Optimal Fully Dynamic k-Center Clustering for Adaptive and Oblivious Adversaries

MohammadHossein Bateni
Google Research
bateni@google.com

Hossein Esfandiari
Google Research
esfandiari@google.com

Hendrik Fichtenberger
Google Research
fichtenberger@google.com

Monika Henzinger
University of Vienna
monika.henzinger@univie.ac.at [*]

Rajesh Jayaram
Google Research
rkjayaram@google.com

Vahab Mirrokni
Google Research
mirrokni@google.com

Andreas Wiese
Technical University of Munich
andreas.wiese@tum.de

Abstract

In fully dynamic clustering problems, a clustering of a given data set in a metric space must be maintained while it is modified through insertions and deletions of individual points. In this paper, we resolve the complexity of fully dynamic k-center clustering against both adaptive and oblivious adversaries. Against oblivious adversaries, we present the first algorithm for fully dynamic k-center in an arbitrary metric space that maintains an optimal $(2+\epsilon)$-approximation in $O(k \cdot \text{polylog}(n, \Delta))$ amortized update time. Here, n is an upper bound on the number of active points at any time, and Δ is the aspect ratio of the metric space. Previously, the best known amortized update time was $O(k^2 \cdot \text{polylog}(n, \Delta))$, and is due to Chan, Gourqin, and Sozio (2018). Moreover, we demonstrate that our runtime is optimal up to $\text{polylog}(n, \Delta)$ factors. In fact, we prove that even offline algorithms for k-clustering tasks in arbitrary metric spaces, including k-medians, k-means, and k-center, must make at least $\Omega(nk)$ distance queries to achieve any non-trivial approximation factor. This implies a lower bound of $\Omega(k)$ which holds even for the insertions-only setting.

For adaptive adversaries, we give the first deterministic algorithm for fully dynamic k-center which achieves a $O\left(\min\left\{\frac{\log(n/k)}{\log\log n}, k\right\}\right)$ approximation in $O(k \cdot \text{polylog}(n, \Delta))$ amortized update time. Further, we demonstrate that any algorithm which achieves a $O\left(\min\left\{\frac{\log n}{k \log f(k,2n)}, k\right\}\right)$-approximation against adaptive adversaries requires $f(k,n)$ update time, for any arbitrary function f. Thus, in the regime where $k = O(\sqrt{\frac{\log n}{\log\log n}})$, we close the complexity of the problem up to $\text{polylog}(n, \Delta)$ factors in the update time. Our lower bound extends to other k-clustering tasks in arbitrary metric spaces, including k-medians and k-means.

Finally, despite the aforementioned lower bounds, we demonstrate that an update time sublinear in k is possible against oblivious adversaries for metric spaces which admit locally sensitive hash functions (LSH), resulting in improved algorithms for a large class of metrics

[*]This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564 "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)" and from the Austrian Science Fund (FWF) project "Fast Algorithms for a Reactive Network Layer (ReactNet)", P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024.

including Euclidean space, ℓ_p-spaces, the Hamming Metric, and the Jaccard Metric. We also give the first fully dynamic $O(1)$-approximation algorithms for the closely related k-sum-of-radii and k-sum-of-diameter problems, with $O(\text{poly}(k, \log \Delta))$ update time.

Contents

1 Introduction

Clustering is a fundamental and well-studied problem in computer science, which arises in approximation algorithms, unsupervised learning, computational geometry, classification, community detection, image segmentation, databases, and other areas [HJ97, Sch07, For10, SM00, AV06, TSK13, CN12]. The goal of clustering is to find a structure in data by grouping together similar data points. Clustering algorithms optimize a given objective function which characterizes the quality of a clustering. One of the classical and best studied clustering objectives is the k-center objective.

Specifically, given a metric space $(\mathcal{X}, d)$ and a set of points $P \subseteq \mathcal{X}$, the goal of k-center clustering is to output a set $C \subset \mathcal{X}$ of at most k "centers", so that the maximum distance of any point $p \in P$ to the nearest center $c \in C$ is minimized. In other words, the goal is to minimize the objective function $\max_{p \in P} d(p, C)$, where $d(p, C) = \min_{c \in C} d(p, c)$. The k-center clustering problem admits several well-known greedy 2-approximation algorithms [Gon85, HS86]. However, it is known to be NP-hard to approximate the objective to within a factor of $(2 - \epsilon)$ for any constant $\epsilon > 0$ [HN79]. Moreover, even restricted to Euclidean space, it is still NP-hard to approximate beyond a factor of 1.822 [FG88, BE97].

While the approximability of many clustering tasks, including k-center clustering, is fairly well understood in the static setting, the same is not true for *dynamic datasets*. Recently, due to the proliferation of data and the rise of modern computational paradigms where data is constantly changing, there has been significant interest in developing dynamic clustering algorithms [CASS16, LV17, CGS18, GHL18, SS19, GHL$^+$21, HK20, HLM20, FLNFS21]. In the incremental dynamic setting, the dataset P is observed via a sequence of insertions of data points, and the goal is to maintain a good k-center clustering of the current set of active points. In the *fully dynamic* setting, points can be both inserted and deleted from P.

The study of dynamic algorithms for k-center was initated by Charikar, Chekuri, Feder, and Motwani [CCFM04], whose "doubling algorithm" maintains an 8-approximation in amortized $O(k)$ update time. However, the doubling algorithm is unable to handle deletions of data points. It was not until recently that the first fully dynamic algorithm for k-center, with update time better than naively recomputing a solution from scratch, was developed. In particular, the work of Chan, Guerqin, and Sozio [CGS18] proposed a randomized algorithm that maintains an optimal $(2 + \epsilon)$-approximation in $O(\frac{\log \Delta}{\epsilon} k^2)$ amortized time per update, where Δ is the aspect ratio of the dataset. The algorithm is randomized against an *oblivious adversary*, i.e., the adversary has to fix the input sequence in advance and cannot adapt it based on the decisions of the algorithm.

Since then, algorithms with improved running time have been demonstrated for the special cases of Euclidean space [SS19] (albeit, with a larger approximation factor), and for spaces with bounded doubling dimension [GHL$^+$21]. However, despite this progress, for general metrics the best known dynamic algorithm for k-center is still the algorithm by Chan et al. [CGS18] and it is open to find a dynamic algorithm with an update time that is sub-quadratic in k with any non-trivial approximation guarantee. Furthermore, it is open to find a dynamic algorithm for k-center with any non-trivial approximation guarantee and update time against an *adaptive adversary*, i.e., an adversary that can choose the next update depending on the previous decisions of the algorithm. Note that deterministic algorithms are necessarily robust against adaptive adversaries, however, most of the aforementioned algorithms are randomized. This raises the following question.

What are the best possible update times and approximation ratios of dynamic algorithms for k-center against oblivious and adaptive adversaries?

In addition to k-center, there are other popular clustering objectives whose complexity in the

k-center	Upper bound		Lower bound	
Adversary	update time	approx. ratio	update time	approx. ratio
Oblivious	$\boldsymbol{\tilde{O}(k/\epsilon)}$	$\boldsymbol{2+\epsilon}$	$\boldsymbol{\Omega(k)}$	**any**
	$\tilde{O}(k^2/\epsilon)$	$2+\epsilon$ [CGS18]		
Adaptive	$\boldsymbol{\tilde{O}(k)}$	$\boldsymbol{O\left(\min\{\frac{\log(n/k)}{\log\log n}, k\}\right)}$	$\boldsymbol{f(k,n)}$	$\boldsymbol{\Omega\left(\min\{\frac{\log(n)}{k\log f(k,2n)}, k\}\right)}$

Table 1: Our main new upper and lower bounds for k-center where $f(k,n)$ is an arbitrary function. The notation $\tilde{O}$ hides factors that are polylogarithmic in n, Δ.

fully dynamic model is not fully understood. For example, in the well-studied k-median and k-means problems, we minimize $\sum_{p\in P} d(p,C)$ and $\sum_{p\in P}(d(p,C))^2$, respectively. The latter problems are special cases of the more general (k,z)-*clustering problem*, where one minimizes $\sum_{p\in S} d(p,C)^z$. There are $O(1)$-approximation algorithms known for these problems in the offline setting [Gon85, HS85, HS86, AS16] as well as in the dynamic setting [HK20]. However, as is the case for k-center, these dynamic algorithms are randomized, and thus far no deterministic dynamic $O(1)$-approximation algorithms for these objectives are known.

In addition to k-means and median, a natural variant of the k-center is the k-*sum-of-radii* problem (also known as the k-*cover problem*) where we choose k centers $C = \{c_1,\dots,c_k\}$, and seek to minimize their sum $\sum_i r_i$ where r_i is the maximum distance $d(p,c_i)$ over all points point p assigned to c_i (i.e., the radius of the clustered centered at c_i). Related to this is the k-*sum-of-diameters* problem, where one minimizes the sum of the diameters of the clusters. For each of these closely related variants to k-center, no algorithms with non-trivial guarantees were known to exist.

1.1 Our Contributions

Our main contribution is the resolution of the complexity of fully dynamic k-center clustering, up to polylogarithmic factors in the update time, against oblivious adversaries, thereby answering the prior open question, as well as the resolution of the problem against adaptive adversaries for the regime when $k = O(\sqrt{\log n/\log\log(n+\Delta)})$. A summary of our upper and lower bounds for fully dynamic k-center is given in Table 1.

Oblivous adversaries. We first design an algorithm with update time that is *linear* in k, and only logarithmic in n and Δ, where n is an upper bound on the number of active points at a given point in time,[1] and Δ denotes the aspect ratio of the given metric space. Our algorithm achieves an approximation ratio of $2+\epsilon$, which is essentially tight since it is NP-hard to obtain an approximation ratio of $2-\epsilon$. This improves on the prior best known quadratic-in-k update time of [CGS18], for the same approximation ratio.

Theorem 1. *There is a randomized fully dynamic algorithm that, on a sequence of insertions and deletions of points from an arbitrary metric space, maintains a $(2+\epsilon)$-approximation to the optimal k-center clustering. The amortized update time of the algorithm is $O(\frac{\log\Delta\log n}{\epsilon}(k+\log n))$ in expectation, and $O(\frac{\log\Delta\log n}{\epsilon}(k+\log n)\log\delta^{-1})$ with probability $1-\delta$ for any $\delta\in(0,\frac{1}{2})$.*

We demonstrate that our update time is tight up to logarithmic factors, even in the insertion only setting. In fact, our lower bound extends immediately to all the other clustering problems

[1]We remark that our algorithm does not need to know n or the total number of updates in advance.

	Oblivious adversary		Adaptive adversary	
	update time	approx. ratio	update time	approx. ratio
k-median	$\boldsymbol{\Omega(k)}$	**any**	$\boldsymbol{f(k,n)}$	$\boldsymbol{\Omega\left(\frac{\log(n)}{\log f(1,2n)}\right)}$
	$\tilde{O}(k^2/\epsilon^{O(1)})$	$5.3+\epsilon$ [HK20]		
k-means	$\boldsymbol{\Omega(k)}$	**any**	$\boldsymbol{f(k,n)}$	$\boldsymbol{\Omega\left(\left(\frac{\log(n)}{\log f(1,2n)}\right)^2\right)}$
	$\tilde{O}(k^2/\epsilon^{O(1)})$	$36+\epsilon$ [HK20]		
(k,z)-clustering	$\boldsymbol{\Omega(k)}$	**any**	$\boldsymbol{f(k,n)}$	$\boldsymbol{\Omega\left(\left(\frac{\log(n)}{z+\log f(1,2n)}\right)^z\right)}$
k-sum-of-radii	$\boldsymbol{\Omega(k)}$	**any**	$\boldsymbol{f(k,n)}$	$\boldsymbol{\Omega\left(\frac{\log(n)}{\log f(1,2n)}\right)^\star}$
	$\boldsymbol{k^{O(1)}\log\Delta}$	$\boldsymbol{O(1)}$		
k-sum-of-diam.	$\boldsymbol{\Omega(k)}$	**any**	$\boldsymbol{f(k,n)}$	$\boldsymbol{\Omega\left(\frac{\log(n)}{\log f(1,2n)}\right)^\star}$
	$\boldsymbol{k^{O(1)}\log\Delta}$	$\boldsymbol{O(1)}$		

Table 2: Our lower bounds for k-median, k-means, k-sum-of-radii, and k-sum-of-diameter and our upper bounds for the latter two problems, where $f(k,n)$ is an arbitrary function. For the ratios marked with $\star$, we assume that the algorithm also outputs an estimate of the cost of an optimal solution.

defined above, see Table 2 for a further overview of our lower bounds beyond those given in Table 1.

Theorem 2. *If a dynamic algorithm for k-center, k-median, k-means, (k,z)-clustering for any $z>0$, k-sum-of-radii, or k-sum-of-diameter has an approximation ratio that is bounded by any function in k and n, then it has an update time of $\Omega(k)$. This holds already if points can only be inserted but not deleted.*

We remark that our lower bound is unconditional, i.e., it does not depend on any complexity theoretic assumptions. In fact, we prove an even stronger statement: any *offline* algorithm must query the distances of $\Omega(nk)$ pairs of points in order to guarantee any bounded approximation ratio with constant probability (see Section 8). The amortized per-update time of $\Omega(k)$ for dynamic insertion-only algorithms then follows immediately.

Deterministic algorithms and adaptive adversaries. For arbitrary metrics, our algorithms access the metric via queries to a distance oracle returning $d(x,y)$ between any two points which were inserted (and possibly deleted) up to that point. In the oblivious model, both the underlying metric and point insertions and deletions are fixed in advance. However, in the adaptive adversary model, the adversary can adaptively choose the values of $d(x,y)$ based on past outputs of the algorithm. Thus, the values of $d(x,y)$ are fixed only as they are queried (subject to obeying the triangle inequality). We refer to this as the *metric-adaptive* model (see discussion below).

Given our bounds against oblivious adversaries, it is natural to ask whether constant factor approximations for k-center are also achievable in $\tilde{O}(k)$, or even $\tilde{O}(\text{poly}(k))$, time against metric-adaptive adversaries. Perhaps surprisingly, we show that any k-center algorithm with update time poly-logarithmic in $n+\Delta$ (and arbitrary dependency on k) must incur an essentially logarithmic approximation factor if it needs to report an estimation of the cost. Moreover, for a large range of k, such a runtime is not possible even if the algorithm is tasked only with returning an approximately optimal set of k-centers.

Theorem 3. *For any $k \geq 1$, any dynamic algorithm which returns a set of k-centers against a metric-adaptive adversary with an amortized update time of $f(k,n)$, for an arbitrary function f, must have an approximation ratio of $\Omega\left(\min\{\frac{\log(n)}{k\log f(k,2n)}, k\}\right)$ for the k-center problem. In addition, even for the case of $k=1$, we show that* any *algorithm with an update time of $f(k,n)$*

- *for 1-median has an approximation ratio of $\Omega\left(\frac{\log(n)}{\log f(1,2n)}\right)$,*
- *for 1-means has an approximation ratio of $\Omega\left(\left(\frac{\log(n)}{\log f(1,2n)}\right)^2\right)$,*
- *for $(1,z)$-clustering has an approximation ratio of $\Omega\left(\left(\frac{\log(n)}{z+\log f(1,2n)}\right)^z\right)$,*
- *for 1-center, 1-sum-of-radii or 1-sum-of-diameters has an approximation ratio of $\Omega\left(\frac{\log(n)}{\log f(1,2n)}\right)$ if the algorithm is also able to estimate the cost of the optimal clustering.*

In particular, Theorem 3 demonstrates that for any k satisfying $\omega(1) \leq k \leq o(\frac{\log n}{\log\log(n+\Delta)})$, there is no constant-factor approximation algorithm for k-center running in polylog(n,Δ) time which is correct against an adaptive adversary. In fact, we prove even more fine-grained trade-offs between the update time of an algorithm and the possible approximation ratio (see Section 9 for details).

Separation between Adversarial Models. An important consequence of Theorem 3 is a separation in the complexity of k-clustering tasks between two distinct types of adaptive adversaries. Specifically, Theorem 3 is a lower bound for adversaries which decide on the answers to distance queries on the fly, without having fixed the metric beforehand, subject to the constraint that the answers are always consistent with some metric. In other words, the metric itself, in addition to the points which are inserted and deleted, are adaptively chosen (i.e., they are metric-adaptive adversaries). This is as opposed to the setting where the metric is fixed in advance, and only the insertions and deletions of points from that space are adaptively chosen (call these *point-adaptive* adversaries).

To illustrate the difference, suppose the current active point set is P. When a point-adaptive adversary inserts a new point q, since the metric is fixed, it must irrevocably decide on the value of $d(p,q)$ for all $p \in P$. On the other hand, a metric-adaptive adversary can defer fixing these value until it is queried for them. For example, suppose $P = \{a,b\}$ and then a point c is inserted, after which the algorithm queries the adversary for the value of $d(a,c)$, and then makes a (possibly randomized) decision $\mathcal{D}$ based on $d(a,c)$ (e.g. $\mathcal{D}$ could be whether to make c a center). Then a metric-adaptive adversary can adaptively decide on the value of $d(b,c)$ based on the algorithm's decision $\mathcal{D}$ (subject to not violating the triangle inequality), whereas a point-adaptive algorithm must have fixed $d(b,c)$ independent of $\mathcal{D}$.

This distinction is nuanced, but has non-trivial consequences. Namely, while Theorem 3 rules out fully dynamic $O(1)$-approximation algorithms with polylog(n,Δ) update time for k-means and k-medians for metric-adaptive adversaries, in [HK20] the authors design such algorithms against point-adaptive adversaries. Thus, Theorem 3 demonstrates that the algorithms of [HK20] would not have been possible against metric-adaptive adversaries. To the best of our knowledge, this is the first separation between such adversarial models for dynamic clustering.

Now while the *randomized* algorithms of [HK20] apply only to the weaker point-adaptive adversaries, note that deterministic algorithms are necessarily correct even against the stronger metric-adaptive adversaries. We show that such a deterministic algorithm in fact exists, with complexity matching the lower bound of Theorem 3 for $k = O(\sqrt{\log n/\log\log n})$.

Theorem 4. *Fix any $B \geq 2$ and $\epsilon \in (0,1)$. Then there is a deterministic dynamic algorithm for k-center with an amortized update time of $O\left(\frac{kB \log n \log \Delta}{\epsilon}\right)$ and an approximation factor of $(4+\epsilon)\min\left\{\frac{\log(n/k)}{\log B}, k\right\}$. Furthermore, the worst-cast insertion time is $O\left(\frac{kB \log n \log \Delta}{\epsilon}\right)$, and the worst-case deletion time is $O\left(\frac{k^2 B \log n \log \Delta}{\epsilon}\right)$.*

In particular, by setting $B = \log n$, we obtain a fully dynamic and deterministic algorithm for k-centers with an approximation ratio of $O\left(\min\left\{\frac{\log(n/k)}{\log \log n}, k\right\}\right)$, which, by Theorem 3, is optimal for the regime when $k = O(\sqrt{\frac{\log n}{\log \log n}})$ among algorithms that are robust against metric-adaptive adversaries and whose update time is polylogarithmic in n and Δ. Moreover, that by setting $B = n^{\epsilon}$ for any $\epsilon \in (0,1)$ we obtain a $O(1/\epsilon)$-approximation in time $\tilde{O}(kn^{\epsilon})$. Thus, Theorem 4 gives a deterministic $O(1)$-approximate algorithm for fully dynamic k-centers running in time $\tilde{O}(kn^{\epsilon})$ for any constant $\epsilon > 0$.

Improved Fully Dynamic k-center via Locally Sensitive Hashing. The lower bound of Theorem 2 demonstrates that, even against an oblivious adversary, one cannot beat $\Omega(k)$ amortized update time for fully dynamic k-center. However, in [SS19, GHL+21] it was shown that for the case of Euclidean space or metrics with bounded doubling dimension, update times *sublinear* in k are possible. These results rely on nearest neighbor data structures with running times that are sublinear in k, which are designed specifically for the respective metric spaces, along with specialized clustering algorithms to employ them. Note that for the case of Euclidean space, the resulting approximation factors were still logarithmic.

We significantly generalize and strengthen the above results, by demonstrating that *any* metric space admitting sublinear time nearest neighbor search data structures also admits fully dynamic k-center algorithms whose update time is sublinear in k. We do this via a black-box reduction, showing that we can obtain a fully dynamic algorithm for k-center for any metric space, given a *locally sensitive hash function* (LSH) for that space.

Informally, an LSH for a space $(\mathcal{X}, d)$ is a hash function h mapping $\mathcal{X}$ to some number of hash buckets, such that closer points in the metric are *more* likely to collide than far points. Thus, after hashing a subset $S \subset \mathcal{X}$ into the hash table, given a query point $q \in \mathcal{X}$, to find the closest points to q in S, it (roughly) suffices to search only through the hash bucket $h(q)$. The "quality" of an LSH family $\mathcal{H}$ is parameterized by four values (r, cr, p_1, p_2), meaning that for any $x, y \in \mathcal{X}$:

- If $d(x,y) \leq r$, then $\mathbf{Pr}_{h\sim\mathcal{H}}[h(x) = h(y)] \geq p_1$.
- If $d(x,y) > cr$, then $\mathbf{Pr}_{h\sim\mathcal{H}}[h(x) = h(y)] \leq p_2$.

Given the above definition, we can now informally state our main result for LSH-spaces (see Section 5 for formal statements).

Theorem 5 (informal)**.** *Let $(\mathcal{X}, d)$ be a metric space that admits an LSH $\mathcal{H}$ with parameters (r, cr, p_1, p_2) for every $r \geq 0$, and a running time of* $\text{Time}(\mathcal{H})$. *Then there is a fully dynamic algorithm for k-center on $(\mathcal{X}, d)$ with an approximation ratio of $c(2+\epsilon)$ and an update time of $O\left(\frac{\log \Delta}{\epsilon p_1} n^{2\rho} \cdot \text{Time}(\mathcal{H})\right)$, where $\rho = \ln p_1^{-1} / \ln p_2^{-1}$.*

Using known LSH hash functions from the literature, Theorem 5 immediately yields improved state of the art algorithms for Euclidean spaces, the Hamming metric, the Jaccard Metric, and the Earth Mover Distance (EMD). In particular, our bounds significantly improve the prior best known results of [SS19] for Euclidean space, by at least a logarithmic factor in the approximation ratio.

Metric space	**Our approx.**	**Our runtime**	Prior approx.	Prior runtime
Arbitrary metric space	$\boldsymbol{2+\epsilon}$	$\boldsymbol{\tilde{O}(k)}$	$2+\epsilon$	$\tilde{O}(k^2)$ [CGS18]
$(\mathbb{R}^d, \ell_p)$, $p \in [1,2]$	$\boldsymbol{c(4+\epsilon)}$	$\boldsymbol{\tilde{O}(n^{1/c})}$	$O(c \cdot \log n)$	$\tilde{O}(n^{1/c})$ [SS19]
Eucledian space $(\mathbb{R}^d, \ell_2)$	$\boldsymbol{c(\sqrt{8}+\epsilon)}$	$\boldsymbol{\tilde{O}(n^{1/c^2+o(1)})}$	$O(c \cdot \log n)$	$\tilde{O}(n^{1/c})$ [SS19]
Hamming metric	$\boldsymbol{c(4+\epsilon)}$	$\boldsymbol{\tilde{O}(n^{1/c})}$	–	–
Jaccard metric	$\boldsymbol{c(4+\epsilon)}$	$\boldsymbol{\tilde{O}(n^{1/c})}$	–	–
EMD over $[D]^d$ $d = O(1)$	$\boldsymbol{O(c \cdot \log D)}$	$\boldsymbol{\tilde{O}(n^{1/c})}$	–	–
EMD over $[D]^d$ sparsity s	$\boldsymbol{O(c \cdot \log sn \log d)}$	$\boldsymbol{\tilde{O}(n^{1/c})}$	–	–

Table 3: Our upper bounds for Fully Dynamic k-Centers against oblivious adversaries in different metric spaces. Results for specific metric spaces are obtained by applying known LSH families with Theorem 5.

See Table 3 for an summary of these results, and see Section 5.2 for the formal theorem statements for each metric space.

k-sum-of-radii and k-sum-of-diameter. Finally, we study the k-sum-of-radii and k-sum-of-diameter problems, for which there were previously no fully dynamic $O(1)$-approximation algorithms known with non-trivial update time. We design the first such algorithms, which hold against oblivious adversaries. Note that, as a consequence of Theorem 3, such a constant-factor approximation is only possible against an oblivious adversary.

Theorem 6. *There are randomized dynamic algorithms for the k-sum-of-radii and the k-sum-of-diameters problems with update time $k^{O(1/\epsilon)} \log \Delta$ and with approximation ratios of $13.008+\epsilon$ and $26.016+\epsilon$, respectively, against an oblivious adversary.*

Thus, we complete the picture that all clustering problems defined above admit dynamic $O(1)$-approximation algorithms against an oblivious adversary, but only (poly-)logarithmic approximation ratios against an adaptive adversary.

1.2 Technical Overview

We now describe the main technical steps employed in the primary results of the paper. For the remainder of the section, we fix a metric space $(\mathcal{X}, d)$. Let $r_{\min}$ and $r_{\max}$ be values such that $r_{\min} \le d(x,y) \le r_{\max}$ for any $x, y \in \mathcal{X}$. We set $\Delta = r_{\max}/r_{\min}$ as an upper bound on the aspect ratio of $(\mathcal{X}, d)$. For any fixed point in time, we denote the current input points of a dynamic clustering algorithm by P, and write n to denote the maximum size of P during the dynamic stream.

1.2.1 Algorithm for General Metric Spaces

Our starting point is the well-known reduction of Hochbaum and Shmoys [HS86] from approximating k-center to computing a maximal independent set (MIS) in a collection of *threshold* graphs. Formally, given a real $r > 0$, the r-threshold graph of a point set P is the graph $G_r = (V, E_r)$ with vertex set $V = P$, and where $(x,y) \in E_r$ is an edge if and only if $d(x,y) \le r$. One computes an

MIS $\mathcal{I}_r$ in the graph G_r for each $r = (1+\epsilon)^i r_{\min}$ with $i = 0, 1, \ldots, \lceil \log_{1+\epsilon} \Delta \rceil$. If $|\mathcal{I}_r| \leq k$, then $\mathcal{I}_r$ is a k-center solution of cost at most r. If $|\mathcal{I}_r| > k+1$, then there are $k+1$ points whose pair-wise distance is at least r. Therefore, by the triangle inequality, the optimal cost is at least $r/2$. These facts together yield a $(2+\epsilon)$-approximation.

By the above, it suffices to maintain an MIS in $O(\epsilon^{-1} \log \Delta)$ threshold graphs. Now the problem of maintaining an MIS in a fully dynamic sequence of *edge* insertions and deletions to a graph is very well studied [AOSS19, GK18, OSSW18, DZ18, CHHK16, CZ19, BDH$^+$19]. Notably, this line of work has culminated with the algorithms of [CZ19, BDH$^+$19], which maintain an MIS in expected $\operatorname{polylog} n$ update time per edge insertion or deletion. Unfortunately, point insertions and deletions from a metric space correspond to *vertex* insertions and deletions in a threshold graph. Since a single vertex update can change up to $O(n)$ edges in the graph at once, one cannot simply apply the prior algorithms for fully dynamic edge updates. Moreover, notice that in this vertex-update model, we are only given access to the graph via queries to the adjacency matrix. Thus, even finding a single neighbor of v can be expensive.

On the other hand, observe that in the above reduction to MIS, one does not always need to compute the entire MIS; for a given threshold graph G_r, the algorithm can stop as soon as it obtains an independent set of size at least $k+1$. This motivates the following problem, which is to return either an MIS of size at most k, or an independent set of size at least $k+1$. We refer to this as the k-Bounded MIS problem. Notice that given an MIS $\mathcal{I}$ of size at most k in a graph G, and given a new vertex v, if v is not adjacent to any $u \in \mathcal{I}$, then $\mathcal{I} \cup \{v\}$ is an MIS, otherwise $\mathcal{I}$ is still maximal. Thus, while an insertion of a vertex v can add $\Omega(n)$ edges to G, for the k-Bounded MIS problem, one only needs to check the $O(k)$ potential edges between v and $\mathcal{I}$ to determine if $\mathcal{I}$ is still maximal. Thus, our goal will be to design a fully dynamic algorithm for k-Bounded MIS with $\tilde{O}(k)$ amortized update time in the vertex-update model.

The Algorithm for k-Bounded MIS. To accomplish the above goal, we will adapt several of the technical tools employed by the algorithms for fully dynamic MIS in the edge-update model. Specifically, one of the main insights of this line of work is to maintain the *Lexicographically First Maximal Independent Set* (LFMIS) with respect to a random permutation $\pi : V \to [0,1]$ of the vertices.[2] The LFMIS is a natural object obtained by greedily adding the vertex with smallest $\pi(v)$ to the MIS, removing it and all its neighbors, and continuing iteratively until no vertices remain. Maintaining an LFMIS under a random ranking has several advantages from the perspective of dynamic algorithms. Firstly, it is *history-independent*, namely, once π is fixed, the current LFMIS depends only on the current graph, and not the order of insertions and deletions which led to that graph. Secondly, given a new vertex v, the probability that adding v to the graph causes a large number of changes to be made to the LFMIS is small, since $\pi(v)$ must have been similarly small for this to occur.

Given the above advantages of an LFMIS, we will attempt to maintain the set LFMIS_{k+1} consisting of the first $\min\{k+1, |\text{LFMIS}|\}$ vertices in the overall LFMIS with respect to a random ranking π; we refer to LFMIS_{k+1} as the top-k LFMIS (see Definition 8). Notice that maintaining this set is sufficient to solve the k-Bounded MIS problem. The challenge in maintaining the set LFMIS_{k+1} will be to handle the "excess" vertices which are contained in the LFMIS but are not in LFMIS_{k+1}, so that their membership in LFMIS_{k+1} can later be quickly determined when vertices with smaller rank in LFMIS_{k+1} are removed. To handle these excess vertices, we make judicious use of a priority queue $\mathcal{Q}$, with vertex priorities given by the ranking π.

Now the key difficulty in dynamically maintaining an MIS is that when a vertex v in an MIS is

[2]LFMIS with respects to random orderings were considered in [CHHK16, AOSS19, CZ19, BDH$^+$19].

deleted, potentially all of the neighbors of v may need to be added to the MIS, resulting in a large update time. Firstly, in order to keep track of which vertices could possibly enter the LFMIS when a vertex is removed from it, we maintain a mapping $\ell : V \to \text{LFMIS}$, such that for each $u \notin \text{LFMIS}$, we have $\ell(u) \in \text{LFMIS}$ and $(u, \ell(u))$ is an edge. The "leader" $\ell(u)$ of u serves as a certificate that u cannot be added to the MIS. When a vertex $v \in \text{LFMIS}$ is removed from the LFMIS, we only need to search through the set $\mathcal{F}_v = \{u \in V \mid \ell(u) = v\}$ to see which vertices should be added to the LFMIS. Note that this can occur when v is deleted, or when a neighbor of v with smaller rank is added to the LFMIS. Consequentially, the update time of the algorithm is a function of the number points u whose leader $\ell(u)$ changes on that step. For each such u, we can check in $O(k)$ time if it should be added to LFMIS_{k+1} by querying the edges between u and the vertices in LFMIS_{k+1}. By a careful amortized analysis, we can prove that the total runtime of this algorithm is indeed at most an $O(k)$ factor larger than the total number of leader changes. This leaves the primary challenge of designing and maintaining a leader mapping which changes infrequently.

A natural choice for such a leader function is to set $\ell(u)$ to be the *eliminator* of v in the LFMIS. Here, for any vertex u not in the LFMIS, the eliminator $\text{elim}_\pi(u)$ of u is defined to be its neighbor with lowest rank that belongs to the LFMIS. The eliminators have the desirable property that they are also history-independent, and therefore the number of changes to the eliminators on a given update depends only on the current graph and the update being made. An important key result of [BDH+19] is that the expected number of changes to the eliminators of the graph, even after the insertion or removal of an entire vertex, is at most $O(\log n)$. Therefore, if we could maintain the mapping $\ell(v) = \text{elim}_\pi(v)$ by keeping track of the eliminators, our task would be complete.

Unfortunately, keeping track of the eliminators will not be possible in the vertex-update model, since we can only query a small fraction of the adjacency matrix after each update. In particular, when a vertex v is inserted, it may change the eliminators of many of its neighbors, but we cannot afford to query all $\Omega(n)$ potential neighbors of v to check which eliminators have changed. Instead, our solution is to maintain a leader mapping $\ell(v)$ which is an "out-of-date" version of the eliminator mapping. Each time we check if a vertex v can be added to LFMIS_{k+1}, by searching through its neighbors in LFMIS_{k+1}, we ensure that either v is added to LFMIS_{k+1} or its leader $\ell(v)$ is updated to the current eliminator of v, thereby aligning $\ell(v)$ with $\text{elim}_\pi(v)$. However, thereafter, the values of $\ell(v)$ and $\text{elim}_\pi(v)$ can become misaligned in several circumstances. In particular, the vertex v may be moved into the queue $\mathcal{Q}$ due to its leader $\ell(v)$ either leaving the LFMIS, or being pushed out of the top $k+1$ vertices in the LFMIS. In the second case, we show that v can follow its leader to $\mathcal{Q}$ without changing $\ell(v)$, however, in the first case $\ell(v)$ is necessarily modified. On the other hand, as noted, the eliminator of v can also later change without the algorithm having to change $\ell(v)$. Our analysis proceeds by a careful accounting, in which we demonstrate that each change in an eliminator can result in at most a constant number of changes to the leaders ℓ, from which an amortized bound of $O(\log n)$ leader changes follows.

Comparison to the prior k-center algorithm of [CGS18]. The prior fully dynamic k-center algorithm of Chan, Gourqin, and Sozio [CGS18], which obtained an amortized $O(\epsilon^{-1} \log \Delta \cdot k^2)$ update time, also partially employed the idea of maintaining an LFMIS (although the connection to MIS under lexicographical orderings was not made explicit in that work). However, instead of consistently maintaining the LFMIS with respect to a random ranking π, they begin by maintaining an LFMIS with respect to the ordering π' in which the points were originally inserted into the stream. Since this ordering is adversarial, deletions in the stream can initially be very expensive to handle. To prevent bad deletions from repeatedly occurring, whenever a deletion to a center c occurs, the algorithm of [CGS18] randomly reorders all points which are contained in clusters that come after c in the current ordering being used. In this way, the algorithm of [CGS18] gradually converts the adversarial ordering π' into a random ordering π. However, by reordering *all* points which occurred after a deleted center c, instead of just the set of points which were led by that center (via a mapping ℓ), the amortized update time of the algorithm becomes $O(k^2)$.[3] In contrast, one of our key insights is to update the entire clustering to immediately reflect a random LFMIS ordering after each update.

1.2.2 Algorithm for LSH Spaces

The extension of our algorithm to LSH spaces is based on the following observation: each time we attempt to add a vertex v to LFMIS_{k+1}, we can determine the fate of v solely by finding the vertex $u \in \mathrm{LFMIS}_{k+1}$ in the neighborhood of v of minimal rank (i.e., the eliminator of v, if it is contained in LFMIS_{k+1}). If $\pi(u) < \pi(v)$, we simply set $\ell(v) = u$ and proceed. Otherwise, we must find all other neighbors w of v in LFMIS_{k+1}, remove them from the LFMIS, and set $\ell(w) = u$. Finding the vertex u can therefore be cast as an r-*near neighbor search* problem: here, one wants to return any $u \in \mathrm{LFMIS}_{k+1}$ which is at distance at most r from u, with the caveat that we need to return such vertices in order based on their ranking. Since, whenever u enters the LFMIS, each point w that we search through which leaves LFMIS_{k+1} had its leader change, if we can find each consecutive neighbor of u in LFMIS_{k+1} in time α, we could hope to bound the total runtime of the algorithm by an $O(\alpha)$ factor more than the total number of leader changes, which we know to be small by analysis of the general metric space algorithm.

To achieve values of α which are sublinear in k, we must necessarily settle for an *approximate near neighbor search* (ANN) algorithm. A randomized, approximate (r, cr)-nearest neighbor data structure will return any point in LFMIS_{k+1} which is at distance at most cr, assuming there is at least one point at distance at most r in LFMIS_{k+1}. In other words, such an algorithm can be used to find all edges in G_r, with the addition of any arbitrary subset of edges in G_{cr}. By relaxing the notion of a threshold graph to allow for such a c-approximation, one can hope to obtain a $c(2+\epsilon)$-approximation to k-center via solving the k-Bounded MIS problem on each relaxed threshold graph.

The key issue above is that, when using an ANN data structure, the underlying relaxed threshold graph is no longer a deterministic function of the point set P, and is instead "revealed" as queries are made to the ANN data structure. We handle this issue by demonstrating that, for the class of ANN algorithms based on locally sensitive hash functions, one can define a graph G which is only a function of the randomness in the ANN data structure, and not the ordering π. The edges of this graph are defined in a natural way — two points are adjacent if they collide in at least one of the hash buckets. By an appropriate setting of parameters, the number of collisions between points at distance larger than cr can be made small. By simply ignoring such erroneous edges as they are queried, the runtime increases by a factor of the number of such collisions.

1.2.3 Lower bounds against an oblivious adversary

For proving the lower bound of $\Omega(k)$ we use the following hard distribution: in one case we randomly plant k clusters each of size roughly n/k, where points within a cluster are close and points in separate clusters are far. In the second case, we do the same, and subsequently choose a point $i \sim [n]$ randomly and move it very far from all points (including its own cluster). Adaptive algorithms can gradually winnow the set of possible locations for i by discovering connected components in the

[3]Consider the stream which inserts k clusters of equal size n/k, and then begins randomly deleting half of each cluster in reverse order. By the time a constant fraction of all the points are deleted, for each deletion the probability a leader is deleted is $\Omega(k/n)$, but such a deletion causes $O(nk)$ work to be done by the algorithm.

clusters, and eliminating the points in those components. Our proof follows by demonstrating that a large fraction of the input distribution results in any randomized algorithm making a sequence of distance queries which eliminates few data points, and therefore gives only a small advantage in discovering the planted point i.

1.2.4 Lower bounds against an adaptive adversary

We sketch our main ideas behind the lower bounds against an adaptive adversary. For ease of presentation, we discuss here a simplified setting in which the algorithm (i) can query only the distances between points that are currently in P (i.e., that have been introduced already but not yet removed) and (ii) has a worst-case update time of $f(k, n)$ (rather than amortized update time) for some function f. Note that these assumptions are not needed for the full proof presented in Section 9. Our goal is to prove a lower bound on the approximation ratio of such an algorithm.

The adversary starts by adding points into P and maintains an auxiliary graph $G = (V, E)$ with one vertex v_p for each point p. Whenever the algorithm queries the distance between two points $p, p' \in P$, the adversary reports that they have a distance of 1 and adds an edge $\{v_p, v_{p'}\}$ of length 1 to G. Intuitively, the adversary uses G to keep track of the previously reported distances. Whenever there is a vertex v_p with a degree of at least $100f(k, n)$, in the next operation the adversary deletes the corresponding point p. There could be several such vertices, and then the adversary deletes them one after the other. Thanks to assumption (i), once a point p is deleted, the degree of v_p cannot increase further. Hence, the degrees of the vertices in G cannot grow arbitrarily. More precisely, one can show that the degree of each vertex can grow to at most $O(\log n \cdot f(k, n))$. In particular, at least half of the vertices in G are at distance at least $\Omega(\log_{O(\log n \cdot f(k,n))} n) = \Omega\left(\log n \,/\, [\log\log n \cdot \log(f(k, n))]\right)$ to v_p.

Now observe that the algorithm knows only the point distances that it queried, i.e., those that correspond to edges in G. For all other distances, the triangle inequality imposes only an upper bound for any pairs of points whose corresponding vertices are connected in G. Thus, the algorithm cannot distinguish the setting where the underlying metric is the shortest path metric in G from the setting where all points are at distance 1 to each other. If $k = 1$, for any of the problems under consideration, then for the selected center c the algorithm cannot distinguish whether all points are at distance 1 to c or if half of the points in P are at distance $\Omega\left(\log n \,/\, [\log\log n \cdot \log(f(1, n))]\right)$ to c. Therefore, any algorithm which is correct with constant probability will suffer an approximation of at least $\Omega\left(\log n \,/\, [\log\log n \cdot \log(f(1, n))]\right)$.

We improve the above construction so that it also works for algorithms that have amortized update times, are allowed to query distances to points that are already deleted, and may output $O(k)$ instead of k centers. To this end, we adjust the construction so that for points with degree of at least $100f(k, n)$, where $f(k, n)$ is the (amortized) number of distance queries per operation, we report distances that might be larger than 1 and that still allow us to use the same argumentation as above. At the same time, we remove the factor of $\log\log n$ in the denominator.

1.2.5 Algorithms for k-Sum-of-radii and k-Sum-of-diameters

Our algorithms for k-sum-of-radii and k-sum-of-diameters against an oblivious adversary use the following paradigm: we maintain bi-criteria approximations, i.e., solutions with a small approximation ratio that might use more than k centers, i.e., up to $O(k/\epsilon)$ centers. In a second step, we use the centers of this solution as the input to an auxiliary dynamic instance for which we maintain a solution with only k centers. These centers then form our solution to the actual problem, by increasing their radii appropriately. Since the input to our auxiliary instance is much smaller than the input to the original instance, we can afford to use algorithms for it whose update times have a much higher dependence on n, and in fact we can even recompute the whole solution from scratch.

More precisely, recall that in the static offline setting there is a $(3.504+\epsilon)$-approximation algorithm with running time $n^{O(1/\epsilon)}$ for k-sum-of-radii [CP04]. We provide a black-box reduction that transforms any such offline algorithm with running time of $g(n)$ and an approximation ratio of α into a fully dynamic $(6+2\alpha+\epsilon)$-approximation algorithm with an update time of $O(((k/\epsilon)^5 + g(\mathrm{poly}(k/\epsilon)))\log\Delta)$. To this end, we introduce a dynamic primal-dual algorithm that maintains a bi-criteria approximation with up to $O(k/\epsilon)$ centers. After each update, we use these $O(k/\epsilon)$ centers as input for the offline α-approximation algorithm, run it from scratch, and increase the radii of the computed centers appropriately such that they cover all points of the original input instance.

For computing the needed bi-criteria approximation, there is a polynomial-time offline primal-dual $O(1)$-approximation algorithm with a running time of $\Omega(n^2k)$ [CP04] from which we borrow ideas. Note that a dynamic algorithm with an update time of $(k\log n)^{O(1)}$ yields an offline algorithm with a running time of $n(k\log n)^{O(1)}$ (by inserting all points one after another) and no offline algorithm is known for the problem that is that fast. Hence, we need new ideas. First, we formulate the problem as an LP (P) which has a variable $x_p^{(r)}$ for each combination of a point p and a radius r from a set of (suitably discretized) radii R, and a constraint for each point p. Let (D) denote its dual LP, see below, where $z := \epsilon\,\mathrm{OPT}'/k$ and OPT' is a $(1+\epsilon)$-approximate estimate on the value OPT of the optimal solution.

$$\begin{aligned} \min \sum_{p\in P}\sum_{r\in R} x_p^{(r)}(r+z) & & \\ \text{s.t.} \sum_{p'\in P}\sum_{r:d(p,p')\le r} x_{p'}^{(r)} \ge 1 \ \forall p\in P & & (P) \\ x_p^{(r)} \ge 0 \ \forall p\in P\ \forall r\in R & & \end{aligned}$$

$$\begin{aligned} \max \sum_{p\in P} y_p & & \\ \text{s.t.} \sum_{p'\in P: d(p,p')\le r} y_{p'} \le r+z \ \forall p\in P, r\in R & & (D) \\ y_p \ge 0 \qquad \forall p\in P & & \end{aligned}$$

We select a point c randomly and raise its dual variable y_c. Unlike [CP04], we raise y_c only until the constraint for c and some radius r becomes *half-tight*, i.e., $\sum_{p':d(c,p')\le r} y_{p'} = r/2+z$. We show that then we can guarantee that no other dual constraint is violated, which saves running time since we need to check only the (few, i.e. $|R|$ many) dual constraints for c, and not the constraints for all other input points when we raise y_c. We add c to our solution and assign it a radius of $2r$. Since the constraint for (c,r) is half-tight, our dual can still "pay" for including c with radius $2r$ in our solution.

In primal-dual algorithms, one often raises all primal variables whose constraints have become tight, in particular in the corresponding routine in [CP04]. However, since we assign to c a radius of $2r$, we argue that we do not need to raise the up to $\Omega(n)$ many primal variables corresponding to dual constraints that have become (half-)tight. Again, this saves a considerable amount of running time. Then, we consider the points that are not covered by c (with radius $2r$), select one of these points uniformly at random, and iterate. After a suitable pruning routine (which is faster than the corresponding routine in [CP04]) this process opens $k' = O(k/\epsilon)$ centers at cost $(6+\epsilon)\,\mathrm{OPT}'$, or asserts that $\mathrm{OPT} > \mathrm{OPT}'$. We show that we can maintain this solution dynamically.

As mentioned above, after each update we feed the centers of the bi-criteria approximation as input points to our (arbitrary) static offline α-approximation algorithm for k-sum-of-radii and run it. Finally, we translate its solution to a $(6+2\alpha+\epsilon)$-approximate solution to the original instance. For the best known offline approximation algorithm [CP04] it holds that $\alpha = 3.504$, and thus we obtain a ratio of $13.008 + \epsilon$ overall. For k-sum-of-diameters the same solution yields a $(26.016 + 2\epsilon)$-approximation.

1.3 Other Related Work

k-means and k-median. For k-means [HK20] gave a conditional lower bound against an oblivious adversary for any dynamic better-than-4 approximate k-means algorithm showing that for any $\gamma > 0$ no algorithm with $O(k^{1-\gamma})$ update time and $O(k^{2-\gamma})$ query time exists. There exists a fully dynamic constant approximation algorithm for k-means and k-median with expected amortized update time $\tilde{O}(n + k^2)$ that tries to minimize the number of center changes [CAHP+19]. For d-dimensional Euclidean metric space dynamic algorithms for k-median and k-means exist by combining dynamic coreset algorithms for that metric space with a static algorithm, but their running time is in $O(poly(\epsilon^{-1}, k, \log \Delta, d))$ [FS05, FSS13, BFL+17, SW18].

k-sum-of-radii. For a variant of the sum-of-radii problem [HLM20] gives a fully dynamic algorithm in metric spaces with doubling dimension κ that achieves a $O(2^{2^\kappa})$ approximation in time $O(2^{6\kappa} \log \Delta)$.

2 Preliminaries

We begin with basic notation and definitions. For any positive integer n, we write $[n]$ to denote the set $\{1, 2, \dots, n\}$. In what follows, we will fix any metric space $(\mathcal{X}, d)$. A fully dynamic stream is a sequence $(p_1, \sigma_1), \dots, (p_M, \sigma_M)$ of M updates such that $p_i \in \mathcal{X}$ is a point, and $\sigma_i \in \{+, -\}$ signifies either an insertion or deletion of a point. Naturally, we assume that a point can only be deleted if it was previously inserted. Moreover, we assume that each point is inserted at most once before being deleted.

We call a point $p \in \mathcal{X}$ active at time t if p was inserted at some time $t' < t$, and not deleted anytime between t' and t. We write $P^t \subset \mathcal{X}$ to denote the set of active points at time t. We let $r_{\min}, r_{\max}$ be reals such that for all $t \in [M]$ and $x, y \in P^t$, we have $r_{\min} \leq d(x, y) \leq r_{\max}$, and set $\Delta = r_{\max}/r_{\min}$ to be the aspect ratio of the point set. As in prior works [CGS18, SS19], we assume that an upper bound on Δ is known.

Clustering Objectives. In the k-center problem, given a point set P living in a metric space $(\mathcal{X}, d)$, the goal is to output a set of k *centers* $\mathcal{C} = \{c_1, \dots, c_k\} \subset \mathcal{X}$, along with a mapping $\ell : P \to \{c_1, \dots, c_k\}$, such that the following objective is minimized:

$$\text{cost}_k^\infty(\mathcal{C}, P) = \max_{p \in P} d(p, \ell(p))$$

In other words, one would like for the maximum distance from p to the $\ell(p)$, over all $p \in P$, to be minimized. A related objective is k-sum-of-radii, where the maximum distance to a point is considered for each cluster and added instead of taking the maximum:

$$\text{cost}_k^{R\Sigma}(\mathcal{C}, P) = \sum_{c \in \mathcal{C}} \max_{p \in \ell^{-1}(c)} d(p, c)$$

For the k-sum-of-diameters objective, one considers the pairwise distances of points assigned to the same clusters instead of the distances from each point to its center.

$$\text{cost}_k^{D\Sigma}(\mathcal{C}, P) = \sum_{c \in \mathcal{C}} \max_{p, q \in \ell^{-1}(c)} d(p, q)$$

Additionally, for any real $z > 0$, we introduce the (k, z)-clustering problem, which is to minimize

$$\text{cost}_k^z(\mathcal{C}, P) = \sum_{p \in P} d^z(p, \ell(p))$$

We will be primarily concerned the the k-center objective, but we introduce the more general (k, z)-clustering objective, which includes both k-median (for $z = 1$) and k-means (for $z = 2$), as our lower bounds from Section 8 will hold for these objectives as well.

We remark that while $\ell(p)$ is usually fixed by definition to be the closest point to p in C, the closest point may not necessarily be easy to maintain in a fully dynamic setting. Therefore, we will evaluate the cost of our algorithms with respects to both the centers and the mapping ℓ from points to centers. For any $p \in P$, we will refer to $\ell(p)$ as the *leader* of p under the mapping ℓ, and the set of all points lead by a given c_i is the cluster led by c_i.

In addition to maintaining a clustering with approximately optimal cost, we would like for our algorithms to be able to quickly answer queries related to cluster membership, and enumeration over all points in a cluster. Specifically, we ask that our algorithm be able to answer the following queries at any time step t:

Queries to a Fully Dynamic Clustering Algorithm

1. **Membership Query:** Given a point $p \in P^t$, return the center $c = \ell(p)$ of the cluster C containing p.
2. **Cluster Enumeration:** Given a point $p \in P^t$, list all points in the cluster C containing p.

In particular, after processing any given update, our algorithms will be capable of responding to membership queries in $O(1)$-time, and to clustering enumeration queries in time $O(|C|)$, where C is the clustering containing the query point.

3 From Fully Dynamic k-Center to k-Bounded MIS

In this section, we describe our main results for fully dynamic k-center clustering, based on our main algorithmic contribution, which is presented in Section 4. We begin by describing how the problem of k-center clustering of P can be reduced to maintaining a maximal independent set (MIS) in a graph. In particular, the reduction will only require us to solve a weaker version of MIS, where we need only return a MIS of size at most k, or an independent set of size at least $k + 1$. Formally, this problem, which we refer to as the k-Bounded MIS problem, is defined as follows.

Definition 7 (k-Bounded MIS)**.** *Given a graph $G = (V, E)$ and an integer $k \geq 1$, the k-bounded MIS problem is to output a maximal independent set $\mathcal{I} \subset V$ of size at most k, or return an independent set $\mathcal{I} \subset V$ of size at least $k + 1$.*

Reduction from k-center to k-Bounded MIS. The reduction from k-center to computing a maximum independent set in a graph is well-known, and can be attributed to the work of Hochbaum and Shmoys [HS86]. The reduction was described in Section 1.2, however, both for completeness and so that it is clear that only a k-Bounded MIS is required for the reduction, we spell out the full details here.

Fix a set of points X in a metric space, such that $r_{\min} \leq d(x,y) \leq r_{\max}$ for all $x,y \in \mathcal{X}$. Then, for each $r = r_{\min}, (1+\epsilon/2)r_{\min}, (1+\epsilon/2)^2 r_{\min}, \dots, r_{\max}$, one creates the *$r$-threshold graph* $G_r = (V, E_r)$, which is defined as the graph with vertex set $V = X$, and $(x,y) \in E_r$ if and only if $d(x,y) \leq r$. One then runs an algorithm for k-Bounded MIS on each graph G_r, and finds the smallest value of r such that the output of the algorithm $\mathcal{I}_r$ on G_r satisfies $|\mathcal{I}_r| \leq k$ — in other words, $\mathcal{I}_r$ must be a MIS of size at most k in G_r. Observe that $\mathcal{I}_r$ yields a solution to the k-center problem with cost at most r, since each point in X is either in $\mathcal{I}_r$ or is at distance at most r from a point in $\mathcal{I}_r$. Furthermore, since the independent set $\mathcal{I}_{r/(1+\epsilon/2)}$ returned from the algorithm run on $G_{r/(1+\epsilon/2)}$ satisfies $|\mathcal{I}_{r/(1+\epsilon/2)}| \geq k+1$ it follows that there are $k+1$ points in X which are pair-wise distance at least $r/(1+\epsilon/2)$ apart. Hence, the cost of any k-center solution (which must cluster two of these $k+1$ points together) is at least $r/(2+\epsilon)$ by the triangle inequality. It follows that $\mathcal{I}$ yields a $2+\epsilon$ approximation of the optimal k-center cost.

Note that, in addition to maintaining the centers $\mathcal{I}$, for the purposes of answering membership queries, one would also like to be able to return in $O(1)$ time, given any $x \in \mathcal{X}$, a fixed $y \in \mathcal{I}$ such that $d(x,y) \leq r$. We will ensure that our algorithms, whenever they return a MIS $\mathcal{I}$ with size at most k, also maintain a mapping $\ell : V \setminus \mathcal{I} \to \mathcal{I}$ which maps any x, which is not a center, to its corresponding center $\ell(x)$.

Observe that in the context of k-clustering, insertions and deletions of points correspond to insertions and deletions of entire vertices into the graph G. This is known as the fully dynamic *vertex update model*. Since one vertex update can cause as many as $O(n)$ edge updates, we will not be able to read all of the edges inserted into the stream. Instead, we assume our dynamic graph algorithms can query for whether (u,v) is an edge in constant time (i.e., constant time oracle access to the adjacency matrix).[4]

Our algorithm for k-Bounded MIS will return a very particular type of MIS. Specifically, we will attempt to return the first $k+1$ vertices in a *Lexicographically First MIS* (LFMIS), under a random lexicographical ordering of the vertices.

Lexicographically First MIS (LFMIS). The LFMIS of a graph $G = (V,E)$ according to a ranking of the vertices specified by a mapping $\pi : V \to [0,1]$ is a unique MIS defined as by the following process. Initially, every vertex $v \in V$ is alive. We then iteratively select the alive vertex with minimal rank $\pi(v)$, add it to the MIS, and then kill v and all of its alive neighbors. We write $\mathrm{LFMIS}(G,\pi)$ to denote the LFMIS of G under π. For each vertex v, we define the *eliminator* of v, denoted $\mathrm{elim}_{G,\pi}(v)$ to be the vertex u which kills v in the above process; namely, $\mathrm{elim}_{G,\pi}(v)$ is the vertex with smallest rank in the set $(N(v) \cup \{v\}) \cap \mathrm{LFMIS}(G,\pi)$.

Definition 8. *Given a graph $G = (V,E)$, $\pi : V \to [0,1]$, and an integer $k \geq 1$, we define the top-k LFMIS of G with respect to π, denoted $\mathrm{LFMIS}_k(G,\pi)$ to be the set consisting of the first $\min\{k, |\mathrm{LFMIS}(G,\pi)|\}$ vertices in $\mathrm{LFMIS}(G,\pi)$ (where the ordering is with respect to π). When G,π are given by context, we simply write LFMIS_k.*

It is clear that returning a top-$(k+1)$ LFMIS of G with respect to any ordering will solve the k-Bounded MIS problem. In order to also obtain a mapping ℓ from points to their centers in the independent set, we define the following augmented version of the top-k LFMIS problem, which we refer to as a *top-k LFMIS with leaders.*

Definition 9. *A top-k LFMIS with leaders consists of the set $\mathrm{LFMIS}_k(G,\pi)$, along with a leader mapping function $\ell : V \to V \cup \{\bot\}$, such that $(v, \ell(v)) \in E$ whenever $\ell(v) \neq \bot$, and such that*

[4]This is equivalent to assuming that distances in the metric space can be computed in constant time, however if such distances require α time to compute, this will only increase the runtime of our algorithms by a factor of α.

if $\mathrm{LFMIS}_k(G,\pi) = \mathrm{LFMIS}(G,\pi)$*, then* $\ell(v) \in \mathrm{LFMIS}_k(G,\pi)$ *for all* $v \in V \setminus \mathrm{LFMIS}_k(G,\pi)$*, and* $\ell(v) = \perp$ *for all* $v \in \mathrm{LFMIS}_k(G,\pi)$*.*

Within our algorithm, the event that $\ell(v) = \perp$ will occur only if v itself is a leader in LFMIS_k, or if v is among a set of "unclustered" points whose leader mapping to a point in LFMIS may be out of date. We choose to set $\ell(v) = \perp$ when $v \in \mathrm{LFMIS}_k$ is a leader, rather than setting $\ell(v) = v$, to maintain the invariant that $(v, \ell(v)) \in E$ whenever $\ell(v) \neq \perp$.

The main goal of the following Section 4 will be to prove the existence of a $\tilde{O}(k)$ amortized update time algorithm for maintaining a top-k LFMIS with leaders of a graph G under a fully dynamic sequence of insertions and deletions of vertices from G. Specifically, we will prove the following theorem.

Theorem 22. *There is a algorithm which, on a fully dynamic stream of insertions and deletions of vertices to a graph G, maintains at all time steps a top-k LFMIS of G with leaders under a random ranking $\pi : V \to [0,1]$. The expected amortized per-update time of the algorithm is $O(k \log n + \log^2 n)$, where n is the maximum number active of vertices at any time. Moreover, the algorithm does not need to know n in advance.*

Next, we demonstrate how Theorem 22 immediately implies the main result of this work (Theorem 1). Firstly, we prove a proposition which demonstrates that any fully dynamic Las Vegas algorithm with small runtime in expectation can be converted into a fully dynamic algorithm with small runtime with high probability.

Proposition 10. *Let $\mathcal{A}$ be any fully dynamic randomized algorithm that correctly maintains a solution for a problem $\mathcal{P}$ at all time steps, and runs in amortized time at most $\mathrm{Time}(\mathcal{A})$ in expectation. Then there is a fully dynamic algorithm for $\mathcal{P}$ which runs in amortized time at most $O(\mathrm{Time}(\mathcal{A}) \log \delta^{-1})$ with probability $1-\delta$ for all $\delta \in (0, 1/2)$.*

Proof. The algorithm is as follows: we maintain at all time steps a single instance of $\mathcal{A}$ running on the dynamic stream. If, whenever the current time step is t, the total runtime of the algorithm exceeds $4t\,\mathrm{Time}(\mathcal{A})$, we delete $\mathcal{A}$ and re-instantiate it with fresh randomness. We then run the re-instantiated version of $\mathcal{A}$ from the beginning of the stream until either we reach the current time step t, or the total runtime again exceeds $4t\,\mathrm{Time}(\mathcal{A})$, in which case we re-instantiate again.

Let M be the total length of the stream. For each $i = 0, 1, 2, \dots, \lceil \log M \rceil$, let $\mathbf{Z}_i$ be the number of times that $\mathcal{A}$ is re-instantiated while the current time step is between 2^i and 2^{i+1}. Note that the total runtime is then at most

$$4\,\mathrm{Time}(\mathcal{A}) \cdot \left(M + \sum_{i=0}^{\lceil \log M \rceil} 2^{i+1} \mathbf{Z}_i \right)$$

Fix any $i \in \{0, 1, \dots, \lceil \log M \rceil\}$, and let us bound the value $\mathbf{Z}_i$. Each time that $\mathcal{A}$ is restarted when the current time t step is between 2^i and 2^{i+1}, the probability that the new algorithm runs in time more than $2^{i+2}\,\mathrm{Time}(\mathcal{A}) \geq 4t\,\mathrm{Time}(\mathcal{A})$ on the first 2^{i+1} updates is at most $1/2$ by Markov's inequality. Thu, the probability that $\mathbf{Z}_i > T_i + 1$ is at most 2^{-T_i}, for any $T_i \geq 0$. Setting $T_i = \log(2/\delta) + \lceil \log M \rceil - i$, we have

$$\begin{aligned} \sum_{i=0}^{\lceil \log M \rceil} \mathbf{Pr}\left[\mathbf{Z}_i > T_i + 1\right] &\leq \frac{\delta}{2} \sum_{i=0}^{\lceil \log M \rceil} \left(\frac{1}{2}\right)^{\lceil \log M \rceil - i} \\ &< \delta \end{aligned} \tag{1}$$

In other words, by a union bound, we have $\mathbf{Z}_i \leq T_i + 1$ for all $i = 0, 1, 2, \dots, \lceil \log M \rceil$, with probability at least $1-\delta$. Conditioned on this, the total runtime is at most

$$\begin{aligned}4\,\mathrm{Time}(\mathcal{A})\cdot\left(M+\sum_{i=0}^{\lceil\log M\rceil}2^{i+1}(T_i+1)\right)&=O\left(\mathrm{Time}(\mathcal{A})\left(\sum_{i=0}^{\lceil\log M\rceil}2^{i}(\log\delta^{-1}+\lceil\log M\rceil-i)\right)\right)\\&=O\left(\mathrm{Time}(\mathcal{A})\left(M\log\delta^{-1}+M\sum_{i=0}^{\lceil\log M\rceil}\frac{i}{2^i}\right)\right)\\&=O\left(\mathrm{Time}(\mathcal{A})\cdot M\log\delta^{-1}\right)\end{aligned}\tag{2}$$

which is the desired total runtime. □

Given Theorem 22, along with the reduction to top-k LFMIS from k-center described in this section, which runs $O(\epsilon^{-1}\log\Delta)$ copies of a top-k LFMIS algorithm, we immediately obtain a fully dynamic k-center algorithm with $\tilde{O}(k)$ expected amortized update time. By then applying Proposition 10, we obtain our main theorem, stated below.

Theorem 1. *There is a fully dynamic algorithm which, on a sequence of insertions and deletions of points from a metric space $\mathcal{X}$, maintains a $(2+\epsilon)$-approximation to the optimal k-center clustering. The amortized update time of the algorithm is $O(\frac{\log\Delta\log n}{\epsilon}(k+\log n))$ in expectation, and $O(\frac{\log\Delta\log n}{\epsilon}(k+\log n)\log\delta^{-1})$ with probability $1-\delta$ for any $\delta\in(0,\frac{1}{2})$, where n is the maximum number of active points at any time step.*

The algorithm can answer membership queries in $O(1)$-time, and enumerate over a cluster C in time $O(|C_i|)$.

4 Fully Dynamic k-Bounded MIS with Vertex Updates

Given the reduction from Section 3, the goal of this section will be to design an algorithm which maintains a top-k LFMIS with leaders (Definition 9) under a graph which receives a fully dynamic sequence of *vertex* insertions and deletions. As noted, maintaining a top-$(k+1)$ LFMIS immediately results in a solution to the k-Bounded MIS problem. We begin by formalizing the model of vertex-valued updates to dynamic graphs.

Fully Dynamic Graphs with Vertex Updates In the vertex-update fully dynamic setting, at each time step a vertex v is either inserted into the current graph G, or deleted from G, along with all edges incident to v. This defines a sequence of graphs $G^1, G^2, \ldots, G^M$, where $G^t=(V^t,E^t)$ is the state of the graph after the t-th update. Equivalently, we can think of there being an "underlying" graph $G=(V,E)$, where at the beginning all vertices are *inactive*. At each time step, either an active vertex is made inactive, or vice-versa, and G^t is defined as the subgraph induced by the active vertices at time t. The latter is the interpretation which will be used for this section.

Since the degree of v may be as large as the number of active vertices in G, our algorithm will be unable to read all of the edges incident to v when it arrives. Instead, we require only query access to the adjacency matrix of the underlying graph G. Namely, we assume that we can test in constant time whether $(u,v)\in E$ for any two vertices u,v.

For the purpose of k-center clustering, we will need to maintain a top-k LFMIS with leaders $(\mathrm{LFMIS}_k(G,\pi),\ell)$, along with a Boolean value indicating whether $\mathrm{LFMIS}_k(G,\pi)=\mathrm{LFMIS}(G,\pi)$. To do this, we can instead attempt to maintain the set $\mathrm{LFMIS}_{k+1}(G,\pi)$, as well as a leader mapping function $\ell : V \to V \cup \{\perp\}$, with the relaxed property that if $\text{LFMIS}_k(G, \pi) = \text{LFMIS}(G, \pi)$, then $\ell(v) \in \text{LFMIS}(G, \pi)$ for all $v \in V \setminus \text{LFMIS}(G, \pi)$ and $\ell(v) = \perp$ for all $v \in \text{LFMIS}(G, \pi)$. We call such a leader function ℓ with this relaxed property a *modified* leader mapping. Thus, in what follows, we will focus on maintaining a top-k LFMIS with this modified leader mapping.

4.1 The Data Structure

We now describe the main data structure and algorithm which will maintain a top-k LFMIS with leaders in the dynamic graph G. We begin by fixing a random mapping $\pi : V \to [0, 1]$, which we will use as the ranking for our lexicographical ordering over the vertices. It is easy to see that if $|V| = n$, then by discretizing $[0, 1]$ so that $\pi(v)$ can be represented in $O(\log n)$ bits we will avoid collisions with high probability. At every time step, the algorithm will maintain an ordered set $\mathcal{L}_{k+1}$ of vertices in a linked list, sorted by the ranking π, with $|\mathcal{L}_{k+1}| \leq k + 1$. We will prove that, after every update t, we have $\mathcal{L}_{k+1} = \text{LFMIS}_{k+1}(G^t, \pi)$.[5] We will also maintain a mapping $\ell : V \to V \cup \{\perp\}$ which will be our leader mapping function. Initially, we set $\ell(v) = \perp$ for all v. Lastly, we will maintain a (potentially empty) priority queue $\mathcal{Q}$ of *unclustered* vertices, where the priority is similarly given by π.

Each vertex v in G_t will be classified as either a *leader*, a *follower*, or *unclustered*. Intuitively, when $\text{LFMIS}_k(G, \pi) = \text{LFMIS}(G, \pi)$, the leaders will be exactly the points in $\text{LFMIS}_k(G, \pi)$, the followers will be all other points v which are mapped to some $\ell(v) \in \text{LFMIS}_k(G, \pi)$ (in other words, v "follows" $\ell(v)$), and there will be no unclustered points. At intermediate steps, however, when $|\text{LFMIS}(G, \pi)| \geq k+1$, we will be unable to maintain the entire set $\text{LFMIS}(G, \pi)$ and, therefore, we will store the set of all vertices which are not in LFMIS_{k+1} in the priority queue $\mathcal{Q}$ of unclustered vertices. The formal definitions of leaders, followers, and unclustered points follow.

Every vertex currently maintained in $\mathcal{L}_{k+1}$ is be a leader. Each leader v may have a set of follower vertices, which are vertices u with $\ell(u) = v$, in which case we say that u follows v. By construction of the leader function ℓ, every follower-leader pair $(u, \ell(u))$ will be an edge of G. We write $\mathcal{F}_v = \{u \in V : \ell(u) = v\}$ to denote the (possibly empty) set of followers of a leader v. For each leader, the set $\mathcal{F}_v$ will be maintained as part of the data structure at the vertex v.

Now when the size of LFMIS exceeds $k + 1$, we will have to remove the leader v in LFMIS with the largest rank, so as to keep the size of $\mathcal{L}_{k+1}$ at most $k + 1$. The vertex v will then be moved to the queue $\mathcal{Q}$, along with its priority $\pi(v)$. The set $\mathcal{F}_v$ of followers of v will continue to be followers of v — their status remains unchanged. In this case, the vertex v is now said to be an *inactive* leader, whereas each leader currently in $\mathcal{L}_{k+1}$ is called an *active* leader. If, at a later time, we have $\pi(v) < \max_{u \in \mathcal{L}_{k+1}} \pi(u)$, then it is possible that v may be part of LFMIS_{k+1}, in which case we will attempt to reinsert the inactive leader v from $\mathcal{Q}$ back into $\mathcal{L}_{k+1}$. Note, importantly, that whenever $\pi(v) < \max_{u \in \mathcal{L}_{k+1}} \pi(u)$ occurs at a future time step for a vertex $v \in \mathcal{Q}$, then either v is part of LFMIS_{k+1}, or it is a neighbor of some vertex $u \in \text{LFMIS}_{k+1}$ of lower rank. In both cases, we can remove v from $\mathcal{Q}$ and attempt to reinsert it, with the guarantee that after this reinsertion v will either be an active leader, or a follower of an active leader.

The Unclustered Queue. We now describe the purpose and function of the priority queue $\mathcal{Q}$. Whenever either a vertex v is inserted into the stream, or it is a follower of a leader $\ell(v)$ who is removed from the LFMIS, we must attempt to reinsert v, to see if it should be added to LFMIS_{k+1}. However, if $|\mathcal{L}_{k+1}| = k + 1$, then the only way that v should be a part of LFMIS_{k+1} (and therefore added to $\mathcal{L}_{k+1}$) is if $\pi(v) < \max_{u \in \mathcal{L}_{k+1}} \pi(u)$. If this does not occur, then we do not

[5] We use a separate notation $\mathcal{L}_{k+1}$, instead of LFMIS_{k+1}, to represent the set maintained by the algorithm, until we have demonstrated that we indeed have $\mathcal{L}_{k+1} = \text{LFMIS}_{k+1}(G^t, \pi)$ at all time steps t.

need to insert v right away, and instead can defer it to a later time when either $|\mathcal{L}_{k+1}| < k+1$ or $\pi(v) < \max_{u \in \mathcal{L}_{k+1}} \pi(u)$ holds. Moreover, by definition of the modified leader mapping ℓ, we only need to set $\ell(v)$ when $|\operatorname{LFMIS}_{k+1}| < k+1$. We can therefore add v to the priority queue $\mathcal{Q}$.

Every point in the priority queue is called an *unclustered point*, as they are not currently part of a valid k-clustering in the graph. By checking the top of the priority queue at the end of processing each update, we can determine whenever either of the events $|\mathcal{L}_{k+1}| \le k$ or $\pi(v) < \max_{u \in \mathcal{L}_{k+1}} \pi(u)$ holds; if either is true, we iteratively attempt to reinsert the top of the queue until the queue is empty or both events no longer hold. This will ensure that either all points are clustered (so $\mathcal{Q} = \emptyset$), or $|\mathcal{L}_{k+1}| = k+1$ and $\mathcal{L}_{k+1} = \operatorname{LFMIS}_{k+1}$ (since no point in the queue could have been a part of $\operatorname{LFMIS}_{k+1}$).

The Leader Mapping. Notice that given a top-k LFMIS, a valid leader assignment is always given by $\ell(v) = \operatorname{elim}_{G,\pi}(v)$, where $\operatorname{elim}_{G,\pi}(v)$ is the eliminator of v via π as defined in Section 3; this is the case since if $\operatorname{LFMIS}_k(G,\pi) = \operatorname{LFMIS}(G,\pi)$ then each vertex is either in $\operatorname{LFMIS}_k(G,\pi)$ or eliminated by one of the vertices in $\operatorname{LFMIS}_k(G,\pi)$. Thus, intuitively, our goal should be to attempt to maintain that $\ell(v) = \operatorname{elim}_{G,\pi}(v)$ for all $v \notin \operatorname{LFMIS}_k(G,\pi)$. However, the addition of a new vertex which enters $\operatorname{LFMIS}_k(G,\pi)$ can change the eliminators of many other vertices *not* in $\operatorname{LFMIS}_k(G,\pi)$. Discovering which points have had their eliminator changed immediately on this time step would be expensive, as one would have to search through the followers of all active leaders to see if any of their eliminators changed. Instead, we note that at this moment, so long as the new vertex does not share an edge with any other active leader, we do not need to modify our leader mapping. Instead, we can *defer* the reassignment of the leaders of vertices v whose eliminator changed on this step, to a later step when their leaders are removed from $\operatorname{LFMIS}_{k+1}(G,\pi)$. Demonstrating that the number of changes to the leader mapping function ℓ, defined in this way, is not too much larger than the number of changes to the eliminators of all vertices, will be a major component of our analysis.

The Algorithm and Roadmap. Our main algorithm is described in three routines: Algorithms 1, 2, and 3. Algorithm 1 handles the inital insertion or deletion of a vertex in the fully dynamic stream, and then calls at least one of Algorithms 2 or 3. Algorithm 2 handles insertions of vertices in the data structure, and Algorithm 3 handles deletions of vertices from the data structure. We begin in Section 4.2 by proving that our algorithm does indeed solve the top-k LFMIS problem with the desired modified leader mapping. Then, in Section 4.3, we analyze the amortized runtime of the algorithm.

```
Algorithm 1: Process Update
  Data: An update $(v,\sigma)$, where $\sigma \in \{+,-\}$.
1 if $\sigma = +$ is an insertion of $v$ then
2     Generate $\pi(v)$, and set $\ell(v) = \perp$.
3     Call Insert$(v, \pi(v))$.
4 if $\sigma = -$ is a deletion of $v$ then
5     Call Delete$(v)$.
6 while $|\mathcal{Q}| \neq \emptyset \wedge \big(|\mathcal{L}_{k+1}| \le k+1 \vee \min_{w \in \mathcal{Q}} \pi(w) < \max_{w \in \mathcal{L}_{k+1}} \pi(w)\big)$ do
7     $u \leftarrow \arg\min_{w \in \mathcal{Q}} \pi(w)$.
8     Delete $u$ from $\mathcal{Q}$, and call Insert$(u, \pi(u))$.
```

```
Algorithm 2: Insert$(v, \pi(v))$
1  if $|\mathcal{L}_{k+1}| = k+1 \wedge \pi(v) > \max_{u \in \mathcal{L}_{k+1}} \pi(u)$ then
2      Insert $(v, \pi(v))$ into $\mathcal{Q}$.
3  else
4      Compute $S = \mathcal{L}_{k+1} \cap N(v)$
5      if $S = \emptyset$ then
6          Add $v$ to $\mathcal{L}_{k+1}$.
7          if $|\mathcal{L}_{k+1}| = k+2$ then
8              Let $u = \arg\max_{u' \in \mathcal{L}_{k+1}} \pi(u')$.
9              Remove $u$ from $\mathcal{L}_{k+1}$, and insert $(u, \pi(u))$ into $\mathcal{Q}$.
10     else
11         $u^* = \arg\min_{u' \in S} \pi(u')$
12         if $\pi(u^*) < \pi(v)$ then
13             if $v$ is a leader then
14                 For each $w \in \mathcal{F}_v$, insert $(w, \pi(w))$ to $\mathcal{Q}$, and set $\ell(w) = \perp$
15                 Delete the list $\mathcal{F}_v$.
16             Add $v$ to $\mathcal{F}_{u^*}$ as a follower of $u^*$, set $\ell(v) = u^*$.
17         else
18             For each $w \in \cup_{u \in S} \mathcal{F}_u$, add $(w, \pi(w))$ to $\mathcal{Q}$, and set $\ell(w) = \perp$.
19             For each $u \in S$, set $\ell(u) = v$ to be a follower of $v$, remove $u$ from $\mathcal{L}_{k+1}$, and
               delete the list $\mathcal{F}_u$.
```

```
Algorithm 3: Delete$(v)$
1  if $v$ is a follower then
2      Delete $v$ from $\mathcal{F}_{\ell(v)}$, and remove $v$ from the set of vertices.
3  else if $v \in \mathcal{Q}$ then
4      if $v$ is a leader then
5          For each $w \in \mathcal{F}_v$, insert $(w, \pi(w))$ to $\mathcal{Q}$, and set $\ell(w) = \perp$
6          Delete the list $\mathcal{F}_v$, and remove $v$ from $\mathcal{Q}$ and the set of vertices.
7      else
8          Delete $v$ from $\mathcal{Q}$ and the set of vertices.
9  else                                                    /* Must have $v \in \mathcal{L}_{k+1}$ */
10
11     For each $w \in \mathcal{F}_v$, insert $(w, \pi(w))$ to $\mathcal{Q}$, and set $\ell(w) = \perp$
12     Delete the list $\mathcal{F}_v$, and remove $v$ from $\mathcal{L}_{k+1}$ and the set of vertices.
```

4.2 Correctness of the Algorithm

We will now demonstrate the correctness of the algorithm, by first proving two Propositions.

Proposition 11. *After every time step t, the set $\mathcal{L}_{k+1}$ stored by the algorithm is an independent set in $G^{(t)}$.*

Proof. Suppose otherwise, and let $v, u \in \mathcal{L}_{k+1}$ be vertices with $(v, u) \in E$. WLOG we have that v is the vertex which entered $\mathcal{L}_{k+1}$ most recently of the two. Then we had $u \in ALG$ at the moment that Insert$(v, \pi(v))$ was most recently called. Since on the step that Insert$(v, \pi(v))$ was most recently called the vertex p was added to $\mathcal{L}_{k+1}$, it must have been that $\min_{w\in N(v)\cap\mathcal{L}_{k+1}} \pi(w) > \pi(v)$, thus, in particular, we must have had $\pi(v) < \pi(u)$. However, in this case we would have made u a follower of v at this step and set $\ell(u) = v$, which could not have occurred since then u would have been removed from $\mathcal{L}_{k+1}$, which completes the proof. ☐

Claim 12. *We always have $|\mathcal{L}_{k+1}| \le k+1$ at all time steps.*

Proof. After the first insertion the result is clear. We demonstrate that the claim holds inductively after each insertion. Only a call to `Insert`(v) can increase the size of $\mathcal{L}_{k+1}$, so consider any such call. If $N(v) \cap \mathcal{L}_{k+1}$ is empty, then we add v to $\mathcal{L}_{k+1}$, which can possibly increase its size to $k+2$ if it previously had $k+1$ elements. In this case, we remove the element with largest rank and add it to $\mathcal{Q}$, maintaining the invariant. If there exists a $u^* \in N(v)\cap\mathcal{L}_{k+1}$ with smaller rank than v, we make v the follower of the vertex in $N(v)\cap\mathcal{L}_{k+1}$ with smallest rank, in which case the size of $\mathcal{L}_{k+1}$ is unaffected. In the final case, all points in $N(v)\cap\mathcal{L}_{k+1}$ have larger rank than v, in which case all of $N(v)\cap\mathcal{L}_{k+1}$ (which is non-empty) is made a follower of v and removed from $\mathcal{L}_{k+1}$, thereby decreasing or not affecting the size of $\mathcal{L}_{k+1}$, which completes the proof. ☐

Proposition 13 (Correctness of Leader Mapping)**.** *At any time step, if $\mathcal{L}_{k+1} \le k$ then every vertex $v \in V$ is either contained in $\mathcal{L}_{k+1}$, or has a leader $\ell(v) \in \mathcal{L}_{k+1}$ with $(v, \ell(v)) \in E$.*

Proof. After processing any update, we first claim that if $\mathcal{L}_{k+1} \le k$ we have $\mathcal{Q} = \emptyset$. This follows from the fact that the while loop in Line 6 of Algorithm 1 does not terminate until one of these two conditions fails to hold. Thus if $\mathcal{L}_{k+1} \le k$, every vertex $v \in V$ is either contained in $\mathcal{L}_{k+1}$ (i.e., an active leader), or is a follower of such an active leader, which completes the proof of the proposition, after noting that we only set $\ell(u) = v$ when $(u,v) \in E$ is an edge. ☐

Lemma 14 (Correctness of the top-k LFMIS)**.** *After every time step, we have $\mathcal{L}_{k+1} = \mathrm{LFMIS}_{k+1}(G,\pi)$.*

Proof. Order the points in $\mathcal{L}_{k+1} = (v_1,\dots,v_r)$ and $\mathrm{LFMIS}_{k+1}(G,\pi) = (u_1,\dots,u_s)$ by rank. We prove inductively that $v_i = u_i$. Firstly, note that u_1 is the vertex with minimal rank in G. As a result, u_1 could not be a follower of any point, since we only set $\ell(u) = v$ when $\pi(v) < \pi(u)$. Thus u_1 must either be an inactive leader (as it cannot be equal to v_j for $j > 1$) or an unclustered point. In both cases, one has $u_1 \in \mathcal{Q}$, which we argue cannot occur. To see this, note that at the end of processing the update, the while loop in Line 6 of Algorithm 1 would necessarily remove u_1 from $\mathcal{Q}$ and insert it. It follows that we must have $u_1 = v_1$.

In general, suppose we have $v_i = u_i$ for all $i \le j$ for some integer $j < s$. We will prove $v_{j+1} = u_{j+1}$. First suppose $r, s \ge j+1$. Now by definition of the LSFMIS, the vertex u_{j+1} is the smallest ranked vertex in $V \setminus \cup_{i\le j} N(v_i) \cup \{u_i\}$. Since we only set $\ell(u) = v$ when $(u,v) \in E$ is an edge, it follows that u_{j+1} cannot be a follower of $u_i = v_i$ for any $i \le j$. Moreover, since we only set $\ell(u) = v$ when $\pi(v) < \pi(u)$, it follows that u_{j+1} cannot be a follower of v_i for any $i > j$, since $\pi(v_i) \ge \pi(u_{j+1})$ for all $i > j$. Thus, if $v_{j+1} \ne u_{j+1}$, it follows that either $u_{j+1} \in \mathcal{Q}$, or u_{j+1} is a follower of some vertex $u' \in \mathcal{Q}$ with smaller rank than u_{j+1}. Then, similarly as above, in both cases the while loop in Line 6 of Algorithm 1 would necessarily remove u_{j+1} (or u' in the latter case) from $\mathcal{Q}$ and insert it, because $\pi(u_{j+1}) < \pi(v_r)$, and in the latter case if such a u' existed we would have $\pi(u') < \pi(u_{j+1}) < \pi(u_r)$. We conclude that $v_{j+1} = u_{j+1}$.

The only remaining possibility is $r \ne s$. First, if $r > s$, by Claim 12 we have $r \le k+1$, and by Proposition 11 $\mathcal{L}_{k+1}$ forms a independent set. Thus $v_1, v_2, \dots, v_s, v_{s+1}$ is an independent set, but since $v_i = u_i$ for $i \le s$ and $\mathrm{LFMIS}_{k+1}(G,\pi) = \{u_1,\dots,u_s\}$ is a maximal independent set whenever $s \le k$, this yields a contradiction. Finally, if $r < s$, consider the vertex u_{r+1}. Since $u_i = v_i$ for all $i \le r$, u_{r+1} cannot be a follower of v_i for any $i \in [r]$. As a result, it must be that either $u_{r+1} \in \mathcal{Q}$ or u_{r+1} is a follower of a vertex in $\mathcal{Q}$. In both cases, at the end of the last update, we had $|\mathcal{L}_{k+1}| = r \le k$ and $\mathcal{Q} \ne \emptyset$, which cannot occur as the while loop in Line 6 of Algorithm 1 would not have terminated. It follows that $r = s$, which completes the proof. $\square$

4.3 Amortized Update Time Analysis

We now demonstrate that the above algorithm runs in amortized $\tilde{O}(k)$-time per update. We begin by proving a structural result about the behavior of our algorithm. In what follows, let G^t be the state of the graph *after* the t-th update. Similarly, let $\ell_t(v) \in V \cup \{\perp\}$ be the value of $\ell(v)$ after the t-th update.

Proposition 15. *Let $\texttt{Insert}(v_1, \pi(v_1)), \texttt{Insert}(v_2, \pi(v_2)), \dots, \texttt{Insert}(v_r, \pi(v_r))$ be the ordered sequence of calls to the* `Insert` *function (Algorithm 2) which take places during the processing of any individual update in the stream. Then we have $\pi(v_1) < \pi(v_2) < \cdots < \pi(v_r)$. As a corollary, for any vertex v the function $\texttt{Insert}(v, \pi(v))$ is called at most once per time step.*

Proof. Assume $r > 1$, since otherwise the claim is trivial. To prove the proposition it will suffice to show two facts: (1) whenever a call to $\texttt{Insert}(v_i, \pi(v_i))$ is made, $\pi(v_i)$ is smaller than the rank of all vertices in the queue $\mathcal{Q}$, and (2) a call to $\texttt{Insert}(v_i, \pi(v_i))$ can only result in vertices with larger rank being added to $\mathcal{Q}$.

To prove (1), note that after the first call to $\texttt{Insert}(v_1, \pi(v_1))$, which may have been triggered directly as a result of v_1 being added to the stream at that time step, all subsequent calls to `Insert` can only be made via the while loop of Line 6 in Algorithm 1, where the point with smallest rank is iteratively removed from $\mathcal{Q}$ and inserted. Thus, fact (1) trivially holds for all calls to `Insert` made in this while loop, and it suffices to prove it for $\texttt{Insert}(v_1, \pi(v_1))$ in the case that v_1 is added to the stream at the current update (if v_1 was added from the queue, the result is again clear). Now if $\mathcal{Q} \neq \emptyset$ at the moment $\texttt{Insert}(v_1, \pi(v_1))$ is called, it must be the case that $|\mathcal{L}_{k+1}| = k + 1$ and $\min_{w \in \mathcal{Q}} \pi(w) > \max_{w \in \mathcal{L}_{k+1}} \pi(w)$ (otherwise the queue would have been emptied at the end of the prior update). Thus, if it were in fact the case that $\pi(v_1) > \min_{w \in \mathcal{Q}} \pi(w)$, then we also have $\pi(v_1) > \max_{w \in \mathcal{L}_{k+1}} \pi(w)$, and therefore the call to $\texttt{Insert}(v_1, \pi(v_1))$ would result in inserting v_1 into $\mathcal{Q}$ on Line 1 of Algorithm 2. Such an update does not modify $\mathcal{L}_{k+1}$, and does not change the fact that $\min_{w \in \mathcal{Q}} \pi(w) > \max_{w \in \mathcal{L}_{k+1}} \pi(w)$, thus the processing of the update will terminate after the call to $\texttt{Insert}(v_1, \pi(v_1))$ (contradicting the assumption that $r > 1$), which completes the proof of (1).

To prove (2), note that there are only three ways for a call to $\texttt{Insert}(v_i, \pi(v_i))$ to result in a vertex u being added to $\mathcal{Q}$. In the first case, if $\ell(u)$ was an active leader which was made a follower of v_i as a result of $\texttt{Insert}(v_i, \pi(v_i))$, then we have $\pi(u) < \pi(\ell(u)) < \pi(v_i)$. Next, we could have had $\ell(u) = v_i$ (in the event that v_i was an inactive leader being reinserted from $\mathcal{Q}$), in which case $\pi(u) < \pi(v_i)$. Finally, it could be the case that u was the active leader in $\mathcal{L}_{k+1}$ prior to the call to $\texttt{Insert}(v_i, \pi(v_i))$, and was then removed from $\mathcal{L}_{k+1}$ as a result of the size of $\mathcal{L}_{k+1}$ exceeding $k + 1$ and u having the largest rank in $\mathcal{L}_{k+1}$. This can only occur if v_i was added to $\mathcal{L}_{k+1}$ and had smaller rank than u, which completes the proof of (2).

Since by (1) every time $\texttt{Insert}(v_i, \pi(v_i))$ is called $\pi(v_i)$ is smaller than the rank of all vertices in the queue, and by (2) the rank of all new vertices added to the queue as a result of $\texttt{Insert}(v_i, \pi(v_i))$ will continue to be larger than $\pi(v_i)$, it follows that v_{i+1}, which by construction must be the vertex with smallest rank in $\mathcal{Q}$ after the call to $\texttt{Insert}(v_i, \pi(v_i))$, must have strictly larger rank than v_i, which completes the proof of the proposition. $\square$

The following proposition is more or less immediate. It implies, in particular, that a point can only be added to $\mathcal{Q}$ once per time step (similarly, v can be removed from $\mathcal{Q}$ once per time step).

Proposition 16. *Whenever a vertex v in the queue $\mathcal{Q}$ is removed and $\texttt{Insert}(v, \pi(v))$ is called, the vertex v either becomes a follower of an active leader, or an active leader itself.*

Proof. If v shares an edge with a vertex in $\mathcal{L}_{k+1}$ with smaller rank, it becomes a follower of such a vertex. Otherwise, all vertices in $N(v) \cap \mathcal{L}_{k+1}$ become followers of v, and v becomes an active leader by construction (possibly resulting in an active leader of larger rank to be removed from $\mathcal{L}_{k+1}$ as a result of it no longer being contained in LFMIS_{k+1}). □

Equipped with the prior structural propositions, our approach for bounding the amortized update time is to first observe that, on any time step t, our algorithm only attempts to insert a vertex v, thereby spending $O(k)$ time to search for edges between v and all members of $\mathcal{L}_{k+1}$, if either v was the actual vertex added to the stream on step t, or when v was added to $\mathcal{Q}$ on step t or before. Thus, it will suffice to bound the total number of vertices which are ever added into $\mathcal{Q}$ — by paying a cost $O(k + \log n)$ for each vertex v which is added to the queue, we can afford both the initial $O(\log n)$ cost of adding it to the priority queue, as well as the $O(k)$ runtime cost of possibly later reinserting v during the while loop in Line 6 of Algorithm 1. We formalize this in the following proposition.

Proposition 17. *Let T be the total number of times that a vertex is inserted into the queue $\mathcal{Q}$ over the entire execution of the algorithm, where two insertions of the same vertex v on separate time steps are counted as distinct insertions. Then the total runtime of the algorithm, over a sequence of M insertions and deletions, is at most $O(T(k + \log n) + Mk)$, where n is the maximum number of vertices active at any given time.*

Proof. Note that the only actions taken by the algorithm consist of adding and removing vertices v from $\mathcal{Q}$ (modifying the value of $\ell(v)$ in the process, and possibly deleting $\mathcal{F}_v$), and computing $\mathcal{L}_{k+1} \cap N(v)$ for some vertex v. The latter requires $O(k)$ time since we have $|\mathcal{L}_{k+1}| \le k+1$ at all time steps. Given $O(\log n)$ time to insert or query from a priority queue with at most n items, we have that $O(T \log n)$ upper bounds the cost of all insertions and deletions of points to $\mathcal{Q}$. Moreover, all calls to compute $\mathcal{L}_{k+1} \cap N(v)$ for some vertex v either occur when v is the vertex added to the stream on that time step (of which there is at most one), or when v is inserted after previously having been in $\mathcal{Q}$. By paying each vertex v a sum of $O(k)$ when it is added to $\mathcal{Q}$, and paying $O(k)$ to each vertex when it is first added to the stream, it can afford the cost of later computing $\mathcal{L}_{k+1} \cap N(v)$ when it is removed. This results in a total cost of $O(T(k + \log n) + Mk)$, which completes the proof. □

In what follows, we focus on bounding the quantity T. To accomplish this, observe that a vertex v can be added to $\mathcal{Q}$ on a given time step t in one of three ways:

Scenarios where v is added to $\mathcal{Q}$

1. The vertex v was added in the stream on time step t. In this case, v is added to $\mathcal{Q}$ when the if statement on Line 1 of Algorithm 2 executes.

2. The vertex v is added to $\mathcal{Q}$ when it was previously led by $\ell_{t-1}(v) \in V$, and either $\ell_{t-1}(v)$ becomes a follower of another leader during time step t (resulting in v being added to $\mathcal{Q}$), or $\ell_{t-1}(v)$ is deleted. This can occur in either Lines 14 or 18 of Algorithm 2 for the first case, or in Lines 5 or 11 of Algorithm 3 in the case of $\ell_{t-1}(v)$ being deleted.

3. The vertex v was previously in LFMIS_{k+1}, and subsequently left LFMIS_{k+1} because $|\text{LFMIS}_{k+1}| = k+1$ and a new vertex u was added to LFMIS_{k+1} with smaller rank. This occurs in Line 9 of Algorithm 2.

Obviously, the first case can occur at most once per stream update, so we will focus on bounding the latter two types of additions to $\mathcal{Q}$. For any step t, define $\mathcal{A}_\pi^t$ to be the number of vertices that are added to $\mathcal{Q}$ as a result of the second form of insertions above. Namely, $\mathcal{A}_\pi^t = |\{v \in G^t : \ell_{t-1}(v) \in V, \text{ and } \ell_t(v) \neq \ell_{t-1}(v)\}|$. Next, define $\mathcal{B}_\pi^t$ to be the number of leaders which were removed from $\mathcal{L}_{k+1}$ Line 9 of Algorithm 2 (i.e., insertions into $\mathcal{Q}$ of the third kind above). Letting T be as in Proposition 17, we have $T \leq M + \sum_t \mathcal{A}_\pi^t + \mathcal{B}_\pi^t$.

To handle $\sum_t \mathcal{A}_\pi^t$ and $\sum_t \mathcal{B}_\pi^t$, we demonstrate that each quantity can be bounded by the total number of times that the *eliminator* of a vertex changes. Recall from Section 3 that, given a graph $G = (V, E)$, $v \in V$, and ranking $\pi : V \to [0, 1]$, the eliminator of v, denoted $\text{elim}_{G,\pi}(v)$, is defined as the vertex of smallest rank in the set $(N(v) \cup \{v\}) \cap \text{LFMIS}(G, \pi)$. Now define $\mathcal{C}_\pi^t$ to be the number of vertices whose eliminator changes after time step t. Formally, for any two graphs G, G' differing in at most once vertex, we define $\mathcal{C}_\pi(G, G') = \{v \in V \,|\, \text{elim}_{G,\pi}(v) \neq \text{elim}_{G',\pi}(v)\}$, and set $\mathcal{C}_\pi^t = |\mathcal{C}_\pi(G^{t-1}, G^t)|$. We now demonstrate that $\sum_t \mathcal{C}_\pi^t$ deterministically upper bounds both $\sum_t \mathcal{A}_\pi^t$ and $\sum_t \mathcal{B}_\pi^t$.

Lemma 18. *Fix any ranking $\pi : V \to [0, 1]$. Then we have $\sum_t \mathcal{A}_\pi^t \leq 5 \sum_t \mathcal{C}_\pi^t$, and moreover for any time step t we have $\mathcal{B}_\pi^t \leq \mathcal{C}_\pi^t$.*

Proof. We first prove the second statement. Fix any time step t, and let $v_1, \dots, v_r$ be the $r = |\mathcal{B}_\pi^t|$ vertices which were removed from $\mathcal{L}_{k+1}$, ordered by the order in which they were removed from $\mathcal{L}_{k+1}$. For this to occur, we must have inserted at least r vertices $u_1, \dots, u_r$ into $\mathcal{L}_{k+1}$ which were not previously in $\mathcal{L}_{k+1}$ on the prior step; in fact, Line 9 of Algorithm 2 induces a unique mapping from each v_i to the vertex u_i which forced it out of $\mathcal{L}_{k+1}$ during a call to `Insert`$(u_i, \pi(u_i))$. Note that, under this association, we have $\pi(u_i) < \pi(v_i)$ for each i. We claim that the eliminator of each such u_i changed on time step t.

Now note that $\{u_1, \dots, u_r\}$ and $\{v_1, \dots, v_r\}$ are disjoint, since u_i was inserted before u_{i+1} during time step t by the definition of the ordering, and so $\pi(u_1) < \pi(u_2) < \cdots < \pi(u_r)$ by Proposition 15, so no u_i could be later kicked out of $\mathcal{L}_{k+1}$ by some u_j with $j > i$. It follows that none of $u_1, \dots, u_r$ were contained in $\text{LFMIS}_{k+1}(G^{t-1}, \pi)$, but they are all in $\text{LFMIS}_{k+1}(G^t, \pi)$. Now note that it could not have been the case that $u_i \in \text{LFMIS}(G^{t-1}, \pi)$, since we had $v_i \in \text{LFMIS}_{k+1}(G^{t-1}, \pi)$ but $\pi(u_i) < \pi(v_i)$. Thus $u_i \notin \text{LFMIS}(G^{t-1}, \pi)$, and therefore the eliminator of u_i changed on step t from $\text{elim}_{G^{t-1},\pi}(u_i) \neq u_i$ to $\text{elim}_{G^t,\pi}(u_i) = u_i$, which completes the proof of the second statement.

We now prove the first claim that $\sum_t \mathcal{A}_\pi^t \leq 5 \sum_t \mathcal{C}_\pi^t$. Because a vertex can be inserted into $\mathcal{Q}$ at most once per time step (due to Proposition 16), each insertion $\mathcal{Q}$ which contributes to $\sum_t \mathcal{A}_\pi^t$ can be described as a vertex-time step pair (v, t), where we have $\ell_t(v) \neq \ell_{t-1}(v) \in V$ because either $\ell_{t-1}(v)$ became a follower of a vertex in $\text{LFMIS}_{k+1}(G^t, \pi)$, or because $\ell_{t-1}(v)$ was deleted on time step t. We will now need two technical claims.

Claim 19. *Consider any vertex-time step pair (v, t) where $\ell_t(v) \in V$ and $\ell_{t-1}(v) \neq \ell_t(v)$. In other words, v was made a follower of some vertex $\ell_t(v)$ during time step t. Then $\ell_t(v) = \text{elim}_{G^t,\pi}(v)$.*

Proof. First note that the two statements of the claim are equivalent, since if $\ell(v)$ is set to $u \in V$ during time step t, then by Proposition 15 we have that $\ell(v)$ is not modified again during the processing of update t, so $u = \ell_t(v)$. Now the algorithm would only set $\ell_t(v) = u$ in one of two cases. In this first case, it occurs during a call to `Insert`$(v, \pi(v))$, in which case $\ell_t(v)$ is set to the vertex with smallest rank in $\text{LFMIS}_{k+1}(G^t, \pi) \cap N(v)$, which by definition is $\text{elim}_{G^t,\pi}(v)$. In the second case, v was previously in $\text{LFMIS}_{k+1}(G^{t-1}, \pi)$, and $\ell(v)$ was changed to a vertex w during a call to `Insert`$(w, \pi(w))$, where $w \in N(v)$ and $\pi(w) < \pi(v)$. Since prior to this insertion v was not a neighbor of any point in $\text{LFMIS}_{k+1}(G^{t-1}, \pi)$, and since by Proposition 15 the `Insert` function will not be called again on time t for a vertex with rank smaller than w, it follows that w has the minimum rank of all neighbors of v in $\text{LFMIS}(G^t, \pi)$, which completes the claim. ☐

Claim 20. *Consider any vertex-time step pair (v,t) where $\ell_t(v) = \perp$ and $\ell_{t-1}(v) = \text{elim}_{G^{t-1},\pi}(v)$. Then $\text{elim}_{G^{t-1},\pi}(v) \neq \text{elim}_{G^t,\pi}(v)$.*

Proof. If $\ell_{t-1}(v) = \text{elim}_{G^{t-1},\pi}(v)$, then $\text{elim}_{G^{t-1},\pi}(v) \in v$ and $\ell_t(v)$ is changed to $\perp$ during time step t, then as in the prior claim, this can only occur if $\ell_{t-1}(v)$ is made a follower of another point in $\text{LFMIS}_{k+1}(G^t,\pi)$, or if $\ell_{t-1}(v)$ is deleted on that time step. In both cases we have $\ell_{t-1}(v) \notin \text{LFMIS}(G^t,\pi)$. Since $\text{elim}_{G^t,\pi}(v)$ is always in $\text{LFMIS}(G^t,\pi)$, the claim follows. □

Now fix any vertex v, and let $\sigma_1,\dots,\sigma_M$ be the sequence of eliminators of v, namely $\sigma_t = \text{elim}_{G^t,\pi}(v)$ (note that σ_t is either a vertex in V or $\sigma_t = \emptyset$). Similarly define $\lambda_1,\dots,\lambda_M$ by $\lambda_t = \ell_t(v)$, and note that $\lambda_t \in V \cup \{\perp\}$. To summarize the prior two claims: each time $\lambda_{t-1} \neq \lambda_t$ and $\lambda_t \in V$, we have $\sigma_t = \lambda_t$; namely, the sequences become aligned at time step t. Moreover, whenever the two sequences are aligned at some time step t, namely $\sigma_t = \lambda_t$, and subsequently $\lambda_{t+1} = \perp$, we have that $\sigma_{t+1} \neq \sigma_t$. We now prove that every five subsequent changes in the value of λ cause at least one unique change in σ.

To see this, let $t_1 < t_2 < t_3$ be three subsequent changes, so that $\lambda_{t_1} \neq \lambda_{t_1-1}$, $\lambda_{t_2} \neq \lambda_{t_2-1}$, $\lambda_{t_3} \neq \lambda_{t_3-1}$, and λ_i does not change for all $i = t_1,\dots,t_2-1$ and $i = t_2,\dots,t_3-1$. First, if $\lambda_{t_1},\lambda_{t_2} \in V$, by Claim 19 we have $\sigma_{t_1} = \lambda_{t_1}$ and $\sigma_{t_2} = \lambda_{t_2}$, and thus $\sigma_{t_1} \neq \sigma_{t_2}$, so σ changes in the interval $[t_1,t_2]$. If $\lambda_{t_1} \in V$ and $\lambda_{t_2} = \perp$, we have $\sigma_{t_1} = \lambda_{t_1}$, and so if σ does not change by time t_2-1 we have $\sigma_{t_2-1} = \lambda_{t_2-1}$, and thus $\sigma_{t_2} \neq \sigma_{t_2-1}$ by Claim 20, so σ changes in the interval $[t_1,t_2]$. Finally, if $\lambda_{t_1} = \perp$, then we must have $\lambda_{t_2} \in V$, and so $\lambda_{t_3} = \perp$. Then by the prior argument, σ must change in the interval $[t_2,t_3]$. Thus, in each case, σ must change in the interval $[t_1,t_3]$. To avoid double counting changes which occur on the boundary, letting $t_1,\dots,t_r$ be the sequence of all changes in λ, it follows that there is at least one change in σ in each of the disjoint intervals $(t_{5i+1},t_{5(i+1)})$ for $i = 0,1,2,\dots,\lfloor r/5 \rfloor$. It follows that $\sum_t \mathcal{A}_\pi^t \leq 5\sum_t \mathcal{C}_\pi^t$, which completes the proof. □

The following theorem, due to [BDH+19], bounds the expected number of changes of eliminators which occur when a vertex is entirely removed or added to a graph.

Theorem 21 (Theorem 3 of [BDH+19])**.** *Let $G = (V,E)$ be any graph on n vertices, and let $G' = (V',E')$ be obtained by removing a single vertex from V along with all incident edges. Let $\pi : V \to [0,1]$ be a random mapping. Let $\mathcal{C}_\pi(G,G') = \{v \in V \,|\, \text{elim}_{G,\pi}(v) \neq \text{elim}_{G',\pi}(v)\}$. Then we have $\mathbb{E}_\pi\left[|\mathcal{C}_\pi(G,G')|\right] = O(\log n)$.*

Theorem 22. *There is a algorithm which, on a fully dynamic stream of insertions and deletions of vertices to a graph G, maintains at all time steps a top-k LFMIS of G with leaders (Definition 9) under a random ranking $\pi : V \to [0,1]$. The expected amortized per-update time of the algorithm is $O(k\log n + \log^2 n)$, where n is the maximum number active of vertices at any time. Moreover, the algorithm does not need to know n in advance.*

Proof. By the above discussion, letting T be as in Proposition 17, we have $T \leq M + \sum_t \mathcal{A}_\pi^t + \mathcal{B}_\pi^t$. By the same proposition, the total update time of the algorithm over a sequence of n updates is at most $O(T(k+\log n) + kM)$. By Lemma 18, we have $\sum_t \mathcal{A}_\pi^t + \mathcal{B}_\pi^t \leq 6\sum_t \mathcal{C}_\pi^t$, and by Theorem 21 we have $\mathbb{E}_\pi\left[\sum_t \mathcal{C}_\pi^t\right] = O(M\log n)$. It follows that $\mathbf{E}[T] = O(M\log n)$, therefore the total update time is $O(kM\log n + M\log^2 n)$, which completes the proof. □

5 Fully Dynamic k-Centers via Locally Sensitive Hashing

In this section, we demonstrate how the algorithm for general metric spaces of Section 4 can be improved to run in *sublinear* in n amortized update time, even when $k = \Theta(n)$, if the metric in question admits good locally sensitive hash functions (introduced below in Section 5.1). Roughly speaking, a locally sensitive hash function is a mapping $h : \mathcal{X} \to U$, for an universe U, which has the property that points which are close in the metric should collide, and points which are far should not. Thus, when searching for points which are close to a given $x \in \mathcal{X}$, one can first apply a LSH to quickly prune far points, and search only through the points in the hash bucket $h(x)$. We will use this approach to speed up the algorithm from Section 4.

There are several serious challenges when using an arbitrary ANN data structure to answer nearest-neighbor queries for our k-center algorithm. Firstly, the algorithm *adaptively* queries the ANN data structure: the points which are inserted into LFMIS_{k+1}, as well as the future edges which are reported by the data structure, depend on the prior edges which were returned by the data structure. Such adaptive reuse breaks down traditional guarantees of randomized algorithms, hence designing such algorithms which are robust to adaptivity is the subject of a growing body of research [BEJWY20, CN20, HKM+20, WZ22, ACSS21]. More nefariously, the adaptivity also goes in the other direction: namely, the random ordering π influences which points will be added to the set LFMIS_{k+1}, in turn influencing the future queries made to the ANN data structure, which in turn dictate the edges which exist in the graph (by means of queries to the ANN oracle). Thus, the graph itself cannot be assumed to be independent of π! However, we show that we can leverage LSH functions to define the graph independent of π.

Summary of the LSH-Based Algorithm. We now describe the approach of our algorithm for LSH spaces. Specifically, first note that the factor of k in the amortized update time in Proposition 17 comes from the time required to compute $S = \mathcal{L}_{k+1} \cap N(v)$ in Algorithm 2. However, to determine which of the three cases we are in for the execution of Algorithm 2, we only need to be given the value $u^* = \arg\min_{u' \in S} \pi(u')$ of the vertex in S with smallest rank, or $S = \emptyset$ if none exists. If $S = \emptyset$, then the remainder of Algorithm 2 runs in constant time. If $\pi(u^*) < \pi(v)$, where v is the query point, then the remaining run-time is constant unless v was a leader, in which case it is proportional to the number of followers of v, each of which are inserted into $\mathcal{Q}$ at that time. Lastly, if $\pi(u^*) > \pi(v)$, then we search through each $u \in S$, make u a follower of v, and add the followers of u to $\mathcal{Q}$. If u was previously in $\mathcal{L}_{k+1}$, it must have also been in LFMIS_{k+1} on the prior time step. Thus it follows that each such $u \in S$ changes its eliminator on this step.

In summary, after the computation of $S = \mathcal{L}_{k+1} \cap N(v)$, the remaining runtime is bounded by the sum of the number of points added to $\mathcal{Q}$, and the number of points that change their eliminator on that step. Since, ultimately, the approach in Section 4.3 was to bound T by the total number of times a point's eliminator changes, our goal will be to obtain a more efficient data structure for returning $S = \mathcal{L}_{k+1} \cap N(v)$. Specifically, if after a small upfront runtime R, such a data structure can read off the entries of S in the order of the rank, each in constant time, one could therefore replace the factor of k at both parts of the sum in the running time bound of Proposition 17 by R. We begin by formalizing the guarantee that such a data structure should have.

We will demonstrate that approximate nearest neighbor search algorithms based on locally sensitive hashing can be modified to have the above properties. However, since such an algorithm will only be approximate, it will sometimes returns points in S which are farther than distance r from v, where r is the threshold. Thus, the resulting graph defined by the locally sensitive hashing procedure will now be an *approximate threshold graph:*

Definition 23. *Fix a point set P from a metric space $(\mathcal{X}, d)$, and real values $r > 0$ and $c \geq 1$. A (r, c, L)-approximate threshold graph $G_{r,c} = (V(G_{r,c}), E(G_{r,c}))$ for P is any graph with $V(G_{r,c}) = P$, and whose whose edges satisfy $E(G_r) \subseteq E(G_{r,c})$ and $|E(G_{r,c}) \setminus E(G_{cr})| \leq L$, where $E(G_r), E(G_{cr})$ are the edge set of the threshold graphs G_r, G_{cr} respectively.*

If $L = 0$, it is straightforward to see that an algorithm for solving the top-k LFMIS problem on a $(r, c, 0)$-approximate threshold graph $G_{r,c}$ can be used to obtain a $c(2+\epsilon)$ approximation to k-center. When $L > 0$, an algorithm can first check, for each edge $e \in E(G_{r,c})$ it considers, whether $e \in G_{rc}$, and discard it if it is not the case. We will see that the runtime of handling a (r, c, L)-approximate threshold graph will depend linearly on L. Moreover, we will set parameters so that L is a constant in expectation.

5.1 Locally Sensitive Hashing and the LSH Algorithm

We begin by introducing the standard definition of a locally sensitive hash family for a metric space [IM98].

Definition 24 (Locally sensitive hashing [IM98, HPIM12])**.** *Let $\mathcal{X}$ be a metric space, let U be a range space, and let $r \geq 0$ $c \geq 1$ and $0 \leq p_2 \leq p_1 \leq 1$ be reals. A family $\mathcal{H} = \{h : \mathcal{X} \to U\}$ is called (r, cr, p_1, p_2)-sensitive if for any $q, p \in \mathcal{X}$:*

- *If $d(p, q) \leq r$, then $\mathbf{Pr}_{\mathcal{H}}[h(q) = h(p)] \geq p_1$.*
- *If $d(p, q) > cr$, then $\mathbf{Pr}_{\mathcal{H}}[h(q) = h(p)] \leq p_2$.*

Given a (r, cr, p_1, p_2)-sensitive family $\mathcal{H}$, we can define $\mathcal{H}^t$ to be the set of all functions $h^t : \mathcal{X} \to U^t$ defined by $h^t(x) = (h_1(x), h_2(x), \ldots, h_t(x))$, where $h_1, \ldots, h_k \in \mathcal{H}$. In other words, a random function from $\mathcal{H}^t$ is obtained by drawing t independent hash functions from $\mathcal{H}$ and concatenating the results. It is easy to see that the resulting hash family $\mathcal{H}^t$ is (r, cr, p_1^t, p_2^t)-sensitive. We now demonstrate how a (r, c)-approximate threshold graph can be defined via a locally sensitive hash function.

Definition 25. *Fix a metric space $(\mathcal{X}, d)$ and a finite point set $P \subset \mathcal{X}$, as well as integers $t, s \geq 1$. Let $\mathcal{H} : \mathcal{X} \to U$ be a (r, cr, p_1, p_2)-sensitive family. Then a graph $G_{r,cr}(P, \mathcal{H}, t, s) = (V, E)$ induced by $\mathcal{H}$ is a random graph which is generated via the following procedure. First, one randomly selects hash functions $h_1, h_2, \ldots, h_s \sim \mathcal{H}^t$. Then the vertex set is given by $V = P$, and then edges are defined via $(x, y) \in E$ if and only if $h_i(x) = h_i(y)$ for some $i \in [s]$.*

We now demonstrate that, if $\mathcal{H}$ is a sufficiently sensitive hash family, the random graph $G_{r,cr}(P, \mathcal{H}, t, s)$ constitutes a (r, c, L)-approximate threshold graph with good probability, where L is a constant in expectation.

Proposition 26. *Fix a metric space $(\mathcal{X}, d)$ and a point set $P \subset \mathcal{X}$ of size $|P| = n$, and let $\mathcal{H} : \mathcal{X} \to U$ be a (r, cr, p_1, p_2)-sensitive family. Fix any $\delta \in (0, \frac{1}{2})$. Set $s = \ln(n^2/\delta)n^{2\rho}/p_1$, where $\rho = \ln \frac{1}{p_1} / \ln \frac{1}{p_2}$, and $t = \lceil 2 \log_{1/p_2} n \rceil$. Then, with probability at least $1 - \delta$, the the random graph $G_{r,cr}(P, \mathcal{H}, t, s)$ is a (r, c, L)-approximate threshold graph, where L is a random variable satisfying $\mathbf{E}[L] < 2$ (Definition 23).*

Proof. First, fix any $x, y \in P$ such that $d(x, y) > cr$. We have that $\mathbf{Pr}_{h \sim \mathcal{H}^k}[h(x) = h(y)] \leq p_2^t < \frac{1}{n^2}$. It follows that

$$\mathop{\mathbf{E}}_{h_1, \ldots, h_s \sim \mathcal{H}^t}[|E(G_{r,cr}(P, \mathcal{H}, t, s) \setminus E(G_r)|] \leq \sum_{(x,y) \in P^2} \frac{1}{n^2} < 1$$

Namely, we have $\mathbf{E}[L] < 1$ where $L = |E(G_{r,cr}(P, \mathcal{H}, t, s) \setminus E(G_r)|$. Next, fix any $(x, y) \in E(G_r)$. We have

$$\mathop{\mathbf{Pr}}_{h \sim \mathcal{H}^k}[h(x) = h(y)] \geq p_1^t > p_1^{2 \log_{1/p_2} n + 1} = p_1 (n^2)^{-\rho}$$

Thus, the probability that at least one h_i satisfies $h_i(x) = h_i(y)$ is at least

$$1 - (1 - p_1 n^{-2\rho})^s > 1 - (1/e)^{\ln(n^2/\delta)} = 1 - \delta/n^2$$

After a union bound over all such possible pairs, it follows that $(x, y) \in E(G_{r,cr}(P, \mathcal{H}, t, s))$ for all $(x, y) \in E(G_r)$ with probability at least $1 - \delta$. Note that since $\delta < 1/2$ and L is a non-negative random variable, it follows that conditioning on the prior event can increase the expectation of L by at most a factor of 2, which completes the proof. □

We will now describe a data structure which allows us to maintain a subset $\mathcal{L}$ of vertices of the point set P, and quickly answer queries for neighboring edges of a vertex v in the graph G defined by the intersection of $G_{r,cr}(P, \mathcal{H}, t, s)$ and G_{cr}. Note that if $G_{r,cr}(P, \mathcal{H}, t, s)$ is a (r, c, L)-approximate threshold graph, then this intersection graph G satisfies $G_r \subseteq G \subseteq G_{cr}$, and is therefore a $(r, c, 0)$-approximate threshold graph. It is precisely this graph G which we will run our algorithm for top-k LFMIS on. However, in addition to finding all neighbors of v in $\mathcal{L}$, we will also need to quickly return the neighbor with smallest rank π, where $\pi : V \to [0, 1]$ is a random ranking as in Section 4. Roughly, the data structure will hash all points in $\mathcal{L}$ into the hash buckets given by $h_1, \dots, h_s$, and maintain each hash bucket via a binary search tree of depth $O(\log n)$, where the ordering is based on the ranking π.

For the following Lemma and Theorem, we fix a metric space $(\mathcal{X}, d)$ and a point set $P \subset \mathcal{X}$ of size $|P| = n$, as well as a scale $r > 0$, and approximation factor c. Moreover, let $\mathcal{H} : \mathcal{X} \to U$ be a (r, cr, p_1, p_2)-sensitive hash family for the metric space $\mathcal{X}$, and let $\text{Time}(\mathcal{H})$ be the time required to evaluate a hash function $h \in \mathcal{H}$. Furthermore, let $\pi : P \to [0, 1]$ be any ranking over the points P, such that $\pi(x)$ is truncated to $O(\log n)$ bits, and such that $\pi(x) \neq \pi(y)$ for any distinct $x, y \in P$ (after truncation). Note that the latter holds with probability $1 - 1/\operatorname{poly}(n)$ if π is chosen uniformly at random. Lastly, for a graph $G = (V(G), E(G))$, let $N_G(v)$ be the neighborhood of v in G (in order to avoid confusion when multiple graphs are present).

Lemma 27. *Let $G_{r,cr}(P, \mathcal{H}, t, s)$ be a draw of the random graph as in Definition 25, where $r, c, P, \mathcal{H}$ are as above, such that $G_{r,cr}(P, \mathcal{H}, t, s)$ is a (r, c, L)-approximate threshold graph. Let $G = G_{r,cr}(P, \mathcal{H}, t, s) \cap G_{cr}$. Then there is a fully dynamic data structure, which maintains a subset $\mathcal{L} \subset V(G)$ and can perform the following operations:*

- `Insert`(v): *inserts a vertex v into $\mathcal{L}$ in time $O(s \log n + ts\,\text{Time}(\mathcal{H}))$*
- `Delete`(v): *deletes a vertex v from $\mathcal{L}$ in time $O(s \log n + ts\,\text{Time}(\mathcal{H}))$*
- `Query-Top`(v): *returns $u^* = \arg\min_{u \in N_G \cap \mathcal{L}} \pi(u)$, or* Empty *if $N_G(v) \cap \mathcal{L} = \emptyset$, in time $O(sL \log n + ts\,\text{Time}(\mathcal{H}))$*
- `Query-All`(v): *returns the set $N_G(v) \cap \mathcal{L}$, running in time $O(s(|N_G(v) \cap \mathcal{L}| + L) \log n + ts\,\text{Time}(\mathcal{H}))$.*

Moreover, given the hash functions $h_1, \dots, h_s \in \mathcal{H}^t$ which define the graph $G_{r,cr}(P, \mathcal{H}, t, s)$, the algorithm is deterministic, and therefore correct even against an adaptive adversary.

Proof. For each $i \in [s]$ and hash bucket $b \in U$, we store a binary tree $T_{i,b}$ with depth at most $O(\log n)$, such that each node z corresponds to an interval $[a, b] \subset [0, 1]$, and the left and right children of z correspond to the intervals $[a, (a + b)/2]$ and $[(a + b/2), b]$, respectively. Moreover, each node z maintains a counter for the number of points in $\mathcal{L}$ which are stored in its subtree. Given a vertex v with rank $\pi(v)$, one can then insert v into the unique leaf of $T_{i,b}$ corresponding to the $O(\log n)$ bit value $\pi(v)$ in time $O(\log n)$. Note that, since the keys $\pi(v)$ are unique, each leaf contains at most one vertex. Similarly, one can remove and search for a vertex from $T_{i,b}$ in time $O(\log n)$.

When processing any query for an input vertex $v \in P$, one first evaluates all ts hash functions required to compute $h_1(v), \dots, h_s(v)$, which requires $ts\,\text{Time}(\mathcal{H})$ time. For insertions and deletions, one can insert v from each of the s resulting trees in time $O(\log n)$ per tree, which yields the bounds for `Insert`(v) and `Delete`(v). For `Query-Top`(v), for each $i \in [s]$, one performs an in-order traversal of $T_{i,h_i(v)}$, ignoring nodes without any points stored in their subtree, and returns the first leaf corresponding to a vertex $u \in N_G(v)$, or Empty if all points in the tree are examined before finding such a neighbor. Each subsequent non-empty leaf in the traversal can be obtained in $O(\log n)$ time, and since by definition of a (r, c, L)-approximate threshold graph, $h_i(v) = h_i(u')$ for at most u' vertices with $(v, u') \notin G_{cr}$, it follows that one must examine at most L vertices u' with $(v, u') \notin G_{cr}$ before one finds $u^* = \arg\min_{u \in N_G \cap \mathcal{L}} \pi(u)$ (or exhausts all points in the tree). Thus, the runtime is $O(L \log n)$ to search through each of the s hash functions, which results in the desired bounds.

Finally, for `Query-All`(v), one performs the same search as above, but instead completes the full in-order traversal of each tree $T_{i,h_i(v)}$. By the (r, c, L)-approximate threshold graph property, each tree $T_{i,h_i(v)}$ contains at most $|N_G(v) \cap \mathcal{L}| + L$ vertices from $\mathcal{L}$, after which the runtime follows by the argument in the prior paragraph. □

Given the data structure from Lemma 27, we will now demonstrate how the algorithm from Section 4 can be implemented in sublinear in k time, given a sufficently good LSH function for the metric. The following theorem summarizes the main consequences of this implementation, assuming the graph $G_{r,cr}(P, \mathcal{H}, t, s)$ is a (r, c, L)-approximate threshold graph.

Theorem 28. *Let $G_{r,cr}(P, \mathcal{H}, t, s)$ be a draw of the random graph as in Definition 25, where $r, c, P, \mathcal{H}$ are as above, such that $G_{r,cr}(P, \mathcal{H}, t, s)$ is a (r, c, L)-approximate threshold graph. Let $G = (V, E)$ be the graph with $V = P$ and $E = E(G_{r,cr}(P, \mathcal{H}, t, s)) \cap E(G_{cr})$. Then there is a fully dynamic data structure which, under a sequence of vertex insertions and deletions from G, maintains a top-n LFMIS with leaders (Definition 9) of G at all time steps. The expected amortized per-update runtime of the algorithm is $O((sL \log n + ts \,\mathrm{Time}(\mathcal{H})) \log n + \log^2 n)$, where the expectation is taken over the choice of π.*

Proof. The algorithm is straightforward: we run the fully dynamic algorithm for top-k LFMIS with leaders from Section 4, however we utilize the data structure from Lemma 27 to compute $S = \mathcal{L} \cap N_G(v)$ in Algorithm 2 (where $\mathcal{L} = \mathcal{L}_{n+1}$), as well as handle deletions from $\mathcal{L}$ in Algorithm 3. Note that to handle a call to `Insert`(v) of Algorithm 2, one first calls `Query-Top`(v) in data structure from Lemma 27. If the result is $\emptyset$, or u with $\pi(u) < \pi(v)$, then one can proceed as in Algorithm 2 but by updating $\mathcal{L}$ via Lemma 27. If the result is u with $\pi(u) > \pi(v)$, one then calls `Query-All`(v) to obtain the entire set $S = N_G(v) \cap \mathcal{L}$, each of which will subsequently be made a follower of v.

Let T be the total number of times that a vertex is inserted into the queue $\mathcal{Q}$ over the entire execution of the algorithm (as in Proposition 17), and as in Section 4.3, we let $\mathcal{C}_\pi^t$ denote the number of vertices whose eliminator changed after the t-th time step. We first prove the following claim, which is analogous to Proposition 17.

Claim 29. *The total runtime of the algorithm, over a sequence of M insertions and deletions of vertices from G, is at most $O(T(\lambda + \log n) + \lambda(M + \sum_{t \in [M]} \mathcal{C}_\pi^t))$, where $\lambda = sL \log n + ts \,\mathrm{Time}(\mathcal{H})$.*

Proof. First note that λ upper bounds the cost of inserting and deleting from $\mathcal{L}$, as well as calling `Query-Top`(v). For every vertex, when it is first inserted into the stream, we pay it a cost of λ to cover the call to `Query-Top`(v). Moreover, whenever a vertex is added to the queue $\mathcal{Q}$, we pay a cost of λ to cover a subsequent call to `Query-Top`(v) when it is removed from the queue and inserted again, plus an additional $O(\log n)$ required to insert and remove the top of a priority queue. The only cost of the algorithm which the above does not cover is the cost of calling `Query-All`(v), which can be bounded by $O(\lambda \cdot |N_G(v) \cap \mathcal{L}|)$. Note that, by correctness of the top-k LFMIS algorithm

(Lemma 14), each vertex in $\mathcal{L}$ is its own eliminator at the beginning of each time step. It follows that each vertex $u \in N_G(v) \cap \mathcal{L}$ had its eliminator changed on step t, since `Query-All`(v) is only called on time step t in the third case of Algorithm 2, where all points in $|N_G(v) \cap \mathcal{L}|$ will be made followers of v. Thus the total cost of all calls to `Query-All`(v) can be bounded by $\lambda \sum_{t \in [M]} \mathcal{C}_\pi^t$, which completes the proof of the claim. $\square$

Given the above, by Lemma 18 we have that $T \le M \le 6\mathcal{C}_\pi^t$, and by Theorem 21 we have $\mathbf{E}_\pi\left[\sum_t \mathcal{C}_\pi^t\right] = O(M \log n)$. It follows that, the expected total runtime of the algorithm, taken over the randomness used to generate π (with $h_1, \dots, h_s$ previously fixed and conditioned on) is at most $O(M \log^2 n + M\lambda \log n)$ as needed. $\square$

Theorem 30. *Let $(\mathcal{X}, d)$ be a metric space, and fix $\delta \in (0, 1/2)$. Suppose that for any $r \in (r_{\min}, r_{\max})$ there exists an (r, cr, p_1, p_2)-sensitive hash family $\mathcal{H}_r : \mathcal{X} \to U$, such that each $h \in \mathcal{H}_r$ can be evaluated in time at most $\text{Time}(\mathcal{H})$, and such that p_2 is bounded away from 1. Then there is a fully dynamic algorithm that, on a sequence of M insertions and deletions of points from $\mathcal{X}$, given an upper bound $M \le \hat{M} \le \text{poly}(M)$, with probability $1 - \delta$, correctly maintains a $c(2 + \epsilon)$-approximate k-center clustering to the active point set P^t at all time steps t, simultaneously for all $k \ge 1$. The total runtime of the algorithm is at most*

$$\tilde{O}\left(M \cdot \frac{\log \Delta \log \delta^{-1}}{\epsilon p_1} n^{2\rho} \cdot \text{Time}(\mathcal{H})\right)$$

where $\rho = \frac{\ln p_1}{\ln p_2}$, and n is an upper bound on the maximum number of points at any time step.

Proof. We first demonstrate that there exists an algorithm $\mathcal{A}(\delta, M)$ which takes as input $\delta \in (0, 1/2)$ and $M \ge 1$, and on a sequence of at most M insertions and deletions of points in $\mathcal{X}$, with probability $1 - \delta$ correctly solves the top-M LFMIS with leaders (Definition 9) on a graph G that is $(r, c, 0)$-approximate threshold graph for P (Definition 23), where $P \subset \mathcal{X}$ is the set of all points which were inserted during the sequence, and runs in total expected time $\alpha_{\delta,M}$, where

$$\alpha_{\delta,M} = \tilde{O}\left(M^{1+2\rho} \log(M/\delta) \,\text{Time}(\mathcal{H})\right)$$

The reduction from having such an algorithm to obtaining a k-center solution for every $k \ge 1$, incurring a blow-up of $\epsilon^{-1} \log \Delta$, and requiring one to scale down δ by a factor of $O(\epsilon^{-1} \log \Delta)$ so that all $\epsilon^{-1} \log \Delta$ instances are correct, is the same as in Section 3, with the modification that the clustering obtained by a MIS $\mathcal{L}$ at scale $r \ge 0$ has cost at most cr, rather than r. Thus, in what follows, we focus on a fixed r.

First, setting $s_{\delta,M} = O(\log(M/\delta) M^{2\rho}/p_1)$ and $t_M = O(\log_{1/p_2} M)$, by Proposition 26 it holds that with probability $1 - \delta$ the graph $G_{r,cr}(P, \mathcal{H}, t_M, s_{\delta,M})$ is a (r, c, L)-approximate threshold graph, with $\mathbf{E}[L] < 2$. Then by Theorem 28, there is an algorithm which maintains a top-M LFMIS with leaders to the graph $G = G_{r,cr}(P, \mathcal{H}, t_M, s_{\delta,M}) \cap G_{rc}$, which in particular is a $(r, c, 0)$-approximate threshold graph for P, and runs in expected time at most

$$O(M(s_{\delta,M} L \log M + t_M s_{\delta,M} \,\text{Time}(\mathcal{H})) \log M + M \log^2 M)$$

where the expectation is taken over the choice of the random ranking π. Taking expectation over L, which depends only on the hash functions $h_1, \dots, h_{s_{\delta,M}}$, and is therefore independent of π, the expected total runtime is at most $\alpha_{\delta,M}$ as needed.

To go from the updated time holding in expectation to holding with probability $1 - \delta$, we follow the same proof of Proposition 10, except that we set the failure probability of each instance to be $O(\delta / \log^2(M/\delta))$. By the proof of Proposition 10, the total number of copies ever run by the algorithm is at most $O(\log(1/\delta) \sum_{i=1}^{\log M} i) = O(\log(1/\delta) \log^2 M)$ with probability at most $1 - \delta/2$, and thus with probability at least $1-\delta$ it holds that both at most $O(\log(1/\delta)\log^2 M)$ copies of the algorithm are run, and each of them is correct at all times. Note that whenever the runtime of the algorithm exceeds this bound, the algorithm can safely terminate, as the probability that this occurs is at most δ by Proposition 10.

Put together, the above demonstrates the existence of an algorithm $\bar{\mathcal{A}}(\delta, M)$ which takes as input $\delta \in (0,1/2)$ and $M \geq 1$, and on a sequence of at most M insertions and deletions of points in $\mathcal{X}$, with probability $1-\delta$ correctly maintains a $c(2+\epsilon)$-approximation to the optimal k-center clustering simultaneously for all $k \geq 1$, with total runtime at most $\tilde{O}(\epsilon^{-1}\log\Delta\alpha_{\delta,M})$. However, we would like to only run instances of $\bar{\mathcal{A}}(\delta, M)$ with $M = O(n)$ at any given point in time, so that the amortized update time has a factor of $n^{2\rho}$ instead of $M^{2\rho}$. To accomplish this, we greedily pick time steps $1 \leq t_1 < t_2 < \cdots < M$ with the property that $t_i - t_{i-1} = \lceil |P^{t_{i-1}}|/2 \rceil$. Observe that for such time steps, we have $|P^{t_i}| < 2|P^{t_{i-1}}|$. We then define time steps $1 \leq t'_1 < t'_2 < \cdots < M$ with the property that t'_i is the first time step where the active point set size exceeds 2^i. We then run an instance of $\bar{\mathcal{A}}(\delta_0, 2^i)$, starting with $i = 1$, where $\delta_0 = \delta/M$. Whenever we reach the next time step t'_{i+1} we restart the algorithm $\bar{\mathcal{A}}$ with parameters $(\delta_0, 2^{i+1})$, except that instead of running $\bar{\mathcal{A}}$ on the entire prefix of the stream up to time step t'_{i+1}, we only insert the points in $P^{t'_{i+1}}$ which are active at that time step. Similarly, when we reach a time step t_j, we restart $\bar{\mathcal{A}}$ with the same parameters, and begin by isnerting the active point set P^{t_j}, before continuing with the stream.

Note that $\bar{\mathcal{A}}$ is only restarted $\log n$ times due to the active point set size doubling. Moreover, each time it is restarted due to the time step being equal to t_j for some j, the at most $2|P^{t_{j-1}}|$ point insertions required to restart the algorithm can be amortized over the $t_j - t_{j-1} \geq |P^{t_{j-1}}|/2$ prior time steps. Since each instance is correct with probability $1-\delta_0$, by a union bound all instances that are ever restarted are correct with probability $1-\delta$. Note that, since the amortized runtime dependency of the overall algorithm on M is polylogarithmic, substituting M with an upper bound $\hat{M}$ satisfying $M \leq \hat{M} \leq \text{poly}(M)$ increases the runtime by at most a constant. Moreover, we never run $\bar{\mathcal{A}}(\delta, t)$ with a value of t larger than $2n$, which yields the desired runtime. □

5.2 Corollaries of the LSH Algorithm to Specific Metric Spaces

We now state our results for specific metric spaces by applying Theorem 30 in combination with known locally sensitive hash functions from the literature. The following corollary follows immediately by application of the locally sensitive hash functions of [DIIM04, HPIM12], along with the bounds from Theorem 30.

Corollary 31. *Fix any $c \geq 1$. Then there is a fully dynamic algorithm which, on a sequence of M insertions and deletions of points from d-dimensional Euclidean space $(\mathbb{R}^d, \ell_p)$, for $p \geq 1$ at most a constant, with probability $1-\delta$, correctly maintains a $c(4+\epsilon)$-approximate k-center clustering to the active point set P^t at all time steps $t \in [M]$, and simultaneously for all $k \geq 1$. The total runtime is at most*

$$\tilde{O}\left(M\frac{\log\delta^{-1}\log\Delta}{\epsilon}dn^{1/c}\right)$$

For the case of standard Euclidean space ($p = 2$), one can used the improved ball carving technique of Andoni and Indyk [AI06] to obtain better locally sensitive hash functions, which result in the following:

Corollary 32. *Fix any $c \geq 1$. Then there is a fully dynamic algorithm which, on a sequence of M insertions and deletions of points from d-dimensional Euclidean space $(\mathbb{R}^d, \ell_2)$, with probability*

$1-\delta$, *correctly maintains a* $c(\sqrt{8}+\epsilon)$*-approximate* k*-center clustering to the active point set* P^t *at all time steps* $t \in [M]$*, and simultaneously for all* $k \geq 1$*. The total runtime is at most*

$$\tilde{O}\left(M \frac{\log \delta^{-1} \log \Delta}{\epsilon} d n^{1/c^2+o(1)}\right)$$

Additionally, one can use the well-known MinHash [Bro97] to obtain a fully dynamic k-center algorithm for the *Jaccard Distance*. Here, the metric space is the set of all subsets of a finite universe X, equipped with the distance $d(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}$ for $A, B \subseteq X$. We begin stating a standard bound on the value of ρ for the MinHash LSH family.

Proposition 33 ([IM98])**.** *Let* $\mathcal{H}$ *be the hash family given by*

$$\mathcal{H} = \{h_\pi : 2^X \to X \mid h_\pi(A) = \arg\min_{a \in A} \pi(a),\ \pi \text{ is a permutation of } X\}$$

Then for any $c \geq 1$ *and* $r \in [0, 1/(2c)]$*, we have that* $\mathcal{H}$ *is* $(r, cr, 1-r, 1-cr)$*-sensitive for the Jaccard Metric over* X*, where* $\rho = 1/c$*.*

Proof. If $d(A,B) = r$, for any $r \in [0,1]$, we have

$$\mathop{\mathbf{Pr}}_{h \sim \mathcal{H}}[h(A) = h(B)] = \frac{|A \cap B|}{|A \cup B|} = 1 - r$$

Thus, we have $p_1 = 1 - r$ and $p_2 = 1 - cr$. Now by Claim 3.11 of [HPIM12], we have that for all $x \in [0,1)$ and $c \geq 1$ such that $1 - cx > 0$, the following inequality holds:

$$\frac{\ln(1-x)}{\ln(1-cx)} \leq \frac{1}{c}$$

Thus we have the desired bound:

$$\rho \leq \frac{\ln(1-r)}{\ln(1-cr)} \leq 1/c$$ □

Corollary 34. *Let* X *be a finite set, and fix any* $c \geq 1$*. Then there is a fully dynamic algorithm which, on a sequence of* M *insertions and deletions of subsets of* X *equipped with the Jaccard Metric, with probability* $1-\delta$*, correctly maintains a* $c(4+\epsilon)$*-approximate* k*-center clustering to the active point set* P^t *at all time steps* $t \in [M]$*, and simultaneously for all* $k \geq 1$*. The total runtime is at most*

$$\tilde{O}\left(M \frac{\log \delta^{-1} \log \Delta}{\epsilon} |X| n^{1/c}\right)$$

Proof. Letting $r_{\min}$ be the minimum distance between points in the stream, we run a copy of the top-M LFMIS algorithm of Theorem 28 for $r = r_{\min}, (1+\epsilon)r_{\min}, \dots, 1/(2c)$, where we set the value of the approximation factor c in Theorem 28 to be scaled by a factor of 2, so that each instance computes a LFMIS on a $(r, 2c, 0)$-approximate threshold graph. Note that for any time step t, if the copy of the algorithm for $r = 1/c$ contains a LFMIS with at least $k+1$ vertices, it follows that the cost of the optimal clustering is at most $1/(4c)$. In this case, we can return an arbitrary vertex as a 1-clustering of the entire dataset, which will have cost at most 1, as the Jaccard metric is bounded by 1, and thereby yielding a $4c$ approximation. Otherwise, the solution is at most a $c(4+2\epsilon)$-approximation, which is the desired result after a re-scaling of ϵ. Note that $\text{Time}(\mathcal{H}) = \tilde{O}(|X|)$ to evaluate the MinHash, which completes the proof. □

6 Algorithm for k-Center Against an Adaptive Adversary

We describe a deterministic algorithm for k-center that maintains an $O(\min\{\log(n/k)/\log\log(n+\Delta), k\})$-approximate solution with an update time of $O(k \log n \cdot \log \Delta \cdot \log(n+\Delta))$. Given some constant $\epsilon > 0$, we maintain one data structure for each value OPT' that is a power of $1+\epsilon$ in $[1, \Delta]$, i.e., $O(\log_{1+\epsilon} \Delta)$ many. The data structure for each such value OPT' outputs a solution of cost at most $O(\min\{\log(n/k)/\log\log(n+\Delta), k\}) \mathrm{OPT}'$ or asserts that $\mathrm{OPT} > \mathrm{OPT}'$. We output the solution with smallest cost which is hence a $O(\min\{\log(n/k)/\log\log(n+\Delta), k\})$-approximation.

Given the value OPT', our algorithm maintains a hierarchy on the input represented by a B-ary tree T, which we call *clustering tree*, where $B = \log(n+\Delta)$. The main property of the clustering tree is that the input points are stored in the leaves and each inner node stores a k-center set for the Bk centers that are stored at its B children.

Definition 35 (clustering tree)**.** *Let* $\mathrm{OPT}' > 0$*, let* P *be a set of points and let* T *be a* B*-ary tree, where* $B \geq 2$*. We call* T *a* clustering tree *on* P *with* node-cost OPT' *if the following conditions hold:*

1. *each node* u *stores at most* Bk *points from* P*, denoted* P_u*,*
2. *for each node* u*, at most* k *points, denoted* C_u*, are marked as centers. Either, their* k*-center cost is at most* OPT' *on all points stored in* u*, or* u *is marked as a witness that there is no center set with cost at most* $\mathrm{OPT}'/2$*,*
3. *each inner node stores the at most* Bk *centers of its children.*

For each node u in T, the algorithm maintains a corresponding graph on the at most Bk points P_u it stores, which is called *blocking graph*. Without loss of generality, we assume that T is a full B-ary tree with $n/(Bk)$ leaves. We explain in the proof of Theorem 44 how to get rid of this assumption. For the sake of simplicity, we identify a node u with its associated blocking graph $N = (V, E)$ in the following. For each node $N = (V, E)$, at most k points are marked as centers (isCenter in Algorithm 4), and the algorithm maintains the invariant that two centers $u, v \in V$ have distance at least OPT' by keeping record of *blocking* edges in the blocking graph between centers and points that have distance less than OPT' to one of these centers. We say that a center u *blocks* a point v (from being a center) if there is an edge (u, v) in the blocking graph. In addition, the algorithm records whether N contains more than k points with pairwise distance greater than OPT' (lowerBoundWitness in Algorithms 4 and 5).

Insertions (see `InsertPoint`). When a point u is inserted into T, a node $N = (V, E)$ with less than Bk points is selected and it is checked whether $d(v, u) \leq \mathrm{OPT}'$ for any center $v \in V$. If this is the case, the algorithm inserts an edge (u, v) for every such center v into E and terminates afterwards. Otherwise, the algorithm checks whether the number of centers is less than k. If this is the case, it marks u as a center and inserts an edge (u, w) for *each point* $w \in V$ with $d(u, w) \leq \mathrm{OPT}'$ and recurses on the parent of N. Otherwise, if there are more than $k-1$ centers in u, the algorithm marks N as witness and terminates.

Deletions (see `DeletePoint`). When a point u is deleted from T, the point is first removed from the leaf N (and the blocking graph) where it is stored. If u was not a center, the algorithm terminates. Otherwise, the algorithm checks whether any points were unblocked (have no adjacent node in the blocking graph) and, if this is the case, proceeds by attempting to mark these points as centers and inserting them into the parent of N one by one (after marking the first point as center, the remaining points may be blocked again). Afterwards, the algorithm recurses on the parent of N.

Algorithm 4: Insertion and deletion of a point P in a node of the clustering tree T (represented by a blocking graph G).

```
Data: isCenter is a boolean array on the elements of $V$, lowerBoundWitness is a boolean
      array on the nodes of $T$
Function InsertIntoNode($N = (V, E), p, \mathrm{OPT}'$)
    insert $p$ into $V$ ;
    foreach $v \in V \setminus \{p\}$ do
        if isCenter[$v$] $\wedge\, d(p, v) \leq \mathrm{OPT}'$ then
            insert $(v, p)$ into $E$ ;
    return TryMakeCenter ($G, p, \mathrm{OPT}'$) ;
Function DeleteFromNode($N = (V, E), p, \mathrm{OPT}'$)
    newCenters $\leftarrow \emptyset$ ;
    neighbors $\leftarrow \Gamma(p)$ ;
    delete $p$ from $N$ ;
    if isCenter[$p$] $= true$ then
        foreach $u \in$ neighbors do
            newCenters $\leftarrow$ newCenters $\cup$ TryMakeCenter($N$, $u$, $\mathrm{OPT}'$) ;
    return newCenters ;
Function TryMakeCenter($N = (V, E), p, \mathrm{OPT}'$)
    if $\deg(p) = 0 \wedge |\{v \mid v \in V \wedge$ isCenter[$v$]$\}| < k$ then
        isCenter[$p$] $\leftarrow true$ ;
        foreach $v \in V \setminus \{p\}$ do
            if $d(p, v) \leq \mathrm{OPT}'$ then
                insert $(p, v)$ into $E$ ;
        lowerBoundWitness[$N$] $\leftarrow false$ ;
        return $\{p\}$ ;
    else if $\deg(p) = 0$ then
        lowerBoundWitness[$N$] $\leftarrow true$ ;
    return $\emptyset$ ;
```

Given a rooted tree T and a node u of T, we denote the subtree of T that is rooted at u by $T(u)$. For a clustering tree T, we denote the set of all points stored at the leaves of $T(u)$ by $\mathcal{P}(u)$. Recall that the points directly stored at u are denoted by P_u. Observe that for each node u in a clustering tree, $T(u)$ is a clustering tree of $\mathcal{P}(u)$.

6.1 Structural Properties of the Algorithm

We show that Algorithm 5 maintains a clustering tree.

Lemma 36. *Let T be a clustering tree on a point set P. After calling* `InsertPoint`(T, p) *(see Algorithm 5) for some point $p \notin P$, T is a clustering tree on $P \cup \{p\}$.*

Proof. We prove the statement by induction over the recursive calls of `InsertIntoNode` in `InsertPoint`. Let $N = (V, E)$ be the leaf in T where p is inserted. Condition 1 in Definition 35 is guaranteed for N by `InsertPoint`. The algorithm `InsertIntoNode` ensures that p is marked as a center only if it is not within distance OPT' of any other center. Let C be the center set of N before inserting p. By

Algorithm 5: Insertion and deletion of a point p in the clustering tree T.

```
Data: lowerBoundWitness is a boolean array on the nodes of $T$
Function InsertPoint($T, p, \mathrm{OPT}'$)
    $N \leftarrow$ leaf in $T$ that contains less than $Bk$ elements ;
    do
        centers $\leftarrow$ InsertIntoNode($N, p$) ;
        $N \leftarrow N.parent$ ;
    while centers $= \{p\}$;
Function DeletePoint($T, p, \mathrm{OPT}'$)
    $N \leftarrow$ leaf in $T$ that contains $p$ ;
    centers $\leftarrow \emptyset$; failed $\leftarrow false$ ;
    do
        centers $\leftarrow$ centers $\cup$ DeleteFromNode($N, p$) ;
        newCenters $\leftarrow \emptyset$ ;
        foreach $v \in$ centers do
            newCenters $\leftarrow$ newCenters $\cup$ InsertIntoNode($N.parent$, $v$) ;
        centers $\leftarrow$ newCenters ;
        $N \leftarrow N.parent$ ;
    while $N \neq null$;
```

condition 2, C has $\mathrm{cost}_k(C, V) \leq \mathrm{OPT}'$. Let C' be the center set of any optimal solution on V. If $\mathrm{cost}_k(C', V) \leq \mathrm{OPT}'/2$, each center of C covers at least one cluster of C'. By pigeonhole principle, p is not blocked by a center in C if and only if $|C| < k$. Otherwise, if $\mathrm{cost}_k(C', V) > \mathrm{OPT}'/2$, p is chosen if no center in C covers p and $|C'| < k$, or N is marked as witness. It follows that condition 2 is still satisfied for N after `InsertIntoNode` terminates.

Now, let $N = (V, E)$ be any inner node in a call to `InsertIntoNode`(N, p). We note that such call is only made if p was marked as a center in its child N' on which `InsertIntoNode` was called before by `InsertPoint`. Thus, p is inserted into V if and only if p is a center in N'. Therefore, condition 2 is satisfied for N. By the above reasoning, condition 3 is also satisfied. □

Lemma 37. *Let T be a clustering tree on a point set P. After calling* `DeletePoint`(T, p) *(see Algorithm 5) for some point $p \in P$, T is a clustering tree on $P \setminus \{p\}$.*

Proof. We prove the statement by induction over the loop's iterations in `DeletePoint`. Let $N = (V, E)$ be the leaf in T where p was inserted. `DeleteFromNode` deletes p from N and iterates over all unblocked points to mark them as centers one by one. After deleting p, condition 1 holds for N. Let U be the set of unblocked points after removing p, let C be the center set after removing p from V, and let C' be the center set of any optimal solution on $V \setminus \{p\}$. If $\mathrm{cost}_k(C', V) \leq \mathrm{OPT}'/2$, each unblocked point from U covers at least one cluster of C' with cost OPT'. Therefore, any selection of $k - |C|$ points from U that do not block each other together with C is a center set for V with cost OPT'. Otherwise, if $\mathrm{cost}_k(C', V) > \mathrm{OPT}'/2$, a set of at most $k - |C|$ points from U is chosen, or N is marked as witness. It follows that condition 2 is still satisfied for N after `DeleteFromNode` terminates. Let C'' be the center set that is returned by `DeleteFromNode`. `DeletePoint` inserts all points from C'' into the parent node. This reinstates condition 3 on the parent node. Then, `DeleteFromNode` recurses on the parent. □

6.2 Correctness of the Algorithm

We use the following notion of super clusters and its properties to prove the $O(k)$ upper bound on the approximation ratio of the algorithm.

Definition 38 (super cluster)**.** *Let P be a set of points and let C be a center set with $\mathrm{cost}_k(C,P) \le 2\,\mathrm{OPT}'$. Consider the graph $G=(C,E)$, where $E=\{(u,v) \mid d(u,v) \le 2\,\mathrm{OPT}'\}$. For every connected component in G, we call the union of clusters corresponding to this component a* super cluster*.*

Lemma 39. *Let T be a clustering tree constructed by Algorithm 5 with node-cost OPT' and no node marked as witness. For any node N in T, N contains one point from each super cluster in $\mathcal{P}(N)$ that is marked as center.*

Proof. Let S be a supercluster of P. Let N be a node in T that contains a point $p \in S$. For the sake of contradiction, assume that there exists no point $q \in S \cap N$ that is marked as center. By Definition 38, the k-center clustering cost of the centers in N is greater than OPT'. By Definition 35.2, this implies that a point must be marked as witness. This is a contradiction to the assumption that no node is marked as witness. □

The following simple observation leads to the bound of $O(\log(n/k)/\log\log(n+\Delta))$ on the approximation ratio.

Observation 40. *Let $c > 0$, $\ell \in \mathbb{N}$, let $P_1,\dots,P_\ell$ be sets of points and let $C_1,\dots,C_\ell$ so that $\mathrm{cost}_k(C_i,P_i) \le c$ for every $i \in [\ell]$. For every k-center set C' with $\mathrm{cost}_k(C', \cup_i C_i) \le c'$ we have $\mathrm{cost}_k(C', \cup_i P_i) \le c + c'$ by the triangle inequality.*

We combine the previous results and obtain the following approximation ratio for our algorithm.

Lemma 41. *Let T be a clustering tree on a point set P that is constructed by Algorithm 5 with node-cost OPT'. Let C be the points in the root of T that are marked as centers. If no node in C is marked as witness, $\mathrm{cost}_k(C,P) \le \min\{k, \log(n/k)/\log B\} \cdot 4\mathrm{opt}_k(P)$. Otherwise, $\mathrm{opt}_k(P) \ge \mathrm{OPT}'/2$.*

Proof. Assume that $\mathrm{opt}_k(P) \le \mathrm{OPT}'/2$, as otherwise, there exists a node that stores at least $k+1$ points that have pairwise distance OPT', which implies the claim. First, we prove $\mathrm{cost}_k(C,P) \le \log(n/(Bk))/\log B \cdot \mathrm{OPT}'$. It follows from Observation 40 that C has k-center cost $2\,\mathrm{OPT}'$ on the points stored in the root's children. Since T has depth at most $\log(n/(Bk))/\log B$, it follows by recursively applying Observation 40 on the children that C also has cost $\log(n/(Bk))/\log B \cdot 2\,\mathrm{OPT}'$ on P.

Now, we prove $\mathrm{cost}_k(C,P) \le k$. Let $p \in P$, and let S be the super cluster of p with corresponding optimal center set C'. We have $d(p,C') \le \mathrm{OPT}'/2$. By Lemma 39, there exists a center $q \in C$ so that $d(q,C') \le \mathrm{OPT}'/2$. By the definition of super clusters, for every $x,y \in C'$, $d(x,y) \le (k-1)\cdot 2\,\mathrm{OPT}'$. It follows from the triangle inequality that $d(p,q) \le 2k\,\mathrm{OPT}'$. □

6.3 Update Time Analysis

Lemma 42. *The amortized update time of Algorithm 5 is $O(Bk\log(n/k)/\log B)$.*

Proof. To maintain blocking graphs efficiently, the graphs are stored in adjacency list representation and the degrees of the vertices as well as the number of centers are stored in counters. When a point p is inserted, $3Bk\log(n/(Bk))/\log B$ tokens are paid into the account of p. Each token pays for a (universally) constant amount of work.

Each point is inserted at most once in each of the $\log(n/(Bk))/\log B$ nodes from the leaf where it is inserted up to the root. The key observation is that it is marked as a center in each of these nodes at most once (when it is inserted, or later when a center is deleted): Marking a point as center is irrevocable until it is deleted. For each node N and point $p \in N$, it can be checked in constant time whether p can be marked as a center in N by checking its degree in the blocking graph. Marking p as a center takes time $O(Bk)$ because it is sufficient to check the distance to all other at most Bk points in N and insert the corresponding blocking edges. For each point p, we charge the time it takes to *mark* p as a center to the account of p. Therefore, marking p as center withdraws a most $Bk \log(n/(Bk))$ tokens from its account in total.

Consider the insertion of a point p into a node. As mentioned, p is inserted in at most $\log(n/(Bk))/\log B$ nodes, and in each of these nodes, it is inserted at most once (when it is inserted into the tree, or when it is marked as a center in a child node). Inserting a point into a node N requires the algorithm to check the distance to all at most k centers in N to insert blocking edges, which results in at most $k \log(n/(Bk))/\log B$ work in total.

It remains to analyze the time that is required to update the tree when a point p is deleted. For any node N, if p is not a center in N, deleting p takes constant time. Otherwise, if p is a center, the algorithm needs to check its at most Bk neighbors in the blocking graph one by one whether they can be marked as centers. Checking a point q takes only constant time, and marking q as a center has already been charged to q by the previous analysis. All centers that have been marked have to be inserted into the parent of N, but this has also been charged to the corresponding points. Therefore, deleting p consumes at most $Bk \log(n/(Bk))/\log B$ tokens from the account of p. □

Lemma 43. *The worst-case insertion time of Algorithm 5 is $O(Bk \log(n/k)/\log B)$, and the worst-cast deletion time is $O(Bk^2 \log(n/k)/\log B)$.*

Proof. To maintain blocking graphs efficiently, the graphs are stored in adjacency list representation and the degrees of the vertices as well as the number of centers are stored in counters.

Each point is inserted at most once in each of the $\log(n/(Bk))/\log B$ nodes from the leaf where it is inserted up to the root. For each node N and point $p \in N$, it can be checked in constant time whether p can be marked as a center in N by checking its degree in the blocking graph. Marking p as a center takes time $O(Bk)$ because it is sufficient to check the distance to all other at most Bk points in N and insert the corresponding blocking edges.

Consider the insertion of a point p into the tree. As mentioned, p is inserted in at most $\log(n/(Bk))/\log B$ nodes. Therefore, during the initial insertion of p into the tree, the insertion procedure may insert p into at most $\log(n/(Bk))/\log B$ nodes. Inserting a point into a node N requires the algorithm to check the distance to all at most k centers in N to insert blocking edges and possibly marking it as a center, which results in at most $O(Bk \log(n/(Bk))/\log B)$ time in total.

Now, consider the deletion of p from the tree. For any node N, if p is not a center in N, deleting p takes constant time. Otherwise, if p is a center, the algorithm needs to check its at most Bk neighbors in the blocking graph one by one whether they can be marked as centers. Checking a point q takes only constant time, and marking q as a center takes time $O(Bk)$. All at most k points that have been marked have to be inserted into the parent of N, which takes time $O(Bk \cdot k)$. As the length of any path from a leaf to the root is at most $\log(n/(Bk))/\log B$, deleting p requires at most $O(Bk^2 \log(n/(Bk))/\log B)$ time. □

6.4 Main Theorem

It only remains to combine all previous results to obtain Theorem 4.

Theorem 44 (Theorem 4)**.** *Let $\epsilon, k > 0$ and $B \geq 2$. There exists a deterministic algorithm for the dynamic k-center problem that has amortized update time $O(Bk \log n \log(\Delta)/\log(1+\epsilon))$ and approximation factor $(1+\epsilon) \cdot \min\{4k, 4\log(n/k)/\log B\}$. The worst-cast insertion time is*

$O(Bk \log n \log \Delta / \log(1+\epsilon))$*, and the worst-case deletion time is* $O(Bk^2 \log n \log \Delta / \log(1+\epsilon))$.

Proof. Since $P = (P_1, \ldots, P_n)$ is a dynamic point set, its size $n_i := |P_i|$ can increase over time. Therefore, we need to remove the assumption that the clustering tree has depth $\log(\max_{t\in[n]} n_t)$. First, we note that we can insert and delete points so that the clustering tree T is a complete B-ary tree (the inner nodes induce a full B-tree, and all leaves on the last level are aligned left): We always insert points into the left-most leaf on the last level of T that is not full; when a point is deleted from a leaf N that is not the right-most leaf N' on the last level of T, we delete an arbitrary point q from N' and insert q into N. Deleting and reinserting points this way can be seen as two update operations, and therefore, it can only increase time required to update the tree by a factor of 3. Furthermore, any leaf can be turned into an inner node by adding a copy of itself as its left child and adding an empty node as its right child. Vice versa, an empty leaf and its (left) sibling can be contracted into its parent. This way, the algorithm can guarantee that the depth of the tree is between $\lfloor \log(n_t/(Bk))/\log B \rfloor$ and $\lceil \log(n_t/(Bk))/\log B \rceil$ at all times $t \in [n]$.

Recall that $d(x,y) \geq 1$ for every $x, y \in \mathcal{X}$, and $\Delta = \max_{x,y\in\mathcal{X}} d(x,y)$. For every $\Gamma \in \{(1+\epsilon)^i \mid i \in [\lceil \log_{1+\epsilon}(\Delta) \rceil]\}$, the algorithm maintains an instance T_Γ of a clustering tree with node-cost $\mathrm{OPT}' = (1+\epsilon)^\Gamma$ by invoking Algorithm 5. After every update, the algorithm determines the smallest Γ so that no node in T_Γ is marked as witness, and it reports the center set of the root of T_Γ. The bound on the cost follows immediately from Lemma 41. Since there are at most $\log(\Delta)/\log(1+\epsilon)$ instances, the bounds on the running time follow from Lemmas 42 and 43. □

7 Algorithms for k-Sum-of-Radii and k-Sum-of-Diameters

In this section we present our (randomized) dynamic $(13.008+\epsilon)$-approximation algorithm for k-sum-of-radii with an amortized update time of $k^{O(1/\epsilon)} \log \Delta$, against an oblivious adversary. Our strategy is to maintain a bi-criteria approximation with $O(k/\epsilon)$ clusters whose sum of radii is at most $(6+\epsilon)\,\mathrm{OPT}$ (and which covers all input points). We show how to use an arbitrary offline α-approximation algorithm to turn this solution into a $(6+2\alpha+\epsilon)$-approximate solution with only k clusters. Using the algorithm in [CP04] (for which $\alpha = 3.504+\epsilon$), this yields a dynamic $(13.008+\epsilon)$-approximation for k-sum-of-radii, and hence a $(26.015+\epsilon)$-approximation for k-sum-of-diameters. We provide the algorithm and a high-level overview of its analysis in Sections 7.1 and 7.2. The details of the formal proofs can be found in Section 7.3.

7.1 Bicriteria Approximation

Assume that we are given an $\epsilon > 0$ such that w.l.o.g. it holds that $1/\epsilon \in \mathbb{N}$. We maintain one data structure for each value OPT' that is a power of $1+\epsilon$ in $[1,\Delta]$, i.e., $O(\log_{1+\epsilon}\Delta)$ many. The data structure for each such value OPT' outputs a solution of cost at most $(13.008+\epsilon)\,\mathrm{OPT}'$ or asserts that $\mathrm{OPT} > \mathrm{OPT}'$. We output the solution with smallest cost which is hence a $(13.008+O(\epsilon))$-approximation. We describe first how to maintain the mentioned bi-criteria approximation. We define $z := \epsilon\,\mathrm{OPT}'/k$. Our strategy is to maintain a solution for an auxiliary problem based on a Lagrangian relaxation-type approach. More specifically, we are allowed to select an arbitrarily large number of clusters, however, for each cluster we need to pay a fixed cost of z plus the radius of the cluster. For the radii we allow only integral multiples of z that are bounded by OPT', i.e., only radii in the set $R = \{z, 2z, ..., \mathrm{OPT}' - z, \mathrm{OPT}'\}$.

This problem can be modeled by an integer program. We formulate the problem as an LP (P) which has a variable $x_p^{(r)}$ for each combination of a point p and a radius r from a set of (suitably discretized) radii R, and a constraint for each point p. Let (D) denote its dual LP, see below, where

$z := \epsilon\,\mathrm{OPT}'/k$.

$$\begin{aligned} \min \quad & \sum_{p\in P}\sum_{r\in R} x_p^{(r)}(r+z) \\ \text{s.t.} \quad & \sum_{p'\in P}\sum_{r:d(p,p')\le r} x_{p'}^{(r)} \ge 1 \ \ \forall p\in P \qquad (P) \\ & x_p^{(r)} \ge 0 \ \ \forall p \in P\ \forall r\in R \end{aligned} \qquad \begin{aligned} \max \quad & \sum_{p\in P} y_p \\ \text{s.t.} \quad & \sum_{p'\in P: d(p,p')\le r} y_{p'} \le r+z \ \ \forall p\in P, r\in R \qquad (D) \\ & y_p \ge 0 \qquad \forall p\in P \end{aligned}$$

We describe first an offline primal-dual algorithm that computes an integral solution to (P) and a fractional solution to the dual (D), so that their costs differ by at most a factor 6. By weak duality, this implies that our solution for (P) is a 6-approximation. We initialize $x \equiv 0$ and $y \equiv 0$, define $U_1 := P$, and we say that a pair (p', r') *covers a point* p if $d(p', p) \le r'$. Our algorithm works in iterations.

At the beginning of the i-th iteration, assume that we are given a set of points U_i, a vector $\left(x_p^{(r)}\right)_{p\in P, r\in R}$, and a vector $(y_p)_{p\in P}$, such that U_i contains all points in P that are not covered by any pair (p,r) for which $x_p^{(r)} = 1$ (which is clearly the case for $i = 1$ since $U_1 = P$ and $x \equiv 0$). We select a point $p \in U_i$ uniformly at random among all points in U_i. We raise its dual variable y_p until there is a value $r \in R$ such that the dual constraint for (p,r) becomes *half-tight,* meaning that $\sum_{p':d(p,p')\le r} y_{p'} = r/2 + z$.

Note that it might be that the constraint is already at least half-tight at the beginning of iteration i in which case we do not raise y_p but still perform the following operations. Assume that r is the largest value for which the constraint for (p,r) is at least half-tight. We define $x_p^{(2r)} := 1$, $(p_i, r_i) := (p, 2r)$ and set U_{i+1} to be all points in U_i that are not covered by $(p, 2r)$.

Lemma 45. *Each iteration i needs a running time of $O(|U_i| + ik/\epsilon)$.*

We stop when in some iteration i^* it holds that $U_{i^*} = \emptyset$ or if we completed the $(2k/\epsilon)^2$-th iteration and $U_{(2k/\epsilon)^2+1} \neq \emptyset$. Suppose that x and y are the primal and dual vectors after the last iteration. In case that $U_{(2k/\epsilon)^2} \neq \emptyset$ we can guarantee that our dual solution has a value of more than OPT' from which we can conclude that $\mathrm{OPT} > \mathrm{OPT}'$; therefore, we stop the computation for the estimated value OPT' in this case.

Lemma 46. *If $U_{(2k/\epsilon)^2+(2k/\epsilon)} \neq \emptyset$ then $\mathrm{OPT} > \mathrm{OPT}'$.*

If $U_{(2k/\epsilon)^2} = \emptyset$ we perform a pruning step in order to transform x into a solution whose cost is at most by a factor 3 larger than the cost of y. We initialize $\bar{S} := \emptyset$. Let $S = \{(p_1, r_1), (p_2, r_2), ...\}$ denote the set of pairs (p, r) with $x_p^{(r)} = 1$. We sort the pairs in S non-increasingly by their respective radius r. Consider a pair (p_j, r_j). We insert the cluster $(p_j, 3r_j)$ in our solution $\bar{S}$ and delete from S all pairs $(p_{j'}, r_{j'})$ such that $j' > j$ and $d(p_j, p_{j'}) < r_j + r_{j'}$. Note that $(p_j, 3r_j)$ covers all points that are covered by any deleted pair $(p_{j'}, r_{j'})$ due to our ordering of the pairs. Let $\bar{S}$ denote the solution obained in this way and let $\bar{x}$ denote the corresponding solution to (P), i.e., $\bar{x}_p^{(r)} = 1$ if and only if $(p,r) \in \bar{S}$. We will show below that $\bar{S}$ is a feasible solution to (P) with at most $O(k/\epsilon)$ clusters. Let $\bar{C} \subseteq P$ denote their centers, i.e., $\bar{C} = \{p \mid \exists r \in R : (p,r) \in \bar{S}\}$.

Lemma 47. *Given x we can compute $\bar{x}$ in time $O((k/\epsilon)^4)$ and $\bar{x}$ selects at most $O(k/\epsilon)$ centers.*

We transform now the bi-criteria approximate solution $\bar{x}$ into a feasible solution $\tilde{x}$ with only k clusters. To this end, we invoke the offline $(3.504+\epsilon)$-approximation algorithm from [CP04] on the input points $\bar{C}$. Let $\hat{S}$ denote the set of pairs $(\hat{p}, \hat{r})$ that it outputs (and note that not necessarily $\hat{r} \in R$ since we use the algorithm as a black-box). Note that $\hat{S}$ covers only the points in $\bar{C}$, and not necessarily all points in P. On the other hand, the solution $\hat{S}$ has a cost of at most $2\,\mathrm{OPT}$ since we can always find a solution with this cost covering $\bar{C}$, even if we are only allowed to select centers from $\bar{C}$. Thus, based on $\bar{S}$ and $\hat{S}$ we compute a solution $\tilde{S}$ with at most k clusters that covers P. We initialize $\tilde{S} := \emptyset$. For each pair $(\hat{p}, \hat{r}) \in \hat{S}$ we consider the points $\bar{C}'$ in $\bar{C}$ that are covered by $(\hat{p}, \hat{r})$. Among all these points, let $\bar{p}$ be the point with maximum radius $\bar{r}$ such that $(\bar{p}, \bar{r}) \in \bar{S}$. We add to $\tilde{S}$ the pair $(\hat{p}, \hat{r} + \bar{r})$ and remove $\bar{C}'$ from $\bar{C}$. We do this operation with each pair $(\hat{p}, \hat{r}) \in \hat{S}$. Let $\tilde{S}$ denote the resulting set of pairs.

Lemma 48. *Given $\bar{S}$ and $\hat{S}$ we can compute $\tilde{S}$ in time $O(k^3/\epsilon^2)$.*

We show that $\tilde{S}$ is a feasible solution with small cost. We start by bounding the cost of $\bar{x}$ via y.

Lemma 49. *We have that $\bar{x}$ and y are feasible solutions to (P) and (D), respectively, for which we have that $\sum_{p \in P} \sum_{r \in R} \bar{x}_p^{(r)} (r + z) \le 6 \cdot \sum_{p \in P} y_p \le 6\,\mathrm{OPT}'$.*

Next, we argue that $\tilde{S}$ is feasible and bound its cost by the cost of S' and the cost of $\bar{x}$.

Lemma 50. *We have that $\tilde{S}$ is a feasible solution with cost at most $\sum_{(\hat{p},\hat{r}) \in \hat{S}} \hat{r} + \sum_{p \in P} \sum_{r \in R} \bar{x}_p^{(r)} (r + z) \le (13.008 + \epsilon)\,\mathrm{OPT}'$.*

7.2 Dynamic Algorithm

We describe now how we maintain the solutions $x, \bar{x}, y, S, \bar{S}$, and $\tilde{S}$ dynamically when points are inserted or deleted. Our strategy is similar to [CGS18].

Suppose that a point p is inserted. For each $i \in \{1, ..., 2k/\epsilon + 1\}$ we insert p into the set U_i if $U_i \neq \emptyset$ and p is not covered by a pair (p_j, r_j) with $j \in \{1, ..., i-1\}$. If there is an index $i \in \{1, ..., 2k/\epsilon + 1\}$ such that $U_{i-1} \neq \emptyset$ (assume again that $U_0 \neq \emptyset$), $U_i = \emptyset$, and p is not covered by any pair (p_j, r_j) with $j \in \{1, ..., i-1\}$, we start the above algorithm in the iteration i, being initialized with $U_i = \{p\}$ and the solutions x, y as being computed previously.

Suppose now that a point p is deleted. We remove p from each set U_i that contains p. If there is no $r \in R$ such that $(p, r) \in S$ then we do not do anything else. Assume now that $(p, r) \in S$ for some $r \in R$. The intuition is that this does not happen very often since in each iteration i we choose a point uniformly at random from U_i. More precisely, in expectation the adversary needs to delete a constant fraction of the points in U_i before deleting p. Consider the index i such that $(p, r) = (p_i, r_i)$. We restart the algorithm from iteration i. More precisely, we initialize y to the values that they have after raising the dual variables $y_{p_1}, ..., y_{p_{i-1}}$ in this order as described above until a constraint for the respective point p_j becomes half-tight. We initialize x to the corresponding primal variables, i.e., $x_{p_j}^{(2r_j)} = 1$ for each $j \in \{1, ..., i-1\}$ and $x_{p'}^{(r')} = 0$ for all other values of p', r'. Also, we initialize the set U_i to be the obtained set after removing p. With this initialization, we start the algorithm above in iteration i, and thus compute like above the (final) vectors y, x and based on them $S, \bar{S}$, and $\tilde{S}$.

When we restart the algorithm in some iteration i then it takes time $O(|U_i| k^2)$ to compute the new set S. We can charge this to the points that were already in U_i when U_i was recomputed the last time and to the points that were inserted into U_i later. After a point p was inserted, it is charged at most $O(k/\epsilon)$ times in the latter manner since it appears in at most $O(k/\epsilon)$ sets U_i. Finally, given S, we can compute the sets $\bar{S}$, $\hat{S}$, and $\tilde{S}$ in time $O(k^3/\epsilon^2)$. The algorithm from [CP04] takes time $n^{O(1/\epsilon)}$ if the input has size n. One can show that this yields an update time of $k^{O(1/\epsilon)} + (k/\epsilon)^4$ for each value OPT'. Finally, the same set $\tilde{S}$ yields a solution for k-sum-of-diameters, increasing the approximation ratio by a factor of 2.

Theorem 6. *There are randomized dynamic algorithms for the k-sum-of-radii and the k-sum-of-diameters problems with update time $k^{O(1/\epsilon)} \log \Delta$ and with approximation ratios of $13.008 + \epsilon$ and $26.016 + \epsilon$, respectively, against an oblivious adversary.*

7.3 Deferred Proofs

In this section, we formally prove the correctness of Algorithm 6 for the k-sum-of-radii and the k-sum-of-diameter problem. To compute a solution in the static setting, the algorithm is invoked as `PrimalDual`$(P, \emptyset, R, z, k, \epsilon, 0, 0, 0)$, where P is the set of input points, R is the set of radii, $z = \epsilon \,\mathrm{OPT}'/k$ is the facility cost and k is the number of clusters. Algorithm 6 is a pseudo-code version of the algorithm described earlier.

Algorithm 6: Pseudo code of the primal-dual algorithm for k-sum-of-radii as described in Section 7.

```
Data: point set $P$, unassigned point sets $U = \{U_0, \dots\}$, radii set $R$, facility cost $z$, number
      of clusters $k$, precision $\epsilon$, primal vector $x = \{x_p^{(r)} \mid r \in R \wedge p \in P\}$, dual vector
      $Y = \{y_p \mid p \in P\}$
Function PrimalDual $(P, U, R, z, k, \epsilon, x, y, i)$
    while $U_i \neq \emptyset$ do
        $p_i \leftarrow$ uniformly random point from $U_i$ ;
        $\delta_i \leftarrow \max(\{0\} \cup \{\delta' \mid \delta' \in \mathbb{R} \wedge \forall r' \in R : \sum_{p' \in P : d(p_i, p') \leq r'} y_{p'} + \delta' \leq r'/2 + z\})$ ;
        if $\delta_i > 0$ then
            $r_i \leftarrow \max\{r' \mid r' \in R \wedge \sum_{p' \in P : d(p_i, p') \leq r'} y_{p'} + \delta_i = r'/2 + z\}$ ;
        else
            $r_i \leftarrow \max\{r' \mid r' \in R \wedge \sum_{p' \in P : d(p_i, p') \leq r'} y_{p'} \geq r'/2 + z\}$ ;
        $y_{p_i} \leftarrow \delta_i$; $x_{p_i}^{(2r_i)} \leftarrow 1$ ;
        $U_{i+1} \leftarrow U_i \setminus \{p' \mid p' \in U_i \wedge d(p_i, p') \leq 2r_i\}$ ;
        $i \leftarrow i + 1$ ;
        if $i > (2k/\epsilon)^2$ then
            return "$\mathrm{OPT}' < \mathrm{OPT}$" ;
    $\bar{S} \leftarrow \emptyset$; $S \leftarrow$ sort $(p_j, r_j)_{j \in [i-1]}$ non-increasingly according to $r_j$ ;
    forall $(p, r) \in S$ do
        if $\nexists (p_j, r_j) \in \bar{S} : d(p, p_j) \leq r + r_j$ then
            $\bar{S} \leftarrow \bar{S} \cup \{(p, 3r)\}$ ;
    return $\bar{S}, U, x, y, r, i$ ;
```

Proof of Lemma 45 We bound the running time of a single primal-dual step. This implies Lemma 45.

Lemma 51 (Lemma 45)**.** *The running time of the ith iteration of the while-loop in* `PrimalDual` *is $O(ik/\epsilon + |U_i|)$.*

Proof. In each iteration, at most one entry of y, i.e., y_{p_i}, increases. Therefore, at most $i - 1$ entries of y are non-zero at the beginning of iteration i. By keeping a list of non-zero entries, each sum corresponding to a constraint of the dual program can be computed in time $O(i)$. Since $|R| \leq k/\epsilon$, it follows that δ_i and r_i can be computed in time $O(ik/\epsilon)$. To construct U_{i+1}, it is sufficient to iterate over U_i once. □

Proof of Lemma 46 We show that our choice of $z = \epsilon \operatorname{OPT}' / k$ will result in a solution $\bar{S}$ if $\mathrm{OPT} \le \mathrm{OPT}'$.

Lemma 46. *If $U_{(2k/\epsilon)^2+(2k/\epsilon)} \neq \emptyset$ then* $\mathrm{OPT} > \mathrm{OPT}'$.

Proof. Since, by weak duality, $\sum_{p\in P} y_p$ is a lower bound to the optimum value of (P), we bound the number of iterations that are sufficient to guarantee $\sum_{p\in P} y_p > \mathrm{OPT}'$. Call an iteration of the while-loop in `PrimalDual` *successful* if $\delta_i > 0$, and *unsuccessful* otherwise. First, observe that for each successful iteration i, the algorithms increases y_{p_i} by at least $z/2$. This is due to the fact that all values in R are multiples of z, and therefore $r/2 + z$ is a multiple of $z/2$ for any $r \in R$. Since the algorithm increases y_{p_i} as much as possible, all y_{p_i} are multiples of $z/2$. Therefore, after $\operatorname{OPT}'/(z/2) + 1 = 2k/\epsilon + 1$ successful iterations, it holds that $\mathrm{OPT} \ge \sum_{p\in P} y_p > \mathrm{OPT}'$.

Now, we prove that for each successful iteration, there are at most $|R|$ unsuccessful iterations. Then, it follows that after $((2k/\epsilon)+1)\cdot|R| \le (2k/\epsilon)^2 + (2k/\epsilon)$ iterations, $\mathrm{OPT} > \mathrm{OPT}'$. Let i be an unsuccessful iteration. The crucial observation for the following argument is that for the maximum $r_i \in R$ so that (p_i, r_i) is at least half-tight and for any $j < i$ so that $d(p_i, p_j) \le r_i$ and $y_{p_j} > 0$, we have $r_j < r_i$: otherwise, p_i would have been removed from U_j. On the other hand, such p_j must exist because the dual constraint (p_i, r_i) is at least half-tight but $y_{p_i} = 0$. Now, we *charge* the radius r_i to the point p_j and observe that we will never charge r_i to p_j again. This is due to the fact that all points $p \in U_{i-1}$ with distance $d(p_j, p) \le r_i$ are removed from U_i because $d(p_i, p) \le d(p_i, p_j) + d(p_j, p) \le 2r_i$. Therefore, for any successful iteration j and any $r \in R$, there is at most one $i > j$ so that $r = r_i$ is charged to p_j, each accounting for an unsuccessful iteration. ☐

Proof of Lemma 47 We argue that the pruning step has running time $\mathrm{poly}(k/\epsilon)$ to prove Lemma 47.

Lemma 52 (Lemma 47)**.** *The for-loop in* `PrimalDual` *has running time $O((k/\epsilon)^4)$.*

Proof. Since $i \le 2(2k/\epsilon)^2$, it holds that $|S| \le |\{p \in P \mid \exists r \in R : x_p^{(r)} > 0\}| \le 2(2k/\epsilon)^2$. Therefore, sorting S requires at most $O((2k/\epsilon)^2 \log(2k/\epsilon))$ time. In each iteration of the loop, it suffices to iterate over $\bar{S}$. Since $|\bar{S}| \le |S|$, the claim follows. ☐

Proof of Lemma 48

Lemma 48. *Given $\bar{S}$ and $\hat{S}$ we can compute $\tilde{S}$ in time $O(k^3/\epsilon^2)$.*

Proof. Recall that we sort points in S non-increasingly. Therefore, for every point p, there exists only one $(p, r) \in S$ that is inserted into $\bar{S}$. It follows that $|\bar{S}| \le |S| \le (2k/\epsilon)^2$ and since $|\hat{S}| \le k$, it suffices to iterate over $\bar{S}$ for every $(\hat{p}, \hat{r}) \in \hat{S}$. ☐

Proof of Lemma 49 We turn to the feasibility and approximation guarantee of the solution that is computed by Algorithm 6. First, we observe that the dual solution is always feasible.

Lemma 53. *During the entire execution of Algorithm 6, no dual constraint (p, r) is violated.*

Proof. For the sake of contradiction, let i be the first iteration of the while-loop in `PrimalDual` after which there exists (p, r) such that $\sum_{p'\in P: d(p,p')\le r} > r + z$. From this definition, it follows that $d(p, p_i) \le r$ as only y_{p_i} has increased in iteration i. By the triangle inequality, for all $p' \in P$ so that $d(p, p') \le r$, we have that $d(p_i, p') \le 2r$. Therefore, the dual constraint $(p_i, 2r)$ is more than half-tight:

$$\sum_{p'\in P: d(p_i,p')\le 2r} y_{p'} \ge \sum_{p'\in P: d(p,p')\le r} y_{p'} > r + z = \frac{2r}{2} + z.$$

This is a contradiction to the choice of (p_i, r_i). ☐

The primal solution is also feasible to the LP, and its cost is bounded by $6\,\mathrm{OPT}'$.

Lemma 49. *We have that $\bar{x}$ and y are feasible solutions to (P) and (D), respectively, for which we have that $\sum_{p\in P}\sum_{r\in R}\bar{x}_p^{(r)}(r+z) \le 6\cdot\sum_{p\in P} y_p \le 6\,\mathrm{OPT}'$.*

Proof. Since $U_i = \emptyset$, x is feasible at the end of Algorithm 6. By the construction of $\bar{S}$, it follows that $\bar{x}$ is feasible. By Lemma 53, y is always feasibile for (D). It remains to bound the cost of $\bar{S}$.

By the construction of $\bar{S}$, for any $(p,r)\in\bar{S}$, $x_p^{(r/3)} = 1$ and $(p, r/3)$ is at least half-tight in y. In other words, $r + z \le 6\cdot(r/(2\cdot 3)+z) = 6\sum_{d(p,p')\le r/3} y_p$. Let $\bar{S}(p,r) = \{p' \in P \mid d(p,p') \le r\}$. For all $(p_1, r_1), (p_2, r_2) \in \bar{S}$, $p_1 \neq p_2$, we have that $\bar{S}(p_1, r_1/3)$ and $\bar{S}(p_2, r_2/3)$ are disjoint due to the construction of $\bar{S}$. Therefore, it holds that

$$\sum_{p\in P}\sum_{r\in R}\bar{x}_p^{(r)}(r+z) = \sum_{(p,r)\in\bar{S}}(r+z) \le 6\cdot\sum_{p\in P} y_p.$$

By weak duality, $6\cdot\sum_{p\in P} y_p \le 6\,\mathrm{OPT}'$. ☐

Proof of Lemma 50 Finally, we prove that the pruned solution is a feasible k-sum-of-radii solution with cost bounded by $(13.008+\epsilon)\,\mathrm{OPT}'$.

Lemma 50. *We have that $\tilde{S}$ is a feasible solution with cost at most $\sum_{(\hat{p},\hat{r})\in\hat{S}}\hat{r} + \sum_{p\in P}\sum_{r\in R}\bar{x}_p^{(r)}(r+z) \le (13.008+\epsilon)\,\mathrm{OPT}'$.*

Proof. First observe that the cost of $\hat{S}$ is bounded by $2\cdot 3.504\cdot\mathrm{OPT}$ since we can construct a solution with cost at most $2\,\mathrm{OPT}$ that covers $\bar{C}$ using only centers from $\bar{C}$: for each point $p\in\bar{C}$, take the point $p'\in\mathrm{OPT}$ covering p with some radius r', and select p with radius $2r'$.

Let $p\in P$, let $(\bar{p},\bar{r})\in\bar{S}$ so that $d(p,\bar{p})\le\bar{r}$ and let $(\hat{p},\hat{r})\in\hat{S}$ that was chosen to cover $\bar{p}$. By the triangle inequality, $d(p,\bar{p}) \le \hat{r}+\bar{r}$. Let $(\hat{p},\tilde{r})$ be the corresponding tuple in $\tilde{S}$. By the construction of $\tilde{S}$, we have that $\hat{r}+\bar{r}\le\tilde{r}$. Therefore, $\tilde{S}$ is feasible. It follows that the cost of $\tilde{S}$ is at most

$$\sum_{(\hat{p},\hat{r})\in\hat{S}}\hat{r} + \sum_{p\in P}\sum_{r\in R}\bar{x}_p^{(r)}(r+z) \le (2\cdot 3.504+6+\epsilon)\,\mathrm{OPT}'.$$ ☐

Proof of Theorem 6

Theorem 6. *There are randomized dynamic algorithms for the k-sum-of-radii and the k-sum-of-diameters problems with update time $k^{O(1/\epsilon)}\log\Delta$ and with approximation ratios of $13.008+\epsilon$ and $26.016+\epsilon$, respectively, against an oblivious adversary.*

Proof. Consider a fixed choice of OPT'. This assumption will be removed at the end of the proof. We invoke Algorithm 6 on the initial point set as for the static setting. Consider an operation t, and let $\breve{S}, \breve{U}, \breve{x}, \breve{y}, \breve{r}, \breve{i}$ be the state before this operation.

If a point p is inserted, the algorithm checks, for every $j\in\{1,\dots,i-1\}$, if $d(p_j,p)\le 2r_i$. If this is not the case for any j, p is added to $\breve{U}_j$ and the algorithm proceeds. Otherwise, the algorithm stops. If the last check fails and $i > (2k/\epsilon)^2$, the algorithm stops, too. Otherwise, it runs `PrimalDual`$(P, \breve{U}, R, z, k, \epsilon, x, y, i)$ with the updated $\breve{U}$. In any case, the point p deposits a budget of $2k/\epsilon$ tokens for each possible U_i, $i\in[2k/\epsilon]$, i.e., $(2k/\epsilon)^2$ tokens in total. By Lemmas 45 and 52, the total running time is $O((2k/\epsilon)k/\epsilon + 0 + (k/\epsilon)^4 + (2k/\epsilon)^2) = O((k/\epsilon)^4)$.

If a point p is deleted, the algorithm deletes p from all $\breve{U}_i$ it is contained in. If for all radii $r\in R$ we have that $\breve{x}_p^{(r)} = 0$ then the algorithms stops. Otherwise, let j be so that $p\in\breve{U}_j\setminus\breve{U}_{j+1}$, i.e., p is the center of the j^{th} cluster. The algorithm sets, for all $r \in R$ and all $j' \geq j$, $\breve{x}_{p_{j'}}^{(r)} = 0$, $\breve{y}_{p_{j'}} = 0$ and, for all $j' > j$, $\breve{U}_{j'} = \emptyset$. Then, it calls $\texttt{PrimalDual}(P, \breve{U}, R, z, k, \epsilon, \breve{x}, \breve{y}, j)$. By Lemmas 45 and 52, the total running time is $O((2k/\epsilon)^2 k/\epsilon + (2k/\epsilon)|U_j| + (k/\epsilon)^4)$.

The correctness of the algorithm follows from the fact that if $\mathrm{OPT}' \geq \mathrm{OPT}$, the algorithm produces a feasible solution irrespective of the choice of the p_i, $i \in [2k/\epsilon]$, by Lemma 50 and observing that the procedure described above simulates a valid run of the algorithm for $P = P_{t-1} \cup \{p\}$ and $P = P_t \setminus \{p\}$, respectively. Finally, we prove that the expected time to process all deletions up to operation t is bounded by $O(t \cdot 2k/\epsilon)$. The argument runs closely along the running time analysis in [CGS18].

Let $t' \leq t$, $j \in [2k/\epsilon]$ and let $\bar{U}_j^{(t')}$ be the set $U_i^{(t')}$ that was returned by $\texttt{PrimalDual}$ after the last call that took place before operation t' so that the argument i is such that $i \leq j$. Note that this is the last call to $\texttt{PrimalDual}$ before operation t' when U_j is reclustered. We decompose $U_j^{(t')}$ into $A_j^{(t')} = U_j^{(t')} \setminus \bar{U}_j^{(t')}$ and $B_j^{(t')} = U_j \cap \bar{U}_j^{(t')}$ and define the random variable $T_i^{t'}$, where $T_i^{t'} = |B^{(t')}|$ if operation t' deletes center p_i and $T_i^t = 0$ otherwise. Next, we bound $E[\sum_{t'<t} \sum_{i \in [2k/\epsilon]} T_i^{(t')}]$. For $t' < t$ and $i \in [2k/\epsilon]$, consider $E[T_i^{(t')}]$. Since p_i was picked uniformly at random from $B^{(t')}$, the probability that operation t' deletes p_i is $1/|B^{(t')}|$. Therefore, $E[T_i^{t'}] = 1$. By linearity of expectation, $E[\sum_{t'<t} \sum_{i \in [2k/\epsilon]} T_i^{(t')}] \leq 2k/\epsilon$. If operation t' deletes p_j, U_j is reclustered at operation t' and any point in A is not in $A^{(T_i^{t'})}$ for any $t'' \geq t'$. Therefore, each point $p \in A$ can pay $(2k/\epsilon)$ tokens from its insertion budget if p_j is deleted. The expected amortized cost for all operations up to operation t is therefore at most $O((2k/\epsilon)^2 k/\epsilon + (2k/\epsilon)^2 + (k/\epsilon)^4) = O((k/\epsilon)^4)$.

Now, we remove the assumption that OPT' is known. Recall that $d(x,y) \geq 1$ for every $x, y \in X$, and $\Delta = \max_{x,y \in \mathcal{X}} d(x,y)$. For every $\Gamma \in \{(1+\epsilon)^i \mid i \in [\lceil \log_{1+\epsilon}(k\Delta) \rceil]$, the algorithm maintains an instance of the LP with $\mathrm{OPT}' = (1+\epsilon)^\Gamma$. After every update, the algorithm determines the smallest Γ for which a solution is returned. Recall that the algorithm from [CP04] takes time $O(n^{O(1/\epsilon)})$. The total expected amortized cost is $O(k^{O(1/\epsilon)} \log \Delta)$. □

8 Lower Bound for Arbitrary Metrics

We now demonstrate that any algorithm which approximates the optimal k-center cost, in an arbitrary metric space of n points, must run in $\Omega(nk)$ time. Specifically, the input to an algorithm for k-center in arbitrary metric spaces is both the point set P *and* the metric d over the points. In particular, the input can be represented via the distance matrix distance matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ over the point set P, and the behavior of such an algorithm can be described by a sequences of adaptive queries to $\mathbf{D}$.

The above setting casts the problem of approximating the cost of the optimal k-center clustering as a *property testing* problem [GGR98, Gol17], where the goal is to solve the approximation problem while making a small number of queries to $\mathcal{D}$. Naturally, the query complexity of such a clustering algorithm lower bounds its runtime, so to prove optimality of our dynamic k-center algorithms it suffices to focus only on the query complexity. In particular, in what follows we will demonstrate that any algorithm that approximates the optimal k-center cost to any non-trivial factor with probability 2/3 must query at least $\Omega(nk)$ entries of the matrix. In particular, this rules out any fully dynamic algorithm giving a non-trivial approximation in $o(k)$ amortized update time for general metric spaces.

Moreover, we demonstrate that this lower bound holds for the (k,z)-clustering objective, which includes the well studied k-medians and k-means. Recall that this problem is defined as outputting a set $\mathcal{C} \subset \mathcal{X}$ of size at most k which minimizes the objective function

$$\mathrm{cost}_k^z(\mathcal{C}, P) = \sum_{p \in P} d^z(p, \ell(p))$$

where $\ell(p)$ is the cluster center associated with the point p. Note that $(\mathrm{cost}_k^z(\mathcal{C}, P))^{1/z}$ is always within a factor of n of the optimal k-center cost. Thus, it follows that if k-center cannot be approximated to any non-trivial factor (including factors which are polynomial in n) in $o(nk)$ queries to $\mathbf{D}$, the same holds true for (k, z)-clustering for any constant z. Thus, in the proofs of the following results we focus solely on proving a lower bound for approximation k-center to any factor R, which will therefore imply the corresponding lower bounds for (k, z)-clustering.

We do so by first proving Theorem 54, which gives a $\Omega(nk)$ lower bound when $n = \Omega(k \log k)$. Next, in Proposition 59, we prove a general $\Omega(k^2)$ lower bound for any $n > k$, which will complete the proof of Theorem 2. We note that the proof of Proposition 59 is fairly straightforward, and the main challenge will be to prove Theorem 54.

Theorem 54. *Fix and $k \geq 1$ and $n > Ck \log k$ for a sufficiently large constant C. Then any algorithm which, given oracle access the distance matrix $\mathcal{D} \in \mathbb{R}^n$ of a set X of n points in a metric space, determines correctly with probability $2/3$ whether the optimal k-center cost on X is at most 1 or at least R, for any value $R > 1$, must make at least $\Omega(kn)$ queries in expectation to $\mathcal{D}$. The same bound holds true replacing the k-center objective with (k, z)-clustering, for any constant $z > 0$.*

Proof. We suppose there exists such an algorithm that makes at most an expected $kn/8000$ queries to $\mathcal{D}$. By forcing the the algorithm to output an arbitrary guess for c whenever it queries a factor of 20 more entries than its expectation, by Markov's inequality it follows that there is an algorithm which correctly solves the problem with probability $2/3 - 1/20 > 6/10$, and always makes at most $kn/400$ queries to $\mathcal{D}$.

The Hard Distribution. We define a distribution $\mathcal{D}$ over $n \times n$ distance matrices $\mathcal{D}$ as follows. First, we select a random hash function $h : [n] \to [k]$, a uniformly random coordinate $i \sim [n]$. We then set $\mathbf{D}(h)$ to be the matrix defined by $\mathbf{D}_{p,q}(h) = 1$ for $p \neq q$ if $h(p) = h(q)$, and $\mathbf{D}_{p,q}(h) = R$ otherwise, where R is an arbitrarily large value which we will later fix. We then flip a coin $c \in \{0, 1\}$. If $c = 0$, we return the matrix $\mathbf{D}(h)$, but if $c = 1$, we define the matrix $\mathbf{D}(h, i)$ to be the matrix resulting from changing every 1 in the i-th row and column of $\mathbf{D}(h)$ to the value $2R$. It is straightforward to check that the resulting distribution satisfies the triangle inequality, and therefore always results in a valid metric space. We write $\mathcal{D}_0, \mathcal{D}_1$ to denote the distribution $\mathcal{D}$ conditioned on $c = 0, 1$ respectively. In the testing problem, a matrix $\mathbf{D} \sim \mathcal{D}$ is drawn, and the algorithm is allowed to make an adaptive sequence of queries to the entries of $\mathbf{D}$, and thereafter correctly determine with probability $2/3$ the value of c corresponding to the draw of $\mathbf{D}$.

Note that a draw from $\mathcal{D}$ can then be described by the values (h, i, c), where $h \in \mathcal{H} = \{h' : [n] \to [k]\}$, $i \in [n]$, and $c \in \{0, 1\}$. Note that, under this view, a single matrix $\mathbf{D} \sim \mathcal{D}_0$ can correspond to multiple draws of $(h, i, 0)$. Supposing there is a randomized algorithm which is correct with probability $6/10$ over the distribution $\mathcal{D}$ and its own randomness, it follows that there is a deterministic algorithm $\mathcal{A}$ which is correct with probability $6/10$ over just $\mathcal{D}$, and we fix this algorithm now.

Let $(d_1, p_1), (d_2, p_2), \ldots, (d_t, p_t)$ be an adaptive sequence of queries and observations made by an algorithm $\mathcal{A}$, where $d_i \in \{1, R, 2R\}$ is a distance and $p_i \in \binom{n}{2}$ is a position in $\mathcal{D}$, such that the algorithm queries position p_i and observed the value d_i in that position.

Claim 55. *There is an algorithm with optimal query vs. success probability trade-off which reports $c = 1$ whenever it sees an entry with value $d_i = 2R$, otherwise it reports $c = 0$ d if it never sees a distance of value $2R$.*

Proof. To see this, first note that if $c = 0$, one never sees a value of $2R$, so any algorithm which returns $c = 0$ after observing a distance of size $2R$ is always incorrect on that instance.

For the second claim, suppose an algorithm $\mathcal{A}$ returned that $c = 1$ after never having seen a value of $2R$. Fix any such sequence $S = \{(d_1, p_1), (d_2, p_2), \dots, (d_t, p_t)\}$ of adaptive queries and observations such that $d_i \neq 2R$ for all $i = 1, 2, \dots, t$. We claim that $\mathbf{Pr}[c = 0|S] \geq \mathbf{Pr}[c = 1|S]$. To see this, let $(h, i, 1)$ be any realization of a draw from $\mathcal{D}_1$, and note that $\mathbf{Pr}[(h, i, 1)] = \mathbf{Pr}[(h, i, 0)] = \frac{1}{2n} k^{-n}$. Let $F_0(S)$ be the set of tuples (h, i) such that the draw $(h, i, 0)$ could have resulted in S, and $F_1(S)$ the set of tuples (h, i) such that $(h, i, 1)$ could have resulted in s. Let $(h, i, 1)$ be a draw that resulted in S. Then $(h, i, 0)$ also results in S, because the difference between the resulting matrices $\mathcal{D}$ is supported only on positions which were initially $2R$ in the matrix generated by $(h, i, 1)$. Thus $F_1(S) \subseteq F_2(S)$, which demonstrates that $\mathbf{Pr}[c = 0|S] \geq \mathbf{Pr}[c = 1|S]$. Thus the algorithm can only improve its chances at success by reporting $c = 0$, which completes the proof of the claim. □

Decision Tree of the Algorithm. The adaptive algorithm $\mathcal{A}$ can be defined by a 3-ary decision tree T of depth at most $kn/400$, where each non-leaf node $v \in T$ is labelled with a position $p(v) = (x_v, y_v) \in [n] \times [n]$, and has three children corresponding to the three possible observations $\mathbf{D}_{p(x)} \in \{1, R, 2R\}$. Each leaf node contains only a decision of whether to output $c = 0$ or $c = 1$. For any $v \in T$, let v_1, v_R, v_{2R} denote the three children of v corresponding to the edges labelled $1, R$ and $2R$, Every child coming from a "$2R$" edge is a leaf, since by the above claim the algorithm can be assumed to terminate and report that $c = 1$ whenever it sees the value of $2R$. For any vertex $v \in T$ at depth ℓ, let $S(v) = \{(d_1, p_1), \dots, (\cdot, p(v))\}$ be the unique sequence of queries and observations which correspond to the path from the root to v. Note that the last entry $(\cdot, p(v)) \in S(v)$ has a blank observation field, meaning that at v the observation $p(v)$ has not yet been made.

For any $v \in T$ and $i \in [n]$, we say that a point i is *light* at v if the number of queries $(d_j, p_j) \in S$ with $i \in p_j$ is less than $k/2$. If i is not light at v we say that it is *heavy* at v. For any $i \in [n]$, if in the sequence of observations leading to v the algorithm observed a 1 in the i-th row or column, we say that i is *dead* at v, otherwise we say that i is *alive*. We write $\mathbf{Pr}[v]$ to denote the probability, over the draw of $\mathbf{D} \sim \mathcal{D}$, that the algorithm traverses the decision tree to v, and $\mathbf{Pr}[v|\, c = b]$ for $b \in \{0, 1\}$ to denote this probability conditioned on $\mathbf{D} \sim \mathcal{D}_b$. Next, define $F_b(v) = F_b(S(v))$ for $b \in \{0, 1\}$, where $F_b(S)$ is as above. Note that if $(h, i) \in F_0(v)$ for some $i \in [n]$, then $(h, j) \in F_0(v)$ for all $j \in [n]$, since the matrices generated by $(h, i, 0)$ are the same for all $i \in [n]$. Thus, we can write $h \in F_0(v)$ to denote that $(h, i) \in F_0(v)$ for all $i \in [n]$.

Claim 56. *Let $v \in T$ be a non-leaf node where at least one index i in $p(v) = (i, j)$ is alive and light at v. Then we have*

$$\mathop{\mathbf{Pr}}_{\mathbf{D} \sim \mathcal{D}} \left[\mathbf{D}_{p(v)} = 1 | S(v), c = 0\right] \leq \frac{2}{k}$$

Proof. Fix any function $h \in \mathcal{H}$ such that $h \in F_0(v)$: namely, h is consistent with the observations seen thus far. Let $h_1, \dots, h_k \in \mathcal{H}$ be defined via $h_t(j) = h(j)$ for $j \neq i$, and $h_t(i) = t$, for each $t \in [k]$. We claim that $h_t \in F_0(v)$ for at least $k/2$ values of t. To show this, first note that the values of $\{h(j)\}_{j \neq i}$ define a graph with at most k connected components, each of which is a clique on the set of values $j \in [n] \setminus \{i\}$ which map to the same hash bucket under h. The only way for $h_t \notin F_0(v)$ to occur is if an observation (i, ℓ) was made in $S(v)$ such that $h(\ell) = t$. Note that such an observation must have resulted in the value R, since i is still alive (so it could not have been 1). In this case, one knows that i was not in the connected component containing ℓ. However, since i

is light, there have been at most $k/2$ observations involving i in $S(v)$. Each of these observations eliminate at most one of the h_t's, from which the claim follows.

Given the above, it follows that if at the vertex v we observe $\mathbf{D}_{p(v)} = \mathbf{D}_{i,j} = 1$, then we eliminate every h_t with $t \neq h(j)$ and $h_t \in F_0(v)$. Since for every set of values $\{h(j)\}_{j \neq i}$ which are consistent with $S(v)$ there were $k/2$ such functions $h_t \in F_0(v)$, it follows that only a $2/k$ fraction of all $h \in \mathcal{H}$ result in the observation $\mathbf{D}_{p(v)} = 1$. Thus, $|\mathcal{F}_0(v_1)| \leq \frac{2}{k}|\mathcal{F}_0(v)|$, which completes the proof of the proposition. □

Let $\mathcal{E}_1$ be the set of leafs v which are children of a $2R$ labelled edge, and let $\mathcal{E}_0$ be the set of all other leaves. Note that we have $\mathbf{Pr}[v \mid c = 0] = 0$ for all $v \in \mathcal{E}_1$, and moreover $\sum_{v \in \mathcal{E}_0} \mathbf{Pr}[v|c = 0] = 1$. For $v \in T$, let $\theta(v)$ denote the number of times, on the path from the root to v, an edge (u, u_1) was crossed where at least one index $i \in p(u)$ was alive and light at u. Note that such an edge kills i, thus we have $\theta(v) \leq n$ for all nodes v. Further, define $\hat{\mathcal{E}}_0 \subset \mathcal{E}_0$ to be the subset of vertices $v \in \mathcal{E}_0$ with $\theta(v) < n/20$. We now prove two claims regarding the probabilities of arriving at a leaf $v \in \hat{\mathcal{E}}_0$.

Claim 57. *Define $\hat{\mathcal{E}}_0$ as above. Then we have*

$$\sum_{v \in \hat{\mathcal{E}}_0} \mathbf{Pr}[v|c = 0] > 9/10$$

Proof. We define indicator random variables $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_t \in \{0, 1\}$, where $t \leq kn/400$ is the depth of T, such that $\mathbf{X}_i = 1$ if the i-th observation made by the algorithm causes a coordinate $i \in [n]$, which was prior to observation i both alive and light, to die, where the randomness is taken over a uniform draw of $\mathbf{D} \sim \mathcal{D}_0$. Note that the algorithm may terminate on the t'-th step for some t' before the t-th observation, in which case we all trailing variables $\mathbf{X}_{t'}, \dots, \mathbf{X}_t$ to 0. By Claim 56, we have $\mathbf{E}[\mathbf{X}_i] \leq 2k$ for all $i \in [t]$, so $\mathbf{E}\left[\sum_{i \in [t]} \mathbf{X}_i\right] < n/200$. By Markov's inequality, we have $\sum_{i \in [t]} \mathbf{X}_i < n/20$ with probability at least $9/10$. Thus with probability at least $9/10$ over the draw of $\mathbf{D} \sim \mathcal{D}_0$ we land in a leaf vertex v with $\theta(v) < n/20$, implying that $v \in \hat{\mathcal{E}}_0$ as needed. □

Claim 58. *For any $v \in \hat{\mathcal{E}}_0$, we have $\mathbf{Pr}[v \mid c = 1] > (9/10)\,\mathbf{Pr}[v \mid c = 0]$.*

Proof. Fix any $v \in \hat{\mathcal{E}}_0$. By definition, when the algorithm concludes at v, at most $n/5$ indices were killed while having originally been light. Furthermore, since each heavy index requires by definition at least $k/2$ queries to become heavy, and since each query contributes to the heaviness of at most 2 indices, it follows that at most $kn/400(4/k) = n/100$ indices could ever have become heavy during any execution. Thus there are at least $n - n/20 - n/100 > (9/10)n$ indices i which are both alive and light at v.

Now fix any $h \in \mathcal{F}_0(v)$. We show that $(h, i) \in \mathcal{F}_1(v)$ for at least $(9/10)n$ indices $i \in [n]$, which will demonstrate that $|\mathcal{F}_1(v)| > (9/10)|\mathcal{F}_0(v)|$, and thereby complete the proof. In particular, it will suffice to show that is true for any $i \in [n]$ which is alive at v. To see why this is the case, note that by definition if i is alive at $v \in \mathcal{E}_0$, then $S(v)$ includes only observations in the i-th row and column of $\mathcal{D}$ which are equal to R. It follows that none of these observations would change if the input was instead specified by $(h, i, 1)$ instead of $(h, j, 0)$ for any $j \in [n]$, as the difference between the two resulting matrices are supported on values where $\mathcal{D}$ is equal to 1 in the i-th row and column of $\mathcal{D}$. Thus if i is alive at v, we have that $(h, i) \in \mathcal{F}_1(v)$, which completes the proof of the claim. □

Putting together the bounds from Claims 57 and 58, it follows that

$$\sum_{v \in \hat{\mathcal{E}}_0} \mathbf{Pr}[v \mid c = 1] > (9/10)^2 = .81$$

Moreover, because by Claim 55 the algorithm always outputs $c = 0$ when it ends in any $v \in \mathcal{E}$, it follows that the algorithm incorrectly determined the value of c with probability at least .81 conditioned on $c = 1$, and therefore is incorrect with probability at least $.405 > 4/10$ which is a contradiction since $\mathcal{A}$ was assumed to have success probability at least $6/10$.

From the Hard Distribution to k-Centers. To complete the proof, it suffices to demonstrate that the optimal k-center cost is at most 1 when $\mathbf{D} \sim \mathbf{D}_0$, and at least R when $\mathbf{D} \sim \mathbf{D}_1$. The first case is clear, since we can choose at least one index in the pre-image of $h^{-1}(t) \subseteq [n]$ for each $t \in [k]$ to be a center. For the second case, note that conditioned on $|h^{-1}(t)| \geq 2$ for all $t \in [k]$, the resulting metric contains $k+1$ points which are pairwise-distance at least R from each other. In particular, for the resulting metric, at least one point must map to a center which it is distance at least R away from, and therefore the cost is at least R. Now since $n = \Omega(k \log k)$ with a sufficently large constant, it follows by the coupon collector's argument that with probability at least $1/1000$, we have that $|h^{-1}(t)| \geq 2$ for all $t \in [k]$. Moreover, that the $1/1000$ probability under which does not occur can be subtracted into the failure probability of .405 in the earlier argument, which still results in a $.404 > 4/10$ failure probability, and therefore leads and leading to the same contradiction, which completes the proof. □

Proposition 59. *Fix any $1 \leq k < n$. Then any algorithm which, given oracle access the distance matrix $\mathcal{D} \in \mathbb{R}^n$ of a set X of n points in a metric space, determines correctly with probability $2/3$ whether the optimal k-center cost on X is at most 1 or at least R, for any value $R > 1$, must make at least $\Omega(k^2)$ queries in expectation to $\mathcal{D}$. The same bound holds true replacing the k-center objective with (k,z)-clustering, for any constant $z > 0$.*

Proof. By the same arguements given in Theorem 54, one can assume that the existence of such an algorithm that would violate the statement of the proposition implies the existence of a deterministic algorithm which always makes at most ck^2 queries to $\mathcal{D}$ and is correct with probability $3/5$, for some arbitrarily small constant c. In what follows, we assume $n = k+1$, and for larger n we will simply add $n-(k+1)$ duplicate points on top of the first point in the following distribution; note that any algorithm for the dataset with the duplicate point can be simulated, with no increase in query complexity, via access to the distance matrix on the first $k+1$ points.

The hard instance is as then as follows. With probability $1/2$, we give as input the distance matrix $\mathbf{D} \in \mathbb{R}^{k+1 \times k+1}$ with $\mathbf{D}_{i,j} = R$ for all $i \neq j$. Note that any k-center solution must have one of the points in a cluster centered at another, and therefore the optimal k-center cost is at least R for this instance. In the second case, the input is $\mathbf{D}$ but with a single entry $\mathbf{D}_{i,j} = \mathbf{D}_{j,i} = 1$, where (i,j) is chosen uniformly at random. Note that the result is still a valid metric space in all cases. Moreover, note that the optimal k-center cost is 1, and is obtained by choosing all points except i (or alternatively except j).

By the same (and in fact simplified) argument as in Claim 55, we can assume the algorithm returns that the k-center cost is at most 1 if and only if it sees an entry with value equal to 1. Since the algorithm is deterministic, and since the only distance other than 1 is R, we can define a deterministic set S of ck^2 indices in $\binom{k+1}{2}$ such that the adaptive algorithm would choose exactly the set S if, for every query it made, it observed the distance R (and therefore would return that the k-center cost was at most 1 at the end). Then in the second case, the probability that (i,j) is contained in S is at most $\frac{4}{c}$. Setting $c > 40$, it follows that the algorithm is incorrect with probability at least $1/2 - \frac{4}{c} > 2/5$, contradicting the claimed success probability of $3/5$, and completing the proof. □

9 Lower Bound Against Adaptive Adversaries

In this section we present our lower bounds on the approximation guarantees of dynamic algorithms for k-center, k-median, k-means, k-sum-of-radii, k-sum-of-diameters, and (k, z)-clustering against a *metric-adaptive* adversary. Recall that a point-adaptive adversary needs to fix the metric space in advance, while a metric-adaptive adversary only needs to answer a distance query consistently with all answers it gave previously (so the metric itself is chosen adaptively). First, we define a generic strategy for an adversary that in each operation creates or deletes a point and answers the distance queries of the algorithm. Then we show how to derive lower bounds for all of our problems from this single strategy. Some formal details of the analysis are deferred to Section 9.2.

9.1 Strategy of the Adaptive Adversary

Let $f(k, n)$ be a positive function that is, for every fixed k, non-decreasing in n. Suppose that there is an algorithm (for any of the problems under consideration) that in an amortized sense queries the distances between at most $f(k, n)$ pairs of points per operation of the adversary, where n denotes the number of points at the beginning of the respective operation. Note that any algorithm with an amortized update time of at most $f(k, n)$ fulfills this condition. To determine the distance between two points the algorithm asks a distance query to the adversary. We present now an adversary $\mathcal{A}$ whose goal is to maximize the approximation ratio of the algorithm. To record past answers and to give consistent answers, $\mathcal{A}$ maintains a graph $G = (V, E)$ which contains a vertex $v_p \in V$ for each point p that has been inserted previously (including points that have been deleted already). Intuitively, with the edges in E the adversary keeps track of previous answers to distance queries. Each vertex v_p is labeled as *open, closed,* or *off.* If a point p has not been deleted yet, then its vertex v_p is labeled as open or closed. Once point p is deleted, then v_p is labeled as off. Intuitively, if p has not been deleted yet, then v_p is open if it has small degree and closed if it has large degree. In the latter case, $\mathcal{A}$ will delete p soon. All edges in G have length 1, and for two vertices $v, v' \in V$ we denote by $d_G(v, v')$ their distance in G.

When choosing the next update operation, $\mathcal{A}$ checks whether there is a closed vertex v_p. If yes, $\mathcal{A}$ picks an arbitrary closed vertex v_p, deletes the corresponding point p, and labels the vertex v_p as off. Otherwise, it adds a new point p, adds a corresponding vertex v_p to G, and labels v_p as open.

Suppose now that the algorithm queries the distance $d(p, p')$ for two points p, p' while processing an operation. Note that p and/or p' might have been deleted already. If both v_p and $v_{p'}$ are open then $\mathcal{A}$ reports to the algorithm that $d(p, p') = 1$ and adds an edge $\{v_p, v_{p'}\}$ to E. Intuitively, due to the edge $\{v_p, v_{p'}\}$ the adversary remembers that it reported the distance $d(p, p') = 1$ before and ensures that in the future it will report distances consistently. Otherwise, $\mathcal{A}$ considers an augmented graph G' which consists of G and has in addition an edge $\{v_{\bar{p}}, v_{\bar{p}'}\}$ of length 1 between any pair of open vertices $\bar{p}, \bar{p}'$. The adversary computes the shortest path P between p and p' in G' and reports that $d(p, p')$ equals the length of P. If P uses an edge between two open vertices $\bar{p}, \bar{p}'$, then $\mathcal{A}$ adds the edge $\{v_{\bar{p}}, v_{\bar{p}'}\}$ to G. Note that P can contain at most one edge between two open vertices since it is a shortest path in a graph in which all pairs of open vertices have distance 1. Observe that if both v_p and $v_{p'}$ are open then this procedure reports that $d(p, p') = 1$ and adds an edge $\{v_p, v_{p'}\}$ to E which is consistent with our definition above for this case. If a vertex v_p has degree at least $100f(k, t)$ for the current operation t, then v_p is labeled as closed. A closed vertex never becomes open again.

In the next lemma we prove some properties about this strategy of $\mathcal{A}$. For each operation t, denote by $G_t = (V_t, E_t)$ the graph G at the beginning of the operation t. Recall that the value of n right before the operation t (which is the number of current points) equals the number of open and closed vertices in G_t. We show that the the number of open vertices is $\Theta(t)$, each vertex has bounded degree, and there exist arbitrarily large values t such that in G_t there are no closed vertices (i.e., only open and off vertices).

Lemma 60. *For every operation $t > 0$ the strategy of the adversary ensures the following properties for G_t*

1. *the number of open vertices in G_t is at least $92t/100$,*
2. *each vertex in G_t has a degree of at most $101f(k,n)$,*
3. *there exists an operation t' with $t < t' \leq 2t$ such that $G_{t'}$ contains only open and off vertices, but no closed vertices.*

We say that an operation $t \in \mathbb{N}$ is a *clean operation* if in G_t there are no closed vertices. For any clean operation t, denote by $\bar{G}_t = (\bar{V}_t, \bar{E}_t)$ the subgraph of G_t induced by the open vertices in V_t.

Consistent metrics. The algorithm does not necessarily know the complete metric of the given points, it knows only the distances reported by the adversary. In particular, there might be many possible metrics that are consistent with the reported distances. For each $t \in \mathbb{N}$ denote by Q_t the points that were inserted before operation t, including all points that were deleted before operation t, and let $P_t \subseteq Q_t$ denote the points in Q_t that are not deleted. Given a metric M on any point set P', for all pairs of points $p, p' \in P'$ we denote by $d_M(p,p') \geq 0$ the distance between p and p' according to M. For any $t \in \mathbb{N}$ we say that a metric M for the point set Q_t is *consistent* if for any pair of points $p, p' \in Q_t$ for which the adversary reported the distance $d(p,p')$ before operation t, it holds that $d(p,p') = d_M(p,p')$. In particular, any consistent metric might be the true underlying metric for the point set Q_t.

The key insight is that for each clean operation t, we can build a consistent metric with the following procedure. Take the graph G_t and insert an arbitrary set of edges of length 1 between pairs of open vertices (but no edges that are incident to off vertices), and let G'_t denote the resulting graph. Let M be the shortest path metric according to G'_t. If a metric M for Q_t is constructed in this way, we say that M is an *augmented graph metric for t.*

Lemma 61. *Let $t \in \mathbb{N}$ be a clean operation and let M be an augmented graph metric for t. Then M is consistent.*

In particular, there are no shortcuts via off vertices in G_t that could make the metric M inconsistent.

We fix a clean operation $t \in \mathbb{N}$. We define some metrics that are consistent with Q_t that we will use later for the lower bounds for our specific problems. The first one is the "uniform" metric M_{uni} that we obtain by adding to G_t an edge between *each* pair of open vertices in G_t. As a result, $d_{M_{\text{uni}}}(p,p') = 1$ for any $p, p' \in P_t$.

Lemma 62. *For each clean operation t the corresponding metric M_{uni} is consistent.*

In contrast to M_{uni}, our next metric ensures that there are distances of up to $\Omega(\log n)$ between some pairs of points. Let $p^* \in P_t$ be a point such that v_{p^*} is open. For each $i \in \mathbb{N}$ let $V^{(i)} \subseteq V_t$ denote the open vertices $v \in V_t$ with $d_{G_t}(v_{p^*}, v) = i$, and let $V^{(n)} \subseteq V_t$ denote the vertices in G_t that are in a different connected component than p^*. Since the vertices in G_t have degree at most $100f(k,n)$, more than half of all vertices are in sets $V^{(i)}$ with $i \geq \Omega(\log n / \log f(k,n))$.

We define now a metric $M(p^*)$ as the shortest path metric in the graph defined as follows. We start with G_t; for each $i, i' \in \mathbb{N}$ we add to G_t an edge $\{v_p, v_{p'}\}$ between any pair of vertices $v_p \in V^{(i)}$,

$v_{p'} \in V^{(i')}$ such that $|i - i'| \leq 1$. As a result, for any $i, i' \in \mathbb{N}$ and any $v_p \in V^{(i)}$, $v_{p'} \in V^{(i')}$ we have that $d_{M(p^*)}(p, p') = \max\{|i - i'|, 1\}$, i.e., $d_{M(p^*)}(p, p') = 1$ if $i = i'$ and $d_{M(p^*)}(p, p') = |i - i'|$ otherwise.

Lemma 63. *For each clean operation t and each point $p^* \in V_t$ the metric $M(p^*)$ is consistent.*

For any two thresholds $\ell_1, \ell_2 \in \mathbb{N}_0$ with $\ell_1 < \ell_2$ we define a metric $M_{\ell_1,\ell_2}(p^*)$ (which is a variation of $M(p^*)$) as the shortest path metric in the following graph. Intuitively, we group the vertices in $\bigcup_{i=0}^{\ell_1} V^{(i)}$ to one large group and similarly the vertices in $\bigcup_{i=\ell_2}^{\infty} V^{(i)}$. Formally, in addition to the edges defined for $M(p^*)$, for each pair of vertices $v_p \in V^{(i)}$, $v_{p'} \in V^{(i')}$ we add an edge $\{v_p, v_{p'}\}$ if $i \leq i' \leq \ell_1$ or $\ell_2 \leq i \leq i'$.

Lemma 64. *For each clean operation t, each point $p^* \in V_t$, and each $\ell_1, \ell_2 \in \mathbb{N}_0$ the metric $M_{\ell_1,\ell_2}(p^*)$ is consistent.*

Lower bounds. Consider a clean operation t. The algorithm cannot distinguish between M_{uni} and $M(p^*)$ for any $p^* \in P_t$. In particular, for the case that $k = 1$ (for any of our clustering problems) the algorithm selects a point p^* as the center, and then for each i it cannot determine whether the distance of the points in $V^{(i)}$ to p^* equals 1 or i. However, there are at least $n/2$ points in sets $V^{(i)}$ with $i \geq \Omega(\log n / \log f(k, n)))$ and hence they contribute a large amount to the objective function value. This yields the following lower bounds, already for the case that $k = 1$. With more effort, we can show them even for bi-criteria approximations, i.e., for algorithms that may output $O(k)$ centers, but where the approximation ratio is still calculated with respect to the optimal cost on k centers.

For k-center the situation changes if the algorithm does not need to be able to report an upper bound of the value of its computed solution (but only the solution itself), since if $k = 1$, then any point is a 2-approximation. However, for arbitrary k we can argue that there must be $3k$ consecutive sets $V^{(i)}, V^{(i+1)}, ..., V^{(i+3k-1)}$ such that the algorithm does not place any center on any point corresponding to the vertices in these sets and hence incurs a cost of at least $3k/2$. On the other hand, for the metric $M_{i,i+3k-1}(p^*)$ the optimal solution selects one center from each set $V^{(i+1)}, V^{(i+4)}, V^{(i+7)}, ...$ which yields a cost of only 1.

Theorem 3. *For any $k \geq 1$, any dynamic algorithm which returns a set of k-centers against a metric-adaptive adversary with an amortized update time of $f(k, n)$, for an arbitrary function f, must have an approximation ratio of $\Omega\left(\min\{\frac{\log(n)}{k \log f(k,2n)}, k\}\right)$ for the k-center problem. In addition, even for the case of $k = 1$, we show that* any *algorithm with an update time of $f(k, n)$*

- *for 1-median has an approximation ratio of $\Omega\left(\frac{\log(n)}{\log f(1,2n)}\right)$,*
- *for 1-means has an approximation ratio of $\Omega\left(\left(\frac{\log(n)}{\log f(1,2n)}\right)^2\right)$,*
- *for $(1, z)$-clustering has an approximation ratio of $\Omega\left(\left(\frac{\log(n)}{z+\log f(1,2n)}\right)^z\right)$,*
- *for 1-center, 1-sum-of-radii or 1-sum-of-diameters has an approximation ratio of $\Omega\left(\frac{\log(n)}{\log f(1,2n)}\right)$ if the algorithm is also able to estimate the cost of the optimal clustering.*

9.2 Deferred Proofs

We introduce some formal notation and definitions we use to revisit the adversarial strategy that generates an input stream and answers distance queries on the set of currently known points. Then, we derive lower bound constructions for the aforementioned problems that are based on the metric space defined by the stream and the answers to the algorithm.

We describe a strategy for an adversary $\mathcal{A}$ that generates a stream of update operations σ and answers distance queries q on pairs of points by any dynamic algorithm with a guarantee on its amortized complexity. In the following presentation, the adversary constructs the underlying metric space ad hoc. More precisely, the adversary constructs two metric spaces simultaneously that cannot be distinguished by the algorithm and its queries. All subsequent lower bounds stem from the fact that the problem at hand has different optimal costs on the input for the two metric spaces. When the algorithm outputs a solution, the adversary can fix a metric space that induces high cost for the centers chosen by the algorithm.

During the execution of the algorithm, the adversary maintains a graph G. Each point that was inserted by the adversary is represented by a node in G. All query answers given to the algorithm by the adversary can be derived from G using the shortest path metric $d_G(\cdot,\cdot)$ on G. We denote the algorithm's i^{th} query after update operation t by $q_{t,i}$, and the adversary's answer by $\operatorname{ans}(q_{t,i})$. For $t > 0$, we denote the number of queries asked by the algorithm between the t^{th} and the $(t+1)^{\text{th}}$ update operation by $c(t)$. If t is clear from context, we simplify notation and write $c := c(t)$. Note that, in this section, we use a slightly extended notation when indexing graphs when compared to other sections. Details follow.

We number the update operations consecutively starting with 1 using index t and after each update operation, we index the distance queries that the algorithm issues while processing the update operation and the immediately following value- or solution-queries using index i. Let $c > 0$ and let $G_{0,c(0)}$ be the empty graph. For every $t > 0, i \in [c(t)]$, consider i-th query issued by the algorithm processing the t-th update operation. The graph $G_{t,i}$ has the following structure. For every point x that is inserted in the first t operations, $V(G_{t,0})$ contains a node x. All edges have length 1, and it holds that $V(G_{t,i}) \supseteq V(G_{t,i-1})$ and $E(G_{t,i}) \supseteq E(G_{t,i-1})$.

Edges are inserted by the adversary as detailed below. Let $\preceq$ denote the predicate that corresponds to the lexicographic order. In particular, the adversary maintains the following invariant, which is parameterized by the update operation t and the corresponding query i: for all $(t', j) \preceq (t, i)$ and $(u, v) := q_{t',j}$, $\operatorname{ans}(q_{t',j}) = d_{G_{t,i}}(u, v)$. In other words, any query given by the adversary remains consistent with the shortest path metric on all versions of G after the query was answered. The adversary distinguishes the following types of nodes in $G_{t,i}$ to answer a query. Recall that $f := f(k, n)$ is an upper bound on the amortized complexity per update operation of the algorithm, which is non-decreasing in n for fixed k.

Definition 65 (type of nodes). *Let $t \geq 0, i \in [c]$ and let $u \in V(G_{t,i})$. If u has degree less than $100f(k, i)$ for all $i \in [t]$, it is* open *after update t, otherwise it is* closed*. In addition, the adversary can mark closed nodes as* off*. We denote the set of open, closed and off nodes in $G_{t,i}$ by $A_{t,i}$, $P_{t,i}$ and $D_{t,i}$, respectively.*

For operation t, the adversary answers the i^{th} query $q_{t,i}$ according to the shortest path metric on $G_{t,i-1}$ with the additional edge set $A_{t,i-1} \times A_{t,i-1}$. In other words, the adversary (virtually) adds edges between all open nodes in $G_{t,i-1}$ and reports the length of a shortest path between the query points in the resulting graph. After the adversary answered query $q_{t,i}$, the resulting graph $G_{t,i}$ is $G_{t,i-1}$ plus the (unique) edge e of the shortest path between two open nodes that is not in $G_{t,i-1}$ if such edge exists. If the connected component of e in the resulting graph does not contain any open node, we also add an edge between the connected component and a node with degree at most $50f(k, i)$. Thus, the algorithm maintains the invariant that each connected component has at least one open vertex (see Lemma 66 for details). A key element of our analysis is that all answers up to operation t and query i are equal to the length of the shortest paths between the corresponding query points in $G_{t,i}$. The generation of the input stream and the answers to all queries are formally given by Algorithm 7 and 8, respectively.

Algorithm 7: Construction of element σ_t of σ

```
1  Function GenerateStream($t$)
2      if there exists a closed node $x \in V(G_{t-1,c})$ then
3          mark $x$ as off in $G_{t,0}$;
4          return ⟨ delete $x$ ⟩
5      else
6          let $x$ be a new point, i.e., that was not returned by the adversary before;
7          return ⟨ insert $x$ ⟩
```

Algorithm 8: Answer of the adversary to query $q_{t,i}$

```
1  Function AnswerQuery($q_{t,i} = (x, y)$)
2      let $H_{t,i} = (V(G_{t,i-1}), E(G_{t,i-1}) \cup (A_{t,i-1} \times A_{t,i-1}))$;
3      let $p = (e_1, \dots, e_k)$ be a shortest path between $x$ and $y$ in $H_{t,i}$;
4      set $G_{t,i} := G_{t,i-1}$;
5      if $\exists e_i \notin E(G_{t,i-1})$ then
6          insert $e_i$ into $G_{t,i}$ ;
7          let $C$ be the connected component of $e_i$ in $G_{t,i}$ ;
8          if $C \cap A_{t,i} = \emptyset$ then                  // make sure $C$ contains an open node
9              let $u = \arg\min_{u' \in C} \deg(u')$ ;          // pick node with degree $100f(k,t)$
10             let $v = \arg\min_{v' \in A_{t,i}} \deg(v')$ ;    // pick node with degree at most $50f(k,t)$
11             insert $(u, v)$ into $G_{t,i}$ ;
12     return length of $p$
```

9.2.1 Adversarial Strategy

Let n_t be the number of open and closed nodes, i.e., the number of current points for the algorithm, after operation t. In the whole section we use the notations of $G_{t',i}$, $A_{t,i}$ etc. from Definition 65.

Proof of Lemma 60 The next three lemmas prove the three claims in Lemma 60.

Lemma 66 (Lemma 60 (1))**.** *For every $t > 0$, the number of open nodes in $G_{t,0}$ is at least $92t/100$, and for every $t > 0, j \geq 0$, there exists at least one node with degree at most $50f(k,t)$ in $G_{t,j}$.*

Proof. Recall that $f(k,n)$ is a positive function that is non-decreasing in n. We prove the first part of the claim by induction. By the properties of f, the case $t = 1$ follows trivially. Let $t \geq 2$. For any $i \in [t]$, the algorithm's query budget increases by $f(k, n_i)$ queries after the i^{th} update. Since $f(k,i)$ is non-decreasing, nodes inserted after update operation $i-1$ can only become closed if their degrees increase to at least $100f(k,i)$. Answering a query i inserts at most two edges into $G_{t,i-1}$, and the sum of degrees increases by at most four. Therefore, it holds that $|P_{t,0}| \leq \sum_{i\in[t]} 4f(k,n_i)/(100f(k,i)) \leq \sum_{i\in[t]} 4f(k,i)/(100f(k,i)) \leq 4t/100$. It follows that the adversary will delete at most $4t/100$ points in the first t operations and insert points in the other at least $(1-4/100)t$ operations. The number of open nodes after update t is $|A_{t,0}| \geq t - |P_{t,0}| - 4t/100 \geq 92t/100$.

To prove the second part of the claim, let $s_{t,j}$ be the number of nodes in $G_{t,j}$ with degree at most $50f(k,t)$. Similarly as before, we have $n_t - s_{t,j} \le \sum_{i\in[t]} 4f(k,n_i)/(50f(k,i)) \le \sum_{i\in[t]} 4f(k,i)/(50f(k,i)) \le 4t/50$. Therefore, $s_{t,j} \ge n_t - 4t/50 \ge (t - 4t/100) - 4t/50 \ge 1$. □

Lemma 67 (Lemma 60 (2))**.** *For every $t > 0, i \in [c]$, all nodes have degree at most $100f(k,t) + 1$ in $G_{t,i}$.*

Proof. By definition, the claim is true for open nodes. Edges are only inserted into G if the algorithm queries for the distance between two nodes x, y and the adversary determines a shortest path between x and y that contains edges that are not present in $G_{t,i-1}$ (see Algorithm 8). Since all such edges are edges between open nodes, only degrees of open nodes in $G_{t,i}$ increase. The adversary finds a shortest path on a supergraph of $A_{t,i} \times A_{t,i}$. Therefore, any shortest path it finds contains at most one edge with two open endpoints. It follows that a query increases the degree of any open node in $G_{t,i}$ by at most one, which may turn it into a closed node with degree $100f(k,t)$. If the connected component of this edge contains no open node, it also inserts an edge from an open node (with degree at most $50f(k,t)$) to the vertex with smallest degree in the component. Since the vertex of the component that became closed most recently always has degree $100f(k,t)$, this increases the degree of every closed node at most once by 1. □

Lemma 68 (Lemma 60 (3))**.** *For every $t > 0$, there exists a clean update operation t', $t < t' \le 2t$, i.e., $G_{t',0}$ only contains open and off nodes, but no closed nodes.*

Proof. We prove the claim by induction over the operations t with the properties that $G_{t,0}$ contains no closed node, but $G_{t+1,0}$ contains at least one closed node. The claim is true for the initial (empty) graph $G_{0,0}$. Let $t > 0$. We prove that in at least one operation $t' \in \{t+1, \dots, 2t\}$, the number of closed nodes is 0. For the sake of contradiction, assume that for all t', $t < t' \le 2t$, the number of closed nodes is non-zero, i.e., $|P_{t',0}| > 0$. We call an open node *semi-open* if it has degree greater than $50f(k,i)$ after some update $i \in [2t]$. Otherwise, we call it *fully-open*. Recall that, similarly, a vertex becomes closed if it has degree at least $100f(k,i)$ after some update $i \in [2t]$ (and never becomes open again).

For any $i \in [2t]$, the algorithm's query budget increases by $f(k,n_i)$ queries after the i^{th} update. Since $f(k,i)$ is non-decreasing, nodes inserted after update operation $i-1$ can only become semi-open if their degrees increase to at least $50f(k,i)$ (resp. $100f(k,i)$) by the definition of semi-open (resp. closed). Also due to the monotonicity of f, the number of semi-open or closed nodes is maximized if the algorithm invests its query budget as soon as possible. It follows that the number of semi-open or closed nodes up to operation $2t$ is at most $\sum_{j\in[2t]} 4f(k,j)/(50f(k,j)) \le 4t/50$. Without loss of generality, we may assume that all semi-open nodes are closed (so the algorithm does not need to invest budget to make them closed).

After update operation $2t$, the algorithm's total query budget from all update operations is at most $\sum_{i\in[2t]} f(k,n_i) \le \sum_{i\in[2t]} f(k,i) \le 2tf(k,2t)$. Answering a query i inserts at most two edges into $G_{t,i-1}$, and the sum of degrees increases by at most four. The algorithm may use its budget to increase the degree of at most $2t \cdot 4f(k,2t)/(50f(k,2t)) \le 8t/50$ fully-open nodes to at least $100f(k,2t)$, i.e., to make them closed nodes. Recall our assumption that $|P_{i,0}| > 0$ for all $i \in \{t+1, \dots, 2t\}$. The adversary deletes one point corresponding to a closed node in each update operation from $\{t+1, \dots, 2t\}$. Therefore, the number of closed nodes after update operation $2t$ is

$$|P_{2t,0}| \le \frac{4t}{50} + \frac{8t}{50} - t \le \frac{12t}{50} - t < 0.$$

This is a contradiction to the assumption. □

Proofs of Lemmas 61 to 64 The following observation follows immediately from the properties of shortest path metrics.

Observation 69. *Let $G = (V, E)$ and $G' = (V, E')$ be two graphs so that $E \subseteq E'$. If a sequence of queries is consistent with the shortest path metric on G as well as on G', then, for any E'', $E \subseteq E'' \subseteq E'$, it is also consistent with the shortest path metric on (V, E'').*

The following lemma together with Observation 69 implies Lemmas 61 to 64 and 71 by setting $G = G_{t,i-1}$ and $G' = (V(G_{t,i-1}), E(G_{t,i-1}) \cup (A_{t,i-1} \times A_{t,i-1}))$ in Observation 69.

Lemma 70. *For any $t, t' > 0$, $i, i' \in [c]$ so that $(t', i') \prec (t, i)$, the answer given to query $q_{t',i'}$ is consistent with the shortest path metric on G'.*

Proof. Let $t' > 0$, $i' \in [c]$ so that $(t', i') \prec (t, i)$ and denote $(x, y) := q := q_{t',i'}$. We prove that $\text{ans}(q) = d_{G_{t,i}}(x, y)$. As neither vertices nor edges are deleted after they have been inserted, for every $(t_1, i_1) \prec (t_2, i_2)$, G_{t_2,i_2} is a supergraph of G_{t_1,i_1}. Thus, if there exists a path P between x and y in $G_{t',i'}$, a shortest path in $G_{t,i}$ between x and y cannot be longer than P.

It remains to prove $\text{ans}(q) \le d_{G_{t,i}}(x, y)$. For the sake of contradiction, assume that there exist t'', i'' so that $\text{ans}(q) = d_{G_{t'',i''-1}}(x, y)$, but $\text{ans}(q) > d_{G_{t'',i''}}(x, y)$. By Definition 65, closed nodes never become open. For any closed node $v \in P_{t'',i''-1}$, it follows that $d_{G_{t'',i''}}(v, A_{t,i}) \ge d_{G_{t'',i''-1}}(v, A_{t,i-1})$ as Algorithm 8 only inserts edges between vertices in $A_{t,i}$ into $G_{t,i}$. Therefore, any shortest path between x and y in the graph $(V(G_{t'',i''-1}), E(G_{t'',i''-1} \cup (A_{t'',i''-1} \times A_{t'',i''-1}))$ has length at least $d_{G_{t'',i''-1}}(x, y) = \text{ans}(q)$. □

9.2.2 Lower Bounds for Clustering

Our lower bounds apply for the case that the algorithm is allowed to choose as centers any points that have ever been inserted as well as to the case where centers must belong to the set of current points. In the whole section we use the notations of $G_{t',i}, A_{t,i}$ etc. from the introduction of Section 9.2.

Proof of Theorem 3 For any $\ell \in \mathbb{N}_0$, we define a metric $M_\ell(P^*)$ on a subset of points P^* as the shortest path metric on the following graph. For each pair of open vertices u, v we add an edge if $d(u, P^*) \ge \ell$ and $d(v, P^*) \ge \ell$.

Lemma 71. *For each clean update operation t, each subset of points $P^* \in V_t$, and each $\ell \in \mathbb{N}_0$ the metric $M_\ell(P^*)$ is consistent.*

Proof. The metric $M_\ell(P^*)$ is an augmented graph metric for t and, thus, for a clean update operation t, it is consistent by Lemma 61. □

Lemma 72 (Theorem 3, part 1)**.** *Let $k \ge 2$. Consider any dynamic algorithm for maintaining an approximate k-center solution of a dynamic point set that (1) queries amortized $f(k, n)$ distances per operation, where n is the number of current points, and (2) outputs at most $g(k) \in O(k)$ centers. For any $t \ge 2$ such that t is a clean operation, the approximation factor of the algorithm's solution (with respect to an optimal k-center solution) against an adaptive adversary right after operation t is at least $\Omega\left(\min\left\{k, \frac{\log n}{k \log f(k,2n)}\right\}\right)$.*

Proof. Denote $G := (V, E) := G_{t,0}$ and $A := A_{t,0}$. By Lemma 66, the number of open nodes in G is at least $92t/100 \ge t$, which implies that $|A| \ge 92t/100$. Thus, after operation t, the number n of current points is at least $92t/100$.

Without loss of generality, we assume that $G[A]$ has exactly one connected component: If this is not the case, let $C_1, \dots, C_s$ be the connected components of $G[A]$ and observe that we may insert a path connecting the connected components by inserting into $G[A]$ edges $(v_1, v_2), \dots, (v_{s-1}, v_s)$ of length 1, where $v_i \in C_i$ are arbitrary vertices. This increases the maximum degree of nodes in G by at most 2 and the shortest-path metric M on the resulting graph that is constructed in this way is an augmented graph metric for t. As t is clean, Lemma 61 shows that M is consistent.

Let $x \in V$ be any node. By Lemma 67, the number $n^{(i)}$ of nodes that have distance at most i to x is at most $\sum_{j\in[i]}(100f(1,t)+3)^j \le (100f(k,t)+3)^{i+1}$. Consider the largest ℓ such that $(100f(k,t)+3)^{\ell+1} < n$. It follows that there exists a node z at distance $\ell+1$ to x. Furthermore $(100f(k,t)+3)^{\ell+3} \ge n^{(\ell+1)} \ge n$, which implies that $\ell+2 \ge \log_{100f(k,t)+3} n$.

Let $S = \{s_1,\dots,s_{g(k)}\}$ be the solution of the algorithm. For $i \in [\ell+1]$, let us define $V^{(i)}$ to be the set of vertices v in $G[A]$ with $d_{G[A]}(x,v) = i$. By pigeon hole principle, there must exist a consecutive sequence $(i_1,\dots,i_m)$ so that $m \ge \ell/(g(k)+1)$ and, for all $i \in \{i_1,\dots,i_m\}$, $S \cap V^{(i)} = \emptyset$. Let $k' = \min\{3k-1, m/2\}$. Consider the metric $M_{i_1,i_{k'}}(x)$ and let $y_{i_1},\dots,y_{k'}$ be elements from the respective sets $V^{(i_1)},\dots,V^{(i_{k'})}$.

The algorithm's solution S has cost at least k' because $d_G(S, V_{i_{k'}}) \ge k'$. The solution $\{y_{3j-2} \mid j \in \mathbb{N} \wedge 3j-2 \in [k']\}$ is optimal and has cost 1. It follows that the approximation factor of S is greater than or equal to $k' = \min\{3k-1, m/2\}$.

Let n be the number of points at iteration t. We calculate that

$$\begin{aligned}
\min\{3k/2, m/2\} &\ge \Omega(\min\{k,m\}) \\
&\ge \Omega\left(\min\left\{k, \frac{\ell}{g(k)+1}\right\}\right) \\
&\ge \Omega\left(\min\left\{k, \frac{1}{g(k)}\left(\frac{\log n}{\log(103f(k,t))} - 1\right)\right\}\right) \\
&\ge \Omega\left(\min\left\{k, \frac{\log n}{k\log(103f(k,2n))}\right\}\right) \\
&\ge \Omega\left(\min\left\{k, \frac{\log n}{k\log 103 + k\log f(k,2n)}\right\}\right) \\
&\ge \Omega\left(\min\left\{k, \frac{\log n}{k\log f(k,2n)}\right\}\right)
\end{aligned}$$

using that $g(k) = O(k)$. □

Lemma 73 (Theorem 3, part 2)**.** *Consider any dynamic algorithm for computing the diameter of a dynamic point set that queries amortized $f(1,n)$ distances per operation, where n is the number of current points, and outputs at most $g \ge 1$ centers. For any $t \ge 2$ such that t is a clean operation, the approximation factor of the algorithm's solution (with respect to the correct diameter) against an adaptive adversary right after operation t is at least $\log(92t/100)/(\log 103f(1,t)) - 1$ and $92t/100 \le n \le t$.*

Proof. Denote $G := (V,E) := G_{t,0}$ and $A := A_{t,0}$. By Lemma 66, the number of open nodes in G is at least $92t/100 \ge t$, which implies that $|A| \ge 92t/100$. Thus, after operation t, $n \ge 92t/100$.

Without loss of generality, we assume that $G[A]$ has exactly one connected component: If this is not the case, let $C_1,\dots,C_s$ be the connected components of $G[A]$ and observe that we may insert a path connecting the connected components by inserting into $G[A]$ edges $(v_1,v_2),\dots,(v_{s-1},v_s)$ of length 1, where $v_i \in C_i$ are arbitrary vertices. This increases the maximum degree of nodes in G by at most 2 and the shortest-path metric M on the resulting graph that is constructed in this way is an augmented graph metric for t. As t is clean, Lemma 61 shows that M is consistent.

Let $x \in V$ be any node. By Lemma 67, the number $n^{(i)}$ of nodes that have distance i to x is at most $\sum_{j\in[i]}(100f(1,t)+3)^j \le (100f(1,t)+3)^{i+1}$. Consider the largest i such that $(100f(1,t)+$

$3)^{i+1} < n$. It follows that there exists a node at distance $i+1$ to x. Furthermore $(100f(1,t)+3)^{i+2} \geq n^{(i+1)} \geq n$, which implies that $i+2 \geq \log_{100f(1,t)+3} n$. As $f(1,t) \geq 1$ for all values of t, there exists a shortest path P starting at x of length at least $i+1 \geq \log_{100f(1,t)+3} n - 1 \geq (\log n / \log(103f(1,t))) - 1 =: \ell$. It follows that the diameter is at least ℓ. On the other hand, M can be extended by adding an edge between any pair of open nodes, resulting in the consistent metric M_{uni}. For this metric the diameter of G is 1. As the algorithm cannot tell whether ℓ or 1 is the correct answer, and it always has to output a value that is as least as large as the correct answer, it will output at least ℓ. Thus, the approximation ratio is at least $\ell \geq \log(92t/100)/\log 103f(1,t) - 1$. Note that this implies a lower bound for the approximation ratio for 1-sum-of-radii, and 1-sum-of-diameter. □

Lemma 74. *Consider any dynamic algorithm for $(1,p)$-clustering that queries amortized $f(1,n)$ distances per operation, where n is the number of current points, and outputs at most $1 \leq g \leq n$ centers. For any $t \geq 1$ such that t is a clean operation, the approximation factor of the algorithm's solution (with respect to the optimal $(1,p)$-clustering cost) against an adaptive adversary right after operation t is at least $\left[\frac{\log(t/4g)}{p+\log(101f(1,t))}\right]^p/4$ and $92t/100 \leq n \leq t$.*

Proof. Denote $G := (V,E) := G_{t,0}$ and $A := A_{t,0}$. By Lemma 66, the number of open nodes in G is at least $92t/100 \geq t$, which implies that $|A| \geq 92t/100$. Thus, after operation t, $n \geq 92t/100$.

Let C be the centers that are picked by the algorithm after operation t. By the assumption of the lemma, $|C| \leq g$. Consider $M_\ell(C)$, where $\ell := \log(t/4g)/(p + \log(101f(1,t)))$. By Lemma 67, for any $s \in C$, the size of $|\{x \mid x \in A \wedge d(x,s) < \ell\}|$ is at most $\sum_{i\in[\ell-1]}(101f(1,t))^i < (101f(1,t))^\ell$. Let $V_+ := \{x \mid x \in A \wedge d(x,C) \geq \ell\}$. Since $|C| \leq g$, it holds that $|V_+| > |A| - g(101f(1,t))^\ell \geq 92t/100 - g(t/4g)^{\log((101f(1,t))/(p+\log(101f(t,1)))} \geq 92t/100 - t/4 \geq t/2$. Since $d(s,V_+) \geq \ell$ for any $s \in C$, the $(1,p)$-clustering cost of S, and thus the cost of the algorithm, is at least $|V_+| \cdot \ell^p \geq t/2 \cdot \ell^p$.

We will show that if instead a single point corresponding to a vertex of V_+ is picked as center, then the cost is at most $2t$, which provides an upper bound on the cost of the optimum solution. It follows that the approximation factor achieved by the algorithm is at least $\ell^p/4$.

To complete the proof consider a point x whose corresponding point v_x belongs to V_+. One can easily show that for each $\alpha \in \mathbb{N}$ it holds that $\alpha^p \leq (101f(1,t)\cdot 2^p)^{2\alpha/3}$ since for each $\alpha \in \mathbb{N}$ it holds that $\alpha^p = 2^{p\log\alpha} \leq 2^{p\cdot\frac{2\alpha}{3}}$. Thus, the $(1,p)$-clustering cost if a single point x is chosen as center is at most

$$\begin{aligned}
\sum_{i=0}^{\ell-1} g(101f(1,t))^i \cdot (\ell-i)^p + t \cdot 1^p &\leq g\left((101f(1,t))^\ell \cdot \sum_{i=1}^{\ell} \frac{1}{(101f(1,t))^i} \cdot i^p\right) + t \\
&\leq g\left((101f(1,t))^\ell \cdot \sum_{i=1}^{\ell} \frac{(101f(1,t)\cdot 2^p)^{2i/3}}{(101f(1,t))^i}\right) + t \\
&\leq g\left((101f(1,t))^\ell \cdot \sum_{i=1}^{\ell} \frac{2^{p\cdot 2i/3}}{(101f(1,t))^{i/3}}\right) + t \\
&\leq g\left((101f(1,t))^\ell \cdot \sum_{i=1}^{\ell} \left(\frac{2^{\frac{2p}{3}}}{(101f(1,t))^{1/3}}\right)^i\right) + t \\
&\leq g(101f(1,t))^\ell 2^{p\ell} + t \\
&\leq g(101f(1,t)\cdot 2^p)^\ell + t \\
&\leq 5t/4 \leq 2t
\end{aligned}$$

using that $\ell = \frac{\log(t/4g)}{p+\log(101f(1,t))} = \frac{\log(t/4g)}{\log(2^p 101 f(1,t))} = \log_{2^p 101 f(1,t)} t/4g$. This yields an approximation ratio of at least $\frac{t/2\cdot\ell^p}{2t} = \Omega(\ell^p)$. □

Lemma 75 (Theorem 3, part 3)**.** *For any $k \geq 1$, any dynamic algorithm which returns a set of k-centers against an adaptive adversary with an amortized update time of $f(k,n)$, for an arbitrary function f, must have an approximation ratio of $\Omega\left(\left(\frac{\log(n)}{z+\log f(1,2n)}\right)^z\right)$ for the $(1,z)$-clustering.*

Proof. Let $t \in \mathbb{N}$. By Lemma 60, there is a value t' with $t < t' \leq 2t'$ such that t' is a clean operation. Let n be the number of open points at iteration t'. By Lemma 66 we know that $t' \geq n \geq 92t'/100$. Note that hence $t' \leq 2n$.

Recall that we assumed that the function $f(k,n)$ is non-decreasing in n (for any fixed k). Suppose that after operation t' we query the solution value of an algorithm for 1-center, 1-sum-of-radii, or 1-sum-of-diameter. By Lemma 73 its approximation ratio is at least $\frac{\log(92t'/100)}{\log(103f(k,t'))} - 1 \geq \frac{\log(92n/100)}{\log(103f(k,2n))} - 1 = \Omega\left(\frac{\log(n)}{\log(f(k,2n))}\right)$. Suppose that instead we query the solution from the algorithm for $(1,p)$-clustering. By Lemma 74, its approximation ratio is at least

$$\begin{aligned}
\left[\frac{\log(t'/4g)}{p+\log(101f(1,t'))}\right]^p/4 &\geq \left[\frac{\log(n/4g)}{p+\log(101f(1,2n))}\right]^p/4\\
&\geq \left[\frac{\log(n)-\log(4g)}{p+\log(101f(1,2n))}\right]^p/4\\
&\geq \left[\frac{\log(n)}{1.1p+1.1\log(101f(1,2n))}\right]^p/4\\
&\geq \left[\frac{\log(n)}{1.1p+1.1(7+\log(f(1,2n))}\right]^p/4\\
&\geq \left[\frac{\log(n)}{1.1p+8+1.1\log(f(1,2n))}\right]^p/4\\
&\geq \left[\frac{\log(n)}{2p+8+2\log(f(1,2n))}\right]^p/4\\
&= \Omega\left(\left(\frac{\log n}{2p+8+2\log f(1,2n)}\right)^p\right)
\end{aligned}$$

using that $g = O(1)$. Hence, for k-median and k-means if we take $p = 1$ and $p = 2$, respectively, this yields bounds of $\Omega\left(\frac{\log n}{10+2\log f(1,2n)}\right)$ and $\Omega\left(\left(\frac{\log n}{12+2\log f(1,2n)}\right)^2\right)$, respectively. □

References

[ACSS21] Idan Attias, Edith Cohen, Moshe Shechner, and Uri Stemmer. A framework for adversarial streaming via differential privacy and difference estimators. *CoRR*, abs/2107.14527, 2021.

[AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*, pages 459–468. IEEE, 2006.

[AOSS19] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *Proceedings of the*

Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 1919–1936. SIAM, 2019.

[AS16] Sara Ahmadian and Chaitanya Swamy. Approximation algorithms for clustering problems with lower bounds and outliers. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[AV06] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[BDH$^+$19] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 382–405. IEEE, 2019.

[BE97] Marshall Bern and David Eppstein. Approximation algorithms for geometric problems. *Approximation algorithms for NP-hard problems*, pages 296–345, 1997.

[BEJWY20] Omri Ben-Eliezer, Rajesh Jayaram, David P Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI symposium on principles of database systems*, pages 63–80, 2020.

[BFL$^+$17] Vladimir Braverman, Gereon Frahling, Harry Lang, Christian Sohler, and Lin F. Yang. Clustering high dimensional dynamic data streams. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 576–585. PMLR, 2017.

[Bro97] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

[CAHP$^+$19] Vincent Cohen-Addad, Niklas Oskar D Hjuler, Nikos Parotsidis, David Saulpic, and Chris Schwiegelshohn. Fully dynamic consistent facility location. In *Advances in Neural Information Processing Systems*, pages 3255–3265, 2019.

[CASS16] Vincent Cohen-Addad, Chris Schwiegelshohn, and Christian Sohler. Diameter and k-center in sliding windows. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[CCFM04] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.

[CGS18] T-H. Hubert Chan, Arnaud Guerqin, and Mauro Sozio. Fully Dynamic k-Center Clustering. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 579–587, Republic and Canton of Geneva, CHE, April 2018. International World Wide Web Conferences Steering Committee.

[CHHK16] Keren Censor-Hillel, Elad Haramaty, and Zohar Karnin. Optimal dynamic distributed mis. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 217–226, 2016.

[CN12] Adam Coates and Andrew Y Ng. Learning feature representations with k-means. In *Neural networks: Tricks of the trade*, pages 561–580. Springer, 2012.

[CN20] Yeshwanth Cherapanamjeri and Jelani Nelson. On adaptive distance estimation. *Advances in Neural Information Processing Systems*, 33, 2020.

[CP04] Moses Charikar and Rina Panigrahy. Clustering to minimize the sum of cluster diameters. *J. Comput. Syst. Sci.*, 68(2):417–441, 2004.

[CZ19] Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 370–381. IEEE, 2019.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[DZ18] Yuhao Du and Hengjie Zhang. Improved algorithms for fully dynamic maximal independent set. *arXiv preprint arXiv:1804.08908*, 2018.

[FG88] Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444, 1988.

[FLNFS21] Hendrik Fichtenberger, Silvio Lattanzi, Ashkan Norouzi-Fard, and Ola Svensson. Consistent k-clustering for general metrics. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2660–2678. SIAM, 2021.

[For10] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[FS05] Gereon Frahling and Christian Sohler. Coresets in dynamic geometric data streams. In Harold N. Gabow and Ronald Fagin, editors, *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 209–217. ACM, 2005.

[FSS13] Dan Feldman, Melanie Schmidt, and Christian Sohler. Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1434–1453. SIAM, 2013.

[GGR98] Oded Goldreich, Shari Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *Journal of the ACM (JACM)*, 45(4):653–750, 1998.

[GHL18] Gramoz Goranci, Monika Henzinger, and Dariusz Leniowski. A tree structure for dynamic facility location. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018,* *Helsinki, Finland*, volume 112 of *LIPIcs*, pages 39:1–39:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[GHL$^+$21] Gramoz Goranci, Monika Henzinger, Dariusz Leniowski, Christian Schulz, and Alexander Svozil. Fully dynamic k-center clustering in low dimensional metrics. In *2021 Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 143–153. SIAM, 2021.

[GK18] Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set and other problems. *arXiv preprint arXiv:1804.01823*, 2018.

[Gol17] Oded Goldreich. *Introduction to property testing.* Cambridge University Press, 2017.

[Gon85] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical computer science*, 38:293–306, 1985.

[HJ97] Pierre Hansen and Brigitte Jaumard. Cluster analysis and mathematical programming. *Mathematical programming*, 79(1):191–215, 1997.

[HK20] Monika Henzinger and Sagar Kale. Fully dynamic coresets. In *28th Annual European Symposium on Algorithms, ESA 2020, September 6-11, 2020, Pisa, Italy*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[HKM$^+$20] Avinatan Hassidim, Haim Kaplan, Yishay Mansour, Yossi Matias, and Uri Stemmer. Adversarially robust streaming algorithms via differential privacy. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020.

[HLM20] Monika Henzinger, Dariusz Leniowski, and Claire Mathieu. Dynamic clustering to minimize the sum of radii. *Algorithmica*, 82:3183–3194, 2020.

[HN79] Wen-Lian Hsu and George L Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, 1979.

[HPIM12] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.

[HS85] Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985.

[HS86] Dorit S Hochbaum and David B Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM (JACM)*, 33(3):533–550, 1986.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[LV17] Silvio Lattanzi and Sergei Vassilvitskii. Consistent k-clustering. In *International Conference on Machine Learning*, pages 1975–1984. PMLR, 2017.

[OSSW18] Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. Fully dynamic mis in uniformly sparse graphs. *arXiv preprint arXiv:1808.10316*, 2018.

[Sch07] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[SS19] Melanie Schmidt and Christian Sohler. Fully dynamic hierarchical diameter k-clustering and k-center. *arXiv preprint arXiv:1908.02645*, 2019.

[SW18] Christian Sohler and David P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 802–813. IEEE Computer Society, 2018.

[TSK13] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Data mining cluster analysis: basic concepts and algorithms. *Introduction to data mining*, pages 487–533, 2013.

[WZ22] David P. Woodruff and Samson Zhou. Tight bounds for adversarially robust streams and sliding windows via difference estimators. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1183–1196, 2022.