run m instances of Algorithm 8 for each value $r \in R_i$ in parallel. Whenever a new point p_i is added, if $\ell_i = \ell_{i-1}$, then $R_i = R_{i-1}$, and the new point is inserted to all parallel instances. If $\ell_i > \ell_{i-1}$, then the set R_i has two types of values. Those values in R_i which are less than $1.2\ell_i$ are also present in L_{i-1}, because t_i/t_{i-1} is a positive power of 2. For these values, we continue executing the corresponding instance. If a value $r \in R_i$ is not present in R_{i-1}, then we have $r \geq 1.2\ell_{i-1} \geq \ell_{i-1}$. Since those points not lying in the candidate solution are saved in the buffer of Algorithm 1 (which has size at most z), all non-outlier points of this algorithm lie in the candidate balls of Algorithm 8 which has center p_1 and radius at most l_{i-1}. These outliers have been stored in a buffer. Since Algorithm 8 maintains two balls with radius at least r, one of which (say B_1) is centered at p_1, then all non-outlier points of Algorithm 1 are in B_1, and hence, they do not make any transition in the states of Algorithm 8. Therefore, for any new value r, it suffices to execute Algorithm 8 with only the outlier points in the buffer of Algorithm 1. □

As described in the proof of Lemma 14, a good estimate for r^* can be obtained by running $O(1/\varepsilon)$ instances of Algorithm 8 in parallel. By adding another level of parallelization to remove the assumption of p_1 being a non-outlier, we get the following.

Theorem 15. *If $\delta^* \leq \alpha r^*$, a $(1.8+\varepsilon)$-approximation to 2-center with z outliers can be maintained in $O(\frac{dz^4}{\varepsilon})$ space and $O(\frac{dz^5}{\varepsilon})$ update/query time.*

Theorems 10 and 15 together yield the following main result of the paper.

Theorem 16. *Given a stream of points in d dimensions, we can maintain a $(1.8+\varepsilon)$-approximation to 2-center with z outliers using $O(d^3z^2 + dz^4/\varepsilon)$ space and $\mathrm{poly}(d, z, \frac{1}{\varepsilon})$ update/query time.*

5 Conclusions

In this paper, we presented a $(1.8+\varepsilon)$-approximation streaming algorithm for the 2-center problem with outliers in Euclidean space. It improves over the previous $(4+\varepsilon)$-approximation algorithm available for the problem due to McCutchen and Khuller [15]. It is interesting to see if the ideas used in this paper can be extended to the k-center problem with outliers in the data stream model, for general k, or even for small values of $k \geq 3$. Finding better approximation factors and/or space complexities for the cases $k = 1, 2$ is another interesting problem that remains open.

Acknowledgement The authors would like to thank Kiana Ehsani and Sahand Mozaffari for their thoughtful discussions, and for their very helpful comments on the first draft of this paper.

References

[1] P. K. Agarwal and J. M. Phillips. An efficient algorithm for 2d Euclidean 2-center with outliers. In *Proc. 16th Annu. European Sympos. Algorithms*, pages 64–75. 2008.

[2] P. K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. In *Proc. 21st ACM-SIAM Sympos. Discrete Algorithms*, pages 1481–1489, 2010.

[3] P. K. Agarwal and H. Yu. A space-optimal data-stream algorithm for coresets in the plane. In *Proc. 23rd Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 2007.

[4] T. M. Chan. More planar two-center algorithms. *Comput. Geom. Theory Appl.*, 13(3):189–198, 1999.

[5] T. M. Chan and V. Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. *Comput. Geom. Theory Appl.*, 47(2):240–247, 2014.

[6] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pages 642–651, 2001.

[7] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21(3):579–597, 1996.

[8] L. Danzer, B. Gruenbaum, and V. Klee. Helly's theorem and its relatives. In *Proc. Symposia in Pure Mathematics 7*, pages 101–180, 1963.

[9] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 434–444, 1988.

[10] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.

[11] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[12] S. Guha. Tight results for clustering and summarizing data streams. In *Proc. 12th Internat. Conf. Database Theory*, pages 268–275, 2009.

[13] S.-S. Kim and H.-K. Ahn. An improved data stream algorithm for clustering. *Comput. Geom. Theory Appl.*, 48(9):635–645, 2015.

[14] J. Matoušek. On geometric optimization with few violated constraints. *Discrete Comput. Geom.*, 14(1):365–384, 1995.

[15] R. M. McCutchen and S. Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Proc. 11th Internat. Workshop Approx. Algorithms*, pages 165–178, 2008.

[16] H. Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. *Algorithmica*, 60(1):46–59, 2011.

[17] H. Zarrabi-Zadeh and T. M. Chan. A simple streaming algorithm for minimum enclosing balls. In *Proc. 18th Canad. Conf. Computat. Geom.*, pages 139–142, 2006.

[18] H. Zarrabi-Zadeh and A. Mukhopadhyay. Streaming 1-center with outliers in high dimensions. In *Proc. 21st Canad. Conf. Computat. Geom.*, pages 83–86, 2009.