# In-Range Farthest Point Queries and Related Problem in High Dimensions

**Ziyun Huang**
Department of Computer Science and Software Engineering,
Penn State Erie, The Behrend College, USA

**Jinhui Xu**
Department of Computer Science and Engineering,
State University of New York at Buffalo, NY, USA

## Abstract

Range-aggregate query is an important type of queries with numerous applications. It aims to obtain some structural information (defined by an *aggregate function* $F(\cdot)$) of the points (from a point set $P$) inside a given query range $B$. In this paper, we study the range-aggregate query problem in high dimensional space for two aggregate functions: (1) $F(P \cap B)$ is the farthest point in $P \cap B$ to a query point $q$ in $\mathbb{R}^d$ and (2) $F(P \cap B)$ is the minimum enclosing ball (MEB) of $P \cap B$. For problem (1), called *In-Range Farthest Point (IFP) Query*, we develop a bi-criteria approximation scheme: For any $\epsilon > 0$ that specifies the approximation ratio of the farthest distance and any $\gamma > 0$ that measures the "fuzziness" of the query range, we show that it is possible to pre-process $P$ into a data structure of size $\tilde{O}_{\epsilon,\gamma}(dn^{1+\rho})$ in $\tilde{O}_{\epsilon,\gamma}(dn^{1+\rho})$ time such that given any $\mathbb{R}^d$ query ball $B$ and query point $q$, it outputs in $\tilde{O}_{\epsilon,\gamma}(dn^{\rho})$ time a point $p$ that is a $(1-\epsilon)$-approximation of the farthest point to $q$ among all points lying in a $(1+\gamma)$-expansion $B(1+\gamma)$ of $B$, where $0 < \rho < 1$ is a constant depending on $\epsilon$ and $\gamma$ and the hidden constants in big-O notations depend only on $\epsilon$, $\gamma$ and Polylog$(nd)$. For problem (2), we show that the IFP result can be applied to develop query scheme with similar time and space complexities to achieve a $(1+\epsilon)$-approximation for MEB. To the best of our knowledge, these are the first theoretical results on such high dimensional range-aggregate query problems. Our results are based on several new techniques, such as *multi-scale construction* and *ball difference range query*, which are interesting in their own rights and could be potentially used to solve other range-aggregate problems in high dimensional space.

**2012 ACM Subject Classification** Theory of computation → Computational geometry

**Keywords and phrases** Farthest Point Query, Range Aggregate Query, Minimum Enclosing Ball, Approximation, High Dimensional Space

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2022.75

**Category** Track A: Algorithms, Complexity and Games

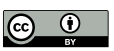
**Related Version** *Full Version*: `https://arxiv.org/abs/2206.07592`

**Funding** *Jinhui Xu*: The research of this author was supported in part by NSF through grant IIS-1910492 and by KAUST through grant CRG10 4663.2.

## 1 Introduction

Range search is a fundamental problem in computational geometry and finds applications in many fields like database systems and data mining [4, 27]. It has the following basic form: Given a set of $n$ points $P$ in $\mathbb{R}^d$, pre-process $P$ into a data structure so that for any query range $B$ from a certain range family (*e.g.,* spheres, rectangles, and halfspaces), it reports or counts the number of the points in $P \cap B$ efficiently. Range search allows us to obtain some basic information of the points that lie in a specific local region of the space.

In many applications, it is often expected to know more information than simply the number of points in the range. This leads to the study of range-aggregate query [2, 3, 6, 10, 13, 20, 21, 23, 25, 26, 32], which is a relatively new type of range search. The goal of







range-aggregate query is to obtain more complicated structural information (such as the diameter, the minimum enclosing ball, and the minimum spanning tree) of the points in the query range. Range-aggregate query can be generally defined as follows: Given a point set $P$, pre-process $P$ into a data structure such that for any range $B$ in a specific family, it outputs $F(P \cap B)$, where $F(\cdot)$ is a given *aggregate function* that computes a certain type of information or structure of $P \cap B$ like "diameter","minimum enclosing ball", and "minimum spanning tree". Range-aggregate queries have some interesting applications in data analytics and big data [16, 28, 29, 32], where it is often required to retrieve aggregate information of the records in a dataset with keys that lie in any given (possibly high dimensional) range.

In this paper, we study the range-aggregate query problem in high dimensions for spherical ranges. Particularly, we consider two aggregate functions for any $\mathbb{R}^d$ query ball $B$: (1) $F(P \cap B)$ is the farthest point in $P \cap B$ to a query point $q$ in $\mathbb{R}^d$ and (2) $F(P \cap B)$ is the minimum enclosing ball (MEB) of $P \cap B$. We will focus in this paper on problem (1), called the *In-Range Farthest Point (IFP) Query*, and show that an efficient solution to IFP query also yields efficient solutions to the MEB problems. We start with some definitions.

▶ **Definition 1** (Approximate IFP (AIFP)). *Let $P$ be a set of $n$ points in $\mathbb{R}^d$, $q$ be a point and $B$ be a $d$-dimensional (closed) ball. A point $p \in P$ is a bi-criteria $(\epsilon, \gamma)$-approximate in-range farthest point (or AIFP) of $q \in P$ in $B$, if there exists a point set $P'$ such that the following holds, where $\epsilon$ and $\gamma$ are small positive constants, and $B(1+\gamma)$ is the ball concentric with $B$ and with radius $(1+\gamma)r$: (1) $P \cap B \subseteq P' \subseteq P \cap B(1+\gamma)$; (2) $p \in P'$; and (3) for any $p' \in P'$, $(1-\epsilon)\|p'-q\| \leq \|p-q\|$.*

Defining AIFP in this way enables us to consider all points in $B$ and exclude all points outside of $B(1+\gamma)$. Points in the fuzzy region $B(1+\gamma) \setminus B$ may or may not be included in the farthest point query. Note that allowing fuzzy region is a commonly used strategy to deal with the challenges in many high dimensional similarity search and range query problems. For example, consider the classic near neighbor search problem, which is equivalent to spherical emptiness range search: Given a query sphere $B$ in $\mathbb{R}^d$, report a data point $p$ that lies in $B$ if such a data point exists. In high dimensional space, obtaining an exact solution to such a query is very difficult. A commonly used technique for this problem is the Locality Sensitive Hashing (LSH) scheme [12]. Given a query ball $B$, LSH could report a data point in $B(1+\epsilon)$ for some given factor $\epsilon > 0$. In other words, a fuzzy region $B(1+\epsilon) \setminus B$ is allowed. Similarly, we can define approximate MEB for points in a given range with a fuzzy region.

▶ **Definition 2** (Minimum Enclosing Ball (MEB)). *Let $P$ be a set of $n$ points in $\mathbb{R}^d$. A $d$-dimensional (closed) ball $B$ is an enclosing ball of $P$ if $P \subset B$ and $B$ is the minimum enclosing ball (MEB) of $P$ if its radius $r$ is the smallest among all enclosing balls. A ball $B'$ is a $(1+\epsilon)$-approximate MEB of $P$ for some constant $\epsilon > 0$ if it is an enclosing ball of $P$ and its radius is no larger than $(1+\epsilon)Rad(P)$, where $Rad(P)$ is the radius of the MEB of $P$.*

▶ **Definition 3** (Approximate MEB (AMEB)). *Let $P$ be a set of $n$ points and $B$ be any ball with radius $r$ in $\mathbb{R}^d$. A ball $B'$ with radius $r'$ is a bi-criteria $(\epsilon, \gamma)$-approximate MEB (or AMEB) of $P$ in range $B$, if there exists a point set $P'$ such that the following holds, where $\gamma$ and $\epsilon$ are small positive constants: (1) $P \cap B \subseteq P' \subseteq P \cap B(1+\gamma)$; and (2) $B'$ is a $(1+\epsilon)$-approximate MEB of $P'$.*

In this paper, we will focus on building a data structure for $P$ so that given any query ball $B$ and a point $q \in \mathbb{R}^d$, an AIFP of $q$ in $P \cap B$ can be computed efficiently (*i.e.,* in sub-linear time in terms of $n$). Below are the main theorems of this paper. Let $\epsilon > 0, \gamma > 0$, $0 < \delta < 1$ be any real numbers.

▶ **Theorem 4.** *For any set $P$ of $n$ points in $\mathbb{R}^d$, it is possible to build a data structure of size $O_{\epsilon,\gamma}(dn^{1+\rho}\log\delta^{-1}Polylog(nd))$ in $O_{\epsilon,\gamma}(dn^{1+\rho}\log\delta^{-1}Polylog(nd))$ pre-processing time, where $0<\rho<1$ is a small constant depending on $\epsilon$ and $\gamma$. With this data structure, it is then possible to find a $(\epsilon,\gamma)$-AIFP of any given query point $q$ and query ball $B$ in $O_{\epsilon,\gamma}(dn^{\rho}\log\delta^{-1}Polylog(nd))$ time with probability at least $1-\delta$.*

*Note: In the above result, the relationship between $\rho$ and $\epsilon,\gamma$ has a rather complicated dependence on several constants of p-stable distribution, which is inherited from the underlying technique of Locality Sensitive Hashing (LSH) scheme [12]. This indicates that for any $\epsilon,\gamma$, we have $0<\rho<1$ and $\rho$ approaches 1 as $\epsilon,\gamma$ approach 0.*

We will also show how to use the AIFP data structure to answer MEB queries efficiently.

▶ **Theorem 5.** *For any set $P$ of $n$ points in $\mathbb{R}^d$, it is possible to build a data structure of size $O_{\epsilon,\gamma}(dn^{1+\rho}\log\delta^{-1}Polylog(nd))$ in $O_{\epsilon,\gamma}(dn^{1+\rho}\log\delta^{-1}Polylog(nd))$ pre-processing time, where $0<\rho<1$ is a small constant depending on $\epsilon$ and $\gamma$. With this data structure, it is then possible to find a $(\epsilon,\gamma)$-AMEB for any query ball $B$ in $O_{\epsilon,\gamma}(dn^{\rho}\log\delta^{-1}Polylog(nd))$ time with probability at least $1-\delta$.*

To our best knowledge, these are the first results on such range-aggregate problems in high dimensions. Each data structure has only a near linear dependence on $d$, a sub-quadratic dependence on $n$ in space complexity, and a sub-linear dependence on $n$ in query time.

**Our Method.** The main result on AIFP is based on several novel techniques, such as *multi-scale construction* and *ball difference range query*. Briefly speaking, multi-scale construction is a general technique that allow us to break the task of building an AIFP query data structure into a number of "constrained" data structures. Each such data structure is capable of correctly answering an AIFP query given that some assumption about the query holds (for example, the distance from $q$ to its IFP is within a certain range). Multi-scale construction uses a number of "constrained" data structures of small size to cover all possible cases of a query, which leads to a data structure that can handle any arbitrary queries. Multi-scale construction is independent of the aggregate function, and thus has the potential be used as a general method for other types of range-aggregate query problems in high dimensional space. Another important technique is a data structure for the ball difference range query problem, which returns a point, if there is one, in the difference of two given query balls. The ball difference data structure is the building block for the constrained AIFP data structures, and is interesting in its own right as a new high dimensional range search problem.

**Related Work.** There are many results for the ordinary farthest point query problem in high dimensional space [11, 17, 19, 24]. However, to the best of our knowledge, none of them is sufficient to solve the IFP problem, and our result is the first one to consider the farthest point problem under the query setting. Our technique for the IFP problem also yields solutions to other range-aggregate queries problems, including the MEB query problem.

A number of results exist for various types of the range-aggregate query problem in fixed dimensional space. In [6], Arya, Mount, and Park proposed an elegant scheme for querying minimum spanning tree inside a query range. They showed that there exists a bi-criteria $(\epsilon_q,\epsilon_w)$-approximation with a query time of $O(\log n+(1/\epsilon_q\epsilon_w)^d)$. In [23], Nekrich and Smid introduced a data structure to compute an $\epsilon$-coreset for the case of orthogonal query ranges and aggregate functions satisfying some special properties. Xue [30] considered the colored closest-pair problem in a (rectangular) range and obtained a couple of data structures with near linear size and polylogarithmic query time. Recently, Xue *et. al.* [31] further studied

more general versions of the closest-pair problem and achieved similar results. For the MEB problem under the range-aggregate settings, Brass *et al.* are the first to investigate the problem in 2D space, along with other types of aggregate functions (like width and the size of convex hull) [10]. They showed that it is possible to build a data structure with $O(n \cdot polylog(n))$ pre-processing space/time and $O(polylog(n))$ query time.

All the aforementioned methods were designed for fixed dimensional space, and thus are not applicable to high dimensions. Actually, range aggregation has rarely been considered in high dimensions, except for a few results that may be viewed as loosely relevant. For example, Abbar *et al.* [1] studied the problem of finding the maximum diverse set for points inside a ball with fixed radius around a query point. Their ideas are seemingly useful to our problem. However, since their ball always has the same fixed radius, their techniques are not directly applicable. In fact, a main technical challenge of our problem is how to deal with the arbitrary radius and location of the query range, which is overcome by our multi-scale construction framework. Another related work by Aumüller *et. al.* [8] has focused on random sampling in a given range. The technique is also not directly applicable to IFP.

## 1.1 Overviews of the Main Ideas

Below we describe the main ideas of our approaches. For simplicity, in the following we ignore the fuzziness of the query range. We approach the AIFP query problem by first looking at an easier version: given ball $B$ and point $q$, find an approximate farthest point in $P \cap B$ to $q$, with the (strong) assumption that the radius of $B$ is a fixed constant $r_B > 0$, and that the distance between $q$ and its IFP in $P \cap B$ is within a range of $(d_{min}, d_{max}]$, where $d_{max} > d_{min} > 0$ are fixed constants. We call such a problem a constrained AIFP problem. We use a tuple $(r_B, d_{min}, d_{max})$ to denote such a constraint.

To solve the constrained AIFP problem, we develop a data structure for the *ball difference (BD) range query* problem, which is defined as follows: given two balls $B_{in}$ and $B_{out}$, find a point that lies in $P \cap B_{in} \setminus B_{out}$. With such a data structure, it is possible to reduce an AIFP query with constraint $(r_B, d_{min}, d_{max})$ to a series of BD queries. Below we briefly describe the idea. Let $r_0 = d_{min}$, and for $i = 1, 2, 3 \ldots$, let $r_i = (1 + \epsilon) r_{i-1}$, where $\epsilon > 0$ is an approximation factor. For $i = 0, 1, \ldots$, we try to determine whether there is a point in $P \cap B$ whose distance to $q$ is larger than $r_i$. Note that this can be achieved by a BD query with $B_{in} := B$ and $B_{out}$ being the ball centered at $q$ and with radius $r_i$. By iteratively doing this, eventually we will reach an index $j$ such that it is possible to find a point $p \in B \cap P$ that satisfies the condition of $\|p - q\| > r_j$, but no point lies in $P \cap B$ whose distance to $q$ is larger than $r_{j+1} = (1 + \epsilon) r_j$. Thus, $p$ is a $(1 - O(\epsilon))$-approximate farthest point to $q$ in $P \cap B$. From the definition of constrained AIFP query, it is not hard to see that this process finds the AIFP after at most $O(\log_{1+\epsilon} \frac{d_{max}}{d_{min}})$ iterations. Every BD data structure supports only $B_{in}$ and $B_{out}$ with fixed radii. This means that we need to build $O(\log_{1+\epsilon} \frac{d_{max}}{d_{min}})$ BD data structures for answering any AIFP query with constraint $(r_B, d_{min}, d_{max})$.

With the constrained AIFP data structure, we then extend it to a data structure for answering general AIFP queries. Our main idea is to use the aforementioned multi-scale construction technique to build a collection of constrained data structures, which can effectively cover (almost) all possible cases of the radius of $B$ and the farthest distance from $q$ to any point in $B \cap P$. More specifically, for any AIFP query, it is always possible to either answer the query easily without using any constrained data structures, or find a constrained data structure such that the AIFP query satisfies the constraint $(r_B, d_{min}, d_{max})$, and thus can be used to answer the AIFP query.

For AMEB query, we follow the main idea of Badoiu and Clarkson [9], and show that an AMEB query is reducible to a series of AIFP queries. More discussions are left to Section 5.

## 2 Constrained AIFP Query

In this section, we discuss how to construct a data structure to answer constrained AIFP queries. Particularly, given any ball $B$ and point $q$ satisfying the constraint $(r_B, d_{min}, d_{max})$,

- the radius of $B$ is $r_B$,
- the distance from $q$ to its farthest point to $P \cap B$ is within the range of $(d_{min}, d_{max}]$,

the data structure can find the AIFP to $q$ in $P \cap B$ in sub-linear time (with high probability).

In the following, we let $\epsilon > 0$ be an approximation factor, $\gamma > 0$ be a factor that controls the region fuzziness and $0 < \delta < 1$ be a factor controlling the query success probability. The main result of this section is summarized as the following lemma.

▶ **Lemma 6.** *Let $P$ be a set of $n$ points in $\mathbb{R}^d$. It is possible to build a data structure for $P$ with size $O_{\epsilon,\gamma}(dn^{1+\rho} \log \delta^{-1} \log(d_{max}/d_{min}))$ in $O_{\epsilon,\gamma}(dn^{1+\rho} \log \delta^{-1} \log(d_{max}/d_{min}))$ time, where $0 < \rho < 1$ is a real number depending on $\epsilon$ and $\gamma$, and the constants hidden in the big-O notation depend only on $\epsilon, \gamma$. Given any query $(B, q)$ that satisfies the constraint of $(r_B, d_{min}, d_{max})$, with probability at least $1 - \delta$, the data structure finds an $(\epsilon, \gamma)$-AIFP for $q$ in $P \cap B$ within time $O_{\epsilon,\gamma}(dn^{\rho} \log \delta^{-1} \log(d_{max}/d_{min}))$.*

In the following, we consider an AIFP query that satisfies constraint $(r_B, d_{min}, d_{max})$. As mentioned in last section, it is possible to reduce a constrained AIFP query to a series of *ball difference(BD) range queries*, which report a point in $P$ that lies (approximately) in $B_{in} \setminus B_{out}$ for a given pair of $\mathbb{R}^d$ balls ($B_{in}$ and $B_{out}$), or return NULL if no such point exists. Below, we describe the reduction using a ball-peeling strategy. We consider a series of balls $B_0, B_1, B_2, \ldots$ concentric at $q$ with an exponentially increasing radius. Let $\xi > 0$ be a to-be-determined approximation factor, and $B_0 := \mathcal{B}(q, d_{min})$ which is the ball centered at $q$ with radius $d_{min}$. For integer $i > 0$, let $B_{i+1} = B_i(1 + \xi)$ which is the ball obtained by enlarging the radius of $B_i$ by a factor of $(1 + \xi)$. [1] For $i = 0, 1, 2, \ldots$, repeatedly perform a BD query with $B_{in} := B$ and $B_{out} := B_i$, until an index $j$ is encountered such that the BD query reports a point $p_j$ that lies in $P \cap B \setminus B_j$, but returns NULL when trying to find a point in $P \cap B \setminus B_{j+1}$. If $\xi$ is a small enough constant, it is not hard to see that $p_j$ is a good approximation of the IFP to $q$ in $P \cap B$. Note that in this process, no more than $\log_{1+\xi}(d_{max}/d_{min})$ BD queries are required. This is because the distance between $q$ and any point in $B$ is at most $d_{max}$. Thus, it is not necessary to increase the radius of $B_{out}$ to be more than $d_{max}$ in the BD range query. The bound on the number of BD range queries then follows from the facts that the series of BD range queries starts with a $B_{out}$ ball of radius $d_{min}$ and each time the radius of $B_{out}$ is increased by a factor of $1 + \xi$. This process is similar to peel a constant portion of $B_{in}$ each time by $B_{out}$. See Figure 1 for an illustration.

The above discussion suggests that a constrained AIFP data structure can be built through (approximate) BD query data structures, which have the following definition. Let $\xi > 0$ be an approximation factor. A data structure is called $\xi$-error BD for a point set $P$, if given any balls $B_{in}$ and $B_{out}$, it answers the following query (with high success probability):

[1] Throughout this paper we use similar notations. Let $q$ be any point and $x > 0$ be real number. Then, $\mathcal{B}(q, x)$ denotes the ball centered at $q$ and with radius $x$. Let $B$ be any ball. For real number $y > 0$, we let $B(y)$ denote the ball obtained by enlarging (or shrinking if $y < 1$) the radius of $B$ by a factor of $y$.

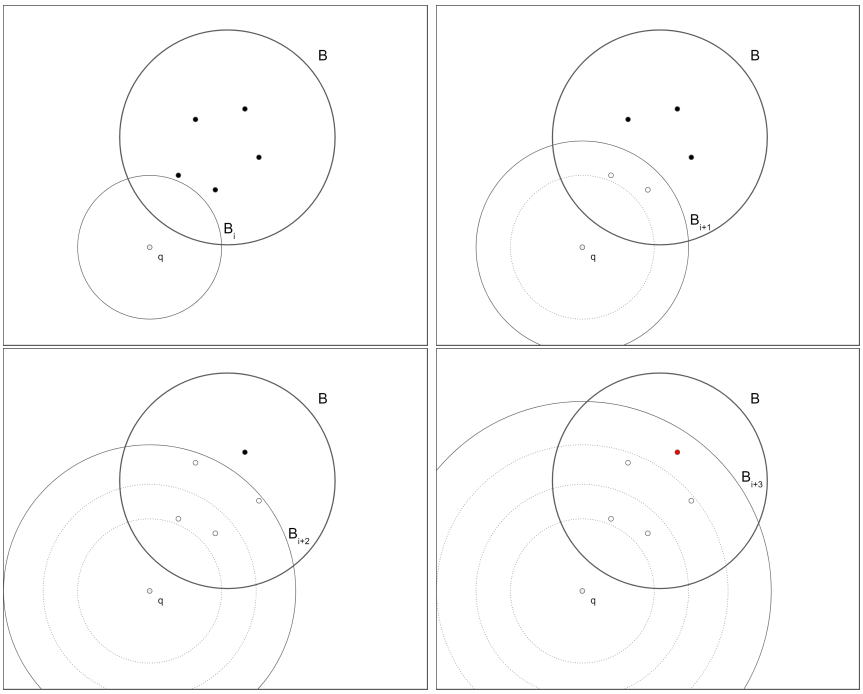



**Figure 1** An illustration of answering a constrained AIFP query using BD queries.

1. If there exists a point in $P \cap (B_{in} \setminus B_{out})$, the data structure returns a point in $P \cap (B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1}))$.
2. Otherwise, it returns a point in $P \cap (B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1}))$ or NULL.

The details of how to construct a $\xi$-error BD data structure is left to the next subsection. Below is the main result of the BD query data structure for $\xi > 0$, fixed constant $r_{in} > 0, r_{out} > 0$ and success probability controlling factor $0 < \delta < 1$.

▶ **Lemma 7.** *It is possible to build a $\xi$-error BD query data structure of size $O_\xi(dn^{1+\rho}\log\delta^{-1})$ in $O_\xi(dn^{1+\rho}\log\delta^{-1})$ time, where $0 < \rho < 1$ depends only on $\xi$. The query time of this data structure is $O_\xi(dn^{\rho}\log\delta^{-1})$. For any pair of query balls $B_{in}$ and $B_{out}$ with radius $r_{in}$ and $r_{out}$, respectively, the data structure answers the query with success probability at least $1-\delta$.*

Note that each BD query data structure works only for query balls $B_{in}$ and $B_{out}$ with fixed radii $r_{in}$ and $r_{out}$, respectively. This means that the constrained AIFP data structure should consist of multiple BD data structures with different values of $r_{in}$ and $r_{out}$.

From the above discussion, we know that a constrained AIFP data structure can be built by constructing a sequence of $\log(d_{max}/d_{min})$ BD data sturctures with $r_{in} := r_B$ and $r_{out}$ being $d_{min}, (1+\xi)d_{min}, (1+\xi)^2 d_{min}, \ldots$. Such a data structure will allow us to answer constrained AIFP queries using the ball peeling strategy.

Given any constants $\epsilon > 0$, $\gamma > 0$, $0 < \delta < 1$, and constraint $(r_B, d_{min}, d_{max})$, the following Algorithm 1 builds a constrained AIFP data structure for a given point set $P$. The data structure is simply a collection of BD query data structures.

With such a collection of BD query data structures, we can answer any constrained AIFP query satisfying $(r_B, d_{min}, d_{max})$ by applying the ball peeling strategy mentioned before. The algorithm is formally described as the Algorithm 2 below.

**Algorithm 1** Build-CAIFP($P; \epsilon, \gamma, \delta; r_B, d_{min}, d_{max}$).

**Input:** A $\mathbb{R}^d$ point set $P$ with cardinality $n$. Constants $\epsilon > 0, \gamma > 0, 0 < \delta < 1$. Constraint tuple $(r_B, d_{min}, d_{max})$.
**Output:** A number of BD-Query data structures built with different parameters.

1: Let $\xi = \min\{(1-\epsilon)^{-1/2}-1, \gamma\}$. Construct a sequence of real numbers $r_0, r_1, r_2, \ldots, r_m$, by letting $r_0 = d_{min}$, $m$ be the integer such that $r_0(1+\xi)^{m-1} < d_{max}$ and $r_0(1+\xi)^m \geq d_{max}$, $r_i = (1+\xi)r_{i-1}$ for $i = 1, 2, \ldots, m$, and $\delta' = \delta/m$.
2: **FOR** $i = 0, 1, 2, \ldots, m$, build a $\xi$-error BD query data structure for query balls with radii $r_{in} = r_B$ and $r_{out} = r_i$, with query success probability at least $1 - \delta'$.

**Algorithm 2** Query-CAIFP($B, q$).

**Input:** A constrained AIFP query $(B, q)$ with constraint $(r_B, d_{min}, d_{max})$.
**Output:** A point $p_{ans}$ that is an approximate farthest point in $B \cap Q$ to $p$, or NULL if no such point exists.

1: Initialize variable $p_{ans} \leftarrow$ NULL.
**Note:** In the following, we use $m$ and $r_i$ for $i = 0, 1, \ldots, m$ as in Algorithm 1.
2: **For** $i$ from 0 to $m$: Make a query $(B, B_{out,i})$ to the BD-Query data structure $BD_i$, by letting $B_{out,i} := \mathcal{B}(q, r_i)$. If the query answer is NULL, **Return** $p_{ans}$. Otherwise update $p_{ans}$ to be the query answer.
3: **Return** $p_{ans}$.

By some simple calculation, we know that the probability that all the BD queries in Algorithm 2 are successful is at least $1 - \delta$, and when this happens, the output point $p_{ans}$ is an $(\epsilon, \gamma)$-AIFP of $q$ in $B \cap P$. This is summarized as the following lemma.

▶ **Lemma 8.** *With probability at least $1 - \delta$, Algorithm 2 outputs a point $p_{ans} \in B(1+\gamma)$ such that for any $q \in B \cap P$, $\|p_{ans} - p\| \geq (1-\epsilon)\|q - p\|$.*

Next we analyze the space/time complexity of the AIFP scheme. The query data structure is a combination of $m = O_{\epsilon,\gamma}(\log(d_{max}/d_{min}))$ BD data structures. From the discussion of BD data structures (see Lemma 7), every BD query data structure we build has space/time complexity $O_{\epsilon,\gamma}(dn^{1+\rho}\log\delta^{-1})$ where $0 < \rho < 1$ depends only on $\epsilon, \gamma$. Each BD query takes $O_{\epsilon,\gamma}(dn^{\rho}\log\delta^{-1})$ time. Lemma 6 then follows.

## 2.1 The BD Query Scheme

In this subsection we present the BD query scheme. To our best knowledge, this is the first theoretical result to consider the BD range search problem. A very special case of BD query called the "annulus queries" where the two balls are co-centered is studied in [7]. Nonetheless, the technique is not directly applicable to general BD queries. Our BD range query scheme is based on the classic Locality Sensitive Hashing (LSH) technique [15, 5, 12] which has been a somewhat standard technique for solving the proximity problems in high dimensional space. The main idea of LSH is to utilize a family of hash functions (called an LSH family) that have some interesting properties. Given two points $p$ and $q$ in $\mathbb{R}^d$, if we randomly pick a function $h$ from the LSH family, the probability that the event of $h(p) = h(q)$ happens will be high if $\|p - q\|$ is smaller than a threshold value, and the probability for the same event will be lower if $\|p - q\|$ is larger. Such a property of the LSH family allows us to develop hashing and bucketing based schemes to solve similarity search problems in high dimensional space. Below is the definition of an LSH family.

▶ **Definition 9.** *Let $0 < r_1 < r_2$ and $1 > \mathsf{P}_1 > \mathsf{P}_2 > 0$ be any real numbers. A family $\mathcal{H} = \{h : \mathbb{R}^d \to U\}$, where $U$ can be any set of objects, is called $(r_1, r_2, \mathsf{P}_1, \mathsf{P}_2)$-sensitive, if for any $p, q \in \mathbb{R}^d$.*

1. *if $\|p - q\| \leq r_1$, then $Pr_{\mathcal{H}}[h(p) = h(q)] \geq \mathsf{P}_1$,*
2. *if $\|p - q\| > r_2$, then $Pr_{\mathcal{H}}[h(p) = h(q)] \leq \mathsf{P}_2$.*

It was shown in [12] that for any dimension $d$ and any $r > 0, c > 1$, an $(r, cr, \mathsf{P}_1, \mathsf{P}_2)$-sensitive family $\mathcal{H}$ exists, where $1 > \mathsf{P}_1 > \mathsf{P}_2 > 0$ depends only on $c$. Every hash function $h(p) : \mathbb{R}^d \to \mathbb{Z}$ maps a point $p$ in $\mathbb{R}^d$ to an integer, and $h(p)$ has the form $h(p) = \lfloor \frac{a \cdot p + b}{r} \rfloor$ for some $\mathbb{R}^d$ vector $a$ and integers $b, r$. It takes $O(d)$ time to sample a hash function $h$ from such a family and compute $h(p)$. Our data structure will make use of two such families. Let $\mathcal{H}_{in}$ be an $(r_{in}, (1+\xi)r_{in}, \mathsf{P}_1, \mathsf{P}_2)$-sensitive family, and $\mathcal{H}_{out}$ be a $((1+\xi)^{-1} r_{out}, r_{out}, \mathsf{P}_1, \mathsf{P}_2)$-sensitive family, where **$\mathbf{0 < \mathsf{P}_1, \mathsf{P}_2 < 1}$ are constants depending only on $\boldsymbol{\xi}$**, as described in [12]. Given any BD-query $(B_{in}, B_{out})$ with the centers of the balls being $o_{in}, o_{out}$ respectively, the family $\mathcal{H}_{in}$ helps us to identify points that are close enough to $o_{in}$ (and therefore lie in $B_{in}$), and $\mathcal{H}_{out}$ helps us to identify points that are far away enough from $o_{out}$ (and therefore lie outside of $B_{out}$).

**High level idea.** Our approach is based on a novel bucketing and query scheme that utilizes the properties of the LSH family. Before presenting the technical details, We first illustrate the high level idea. For convenience, we assume for now that the functions in $\mathcal{H}_{in}$ and $\mathcal{H}_{out}$ have range $\{0, 1\}$ (this is achievable by some simple modification to these hash function families). We use a randomized process to create a hybrid random hash function $S(\cdot)$ that maps any point in $\mathbb{R}^d$ to a bit string. Such a function $S(\cdot)$ is a concatenation of a number of hash functions drawn from $\mathcal{H}_{in}$ and $\mathcal{H}_{out}$. Given $p \in \mathbb{R}^d$, $S(\cdot)$ applies the aforementioned hash functions (drawn from $\mathcal{H}_{in}$ and $\mathcal{H}_{out}$) on $p$ to obtain a bit-string. With such a function $S(\cdot)$, consider comparing the bit-strings of $S(p), S(q)$ for points $p, q \in \mathbb{R}$. Intuitively, based on the properties of $\mathcal{H}_{in}$ and $\mathcal{H}_{out}$, we know that if $p, q$ are close enough, $S(p)$ and $S(q)$ should have many common bits in positions that are determined by functions from $\mathcal{H}_{in}$. Contrarily, if $p, q$ are far away, $S(p)$ and $S(q)$ should have only a few common bits in positions that are determined by functions from $\mathcal{H}_{out}$.

For every point $p \in P$, we use $S(p)$ to compute a bit-string label for $p$, and put $p$ into the corresponding buckets (*i.e.,* labeled with the same bit-strings). To answer a given BD query $B_{in}, B_{out}$ with centers of the balls being $o_{in}, o_{out}$, respectively, we compute $S(o_{in})$ and $S(o_{out})$. Note that, based on the above discussion, we know that if a point $p$ satisfies the condition of $p \in B_{in} \setminus B_{out}$, then $S(p)$ and $S(o_{in})$ should have many common bits in the positions determined by $\mathcal{H}_{in}$, and $S(p)$ and $S(o_{out})$ should have few common bits in the positions determined by $\mathcal{H}_{out}$. Thus, by counting the number of common bits in the labels, we can then locate buckets that are likely to contain points close to $o_{in}$ and far away from $o_{out}$, *i.e.,* points are likely to be in $B_{in} \setminus B_{out}$. To achieve the desired outcome, we will create multiple set of buckets using multiple random functions $S(p)$.

**Details of the Algorithms.** After understanding the above general idea, we now present the data structure and the query algorithm along with the analysis. Let $\mathsf{P}'_1 = (1 + \mathsf{P}_1)/2, \mathsf{P}'_2 = (1+\mathsf{P}_2)/2, \eta = (\mathsf{P}'_1 - \mathsf{P}'_2)/3, a = \lceil (2\mathsf{P}'_1 \ln 3)/\eta^2 \rceil$. $\mathsf{P}''_1 = 2^{-2a} \cdot 4/9$, $\mathsf{P}''_2 = 2^{-2a}/3, b = \lceil \log_{1/\mathsf{P}''_2} n \rceil$, $\rho = \frac{\ln 1/\mathsf{P}''_1}{\ln 1/\mathsf{P}''_2}$, and $c = \lceil n^\rho / \mathsf{P}''_1 \rceil$. Let $F_Z$ be a function that maps every element in $\mathbb{Z}$ randomly to 0 or 1, each with probability 1/2. The following Algorithm 3 shows how to construct a $\xi$-error BD range query data structure for any point set $P$ and radii $r_{in}$ and $r_{out}$. The data structure consists of $c$ groups of buckets, each created using a random function $S(p)$ that maps a point to a bit-string of total length $2ab$.

**Algorithm 3** CreateBuckets$(P, \xi, r_{in}, r_{out})$.

**Input:** A point set $P$. Parameters $\xi > 0, , r_{in} > 0, r_{out} > 0$.

**Output:** $\mathcal{G}_1, \mathcal{G}_2, \ldots \mathcal{G}_c$. Each $\mathcal{G}_i$ is a collection of *buckets* (*i.e.,* sets of points of $P$). Each bucket $G \in \mathcal{G}_i$ is labeled with *a bit-string*, which is a concatenation of sub-bit-strings $\text{LAB}_{in}(G,1)$, $\text{LAB}_{out}(G,1)$, $\text{LAB}_{in}(G,2)$, $\text{LAB}_{out}(G,2)$, ..., $\text{LAB}_{in}(G,b)$, $\text{LAB}_{out}(G,b)$. For every $p \in P$ and $i = 1, 2, \ldots, c$, $p$ appears in one of the buckets in $\mathcal{G}_i$.

1: Initialize $\mathcal{G}_i, i = 1, 2, \ldots, c$, as empty sets. Each $\mathcal{G}_i$ will be used as a container for buckets.

2: Randomly sample $abc$ functions from family $\mathcal{H}_{in}$, and also $abc$ functions from family $\mathcal{H}_{out}$. Denote these functions as $h_{in,i,j,k}$ and $h_{out,i,j,k}$, for integers $1 \le i \le a, 1 \le j \le b, 1 \le k \le c$. For every $h_{in,i,j,k}$, $h_{out,i,j,k}$ and every $p \in Q$, compute $F_Z(h_{in,i,j,k}(p))$ and $F_Z(h_{out,i,j,k}(p))$.

3: **FOR** $k$ from 1 to $c$:

- For every point $p \in P$, we create a bit-string $S(p)$ that concatenates $\text{LAB}_{in}(p,1), \text{LAB}_{out}(p,1), \text{LAB}_{in}(p,2), \text{LAB}_{out}(p,2), \ldots, \text{LAB}_{in}(p,b), \text{LAB}_{out}(p,b)$: For $j$ from 1 to $b$, let $\text{LAB}_{in}(p,j)$, $\text{LAB}_{out}(p,j)$ be a pair of bit-strings of length $a$, each with the $i$-th bit being $F_Z(h_{in,i,j,k}(p))$, $F_Z(h_{out,i,j,k}(p))$, respectively, for $i = 1, 2, \ldots a$.
- **IF** there is already a bucket $G$ in $\mathcal{G}_k$ with label $S(G) = S(p)$, **DO**: Put $p$ into $G$.
- **ELSE, DO**: Create a new bucket $G$ and put $G$ into $\mathcal{G}_k$, set the label of $G$ as $S(G) = S(p)$. Put $p$ into $G$.

With the BD range query data structure created by the Algorithm 3, we can use the Algorithm 4 below to answer a BD range query for any given pair of balls ($B_{in}$ and $B_{out}$). The main idea of the algorithm compute a bit-string label $S$ for the query, then examine points in buckets with labels that satisfy certain properties (*e.g.* should have enough common bits with $S$). Due to the fact that we label these buckets using functions from two LSH families, it can be shown that the chance for us to find a point in $B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$ from one of the examined buckets will be high if there exists a point in $B_{in} \setminus B_{out}$.

In the following we show the correctness of Algorithm 4. Consider the for loop in Step 1 of Algorithm 4 when answering a query $(B_{in}, B_{out})$. Using the notations from Algorithm 4, for any $k$ from 1 to $c$ in Step 1, we have the following lemma, which shows that if a point in $P$ lies in (or outside of) the query range, the number of common bits between its bucket label and the label computed from the query would likely (or unlikely) be high, respectively.

▶ **Lemma 10.** *Let $p \in P$ be a point that lies in $B_{in} \setminus B_{out}$, and $q \in P$ be a point that does NOT lie in $B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$. Let $S(p) = \text{LAB}_{in}(p,1), \text{LAB}_{out}(p,1), \text{LAB}_{in}(p,2), \text{LAB}_{out}(p,2), \ldots, \text{LAB}_{in}(p,b), \text{LAB}_{out}(p,b)$ and $S(q) = \text{LAB}_{in}(q,1), \text{LAB}_{out}(q,1), \text{LAB}_{in}(q,2), \text{LAB}_{out}(q,2), \ldots, \text{LAB}_{in}(q,b), \text{LAB}_{out}(q,b)$ be the labels of the bucket in $\mathcal{G}_k$ that contains $p$ and $q$, respectively. For any $j = 1, 2, \ldots, b$, the following holds.*

- $Pr[COM(\text{LAB}_{in}(p,j), \text{LAB}_{in}(o_{in},j)) \ge t_1 \wedge COM(\text{LAB}_{out}(p,j), \text{LAB}_{out}(o_{out},j)) \ge t_2] \ge 4/9$.
- $Pr[COM(\text{LAB}_{in}(q,j), \text{LAB}_{in}(o_{in},j)) \ge t_1 \wedge COM(\text{LAB}_{out}(q,j), \text{LAB}_{out}(o_{out},j)) \ge t_2] \le 1/3$.

**Proof.** Since $p \in B_{in} \setminus B_{out}$, we have $\|p - o_{in}\| \le r_{in}$ and $\|p - o_{out}\| \ge r_{out}$. For any hash function $h_1 \in \mathcal{H}_{in}$ and $h_2 \in \mathcal{H}_{out}$, $\Pr[h_1(p) = h_1(o_{in})] \ge \mathsf{P}_1$ and $\Pr[h_2(p) = h_2(o_{out})] \le \mathsf{P}_2$. Thus, we have $\Pr[F_Z(h_1(p)) = F_Z(h_1(o_{in}))] \ge (\mathsf{P}_1 + 1)/2$ and $\Pr[F_Z(h_2(p)) = 1 - F_Z(h_2(o_{out}))] \le (1 - \mathsf{P}_2)/2$. This means that for any $i = 1, 2 \ldots, a$, the probability that the

**Algorithm 4** BD-Query($B_{in}, B_{out}$).

**Input:** Two $\mathbb{R}^d$ balls: $B_{in}$ with center $o_{in}$ and radius $r_{in}$, and $B_{out}$ with center $o_{out}$ and radius $r_{out}$. Assume that collections $\mathcal{G}_1, \mathcal{G}_2, \ldots \mathcal{G}_c$ have already been generated by algorithm CreateBucket.

**Output:** A point $p \in Q$, or NULL.

Note: The algorithm probes a number of points in the buckets of $\mathcal{G}_1, \mathcal{G}_2, \ldots \mathcal{G}_c$ until a suitable point is found as the output, or terminates and returns NULL when no such point can be found or the number of probes exceeds a certain limit.

1: Do the following, but terminate and return NULL when $3c$ points are examined: **FOR** $k$ from 1 to $c$:
- Create a bit string $S$ that concatenates $\text{LAB}_{in}(o_{in}, 1)$, $\text{LAB}_{out}(o_{out}, 1)$, $\text{LAB}_{in}(o_{in}, 2)$, $\text{LAB}_{out}(o_{out}, 2)$, …, $\text{LAB}_{in}(o_{in}, b)$, $\text{LAB}_{out}(o_{out}, b)$: For $j$ from 1 to $b$, let $\text{LAB}_{in}(o_{in}, j)$, $\text{LAB}_{out}(o_{out}, j)$ be a pair of bit strings of length $a$, with the $i$-th bit of each string being $F_Z(h_{in,i,j,k}(o_{in}))$, $1 - F_Z(h_{out,i,j,k}(o_{out}))$, respectively, for $i = 1, 2, \ldots a$.
- Create a bit string $S'$ that concatenates $\text{LAB}'_{in}(1), \text{LAB}'_{out}(1), \text{LAB}'_{in}(2), \text{LAB}'_{out}(2), \ldots,$ $\text{LAB}'_{in}(b), \text{LAB}'_{out}(b)$: Each of these sub bit strings is a random bit string of length $a$, drawn uniformly randomly from $\{0,1\}^a$.
- If there exists some integer $j$ such that $\text{COM}(\text{LAB}_{in}(o_{in}, j), \text{LAB}'_{in}(j)) < t_1$ or $\text{COM}(\text{LAB}_{out}(o_{out}, j), \text{LAB}'_{out}(j)) < t_2$, where $\text{COM}(x, y)$ counts the number of common digits of 2 bit strings $x, y$, $t_1 = \mathsf{P}'_1 a - \eta a, t_2 = (1 - \mathsf{P}_2)a/2 - \eta a$ **CONTINUE.**
- If there is no bucket in $\mathcal{G}_k$ that is labeled with $S'$, **CONTINUE.**
- Examine all the points in the bucket $G$ in $\mathcal{G}_k$ that is labeled with $S'$. Stop when there a point $p \in G$ such that $p \in B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$. Return $p$.

2: Return NULL if no point is returned in the above process.

$i$-th bit of $\text{LAB}_{in}(p, j)$ is the same as that of $\text{LAB}_{in}(o_{in}, j)$ is at least $\mathsf{P}'_1 = (\mathsf{P}_1 + 1)/2$. Since the hash functions to determine each of the bits are drawn independently, an estimation of $X = \text{COM}(\text{LAB}_{in}(p, j), \text{LAB}_{in}(o_{in}, j))$ can be obtained by $\Pr[F_Z(h_1(p)) = F_Z(h_1(o_{in}))] \geq (\mathsf{P}_1 + 1)/2$ using the concentration inequalities for binomial distributions. Using a variant of the Chernoff inequalities from [22], we have

$$\Pr[X \leq \mathsf{P}'_1 a - \eta a] \leq e^{-(\eta a)^2/(2\mathsf{P}'_1 a)}.$$

From the definition of the parameters, we know that $\Pr[X \leq \mathsf{P}'_1 a - \eta a] \leq 1/3$ (by simple calculation). Thus, we have $\Pr[X \geq t_1] \geq 2/3$.

Let $Y = \text{COM}(\text{LAB}_{out}(p, j), \text{LAB}_{out}(o_{out}, j))$. From $\Pr[F_Z(h_2(p)) = 1 - F_Z(h_2(o_{out}))] \leq (1 - \mathsf{P}_2)/2$ and using a similar argument as above, we can also obtain $\Pr[Y \geq t_2] \geq 2/3$ (the details are omitted). Since the hash functions are drawn independently, we have $\Pr[X \geq t_1 \wedge Y \geq t_2] \geq 4/9$.

In the following we discuss the case that $q \notin B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$. This means either $\|q - o_{in}\| \geq (1+\xi) r_{in}$ or $\|q - o_{out}\| \leq (1+\xi)^{-1} r_{out}$. We first consider the case $\|q - o_{in}\| \geq (1+\xi) r_{in}$. For any hash function $h_1 \in \mathcal{H}_{in}$, $\Pr[h_1(q) = h_1(o_{in})] \leq \mathsf{P}_2$. Thus, we have $\Pr[F_Z(h_1(q)) = F_Z(h_1(o_{in}))] \leq (\mathsf{P}_2 + 1)/2$. This means that for any $i = 1, 2 \ldots, a$, the probability that the $i$-th bit of $\text{LAB}_{in}(q, j)$ is the same as that of $\text{LAB}_{in}(o_{in}, j)$ is at most $\mathsf{P}'_2 = (\mathsf{P}_2 + 1)/2$. Again, we use a concentration inequality to obtain an estimation of $X = \text{COM}(\text{LAB}_{in}(q, j), \text{LAB}_{in}(o_{in}, j))$. Using a variant of the Chernoff inequalities from [22], we have

$$\Pr[X \geq \mathsf{P}'_2 a + \eta a] \leq e^{-(\eta a)^2/(2\mathsf{P}'_2 a + \eta a/3)}.$$

Note that $a = (2\mathsf{P}'_1 \ln 3)/\eta^2 \geq ((2\mathsf{P}'_2 + \eta/3)\ln 3)/\eta^2$, which implies that $e^{-(\eta a)^2/(2\mathsf{P}'_2 a + \eta a/3)} \leq 1/3$ (by simple calculation). Thus, we have $\Pr[X \geq \mathsf{P}'_2 a + \eta a] \leq 1/3$. Also, since $\mathsf{P}'_2 a + \eta a < \mathsf{P}'_1 a - \eta a = t_1$, we get $\Pr[X \geq t_1] \leq 1/3$. This immediately implies that $\Pr[\mathrm{COM}(\mathrm{LAB}_{in}(q,j), \mathrm{LAB}_{in}(o_{in},j)) \geq t_1 \wedge \mathrm{COM}(\mathrm{LAB}_{out}(q,j), \mathrm{LAB}_{out}(o_{out},j)) \geq t_2] \leq 1/3$.

The argument for the case $\|q - o_{out}\| \leq (1+\xi)^{-1} r_{out}$ is similar. Thus, we omit it here. This completes the proof. ◀

From the above lemma, we can conclude the following by basic calculation. For any $k$ from 1 to $c$ in Step 1 of Algorithm 4 (if the loop is actually executed), let $p \in P$ be a point that lies in $B_{in} \setminus B_{out}$, and $q \in P$ be a point that does NOT lie in $B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$, we have the following.

▶ **Lemma 11.** *Let $G_p$ and $G_q$ be the buckets in $\mathcal{G}_k$ that contain $p$ and $q$, respectively. The probability for $G_p$ to be examined is no smaller than $2^{-2ab}(4/9)^b = (\mathsf{P}''_1)^b$, and the probability for the event "ALL such $G_p$ for $k$ from 1 to $c$ are NOT examined" is at most $(1-(\mathsf{P}''_1)^b)^c \leq 1/e$. The probability for $G_q$ to be examined is no larger than $2^{-2ab}(1/3)^b = (\mathsf{P}''_2)^b \leq 1/n$.*

With the above lemma, we can obtain the following lemma using an argument similar to [15] for near neighbor search with LSH. This proves the correctness of the query scheme.

▶ **Lemma 12.** *If there exists a point in $P$ that lies in $B_{in} \setminus B_{out}$, with probability at least $1/4$, Algorithm 4 reports a point in $P$ that lies in $B_{in}(1+\xi) \setminus B_{out}((1+\xi)^{-1})$.*

The complexity of the data structure is $O(dabcn)$, which is $O_\xi(dn^{1+\rho}\log n)$ from $a = O_\xi(1), b = O_\xi(\log n)$ and $c = O_\xi(n^\rho)$, and $\rho$ and the constant hidden in the big-O notation depends only on $\xi$. The query time is $O(abcd)$, which is $O_\xi(dn^\rho \log n)$. To achieve $1-\delta$ success probability, it suffices to concatenate $O(\log \delta)$ such data structures together.

Due to space limit. we leave the proof of the above 2 lemmas and the full proof of Lemma 7 to the full version of the paper.

## 3 Multi-scale Construction

In this section, we present the *multi-scale construction* method, which is a standalone technique with potential to be used to other high dimensional range-aggregate query problems.

The multi-scale construction method is motivated by several high dimensional geometric query problems that share the following common feature: they are challenging in the general settings, but become more approachable if some key parameters are known in advance. The AIFP query problem discussed in this paper is such an example. In the previous section, we have shown how to construct an AIFP data structure if we fix the size of the query ball and know that the farthest distance lies in a given range.

The basic ideas behind multi-scale construction are the follows. Firstly, we know that if a problem is solvable when one or more key parameters are fixed, a feasible way to solve the general case of the problem is to first enumerate all possible cases of the problem defined by (the combinations of) the values of the parameters. Then, solve each case of the problem, and finally obtain the solution from that of all the enumerated cases. The multi-scale construction method follows a similar idea. More specifically, to obtain a general AIFP query data structure, the multi-scale construction method builds a set of constrained AIFP query data structures that cover all possible radii of $B$ and farthest distance value. Secondly, since it is impossible to enumerate the infinite number of all possible values for these parameters, our idea is to sample a small set of fixed radii (based on the distribution of the points in $P$)

and build constrained AIFP data structures only for the set of sampled values. This will certainly introduce errors. However, good approximations are achievable by using a range cover technique.

Below we first briefly introduce two key ingredients of our method, Aggregation Tree and Range Cover, and then show how they can be used to form a multi-scale construction.

## 3.1 Aggregation Tree and Range Cover

In this subsection, we briefly introduce the two components of the multi-scale construction scheme: the *aggregation tree* and the *range cover* data structure. We first introduce aggregation tree, which is used in [18] as an ingredient of the range cover data structure. It is essentially a slight modification of the Hierarchical Well-Separated Tree (HST) introduced in [14]. Below is the definition of an aggregation tree: (1) Every node $v$ (called *aggregation node*) represents a subset $P(v)$ of $P$, and the root represents $P$; (2)Every aggregation node $v$ is associated with a representative point $re(v) \in P(v)$ and a size $s(v)$. Let $Dia(P(v))$ denotes the diameter of $P(v)$, $s(v)$ is a polynomial approximation of $Dia(P(v))$: $Dia(P(v)) \leq s(v)$, and $\frac{s(v)}{Dia(P)}$ is upper-bounded by a polynomial function $\mathcal{P}_{HST}(n, d) \geq 1$ (called *distortion polynomial*); (3) Every leaf node corresponds to one point in $P$ with size $s(v) = 0$, and each point appears in exactly one leaf node; (4) The two children $v_1$ and $v_2$ of any internal node $v$ form a partition of $v$ with $\max\{s(v_1), s(v_2)\} < s(v)$; and (5) For every aggregation node $v$ with parent $v_p$, $\frac{s(v_p)}{r_{out}}$ is bounded by the distortion polynomial $\mathcal{P}_{HST}(n, d) \geq 1$, where $r_{out}$ is the minimum distance between points in $P(v)$ and points in $P \setminus P(v)$.

The above definition is equivalent to the properties of HST in [14], except that we have an additional distortion requirement (Item 5). See Figure 2 for example of an aggregate tree.

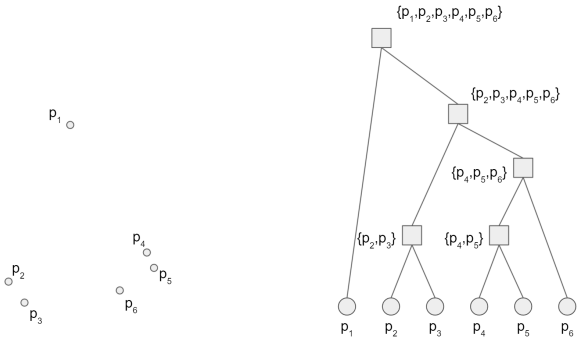



**Figure 2** An illustration of an aggregation tree built for 6 points.

An aggregation tree can be constructed in $O(dn \log^2 n)$ time using the method in [14]. It is proved in [14] that the distortion polynomial is $\mathcal{P}_{HST}(n, d) = dn$. In the rest of the paper, we always assume that the distortion of an aggregation tree is $\mathcal{P}_{HST}(n, d) = dn$.

In the following, we briefly introduce range cover. Range cover is a technique proposed in [18] for solving the truth discovery problem in high dimensions. We utilize it in a completely different way to form a multi-scale construction for the AIFP query problem. Below is the algorithm (Algorithm 5). Given an aggregation tree $T_p$ and real number parameters $\Delta \geq 8n$ and $0 < \lambda < 1$ (whose values will be determined later), the range cover algorithm creates a number of buckets for the nodes of $T_P$. Each bucket $B_t$ is associated with an interval of real

**Algorithm 5** RangeCover($T_P; \lambda, \Delta$).

**Input:** An aggregation tree $T_P$ built over a set $P$ of $n$ points in $\mathbb{R}^d$; controlling factors $0 < \lambda < 1$ and an integer $\Delta \geq 4\mathcal{P}_{HST}(n, d)$.

**Output:** A number of buckets, where each bucket stores a number of tree nodes. Each bucket $B_t$ is indexed by an integer $t$ and associated with an interval $((1+\lambda)^t, (1+\lambda)^{t+1}]$.

1: For every integer $t$ create an empty bucket $B_t$ associated with interval $((1+\lambda)^t, (1+\lambda)^{t+1}]$. (Note that $B_t$ will not be actually created until some tree node $v$ is inserted into it.)

2: For every non-root node $v$ of $T_P$, let $v_p$ be its parent in $T_P$, $r_H$ be $s(v_p)/\lambda$, and $r_L$ be $\max\{s(v)/\lambda, s(v_p)/\Delta\}$. Do

- For every integer $t$ satisfying inequality $r_L \leq (1+\lambda)^t < r_H$, insert $v$ into bucket $B_t$.

number $((1+\lambda)^t, (1+\lambda)^{t+1}]$. If a value $r$ lies in the interval of a bucket $B_t$, it can be shown that the diameter of every aggregation node $v$ is small compared to $r$, and thus all points in $P(v)$ can be approximately viewed as one "heavy" point located at the representative point $re(v)$. Intuitively, every bucket from the range cover algorithm provides a view of $P$ when observed from a distance $r$ in the range of the bucket, where each node in the bucket represents a "heavy" point that is formed by the aggregation of a set of close (compared to the observing distance) clusters of points in $P$. Thus, the buckets of the range cover provides views of the input point set at different scales of observing distances (see Figure 3 for an illustration). The size of the output data structure is only $O(n \log n\Delta)$, as shown in [18].

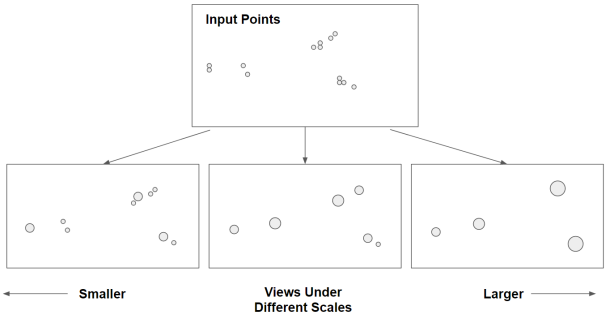



**Figure 3** An illustration of range cover. The nodes in every bucket can be viewed as "heavy" points yielded by the aggregation of a set of close points. Every bucket provides a view of the input point set when observed from a certain distance. All the buckets jointly form a complete set of views of the input points at all possible scales.

Note that for many problems, fixing some key parameters also means fixing the "observing distance" of $P$ from the perspective of solving the problem. This allows us to solve the problem based on the view of $P$ provided by the bucket associated with the corresponding observation distance. We will show that this idea also applies to the AIFP problem.

## 3.2 Multi-scale Construction for AIFP

In this subsection, we use AIFP problem as an example to show how to implement multi-scale construction using the range cover data structure.

We first observe that every bucket of the range cover can be used to solve a constrained AIFP problem (with the proof given later). Given an AIFP query $(B, q)$, if the (approximate) distance from $q$ to a point in $P \cap B$ is known and falls in the interval of $((1+\lambda)^t, (1+\lambda)^{t+1}]$,

then the (approximate) distance from $q$ to a point of $B_t \cap B$ (where every node $v$ in $B_t$ is viewed as a "heavy" point located at $re(v)$) is an AIFP of $q$ in $P \cap B$. This means that $B_t$ provides a good "sketch" of $P$ that allows more efficient computation of the AIFP of $q$ in $P \cap B$. This observation leads to the main idea of the multi-scale construction method. To obtain a general AIFP query data structure, for every bucket $B_t$, we construct a constrained AIFP data structure for $B_t$ (viewed as a set of "heavy" points), exploiting the assumption that the farthest distance is in the interval of $((1+\lambda)^t, (1+\lambda)^{t+1}]$. To answer a general AIFP query, we can find the AIFP for every bucket by querying the constrained AIFP data structures associated with the bucket. In this way, we can compute AIFPs for all possible radii. When answering a general AIFP query, we first determine an approximate farthest distance of $q$ to $P \cap B$, and then query the appropriate constrained AIFP data structures. Despite the necessity of building multiple constrained data structures, the complexity of the multi-scale construction is not high, as the total number of nodes in all buckets is only $\tilde{O}(n)$.

However, the above idea is hard to implement, because each bucket $B_t$ is only responsible for a small range $((1+\lambda)^t, (1+\lambda)^{t+1}]$ of the possible farthest distance from $q$ to $P \cap P$. This means that we need an accurate estimation of this distance when answering the query, which is almost as hard as the query itself. We resolve this issue by merging multiple consecutive buckets into a larger one. The resulting bucket can account for a larger range of the possible farthest distances. We then build a constrained data structure for each bucket.

This leads to the following Multi-Scale algorithm. Let $\Gamma \geq 1$ be an integer constant to be determined, and $\mathcal{A}$ be an algorithm for building a constrained data structure. In this algorithm, for each integer $t$, we try to merge the aggregation nodes in buckets $B_t, B_{t+1}, \ldots, B_{t+\Gamma}$ from the range cover (recall that these buckets are associated with farthest distance ranges $((1+\lambda)^t, (1+\lambda)^{t+1}], ((1+\lambda)^{t+1}, (1+\lambda)^{t+2}] \ldots, ((1+\lambda)^{t+\Gamma}, (1+\lambda)^{t+\Gamma+1}]$, respectively) into one bucket $B_t^+$ that could account for a larger range $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$. We then use $\mathcal{A}$ to build a data structure $\mathcal{S}_t$ for every bucket $B_t^+$ (by viewing every node in $B_t^+$ as a point).

**Algorithm 6** Multi-Scale($T_P; \lambda, \Delta, \Gamma; \mathcal{A}$).

**Input:** An aggregation tree $T_P$ built over a set $P$ of $n$ points in $\mathbb{R}^d$; controlling factors $0 < \lambda < 1$, integer $\Delta \geq 4\mathcal{P}_{HST}(n,d) = 4dn$, and integer $\Gamma \geq 1$. A routine $\mathcal{A}$ which builds a constrained data structure for any given bucket $B_t$ and point set $re(B_t^+) := \{re(v) \mid v \in B_t^+\}$.
**Output:** A number of buckets, with each storing a number of tree nodes. Each bucket $B_t^+$ is indexed by an integer $t$ and associated with an interval $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$. Each bucket $B_t^+$ is associated with data structure $\mathcal{S}_t$ built by $\mathcal{A}$.

1: Create a collection of buckets $\{B_t\}$ by calling RangeCover($T_P; \lambda, \Delta(1+\lambda)^\Gamma$).
2: For each integer $t$ create an empty bucket $B_t^+$ associated with $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$.
3: For every non-root node $v$ of $T_P$, enumerated in a bottom-up manner in $T_P$ so that the children of a node is always visited earlier than the parent node, put $v$ into $B_t^+$ for every $t$ such that the following is satisfied:
   - $s(v) \leq \lambda(1+\lambda)^t$, $v$ appears in $B_{t'} \in \{B_t\}$ for some $t \leq t' \leq t+\Gamma$, and none of $v$'s descendants are put in $B_t^+$ previously.
4: For every non-empty bucket $B_t^+$, create a data structure $\mathcal{S}_t$ using $\mathcal{A}$ for the point set $re(B_t^+) := \{re(v) \mid v \in B_t^+\}$.

For better understanding of this scheme, we first briefly discuss the geometric properties of the buckets created by Algorithm 6. Intuitively speaking, the aggregation nodes of every bucket provide a sketch of *almost* the whole input point set $P$, with the exception being points that satisfying some special isolation property. This can be briefly described as follows:

(1) The diameter of each the aggregation node (viewed as a point set) should be small to the observation distances; (2) The aggregation nodes are mutually disjoint; and (3) Every point $p \in P$ is either in one of the nodes in the bucket, or it is in an aggregation node (not in the bucket) whose distance to other nodes is large. These properties are formalized as follows. (We leave the proofs of all the following claims and lemma to the full version of the paper.)

▷ Claim 13. Let $v$ be any aggregation node $v$ in a created bucket $B_t^+$. Then, $s(v) \leq \lambda(1+\lambda)^t$.

▶ **Lemma 14.** *For any $p \in P$ and and bucket $B_t^+$ created by Algorithm 6, one of the following holds:*

1. *There exists exactly one aggregation node $v \in B_t^+$ such that $p \in P(v)$.*
2. *Either $B_t^+$ is empty or there exists no aggregation node $v \in B_t^+$ such that $p \in P(v)$. There exists an aggregation node $v'$ in $T_P$ such that $s(v')/\lambda \leq (1+\lambda)^t$. Furthermore, let $q$ be any point in $P \setminus P(v')$, then $\|p-q\| > (\Delta/dn)(1+\lambda)^{t+\Gamma}$.*

Although the sketch does not fully cover $P$, in many problems (including AIFP) these points are either negligible or easy to handle by other means due to their special properties.

From [18], we know that the running time of Algorithm 5 and the space complexity of the output data structure is $O_\lambda(n \log n\Delta)$ (where the hidden constant in the big-O notation depends only on $\lambda$). Algorithm 6 essentially merges $\Gamma+1$ consecutive buckets $B_t$, $B_{t+1}$, …, $B_{t+\Gamma}$ created by Algorithm 5 into one bucket $B_t^+$. Thus, we have the following lemma.

▶ **Lemma 15.** *Excluding the time it takes for $\mathcal{A}$ to process each $B_t^+$ in Step 4, the running time of Algorithm 6 and the total number of nodes in all buckets is $O_\lambda(\Gamma^2 n \log n\Delta)$, where the hidden constant in big-O notation depends only on $\lambda$.*

We conclude this subsection by providing a key lemma showing that, given a constrained AIPF query $(B, q)$ satisfying constraint $(r_B, d_{min}, d_{max})$ with $d_{min} \leq r_B \leq d_{max}$, if there is a bucket $B_t^+$ such that $(1+\lambda)^{t+1}/(1-\lambda) < d_{min} < d_{max} \leq (1+\lambda)^{t+\Gamma} - (2+2/\lambda)r_B$ (*i.e.* the range $[d_{min}, d_{max}]$ falls in interval $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$ with some gap), then, with an easy-to-handle exception, an AIFP to $q$ in $B_t^+$ (by viewing every node of $B_t^+$ as one point) in (slightly enlarged) range $B$ is also an AIFP of $q$ to $P \cap B$. Formally, let $re(B_t^+) := \{re(v) \mid v \in B_t^+\}$. Let $p_t$ be the farthest point to $q$ in $re(B_t^+) \cap B(1+\lambda)$ if $re(B_t^+) \cap B(1+\lambda) \neq \emptyset$, and $p$ be a $(\lambda/6, \lambda/6)$-AIFP of $q$ in $re(B_t^+) \cap B(1+\lambda)$ [2]. Let $p_N$ be a $(1+\lambda)$-approximate nearest neighbor of $q$ in $P$.

▶ **Lemma 16.** *One of the following holds: (1) $p_N$ is a $(2\lambda, 2\lambda)$-AIFP of $q$ in $P \cap B$, or (2) $p_t$ exists and $(1+\lambda)^t \leq \|q-p_t\| \leq (1+\lambda)^{t+\Gamma+1}$, and $p$ is a $(2\lambda, 2\lambda)$-AIFP of $q$ in $P \cap B$.*

The above lemma implies that, in Algorithm 6, if routine $\mathcal{A}$ builds a constrained AIFP data structure for farthest distance lies in interval $[(1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$, then either this data structure can be used to answer any query with constraint $(r_B, d_{min}, d_{max})$ (with other parameters, like $r_B$ and the approximate factors for constrained AIFP, set properly), or the AIFP query can be solved easily using a nearest neighbor search. In the following section, we will show how to build a general AIFP query data structure through multi-scale construction by selecting appropriate parameters. With the multi-scale data structure (together with some auxiliary data structures), we can answer an AIFP query by (1) obtaining a rough estimation of the farthest distance, and (2) querying the bucket corresponding to the estimated range.

[2] Note $p$ could be NULL here. This could happen when bucket $B_t^+$ is empty or $re(B_t^+) \cap B(1+\lambda) = \emptyset$.

## 4 General AIFP Query

In this section, we present a general $(\epsilon, \gamma)$-AIFP query scheme. In the following, let $B$ be a closed ball with radius $r_B > 0$ and $q$ be an arbitrary point. Let $0 < \epsilon < 1$ and $0 < \gamma < 1$ be any pair of constants and $\lambda := \min(\epsilon, \gamma)/512$. We assume that $r_B$ is $\lambda$-*aligned*, which means that $r_B = (1+\lambda)^t$ for some integer $t$. The alignment assumption makes it easier to implement the multi-scale construction. Note that if $r_B$ is not $\lambda$-aligned, we can always enlarge $B$ a little to make $r_B$ $\lambda$-aligned and still obtain a good approximation with carefully chosen parameters.

Our main idea is to convert each query $(B, q)$ into one or more AIFP queries $(B', q)$ such that it is possible to find a lower bound $d_{min}$ and an upper bound $d_{max}$ on the farthest distance between $q$ and a point in $B' \cap P$. With such bounds, the AIFP can then be found by querying a pre-built constrained AIFP data structure. To ensure efficiency, the gap between $d_{min}$ and $d_{max}$ cannot be too large, *i.e.,* $d_{max}/d_{min}$ should be bounded by a polynomial of $n$ and $d$. Since the complexity of a constrained data structure depends on $d_{max}/d_{min}$, a small gap will also enable us to control the size of the data structure. We start with a simple claim.

▷ Claim 17. If the distance between $q$ and the center $o_B$ of $B$ is very large compared to $r_B$, *i.e.* $\|q - o_B\| \geq (3+\gamma)\epsilon^{-1} r_B$, then any point in $B(1+\gamma/2)$ is an $(\epsilon, \gamma)$-AIFP of $q$ in $P \cap B$.

This claim suggests that we can safely assume that the farthest distance between $q$ and $P \cap B$ is not too large (compared to $r_B$), as otherwise the AIFP can be easily found with a nearest neighbor query. This helps us establish an upper bound on the farthest distance. In the following, we let $d_{max} := (4+2\gamma)\epsilon^{-1} r_B$, and assume that $\|q - o_B\| \leq (3+\gamma)\epsilon^{-1} r_B$. From simple calculation, this implies that for any $p \in P \cap B(1+\gamma)$, we have $\|p - q\| \leq d_{max}$.

Next, we try to find a lower bound for the farthest distance. Since the full argument will be rather complicated, we will thus describe only the general idea here, and leave the details to the full version of the paper. We start with a simple case.

**Case 1:** $\|q - o_B\| \geq (1+\gamma/64) r_B$, *i.e.* $q$ does not lie in $B$ and is not very close to the boundary of $B$. Then, clearly the farthest distance from $q$ to any point in $P \cap B$ is at least $(\gamma/64) r_B$. Recall that we have obtained an upper bound $d_{max} := (4+2\gamma)\epsilon^{-1} r_B$. The ratio $d_{max}/(\gamma r_B/64)$ is clearly bounded by a polynomial of $n, d$.

In the following, we assume that $\|q - o_B\| \leq (1+\gamma/64) r_B$, which means that $q$ lies in $B$ or is very close to the boundary of $B$. Let $p_N \in P$ be a 2-nearest neighbor of $q$ in $P$ (*i.e.* for any $p' \in P$, $\|p_N - q\| \leq 2\|p' - q\|$) , and denote $r_N := \|p_N - q\|$. We discuss another simple case.

**Case 2:** $r_N > \gamma r_B/64$. Since $p_N$ is a 2-approximate nearest neighbor, we conclude that the distance from $q$ to any point in $P \cap B$ is at least $\gamma r_B/128$. Again, the ratio $d_{max}/(\gamma r_B/128)$ is well bounded.

In the following, we assume that $r_N \leq \gamma r_B/64$. In order to find a good lower bound on the farthest distance, our general strategy is to examine points around $p_N$ and see whether there exists a point $p'$ whose distance to $q$ is sufficiently large, but still upper bounded by $(\gamma/64) r_B$. The upper bound $(\gamma/64) r_B$ ensures that $p' \in B(1+\gamma/16)$, *i.e.* $p'$ "approximately" lies in $B$, and because of the fuzziness of the region, $\|p' - p\|$ can be used as a lower bound on the farthest distance. To efficiently implement this idea, we make use of an aggregation tree $T_P$ with distortion polynomial $\mathcal{P}_{HST}(n, d) = nd$. Later, we will use this $T_P$ to construct the query data structure. Let $v_N$ denote the leaf node of $T_P$ that correspond to singleton set $\{p_N\}$. We walk along the tree path from $v_N$ to the root of $T_P$, and examine the nodes

(and their associated point sets) on this path. Denote by $v$ the highest (closest to the root) node of $T_P$ such that $p_N \in P(v)$ and $s(v) + r_N \leq \gamma r_B/64$. Note that since $r_N \leq \gamma r_B/64$ and $s(v_N) = 0$, such a node $v$ must exist. Since $s(v)$ is an upper bound on the diameter of $P(v)$, every point $p' \in P(v)$ satisfies inequality $\|p' - q\| \leq \gamma r_B/64$, and thus $p' \in B(1+\gamma/16)$. See Figure 4 for a illustrations of the relations between $q$ and points in $P(v)$. We consider two more cases, depending on the value of $s(v) + r_N$.

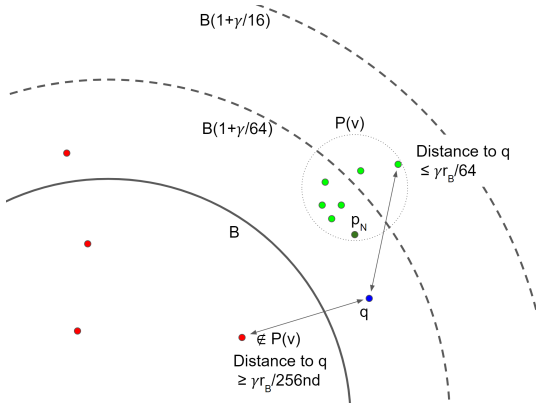



**Figure 4** An illustration of points in $P(v)$, $P \setminus P(v)$ and $q$.

**Case 3:** $s(v) + r_N > \gamma r_B/512n^2d^2$. This means that $s(v) + r_N$ is at least $r_B$ divided by a polynomial of $n, d$. Note that from the low distortion property of $T_P$, $s(v)$ is a polynomial approximation of the diameter of $P(v)$. It is possible to show by standard inequality arguments that, $s(v) + r_N$ is a polynomial approximation of $q$'s farthest distance to any point in $P(v)$. Since $P(v) \subset B(1 + \gamma/16)$, we conclude that the farthest distance from $q$ to any point in $B(1 + \gamma/16) \cap P$ is lower bounded by $r_B$ divided by a polynomial of $n, d$. A careful calculation gives us an estimated lower bound $r_B/2048n^3d^3$. Since an upper bound for the farthest distance is $d_{max} = (4 + 2\gamma)\epsilon^{-1}r_B$, the quality of this lower bound is satisfactory because the gap between it and $d_{max}$ is a polynomial of $n, d$.

**Case 4:** $s(v) + r_N \leq \gamma r_B/512n^2d^2$. We show that in this case, either there is a lower bound on the farthest distance, or the query can be reduced to another AIFP query where the range of farthest distances can be bounded. Let $v_p$ be the parent of $v$ (if $v$ is the root of $T_P$, the following result still holds), then $s(v_p) + r_N \geq \gamma r_B/64$. This means that $s(v_p)$ is much larger than $s(v)$. From the property of $T_P$, we know that for any $p_{out} \in P \setminus P(v)$, the ratio $s(v_p)/\|p_{out} - p_N\|$ is upper bounded by the distortion polynomial $\mathcal{P}_{HST}(n, d) = nd$. In the current case, $s(v_p)/s(v)$ is indeed much larger than $nd$. Thus $\|p_{out} - p_N\|$ is very large compared to $s(v)$, which gives us a lower bound on $\|p_{out} - q\|$. In fact, by a careful calculation, it is possible to show that $\|q - p_{out}\| \geq \gamma r_B/256nd$ for any $p_{out} \in P \setminus P(v)$. Furthermore, since the distance between $q$ and any point in $P(v)$ is at most $\gamma r_B/512n^2d^2$, which is much smaller than $\gamma r_B/256nd$. This implies that it is possible to use a ball centered at $q$ to separate $P(v)$ and $P \setminus P(v)$. In fact, if let $r_{sep} := [s(v) + r_N]_\lambda$, where $[x]_\lambda$ denote the smallest real number that can be written as $(1+\lambda)^t$ for some integer $t$ such that $(1+\lambda)^t \geq x$, then it can be shown that for every $p \in P(v)$, $p \in \mathcal{B}(q, r_{sep})$ and for every $p \in P \setminus P(v)$, $p \notin \mathcal{B}(q, (1+\gamma)r_{sep})$.

With the above argument, we divide Case 4 into 2 sub-cases. Let $p_F$ be the actual farthest point to $q$ in $P \cap B$. We first discuss **Case 4a**, where $p_F \in P \setminus P(v)$. From the above discussion, we have $\|q - p_F\| \geq \gamma r_B/256nd$, which gives a good lower bound on the

farthest distance. For **Case 4b** where $p_F \in P(v)$, we know that the query reduces to a problem of finding the farthest point in $p \cap \mathcal{B}(q, r_{sep})$. For this problem, it is possible to prove a $r_{sep}/8nd$ lower bound and $r_{sep}$ upper bound on the farthest distance. The ratio of the two bounds is satisfactory.

To summarize, for Case 1,2 and 4a, let $d^{(1)}_{max} := (4+2\gamma)\epsilon^{-1}r_B$, $d^{(1)}_{min} := \gamma r_B/256nd$ and $r^{(1)}_B := r_B$, the AIFP query satisfies constraint $(r^{(1)}_B, d^{(1)}_{min}, d^{(1)}_{max})$; for Case 3, the AIFP query can be answered from constraint query $(\mathcal{B}(o_B, [(1+\gamma/16)r_B]_\lambda), q)$, which satisfies constraint $(r^{(2)}_B, d^{(2)}_{min}, d^{(2)}_{max})$, where $d^{(2)}_{min} := \gamma r_B/2048n^3d^3$, $d^{(2)}_{max} := (4+2\gamma)\epsilon^{-1}r_B$ and $r^{(2)}_B := [(1+\gamma/16)r_B]_\lambda$; for Case 4a, the AIFP query reduces to query $(\mathcal{B}(q, r_{sep}), q)$, with constraint $(r^{(3)}_B, d^{(3)}_{min}, d^{(3)}_{max})$ where $d^{(3)}_{max} := r^{(2)}_B := r_{sep}$, $d^{(3)}_{min} := d^{(3)}_{max}/8nd$. Any AIFP query can be answered from one of the three constraint queries, and we have $d^{(i)}_{max}/d^{(i)}_{min}$ no larger than $2048(4+2\gamma)n^3d^3/\epsilon$ for any $i = 1, 2, 3$.

## 4.1 Multi-scale Construction for General AIFP Query

In this subsection, we show how to build a Multi-Scale data structure to answer general AIFP queries. Our goal is to choose the appropriate parameters $\Gamma \geq 1$, $0 < \lambda < 1$ and $\Delta \geq 4nd$ for Algorithm 6 so that for every AIFP query with constraint $(r^{(i)}_B, d^{(i)}_{min}, d^{(i)}_{max})$, there exists a bucket $B^+_t$ whose range $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$ wholly covers the interval $[d^{(i)}_{min}, d^{(i)}_{max}]$, and the constrained AIFP data structure built for the bucket with $d_{min} := (1+\lambda)^t$ and $d_{max} := (1+\lambda)^{t+\Gamma+1}$ can be used to answer the query. Note that we have already defined $\lambda := \min(\epsilon, \gamma)/512$ and $\Delta := 4nd$. The remaining task is to determine the value of $\Gamma$.

Observe that $d^{(i)}_{max}/d^{(i)}_{min}$ is bounded by a polynomial $\mathcal{P}_{gap}(n, d) := 2048(4+2\gamma)n^3d^3/\epsilon$. Let $\Gamma' := \lceil \log_{1+\lambda} \mathcal{P}_{gap}(n, d) \rceil$, and $\Gamma_L \geq \Gamma'$ and $\Gamma_R \geq \Gamma'$ be integer parameters to be determined later. Denote by $\Gamma$ the sum of $\Gamma_L$ and $\Gamma_R$, *i.e.,* $\Gamma := \Gamma_L + \Gamma_R$. For every integer $t$, define $r_{mid}(t) := (1+\lambda)^{t+\Gamma_L}$. Therefore, we have $r_{mid}(t)/(1+\lambda)^t \geq (1+\lambda)^{\Gamma_L} \geq \mathcal{P}_{gap}(n, d)$ and $(1+\lambda)^{t+\Gamma+1}/r_{mid}(t) \geq (1+\lambda)^{\Gamma_R} \geq \mathcal{P}_{gap}(n, d)$. For any AIFP query $(B, q)$ with constraint $(r^{(i)}_B, d^{(i)}_{min}, d^{(i)}_{max})$, it is always possible to find a bucket $B^+_t$ such that $r^{(i)}_B = r_{mid}(t)$. Clearly, interval $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$ wholly covers the interval $[d^{(i)}_{min}, d^{(i)}_{max}]$. If a constrained AIFP data structure is constructed for $B^+_t$ with constraint $((1+\lambda)r_{mid}(t), (1+\lambda)^t, (1+\lambda)^{t+\Gamma+1})$, it can be used to answer the AIFP query $(B, q)$. (See Lemma 16.)

To summarize the above discussions, we set the parameters of Algorithm 6 as the following. The algorithm then produces the data structure for $(\epsilon, \gamma)$-AIFP query. Assume that a real number $0 < \delta < 1$ is given and we would like to achieve $1 - \delta$ query success probability.

- $\lambda := \min(\epsilon, \gamma)/512, \Gamma_L := \Gamma' + \lceil \log_{1+\lambda} 8 \rceil, \Gamma_R := \Gamma' + \lceil \log_{1+\lambda} 8 \rceil, \Gamma := \Gamma_L + \Gamma_R, \Delta := 4nd$.
- Routine $\mathcal{A}$: Given a non-empty bucket $B^+_t$, $\mathcal{A}$, it uses Algorithm 1 to creates a constrained $(\lambda/6, \lambda/6)$-AIFP data structure for point set $re(B^+_t)$ for constraint $((1+\lambda)r_{mid}(t), (1+\lambda)^t, (1+\lambda)^{t+\Gamma+1})$, with success probability at least $1 - \delta/4$.

Note that we let $\Gamma_L := \Gamma_R := \Gamma' + \lceil \log_{1+\lambda} 8 \rceil$. This allows more gap when fitting the interval $[d^{(i)}_{min}, d^{(i)}_{max}]$ in $((1+\lambda)^t, (1+\lambda)^{t+\Gamma+1}]$, which is required by Lemma 16.

With the multi-scale data structure constructed by Algorithm 6 using the above parameters, we are able to answer any general AIFP query by reducing it to at most three constrained $(\lambda/6, \lambda/6)$-AIFP queries (where $\lambda = \min(\epsilon, \gamma)/512$) with constraints $(r^{(i)}_B, d^{(i)}_{min}, d^{(i)}_{max})$, $i = 1, 2, 3$. From Lemma 6 and the fact that the ratio $d_{max}/d_{min}$ satisfies $d_{max}/d_{min} = (1+\lambda)^{t+\Gamma+1}/(1+\lambda)^t = (1+\lambda)^{\Gamma+1}$, which is bounded by a polynomial of $n, d$, we know that $\log d_{max}/d_{min}$ is $O_{\epsilon,\gamma}(\log nd)$ and the query time is $O_{\epsilon,\gamma}(dn^\rho \log \delta^{-1} \mathrm{Polylog}(nd))$

for some $\rho < 1$ depending on $\epsilon, \gamma$. The multi-scale data structure consists of multiple AIFP structures built on buckets of points with total size $O_\lambda(\Gamma^2 n \log(n\Delta))$ (from Lemma 15). From Lemma 6, we know that the complexity of the data structure is $O_{\epsilon,\gamma}(dn^{1+\rho} \log \delta^{-1} \text{Polylog}(nd))$. We leave the detailed analysis to the full version of the paper.

## 5 MEB Range Aggregate Query

Given any query ball $B$, we find the AMEB of $P \cap B$ using an iterative algorithm by Badoiu and Clarkson [9]. Their algorithm was originally designed for finding an approximate MEB for a fixed point set $P$. With careful analysis we show that their approach, after some modifications, can still be used to find AMEB in any given range $B$. Briefly speaking, our idea is to construct a small-size coreset of $P \cap B$. The MEB of the coreset is then a $(\epsilon, \gamma)$-approximate MEB of $P \cap B$. The algorithm selects the coreset in an iterative fashion. It starts with an arbitrary point $p$ from $P \cap B$. At each iteration, it performs the following operation to add a point to the coreset: (1) Compute an (approximate) MEB of the current coreset (directly using the algorithm in [9]); (2) Identify the AIFP in $P \cap B$ to the center of the current MEB, and add it to the coreset. We can show that after $O_{\epsilon,\gamma}(\log n)$ iterations, the MEB of the coreset is then a $(\epsilon, \gamma)$-AMEB of $P \cap B$.

To answer an $(\epsilon, \gamma)$-AMEB query with success probability at least $1 - \delta$, we will need only one $(\epsilon', \gamma')$-AIFP data structure with success probability $1 - \delta / O_{\epsilon'}(\log n)$ (where $\epsilon', \gamma'$ depends on $\epsilon, \gamma$ polynomially), whose size is $O_{\epsilon,\gamma}(dn^{1+\rho} \log \delta^{-1} \text{Polylog}(nd))$ for some $\rho < 1$ depending on $\epsilon, \gamma$. Every AMEB query is reduced to $O_{\epsilon'}(\log n)$ AIFP queries (the time for computing the MEB every iteration is negligible compared to AIFP queries). Thus, the query time is $O_{\epsilon,\gamma}(dn^{\rho} \log \delta^{-1} \text{Polylog}(nd))$. The detailed analysis is left to the full version of the paper.

## References


1. Sofiane Abbar, Sihem Amer-Yahia, Piotr Indyk, Sepideh Mahabadi, and Kasturi R Varadarajan. Diverse near neighbor problem. In *Proceedings of the twenty-ninth annual symposium on Computational geometry*, pages 207–214, 2013.
2. Mikkel Abrahamsen, Mark de Berg, Kevin Buchin, Mehran Mehr, and Ali D Mehrabi. Range-clustering queries. *arXiv preprint*, 2017. `arXiv:1705.06242`.
3. Pankaj K Agarwal, Lars Arge, Sathish Govindarajan, Jun Yang, and Ke Yi. Efficient external memory structures for range-aggregate queries. *Computational Geometry*, 46(3):358–370, 2013.
4. Pankaj K Agarwal, Jeff Erickson, et al. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.
5. Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 1018–1028. SIAM, 2014.
6. Sunil Arya, David M Mount, and Eunhui Park. Approximate geometric mst range queries. In *31st International Symposium on Computational Geometry (SoCG 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
7. Martin Aumüller, Tobias Christiani, Rasmus Pagh, and Francesco Silvestri. Distance-sensitive hashing. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 89–104, 2018.
8. Martin Aumüller, Sariel Har-Peled, Sepideh Mahabadi, Rasmus Pagh, and Francesco Silvestri. Sampling a near neighbor in high dimensions–who is the fairest of them all? *arXiv preprint*, 2021. `arXiv:2101.10905`.
9. Mihai Badoiu and Kenneth L Clarkson. Smaller core-sets for balls. In *SODA*, volume 3, pages 801–802, 2003.

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

11 Ryan R Curtin and Andrew B Gardner. Fast approximate furthest neighbors with data-dependent candidate selection. In *International Conference on Similarity Search and Applications*, pages 221–235. Springer, 2016.

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

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

14 Sariel Har-Peled. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011.

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

16 Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range queries in olap data cubes. *ACM SIGMOD Record*, 26(2):73–88, 1997.

17 Qiang Huang, Jianlin Feng, and Qiong Fang. Reverse query-aware locality-sensitive hashing for high-dimensional furthest neighbor search. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 167–170. IEEE, 2017.

18 Ziyun Huang, Hu Ding, and Jinhui Xu. A faster algorithm for truth discovery via range cover. *Algorithmica*, 81(10):4118–4133, 2019.

19 Piotr Indyk. Better algorithms for high-dimensional proximity problems via asymmetric embeddings. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 539–545, 2003.

20 Sankalp Khare, Jatin Agarwal, Nadeem Moidu, and Kannan Srinathan. Improved bounds for smallest enclosing disk range queries. In *CCCG*. Citeseer, 2014.

21 Zhe Li, Tsz Nam Chan, Man Lung Yiu, and Christian S Jensen. Polyfit: Polynomial-based indexing approach for fast approximate range aggregate queries. *arXiv preprint*, 2020. `arXiv:2003.08031`.

22 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

23 Yakov Nekrich and Michiel HM Smid. Approximating range-aggregate queries using coresets. In *CCCG*, pages 253–256, 2010.

24 Rasmus Pagh, Francesco Silvestri, Johan Sivertsen, and Matthew Skala. Approximate furthest neighbor in high dimensions. In *International Conference on Similarity Search and Applications*, pages 3–14. Springer, 2015.

25 Saladi Rahul, Haritha Bellam, Prosenjit Gupta, and Krishnan Rajan. Range aggregate structures for colored geometric objects. In *CCCG*, pages 249–252, 2010.

26 Saladi Rahul, Ananda Swarup Das, KS Rajan, and Kannan Srinathan. Range-aggregate queries involving geometric aggregation operations. In *International Workshop on Algorithms and Computation*, pages 122–133. Springer, 2011.

27 Yufei Tao, Xiaokui Xiao, and Reynold Cheng. Range search on multidimensional uncertain data. *ACM Transactions on Database Systems (TODS)*, 32(3):15–es, 2007.

28 Jeffrey Scott Vitter and Min Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. *Acm Sigmod Record*, 28(2):193–204, 1999.

29 Eugene Wu and Samuel Madden. Scorpion: Explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment*, 6(8), 2013.

30 Jie Xue. Colored range closest-pair problem under general distance functions. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 373–390. SIAM, 2019.

**31** Jie Xue, Yuan Li, and Ravi Janardan. Approximate range closest-pair queries. *Computational Geometry*, 90:101654, 2020.

**32** Xiaochun Yun, Guangjun Wu, Guangyan Zhang, Keqin Li, and Shupeng Wang. Fastraq: A fast approach to range-aggregate queries in big data environments. *IEEE Transactions on Cloud Computing*, 3(2):206–218, 2014.