Spherical LSH for Approximate Nearest Neighbor Search on Unit Hypersphere

Kengo Terasawa and Yuzuru Tanaka

Meme Media Laboratory, Hokkaido University,
N-13, W-8, Sapporo, 060–8628, Japan
{terasawa,tanaka}@meme.hokudai.ac.jp

Abstract. LSH (Locality Sensitive Hashing) is one of the best known method for solving the c-approximate nearest neighbor problem in high dimensional spaces. This paper presents a variant of LSH algorithm, focusing on the special case of where all points in the dataset lie on the surface of the unit hypersphere in the d-dimensional Euclidean space. LSH scheme is based on the family of hash functions that preserves locality of points. This paper points out that, when all points are constrained to lie on the surface of the unit hypersphere, there exists hash functions that partition the space more efficiently than the previously proposed methods. The design of these hash functions uses the randomly rotated regular polytopes and partitions the surface of the unit hypersphere like a Voronoi diagram. Our new scheme improves the exponent ρ, the main indicator of the performance of LSH algorithm.

1 Introduction

Nearest Neighbor Search is one of the fundamental problems in computer science with applications in information retrieval, pattern recognition, clustering, machine learning, data mining, and so forth. If we have the set of data points with n population and each data point has dimensionality of d, the brute-force search can find nearest neighbor in $O(dn)$ time. Nowadays, since the size of the data we should treat tends to become larger and larger, the linearity of the complexity to the size of population has posed a problem. Therefore, we need some new data structure that returns the nearest neighbor of an arbitrarily given point faster than brute-force method. The main objective of the algorithms for nearest neighbor search is to build a data structure which, given any query point q, reports the data point quickly that is closest to q.

As a result of intensive research effort, this problem was well solved particularly for low dimensional spaces. However, all of them tend to fail when the dimensionality goes higher: sometimes the time complexity asymptotically tends to $O(dn)$, which means no improvement over brute-force method, and sometimes they need memory space exponential to d, which is of course infeasible when d is large. In this way the nearest neighbor search in high dimensions is still a difficult problem. This difficulty is known as “the curse of dimensionality.”

In recent years, an approximation method has been proposed to overcome the curse of dimensionality. c-approximate nearest neighbor problem is defined as follows: for a given query point q, reports a point within the distance that is c times larger than the distance to the nearest neighbor. Randomized (or probabilistic) algorithm is also often employed to overcome the curse of dimensionality: with a fixed parameter δ, the algorithm should return requested point with probability no less than $1-\delta$. Using these approximation, many algorithms have been proposed. Among them, one of the best known algorithm is Locality Sensitive Hashing[1–3], which is also called LSH.

LSH [1–3] is a randomized algorithm for approximate nearest neighbor search problem that runs significantly faster than other existing method especially in high dimensional spaces. The basic idea of LSH is to hash each point into the hash tables using a hash function randomly chosen from locality sensitive hash function family. In finding the nearest neighbor, LSH scans only the points which have the same hash index as the query point. It runs in $\tilde{O}(n^\rho)$ time, where ρ is the the main indicator of the performance and it satisfies $\rho < 1$. The value of ρ has been improved several times during this decade. It was reported that the state-of-the-art algorithm [3] for high dimensional spaces has reduced the value ρ to 0.5563 for $c = 1.5$, and to 0.3641 for $c = 2.0$.

The main indicator of the performance, noted as ρ, was determined by the design of the hash functions. The firstly proposed LSH [1] used random bit extraction from the unary expressions, and showed the performance $\rho = 1/c$. In later improvement of LSH [2], random projection based on p-stable distributions was employed and ρ was slightly improved from earlier versions. The most recent improvement of LSH is found in [3], where they proposed novel hash functions and ρ was significantly improved especially for $d \leq 24$.

As seen before, much effort has been made for solving approximate nearest neighbor problem in $\mathbb{R}^d$ space. In practical applications of pattern recognition, however, it is often the case that to find the nearest neighbor on the hypersphere is more important than that in the entire $\mathbb{R}^d$ space. In such cases, descriptor-vectors are sometimes normalized to unit length in preprocessing because their magnitudes are less important than their directions. For example, SIFT (Scale Invariant Feature Transform) descriptor [4, 5], which is one of the most famous descriptor in computer vision, uses 128-dimensional descriptor-vector normalized to unit length. In text processing, the cosine similarity [6] is widely used. Note that nearest neighbor search with cosine similarity measure is equivalent to nearest neighbor search with Euclidean distance measure after normalizing all vectors to unit length. Other than mentioned here, we have many examples of pattern recognition algorithms that use vectors normalized to unit length.

In this paper, we focus on the special case of the approximate nearest neighbor problem — all points in the dataset are constrained to lie on the surface of the unit hypersphere — and propose a variant of LSH that efficiently solves this special problem. Since our algorithm works for datasets of points on hypersphere, we named our algorithm as Spherical LSH (SLSH).

2 Locality Sensitive Hashing (LSH)

Since our proposal is a variant of LSH algorithm, we must describe about LSH. The following is a brief introduction to LSH algorithm. More detailed description will be found in [1].

LSH is a randomized algorithm for solving the (R, c)-NN problem. (R, c)-NN problem is a decision version of the approximate nearest neighbor problem. It is known that c-approximate nearest neighbor problem can be reduced to (R, c)-NN problem with complexity $O(\log(n/\varepsilon))$. In the following, c is an approximation factor, and let $c = 1 + \varepsilon$.

Definition 1 **c-approximate nearest neighbor problem** *is defined as: Given a set P of points in a d-dimensional space $\mathbb{R}^d$, devise a data structure which for any query point $q \in \mathbb{R}^d$ find a point $p \in P$ that satisfies for all $p' \in P$, $d(p, q) \leq c \cdot d(p', q)$.*

Definition 2 **(R, c)-NN problem** *is defined as: Given a set P of points in a d-dimensional space $\mathbb{R}^d$, and a parameter $R > 0$, devise a data structure which for any query point $q \in \mathbb{R}^d$ does the following:*

- *if there exists point $p \in P$ s.t. $d(p, q) \leq R$ then return YES and a point $p' \in P$ s.t. $d(p', q) \leq cR$,*
- *if $d(p, q) > cR$ for all $p \in P$ then return NO.*

In [7], Har-Peled showed the following theorem.

Theorem 1. *c-approximate nearest neighbor problem can be reduced to (R, c)-NN problem with complexity $O(\log(n/\varepsilon))$.*

LSH can solve (R, c)-NN problem significantly faster than other existing method especially in high dimensional spaces. The basic idea of LSH is to hash every point in the dataset into the hash tables using a hash function randomly chosen from the locality sensitive hash function family. Finding the nearest neighbor of a query point involves applying the hash functions to the query point and accumulating the points in the dataset that appear in the corresponding buckets.

The locality sensitive hash function family is an important constituent of the LSH algorithm. For a domain S of the point set, an LSH family is defined as:

Definition 3 *A family $\mathcal{H} = \{h : S \to U\}$ is called (r_1, r_2, p_1, p_2)-sensitive if for any $u, v \in S$,*

- *if $d(u, v) \leq r_1$ then $Pr_{\mathcal{H}}[h(u) = h(v)] \geq p_1$,*
- *if $d(u, v) > r_2$ then $Pr_{\mathcal{H}}[h(u) = h(v)] \leq p_2$,*

where $d(u, v)$ is the distance between u and v.

In order for a LSH family to be useful, it has to satisfy inequalities $p_1 > p_2$ and $r_1 < r_2$.

For solving the (R, c)-NN problem, LSH sets $r_1 = R$ and $r_2 = cR$, and then amplifies the difference of collision probabilities by concatenating them, i.e.,

$$g(p) = \{\ h_1(p), h_2(p), \ldots, h_k(p)\ \}, \tag{1}$$

where h_i is randomly chosen from (r_1, r_2, p_1, p_2)-sensitive hash function family $\mathcal{H}$. For a query point q, LSH scans only the points who stay in the same bucket as $g(q)$. Since the process is probabilistic, it could occur that the query point and nearest point stay away from each other. In order to reduce such false negatives, LSH algorithm makes L hash tables, and scans the points in the union of the buckets corresponding to each of $g_1(p), g_2(p), \ldots, g_L(p)$.

From the settings above, we can obtain the following theorem:

Theorem 2. *LSH can solve (R, c)-NN problem with $O(dn + n^{1+\rho})$ space and $\tilde{O}(n^\rho)$ time, where $\rho = \frac{\log 1/p_1}{\log 1/p_2}$*

Now, the remaining problem is to design the locality sensitive hash functions. The firstly proposed LSH [1], which works for Hamming metric space, used random bit extraction from the unary expressions. It showed the performance $\rho = 1/c$. The later improvement of LSH [2] extended the target metric space to arbitrary l_p-norm space with $p \in (0, 2]$, and ρ is slightly improved than the earlier version by using random projection based on p-stable distributions as the locality sensitive hash functions. Most recent improvement of LSH [3] employed "ball partitioning" instead of the former "grid partitioning" to partition the space and bounded the complexity by

$$\rho = \frac{1}{c^2} + O\left(\frac{\log \log n}{\log^{1/3} n}\right). \tag{2}$$

For the practical variant, they also proposed to use the Leech Lattice-based partitioning, which is likely to perform better than aforementioned "ball partitioning" due to much lower "big-Oh" constants. It uses lattice called Leech Lattice [8], which is very symmetric lattice embedded in the 24-dimensional space. To use Leech Lattice, they have to do dimensionality reduction when the dimensionality is larger than 24.

These improvements are all concerned with the problem of: "How to partition the space well?" Leech Lattice-based partitioning, which is round and symmetric, is the quite nice partitioning except that it can be applied only to 24-dimensional space.

Now remind that the purpose of this paper is to solve the nearest neighbor problem on the unit hypersphere. We have to solve the special problem to partition the unit hypersphere embedded in $\mathbb{R}^d$, while the example presented above are all considering general problem to partition entire $\mathbb{R}^d$ space. Is there any partitioning that works nicely especially for the hypersphere? Our answer will be described in the next section.

3 Spherical LSH (SLSH)

Here we propose a novel locality sensitive hash functions which perform better than previously proposed ones. While earlier LSH families are considering arbitrary points in $\mathbb{R}^d$ space as in [1–3], we are considering arbitrary points on the unit $(d-1)$-sphere embedded in $\mathbb{R}^d$ space with center at the origin. In other words, all we have to do is to partition the surface of the unit hypersphere in $\mathbb{R}^d$, in contrast to the fact that the earlier LSH algorithm [1–3] had to partition entire $\mathbb{R}^d$ space. This section describes the locality sensitive hash functions for partitioning the surface of the unit hypersphere in high dimensions. We named this process as SLSH (Spherical LSH).

3.1 Problem Description

The problem of interest in this paper is defined as follows.

Definition 4 **(R, c)-NN problem on Unit Hypersphere:** *Given a set P of points in a d-dimensional space $\mathbb{R}^d$, and all points $p \in P$ satisfies that $||p|| = 1$. Given a parameter $R > 0$, devise a data structure which for any query point $q \in \mathbb{R}^d$ does the following:*

- *if there exists point p s.t. $d(p, q) \leq R$ then return YES and a point p' s.t. $d(p', q) \leq cR$,*
- *if $d(p, q) > cR$ for all $p \in P$ then return NO.*

This is a special case of (R, c)-NN problem (Def. 2), but with wide application area as described in Section 1.

3.2 Locality Sensitive Hash Functions using Regular Polytope

SLSH uses the randomly rotated regular polytope for partitioning the surface of the unit hypersphere. After rotating the polytope at random, then the hash function $h(p)$ is defined as the number assigned to the vertex which is nearest to p. In other words, our hash function partitions the surface of the unit hypersphere like a Voronoi diagram.

First let us go through some notations.

Hypersphere: Hypersphere is the generalization of the sphere to higher dimensions. Often the symbol $\mathbb{S}^n$ is used to repsesent the n-sphere that has n surface dimensions and embedded in $(n+1)$-dimensional space.
Unit hypersphere is the hypersphere whose radius is unity. From now on, we consider the unit $(d-1)$-sphere, it means the unit hypersphere embedded in $\mathbb{R}^d$, whose center is located at the origin.

Regular polytope: Regular polytope is the generalization of the regular polygon (in two-dimensional space) and the regular polyhedron (in three-dimensional space) to higher dimensions. It has a high degree of symmetry such as:

- All edges have the equal length. It means that the distance between the adjacent vertexes are always the same.
- All faces are congruent.

It is known that there exists only three kinds of regular polytopes in higher ($d \geq 5$) dimensions, namely,

simplex, having $d+1$ vertices, is analogous to the tetrahedron.
orthoplex (Cross polytope), having $2d$ vertices, is analogous to the octahedron.
hypercube (Measure polytope), having 2^d vertices, is analogous to the cube.

Suppose that we randomely rotate the regular polytope inscribed in a unit $(d-1)$-sphere. We can partition the $(d-1)$-sphere so that all point belongs to the nearest vertex of the rotated regular polytope.

Definition 5 (Key idea of our algorithm): *Let $\{\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_N\}$ ($||\tilde{v}_i||^2 = 1$) be a set of vertices that forms a regular polytope in $\mathbb{R}^d$ where N represents the number of vertices of the employed polytope, and let A be a rotation matrix. For an arbitrary unit vector p, a hash function $h_A(p)$ is defined as:*

$$h_A(p) = \text{argmin}_i \ ||A\tilde{v}_i - p||^2. \tag{3}$$

Note that for a given p, we can obtain $h_A(p)$ in $O(d^2)$ time for every type of regular polytope (it will be discussed later).

By considering A as arbitrary rotation matrix in $\mathbb{R}^d$ space, $\mathcal{H} = \{h_A\}$ satisfies the definition of locality sensitive hash function family. SLSH uses this LSH family for hashing.

3.3 The Algorithm

Here we will describe the details of the algorithm.

The coordinate of the vertices of the regular polytope in d-dimensional space is given by:

simplex:

$$[\tilde{\boldsymbol{v}}_i]_j = \delta_{ij} - \frac{d+1-\sqrt{d+1}}{d(d+1)} \qquad (i = 1, 2, \ldots, N) \tag{4}$$

$$[\tilde{\boldsymbol{v}}_{N+1}]_j = \frac{1-\sqrt{d+1}}{d} - \frac{d+1-\sqrt{d+1}}{d(d+1)} \tag{5}$$

where $[\tilde{\boldsymbol{v}}_i]_j$ represents j-th coordinate of the i-th vertex $\tilde{\boldsymbol{v}}_i$, and δ_{ij} is 1 for $i = j$ and 0 otherwise.

orthoplex: All permutations of $(\pm 1, 0, 0, \cdots, 0)$ give the coordinate of the vertices. It follows that orthoplex has $2d$ vertices.

hypercube: $\frac{1}{\sqrt{d}} \times (\pm 1, \pm 1, \cdots, \pm 1)$ give the coordinate of the vertices. It follows that hypercube has 2^d vertices.

Let us consider how to obtain the nearest vertex efficiently. Instead of directry solving Eq.(3), it is computationally easier to solve

$$h_A(p) = \text{argmax}_i (A\tilde{v}_i \cdot p). \tag{6}$$

If we calculate $\{v_i = A\tilde{v}_i \mid i = 1, \cdots, N\}$ in advance, $d{+}1$ dot-product calculation would suffice to return $h_A(p)$ for simplex. For orthoplex, doing $2d$ dot-product calculation is also possible, however, more efficient way exists. If v is a vertex of the orthoplex, then $-v$ is also a vertex of the orthoplex. The dot-product of $-v$ and p is just $-(v, p)$. Therefore, we do not have to calculate dot-product $2d$ times — only d times calculation would suffice. For hypercube, naively solving Eq.(6) needs 2^d times dot-product calculation that is of course infeasible. However, the way exists to avoid such prohibitive calculations. The same partitioning can obtained by d times bisection using orthonormal basis vectors $\{e_1, \cdots, e_d\}$. We can bisection the hypersphere using each one of the basis vectors, i.e., $Bisec_{e_i}(p) = 1$ if $(e_i \cdot p) \geq 0$ and $Bisec_{e_i}(p) = 0$ otherwise. Then we could construct the map $p \in \mathbb{S}^{d-1} \mapsto \{0, 1\}^d \mapsto \{0, 1, \cdots, 2^d{-}1\}$. This partitioning is equivalent to partitioning based on nearest vertexes of hypercube. Thus, we can conclude that: for every type of regular polytope, $h_A(p)$ can be calculated in $O(d^2)$ time.

Let us discuss about preprocessing cost. Preprocessing of SLSH costs $O(d^3 + dn)$ time for one hash table. The former d^3 is the cost to make random rotation matrix. The latter dn is to hash for all points in the dataset. The memory space overhead is $O(d^2 L)$ to storage the rotated vertices, and $O(nL)$ to storage the hash index of all points in the datasets.

The algorithm is summerized in Fig. 3 and Fig. 4

4 Performance Evaluation

As mentioned before, the performance of LSH is evaluated by the index $\rho = \frac{\log 1/p_1}{\log 1/p_2}$. In this section we describe about the evaluation of the ρ.

Again p_1 is the probability of collision of two points with distance R, and p_2 is the probability of collision of two points with distance cR. In the original LSH, since they can change the scale of the coordinate, they can assume $R = 1$ without loss of generality. On the other hand, we can not scale the coordinate because our method works only to $\mathbb{S}^{d-1}$. Therefore we must evaluate ρ for several Rs.

For ease of the comparison, we evaluated several types of locality sensitive hash functions. The candidates were,

SLSH: our proposal. Partitioning based on the rotated regular polytope.

Leech Lattice: proposed in [3], with dimensionality reduction d to 24.

Table 1. Probabilities of collision for two points with distance r. The value of Leech Lattice is cited from [3], and other values are obtained through our Monte-Carlo simulation for 10^6 trials (We omitted the value of spherical bisection. It is just $1 - \cos^{-1}(1 - r^2/2)/\pi$.)

Distance r	LeechLattice ($d > 24$)	16-dim simplex	16-dim orthoplex	16-dim hypercube	64-dim simplex	64-dim orthoplex	64-dim hypercube
0.10		0.90133	0.88612	0.59084	0.87169	0.85846	0.12152
0.20		0.80746	0.77939	0.33587	0.75356	0.73061	0.01271
0.30		0.71800	0.67894	0.18092	0.64590	0.61412	0.00116
0.40		0.63309	0.58535	0.09186	0.54531	0.50879	0.00008
0.50		0.55276	0.49754	0.04315	0.45407	0.41365	0.00000
0.60		0.47649	0.41595	0.01836	0.37078	0.32937	0.00000
0.70	0.08535	0.40459	0.34066	0.00676	0.29652	0.25503	0.00000
0.80	0.05259	0.33750	0.27211	0.00212	0.23071	0.19144	0.00000
0.90	0.03117	0.27473	0.21051	0.00050	0.17326	0.13748	0.00000
1.00	0.01779	0.21676	0.15533	0.00006	0.12449	0.09314	0.00000
1.10	0.00975	0.16419	0.10797	0.00000	0.08456	0.05854	0.00000
1.20	0.00515	0.11785	0.06906	0.00000	0.05260	0.03326	0.00000
1.30	0.00266	0.07826	0.03872	0.00000	0.02921	0.01656	0.00000
1.40	0.00133	0.04622	0.01789	0.00000	0.01378	0.00644	0.00000
1.50	0.00067	0.02253	0.00587	0.00000	0.00504	0.00181	0.00000
1.60	0.00033	0.00775	0.00108	0.00000	0.00117	0.00028	0.00000
1.70	0.00016	0.00124	0.00006	0.00000	0.00010	0.00001	0.00000
1.80	0.00008	0.00003	0.00000	0.00000	0.00000	0.00000	0.00000
1.90	0.00004	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
2.00	0.00002	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

Spherical Bisection: another candidate for partitioning the hypersphere. It bisection the hypersphere by random set of vectors, in contrast to our "hypercube-partitioning" bisection the hypersphere by orthonormal set of vectors.

Let $p(r)$ represent the collision probability of single hash function with respect to distance r, i.e., for point u and v that satisfy $||u - v|| = r$, $p(r)$ is defined as $p(r) = \Pr_{\mathcal{H}}[h(u) = h(v)]$.

In the Table 1, the value of $p(r)$ is displayed. The value of Leech Lattice was cited from [3]. The value of Spherical Bisection was obtained by the following equation:

$$p(r) = 1 - \theta/\pi, \tag{7}$$

where θ is the angle between two vectors measured in radians. The cosine of θ and Euclidean distance r has the relationship as:

$$r^2 = 2(1 - \cos\theta). \tag{8}$$

For SLSH, we obtained $p(r)$ by Monte-Carlo simulation for 10^6 trials.

Table 2. The values of ρ

R	c	Leech Lattice ($d > 24$)	SLSH for 64-dim orthoplex
0.64	1.5	0.5563	0.5471
0.72	1.5	0.5563	0.5189
0.80	1.5	0.5563	0.4858
0.56	2.0	0.3641	0.3456
0.64	2.0	0.3641	0.3063

Figure 1 and 2 plots the value of ρ vs. $R = r_1$. The value of $\rho = \frac{\log 1/p_1}{\log 1/p_2}$ was calculated from $p(r)$ displayed in Table 1 for SLSH and Spherical Bisection. In the calculation, we did not use the value $p(r)$ less than 0.00001 since such values are less reliable. For Leech Lattice case, since the coordinate scale are changeable, only the best ρ over R is plotted.

From these figure, we can observe the following. For any dimensionalities and for any polytopes, SLSH performs better than Spherical Bisection method. Comparing ρ on the same polytopes, larger dimensionality implies better ρ. It means that our method could avoid the curse of dimensionality. Comparing ρ on the various types of polytopes, simplex and orthoplex tend to show the similar result except orthoplex shows slightly better result when R becomes larger. Hypercube shows quite different behavior from the others. The collision probability $p(r)$ of hypercube rapidly drops to near zero especially in high dimensional spaces. Although the index ρ is lower than the others, its use for practical application seems to be difficult. It is because too small $p(r)$ implies the necessity to use large L. Comparing SLSH with Leech Lattice-based method, in the case of $c = 1.5$, for almost all dimensionalities, SLSH performs better than Leech Lattice-based method when R is larger than 0.60–0.64. In the case of $c = 2.0$, for almost all dimensionalities, SLSH performs better than Leech Lattice-based method when R is larger than 0.48–0.52. We displayed some of the representative values in Table 2.

Here we note that $R = 0.65$ and $c = 1.5$ is a practical value found in the literature. The paper [9] says that all but 3% of the data points have their nearest neighbor within distance $R = 0.65$ in their experiment. Furthermore, if we want to implement c-approximate nearest neighbor, its reduction to (R, c)-NN needs hierarchical implementation of LSH such as $R = C_0 c^\lambda$ for $\lambda = 1, 2, \cdots, \Lambda$. SLSH helps substitute for LSH-based c-approximate nearest neighbor in some hierarchies of the implementation.

5 Discussion

Why SLSH could outperform the original LSH? One of the reasons is that SLSH could work on the original dimension. It does not need the dimensionality reduction. In dimensionality reduction, one can not avoid the chance of far away points colliding to the near point. It is the disadvantage of dimensionality reduction.

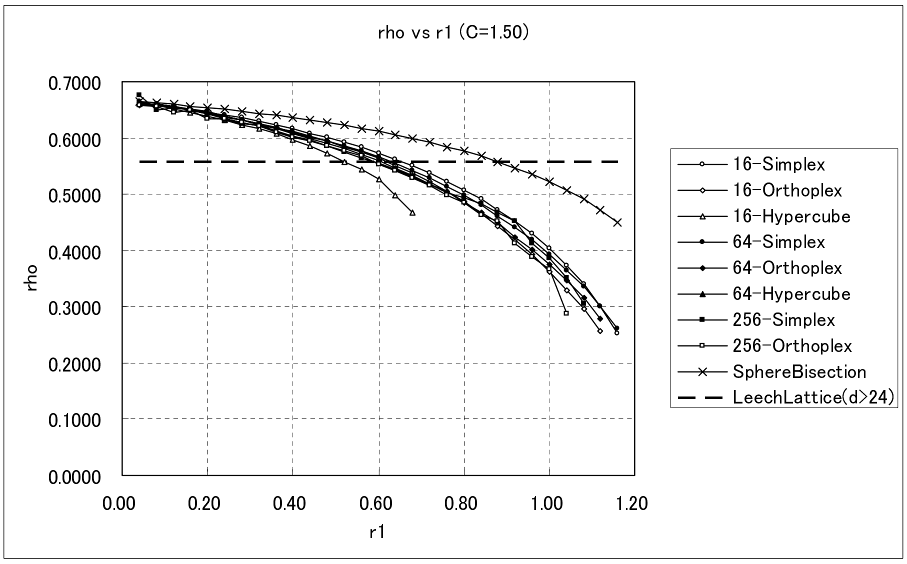

Fig. 1. The values of $\rho = \frac{\log p_1}{\log p_2}$ in case $c = 1.5$

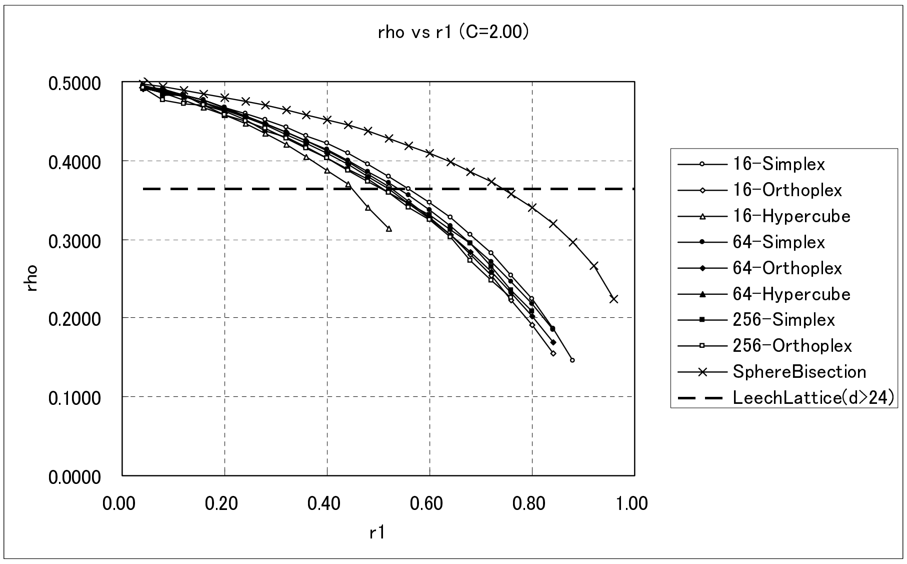

Fig. 2. The values of $\rho = \frac{\log p_1}{\log p_2}$ in case $c = 2.0$

We can avoid such disadvantages by not-using the dimensionality reduction. We have found the good partitioning that exists in any dimensional hypersphre. In our partitioning, the point $\boldsymbol{p_1}$ and $\boldsymbol{p_2} = -\boldsymbol{p_1} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon}$ is an arbitrary small vector, will *never* collide for any case of polytope, say, simplex, orthoplex, hypercube. If we try to guarantee that $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ will never collide by spherical bisection method, we need at least d times partitioning (It is similar to our hypercube method). However, like our hypercube method, it tends to partition the space too thin. It may be understood by the fact that it partitions the space into 2^d fragments. On the other hand, our simplex method and orthoplex method partition the space into $d + 1$ or $2d$ fragments. It is a milder partitioning than 2^d fragments, but far away points will collide with very little probability. Those are the keys of the efficiency of our algorithm.

6 Conclusion

In this paper we have proposed the algorithm to solve the approximate nearest neighbor problem when all points are constrained to lie on the surface of the unit hypersphere. Our algorithm, named SLSH, is based on the LSH scheme, and outperforms the state-of-the-art LSH variants.

References

1. A. Gionis, P. Indyk, and R. Motwani, "Similarity Search in High Dimensions via Hashing," Proc. 25th International Conference on Very Large Data Bases, VLDB1999, pp.518–529, 1999.
2. M. Datar, P. Indyk, N. Immorlica, and V. Mirrokni, "Locality-Sensitive Hashing Scheme Based on p-Stable Distributions," Proc. Symposium on Computational Geometry 2004, pp.253–262, 2004.
3. A. Andoni and P. Indyk, "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," Proc. 47th Annual IEEE Symposium on Foundations of Computer Science, FOCS'06, pp.459–468, 2006.
4. D.G. Lowe, "Object recognition from local scale-invariant features," Proc. 7th International Conference on Computer Vision, ICCV'99, vol.2, pp.1150–1157, 1999.
5. D.G. Lowe, "Distinctive image features from scale-invariant keypoints," International Journal of Computer Vision, vol.60, no.2, pp.91–110, 2004.
6. G. Salton and M.J. McGill, "Introduction to Modern Information Retrieval," McGraw Hill, 1983.
7. S. Har-Peled, "A Replacement for Voronoi Diagrams of Near Linear Size," Proc. 42nd Annual Symposium on Foundations of Computer Science, FOCS'01, pp.94–103, 2001.
8. J. Leech, "Notes on sphere packings," Canadian Journal of Mathematics, pp.251–267, 1967.
9. T. Darrell, P. Indyk, and G. Shakhnarovich (eds.), "Nearest Neighbor Methods in Learning and Vision: Theory and Practice," MIT Press, 2006.

```
simplex — The range of the function is: $h_A(p) \in \{1,\dots,d+1\}$
  Preprocessing:
    Let $\{\tilde{v}_i \mid i = 1,\dots,d+1\}$ be a set of vectors described in Eq. (4) and (5).
    Let $A$ be a random rotation matrix.
    For $i \leftarrow 1$ to $d+1$ do
      $v_i \leftarrow A\tilde{v}_i$
    (Then $\{v_1, v_2, \cdots, v_d\}$ forms a randomly rotated simplex.)
  Calculation of $h_A(p)$ for input $p$:
    $h_A(p) \leftarrow \mathrm{argmax}_i(v_i \cdot p)$
    Return $h_A(p)$

orthoplex — The range of the function is: $h_A(p) \in \{1,\dots,2d\}$
  Preprocessing:
    Let $v_i$ be the $i$-th column of a random rotation matrix $A$.
    (Then $\{v_1, v_2, \cdots, v_d, -v_1, -v_2, \cdots, -v_d\}$ forms a randomly rotated ortho-
    plex.)
  Calculation of $h_A(p)$ for input $p$:
    $h_A(p) \leftarrow \mathrm{argmax}_i|v_i \cdot p|$.
    if $(v_{h_A(p)} \cdot p) < 0$ then $h_A(p) \leftarrow h_A(p) + d$
    Return $h_A(p)$

hypercube — The range of the function is: $h_A(p) \in \{0,\dots,2^d-1\}$
  Preprocessing:
    Let $v_i$ be the $i$-th column of a random rotation matrix $A$.
    (Then $\{u_{s_1 s_2 \cdots s_d} = \sum s_i v_i \mid s_i = \pm 1\}$ forms a randomly rotated hypercube.)
  Calculation of $h_A(p)$ for input $p$:
    $h_A(p) \leftarrow 0$
    For $i \leftarrow 1$ to $d$ do
      if $(v_i \cdot p) \geq 0$ then $h_A(p) \leftarrow h_A(p) + 2^{i-1}$
    Return $h_A(p)$
```

Fig. 3. The implementation of $h_A(p)$ for each type of regular polytope

```
How to make a random rotation matrix:

    For $i \leftarrow 1$ to $d$ do
      Let $v_i$ a random vector from the $d$-dimensional Gaussian distribution.
      For $j \leftarrow 1$ to $i-1$ do
        $v_i \leftarrow v_i - (v_i \cdot v_j)v_j/|v_j|$      (Gram-Schmidt orthogonalization)
      $v_i \leftarrow v_i/|v_i|$      (normalize to unit length)
    Return $(v_1\ v_2\ \cdots\ v_d)$
```

Fig. 4. Algorithm for making a random rotation matrix