Faster and Simpler Greedy Algorithm for k-Median and k-Means

Max Dupré la Tour
McGill University, Montreal, Canada

David Saulpic
CNRS & Université Paris Cité, IRIF, F-75013, Paris, France

Abstract

Clustering problems such as k-means and k-median are staples of unsupervised learning, and many algorithmic techniques have been developed to tackle their numerous aspects.

In this paper, we focus on the class of greedy approximation algorithm, that attracted less attention than local-search or primal-dual counterparts. In particular, we study the recursive greedy algorithm developed by Mettu and Plaxton [SIAM J. Comp 2003]. We provide a simplification of the algorithm, allowing for faster implementation: our algorithm matches the state-of-the-art running time for computing a constant-factor approximation in Euclidean space and graph metrics, and, in addition, is the first near-linear-time to compute a polylogarithmic approximation in Euclidean space.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Clustering, k-means, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.84

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2407.11217

1 Introduction

Clustering problems such as k-median and k-means lie at the intersection of applied and theoretical algorithms. Applied, because they are staples of unsupervised learning and are widely used for both classification and data analysis; and theoretical, because they have simple and elegant formulations, serving as testbeds for many algorithmic techniques.

In this paper, we focus on approximation algorithms. In general, we can distinguish three big families of techniques: those based on linear programming (e.g., primal-dual), local search, and greedy. For clustering, the primal dual method is the basis for the most accurate approximation algorithm, where a long line of work reached a $2+\varepsilon$ approximation for k-median [18] and 5.93 for k-means [9]. In addition, primal-dual based algorithms are quite flexible with respect to changes in the objective function – for example, they can be extended to handle the ordered k-median problem [8], clustering with outliers [34], or give improved approximation for high-dimensional Euclidean space [10]. However, those have quite slow running time and are not expected to be applied. Local search is competitive for specific input: it yields an almost linear time approximation scheme in low-dimensional Euclidean space [15], or recover the optimal clustering under some stability assumption [24]. Local search also provides great approximation, as small as $2.836+\varepsilon$ for k-median [19]. One drawback is, here as well, the running time, as evaluating the impact of a single local step takes in general linear time. We note here that, following the initial release of our work, Jiang, Jin, Lou and Lu [32] presented an implementation of local-search based on Locality Sensitive Hashing, to get an $O(c)$-approximation in time $n^{1+1/c+o(1)}$ in Euclidean space and graph metrics. We discuss this further below.

53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026).
Editors: Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis;
Article No. 84; pp. 84:1–84:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The third category, greedy algorithm, has been less investigated. The usual advantage of greedy algorithm is their simplicity, that often highlights structural properties of the problem at hand and allow for fast implementation. However, for clustering the two most "natural" greedy have strong lower bound. Iteratively adding the center that decreases most the cost does not give better than $\Omega(n)$ approximation, and its reversed version, which starts from all points being centers and removing the point that increases least the cost, is a $\Omega(\log n/\log\log n)$-approximation [13]. On the other hand, the most famous practical algorithm for solving k-means is a randomized greedy, k-means++ [5]. Its approximation guarantee is however a super-constant $\Theta(\log k)$, and its running time $O(nkd)$ (in Euclidean space $\mathbb{R}^d$) becomes prohibitive for modern, very large scale data and values of k. The only greedy known to be a constant-factor approximation is a recursive greedy algorithm, due to Mettu and Plaxton [39]. However, this algorithm has a strong reputation of being intricate. In addition, although it is among the first and few constant-factor approximation algorithm, it has not seen subsequent improvement, extension or application.

In this paper, our goal is to simplify the recursive greedy algorithm of [39], in order to provide new structural hindsight on clustering problems. We also hope that simplification will allow for more applications. And, indeed, we present several of those, including the first linear time polylog k approximation algorithm for k-means in $\mathbb{R}^d$.

1.1 Our contribution

The algorithm works not only for k-median and k-means, but for the more general (k, z)-clustering, which seeks to minimize the sum of the z-th power of the distance from each client to its center (k-means is $z = 2$, k-median $z = 1$). We show the following theorem:

▶ **Theorem 1** (see Theorem 5 and Theorem 6). *Let $(P, dist)$ be a metric space with aspect-ratio Δ,*[1] *and $c > 5$ be a constant. Suppose there is:*

- *an algorithm that computes the number of points in all balls centered on input point with radius $(2c)^i$ in time T_{Value}, and*
- *a datastructure with preprocessing time T_{Pre} and that is able to: remove any point from the ground set in time T_{rm}, and, given x and r, compute a set $N(x, r)$ points of the current ground set such that $B(x, r) \subseteq N(x, r) \subseteq B(x, c \cdot r)$ in time $T_N \cdot |N(x, r)|$.*

Then the recursive greedy algorithm can be implemented such that it is a poly(c)*- approximation and has running time $T_{\text{Value}} + O\left((T_{Pre} + (T_{rm} + T_N) \cdot |P|) \cdot \log \Delta\right)$.*

Furthermore, this algorithm not only computes a solution to (k, z)-clustering, but it also provides an ordering of the input points $p_1, \ldots, p_n$ such that for any k, the set $\{p_1, \ldots, p_k\}$ forms an $O(1)$-approximation to (k, z)-clustering. This variant of (k, z)-clustering is referred to as online [39] or incremental [41, 12] in the literature. This is a shared feature with algorithms based on k-means++ seeding, where each prefix is a $O(\log k)$ approximation.

We apply this theorem in Euclidean space and sparse graph with different datastructure, and get the following corollaries:

▶ **Corollary 2.** *The recursive greedy algorithm can be implemented such that it computes, for any constant $c \geq 5$:*

- *a* poly(c)*-approximation to incremental (k, z)-clustering in graphs, with running time $O(m^{1+1/c} \log \Delta)$,*

[1] The aspect-ratio is the ratio between the largest distance and the smallest non-zero distance in the metric.

- *a* poly(c)*-approximation to incremental* (k,z)*-clustering in Euclidean space, running in time* $O\left(n^{1+1/c+o(1)} d \log \Delta\right)$,
- *a* polylog(k)*-approximation to* (k,z)*-clustering in Euclidean space, running in time* $\tilde{O}(nd + n \log\log \Delta)$.

We note that, in Euclidean space, the $(n-1,z)$-clustering problem is equivalent to the closest pair problem : as remarked in [7], the current tradeoff is poly(c)-approximation in time $n^{1+1/c}$, [2, 3] which is believed to be tight [40]. In this case, our Euclidean constant-factor approximation would be tight too.

We also note that, for the standard (k,z)-clustering, it is easy to reduce the aspect-ratio Δ to poly(n): we show in the full version of our paper how to do so in near-linear time in graphs, and [26] show how to do it in time $O(nd + n \log\log \Delta)$ in Euclidean space. Hence, for the standard k-means and k-median in sparse graphs, the Δ in the bounds above can be replaced with n – and, for Euclidean space, this combined with dimension reduction to replace d with $O(\log k)$ [37]. The last result of the corollary is phrased to reflect those improvements.

As mentioned previously, this theorem relies on a simplification of the recursive greedy algorithm, which is as follows. The algorithm places centers one by one, with the following rule to place the $(i+1)$-th center. For any ball of the metric space, define its value to be the number of input point inside the ball, times its radius to the power z. Out of the balls that are "far" from the first i centers, select the one with maximal value. Then, recursively select a ball of radius divided by 2 and with a center "close" to the current ball, until there is a single point in the ball. This final point is the $(i+1)$-th center.

Hence, it is merely necessary to be able to count efficiently the number of point in a ball, and to identify points "close" to a given ball. This can be done efficiently in many metric spaces, e.g. using locally sensitive hashing. In addition, we show that all the quantities involved – number of points, radius, "far" and "close" – can be approximated, allowing for efficient implementation.

Comparison with prior work. We see our main contribution as a deeper understanding of the greedy algorithm from Mettu and Plaxton. This results in some novel tradeoff between approximation and running time for (k,z)-clustering, that we compare below with other existing techniques.

We note first that our $n^{1+1/c}$ algorithms (in graphs and Euclidean space) results do not beat the state-of-the-art, in terms of approximation and running time: indeed, in sparse graph, an algorithm from [42] (which uses the primal-dual techniques as subroutine) runs in near-linear time and computes a constant-factor approximation to k-median. An equivalent result is not shown for k-means, or more generally (k,z)-clustering. While it is likely that the analysis would follow, using recent results on the primal-dual method for k-means, our result gives the first formal proof for a fast constant-factor approximation to k-means in graphs.

As any set of points in Euclidean space can be turned into a sparse graph in time $O\left(n^{1+1/c+o(1)} d\right)$ while preserving the pairwise distance up to a factor $O(c)$ (using spanners [31]), a constant-factor approximation for Euclidean inputs running in time $O\left(n^{1+1/c+o(1)} d\right)$ follows from Thorup's result. Again, this formally holds for k-median, but not k-means.

However, in the own words of Thorup, this algorithm "is rather complicated and hence unlikely to be of direct practical relevance." Our contribution here is to present a different algorithm, greedy and much simpler than the one of [42]. In addition, our algorithms works for the incremental version of the problems as well.

In the realm of linear time, we are not aware of any sub-polynomial approximation algorithm for Euclidean (k, z)-clustering: only the algorithm from [11] runs in near-linear time but yields a $O(k^4)$-approximation. The paper [26] claims that an algorithm from [22] would produce a $O(\text{polylog}\, n)$ approximation in near-linear time: however, their proof is not correct. More precisely, [22] present a near linear-time algorithm with no approximation guarantee: to achieve a polylog(n)-approximation ratio, they need a rejection-sampling step that adds a $n^{1+o(1)}$ running-time – as opposed to our near linear $n\,\text{polylog}(n)$. The other algorithm indeed runs in near-linear time, but does not have any approximation guarantee: indeed, their multi-tree embedding is shown to preserve squared distances, in expectation, up to a polylog factor. However, this embedding does not yield a metric space (in particular, it does not respect triangle inequality[2]): we are not aware of any approximation algorithm in this setting – in particular, the k-means++ algorithm employed by [22] does not.

Our near linear-time approximation algorithm is therefore new, and, importantly, corrects the claim made in [26].

Comparison with subsequent work. Jiang, Jin, Lou and Lu [32] took inspiration from an initial pre-print of our paper to show how to implement fast local search, in spaces that admit sparse spanners: they get a $O(c)$-approximation for (k, z)-clustering in time $n^{1+1/c+o(1)}$. Hence, they improve our tradeoff between run-time and approximation ratio, and achieve the same result as combining spanners and Thorup's algorithm. Compared to the latter, their local search procedure has the advantage being much simpler, and their proof directly generalizes to k-means. Our initial pre-print only mentioned results in Euclidean space, while [32] applies to any metric with sparse spanners. We concurrently generalized our techniques to graphs – hence, to any metric that admits efficiently computable sparse spanners.

We note that this paper explicitly cites our work as a motivation: we present a poly(c)-approximation in time $n^{1+1/c+o(1)}$, and their goal was to bring the approximation-ratio to $O(c)$, closer to the "LSH tradeoff". In addition, we note that their algorithm fails to get any near-linear time run-time, as they deeply rely on LSH.

Another preprint appeared on arxiv [25] after the initial release of the present article. They claim a running time $\tilde{O}(nd)$: however, in their proof, they have a total running of $\Omega(k^4)$; hence, the claim of near-linear running time is only valid under the assumption $n \geq k^4$ (see their proof of Theorem 3.23).

1.2 Further related work

Specific to Euclidean space. The k-median and k-means problems are NP-hard even when the input is in the Euclidean plane $\mathbb{R}^2$ [38, 36]. However, in low-dimensional spaces, it is possible to compute a $(1 + \varepsilon)$-approximation, for any $\varepsilon > 0$, in time $f(\varepsilon, d)\tilde{O}(n)$ [17]. If the target running-time is polynomial in the dimension d, the problems becomes NP-hard to approximate: within a factor of 1.015 for k-median and 1.06 for k-means [20]. The best

[2] The multi-tree embedding consists of the following: build several quadtrees, and define the distance between two points to be the minimum distance accross all quadtrees. This minimum may not respect triangle inequality. As an illustration, consider three points on the line, a at $x = 0$, b at $x = 2$ and c at $x = 4$. Consider two quadtrees, one with a split at $x = 1$ (making all points with $x < 1$ very far from all points with $x > 1$, say distance 10, while points with $x < 1$ or $x > 1$ are at their exact distance), the other one with a split at $x = 3$. In both trees, a and c are at distance 10, hence they are at distance 10 in the multi-tree. However, in the first one b and c are at distance 2, and in the second a and b are at distance 2: hence, with triangle inequality a and c should be at distance 4.

approximation ratios are $1+\sqrt{2}$ for k-median and 5.912 for k-means, based on a primal-dual algorithm running in large polynomial time [16]. For faster and practical algorithms, [22] improves the running time of k-means++ to almost linear, while roughly preserving the approximation guarantee, and [35] that improves the approximation guarantee to $O(1)$ albeit with a running time of $O(nkd)$. For linear-time algorithm, the one from [11] stays the best we know of. Embedding the input metric into a quadtree yields an expected distortion on distances of $\mathrm{poly}(d)\log\Delta$: hence, combined with dimension-reduction to turn d into $\log k$, with reduction of the diameter to $\mathrm{poly}(n)$, one can easily compute a $\mathrm{polylog}(k)\log n$-approximation to k-median. However, this does not work for k-means as the quadtree does not preserve squared distances (see e.g. [17]).

Several sketching techniques are applicable to clustering in Euclidean space: it is possible to reduce the dimension to $O(\varepsilon^{-2}\log k)$ in near-linear time $\tilde{O}(nd)$, while preserving the cost of any clustering up to a multiplicative $(1\pm\varepsilon)$ factor. It is also possible to build coresets in time $\tilde{O}(ndk^{o(1)}+n\log\log\Delta)$, which reduces the number of distinct points to $O(k\varepsilon^{-2-z})$ (see [26] for the specific running time,[3] which uses the coreset algorithms from [23, 21]). Combining those techniques with e.g. [35], it is possible to compute an $O(1)$-approximation to (k,z)-clustering in time $\tilde{O}(ndk^{o(1)}+k^2)$.

General metric spaces. Beyond Euclidean space, k-median is NP-hard to approximate within a factor of $1+2/e$ and k-means within $1+8/e$ [29].

For the incremental version of k-median, the best known approximation ratio is 7.656 for general metric spaces [12] and 7.076 for Euclidean spaces [41]. The approximation ratio cannot be better than 2.01 [12].

Last, Mettu and Plaxton [39] introduced an algorithm related to recursive greedy, for the Facility Location problem. In this algorithm as well they weight balls with a value: [6] showed how to use a value proportional to the number of points in the ball, to estimate the optimal cost of Facility Location problem in a streaming setting.

2 Preliminaries

The (k,z)-clustering problem is defined as follows: the input is a metric space (P,dist) with $|P|=n$, an integer k, and a $z\geq 1$. The goal is to find a set of k points $S\subseteq P$ that minimizes $\textsc{cost}(P,S):=\sum_{x\in P}\mathrm{dist}(x,S)^z$, where $\mathrm{dist}(x,S):=\min_{s\in S}\mathrm{dist}(x,s)$. We say that a set of k points C_k is an α-approximation to (k,z)-clustering when $\textsc{cost}(P,C_k)\leq \alpha\cdot\min_{S,|S|=k}\textsc{cost}(P,S)$.

A list of n points $c_1,...,c_n$ is an α-approximation to the incremental (k,z)-clustering problem on input P when for any $k=1,...,n$, the prefix $c_1,...,c_k$ is an α-approximation to (k,z)-clustering on P.

We assume without loss of generality that the smallest pairwise distance between points of P is 1, and we let Δ be an upper bound on the diameter of the input P (i.e., the largest pairwise distance).

We will consider different metric spaces. The first type is metric spaces induced by a positively weighted connected graph. In this setting, P is a subset of the vertices of a graph weighted $G=(V,E,w)$, where $w\colon E\to\mathbb{N}\setminus\{0\}$, $|V|=n$, and $|E|=m$. The distance

[3] As we noted previously, this paper uses polylogarithmic approximation in near-linear time, which was actually not known; we state their run-time when using an almost-linear time algorithm instead.

$\mathrm{dist}(u, v)$ between two vertices $u, v \in V$ is defined as the length of the shortest (weighted) path in G. The second metric is when P is a multiset of points in $\mathbb{R}^d$. In this case, dist is the usual Euclidean distance.

In our algorithm, we will often use a subroutine to compute the size of a union of sets.

▶ **Lemma 3.** *Given a set of items P, a collections $S_1, ..., S_m$ of subset of P, and a collection of queries $Q_1, ..., Q_t \subseteq \{1, ..., m\}$, there is an algorithm with running-time $O\left((\sum_i^m |S_i| + \sum_i^t |Q_i|) \cdot \log t\right)$ that is able to compute, with probability $1 - 1/t^2$, an estimate for any i of $|\cup_{j \in Q_i} S_j|$ correct up to a factor 3.*

Proof. We rely on the sketching technique introduced by [28, 1]. They show that there is a function $r : \mathbb{R}^d \to \mathbb{R}$ such that, for any fixed set U, $|U|$ is well approximated by $2^{Y_U} := 2^{\max_{u \in U} r(u)}$. Formally, with probability 2/3, it holds that $\frac{1}{3} \le \frac{|U|}{2^{Y_U}} \le 3$. (See Proposition 2.3 in [1]). The running time to compute the function r is the time to evaluate a pairwise independent hash function, e.g. $O(1)$.

Our algorithm therefore computes, for each S_j, the value $S_j := \max_{p \in S_j} r(p)$ in times $O(\sum |S_j|)$. For any Q_i, it holds that $Y_{Q_i} := \max_{j \in Q_i} Y_j$ satisfies with probability 2/3 that

$$\frac{1}{3} \le \frac{|\cup_{j \in Q_i} S_j|}{2^{Y_{Q_i}}} \le 3$$

Computing each Y_{Q_i} takes time $O(\sum |Q_i|)$. Therefore, it only remains to boost the probability to ensure the guarantee holds for all Q_i simultaneously: for this, we run $3\log(t)$ many copies of the algorithm and let $\mathrm{Count}(Q_i)$ be the median of those estimates. A standard argument shows that, with probability $1 - 1/t^2$, it holds for all Q_i that $\frac{1}{3} \le \frac{|\cup_{j \in Q_i} S_j|}{\mathrm{Count}(Q_i)} \le 3$, which implies the lemma.

The overall running time is $O\left((\sum |S_j| + \sum |Q_i|) \log(t)\right)$. ◀

3 The Greedy Algorithm

3.1 Description of the original algorithm

In what follows, we assume that $z \ge 1$ is fixed. We start by presenting the original algorithm and definitions from Mettu and Plaxton [39].

- Given a ball $B = B(x, r) := \{y \in P, \mathrm{dist}(x, y) \le r\}$, the *value* of B is $\mathrm{Value}_{\mathrm{MP}}(B) := \sum_{y \in B} (r - \mathrm{dist}(x, y))^z$.
- A *child* of a ball $B(x, r)$ is any ball $B(y, r/2)$, where $y \in P$ and $\mathrm{dist}(x, y) \le 10r$.
- For any point $x \in P$ and a set of centers C, let $\mathrm{isolated}(x, C)$ denote the ball $B(x, \mathrm{dist}(x, C)/100)$ if C is not empty; and $B(x, \max_{y \in P} d(x, y))$ if $C = \emptyset$. Intuitively, this corresponds to very large ball centered at x that is far away from any center of C.[4]

The algorithm is a recursive greedy procedure, that starts with $C = \emptyset$ and repeats n times the following steps: start with the ball $\mathrm{isolated}(x, C)$ with maximum value over all $x \in P$ (with ties broken arbitrarily), and as long as this ball has more than one child (i.e. as long that there are at least two distinct points of the input P "close" to the ball) replace it with the child with maximum value. Let x be the center of the last chosen ball: add x to C, and repeat – see Algorithm 1 for a pseudo-code.

[4] In those definitions, we chose the scalar constants $2, 10, 100$ for convenience: the whole analysis can be parameterized more carefully in order to optimize the approximation ratio. We opted for simplicity.

Algorithm 1 Recursive Greedy.

```
Let $C_0 = \emptyset$
for $i$ from 1 to $n$ do
    Let $B$ be a maximum value ball in $\{\text{isolated}(x, C_i) | x \in P \setminus C_i\}$
    while $B$ contains more than one point do
        Replace $B$ by a maximum value child of $B$.
    end while
    $C_i = C_{i-1} \cup \{c_i\}$, where $c_i$ is the center of $B$.
end for
```

▶ **Theorem 4** ([39]). *For any fixed z and for all k, the cost of C_k is a $O(1)$-approximation of the optimal (k, z)-clustering cost.*

Mettu and Plaxton prove that this algorithm can be implemented in $O(n^2)$ time, which is linear with respect to the input size (when the metric space is given as a full matrix of pairwise distances). We modify this algorithm to achieve a fast implementation when the metric is described as a sparse graph or a Euclidean space. The general idea is that the recursive greedy algorithm still achieves an $O(1)$-approximation even all quantities are approximated: the value function, child sets, and the sets of isolated balls need not be exactly computed.

3.2 Simplification and extension

The main source of conceptual difficulty of Algorithm 1 is the notion of value, which is not easy to grasp intuitively: the algorithm recurses down to denser region, but the notion of denser it uses is not the most natural. Our first contribution is to show that the value can be replaced essentially by the number of points inside the ball, times the radius of the ball. With this definition of value, the interpretation of the algorithm is more straightforward: the algorithm recurses down to the children containing most points.

In order to simplify the application of the algorithm, we show in addition that the number of points can be approximated, and that the children of a ball can be computed approximately as well. We introduce a parameter $c \geq 5$, which governs the trade-off between run-time and approximation ratio.

Simplifying the value function. Instead of $\text{Value}_{\text{MP}}(B(x, r)) = \sum_{y \in B(x,r)} (r - \text{dist}(x, y))^z$, we will use a function Value approximating $r^z \cdot |B(x, r)|$ as follows: first, count the number of points inside the ball $B(x, r)$ up to a factor 3, and then multiply by r^z. Formally, we show that it is enough to use a function Value that satisfies

$$\forall x \in P, \; r^z/3 \cdot |B(x, r)| \leq \text{Value}(B(x, r)) \leq 3r^z \cdot |B(x, c \cdot r)|.$$

This is our key new insight, that allows for fast implementation: it is indeed much easier to approximately count the number of points in a ball than to evaluate Value_{MP}.

Approximating balls. In addition to this key simplification, we allow for some approximations in the computation of the different ball considered by the algorithm.

Allowing approximation of balls to redefine children. We say that $N(x, r)$ is a c-approximate ball of x at radius r if it satisfies $B(x, r) \subseteq N(x, r) \subseteq B(x, c \cdot r)$. When c is fixed, we simply say that $N(x, r)$ is an approximate ball. The algorithm uses this notion instead of the "child" used in Algorithm 1: it considers all balls (of radius $r/2c$) that are centered at points that are in an approximate ball of x (of radius r).

Forbidding balls. To select the starting ball at the beginning of an iteration, we move away from the notion of isolated. Instead, our algorithm maintain a set of available balls. Initially, those are all balls; when center c_j is placed at the end of j-th iteration, our algorithm *forbids* (i.e., remove them from the set of available balls) all balls that are too close to c_j. More precisely, the algorithm computes, for all r powers of $2c$ such that $1 \leq r \leq \Delta$, an approximate ball $N(c_j, 100c^4 \cdot r)$. It removes from the set of available balls all balls of the form $B(p, r)$ with $p \in N(c_j, 100c^4 \cdot r)$. There are $O(n \log \Delta)$ such balls.

Reducing the number of balls. Perhaps not surprisingly, the radius of balls can be rounded, in order to limit the number of distinct balls to consider. For this, we assume for simplicity that the diameter Δ is a power of $2c$. The algorithm will consider only balls of the form $B(x, r)$, where $x \in P$ is an input point and r is a power of $2c$, such that[5] $1/(2c)^7 \leq r \leq \Delta$.

We give the pseudocode of our modified algorithm in Algorithm 2.

Algorithm 2 Simplified recursive greedy.

Input: A metric space (P, dist) and a number of clusters k.
Output: a set of C of at most k centers.

Define the set of *available* balls to be $\left\{B(x, \Delta/(2c)^\ell), x \in P, l \in \{0, \ldots, \log_{2c}(\Delta) + 7\}\right\}$.
Compute Value$(B(x, r))$ for all the available balls.
for i from 1 to k **do**
  if The set of available ball is empty[6] **then**
    output the solution $C_{i-1} = \{c_1, \ldots, c_{i-1}\}$.
  end if
  Let $B = B(x, r)$ be an available ball with largest Value
  while $r > 1/(2c)^7$ **do** ▷ *Center Selection loop*
    Compute an approximate ball $N = N(x, 10c \cdot r)$
    Update x and r: select $x \in \arg\max_{y \in N} \text{Value}\left(B\left(y, \frac{r}{2c}\right)\right)$ and $r \leftarrow \frac{r}{2c}$
  end while
  $c_i \leftarrow x$
  for all radius $r \in \left\{\Delta/(2c)^\ell, \ell \in \{0, ..., \log_{2c}(\Delta) + 7\}\right\}$ **do** ▷ *Forbidding loop*
    for $x \in N\left(c_i, 100c^4 \cdot r\right)$ **do**
      Remove $B(x, r)$ from the set of available balls.
    end for
  end for
end for
Output the solution $C_k = \{c_1, \ldots, c_k\}$.

3.3 Analysis

To clarify the analysis, we stop the algorithm after k iterations and prove that the set of centers C_k is a poly(c)-approximation of the optimal (k, z)-clustering. However, the algorithm does not depend on k, and therefore the set of centers after k' iterations for $k' \leq k$ is also a poly(c)-approximation of the optimal (k', z)-clustering. In particular, if we modify the algorithm to stop after n iterations instead, it provides a poly(c)-approximation of the *incremental* (k, z)-clustering problem.

[5] Although points are at distance at least 1, it is important for our algorithm to consider balls with smaller radius $1/(2c)^7$ in order to be sure that, around any point, there is a ball available unless there is a center at the point.

[6] In that case, as explained in the footnote 2, i is more than the number of distinct points.

For clarity, we split the proof of Theorem 1 into two parts: first the approximation guarantee, then the running time.

▶ **Theorem 5.** *For any k, the set of centers output by the simplified recursive greedy Algorithm 2 gives a* $\mathrm{poly}(c)$*-approximation of the optimal* (k,z)*-clustering solution.*

We provide the full proof of this theorem in the full version of our paper,[7] and focus here only on the running time.

In our applications, computing the values of ball turns out to be easy – since it boils down to estimating the number of points in balls, which is well studied. Hence, the main remaining task when applying this algorithm to a specific metric space is to bound the running time of the Forbidding loop and of the Center Selection loop.

▶ **Theorem 6.** *Let $(P, dist)$ be a metric space. Suppose there is an algorithm that computes all values in time T_{Value} and a datastructure with preprocessing time T_{Pre} and that is able to, for any fixed r: remove any point from the ground set in time T_{rm}, and, given x, computes an approximated ball $N(x,r)$ of the current ground set in time $T_N \cdot |N(x,r)|$.*

Then the running time of Algorithm 1 is bounded by

$$T_{\text{Value}} + O\left(\log \Delta (T_{Pre} + |P|(T_N + T_{rm}))\right)$$

To prove this theorem, it suffices to bound the complexity of the **Center Selection** loop and the **Forbidding** loop, since by definition the value of all balls can be computed in time T_{Value}.

In the **Center Selection** loop, we will show that each ball is considered at most once when the algorithm selects the ball with the maximum value at line 10.

In the **Forbidding** loop, we use a data structure that maintains $O(\log n)$ copies of the input–one for each radius considered by the algorithm. After a center is selected, for each radius r, we forbid all balls $B(x,r)$ with $x \in N(c_i, 100c^4 \cdot r)$. These points are deleted from the corresponding data structure to prevent them from being reconsidered later in the Forbidding loop when subsequent centers are selected.

3.4 Running time analysis of the Center Selection loop

We now introduce a new definition. We say that a ball B' is a *potential descendant* of a ball B if there exists a sequence of balls $B_0, \dots, B_\ell$, with $B_i = B(x_i, r_i)$, such that $B_0 = B$, $B_\ell = B'$, and for all i, $\mathrm{dist}(x_i, x_{i+1}) \le 10c^2 r_i$ and $r_{i+1} = r_i/2c$.

Note that if two balls B and B' appear in the same center selection loop with B' appearing after B, then B' is a potential descendant of B.

▶ **Fact 7.** *If $B(y, r_y)$ is a potential descendant of $B(x, r_x)$, then $dist(x,y) \le 20c^2 \cdot r_x$.*

Proof. Let $B(y, r_y)$ be a descendant of $B(x, r_x)$, and let $x_0 = x, ..., x_\ell = y$ be the centers of the sequence of balls $(B_0, \dots, B_\ell)$ from the definition of descendant. For every i, we have $\mathrm{dist}(x_i, x_{i+1}) \le \frac{10c^2 \cdot r_x}{(2c)^i}$. By triangle inequality, this implies

$$\mathrm{dist}(x,y) \le \sum_{i=0}^{\ell-1} \frac{10c^2 \cdot r_x}{(2c)^i} \le 20c^2 \cdot r_x. \qquad \blacktriangleleft$$

[7] https://arxiv.org/abs/2407.11217

A consequence of this fact is that any ball appears at most once in an approximate ball of the Center Selection loop:

▶ **Lemma 8.** *For any $x \in P, r \in \mathbb{R}^+$, the ball $B(x,r)$ appears at most once in the center selection loop.*

Proof. We split the proof into two parts: first, if a ball appears in an approximate ball, then it is forbidden at the next Forbidding step. Second, if a ball is available, then all its potential descendants are available. Combined, those two results conclude our lemma.

▶ **Fact 9.** *If a ball $B(x,r_x)$ is a potential descendant of a ball B^j selected at the j-th iteration of the center selection loop, then $B(x,r_x)$ is forbidden during the forbidding procedure after c_j is selected.*

Proof. Let y be the center of the ball B^j, and let its radius be $r_y = 2c \cdot r_x$. Fact 7 ensures that both x and c_j are at distance at most $20c^2 \cdot 2c \cdot r_x$ from y: therefore, $\mathrm{dist}(x,c_j) \leq 40c^3 r_x < 100c^4 r_x$, hence the ball $B(x,r_x)$ is in the set $N(c_j, 100c^4 r_x)$ and is forbidden on line 12 when c_j is selected as a center. ◀

▶ **Fact 10.** *If, at the beginning of an iteration of the loop line 4, a ball $B(x,r_x)$ is available, then all its potential descendants are available.*

Proof. Let $B(x,r_x)$ be any ball, and let $B(y,r_y)$ be a potential descendant of $B(x,r_x)$. Suppose that $B(y,r_y)$ is not available at the beginning of an iteration of the loop in line 4. Then there exists a center c_i selected by the algorithm such that $y \in N(c_i, 100c^4 \cdot r_y)$ and therefore $\mathrm{dist}(y,c_i) \leq 100c^5 \cdot r_y$. Because $B(y,r_y)$ is a descendant of $B(x,r_x)$, we know by Fact 7 that $\mathrm{dist}(x,y) \leq 20c^2 \cdot r_x$. Using the triangle inequality, we get $\mathrm{dist}(x,c_i) \leq \mathrm{dist}(x,y) + \mathrm{dist}(y,c_i) \leq 20c^2 \cdot r_x + 100c^5 \cdot r_y$. We also know that $r_y \leq r_x/2c$ and therefore $\mathrm{dist}(x,c_i) \leq (50c^4 + 20c^2) \cdot r_x \leq 100c^4 \cdot r_x$. Hence $x \in B(c_i, 100c^4 \cdot r_x) \subset N(c_i, 100c^4 \cdot r_x)$, and $B(x,r_x)$ is also forbidden at the i-th iteration. ◀

Combining those two facts concludes the proof. ◀

Proof of Theorem 6. First, by definition, the running time of computing all values in line 2 of the algorithm takes time T_{Value}.

For analyzing the Forbidding loop, we note the following: instead of computing $N(c_i, 100c^4 r)$ in line 14, it is merely enough to compute the set of balls that have not been forbidden yet in that set. To do so with the datastructure from the theorem statement, one can merely do the following: for each valid r, initialize the datastructure with all points in the ground set. Then, after each computation of an approximate ball $N(c_i, 100c^4 r)$ of the current set, remove all $x \in N(c_i, 100c^4 r)$ from the ground set of the corresponding r in line 15. Hence, the preprocessing time is $T_{Pre} \cdot O(\log \Delta)$, and the total running time is $O((T_N + T_{rm})|P|\log \Delta)$. Indeed, there are $O(\log \Delta)$ different radius r, and at each level any point appears at most once before being removed. The guarantees of the datastructure therefore ensure that, at any given level, the total running time is bounded by $O((T_N + T_{rm})|P|)$.

Last, it remains to analyze the Center Selection loop. For this, we can use the datastructure from the lemma, without removing any point: Lemma 8 ensures that each ball $B(x,r)$ appears only once. Hence, the total running time is again $O\left((T_{Pre} + T_N|P|)\log \Delta\right)$. ◀

4 Implementation in the Euclidean setting

4.1 Near-linear time approximation via multiple quadtrees

Tree embeddings are a common tool for designing approximation algorithms on metric spaces. A particularly useful tree embedding in Euclidean spaces is the *quadtree*; see, for example, [30, 4] for general reference, and [33, 22] for applications to clustering.

In a quadtree, the input space is enclosed in an axis-aligned hypercube that is recursively subdivided into 2^d smaller hypercubes of half the side length. If the entire input is translated by a uniformly random vector in $[0, \Delta]^d$, then the standard analysis of distortion induced by the quadtree (see e.g. [26]) shows that with probability at least 1/2, the smallest quadtree cell containing both p and q has side length at most $4\sqrt{d} \cdot \|p - q\|$. We define $\text{dist}_T(p, q)$ as the diagonal of the smallest quadtree cell containing both p and q.

To boost the probability, one can take $t = O(\log n)$ independent random shifts and construct t quadtrees, with the following guarantees:

- for all quadtree T, and any p, q, $\|p - q\| \le \text{dist}_T(p, q)$
- with probability $1 - 1/n^2$, for any p, q there exist a quadtree T such that $\text{dist}_T(p, q) \le 4d\|p - q\|$.

The construction of a single quadtree takes time $O(nd \log \Delta)$ [22], hence the total construction time is $O(nd \log n \log \Delta)$.

We will use these quadtrees to run Algorithm 2 with parameter $c = 4d$. In order to evaluate the value of all balls of radius r, we compute all quadtree cells of side length $L = 4\sqrt{d} \cdot r$. Now, to compute an approximate ball $N(x, r)$, it is merely enough to take the union of all cells with side length L that contain x. Indeed, with probability 1 all points in those cells are at distance at most $L\sqrt{d}$ of x; and, with probability $1 - 1/n^2$, any point at distance r is in the same cell as x in one of the quadtrees.

Therefore, computing the set $N(x, r)$ can be done in time $O(\log n)|N(x, r)|$, hence $T_N = O(\log n)$ – as there are $O(\log n)$ different quadtrees, the union can be computed with this running time. Removing a point from the datastructure simply takes time $T_{rm} = O(\log n \log \Delta)$, to remove it from the $O(\log \Delta)$ levels of each of the $O(\log n)$ quadtree.

In addition, computing the size of $N(x, r)$ is a direct application of Lemma 3 – and computing $N(x, r)$ for all point $x \in P$ and r power of 2 takes time $T_{\text{Value}} = O(n \log \Delta \log^2 n)$.

Theorem 6 directly implies:

▶ **Corollary 11.** *In Euclidean space, the recursive greedy algorithm can be implemented to provide a* poly(d)*-approximation to incremental* (k, z)*-clustering in time* $O(nd \log n \log \Delta + n \log \Delta \log^2 n)$*.*

Note that, using dimension-reduction, the dimension for (k, z)-clustering can be reduced to $O(\log k)$ [37] in time $O(nd \log d)$. This, with the above corollary, shows the last two items of Corollary 2.

4.2 Constant-factor approximation via Locality-sensitive hashing

The tool we use in the Euclidean setting is Locality-sensitive hashing [2]. The precise result we use is the following:

▶ **Lemma 12** (See section D in [22])**.** *Let* $P \subseteq \mathbb{R}^d$*,* $r \in \mathbb{R}^+$*, and* $\ell = (n/\delta)^{1/c^2}$ *; there is a family of hash functions from* $\mathbb{R}^d$ *to some universe* U *such that, with probability* $1 - \delta$*, if* $f_1, ..., f_\ell$ *are drawn from this family:*

- *For any* $p, q \in P$ *with dist*$(p, q) \ge c \cdot r$*, then for all* $i = 1, ..., \ell$ $f_i(p) \ne f_i(q)$

- *For any $p, q \in P$ with dist$(p,q) \leq r$, then there exists $i \in \{1, ..., \ell\}$ with $f_i(p) = f_i(q)$.*

Furthermore, the hash functions satisfy the following:

- *for any $i, p \in \mathbb{R}^d$, computing $f_i(p)$ takes time $O\left(dn^{o(1)}\right)$,*
- *after preprocessing time $O\left(\ell d \cdot n^{1+o(1)}\right)$, one can compute for any i, p the set $T_i[u] := \{p : f_i(p) = u\}$ in time $O(|T_i[u]|)$.*

We use the previous lemma in two ways: first, it allows us to compute an approximate neighborhood of each point quickly, and second, combined with streaming techniques, to estimate the size of this neighborhood efficiently. We start with the former (where we replaced, for simplicity of notation, the success probability $1-\delta$ with $1-1/n^2$):

▶ **Corollary 13.** *For any $r \in \mathbb{R}^+$ and $P \subseteq \mathbb{R}^d$, there is a datastructure with preprocessing time $T_{pre} = O\left(dn^{1+3/c^2+o(1)}\right)$ that can, with probability $1-1/n^2$:*

- *remove a point from P in time $O\left(n^{3/c^2}\right)$*
- *answer the following query: for any point $p \in P$, compute a set $N(p,r)$ of points of P such that $B(p,r) \cap P \subseteq N(p,r) \subseteq B(p, c \cdot r) \cap P$. The query time is $O\left(n^{3/c^2}|N(p,r)|\right)$.*

Proof. This is a direct application of Lemma 12: given r and $\delta = 1/n^2$, compute $f_i(p)$ for all i and p, in time $O\left(dn^{1+3/c^2+o(1)}\right)$. First, to remove a point p from P, simply remove it from all the tables $T_i[f_i(p)]$ for $i = 1, ..., \ell$: this takes time $O(\ell) = O\left(n^{3/c^2}\right)$.

To answer a query given a point p, compute $T_i[f_i(p)]$ for all i, in time $O(|T_i[f_i(p)]|)$ and define $N(p,r) := \cup_{i=1}^{\ell} T_i[f_i(p)]$. The running time to compute the union is at most $\ell \cdot O(|N(p,r)|) = O(n^{3/c^2}|N(p,r)|)$. The first two bullets of Lemma 12 ensure the desired accuracy guarantee. ◀

Hence, in the vocabulary of Theorem 6, $T_{rm} = T_N = O\left(n^{3/c^2}\right)$. It only remains to compute the values: this can be easily done combining Lemma 12 with the sketching techniques of Lemma 3, as follows:

▶ **Lemma 14.** *Given a radius r, there is an algorithm that runs in time $O\left(dn^{1+3/c^2+o(1)}\right)$ and computes, for all $p \in P$, $\text{Value}(B(p,r))$ such that, with probability $1-1/n^2$, it holds that $\forall p,\ r^z \cdot |B(p,r) \cap P|/3 \leq \text{Value}(B(p,r)) \leq 3r^z \cdot |B(p, c \cdot r) \cap P|$.*

Proof. We show how to compute, for all $p \in P$, an approximate count of the number of points in $B(p,r)$, namely a value $\text{Count}(p,r)$ such that $|B(p,r) \cap P|/3 \leq \text{Count}(p,r) \leq 3r \cdot |B(p, c \cdot r) \cap P|$. Multiplying Count by r^z gives the lemma.

To build the estimates $\text{Count}(p,r)$, the first step of the algorithm is to compute $f_i(p)$, for all $i \in \{1, ..., \ell\}$ and all $p \in P$, using Lemma 12 with r and $\delta = 1/n^2$. This takes time $O\left(dn^{1+3/c^2+o(1)}\right)$. Due to Lemma 12, we have the guarantee that, with probability $1-1/n^2$,

$$|B(p,r) \cap P| \leq \left|\cup_{i=1}^{\ell} T_i[f_i(p)]\right| \leq |B(p, c \cdot r) \cap P|.$$

Therefore, it is merely enough to estimate $\left|\cup_{i=1}^{\ell} T_i[f_i(p)]\right|$ using Lemma 3 (with $S_{i,u} = T_i[u]$, $t = n$, and for all p, $Q_p = \{f_i(p), i = 1, ..., \ell\}$). As each point p is in at most ℓ sets $S_{i,u}$, and each Q_p has size at most ℓ, the running time of the algorithm from Lemma 3 is $O(\ell n)$

Hence, the overall running time is $O\left(dn^{1+3/c^2+o(1)}\right) + O(n\ell \log(n)) = O\left(dn^{1+3/c^2+o(1)}\right)$. ◀

Thus, Theorem 6 implies:

▶ **Corollary 15.** *In Euclidean space, the recursive greedy algorithm can be implemented to provide a* poly(c)*-approximation to incremental* (k, z)*-clustering in time* $O(n^{1+3/c^2+o(1)} d \log \Delta)$*.*

5 Almost-linear time implementation for Graphs

In sparse graphs, Filtser [27] introduced a datastructure very similar to LSH, dubbed *ThProbabilistic Decomposition*: for any $r > 0$, and any $c \geq 1$, there is a distribution over partitions such that:

- for any partition in the support of the distribution, the diameter of any part is bounded by $2c \cdot r$,
- any u, v at distance at most r are in the same part with probability at least $\frac{1}{8} n^{-1/c-1}$.

Furthermore, there is an algorithm to draw a partition according to this distribution in time $O(m \log n)$ (see Theorem 4 in [27]).

To achieve the same guarantee as in Lemma 12, we merely draw $n^{1/c}$ partition at random, and we get the properties that, for any vertices at distance more than $2cr$, they are in different part in all partitions ; and for any vertices at distance less than r, there is one partition where they are in the same part with high probability.

As in Corollary 13, we can use this datastructure to build the one required by Theorem 6 to compute approximate balls.

It therefore just remains to compute the values of all balls. In sparse graphs, Cohen showed how to approximate the number of points in all balls in near-linear time:

▶ **Lemma 16** (Theorem 5.1 in [14])**.** *There exists an algorithm that takes as input a metric induced by an edge-weighted graph* $G = (V, E, w)$ *with* n *vertices and* m *edges. The algorithm has expected preprocessing time* $O(m \log^2 n + n \log^3 n)$ *and allows queries* $\tilde{n}(v, d)$ *for any pair* $(v, d) \in V \times \mathbb{R}^+$ *that estimate the number of points in the ball* $B(v, d)$ *such that:*

- *The expected query time is* $O(\log \log n)$*.*
- *With probability* $1 - O(1/\text{poly}(n))$*, for all* $(v, d) \in V \times \mathbb{R}^+$*,*

$$\frac{\big||B(v,d)| - \tilde{n}(v,d)\big|}{|B(v,d)|} \leq 1/10.$$

Hence, using this result, we can directly compute the values of all the $O(n \log \Delta)$ balls, in time $T_{\text{Value}} = O\left(m \log n^2 + n \log^3 n + n \log \Delta \log \log n\right)$.

References

1 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, 1996. doi:10.1145/237814.237823.

2 Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008. doi:10.1145/1327452.1327494.

3 Alexandr Andoni and Ilya P. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 793–801. ACM, 2015. doi:10.1145/2746539.2746553.

4 Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998. doi:10.1145/290179.290180.

5 David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 1027–1035, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283494.

6 Mihai Bădoiu, Artur Czumaj, Piotr Indyk, and Christian Sohler. Facility location in sublinear time. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, pages 866–877, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11523468_70.

7 Sayan Bhattacharya, Martín Costa, Ermiya Farokhnejad, Shaofeng H. C. Jiang, Yaonan Jin, and Jianing Lou. Fully dynamic euclidean k-means, 2025. doi:10.48550/arXiv.2507.11256.

8 Jaroslaw Byrka, Krzysztof Sornat, and Joachim Spoerhase. Constant-factor approximation for ordered k-median. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 620–631. ACM, 2018. doi:10.1145/3188745.3188930.

9 Moses Charikar, Vincent Cohen-Addad, Ruiquan Gao, Fabrizio Grandoni, Euiwoong Lee, and Ernest van Wijland. An improved greedy approximation for (metric) k-means. In *66th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2025, Sydney, Australia, December 14-17, 2025*, pages 233–240. IEEE, 2025. doi:10.1109/FOCS63196.2025.00016.

10 Moses Charikar, Vincent Cohen-Addad, Ruiquan Gao, Fabrizio Grandoni, Euiwoong Lee, and Ernest van Wijland. A $(4+\varepsilon)$-approximation for euclidean k-means via non-monotone dual-fitting. In *Proceedings of the 58th Annual ACM Symposium on Theory of Computing (STOC'26, June 22–26, 2026, Salt Lake City, UT, USA*. ACM, 2026. doi:10.1145/3798129.3800894.

11 Moses Charikar, Monika Henzinger, Lunjia Hu, Maximilian Vötsch, and Erik Waingarten. Simple, scalable and effective clustering via one-dimensional projections. *Advances in Neural Information Processing Systems*, 36:64618–64649, 2023.

12 Marek Chrobak and Mathilde Hurand. Better bounds for incremental medians. *Theoretical Computer Science*, 412(7):594–601, 2011. Selected papers from WAOA 2007: Fifth Workshop on Approximation and Online Algorithms. doi:10.1016/j.tcs.2009.07.006.

13 Marek Chrobak, Claire Kenyon, and Neal E. Young. The reverse greedy algorithm for the metric k-median problem. *Inf. Process. Lett.*, 97(2):68–72, 2006. doi:10.1016/J.IPL.2005.09.009.

14 Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997. doi:10.1006/jcss.1997.1534.

15 Vincent Cohen-Addad. A fast approximation scheme for low-dimensional k-means. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 430–440. SIAM, 2018. doi:10.1137/1.9781611975031.29.

16 Vincent Cohen-Addad, Hossein Esfandiari, Vahab S. Mirrokni, and Shyam Narayanan. Improved approximations for euclidean k-means and k-median, via nested quasi-independent sets. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1621–1628. ACM, 2022. doi:10.1145/3519935.3520011.

17 Vincent Cohen-Addad, Andreas Emil Feldmann, and David Saulpic. Near-linear time approximation schemes for clustering in doubling metrics. In *J. ACM*, volume 68, 2021. doi:10.1145/3477541.

18 Vincent Cohen-Addad, Fabrizio Grandoni, Euiwoong Lee, Chris Schwiegelshohn, and Ola Svensson. A $(2+\varepsilon)$-approximation algorithm for metric k-median. *To appear at STOC*, 2025. doi:10.48550/arXiv.2503.10972.

19 Vincent Cohen-Addad, Anupam Gupta, Lunjia Hu, Hoon Oh, and David Saulpic. An improved local search algorithm for k-median. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1556–1612. SIAM, 2022. doi:10.1137/1.9781611977073.65.

20 Vincent Cohen-Addad, Karthik C. S., and Euiwoong Lee. Johnson coverage hypothesis: Inapproximability of k-means and k-median in ℓ_p-metrics. In *Symposium on Discrete Algorithms, SODA*, pages 1493–1530, 2022. doi:10.1137/1.9781611977073.63.

21 Vincent Cohen-Addad, Kasper Green Larsen, David Saulpic, and Chris Schwiegelshohn. Towards optimal lower bounds for k-median and k-means coresets. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1038–1051. ACM, 2022. doi:10.1145/3519935.3519946.

22 Vincent Cohen-Addad, Silvio Lattanzi, Ashkan Norouzi-Fard, Christian Sohler, and Ola Svensson. Fast and accurate k-means++ via rejection sampling. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/babcff88f8be8c4795bd6f0f8cccca61-Abstract.html.

23 Vincent Cohen-Addad, David Saulpic, and Chris Schwiegelshohn. A new coreset framework for clustering. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 169–182. ACM, 2021. doi:10.1145/3406325.3451022.

24 Vincent Cohen-Addad and Chris Schwiegelshohn. On the local structure of stable clustering instances. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 49–60. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.14.

25 Vincent Cohen-Addad, Liudeng Wang, David P. Woodruff, and Samson Zhou. Fast, space-optimal streaming algorithms for clustering and subspace embeddings. *arXiv prepublication*, abs/2504.16229, 2025. doi:10.48550/arXiv.2504.16229.

26 Andrew Draganov, David Saulpic, and Chris Schwiegelshohn. Settling time vs. accuracy tradeoffs for clustering big data. *SIGMOD 2024*, 2024.

27 Arnold Filtser. On strong diameter padded decompositions. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, volume 145 of *LIPIcs*, pages 6:1–6:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.6.

28 Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985. doi:10.1016/0022-0000(85)90041-8.

29 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *J. Algorithms*, 31(1):228–248, 1999. doi:10.1006/jagm.1998.0993.

30 Sariel Har-Peled. *Geometric approximation algorithms*. American Mathematical Soc., 2011.

31 Sariel Har-Peled, Piotr Indyk, and Anastasios Sidiropoulos. Euclidean spanners in high dimensions. 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 804–809. SIAM, 2013. doi:10.1137/1.9781611973105.57.

32 Shaofeng H. C. Jiang, Yaonan Jin, Jianing Lou, and Pinyan Lu. Local search for clustering in almost-linear time, 2025. To appear at SODA 2026. doi:10.48550/arXiv.2504.03513.

33 Stavros G. Kolliopoulos and Satish Rao. A nearly linear-time approximation scheme for the euclidean k-median problem. *SIAM J. Comput.*, 37(3):757–782, 2007. doi:10.1137/S0097539702404055.

34 Ravishankar Krishnaswamy, Shi Li, and Sai Sandeep. Constant approximation for k-median and k-means with outliers via iterative rounding. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 646–659. ACM, 2018. doi:10.1145/3188745.3188882.

35 Silvio Lattanzi and Christian Sohler. A better k-means++ algorithm via local search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3662–3671. PMLR, 2019. URL: http://proceedings.mlr.press/v97/lattanzi19a.html.

36 Meena Mahajan, Prajakta Nimbhorkar, and Kasturi R. Varadarajan. The planar k-means problem is np-hard. *Theor. Comput. Sci.*, 442:13–21, 2012. doi:10.1016/J.TCS.2010.05.034.

37 Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Performance of johnson-lindenstrauss transform for k-means and k-medians clustering. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1027–1038, 2019. doi:10.1145/3313276.3316350.

38 Nimrod Megiddo and Kenneth J Supowit. On the complexity of some common geometric location problems. *SIAM journal on computing*, 13(1):182–196, 1984. doi:10.1137/0213014.

39 Ramgopal R. Mettu and C. Greg Plaxton. The online median problem. *SIAM Journal on Computing*, 32(3):816–832, 2003. doi:10.1137/S0097539701383443.

40 Aviad Rubinstein. Hardness of approximate nearest neighbor search. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1260–1268. ACM, 2018. doi:10.1145/3188745.3188916.

41 Vladimir Shenmaier. An approximation algorithm for the euclidean incremental median problem. *Discrete Optimization*, 22:312–327, 2016. doi:10.1016/j.disopt.2016.08.005.

42 Mikkel Thorup. Quick k-median, k-center, and facility location for sparse graphs. *SIAM J. Comput.*, 34(2):405–432, 2004. doi:10.1137/S0097539701388884.