Approximation Algorithm for the Kinetic Robust K-Center Problem

Sorelle A. Friedler* David M. Mount†

Department of Computer Science, University of Maryland, College Park, MD 20742, USA,
{sorelle, mount}@cs.umd.edu,
http://www.cs.umd.edu/~sorelle/

Abstract

Clustering is an important problem and has numerous applications. In this paper we consider an important clustering problem, called the k-center problem. We are given a discrete point set S and a constant integer k, and the goal is to compute a set of k center points to minimize the maximum distance from any point of S to its closest center. We consider both the discrete formulation, in which center points are restricted to be selected from S, and the absolute formulation, in which the centers may be chosen from any point in space. We consider two generalizations of this problem, inspired by issues that arise in real applications. First, we consider a robust version of the problem, in which the user provides a parameter $0 < t \leq 1$, and the algorithm is required to cluster only a fraction t of the points, thus allowing some fraction of outlying points to be ignored. Second, we consider the problem in a kinetic context, where points are assumed to be in motion. We present a kinetic data structure (in the KDS framework) that maintains a $(3+\varepsilon)$-approximation for the robust discrete k-center problem, and a $(4+\varepsilon)$-approximation for the robust absolute k-center problem. We also improve on a previous 8-approximation for the non-robust kinetic k-center problem for arbitrary k and show that our data structure supports a $(4+\varepsilon)$-approximation.

1 Introduction

The calculation of clustering properties of points is a frequently studied problem in operations research and computer science. Many problem variations have been considered including k-center, k-means, and facility location problems [8, 14, 16, 17].

The (non-robust) *k-center problem* is defined as follows: Given a set of n points, find k center points that minimize the maximum distance from any point to its closest center. The version of this problem where each of the k centers must be one of the original n points is known as the *discrete k-center problem.* The version where the centers may be any point in space is known as the *absolute k-center problem* [16]. We consider the geometric version of this problem, where distance is the Euclidean metric.

Kariv and Hakimi [16] proved that the discrete and absolute versions of the k-center problem are NP-hard. Feder and Greene show that the problem in a geometric context cannot be approximated to within a factor of 1.822 (assuming P = NP) [8]. It is also known that the problem of

*The work of Sorelle Friedler has been supported by the AT&T Labs Fellowship Program.

†The work of David Mount has been supported in part by the National Science Foundation under grant CCR–0635099.

finding a $(2-\varepsilon)$-approximation for the k-center problem is NP-complete when the k-center problem is considered in a graph theoretic context [15].

Since the k-center problem is NP-hard, considering approximation algorithms is appropriate. An algorithm provides a *c-approximation* to the k-center problem if the radius associated with the k chosen centers is no more than c times the optimal radius. Feder and Greene [8] gave a 2-approximation for the geometric k-center problem and Hochbaum and Shmoys [14] and Gonzalez [10] gave 2-approximation algorithms for the graph theoretic version of the k-center problem. Since there is a known lower bound of $(2-\varepsilon)$ for the graph version of the k-center problem, these approximation algorithms give the best approximation bound possible.

In many applications of clustering, it is desirable to ignore some number of distant points called *outliers*. The study of statistical estimators that are insensitive to outliers is the domain of *robust statistics* [18]. Robust statistics have been extensively studied in mathematics, operations research, and computer science. The flexibility to ignore outliers is important when considering experimental data which may contain measurement errors or isolated and unusual data points. It is also often a necessity when considering business-related problems where cost is an important factor. Charikar *et al.* [7] explored the robust facility location problem, which determines the locations of stores while minimizing the distance from customers to the stores and the total cost of opening facilities. When considering cost it is desirable to have a model which will not require a new facility for very isolated customers.

In this paper we focus on the robust version of the k-center problem. The *robust k-center problem* modifies the k-center problem to allow flexibility in the number of points that satisfy the distance criteria. In our formulation we are given a set of n points in Euclidean space, a constant k, and a threshold parameter t, where $0 < t \leq 1$. The objective is to compute the smallest radius r such that there exist k disks of radius r that cover at least $\lceil tn \rceil$ points. For $t = 1$ this problem is the same as the non-robust formulation.

Since the robust k-center problem is a generalization of the non-robust version, the 1.822 approximation lower bound [8] (assuming P $\neq$ NP) for geometric contexts holds for the robust k-center problem as well. Charikar *et al.* [7] showed that in the graph-theoretic context for the case of the robust k-center problem with forbidden centers (in which some locations cannot be chosen as centers), this lower bound increases to $3-\varepsilon$. They also gave a 3-approximation algorithm for the robust k-center problem.

Objects in motion (e.g., airplanes or cell-phone users) present unique data structure challenges. Many frameworks have been studied for handling *kinetic data* [4, 13, 19, 20]. These generally rely on *a priori* information about point motion (e.g., a "flight plan" in the form of an algebraic expression). We focus on the handling of kinetic data through the model of kinetic data structures (KDSs) proposed by Basch, Guibas, and Hershberger [5]. KDSs have become the standard for kinetic data and offer the flexibility of allowing flight plans to change.

Sometimes, in addition to locations of points in motion, it is also desirable to answer questions about the structure of these points. Problems formerly framed on static data sets are now being considered in a kinetic context. Standard problems which have been tackled include finding the convex hull [5], Voronoi diagram [2], and minimum spanning tree on geometric graphs [6].

Kinetic data structures track specific properties of moving points. This is done through a set of boolean conditions, called *certificates*, and a corresponding set of update rules. *Certificates* are boolean conditions that guarantee geometric relations necessary to a particular problem's solution, and the rules specify how to respond when a certificate fails. Certificate failures are predicted and

queued based on information about the points' planned paths of motion, assumed to be in the form of algebraic expressions.

There are four criteria under which the computational cost of a KDS is evaluated: responsiveness, efficiency, compactness, and locality [11]. *Responsiveness* measures the complexity of the cost to repair the solution after a certificate fails. *Efficiency* measures the number of certificate failures as compared to the number of required changes to the solution as the points move. *Compactness* measures the size of the certificate set. *Locality* measures the number of certificates in which each point participates. Guibas provides a more detailed overview of kinetic data structures in [11].

The kinetic k-center problem is a generalization of the static version, so the 1.822 approximation lower bound [8] (assuming P $\neq$ NP) for the geometric version which we consider holds for this problem as well. No other lower bounds are known for the kinetic problem. Gao, Guibas, and Nguyen [9] give an 8-approximation algorithm for the discrete k-center problem.

1.1 Contributions

In this paper we consider, for the first time, a combination of these three important paradigms — clustering, robust statistics, and kinetic data structures — and present an approximation algorithm and corresponding efficient kinetic data structure to solve the kinetic robust k-center problem. We assume throughout that k is a constant. The static k-center, kinetic k-center, and robust k-center problems are all special cases of our problem and as such our approximation algorithm solves these cases as well.

We obtain a $(3+\varepsilon)$-approximation for the static, kinetic, and robust forms of the discrete k-center problem and a $(4+\varepsilon)$-approximation for the absolute version of the k-center problems. Note that the first bound improves upon the 8-approximation for the kinetic discrete k-center problem as given by Gao, Guibas, and Nguyen [9] and generalizes it to the robust setting. However, our result assumes that k is constant while theirs holds for arbitrary k. We improve their result for the non-robust kinetic problem for arbitrary k by showing that our data structure achieves a $(4+\varepsilon)$-approximation while maintaining the responsiveness of their KDS (see Section D).

To our knowledge, our kinetic robust algorithm is the first approximation algorithm for the kinetic absolute k-center problem (even ignoring robustness), and the first to tackle any kinetic, robust problem. We give an example to show that our $(3+\varepsilon)$-approximation for the robust, discrete k-center problem is tight.

The KDS used by our algorithm is efficient. We achieve bounds of $O(\log \alpha/(\varepsilon^d))$ for locality and $O(n/(\varepsilon^d))$ for compactness (where α is an upper bound on the ratio between the largest and smallest inter-point distances of the point set under motion) so we do not create too many certificates. Our responsiveness bound is $O((\log \alpha \log n)/(\varepsilon^d))$, so the data structure is able to update quickly. Our efficiency bound of $O(n^2 \log \alpha)$ is reasonable since the combinatorial structure of the spanner requires $\Omega(n^2)$ updates [9], so any approach based on a spanner requires $\Omega(n^2)$ updates.

2 Weak Hierarchical Spanner

Our approach is to extend a spanner construction for kinetic data structures developed by Gao *et al.* [9] which they call a *deformable spanner*. We first describe their spanner and results.

The deformable spanner is defined assuming a point set S in $\mathbb{R}^d$. The *aspect ratio*, denoted α, is defined as the ratio between the maximum distance between two points and the minimum distance between two points. Since our point set is not static, the aspect ratio is actually a function of time,

$\alpha(\tau)$. When considering the aspect ratio in the context of time complexity, we actually consider the maximum $\alpha(\tau)$ over all τ. For shorthand, we will refer to both meanings as α and rely on context to distinguish between them.

Using the terminology of Gao *et al.* [9] the spanner is constructed based on a *hierarchy of discrete centers*. To avoid confusion with the use of the term "center," henceforth we will use the term *node* for points in the spanner hierarchy and use the term *center* when referring to disk centers in the solution to the k-center problem. The following properties hold for the hierarchy for $0 \leq i \leq \lceil \log \alpha \rceil$:

- S_0 is the entire point set S.
- Each point in S_{i-1} is within distance 2^i of some node in S_i, the ith level of the hierarchy.
- Nodes in S_i are chosen from S_{i-1}.
- If x is a node in S_i then x is also in levels 0 through $i-1$.

Any set of nodes for which these properties hold is valid.

A node is said to *cover* a point in the level below if that point is within distance 2^i of the node. The node which covers a point is that point's *parent* (if there are multiple such nodes, one is chosen arbitrarily). The point covered is called the *child* of that node. Similarly, the parent of a point's parent is its *ancestor* and so on recursively. A point in the complementary relationship is called a *descendant*. Some properties about the deformable spanner as proven in [9] are given below:

- $S_i \subseteq S_{i-1}$
- For any two points $p, q \in S_i, \|pq\| \geq 2^i$.
- There is an edge from each point to its parent.
- The hierarchy has a height of at most $\lceil \log_2 \alpha \rceil$.
- Any point in S_0 is of a distance at most 2^{i+1} away from its ancestor in level S_i.

The deformable spanner maintains four types of certificates; parent-child certificates, edge certificates, separation certificates, and potential neighbor certificates (to keep track of cousins). The KDS for this spanner is appropriately efficient, local, compact, and responsive (see Section 4.3).

Our *weak hierarchical spanner* is a variant of the deformable spanner. We combine the Gao *et al.* [9] hierarchy of discrete centers with the neighbor relationships used in their spanner. We then create $s(\varepsilon)$ copies of this structure with varying distances certified (see Section 4). This structure is a *weak spanner* in the sense that it has a constant stretch factor. For the rest of the paper we call this structure the "spanner."

For most of this paper, we consider the processing of only one of these spanners. The manipulation of each of these spanners is determined by the spanner's *base distance*. In the description given above, this distance is 2^i, however this distance varies slightly for each spanner we create. Instead, each spanner has a base distance of $2^i b$ where $b = (1 + \frac{p}{s})$ and $0 \leq p < s(\varepsilon)$ is the counter identifying the current spanner. For ease of reading, we assume for most of the paper that $p = 0$ and so $b = 1$.

3 Robust K-Center Algorithm

Gao *et al.* [9] gave an 8-approximation for the non-robust discrete version of the k-center problem (for arbitrary k) on their deformable spanner. In Section D we improve this to a $(4+\varepsilon)$-approximation for arbitrary k. We now give a $(3+\varepsilon)$-approximation for the robust discrete version of this problem and a $(4+\varepsilon)$-approximation for the robust absolute version, both for constant k. Recall that the non-robust version can be expressed as the robust version in which the required threshold fraction of covered points is $t = 1$, so these algorithms also hold for the non-robust case.

3.1 Intuitive Explanation

For the sake of intuition regarding some of the more complex technical elements of our algorithm we first present the algorithm by Charikar *et al.* [7] for the static robust k-center problem, which is the basis for our algorithm, and explain why it can not be directly applied to this problem. Henceforth we refer to it as the *expanded greedy algorithm*.

This algorithm creates two types of disks centered at a point v in the input set S. The disks G_v have radius r and the disks E_v have radius $3r$ where r is a pre-determined input to this algorithm which is designed to be repeatedly called in a parametric search so that all potential values for r are included. In a slight abuse of notation we sometimes let G_v and E_v represent the geometric disk and sometimes the points contained within the disk. It is clear from context which interpretation is being used. These disks are referred to, respectively, as *greedy disks* and *expanded disks* due to their role in the expanded greedy algorithm (see Figure 1).

The proof of correctness for the expanded greedy algorithm uses a charging argument where each point covered by an optimal disk is charged to the expanded disk that covers it or to a non-overlapping greedy disk. The proof relies on two main points. First, that for any G_v each optimal disk is either disjoint from G_v or completely covered by E_v due to the expansion factor. Second, that it is possible to choose a greedy disk that covers some optimal disk.

When initially considering adapting the expanded greedy algorithm to the kinetic problem on the spanner described in Section 2, we considered using this algorithm as a per-level subroutine. This initial algorithm starts at the highest level of the spanner and works down. For each level i, it runs the expanded greedy algorithm with radius 2^i. It returns the set of centers associated with the lowest level which succeeds in covering $\lceil tn \rceil$ points. There are, however, some shortcomings to this approach due to assumptions that are not true in our context.

All important radii will be considered. The proof of the expanded greedy algorithm, and of our algorithm which follows, relies on the possibility for the algorithm to pick a node and cover all points within the optimal radius of that node. However, in this initial algorithm, if the optimal radius is slightly larger than 2^i then our algorithm would be forced to choose the centers at level $i+1$ in order to cover as many points as the optimal algorithm. This produces a result that is roughly twice that of the optimal radius.

We solve this problem by creating multiple spanners with certification distances that vary, thus partitioning the interval between 2^i and 2^{i+1} (see Section 4). The algorithm is then applied to all levels of all spanners, and the set of centers with the smallest radius is chosen. The number of

---

Figure 1: An overview of the expanded greedy algorithm [7].

Given radius r, point set S, t, k, and n:
  $C_r \leftarrow \emptyset$ (C_r is the set of k centers being created for radius r)
  $V \leftarrow S$ (V is the set of candidate centers, S is the complete point set)
  for each $v \in V$
    construct G_v and E_v
    compute counts $|G_v|$ and $|E_v|$ of points in S within r and $3r$ of v
  for $j = 1$ to k and $V \neq \emptyset$
    let v_j be the $v \in V$ where $|G_v|$ counts the most uncovered points
    $C_r \leftarrow C_r \cup \{v_j\}$
    mark all points in E_v as covered
  if at least $\lceil tn \rceil$ points are covered return C_r otherwise return "failure"

spanners we create depends on ε, and there are enough so that we can make the approximation error due to this issue arbitrarily small.

All points in S are candidate centers. The initial algorithm only chooses nodes in level i as possible centers, so points are excluded from consideration. If some of these excluded centers are in the optimal solution, then the algorithm could miss a solution at radius 2^i and increase the radius.

Our solution to this problem is to consider all nodes in a level far enough below i so that the candidate centers will be sufficiently dense when we are processing nodes in level i. Since the optimal solution may still not be one of these points, we expand the radius we consider to cover all the descendants of the points in this lower level. We are now able to cover all of the points covered by an optimal solution since the optimal center is a descendant of some center in this lower level. The number of levels to descend so that the points will be sufficiently dense is chosen based on ε so that the approximation error due to this issue can be made arbitrarily small.

$|G_v|$ and $|E_v|$ are known exactly. For static points it is reasonable to keep information accurately identifying the number of points within the greedy and expanded radii of each node. However, maintaining these counts under motion would require keeping certificates between each point in S and any geometrically covering centers. This would increase the compactness and locality complexity described in Section 4.3.

To reduce compactness and locality complexity, we only keep the information maintained by the deformable spanner and any information that can be derived from that. The spanner allows us to compute points in a *fuzzy disk* in which all points within some inner distance are guaranteed to be counted and no points outside of some distance are counted. Note that this means we may cover more points than we count. Since we want to cover at least $\lceil tn \rceil$ points, this is not a problem.

We maintain these fuzzy disks without the addition of new certificates to our data structure by using the neighbor and parent-child relationships to count all descendants of nodes as specified in Section 3.3. Some of these descendants may be outside of the inner radius which we maintain, so we add enough to the final radius so that it covers all these descendants.

3.2 Preconditions

In our algorithm, we assume that we have access to the following information. In other words, we assume for now that the points are static and rely on the description of the kinetic data structure in Section 4 for proof that these values are maintained correctly.

Throughout the algorithm we use a "greedy" radius, $g_i(\varepsilon, p) = 2^i(1 + \frac{p}{s})(1 + 2^{-l+1})$ and an "expanded" radius, $e_i(\varepsilon, p) = 3 \cdot 2^i(1 + \frac{p}{s})(1 + 2^{-l+1})$. From ε we derive additional parameters $s(\varepsilon)$ (abbreviated as s) and $l(s, \varepsilon)$ (abbreviated as l) where s represents the number of spanners we maintain, and l represents the number of levels of the spanner which we will descend at each step of the algorithm in order to find candidate centers. Choices for s and l are justified in the proof to Theorem 3.2. Since ε is constant throughout the algorithm and p is kept in the general form, we abbreviate the radii as g_i and e_i. We also use slightly smaller radii $g_i^- = 2^i(1 + \frac{p}{s})(1 + 2^{-l})$ and $e_i^- = 3 \cdot 2^i(1 + \frac{p}{s})(1 + 2^{-l})$.

For all points we maintain:

- The parent/child and neighbor relationships between points in the spanner.
- The hierarchical description of the spanner, including the levels that each point exists in.

For each level i in the spanner hierarchy we maintain:

- A count of the number of points that are descendants of each node in level i.

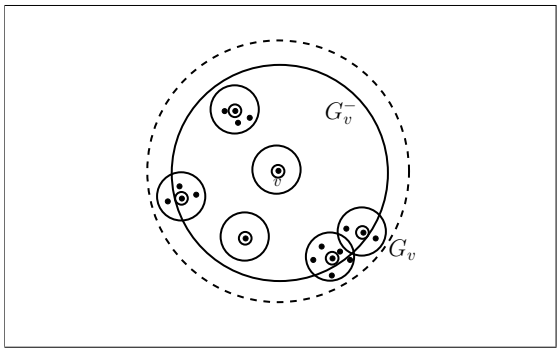

Figure 2: For $v \in S_{i-l-1}$ the range sketching query returns the nodes in S_{i-l-1} within G_v^-. These are circled. All points shown are in $D(\mu(G_v^-))$ and are counted for the priority queue.

Given node v and level $i-l-1$:
  $sum \leftarrow 0$
  $\mu(G_v^-) \leftarrow$ result of Lemma 3.3 part (i)
    applied to v
  for each $u \in \mu(G_v^-)$
    if $||uv|| > g_i^-$
      $\mu(G_v^-) \leftarrow \mu(G_v^-) \setminus u$
    else
      $sum \leftarrow sum + |D(u)|$
  return $(\mu(G_v^-), sum)$

Figure 3: The range sketching and counting subroutine for the $\mu(G_v^-)$ query. To answer the $\mu(E_v^-)$ query, all references to $\mu(G_v^-)$ change to $\mu(E_v^-)$, g_i^- to e_i^-, and Lemma 3.3 part (ii) is used.

- A count indicating the number of points within distance g_i of each node in S_{i-l-1}. This count must include all points within distance g_i^- of the node and may not include any points farther than g_i away.
- A count indicating the number of points within distance e_i of each node in S_{i-l-1}. This count must include all points within distance e_i^- of the node and may not include any points farther than e_i away.
- A priority queue associated with level i that has the nodes in S_{i-l-1} as keys and the counts of points within distance g_i as described above as values.

3.3 The Discrete Problem

Our algorithm picks a set of k centers from the given n input points for a $(3+\varepsilon)$-approximation of the optimal solution to the kinetic, robust k-center problem. It takes as input the set of n points S, ε, k, t, and α. Appropriate choices for s and l are justified in the proof to Theorem 3.2 and α is an upper bound on the aspect ratio which is assumed to be provided as part of the input.

Algorithm Overview. The algorithm iterates over all s spanners and for each level, starting from the highest and moving to S_0, calculates the k best centers for that level using the per-level subroutine described later in this section. If the k centers found for a given level i don't cover $\lceil tn \rceil$ points, the k centers found for $i+1$ are stored as representative for that spanner. The k centers with minimum radius e_i out of those s sets is output as the solution.

Range Sketching. In order to maintain the counts described as necessary preconditions, we need to answer a range sketch query. A *range sketch query* is given a node v in S_{i-l-1} and returns a set of nodes $\mu(G_v^-) \subseteq S_{i-l-1}$ such that for $u \in \mu(G_v^-)$, $D(u) \subseteq G_v$ and $G_v^- \cap S \subseteq D(\mu(G_v^-))$ where $D(X)$ is the set of descendants of a node or set of nodes X, G_v is the disk of radius g_i centered at node v, and G_v^- is the disk of radius g_i^- centered at node v. These nodes correspond roughly to the nodes of the partition tree used for approximate range searching by Arya and Mount [3]. See Figure 2 for an example range sketching query.

We answer the range sketching query by (1) identifying all nodes that *could* be in $\mu(G_v^-)$ and (2) examining these nodes individually. To determine which nodes could be in $\mu(G_v^-)$ we develop the following lemmas whose proofs have been moved to Section A due to space considerations.

```
Given node v and level i − l − 1:
    U ← overlap range sketch(v, i − l − 1)
    μ(E_v^−) ← range sketch(v, i − l − 1)
    for each u ∈ U
        (μ(G_u^−), |G_u|) ← range sketch(u, i − l − 1)
        for each w ∈ μ(E_v^−)
            if w ∈ μ(E_v^−) ∩ μ(G_u^−)
                |G_u| ← |G_u| − |D(w)|
```

Figure 4: Subroutine to update greedy disk counts.

```
Given node v and level i − l − 1:
    U ← result of Lemma 3.3 part (iii)
          applied to v
    for each u ∈ U
        if ||uv|| > 2^{i+2}(1 + 1/2^{l+1})
            U ← U \ u
    return U
```

Figure 5: The overlap range sketch subroutine.

Lemma 3.1. *For some node v in level i all points within distance 2^{i+1} are descendants of v or are descendants of one of v's neighbors.*

Lemmas 3.2 and 3.3 extend Lemma 3.1 in order to maintain information about points within specific distances of a node. We define $N^{(h)}(v)$ to be the set of nodes (at the same level as v) containing all f-fold neighbors of v for $0 \leq f \leq h$ where $h \geq 1$ is some integer.

Lemma 3.2. *For some node v in level i, all points within distance $h \cdot 2^{i+2} - 2^{i+1}$ of v are in the set $N^{(h)}(v)$.*

Lemma 3.3. *For some node v in S_{i-l-1}: (i) All points within distance g_i^- are in $N^{(2^{l-1}+3/2)}(v)$. (ii) All points within distance e_i^- are in $N^{(3\cdot 2^{l-1}+7/2)}(v)$. (iii) All points within distance $2^{i+2}(1+2^{-l+1})$ are in $N^{(2^{l+1}+9/2)}(v)$.*

Based on Lemma 3.3, the range sketching subroutine is defined in Figure 3. The count of the fuzzy disk G_v is approximated by $|D(\mu(G_v^-))|$ and determined by summing the counts indicating the number of descendants of a node. The range sketching subroutine also returns this sum.

The range sketching subroutine is used when the count of uncovered points within a disk changes. The update greedy disk counts subroutine (Figure 4) is used when an expanded disk is added to the list of centers. It does a range sketching query (Figure 5) for all nodes which could belong to a greedy disk that overlaps the changed expanded disk.

Main Subroutine and Analysis. We now have the tools needed to introduce the main subroutine for the algorithm. The per-level subroutine (presented in Figure 6) calculates the candidate list of k centers for a given spanner p and level i. It is called from the algorithm overview presented earlier.

We consider the static version of this algorithm in order to create a kinetic version; we focus on one stage of the algorithm. In the static context this algorithm requires preprocessing to create the priority queue and do range sketching to determine initial counts for G_v and E_v in all levels of all spanners. We comment that this can be done in time $O(n \log n \log \alpha/(\varepsilon^d))$ since there are n points, priority queue insertions take time $\log n$, centers are calculated for $\log \alpha$ levels, and $O(1/\varepsilon^d)$ neighbors are considered for range sketching as shown by the following lemma whose proof (which is based on a standard packing argument) has been moved to Section A.

Lemma 3.4. *There are $O(1/\varepsilon^d)$ nodes in S_{i-l-1} within E_v where $v \in S_{i-l-1}$ and E_v has radius e_i.*

Theorem 3.1. *After preprocessing, our algorithm takes time $O((sk \log \alpha \log n)/\varepsilon^d)$, which is $O((\log \alpha \log n)/\varepsilon^d)$.*

Given k, ε, t, n, S, p, and level i:
  $C_{p,i} \leftarrow \emptyset$ ($C_{p,i}$ is the set of k centers being created for spanner p and level i)
  $V \leftarrow S_{i-l-1}$ (V is the set of candidate centers, S_{i-l-1} is set of nodes in level $i-l-1$)
  for each $v \in V$
    $(\mu(G_v^-), |G_v|) \leftarrow$ range sketch(v, $i-l-1$); $G_v = D(\mu(G_v^-))$
    $(\mu(E_v^-), |E_v|) \leftarrow$ range sketch(v, $i-l-1$); $E_v = D(\mu(E_v^-))$
  for $j = 1$ to k and $V \neq \emptyset$ let v_j be the $v \in V$ with the largest $|G_v|$
    $C_{p,i} \leftarrow C_{p,i} \cup \{v_j\}$
    update greedy disk(v_j, $i-l-1$)
  if at least $\lceil tn \rceil$ points are covered return $(C_{p,i}, e_i)$

Figure 6: The per-level subroutine for spanner p and level i of the kinetic robust k-center algorithm

Proof. The per-level subroutine takes time $O((k \log n)/\varepsilon^d)$ since it repeats for k iterations and must update the counts for $O(1/\varepsilon^d)$ nodes (those within disk E_j by Lemma 3.4) in a priority queue holding at most n points. This subroutine is repeated $O(\log \alpha)$ (the number of levels in the spanner, which may vary over time) times for each of s spanners, so the total time required is $O((sk \log \alpha \log n)/\varepsilon^d)$. Since s is a function of ε and k is a constant, our algorithm takes time $O((\log \alpha \log n)/(\varepsilon^d))$. □

Theorem 3.2. *Let r_{opt} be the optimal radius for k-centers chosen from S and r_{apx} be the radius found by our algorithm, then $r_{apx} \leq (3+\varepsilon) r_{opt}$.*

Proof. Call the k optimal points $v_1, v_2, ..., v_k$. Let r_{opt} be expressed as $2^i + x$ for some integer i and $0 \leq x < 2^i$. Let the disk of radius $2^i + x$ centered at v_j be denoted by O_j.

We first show that it is *possible* for us to cover as many points as are covered by the optimal solution. The optimal centers v_j are only guaranteed to exist in level S_0, so we may not have a chance to choose v_j as our center. However, each $v_j \in S_0$ has an ancestor in level S_{i-l-1} that is within 2^{i-l} of v_j. Call this ancestor u_j; u_j will be considered by our algorithm and when G_j^- centered at u_j is considered, some G_j^- (with radius g_i^- where $\frac{p}{s} 2^i (1 + 1/2^l) \geq x$) will cover all points covered by O_j. Since u_j is in level S_{i-l-1}, it is possible for us to choose k centers which cover as many points as the optimal solution covers. Note that we described G_j^- with smaller radius g_i^- — this inner radius is sufficient to cover the optimal points and is the radius within which the preconditions guarantee that we will cover all points.

Now we show that our algorithm *will* choose k centers that cover as many points as are covered by the optimal solution. Specifically, we show that the number of points covered by $E_1 \cup E_2 \cup ... \cup E_k \geq O_1 \cup O_2 \cup ... \cup O_k$ for our chosen k points and the optimal k points. A similar theorem was proven by Charikar *et al.* [7] and our proof uses a similar charging argument which charges a point in the optimal solution either to itself or to some point in the greedy disk that was greedily chosen instead of that optimal disk. For the full proof of this charging argument, see Section A.

Our algorithm finds k points and radius e_i which cover at least as many points as the optimal solution and optimal radius would. To find the approximation ratio, we express r_{apx} in terms of $r_{opt} = 2^i + x$. Since $0 \leq x < 2^i$, there is some p for which $\frac{p-1}{s} 2^i \leq x < \frac{p}{s} 2^i$, so $\frac{p}{s} 2^i \leq x + \frac{1}{s} 2^i$, so $r_{apx} = e_i \leq 3(1 + 2^{-l+1})(2^i + x + \frac{1}{s} 2^i)$. Consider the ratio between this and $r_{opt} = 2^i + x$. We will show that this ratio is less than or equal to $3 + \varepsilon$.

Let $l = -\lceil \log(\frac{\varepsilon/6 - 1/s}{2/s + 2}) \rceil$ and $\varepsilon > 0$, then $\frac{3(1 + 2^{-l+1})(2^i + x + \frac{1}{s} 2^i)}{2^i + x} = \frac{3(1 + \frac{\varepsilon/6 - 1/s}{1/s+1})(2^i + x + \frac{1}{s} 2^i)}{2^i + x}$.

Allowing this to be as large as possible, we let $s \to \infty$ for the first multiplicand and require $s \geq \frac{6+\varepsilon}{\varepsilon}$ for the second. This gives a bound of $\frac{3(1+\frac{\varepsilon}{6})(2^i + x + \frac{\varepsilon}{6+\varepsilon}2^i)}{2^i + x} = 3 + \frac{\varepsilon 2^i + \frac{\varepsilon}{2}x}{2^i + x} \leq 3 + \frac{2^i\varepsilon + x\varepsilon}{2^i + x} = 3+\varepsilon$.

So $r_{apx}/(2^i+x) \leq 3+\varepsilon$ and the algorithm gives us a $(3+\varepsilon)$-approximation of the optimal radius. □

We present an example point set for which the algorithm's approximation ratio is tight in Section B.

3.4 The Absolute Problem

Recall that in the absolute formulation the centers may be any point in space. This algorithm is the same as for the discrete problem, except $g_i = 2^{i+1}(1+\frac{p}{s})(1+2^{-l+1})$ and $e_i = 2^{i+2}(1+\frac{p}{s})(1+2^{-l+1})$. We double the original radius g_i' so that it has the possibility of covering some optimal disk with radius r_{opt}. Since $g_i' \geq r_{opt}$, doubling g_i' allows g_i to span the distance from the optimal center to some point in S. Increasing e_i by the same amount ensures that if G_j covers part of the optimal disc then E_j expands to encompass it.

Theorem 3.3. *Let r_{opt} be the optimal radius for the absolute k-center problem and let r_{apx} be the radius found by the absolute algorithm, then $r_{apx} \leq (4+\varepsilon)r_{opt}$.*

Proof. Call the k optimal points $v_1, v_2, ..., v_k$. Let each of these points be "rounded" to some point v_j' they cover in the input set. The proof proceeds as for the discrete problem where v_j' is the optimally chosen center and g_i and e_i are as given above. These new radii fulfill the same requirements as the old: G_j centered at u_j (the ancestor of v_j') can cover all points in O_j and E_j covers all points in O_j if G_j intersects O_j. Rounding v_j to v_j' adds r_{opt} to r_{apx}, so in the absolute case we obtain a $(4+\varepsilon)$-approximation. □

4 Kinetic Spanner Maintenance and Quality

4.1 Certificates

The Gao, Guibas, and Nguyen [9] deformable spanner maintains four types of certificates. *Parent-child certificates* guarantee that a child in level i is within distance 2^{i+1} of its parent. *Edge certificates* guarantee that neighbors in level i are within distance $c \cdot 2^i$ of each other (where $c = 4+\frac{16}{\varepsilon}$) [9]. *Separation certificates* guarantee that neighbors in level i are at least 2^i away from each other. *Potential neighbor certificates* guarantee that two points have parents which are connected by an edge (i.e. are cousins). Potential neighbor certificates are maintained so that parent-child certificates can be easily maintained [9]. Recall that in our data structure we have multiple spanners with differing base distances b. For $b = 1$, the distances that are certified are the same as in the deformable spanner, otherwise the distances are multiplied by b. For the remainder of this section we describe the data structure when $b = 1$.

We add to the update rules for parent-child, edge, and separation certificates, keeping the previous rules and adding the following for these actions:

Addition of a spanner edge. Increment the counts for G_v and E_v for all neighbors with counts that are affected according to Lemma 3.3 parts (i) and (ii) and update the priority queue appropriately.

Deletion of a spanner edge. Decrement the counts for G_v and E_v for all neighbors with counts that are affected according to Lemma 3.3 parts (i) and (ii) and update the priority queue appropriately.

Addition of parent-child certificate. Increment the count for all new ancestor nodes and for all counts for G_v and E_v for neighbors with counts that are affected according to Lemma 3.3 parts (i) and (ii) and update the priority queue appropriately.

Failure of parent-child certificate. Decrement the count for all new ancestor nodes and for all counts for G_v and E_v for neighbors with counts that are affected according to Lemma 3.3 parts (i) and (ii) and update the priority queue appropriately.

At each level i, a list of k-centers is maintained as well as a priority queue. The entries in this priority queue are nodes v in S_{i-l-1} and the values are the counts of points with G_v. For each of these updates, any time a count increases, if that node is in the list of k-centers only the counts and the priority queue are updated, otherwise the solution at that level is recalculated. Any time a count decreases, if that node is in the list of k-centers then the solution at that level is recalculated, otherwise only the counts and priority queue are updated. In the cases when recalculating the solution is necessary, our static algorithm is applied.

4.2 Preconditions

For all points. The preconditions needed for all points are maintained by the certificates and update conditions given in the original deformable spanner [9].

For each level i in the spanner hierarchy. In order to maintain the precondition counts necessary to the statically explained algorithm from Section 3.2, we maintain a count for each node in the spanner hierarchy of the number of points which are descendants of that node. These counts are updated when parent-child certificates are added or fail. This count is necessary to maintain the preconditions but not sufficient, since descendants of a node must not be farther than 2^{i+1} from a node in level i [9], but there may be points within 2^{i+1} that are not descendants of that node. To ensure that we have counted all points within g_i and e_i we refer to Section 3.3 and use Lemma 3.3 parts (i) and (ii) to determine which neighbors to consider. Using this lemma and the count maintained at the root of each subtree, we maintain the number of points within G_v and E_v for some node v. The count of points within the fuzzy disk G_v is maintained in a priority queue. When a G_v count changes, the priority queue is updated accordingly. If this update occurs during the course of the algorithm because a center was chosen and nodes were marked as covered, these changes are kept track of until all centers are chosen. The changes are undone in reverse order so that the counts are correct. The preconditions for each level i are maintained according to the update rules given in Section 4.1.

4.3 Quality

In order to assure the quality of the spanner, we must reason about following actions. Recall that n is the total number of points, d the dimension, and α the user given upper bound on the aspect ratio. Compactness and locality conditions ensure that maintaining certificates for the kinetic data structure is not too costly by bounding the number of certificates. Compactness bounds the total number of certificates and locality bounds the number of certificates in which each point can participate. The efficiency condition ensures that maintaining the kinetic data structure is not too expensive by bounding the number of certificate failures that can occur. This is compared to the number of required changes to the combinatorial structure of the spanner to determine if the number of certificate failures is reasonable. These bounds are, respectively, $O(\frac{n}{\varepsilon^d})$, $O(\log\alpha/(\varepsilon^d))$, and $O(n^2\log\alpha)$ and match those given by Gao *et al.* [9]. For further explanation see Section C.

The responsiveness condition ensures that maintaining the kinetic data structure is not too expensive by bounding the amount of time taken to repair failed certificates. Our spanner satis-

fies responsiveness with $O((\log \alpha \log n)/(\varepsilon^d))$ time per certificate update. This time is due to the possibility that a failure or addition of a certificate could require the algorithm to be re-run. See Section C for proof of this assertion.

References

[1] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. In *Algorithmica*, volume 33, pages 201–226, 2002.

[2] G. Albers, Leonidas J. Guibas, Joseph S. B. Mitchell, and T. Roos. Voronoi diagrams of moving points. In *International Journal of Computational Geometry Applications*, volume 8, pages 365–380, 1998.

[3] Sunil Arya and David M. Mount. Approximate range searching. In *Computational Geometry: Theory and Applications*, volume 17, pages 135–163, 2000.

[4] M. J. Atallah. Some dynamic computational geometry poblems. In *Comput. Math. Appl*, volume 11(12), pages 1171–1181, 1985.

[5] Basch, Guibas, and Hershberger. Data structures for mobile data. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1997.

[6] J. Basch, L. J. Guibas, and L. Zhang. Proximity problems on moving points. In *Proceedings of the 13th Annual ACM symposium on Computational Geometry*, pages 344–351, 1997.

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

[8] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 434–444, 1988.

[9] Jie Gao, Leonidas J. Guibas, and A. Nguyen. Deformable spanners and applications. In *Proc. of the 20th ACM Symposium on Computational Geometry (SoCG'04)*, pages 179–199, 2004.

[10] T. Gonzalez. Clustering to minimize the maximum intercluster distance. In *Theoretical Computer Science*, volume 38, pages 293–306, 1985.

[11] L. Guibas. Kinetic data structures. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, pages 23–1–23–18. Chapman and Hall/CRC, 2004.

[12] Leonidas J. Guibas. Kinetic data structures: A state of the art report. 3rd Workshop on Algorithmic Foundations of Robotics, P. K. Agarwal, L. Kavraki, and M. Mason, editors, A. K. Peters, 1998.

[13] P. Gupta, R. Janardan, and M. Smid. Fast algorithms for collision and proximity problems involving moving geometric objects. In *Comput. Geom. Theory Appl*, volume 6, pages 371–391, 1996.

[14] D. S. Hochbaum and D. B. Shmoys. A best possible approximation algorithm for the k-center problem. In *Mathematics of Operations Research*, volume 10(2), pages 180–184, 1985.

[15] Dorit Hochbaum, editor. *Approximation Algorithms for NP-hard Problems.* PWS publishing company, Boston, 1995.

[16] O. Kariv and S.L. Hakimi. An algorithmic approach to network location problems. In *Part 1: The p-centers, SIAM Journal on Applied Mathematics*, volume 37, pages 513–538, 1979.

[17] J. Plesnik. A heuristic for the p-center problem in graphs. In *Discrete Applied Mathematics*, volume 17, pages 263–268, 1987.

[18] P. J. Rousseeuw and A. Leroy. *Robust Regression and Outlier Detection.* New York: Wiley, 1987.

[19] Elmar Schomer and Christian Theil. Efficient collision detection for moving polyhedra. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 51–60, 1995.

[20] Elmar Schomer and Christian Theil. Subquadratic algorithms for the general collision detection problem. In *Abstracts 12th European Workshop Comput. Geom*, pages 95–101, 1996.

A Lemma and Theorem Proofs From Section 3.3

Proof of Lemma 3.1. Consider v and the neighbors of v, denoted $N(v)$. By the construction of the spanner and the maintenance of the edge certificates and separation certificates, all points in $N(v)$ are less than distance $c \cdot 2^i$ and greater than distance 2^i away from v where Gao *et al.* [9] show that $c = 4 + 16/\varepsilon$ ensures a $(1+\varepsilon)$-spanner. All nodes that are less than distance $2^{i+2} + \frac{16}{\varepsilon}2^i$ away from v are in $N(v)$, so the closest a non-neighboring point could be to v is 2^{i+2} away. Since the farthest a descendant can be from its ancestor in level i is 2^{i+1} away, such a non-neighboring point can't have descendants within 2^{i+1} of v. So all points within distance 2^{i+1} of v are descendants of v or are descendants of some node in $N(v)$. □

Proof of Lemma 3.2. By Lemma 3.1 each composition forces a distance of 2^{i+2} to the nearest non-neighbor node, or a total distance of $h \cdot 2^{i+2}$. Due to the descendants of the nearest non-neighbor node, all points within $h \cdot 2^{i+2} - 2^{i+1}$ are in $N^{(h)}(v)$. □

Proof of Lemma 3.3. Each distance can be rewritten in the following way:

(*i*) $g_i^- = 2^i(1 + 2^{-l+1}) = (2^{l-1} + 3/2)2^{(i-l-1)+2} - 2^{(i-l-1)+1}$

(*ii*) $e_i^- = 3 \cdot 2^i(1 + 2^{-l+1}) = (3 \cdot 2^{l-1} + 7/2)2^{(i-l-1)+2} - 2^{(i-l-1)+1}$

(*iii*) $2^{i+2}(1 + 2^{-l+1}) = (2^{l+1} + 9/2)2^{(i-l-1)+2} - 2^{(i-l-1)+1}$

The proof follows from Lemma 3.2. □

Proof of Lemma 3.4. We say that a point set is *δ-sparse* if every pair of points from the set is separated by a distance of at least δ. Given a circle of radius r in the plane, it follows from a standard packing argument that any δ-sparse set that lies entirely within this circle has cardinality $O(1 + (r/\delta)^2)$. Since, by the hierarchy construction, nodes in S_{i-l-1} are of distance at least 2^{i-l-1}

away from each other, S_{i-l-1} is 2^{i-l-1}-sparse. So the number of nodes in S_{i-l-1} within E_v is $O(1 + (2^i(1 + 2^{-l+1})/2^{i-l-1})^d)$ which is $O(1/\varepsilon^d)$ □

Charging Argument for the Proof of Theorem 3.2. Here we show that our algorithm *will* choose k centers that cover as many points as are covered by the optimal solution. Specifically, we show that the number of points covered by $E_1 \cup E_2 \cup ... \cup E_k \geq O_1 \cup O_2 \cup ... \cup O_k$ for our chosen k points and the optimal k points. A similar theorem was proven by Charikar *et al.* [7] and our proof uses a similar charging argument. Let the *size of a disk* D be defined as the number of points which it covers and denoted $|D|$. Let the *size of a collection of disks* $\{D_1, D_2, ..., D_n\}$ be defined as the number of points which the union of those disks covers and denoted $|D_1, D_2, ..., D_n|$. We show that it is possible to order the optimum disks O_j such that for each j, $|O_1, O_2, ..., O_j| \leq |E_1, E_2, ..., E_j|$. We prove this by induction on j and use a charging argument that assigns each point in $\{O_1, O_2, ..., O_j\}$ to some point in $\{E_1, E_2, ..., E_j\}$.

For $j = 1$, we know by greediness that $|E_1| \geq |O_1|$. Assuming by induction that $|O_1, O_2, ..., O_{j-1}| \leq |E_1, E_2, ..., E_{j-1}|$, we consider G_j.

- If G_j intersects any of the remaining optimal disks, let O_j be that disk. If we expand distances g_i to e_i we cover all of O_j (which has radius $2^i + x$) for some p. Since r_{apx} will equal the smallest radius such that the threshold is still covered, it is possible to obtain a solution which covers all points covered by the optimal solution, so we consider the p such that all points in O_j are covered by E_j. We charge each point in O_j to itself.

- If G_j does not intersect any of the remaining O_j, then let O_j be the remaining optimal disk with the greatest size. Since we chose G_j greedily, $|G_j| \geq |O_j|$. We charge each point in O_j to a point in G_j. Since G_j is disjoint from all other O_j, no point in G_j will be charged to twice.

Points are either charged to themselves or to some point in an E_j which is disjoint from all O_j and therefore only used once. So no point in an E_j is charged to more than once. Every point in every O_j is accounted for by one of these cases, so each point in the optimal solution is charged to some point in the algorithm's solution. So $|O_1, O_2, ..., O_k| \leq |E_1, E_2, ..., E_k|$. □

B Example $(3 + \varepsilon)$-approximation Point Set

We present an example in order to justify that the discrete $(3 + \varepsilon)$-approximation ratio is a tight upper bound for our algorithm. Figures 7 and 8 give an example point set with $k = 3$ and $t = 21/23$. Here we will run the basic steps of the algorithm on this point set.

We start with $i = \log 2^5 = 5$, $l = 2$, $\varepsilon = 3/4$, $s = 10$, and $p = 0$. All options for $5 \geq i \geq 3$ are trivial since all points are easily covered for any greedy choices given the expanded radius.

Consider $i = 2$ with points at level S_0 and radius of $2^2(1 + 1/2)$. We greedily choose the node farthest to the left marked with a cross as our first center and cover all 9 points within $3 \cdot 2^2(1+1/2)$ of it. We then greedily choose the node in the middle as shown in Figure 7. Note that this is the first choice which deviates from the optimal solution. We cover all 13 points within $3 \cdot 2^2(1 + 1/2)$ of this point which have not earlier been covered. Finally we greedily choose the single point on the right of the point set.

This solution at $i = 2$ satisfies our restrictions for k and t so we consider $i = 1$. We greedily choose any circle that covers two points, since the points are far enough apart so that with circles of radius $2^1(1+1/2)$ this is the best that we can do. We then cover all points within distance $3 \cdot 2^1(1+$

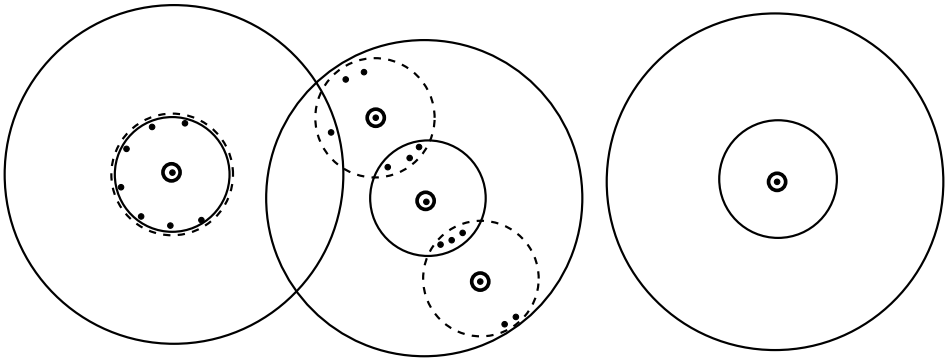

Figure 7: This is an example of a set of points for which our algorithm gives an approximation ratio of 3 ($k = 3$, $t = 21$ of 23 points). If greedy disks are expanded by a ratio of $R < 3$ this example is still forced to give an approximation ratio of 3. Gray circles represent the outer limit of the optimal disks. Black circles represent greedy disks and their corresponding expanded disks. Circled points are nodes in level S_2. The centers of the greedy disks were chosen in order from left to right.

1/2) of these nodes. There are more than three nodes that can be chosen so that the inner circle covers two points, so we always choose such a node. However none of these points lie in the circular pattern of 8 points on the left. These points are farther than $3 \cdot 2^1(1+1/2)$ away from the centers which are chosen. So there are at least 8 points which are not covered by this size radius. This does not satisfy our threshold t, so the algorithm returns the previously found set of centers with radius $3 \cdot 2^2(1+1/2)$. The optimal radius for this point set is 2^2, so the algorithm produced a $(3+\varepsilon)$-approximation.

We then run the algorithm 9 more times with p set from 1 to 9 and take the minimum radius output. The minimum output will be the radius when $p = 0$, since for $p = 0$ the points covered in the algorithm's solution are right on the perimeter of the covered space, so expanding the radius to cover a larger space will not capture more points and will only increase the radius that is output.

C Kinetic Spanner Quality Details

C.1 Compactness and Locality

Compactness and locality conditions ensure that maintaining certificates for the kinetic data structure is not too costly by bounding the number of certificates. Compactness bounds the total number of certificates and locality bounds the number of certificates in which each point can participate.

Theorem C.1. *Our KDS satisfies compactness and locality with $O(\frac{n}{\varepsilon^d})$ total certificates and $O(\log \alpha/(\varepsilon^d))$ certificates per point.*

Proof. Before our modifications, the deformable spanner had $O(\frac{n}{\varepsilon^d})$ total certificates and $O(\log \alpha/(\varepsilon^d))$ certificates per point [9]. No certificates were added or removed, so these bounds remain the same. □

C.2 Efficiency

The efficiency condition ensures that maintaining the kinetic data structure is not too expensive by bounding the number of certificate failures that can occur. This is compared to the number

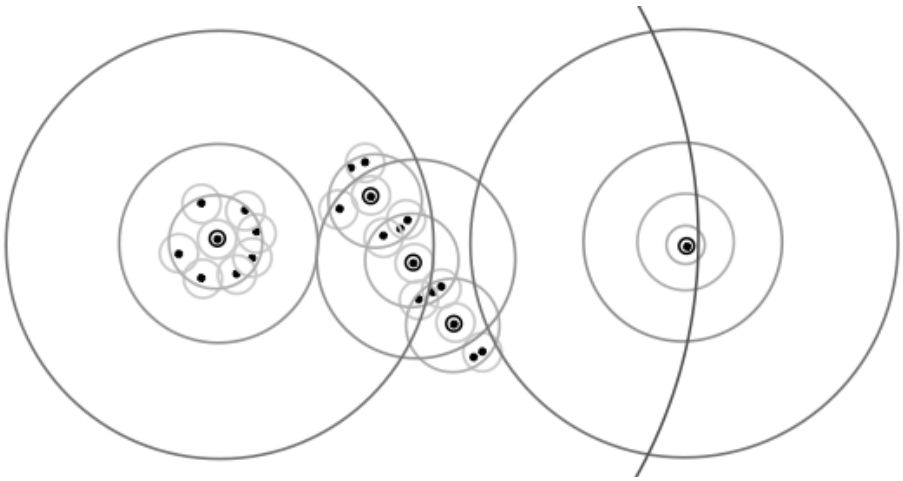

Figure 8: Heirarchy of points from Figure 7 when put into a deformable spanner [9]. Darker, larger circles represent higher levels of the spanner. The circle at level S_5 (the highest level in this spanner) is only partially shown. The circles at level S_0 which only contain individual points are not shown. Circled points are nodes in level S_2.

---

of required changes to the combinatorial structure of the spanner to determine if the number of certificate failures is reasonable.

Theorem C.2. *Our KDS satisfies efficiency with $O(n^2 \log \alpha)$ possible certificate maintenance events.*

Proof. The deformable spanner had $O(n^2 \log \alpha)$ possible maintenance events because, under pseudo-algebraic motion, events only occurred when the distance between two points was at the boundary of some certificate on a given level i, namely 2^i or $c \cdot 2^i$. Since there are $\log \alpha$ possible levels and $2 \cdot n^2$ of these inter-point distances, there were $O(n^2 \log \alpha)$ possible maintenance events [9]. Our spanners change these base distances, but the number of maintenance events stays the same. Since the spanner has not changed except for the base distance, the number of changes required by the combinatorial structure of the spanner remains $\Omega(n^2)$ [9], so any approach based on a spanner requires $\Omega(n^2)$ changes. So the kinetic data structure is efficient. □

C.3 Responsiveness

The responsiveness condition ensures that maintaining the kinetic data structure is not too expensive by bounding the amount of time taken to repair failed certificates.

Theorem C.3. *Our KDS satisfies responsiveness with $O((\log \alpha \log n)/(\varepsilon^d))$ time per certificate update.*

Proof. Before our modifications, the spanner could be updated in $O(1)$ or $O(\frac{\log \alpha}{\varepsilon^d})$ time [9]. Now, the failure or addition of a certificate could require our algorithm to be re-run. When the priority queue is updated during the re-run, a list of changes is maintained. After k centers for a level are chosen, the priority queue is returned to its original state (the state assuming all points are uncovered). Since the changes made are undone, this takes only a constant factor of extra time.

Re-caluclating solution takes the same time as running the per-level subroutine (see Section 3.3) that takes $O((k \log n \log \alpha)/\varepsilon^d)$ which is $O((\log \alpha \log n)/(\varepsilon^d))$. Note that it is possible for the recalculation of the k centers to be forced any time a point moves. The failure or addition of a certificate could also require points to be updated for $O(1/\varepsilon^d)$ nodes (see Lemma 3.4) on $\log \alpha$ levels. The total time a certificate could take to be repaired is $O((\log \alpha \log n)/(\varepsilon^d))$. □

D Non-Robust Kinetic K-Center Algorithm

In this section we mention that using the structure presented in Section 2 and refined throughout the paper, it is possible to improve on the non-robust discrete k-center algorithm presented by Gao *et al.* [9]. That algorithm achieved an 8-approximation and we improve that to a $(4+\varepsilon)$-approximation (both algorithms are for arbitrary k). This improvement follows the algorithm presented in the Gao *et al.* paper, but performs the algorithm on all s spanners simultaneously and chooses the minimum radius found as the output of the algorithm (shown in Figure 9). The value of s is restricted in the proof of Theorem D.1. For proof of maintenance under motion, we refer the reader to the proof of a similar algorithm given by Gao *et al.* [9].

```
Given k, S, α, and ε:
for p = 1 to s
    for i = log α to 0
        if |S_i| > k
            r_p ← 2^{i+2}(1 + p/s)
            K_p ← S_{i+1}
            if |S_{i+1}| < k
                add arbitrary children of nodes in S_{i+1} to K_p
            break out of inner loop
  r ← minimum r_p
  output (K_p, r), each point is serviced by its ancestor in K_p
```

Figure 9: The non-robust kinetic k-center algorithm for arbitrary k.

Theorem D.1. *Let r_{opt} be the optimal radius for k-centers chosen from the input points and r_{apx} be the radius found by our non-robust kinetic k-center algorithm, then $r_{apx} \leq (4+\varepsilon) r_{opt}$.*

Proof. The algorithm will choose some S_i on spanner p with associated radius $2^{i+1}(1+\frac{p}{s})$ where S_i has the minimum radius such that $|S_i| \geq k$. Since $2^{i+1}(1+\frac{p-1}{s}) < 2^{i+1}(1+\frac{p}{s})$, $|S_i| > k$ on spanner $p-1$. So at least two points from S_i are assigned to the same center in the optimal solution. These points are of distance at least $2^i(1+\frac{p-1}{s})$ apart from each other, so the optimal radius must be at least $2^{i-1}(1+\frac{p-1}{s})$. To determine the approximation ratio we consider the ratio between $r_{opt} \geq 2^{i-1}(1+\frac{p-1}{s})$ and $r_{apx} \leq 2^{i+1}(1+\frac{p}{s})$. Choose $s \geq \frac{2}{\varepsilon} + \frac{1}{2}$ and the algorithm is determined to be a $(4+\varepsilon)$-approximation algorithm. □