Computational Geometry in High-Dimensional Spaces

Paul Liu

Abstract

In this project, we provide a review of efficient solutions to a few high-dimensional CG problems, with particular focus on the problem of *nearest neighbour search.* For the problems we examine, the general approach will be to either reduce the dimension (using the JL transform) or to preprocess the data and group together points that are close together (using locality sensitive hashing). Finally, we apply these techniques to a relaxed version of the *unit disc cover problem*, an NP-Hard facility location problem that has only been effectively approximated in the low-dimensional case.

1 Introduction

Over the past 30 years, research in computational geometry (CG) have produced many fruitful and ingenious data structures for geometrical problems in the low-dimensional case. These data structures are typically efficient in 2D or 3D but suffer from hidden constant factors in the dimension d [10]. In many cases, these hidden factors are $d^{O(1)}$ or even $c^{O(d)}$ for some $c > 1$ (this is often referred to as the *curse of dimensionality*). Some of these problems, such as *nearest neighbour search*, are commonly encountered in real world applications involving high-dimensional data sets. These data sets are often impossible or extremely inefficient to handle with low-dimensional methods. As such, high-dimensional CG has received a fair amount of attention in the past two decades, and *approximate* practical solutions have been discovered for a variety of interesting problems.

In this project, we provide a review of efficient solutions to a few high-dimensional CG problems. The paper is composed of two parts: exposition and exploration. Section 2 is expository and explains some key ideas in high-dimensional CG. As an example, we will focus on the problem of *nearest neighbour search* in $\mathbb{R}^d$ under the l_1 and l_2 norms. In particular, we will survey the results of Indyk et al. [7, 8]. For the problems we examine, the general approach will be to either reduce the dimension (using the JL transform) or to preprocess the data and group together points that are close together (using locality sensitive hashing). Section 3 is exploratory and attempts to apply these techniques to the *unit disc cover problem*, an NP-Hard facility location problem that has only been effectively approximated in the low-dimensional case.

2 Nearest neighbour search

Given a set P of n points in $\mathbb{R}^d$, the nearest neighbour (NN) problem asks for the closest point in P to q under some distance function d. In particular, we wish to preprocess P as to quickly answer queries q which are not known in advance. For the class of l_p metrics, this can be done naively in $\langle 1, dn\rangle$[1] time by checking the distance of every point in P to q. In the plane, many provably optimal $\langle n\log n, \log n\rangle$ data structures have been discovered [9]. When the dimension is large[2] however, the similar "low-dimensional" ideas lead to running times that are no better than the naive algorithm, or even worse, have an exponential dependence on the dimension d. One such example is the kd-tree, which exhibits time complexity $\left\langle n\log n, dn^{1-1/d}\right\rangle$. The apparent difficulty of high dimensional NN has led to conjectures that no efficient solution exists when the dimension is sufficiently large [10].

However, in many cases of practical interest we only require an ϵ-approximate nearest neighbour (ϵ-ANN):

Definition 1 *Let* $\mathrm{d}(q, Q) := \min_{p\in Q}\mathrm{d}(q, p)$ *denote the distance of the closest point in* Q *to* q*. Given a set* P *of*

[1]We say that an algorithm with preprocessing time $O(p(n))$ and query time $O(q(n))$ has complexity $\langle p(n), q(n)\rangle$.

[2]By large, we mean that the dimension d is $n^{O(1)}$.

n points in $\mathbb{R}^d$ and a parameter $\epsilon > 0$, the ϵ-approximate nearest neighbour (ϵ-ANN) problem asks for any point $p \in P$ such that $\mathrm{d}(q, p) \leq (1 + \epsilon)\mathrm{d}(q, P)$.

That is, we wish to find *any* point with distance at most $(1+\epsilon)$ times the distance to the nearest neighbour of the query. Surprisingly, recent results have shown that efficient data structures for ANN exist. These data structures only have polynomial dependence on the dimension and sublinear dependency on n (usually on the order of $n^{\frac{1}{1+\epsilon}}$ for a given ϵ), but possibly exponential dependence on ϵ. We describe three such data structures for the Hamming, l_1, and l_2 norm in the Sections below. The data structures we describe are due to [8], and our presentation follows roughly that of [5].

2.1 The general approach

Instead of solving ϵ-ANN, we will first solve the following similar problem:

Definition 2 *Given a set P of n points in $\mathbb{R}^d$, and parameters $\epsilon, r > 0$, the ϵ-point location in equal balls ((ϵ, r)-PLEB) problem asks for:*

1. *Any point $p \in P$ such that $\mathrm{d}(q, p) \leq (1 + \epsilon)r$ if $\mathrm{d}(q, P) \leq r$.*
2. *Nothing if $\mathrm{d}(q, P) > (1 + \epsilon)r$.*
3. *Otherwise either a point $p \in P$ such that $\mathrm{d}(q, p) \leq (1 + \epsilon)r$ or nothing.*

Note that any solution to the ϵ-ANN solves the (ϵ, r)-PLEB problem by checking if the point p returned by ϵ-ANN satisfies $\mathrm{d}(q, p) \leq (1 + \epsilon)r$. More importantly, the reduction also goes the other way when certain constraints are placed upon the query point, and we can solve ϵ-ANN by constructing and binary searching on multiple instances of (ϵ, r)-PLEB:

Theorem 1 *Given a set P of n points in $\mathbb{R}^d$, and a query point q such that $\mathrm{d}(q, P) \in [a, b]$, we can solve ϵ-ANN using $O\left(\epsilon^{-1} \log(b/a)\right)$ instances of (ϵ, r)-PLEB.*

Proof. First, we create $O\left(\epsilon^{-1} \log(b/a)\right)$ instances of (ϵ', r_i)-PLEB, by choosing $r_i = (1 + \epsilon)^i a$ for $i = 0, 1, \ldots, m$ where $m := \lceil \log_{1+\epsilon}(b/a) \rceil$, and choosing ϵ' so that $(1 + \epsilon')^2 = 1 + \epsilon$.

Since $d(q, P) \in [a, b]$, there exists some i for which $r_{i-1} \leq \mathrm{d}(q, P) < r_i$. In this case, the first i instances of (ϵ, r)-PLEB will return nothing while the $i + 1$-st instance of (ϵ, r)-PLEB (with radius r_i) must return a point p such that

$$\begin{aligned} \mathrm{d}(q, p) &\leq (1 + \epsilon')r_i \\ &\leq (1 + \epsilon')^2 \mathrm{d}(q, P) \\ &\leq (1 + \epsilon)\mathrm{d}(q, P). \end{aligned}$$

Hence at least one instance of (ϵ', r_i)-PLEB will return an answer to ϵ-ANN. Furthermore, we can binary search over the m instances of (ϵ', r_i)-PLEB in increasing order of r_i to answer ϵ-ANN in only $O(\log m)$ (ϵ', r_i)-PLEB queries. □

It's not difficult to further prove that we can solve any ϵ-ANN problem with $\mathrm{d}(q, P) \in [a, \infty)$ by taking $m = \lceil \log_{1+\epsilon}(\Delta(P)/\epsilon) \rceil$ in the proof above (where $\Delta(P)$ is the maximum distance between any two points of P). However, for query points q very close to the point set P, the binary search above is not applicable, since fixing a enforces a minimum separation between P and the query point q. This makes the reduction above unsuitable for direct application to continuous metrics such as l_1 and l_2.

One may wonder if there is a more efficient approach that is applicable to continuous metrics, rather than the binary search solution we developed above. Furthermore, we might need to use $O\left(\log_{1+\epsilon}(\Delta(P)/\epsilon)\right)$ different PLEB structures with binary search. When $\Delta(P)$ is unbounded for the possible given inputs, this is unsatisfying. Indeed, Indyk et. al [8] make use of a *ring-cover tree* to achieve an data structure that removes the dependence on $\Delta(P)$ while only adding poly-logarithmic factors onto the query time and space.

Regardless, the general approach to solving ϵ-ANN will then be to develop solutions to (ϵ, r)-PLEB, and then binary search on several instances of (ϵ, r)-PLEB for the solution. We describe the approach for the Hamming norm next.

2.2 (ϵ, r)-PLEB in the Hamming norm

In this section, let our point set P be drawn from the d-dimensional hypercube $\{0, 1\}^d$. That is, each point in

P is a d-bit binary string. Our distance function $\mathrm{d}(p, q)$ will be the Hamming norm, i.e. the number of bits in which two points p and q differ.

The binary search reduction developed in the previous section is perfect for the Hamming norm, two points in this space cannot be arbitrarily close without being the same. We can use Theorem 1 with $a = 1$ and $b = d$ to get a data structure for ϵ-ANN that only requires constructing $O(\epsilon^{-1} \log d)$ instances of (ϵ, r)-PLEB and $O(\log \log d + \log \epsilon^{-1})$ (ϵ, r)-PLEB queries to obtain an ϵ-approximate nearest neighbour. The case of $\mathrm{d}(q, P) = 0$ for a query point q can be answered in $O(1)$ time by storing points of P in a hash table and checking if q is present in the table.

To solve (ϵ, r)-PLEB in the Hamming norm, we use a technique called locality sensitive hashing (LSH). The approach will be create a hash function H for which $H(p) = H(q)$ if p and q are "close". For each point $p \in P$, we will then store p in a hash table using $H(p)$ as the index (we'll actually use multiple hash functions and hash tables). To query for a point q, we simply scan through all points in the table located at index $H(q)$.

Definition 3 *Given parameters $0 < r < R$, a family $\mathcal{H}$ of functions (defined on $\{0, 1\}^d$) is (r, R, α, β)-sensitive if for any $p, q \in \{0, 1\}^d$ and any randomly chosen function $h \in \mathcal{H}$,*

1. *$Pr[h(p) = h(q)] \geq \alpha$ if $\mathrm{d}(p, q) \leq r$.*
2. *$Pr[h(p) = h(q)] \leq \beta$ if $\mathrm{d}(p, q) \geq R$.*

The variables α and β are called the lower and upper sensitivities respectively.

We want the hash function to group close points together and separate points that are far apart. In other words, must satisfy $\alpha > \beta$.

For the Hamming norm, there happens to be a very simple family of locality sensitive hash functions:

Lemma 2 *Let $\mathcal{H} = \{h_1, h_2, \ldots, h_d\}$, where $h_i(p)$ returns the i-th bit of point p. Then $\mathcal{H}$ is $(r, R, 1-r/d, 1-R/d)$-sensitive for any $r, R > 0$.*

Proof. Suppose $\mathrm{d}(p, q) \leq r$, then q differs from p in at most r bits and so the probability that $h_i(p) = h_i(q)$ for a randomly chosen i is at least $1 - r/d$. Similarly, if $\mathrm{d}(p, q) \geq R$, then p and q have at most $d - R$ bits in common, which implies $h_i(p) = h_i(q)$ has probability of at most $1 - R/d$. □

In applying Lemma 2 to our applications, we choose $R = r(1 + \epsilon)$. When ϵ is large, we have a large ratio between the upper sensitivity $1 - r/d$ and lower sensitivity $1 - R/d$. Unfortunately, when ϵ is small, the ratio shrinks. Ideally, we'd like a way to increase the ratio between the two sensitivities when ϵ is small.

Lemma 3 *Given a family $\mathcal{H}$ of (r, R, α, β)-sensitive hash functions, we can get another family $\tilde{\mathcal{H}}$ of $(r, R, \alpha^k, \beta^k)$-sensitive hash functions for any given integer $k > 0$.*

Proof. Consider the hash function $\tilde{h}_\sigma(p) = (h_{\sigma_1}(p), h_{\sigma_2}(p), \ldots, h_{\sigma_k}(p))$ where σ is a randomly chosen sequence of k integers from $\{1 \ldots d\}$. Since the h_{σ_i} 's are independent, for two points p and q we have

1. $\Pr[\tilde{h}_\sigma(p) = \tilde{h}_\sigma(q)] \geq \alpha^k$ if $\mathrm{d}(p, q) \leq r$.
2. $\Pr[\tilde{h}_\sigma(p) = \tilde{h}_\sigma(q)] \leq \beta^k$ if $\mathrm{d}(p, q) \geq R$.

□

We now have a family $\mathcal{H}_1$ of hash functions with a large ratio between the lower and upper sensitivity. Furthermore, for a function $\tilde{h} \in \tilde{\mathcal{H}}$ the probability that $\tilde{h}(p) = \tilde{h}(q)$ for the case $\mathrm{d}(p, q) \geq R$ can be made arbitrarily small. So we can now separate points far away from each other. However, this has come at a cost: we also want $\tilde{h}(p) = \tilde{h}(q)$ to occur with high probability when $\mathrm{d}(p, q) \leq r$, but α^k could be disastrously small. To fix this, we can construct L hash tables simultaneously, and insert each point $p \in P$ into each table. When inserting into table i, we store the point p at index $\tilde{h}_i(p)$. If the index $\tilde{h}_i(p)$ already contains another point $p' \in P$, then do not insert p at all. This way, each of the L hash tables contain at most n points, and each cell of the hash table contains at most 1 point. By storing pointers to p instead of the actual point, we can ensure that the storage cost is $O(nL)$ and the setup cost is $O(nkL)$.

To query a point q, we look through all the tables. On table i we check if there is a point stored at $\tilde{h}_i(q)$. If there is, we compute the $\mathrm{d}(q, p)$ and return p if $\mathrm{d}(q, p) \leq$

$r(1+\epsilon)$. If none of the tables yielded such a point, then we return nothing. By choosing the right value of L, the query can solve ϵ-ANN with decent probability:

Lemma 4 *The query scheme above solves ϵ-ANN with probability* $\min\left\{(1-\beta^k)^L, 1-(1-\alpha^k)^L\right\}$. *Furthermore, the query time complexity is $O(kL+dN)$, where N is the number of points we encounter over all the tables.*

1. *If $\mathrm{d}(q,P) \le r$, we return a point with probability at least $1-(1-\alpha^k)^L$.*
2. *If $\mathrm{d}(q,P) \ge r(1+\epsilon)$, we return nothing with probability at least $(1-\beta^k)^L$.*

Proof. Let $p_i(q)$ denote the probability that a point is returned on table i for a query point q. If $\mathrm{d}(q,P) \le r$, then for a randomly chosen $\tilde{h} \in \mathcal{H}$ we have

$$\begin{aligned} \Pr[\text{a point is returned}] &= 1 - \prod_{i=1}^{L} p_i(q) \\ &\ge 1-(1-\alpha^k)^L. \end{aligned}$$

If $\mathrm{d}(q,P) \ge r(1+\epsilon)$, then similarly

$$\begin{aligned} \Pr[\text{returns nothing}] &= \prod_{i=1}^{L}(1-p_i(q)) \\ &\ge (1-\beta^k)^L. \end{aligned}$$

The bound on the time complexity comes from the fact that we must scan L tables, computing the hash function at q in each table in $O(k)$ time, and possibly doing one distance computation per table in $O(d)$ time with N computations in total. □

To see the power of Lemma 4, suppose that we choose k high enough so that $\alpha^k = \beta^k/10$. Furthermore, we choose $L = 1/\alpha^k$. Then the two probabilities above work out to be roughly $1-e$ and $e^{1/10}$. Hence by choosing k and L correctly, we may make our probability of success as large as we wish. It remains to optimize our choice of k and L so that we retain fast query time with constant probability of success. Then we may build multiple instances of the $\langle nkL, kL+dN\rangle$ data structure above to boost the probability of success arbitrarily high.

As it turns out (see calculations of [5] for more details), the choice of $k = \log_{1/\beta} n$ and $L = \lceil 4/\alpha^k \rceil$ is optimal for obtaining a probability of success of at least 3/4. Using the family of $(r, R, 1-r/d, 1-r(1+\epsilon)/d)$-sensitive hash functions from Lemma 5, we can further show that the this data structure has complexity $\left\langle n^{1+\frac{1}{1+\epsilon}} \log n, dn^{\frac{1}{1+\epsilon}} \right\rangle$ with space $O\left(n^{1+\frac{1}{1+\epsilon}} \log n\right)$. To boost the probability, we need to build $O(\log n)$ instances of this data structure.

Theorem 5 *There exists a data structure for (ϵ, r)-PLEB that succeeds with high probability. Furthermore, this data structure has complexity $\left\langle n^{1+\frac{1}{1+\epsilon}} \log^2 n, dn^{\frac{1}{1+\epsilon}} \log n \right\rangle$ with space usage $O\left(nd + n^{1+\frac{1}{1+\epsilon}} \log n\right)$.*

Proof. By the paragraph previous. □

When ϵ is small, the data structure we've just developed does hardly any better than brute force search. However, when ϵ is large (say $\epsilon > 1$), the query complexity is quite reasonable, and depends only linearly on the dimension.

2.3 (ϵ, r)-PLEB in the l_1 norm

To solve (ϵ, r)-PLEB in the l_1 norm, we reduce it to the Hamming norm case.

Let's simplify the problem, and suppose the points of P were at integer locations, with the maximum distance (under the l_1 norm) between any two points bounded an integer M. Then we may express the coordinate of any point as a d-tuple of $O(\log M)$ bits in each coordinate, or more simply $O(d \log M)$ bits. For instance suppose $M = 3$, the point $(1,2,3)$ on the grid would be translated to the tuple $(01, 10, 11)$ or more simply 011011. It's easy to confirm that after transforming each point to their bit-representation, the Hamming norm on the set of transformed points yields exactly the same distances as the l_1 norm of the original point set. Hence the results of the previous section implies an (ϵ, r)-PLEB data structure for the l_1 norm, when the points are at integer location.

To get around the restriction of integer locations, we round the given points to their closest points on a uniform grid of side length δ. Then we interpret each grid point as an integer point with a simple scaling. By choosing $\delta = O\left(\frac{\epsilon r}{d}\right)$, the distances between pairs of the rounded points and pairs of the original points differ by at most $O(\epsilon r)$, and so we can still get an instance of (ϵ, r)-PLEB data structure for the l_1 norm, with extra factors of $\log \delta$ in the query and space complexity. One may wonder if there is a way to get around the rounding to grid points and avoid incurring the extra factor of $\log \delta$. Indeed, there exists LSH families that can be used directly in the l_1 norm ([2]), so that a similar analysis of the type we did in the previous Section could be used.

2.4 (ϵ, r)-PLEB in the l_2 norm

To solve (ϵ, r)-PLEB in the l_1 norm, we reduce it to the l_1 case.

The crux of the reduction relies on the following lemma:

Lemma 6 *(Johnson-Lindenstrauss) Let P be an arbitrary point set of n points in $\mathbb{R}^d$. There exists a $t \times d$ linear map T, such that $t = O(\epsilon^{-2}\log(n))$ and with high probability,*

$$(1-\epsilon)\|p-q\|_2 \leq \|Tp - Tq\|_2 \leq (1+\epsilon)\|p-q\|_2$$

for all $p, q \in P$, where $\|\cdot\|_p$ is the l_p norm.

In fact, the linear map T is simply a matrix where each entry is drawn randomly from $\{-1/\sqrt{t}, 1/\sqrt{t}\}$. The proof of this lemma is out of the scope of this project, but the reader can refer to [5] for more details. Furthermore, this matrix can be applied in $O(d\log d + t^3)$ as opposed to $O(td)$, our preprocessing and query complexity is only increased by an addtive factor no more than the query cost of our (ϵ, r)-PLEB structure in the previous section.

We start by mapping the points of P onto $\mathbb{R}^t$ using the linear map T. Once the input data is reduced to dimension $O(\epsilon^{-2}\log(n))$, we can perform another mapping f to map each point to another dimension of size $\mathbb{R}^{t'}$, where $t' = O(k/\epsilon^2)$. This mapping embeds the dimension reduced points into $\mathbb{R}^{t'}$ with distortion $(1 \pm \epsilon)$ under the l_1 norm. That is,

$$(1-\epsilon)\|p-q\|_1 \leq \|f(p) - f(q)\|_2 \leq (1+\epsilon)\|p-q\|_1$$

for any p and q in the dimension reduced point set P. This mapping is described in detail in [5]. Finally, we apply the l_1 norm solution described in the previous Section.

Of particular interest is whether we could improve the result by either improving the Johnson-Lindenstrauss (JL) Lemma or avoiding it altogether through using LSH in l_2. Indeed, we can avoid the JL lemma by using the LSH families developed in [2, 8]. Unfortunately, there is no simple way to improve the JL lemma as the result is essentially optimal [1]. We provide a simple packing argument below:

Theorem 7 *Let $0 < \epsilon < 1$ and suppose that we have a function $f : P \to \mathbb{R}^t$ for which*

$$(1-\epsilon)\|p-q\|_2 \leq \|f(p) - f(q)\|_2 \leq (1+\epsilon)\|p-q\|_2$$

for all $p, q \in P$. Then $t = \Omega(\log n)$.

Proof. Consider the set P in $\mathbb{R}^n$ of n-vectors $u_i := e_i/\sqrt{2}$ for $i = 1, 2 \ldots, n$, where e_i is the ith standard unit vector. Since $||u_i - u_j||_2 = 1$ for $i \neq j$, we have $||f(u_i) - f(u_j)||_2 \geq 1 - \epsilon$. Hence we may replace each point $f(u_i)$ with a ball of radius $(1-\epsilon)/2$, such that any two balls $B(u_i, (1-\epsilon)/2)$ and $B(u_j, (1-\epsilon)/2)$ are disjoint. On the other hand, $||f(u_1) - f(u_j)||_2 \leq 1 + \epsilon$, so all the balls are contained inside the ball $B(u_1, \frac{3+\epsilon}{2})$. This implies that $n \leq \left(\frac{3/2+\epsilon/2}{(1-\epsilon)/2}\right)^t$, which implies that $t = \Omega(\log n)$ for fixed ϵ. □

The problem of approximate nearest neighbours is well-studied and has received much attention in the last decade. There are however, many more aspects of nearest neighbours that we have not discussed. We refer the reader to the thesis of Indyk [7] for more information.

3 The unit disc cover problem

Definition 4 *Given a set P of n points in $\mathbb{R}^d$, the r-disc cover (r-DC) problem seeks to find a minimum number of unit radius balls[3] that covers all of P. Since in most cases we can scale r to 1, the special case of $r = 1$ is called the unit disc cover (UDC) problem.*

[3]We will also use the word discs interchangeably with balls.

This problem arises in many applications to facility location, motion planning, and image processing. For example, one such scenario would be to interpret the n points as houses, and the unit radius balls as cellphone towers that broadcast to a fixed radius.

In both the l_1 and l_2 norm, UDC is NP-hard [3]. Regardless, there exist polynomial time approximation schemes (PTAS) in arbitrary dimensions. In $\mathbb{R}^d$ dimensions, it is possible to approximate UDC in l_∞ to within $\left(1+\frac{1}{\ell}\right)^{d-1}$ [6, 4] for any integer $\ell > 0$. Since these algorithms rely on optimally solving the problem in an $\ell \times \ell$ hypercube through exhaustive enumeration, they tend to have a slow time complexity that scales exponentially with ℓ and d, making them impractical for large dimensional data sets. Currently, all state of the art approximation algorithms for UDC use some sort of inherently low dimensional packing argument to prove an adequate approximation factor. In the case of the aforementioned PTAS, one needs to find a covering of an $\ell \times \ell$ hypercube by unit spheres in $\mathbb{R}^d$. Since the covering size scales exponentially as d increases, both the running time and approximation factor of the PTAS scales exponentially as well. In low dimensional settings, this is not an issue. However, since n unit balls trivially cover P, the approximation factors of all known UDC algorithms are impractical or ineffective when $d = \Omega(\log n)$.

We will focus on the l_2 case of UDC. Instead of tackling the UDC problem head on, we look at a relaxed version:

Definition 5 *Given a set P of n points in $\mathbb{R}^d$ and a parameter $0 < \epsilon < 1$, the ϵ-relaxed r-disc cover ((ϵ, r)-RDC) problem seeks to find a minimum number of r radius balls that covers all of P, with the condition that there exists an optimal solution for which every radius r ball can be shrunk to radius $\frac{r}{1+\epsilon}$ while still covering all points of P.*[4] *In the special case that $r = 1$, we call this the ϵ-relaxed unit disc cover (ϵ-RUDC) problem*

That is, all of the balls in the optimal solution have some wiggle-room, and can be replaced with slightly smaller balls.

To our knowledge, this paper is the first time UDC is considered in a high dimensional setting, and the first time an algorithm for ϵ-RUDC is given. We offer an efficient approximation algorithm with running time $O(nd \log d + k^3 + n^2/\epsilon^{O(k)})$ and approximation factor $O\left(\epsilon^{-O(1)} \log^{5/2} n\right)$, where $k = O\left(\epsilon^{-O(1)} \log n\right)$.

3.1 The general approach

Our general approach will be to find an approximation for the maximum number of points a single unit ball can cover (we call these maximum coverage balls of radius 1). Then we greedily cover P by finding unit radius maximum coverage (MC) balls, removing from P the points they cover, and repeating this until there are no more points in P. We prove that this provides a simple $O(\log n)$ approximation algorithm.

To improve the running time of our algorithm, we will use the JL transform to reduce the dimension, and some basic hashing, at the cost of increasing the approximation factor.

3.2 An algorithm for ϵ-RUDC in low dimensions

First, let us develop an approximation algorithm for determining an MC ball of P. Instead of approximating the number of points in the ball, we approximate the radius of the ball:

Lemma 8 *In $O\left(n\epsilon^{-d}\right)$ time and space, we can determine a ball of radius $(1+\epsilon)r$ that covers at least as many points as a radius r MC ball of P.*

Proof. We do this by simple hashing. Partition the space using a uniform grid of side length $\epsilon r/\sqrt{d}$, and keep a counter initialized to 0 located at each cell of the grid (we only store grid cells with counts at least 1). For each point p in P, we increment the counter of every grid cell overlapped by the ball $B(p, r)$. Then we scan through all the counters, and return an arbitrary point in the grid cell with the maximum count. Clearly the count in this grid cell is at least as large as the number of points covered by the radius r MC ball. Furthermore, any ball in this grid cell of radius $(1 + \epsilon)r$ will cover all points that incremented this grid cell's counter, since the diameter of the grid cell is ϵr.

Since the number of cells overlapped by any ball

[4]The odd radius of $r(1+\epsilon)^{-1}$ is used for the convenience of the sections below. Using some simple inequalities, a simpler radius of the form $1 - \Theta(\epsilon)$ can be used in the definition instead.

$B(p, r)$ is at most $O(\epsilon^{-d})$, we have the desired result. □

Algorithm 1 A procedure for computing ϵ-RUDC

```
Input: A point set $P$ in $\mathbb{R}^d$.
Output: A set $\mathcal{S}$ of the unit discs that cover $P$.
1: $\mathcal{S} \leftarrow \emptyset$
2: while $P$ is not empty do
3:     $C \leftarrow$ an approximate MC ball of radius 1
4:     $\mathcal{S} \leftarrow \mathcal{S} + \{C\}$
5:     $P \leftarrow P - C$
6: end while
7: return $\mathcal{S}$
```

For convenience, let OPT denote the set of optimal balls in an instance of ϵ-RUDC. Then we have the following result:

Theorem 9 *In $O\left(n^2\epsilon^{-d}\right)$ time and $O\left(n\epsilon^{-d}\right)$ space, Algorithm 1 gives an $O(\log n)$ approximation to ϵ-RUDC.*

Proof. Since each C on line 3 of Algorithm 1 covers at least as many points as a radius $1/(1+\epsilon)$ MC ball of P, this means that C covers at least as many points as any ball from OPT. By the pigeonhole principle, this means C covers at least $n/|OPT|$ points.

We can now prove by induction that Algorithm 1 that on the m-th step of the algorithm, there are at most $(1 - 1/|OPT|)^m$ uncovered elements.[5] Once we reach a step m with $(1 - 1/|OPT|)^m < 1/n$, we are done. Suppose instead we run the algorithm until $\exp(-m/|OPT|) < 1/n$. This will take just as many steps, since $(1 - 1/|OPT|)^m \leq \exp(-m/|OPT|)$. Furthermore, we have

$$\begin{aligned} \exp(-m/|OPT|) &< 1/n \\ m/|OPT| &< \log n \\ m &> |OPT| \log n. \end{aligned}$$

Hence after the $\lceil |OPT| \log n \rceil$-th step of the algorithm, we will have covered all of P. Since we run the algorithm of Lemma 13 at most n times, the running time of the algorithm is $O\left(n^2\epsilon^{-d}\right)$. □

[5]This result is well known in folklore.

3.3 An algorithm for ϵ-RUDC in high dimensions

Now we have a simple approximation in any dimension d. Unfortunately when d is large, the algorithm of Lemma 8 is extremely inefficient. To solve this we simply apply the JL transform to the input point set P with distortion δ, and solve the $\left(\frac{1+\delta}{1+\epsilon}\right)$-DC problem on a smaller dimension of size $k = O(\delta^{-2} \log n)$.

We will first have to handle some issues of distortion. For each ball C of OPT, there will be another ball in the reduced space that covers the same points C did in the original space (since the JL transform will preserve the distance from the center of the ball to all points of P inside the ball). However, since the JL transform may increase distances by a factor of $1 + \delta$, this ball may be radius $\frac{1+\delta}{1+\epsilon}$ in the reduced dimension. Hence any solution to $\left(\frac{1+\delta}{1+\epsilon}\right)$-DC in the reduced dimension will use at most $|OPT|$ discs. Furthermore, any ball of radius r in the reduced dimension might have radius $r/(1-\delta)$ in the original dimension (since the JL transform may decrease distances by a factor of $1 - \delta$). Hence if we want to solve $\left(\frac{1+\delta}{1+\epsilon}\right)$-DC in the reduced dimension, we are limited to balls of radius at most $1 - \delta$.

To proceed further, we use the following result of Verger-Gaugry [11]:

Fact 1 *Any l_2 ball of radius R in $\mathbb{R}^k$ can be covered by $O\left(k^{3/2}(R/r)^k\right)$ balls of radius $r < R$.*

Now consider using (ϵ, r)-RDC to approximate the solution to R-DC when $r < R$. Since the discs in (ϵ, r)-RDC are smaller, we know that (ϵ, r)-RDC will need to use at least as many discs as R-DC. By a packing argument, we can quantify exactly how many extra discs:

Lemma 10 *Fix a point set P in $\mathbb{R}^k$, and let OPT be an optimal solution to the R-DC problem. Suppose an optimal solution to (ϵ, r)-RDC for $r < R$ uses at most m discs. Then we know that*

$$m \leq O\left(k^{3/2}(R/r)^k |OPT|\right)$$

Proof. By Fact 15, we can replace each unit disc in OPT with $O\left(k^{3/2}(R/r)^k\right)$ discs. This is a disc cover with radius r discs that uses at most $O\left(k^{3/2}(R/r)^k |OPT|\right)$ discs, hence any optimal solution to (ϵ, r)-RDC is upper bounded by this number. □

By Lemma 10, we can approximate $\left(\frac{1+\delta}{1+\epsilon}\right)$-DC with $\left(\frac{1-\delta}{1+\delta},\delta\right)$-RDC. We can also approximate $\left(\frac{1-\delta}{1+\delta},\delta\right)$-RDC by Algorithm 1. Hence, we have the following:

Theorem 11 *Given a point set P in $\mathbb{R}^d$ and a parameter ϵ, there exists an algorithm for ϵ-RUDC with running time $O(nd\log d+k^3+n^2/\epsilon^{O(k)})$ and approximation factor $O\left(\epsilon^{-O(1)}\log^{5/2} n\right)$, where $k=O\left(\epsilon^{-O(1)}\log n\right)$.*

Proof. By approximating $\left(\frac{1+\delta}{1+\epsilon}\right)$-DC with $\left(\frac{1-\delta}{1+\delta},\delta\right)$-RDC and approximating $\left(\frac{1-\delta}{1+\delta},\delta\right)$-RDC with Algorithm 1, we have an algorithm with approximation factor $O\left(k^{3/2}\left(\frac{1+\delta}{1+\epsilon}\cdot\frac{1+\delta}{1-\delta}\right)^k\log n\right)$. To minimize the approximation factor, we choose δ such that $\frac{(1+\delta)^2}{1-\delta}=1+\epsilon$. Since $\epsilon>0$, such a solution always exists with $0<\delta<1$. Furthermore, $\delta=O\left(\epsilon^{O(1)}\right)$ for fixed ϵ. Hence the approximation factor under this choice of δ is $O\left(k^{3/2}\log n\right)=O\left(\epsilon^{-O(1)}\log^{5/2} n\right)$. The running time comes from applying the JL transform to all points and then using Algorithm 4 with tolerance δ. □

References

[1] N. Alon. Problems and results in extremal combinatorics. I. *Discrete Math.*, 273(1-3):31–53, 2003. EuroComb'01 (Barcelona).

[2] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 253–262, New York, NY, USA, 2004. ACM.

[3] R. J. Fowler, M. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981.

[4] T. F. Gonzalez. Covering a set of points in multidimensional space. *Inf. Process. Lett.*, 40(4):181–188, 1991.

[5] S. Har-Peled. *Geometric approximation algorithms*, volume 173 of *Mathematical Surveys and Monographs.* American Mathematical Society, Providence, RI, 2011.

[6] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985.

[7] P. Indyk. *High-dimensional Computational Geometry.* PhD thesis, Stanford, CA, USA, 2001. AAI3000045.

[8] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98 (Dallas, TX)*, pages 604–613. ACM, New York, 1999.

[9] J. O'Rourke. *Computational geometry in C.* Cambridge University Press, Cambridge, second edition, 1998.

[10] J.-R. Sack and J. Urrutia, editors. *Handbook of computational geometry.* North-Holland, Amsterdam, 2000.

[11] J.-L. Verger-Gaugry. Covering a ball with smaller equal balls in $\mathbb{R}^n$. *Discrete Comput. Geom.*, 33(1):143–155, 2005.