Balancing Distance and Choice: (k, α)-Center Clustering

Eliot W. Robson
Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA

Sariel Har-Peled
Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA

Abstract

In this work, we examine a variant of k-center that allows for fault tolerance based on an input parameter α. Here, each client cares about its distance to the αth closest center. This is slightly different than similar variants that have been presented in prior work, and this necessitates different machinery. In particular, we show an elegant 2-approximation algorithm for this problem, and how this algorithm can be sped up to run in linear time. Finally, we present an algorithm computing a greedy α-permutation that provides this approximation for all k, and this algorithm runs in $O(n \log n)$ time (if α and the doubling dimension are constant). We thus establish that (k, α)-center clustering and the "regular" k-center clustering have the same approximation complexity.

Keywords and phrases Points, clustering, fault tolerance

Digital Object Identifier 10.57717/cgt.v5i1.95

1. Introduction

Clustering is a natural and well studied problem in computer science. One of the classic variants is k-center clustering [5, 7] – here, one is given a set $P \subset \mathcal{X}$ of n points in a metric space $\mathcal{X}$, along with a parameter k. The goal is to find a set $C \subseteq P$ of size k minimizing the maximum distance of any point of P to the closest center in C. This problem is known to be NP-Hard to approximate within a factor of 1.86 [4] in the plane, and a better factor than 2 in general settings. However, somewhat surprisingly, there are simple 2-approximation algorithms [5, 11].

Doubling metrics. For a metric space, its ***doubling constant*** is the smallest constant $\mathfrak{c}$ such that any ball can be covered by $\mathfrak{c}$ balls of half its radius. The quantity $\xi = \log \mathfrak{c}$ is the *doubling dimension* of the metric space – it is the analog of the dimension d in $\mathbb{R}^d$. Indeed, any ball in $\mathbb{R}^d$ can be covered by $2^{O(d)}$ balls of half the radius. Metric spaces with constant doubling dimension/constant are abstract metric spaces that intuitively behave like $\mathbb{R}^d$ for constant d.

Greedy permutation. Consider the permutation $P = \{p_1, \ldots, p_n\}$ selected by choosing $P_1 = \{p_1\}$ arbitrarily, and then repeatedly selecting the point furthest away from all the points of P_i, and adding it to the set to form P_{i+1}. This is the *greedy permutation*, and the first k elements of this permutation are a 2-approximation to the k-center clustering problem. This was first shown by Gonzalez [5] along with an $O(nk)$ algorithm for computing it. This algorithm was later improved to $O(\mathfrak{c}^{O(1)} n \log n)$ for metric spaces with doubling constant $\mathfrak{c} = 2^{O(\xi)}$ [8].

Generalization of greedy permutation. More recently, a generalization of the greedy permutation was studied. Buchet *et al.* [1] introduced the *α-distance permutation*, where the

© Eliot W. Robson and Sariel Har-Peled
licensed under Creative Commons License CC-BY 4.0

furthest distance is with respect to the αth closest point of P_i rather than just the closest one – that is, the distance of a point p from P_i is the distance p to its αth closest point in P_i. The intuition is that this formulation is less sensitive to noise, as it smooths away the distance function induced by P_i. They also gave an $O(n\alpha \log n \log \Phi)$ time algorithm for computing the greedy permutation in this situation, where Φ is the *spread* of the input point set (the maximum distance of any pair of points divided by the minimum distance).

Approximating k-center directly. In low-dimensional Euclidean space, Feder and Greene [4] showed how to compute a 2-approximation for the k-center problem in $O(n \log k)$ time. This was later improved to $O(n)$ time (assuming randomization and the floor function) by Har-Peled [6]. Note this does not directly solve the greedy permutation problem and the techniques used are not directly applicable.

Fault tolerant k-center. Algorithms for many problems in computing produce solutions which are not robust to failure, and so it is natural to define and solve variants which include robustness guarantees. For example, there has been a line of work on graph spanners which are resilient to failures [2, 9]. One such variant of clustering tries to provide reliability guarantees in the case where some centers fail, called the *α-neighbor k-center problem*, first introduced by Krumke [14] in the context of graphs. Intuitively, we allow up to $\alpha - 1$ centers to fail, and thus the distance to the αth closest center is the "guaranteed" service quality provided by the center set.

▶ **Definition 1.** *Given a set P of points, and a (multi)set of centers C, for a point $p \in P$, let $\mathsf{d}^\alpha(p, C)$ be the distance of p to the αth closest point in C to p. The* quality *of the clustering of P provided by C is the maximum such distance; that is,* $\text{radius}^\alpha(P, C) = \max_{p \in P} \mathsf{d}^\alpha(p, C)$.

Here, we are interested in the following specific variant.

▶ **Definition 2.** *The* **(k, α)-center clustering** *problem is the task of computing a multiset $C \subseteq P$ of centers such that the quantity* $\text{radius}^\alpha(P, C)$ *is minimized and $|C| = k$. Each point from P may be selected as a center multiple times. We refer to this variant as the* **P variant**.

The task at hand is to compute the set C that results in the clustering of the best quality (i.e., minimize $\text{radius}^\alpha(P, C)$) under the constraint that $|C| = k$. If $\alpha = 1$ this recovers the original k-center problem.

▶ **Definition 3.** *Alternatively, one can consider the problem from Definition 2, where instead we wish to minimize the quantity* $\text{radius}^\alpha(P \setminus C, C)$. *In other words, the points selected as centers do not count towards the clustering quality. We refer to this variant as the* **$P \setminus C$ variant**.

A 4-approximation algorithm was provided by Krumke [14], for the $P \setminus C$ variant and for $\alpha \geq 2$, by extending a framework of Hochbaum and Shmoys [12]. This was later improved by Chaudhuri *et al.* to a 2-approximation using similar graph-theoretic techniques [3]. This problem was also analyzed by Khuller *et al.* [13], who also gave a 2-approximation algorithm.

Khuller *et al.* [13] worked on the P variant, and gave a 2-approximation, for $\alpha < 4$, and 3-approximate for $\alpha \geq 4$. More recently, Kumar and Raichel [15] showed that any c-approximate algorithm for "regular" k-center, with some post-processing applied, can be adapted to obtain a $(1 + 2c)$-factor approximation for the P variant. In addition, they showed that this procedure applied with the 2-factor approximation algorithm of Gonzalez [5] gives a better approximation ratio (3 when $\alpha \mid k$ and 4 otherwise). The approximation ratios of these algorithms are summarized in Table 1.1.

Why (k, α)-center clustering. The distance to the closest center is a noisy function that does not capture well the structure of the centers and is very sensitive to the exact locations of the centers. In the literature, prior work examined the αth nearest neighbor as a way to smooth this function, and better capture the local structure. In meshing, the αth nearest neighbor distance (i.e., using $\alpha = d + 1$) is known as the local-feature size, and it is used in measuring the quality of the meshing generated. For $\alpha = 2$, where the centers themselves have to be served by α-centers, this captures the notion that each centers requires to be supported by another center (or potentially $\alpha - 1$ clusters). This is known as a "base cluster" in a military context.

The case of $\alpha = 3$ in the plane is of special interest, as it corresponds to each client having three bases that are relatively close by, thus facilitating triangulation (i.e., compute one's location by measuring angles to known locations).

1.1. Our settings

Here, we define a fault tolerant variant of k-center that is 2-approximated by a variant of the greedy permutation (similar to that of [1]). We provide an algorithm for computing this permutation, and the techniques improve the runtime of the algorithm of the greedy permutation in [1].

1.1.1. Formal definition of the problem

In the following, let $(\mathcal{X}, \mathsf{d})$ be a metric space with its associated metric d. For a point $x \in \mathcal{X}$, and a set $P \subseteq \mathcal{X}$, let $\mathsf{D}(x, P) = \{\mathsf{d}(x, p) \mid p \in P\}$ be the (multi)set of all distances between x and the points of P.

▶ **Definition 4** (α-distance). *The* **α-distance** *of $x \in \mathcal{X}$ to P is the αth smallest number in $\mathsf{D}(x, P)$. Formally, as $\mathsf{D}(x, P)$ is a multiset, consider ordering its elements in a non-decreasing order, $d_1 \leq d_2 \leq \cdots \leq d_n$, where n is the cardinality of P. Then the α-distance of x to P is $\mathsf{d}^\alpha(x, P) = d_\alpha$.*

▶ **Definition 5.** *The* greedy α-distance permutation *(or simply* **the α-permutation***) of a point set P is defined incrementally as follows. Choose $p_1, \ldots, p_\alpha$ arbitrarily. Then, for i, $\alpha < i \leq \alpha n$, set $p_i = \operatorname{argmax}_{q \in P} \mathsf{d}^\alpha(q, P_{i-1})$, for $P_j = \{p_1, \ldots, p_j\}$.*

Unlike the definition given by Buchet *et al.* [1], points may appear in this permutation up to α times. This subtle issue seems to be key in what follows.

▶ **Definition 6.** *For a point set P as above, we define the collection $\lambda_1, \ldots, \lambda_{\alpha n}$ as follows. For $i \leq \alpha$, we let $\lambda_1 = \cdots = \lambda_\alpha = \infty$, and for $i > \alpha$, we let $\lambda_i = \mathsf{d}^\alpha(p_i, P_{i-1})$.*

Thus, λ_i is the clustering radius/quality when using $P_{i-1} = \{p_1, \ldots, p_{i-1}\}$ as the set of centers. In particular, we have $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{\alpha n}$.

Observe that a point might appear as a center in C at most α times (thus, we implicitly assume that $k \geq \alpha$). There are several subtleties in the above definition:

(I) SET/MULTISET: Can a point serve as a center more than once? If so, then C can be a multiset (as in Definition 2), otherwise, it is a set.

(II) CENTERS NEED NO SERVICE: In the above, we assumed that even the points in C must be served. However, a natural variant is when we consider the clustering radius only for points in $P \setminus C$ (as in Definition 3).

1.1.2. Differences between this and other variants

In the $P \setminus C$ variant introduced in Definition 3, points chosen as centers are exempt from the reliability requirements as non-centers (in other words, points selected as centers only need to be covered once and not α times by other centers). Although seemingly a small difference, this subtlety means it is not immediate how to apply the techniques as used in this work, and we are unsure how much of the machinery used here can be adapted to this setting.

Recall the P variant introduced in Definition 2 differs from the above in that points selected as centers are not exempt from the reliability requirements (even centers must be covered α times). However, this still differs from another variant considered in prior works, where each center cannot be selected multiple times. This difference also causes issues with the arguments presented here. See Table 1.1 for a comparison with references.

	Centers			
Ref / Comment	must be covered	reusable	Running Time	Approx.
[14]	X	X	$\mathsf{poly}(n)$	4
[3, 13]	X	X	$\mathsf{poly}(n)$	2
[13], $\alpha < 4$	✓	X	$\mathsf{poly}(n)$	2
[13], $\alpha \geq 4$	✓	X	$\mathsf{poly}(n)$	3
[15], $\alpha \mid k$	✓	X	$O(nk)$	3
[15], $\alpha \nmid k$	✓	X	$O(nk)$	4
This work	✓	✓	$O(c^{O(1)} n\alpha \log \alpha \log n)$	2

Table 1.1 Table with variants of fault tolerant clustering, their corresponding approximation ratios, and running times.

1.2. Our results

In Section 2 we observe that a minor modification of the standard greedy permutation provides us with a 2-approximation to (k, α)-center clustering – a surprisingly simple and elegant algorithm for this case compared to previous work.

In Section 3 we present an efficient algorithm for solving this problem in low-dimensional Euclidean space. This problem is well behaved, and can be (roughly) 4-approximated using the Net & Prune framework of Har-Peled and Raichel [10]. This results in an algorithm with expected $O(n)$ running time (see Lemma 16). This approximation can be then boosted into a 2-approximation, as described in Section 3.3, while maintaining the same running time. The resulting algorithm has linear running time assuming α is a constant and $k = O(n/\log n)$.

Greedy α-permutation. An added benefit of the greedy algorithm mentioned above, is that if one runs it exhaustively, it results in a greedy permutation of the input points. This permutation has the property that, for any k, the prefix of the first k points in the permutation forms a 2-approximation to the optimal (k, α)-clustering (that is, it is a clustering that works for all k simultaneously). It is thus interesting to compute this permutation quickly.

In Section 4.1, we show how to compute this greedy α-permutation in near linear time, under the assumption that the input is in a metric space with bounded doubling dimension,

and has polynomially bounded spread. The algorithm uses the key notion of friends lists to keep track of the relevant centers available for each point. This is described in Section 4.1.

Removing the dependency on the spread is a technically messy affair, described in Section 4.2. This is our main result, Theorem 28, which provides an algorithm to compute the greedy α-permutation in $O(n \log n)$ time, assuming both the doubling dimension and α are constants. The idea behind the new algorithm is to use a hierarchical clustering of the input (i.e., a HST), and open new clusters as the resolution of interest becomes smaller and smaller. This idea was described by Har-Peled and Mendel [8], and adapting it to our setting requires care.

2. (k, α)-center clustering via the greedy permutation

Our claim is that the first k elements of the α-distance permutation (see Definition 5) can be used to approximate the (k, α)-center clustering. Note that setting $\alpha = 1$ recovers the classical result for k-center clustering and the greedy permutation. In Section 4 we describe how to compute such an α-distance permutation, in a space with constant doubling dimension, in near linear time, assuming the spread of the input is bounded. We use the notation that $\mathbb{b}(p, r)$ denotes the ball of radius r around the point p.

▶ **Lemma 7.** *For $k \geq \alpha$ and a set P of n points in a metric space, the set of the first k points of the α-distance permutation of P is a 2-approximation (in the radius) to the optimal (k, α)-center clustering.*

Proof. Let C_{opt} be the multiset of centers used by the optimal solution. Let $\mathsf{r}_{\mathsf{opt}}$ be the radius. Let $P_{k+1} = \{p_1, \ldots, p_{k+1}\}$ be the multiset of the first $k+1$ centers picked by the greedy algorithm (the choices may not be unique). Consider the bipartite graph G over $C_{\mathsf{opt}} \cup P_{k+1}$ (assume for simplicity that the two sets are disjoint), where $c \in C_{\mathsf{opt}}$ is connected by an edge to $p \in P_{k+1}$ if $\|c - p\| \leq \mathsf{r}_{\mathsf{opt}}$. By the properties of C_{opt}, every vertex of P_{k+1} in G has degree at least α. Thus, the average degree of a vertex of C_{opt} in G is at least $(k+1)\alpha/k > \alpha$. Namely, there must be a point $c_{\mathsf{opt}} \in C_{\mathsf{opt}}$, that has at least $\alpha + 1$ points of P_{k+1} in distance at most $\mathsf{r}_{\mathsf{opt}}$ from it. Let $Q = \mathbb{b}(c_{\mathsf{opt}}, \mathsf{r}_{\mathsf{opt}}) \cap P_{k+1}$ be this set of "close" centers.

For $j > \alpha$, let $\lambda_j = \mathsf{d}^{\alpha}(p_j, P_{j-1})$, and observe that $\lambda_{\alpha+1} \geq \lambda_{\alpha+2} \geq \cdots \geq \lambda_{k+1}$. Observe that using $P_k = \{p_1, \ldots, p_k\}$ as the set of centers provides a (k, α)-cover with radius λ_{k+1} (i.e., the set of k disks of radius λ_{k+1} with centers at P_k covers each point of P α times).

Consider the last point $p_j \in P_k$ appearing in Q (i.e., max j). Since $|Q| \geq \alpha + 1$, it must be that $j > \alpha$.

By the previous inequality and the fact that $Q - p_j \subset P_{j-1}$, we have that

$$2\mathsf{r}_{\mathsf{opt}} \geq \operatorname{diam}(Q) \geq \mathsf{d}^{\alpha}(p_j, Q - p_j) \geq \mathsf{d}^{\alpha}(p_j, P_{j-1}) = \lambda_j \geq \lambda_{k+1},$$

as $|Q - p_j| \geq \alpha$, and $Q \subseteq \mathbb{b}(c_{\mathsf{opt}}, \mathsf{r}_{\mathsf{opt}})$. ◀

This proof is a generalization of the analogous result for the greedy permutation and k-center clustering. We have the following observations.

▶ Remark 8.

(A) A more efficient algorithm is possible in the case where α divides k ([15] has a similar observation). Let $\ell = k/\alpha$ and repeat the above proof where, in each iteration, we place α centers on each point we wish to select as a center (i.e. the worst served point) instead

of just one. This continues for ℓ iterations, placing all $k = \ell \cdot \alpha$ centers, a factor α fewer than the k iterations the standard algorithm runs.

Observe then that this provides the same lower bound on the maximum clustering radius, as we may still use the triangle inequality to bound the cost of the worst served point. Although this does not improve the approximation ratio, the algorithm needed to compute this is identical to that of the standard greedy permutation.

However, in the case that $\alpha \nmid k$, it is not clear how to use a similar strategy without worsening the approximation ratio.

(B) To obtain the above approximation ratio, note that we implicitly use the property that each point can be selected as a center α times. This is because we need to use the fact that many centers are close together to bound the greedy clustering radius, and this argument fails if we cannot always place a center on the worst served point.

As this technique achieves an approximation ratio of 2, it seems unlikely that the technique of Kumar and Raichel [15] can improve on this approximation ratio. In particular, the approximation ratio of the preceding proof does not depend on whether $\alpha \mid k$, and matches the best known approximation ratio when $\alpha = 1$.

3. Fast approximation for (k, α)-center clustering

In this section, we apply the framework of Har-Peled and Raichel [10] to obtain a fast algorithm for the case of fixed k. We begin by restating the core definitions of this framework. For greater intuition and full details, refer to the original work.

3.1. Background

▶ **Definition 9** (r-net). *For a point set P with a metric d and a parameter $r > 0$, an r-net of P is a subset $\mathsf{N} \subseteq P$ such that,*

(i) For every $p, q \in \mathsf{N}$ where $p \neq q$, $\mathsf{d}(p, q) \geq r$,

(ii) For all $p \in P$, we have that $\min_{q \in \mathsf{N}} \mathsf{d}(p, q) < r$.

We let $\mathsf{net}(r, P)$ denote an r-net of P. One can compute $\mathsf{net}(r, P)$ in $O(n)$ time [10, Lemma 2.2.1].

▶ **Definition 10** (φ-decider). *Given a function $f : X \to \mathbb{R}$, a φ-decider for f is a procedure that, given any $x \in X$ and $r > 0$ returns either:*

(i) $f(x) \in [\alpha, \varphi\alpha]$, where α is some real number.

(ii) $f(x) < r$.

(iii) $f(x) > r$.

▶ **Definition 11** (φ-NDP). *An instance of a φ-NDP (Nice Distance Problem) is a pair (W, Γ), where $W \subseteq \mathbb{R}^d$ is a set of n distinct weighted points and Γ is the context (of size $O(n)$) of the problem, containing relevant parameters.*

The task is to evaluate a function $f(W, \Gamma) \to \mathbb{R}^+$ defined over the input pairs, with the following properties:

(i) Decider: There exists an $O(n)$ time φ-decider for f, for some constant φ.

(ii) Lipschitz: Let Q be any Δ-translation of W. Then $|f(W, \Gamma) - f(Q, \Gamma)| \leq 2\Delta$.

(iii) Prune: If $f(W, \Gamma) < r$, then $f(W, \Gamma) = f(W^{<r}, \Gamma')$, where $W^{<r}$ is the set of r-close points, and Γ' is an updated context which can be computed in $O(n)$ time.

The utility of these definitions lies in the following lemma.

▶ **Lemma 12** ([10]). *Given an instance of a φ-NDP defined by a set of n points in $\mathbb{R}^d$, one can get a φ-approximation to its optimal value, in $O(T(n))$ expected time, where $T(n)$ is the running time of the decider procedure, and $\varphi \geq 2$.*

3.2. Applying the framework of net & prune

With respect to the above framework, the function of interest for our problem is $f_{cen}(P, k, \alpha)$ – the radius of the optimal (k, α)-center clustering of P. The following provides a way to construct a decider.

▶ **Lemma 13.** *For a set P of n points, consider computing an r-net N_r. If $|\mathsf{N}_r|\,\alpha \leq k$, then $f_{cen}(P, k, \alpha) < r$. Otherwise, $r/2 < f_{cen}(P, k, \alpha)$.*

Proof. For the first case, consider any $x \in \mathsf{N}_r$ and observe that placing α centers at x means each point is served with cost $\leq r$. The requirement that $|\mathsf{N}_r|\,\alpha \leq k$ means the previously described placement is a feasible solution with cost at most r.

Otherwise, consider constructing a (k, α)-cover of P with balls of radius $r/2$. By the definition of N_r, each ball used can only cover at most one of these net points. Since $|\mathsf{N}_r| \cdot \alpha > k$, then clearly $r/2 < f_{cen}(P, k, \alpha)$. ◀

▶ **Lemma 14.** *One can implement an $(4 + \varepsilon)$-decider for (k, α)-center clustering that runs in $O(n)$ time.*

Proof. The decider algorithm proceeds as follows. Compute an r-net N_r and apply the procedure from Lemma 13. If $|\mathsf{N}_r|\,\alpha \leq k$, report that $f_{cen}(P, k, \alpha) < r$. Otherwise, repeat this process with a $(2 + \varepsilon/2)r$-net, but if the process succeeds then report $f_{cen} \in [r/2, (2 + \varepsilon/2)r]$, and if it fails, report that $f_{cen}(P, k, \alpha) > r$. The correctness of this algorithm follows immediately from Lemma 13. ◀

▶ **Lemma 15.** *Consider an instance (W, k, α) of (k, α)-center clustering, where W is a set of weighted points. The problem of computing (k, α)-center clustering for (W, k, α) is an $(4 + \varepsilon)$-NDP.*

Proof. We verify each part of the definition.

(i) Decider: Follows from the above lemma.

(ii) Lipschitz: Follows as any Δ-translation of W, the distance between a point and any center it is served by increases by at most 2Δ.

(iii) Prune: Points at distance $\geq r$ from all others must only be served by themselves α times and cannot serve any other point (as then the service cost would be $\geq r$). Thus, they can be removed and the value of k (number of centers to be chosen) decremented by α.

◀

With the above and Lemma 12, we can apply the framework of Har-Peled and Raichel to obtain the following.

▶ **Lemma 16.** *Let $\varepsilon \in (0, 1)$ be a fixed constant. Given a set P of n points in $\mathbb{R}^d$, and parameters k, α, one can compute a $(4 + \varepsilon)$-approximation to the optimal (k, α)-clustering of P in $O(n)$ expected time.*

3.3. Nearly linear time approximation

With this constant factor approximation, we want to improve this to a 2-factor approximation in a manner similar to [6, Lemma 6.5]. We delegate the proof of the following lemma to Appendix A.

▶ **Lemma 17.** *Let $P \subset \mathbb{R}^d$ be a set of n points, and let k, r, α be parameters such that $\mathsf{r}_{\mathsf{opt}} \leq r \leq c \cdot \mathsf{r}_{\mathsf{opt}}$, $k \leq n\alpha$, where $c \geq 1$ is a constant. Then, we can compute a 2-approximation to the optimal (k, α)-center clustering of P in $O(n\alpha \log \alpha + k \log k)$ time.*

▶ Remark 18. Although this runtime seems excessive, recall that the idea from [6] was used to eliminate the scheduling structure introduced by the friends list (see Section 4.1.1 for a description of this data structure and the impact on running time). However, this is not the data structure that contributes most to the runtime of Section 4.

Furthermore, removing the factor of α in the runtime can be done by using a grid which has cell diameter proportional to ε, and then treating points in the same cell as weighted points (at the cost of a multiplicative factor of ε error).

4. Computing the greedy α-distance permutation

4.1. Polynomial spread

4.1.1. Construction algorithm

Here, we present a combinatorial algorithm for computing the α-distance permutation. By the preceding section, this gives a 2-approximation algorithm for the desired clustering problem. We modify this algorithm as-needed in later sections. This combines elements of the algorithm of Buchet *et al.* [1] using additional ideas from Feder and Greene [4].

At a high level, the algorithm maintains, for each point, its α closest centers (with multiplicity). It then opens a center at the worst served point (which may already be a center). This algorithm can compute any prefix of the full α-distance permutation by stopping early. Additional data structures are used to perform this scheduling.

We first give a description of the data structures involved. The main heap M is used to schedule which point is to be added next to the permutation, where each point p has an additional heap that maintains the α closest points from the permutation to p.

(i) For each point p which can still be selected as a center, let T_p be a max-heap with the distances to the α closest centers to p (with multiplicity). This heap is tracking the distance $\mathsf{d}^\alpha(p, P_i)$ in the ith iteration, along with bi-directional pointers to the centers stored.

(ii) Maintain a meta-max-heap M which stores each point p keyed on the max distance in T_p. Points p are removed from M after they have been selected to the permutation α times.

(iii) Each point p selected as a center maintains a friends list F_p that contains all centers (with corresponding multiplicity) in distance $\leq 4\lambda_i$ and none at distance $\geq 8\lambda_i$. Centers are removed from this friends list lazily. Points which have never been selected as centers do not have a friends list.

Conceptually, the algorithm is maintaining a radius $\mathsf{r}_{\mathsf{curr}}$ that is decreasing, corresponding to the service cost of the worst served point. As $\mathsf{r}_{\mathsf{curr}}$ decreases, we add points with service cost $\mathsf{r}_{\mathsf{curr}}$ from M to the set of centers and update data structures accordingly.

In the ith iteration, the algorithm proceeds as follows: First, the point p with max value in M is selected (once) as a center. Let $c_p \in P_{i-1}$ be the closest center to p in the previous

iteration (possibly p itself). Then, we add p to the heaps T_q corresponding to the points q served by c_p and centers in the friends list of c_p. The new friends list of p is computed by taking the friends list of the closest center to p from two phases (see Definition 19) ago, c'_p, and choosing those in $\mathcal{b}(p, 4\lambda_i)$.

4.1.2. Correctness

We first show that the newly computed friends list is correct whenever a point is chosen as a center.

▶ **Definition 19.** *A* **phase** *starting at the jth iteration of the algorithm terminates at the first iteration $i > j$ such that $\lambda_{i-1} \leq \lambda_{j-1}/2$.*

▶ **Observation 20.** *At the ith iteration, the friends list of p_j (for $i \geq j$) contains all centers within distance at most $\min\{8\lambda_i, 4\lambda_j\}$ from p_j.*

We apply this in the following lemma.

▶ **Lemma 21.** *Selecting all centers in the friends list of c'_p in $\mathcal{b}(p, 4\lambda_i)$ suffices for the construction of the friends list of a point p selected as a center in the ith iteration.*

Proof. Define the set $S = \{p_j \mid j < i, \mathsf{d}(p_j, p_i) \leq 4\lambda_i\}$ and observe that S is all elements that are required to be included in the friends list F_{p_i} (multiplicity does not matter here). Thus, it suffices to show that all elements of S are scanned in the creation of the friends list of p_i.

Recall that c'_p is the center closest to p from two phases ago, and let λ'_p be the radius when c'_p was selected as a center. By definition, $2\lambda_i \leq \lambda'_p \leq 4\lambda_i$, meaning that $\mathsf{d}(p_i, c'_p) \leq 4\lambda_i$. Thus, at the start of the ith iteration, the friends list of c'_p contains all centers in distance $\leq \min\{8\lambda_i, 4\lambda'_p\} = 8\lambda_i$ from c'_p.

By the triangle inequality, any center $p_j \in S$ is such that

$$\mathsf{d}(p_j, c'_p) \leq \mathsf{d}(p_j, p_i) + \mathsf{d}(p_i, c'_p) \leq 4\lambda_i + 4\lambda_i = 8\lambda_i,$$

so it follows that each $p_j \in S$ is scanned in the creation of the friends list of p_i as desired. ◀

4.1.3. Bounding the runtime

To bound the runtime, we pay attention to the cost of maintaining each of the data structures in each iteration.

▶ **Lemma 22.** *The algorithm runs in $O(\mathcal{c}^{O(1)} n\alpha \log n \log \Phi)$ time, where Φ is the spread of the input point set P (the ratio of diameter and smallest distance of two distinct points), and $\mathcal{c}$ is the doubling constant of the ambient space.*

Proof. We first consider the initialization time. As the first α points can be selected arbitrarily, we simply choose an arbitrary point and select it α times to be a center, then initialize the heap data structures with this point of multiplicity α. This takes $O(n)$ time.

As the algorithm runs for $n\alpha$ iterations, we next analyze the runtime of maintaining each data structure. Key to this analysis is the observation that the friends list F_p for any point $p \in P$ is such that $|F_p| = \mathcal{c}^{O(1)}$. This is because, at the ith iteration, the friends list consists of centers (counted without multiplicity) at most $8\lambda_i$ distance from p, so the bound follows from the doubling constant.

(i) The max-heaps T_p for nearby points p must be updated each time a new center is chosen. For some iteration $j-1$, observe that the ball of size $4\lambda_{j-1}$ around each point $q \in P$ contains at most $\mathfrak{c}^3$ points of P_j (counted without multiplicity). As each point can only appear in P_j at most α times, then in each phase of the algorithm, each point of P can only receive updates about new centers $\alpha\mathfrak{c}^3$ times at most, and each of these updates takes $O(\log \alpha)$ time since the heap T_p holds at most α elements.

We can bound the number of phases by $O(\log \Phi)$, where Φ is the spread of the initial point set. Putting it all together, updating these data structures takes $O(\mathfrak{c}^{O(1)} n\alpha \log \alpha \log \Phi)$ time.

(ii) From the previous discussion, we have that in each iteration, only $\mathfrak{c}^{O(1)}$ points have their heaps updated. Thus, this only induces $\mathfrak{c}^{O(1)}$ updates to elements of M, where each update takes $O(\log n)$ time. Thus, maintaining M takes $O(\mathfrak{c}^{O(1)} n\alpha \log n)$ time as there are $n\alpha$ iterations at most.

(iii) As the friends list of each point is of size at most $\mathfrak{c}^{O(1)}$ (lazily discarding points which are too far away), updating the friends list takes $\mathfrak{c}^{O(1)}$ time in each iteration. Thus, these updates take $O(\mathfrak{c}^{O(1)} n\alpha)$ time overall.

From the above discussion, the algorithm runs in $O(\mathfrak{c}^{O(1)} n\alpha \log n \log \Phi)$ time. ◀

4.2. Removing the dependency on the spread

In this section, we introduce the *hierarchically separated tree (HST)*. We describe how to apply this data structure to remove the spread from the runtime of the algorithm in Section 4.1. We mostly follow the treatment of [8].

4.2.1. Background on HSTs

The following definition is from [8].

▶ **Definition 23** (HST). *A* Hierarchically well-separated tree *(HST) is a metric space defined on the leaves of a rooted tree T. With each vertex $u \in T$, there is an associated label $\Delta_u \geq 0$ such that $\Delta_u = 0$ iff u is a leaf of T. The labels are such that if a vertex u is a child of v, then $\Delta_u \leq \Delta_v$. The distance between two leaves x and y of T is defined as $\Delta_{\mathtt{lca}(x,y)}$, where $\mathtt{lca}(x, y)$ is the least common ancestor of x and y in T.*

We also associate with every vertex $u \in T$ a leaf rep_u of the subtree rooted at u. We also require that $rep_u \in \{rep_v : v \text{ is a child of } u\}$.

Furthermore, we assume that all of the underlying trees used for our HSTs are binary, as any HST can be converted to a binary HST in linear time while retaining the underlying metric.

As a combination of Lemma 3.6 and Theorem 5.3 from [8], we have the following.

▶ **Lemma 24.** *Given an n-point metric space P, one can construct an HST T in (expected) nearly linear time that $O(n)$ approximates P (specifically, $O(n)$ approximates the metric over pairs of points of P).*

This is the main ingredient in removing the spread from the runtime. At a high level, we combine points that are close in the HST while the distance for the worst served point is close to the active distance for the HST. Once the current distance drops enough, we introduce new points through corresponding split operations. This introduces a new type of event into the algorithm of Section 4.1.

4.2.2. The improved algorithm

In addition to the previously defined data structures, the algorithm uses an HST T as defined in the previous section and runs over a dynamic set of points, corresponding to levels in the HST. Specifically, the current set of points handled by the algorithm corresponds to representatives rep_v such that $\Delta_v \leq \mathsf{r}_{\mathsf{curr}}/n^4 \leq \Delta_{\bar{\mathrm{p}}(v)}$, where $\mathsf{r}_{\mathsf{curr}}$ is the current greedy radius of the algorithm and the label Δ_v is the diameter of the subtree rooted at v in T. Also, $\bar{\mathrm{p}}(v)$ is the parent of v in T.

As a result, we must now handle the event where the current greedy radius $\mathsf{r}_{\mathsf{curr}}$ decreases such that $\Delta_v > \mathsf{r}_{\mathsf{curr}}/n^4$ for some current point rep_v. In this case, we must replace v with its two children, u and w.

By the construction of T, we have that $\text{rep}_v \in \{\text{rep}_u, \text{rep}_w\}$, meaning we only have to deal with the insertion of one new point at each split event. We can schedule these split events by sorting all of the nodes v of the HST T in a heap, ordered by their label Δ_v. Then, along with the heap M, process events from either heap based on the decreasing radius $\mathsf{r}_{\mathsf{curr}}$.

In detail, processing a split event occurs as follows. Let $v \in T$ be the node undergoing the split event, with child nodes u, w. Without loss of generality, let u be the child of v such that $\text{rep}_u \neq \text{rep}_v$ (so $\text{rep}_w = \text{rep}_v$). Let c_v be the center closest to rep_v (the case where $c_v = \text{rep}_v$ is identical). We update each of the data structures as follows:

(i) As u is a new point which is not a center, we must create a new heap T_u holding the α closest centers to u. To do this, we scan F_{c_v} of c_v and populate T_u with the α closest centers to u. Since $|F_{c_v}| = \mathscr{c}^{O(1)}$, this takes $O(\mathscr{c}^{O(1)} + \alpha \log \alpha)$ time.
(ii) Now that we have T_u, we must insert it into M, which takes $O(\log n)$ time.
(iii) New points introduced by split events are not centers, so these events do not require any updates to any friends lists.

4.2.3. Correctness

The use of the HST introduces a small amount of distortion into the result of the algorithm. Although this error is small enough for most applications, it requires a precise statement.

▶ **Lemma 25.** *Let $\pi = \langle p_1, \ldots, p_{\alpha n} \rangle$ be the permutation produced by the algorithm (where points can be repeated according to multiplicity). Let $r_{k+1} = \mathsf{d}^{\alpha}(p_{k+1}, P_k)$ be the current clustering radius after the first k centers are selected. Then, the multiset of balls $B_{k+1} = \left\{ \mathscr{b}\left(p_i, (1+n^{-2})r_{k+1}\right) \right\}_{i=1,\ldots,k}$ covers all points of P at least α times each, and any open ball of radius $(1-n^{-2})r_{k+1}/2$ contains at most α points of P_k.*

Proof. Starting with the first claim, by the invariants of the algorithm, balls of radius r_{k+1} around $p_1, \ldots, p_k$ cover all of the points of P α times each. Because of the HST, then the points considered by the algorithm may represent points which are in distance $n^{-2}r_{k+1}$ further away from them. Thus, by expanding the balls around each of $p_1, \ldots, p_k$ to radius $(1+n^{-2})r_{k+1}$, we cover all points of P.

For the second claim, observe that the algorithm is greedily choosing the worst served point, and the active radius when the points $P_k = \{p_1, \ldots, p_k\}$ were chosen is r_{k+1}. Thus, if some open ball $\mathscr{b}$ of radius $r_{k+1}/2$ contains $\geq \alpha + 1$ (with multiplicity) points of P_k, by the triangle inequality, this means the worst served point (and thus the greedy radius) is $< r_{k+1}$, a contradiction. The final claim follows from the observation that points considered by the algorithm represent points from P with distortion at most $n^{-2}r_{k+1}$. ◀

4.2.4. Running time

To bound the running time, we use a charging argument by charging a certain amount of work to each of the nodes of the HST T. As the number of nodes is linear, this leads to a nearly linear time algorithm. The following lemma makes this precise.

▶ **Lemma 26.** *The expected running time of this algorithm is $O(\mathfrak{c}^{O(1)} n\alpha \log \alpha \log n)$, where $\mathfrak{c}$ is the doubling constant.*

Proof. By Lemma 24 the construction of the HST runs in $O(\mathfrak{c}^{O(1)} n \log n)$ time. Observe that there are a total of n split events, so by previous discussion, processing these events takes total work $O(n(\mathfrak{c}^{O(1)} + \alpha \log \alpha + \log n))$. Observe also that updating all friends lists takes $O(\mathfrak{c}^{O(1)} n\alpha)$ time (as split events do not treat the new point as a center) and updating M takes $O(\mathfrak{c}^{O(1)} n\alpha \log n)$ time in total.

Thus, to show the claimed runtime, we must bound the total time taken to update the heap data structures (their initial creation was handled earlier). As in the previous analysis, we divide the execution of the algorithm into phases where a phase of the algorithm starts at the jth iteration and ends at the first $i > j$ such that $\lambda_{i-1} \leq \lambda_{j-1}/2$. Consider a point $p \in P$ being inserted into the point set when a node $v \in T$ in the HST is being split during the ith phase. After the initial creation of the heap T_p, we consider all of the updates being made to T_p until the next split event occurs on p.

For the next $L = 10 \log n$ phases of the algorithm, we charge the work incurred by p to v. Otherwise, when p is $> L$ phases away from the split event where it was introduced, then p represents a set of points which has diameter $\leq \mathsf{r}_{\mathsf{curr}}/n^2$ and, by the condition on L, $\mathsf{r}_{\mathsf{curr}} \leq \Delta_v/n^{10}$. From this, we have that $P \cap \mathcal{b}\left(p, \mathsf{r}_{\mathsf{curr}} n^2\right) \subset P \cap \mathcal{b}\left(p, \Delta_v/n^4\right) \subset P \cap \mathcal{b}\left(p, \mathsf{r}_{\mathsf{curr}}/n^2\right)$. This implies that the points represented by p are far enough away from other points of P (with respect to $\mathsf{r}_{\mathsf{curr}}$), that p cannot be part of a friends list (and thus, not require any updates of the heap T_p) until it undergoes another split event.

As each individual update takes $O(\log \alpha)$ time to process, each node in the HST is charged with $O(\mathfrak{c}^{O(1)} \alpha \log \alpha \log n)$ work because there are only $O(\mathfrak{c}^{O(1)} \alpha)$ many updates that can be initiated from the current phase. Thus, the total work is $O(\mathfrak{c}^{O(1)} n\alpha \log \alpha \log n)$. ◀

▶ Remark 27. By removing the spread from the runtime of our algorithm, we partially answer the open question of [1], as the output of our algorithm differs from theirs only slightly. In their definition of the α-distance permutation, the only difference is that each point can only be selected once. Thus, to achieve this, we use the same scheduling machinery but simply remove a point from M once it is selected instead of reinserting it.

4.2.5. The result

▶ **Theorem 28.** *Given a set of n points P in a metric space of doubling constant $\mathfrak{c}$, and a parameter α, the algorithm of Section 4.2.2 computes (exactly) the α-distance permutation in expected time $O(\mathfrak{c}^{O(1)} n\alpha \log \alpha \log n)$.*

▶ **Corollary 29.** *Suppose we have $k > \alpha$ and a set of n points P, in a metric space of doubling constant $\mathfrak{c}$. By Lemma 7, Section 4.2.2 gives a 2-approximation to the optimal (k, α)-center clustering in expected time $O(\mathfrak{c}^{O(1)} n\alpha \log \alpha \log n)$.*

References

1 Mickaël Buchet, Bianca B. Dornelas, and Michael Kerber. Sparse higher order čech filtrations. In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on*

Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA, volume 258 of *LIPIcs*, pages 20:1–20:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

2 Kevin Buchin, Sariel Har-Peled, and Dániel Oláh. A spanner for the day after. *Discrete Comput. Geom.*, 64(4):1167–1191, 2020.

3 Shiva Chaudhuri, Naveen Garg, and R. Ravi. The p-neighbor k-center problem. *Information Processing Letters*, 65(3):131–134, 1998.

4 Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 434–444, New York, NY, USA, 1988. Association for Computing Machinery.

5 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

6 Sariel Har-Peled. Clustering motion. *Discret. Comput. Geom.*, 31(4):545–565, 2004.

7 Sariel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, USA, 2011.

8 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006.

9 Sariel Har-Peled, Manor Mendel, and Dániel Oláh. Reliable spanners for metric spaces. *ACM Trans. Algo.*, 19(1):7:1–7:27, 2023.

10 Sariel Har-Peled and Benjamin Raichel. Net and prune: A linear time algorithm for euclidean distance problems. *J. ACM*, 62(6):44:1–44:35, 2015.

11 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, 1986.

12 Dorit S. Hochbaum and David B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *J. ACM*, 33(3):533–550, May 1986.

13 Samir Khuller, Robert Pless, and Yoram J. Sussmann. Fault tolerant k-center problems. *Theoretical Computer Science*, 242(1):237–245, 2000.

14 S.O. Krumke. On a generalization of the p-center problem. *Information Processing Letters*, 56(2):67–71, 1995.

15 Nirman Kumar and Benjamin Raichel. Fault tolerant clustering revisited. In *Proceedings of the 25th Canadian Conference on Computational Geometry, CCCG 2013, Waterloo, Ontario, Canada, August 8-10, 2013*. Carleton University, Ottawa, Canada, 2013.

A. Proof of Lemma 17

Restatement of Lemma 17. *Let $P \subset \mathbb{R}^d$ be a set of n points, and let k, r, α be parameters such that $\mathsf{r}_{\mathsf{opt}} \leq r \leq c \cdot \mathsf{r}_{\mathsf{opt}}$, $k \leq n\alpha$, where $c \geq 1$ is a constant. Then, we can compute a 2-approximation to the optimal (k, α)-center clustering of P in $O(n\alpha \log \alpha + k \log k)$ time.*

Proof. At a high level, we will use the constant factor approximation along with grid cells to play the role of the friends list in our first algorithm. This allows for a running time that has reduced logarithmic factors.

Let G_Δ be a partition of $\mathbb{R}^d$ into a uniform grid with side length $\Delta = r/(2c\sqrt{d}) \geq \mathsf{r}_{\mathsf{opt}}/(2c\sqrt{d})$. Also, $\Delta \leq c \cdot \mathsf{r}_{\mathsf{opt}}/(2c\sqrt{d}) = \mathsf{r}_{\mathsf{opt}}/(2\sqrt{d})$. Compute the cells of G_Δ containing points of P in overall linear time (using hashing and the floor function).

Each ball in the optimal clustering can only intersect $\leq (2\mathsf{r}_{\mathsf{opt}}/\Delta + 1)^d = O((4c\sqrt{d}+1)^d) = O(1)$ grid cells, and it follows that two non-empty grid cells $\geq 2r$ from each other can't be served by the same cluster in a 2-approximation to the optimal solution. Thus, we compute, for each non-empty grid cell c, the $O(1)$ non-empty cells in distance $\leq 2r$ from c, which we denote $N(\mathsf{c})$.

Now, in a similar manner to the original algorithm, we repeatedly choose the worst served point to be a center, except now we do not need to update the friends list data structures.

As in the original algorithm, each point $p \in P$ maintains a heap T_p that holds its distance to the α closest centers. Each time a new center is chosen in a cell c, we visit all of the neighboring cells $N(\mathsf{c})$ to broadcast distance updates to the individual points. We also have each cell maintain its worst-served point (the one that could be chosen as the next center) after each distance update it receives. This worst-served point is updated when the points in each cell are scanned.

The initial scan of points takes $O(n)$ time. To find the next point to select as a center, the algorithm uses a heap M of size $O(k)$, with one entry for each non-empty cell, where the key for each cell is the point in the cell which is worst served. Running for k steps, the algorithm performs $O(k)$ insertions and weight updates on M, where each operation takes $O(\log k)$ time, so maintaining M takes $O(k \log k)$ time in total.

Each cell can only contain at most α centers, as the cells have diameter $\leq \Delta\sqrt{d} < \mathsf{r}_{\mathsf{opt}}$, and so each cell can only initiate α distance updates to its neighbors. Therefore, each point in each cell is scanned at most $\alpha \cdot |N(\mathsf{c})| = O(\alpha)$ times, and each of these updates is operating on a heap of size $O(\alpha)$, so this results in a total of $O(n\alpha \log \alpha)$ update time overall. With the preceding discussion, this gives a final runtime of $O(n\alpha \log \alpha + k \log k)$. ◀