Streaming 2-Center with Outliers in High Dimension

Hamid Zarrabi-Zadeh*

Abstract

1 Introduction

2 Preliminaries

Let $B(c, r)$ denote a ball with center c and radius r. We denote by pq the straight line segment between these two points and by $|pq|$ the length of this segment.

Let $B_1^*(c_{2,1}^*, r_2^*)$ and $B_2^*(c_{2,2}^*, r_2^*)$ denote the optimal solution to 2-center problem with z outliers. By δ^* we denote the distance between B_1^* and B_2^*, that is, $\delta^* = \max(0, |c_1^* c_2^*|\, 2r^*)$.

Two congruent balls B_1 and B_2 with radius r are said to be *C-separated*, whenever the distance δ of two balls is at least Cr.

Given an n-point set $\mathcal{P}$ in d dimensions, a point $c \in R^d$ is called a *centerpoint* of $\mathcal{P}$ if any halfspace containing c contains at least $\left\lceil \frac{n}{(d+1} \right\rceil e$ points of $\mathcal{P}$. In other words, any halfspace (or convex set) that avoids a centerpoint can contain at most $\left\lfloor \frac{dn}{d+1} \right\rfloor$ points of $\mathcal{P}$.

3 A Simple 2-Approximation Algorithm for 1-Center with Outliers

Algorithm 1

function 1-CENTERWITHOUTLIERS($\mathcal{P}, z$)
    Let p_i denote the elements of $\mathcal{P}$, for $i \in \{1, \ldots, n\}$.
    for all $i \leftarrow 1, \ldots, z+1$ ***do***
        Assume p_i is a non-outlier point in optimal solution.
        $B(c_i, r_i) \leftarrow$ KNOWNPOINT($\mathcal{P}, z, p_i$)
    $i^* \leftarrow \arg\min_{i=1}^{z+1} r_i$
    return $B(c_{i^*}, r_{i^*})$

function KNOWNPOINT($\mathcal{P}, z, p$)
    $B \leftarrow B(p, 0)$
    $Q \leftarrow \emptyset$
    for p' *in* $\mathcal{P}$ ***do***
        if $p' \notin B$ ***then***
            Insert p' into Q
            if $|Q| = z+1$ ***then***
                Remove from Q a point q which is closest to p.
                $B \leftarrow B(p, |pq|)$
    return B

Function KNOWNPOINT in Algorithm 1, takes a point p as an argument which is guaranteed to be a non-outlier point in the optimal solution. To overcome the lack of knowledge such a point, function 1-CENTERWITHOUTLIERS tries each of the first $z+1$ points of the stream as a candidate for a non-outlier point and generates a ball for each, using function KNOWNPOINT. It returns the ball with the minimum radius. Note that, due to space limitations inherent to streaming data model, the body of for-all loop in function 1-CENTERWITHOUTLIERS is run in parallel.

Theorem 1 *Algorithm 1 is a 2-approximation algorithm for 1-center problem with z outliers, in any dimension.*

Proof. Let $B^*(c^*, r^*)$ be the optimal solution. Clearly, there exists a point p among the first $z+1$ points of $\mathcal{P}$, which is not an outlier in the optimal solution. Since $p \in B^*$, then $|pp'| \leqslant 2r^*$ for any point $p' \in B^*$. There is at least one point q' among the $(z+1)$-furthest points from p that is not an outlier in the optimal solution. Thus $|pq| \leqslant |pq'| \leqslant 2r^*$ and Algorithm 1 returns a valid solution of radius at most $2r^*$. So the proof is complete. □

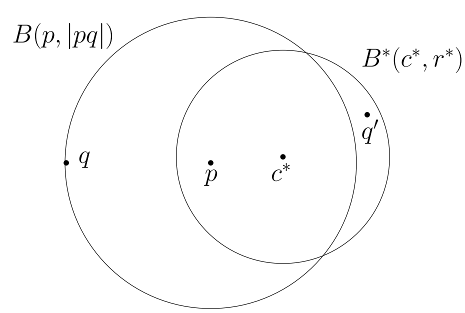

Figure 1: Output of Algorithm 1

The space complexity of Algorithm 1 is $O(z^2 + zd)$, the update time is $O(z \log z + zd)$ and the query time is $O(z)$. If a non-outlier point p is known then the space, update time and query time complexity of algorithm can be reduced by a factor of z.

*Department of Computer Engineering, Sharif University of Technology. zarrabi@sharif.edu

4 The 2-Center Problem with z Outliers

Theorem 2 *For any fixed $d > 1$, our single-pass data stream algorithm whose working space and update time are polynomial in d and sub-linear in n and returns two centers that guarantees a $(1.8+\varepsilon)$-approximation for the Euclidean 2-center problem with outliers.*

In all the algorithms given this section it is assumed that p_1 is a non-outlier point. It is not hard to see that, this limitation can be remedied by considering $\mathrm{O}(z)$ parallel instances of the algorithm, similar to Algorithm 1.

In Section 4.1, we give a description of our algorithm for a given $r' > 0$ for the case where $\delta^* < Cr^*$ and $1.2r^* \leqslant r' < (1.2 + 2\varepsilon/3)r^*$, our algorithm returns a $(1.8 + \varepsilon)$-approximate solution. We explain how to find such an r' and present a full description of our algorithm for the case where $\delta^* < Cr^*$ in Subsection 4.1.1. Then we inspect the case where $\delta^* \geqslant Cr^*$ in Subsection 4.2.

4.1 The Case $\delta^* < Cr*$

Our idea in this section is derived from [?, Sec. 3.1]. To avoid duplication, we just outline the important parts of their algorithm and our modifications to it. Kim's algorithm has 10 different states. In each step one or many of the states can be valid and their algorithm considers all of them in parallel. In each state, we have two balls. A transition between the states occurs whenever a point not covered by any of the two balls has arrived in the stream. This transition graph starts with a blank node (with no balls) and has depth four (without considering the end nodes). It is straightforward to see that this transition graph is a DAG. Our modification is on the transition part. In each state the number of points that should be considered as outlier is unknown. So all the possible choices should be considered. It is sufficient to have four integers, $n_1, \ldots, n_4$ representing the number of outliers in depth 1 to 4, such that $\sum_{i=1}^{4} n_i = z$. To sum up, the modified algorithm is defined as follows:

Algorithm 2

function $\textsc{Known}r'(\mathcal{P}, z, r')$
 for $(n_1, \ldots, n_4) \leftarrow \{(n_1, \ldots, n_4) : \sum n_i = z\}$ ***do***
  $CS \leftarrow B(p_1, r')$.
  $p_{o_2} \leftarrow (n_1 + 1)^{st}$ *point of* $\mathcal{P}$ *that arrives after* p_1 *and does not lie in the corresponding candidate solution.*
   if p_{o_2} *arrives* ***then***
    $CS \leftarrow$ *the case of* $p_{o_2} \in B_1^* \cup$ *the case of* $p_{o_2} \in B_2^*$.
   for *each case* S ***do***
    $p_{o_3} \leftarrow (n_2 + 1)^{st}$ *point of* $\mathcal{P}$ *that arrives after* p_{o_2} *and does not lie in the corresponding candidate solution.*
     if p_{o_3} *arrives* ***then***
      $CS \leftarrow CS \setminus S \cup$ *the sub-case of* $p_{o_3} \in B_1^* \cup$ *the sub-case of* $p_{o_3} \in B_2^*$.
     for *each sub-case* S' ***do***
      $p_{o_4} \leftarrow (n_3 + 1)^{st}$ *point of* $\mathcal{P}$ *that arrives after* p_{o_3} *and does not lie in the corresponding candidate solution.*
       if p_{o_4} *arrives* ***then***
        Replace S' *with a new candidate solution.*
       if *more than* n_4 *points arrive after* p_{o_4} *that lie outside of the corresponding candidate solutions* ***then***
        Abandon the solution
      $Sol_{n_1,n_2,n_3,n_4} \leftarrow$ *the solution with the smallest larger radius, among all the sub-cases.*
 return *the solution* Sol_{n_1,n_2,n_3,n_4} *with the smallest radius, for all valid values of* (n_1, n_2, n_3, n_4).

Theorem 3 *Given $\delta^* < Cr*$ and $1.2r^* \leqslant r' < (1.2 + 2\varepsilon/3)r^*$, Algorithm 2 uses $\mathrm{O}(dz^3)$ space and returns a $(1.8 + \varepsilon)$-approximate solution. Algorithm 2 spends $\mathrm{O}(dz^3)$ update time for each point in $\mathcal{P}$ and answers a query in $\mathrm{O}(dz^3)$.*

Proof. Since our algorithm considers all possible input cases of streaming points, there is at least one feasible solution. Since every feasible solution has its larger radius at most $3r'/2$, the final solution has larger radius at most $3r'/2 \leqslant (1.8 + \varepsilon)r^*$. For space complexity, our algorithm maintains at most two balls in each case, and therefore it uses $\mathrm{O}(dz^3)$ space. Whenever the next point is inserted, the algorithm updates the solution for each sub-case in $\mathrm{O}(d)$ time. Therefore, the algorithm spends $\mathrm{O}(dz^3)$ update time for each point of $\mathcal{P}$. Answering a query consists of choosing the minimum radius among all the candidate solutions, which amounts to $\mathrm{O}(dz^3)$ time. □

4.1.1 Finding r'

Lemma 4 *A 1-center optimal solution with z outliers for a streaming set of points is also an upper bound for the 2-center optimal solution with z outliers.*

Proof. Let $B_1^*(c^*, r_1^*)$ be the optimal solution for the 1-center problem. Let B_2 be a ball with radius r_1^* and center at ∞. Clearly, (B_1^*, B_2) is a solution for 2-center problem. Thus, the proof is complete. □

Lemma 5 *Let r_1^*, r_2^* be the radius of 1-center and 2-center optimal solutions with z outliers, respectively. Then $r_1^* \leqslant \left(\frac{C}{2} + 2\right) r_2^*$.*

Proof. Suppose that $(B_1^*(c_1^*, r_2^*), B_2^*(c_2^*, r_2^*))$ is an optimal solution for 2-center problem. Consider m as the midpoint of segment $c_1^* c_2^*$. Clearly, $B\left(m, \frac{\delta}{2} + 2r_2^*\right)$ covers both B_1^* and B_2^*. So it is a solution for 1-center problem and the proof is complete. □

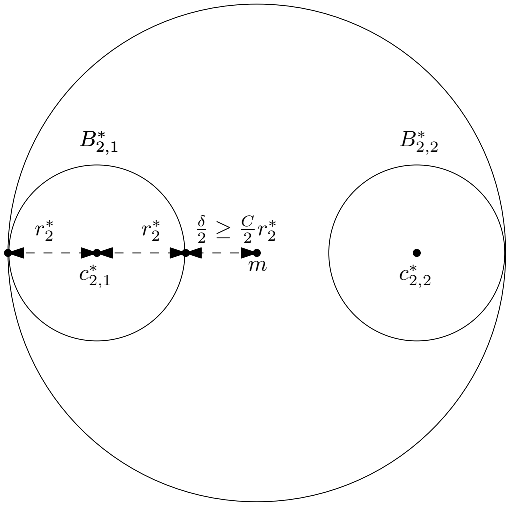

Figure 2: Proof of Lemma 5

Algorithm 1 finds a 2-approximation for r^*_1. By Lemma 4 and 5, it gives a $2\left(\frac{C}{2}+2\right)$-approximation for r^*_2.

Observation 1 *Let r_1 and k_0 be two positive real numbers. Define $k = k_0 2^i$ to be the smallest real number satisfying the inequality $k \geqslant r_1$, where i is a non-negative integer. Clearly, k is a 2-approximation for r_1.*

Our algorithm maintains $m = \left\lceil \frac{1.2(3C+12)}{\varepsilon} \right\rceil$ candidate lengths. For each candidate length t, we assume that $r' = t$ and use it to run Algorithm 2. We use Algorithm 1 to estimate r'. Let k_i be the i^{th} non-zero radius calculated by the algorithm. Obviously, the sequence of answers given by the algorithm is increasing.

Let $l_1 = k_1$ and $l_i = 2^j l_1$, where l_i is the smallest number satisfying $l_i \geqslant k_i$. By Observation 1, l_i is a $(2C+8)$-approximation for r^*_2. So the following inequalities hold:

$$\frac{2\varepsilon}{3}\frac{1.2l_i}{2C+8} \leqslant \frac{1.2l_i}{2C+8} \leqslant 1.2r^*_2 \leqslant 1.2l_i$$

Define $t_i = \frac{1.2l_i}{m}$. If we divide the interval $(0, 1.2l_i]$ into m equal segments, then length of each segment is at most $2/3\varepsilon r^*_2$ and the endpoints as candidates are $\mathcal{L}_i = \{j \times t_i : j = 1, \ldots, m\}$. The inequality $1.2r^*_2 \leqslant j \times l_i \leqslant (1.2 + \frac{2\varepsilon}{3})r^*_2$ holds for at least one of these candidates.

When the first non-zero radius is observed, l_1 is set. Algorithm 2 should be executed for each candidate in $\mathcal{L}_1$ for all the previous points, which have been stored in a buffer. It is not hard to see that the space complexity for storing these points is $\mathrm{O}(z)$. For each new point, if $l_i = l_{i-1}$ then $\mathcal{L}_i = \mathcal{L}_{i-1}$ and it suffices to add the new point to all parallel instances corresponding to candidates in $\mathcal{L}_i$. Otherwise, Algorithm 2 should be executed for all of the candidates in $\mathcal{L}_i$, with all the

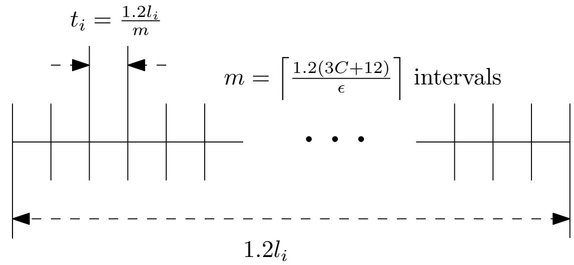

Figure 3: Subdivision of $(0, 1.2l_i]$ interval

points. But this is not achievable due to space limitations. To remedy this problem, note that if the candidate $x_j = j \times t_i \in \mathcal{L}_i$ is also in $\mathcal{L}_{i-1}$, there is no need to start Algorithm 2 over and the current execution will be continued. If $x_j \notin \mathcal{L}_{i-1}$, then it is certain that $x_j \geqslant l_{i-1}$ (Observe that $l_i = ul_{i-1}$, for some $u \in \mathbb{N}$). Since the ball $B_1(p_1, l_{i-1})$, obtained from Algorithm 1, has at most z outliers, all the non-outlier points lie in the candidate balls in Algorithm 2. These outliers have been stored in a buffer, so for the new candidate length, it suffices to execute Algorithm 2 with only the outlier points of B_1.

Theorem 6 *Algorithm 2 uses $\mathrm{O}(\frac{dz^3}{\varepsilon})$ space, and takes $\mathrm{O}(\frac{dz^4}{\varepsilon})$ and $\mathrm{O}(\frac{dz^3}{\varepsilon})$ time for update and query, respectively.*

4.2 The Case $\delta^* \geqslant Cr^*$

Observation 2 *Let d be the distance between any pair of points from two C-separated balls. Then $1 \leqslant \frac{d}{\delta} \leqslant \frac{C+4}{C}$.*

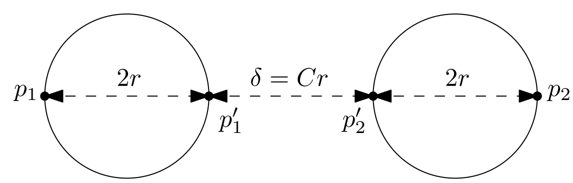

Figure 4: Depiction of two extreme case of Observation 2

Observation 3 *Let B_1 and B_2 be two disjoint balls with distance δ and B be an arbitrary ball of radius at most $\frac{\delta}{2}$. Then B can intersect at most one of B_1 and B_2.*

Lemma 7 *Let $\mathcal{P}$ be a set of points and assume that $B^*_{2,1}$ and $B^*_{2,2}$ are C-separated $(C \geqslant 4)$ and let p_1 be an arbitrary point in $B^*_{2,1}$. Define S as the $z+1$ furthest points of $\mathcal{P}$ from p_1. Then $S \cap B^*_{2,2}$ is nonempty.*

Proof. By contradiction, assume that $S \cap B^*_{2,2}$ is empty. It is clear that S contains at least one non-outlier point.

Let q be the furthest point of $S \cap B^*_{2,1}$ from p_1. Consider $B(p_1, |p_1q|)$. It is clear that B covers $\mathcal{P} \setminus S$. Since $p_1, q \in B^*_{2,1}$, then $|p_1q|$ is at most $2r^*_2$. Thus by Observation 3, $B^*_{2,2} \cap B = \emptyset$. Therefore $B^*_{2,2} \cap \mathcal{P} = \emptyset$ and hence $B^*_{2,2}$ is empty, which contradicts the optimality of this solution. □

Let B_1, B_2 and B_u, be instances of a data structure that supports adding a point and gives a β-approximation for 1-center problem with k outliers, for $k = 0, \ldots, z$ (This data structure, does not need to maintain all the points. Suppose that it has a buffer with length $(d+1)(z+1)$ that maintains the most recently added points). Algorithm 3 assumes point p_1 to be in $B^*_{2,1}$. This assumption can be omitted by executing a parallel instances of the algorithm for each of the first $z+1$ points in the stream, similar to the approach in Algorithm 2. This algorithm maintains c_1 as an arbitrary point in $B^*_{2,1}$ and c_2 as a candidate that may be in $B^*_{2,2}$. It tries to divide points into three subdivisions:

- B_1, a subset of $\mathcal{P}$ where $B_1 \cap B^*_{2,2} = \emptyset$, that is a set of points that provably lie outside $B^*_{2,2}$.
- B_2, a subset of $\mathcal{P}$ along with a *secondary point*, such that if $B^*_{2,2}$ covered the secondary point, then $B_2 \cap B^*_{2,1} = \emptyset$.
- Buffer, with size at most z that if $c_2 \in B^*_{2,2}$ then all of them are outliers.

Algorithm 3 also maintains B_u as union of B_1 and B_2.

Lemma 8 *Let $\mathcal{P}$ be a set of points and B_1 and B_2 be two C-separated balls covering all but at most z of the points in $\mathcal{P}$. Furthermore, let p_1 be arbitrary points in B_1 and q be the $(z+1)^{st}$ furthest point from p_1. Then $\frac{C}{C+4}|qp_1|$ is a lower bound for the distance between B_1 and B_2.*

Proof. Due to Lemma 7, there exists $p_2 \in B^*_{2,2}$ such that $|p_1p_2| \geqslant |qp_1|$. Thus, by Observation 2, $\frac{C}{C+4}|p_1p_2| \leqslant \delta$. The claim follows immediately. □

Function $\textsc{AddTo}B_1$ is used to add a point p to B_1. The point will be added only if the point is within δ-radius of the center c_1, where δ, calculated by Algorithm 3, is $\frac{C}{C+4}|qc_1|$ and q is the $(z+1)^{th}$ furthest point from c_1. By Lemma 8, δ is a lower bound for δ^*. So any point within δ-radius of c_1 is guaranteed to lie outside $B^*_{2,2}$. Thus Function $\textsc{AddTo}B_1$ does not violate the property $B_1 \cap B^*_{2,2} = \emptyset$.

function $\textsc{AddTo}B_1(p)$
  if $p \in B(p_1, \delta)$ **then**
    $B_1 = B_1 \cup \{p\}, B_u = B_u \cup \{p\}$
    return true
  else
    return false

Function $\textsc{AddTo}B_2$ is used to add a point p to B_2. Initially, B_2 is empty and c_2 and r_c are not initialized. When an appropriate candidate is found, Algorithm 3 sets c_2 to that candidate and r_c to $\frac{2|c_1c_2|}{C}$ and thereafter points are added to B_2 only if they lie within r_c-radius of c_2. At this point c_2 is considered as the secondary point for B_2. Clearly the subdivision property for B_2 holds initially when B_2 consists of the single point c_2. All the subsequent points added to B_2 are within r_c-radius of c_2 and given $C \geqslant 7$, by Observation 3 subdivision property is invariant under this operation.

When $|B_2|$ reaches $(d+1)(z+1)$, r_c is increased by a factor of $(2+\frac{2}{C})$. It is easy to see that $(2+\frac{2}{C})r_c < \frac{\delta}{2}$ provided $C \geqslant 14$ and thus by Observation 3 the property is not violated.

function $\textsc{AddTo}B_2$(p)
  if c_2 is set and $p \in B(c_2, r_c)$ **then**
    $B_2 \leftarrow B_2 \cup \{p\}, B_u \leftarrow B_u \cup \{p\}$
    if $|B_2| = (d+1)(z+1)$ **then**
      $r_c \leftarrow (2 + \frac{2}{C}) \times r_c$
      for p in Buffer **do**
        if p in $B(c_2, r_c)$ **then**
          $B_2 \leftarrow B_2 \cup \{p\}, B_u \leftarrow B_u \cup \{p\}$
    return true
  else
    return false

Proposition 9 *Any set $\mathcal{P}$ of points in d-dimensional Euclidean space has a centerpoint.[?]*

Corollary 10 *Given a set $\mathcal{P}$ of $k(d+1)$ points in d-dimensional Euclidean space, there is a point c_p, not necessarily in $\mathcal{P}$, that any convex object that does not cover c_p, leaves at least k point of $\mathcal{P}$ uncovered.*

Let c_p be a centerpoint of points in B_2, when $(z+1)(d+1)$ points have been added to B_2. We claim that with $B_2 = B(c_2, (2+\frac{2}{C})r_c)$, c_p can be chosen as the secondary point for $B_2(c_2, (2+\frac{2}{C})r_c)$. If $c_p \in B^*_{2,2}$ then by Observation 2, $r^* \leqslant \frac{|c_1c_p|}{C}$. Thus $B^*_{2,2} \subseteq B(c_p, \frac{2|c_1c_p|}{C})$. Since:

$$\frac{2\,|c_1c_p|}{C} \leqslant \frac{2}{C}(|c_1c_2| + r_c) \leqslant (1 + \frac{2}{C})r_c$$

then $B^*_{2,2} \subseteq B(c_p, (1+\frac{2}{C})r_c) \subseteq B(c_2, (2+\frac{2}{C})r_c)$. So $B_2(c_2, (2+\frac{2}{C})r_c) \cap B^*_{2,2} \neq \emptyset$. Thus supposing $C \geqslant 14$ and by Observation 3, $B_2 \cap B^*_{2,1} = \emptyset$. Thus the claim follows.

At some point during the execution of Algorithm 3, the assumption about c_2 lying in $B^*_{2,2}$ might be contradicted when the buffer overflows. We can then conclude that $c_2 \notin B^*_{2,2}$. At this point, behavior of Algorithm 3 depends on size of B_2 relative to $(d+1)(z+1)$:

- If $|B_2| < (d+1)(z+1)$ then c_2 can be added to B_1, by definition of B_1. The algorithm then tries all points in $B_2 \cup \text{Buffer} \setminus \{c_2\}$, until it finds a suitable candidate point for c_2 that does not leave too many points uncovered. Note that at most $(z+1)(d+1)+z$ points have to be tried.

- If $|B_2| \geqslant (d+1)(z+1)$, let q be the $(z+1)^{st}$ furthest point from p_1 and p' be the nearest point of ball $B_2(c_2, r_c)$ to p_1. Then $|qp_1| > |p'p_1|$, since B_2 contains more than $z+1$ points. So by Lemma 8:

$$\delta \geqslant \frac{C\,|qp_1|}{C+4} \geqslant \frac{C}{C+4}(|c_1c_2| - r_c) = \frac{C^2-2C}{2C+8} r_c$$

As a result, if $C \geqslant 14$ then $(2+\frac{2}{C})r_c < \frac{\delta}{2}$. As shown in Observation 3, the ball $B_2(c_2, (2+\frac{2}{C})r_c)$ has nonempty intersection with exactly one of $B^*_{2,1}$ and $B^*_{2,2}$ (not all of its points can be outliers). Since the buffer overflowed, the assumption about the secondary point, c_p, being in $B^*_{2,2}$ must have been invalid. By Corollary 10 if c_p is not covered, then circle $B^*_{2,1}$ leaves at least $z+1$ points of B_2 uncovered, and there will be more than z outliers. Thus, c_p has to be covered. As a result, $c_p \in B^*_{2,1}$ and all the points of B_2 can be added to B_1. This is done by replacing B_1 with B_u and initializing B_2 over again.

Algorithm 3

```
δ ← 0, r_c ← 0
c_1 ← p_1
for p ∈ P do
    if at least z + 1 points have been processed then
        q ← (z + 1)-furthest point from c_1
    else
        q ← c_1
    δ ← C/(C+4) |c_1 q|
    if !AddToB_1(p) and !AddToB_2(p) then
        add p to Buffer
        while |Buffer| > z do
            if |B_2| ⩾ (d + 1)(z + 1) then
                B_1 ← B_u, B_2 ← ∅
            else if c_2 is set then
                B_1 ← B_1 ∪ {c_2}
                remove c_2 from B_2
            T ← Buffer ∪ B_2
            Buffer ← ∅, B_2 ← ∅
            if first iteration of while then
                for p ∈ T do
                    if AddToB_1(p) then
                        remove p from T
            c_2 ← arbitrary point in T
            r_c ← 2/C |c_1 c_2|
            for p ∈ T do
                AddToB_2(p)
            Buffer ← T \ B_2
```

Let B'_1, B'_2 and B'_u be the 1-centers obtained from B_1, B_2 and B_u by the β-approximation algorithm, respectively. To answer a query, we use these balls. By the initial assumption about p_1, B_1 contains $B^*_{2,1}$ and B'_1 is a β-approximation for $B^*_{2,1}$. But for B_2 it may be the case that our assumption about the secondary point was incorrect or non-optimal, so B'_2 is not a good approximation for $B^*_{2,2}$. There are two cases for $|B_2|$ to be considered. If $|B_2| < (z+1)(d+1)$, then we can try each point in $B_2 \cup \text{Buffer}$ as a candidate for the secondary point. If $|B_2| \geqslant (d+1)(z+1)$, then there is no need for points in B_2 to be considered, since for those points as secondary point, B_1 would be replaced by B_u and B_2 would be empty. Thus it suffices to consider points of Buffer as candidates and compare them to the solution $(B_u, \emptyset)$.

Algorithm 4

```
function Query
    solutions = [query B_1 B_2 Buffer]
    if |B_2| ⩾ (d + 1)(z + 1) then
        candidates ← Buffer
    else
        candidates ← Buffer ∪ B_2 − {c_2}
    for c_2 ∈ candidates do
        r_c ← 2/C |c_1 c_2|
        B_2 ← ∅, Buffer ← ∅
        for p ∈ candidates do
            if not AddToB_2(p) then
                add p to Buffer
        add QuerySubdivision(B_1, B_2, Buffer) to solutions
    return min(solutions)

function QuerySubdivision(B_1, B_2, Buffer)
    solutions ← empty list
    for k ← 0, ..., (z − |Buffer|) do
        r ← max(1 − center(B_1, k), 1 − center(B_2, z − |Buffer| − k))
        add r to solutions
    return min(solutions)
```

Suppose that our data structure for maintaining B_1, B_2 and B_u uses $S(n,z,d)$ space, $T(n,z,d)$ update and $Q(n,z,d)$ query time. Note that by $Q_o(zd,z,d)$ we mean the query time for the offline 1-center problem with z outlier in d dimensions. The best known algorithm for 1-center without outliers is 1.22-approximation by [?]. The buffering framework introduced in [?] can use this algorithm to achieve a $(1.22\sqrt{2})$-approximation $(1.22\sqrt{2} < 1.8)$ for 1-center problem with outliers.

Theorem 11 *Space complexity of Algorithm 3 is* $O(dz + S(n,z,d))$. *It takes* $O^*(dzT(n,z,d))$ *time for* *update. Query time in Algorithm 4 is* $O(zQ(n,z,d) + dz(dz + zQ_o(zd,z,d)))$ *and it returns two centers that guarantee a* $(1.8+\varepsilon)$*-approximation for* 2*-center problem with outliers.*

Proof. The space complexity for this algorithm is derived from the space used by B_1, B_2, B_u and the Buffer which is obviously $O(dz + S(n,z,d))$. The while loop runs at most once for each point, so amortized time is $O(zdT(n,z,d))$. The candidates in the Algorithm 4 are at most zd points so the query time will easily follows. $\square$

Theorem 12 *Our algorithm uses* $O(zS(n,z,d) + \frac{dz^4}{\varepsilon})$ *space and has time complexity* $O^*(dz^2T(n,z,d) + \frac{dz^5}{\varepsilon})$ *and* $O(z^2Q(n,z,d) + dz^2(dz + zQ_o(zd,z,d)) + \frac{dz^4}{\varepsilon})$ *for update and query operations, respectively. It returns a* $(1.8+\varepsilon)$*-approximate solution for* 2*-center problem with outliers in any dimension.*

Proof. Result follows immediately from Theorems 6 and 11 and the fact that p_1 was assumed to be a non-outlier. $\square$