Fully Dynamic k-Center Clustering in Low Dimensional 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 nearly 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 large values of k, our algorithmic construction outperforms the state-of-the-art algorithm in terms of solution quality and running time.

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. [15, 39, 41]).

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 [31], 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 [19] 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 $(2-\varepsilon)$-approximate k-center clustering is known to be NP-hard [19].

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 [1]. However, most of the traditional clustering algorithms are not capable of capturing the dynamic nature of data and complete reclustering from scratch is used to obtain desirable clustering guarantees.

To address the above challenges, in this paper we study the *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. [7] designed an 8-approximation algorithm with $O(k \log k)$ amortized time per point insertion. This result was

*Department of Computer Science, University of Toronto, Canada.

†Theory and Application of Algorithms, University of Vienna, Austria.

‡Faculty of Mathematics and Computer Science, Heidelberg University, Germany.

‡The research leading to these results has received funding from the European Research Council under the European Unions Seventh Framework Programme (FP/2007–2013) / ERC Grant Agreement no. 340506. and from the Vienna Science and Technology Fund (WWTF) through project ICT15–003.

later improved to a $(2+\varepsilon)$-approximation by McCutchen and Khuller [38]. Recently, Chan et al. [4] 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 $(2+\varepsilon)$-approximation with $O(k^2 \cdot \varepsilon^{-1} \cdot \log \Delta)$ update time per operation, where Δ is the aspect ratio of the underlying metric space, i.e., the ratio of the largest and smallest distance in the metric space.

In many practical applications, real-world clustering instances lie on low dimensional manifolds (cf. [43]) and a natural question is whether one can exploit this structural property to obtain faster algorithms. To answer this question, we study the k-center clustering problem for "low-dimensional" general metrics spaces. More specifically, we consider the well-studied notion of doubling dimension [22, 33, 42, 37, 6, 12, 2]. 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 $r/2$ [33]. This notion can be thought of as a generalization of the Euclidean dimension since $\mathbb{R}^d$ has doubling dimension $\Theta(d)$. The opposite is not true and in fact, the family of metrics with bounded doubling dimension is notably larger than that of bounded-dimensional Euclidean metrics [34, 36, 35, 17].

The k-center clustering problem has been studied in the low dimensional regime from both the static and dynamic perspective. Feder and Greene [13] 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 [26] devised an algorithm that achieves a 2-approximation and runs in $O(n \log n)$ time. In the dynamic setting, Har-Peled [24] implicitly gave a fully-dynamic algorithm for metrics with bounded doubling dimension that reports a $(2+\varepsilon)$-clustering at any time while supporting insertion and deletions of points in $O(\text{poly}(k, \varepsilon^{-1}, \log n))$ time, where 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. A prominent example where this is justified is learning features from image data [9], where it is observed that increasing k leads to siginficant gains in the accuracy of their prediction model. Other applications where k is large include [18] and [14]. Unfortunately, the dependency on k seems inherent in the state-of-the-art dynamic algorithms; for example, the algorithm due to Chan et al. [4] requires examining the set of current centers upon insertion of a point, while the algorithm due to Har-Peled [24] employs the notion of *coresets*, which in turn requires dependency on the number of centers.

Taking cue from the discussion above, a natural question is whether the dependency on k is necessary or is just an artifact of the current algorithmic techniques.

> **Open Question 1.** Are there fully-dynamic algorithms for the k-center clustering problem with running time independent of k while still guaranteeing a $(2+\varepsilon)$-approximation ratio?

Even if we are able to come up with an algorithm with provable guarantees, in order to be competitive with the current approaches to dynamic k-center clustering, the new feature of our algorithm should translate in lower update times in benchmark instances from practice.

> **Open Question 2.** Do algorithms with provable update time independent of the number of centers k achieve a speed-up compared to the state-of-the-art dynamic k-center algorithms in benchmark instances?

In this paper we answer the first algorithmic question in the affirmative.

THEOREM 1.1. *Let $\varepsilon > 0$. There is a deterministic, fully-dynamic algorithm for the k-center clustering problem, where points are taken from a metric space with doubling dimension κ and aspect ratio Δ, such that the cost of the maintained solution is within a factor $(2+\varepsilon)$ to the cost of the optimal solution and the insertions and deletions of points are supported in $O((2/\epsilon)^{O(\kappa)} \log \Delta \log\log \Delta \cdot \ln \varepsilon^{-1})$ 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.*

For bounded doubling dimension metrics, which include d-dimensional Euclidean spaces and other metrics that usually appear in practice, our algorithm achieves nearly logarithmic update and query time in the parameters of the problem. Since by construction our algorithm is deterministic, it works against an adaptive adversary (i.e., updates are allowed to depend on prior decisions made by the algorithm). This is in contrast to the state-of-the art algorithm by Chan et al. [4] that only works against an oblivious adversary who must fix the sequence of updates before the dynamic algorithm starts.

To answer the second open question, 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 [3, 32]. We compare our results against the implementation of Chan et al. [4], which is the state-of-the-art algorithm for the problem. Our findings answer the open question in the affirmative: the proposed algorithm has a significant advantage in running time and solution quality for larger values of k, as suggested by our theoretical results. Another upside of the presented algorithm in constrast to Chan et al. [4] is that we do not need to know lower and upper bounds on the maximum and minimum distance between any two points that are ever inserted when initializing the algorithm. In a practical scenario these values might be unknown. This would, in the worst case, force Chan et al. [4] to choose MAX_DOUBLE and MIN_DOUBLE as upper and lower bounds respectively to guarantee correctness. Consequently, their running time for each update might be significantly slower in practice. We compare running times assuming that these values are precomputed for the respective datasets as it was assumed in their paper. Still, our algorithm outperforms Chan et al. [4] for large k in terms of running time as their algorithm has a quadratic dependence on k. Regarding solution quality, both algorithms obtain $(2+\varepsilon)$ approximations of the optimal solution but for $\varepsilon > 0.5$ we always obtain a better solution quality in the experiments. Most significantly, our experiments show that when one is willing to sacrifice only about 25% of solution quality we are orders of magnitudes faster than Chan et al. [4] even for smaller values of k.

1.1 Related work For an in-depth overview of clustering and its wide applicability we refer the reader to two excellent surveys [39, 23]. 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. [17] showed an algorithm that achieves an 8-approximation factor. Their result was subsequently improved to a $(4+\varepsilon)$ guarantee by Friedler and Mount [16]. In the streaming setting Cohen-Addad et al. [11] designed a $(6+\varepsilon)$-approximation algorithm with an expected update time of $O(k^2 \cdot \varepsilon^{-1} \cdot \log \Delta)$. However, their algorithm only works in the sliding window model and does not support arbitrary insertions and deletions of points. Another important result in the streaming setting is [21] where the authors implement the first single pass constant approximation algorithm for k-median, which was subsequently improved in [8]. Kale [29] studied streaming algorithms for a generalization of the k-center clustering problem, known as the matroid center problem where he gives a $(7+\varepsilon)$-approximate solution in one pass with space depending on the rank of the matroid.

Recently and independently of our work, Schmidt and Sohler [40] 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, k-median and k-means, clustering, Cohen-Addad et al. [10] and Henzinger and Kale [27] obtain fully dynamic algorithms with non-trivial running time and approximation guarantees for general metric spaces.

1.2 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 [25]. Given a metric space (M, d), and an integer radius $r \geq 0$, an *r-net* $Y_r \subseteq M$ 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 r and (b) the *separating* property, i.e., all distinct points $y, y' \in Y_r$ are at distance larger than r.

Consider $O(\varepsilon^{-1} \cdot \log \Delta)$ many r-nets of the metric space (M, d) with different values of r such that r is a power of $(1+\varepsilon)$. The union over all such r-nets naturally defines a hierarchy Π. Chan et al. [4] show that the smallest r in Π such that $|Y_r| \leq k$ yields a feasible k-center clustering whose cost is within $(2+\varepsilon)$ to the optimal one.

A natural attempt to extend the above static algorithm to the incremental setting is to maintain the hierarchy Π under insertions and deletion of points. In fact, Chan et al. [4] 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 on k in metrics with bounded doubling dimension and maintain a similar hierarchy under insertion and deletion of points. Concretely, our algorithm uses *navigating nets*, which is a hierarchy of r-nets defined over metric subspaces of M. Navigating nets were introduced by Krauthgamer and Lee [33] to build an efficient data structure for the nearest-neighbor search problem. We observe that the navigating nets data

structure can be extended to a fully dynamic algorithm for the k-center clustering problem with an update time independent of k. While navigating nets remove the dependency on the number of clusters, they only guarantee an 8-approximation of the dynamic k-center clustering problem. We improve the approximation factor from 8 to $(2+\varepsilon)$ by maintaining a carefully defined *collection* of navigating nets, while increasing the running time by a factor of $O(\varepsilon^{-1}\ln\varepsilon^{-1})$. The definition of this carefully defined collection of navigating nets is inspired by a technique first used for incremental streaming algorithms [38].

Similar hierarchical structures have been recently employed for solving the dynamic sum-of-radii clustering problem [28] and the dynamic facility location problem [20]. In comparison to our result that achieves a $(2+\varepsilon)$-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 $\phi(C) = \max_{x\in M} d(x, C)$, 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., $B(x, r) := \{y \in M \mid d(x, y) \le r\}$.

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 $(2+\varepsilon)$-approximation with a running time not depending on the number of clusters k. The first component of our algorithm is to use a nearest neighbor search data structure called "navigating nets" (Krauthgamer and Lee [33]). Unfortunately, this technique by itself only guarantees an 8-approximation, which does not suffice to obtain the claimed $(2+\varepsilon)$-approximation factor. The second part of our construction is a seemingly unrelated scaling argument by McCutchen and Khuller [38] used in in the context of incremental streaming algorithms. Our work is the first to apply this argument together with the nearest neighbor search data structures.

We start by reviewing notation from [33].

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

1. (separating) Distinct $x, y \in Y_r$ have $d(x, y) \ge r$.
2. (covering) $X \subseteq \bigcup_{y\in Y_r} B(y, r)$.

3.1 Navigating Nets Given a metric space (M, d) which evolves over time with parameters $d_{\min}$ and $d_{\max}$ we define $\Gamma := \{\alpha^i : i \in \mathbb{Z}\}$ as the set of *scales* of (M, d) for some $\alpha > 1$. We define an r-net for all $r \in \Gamma$ as follows: For all $r < d_{\min}$, we define $Y_r := M$ as a trivial r-net of M. For $r > d_{\min}$, define Y_r to be an r-net of $Y_{r/\alpha}$. Note that Y_r is a subset of $Y_{r/\alpha}$ and contains a point with distance $\le r$ to every point in $Y_{r/\alpha}$. A *navigating net* Π is defined as the union over all Y_r for $r \in \Gamma$. 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 $r_{\max} = \min\{r \in \Gamma : \forall r' \ge r, |Y_{r'}| = 1\}$, and (ii) the largest scale $r_{\min}$ defined by $r_{\min} = \max\{r \in \Gamma : \forall r' \le r, Y_{r'} = M\}$. All scales $r \in \Gamma$ such that $r \in [r_{\min}, r_{\max}]$ are referred to as *nontrivial* scales. Figure 1 illustrates an example of a navigating net.

3.2 Navigating nets with differing base distances In what follows, we describe how to obtain a $(2+\varepsilon)$-approximation for the k-center clustering problem by maintaining navigating nets in parallel. This technique was originally introduced by McCuthen and Khuller [38] for improving the approximation ratio of the incremental doubling algorithm for the k-center problem due to Charikar et al. [7].

The key idea behind the construction is that instead of maintaining one navigating net, we maintain $m \ge 1$ navigating nets with differing base distances. The constant m depends on the desired approximation factor: For a better approximation, we maintain more navigating nets. We define m later on. The navigating nets differ *only* in the corresponding set Γ which is used to define them. More concretely, for each integer

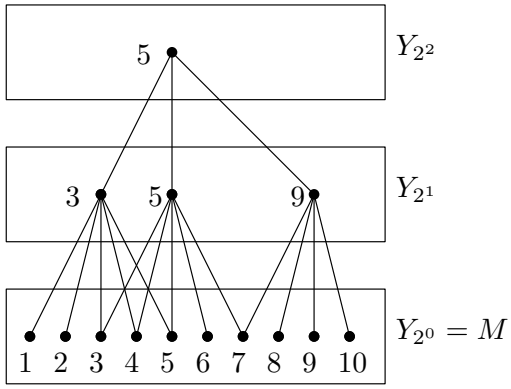

Figure 1: An example for a navigating net with $M = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ where d is Euclidean and $\alpha = 2$. Consequently, $\Gamma = \{2^i : i \in \mathbb{Z}\}$. In the navigating net, the nontrivial scales are given by $\{2^0, 2^1, 2^2\}$. Consequently, $r_{\max} = 2^2$, $r_{\min} = 2^0$.

$1 \le p \le m$, let $\Gamma^p = \{\alpha^{i+(p/m)-1} \mid i \in \mathbb{Z}_+\}$.

Let $Y_r^p := M$ for all $r \le d_{\min}$ and for all $r \in \Gamma^p$, let Y_r^p be an r-net of $Y_{r/\alpha}^p$. A *navigating net* Π^p is defined as the union over all Y_r^p for $r \in \Gamma^p$. Similarly, we maintain $r_{\max}^p$ and $r_{\min}^p$, such that $r_{\max}^p = \min\{r \in \Gamma^p \mid \forall r' \ge r, |Y_{r'}^p| = 1\}$ and $r_{\min}^p = \max\{r \in \Gamma^p \mid \forall r' \le r, Y_{r'}^p = M\}$, respectively. By definition of Γ^p, there is an $\alpha^{j/m-1}$-net of $\alpha^{j/m-2}$ 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 $\{\Pi^p\}_{p=1}^m$. For each navigating net Π^p $1 \le p \le m$, define $i_p^* = i + (p/m) - 1$ to be the index such that the α^{i^*}-net $Y_{\alpha^{i_p^*}}^p$ has at most k centers and $Y_{\alpha^{i_p^*-1}}^p$ has more than k centers. We will omit from now on the p from i_p^* if it is clear from context which navigating net i_p^* belongs to. Define $cost_p = \frac{\alpha}{\alpha-1}\alpha^{i^*}$ for all $1 \le p \le m$. We compare the costs of all navigating nets and pick the navigating net p^* with minimal cost $p^* = \arg\min_{1 \le p \le m} cost_p$. The set of centers $Y_{\alpha^{i^*}}^{p^*}$ 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 $Y_{\alpha^{i^*}}^p$.

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

Proof. By construction, the set $Y_{\alpha^{i^*}}^p$ is an α^{i^*}-net of $Y_{\alpha^{i^*-1}}^p$ and all elements of $Y_{\alpha^{i^*-1}}^p$ are within distance α^{i^*} to a center in $Y_{\alpha^{i^*}}^p$. Similarly, the elements of $Y_{\alpha^{i^*-2}}^p$ are within distance $\alpha^{i^*} + \alpha^{i^*-1}$ to a center in $Y_{\alpha^{i^*}}^p$ and so on. Note that the set $Y_{r_{\min}^p}^p$ contains all points currently in M and thus the distance of every point in M to some center in $Y_{\alpha^{i^*}}^p$ forms a geometric series. Formally, let $x \in M$ be arbitrary and let c be its ancestor in $Y_{\alpha^{i^*}}^p$. Then the distance between c and x is bounded as follows:

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

□

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

LEMMA 3.2. *If $\alpha = 2/\varepsilon = O(\varepsilon^{-1})$ and $m \ge \varepsilon^{-1} \ln 2 + \varepsilon^{-1} \ln \varepsilon^{-1} = O(\varepsilon^{-1} \ln \varepsilon^{-1})$ then $cost_{p^*} \le (2 + \varepsilon)\mathsf{OPT}$.*

Proof. We set $p^* = \arg\min_{1 \le p \le m} cost_p$, 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 $Y_{\hat{\alpha}}^{\hat{p}}$. Note that we returned $Y_{\alpha^{i^*}}^{p^*}$ instead of $Y_{\hat{\alpha}}^{\hat{p}}$ as a solution even though $\alpha_{i^*} > \hat{\alpha}$. Consequently, $|Y_{\hat{\alpha}}^{\hat{p}}| > k \ge |Y_{\alpha_{i^*}}^{p^*}|$. Because $|Y_{\hat{\alpha}}^{\hat{p}}| > k$, at least two points $c_1, c_2 \in Y_{\hat{\alpha}}^{\hat{p}}$ are assigned to the same center c^* in the optimal solution. By the separation property of $Y_{\hat{\alpha}}^{\hat{p}}$ we get that $d(c_1, c_2) \ge \hat{\alpha}$. Using the triangle inequality we obtain

$$\begin{aligned} 2\mathsf{OPT} &\ge d(c_1, c^*) + d(c^*, c_2) \\ &\ge d(c_1, c_2) \ge \hat{\alpha} = \alpha^{(j-1)/m-1} \end{aligned}$$

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

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

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

3.3 Fully dynamic k-center clustering In this section, we present the details of the data structure presented in Section 3.2.

The metric spaces that we consider throughout our running time analysis satisfy the following property.

DEFINITION 3.1. (DOUBLING DIMENSION) *The doubling dimension of a metric space (M,d) is said to be bounded by $\kappa \geq 0$ if any ball $B(x,r)$ in (M,d) can be covered by 2^κ balls of radius $r/2$.*

The key idea for obtaining a data structure which has update time independent of k is to implement navigating nets [33], where the running time of the insertion and deletion operation depends on κ[1].

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

For (1) we use the data structure described in [33]: Given ε, we set $\alpha = 2/\varepsilon$ and $m = \varepsilon^{-1}\ln 2 + \varepsilon^{-1}\ln\varepsilon^{-1}$.

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., $L^p_{x,\alpha^i} = \{z \in Y^p_{\alpha^{i-1}} : d(z,x) \leq \psi\cdot\alpha^i\}$ where $\psi \geq 4$ (this is mandatory to make navigating nets work [33, Section 2.1]). Additionally, for each $x \in M$ and each $1 \leq p \leq m$, 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), i.e., answering queries, 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.

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

3.3.2 Preprocessing Consider the construction of a single navigating net Π^p. We start by inserting the $|M|$ points using the routine described in [33, Chapter 2.5] whose running time is $O((2/\epsilon)^{O(\kappa)}\log\Delta\log\log\Delta)$. Additionally we construct the lists M^p_{x,α^i} for every $1 \leq p \leq m$, $x \in M$ 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 [33] and by the choice of $\alpha = 2/\epsilon$ every navigation list has size $O((2/\epsilon)^{O(\kappa)})$. By Lemma 2.3 in [33] every navigating net has only $O(\log\Delta)$ nontrivial scales. Consequently, the sum of all navigation lists in a navigating net Π^p is of size $\sum_{x,\alpha^i}|L^p_{x,\alpha^i}| = O(|M|(2/\epsilon)^{O(\kappa)}\log\Delta)$. 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 $m = O(\varepsilon^{-1}\ln\varepsilon^{-1})$ navigating nets, the latter yields a construction time of $O(|M|(2/\epsilon)^{O(\kappa)}\log\Delta\log\log\Delta\ln\varepsilon^{-1})$.

3.3.3 Handling Point Updates and Queries To handle point insertions and deletions in the m navigating nets, we invoke the routines described in [33, Chapters 2.5–2.6] for all the navigating nets. We review them here for the sake of completeness.

Inserting a point q into a navigating net Π^p. To insert a point q into Π^p we decide for each Y^p_r, $r \in \Gamma^p$ whether to include q or not. For $r = r^p_{\min}$, insert q into Y^p_r because it contains all points in M. From then on, consider the next level $r = r\cdot\alpha$ and insert q when $d(q,Y^p_r) \geq r$ until $d(q,Y^p_r) \leq r$. Notice that this procedure guarantees that Y^p_r is an r-net of $Y^p_{r/\alpha}$. To implement the query $d(q,Y^p_r) \geq r$ efficiently using navigation lists we construct a small set $Z^p_r \subseteq Y^p_r$ containing all points close to q. Instead of computing the distance from q to all elements in Y^p_r we compute $d(q,Z^p_r)$. We compute the sets Z^p_r for $r \in \Gamma^p$ as follows: Initially, set $r = r^p_{\max}$ and $Z^p_r = \{y_{top}\}$ where y_{top} is the only element in $Y^p_{r^p_{\max}}$.

1. $Z^p_{r/\alpha} = \{y \in \bigcup_{z\in Z^p_r} L^p_{z,r} \mid d(q,y) \leq d(q,Z^p_r) + r\}$.
2. If $r/\alpha > r^p_{\min}$: set $r = r/\alpha$ and go to step 1. Otherwise terminate.

For Z^p_r we obtain the following properties:

[1] Krauthgamer and Lee [33] denote the doubling dimension κ of S with $\dim(S)$.

LEMMA 3.3. *For all nontrivial scales $r \in \Gamma^p$, the set Z_r^p contains $\{x \in Y_r^p \mid d(q,x) \leq \psi r\}$.*

Proof. By induction on r starting from $r = r_{\max}^p$. Base case is trivial, as $Z_r^p = Y_r^p$. Assume for the induction step that Z_r^p contains $\{x \in Y_r^p \mid d(q,x) \leq \psi r\}$. Let $y \in Y_{r/\alpha}^p$ be arbitrary and $d(q,y) \leq \psi \cdot r/\alpha$. Because Y_r^p is an r-net of $Y_{r/\alpha}^p$, there is a $y' \in Y_r^p$ such that $d(y,y') \leq r$. Note that Z_r^p contains y' by the induction hypothesis and trivially, the navigation list of y' also contains y as $L_{y',r}^p = \{z \in Y_{r/\alpha}^p : d(z,y') \leq \psi \cdot r\}$. Note that $d(q,y) \leq d(q,y') + r \leq d(q,Z_r^p) + r$ and thus we include y in $Z_{r/\alpha}^p$ in Step 1. ∎

LEMMA 3.4. *For all r, $|Z_r^p| \leq (2/\varepsilon)^{O(\kappa)}$*

Proof. Again by induction on r starting from $r = r_{\max}^p$. The inital set Z_r^p has size one, so consider any set $Z_{r/\alpha}^p$ constructed in step 1. Let $\Delta_{Z_{r/\alpha}^p}$ be the aspect ratio of $Z_{r/\alpha}^p$. Note that $Z_{r/\alpha}^p \subseteq Y_{r/\alpha}^p$ and due to the separating property any two points have distance at least r/α from each other. Also, for all $y \in Z_{r/\alpha}^p$ we have $d(q,y) \leq \psi r + r$ due to Lemma 3.3 and step 1. The aspect ratio $\Delta_{Z_{r/\alpha}^p}$ is $\frac{2(\psi r + r)}{r/\alpha}$. Thus, $Z_{r/\alpha}^p$ is contained in a ball of radius $2(\psi r + r) \leq 2\Delta_{Z_{r/\alpha}^p} r/\alpha$ centered at any point of Z_r^p. Applying the definition of doubling dimension $O(\log(\Delta_{Z_{r/\alpha}^p}))$ times, yields that the ball has size $r/(2\alpha)$. Each of these balls can cover at most one element in $Z_{r/\alpha}^p$ by the fact that any two points have distance at least r/α from each other. The number of balls is $|Z_{r/\alpha}^p| = (2/\varepsilon)^{O(\kappa)}$ because $\alpha = O(1/\varepsilon)$. ∎

Remember that we implemented Y_r^p using navigation lists: To construct the navigation list $L_{q,r}^p$ for all r where $q \in Y_{r/\alpha}^p$ we use the set $Z_{r/\alpha}^p$ to determine the elements which are within distance $\psi \cdot r$. Then we insert q into all navigation lists $L_{y,r}^p$ where $y, q \in Y_{r/\alpha}^p$ and $d(q,y) \leq \psi \cdot r$. The running time for a point insertion can thus be characterized by the product of the size of the lists Z_r^p ($O((2/\varepsilon)^{O(\kappa)})$, Lemma 3.4), the number of nontrivial scales ($O(\log \Delta)$, Lemma 2.3 in [33]) and the time needed to insert a new navigation list for a point into a balanced search tree of height $\log \Delta$ ($O(\log\log \Delta)$). We obtain a running time of $O((2/\epsilon)^{O(\kappa)} \log \Delta \log\log \Delta)$ for inserting a point.

Deleting a point q from a navigating net Π^p. Deleting a point is done similarly to inserting a point: First, find all navigation lists $L_{y,r}^p$ which contain q and delete q using the sets Z_r^p. Note that when we delete q from Y_r^p, Y_r^p might not be an r-net of $Y_{r/\alpha}^p$ anymore. In this case, we add points from $Y_{r/\alpha}^p$ to Y_r^p until Y_r^p satisfies the covering property again. All of these operations can again be implemented efficiently with the sets Z_r^p. We obtain a running time of $O((2/\epsilon)^{O(\kappa)} \log \Delta \log\log \Delta)$ for deleting a point. In summary, the time for handling a point insertion and a point deletion in a single navigating net is $O((2/\epsilon)^{O(\kappa)} \log \Delta \log\log \Delta)$ (Theorem 2.5 in [33]). Since we maintain $m = O(\varepsilon^{-1} \ln \varepsilon^{-1})$ navigating nets, the overall time to handle a point insertion or deletion is $O((2/\epsilon)^{O(\kappa)} \log \Delta \log\log \Delta \cdot \ln \varepsilon^{-1})$.

Queries. We also keep track of the counters $c_{\alpha^i}^p$ and sets M_{x,α^i}^p when we handle the insertion and deletions of points in the navigating nets. While updating the counters $c_{\alpha^i}^p$ 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 $L_{x,\alpha^{i^*}}^{p^*}$ exists. Second, given a point $x \in M$ we return its corresponding center in $Y_{\alpha^{i^*}}^{p^*}$ as follows: First we check if x is a center. If not, we consider $L_{x,\beta}^{p^*} = \{x\}$. Note that $\beta = \alpha^i$ for some i. Then we repeatedly determine the navigation list $L_{y',\alpha^{i+1}}^{p^*}$ where y' is the center in $Y_{\alpha^{i+1}}^p$ which contains x within radius α^{i+1} using the min-heap $M_{x,\alpha^{i+1}}^{p^*}$. Then we increase i by 1 until $i = i^* - 1$. Once we arrive at the list $L_{y'',\alpha^{i^*-1}}^{p^*}$ we return y'' as the center x is assigned to. We finally analyze the running time of the query operations. It is straightforward to see that maintaining the counters $c_{\alpha^i}^p, \alpha^{i^*}, p^*, \beta$ and min-heaps M_{x,α^i}^p 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 [33]) and thus there are $O(\log \Delta)$ iterations in the lookup algorithm until the scale α^{i^*} is reached.

The correctness of the maintained hierarchies follows from the correctness in [33]. Due to Lemma 3.2 the set $Y_{\alpha^{i^*}}^{p^*}$ is a feasible solution to the k-center problem whose cost is guaranteed to be within $(2+\varepsilon)$ times the optimum cost. 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 [3, 32], 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. Beygelzimer et al. [3] 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 $(2+\varepsilon)$-approximation for the k-center clustering problem. 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 $r > 0$. Our algorithm maintains $O(\varepsilon^{-1} \ln \varepsilon^{-1})$ 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 [3], we are guaranteed to add all nodes of the desired level $Y_{i^*}^p$ described in Section 3.2. From now on we call our algorithm $\mathcal{A}_{\text{Cov}}$.[2]

We compare our algorithm against the algorithm of Chan et al. [4] which is the state-of-the-art approach for the fully dynamic k-center problem in practice.

4.1 The algorithm of Chan et al. [4] 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. [4]: The algorithm maintains a clustering for each $r \in \Gamma := \{(1+\varepsilon)^i : d_{\min} \leq (1+\varepsilon)^i \leq d_{\max}, i \in \mathbb{N}\}$. Their algorithm is a $(2+\varepsilon)$-approximation of the optimal solution and has an average running time of $O(k^2 \cdot \frac{\log(\Delta)}{\varepsilon})$ per update. Note that the algorithm needs $d_{\min}$ and $d_{\max}$ as input and that it is not guaranteed that these values are available in practice. In contrast, $\mathcal{A}_{\text{Cov}}$ does not need these parameters. For our empirical analysis we provided these special parameters to the algorithm of Chan et al. [4]. 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 $\mathcal{A}_{\text{CGS}}$.

4.2 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 $\mathcal{A}_{\text{CGS}}$ and $\mathcal{A}_{\text{Cov}}$ with the following pairwise combinations of $\varepsilon \in \{0.1, 0.5, 1, 4\}$ 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 two competing algorithms $\mathcal{A}_{\text{Cov}}$ and $\mathcal{A}_{\text{CGS}}$. 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.

4.3 Instances and Update Sequences To compare the performance of the two algorithms, we use the instances of Chan et al. [4] with Euclidean distance and add an additional random instance.

- *Twitter.* The twitter data set [5] is introduced in Chan et al. [4] 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 [44] 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 [30]. 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

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

any given timestep is given by $1-$ the respective deletion percentage above -0.0005.

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

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 $\mathcal{A}_{\text{Cov}}$ over $\mathcal{A}_{\text{CGS}}$ for this case. $\mathcal{A}_{\text{Cov}}$ gives better solutions for *all* instances as soon as $\varepsilon \geq 0.5$. Generally speaking, the larger ε gets, the larger is our improvement in the solution: For $\varepsilon = 0.5$ our algorithm gives 10–12% better solutions. Setting $\varepsilon = 1$ we already obtain 12–36% better solutions and finally, when setting $\varepsilon = 4$ we obtain 7–114% better solutions. For $\varepsilon = 0.1$ our solutions are about 3–4% worse than the solutions of $\mathcal{A}_{\text{CGS}}$. We conclude that our algorithm has a significant advantage in running time *and* solution quality for slightly larger values of k and ε.

We now fix the value of ε in our algorithm to 1 and 4 and compare it with $\mathcal{A}_{\text{CGS}}$ 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 $\varepsilon = 1$ in our algorithm. Notice that we obtain a speedup of at least one magnitude when $k \geq 50$ comparing to $\mathcal{A}_{\text{CGS}}$ with $\varepsilon = 0.1$ while sacrificing only 9–12% in solution quality over $\mathcal{A}_{\text{CGS}}$. Most significantly, $\mathcal{A}_{\text{Cov}}$ is faster than $\mathcal{A}_{\text{CGS}}$ with $\varepsilon = 0.5$ and $k \geq 50$ while also obtaining *better* solution quality. Similarly, we set $\varepsilon = 4$ for $\mathcal{A}_{\text{Cov}}$ and compare the results to $\mathcal{A}_{\text{CGS}}$ 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 $\mathcal{A}_{\text{CGS}}$ with $\varepsilon = 0.1$ we

Table 1: Top: Geometric mean speedup of our algorithm $\mathcal{A}_{\text{Cov}}$ over $\mathcal{A}_{\text{CGS}}$. Bottom: Geometric mean improvement in solution quality of $\mathcal{A}_{\text{Cov}}$ over $\mathcal{A}_{\text{CGS}}$. Both algorithms use the same ε and k. Higher numbers are better.

$\varepsilon \rightarrow$ $k \downarrow$	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 $\mathcal{A}_{\text{CGS}}$ when fixing $\varepsilon = 1$ for our algorithm $\mathcal{A}_{\text{Cov}}$. Bottom: Geometric mean improvement in solution quality when fixing $\varepsilon = 1$ for $\mathcal{A}_{\text{Cov}}$. Higher numbers are better.

$\varepsilon \rightarrow$ $k \downarrow$	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 $\mathcal{A}_{\text{CGS}}$ when fixing $\varepsilon = 4$ for for our algorithm $\mathcal{A}_{\text{Cov}}$. Bottom: Geometric mean improvement in solution quality when fixing $\varepsilon = 4$ $\mathcal{A}_{\text{Cov}}$. Higher numbers are better.

$\varepsilon \rightarrow$ $k \downarrow$	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

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 $(2+\varepsilon)$ approximation algorithm for k-center clustering with running time independent of the number of centers k. 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 $(2+\varepsilon)$. 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. An interesting direction for future research includes parallelization of the two algorithms as well as implementing the streaming algorithms by Schmidt and Sohler [40, 38] and Charikar et al. [7].

References

[1] Internet live stats. `https://www.internetlivestats.com/one-second`, 2020. Accessed: 2020-07-20.

[2] Yair Bartal, Lee-Ad Gottlieb, and Robert Krauthgamer. The traveling salesman problem: Low-dimensionality implies a polynomial time approximation scheme. *SIAM J. Comput.*, 45(4):1563–1581, 2016.

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

[4] 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.

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

[6] T.-H. Hubert Chan and Shaofeng H.-C. Jiang. Reducing curse of dimensionality: Improved PTAS for TSP (with neighborhoods) in doubling metrics. In Robert Krauthgamer, editor, *SODA*, pages 754–765. SIAM, 2016.

[7] 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.

[8] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In Lawrence L. Larmore and Michel X. Goemans, editors, *STOC*, pages 30–39. ACM, 2003.

[9] Adam Coates and Andrew Y. Ng. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 561–580. Springer, 2012.

[10] 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.

[11] 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.

[12] Hu Ding, Fan Yang, and Mingyue Wang. On metric DBSCAN with low doubling dimension. In Christian Bessiere, editor, *IJCAI*, pages 3080–3086. ijcai.org, 2020.

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

[14] Sergio Mourelo Ferrandez, Timothy Harbison, Troy Weber, Robert Sturges, and Robert Rich. Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm. *Journal of Industrial Engineering and Management (JIEM)*, 9(2):374–388, 2016.

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

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

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

[18] Yong Ge, Hui Xiong, Chuanren Liu, and Zhi-Hua Zhou. A taxi driving fraud detection system. In Diane J. Cook, Jian Pei, Wei Wang, Osmar R. Zaïane, and Xindong Wu, editors, *ICDM*, pages 181–190. IEEE Computer Society, 2011.

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

[20] 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.

[21] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *FOCS 2000*, pages 359–366. IEEE Computer Society, 2000.

[22] Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low-distortion

embeddings. In *FOCS*, pages 534–543, 2003.
[23] Pierre Hansen and Brigitte Jaumard. Cluster analysis and mathematical programming. *Math. Program.*, 79:191–215, 1997.
[24] Sariel Har-Peled. Clustering motion. *Discret. Comput. Geom.*, 31(4):545–565, 2004.
[25] Sariel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, USA, 2011.
[26] 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.
[27] Monika Henzinger and Sagar Kale. Fully-Dynamic Coresets. In *ESA 2020*, volume 173, pages 57:1–57:21, 2020.
[28] Monika Henzinger, Dariusz Leniowski, and Claire Mathieu. Dynamic clustering to minimize the sum of radii. *Algorithmica*, 82:3183–3194, 2020.
[29] Sagar Kale. Small space stream summary for matroid center. In *APPROX-RANDOM*, pages 20:1–20:22, 2019.
[30] 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.
[31] 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.
[32] Thomas Kollar. Fast nearest neighbors. Technical report, MIT, 2006.
[33] Robert Krauthgamer and James R. Lee. Navigating nets: simple algorithms for proximity search. In *Symposium on Discrete Algorithms (SODA)*, pages 798–807, 2004.
[34] T. Laakso. Ahlfors Q-regular spaces with arbitrary $Q > 1$ admitting weak poincar inequality. *Geometric And Functional Analysis*, 10:111–123, 04 2000.
[35] Tomi J Laakso. Plane with A_∞-weighted metric not bilipschitz embeddable to R^n. *Bulletin of the London Mathematical Society*, 34(6):667–676, 2002.
[36] Urs Lang and Conrad Plaut. Bilipschitz embeddings of metric spaces into space forms. *Geometriae Dedicata*, 87(1-3):285–307, 2001.
[37] Yi Li and Philip M. Long. Learnability and the doubling dimension. In *NIPS*, pages 889–896, 2006.
[38] Richard Matthew McCutchen and Samir Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *APPROX-RANDOM*, pages 165–178, 2008.
[39] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.
[40] Melanie Schmidt and Christian Sohler. Fully dynamic hierarchical diameter k-clustering and k-center. *CoRR*, abs/1908.02645, 2019.
[41] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.
[42] Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *STOC*, pages 281–290, 2004.
[43] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.
[44] 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.