Advanced Algorithms - Final Project

Reading on LSF method on spherical ANN

Yufei Guo - yg2892

Abstract

We study the problem of finding the approximate nearest neighbor when all points lie on a hypersphere in $\mathbb{R}^d$. We briefly explain hyperplane LSH method, cross-polytope method in [And+15], the LSF algorithm from [BDGL16], provide some thoughts on their differences and actual behavior. We further analyze the performance of LSF algorithm when dimension is restricted to $d = \Theta(\log n)$, which gives a better LSH exponent. We also show how to exploit the definition of locality sensitive filters to get asymmetric preprocessing time and query time as described in [Laa15].

Contents

1 Introduction

The problem of similarity search is when given a query item, return the most similar item in the data set. This problem is particularly important in solving real-life tasks in fields including machine learning. The similarity between items can be defined in many ways, via certain embeddings or kernel functions. When a data point characterized as a vector, finding the most similar item becomes finding the nearest neighbor in some given space. In general, the problem of finding nearest neighbor is described as: Given a set $\mathcal{D}$ of n vectors resides in a d dimensional space, upon query of vector $\boldsymbol{q}$, find $\boldsymbol{v} \in \mathcal{D}$ which is closest to q under specified similarity description. When similarity is defined properly, nearest neighbor search can be used to find most similar data points under edit distance, Wasserstein metric or Earth-Mover Distance. There are efficient solutions for finding the exact solution when dimension is low, by using binary search, kd-trees, Voronoi diagrams or even a simple linear search is very fast in $\mathbb{R}^1$ or $\mathbb{R}^2$. Yet these algorithms fail to achieve the same performance when the dimension is high. Solving the same problem can be costly due to "curse of dimensionality", when dimension is d, even a naive linear search through all points require $O(nd)$ time (given that doing one norm calculation $\|x-y\|, x, y \in D$ take $O(d)$ time and $n \gg d$). In order to obtain a solution in reasonable amount of time, we restrain ourselves to find an approximate answer.

The problem of approximate nearest neighbor (ANN) is, under the same setting, rather than finding an exact solution, is to find an approximate solution not far from the optimal solution if there is such optimal solution. Algorithms including sketching and dimension reduction provide valid solutions for this problem. In this survey we focus on Locality Sensitive Filters that was described in[BDGL16][Laa15], applied to points lie on the unit hypersphere in Euclidean space. On unit hypersphere, one can easily calculate angular distance between two vectors by their Euclidean distance, thus it is easy to generalize to cosine similarity. And in [AR15a] showed a reduction from search in entire Euclidean space to search on the unit sphere, providing an extension to finding nearest neighbor in the entire $\mathbb{R}^d$.

2 Preliminaries

Definition 1 (Definition of cr-ANN)**.** *Given D which is a collection of points, we wish to find an algorithm that, upon query q, if there exists $v \in D$ such that $\|v-q\| \leq r$, report some $v' \in D$ that $\|v'-q\| \leq cr$.*

We will describe a slightly different problem and work on this variant, as described in [BDGL16]

Definition 2 (Variant of cr-ANN)**.** *Upon query q, given that all points $v \in D$ are at least cr away except one p which is r close. Report this p.*

In the scope of this survey, we will assume the data set D is a *random* instance, that the data set is obtained by randomly sampling n points on $\mathcal{S}^{d-1}$, as described in [AR15a].

Definition 3 ((r, cr, p_1, p_2)-LSH)**.** *For fixed r, c, given a family of hash function $\mathcal{H}$, $h \in \mathcal{H}, h : \mathbb{R}^d \to U$. $\mathcal{H}$ is (r, cr, p_1, p_2)-LSH if $\forall p, q \in \mathbb{R}^d$*

$$\|p-q\| \leq r \Rightarrow \Pr[h(q) = h(q)] \geq p_1$$

$$\|p-q\| \geq cr \Rightarrow \Pr[h(q) = h(q)] \leq p_2$$

In both definitions, c is the approximation factor and $c > 1$. And r close points are mapped to same value with probability at least p_1, cr far points are mapped to different value with probability at most p_2. Unlike normal hashing techniques that aim to avoid collisions as much as possible, LSH aims to cause collisions between close points and avoid those of far points. Ideally for some LSH technique to work, we would want p_1 to be as large as possible, and p_2 to be as small as possible. Thus we define the LSH exponent $\rho = \frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}}$ to represent this ratio and represent the performance of such LSH technique.

Lemma 1 (Optimality of LSH exponent[AR15b]). *For some fixed $c > 0, r > 0$, one can solve cr-ANN in Euclidean space $\mathbb{R}^d$ with n data points for query time $\tilde{O}(dn^\rho)$ and $\tilde{O}\left(n^{1+\rho}\right)$ extra space (in addition to $O(nd)$ to store all the points). Where $\rho = \frac{1}{2c^2-1}$*

Lemma 2 (Johnson-Lindenstrauss Lemma [JL84]). *There exists a randomized linear map φ such that, $\forall p, q \in \mathbb{R}^k$,*

$$\Pr[\|\varphi(q) - \varphi(q)\| \in (1 \pm \epsilon)\|p - q\|] \geq 1 - e^{\frac{-\epsilon^2 d}{9}}$$

Corollary 1. *With high probability, n points in Euclidean space can be projected into a space of dimension $O\left(\epsilon^{-2} \log n\right)$, while causing at most a $1 + \epsilon$ distortion to pairwise distances.*

Then, without the loss of generality, given fixed number of points, we can safely assume all data are from $\mathbb{R}^d$ where $d = \Theta(\log n \log\log n)$, as done by [AR15a] one can do dimension reduction to d with distance between all pairs of points distorted by at most a factor of $1 + \frac{1}{(\log\log n)^{\Omega(1)}}$ by applying Johnson-Lindenstrauss lemma. We cannot further decrease the dimension to $d = \Theta(\log n)$ since doing so would introduce a constant factor distortion even before doing approximate nearest neighbor search.

One naive approach is to sample the $d \times n$ projection matrix from space of Gaussians, yet this naive approach takes $O(d \log n)$ time. However, there is an efficient algorithm[AC06] to perform Johnson-Lindenstrauss transform faster by using sparse matrices and fast Hadamard transform. Similar techniques were used in [And+15] in designing cross-polytope LSH.

We will use $\|\cdot\|$ to denote the l_2 norm applied to some vector, and $\theta_{p,q}$ to denote the angular distance between p, q, assuming p, q lies on unit hypersphere. Due to simplicity between the conversion of Euclidean distance and angular distance, we might switch between angular distance or Euclidean distance to simplify the proofs.

Obviously, since p, q are unit vectors, $\theta_{p,q} = 2\arcsin\left(\frac{\|p-q\|}{2}\right) = \arccos\left(1 - \frac{\|p-q\|^2}{2}\right)$

Definition 4 (Spherical Structures in $\mathbb{R}^d$ [Laa15][BDGL16]). *Using the same notation, we denote the hypersphere centered at origin in $\mathbb{R}^d$ to be $\mathcal{S}^{d-1} = \{\|x\| = 1, \forall x \in \mathbb{R}^d\}$. And the hyperspherical caps $\mathcal{C}_{\boldsymbol{u},\alpha}$ to be the intersection of $\mathcal{S}^{d-1}$ and half-space $\mathcal{H}_{\boldsymbol{u},\alpha} = \{\langle \boldsymbol{u}, \boldsymbol{x}\rangle \geq \alpha, \forall \boldsymbol{x} \in \mathbb{R}^d, \forall a \in (0,1)\}$, that $\mathcal{C}_{\boldsymbol{u},\alpha} = \mathcal{S}^{d-1} \cup \mathcal{H}_{\boldsymbol{u},\alpha}$. Hyperspherical wedge $\mathcal{W}_{\boldsymbol{u},\alpha,\boldsymbol{v},\beta}$ is the intersection of $\mathcal{C}_{\boldsymbol{u},\alpha}$ and $\mathcal{C}_{\boldsymbol{v},\beta}$.*

3 Overview of different ANN approaches

Many LSH approaches for ANN on $\mathcal{S}^{d-1}$ have been published over past decades. Including

- Random hyperplane LSH introduced in [Cha02] and briefly described in Section 4.
- Cross-polytope LSH that has been proved to be asymptotically optimal in [And+15] and briefly described in Section 5

- Other polytope LSH, using simplices, hypercube, or lattices. Some of which have been proved to be asymptotically optimal, but we will not discuss them here.

All of such techniques involving *partitioning* the space. However, if the function does not do exact partitioning, it might achieve asymptotically better bounds with smaller dimensions [Laa19][BDGL16]. We will describe what is this filter, the algorithm and asymptotic performance analysis in Section 6. Furthermore, we will try to give a higher level comparison between these three on the cause of this asymptotic improvement in Section 7.

4 Hyperplane LSH

One of the simplest method that one may thought of is to define a random hyperplanes (using supporting vector w) that go through the origin, the hyperplane maps each point $p \in D$ such that $h(p) = \text{sign}(w \cdot p)$. Notice that sampling a point from $\mathcal{S}^{d-1}$ is equivalent to sample from a d dimensional Gaussian distribution $N(0, I_d)$ (and normalize). Since Gaussian distribution is 2-stable and spherical symmetry, we can project the hypersphere onto the plane spanned by p, q, and then the hyperplane must lie between p, q (or on the opposite side) to separate p, q. It is obvious that the probability for p, q that have angular distance at most θ to be separated by a random hyperplane is at most $\frac{\theta}{\pi}$. And $\Pr[h(q) = h(p)] = 1 - \frac{\theta}{\pi}$. Note we can sample more hyperplanes and combine k such hash functions to amplify the probability.

The definition of hyperplane LSH suggests that hyperplane LSH can be sampled and decoded very efficiently, and very easy to understand. But it can only achieve $\rho \leq \frac{1}{c}$, which is much worse asymptotically compared to the optimal bound for large c. However, this probability is independent of d or n, and does not include any asymptotic analysis or any $o(1)$ constants that will disappear with high enough d and n. Even though this algorithm is theoretically sub-optimal, it is easy to implement and works well in practice (when d, n are small). From now on we will assume d, n is sufficiently large so that we can do asymptotic analysis.

5 Cross-polytope LSH

We first define what is a cross-polytope.

Definition 5 (Cross-polytope/orthoplex)**.** *A cross-polytope β_d is defined to be the convex polytope where its vertices are unit vectors along both directions of every axis (e.g., $\{\pm e_i\}$). Or, the unit ball in l_1 norm in $\mathbb{R}^d$*

Then, upon a query q, we apply a random rotation A to q, and define $h(q)$ to be the vertex in β_d that Aq is closest to.

It has been shown in [And+15] that this method is optimal as $\rho \to \frac{1}{2c^2 - 1}$ when d is sufficiently large. When A is sampled from a random Gaussian space, there is a large overhead in the calculation which is the random rotation A, as $d \times d$ (dense) matrix vector multiplication takes $O(d^2)$ time. Yet we can apply the technique similar to the fast JL transform as [AC06], and replace A with $HD_1HD_2HD_3$, where H is the fast Hadamard transform, D_i are independent random ± 1 diagonal matrices. As stated in [And+15], even though this approach does not has a rigorous proof that it is sufficiently *random*, it works well in practice when $d \to \infty$, which matches our assumption. Evaluating $HD_1HD_2HD_3$ takes only $O(d \log d)$ time, with space complexity $O(d)$ to store random diagonal matrices.

6 Spherical Cap LSF Algorithm

6.1 Overview

We first define what is a Locality sensitive filter. As opposed to LSH, which maps a data point to a value, locality sensitive filter maps a point to a result, indicating if this point "survives" the filtering or not. Formally, given a filter f, a list of points D, $f : D \to D_f, D_f \subseteq D$. The goal is similar to that of LSH, which requires carefully choosing some filter f, that close points survive the same filter with high probability, and far points survive with low probability. Notice that we also want the calculation of the map D_f to be cheap. We will then describe the approach done in [BDGL16], and how they choose such f to allow efficient calculations.

The key difference between LSH and LSF is, while given a LSH h, it *always* maps p to $h(p)$, it thus gives a partition of the space, where LSF does not. In the scope of spherical caps, if p does not fall in that cap, then it is undefined.

Given the random instance we are considering, and the slightly tweaked cr-ANN problem, we want to bound the probability a query q has only one close neighbor in some range. Then it is essentially calculate the probability that a randomly generated point on the hypersphere lies in the cap $\mathcal{C}_{\boldsymbol{q},\alpha}$, for some α

It is not hard to see this is essentially calculating the ratio between the volume of the hyperspherical cap and the volume of the hypersphere (Depending on the sampling method, this might also be the ratio between surface area instead of volume, but they essentially make no difference). And we can directly calculate such ratio.

Lemma 3 (Volume ratio between spherical structures)**.** $\frac{V(\mathcal{C}_{u,\alpha})}{V(\mathcal{S}^{d-1})} \propto d^{\Theta(1)}\left(\sqrt{1-\alpha^2}\right)^d = d^{\Theta(1)}(\sin\theta)^d$.

Where α is the parameter defines the height of the cap, and θ is the maximum angular distance between any two points on the cap. We show the brief proof of this lemma in **??**.

Similarly, one can easily get that the probability two random points p, q on $\mathcal{S}^{d-1}$ have Euclidean distance $\|p-q\|$ smaller than τ is proportional to $d^{\Theta(1)}\left(\frac{\tau}{\sqrt{2}}\right)^d$. Then the probability that no points except the one are near is $\left(1 - d^{\Theta(1)}\left(\frac{\tau}{\sqrt{2}}\right)^d\right)^{n-1}$.

When d is sufficiently large (and $d \ll n$), with high probability, no far points will be in $\tau = \sqrt{2} - o(1)$. Or, in the scope of angular distance, all far points will be at least $\frac{\pi}{2} - o(1)$ away with high probability, thus they are almost orthogonal to the query. Now τ corresponds to cr, thus we can pick $r = \frac{\tau}{c}$. Essentially this provides a connection between cr-ANN and its variant.

We will describe the algorithm from [BDGL16] on a higher level, and then dive into details on how each parameters. Note that we explicitly skipped combination of k LSFs since it is harder to analyze and $k = 1$ eventually.

6.2 Procedure of algorithm

Initialization

We draw t filters from the family of all filters $\mathcal{F}$. We will describe the parameters of each filter in Section 6.3. The number of filters and their construction t will be discussed in Section 6.5.

Preprocessing

$\forall p \in D$, calculate whether p survives f_i or not, and return the bucket for each filter which contains all the points survives that filter, such that $B_i \coloneqq \{f_i(p) = 1, \forall p \in D\}$. Note that the most

important thing is to assume the existence of an oracle $\mathcal{O}$. When given all filter f_1 through f_t, and a point p, instead of going through all t filters, we can query $\mathcal{O}$ to get the set of filters f_i that $f_i(p) = 1$ more efficiently. [BDGL16] suggests the existence of such oracle in section 5, which runs in $O(k)$, where k is the number of filters q survives. If we allow pseudo randomness over the LSF family, we can obtain such oracle. We will describe and analyze such oracle in Section 6.5.

At query time

Upon getting a query q, ask oracle to get the list of filters q survives, $F \coloneqq \{i \in [t], f_i(q) = 1\}$. Then, go to all those buckets and compare q with the points in the bucket, this is a two step query. Balancing the runtime of this two step query is discussed in Section 6.3.

6.3 Performance analysis

We pick the family of spherical LSFs to be the family of spherical caps. Since we assume a random data set, we can have every cap to cover almost a hemisphere, since all except one points are $\sqrt{2} - o(1)$ far. For now, we assume parameters are fixed during preprocessing and querying, we also use the same α. However, we can also use different threshold, namely α, β when preprocessing the data set and handling query. Intuition behind this trade is discussed in section 7.

Definition 6 (Spherical LSF)**.** *An LSF $f \in \mathcal{F}$ is construct by picking a random vector $v \in \mathcal{S}^{d-1}$, that filters a point p ($f(p) = 1$) when $\langle v, p\rangle \geq \alpha$ for some α.*

Notice LSF and LSH are different when outputting a result, even though (for simplicity) $f(p) = 1$ when p lies in that spherical cap f defines, $f(p)$ is not defined if p does not lie in that cap. And thus $f(p) = f(q) \iff$ both p, q survives f. Worth noting this does not imply the probability of events. We claim that the actual probability conditions on either p or q survives, otherwise this event is not defined [AR15a].

Naturally we want to calculate p_1 and p_2 in the definition of LSH exponent. For some f defined by v and α, $f(p) = f(q) \iff \langle v, p\rangle \geq \alpha, \langle v, q\rangle \geq \alpha$. This is equivalent to v falls in the intersection of $\mathcal{C}_{\boldsymbol{p},\alpha}$ and $\mathcal{C}_{\boldsymbol{q},\alpha}$. For simplicity, denote θ to be the angle between p and q.

Lemma 4 (Volume ratio between hyperspherical wedge and hypersphere[Laa15][BDGL16])**.** $\dfrac{V(\mathcal{W}_{\boldsymbol{u},\alpha,\boldsymbol{v},\beta})}{V(\mathcal{S}^{d-1})} \propto d^{\Theta(1)}\left(\sqrt{1-\gamma^2}\right)^d$ *where* $\gamma = \sqrt{\dfrac{\alpha^2+\beta^2-2\alpha\beta\cos\theta}{\sin\theta}}$*, Given the angular distance between u, v is θ.*

The detailed calculation and proof can be found in Appendix A in [BDGL16].

Note that since we are dealing with symmetric cost between preprocessing and handling a query, $\alpha = \beta$ and $\gamma^2 = \dfrac{2\alpha^2}{1+\cos\theta}$. Due to the fact that a filter does not guaranteed to map a point to a value and above analysis on defining the event, we have different representations for p_1 and p_2, which is the probability that both p, q survives f, over the probability that p survives f. The improvement of this entire algorithm is based on these three equations, we further discuss the understanding in section 7.

$$p_1 \geq d^{\Theta(1)}\left(\sqrt{1-\frac{2\alpha^2}{1+\cos\theta_1}}\right)^d \Big/ d^{\Theta(1)}\left(\sqrt{1-\alpha^2}\right)^d \tag{1}$$

$$p_2 \leq d^{\Theta(1)} \left(\sqrt{1-\frac{2\alpha^2}{1+\cos\theta_2}}\right)^d \Big/ d^{\Theta(1)} \left(\sqrt{1-\alpha^2}\right)^d \tag{2}$$

$$\rho = \frac{\frac{d}{2}\log\left(1-\alpha^2\right) - \frac{d}{2}\log\left(1-\frac{2\alpha^2}{1+\cos\theta_1}\right) - \log d}{\frac{d}{2}\log\left(1-\alpha^2\right) - \frac{d}{2}\log\left(1-\frac{2\alpha^2}{1+\cos\theta_2}\right) - \log d} = \frac{\log\left(1-\alpha^2\right) - \log\left(1-\frac{2\alpha^2}{1+\cos\theta_1}\right)}{\log\left(1-\alpha^2\right) - \log\left(1-\frac{2\alpha^2}{1+\cos\theta_2}\right)} + o\left(1\right)$$

We go back and analyze ρ in terms of α. With the definition of spherical caps, when $\alpha \to 0$, the algorithm is essentially "carving" caps that are almost hemispheres. And with $\alpha = \epsilon$, we can have $\rho \leq \frac{1}{2c^2-1} + o_c\left(1\right)$. The term "carve" is borrowed from [AR15a] and the calculation can be found there. Observe that the function is decreasing as α increases, thus when α is picked large enough, it should provide some non asymptotic improvement to the standard $\rho = \frac{1}{2c^2-1}$.

Balancing the number of filters and the items to compare

We define $p\left(\theta\right) \coloneqq \Pr$[a filter captures two points that are θ away]

Upon receiving a query q, we ask the oracle for the list of filters q survives, which takes $O\left(k\right)$ time using the oracle, where k is the size of this list, and $k = O\left(t \cdot p\left(0\right)\right)$. Then we go over each point in the buckets correspond to those filters. In each bucket, the number of points in that bucket is about $O\left(n \cdot p\left(\theta_2\right)\right)$, as all points are $\theta_2 = \frac{\pi}{2} - o\left(1\right)$ away. Thus the total query time is $O\left(t \cdot p\left(0\right) + t \cdot n \cdot p\left(\theta_2\right)\right)$. Ideally we want $p\left(0\right) = n \cdot p\left(\theta_2\right)$, so that we are not spending too much time on one step of the search. Then we have $d^{\Theta(1)}\left(1-\alpha^2\right)^{\frac{d}{2}} = nd^{\Theta(1)}\left(1-\frac{2\alpha^2}{1+\cos\theta_2}\right)^{\frac{d}{2}}$. Since $\lim_{d\to\infty} d^{O\left(\frac{2}{d}\right)} = 1$, we can solve for $\alpha = \sqrt{\frac{\left(n^{\frac{2}{d}}-1\right)\left(1+\cos\theta_2\right)}{2n^{\frac{2}{d}}-1-\cos\theta_2}} = \sqrt{\frac{n^{\frac{2}{d}}\left(\cos\theta_2-1\right)}{2n^{\frac{2}{d}}-1-\cos\theta_2}+1}$.

α is related to n and d, we have not yet made any assumption on the connection them. We will start discussion on dimensionality to show the performance of this algorithm in diifferent cases.

6.4 Dimension analysis

6.4.1 Sparse regime

In general, the LSH-based cr-ANN on unit hypersphere has a query time of $\tilde{O}\left(dn^{\rho}\right)$ where $\rho = \frac{1}{2c^2-1}$. We borrow the term "sparse regime" from [Laa15]. And we wish to analyze the performance of the algorithm where $d = \omega\left(\log n\right)$, note we have used the fact in section 6.3 that the exponent of this algorithm is upper bounded by $\frac{1}{2c^2-1} + o_c\left(1\right)$. By Corollary 1, we claim that we can always safely assume that dimension is $\Theta\left(\log n \log\log n\right)$, when n is fixed, by applying Johnson-Lindenstrauss Lemma. Thus any instance with higher dimensions can be reduced to one specific dimension, when n is fixed.

Instead of investigating $d = \Theta\left(\log n \log\log n\right)$, we will assume $d = \frac{1}{\kappa}\log n$, but with $\kappa \to 0$, as d will be very large, even though proportional to $\log n$. We wish to show $\rho \sim \frac{1}{2c^2-1}$.

Let $d = \frac{1}{\kappa}\log n$. We first look at α, as α determines ρ. We have

$$\alpha = \sqrt{\frac{\left(n^{\frac{2}{d}} - 1\right)(1 + o_d(1))}{2n^{\frac{2}{d}} - 1 - o_d(1)}} \geq \sqrt{\frac{n^{\frac{2}{d}} - 1}{2n^{\frac{2}{d}} - 1}} = \sqrt{\frac{e^{2\kappa} - 1}{2e^{2\kappa} - 1}}$$

Since ρ deceases as α increases, having this α gives a lower bound on the performance of the algorithm.

We wish to analyze the behavior of α and ρ when $\kappa \to 0$ by doing second order Taylor expansion on $\kappa \approx 0$, and get $\alpha = \left(2\kappa - \frac{\kappa^3}{2} + O\left(\kappa^5\right)\right)^{\frac{1}{2}}$. This suggests as the dimension goes higher, $\alpha \to 0$ and are 'carving' almost hemispheres.

We continue to analyze ρ, and by Taylor expansion, we have
$\rho \leq \frac{1}{2c^2 - 1} - \frac{2\left(c^2 - 1\right)\kappa}{\left(2c^2 - 1\right)^2} + O\left(\kappa^2\right) \propto \frac{1}{2c^2 - 1}$.
The assumption that $\kappa \to 0$ implies d is sufficiently large and we assume that all $o(1)$ terms vanish for simplicity of calculation.

This result matches the bound obtained by LSH, and suggests this algorithm does not provide asymptotic improvement when $d = \omega(\log n)$. Though the limit does approach $\frac{1}{2c^2 - 1}$ from below with $o_\kappa(1)$.

6.4.2 Dense regime

However, [BDGL16] showed that the lower bound of the LSH exponent ρ does not hold when dimension is small, using the above algorithm. We borrow the term "dense regime" from [Laa15] to refer to this case. Notice that this assumption on d lack generality, compared to the assumption that $d = \omega(\log n)$.

Since the probability for two points two be θ_1 close is $d^{\Theta(1)}(\sin\theta)^d$, total number of close points is about $n \cdot d^{\Theta(1)}(\sin\theta)^d$. One can always sample $\bar{n} = O\left((\sin\theta)^{-d}\right)$ points from D, such that there is only constant number of close point in expectation. Thus we can always solve some (very) dense instance by random sampling and solve on each sparser cases.

We use similar methods as in the analysis of sparse regimes, when $\kappa \to \infty$, easy to see that $\alpha \to \frac{\sqrt{2}}{2}$.

The Taylor expansion at κ sufficiently large suggest that
$\rho = -\frac{1}{2\kappa}\log\left(1 - \frac{1}{2c^2 - 1}\right) + O\left(\frac{1}{2\kappa^2}\right)$
$\log\left(1 - \frac{1}{2c^2 - 1}\right) = -\frac{1}{2c^2} + O\left(\left(\frac{1}{c}\right)^3\right)$ By Laurent series·
$\rho \approx \frac{1}{4\kappa c^2}$
Such results imply that for some regimes sufficiently dense, given reasonable c, we can pick a spherical cap non-trivially smaller than hemisphere, and get a asymptotically better exponent than applying other LSH methods.

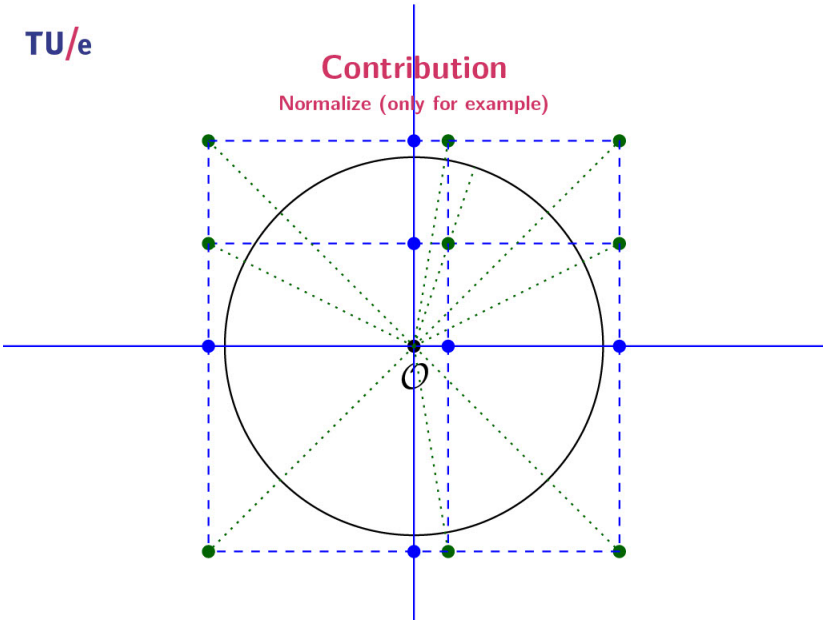

Figure 1: **From random sub codes to code words**[Bec+16], page 61

6.5 List decoding oracle

In this section we describe the the oracle that enables fast decoding mentioned in Section 6.2. The original description can be found in section 5 of [BDGL16]. Since each filter is essentially a vector in $\mathcal{S}^{d-1}$, then calculation whether a query q survives is calculating Aq, where $A \in \mathbb{R}^{t \times n}$ and each row of A corresponds to a filter vector. Clearly this requires $O(t \cdot n)$ operation, assuming each calculation between two scalars is one operation. We wish to achieve a non-trivial computation cost of this matrix-vector multiplication. One may immediately think of use pseudo random matrices to speed up this calculation. However A is not necessarily a square matrix, and t can be quite large. But we also notice that the exact value of the inner product (the output of Aq) is not useful. What we need is only whether a position in Aq (which is the result of $\langle v, p \rangle$) is greater than α or not, which immediately suggest ordering inner products, such that after we find one filter that fails, we know some others will also fail.

Instead of defining LSFs by choosing vectors that are uniformly random from $\mathcal{S}^{d-1}$, we repeatedly pick shorter vectors from a much smaller space $\mathcal{S}^{b-1}$ randomly, and concatenate them together to get a d dimensional vector. One can see a low dimensional example in Fig. 1, where 3 the code word is divided into two parts and 3 sub words are sampled for each part to get 9 code words. Parameter t is the number of filters required in the algorithm, and we assume t satisfies the requirements stated above in Section 6.2. These requirements will be specified near the end of this section.

We pick $m = \Theta(\log d)$, and that $m, b := \frac{d}{m} \in \mathbb{Z}$. This allows us to partition the coordinates into m regions, each with length b. And in each region, we sample $t^{\frac{1}{m}}$ vectors from $\mathcal{S}^{b-1}$. We then give all possible concatenations between regions, which gives $\left(t^{\frac{1}{m}}\right)^m = t$ vectors in $\mathcal{S}^{d-1}$, assuming they have been normalized by $\frac{1}{\sqrt{m}}$. Formally, we define the list of LSF vectors to be C (a.k.a. 'code word' in the original paper) where $C = \left(C_1 \times C^2 \times \cdots \times C^m\right)$ and that $C^i \subseteq \mathcal{S}^{b-1}$. Each C^i contains $t^{\frac{1}{m}}$ 'sub codes' c^i_j, and assume all vectors (shorter and concatenated ones) are standardized.

Now, without the loss of generality, assume there is at least one filter that captures the query

q. Then clearly q must survives the filter defined by the code word that has the largest inner product. We may replace each block of this code word and see how it influence the inner product due to inner product is calculated as sums. Then, if the sub codes we currently picked + the largest sub codes in regions we have not yet picked is smaller than α, the current choice cannot be a potential prefix, and we know any sub codes smaller than what we have picked also cannot be a candidate, thus pruning the search area. The algorithm proceed as follows.

Algorithm 1: List Decoding

Data: A query q, the set of sub codes C^1 through C^m, parameter α
Result: A list L which contains all filters q survives
begin
  Partition q into m segments $q^1, q^2, \cdots, q^m$, then $\forall C^i, \forall c_j^i \in C^i$, calculate $\langle c_j^i, q^i \rangle$ and store them in D^i;
  Sort each D^i in decreasing order, and denote d_j^i to be the j largest element in D^i;
  Initialize $S \longleftarrow \varnothing$;
  for $j_1 \in [2, t^{\frac{1}{m}}]$, ***assert*** $\sum_{i=1}^{1} d_{j_i}^i + \sum_{k=2}^{m} d_1^k \geq \alpha$ **do**
    for $j_2 \in [2, t^{\frac{1}{m}}]$, ***assert*** $\sum_{i=1}^{2} d_{j_i}^i + \sum_{k=3}^{m} d_1^k \geq \alpha$ **do**
      $\vdots$
      for $j_m \in [2, t^{\frac{1}{m}}]$, ***assert*** $\sum_{i=1}^{m} d_{j_i}^i \geq \alpha$ **do**
        $S \longleftarrow S \cup \{\left(d_{j_1}^1, d_{j_2}^2, \cdots, d_{j_m}^m\right)\}$
return S

Note that the original paper used a random rotation, which takes $O\left(n^2\right)$ computation, or $O\left(\log n\right)$ if replaced using several fast Hadamard transforms. If the code word achieved by this algorithm is sufficiently random, it should not need this random rotation. There is a small typo in algorithm description.

Calculating inner product requires $O\left(t^{\frac{1}{m}} n\right)$, sorting each D^i takes $O\left(t^{\frac{1}{m}} \log t^{\frac{1}{m}}\right)$. Easy to see that each time we reach the last for loop, we are guaranteed to at least include 1 code word to S, or it may fail at somewhere before the last for loop. And thus total of $2mk$ evaluations will be made, where k is the number of filters q survives. We have $2k$ dominates $t^{\frac{1}{m}} \log t^{\frac{1}{m}}$ if we simply set $t = \Theta\left(n\right)$. And in sparser settings, we can amplify m by the scale of d compared to $\log n$.

We also need the algorithm to return an answer with high probability, e.g., $1 - \epsilon$. Thus we need at least one filter captures the query q and its θ_1 close neighbor p. Then with t filters, we achieve this goal with probability $1 - \left(1 - p\left(\theta_1\right)\right)^t = O\left(t \cdot p\left(\theta_1\right)\right) \geq 1 - \epsilon$ by Union Bound. And we need $t \propto \frac{1}{p(\theta_1)}$.

This concatenated code word family has been shown to behave similarly as fully random code words, in [BDGL16], theorem 5.1. They also argued the set of sub codes C^1 through C^m can be identical to save space.

7 Intuition behind the algorithm

The analysis in 6.4.1 has shown spherical LSF method is asymptotically same as cross-polytope (and many other LSH methods). One might be curious why this LSF is asymptotically same as other

methods when dimension is high, but achieves a better asymptotic performance when dimension is low, and how fast will this exponent change. In this section we wish to give the intuition how κ influence ρ.

By looking at the definition of LSF, we notice it is a more "relaxed" version compared to LSH, that we do not guarantee that a filter returns a value when given a query. As a result, we got the two expressions for p_1 and p_2 in Equation 1 and Equation 2. Notice we replaced $\frac{1}{p_1}$ with $\frac{p(0)}{p_1}$ in the expression of ρ (compared to the normal LSH exponent) due to the special property of LSFs. Since $p(0)$ is no more than 1, by evaluating the equation, ρ decreases as $p(0)$ gets smaller, which suggest a place where this algorithm can get a better performance.

If ρ decreases as $p(0)$ decreases, ideally we would make $p(0)$ to be as small as possible. However, making $p(0)$ too small would cause our query time no longer being balanced. We gave the expression for α when we were balancing the number of filters q survives and the number of items, where $\left(1-\alpha^2\right) = n^{\frac{2}{d}}\left(1-2\alpha^2\right) = e^{2\kappa}\left(1-2\alpha^2\right)$ (For simplicity we assume $\theta_2 = \frac{\pi}{2}$). We would need an exponential decrease in α^2 if we decrease κ linearly, which cause α to quickly approach 0 much faster than κ decrease, even $d = \Theta(\log n)$. Geometrically, this make sense since as dimension gets smaller (without changing the size of data set), number of points in the same bucket dominates the number of filters q survives, and we can increase α. And in an more idealized setting, (this is more similar to hypercube LSH, which partitions the space into $2^d = \Theta(n)$ areas), we wish to have in expectation 1 point in a partition (cap), thus one cap should occupy $\frac{1}{n}$ of all the sphere. And we can get $\frac{1}{n} = d^{\Theta(1)}\left(\sqrt{1-\alpha^2}\right)^d = \frac{1}{e^{\kappa d}}$. We obtain $1 - e^{-2\kappa} = \alpha^2$ further support the relationship between α and κ (We are not restricting $p(\theta_2)$, restricting it only requires to replace α by γ and would give similar relationship).

This suggest that even though the algorithm gives a better LSH exponent, it returns to the original exponent exponentially fast as dimension scales, but it does provide a valid solution when dimension has certain restrictions. One may use asymmetric parameters during preprocessing and at query time. The conjecture of the optimality of this trade-off was stated in [Laa15] and proven in [And+16]. If we allow asymmetric cost, we can get better ρ for κ in a extended range (though it would still go back to $\frac{1}{2c^2-1}$ exponentially fast), at the cost of extra space, or vice versa.

8 Discussions

Finding better code words

From the process of determining ρ, we have found ρ is directly determined by α, which is the outcome of balancing the number of buckets and the number of points in those buckets. The oracle described in Section 6.5 gives a fast decoding method that requires $t = \Theta(n)$. One possible advancement is to find a family of code words, that is faster to compute, while maintaining enough randomness over the hypersphere. It may be possible to use sparser vectors, however, sparse vectors may fail to capture the features of data points, if they are also sparse. Yet, since cross-polytope LSH is asymptotically optimal as spherical LSH (which is impractical to implement), we might be able to adapt the method used in [AC06][AR15a] to get a construction of code words which is sparse enough to enable fast calculation, and also sufficiently "nice" (behaves similar to full randomness) over the sphere.

Extend to other LSH techniques

In Section 7, we gave a higher level overview of the reasons that LSF performs asymptotically better when dimension is low, which is the relaxation on LSF functions. One possible extension is to apply similar probabilistic techniques to other LSH approaches, and possibly get similar results when dimension is low. One might notice if the relaxation on spherical caps is removed (that all $t = \Theta(n)$ filter functions are guaranteed to return 1 or 0), the partition will be similar to Hypercube LSH, which maps points to all possible vertices of hypercube. However, in [Laa19] they claim Hypercube hashing is asymptotically sub optimal. It might be possible to make Hypercube LSH also asymptotically optimal, using techniques from LSF.

References

[AC06] Nir Ailon and Bernard Chazelle. "Approximate Nearest Neighbors and the Fast Johnson-Lindenstrauss Transform". In: *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '06. Seattle, WA, USA: Association for Computing Machinery, 2006, pp. 557–563. ISBN: 1595931341. DOI: 10.1145/1132516.1132597. URL: https://doi.org/10.1145/1132516.1132597.

[AR15a] Alexandr Andoni and Ilya P. Razenshteyn. "Optimal Data-Dependent Hashing for Approximate Near Neighbors". In: *CoRR* abs/1501.01062 (2015). arXiv: 1501.01062. URL: http://arxiv.org/abs/1501.01062.

[AR15b] Alexandr Andoni and Ilya P. Razenshteyn. "Tight Lower Bounds for Data-Dependent Locality-Sensitive Hashing". In: *CoRR* abs/1507.04299 (2015). arXiv: 1507.04299. URL: http://arxiv.org/abs/1507.04299.

[And+15] Alexandr Andoni et al. "Practical and Optimal LSH for Angular Distance". In: *CoRR* abs/1509.02897 (2015). arXiv: 1509.02897. URL: http://arxiv.org/abs/1509.02897.

[And+16] Alexandr Andoni et al. "Optimal Hashing-based Time-Space Trade-offs for Approximate Near Neighbors". In: *CoRR* abs/1608.03580 (2016). arXiv: 1608.03580. URL: http://arxiv.org/abs/1608.03580.

[BDGL16] Anja Becker et al. "New directions in nearest neighbor searching with applications to lattice sieving". In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*. Ed. by Robert Krauthgamer. SIAM, 2016, pp. 10–24. DOI: 10.1137/1.9781611974331.ch2. URL: https://doi.org/10.1137/1.9781611974331.ch2.

[Bec+16] Anja Becker et al. *New directions in nearest neighbor searching with applications to lattice sieving*. Last visited on 05/11/2023. 2016. URL: https://thijs.com/docs/soda16-soda.pdf.

[Cha02] Moses S. Charikar. "Similarity Estimation Techniques from Rounding Algorithms". In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC '02. Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 380–388. ISBN: 1581134959. DOI: 10.1145/509907.509965. URL: https://doi.org/10.1145/509907.509965.

[JL84] William Johnson and Joram Lindenstrauss. "Extensions of Lipschitz maps into a Hilbert space". In: *Contemporary Mathematics* 26 (Jan. 1984), pp. 189–206. DOI: 10.1090/conm/026/737400.

[Laa15] Thijs Laarhoven. "Tradeoffs for nearest neighbors on the sphere". In: *CoRR* abs/1511.07527 (2015). arXiv: 1511.07527. URL: http://arxiv.org/abs/1511.07527.

[Laa19] Thijs Laarhoven. "Polytopes, lattices, and spherical codes for the nearest neighbor problem". In: *CoRR* abs/1907.04628 (2019). arXiv: 1907.04628. URL: http://arxiv.org/abs/1907.04628.