Robust and Efficient Locality Sensitive Hashing for Nearest Neighbor Search in Large Data Sets

Byungkon Kang
KAIST
Daejeon, Korea
byungkon@kaist.ac.kr

Kyomin Jung
KAIST
Daejeon, Korea
kyomin@kaist.edu

Abstract

Locality sensitive hashing (LSH) has been used extensively as a basis for many data retrieval applications. However, previous approaches, such as random projection and multi-probe hashing, may exhibit high query complexity of up to $\Theta(n)$ when the underlying data distribution is highly skewed. This is due to the imbalance in the number of data stored per each bucket, which leads to slow query time in large data sets. In this paper, we introduce a distribution-free LSH algorithm that addresses this problem by maintaining nearly uniform number of points per bucket. As a consequence, our algorithm allows one to reduce the number of hash tables, and is hence memory-efficient, while achieving high accuracy. Through extensive experiments, we show that our algorithm accurately retrieves nearest neighbors faster than other standard LSH algorithms do in large data sets, and maintains nearly uniform number of per-bucket points.

1 Introduction

Recent explosion in the amount of data has placed huge importance in big-data management. As a consequence, many applications over diverse fields such as database, artificial intelligence, and social network analysis have found the need to modify existing algorithms to scale the performance. Data retrieval is one such area of importance, due to its role of providing a "base system" for many other applications. In particular, we focus on *locality-sensitive hashing* (LSH), which is one of the most popular data retrieval frameworks, and present an algorithm that has low query time complexity under any metric.

LSH can be thought of as a meta-algorithm that allows for fast storing and retrieval of large number of data points, by grouping closer data items together with high probability. Some of the popular LSH algorithms that have been proposed include MinHash [6, 10], random projection [2], multi-probe LSH [7], and p-stable distribution projection [1], which have good provable accuracy guarantees. However, they may perform poorly when the underlying distribution of the data is highly skewed, resulting in serious imbalance in the number of items per bucket. This is because if the size of bucket i is b_i, the total query time becomes $\Theta(\sum_i b_i^2)$ if the query comes randomly from the underlying data distribution. This leads to highly variable query time depending on the bucket size, and is hence inefficient for large data sets. Our algorithm minimizes this quantity by evenly distributing the points in each bucket. A work by [11] proposed a similar idea, where the partitioning is done along the axes of largest variance. However, this only allows for axis-parallel partitions, whereas our approach constructs more flexible boundaries.

Another shortcoming of previous approaches is that the metric under which the similarity is measured may be restricted (*i.e.*, Hamming distance or L_2-norm). In this work, we present Distribution-Free LSH (DFLSH), that overcomes the high query complexity under any distance metric. Compared to the standard LSH algorithms mentioned above, our algorithm lowers the worst-case query complexity by gathering a uniform number of points per bucket regardless of the data distribution and

distance metric. Also, DFLSH requires less hash tables to achieve similar level of accuracy of other LSH algorithms, leading to faster running time and more efficient memory usage. We verify this fact empirically in Section 4. Finally, DFLSH efficiently operates on a variety of metrics, including the L_p-norm, shortest-path graph distance, and the kernel-induced distance metric.

1.1 Preliminaries

The problem we aim to solve is the k-*nearest neighbor* problem (k-NN problem), which provides a basis to many popular applications.

Definition 1 (k-NN problem)**.** *Given a set $P \subset U$, a distance metric $d : U \times U \mapsto \mathbb{R}^+$, and a query point $q \in U$, return a set $S = \{p_i | i = 1, \cdots, k\} \subseteq P$ of k points such that for all $r \in P - S$, $d(r, q) \geq d(p_i, q),\ \forall p_i \in S$.*

Our algorithm can be applied to any metric space U, including graph distance and the kernel-based distance, but we focus on $U \subseteq \mathbb{R}^D$ for ease of explanation. We assume that the data points P come from an unknown underlying probability density $\mathcal{P}$ on U.

To solve the k-NN problem, the LSH framework uses a collection $\mathcal{H}$ of functions that satisfies the *locality sensitive* property [3]:

Definition 2. *Given $r > 0, c > 1, 1 > p_1 > p_2 > 0$, a family of functions $\mathcal{H} = \{h | h : U \mapsto C\}$ for a set of buckets C, is called a (r, cr, p_1, p_2)-sensitive hash family if for any $p, q \in U$,*

$$Pr(h(p) = h(q)) \geq p_1, \text{ if } d(p, q) \leq r$$
$$Pr(h(p) = h(q)) \leq p_2, \text{ if } d(p, q) \geq cr,$$

for all $h \in \mathcal{H}$, where the probability is over the choice of h from $\mathcal{H}$.

The de-facto standard procedure for solving the k-NN problem using (r, cr, p_1, p_2)-sensitive hash functions is as follows. The main idea is to maintain L hash tables in order to to amplify the probability of correctly locating any given element (*i.e.*, inducing *collision*). That is, when r is the maximum distance between q and its nearest neighbor, the probability that at least one table yields the correct nearest neighbor is at least $1-(1-p_1)^L$. In the next section, we present our novel design of h for tackling the k-NN problem for arbitrary $\mathcal{P}$.

2 Algorithms

We present our Distribution-Free LSH (DFLSH) algorithm for solving the k-NN problem. The main idea of DFLSH is to randomly partition the data space into t disjoint cells in a way that each cell has roughly the same number of points. To create such a partition, we first sample $\{c_1, \cdots, c_t\} \subset P$ uniformly at random, in order to sample from the underlying $\mathcal{P}$. Based on these t points, which we call the *centroids*, we construct a *Voronoi partition* of P: a partition $\{C_1, \cdots, C_t\}$ such that $C_i = \{p | d(p, c_i) \leq d(p, c_l)\ \forall c_l\}$, where ties are broken arbitrarily. We empirically show in Section 3 that this construction partitions the data space nearly uniformly (*i.e.*, $O(n/t)$ points per bucket). We repeat the construction L times, each in a uniformly random manner, to create L tables. The overall procedure is outlined in Table 1 (Appendix A).

Querying this hash table is straightforward: 1) Iterate through all t centroids to choose the closest cell C_{i^*} (in time $O(t)$), and 2) linearly search through all points contained in C_{i^*} to select NN candidates (in time $O(n/t)$).

A reasonable value to use for t is $\sqrt{n}$, which makes the time to locate the nearest Voronoi cell and the time to linearly search through points in that cell equal. That is, setting $t = \sqrt{n}$ equates the two complexities, yielding $O(\sqrt{n})$ per table. Overall, the construction takes $O(LDn^{3/2})$ time, and the query complexity becomes $\Theta(LD\sqrt{n})$.

The above algorithm assumes that P is given in the beginning. But it is possible to extend our algorithm to an online setting, where the data points are streamed in. In this case, we adopt a dynamic centroid selection/deletion procedure, where the m^{th} point becomes a centroid with probability $m^{-1/2}$ and is removed from the centroid set at the $(m + v)^{\text{th}}$ step with probability $\sqrt{m/(v-1)} - \sqrt{m/v}$. Once a point is removed from the centroid set, it is never re-selected. This selection/deletion procedure ensures that each point is selected with probability $m^{-1/2}$ at the m^{th} step.

We also present a multi-probe DFLSH for querying, where we select ℓ nearest centroids, instead of just one, to ensure wider coverage. Multi-probing allows to maintain a small number of hash tables to maintain high accuracy and fast running time. We show this through extensive experiments in Section 4. The order of the query complexity for the multi-probe DFLSH is the same as a single-probe version, since ℓ is fixed constant. The procedure for multi-probe DFLSH is outlined in Appendix A.

It is straightforward to extend DFLSH to a kernelized setting [5] (KLSH), where the items are mapped to some feature space via the kernel trick. Given a kernel function $\kappa : U \times U \mapsto \mathbb{R}$, we can compute the distance as $d_\kappa(p,q) \triangleq \kappa(p,p) - 2\kappa(p,q) + \kappa(q,q)$. This approach yields a faster pre-processing step than the previous KLSH algorithm [5], since only the distance in the feature space is needed.

Finally, we propose a hierarchical DFLSH in Appendix A.2, which recursively partitions each Voronoi cell h times in the same manner, where h is the recursive depth. This approach yields construction and query complexities of $O(LDn^{4/3})$ and $O(LDn^{1/3})$, respectively for $h = 2$, which are less than those of the non-hierarchical DFLSH ($h = 1$). Because of the exponential decrease in the number of points per cell with h, the hierarchical scheme works best when n is very large (about > 1 million).

3 Analysis

The analysis of our algorithm is done over two aspects: 1) whether DFLSH satisfies the locality-sensitive property, and 2) the time complexity of querying a point.

Locality-Sensitivity We first analyze the probability that a miss occurs, in a single table, between two items p and q when $d(p,q) = r$. When computing this probability, we assume that there is on average one point per unit volume, without loss of generality. Although the condition for Theorem 1 is that each cell's density be uniform, the theorem is still applicable to arbitrary $\mathcal{P}$, since the distribution within each cell becomes close to a uniform distribution as n increases.

Theorem 1. *The probability that points $p, q \in \mathbb{R}^D$ do not collide, given their distance r, is*

$$p(r) = Pr(h(p) \neq h(q)|r) = \int_0^\infty u(r_1) \int_{|r-r_1|}^{r+r_1} s(r_2)(1 - \exp(-C_D r_2^D n^{-1/2}))dr_2 dr_1,$$

where C_D is the volume of a unit ball in $\mathbb{R}^D$ and,

$$u(r_1) = DC_D n^{-1/2} r_1^{D-1} \exp(-C_D n^{-1/2} r_1^D), \qquad s(r_2) = \frac{(D-1)C_{D-1} g^{D-3}}{DC_D r_1^{D-2}},$$

$$g = \frac{2}{r}\sqrt{l(l-r)(l-r_1)(l-r_2)}, \qquad l = \frac{r + r_1 + r_2}{2}.$$

Because $p(r)$ monotonically increases with r, our hash functions satisfy the $(1, c, p_1, p_2)$-sensitivity property by setting $p_1 = 1 - p(1)$ and $p_2 = 1 - p(c)$ for any $c > 1$. For the bound on accuracy, we provide a numerical analysis of $p(r)$ in Appendix C.

Query Complexity Next, we empirically show that the query complexity is low compared to other algorithms. As mentioned above, the query complexity is the squared sum of the bucket sizes: $\Theta(\sum_i b_i^2)$, where b_i is the number of points in bucket i. We plot this quantity for Multi-probe LSH (MPLSH), Non-uniform partitioning LSH (NULSH), DFLSH, and Hier-DFLSHin Figure 1, each with equal number of buckets.
The data we use is a synthetically-generated mixture of three Gaussians of dimensions 10 and 50 (see Section 4 for details). Figure 1 shows this quantity is lowest for our two algorithms for the synthetic data set.

4 Experiments

In this section, we verify the efficiency of DFLSH and Hier-DFLSH through extensive experiments. The data sets we use are the SIFT image feature data set (SIFT) [4], and an artificially-generated

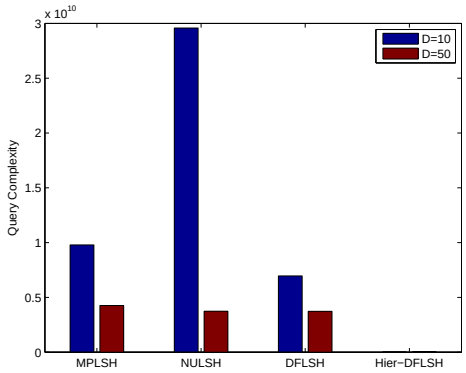

Figure 1: $\sum_i b_i^2$ of each algorithm for the synthetic data set.

set of Gaussian mixtures (AG) with varying dimensions of 10, 30, and 50. The SIFT set consits of 1 million 128-dimensional image features. The algorithms we compare against are the multi-probe LSH [7] (MPLSH)[1], and the non-uniform LSH [11] (NULSH).

In each experiment, we randomly generate 100 queries and their corresponding 100 true nearest neighbors. The accuracy of the algorithms is measured by *recall* as done by [8]: $|R \cap G|/|G|$, where G is the ground-truth set and R is the set retrieved by the algorithms. All algorithms, including the undisclosed NULSH, are implemented in C++.

Parameter Scaling For hash table parameter experiements, we examine the effects of the number of hash tables L, and the number of probes ℓ for the multi-probe variant of our algorithm. Figures 2 and 3 show that both `DFLSH` and Hier-`DFLSH` require small values of ℓ and L to achieve high accuracy, while other algorithms are more sensitive to those parameters.

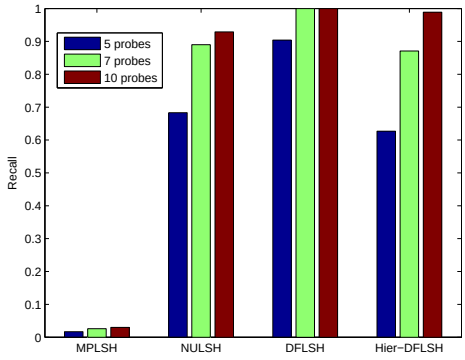

Figure 2: Accuracy as ℓ increases with $L = 5$.

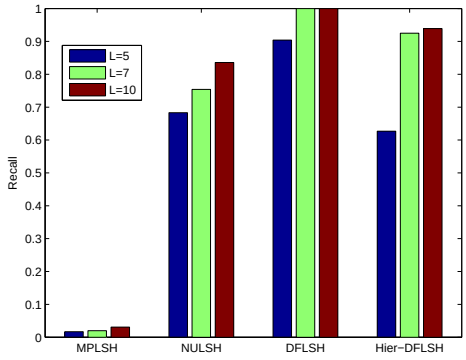

Figure 3: Accuracy as L changes, with $\ell = 2$.

Dimension Scaling This section presents the results on how performance varies with respect to the dimension of the data. In order to keep the nature of the data set as consistent as possible, we use the artificially-generated AG set, with differing dimensions. Each algorithm is tested with 10 hash tables and 2 probes. The result, which is given in Figure D, shows that `DFLSH` achieves near-perfect recall as dimension increases, while other algorithms degrade (Appendix D).

To see how long each algorithm requires to achieve the recall performances given above, we show the timing results in Table 4 (Appendix D). The timing results show that `DFLSH` requires less than 0.05 seconds per query to achieve perfect recall, whereas other algorithms either performs worse, or require ten times more query time.

[1] `LSHKIT`: `http://lshkit.sourceforge.net`

References

[1] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of SCG*, 2004.

[2] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of VLDB*, 1999.

[3] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. In *Proceedings of STOC*, 1998.

[4] H. Jégou, M. Douze, and C. Schmid. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:117–128, 2011.

[5] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of ICCV*, 2009.

[6] P. Li and A. C. König. Theory and applications of b-bit minwise hashing. *Communications of the ACM*, 54(8):101–109, 2011.

[7] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In *Proceedings of VLDB*, 2007.

[8] L. Paulevé, H. Jégou, and L. Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31:1348–1358, 2010.

[9] J. H. Pollard. On distance estimators of density in randomly distributed forests. *Biometrics*, 27:991–1002, 1971.

[10] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of SIGMOD*, 2003.

[11] Z. Yang, W. T. Ooi, and Q. Sun. Hierarchical non-uniform locality sensitive hashing and its application to video identification. In *Proceedings of ICME*, 2004.

Appendices

A Additional Algorithms

A.1 Pseudocode for Construction and Querying

Input: Set of points P, Number of cells t, Number of tables L
Output: L Hash tables $\{T_i\}_{i=1}^{L}$
  for all $i \in \{1, \cdots, L\}$ **do**
    $S_i \leftarrow t$ uniform random samples from P
    $T_i \leftarrow$ `VoronoiPartition`(P, S_i)
  end for

Table 1: `DFLSH`: High-level hash table construction.

A.2 Hierarchical Scheme

To lower the query complexity, we propose a hierarchical Voronoi partitioning scheme. In this approach, we recursively partition each Voronoi cell in the same manner up to some depth h. In order to reflect the hierarchical structure, we modify the L hash tables to be L trees, where each tree has depth h. The internal nodes of each tree indicate the centroids by which the next level is decided, and the leaves specify the final ground partition of the data.
Given a partition hierarchy, a query point is simply matched to the nearest internal node (*i.e.*, first-level centroid) and traverses down the tree until it reaches a leaf node.
Similarly to the non-hierarchical case, we choose $t = n^{1/3}$ and $h = 2$ to balance the time taken to locate the cell and the time taken to linearly search through the points in the cells. Then, the

Input: Query point q, probe number ℓ
Output: Set of nearest neighbor candidates C
  $S \leftarrow \{\}$
  for all $i \in \{1, \cdots, L\}$ **do**
    $V \leftarrow \ell$ closest centroids (in T_i) to q
    for all $c \in V$ **do**
      $S \leftarrow S \cup \{x | x \in c\}$
    end for
  end for
  Linearly search through S to return k nearest neighbors

Table 2: Query procedure for the Multi-Probe `DFLSH`.

construction and query complexities are $O(LDn^{4/3})$ and $O(LDn^{1/3})$, respectively. The following algorithm shows how to construct *one* of hierarchical hash tables. One can repeat this procedure L times to get a full table set. Because the number of points in the lowest-level cell becomes

Input: Set of points P, Number of buckets t, Current depth d
Output: Tree T
  if $d = 0$ **then**
    return P
  end if
  for all $i \in \{1, \cdots, L\}$ **do**
    $S_i \leftarrow t$ Uniform random samples from P
    $R \leftarrow$ `VoronoiPartition`(P, S_i)
    for all $R_j \in R$ **do**
      $T_j \leftarrow$`Hier-DFLSH`$(R_j, t, d-1)$
    end for
  end for

Table 3: `Hier-DFLSH`

exponentially small as the recursive depth increases, the hierarchical scheme works best when n is very large.

B Proofs

B.1 Proof of Theorem 1

Let c_p be the nearest centroid associated with point p such that $d(p, c_p) = r_1$, and $r_2 = d(q, c_p)$. Then, p and q are hashed into the same cell only if there is no centroid in $B(q, r_2)$. To compute the probability of such an event, we use the following auxiliary lemma:

Lemma 1. *Suppose the points are distributed at random with an average density of λ per unit area in $\mathbb{R}$. Then the distribution of the distance r from a random point to its nearest neighbor is given as:*

$$u(r) = DC_D\lambda r^{D-1}\exp(-C_D\lambda r^D), \tag{1}$$

where C_D is the volume of the unit ball in $\mathbb{R}^D$.

Proof. The proof is a high-dimensional extension of the one given in [9]. □

As a consequence, the cumulative distribution function (CDF)

$$U(r) = 1 - \exp(-C_D\lambda r^D) \tag{2}$$

of $u(r)$ specifies the probability that there is no point in $B(x, r)$ for a random point x.
Another probability we should consider is that of r_2; *i.e.*, the probability density $s(r_2)$ when r and r_1 are fixed. The region over the hypersphere $B(q, r_2)$ in $\mathbb{R}^D$ in which r_2 is held constant forms a

hypersphere of radius g in $\mathbb{R}^{D-1}$. Here, g is the height of the triangle whose base length is r, and the lengths of the other two sides are r_1 and r_2. By Heron's formula, g is given as:

$$g = \frac{2}{r}\sqrt{l(l-r)(l-r_1)(l-r_2)},$$

where $l = (r + r_1 + r_2)/2$. Hence, the final density $s(r_2)$ is given as:

$$s(r_2) = \frac{(D-1)C_D g^{D-2} dg}{DC_D r_1^{D-1}} = \frac{(D-1)C_D g^{D-3}}{DC_D r_1^{D-2}}, \tag{3}$$

where $dg = r/g$.
Based on Equation (1), Equation (2), and Equation (3), the probability that q is not assigned to c_p is a double integral of the equations over all possible values of r_1 and r_2. The latter is restricted in the range $[|r - r_1|, r + r_1]$ by the triangle inequality.

$$p(r) = \int_0^\infty u(r_1) \int_{|r-r_1|}^{r+r_1} s(r_2)U(r_2)dr_2 dr_1.$$

Note that this expression is exact for uniform $\mathcal{P}$, due to Lemma 1, but the density of each cell becomes locally uniform as n grows. Since we are dealing with very large values of n, the error induced by non-uniformity becomes negligible.

C Numerical Analysis of $p(r)$

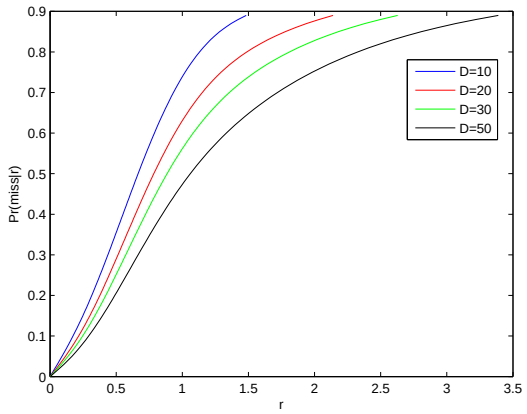

Figure 4: The plotting of r vs. $p(r)$, with varying dimensions.

Figure 4 shows the evaluation of $p(r)$ of Theorem 1 with respect to varying values of r. Note that r is the distance between a random pair of nearest neighbors. Also, since we assume that there is one point per unit volume on average, most nearest neighbors will have distance $r = 1$. This coincides with the steep increase of the miss probability around $r = 1$.
From the definitoin of LSH, choosing $L = O(1/p_1)$ yields a collision probability of $1 - \exp(-O(1))$. Thus, the plot suggests that about $1/p_1 = 1/(1 - p(1)) \approx 2,3$ tables suffice to achieve high probability of collision for our $(1, c, p_1, p_2)$-sensitive LSH functions in 50 dimensions.

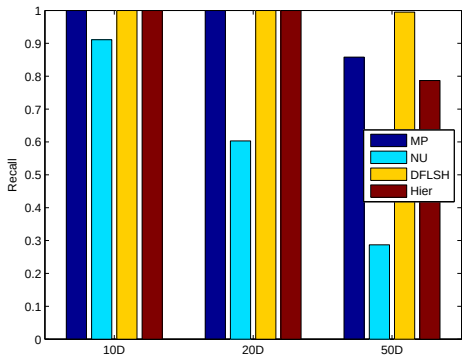

Figure 5: Recall change as dimension increases.

D Additional Experiments

Figure D shows the recall change as the data dimension increases. The plot shows that DFLSH maintains recall higher than 0.995 as the dimension is increased to 50, while the recalls of competing algorithms degrade noticeably. The numbers in the last column (SIFT) are gathered by running

	AG ($D = 10$)	AG ($D = 20$)	AG ($D = 50$)	SIFT
MPLSH	0.086 (1.0)	0.09 (1.0)	0.091 (0.858)	0.009 (0.004)
NULSH	0.175 (0.911)	0.04 (0.603)	0.04 (0.287)	0.781 (0.772)
DFLSH	0.008 (1.0)	0.012 (1.0)	0.034 (0.995)	0.045 (0.884)
Hier-DFLSH	0.002 (1.0)	0.004 (1.0)	0.009 (0.787)	0.011 (0.804)

Table 4: Comparison of average query times (in seconds) over 100 queries. Recalls are duplicated in parentheses to aid the comparison of time-recall tradeoff.

each algorithm with 5 hash tables and 2 probes. It can be seen that DFLSH and Hier-DFLSH require an order of magnitude less time than the others to achieve similar or higher recall level.