I-LSH: I/O efficient c-Approximate Nearest Neighbor Search in High-dimensional Space

Wanqi Liu[†], Hanchen Wang[†], Ying Zhang[†], Wei Wang[§], Lu Qin[†]

[†]*CAI, University of Technology Sydney,* [§]*University of New South Wales*

{wanqi.liu, hanchen-l.wang}@student.uts.edu.au,
{ying.zhang, lu.qin} @uts.edu.au,
{weiw}@cse.unsw.edu.au

***Abstract*—Nearest Neighbor search has been well solved in low-dimensional space, but is challenging in high-dimensional space due to the curse of dimensionality. As a trade-off between efficiency and result accuracy, a variety of c-approximate nearest neighbor (c-ANN) algorithms have been proposed to return a c-approximate NN with confident at least δ. We observe that existing c-ANN search algorithms have some limitations on I/O efficiency when their indexes are resided on the external memory, which is critical for handling large scale high-dimensional data.**

In this paper, we introduce an incremental search based c-ANN search algorithm, named I-LSH. Unlike the previous LSH methods, which expand the bucket width in an exponential way, I-LSH adopts a more natural search strategy to incrementally access the hash values of the objects. We provide rigorous theoretical analysis to underpin our incremental search strategy. Our comprehensive experiment results show that, compared with state-of-the-art I/O efficient c-ANN techniques, our algorithm can achieve much better I/O efficiency under the same theoretical guarantee.

I. Introduction

Given a set of d-dimensional objects (points) and a query object (point), Nearest Neighbor (NN) search finds the object which has the smallest distance to a query object. Due to the "curse of dimensionality" problem, c-ANN was proposed, and has been applied in many domains such as database, computer vision, multimedia, machine learning and recommendation system.

In this paper, we aim to develop an I/O efficient c-ANN search algorithm.

Motivation. Locality sensitive hashing (LSH) [1] is a widely adopted method to support c-ANN search. In addition to theoretical guarantee, it also enjoys great success in practice due to its excellent performance and ease of implementation. Most LSH algorithms are designed for external memory, such as SRS [2] and QALSH [3], while both of them are not I/O efficient.

C2LSH and QALSH adopt the *bucket exponential expansion* strategy where the bucket widths must be a power of c (i.e., bucket width grows exponentially), which may lead to some counter-intuitive scenarios as shown in Fig. 1(a) and (b). In this paper, we follow the framework of QALSH to enjoy the high efficient sequential I/O brought by the B+ tree. In addition, we provide rigorous analysis to show that we can conduct the natural incremental search strategy as shown in Fig. 1(c) on each projected dimension and identify c-ANN object with desired confidence. By doing this, we show in Section IV

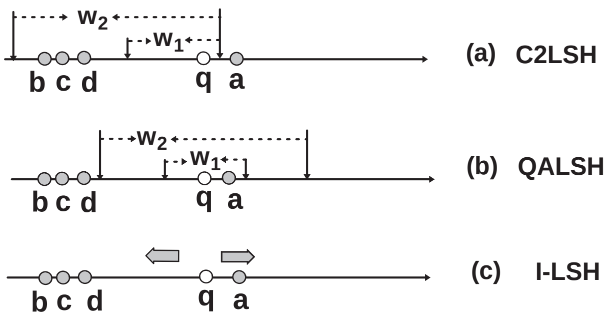


Fig. 1. Motivation for I-LSH

that our proposed algorithm, namely I-LSH, can significantly improve the performance by using approximate ratio c in our search algorithm, instead of $\sqrt{c}$ used in C2LSH and QALSH. Our experiments on real-life datasets demonstrate that I-LSH can achieve the same theoretical guarantee with much less I/O costs.

Contributions. Our principal contributions are summarized as follows.

- We propose a new c-approximate nearest neighbor search algorithm, namely I-LSH, for high-dimensional data, which uses a natural incremental search strategy on the projected dimensions.
- We provide rigorous analysis to demonstrate the correctness and efficiency of our proposed methods.
- We perform an extensive performance evaluation against two state-of-the-art I/O efficient c-ANN algorithms regarding I/O costs and result accuracy. The results demonstrate that our proposed methods can achieve the best I/O performance under the same theoretical guarantee.

II. Preliminaries

In this section, we present the problem definition and relevant existing work. Some important notations used throughout the paper are summarized below.

We use n to denote the number of data objects in the d-dimensional dataset. q denotes the query object. m is the number of hash functions to project the original dataset. $\parallel o_1, o_2 \parallel$ denotes the Euclidean distance between two objects o_1 and o_2. o_{min} means the current found nearest neighbor to q. $h_{\vec{a}}(o)$ denotes the hash value of point o using the vector $\vec{a}$ and $d_i(o)$ denotes the projected distance w.r.t i-th hash function with $d_i(o) = |h_i(o) - h_i(q)|$. c is the approximate ratio. w is the initial bucket width. r and $R = \frac{2r}{w}$ denote the search radius on projected dimensions and the radius of ball $B(q, R)$ in $\mathcal{R}^d$ space regarding search radius r respectively.

A. Problem Definition

Given a dataset D with n points in a d-dimensional space, denoted by $\mathcal{R}^d$, the coordinate value of each object o on the i_{th} dimension is denoted as $o[i]$. The c-approximate nearest neighbor is defined as follows:

Definition 1. ***c-approximate nearest neighbor (c-ANN).*** *For a given query object q and a d-dimensional dataset $\mathcal{D}$, suppose o^* is the nearest neighbor of q with distance R^*, a c-approximate nearest neighbor of q is a data object $o \in \mathcal{D}$ such that $\| o, q \| \leq cR^*$ where c is the approximate ratio.*

Problem statement. In this paper, we aim to propose an efficient algorithm to solve the c-ANN problem in d-dimensional Euclidean space with theoretical guarantee; that is, given the approximate ratio c and the probabilistic threshold δ, the algorithm should return the c-approximate nearest neighbor (c-ANN) with probability at least δ.

B. Existing LSH approaches

Locality sensitive hashing (LSH) was first introduced by Indyk *e*t al. in 1998 [1] and was extended by Datar *e*t al. [4] to Euclidean space. A LSH function can maintain the distance relationships between data objects, which means for two points o_1 and o_2, if $\| o_1, o_2 \|$ is a small value, then $\| h(o_1), h(o_2) \|$ is likely to be a small value as well.

LSH was designed for (R, c)-NN queries, which is simply a decision version of the c-ANN problem. To solve c-ANN problem, virtual rehashing was proposed [5] [6] [3].

QALSH [6] first proposed a query-aware LSH functions, which can be formally represented by $H_{\vec{a}}(o) = \vec{a} \cdot \vec{o}$. In our algorithm, we use the same hash function to project the d-dimensional dataset. It is based on a set of B-trees, which is I/O efficient. But QALSH is a c^2-approximate algorithm, which means it requires a large number of hash functions. SRS uses a multi-dimensional index R+ tree to solve the problem [2]. It can keep the theoretical guarantee with a tiny size of index. However, due to the property of R+ tree, it doesn't work well when $m > 5$.

There are also some I/O efficient ANN search algorithms proposed in the literature without theoretical guarantee. For instance, Liu *et al* proposed SK-LSH [7] for approximate NN problem as an improvement of LSB-tree, which used linear order instead of Z-order for better I/O efficiency. In [8], I/O efficient algorithm was proposed based on PQ method [9].

III. Our Approach

In this section, we present our incremental LSH technique which is I/O efficient with rigorous theoretical guarantee. Section III-A describes our LSH algorithm for c-ANN problem and Section III-B shows that our approach can be immediately extended to support top k c-approximate nearest neighbor search (c-k-ANN).

A. Incremental LSH

In this subsection, we describe our incremental LSH (I-LSH) approach in details. Our technique consists of two parts: *indexing* and *query processing*.

Indexing
The index part is similar to QALSH. For a given d-dimensional data D with n objects, we use m 2-stable hash functions to randomly project each object o into m hash values, denoted by $h_i(o = \vec{a}_i \cdot \vec{o}$ for the i-th hash function. For the i-th random projection, we use a B+ tree to keep the pair $(ID(o), h_i(o))$ for each object, where $ID(o)$ is the ID of the object o and the hash value $h_i(o)$ is the search key.

Algorithm 1: **Incremental LSH search($\mathcal{B}$, q)**

```
Input  : $\mathcal{B}$: $m$ B+ tree indices for object IDs and hash values;
         $q$: the query object;
         $w$: the initial bucket width;
Output : $o$: the $c$-ANN object
1  $n_{can} := 0$; $d_{min} := 0$;
2  Apply $m$ hash functions on $q$;
3  while $n_{can} < \beta n$ do
4      $o \leftarrow$ next object with smallest projected distance;
5      $i \leftarrow$ the projection dimension $o$ comes from;
6      $r \leftarrow |h_i(o) - h_i(q)|$;
7      $R \leftarrow \frac{2r}{w}$;
8      $cn(o) := cn(o) + 1$;
9      if $cn(o) == \alpha m$ then
10         compute $\| o, q \|$ and update $o_{min}$;
11         $n_{can} := n_{can} + 1$;
12     if $R_{min} \leq cR$ then
13         break;
14 return $o_{min}$
```

Query processing.
In general, a query object q in the d-dimensional space will be mapped into m projected dimensions, then the objects and their hash values will be incrementally accessed according to their projected distances in m projected dimensions. An object becomes a candidate if αm of its hash values have been accessed. There are two termination conditions: C_1: if there are βn candidate objects found by I-LSH, the algorithm will stop, C_2: if the current nearest object o_{min} satisfies $\| o_{min}, q \| \leq cR$ where R is the current radius, the algorithm will stop. Note that α, β and w are pre-defined parameters, and their settings will be discussed in Section IV.

Algorithm 1 presents the pseudo-code of our incremental LSH technique. In each iteration, the algorithm chooses the data point o with smallest projected distance to q as the current point (line 4), and compute the corresponding radius R in line 7. If o has been visited for αm times, the algorithm will compute its Euclidean distance to q and add it to the candidate set (line 9 to line 11). If either of the two termination conditions is satisfied, the algorithm will return the current nearest point as the c-ANN.

B. Extension for c-k-ANN problem

In many real-life applications, in addition to the nearest neighbor, users are interested in the k nearest neighbors. In this paper we also study the problem of c-approximate k nearest neighbor (**c-k-ANN**) search. We say an object o is a correct result of of c-k-ANN search if $\mid o, q \mid \leq c \mid o_k, q \mid$, where o_k is the k-th nearest neighbor of q in the objects $\mathcal{D}$.

Algorithm 1 can be easily extended to solve the c-k-ANN problem by the following changes: (1) instead of βn, we need to access $\beta n + k - 1$ candidate objects; (2) instead of o_{min}, we maintain the k-th closest candidate object o_k and (3) the k most closest candidate objects will be return as the result of c-kANN search.

IV. Theoretical Analysis

In this section, we show the correctness of our incremental LSH method; that is, for any given query q, approximate ratio c ($c > 1$) and success probability δ ($0 \leq \delta \leq 1$), our method can return the c-ANN with probability at least δ.

A. Correctness of I-LSH

There are two steps to show the correctness of I-LSH: (1) For the (R, c)-NN problem, our scheme holds the guarantee. (2) The expanding of r won't break the guarantee. Before the formal proof, we stress some important notations frequently used. By r, we denote the current search radius in the projected dimensions, which corresponds to the anchored bucket with width $2r$ and a ball $B(q, R)$ in the high-dimensional space $\mathcal{R}^d$, where $R = \frac{2r}{w}$ and w is the initial bucket width.

(R, c)-NN correctness

Suppose there is a d-dimensional dataset D with n points and a query point q, the radius is R. If the theoretical guarante holds, we hope that,

- $P1$:for data object o,if $\parallel o, q \parallel R$, o can be found as a candidate.
- $P2$:the total number of false positives won't be greater than βn, the false positive means that for a point o, if $\parallel o, q \parallel cR$ but o is found as a candidate.

According to QALSH, if we set $\delta = 1/2 - \delta'$ (So when $\delta = 1/2 - 1/e, \delta' = 1/e$), $\beta = 0.01$ and $\alpha = \frac{\sqrt{\frac{ln \frac{2}{\beta}}{ln \frac{1}{\delta'}}} p_1 + p_2}{1 + \sqrt{\frac{ln \frac{2}{\beta}}{ln \frac{1}{\delta'}}}}$, where $p_1 < \alpha < p_2$, and $m = \lceil \frac{ln \frac{1}{\delta'}}{2(p_1 - p_2)^2} (1 + \sqrt{\frac{ln \frac{2}{\beta}}{ln \frac{1}{\delta'}}})^2 \rceil$, I-LSH can return a point o within $B(q, cR)$ with a probability which is at least δ.

c-ANN correctness

In this paper, our LSH function is query-aware because we enforce that the bucket, namely *anchored bucket*, is always centered at $h(q)$, in the projected dimension. We say the hash function h is (r_1,r_2,p_1,p_2)-sensitive with regarding to the bucket width w if we have $Pr(|h(o) - h(q)| \leq \frac{w}{2}) \geq p_1$ given $\parallel o, q \parallel < r_1$ and $Pr(|h(o) - h(q)| \leq \frac{w}{2}) \leq p_2$ given $\parallel o, q \parallel > r_2$. The following lemma shows that a (1,c,p_1,p_2)-sensitive hash function with bucket width w can become (ξ, $c\xi$, p_1, p_2)-sensitive if the bucket width is set to ξw.

Lemma 1. *Given a hash function h, if it is (1,c,p_1,p_2)-sensitive w.r.t the bucket width w, then it is (ξ, $c\xi$, p_1, p_2) sensitive if the bucket width is set to ξw for any real value $\xi > 0$.*

Proof. According to the definition of (r_1, r_2, p_1, p_2)-sensitive hash function, for the bucket width w, we have

$p_1 = \eta(1, w) = \int_{-\frac{w}{2}}^{\frac{w}{2}} \phi(x)dx$, and $p_2 = \eta(c, w) = \int_{-\frac{w}{2c}}^{\frac{w}{2c}} \phi(x)dx$.

Then for the bucket width ξw, we have

$\eta(\xi, \xi w) = \int_{\frac{\xi w}{2\xi}}^{\frac{\xi w}{2\xi}} \phi(x)dx = \int_{-\frac{w}{2}}^{\frac{w}{2}} \phi(x)dx = p_1$, $\eta(c\xi, \xi w) = \int_{\frac{\xi w}{2c\xi}}^{\frac{\xi w}{2c\xi}} \phi(x)dx = \int_{-\frac{w}{2c}}^{\frac{w}{2c}} \phi(x)dx = p_2$

Therefore, the function h is $(\xi, c\xi, p_1, p_2)$-sensitive with bucket width ξw for any $\xi > 0$. □

According to Lemma 1, it is immediate that for any given bucket width ξw, it corresponds to a ball $B(q, R)$ in $\mathcal{R}^d$ centered by q with $R = \frac{\xi}{2}$ space such that for any object $o \in B(q, R)$, $h(o)$ will fall in the anchored bucket (i.e., $|h(o) - h(q)| \leq \frac{\xi}{2}$) with probability at least p_1, while for any object $o \notin B(q, cR)$, it will collide with q with probability less than p_2.

Let r denote the current search radius in the projected dimensions (i.e., the width of the corresponding anchored bucket is $2r$). As ξ can be any non-negative real value in Lemma 1, the search radius r can be any real value with $r > 0$. And the hash function is (ξ, $c\xi$, p_1, p_2)-sensitive where $\xi = \frac{2r}{w}$.

B. Approximate ratio

Unlike the most of the existing LSH methods such as QALSH and C2LSH, the approximate ratio of I-LSH is c but not c^2.

Lemma 2. *Given a query point q, suppose the NN of q is o^*, $\parallel q, o^* \parallel = R^*$. If I-LSH stops at $B(q, R)$, where r is the corresponding projection radius, $R \leq R^*$ when both P_1 and P_2 are satisfied.*

Proof. Because P_1 is satisfied, o^* must can be found as a candidate if all the data objects within $B(q, R^*)$ have been checked. (1) If there are more than βn candidates in $B(q, R^*)$, then I-LSH will be terminated by C_2 before R^* has been reached, so $R \leq R^*$. (2) Otherwise, because $\parallel o^*, q \parallel = R^* < cR^*$, when o^* is found as a candidate, I-LSH will be terminated by C_1. According to P_1, we get $R \leq R^*$. □

Section IV-A has proved that I-LSH can return a c-approximate result for (R, c)-NN problem. According to lemma 2, I-LSH will stop before or at R^* with a constant possibility δ. So the approximate ratio of I-LSH is c.

V. Evaluation

In this section, we conduct comprehensive experiments to demonstrate the I/O efficiency of our proposed algorithm, compared with two state-of-the-art I/O efficient c-ANN algorithms.

A. Experiment Setup

In this subsection, we present the experiment settings of our performance evaluation.

I-LSH is a LSH approach with theoretical guarantee, so we only compare with QALSH and SRS, two of the recent LSH with theoretical guarantee as well. We evaluate the performance on the c-k-ANN version of these algorithms in the experiments where k varies from 1 to 100, with default value 40.

Datasets.

- **`Tiny`** contains around 5 million GIST feature vectors with dimensionality 384.
- **`Million Song`** is a collection of audio features and meta-data for a million contemporary popular music tracks with 420 dimensions.

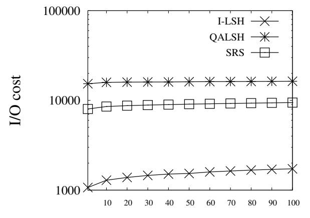
Evaluation Metrics. We count the number of I/Os during the computation for three algorithms, where each random I/O read counts one and each sequential I/O read contributes 0.1 considering the difference cost between random and sequential I/O.

Parameter Setting. The default approximate ratio c is set to 4. Note that QALSH is c^2 approximate method and hence need to set c to $\sqrt{4.0} = 2$ to achieve the 4-approximation, while both SRS and I-LSH are c-approximate methods. We use the default settings of QALSH and SRS unless otherwise specified. The parameter of I-LSH is given in section IV

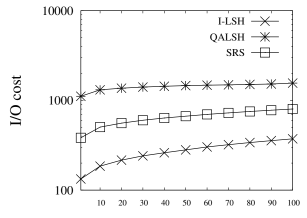
The page size B is set to $8,192$ bytes for all the datasets and algorithms.

B. Evaluate I/O cost

For a c-k-NN search, k is varying from 1 to 100 in the experiments. For 4-NN search, I-LSH has the best I/O performance on all the four datasets. QALSH is not competitive compared with SRS and I-LSH mainly because it has to set $c = \sqrt{c}$ to achieve the same approximate ratio with SRS and I-LSH, which causes a much larger m. SRS has the smallest index size, but because of the penalty of random I/O, SRS requires more I/O costs to find enough candidates. On Tiny and Million song datasets with dimensionality 384 and 420, respectively, I-LSH outperforms SRS by a big margin.



(a) Tiny

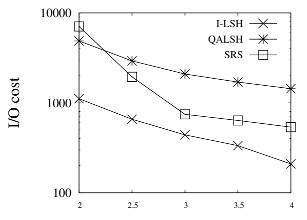


(b) Million Song

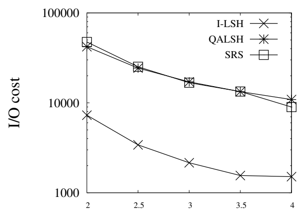
Fig. 2. I/O cost vaying k

C. Effect of the approximate ratio c

A good c-ANN algorithm should support different approximate ratio, especially when c is a small value. In the experiments, c is varying from 2.0 to 4.0. The results are reported in Fig 8. For instance, when $c = 4$, SRS and I-LSH only use about 50% of I/O than QALSH, but the I/O cost of SRS rises very fast, when $c = 2$, it becomes 140% of QALSH and about 7 times of I-LSH. As expected, the performance of three algorithm degrades against the decrease of the approximate ratio. It is shown that the performance of SRS is most sensitive to the ratio. This is because the higher



(a) million song (diff. c)



(b) tiny (diff. c)

Fig. 3. I/O vs c

requirement of accuracy leads to larger m, and will seriously hurt the performance of R-tree in SRS.

D. Summary

Based on the experimental results, we have the following observations:

- Under the same theoretical guarantee, I/O performance of I-LSH consistently outperforms QALSH and SRS under all settings. This is because: (1) I-LSH adopts a natural incremental search strategy, which can achieve c-approximate estimation (unlike the c^2-approximation of QALSH); (2) I-LSH can take advantage of efficient sequential I/O brought by the B+ tree.
- SRS has a good performance when $c = 4$, but it cannot comfortably support smaller c (higher accuracy) because of the curse of dimensionality for the exact NN on multi-dimensional index including R-tree. Moreover, the random I/O caused by R-tree is also a limit for SRS.

Acknowledgments

Ying Zhang is supported by FT170100128 and DP180103096, Lu Qin is supported by DP160101513, Wei Wang is supported by ARC DPs 170103710 and 180103411, and D2DCRC DC25002 and DC25003.

References

[1] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, 1998, pp. 604–613.

[2] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin, "Srs: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index," *Proceedings of the VLDB Endowment*, vol. 8, no. 1, pp. 1–12, 2014.

[3] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng, "Query-aware locality-sensitive hashing for approximate nearest neighbor search," *Proceedings of the VLDB Endowment*, vol. 9, no. 1, pp. 1–12, 2015.

[4] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 2004, pp. 253–262.

[5] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Quality and efficiency in high dimensional nearest neighbor search," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 563–576.

[6] J. Gan, J. Feng, Q. Fang, and W. Ng, "Locality-sensitive hashing scheme based on dynamic collision counting," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 541–552.

[7] Y. Liu, J. Cui, Z. Huang, H. Li, and H. T. Shen, "Sk-lsh: an efficient index structure for approximate nearest neighbor search," *Proceedings of the VLDB Endowment*, vol. 7, no. 9, pp. 745–756, 2014.

[8] Y. Liu, H. Cheng, and J. Cui, "PQBF: i/o-efficient approximate nearest neighbor search by product quantization," in *CIKM*, 2017, pp. 667–676.

[9] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, 2011.