Approximate Range Queries for Clustering

Eunjin Oh
Max Planck Institute for Informatics
Saarbrücken, Germany
eoh@mpi-inf.mpg.de
https://orcid.org/0000-0003-0798-2580

Hee-Kap Ahn
Pohang University of Science and Technology
Pohang, Korea
heekap@postech.ac.kr
https://orcid.org/0000-0001-7177-1679

Abstract

We study the approximate range searching for three variants of the clustering problem with a set P of n points in d-dimensional Euclidean space and axis-parallel rectangular range queries: the k-*median*, k-*means*, and k-*center range-clustering query problems*. We present data structures and query algorithms that compute $(1+\varepsilon)$-approximations to the optimal clusterings of $P \cap Q$ efficiently for a query consisting of an orthogonal range Q, an integer k, and a value $\varepsilon > 0$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Approximate clustering, orthogonal range queries

Digital Object Identifier 10.4230/LIPIcs.SoCG.2018.62

Related Version A full version of this paper is available at `http://arxiv.org/abs/1803.03978`.

Funding This research was supported by NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea, and the MSIT (Ministry of Science and ICT), Korea, under the SW Starlab support program (IITP-2017-0-00905) supervised by the IITP (Institute for Information & communications Technology Promotion).

1 Introduction

Range searching asks to preprocess a set of objects and to build a data structure so that all objects intersecting a given query range can be reported quickly. There are classical variants of this problem such as computing the number of objects intersecting a query range, checking whether an object intersects a query range, and finding the closest pair of objects intersecting a query range. It is a widely used technique in computer science with numerous applications in geographic information systems, computer vision, machine learning, and data mining. These range searching problems have been studied extensively in computational geometry over the last decades. For more information on range searching, refer to the surveys [2, 20].

However, there are a large number of objects intersecting a query range in many real-world applications, and thus it takes long time to report all of them. Thus one might want to obtain a property of the set of such objects instead of obtaining all such objects. Queries of this kind are called *range-analysis queries*. More formally, the goal of this problem is to preprocess a set P of objects with respect to a fixed range-analysis function f and to build a data structure so that $f(P \cap Q)$ can be computed efficiently for any query range Q. These query problems have been studied extensively under various range-analysis functions such as the diameter or width of a point set [8] and the length of the minimum spanning tree of a point set [6]. Note that the classical variants mentioned above are also range-analysis query problems. A clustering cost can also be considered as a range-analysis function.

Clustering is a fundamental research topic in computer science and arises in various applications [19], including pattern recognition and classification, data mining, image analysis, and machine learning. In clustering, the objective is to group a set of data points into clusters so that the points from the same cluster are similar to each other and points from different clusters are dissimilar. Usually, input points are in a high-dimensional space and the similarity is defined using a distance measure. There are a number of variants of the clustering problem in the geometric setting depending on the similarity measure such as the k-median, k-means, and k-center clustering problems.

In this paper, we study the approximate range-analysis query problems for three variants of the clustering with a set P of n points in d-dimensional Euclidean space with $d \geq 2$ and axis-parallel rectangular range queries: the *k-median*, *k-means*, and *k-center range-clustering query problems*. The approximate k-median, k-means and k-center range-clustering query problems are defined as follows: Preprocess P so that given a query range Q, an integer k with $1 \leq k \leq n$ and a value $\varepsilon > 0$ as a query, an $(1+\varepsilon)$-approximation to the k-median, k-means, and k-center clusterings of $P \cap Q$ can be computed efficiently. Our desired query time is polynomial to $\log n$, k and $(1/\varepsilon)$.

Previous work. The k-median and k-means clustering problems have been studied extensively and there are algorithms achieving good approximation factors and polynomial running times to the problem. Har-Peled and Mazumdar [18] presented an $(1+\varepsilon)$-approximation algorithm for the k-means and k-median clustering using coresets for points in d-dimensional Euclidean space. Their algorithm constructs a (k, ε)-coreset with property that for any arbitrary set C of k centers, the clustering cost on the coreset with respect to C is within $(1 \pm \varepsilon)$ times the clustering cost on the original input points with respect to C. Then it computes the clusterings for the coreset using a known weighted clustering algorithm. Later, a number of algorithms for computing smaller coresets for the k-median and k-means clusterings have been presented [9, 13, 17].

The k-center clustering problem has also been studied extensively. It is NP-Hard to approximate the 2-dimensional k-center problem within a factor of less than 2 even under the L_∞-metric [12]. A 2-approximation to the k-center can be computed in $O(kn)$ time for any metric space [12], and in $O(n \log k)$ time for any L_p-metric space [14]. The exact k-center clustering can be computed in $n^{O(k^{1-1/d})}$ time in d-dimensional space under any L_p-metric [4]. This algorithm combined with a technique for grids yields an $(1+\varepsilon)$-approximation algorithm for the k-center problem that takes $O(n \log k + (k/\varepsilon)^{O(k^{1-1/d})})$ time for any L_p-metric [4]. Notice that all these algorithms are *single-shot* algorithms, that is, they compute a clustering of given points (without queries) just once.

There have been some results on range-analysis query problems related to clustering queries. Brass et al. [8] presented data structures of finding extent measures: the width, area, or perimeter of the convex hull of $P \cap Q$. Arya et al. [6] studied data structures that support clustering queries on the length of the minimum spanning tree of $P \cap Q$. Various types of range-aggregate nearest neighbor queries have also been studied [23, 24].

Nekrich and Smid [22] considered approximate range-aggregate queries such as the diameter or radius of the smallest enclosing ball for points in d-dimensional space. Basically, their algorithm constructs a d-dimensional range tree as a data structure, in which each node stores a δ-coreset of points in its subtree (but not explicitly), and applies range-searching query algorithms on the tree, where δ is a positive value. Their algorithm works for any aggregate function that can be approximated using a decomposable coreset including coresets for the k-median, k-means and k-center clusterings. In this case, the size of the data structure is $O(kn\log^d n/\delta^2)$, and the query algorithm computes a (k,δ)-coreset of size $O(k\log^{d-1} n/\delta^2)$. However, their algorithm uses k and δ in constructing the data structure for the clusterings, and therefore k and δ are fixed over range-clustering queries.

Very recently, Abrahamsen et al. [1] considered k-center range-clustering queries for n points in d-dimensional space. They presented a method, for a query consisting of a range Q, an integer k with $1 \le k \le n$ and a value $\varepsilon > 0$, of computing a (k,ε)-coreset S from $P \cap Q$ of size $O(k/\varepsilon^d)$ in $O(k(\log n/\varepsilon)^{d-1} + k/\varepsilon^d)$ time such that the k-center of S is an $(1+\varepsilon)$-approximation to the k-center of $P \cap Q$. After computing the coreset, they compute an $(1+\varepsilon)$-approximation to the k-center of the coreset using a known single-shot algorithm. Their data structure is of size $O(n\log^{d-1} n)$ and its query algorithm computes an $(1+\varepsilon)$-approximate to a k-center range-clustering query in $O(k(\log n/\varepsilon)^{d-1} + T_{\mathrm{ss}}(k/\varepsilon^d))$ time, where $T_{\mathrm{ss}}(N)$ denotes the running time of an $(1+\varepsilon)$-approximation single-shot algorithm for the k-center problem on N points.

The problem of computing the diameter of input points contained in a query range can be considered as a special case of the range-clustering problem. Gupta et al. [15] considered this problem in the plane and presented two data structures. One requires $O(n\log^2 n)$ size that supports queries with arbitrary approximation factors $1+\varepsilon$ in $O(\log n/\sqrt{\varepsilon} + \log^3 n)$ query time and the other requires a smaller size $O(n\log n/\sqrt{\delta})$ that supports only queries with the *fixed* approximation factor $1+\delta$ with $0 < \delta < 1$ that is used for constructing the data structure. The query time for the second data structure is $O(\log^3 n/\sqrt{\delta})$. Nekrich and Smid [22] presented a data structure for this problem in a higher dimensional space that has size $O(n\log^d n)$ and supports diameter queries with the fixed approximation factor $1+\delta$ in $O(\log^{d-1} n/\delta^{d-1})$ query time. Here, δ is an approximation factor given for the construction of their data structure, and therefore it is fixed for queries to the data structure.

Our results. We present algorithms for k-median, k-means, and k-center range-clustering queries with query times polynomial to $\log n$, k and $1/\varepsilon$. These algorithms have a similar structure: they compute a (k,ε)-coreset of input points contained in a query range, and then compute a clustering on the coreset using a known clustering algorithm. We use $T_{\mathrm{ss}}(N)$ to denotes the running time for any $(1+\varepsilon)$-approximation single-shot algorithm of each problem on N points. For a set P of n points in $\mathbb{R}^d$, we present the following results.

- There is a data structure of size $O(n\log^d n)$ such that an $(1+\varepsilon)$-approximation to the k-median or k-means clustering of $P \cap Q$ can be computed in time

  $$O(k^5\log^9 n + k\log^d n/\varepsilon + T_{\mathrm{ss}}(k\log n/\varepsilon^d))$$

  for any box Q, any integer k with $1 \le k \le n$, and any value $\varepsilon > 0$ given as a query.
- There is a data structure of size $O(n\log^{d-1} n)$ such that an $(1+\varepsilon)$-approximation to the k-center clustering of $P \cap Q$ can be computed in time

  $$O(k\log^{d-1} n + k\log n/\varepsilon^{d-1} + T_{\mathrm{ss}}(k/\varepsilon^d))$$

  for any box Q, an integer k with $1 \le k \le n$, and a value $\varepsilon > 0$ given as a query.
- There is a data structure of size $O(n\log^{d-1} n)$ such that an $(1+\varepsilon)$-approximation to the diameter (or radius) of $P \cap Q$ can be computed in time

  $$O(\log^{d-1} n + \log n/\varepsilon^{d-1})$$

  for any box Q and a value $\varepsilon > 0$ given as a query.

Our results are obtained by combining range searching with coresets. The k-median and k-means range-clusterings have not been studied before, except the work by Nekrich and Smid. They presented a general method to compute approximate range-aggregate queries, which can be used for the k-median or k-means range-clustering. Here, the approximation factor is required to be given in the construction of their data structure as mentioned above. However, it is not clear how to use or make their data structure to support approximate range-clustering queries with various approximation factors we consider in this paper. Indeed, the full version of the paper by Abrahamsen et al. [1] poses as an open problem a data structure supporting $(1+\varepsilon)$-approximate k-median or k-means range-clustering queries with various values ε. Our first result answers to the question and presents a data structure for the k-median and k-means range-clustering problems for any value ε.

Our second result improves the best known previous work by Abrahamsen et al. [1]. Recall that the query algorithm by Abrahamsen et al. takes $O(k(\log n/\varepsilon)^{d-1} + T_{\text{ss}}(k/\varepsilon^d))$ time. We improved the first term of their running time by a factor of $\min\{1/\varepsilon^{d-1}, \log^{d-2} n\}$.

Our third result improves the best known previous work by Nekrich and Smid [22]. Our third result not only allows queries to have arbitrary approximation factor values $1+\varepsilon$, but also improves the size and the query time of these data structures. The size is improved by a factor of $\log n$. Even when ε is fixed to δ, the query time is improved by a factor of $\min\{1/\delta^{d-1}, \log^{d-2} n\}$ compared to the one by Nekrich and Smid [22].

A main tool achieving the three results is a new data structure for range-emptiness and range-counting queries. Consider a grid Γ with side length γ covering an axis-parallel hypercube with side length ℓ that is aligned with the standard quadtree. For a given query range Q and every cell $\Box$ of Γ, we want to check whether there is a point of P contained in $\Box \cap Q$ efficiently. (Or, we want to compute the number of points of P contained in $\Box \cap Q$.) For this purpose, one can use a data structure for orthogonal range-emptiness queries supporting $O(\log^{d-1} n)$ query time [10]. Thus, the task takes $O((\ell/\gamma)^d \log^{d-1} n)$ time for all cells of Γ in total. Notice that $(\ell/\gamma)^d$ is the number of grid cells of Γ.

To improve the running time for the task, we present a new data structure that supports a range-emptiness query in $O(\log^{d-t-1} n + \log n)$ time for a cell of Γ intersecting no face of Q with dimension smaller than t for any fixed t. Using our data structure, the running time for the task is improved to $O(\log^{d-1} n + (\ell/\gamma)^d \log n)$. To obtain this data structure, we observe that a range-emptiness query for $\Box \cap Q$ can be reduced to a $(d-t-1)$-dimensional orthogonal range-emptiness query on points contained in $\Box$ if $\Box$ intersects no face of Q with dimension smaller than t. We maintain a data structure for $(d-t-1)$-dimensional orthogonal range-emptiness queries for each cell of predetermined grids, which we call the compressed quadtree, for every t. However, this requires $\Omega(n^2)$ space in total if we maintain these data structures explicitly. We can reduce the space complexity using a method for making a data structure partially persistent [11].

Another tool to achieve an efficient query time is a *unified grid* based on quadtrees. For the k-median and k-means clusterings, we mainly follow an approach given by Har-Peled and Mazumdar [18]. They partition input points with respect to the approximate centers explicitly, and construct a grid for each subset of the partition. They snap each input point p to a cell of the grid constructed for the subset that p belongs to. However, their algorithm is a single-shot algorithm and requires $\Omega(|P \cap Q|)$ time due to the computation of a coreset from approximate centers of the points contained in a given query box. In our algorithm, we do not partition input points explicitly but we use only one grid instead, which we call the unified grid, for the purpose in the implementation so that a coreset can be constructed more efficiently.

The tools we propose in this paper to implement the algorithm by Har-Peled and Mazumdar can be used for implementing other algorithms based on grids. For example, if Monte Carlo algorithms are allowed, the approach given by Chen [9] for approximate range queries can be implemented by using the tools we propose in this paper.

Due to lack of space, some proofs and details are omitted. The missing proofs and details can be found in the full version of the paper.

2 Preliminaries

For any two points x and y in $\mathbb{R}^d$, we use $d(x, y)$ to denote the Euclidean distance between x and y. For a point x and a set Y in $\mathbb{R}^d$, we use $d(x, Y)$ to denote the smallest Euclidean distance between x and any point in Y. Throughout the paper, we use the terms *square* and *rectangle* in a generic way to refer axis-parallel hypercube and box in $\mathbb{R}^d$, respectively.

Clusterings. For any integer k with $1 \leq k \leq n$, let $\mathcal{C}_k$ be the family of the sets of at most k points in $\mathbb{R}^d$. Let $\Phi : \mathcal{C}_n \times \mathcal{C}_k \to \mathbb{R}_{\geq 0}$ be a cost function which will be defined later. For a set $P \subseteq \mathbb{R}^d$, we define $\textsc{Opt}_k(P)$ as the minimum value $\Phi(P, C)$ over all sets $C \in \mathcal{C}_k$. We call a set $C \in \mathcal{C}_k$ realizing $\textsc{Opt}_k(P)$ a *k-clustering of P under the cost function Φ*.

In this paper, we consider three cost functions $\Phi_\mathsf{M}, \Phi_\mathsf{m}$ and Φ_c that define the k-median, k-means, and k-center clusterings, respectively. Let $\phi(p, C) = \min_{c \in C} d(p, c)$ for any point p in P. The cost functions are defined as follows: for any set C of $\mathcal{C}_k$,

$$\Phi_\mathsf{M}(P, C) = \sum_{p \in P} \phi(p, C), \quad \Phi_\mathsf{m}(P, C) = \sum_{p \in P} (\phi(p, C))^2, \quad \Phi_\mathsf{c}(P, C) = \max_{p \in P} \phi(p, C).$$

We consider the query variants of these problems. We preprocess P so that given a query rectangle Q, an integer k with $1 \leq k \leq n$, and a value $\varepsilon > 0$, an $(1 + \varepsilon)$-approximate k-clustering of the points contained in Q can be reported efficiently. Specifically, we want to report a set $C \in \mathcal{C}_k$ with $\Phi(P_Q, C) \leq (1+\varepsilon)\textsc{Opt}_k(P_Q)$ in sublinear time, where $P_Q = P \cap Q$.

Coreset for clustering. Consider a cost function Φ. We call a set $S \subseteq \mathbb{R}^d$ a (k, ε)-*coreset* of P for the k-clustering under the cost function Φ if the following holds: for any set C of $\mathcal{C}_k$,

$$(1 - \varepsilon)\Phi(P, C) \leq \Phi(S, C) \leq (1 + \varepsilon)\Phi(P, C).$$

Here, the points in a coreset might be weighted. In this case, the distance between a point p in d-dimensional space and a weighted point s in a coreset is defined as $w(s) \cdot d(p, s)$, where $w(s)$ is the weight of s and $d(p, s)$ is the Euclidean distance between p and s.

By definition, an $(1 + \varepsilon)$-approximation to the k-clustering of S is also an $(1 + \varepsilon)$-approximation to the k-clustering of P. Thus, (k, ε)-coresets can be used to obtain a fast approximation algorithm for the k-median, k-means, and k-center clusterings. A (k, ε)-coreset of smaller size gives a faster approximation algorithm for the clusterings. The followings are the sizes of the smallest (k, ε)-coresets known so far: $O(k/\varepsilon^2)$ for the k-median and k-means clusterings in $\mathbb{R}^d$ [13], and $O(k/\varepsilon^d)$ for the k-center clustering in $\mathbb{R}^d$ [3].

It is also known that (k, ε)-coresets for the k-median, k-means, and k-center clusterings are *decomposable*. That is, if S_1 and S_2 are (k, ε)-coresets for disjoint sets P_1 and P_2, respectively, then $S_1 \cup S_2$ is a (k, ε)-coreset for $P_1 \cup P_2$ by [18, Observation 7.1]. Using this property, one can obtain data structures on P that support an $(1 + \delta)$-approximation to the k-median, k-means, and k-center range-clustering queries for constants $\delta > 0$ and k with $1 \leq k \leq n$ which are given in the construction phase.

▶ **Lemma 1.** *Given a set P of n points in $\mathbb{R}^d$ and a value $\delta > 0$ given in the construction phase, we can construct a data structure of size $O(n \log^d n)$ so that a (k, δ)-coreset of $P \cap Q$ for the k-median and k-means clusterings of size $O(k \log^d n)$ can be computed in $O(k \log^d n)$ time for any orthogonal range Q and any integer k with $1 \leq k \leq n$ given as a query.*

Note that the approximation factor of the coreset is fixed to queries. In Section 4, we will describe a data structure and its corresponding query algorithm answering k-median and k-means range-clustering queries that allow queries to have arbitrary approximation factor values $1 + \varepsilon$ using the algorithm in Lemma 1 as a subprocedure.

Single-shot algorithms for the k-median and k-means clusterings. The single-shot version of this problem was studied by Har-Peled and Mazumdar [18]. They gave algorithms to compute (k, ε)-coresets of size $O(k \log n/\varepsilon^d)$ for the k-median and k-means clusterings. We extensively use their results. The algorithm for the k-means clustering works similarly.

They first provide a procedure that given a constant-factor approximation A to the k-means clustering of P consisting of possibly more than k centers, computes a (k, ε)-coreset of P for the k-median clustering of size $O(|A| \log n/\varepsilon^d)$. To do this, the procedure partitions P into m pairwise disjoint subsets with respect to A and constructs a grid for each subset with respect to the approximate centers. For every grid cell $\square$ containing a point of a subset P' of the partition, the procedure picks an arbitrary point q of P' contained in it and assigns the number of points of P' contained in $\square$ to q as its weight. They showed that the set of all weighted points is a (k, ε)-coreset for P of size $O(|A| \log n/\varepsilon^d)$.

Using this procedure, the algorithm constructs a (k, ε)-coreset S of size $O(k \log^4 n/\varepsilon^d)$ using a constant-factor approximation of size $O(k \log^3 n)$. Using the coreset S, the algorithm obtains a smaller coreset of size $O(k \log n/\varepsilon^d)$ as follows. The algorithm computes a constant-factor approximation $\mathcal{C}_0$ to the k-center clustering of S using the algorithm in [14]. This clustering is an $O(n)$-approximation to the k-median clustering. Then it applies the local search algorithm by Arya et al. [7] to $\mathcal{C}_0$ and S to obtain a constant-factor approximation of P of size at most k. It uses this set to compute a (k, ε)-coreset of size $O(k \log n/\varepsilon^d)$ by applying the procedure mentioned above again.

3 Data structures for range-clustering queries

We maintain two data structures constructed on P. One is a compressed quadtree [5], and the other is a variant of a range tree, which we introduce in this paper.

3.1 Compressed quadtree

We use the term *quadtrees* in a generic way to refer the hierarchical spatial tree data structures for d-dimensional data that are based on the principle of recursive decomposition of space, also known as quadtrees, octrees, and hyperoctrees for spatial data in $d = 2, 3$, and higher dimensions, respectively. We consider two types of quadtrees: a standard quadtree and a compressed quadtree.

Without loss of generality, we assume that the side length of the square corresponding to the root is 1. Then every cell of the standard quadtree has side length of 2^{-i} for an integer i with $0 \leq i \leq t$ for some constant t. We call a value of form 2^{-i} for an integer i with $0 \leq i \leq t$ a *standard length*. Also, we call a grid a *standard grid* if every cell of the grid is also in the standard quadtree. In other words, any grid aligned with the standard quadtree is called a standard grid. We use $\mathcal{T}_s$ and $\mathcal{T}_c$ to denote the standard and compressed quadtrees constructed on P, respectively.

For ease of description, we will first introduce our algorithm in terms of the standard quadtree. But the algorithm will be implemented using the compressed quadtree to reduce the space complexity. To do this, we need the following lemma. In the following, we use a node and its corresponding cell of $\mathcal{T}_s$ (and $\mathcal{T}_c$) interchangeably. For a cell $\square$ of the standard quadtree on P, there is a unique cell $\overline{\square}$ of the compressed quadtree on P satisfying $\square \cap P = \overline{\square} \cap P$. We call this cell the *compressed cell* of $\square$.

▶ **Lemma 2** ([16]). *We can find the compressed cell of a given cell $\square$ of $\mathcal{T}_s$ in $O(\log n)$ time.*

3.2 Data structure for range-emptiness queries

In our query algorithm, we consider a standard grid Γ of side length γ covering an axis-parallel hypercube of side length ℓ. For a given query range Q and every cell $\square$ of Γ, we want to check whether there is a point of P contained in $\square \cap Q$ efficiently. For this purpose, one can use a data structure for orthogonal range-emptiness queries supporting $O(\log^{d-1} n)$ query time [10]. Thus, the task takes $O((\ell/\gamma)^d \log^{d-1} n)$ time for all cells of Γ in total. Notice that $(\ell/\gamma)^d$ is the number of grid cells of Γ.

However, we can accomplish this task more efficiently using the data structure which we will introduce in this section. Let t be an integer with $0 < t \leq d$. We use $<_t$-face to denote a face with dimension smaller than t among faces of a rectangle in $\mathbb{R}^d$. Note that a corner of a rectangle in $\mathbb{R}^d$ is a $<_t$-face of a rectangle for $t = 1$. Our data structure allows us to check whether a point of P is contained in $Q \cap \overline{\square}$ in $O(\log^{d-t-1} n + \log n)$ time for a cell $\overline{\square}$ of $\mathcal{T}_c$ intersecting no $<_t$-face of Q with $0 < t < d$. Here, we first compute the compressed cell $\overline{\square}$ of each cell $\square$ in Γ, and then apply the query algorithm to $\overline{\square}$. Recall that $\square \cap P$ coincides with $\overline{\square} \cap P$ for any cell $\square$ of Γ and its compressed cell $\overline{\square}$. In this way, we can complete the task in $O(\sum_{t=1}^{d-1} x_t \log^{d-t-1} n + |\Gamma| \log n + \log^{d-1} n)$ time in total, where x_t is the number of cells of Γ intersecting no $<_t$-face of Q but intersecting a t-dimensional face of Q. Notice that for any cell $\square$ intersecting Q, there is an integer t with $0 < t \leq d$ such that $\square$ intersects no $<_t$-face of Q, but intersects a t-dimensional face of Q unless $\square$ contains a corner of Q. Here, x_t is $O((\ell/\gamma)^t)$. Therefore, we can accomplish the task for every cell of Γ in $O(\log^{d-1} n + (\ell/\gamma)^d \log n)$ time in total.

For a nonempty subset I of $\{1, \ldots, d\}$ of size t, the *I-projection range tree* on a point set $A \subseteq \mathbb{R}^d$ is the range tree supporting fractional cascading constructed on the projections of A onto a $(d-t)$-dimensional hyperplane orthogonal to the ith axes for all $i \in I$.

▶ **Lemma 3.** *Given the I-projection range tree on $P \cap \square$ for every cell $\square$ of $\mathcal{T}_c$ and every nonempty subset I of $\{1, \ldots, d\}$, we can check whether a point of P is contained in $Q \cap \square$ for any query rectangle Q and any cell $\square$ of $\mathcal{T}_c$ intersecting no $<_t$-face of Q in $O(\log^{d-t-1} n + \log n)$ time.*

However, the I-projection range trees require $\Omega(n^2)$ space in total if we store them explicitly. To reduce the space complexity, we use a method of making a data structure *partially persistent* [11]. A partially persistent data structure allows us to access any elements of an old version of the data structure by keeping the changes on the data structure. Driscoll et al. [11] presented a general method of making a data structure based on pointers partially persistent. In their method, both time and space overheads for an update are $O(1)$ amortized, and the access time for any version is $O(\log n)$.

3.2.1 Construction of the I-projection range trees

Consider a fixed subset I of $\{1, \dots, d\}$. We construct the I-projection range trees for the cells of $\mathcal{T}_c$ in a bottom-up fashion, from leaf cells to the root cell. Note that each leaf cell of the compressed quadtree contains at most one point of P. We initially construct the I-projection range tree for each leaf cell of $\mathcal{T}_c$ in total $O(n)$ time.

Assume that we already have the I-projection range tree for every child node of an internal node v with cell $\square$ of $\mathcal{T}_c$. Note that an internal node of the compressed quadtree has at least two child nodes and up to 2^d child nodes. We are going to construct the I-projection range tree for v from the I-projection range trees of the child nodes of v. One may consider to merge the I-projection range trees for the child nodes of v into one, but we do not know any efficient way of doing it. Instead, we construct the I-projection range tree for v as follows. Let u be a child node of v on $\mathcal{T}_c$ that contains the largest number of points of P in its corresponding cell among all child nodes of v. We insert all points of P contained in the cells for the child nodes of v other than u to the I-projection range tree of u to form the I-projection range tree of v. Here, we do not destroy the old version of the I-projection range tree of u by using the method by Driscoll et al. [11]. Therefore, we can still access the I-projection range tree of u. For the insertion, we use an algorithm that allows us to apply fractional cascading on the range tree under insertions of points [21]. We do this for every subset I of $\{1, \dots, d\}$ and construct the I-projection range trees for nodes of $\mathcal{T}_c$.

In this way, we can access any I-projection range tree in $O(\log n)$ time, and therefore we can check if a point of P is contained in $Q \cap \square$ in $O(\log^{d-t-1} n + \log n)$ time for any rectangle Q and any cell $\square$ of $\mathcal{T}_c$ intersecting no $<_t$-face of Q for any integer t with $0 < t \leq d$ by Lemma 3.

3.2.2 Analysis of the construction

The construction of the dynamic range tree [21, Theorem 8] requires $O(\delta \log^{d-t-1} \delta)$ space on the insertions of δ points in $\mathbb{R}^{d-t}$. The method by Driscoll et al. requires only $O(1)$ overhead for each insertion on the space complexity. Thus, the space complexity of the I-projection range trees for a fixed subset I consisting of t indices over all cells of $\mathcal{T}_c$ is $O(n + \delta \log^{d-t-1} \delta)$, where δ is the number of the insertions performed during the construction in total.

The update procedure for the dynamic range tree [21, Theorem 8] takes $O(\log^{d-t-1} n)$ time if only insertions are allowed. The method by Driscoll requires only $O(1)$ overhead for each insertion on the update time. Thus, the construction time is $O(n + \delta \log^{d-t-1} n)$, where δ is the number of the insertions performed during the construction in total.

The following lemma shows that δ is $O(n \log n)$, and thus our construction time is $O(n \log^{d-t} n)$ and the space complexity of the data structure is $O(n \log^{d-t} n)$ for each integer t with $0 < t \leq d$. Note that there are $2^d = O(1)$ subsets of $\{1, \dots, d\}$. Therefore, the total space complexity and construction time are $O(n \log^{d-1} n)$.

▶ **Lemma 4.** *For a fixed subset I of $\{1, \dots, d\}$, the total number of insertions performed during the construction of all I-projection range trees for every node of $\mathcal{T}_c$ is $O(n \log n)$.*

▶ **Lemma 5.** *We can construct a data structure of size $O(n \log^{d-1} n)$ in $O(n \log^{d-1} n)$ time so that the emptiness of $P \cap Q \cap \square$ can be checked in $O(\log^{d-t-1} n + \log n)$ for any query rectangle Q and any cell $\square$ of $\mathcal{T}_c$ intersecting no $<_t$-face of Q for an integer t with $0 < t \leq d$.*

For a cell $\square$ containing a corner of Q, there is no index t such that $\square$ intersects no $<_t$-face of Q. In this case, we use the standard range tree on P and check the emptiness of $P \cap Q \cap \square$ in $O(\log^{d-1} n)$ time. Notice that there are 2^d such cells because the cells are pairwise disjoint.

3.3 Data structure for range-counting queries

The data structure for range-emptiness queries described in Section 3.2 can be extended to a data structure for range-reporting queries. However, it does not seem to work for range-counting queries. This is because the dynamic range tree with fractional cascading [21] does not seem to support counting queries. Instead, we use a dynamic range tree without fractional cascading, which increases the query time and update time by a factor of $\log n$. The other part is the same as the data structure for range-emptiness queries.

▶ **Lemma 6.** *We can construct a data structure of size $O(n \log^{d-1} n)$ in $O(n \log^{d-1} n)$ time so that the number of points of P contained in $Q \cap \square$ can be computed in $O(\log^{d-t} n + \log n)$ time for any query rectangle Q and any cell $\square$ of $\mathcal{T}_c$ intersecting no $<_t$-face of Q for an integer t with $0 < t \leq d$.*

4 k-Median range-clustering queries

In this section, we present a data structure and a query algorithm for k-median range-clustering queries. Given a set P of n points in $\mathbb{R}^d$ for a constant $d \geq 2$, our goal is to preprocess P such that k-median range-clustering queries can be answered efficiently. A k-median range-clustering query consists of a box Q, an integer k with $1 \leq k \leq n$, and a value $\varepsilon > 0$. We want to find a set $C \in \mathcal{C}_k$ with $\Phi_{\mathsf{M}}(P_Q, C) \leq (1+\varepsilon)\textsc{Opt}_k(P_Q)$ efficiently, where $P_Q = P \cap Q$. Throughout this section, we use Φ to denote Φ_{M} unless otherwise specified.

Our query algorithm is based on the single-shot algorithm by Har-Peled and Mazumdar [18]. A main difficulty in the implementation for our setting is that they construct a grid with respect to each point in an approximate center set. Then for each grid cell, they compute the number of points of P_Q contained in the grid cell. Thus to implement their approach in our setting directly, we need to apply a counting query to each grid cell. Moreover, we have to avoid overcounting as a point might be contained in more than one grid cell of their grid structures.

To do this efficiently without overcounting, we use a *unified grid* based on the standard quadtree. Although this grid is defined on the standard quadtree, we use the grid on the compressed quadtree in the implementation To make the description easier, we use the standard quadtree instead of the compressed quadtree in defining of the unified grid.

4.1 Coreset construction from approximate centers

Assume that we are given a constant-factor approximation $A = \{a_1, \ldots, a_m\}$ to the k-median clustering of P_Q, where m is possibly larger than k. In this subsection, we present a procedure that computes a (k, ε)-coreset of size $O(|A| \log n/\varepsilon^d)$. We follow the approach by Har-Peled and Mazumdar [18] and implement it in our setting.

General strategy. We describe our general strategy first, and then show how to implement this algorithm. For the definitions of the notations used in the following, refer to those in Section 2 unless they are given. We compute a $2\sqrt{d}$-approximation R to the maximum of $d(p, A)/(c_1|P_Q|)$ over all points $p \in P_Q$, that is, a value R satisfying that the maximum value lies between $R/(2\sqrt{d})$ and $2\sqrt{d}R$, where $c_1 > 1$ is the approximation factor of A.

Let Q_{ij} be the cell of the standard quadtree containing a_i with side length $\bar{R}_j$ satisfying $R2^j \leq \bar{R}_j < R2^{j+1}$ for $j = 0, \ldots, M = \lceil 2\log(2\sqrt{d}c_1|P_Q|) \rceil$. By construction, note that $Q_{ij_1} \subset Q_{ij_2}$ for any two indices j_1 and j_2 with $j_1 < j_2$. For any point p in P_Q, we have at least one cell Q_{ij} containing p since there is a value $\bar{R}_j$ at least four times the maximum of $d(p, A)$.

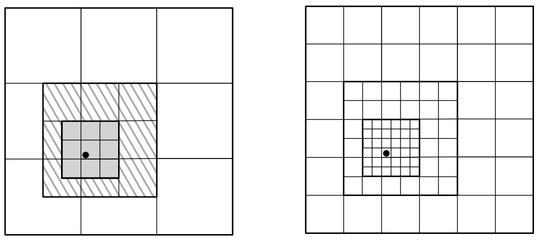

Figure 1 (a) Exponential grid for an algorithm for the query version constructed with respect to the point, say a_1, in the middle. The point is not the center of the grid. The gray region is the grid cluster for Q_{10}, the dashed region is the grid cluster for Q_{11}, and the largest box is the grid cluster for Q_{12}. (b) Only first-level grid is depicted.

We define the *grid cluster* for Q_{ij} as the union of at most 3^d grid cells of the standard quadtree with side length $\bar{R}_j$ that share their faces with Q_{ij} including Q_{ij}. Note that the grid cluster for Q_{ij} contains all points of $\mathbb{R}^d$ that are within distance from a_i at most $\bar{R}_j$. Also, every point in $\mathbb{R}^d$ contained in the grid cluster for Q_{ij} has its distance from a_i at most $2\sqrt{d}\bar{R}_j$. See Figure 1(a). Let V_{i0} denote the grid cluster for Q_{i0} and V_{ij} be the grid cluster for Q_{ij} excluding the grid cluster for $Q_{i(j-1)}$. Note that V_{ij} is the union of at most $3^d(2^d-1)$ cells of the standard quadtree with side length $\bar{R}_j/2$, except for $j=0$. For $j=0$, the region V_{i0} is the union of at most 3^d such cells.

The *first-level grid* for a fixed index i consists of all cells of the standard quadtree with side length $\bar{R}_j/2$ contained in V_{ij}. For an illustration, see Figure 1(b). We partition each cell of the first-level grid into the cells of the standard quadtree with side length $\bar{r}_j$ satisfying $\varepsilon\bar{R}_j/(40c_1d) \leq \bar{r}_j \leq 2\varepsilon\bar{R}_j/(40c_1d)$. The *second-level grid* for i consists of all such cells. Let $\mathcal{V}$ be the set of all grid cells which contain at least one point of P_Q. Note that the size of $\mathcal{V}$ is $O(|A|\log n/\varepsilon^d)$. We will compute it in $O(|A|\log^d n/\varepsilon + |A|\log n/\varepsilon^d)$ time.

We consider the grid cells $\square$ of $\mathcal{V}$ one by one in the increasing order of their side lengths, and do the followings. Let $P(\square)$ be the set of points of P_Q that are contained in $\square$, but are not contained in any other grid cells we have considered so far. We compute the number of points of $P(\square)$, and assign this number to an arbitrary point of $P(\square)$ as its weight. We call this weighted point the *representative* of $\square$. Also, we say that a point of $P(\square)$ is *charged* to $\square$. Notice that every point of P_Q is charged to exactly one cell of $\mathcal{V}$. We describe the details of this procedure in Section 4.1.2. Let S be the set of all such weighted points. Then S is a (k,ε)-coreset for P_Q of size $O(|A|\log n/\varepsilon^d)$.

We implement the algorithm using the compressed quadtree, not the standard quadtree. We provide an implementation of the algorithm in the following subsections briefly.

4.1.1 Computing an approximation to the average radius

The first step is to compute a $2\sqrt{d}$-approximation R to the maximum $\textsc{max}$ of $d(p,A)/(c_1|P_Q|)$ over all points $p \in P_Q$, where $c_1 > 1$ is the approximation factor of A. More precisely, we compute R such that $R/(2\sqrt{d}) \leq \textsc{max} \leq 2\sqrt{d}R$. We can compute it in $O(|A|\log^d n)$ time.

Let r^* be the maximum of $d(p,A)$ over all points $p \in P_Q$. We compute a $2\sqrt{d}$-approximation of r^* and divide it by $c_1|P_Q|$ to compute R. Note that we can compute $|P_Q|$ in $O(\log^{d-1} n)$ time using the range tree constructed on P. Imagine that we have a standard grid with side length $\alpha > 0$ covering Q. Consider the grid cells in this grid each of which contains a point of A. If the union of the grid clusters of all these grid cells contains P_Q, it holds that $d(p,A) \leq 2\alpha\sqrt{d}$ for any $p \in P_Q$. Otherwise, $d(p,A) > \alpha$ for some $p \in P_Q$. We use this observation to check whether $2\alpha\sqrt{d} \geq r^*$ or $\alpha \leq r^*$.

We apply binary search on the standard lengths. For each iteration with standard length α, we check whether α is at most r^* or at least $r^*/(\alpha\sqrt{d})$. As a result, we obtain an interval whose endpoints are standard lengths, and return the larger endpoint as an output. However, there are an arbitrarily large number of distinct standard lengths. We consider only $O(\log n)$ distinct standard lengths among them.

▶ **Lemma 7.** *We can compute a $2\sqrt{d}$-approximation to the maximum of $d(p, A)/(c_1|P_Q|)$ for all points p in P_Q in $O(|A| \log^d n)$ time.*

4.1.2 Computing the compressed cells in the grid

As described in Section 4.1, we construct the second-level grid for each index i for $i = 1, \dots, m$, and check whether each grid cell contains a point of P_Q. The set of the grid cells in the second-level grids containing a point of P_Q is denoted by $\mathcal{V}$. Then we consider the grid cells $\Box$ of $\mathcal{V}$ one by one in the increasing order of their side lengths, and compute the number of points of P_Q contained in $\Box$, but not contained in any other grid cells we have considered so far. Computing this number is quite tricky.

To handle this problem, we observe that for any two cells in $\mathcal{V}$, either they are disjoint or one is contained in the other because they are cells of the standard quadtree. For two cells $\Box_1$ and $\Box_2$ with $\Box_1 \subseteq \Box_2$, let i_1 and i_2 be the indices such that $\Box_1$ and $\Box_2$ are the grid cells of the second-level grids for i_1 and i_2, respectively. Since the grid cells in the same second-level grid are pairwise interior disjoint, we have $i_1 \neq i_2$. In this case, for any point $p \in \Box_2$, there is another grid cell $\Box_1'$ containing p in the second-level grid for i_1 with side length smaller than the side length of $\Box_2$. Therefore, we do not consider any cell of $\mathcal{V}$ containing another cell of $\mathcal{V}$. Imagine that we remove all such cells from $\mathcal{V}$. Then the cells of $\mathcal{V}$ are pairwise interior disjoint. Therefore, if suffices to compute the number of points of P_Q contained in each cell of $\mathcal{V}$, which can be done efficiently using the data structure in Section 3.

We show how to compute the set $\mathcal{V}$ after removing all cells containing another cell efficiently. To do this, we first compute the cells in the first-level grids, and discard some of them. Then we subdivide the remaining cells into cells in the second-level grids. More specifically, let $\mathcal{V}_1$ be the set of the cells of the first-level grids. We first compute the cells in $\mathcal{V}_1$, and then remove all cells in $\mathcal{V}_1$ containing another cell in $\mathcal{V}_1$. Then the cells in $\mathcal{V}_1$ are pairwise interior disjoint. And then we compute the second-level grid cells in each cell of $\mathcal{V}_1$. The second-level grid cells we obtain are the cells of $\mathcal{V}$ containing no other cell in $\mathcal{V}$.

Range-counting for each compressed cell. The next step is to compute the number of points of P_Q contained in each cell $\Box$ of $\mathcal{V}$. If $\Box$ is contained in Q, we already have the number of points of P_Q contained in $\Box$, which is computed in the preprocessing phase. If $\Box$ contains a corner of Q, we use the range tree constructed on P. Since there are $O(1)$ such cells, we can handle them in $O(\log^{d-1} n)$ time in total. For the remaining cells, we use the data structure in Section 3.3. Then we can handle them in $O(\sum_{t=1}^{d-1} m_t \log^{d-t} n)$ time, where m_t is the number of the cells of $\mathcal{V}$ intersecting no $<_t$-face of Q but intersecting a t-dimensional face of Q for an integer with $0 < t < d$. We have $m_t = O(|A| \log n/\varepsilon^t)$. Therefore, the total running time for the range-counting queries is $O(|A| \log^2 n/\varepsilon^{d-1} + |A| \log^d n/\varepsilon + \log^{d-1} n + |A| \log n/\varepsilon^d)$ in total, which is $O(|A| \log^d n/\varepsilon + |A| \log n/\varepsilon^d)$.

▶ **Lemma 8.** *Given a constant-factor approximation A to the k-median clustering of a set P of n points in $\mathbb{R}^d$ such that $|A|$ is possibly larger than k, we can compute a (k, ε)-coreset of P_Q of size $O(|A| \log n/\varepsilon^d)$ in $O(|A| \log^d n/\varepsilon + |A| \log n/\varepsilon^d)$ time for any rectangle Q, any integer k with $1 \leq k \leq n$ and any value $\varepsilon > 0$.*

4.2 Smaller coreset

Due to Lemma 1, we can obtain a $(k,2)$-coreset S of P_Q of size $O(k \log^d n)$ in $O(k \log^d n)$ time for any query rectangle Q using a data structure of size $O(n \log^d n)$. A (k,c)-coreset of S is also a $(k,2c)$-coreset of P_Q for any constant $c > 1$ by the definition of the coreset. We compute a $(k,2)$-corset S' of S, which is a $(k,4)$-coreset of P_Q, of size $O(k \log n)$ in $O(k \log^d n + k^5 \log^9 n)$ time by [18, Lemma 5.1] by setting $\varepsilon = 2$.

Using this $(k,4)$-coreset of size $O(k \log n)$ of P_Q, we can obtain constant-factor approximate centers of size k as Har-Peled and Mazumdar [18] do. Then we can compute a (k,ε)-coreset of size $O(k \log n/\varepsilon^d)$ using Lemma 8 again using the constant-factor approximation centers to $\textsc{Opt}_k(S)$ of size k.

This algorithm is extended to the k-means range-clustering problem.

▶ **Theorem 9.** *There is a data structure of size $O(n \log^d n)$ on a set P of n points such that given a box $Q \subseteq \mathbb{R}^d$, an integer k with $1 \le k \le n$, and a value $\varepsilon > 0$ as a query, an $(1+\varepsilon)$-approximation to the k-median range-clustering of $P \cap Q$ can be computed in $O(k^5 \log^9 n + k \log^d /\varepsilon + T_{\text{ss}}(k \log n/\varepsilon^d))$ time, where $T_{\text{ss}}(N)$ denotes the running time of an $(1+\varepsilon)$-approximation single-shot algorithm for the k-median clustering of N weighted points.*

Remark. The construction of the coreset for the k-means clustering is similar to the construction of the coreset for the k-median clustering in [18]. The only difference is that for the k-means clustering Φ_{m} is used instead of Φ_{M} and $R = \sqrt{\Phi_{\mathsf{m}}(P,A)/(c_1 n)}$ is used instead of $R = \Phi_{\mathsf{M}}(P,A)/(c_1 n)$. Therefore, we can compute a (k,ε)-coreset for the k-means clustering of size $O(k \log n/\varepsilon^d)$ in $O(k^5 \log^9 n + k \log^d n/\varepsilon + k \log n/\varepsilon^d)$ time.

▶ **Theorem 10.** *Let P be a set of n points in d-dimensional space. There is a data structure of size $O(n \log^d n)$ such that given a query range $Q \subseteq \mathbb{R}^d$, an integer k with $1 \le k \le n$, and a value $\varepsilon > 0$ as a query, an $(1+\varepsilon)$-approximation to the k-means range-clustering of $P \cap Q$ can be computed in $O(k^5 \log^9 n + k \log^d /\varepsilon + T_{\text{ss}}(k \log n/\varepsilon^d))$ time, where $T_{\text{ss}}(N)$ denotes the running time of an $(1+\varepsilon)$-approximation single-shot algorithm for the k-means clustering of N weighted input points.*

5 k-Center range-clustering queries

We are given a set P of n points in $\mathbb{R}^d$ for a constant $d \ge 2$. Our goal is to process P so that k-center range-clustering queries can be computed efficiently. A range-clustering query consists of a rectangle $Q \subseteq \mathbb{R}^d$, an integer k with $1 \le k \le n$, and a value $\varepsilon > 0$. We want to find a set $C \in \mathcal{C}_k$ with $\Phi_{\mathsf{c}}(P_Q, C) \le (1+\varepsilon)\textsc{Opt}_k(P_Q)$ efficiently, where $P_Q = P \cap Q$. Combining the algorithm by Abrahamsen et al. [1] and the data structure in Section 3, we can improve the algorithm by Abrahamsen et al.

▶ **Theorem 11.** *There is a data structure of size $O(n \log^{d-1} n)$ on a set P of n points in $\mathbb{R}^d$ such that given a box $Q \subseteq \mathbb{R}^d$, an integer k with $1 \le k \le n$, and a value $\varepsilon > 0$ as a query, an $(1+\varepsilon)$-approximation to the k-center range-clustering of $P \cap Q$ can be computed in $O(k \log^{d-1} n + k \log n/\varepsilon^{d-1} + T_{\text{ss}}(k/\varepsilon^d))$ time, where $T_{\text{ss}}(N)$ denotes the running time of an $(1+\varepsilon)$-approximation single-shot algorithm for the k-center clustering of N points.*

6 Approximate diameter and radius of a point set

In this section, we are given a set P of n points in $\mathbb{R}^d$. Our goal in this section is to preprocess P so that given any orthogonal range Q and a value $\varepsilon > 0$, an approximate diameter (or radius) of $P \cap Q$ can be computed efficiently. We give a sketch of the algorithm.

Our query algorithm starts by sampling a set S of points from $P \cap Q$, which we call an ε-coreset of $P \cap Q$, such that the diameter of S is an $(1+\varepsilon)$-approximation of the diameter of $P \cap Q$. Let $\textsc{apx}$ be a value such that $D \leq \textsc{apx} \leq c \cdot D$ for a constant $c > 1$, where D is the diameter of $P \cap Q$. Consider a standard grid of side length $\varepsilon\textsc{apx}$ covering Q. Assume that we pick an arbitrary point in each grid cell containing a point of $P \cap Q$. Then the set of all picked points is an ε-coreset of $P \cap Q$ of size $O(1/\varepsilon^d)$.

Let $\mathcal{D}$ be the set of all grid cells containing a point of $P \cap Q$. We can obtain a smaller ε-coreset as follows. We first obtain a subset $\mathcal{D}' \subseteq \mathcal{D}$ and choose an arbitrary point in each grid cell of $\mathcal{D}'$ for a ε-coreset. If a grid cell of $\mathcal{D}$ intersects the boundary of Q, we move it from $\mathcal{D}$ to $\mathcal{D}'$. For the remaining cells of $\mathcal{D}$, consider the grid cells of $\mathcal{D}$ whose centers have the same coordinates, except for only one coordinate, say the ith coordinate. We add the grid cells with the largest ith coordinate and smallest ith coordinate to $\mathcal{D}'$. Then $\mathcal{D}'$ consists of $O(1/\varepsilon^{d-1})$ grid cells. We choose an arbitrary point of P contained in each grid cell of $\mathcal{D}'$. The set S of all chosen points is an ε-coreset of $P \cap Q$ of size $O(1/\varepsilon^{d-1})$. Using the data structure described in Section 3, we can compute this coreset in $O(\log^{d-1} n + \log n/\varepsilon^{d-1})$ time. Also, this approach works for the problem of computing the radius of the smallest enclosing ball of $P \cap Q$.

▶ **Theorem 12.** *Given a set P of n points in $\mathbb{R}^d$, we can compute an $(1+\varepsilon)$-approximate diameter (or radius) of $P \cap Q$ in $O(\log^{d-1} n + \log n/\varepsilon^{d-1})$ time for a query consisting of an orthogonal range Q and a value $\varepsilon > 0$ using a data structure of size $O(n \log^{d-1} n)$.*

References

1 Mikkel Abrahamsen, Mark de Berg, Kevin Buchin, Mehran Mehr, and Ali D. Mehrabi. Range-Clustering Queries. In *Proceedings of the 33rd International Symposium on Computational Geometry (SoCG 2017)*, volume 77, pages 5:1–5:16, 2017.

2 Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In *Advances in Discrete and Compputational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society Press, 1999.

3 Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. Geometric approximation via coresets. In *Combinatorial and Computational Geometry*, volume 52, pages 1–30. MSRI Publications, 2005.

4 Pankaj. K. Agarwal and Cecillia M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.

5 Srinivas Aluru. Quadtrees and octrees. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, chapter 19. Chapman & Hall/CRC, 2005.

6 Sunil Arya, David M. Mount, and Eunhui Park. Approximate geometric MST range queries. In *Proceedings of the 31st International Symposium on Computational Geometry (SoCG 2015)*, pages 781–795, 2015.

7 Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

8 Peter Brass, Christian Knauer, Chan-Su Shin, Michiel Smid, and Ivo Vigan. Range-aggregate queries for geometric extent problems. In *Proceedings of the 19th Computing: The Australasian Theory Symposium (CATS 2013)*, volume 141, pages 3–10, 2013.

9 Ke Chen. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM Journal on Computing*, 39(3):923–947, 2009.

10 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag TELOS, 2008.

11 James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.

12 Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC 1988)*, pages 434–444, 1988.

13 Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43th Annual ACM Symposium on Theory of Computing (STOC 2011)*, pages 569–578, 2011.

14 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(Supplement C):293–306, 1985.

15 Prosenjit Gupta, Ravi Janardan, Yokesh Kumar, and Michiel Smid. Data structures for range-aggregate extent queries. *Computational Geometry*, 47(2, Part C):329–347, 2014.

16 Sariel Har-Peled. *Geometric Approximation Algorithms.* Mathematical surveys and monographs. American Mathematical Society, 2011.

17 Sariel Har-Peled and Akash Kushal. Smaller coresets for k-median and k-means clustering. *Discrete & Computational Geometry*, 37(1):3–19, Jan 2007.

18 Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 291–300, 2004.

19 Anil Kumar Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

20 Jiří Matoušek. Geometric range searching. *ACM Computing Surveys*, 26(4):422–461, 1994.

21 Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(1):215–241, 1990.

22 Yakov Nekrich and Michiel H. M. Smid. Approximating range-aggregate queries using coresets. In *Proceedings of the 22nd Annual Canadian Conference on Computational Geometry (CCCG 2010)*, pages 253–256, 2010.

23 Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. Aggregate nearest neighbor queries in spatial databases. *ACM Transactions on Database System*, 30(2):529–576, 2005.

24 Jing Shan, Donghui Zhang, and Betty Salzberg. On spatial-range closest-pair query. In *Proceedings of the 8th International Symposium on Advances in Spatial and Temporal Databases (SSTD 2003)*, pages 252–269, 2003.