Poster Paper

Hybrid LSH: Faster Near Neighbors Reporting in High-dimensional Space

Ninh Pham
University of Copenhagen
Denmark
pham@di.ku.dk

ABSTRACT

We study the r-near neighbors reporting problem (rNNR) (or *spherical range reporting*), i.e., reporting *all* points in a high-dimensional point set S that lie within a radius r of a given query point. This problem has played building block roles in finding near-duplicate web pages, solving k-diverse near neighbor search and content-based image retrieval problems. Our approach builds upon the locality-sensitive hashing (LSH) framework due to its appealing asymptotic sublinear query time for near neighbor search problems in high-dimensional space. A bottleneck of the traditional LSH scheme for solving rNNR is that its performance is sensitive to data and query-dependent parameters. On data sets whose data distributions have diverse local density patterns, LSH with inappropriate tuning parameters can sometimes be outperformed by a simple linear search.

In this paper, we introduce a hybrid search strategy between LSH-based search and linear search for rNNR in high-dimensional space. By integrating an auxiliary data structure into LSH hash tables, we can efficiently estimate the computational cost of LSH-based search for a given query regardless of the data distribution. This means that we are able to choose the appropriate search strategy between LSH-based search and linear search to achieve better performance. Moreover, the integrated data structure is time efficient and fits well with many recent state-of-the-art LSH-based approaches. Our experiments on real-world data sets show that the hybrid search approach outperforms (or is comparable to) both LSH-based search and linear search for a wide range of search radii and data distributions in high-dimensional space.

1. INTRODUCTION

We study the r-near neighbors reporting problem (rNNR) (or *spherical range reporting*) [2, 5]: *Given a d-dimensional point set S of size n, reporting* all *points in S that lie within a radius r of a given query point.* This problem has played building block roles in finding near-duplicate web pages [11], solving k-diverse near neighbor search [1] and content-based image retrieval problems [15]. Recent theoretical work [2, 3] conjectures that solving rNNR exactly in time truly sublinear in n seems to demand space exponential in d, which is an example of the phenomenon "curse of dimensionality".

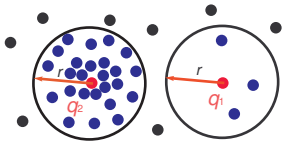


Figure 1: An example of LSH bottleneck. Given a radius r, LSH works efficiently with the query q_1 on sparse area, since it will report just a few points. However, LSH is worse than linear search with the "hard" query q_2 on dense area. Since the output size of q_2 is nearly the data set size and many points are very close to q_2, duplicates show up in most hash tables and the cost of removing duplicates will be the computational bottleneck.

Since exact solutions of rNNR generally degrade as dimensionality increases, we investigate an *approximate* variant of rNNR. That is, given a parameter $0 < \delta < 1$, we allow the algorithm to return each point in S that lie within a radius r of the query point with probability $1 - \delta$. Our approach builds upon on the *locality-sensitive hashing* (LSH) [4, 12], one of the most widely used solution for near neighbor search problems. In a nutshell, LSH hashes near points into the same bucket with good probability, and increases the gap of collision probability between near and far points. It typically needs to use multiple hash tables to obtain probabilistic guarantees. Search candidates are *distinct* data points that are hashed into the same bucket as the query in hash tables.

Since its first introduction, several LSH schemes [6, 7, 8, 10, 13, 14] have been proposed for a wide range of metric distances in high-dimensional space. However, a bottleneck of using LSH for solving rNNR is that its performance is sensitive to the parameters which depend on the distance distribution between data points and query points. Such parameters are hard to tune on data sets whose data distributions have diverse *local* density patterns. Figure 1 shows an illustration of this bottleneck.

In practice, LSH needs to use significant space (i.e., hundreds of hash tables) [10] or the multi-probe approach [13] which examines several "close" buckets in a hash table. In other words, the number of examined buckets needs to be sufficiently large to obtain high accuracy. In turn, the cost

of removing duplicates (i.e., points colliding with the query in several hash tables) turns out to be the computational bottleneck when there are many points close to the query. This observation has been shown on the Webspam dataset in the experiment section even with very small radii.

In this work, we study a hybrid search strategy between LSH-based search and linear search for rNNR in an arbitrary high-dimensional space and distance measure that allows LSH. By integrating the so-called HyperLogLog data structures [9] into LSH hash tables, we can quickly and accurately estimate the output size and derive the computational cost of LSH-based search for a given query point regardless of the data distribution. In other words, we are able to choose the appropriate search strategy between LSH-based search and linear search to achieve better performance (i.e., running time and recall ratio). Moreover, the proposed solution can be adapted to many recent state-of-the-art LSH-based approaches [2, 13, 14]. Our experiments on real-world datasets demonstrate that the proposed hybrid search outperforms (or is comparable to) both LSH-based search and linear search for a wide range of search radii and data distributions in high-dimensional space.

2. BACKGROUND AND PRELIMINARIES

Problem setting. Our problem, r-near neighbor reporting under any distance measure, is defined as follows:

DEFINITION 1. *(r-near neighbor reporting or rNNR) Given a set $S \subset \mathbf{R}^d$, $|S| = n$, a distance function f, and parameters $r > 0$, $\delta > 0$, construct a data structure that, given any query $\boldsymbol{q} \in \mathbf{R}^d$, return* each *point $\boldsymbol{x} \in S$ where $f(\boldsymbol{x}, \boldsymbol{q}) \leq r$ with probability $1 - \delta$.*

We call this the "exact" rNNR problem in case $\delta = 0$, otherwise it is the "approximate" variant.

Locality-sensitive hashing (LSH). LSH can be used for solving approximate rNNR in high-dimensional space because its running time is usually better than linear search with appropriate tuning parameters [4].

DEFINITION 2. (Indyk and Motwani [12]) *Fix a distance function $f : \mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}$. For positive reals r, c, p_1, p_2, and $p_1 > p_2$, $c > 1$, a family of functions $\mathcal{H}$ is* (r, cr, p_1, p_2)-sensitive *if for uniformly chosen $h \in \mathcal{H}$ and all $\boldsymbol{x}, \boldsymbol{y} \in \mathbf{R}^d$:*

- *If $f(\boldsymbol{x}, \boldsymbol{y}) \leq r$ then $\mathbf{Pr}[h(\boldsymbol{x}) = h(\boldsymbol{y})] \geq p_1$;*
- *If $f(\boldsymbol{x}, \boldsymbol{y}) \geq cr$ then $\mathbf{Pr}[h(\boldsymbol{x}) = h(\boldsymbol{y})] \leq p_2$.*

Given an LSH family $\mathcal{H}$, the classic LSH algorithm constructs L hash tables by hashing data points using L hash functions g_j, $j = 1, \ldots, L$, by setting $g_j = \left(h_j^1, \ldots, h_j^k\right)$, where h_j^i, $i = 1, \ldots, k$, are chosen randomly from the LSH family $\mathcal{H}$. Concatenating k such random hash functions h_j^i increases the gap of collision probability between near points and far points. To process a query $\boldsymbol{q}$, one needs to get a candidate set by retrieving all points from the bucket $g_j(\boldsymbol{q})$ in the jth hash table, $j = 1, \ldots, L$. Each *distinct* point $\boldsymbol{x}$ in the candidate set is reported if $f(\boldsymbol{x}, \boldsymbol{q}) \leq r$.

For the approximate rNNR, a near neighbor has to be reported with a probability at least $1 - \delta$. Hence, one can fix the number of hash tables, L, and set the value k as a function of L and δ. A simple computation indicates that $k = \left\lceil \log\left(1 - \delta^{1/L}\right) / \log p_1 \right\rceil$ leads to good performance[1].

[1]This is a practical setting used in E2LSH package (http://www.mit.edu/∼andoni/LSH/)

Note that our parameter setting is different from the standard setting $k = \log n, L = n^\rho$, where $\rho = \log p_1 / \log p_2$ [12], since we focus on reporting *every* r-near neighbor.

Although LSH-based algorithm can efficiently solve rNNR problem, it might run in $\mathcal{O}(nL)$ time in the worst case, see Figure 1 as an example. Tuning appropriate parameters k, L for a given dataset whose data distribution has diverse local density patterns remains a tedious process.

HyperLogLog (HLL) for count-distinct problem. While counting the exact number of distinct elements in a data stream is simple with space linear to the cardinality, approximating such the cardinality using limited memory is an important problem with broad industrial applications. Among efficient algorithms for the problem, HyperLogLog (HLL) [9] constitutes the state-of-the-art (i.e., a near-optimal probabilistic algorithm) when there is no prior estimate of the cardinality. This means that it achieves a superior accuracy for a given fixed amount of memory over other techniques.

HLL builds an array M of m zero registers. For an element i, it generates a random *integer* pair $\{m_i, v_i\}$ where $m_i \sim \texttt{Uniform}([m])$ indicates a position in M, and $v_i \sim \texttt{Geometric}(1/2)$ is an update value. The array M updates the value at the position m_i by $\max(M[m_i], v_i)$. After processing all elements, the cardinality estimator of the stream is $\theta_m m^2 \left(\sum_{j=1}^m 2^{-M[j]}\right)$, where θ_m is a constant to correct the bias. HLL works optimally with *distributed* data streams since we can merge several HLLs by collecting register values and applying component-wise a `max` operation. The relative error of HLL is $1.04/\sqrt{m}$. More details of the theoretical analysis and a practical version of HLL can be seen in [9].

3. ALGORITHM

This section describes our novel hybrid search strategy which interchanges LSH-based search and linear search for solving rNNR. We first present a simple but accurate computational cost model to measure the performance of LSH-based search. By constructing an HLL data structure in each bucket of hash tables, we are able to estimate the computational cost of LSH-based search, and then identify the condition whether LSH-based search or linear search is used.

3.1 Computational Cost Model

For each query, LSH-based search needs to process following operations: (1) Step S1: Compute hash functions to identify the bucket of query in L hash tables, (2) Step S2: Look up in each hash table the points of the same bucket of query, and merge them together for removing duplicate to form a candidate set, and (3) Step S3: Compute the distance between candidates and the query to report near neighbor points. Typically, the cost of S1 is very small and dominated by the cost of S2 and S3, which significantly depend on the distance distribution between the query and data points.

To process Step S2, one typically uses a hash table or a bitvector of n bits to store non-duplicate entries. The cost of such techniques is proportional to the total number of collisions (*#collisions*) encountered in L hash tables, which can be directly computed by simply storing the bucket size. The cost of S3 is clearly proportional to the candidate set size (*candSize*). The total cost of LSH-based search is composed of the cost of S2 and S3, as formalized in Equation (1).

Given α as the average cost of removing a duplicate, and β as the cost of a distance computation, we formalize the

total cost of LSH-based search and linear search as follows:

$$\textbf{LSHCost} = \alpha \cdot \#collisions + \beta \cdot candSize \quad (1)$$

$$\textbf{LinearCost} = \beta \cdot n \quad (2)$$

Given such constants α, β, we can compute exactly **LinearCost**, but we need *candSize* for computing **LSHCost**. By constructing an HLL data structure for each bucket, we can derive the HLL of the candidate set. Therefore, we can accurately approximate *candSize*, and then estimate the **LSHCost**. In turn, we can compare **LinearCost** and **LSHCost** in order to *interchange* LSH-based search with linear search to achieve better performance.

3.2 Hybrid Search Strategy

We construct an HLL for each bucket when building LSH hash tables, as shown in Algorithm 1. Given a query $\boldsymbol{q}$, we view point indexes hashed in the buckets $g_1(\boldsymbol{q}), \cdots, g_L(\boldsymbol{q})$ as L partitions of a data stream. We will estimate the number of distinct elements of such data stream, which is the *candSize* in Equation (1). By estimating **LSHCost** and comparing it to **LinearCost**, we can identify the suitable search strategy, as shown in Algorithm 2.

Algorithm 1 Construct LSH hash tables

Require: A point set S, and L hash functions: $g_1, \ldots, g_L$

```
for each x ∈ S do
    for each hash table T_i using hash function g_i do
        Insert x into the bucket g_i(x)
        Update HyperLogLog of the bucket g_i(x)
    end for
end for
```

Algorithm 2 Hybrid search for r-NN

Require: A query point $\boldsymbol{q}$, and L hash tables: $T_1, \ldots, T_L$

1: Get the size of the buckets $g_1(\boldsymbol{q}), \ldots, g_L(\boldsymbol{q})$ to compute *#collisions*
2: Merge HLLs of the buckets $g_1(\boldsymbol{q}), \ldots, g_L(\boldsymbol{q})$ to estimate *candSize*
3: Estimate **LSHCost** using Equation (1), and compute **LinearCost** using Equation (2)
4: Choose LSH-based search if **LSHCost** < **LinearCost**; otherwise, use linear search

The time complexity analysis. Now, we analyze the complexity of the two algorithms. Algorithm 1 uses a space overhead due to the additional HLLs. For each bucket, an HLL needs $O(m)$ space where m is the number of registers of HLL, which governs the accuracy of the *candSize* estimate. In practice, we only need $m = 32 - 128$. This means that the space overhead of HLLs is usually smaller than large buckets (e.g., $\#points > m$). For small buckets (e.g., $\#points < m$), we might not need HLL, since we can update the merged HLL on demand at the query time. This trick can save the space overhead and improve the running time of the algorithm.

Algorithm 2 is more important since it governs the running time of the algorithm. Compared to the classic LSH-based search, the additional cost of the hybrid search approach is from merging L HLL data structures and estimating *candSize*, which takes $O(mL)$. Such cost is often smaller than (or comparable to) the cost of Step S1, i.e., hash functions computation on LSH families [6, 7, 8, 12]. In other words, the cost overhead caused by our hybrid search approach is little and dominated by the total search cost.

4. EXPERIMENT

We implemented algorithms in Python 3 and conducted experiments on an Intel Xeon Processor E5-1650 v3 with 64GB of RAM. We compared the performance of different search strategies, including hybrid search, LSH-based search, and linear search for reporting near neighbors on several metric distances allowing LSH. We used 4 real-world data sets: Corel Images[2] ($n = 68,040, d = 32$), CoverType[2] ($n = 581,012, d = 54$), Webspam[3] ($n = 350,000, d = 254$), and MNIST[3] ($n = 60,000, d = 780$). For each dataset, we randomly remove 100 points and use it as the query set, and report the average of 5 runs of algorithms on the query set.

For each metric distance, we use the corresponding LSH family. Particularly, we applied SimHash [7] to obtain 64-bit fingerprint vectors for MNIST and use bit sampling LSH [12] for Hamming distance. CoverType and Corel Images use random projection-based LSH [8] for L1 and L2 distances, respectively. Webspam uses SimHash [7] for cosine distance.

4.1 Efficiency of HyperLogLog

This subsection presents experiments to evaluate the efficiency of HLLs on estimating the candidate set size for a given query point. For HLL's parameter, we fix $m = 128$ to achieve a relative error at most 10% as suggested in [9]. For LSH's parameters, we fix $L = 50$ and set $k = \left\lceil \log\left(1 - \delta^{1/L}\right) / \log p_1 \right\rceil$, where $\delta = 10\%$ and p_1 is the collision probability for points within the radius r to the query. This setting is used for SimHash [7] and bit sampling LSH [12]. For random projection-based LSH [8] for L1 and L2 distances, in order to achieve $\delta = 10\%$, we have to adjust $k = 8, w = 4r$ and $k = 7, w = 2r$, respectively, where w is an additional parameter of such LSHs. We note that HLL estimation takes $O(mL)$ time, so this cost is almost constant when fixing m and L.

Table 1: Relative cost and error of HLLs

Dataset	Webspam	CoverType	Corel	MNIST
% Cost	1.31%	0.12%	3.18%	17.54%
% Error	5.99%	5.86%	6.74%	6.8%

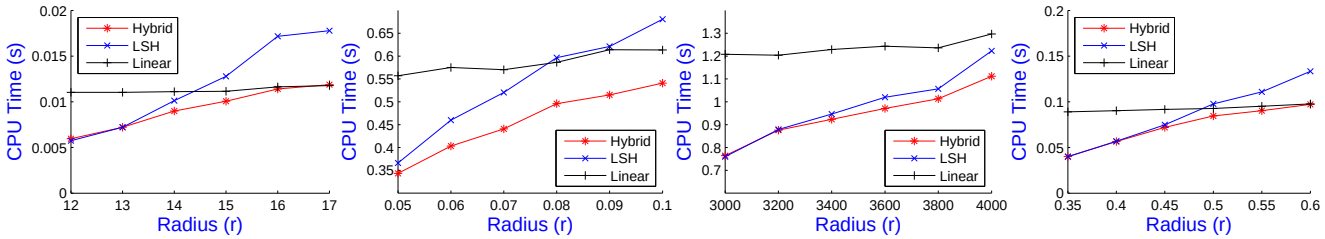
Table 1 shows the average performance of HLL over 4 datasets for a small range of radii where LSH-based search significantly outperforms linear search. It is clear that the cost of HLL is very little, less than 4% of the total cost for the real-value data points. For MNIST, since the distance computation cost is very cheap due to binary representation, the cost of HLL is 17.54% of the total cost. However, since MNIST is very small ($n = 60000$), we can set $m = 32$ to reduce the cost to 4.4% without degrading the performance.

Regarding the accuracy, although theoretical analysis guarantees a relative error of 10%, the practical relative error is even much smaller, less than 7% with standard deviation around 5% for all datasets. The small overhead cost and high accuracy provided by HLL enables us to efficiently estimate the total cost of LSH-based search, see Equation (1), and identify the appropriate search strategy.

4.2 Efficiency of Hybrid Search

This subsection studies the performance of our proposed hybrid search strategy. To compare **LSHCost** and **LinearCost**, we need to identify the ratio β/α, which obviously depends on the implementation, the sparsity of the dataset

[2]https://archive.ics.uci.edu/ml/datasets/
[3]http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/



(a) MNIST with Hamming dist. (b) Webspam with cosine dist. (c) CoverType with L1 dist. (d) Corel with L2 dist.

Figure 2: Comparison of CPU Time (s) for a query set between hybrid search (Hybrid), LSH-based search (LSH), and linear search (Linear) on 4 data sets using different metric distances.

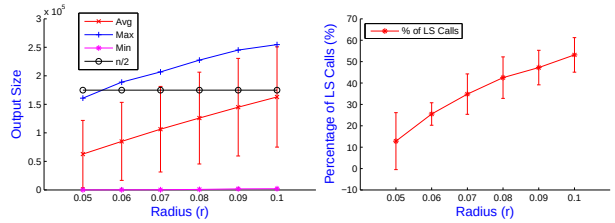


Figure 3: Left: Average, maximum, and minimum output size of queries; Right: Percentage of linear search (LS) calls used in hybrid search for Webspam.

and the used distance metric. We use a random set of 100 queries and 10,000 data points for choosing the ratio β/α as 10, 10, 6, 1 for Webspam, Covertype, Corel, and MNIST, respectively. We use the same setting as the previous section for LSH's and HLL's parameters.

Figure 2 shows the average running time in seconds of the 3 search strategies. For small r, LSH-based search and hybrid search are comparable, but superior to linear search since the output size of each query is rather small. When r increases, hybrid search gains substantial advantages by interchanging LSH-based search with linear search since there are more "hard" queries on the query set. It outperforms LSH-based search and eventually converges to linear search. Specifically, hybrid search provides superior performance compared to both LSH-based search and linear search on Webspam, as shown in Figure 2.b. This is due to the fact that Webspam has several "hard" queries for even very small radii ($r \leq 0.1$).

Figure 3 reveals that the output size varies significantly even with small r. The maximum output size is almost more than half of the point set size ($n/2$) whereas the minimum output size is very tiny. This means that Webspam has many "hard" queries, and therefore hybrid search gives superior average performance. The right figure confirms this observation by showing the average percentage of linear search calls for hybrid search. This amount is at least 10% at $r = 0.05$ and increases to approximate 50% at $r = 0.1$.

We note that hybrid search gives higher recall ratio than LSH-based search since it uses linear search for "hard" queries. Due to the limit of space, we do not report it here.

5. CONCLUSIONS

In this paper, we propose a hybrid search strategy for LSH on rNNR problem in high-dimensional space. By integrating an HyperLogLog data structure for each bucket, we can estimate the total cost of LSH-based search and choose the appropriate search strategy between LSH-based search and linear search to achieve better performance. Our experiments on real-world data sets demonstrate that the proposed approach outperforms (or is comparable to) both LSH-based search and linear search for a wide range of search radii and data distributions in high-dimensional space. We observed that our hybrid search fits well with the multi-probe LSH schemes [2, 13] and the covering LSH [14], which typically require a large number of probes. Applying hybrid search on these LSH schemes for rNNS will be our future work.

6. REFERENCES

[1] S. Abbar, S. Amer-Yahia, P. Indyk, and S. Mahabadi. Real-time recommendation of diverse related articles. In *WWW*, 2013.
[2] T. D. Ahle, M. Aumüller, and R. Pagh. High-dimensional spherical range reporting by output-sensitive multi-probing LSH. SODA'17.
[3] J. Alman and R. Williams. Probabilistic polynomials and hamming nearest neighbors. In *FOCS*, 2015.
[4] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, Jan. 2008.
[5] S. Arya, G. D. da Fonseca, and D. M. Mount. A unified approach to approximate proximity searching. In *ESA*, 2010.
[6] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations (extended abstract). In *STOC*, 1998.
[7] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, 2002.
[8] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, 2004.
[9] Éric Fusy, O. G, and F. Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *AofA*, 2007.
[10] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.
[11] M. R. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *SIGIR*, 2006.
[12] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, 1998.
[13] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *VLDB*, 2007.
[14] R. Pagh. Locality-sensitive hashing without false negatives. In *SODA*, 2016.
[15] F. X. Yu, S. Kumar, Y. Gong, and S. Chang. Circulant binary embedding. In *ICML*, 2014.