Fully Dynamic k-Center Clustering in Doubling Metrics*

Gramoz Goranci† Monika Henzinger‡ Dariusz Leniowski § Christian Schulz ¶

Alexander Svozil‖

Abstract

Clustering is one of the most fundamental problems in unsupervised learning with a large number of applications. However, classical clustering algorithms assume that the data is static, thus failing to capture many real-world applications where data is constantly changing and evolving. Driven by this, we study the metric k-center clustering problem in the fully dynamic setting, where the goal is to efficiently maintain a clustering while supporting an intermixed sequence of insertions and deletions of points. This model also supports queries of the form (1) report whether a given point is a center or (2) determine the cluster a point is assigned to.

We present a deterministic dynamic algorithm for the k-center clustering problem that provably achieves a $(2+\epsilon)$-approximation in poly-logarithmic update and query time, if the underlying metric has bounded doubling dimension, its aspect ratio is bounded by a polynomial and ϵ is a constant. An important feature of our algorithm is that the update and query times are independent of k. We confirm the practical relevance of this feature via an extensive experimental study which shows that for values of k and ϵ suggested by theory, our algorithmic construction outperforms the state-of-the-art algorithm in terms of solution quality and running time.

*The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

†University of Toronto, Canada

‡University of Vienna, Austria

§University of Vienna, Austria

¶University of Vienna, Austria

‖University of Vienna, Austria

1 Introduction

The massive increase in the amount of data produced over the last few decades has motivated the study of different tools for analysing and computing specific properties of the data. One of the most extensively studied analytical tool is clustering, where the goal is to group the data into clusters of "close" data points. Clustering is a fundamental problem in computer science and it has found a wide range of applications in unsupervised learning, classification, community detection, image segmentation and databases (see e.g. [9, 25, 27]).

A natural definition of clustering is the *k-center clustering*, where given a set of n points in a metric space and a parameter $k \leq n$, the goal is to select k designated points, referred to as *centers*, such that their *cost*, defined as the maximum distance of any point to its closest center, is minimized. As finding the optimal k-center clustering is NP-hard [20], the focus has been on studying the approximate version of this problem. For a parameter $\alpha \geq 1$, an α-approximation to the k-center clustering problem is an algorithm that outputs k centers such that their cost is within α times the cost of the optimal solution. There is a simple 2-approximate k-center clustering algorithm by Gonzalez [12] that runs in $O(nk)$ time; repeatedly pick the point furthest away from the current set of centers as the next center to be added. The problem of finding a $(2-\epsilon)$-approximate k-center clustering is known to be NP-complete [12].

In many real-world applications, including social networks and the Internet, the data is subject to frequent updates over time. For example, every second about thousands of Google searches, YouTube video uploads and Twitter posts are generated. However, most of the traditional clustering algorithms are not capable of capturing the dynamic nature of data and often completely reclustering from scratch is used to obtain desirable clustering guarantees.

To address the above challenges, in this paper we study a *dynamic* variant of the k-center clustering problem, where the goal is to maintain a clustering with small approximation ratio while supporting an intermixed update sequence of insertions and deletions of points with a small time per update. Additionally, for any given point we want to report whether this point is a center or determine the cluster this point is assigned to. When only insertions of points are allowed, also known as the *incremental* setting, Charikar et al. [4] designed an 8-approximation algorithm with $O(k \log k)$ amortized time per point insertion. This result was later improved to a $(2+\epsilon)$-approximation by McCutchen and Khuller [23]. Recently, Chan et al. [2] studied the model that supports both point insertions and deletions, referred to as the *fully-dynamic* setting. Their dynamic algorithm is randomized and achieves a $(2+\epsilon)$-approximation with $O(k^2 \cdot \epsilon^{-1} \cdot \log \Delta)$ update time per operation, where Δ is the aspect ratio of the underlying metric space.

It is an open question whether there are fully-dynamic algorithms that achieve smaller running time (ideally independent of k) while still keeping the same approximation guarantee. We study such data structures for metrics spaces with "limited expansion". More specifically we consider the well-studied notion of doubling dimension. The *doubling dimension* of a metric space is bounded by κ if any ball of radius r in this metric can be covered by 2^κ balls of radius $r/2$ [22]. This notion can be thought of as a generalization of the Euclidean dimension since $\mathbb{R}^d$ has doubling dimension $\Theta(d)$.

The k-center clustering problem has been studied in the low dimensional regime from both the static and dynamic perspective. Feder and Greene [7] showed that if the input points are taken from $\mathbb{R}^d$, there is a 2-approximation to the optimal clustering that can be implemented in $O(n \log k)$ time. They also showed that computing an approximation better than 1.732 is NP-hard, even when restricted to Euclidean spaces. For metrics of bounded doubling dimension Har-Peled and Mendel [16] devised an algorithm that achieves a 2-approximation and runs in $O(n \log n)$ time. In the dynamic setting, Har-Peled [15] implicitly gave a fully-dynamic algorithm for metrics with bounded doubling dimension that reports a $(2+\epsilon)$-clustering at any time while supporting insertion and deletions of points in $O(\text{poly}(k, \epsilon^{-1}, \log n))$ time, where $\text{poly}(\cdot)$ is a fixed-degree polynomial in the input parameters.

One drawback shared by the above dynamic algorithms for the k-center clustering is that the update

time is dependent on the number of centers k. This is particularly undesirable in the applications where k is relatively large. For example, one application where this is justified is the distribution of servers on the Internet, where thousands of servers are heading towards millions of routers. Moreover, this dependency on k seems inherent in the state-of-the-art dynamic algorithms; for example, the algorithm due to Chan et al. [2] requires examining the set of current centers upon insertion of a point, while the algorithm due to Har-Peled [15] employs the notion of *coresets*, which in turn requires dependency on the number of centers.

In this paper we present a dynamic algorithm for metrics with bounded doubling dimension that achieves a $(2+\epsilon)$ approximation ratio for the k-center clustering problem (thus matching the approximation ratio of the dynamic algorithm in general metric spaces [2]) while supporting insertions and deletion of points in time *independent* of the number of centers k and poly-logarithmic in the aspect ratio Δ. Our algorithm is deterministic and thus works against an adaptive adversary.

Theorem 1.1. *There is a fully-dynamic algorithm for the k-center clustering problem, where points are taken from a metric space with doubling dimension κ, such that any time the cost of the maintained solution is within a factor $(2+\epsilon)$ to the cost of the optimal solution and the insertions and deletions of points are supported in $O(2^{O(\kappa)} \log\Delta \log\log\Delta \cdot \epsilon^{-1} \ln\epsilon^{-1})$ update time. For any given point, queries about whether this point is a center or reporting the cluster this point is assigned to can be answered in $O(1)$ and $O(\log\Delta)$, respectively.*

We perform an extensive experimental study of a variant of our algorithm, where we replace navigating nets with the closely related notion of *cover trees*, one of the pioneering data-structures for fast nearest-neighbour search [1, 21]. We compare our results against the implementation of [2], which is the state-of-the-algorithm for the problem. Our findings indicate that our algorithm has a significant advantage in running time and solution quality for larger values of k and ϵ, as suggested by our theoretical results.

Related work. For an in-depth overview of clustering and its wide applicability we refer the reader to two excellent surveys [25, 14]. Here we briefly discuss closely related variants of the k-center clustering problem such as the kinetic and the streaming model. In the kinetic setting, the goal is to efficiently maintain a clustering under the continuous motion of the data points. Gao et al. [11] showed an algorithm that achieves an 8-approximation factor. Their result was subsequently improved to a $(4+\epsilon)$ guarantee by Friedler and Mount [10]. In the streaming setting Cohen-Addad et al. [6] designed a $(6+\epsilon)$-approximation algorithm with an expected update time of $O(k^2 \cdot \epsilon^{-1} \cdot \log\Delta)$. However, their algorithm only works in the sliding window model and does not support arbitrary insertions and deletions of points. [18] studied streaming algorithms for a generalization of the k-center clustering problem, known as the Matroid Center.

Recently and independently of our work, Schmidt and Sohler [26] gave an 16-approximate fully-dynamic algorithm for the *hierarchical* k-center clustering with $O(\log\Delta \log n)$ and $O(\log^2\Delta \log n)$ expected amortized insertion and deletion time, respectively, and $O(\log\Delta + \log n)$ query time, where points come from the discrete space $\{1,\ldots,\Delta\}^d$ with d being a constant. This result implies a dynamic algorithm for the k-center clustering problem with the same guarantees. In comparison with our result, our algorithm (i) achieves a better and an almost tight approximation, (ii) is deterministic and maintains comparable running time guarantees, and (iii) applies to any metric with bounded doubling dimension. For variants of facility location and k-means clustering, Cohen-Addad et al. [5] obtained fully dynamic algorithms with non-trivial running time and approximation guarantees for general metric spaces.

There has been growing interest in designing provably dynamic algorithms for graph clustering problems with different objectives. Two recent examples include works on dynamically maintaining expander decompositions [24] and low-diameter decompositions [8]. For applications of such algorithms we refer the reader to these papers and the references therein.

Technical overview. In the static setting, a well-known approach for designing approximation algorithms for the k-center clustering problem is exploiting the notion of r-nets. Given a metric space (M, d), and an integer parameter $r \geq 0$, an *r-net* Y_r is a set of points, referred to as *centers*, satisfying (a) the *covering* property, i.e., for every point $x \in M$ there exists a point $y \in Y_r$ within distance at most $2 \cdot (1+\epsilon)^r$ and (b) the *separating* property, i.e., all distinct points $y, y' \in Y_r$ are at distance strictly larger than $2 \cdot (1+\epsilon)^r$. Restricting the set of possible radii to powers of $(1+\epsilon)$ in (M, d) allows us to consider only $O(\epsilon^{-1} \cdot \log \Delta)$ different r-nets, where Δ is the *aspect ratio*, defined as the ratio between the maximum and the minimum pair-wise distance in (M, d). The union over all such r-nets naturally defines a hierarchy Π. It can be shown that the smallest r in Π such that the size of the r-net Y_r is at most k yields a feasible k-center clustering whose cost is within $(2+\epsilon)$ to the optimal one (see e.g., [2]).

A natural attempt to extend the above static algorithm to the incremental setting is to maintain the hierarchy Π under insertions of points. In fact, Chan et al. [2] follow this idea to obtain a simple incremental algorithm that has a linear dependency on the number of centers k. We show how to remove this dependency in metrics with bounded doubling dimension and maintain the hierarchy under deletion of points. Concretely, our algorithm employs *navigating nets*, which can be thought of as a union over slightly modified r-nets with slightly larger constants in the cover and packing properties. Navigating nets were introduced by Krauthgamer and Lee [22] to build an efficient data-structure for the nearest-neighbor search problem. We observe that their data-structure can be slightly extended to a dynamic algorithm for the k-center clustering problem that achieves an 8-approximation with similar update time guarantees to those in [22]. Following the work of McCutchen and Khuller [23], we maintain a carefully defined collection of navigating nets, which in turn allow us to bring down the approximation factor to $(2+\epsilon)$ while increasing the running time by a factor of $O(\epsilon^{-1} \ln \epsilon^{-1})$.

Similar hierarchical structures have been recently employed for solving the dynamic sum-of-radii clustering problem [17] and the dynamic facility location problem [13]. In comparison to our result that achieves a $(2+\epsilon)$-approximation, the first work proves an approximation factor that has an exponential dependency on the doubling dimension while the second one achieves a very large constant. Moreover, while our data-structure supports arbitrary insertions of points, both works support updates only to a *specific* subset of points in the metric space.

2 Preliminaries

In the k-center clustering problem, we are given a set M of points equipped with some metric d and an integer parameter $k > 0$. The goal is to find a set $C = \{c_1, \ldots, c_k\}$ of k points (centers) so as to minimize the quantity $\phi(C) = \max_{x \in S} d(x, C)$, where $d(x, C) = \min_{c \in C} d(x, c)$. Let OPT denote the cost of the optimal solution.

In the *dynamic* version of this problem, the set M evolves over time and queries can be asked. Concretely, at each timestep t, either a new point is added to M, removed from M or one of the following queries is made for any given point $x \in M$: (i) decide whether x is a center in the current solution, and (ii) find the center c to which x is assigned to. The goal is to maintain the set of centers C after each client update so as to maintain a small factor approximation to the optimal solution.

Let $d_{\min}$ and $d_{\max}$ be lower and upper bounds on the minimum and the maximum distance between any two points that are ever inserted. For each $x \in M$ and radius r, let $B(x, r)$ be the set of all points in M that are within distance r from x, i.e., $B(x, r) := \{y \in M \mid d(x, y) \leq r\}$.

The metric spaces that we consider throughout satisfy the following property.

Definition 2.1 (Doubling Dimension)**.** *The doubling dimension of a metric space (M, d) is said to be bounded by κ if any ball $B(x, r)$ in (M, d) can be covered by 2^κ balls of radius $r/2$.*

3 Fully dynamic k-center clustering using navigating nets

In this section, we present a fully-dynamic algorithm for the k-center clustering problem that achieves a $(2+\epsilon)$-approximation with a running time not depending on the number of clusters k. Our construction is based on navigating nets of Krauthgamer and Lee [22] and a scaling technique of McCutchen and Khuller [23].

We start by reviewing some notation from [22].

r-nets and navigating nets. Let (M, d) be a metric space. For a given parameter $r > 0$, a subset $Y \subseteq M$ is an *r-net* of M if the following properties hold:

1. (separating) For every $x, y \in Y$ we have that $d(x, y) \geq r$ and,
2. (covering) $M \subseteq \bigcup_{y \in Y} B(y, r)$.

Let $\alpha > 1$ be a constant and let $\Gamma := \{\alpha^i : i \in \mathbb{Z}_+\}$ be a set of *scales*. Let $Y_r := M$ for all $r \leq d_{\min}$, and for all $r \in \Gamma$, define Y_r to be an r-net of $Y_{r/\alpha}$. A *navigating net* Π is defined as the union of all Y_r for all $r \in \Gamma$. We refer to the elements in Y_r as *centers*.

Note that for every scale $r > d_{\max}$ the set Y_r contains only one element due to the separating property. A navigating net Π keeps track of (i) the smallest scale $r_{\max}$ defined by $r_{\max} = \min\{r \in \Gamma \mid \forall r' \geq r, |Y_{r'}| = 1\}$, and (ii) the largest scale $r_{\min}$ defined by $r_{\min} = \max\{r \in \Gamma \mid \forall r' \leq r, Y_{r'} = M\}$. All scales $r \in \Gamma$ such that $r \in [r_{\min}, r_{\max}]$ are referred to as *nontrivial* scales.

3.1 Navigating nets with differing base distances

In what follows, we describe how to obtain a $(2+\epsilon)$-approximation for the k-center clustering problem by maintaining navigating nets in parallel. This technique was originally introduced by McCutchen and Khuller [23] for improving the approximation ratio of the incremental doubling algorithm for the k-center problem due to Charikar et al. [4].

The key idea behind the construction is that instead of maintaining one navigating net, we maintain m navigating nets with differing base distances. The navigating nets differ *only* in the corresponding set Γ which is used to define them. More concretely, for each integer $1 \leq p \leq m$, let $\Gamma^p = \{\alpha^{i+(p/m)-1} \mid i \in \mathbb{Z}_+\}$.

Let $Y^p_r := M$ for all $r \leq d_{\min}$ and for all $r \in \Gamma^p$, let Y^p_r be an r-net of $Y^p_{r/\alpha}$. A *navigating net* Π^p is defined as the union over all Y^p_r for $r \in \Gamma^p$. Similarly, we maintain $r^p_{\max}$ and $r^p_{\min}$, such that $r^p_{\max} = \min\{r \in \Gamma^p \mid \forall r' \geq r, |Y^p_{r'}| = 1\}$ and $r^p_{\min} = \max\{r \in \Gamma^p \mid \forall r' \leq r, Y^p_{r'} = M\}$, respectively. By definition of Γ^p, there is an $\alpha^{j/m-1}$-net for all positive integers j.

We next show how to maintain a k-center solution for the set of points M using the family of navigating nets $\{\Gamma^p\}_{p=1}^m$. For each navigating net $1 \leq p \leq m$, define $i^* = i + (p/m) - 1$ to be the index such that the α^{i^*}-net $Y^p_{\alpha^{i^*}}$ has at most k centers and $Y^p_{\alpha^{i^*-1}}$ has more than k centers. Define $cost_p = \frac{\alpha}{\alpha-1}\alpha^{i^*}$ for all $1 \leq p \leq m$. We compare the costs of all navigating nets and pick the navigating net p^* with minimal cost $p^* = \arg\min_{1 \leq p \leq m} cost_p$. The set of centers $Y^{p^*}_{\alpha^{i^*}}$ is the output k-center solution.

The next lemma proves that every point $x \in M$ is within a distance $cost_p = \frac{\alpha}{\alpha-1}\alpha^{i^*}$ of a center in $Y^p_{\alpha^{i^*}}$.

Lemma 3.1. *For $1 \leq p \leq m$ and $x \in M$ there is a center $c \in Y^p_{\alpha^{i^*}}$ such that $d(x, c) \leq cost_p$.*

Proof. By construction, the set $Y^p_{\alpha^{i^*}}$ is an α^{i^*}-net of $Y^p_{\alpha^{i^*-1}}$ and all elements of $Y^p_{\alpha^{i^*-1}}$ are within distance α^{i^*} to a center in $Y^p_{\alpha^{i^*}}$. Similarly, the elements of $Y^p_{\alpha^{i^*-2}}$ are within distance $\alpha^{i^*} + \alpha^{i^*-1}$ to a center in $Y^p_{\alpha^{i^*}}$ and so on. Note that the set $Y^p_{r^p_{\min}}$ contains all points currently in M and thus the distance of every point in M to some center in $Y^p_{\alpha^{i^*}}$ forms a geometric series.

Formally, let $x \in M$ be arbitrary and let c be its ancestor in $Y^p_{\alpha^{i^*}}$. Then the distance between c and x is bounded as follows:

$$\begin{aligned} d(x,c) &\leq \alpha^{i^*} + \alpha^{i^*-1} + \alpha^{i^*-2} + \cdots \\ &\leq \alpha^{i^*} \sum_{i=0}^{\infty} \left(\frac{1}{\alpha}\right)^i \\ &= \alpha^{i^*} \frac{\alpha}{\alpha - 1} = cost_p. \end{aligned}$$

□

The above lemma shows an upper bound for the output k-center solution $Y^{p^*}_{\alpha^{i^*}}$, i.e., $\phi(Y^{p^*}_{\alpha^{i^*}}) \leq cost_{p^*}$. The next lemma proves that $cost_{p^*}$ has the desired approximation guarantee, i.e., $cost_{p^*} \leq (2+\epsilon)\mathrm{OPT}$.

Lemma 3.2. *If $\alpha = O(\epsilon^{-1})$ and $m = O(\epsilon^{-1} \ln \epsilon^{-1})$ then $cost_{p^*} \leq (2+\epsilon)OPT$.*

Proof. We set $p^* \leftarrow \arg\min_{1 \leq p \leq m} cost_p$, where $cost_{p^*} = \frac{\alpha}{\alpha-1}\alpha^{i^*} = \frac{\alpha}{\alpha-1}\alpha^{j/m-1}$ for some $j \in \mathbb{Z}$. For comparison, consider level $\hat{\alpha} = \alpha^{(j-1)/m-1}$ and the corresponding $\hat{\alpha}$-net $Y^{\hat{p}}_{\hat{\alpha}}$. Note that we returned $Y^{p^*}_{\alpha^{i^*}}$ instead of $Y^{\hat{p}}_{\hat{\alpha}}$ as a solution even though $\alpha_{i^*} > \hat{\alpha}$. Consequently, $|Y^{\hat{p}}_{\hat{\alpha}}| > k \geq |Y^{p^*}_{\alpha_{i^*}}|$. Because $|Y^{\hat{p}}_{\hat{\alpha}}| > k$, at least two points $c_1, c_2 \in Y^{\hat{p}}_{\hat{\alpha}}$ are assigned to the same center c^* in the optimal solution. By the separation property we get that $d(c_1, c_2) \geq \hat{\alpha}$. Using the triangle inequality we obtain

$$2\mathrm{OPT} \geq d(c_1, c^*) + d(c^*, c_2) \geq d(c_1, c_2) \geq \alpha^{(j-1)/m-1}$$

and thus $\mathrm{OPT} \geq \alpha^{(j-1)/m-1}/2$. To obtain the desired approximation we compare our result with $cost_{p^*}$:

$$\begin{aligned} \frac{cost_{p^*}}{\mathrm{OPT}} &\leq \frac{\frac{\alpha}{\alpha-1}\alpha^{j/m-1}}{\alpha^{(j-1)/m-1}/2} \\ &= \frac{2\alpha^{j/m+1}}{(\alpha-1)\cdot\alpha^{(j-1)/m}} \\ &= \frac{2\alpha^{(j-1)/m}\cdot\alpha^{1/m+1}}{(\alpha-1)\cdot\alpha^{(j-1)/m}} \\ &= \frac{2\alpha}{(\alpha-1)}\sqrt[m]{\alpha}. \end{aligned}$$

It remains to show that $2\frac{\alpha}{(\alpha-1)}\sqrt[m]{\alpha} \leq 2(1+\epsilon)\cdot(1+\epsilon)$. Set $\alpha = 2/\epsilon$. Clearly, $\alpha = O(\epsilon^{-1})$ and $\frac{\alpha}{\alpha-1} = 1 + \frac{\epsilon}{2-\epsilon} \leq 1+\epsilon$ because $0 < \epsilon \leq 1$. Moreover note that $\alpha^{1/m} \leq (1+\epsilon)$ iff $1/m \log_{1+\epsilon} \alpha \leq 1$. The latter holds for any $m \geq \epsilon^{-1}\ln 2 + \epsilon^{-1}\ln\epsilon^{-1}$, which in turn implies that $m = O(\epsilon^{-1}\ln\epsilon^{-1})$. □

3.2 Fully dynamic k-center clustering

In this section, we present the details of the data structure presented in Section 3.1.

Data structure. Our data-structure needs to (1) maintain m navigating nets and (2) answer queries about our current solution to the given k-center clustering problem.

For (1) we use the data structure described in [22]: Let $1 \leq p \leq m$ and $\alpha^i \in \Gamma^p$: For the navigating net Π^p we do not store the sets $Y^p_{\alpha^i}$ explicitly. Instead, for every nontrivial scale $\alpha^i \in \Gamma^p$ and every $x \in Y^p_{\alpha^i}$ we store the *navigation list* L^p_{x,α^i} which contains nearby points to x in the α^{i-1}-net $Y^p_{\alpha^{i-1}}$, i.e.,

$L^p_{x,\alpha^i} = \{z \in Y^p_{\alpha^{i-1}} : d(z,x) \le \psi \cdot \alpha^i\}$ where $\psi \ge 4$. Additionally, for each $x \in M$ and each $1 \le p \le m$, we store the largest scale $\beta \in \Gamma^p$ such that $L^p_{x,\beta} = \{x\}$ but we do not store any navigation list $L^p_{x,\alpha}$ where $\alpha \in \Gamma^p$ and $\alpha < \beta$.

For (2), we also maintain the reverse information. Specifically, for every x in M and nontrivial scale α^i we maintain M^p_{x,α^i} which contains all the points in the α^{i+1}-net $Y^p_{\alpha^{i+1}}$ whose navigation list contains x, i.e., $M^p_{x,\alpha^i} = \{y \in Y^p_{\alpha^{i+1}} : x \in L^p_{y,\alpha^{i+1}}\}$. We maintain each M^p_{x,α^i} in a min-heap data structure, where each element $y \in M^p_{x,\alpha^i}$ is stored with the distance $d(x,y)$. It is well known that constructing such a min-heap takes $O(|M^p_{x,\alpha^i}|)$ time and the insert and delete operations can be supported in logarithmic time in the size of M^p_{x,α^i}. Let y be the closest point to x in M^p_{x,α^i}. The min-heap allows us to extract y in $O(1)$ time. Note that due to the covering property the closest point to y is also the closest point to x in $Y^p_{\alpha^{i+1}}$.

Additionally, we maintain a counter $c^p_{\alpha^i} = |Y^p_{\alpha^i}|$ for each scale $\alpha^i \in \Gamma^p$ and navigating net $1 \le p \le m$. Also, for each navigating net $1 \le p \le m$, we maintain the largest scale α^{i^*} such that $c^p_{\alpha^{i^*}} \le k$ and $c^p_{\alpha^{i^*+1}} > k$. We store $cost_{p^*} = \min_{1 \le p \le m} \frac{\alpha}{\alpha-1}\alpha^{i^*}$ and $p^* = \arg\min_{1 \le p \le m} cost_p$.

Preprocessing. Consider the construction of a single navigating net Π^p. We start by inserting the $|M|$ points using the routine described in [22][Chapter 2.5] whose running time is $O(2^{O(\kappa)} \log \Delta \log\log \Delta)$. Additionally we construct the lists M^p_{x,α^i} for every $1 \le p \le m$, $x \in M$ and scale α^i. We do this during the insert operation which takes care of the lists L^p_{x,α^i}. Due to Lemma 2.2 in [22] every navigation list has size $O(2^{O(\kappa)})$ and due to Lemma 2.3 in [22] every navigating net has only $\log \Delta$ nontrivial scales. Consequently, the sum of all navigation lists in a navigating net Π^p is of size $\sum_{x,\alpha^i} |L^p_{x,\alpha^i}| = O(|M| 2^{O(\kappa)} \log \Delta)$. Notice that $\sum_{x,\alpha^i} |L^p_{x,\alpha^i}| = \sum_{x,\alpha^i} |M^p_{x,\alpha^i}|$ because the sets M^p_{x,α^i} store the reverse information of the sets L^p_{x,α^i}. Since there are $m = O(\epsilon^{-1} \ln \epsilon^{-1})$ navigating nets, the latter yields a construction time of $O(|M| 2^{O(\kappa)} \log \Delta \log\log \Delta \cdot \epsilon^{-1} \ln \epsilon^{-1})$.

Handling Point Updates and Queries. To handle point insertions and deletions in the m navigating nets, we invoke the routines described in [22][Chapters 2.5-2.6] for all the navigating nets. We also keep track of the counters $c^p_{\alpha^i}$ and sets M^p_{x,α^i} when we handle the insertion and deletions of points in the navigating nets. While updating the counters $c^p_{\alpha^i}$ we simultaneously keep track of α^{i^*} for all navigating nets and maintain p^*.

We next discuss the query operations that our data-structure supports. First, we answer the query whether a given point $x \in M$ is a center by simply checking if the list $L^{p^*}_{x,\alpha^{i^*}}$ exists. Second, given a point $x \in M$ we return its corresponding center in $Y^{p^*}_{\alpha^{i^*}}$ as follows: First we check if x is a center. If not, we consider $L^{p^*}_{x,\beta} = \{x\}$. Note that $\beta = \alpha^i$ for some i. Then we repeatedly determine the navigation list $L^{p^*}_{y',\alpha^{i+1}}$ where y' is the center in $Y^p_{\alpha^{i+1}}$ which contains x within radius α^{i+1} using the min-heap $M^{p^*}_{x,\alpha^{i+1}}$. Then we set $i = i+1$ until $i = i^* - 1$. Once we arrive at the list $L^{p^*}_{y'',\alpha^{i^*-1}}$ we return y'' as the center x is assigned to.

The correctness of the maintained hierarchies follows from the correctness in [22]. Due to Lemma 3.2 the set $Y^{p^*}_{\alpha^{i^*}}$ is a feasible solution to the k-center problem whose cost is guaranteed to be within $(2+\epsilon)$ times the optimum cost.

We finally analyze the running time of the update and query operations. The time for handling a point insertion and a point deletion in a single navigating net is $O(2^{O(\kappa)} \log \Delta \log\log \Delta)$ (Theorem 2.5 in [22]). Since we maintain $m = O(\epsilon^{-1} \ln \epsilon^{-1})$ navigating nets, the overall time to handle a point insertion or deletion

is $O(2^{O(\kappa)} \log \Delta \log\log \Delta \cdot \epsilon^{-1} \ln \epsilon^{-1})$. It is straightforward to see that maintaining the counters $c^p_{\alpha^i}, \alpha^{i^*}, p^*, \beta$ and min-heaps M^p_{x,α^i} in all navigating nets can also be done in the same time per update. Determining if a point $x \in M$ is a center can be done in $O(1)$. Determining the center of a given point $x \in M$ takes $O(\log \Delta)$ time because there are $O(\log \Delta)$ nontrivial scales (Lemma 2.3 in [22]) and thus there are $O(\log \Delta)$ iterations in the lookup algorithm until the scale α^{i^*} is reached.

Combining the above guarantees yields Theorem 1.1.

4 Empirical Analysis

In this section, we present the experimental evaluation for our k-center algorithm. We implemented the algorithm described in the previous sections using cover trees [1, 21], which is a fast variant of navigating nets. The cover tree maintains the same invariants as navigating nets, except that for a point at a certain level in the hierarchy, we store *exactly one* nearby point one level up, instead of a set of points that are nearby. [1] show that all running time guarantees can be maintained for metric spaces with bounded expansion constant. This in turn implies that using a collection of cover trees yields a $(2+\epsilon)$-approximation for the k-center clustering problem. The running time for an insertion/deletion of a point in a cover tree is in $O(c^6 \ln |M|)$ where c is the expansion constant of M [1][1]. Our algorithm maintains $O(\epsilon^{-1} \ln \epsilon^{-1})$ cover trees. To obtain the current centers of a cover tree, we traverse the tree top-down and add all distinct points until we have k points. Due to the *nesting property* of the cover tree, i.e., every point which appears in some level i appears in every lower level $j < i$ in the tree [1] we are guaranteed to add all nodes of the desired level $Y^p_{i^*}$ described in Section 3.1. From now on we call the described algorithm $\mathscr{A}_{\text{Cov}}$.[2]

We compare our algorithm against the algorithm of Chan et al. [2] which is the state-of-the-art approach for the fully dynamic k-center problem in practice.

The algorithm of Chan et al. [2]. To gain some intuition into the state-of-the art algorithm in practice, we give a brief summary of the algorithm described in Chan et al. [2]: The algorithm maintains a clustering for each $r \in \Gamma := \{(1+\epsilon)^i : d_{\min} \leq (1+\epsilon)^i \leq d_{\max}, i \in \mathbb{N}\}$. Their algorithm is a $(2+\epsilon)$-approximation of the optimal solution and has an average running time of $O(k^2 \cdot \frac{\log(\Delta)}{\epsilon})$ per update. Note that the algorithm needs $d_{\min}$ and $d_{\max}$ as input and that these values are usually not available in practice. In contrast, $\mathscr{A}_{\text{Cov}}$ does not need these parameters. For our empirical analysis we provided these special parameters to the algorithm of Chan et al. [2]. For arbitrary instances one would initialize $d_{\min}, d_{\max}$ with the minimum/maximum value for the type double respectively to guarantee the correctness of their algorithm. From now on, we call their algorithm $\mathscr{A}_{\text{CGS}}$.

Setup. We implemented the cover tree in C++ and compiled it with g++-7.4.0. We executed all of our experiments on a Linux machine running on an AMD Opteron Processor 6174 with 2.2GHz and 256GB of RAM. In our experiments we evaluate $\mathscr{A}_{\text{CGS}}$ and $\mathscr{A}_{\text{Cov}}$ with the following pairwise combinations of $\epsilon \in \{0.1, 0.5, 1, 4\}$ and $k \in \{20, 50, 100, 200\}$. In total, we perform 10 different runs for each test instance and compute the arithmetic mean of the solution improvement and speedup on this instance. When further averaging over multiple instances, we use the geometric mean in order to give every instance a comparable influence on the final score. To measure the solution quality of an algorithm at any timepoint i we query for the current set of centers C_i. We do not directly compute the objective function value $\phi(C_i)$, since this is an expensive operation and it is not usually needed in practice. After the termination of the two algorithms we compute the objective function of the k-center solution $\phi(C_i)$ in order to compare the solutions of the

[1]The expansion constant of M is defined as the smallest value $c \geq 2$ such that $|B(p, 2r)| \leq c|B(p, r)|$ for all $p \in M$ and $r > 0$.

[2]Source code and data sets: `http://bit.ly/2S4WvJL`

two competing algorithms $\mathscr{A}_{\text{Cov}}$ and $\mathscr{A}_{\text{CGS}}$. Hence, the running times of both algorithms include the time to perform the point insertions/deletions and the queries (obtaining the centers of the solution), but not computing the objective function.

Instances and Update Sequences. To compare the performance of the two algorithms, we use the instances of Chan et al. [2] with euclidean distance and add an additional random instance.

- *Twitter.* The twitter data set [3] is introduced in [2] and consists of 21 million geotagged tweets. Our experiments consider only the first 200k tweets without duplicates.
- *Flickr.* The Yahoo Flickr Creative Commons 100 Million (YFCC100m) dataset [28] contains the metadata of 100 million pictures posted on Flickr. Unfortunately, we were not able to obtain the full dataset but used a search engine to build a subset of the dataset [19]. This subset entails 800k points with longitude and latitude.
- *Random.* This dataset consists of 2 million points created as follows: First, we sampled 100 points (x, y) uniformly at random for $-1 \leq x, y \leq 1$. Then, for each such point (x, y), we sampled another 20000 points using a normal distribution with (x, y) as mean and a variance of 0.001 respectively.

We use the following update sequences on the data sets inserting at most 200k points:

- *Sliding Window.* In the sliding window query, a point is inserted at some point in time t and will be removed at time $t + W$ where W is the window size. We chose a sliding window of size 60k following the implementation of Chan et al. During the update sequence we perform a query every 2000 insertions. Therefore, we perform 100 queries in total.
- *Random Insertions/Deletions.* We further distinguish between three concrete types of update sequences with 30% deletions, 10% deletions and 5% deletions. Points are inserted uniformly at random and deleted uniformly at random from the set of points already inserted. The chance to perform a query is 0.05%. The chance to insert a point at any given timestep is given by 1 - the respective deletion percentage above - 0.0005.

Results and Interpretation. We now evaluate the performance of $\mathscr{A}_{\text{Cov}}$ and compare it $\mathscr{A}_{\text{CGS}}$. In Table 1 we present the geometric mean speedup of $\mathscr{A}_{\text{Cov}}$ over $\mathscr{A}_{\text{CGS}}$. Here, both algorithm use the same parameter ϵ and have the same number of centers k. First of all, note that the empirical results *reinforce* the theoretical results: The larger k and ϵ are in our experiments, the larger the speedups of $\mathscr{A}_{\text{Cov}}$ become when compared to the algorithm $\mathscr{A}_{\text{CGS}}$. The running time of our algorithm $\mathscr{A}_{\text{Cov}}$ does not depend on k whereas in contrast each updates of $\mathscr{A}_{\text{CGS}}$ depends *quadratically* on k on average. Moreover, speedups improve for larger values of ϵ since the running time of $\mathscr{A}_{\text{Cov}}$ has a multiplicative factor of $O(\epsilon^{-1} \ln \epsilon^{-1})$ and $\mathscr{A}_{\text{CGS}}$'s running time includes a better factor $O(\epsilon^{-1})$. For example, when k is as large as 200, $\mathscr{A}_{\text{Cov}}$ is faster than $\mathscr{A}_{\text{CGS}}$ for all values of ϵ. In contrast, when $\epsilon = 1$, $\mathscr{A}_{\text{Cov}}$ has better speedups than $\mathscr{A}_{\text{CGS}}$ already for small values of k like $k = 50$. When $k = 20$, $\mathscr{A}_{\text{CGS}}$ is faster than $\mathscr{A}_{\text{Cov}}$.

We proceed to compare the solution quality when both algorithm use the *same* parameter ϵ and also use the same number of centers k. In Table 1 we present the geometric mean solution improvement of $\mathscr{A}_{\text{Cov}}$ over $\mathscr{A}_{\text{CGS}}$ for this case. $\mathscr{A}_{\text{Cov}}$ gives better solutions for *all* instances as soon as $\epsilon \geq 0.5$. Generally speaking, the larger ϵ gets, the larger is our improvement in the solution: For $\epsilon = 0.5$ our algorithm gives 10-12% better solutions. Setting $\epsilon = 1$ we already obtain 12-36% better solutions and finally, when setting $\epsilon = 4$ we obtain 7-114% better solutions. For $\epsilon = 0.1$ our solutions are about 3-4% worse than the solutions of $\mathscr{A}_{\text{CGS}}$. We conclude that our algorithm has a significant advantage in running time *and* solution quality for slightly larger values of k and ϵ.

Table 1: Top: Geometric mean speedup of $\mathscr{A}_{\text{Cov}}$ over $\mathscr{A}_{\text{CGS}}$. Bottom: Geometric mean improvement in solution quality of $\mathscr{A}_{\text{Cov}}$ over $\mathscr{A}_{\text{CGS}}$. For every entry both algorithms use the same ϵ and k. Higher is better.

ϵ / k	0.1	0.5	1.0	4.0
20	0.02	0.14	0.32	0.72
50	0.10	0.59	**1.34**	**3.05**
100	0.33	**2.01**	**4.45**	**10.32**
200	**1.15**	**7.66**	**17.74**	**39.60**
20	0.97	**1.12**	**1.27**	**1.07**
50	0.97	**1.10**	**1.36**	**1.46**
100	0.96	**1.12**	**1.12**	**2.14**
200	0.96	**1.12**	**1.19**	**1.28**

Table 2: Top: Geometric mean speedup over $\mathscr{A}_{\text{CGS}}$ when fixing $\epsilon = 1$ for $\mathscr{A}_{\text{Cov}}$. Bottom: Geometric mean improvement in solution quality when fixing $\epsilon = 1$ for our algorithm $\mathscr{A}_{\text{Cov}}$. Higher is better.

ϵ / k	0.1	0.5	1.0	4.0
20	**2.48**	0.55	0.32	0.14
50	**10.01**	**2.27**	**1.34**	0.62
100	**32.17**	**7.51**	**4.45**	**2.08**
200	**130.01**	**29.60**	**17.74**	**8.35**
20	0.91	**1.08**	**1.27**	**1.18**
50	0.90	**1.05**	**1.36**	**1.72**
100	0.89	**1.06**	**1.12**	**2.52**
200	0.88	**1.06**	**1.19**	**1.54**

Table 3: Top: Geometric mean speedup over $\mathscr{A}_{\text{CGS}}$ when fixing $\epsilon = 4$ for $\mathscr{A}_{\text{Cov}}$. Bottom: Geometric mean improvement in solution quality when fixing $\epsilon = 4$ for our algorithm $\mathscr{A}_{\text{Cov}}$. Higher is better.

ϵ / k	0.1	0.5	1.0	4.0
20	**12.09**	2.70	1.58	0.72
50	**48.69**	11.07	6.54	3.05
100	**159.11**	37.17	22.03	10.32
200	**616.51**	140.38	84.13	39.60
20	0.83	0.98	1.16	1.07
50	0.76	0.89	1.16	1.46
100	0.76	0.90	0.95	2.14
200	0.74	0.88	0.99	1.28

We now fix the value of ϵ in our algorithm to 1 and 4 and compare it with $\mathscr{A}_{\text{CGS}}$ for all values of ϵ. Table 2 presents the geometric mean speedup of the results and the geometric mean improvement in

solution quality for the case that we fix $\epsilon = 1$ in our algorithm. Notice that we obtain a speedup of at least one order of magnitude when $k \geq 50$ comparing to $\mathscr{A}_{\text{CGS}}$ with $\epsilon = 0.1$ while sacrificing only 9-12% in solution quality over $\mathscr{A}_{\text{CGS}}$. Most significantly, $\mathscr{A}_{\text{Cov}}$ is faster than $\mathscr{A}_{\text{CGS}}$ with $\epsilon = 0.5$ and $k \geq 50$ while also obtaining *better* solution quality. Similarly, we set $\epsilon = 4$ for $\mathscr{A}_{\text{Cov}}$ and compare the results to $\mathscr{A}_{\text{CGS}}$ for all values of ϵ again. The resulting geometric mean speedups and the geometric mean solution improvement is presented in Table 3. When comparing to $\mathscr{A}_{\text{CGS}}$ with $\epsilon = 0.1$ we obtain speedups of one order when $k \leq 50$ and two orders when $k \geq 100$ while sacrificing at most 26% of the solution quality.

5 Conclusion

We developed a fully dynamic $(2+\epsilon)$ approximation algorithm for k-center clustering with running time independent of k, the number of centers. Our algorithm maintains multiple hierarchies (so called navigating nets), so that each hierarchy stores sets of points which evolve over time through deletions and insertions. Roughly speaking, each of these hierarchies maintains the property that points residing on the same level are at least separated by a specific distance. This allows us to obtain k-center solutions with an approximation of $(2+\epsilon)$. Maintaining the navigating nets can be done in time independent of k. Lastly, we conducted an extensive evaluation of this algorithm which indicates that our algorithm outperforms the state-of-the-art algorithms for values of k and ϵ suggested by theory. In this case, our algorithm obtains significant speedups and improvements in solution quality. Important future work includes parallelization of the two algorithms as well as implementing the streaming algorithms in [26, 23] and [4].

References

[1] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*. ACM, 2006.

[2] T.-H. Hubert Chan, Arnaud Guerqin, and Mauro Sozio. Fully dynamic k-center clustering. In *International World Wide Web Conference (WWW)*, pages 579–587, 2018.

[3] T.-H. Hubert Chan, Arnaud Guerqin, and Mauro Sozio. Fully dynamic k-center clustering GitHub Repository. `https://github.com/fe6Bc5R4JvLkFkSeExHM/k-center`, 2018.

[4] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004. announced at STOC'97.

[5] Vincent Cohen-Addad, Niklas Hjuler, Nikos Parotsidis, David Saulpic, and Chris Schwiegelshohn. Fully dynamic consistent facility location. In *Conference on Neural Information Processing Systems (NeurIPS)*, pages 3250–3260, 2019.

[6] Vincent Cohen-Addad, Chris Schwiegelshohn, and Christian Sohler. Diameter and k-center in sliding windows. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 19:1–19:12, 2016.

[7] Tomás Feder and Daniel H. Greene. Optimal algorithms for approximate clustering. In *Symposium on Theory of Computing (STOC)*, pages 434–444, 1988.

[8] Sebastian Forster and Gramoz Goranci. Dynamic low-stretch trees via dynamic low-diameter decompositions. In *STOC*, pages 377–388, 2019. `doi:10.1145/3313276.3316381`.

[9] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75 – 174, 2010.

[10] Sorelle A. Friedler and David M. Mount. Approximation algorithm for the kinetic robust k-center problem. *Comput. Geom.*, 43(6-7):572–586, 2010.

[11] Jie Gao, Leonidas J. Guibas, and An Thai Nguyen. Deformable spanners and applications. *Comput. Geom.*, 35(1-2):2–19, 2006.

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

[13] Gramoz Goranci, Monika Henzinger, and Dariusz Leniowski. A tree structure for dynamic facility location. In *European Symposium on Algorithms (ESA)*, pages 39:1–39:13, 2018.

[14] Pierre Hansen and Brigitte Jaumard. Cluster analysis and mathematical programming. *Math. Program.*, 79:191–215, 1997.

[15] Sariel Har-Peled. Clustering motion. *Discrete & Computational Geometry*, 31(4):545–565, 2004. announced at FOCS'04.

[16] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006. announced at SoCG'04.

[17] Monika Henzinger, Dariusz Leniowski, and Claire Mathieu. Dynamic clustering to minimize the sum of radii. In *European Symposium on Algorithms (ESA)*, pages 48:1–48:10, 2017.

[18] Sagar Kale. Small space stream summary for matroid center. In *APPROX-RANDOM*, pages 20:1–20:22, 2019.

[19] Sebastian Kalkowski, Christian Schulze, Andreas Dengel, and Damian Borth. Real-time analysis and visualization of the yfcc100m dataset. In *Proceedings of the 2015 workshop on community-organized multimodal mining: opportunities for novel solutions*, pages 25–30, 2015.

[20] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

[21] Thomas Kollar. Fast nearest neighbors. Technical report, MIT, 2006.

[22] Robert Krauthgamer and James R. Lee. Navigating nets: simple algorithms for proximity search. In *Symposium on Discrete Algorithms (SODA)*, pages 798–807, 2004.

[23] Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *APPROX-RANDOM*, pages 165–178, 2008. doi:10.1007/978-3-540-85363-3_14.

[24] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *SODA*, pages 2616–2635, 2019. doi:10.1137/1.9781611975482.162.

[25] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[26] Melanie Schmidt and Christian Sohler. Fully dynamic hierarchical diameter k-clustering and k-center. *CoRR*, abs/1908.02645, 2019. URL: http://arxiv.org/abs/1908.02645.

[27] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[28] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015.