UNIVERSITY OF TECHNOLOGY SYDNEY
Faculty of Engineering and Information Technology

Efficient Query Processing and Analytics on High Dimensional Data

by

Wanqi Liu

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Sydney, Australia

2020

Certificate of Authorship/Originality

I, Wanqi Liu declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise reference or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Candidate signature

Production Note:
Signature removed prior to publication.

Date: 28/03/2020

ABSTRACT

Efficient Query Processing and Analytics on High Dimensional Data

by
Wanqi Liu

As a fundamental problem in query processing, similarity search has been applied in many fields including multimedia, machine learning, database, recommendation systems and so on. Generally, it will be challengeable when it comes to the high-dimensional space due to "the curse of dimensionality". Since it would be too expensive to find exact results, approximate solutions have been studied in many papers. There are various distance metrics to evaluate the similarity between the query object and other points in a dataset. In this thesis, we focus on some well-known distance metrics including Euclidean distance and inner product. Except for Euclidean space, we also study query processing on graphs and propose a novel distance metric on graph. The thesis contains four similarity search problems regarding to different distance metrics, which are Approximate Nearest Neighbour, Approximate Inner Product Search, Approximate Furthest Neighbour and Skyline Nearest Neighbour. Given a query point, the four problems all focus on retrieving a set of "similar" points from the dataset.

Given a set of d-dimensional data points, and a query point q, Approximate Nearest Neighbour Search (ANNS) aims to find the approximate closest object to q in the set. More specifically, in this thesis, we focus on c-ANNS problem, which means given a constant c, the purpose is to find a result whose distance is not larger than c times of the exact smallest distance with a certain possibility. Even though this problem has been researched for a long time, there are still some shortage of current algorithms. We studied the existed works, and proposed a novel I/O efficient algorithm to solve c-approximate nearest neighbour problem in external memory, which can dramatically reduce I/O cost and provide rigorous proof of its correctness.

Maximum Inner Product Search (MIPS) is another valuable problem. It returns an object with maximum inner product value to query point q. There are hundreds of solutions for MIPS but still short of comprehensive evaluation and analysis of these methods' performance. In this thesis, we chose several state-of-the-art algorithms of MIPS using different techniques, and conducted a set of comprehensive experiments to evaluate their performance fairly.

Approximate Furthest Neighbour Search is an opposite problem of Nearest Neigh-

bour Search. It finds the furthest object to query point q in a dataset instead of the closest one. Since most recent works for approximate furthest neighbour search in external memory are only suitable for low-dimensional data, we proposed a new I/O efficient technique to achieve a better performance on I/O cost.

In addition to Euclidian space, similarity search is also a fundamental problem in other spaces like graphs. Considering real-world applications, the multi-layer graph model is extensively studied to reveal the multi-dimensional relations between the graph entities. In this thesis, we formulated a new problem called skyline nearest neighbor search on multi-layer graphs, and proposed a baseline algorithm, and two optimizations instead of naively adopting the traditional skyline procedure as a subroutine. We also investigated the rule to optimize search order in the algorithm so that further improve the algorithmic efficiency.

Acknowledgements

Firstly, I wish to express my deepest gratitude to my supervisor Prof. Ying Zhang who is professional, diligent and patient, for offering the opportunity to study at UTS, and for the continuous support, patient guidance and enthusiastic encouragement he has provided through the three years. Discussions with him always enlighten me to improve my works in this thesis. He also helped me to extend my knowledge and gave me many inspirations. It would never have been possible for me to complete this work without his incredible support and valuable ideas.

Secondly, I wish to express my sincere appreciation to Prof. Wei Wang for his advice, and for his brilliant ideas which inspired me deeply. Additionally, his valuable comments from his abundant research experience helped me on my research work. Acknowledgement also goes to Prof. Xumin Lin and my co-supervisor Dr. Lu Qin for their supporting and guidance.

I am also sincerely grateful to Mr.Hanchen Wang as most of the works in this thesis are conducted collaborating with him. I really appreciate for his patient to assist me in proofreading and revising the drafts. He is always able to find out grammar mistakes which I missed. My sincere thanks also goes to Dr. Dong Wen who shows advanced research skills and helps me to refine my work.

I would also like to thank Mr. Kai Wang, Mr. Mingjie Li, Mr. Bohua Yang, Ms. Conggai Li, Mr. Boge Liu, Mr. Wentao Li, Dr. Yang Yang, Dr. Haida Zhang, Dr. Fan Zhang, Dr. Ouyang Dian, Mr. Junda Lu, Mr. Yufei Wang for giving me a pleasant time when working with them.

Last but not least, I would like to express my great appreciation to my parents for raising me with care and love, providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis, and to my grandparents for their understanding and love. Thanks to my boyfriend Chenhao Pan for his love, understanding and continuing support to complete the research work.

Wanqi Liu
Sydney, Australia, March 2020.

List of Publications

- **Wanqi Liu**, Hanchen Wang, Ying Zhang, Wei Wang and Lu Qin, I-LSH: I/O efficient c-Approximate Nearest Neighbor Search in High-dimensional Space, published in ICDE 2019
- **Wanqi Liu**, Dong Wen, Hanchen Wang, Fan Zhang and Xubo Wang, Skyline Nearest Neighbor Search on Multi-Layer Graphs, published in ICDE 2019 workshop
- **Wanqi Liu**, Hanchen Wang, Ying Zhang, Luqin and Wenjie Zhang, I/O efficient algorithm for c-approximate furthest neighbor search in high-dimensional space, will appear in DASFAA 2020
- **Wanqi Liu**, Hanchen Wang, Ying Zhang, Wei Wang, Lu Qin and Xuemin Lin, EI-LSH: An Early-Termination Driven I/O Efficient Incremental c-Approximate Nearest Neighbor Search, Received by VLDBJ

Contents

List of Figures

Abbreviation

NN: Nearest Neighbour
ANN: Approximate Nearest Neighbour
c-ANN: c-Approximate Nearest Neighbour
MIPS: Maximum Inner Product Search
FN: Furthest Neighbour
AFN: Approximate Furthest Neighbour
c-AFN: c-Approximate Furthest Neighbour
ET: Early-Termination
NT: Normal-Termination
SNNS: Skyline Nearest Neighbour Search

Chapter 1

Introduction

In this chapter, we briefly introduce the research in this thesis, including the research backgrounds, motivations and our contributions.

Similarity search is one of the significant problems to support query processing. Given a query, similarity search aims to find data points in a dataset which are similar to the query, while the definitions of similarity are varying. Many similarity search problems become much more complex and difficult when they are in the high-dimensional space due to the curse of dimensionality, which was first mentioned by Richard E. Bellman [9]. Curse of dimensionality refers to some phenomenas which only appear when analysing or processing high-dimensional data. Space increases exponentially as dimensionality grows, and the growth will cost more unnecessary storage space and time for processing queries. As a trade-off between result quality and time consumption, an acceptable method in high-dimensional space is to retrieve approximate result instead of exact result. In this thesis, we focus on similarity search problem regarding various distances. We studied several representative similarity search problems, including nearest neighbour search, maximum inner product search, furthest neighbour search in high-dimensional space, and proposed approximate algorithms to improve performance, and defined a new distance metric in multi-layer graphs, coming with novel solutions for it.

Euclidean distance is one of the most popular distance metrics used in similarity search, and it has been widely used in Nearest and Furthest Neighbour search problem. In this thesis, we proposed novel I/O efficient algorithms for Approximate

Nearest Neighbour Search (ANNS) problem and Approximate Furthest Neighbour Search (AFNS) problem, by adopting a more reasonable search strategy and a more aggressive termination condition. Maximum inner product is another important distance metric. We notice that although this problem has been well studied, there is a few works which have provided a fair and comprehensive evaluation for existed algorithms. Therefore we conducted a set of comprehensive experiments for Approximate Maximum Inner Product (AMIP) search problem to fairly evaluate state-of-the-art MIPS algorithms from different domains and analysed the result carefully and objectively. We also studied similarity search on multi-layer graphs which can be considered as another multi-dimensional space. We defined a new problem called Skyline Nearest Neighbour Search which is applied on multi-layer graphs, and proposed an efficient algorithm to process such queries.

1.1 Background

1.1.1 Approximate Nearest Neighbour Search

Nearest Neighbour (NN) problem is one of the fundamental similarity search problems in similarity seach, and the common measures of distance in NN are Hamming distance and Euclidean distance. In this thesis, we primarily focus on Euclidean distance.

Given a set of d-dimensional objects and a query object q, Nearest Neighbour (NN) search finds the object which has the smallest distance to a query object q. This problem has a various applications in many fields such as database, computer vision and machine learning. In these applications, each object is usually represented by a point with high-dimensionality. In low-dimensional space, NN is not a complex problem and has been well-solved. But in high-dimensional space, it becomes more tricky due to the curse of dimensionality. Since it is too expensive to find the exact NN point, Approximate Nearest Neighbour Search can be performed

efficiently and are sufficiently useful for many practical problems, thus attracting an enormous number of research efforts. There are two categories of approximate NN search algorithms: (1) c-approximate NN (c-ANN) search algorithms, which have (c, δ)-approximate theoretical guarantee, and (2) approximate NNS algorithms without such guarantee. Our research focuses on c-ANN search. Considering the huge volume of data due to the large number of objects and high dimensionality, we aim to develop an I/O efficient algorithm based on external memory to handle the large scale data which cannot be fitted in the main memory. Locality sensitive hashing (LSH) [39] 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.

1.1.2 Approximate Maximum Inner Product Search

Inner product is another widely-used distance in similarity search, and Maximum Inner Product Search (MIPS) is an important and challenging problem which has been widely used in many domains such as database, recommendation system and machine learning. Since the real-world datasets are generally in a high-dimensional space with substantial size, computing accurate MIPS result could suffer from "curse of dimensionality". To get the result in an acceptable time, Approximate Maximum Inner Product Search (Approximate MIPS) has been proposed and applied in most cases. In this thesis, we mainly focus on the existed Approximate MIPS algorithms.

In many applications, data objects can be represented as points. Given a d-dimensional dataset D and query point q, MIPS aims to find the point $o^* \in D$ which has maximum inner product with q:

$$o^* = arg\ max \sum_{i=1}^{d} o_i \cdot q_i$$

where o_i and q_i denotes the i_{th} coordinate value of point o and query q. In practice, it is generally required to return top-k objects for a given query point, so all of the

algorithms we chose in this thesis can be easily extended to the top-k version.

1.1.3 Approximate Furthest Neighbour Search

Furthest neighbour search is a logical opposite of NN we just mentioned above. Given a set of d-dimensional objects and a query object, similar to Nearest Neighbor search, the Furthest Neighbor search aims to find an object which has the longest distance to the query point. It has been widely applied in many domains such as recommendation systems to increase the diversity of the recommendation [76] [75]. c-AFN usually use same distance metrics with c-ANN, thus the difficulty caused by high-dimensionality also exists. There are several existing solutions for furthest neighbour search in low-dimensional space. It becomes very expensive to find the exact furthest neighbour for a given query object. To solve this problem, an approximate FN is an acceptable solution as a trade-off between efficiency and accuracy. In this thesis, we focus on solving approximate furthest neighbor problem with theoretical guarantee, which is also called c-AFN problem. Given an approximation ratio c ($c > 1$) and a success possibility δ, a c-AFN query returns a c-approximate furthest neighbor with confidence at least δ. Similar to c-AFN problem, a huge dataset is also hard to be fitted in memory, so we also studied external memory algorithms.

1.1.4 Skyline Nearest Neighbour Search

To study they similarity search problem in other spaces other than Euclidean space, we propose the skyline nearest neighbour search problem on graphs.

Unlike the previous problems, Skyline Nearest Neighbour Search is from another aspect to work on similarity search. We have dug into NN problem in high-dimensional space, while in many real-world scenarios, there are usually multiple relationships between objects. For example, in a social network application such as twitter, there are several types of relationships between users: following, reply and

retweet. The multi-layer graph is frequently used to model such multi-dimensional dataset, and has been well studied [12] [73]. However, there is no definition of NN on a multi-layer graph.

Skyline nearest neighbour is a new problem defined by us to solve NN problem on multi-layer graphs. For a query vertex, the NN on each layer can be totally different. To solve such multi-criteria decision problem, skyline is a classical model which can eliminate all the objects which are worse than some other on all the categories. In a multi-layer graph, a skyline nearest neighbor search can eliminate the vertices whose shortest distances to the query vertex on all the layers are larger than some other vertices. For example, in figure 1.1, the distance between v_0 and v_3 on two layers are both 2, and the distance between v_0 and v_2 on two layers are 2 and 3. Since v_2 has a same distance to v_0 on layer 1 with v_3 and a longer distance on layer 2, v_2 should not be considered as a skyline nearest neighbor candidate. Skyline nearest neighbor search returns a set of vertices which are not dominated by any other vertex in the multi-layer graph.

1.2 Motivations

1.2.1 Approximate Nearest Neighbour Search

ANNS problem is one of the typical similarity search problems. It aims to find top-1 or top-k nearest points for a given query in a dataset approximately. The accuracy of most approximate algorithms are dependent on datasets, which means theoretically, an approximate algorithm could return a set of candidates which are not even close to the real NN points if the data distribution is not friendly to this algorithm. So we are more interested in developing c-ANN algorithms, which can at least provide a guarantee for its result accuracy. LSH is the most popular scheme for c-ANN search and there are various of LSH-based c-ANN algorithms which have already been published. However, the recent I/O efficient c-ANN algorithms either

suffered from a large approximation ratio problem or a non-efficient index structure. Motivated by this, In the thesis, 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. The theoretical will always hold regardless of the data distribution.

1.2.2 Approximate Maximum Inner Product Search

Despite Euclidean distance, inner product is also an extensively used distance metric in many applications such as multi-class label prediction [23, 41], matrix factorization [51, 97] and computer vision [27]. Similar to ANNS, Approximate MIPS returns a set of points which have the largest inner product with a query point approximately. This problem has attracted increasing attention and hundreds of related works have been published, but there are only a few comprehensive comparisons among these algorithms. To get an objective and overall evaluation to these existed algorithms, we conduct a set of experiments to evaluate the state-of-the-art approximate maximum inner product search algorithms considering of the following needs:

- **Overall evaluation metrics and settings.** A MIPS algorithm can be measured from a wide variety of aspects: time complexity, space complexity, recall, ratio, precision, index size, index time, scalability and so on. In most papers, due to the limitation of number of pages, they only choose several metrics and parameter settings to conduct experiments. It is common that different papers select different evaluation metrics in their own experiments. For example, ratio and recall can both be used to evaluate result accuracy, but it is difficult to compare the performance between two algorithms straightforward when they are using ratio and recall respectively.

In this thesis we evaluate the algorithms adopting a wide range of settings and metrics on 8 datasets to get a complete understanding of them.

- **Cover algorithms from different domains.** As the MIPS problem has been applied in various fields, there are multiple popular techniques to solve it such as hashing, graph, product quantization and so on. When a new algorithm is proposed, the author usually only compares with the existed papers in the same domain or using similar techniques but ignore alternative algorithms in other fields. For example, KNN-Graph is a well-known model which supports diverse distance metrics including maximum inner product. ip-nsw+, a state-of-the-art Approximate MIPS algorithm based on KNN-Graph which was published on AAAI 2020 [54], only considered other graph-based Approximate MIPS algorithm in the experiments but neglected algorithms adopting other techniques.

1.2.3 Approximate Furthest Neighbour Search

As an opposite problem of ANNS, a general solution for c-AFN is also LSH and its variants since it has a good property that the close objects in the original space are likely to be close as well after projection. We notice that most of the LSH based c-AFN algorithms adopt exponentially reducing searching strategy, which could involve a large number of points at each step. Besides, the current c-AFN algorithms require a sizeable index before processing queries, which could be not acceptable when there is no enough CPU and memory resources.

Therefore, we propose an I/O efficient c-AFN algorithm following the framework of a state-of-the-art c-AFN algorithm called RQALSH but improving the searching strategy to reduce I/O cost and index size. The theoretical guarantee for c-AFN is still held: given a approximate ratio $c < 1$, and a probabilistic threshold δ, our algorithm would return a c-approximate furthest neighbour of query q with a

possibility at least δ. It offers a lower bound of the result quality regardless of datasets.

1.2.4 Skyline Nearest Neighbour Search

In many real-world scenarios, there are usually multiple relationships between objects. For example, in a social network application such as twitter, there are several types of relationships between users: following, reply and retweet. If we hope to get some information about a given user, Euclidean space cannot well represent such relationships, so it is necessary to consider similarity search problem in other spaces such as graph. Multi-layer graph is frequently used to model such multi-dimensional dataset, and has been well studied [12] [73]. However, there is no definition of NN on a multi-layer graph.

Skyline nearest neighbour is a new problem defined by us to solve NN problem on multi-layer graphs. For a query vertex, the NN on each layer can be totally different. To solve such multi-criteria decision problem, skyline is a classical model which can eliminate all the objects which are worse than some other on all the categories. In a multi-layer graph, a skyline nearest neighbor search can eliminate the vertices whose shortest distances to the query vertex on all the layers are larger than some other vertices. For example, in figure 1.1, the distances between v_0 and v_3 on two layers are both 2, and the distances between v_0 and v_2 on two layers are 2 and 3. Since v_2 has a same distance to v_0 on layer 1 with v_3 and a longer distance on layer 2, v_2 should not be considered as a skyline nearest neighbor candidate. Skyline nearest neighbor search returns a set of vertices which are not dominated by any other vertex in the multi-layer graph.

1.3 Contibutions

In this thesis, our contributions are described as follows.

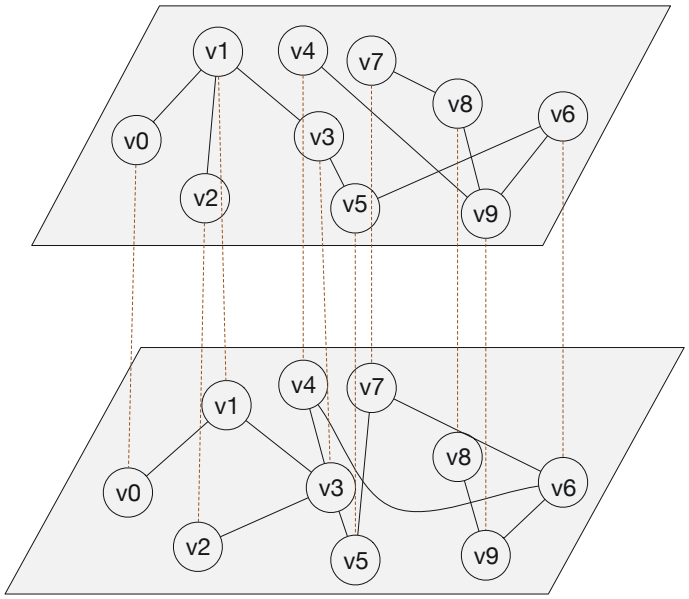


Figure 1.1 : Multi-layer graph example

1.3.1 Approximate Nearest Neighbour Search

- We propose a new c-approximate nearest neighbor search algorithm, namely EI-LSH, for high-dimensional data, which uses a natural incremental search strategy on the projected dimensions.

- We design a new early termination technique to aggressively reduce the number of objects accessed without breaking the theoretical guarantee.

- 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.

1.3.2 Maximum Inner Product Search

As an experimental study, we mainly focus on fairly comparing state-of-the-art MIPS algorithms from different fields and analyse the experiment results. Our contributions are summarised as follows:

- We conducted comprehensive experiments to compare state-of-the-art Approximate MIPS algorithms from different domains. In our experimental study, we test all of the methods adopting a wide range of evaluation metrics. In our experiments, all of the methods are executed in the same environment and all of the implementation tricks have been deleted so that we can get a fair conclusion from the results. We believe that from the experiments, it will be straightforward to select the method which has the best performance.

- We group the algorithms into two categories according to the techniques they used, compare methods within the same group and then report a detail evaluation of the experiment results. We also give a basic analysation and explanation of the weakness and advantage of each algorithm.

1.3.3 Approximate Furthest Neighbour Search

The principal contributions of our work are summarized as follows:

- We propose a novel c-AFN algorithm called RI-LSH for high-dimensional data. It uses continuous searching strategy on each projection dimension.

- We prove that our algorithm has theoretical guarantee and has a more strict approximation bound.

- We conduct extensive performance evaluation against two c-AFN algorithms regarding I/O cost, running time and result accuracy. The results show that

our algorithm can achieve a better performance on both accuracy and efficiency.

1.3.4 Skyline Nearest Neighbour Search

We formulate a new problem to discover skyline nearest neighbors for a given query vertex in a multi-layer graph and propose efficient solutions for it. The main contributions of this thesis are summarized as follows:

- We formulate a new problem which can be applied to discover vertices which are not dominated by others in a multi-layer graph. To the best of our knowledge, the problem has not been formulated before.
- We develop a baseline algorithm to answer the skyline nearest neighbor queries, and then discuss the early-termination condition to improve the performance.
- We perform performance evaluation regarding on real-world graphs. The experimental results demonstrate that the optimizations can improve the performance significantly.

Chapter 2

Literature Survey

2.1 Approximate Nearest Neighbour Search

In this section, we introduce the relevant existing works. Firstly, we introduce some closely related works which are based on Locality-Sensitive Hashing (LSH) with theoretical guarantee, and analyze their pros and cons. Then we will also briefly introduce some other approximate nearest neighbour search (ANNS) algorithms

2.1.1 List-based LSH algorithm

A major solution to overcome the multiple indexes of LSH is the virtual rehashing, which has been used in some LSH papers with a theoretical guarantee such as LSB-tree [83], C2LSH [28] and QALSH [37].

Basically, instead of physically constructing the buckets with different widths to cope with the possible radius (i.e., R values) in c-NN search, the virtual rehashing increases the bucket width based on one hash function only (i.e., random projection of the objects). Specifically, suppose the original hash function $H(o)$ is $(1, c, p_1, p_2)$-sensitive for $R = 1$. To enlarge the search radius from 1 to c, we may simply change the hash function to $H^c(o) = \frac{H(o)}{c}$, where $H^c(o)$ is (c, c^2, p_1, p_2)-sensitive. Then the above procedure is repeated until the termination of the algorithm (i.e., retrieves enough candidates or meets the early termination condition).

Virtual rehashing methods can be grouped into two categories: query-oblivious LSH and query-aware LSH. In the following, we introduce two representative works, C2LSH [28] and QALSH [37], in each category respectively.

C2LSH [28] uses the virtual rehashing function to increase the width of the bucket. Instead of following the AND-then-OR scheme [22], C2LSH does not use the LSH table containing a set of hash functions (i.e., AND scheme) but uses a set of m LSH functions. For an object o, if o collides with query q (i.e., in the same bucket) in at least αm LSH functions, then o will be added to the candidate set. The algorithm will terminate if there are already βn candidates during the search or there exists an object o such that $\| o, q \| \leq cR$ where R is the current search radius. The advantage of C2LSH is that it doesn't require a complex hash table, which can significantly reduce the total number of hash functions. As a result, C2LSH has a much smaller I/O cost compared with LSH and LSH-forest. One disadvantage of C2LSH is that its LSH function is *query-oblivious*; that is, the possible virtual partitions have been determined before the arrival of the query. As illustrated in Fig. 3.1(a), an object close to q in the projected space may be assigned to different bucket even if we keep on increasing the width of the bucket.

LSB-Tree [83] uses a set of LSB-tree to build an LSB-forest instead of hash tables. In each LSH-tree, a data object is projected to a value where $h(o) = \vec{a} \cdot \vec{o} + b^*$. The vector a and o have the same meaning as other LSH methods, and b^* is uniformly distributed in $[0, 2^f w)$ where f is a variable related to the dimensionality of the largest coordinate in each dimension, and w is a constant. LSB-tree uses a Z-order technique to implement the virtual rehashing step. Each data object o is mapped to an m-dimensional data object $G(o) = (h_1(o), ..., h_m(o))$ where $h_i(\cdot)$ is the i_{th} hash function, and then, each $G(o)$ is converted to the Z-order value $z(o)$. The Z-order values are used to build the LSB-tree. The basic idea of an LSB-tree is to use Length of the Longest Common Prefix (LLCP) method to evaluate any two Z-order values' "closeness". The Z-order values stored in the leaf nodes will be visited in a decreasing order of the LLCP with $z(q)$, which simulates the process of processing a set of (R, c)-NN search with increasing radius R.

LSB-tree generally has excellent performance in terms of result accuracy, but the I/O cost is much higher than other algorithms. Furthermore, the LSH function is also *query-oblivious*. Both the LSB-tree and C2LSH have a major defect in that they require c to be a power of 2, and since the approximation ratio of LSB-tree and C2LSH is c^2, the best guarantee they can keep is a 4-approximate nearest neighbor, which might not be accurate enough in some scenario.

QALSH [37] addresses the above issue by proposing the *query-aware* virtual rehashing function. As shown in Fig. 3.1, QALSH always sets the query point as the center of the bucket such that the point close to q in the projected space will have a good chance to share the same bucket. Particularly, the hash function of QALSH is defined as: $h(o) = \vec{a} \cdot \vec{o}$, and for each hash function, the hash values are stored in a $B+$ tree. When q arrives, the query point will always be at the center of the *anchored bucket*, and the data object whose hash value $h(o)$ satisfies $|h(o) - h(q)| \leq \frac{w}{2}$ collides with q. Although QALSH can enhance the search performance with the *query-aware* technique and can accept any c greater than 1, it still has some drawbacks. QALSH has to iteratively increase the bucket width and access all objects touched by the expansion of the bucket. As shown in Fig. 3.1(a), it may explore many unnecessary objects because it is difficult to find a good way to expand the bucket.

2.1.2 Tree-based LSH method

Although LSB-tree and QALSH use B+tree or B-tree to build the index, they are still considered as list-based LSH algorithms since the searching is done on single hash functions whose structures are lists, and the tree structure is simply used to speed up the search. A tree-based LSH method uses a multi-dimensional index. In a nutshell, SRS [82] reduces the problem of "approximate NN search in high dimensional space" to an "exact T-NN search in a low dimensional space". As

shown in [70, 82], if we use m independent 2-stable function to map each data point in $\mathcal{R}^d$ to m-dimensional space $\mathcal{R}^m$, the square of L_2 distance between two objects o_1 and o_2 within $\mathcal{R}^m$ (namely projected distance) follows the χ^2 distribution with m degrees of freedom regarding the square of their distances in $\mathcal{R}^d$. SRS uses the classical multi-dimensional index, e.g., R-tree, to organize the objects in $\mathcal{R}^m$ space. Then, an incremental exact NN search can be conducted to solve the problem. The key idea of SRS is very intuitive and the index size is very small compared with all the other c-ANN methods under the same theoretical guarantee.

Discussion. The main limit of SRS is I/O efficiency when the index cannot be held in the main memory. According to Algorithm 6 of SRS, the number of projected dimensions m increases quickly with regard to the approximate accuracy. Moreover, as shown in the empirical study, a considerable large number of objects (i.e., disk pages) are accessed during the search. On the other hand, it is well known that a multi-dimensional index like R-tree cannot efficiently support an exact NN search for data with projected dimensionality over 5, and the performance drops dramatically when dimensionality increases due to the curse of dimensionality. When the approximation ratio c is set to be a smaller value, such as 2, SRS will require a 15-dimensional R-tree to undertake the exact NN search, but the performance will be extraordinarily bad. Moreover, I/O invoked in SRS is random I/O, which is much more expensive than sequential I/O.

2.1.3 Other relevant work

In this paper, we focus on c-ANN algorithms which have a rigorous theoretical guarantee. 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 [56] for approximate NN problems as an improvement of LSB-tree, which used linear order instead of Z-order for better I/O efficiency. In [55], an I/O efficient

algorithm was proposed based on the PQ method [43]. Arora *et al.* proposed HD-Index in [5] which consisted of a hierarchical structure called RDB-tree to support the c-NN query in high-dimensional datasets. Gu *et al.* [33] applied the PCA [35] technique to project the data from high-dimensional space to low-dimensional space and proposed OR-tree to optimize the I/O cost.

In addition to the aforementioned external memory based techniques, a large body of work has been proposed in the literature (e.g., Database, Machine Learning, Data Mining and Multimedia) to enhance the performance of approximate NNS assuming that the index and data can be held in the main memory without a theoretical guarantee. As shown in [52], they can be classified into four categories based on the nature of the techniques namely: (1) *LSH-based methods* such as multi-prob LSH [57], Bi-level LSH [69]), and DSH [30]); (2) *Encoding-based methods* such as Spectral Hashing [92], Neighbor Sensitive Hashing [72], Selective Hashing [29], and Product Quantization [43], (see [88] for a comprehensive survey); (3) *Tree-based space partition* methods such as randomized kd-tree [81], FLANN [65] and Annoy [11]; and (4) *Neighborhood-based* methods such as KGraph [24], HNSW [61] and DPG [52]. It is also worth mentioning that some recent works take advantage of hardware to speed up the approximate NN search such as GPU [89, 46] and FPGA [96].

2.2 Approximate Maximum Inner Product Search

Except for the algorithms we selected in this thesis, there are more existed methods for Approximate MIPS problem and exact MIPS problem. Similar to H2-ALSH, there some other Locality Sensitive Hashing based Approximate MIPS algorithms such as [50] and [79], both of which also provide theoretical guarantee to their results. Generally, LSH-based Approximate MIPS algorithm transforms the dataset into another data space and then do nearest neighbour search to retrieve MIPS can-

didates. Since LSH generally requires a longer hash code or multiple hash tables to get an accurate enough result, which means a larger index size, and the recall grows slowly when index size increasing , there is another hash-based idea called learning to hash. The basic idea of learning to hash algorithms is to generate hash functions by studying the dataset. More specifically, the purpose is make the hash distance between two points can better present the real distance between them in the original data space. [87] [99] [32] [95] [42] propose algorithms to solve Approximate MIPS by learning to hash, which are evidently more efficient than data-independent hash methods. Another approach is based on product quantization [31] [6] [34]. These algorithms also clustered vectors multiple times, and every time, each cluster contains same number of points. Some approximate nearest neighbour algorithms based on graphs or trees also support Approximate MIPS search such as kd-trees, k-NN graphs [86, 16, 4] and Navigable Small World graphs which was first proposed by J. Kleinberg [49, 48] and then was adopted by many nearest neighbour search and Approximate MIPS algorithms [53, 47, 7]. Hardware also plays an important role to improve seed up Approximate MIPS search such as GPU [77]. Exact MIPS algorithm also exists. LEMP and its variants [85, 84] can retrieve exact and approximate maximum inner product points avoiding too expensive computations.

2.3 Approximate Furthest Neighbour Search

To solve the c-approximate furthest neighbour search (c-AFN) problem, there are basically two classifications: (1) using hash functions or (2) tree-based solutions. Qiang Huang proposed two efficient c-AFN algorithms in TKDE 2017 [36] called RQALSH and RQALSH*. Both of the two algorithms are based on the LSH scheme proposed by Indyk [22], and the difference is that RQALSH holds a theoretical guarantee, while RQALSH* utilized machine learning skill to heuristically pre-process the data. RQALSH used a query-aware reverse hashing function to project the d-

dimensional dataset to a m-dimensional dataset and use B+ trees to store the hash values. The advantage of RQALSH is that it uses a query aware hash function so the query point is always located at the center of each bucket, and it adopts B+ tree to build the index and can have a good I/O performance since searching on B+ tree cost a sequential I/O. DrusillaSelect [21] used a novel hashing strategy for approximate furthest neighbor search that selects projection bases using the data distribution. It selects a small number of data points as candidates based on data distribution and also has a variant which provides an absolute approximation ratio. Another method with theoretical guarantee is QDAFN [68] which has been adapted for external memory. Compared with Indyk's implementation, QDAFN is easier to implement and keeps the theoretical guarantee as well. It also has a heuristic version called QDAFN*. For the tree-based algorithms, two of the typical algorithms are [94] proposed by B Yao and [20] proposed by RR Curtin. Both of the two algorithms construct a tree using the data points and then adopt the branch-and-bound pruning strategy to exclude the nodes which is not possible to be a FN of the query point. Besides, most of the tree-based methods for NN problem can also be used for c-AFN problem as well such as kd-tree [10], R+ tree [8], and etc.

2.4 Skyline Nearest Neighbour Search

Distance queries. There are some algorithms to answer distance queries on a graph. Some methods are based on 2-hop cover [19]. 2-hop cover first computes a spanning tree T for graph G as a part of the index. When computing distance $dis(u, v)$, it outputs the minimum over $dis(u, w_1) + dis_T(w_1, w_2) + dis(w_2, v)$ where w_1 and w_2 are vertices in labels of u and v respectively, and dis_T is the distance in the tree T. There are several methods using 2-hop cover to solve the problem, such as hierarchical hub labeling [1] and highway-centric labeling [45] which are suitable for road networks. Another approach is based on tree decompositions which is also

reported to be efficient. Tree decomposition on graph G is a tree T in which each vertex is mapped to a set of vertices in G called a *bag*. The set of bags containing a vertex in G forms a connected component in T. [67] uses tree decomposition for distance queries on road networks.

For scale-free networks, such as social networks and web graphs, two important algorithms are Pruned Landmark Labeling [2] and Hop-Doubling Labeling [44]. Hop-Doubling Labeling doubles the length of each path in each iteration to compute the 2-hop labels. In [3], [15], [91], tree-decomposition is also used.

Skyline. Skyline and its related problems have been extensively studies. It was first proposed by Borzsonyi et al [14]. The divide-and-conquer approach [63] divides the dataset into several partitions to fit in memory, then compute the partial skyline for each partition, and the final result is obtained by merging the partial skylines. Block Nested Loop (BNL) is a straightforward approach to calculate skyline. It compares each point p with other points, and reports whether p is a part of skyline or not. SFS [17] improves the performance of BNL. It uses a presorting to ensure that if p dominates p' then p must be visited before p'. Index [26] is an algorithm which organizes a set of d-dimensional points into d lists, if and only if the coordinate p_i on the ith axis of the point p is the minimum coordinate among all the d dimensions. Then it uses a set of B-tree to store the lists and compute the skyline. Papadias et al. proposed an efficient algorithm to progressively find the skyline [71]. Sheng and Tao [78] proposed an algorithm to compute skyline efficiently on external memories.

Chapter 3

Approximate Nearest Neighbour Search

3.1 Overview

In this chapter, we introduce two I/O efficient algorithms, called EI-LSH and I-LSH, for c-ANN in high-dimensional space. I-LSH has already been published in our conference paper [90] and EI-LSH was proposed in our extension of this paper. The rest of this chapter is organized as follows. First we formally define the c-ANN problem and give preliminary definitions in Section 3.3. In addition, we also discuss advantages and shortages of some competitive algorithms in this section. Then in Section 3.4, we describe our motivation of the algorithm and then detailed illustrate our approach. The rigorous analysis of correctness of our algorithms are given in Section 3.5. Section 3.6 evaluates our algorithms and state-of-the-art c-AFN methods. At last, Section 3.7 concludes the chapter.

3.2 Motivation

The key of the LSH is that we hope two close objects in the high dimensional space are also close to each other in each projected dimension. Assume there is a nice hash function such that the closeness of the objects to q are well preserved in the projected dimension. Intuitively, we should incrementally access the projected objects according to their distances to q in the projected dimension as shown in Fig. 3.1(c). However, 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. Particularly, the

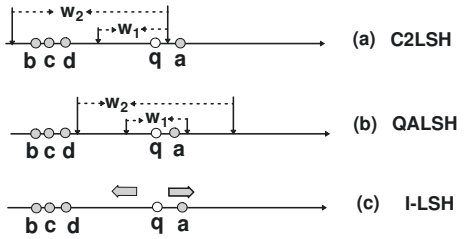


Figure 3.1 : Motivation for EI-LSH

hash values of the objects will be assigned to the buckets in C2LSH and QALSH. We say an object collides with q if it is assigned to the same bucket with q in the projected dimension. The objects collided with q will be explored, and the bucket width will keep growing if necessary (e.g., from w_1 to w_2 in Figs. 3.1(a) and (b)), and hence new objects may collide with q and be processed. It is shown in [37] that the LSH function in C2LSH is *query-oblivious*; that is, the possible boundary of the buckets have been pre-determined before the arrival of the query. In Fig. 3.1(a), although the object a is close to q in the projected dimension, it will not be explored even the bucket width grows from w_1 to w_2. QALSH addresses this issue by proposing *query-aware* LSH function where, as shown in Fig. 3.1(b), the query is always centered at its corresponding bucket. By doing this, the object **a** in the example will be explored at the beginning. Nevertheless, because the distribution of the hash values is not uniformly distributed in practice, the search may encounter too many new objects (e.g., Fig. 3.1(a)) or none new object (e.g., Fig. 3.1(b)) when bucket width w_1 is extended to w_2.

3.3 Preliminary

In this section, we present the problem definitions. Some important notations used throughout the paper are summarized in Table 5.1.

Table 3.1 : Summary of Notations

Notation	**Definition**
n	the number of point objects in the dataset
d	the dimensionality of the dataset
m	the number of hash functions (projected dimensions)
q	the query object
$\parallel o_1, o_2 \parallel$	the Euclidean distance between o_1 and o_2
o^*, R^*	NN object of q, with distance R^*
o_{min}, R_{min}	c-ANN object returned, with distance R_{min}
$h_{\vec{a}}(o)$	the hash value of point o using the vector $\vec{a}$
$d_i(o)$	projected distance w.r.t i-th hash function with $d_i(o) = \|h_i(o) - h_i(q)\|$
c and δ	approximate ratio and success probability
w	the initial bucket width in LSH functions
r	the search radius on projected dimensions
R	$R = \frac{2r}{w}$, radius of ball $B(q, R)$ in $\mathcal{R}^d$ space regarding search radius r
γ	the probability that the algorithm is terminated by the normal termination condition
R^+	$R^+ = \lambda r$, distance threshold for early termination where λ is a pre-computed constant

3.3.1 Problem Definition

First we give the formal definition of the c-ANN problem. Considering a dataset D with n point objects in a d-dimensional space, denoted by $\mathcal{R}^d$. We are particularly interested in the high-dimensional case where d is a large number (e.g., $d \geq 50$). The coordinate value of an object o on the i_{th} dimension is denoted as $o[i]$. In this paper, we focus on Euclidean distance which is one of the most popular distance metrics widely used in a variety of applications. For a given query object q, the Euclidean distance between o and q is denoted by $\parallel o, q \parallel = \sqrt{\sum_{i=1}^{d}(o[i] - q[i])^2}$. For presentation simplicity, we use "distance of the object" to represent the "distance between the object and the query" whenever there is no ambiguity. 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 $\parallel o, q \parallel \leq cR^*$ where c is the approximate ratio.

Problem statement. In this paper, we propose an efficient algorithm to solve the c-ANN problem in d-dimensional Euclidean space with a 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. The theoretical guarantee will always hold regardless of the data distribution.

3.3.2 LSH with bucket partitioning

Locality sensitive hashing (LSH) was first introduced by Indyk *et* al. in 1998 [39] to solve the c-ANN problem in binary Hamming space and was extended by Datar *et* al. [22] to Euclidean space.

In the nutshell, the LSH functions map the objects in a d-dimensional dataset

$\mathcal{D}$ into an m-dimensional space, denoted by $\mathcal{R}^m$, where $m \ll d$ and the relative distance among the objects are kept in the mapped space. That is, if two objects p_1 and p_2 are close to each other in d-dimensional space, hopefully they will be mapped to nearby positions after a random projection. By partitioning these real values (i.e., one dimensional data) with a width w, the objects in $\mathcal{R}^d$ are mapped to different *buckets*. Intuitively, two close objects in space $\mathcal{R}^d$ will have a good chance of being mapped into the same bucket, and vice versa for the two distant objects.

An LSH function family $\mathcal{H}$ for a distance function f is defined as (r_1, r_2, p_1, p_2)-sensitive if and only if for any two data points x and y, there exists two distance thresholds r_1 and r_2 and two probability thresholds p_1 and p_2 which satisfy: $\begin{cases} \Pr_{H\in\mathcal{H}}[H(x) = H(y)] \geq p_1 & \text{, if } f(x,y) < r_1 \\ \Pr_{H\in\mathcal{H}}[H(x) = H(y)] \leq p_2 & \text{, if } f(x,y) > r_2 \end{cases}$. This implies that the chance of mapping two objects x, y to the same hash value (i.e., *bucket*) increases as the distance between x and y, $f(x, y)$ decreases.

In [22], an LSH function H can be formally represented by $H_{\vec{a},b}(o) = \frac{\vec{a}\cdot\vec{o}+b}{w}$ where $\vec{a}$ is a d-dimensional vector for the random projection, o is the d-dimensional data object, w is the bucket width, and b is a variable randomly chosen from $[0, w]$. The p-stable distribution, e.g., the Gaussian/normal distribution for $p = 2$, and Cauchy distribution for $p = 1$, is able to construct $\vec{a}$ of the LSH function for the random projection. A formal definition of p-stable distribution follows.

Definition 2: **p-stable distribution** [22] A distribution D over $\mathcal{R}$ is called *p-stable*, if there exists $p \geq 0$ such that for any n real numbers v_1, ... , v_n and i.i.d. variables X_1 ,..., X_n with distributions D, the random variable $\sum_i^n v_i \cdot X_i$ has the same distribution as $(\sum_{i=1}^n v_i^p)^{1/p}$ where X is a random variable with the distribution D. p-stable distribution has only been found when $p = 1$ and $p = 2$. When $p = 1$, the p-stable distribution is the Cauchy distribution, when $p = 2$, the distribution is the Normal distribution.

Although the p-stable distribution is only found when $p = 1$ and $p = 2$, it is also possible to use LSH to undertake the c-ANN search by conducting a transformation on the dataset [98]. In this paper we only focus on the Euclidean space where $p = 2$. In [22], only one LSH function has some guarantee of the probability, but to boost accuracy and to limit the candidate set size, LSH [22] requires k hash functions to build an LSH table, and all L hash tables are required to keep the theoretical guarantee. For an object p and a query point q, if q falls into the same bucket on the i_{th} function with q, then we say p collides with q on function h_i. If p collides with query q on all the k hash functions in a hash table, then p is considered as a c-NN candidate. The algorithm will stop if (i) there is a candidate p, $\| p, q \| \leq cR$; or (ii) $3L$ candidates have been accessed.

Discussion. LSH was designed for (R, c)-NN queries which is simply a decision version of the c-ANN problem. Particularly, for a given distance R, the algorithm will build an index considering the distance and then will return an object within distance cR to query point q with at least a constant probability. But to solve the c-ANN problem, radius R can't be determined at the beginning. Since the index of LSH is built on radius R, when the radius has to be changed to different values in the c-ANN search, LSH with a fixed bucket partitioning requires multiple indexes for different radii (e.g., 1, c, c^2, ...) because the radius needs to be expanded until enough candidates can be found. However, multiple indexes will lead to indexes of enormous sizes, which is the main drawback of LSH. It is almost impossible to solve a c-ANN search using LSH with a fixed bucket partitioning.

3.4 Our Approach

In this section, we present our early-termination condition with the incremental LSH technique, which is I/O efficient with rigorous theoretical guarantee. In Section 3.4.1, we briefly introduce the motivation of our approach. Section 3.4.2

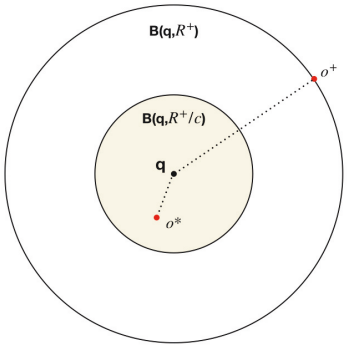


Figure 3.2 : Example: When ET could fail

describes our LSH algorithm for c-ANN problem and Section 3.4.4 shows that our approach can be immediately extended to support top k c-approximate nearest neighbor search (c-k-ANN).

3.4.1 Motivation

For an object o and a hash function h_i, we use $d_i(o)$ to denote $|h_i(a) - h_i(q)|$, namely **projected distance** w.r.t h_i, which is distance between o and q in the i-th projected dimension. Just like other LSH schemes, a point o will be found as a candidate if o has been visited on enough hash functions.

In a list-based LSH algorithm, the early-termination condition is used when a "good enough" candidate has been found. For the traditional Early-termination condition, it only uses a very loose condition $R_{min} \leq cR$ to evaluate if the current best candidate is close enough, where R_{min} is the current smallest distance and R is the current radius. Once a point o with distance $Rmin$ satisfies the condition, it will be a correct c-ANN of q without any false possibility according to the definition of c-ANN. However, if we hope to terminate the algorithm as early as possible without breaking theoretical guarantee, calculating the possibility that the current best candidate is a correct c-ANN could be a better idea. As shown in figure 3.2,

suppose there is only one hash function. q is the query point and o^+ denotes the current nearest candidate to the query point q with distance R^+. If o^+ is returned as the NN of q, the result is wrong if and only if there exist a point o^* which falls in the ball $B(q, \frac{R^+}{c})$ and $\| h(o^*), h(q) \| \geq \| h(o^+), h(q) \|$, in other words, o^* is not found as a candidate before o^+.

In our early-termination condition, an important feature is the projection distances of each data point on all the hash functions. So we also propose an algorithm called incremental LSH to support our ET better. The key idea of our **Early termination driven Incremental LSH** (**EI-LSH**) method is to *incrementally* access the objects (i.e, IDs and their hash values) according to their projected distances, equipped with new aggressive early termination condition. As shown in Fig. 3.3 (a), suppose there is one hash function h_1 and By a_1, we denote the projected distance $d_1(a)$. In this example, a_1, b_1, c_1 and d_1 will be sequentially accessed. When there are more than one hash function (e.g., two hash functions h_1 and h_2 in Fig. 3.3(b)), the search will be conducted simultaneously with the same search radius. In Fig. 3.3(b), the search region is a square. a_1 (i.e, ID of the object a and its hash value on h_1) will be accessed first, followed by b_2, b_1, d_2, etc.

Given the incremental search strategy and the new ET framework, the key challenge is to decide when the search should be terminated and how to identify a set of candidate objects such that the c-ANN can be returned with theoretical guarantee. In Section 3.4.2, we propose our incremental LSH algorithm. Suppose there are m hash functions and n objects, an object becomes a candidate object if it has been accessed at least αm times, and we will have at most βn candidates, where α and β are pre-defined parameters according to the desired theoretical guarantee. The limitation of the candidate set could offer a guarantee on the I/O performance. On the other hand, we also devise early termination technique based on the closest candidate object seen so far. We set a value of p_{ET} which demonstrates the possibility

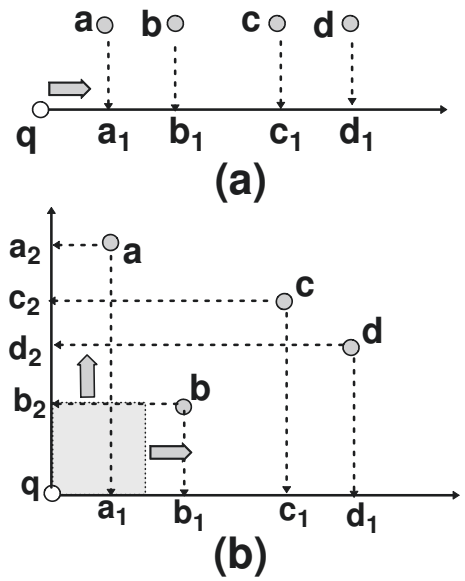


Figure 3.3 : Example for incremental search

that a real c-ANN has already be found at the current projection radius r, and then use this value to calculate a bond $R_{plus} = \lambda r$. We maintain the current best candidate o_{min} with a real distance R_{min} to the query point q, and compare R_{min} with R_{plus}. If R_{min} is smaller than R_{plus}, a value only related to p_{ET} and r, new ET will play a role and o_{min} will be returned as the result. As we explained in Section 3.5, once p_{ET} is given, R_{plus} is only related to r and can be represented as λr, using R_{plus} instead of calculating the possibility for each radius r and comparing with p_{ET} could avoid integral operation which can be very expensive when repeating multiple times. The theoretical underpinning of our approach is provided in Section 3.5.

3.4.2 Incremental LSH with new ET

In this subsection, we describe our incremental LSH (EI-LSH) with new ET approach in details. The setting of the parameters such as m, α, β and λ will be explained in the following theoretical analysis in Section 3.5. Our technique consists of two parts: *indexing* and *query processing*.

Indexing

The indexing step is similar to most LSH algorithms. In the indexing part, the high-dimensional dataset is projected to a low-dimensional space using a set of LSH functions. More specifically, 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 $h_i(o)$ for the i-th hash function. Particularly, we have $h_i(o) = \vec{a}_i \cdot \vec{o}$ where $\vec{a}_i$ is randomly chosen from the normal distribution $\mathcal{N}(0,1)$ (i.e., 2-stable distribution). 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. After indexing, the d-dimensional dataset is mapped to a m-dimensional dataset where $m \ll d$. A good property of LSH is that the required hash function number m is not effected by the dimensionality d, so even the dataset is in a really high dimensionality space, m can still be a small value. Thanks to the good performance of B+ tree, our index can easily handle large scale data and support the insert/deletion of the objects in an I/O efficient way. As shown in Section 3.5, we need to choose a proper m for the desired approximate ratio c and success probability δ. Let m_{max} denote the maximal m value we will support, we build the index with m_{max} B+ trees, denoted by $\mathcal{B}$, and randomly choose m B+ trees from the forest to do the search. Note that we also use a B+ tree to maintain the objects in $\mathcal{R}^d$ space, where the object ID is the search key.

Query processing.

In general, a query object q in the d-dimensional space will also 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. According to LSH property, if a point o is close to q on most LSH functions, it is likely to be a c$-ANN$ of q. So an object becomes a candidate if αm hash values of the point

Algorithm 1: **EI-LSH** ($\mathcal{B}$, q)

Input : $\mathcal{B}$: m B+ tree indexes for object IDs and hash values;

q: the query object;

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 $cn(o) := cn(o) + 1$;
8 **if** $cn(o) == \alpha m$ **then**
9 compute $\| o, q \|$ and update o_{min};
10 $n_{can} := n_{can} + 1$;
11 **if** $R_{min} \leq \lambda r$ **then**
12 break;
13 **return** o_{min}

have been accessed. The search will terminate if there are already βn candidate objects or the new early termination condition is satisfied. Note that α, β and λ are pre-defined parameters, and their settings will be discussed in Section 3.5.

Algorithm 3 presents the pseudo-code of our incremental LSH technique. We use n_{can} to record the number of candidate objects seen so far and o_{min} is the candidate object which is closest to q (Line 3). The query q will be mapped to m hash values (Line 3). Lines 4-3 conduct incremental search according the projected distances of the objects. Note that an object may appear m times. During the search, Line 3 keeps on retrieving the next closest object o with smallest projected distance, say

from the i-th hash function, (Line 3) and the search radius r is set to the projected distance (i.e., $|h_i(o) - h_i(q)|$ at Line 3). Regarding the example in Fig. 3.3(b), we have $o \leftarrow a$ and $i \leftarrow 1$ in the first iteration, while $o \leftarrow b$ and $i \leftarrow 2$ in the second iteration. Then Line 3 increases the number of visited times of o, denoted by $cn(o)$, by one. The object o becomes a candidate object if $cn(o)$ reaches αm. Then we compute its distance to q in $\mathcal{R}^d$ space (i.e., $\| o, q \|$) and o_{min} may be updated by o, where o_{min} keeps the closest candidate object seen so far (Lines 3-3). The incremental search will terminate in two cases: (1) we already have βn candidate objects (Line 4); or (2) the early termination condition is satisfied based on the current o_{min} and search radius r (Lines 3-3). Finally, o_{min} is returned as the c-ANN, which is the closest candidate object.

Correctness. For the given query q, approximate ratio c, and success probability δ, Section 3.5 shows that we can choose proper m, α, λ, and β values such that our EI-LSH algorithm can return c-ANN with probability at least δ.

CPU Costs. The dominant CPU costs in Algorithm 3 are the computation of distance in $\mathcal{R}^d$ space (Line 3) and the retrieval of the next object in the incremental search against m projected dimensions (Line 3). As the hash values are readily sorted by a B+ tree for each projected dimension, the bi-direction search can be conducted after the position of $h_i(q)$ is identified. A min-heap can be used to maintain $2m$ closest objects on each search direction, it takes $O(\ln(2m))$ time to conduct incremental search. Thus, the CPU costs of EI-LSH search are $O(s(d + \ln(2m)))$ where s is the number of iterations at Lines 4-3.

I/O costs. The dominant I/O costs also come from the computation of distance (Line 3) and the incremental search (Line 3). Both hash values and object data IDs are organized by B+ trees, we use the number of leaf-node visited to evaluate the I/O costs because it is the dominant cost. We use I_{seq} and I_{ran} to denote the unit

cost of sequential I/O read and random I/O read, where I_{seq} is much cheaper than I_{ran} in practice. To compute the distance between o and q, one random I/O will be invoked to load the object with d coordinate values. For the incremental search, as the hash values are sorted in B+ trees, we can retrieve the hash values and the corresponding object IDs in a pre-fetch fashion; that is, we use one random I/O to identify the position and apply l sequential I/O to load l pages. Thus, the total I/O costs are $O(\frac{s}{n_e} \times I_{seq} + \frac{s}{n_e \times l} \times I_{ran} + n_{can} \times I_{ran})$ where n_e is the average number of entries per page*, n_{can} is the number of candidate objects accessed, and s is the number of iterations (i.e., the number of hash values visited). In our implementation, we set $l = 10$.

3.4.3 Incremental LSH with traditional ET

Our algorithm can also adopt the traditional early-termination condition which is more safe. There are also two steps of I-LSH: indexing and querying. The indexing step is totally same as EI-LSH, while in the querying step, there is something different. Instead of comparing R_{min}, the current shortest distance to q, with λr, I-LSH compares R_{min} with cR where $R = \frac{r}{w/2}$. Since R is the current radius of $B(q, R)$, if an object o_{min} satisfies $R_{min} < cR$, it is safe to terminate the querying and return o_{min} as the result.

3.4.4 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 thesis 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 c-k-ANN search if $\| o, q \| \leq c \| o_k, q \|$, where o_k is the k-th nearest neighbor of q in the objects $\mathcal{D}$.

*Note that each entry takes 8 bytes for one hash value and the object ID.

Algorithm 2: **I-LSH** ($\mathcal{B}$, q)

Input : $\mathcal{B}$: m B+ tree indexes for object IDs and hash values;

q: the query object;

Output : o: the c-ANN object

```
1  n_can := 0; d_min := 0;
2  Apply m hash functions on q;
3  while n_can < βn do
4      o ← next object with smallest projected distance;
5      i ← the projection dimension o comes from;
6      r ← |h_i(o) − h_i(q)|;
7      cn(o) := cn(o) + 1;
8      R ← 2r/w;
9      if cn(o) == αm then
10         compute ‖ o, q ‖ and update o_min;
11         n_can := n_can + 1;
12     if R_min ≤ cR then
13         break;
14 return o_min
```

Algorithm 3 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 its distance to q will be used for early termination test; and (3) the k most closest candidate objects will be returned as the result of c-k-ANN search.

3.5 Analysis

In this section, we prove that if the new ET framework is used on a proper LSH method, the algorithm can still have theoretical guarantee. Besides, 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 a c-ANN with probability at least δ.

Before the formal proof, we stress some important notations frequently used. By o^*, we denote the NN object with distance R^*. 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. The object o_{min}, which is the closest candidate object seen so far, will be returned as c-ANN in Algorithm 3. We use R_{min} to denote its distance to q, and we say o_{min} is correct if $R_{min} \leq cR^*$.

Since the correctness of the ET framework is also related to the correctness of EI-LSH, we first prove without ET, EI-LSH can return a correct c-ANN with a success possibility of δ.

Incremental increase of search radius r

Same as other LSH based methods, the correctness of our proposed method also relies on the (r_1,r_2,p_1,p_2)-sensitive property of the hash function. The key difference is that we incrementally increase the search radius r in Algorithm 3, following the distance of the hash values w.r.t the projection of q, i.e., $|h_i(o) - h_i(q)|$ with $1 \leq i \leq m$.

In this thesis, we use the 2-stable distribution to construct the hash function h. Let $f_h(o)$ denote the distance between the object o and the query q in the projected dimension resulting from the hash function h, where $f_h(o) = |h(o) - h(q)|$. As shown

in [22], following lemma indicates that the $f_h(o)$ follows the normal distribution related to the distance between o and q in $\mathcal{R}^d$ space.

Lemma 1: For an object o with $\| o, q \| = \zeta$, $f_h(o)$ follows the normal distribution as $\zeta\mathbf{Z}$ where $\mathbf{Z}$ is a random variable drawn from $\mathcal{N}(0,1)$ with density function $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$.

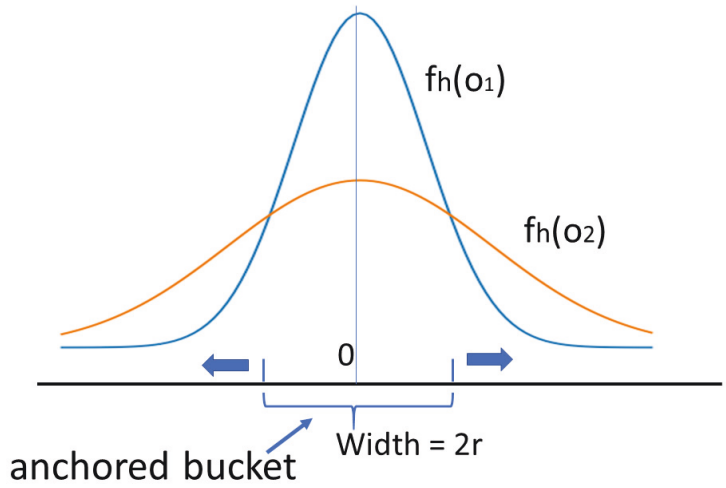


Figure 3.4 : Example of $f_h(x)$

Suppose we have $\| o_1, q \| = 1$ and $\| o_2, q \| = 2$ Fig. 3.4 illustrates the distribution of $f_h(o_1)$ and $f_h(o_2)$, respectively. Given the bucket width w and centered at $h(q)$, we use $\eta(R, w)$ to denote the probability that o and q fall in the same bucket with $\| o, q \| = R$, which is the probability mass $\int_{-\frac{w}{2R}}^{\frac{w}{2R}} \phi(x)dx$.

In this thesis, 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 $\| o, q \| < r_1$ and $Pr(|h(o) - h(q)| \leq \frac{w}{2}) \leq p_2$ given $\| o, q \| > r_2$. 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 2: 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 1: 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 2, 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 2, the search radius r can be any real value with $r > 0$. And the hash function is $(\xi, c\xi, p_1, p_2)$-sensitive where $\xi = \frac{2r}{w}$.

Correctness of the Algorithm

Algorithm 3 may be terminated by normal termination condition (denoted by `NT`) or early termination condition (`ET`), and both may be correct with some probability. Thus, we need to carefully consider the correlation of two termination conditions. Suppose the LSH algorithm will be stopped by termination condition 1 (NT) with γ possibility, and will be terminated by early-termination condition with $1 - \gamma$ possibility, the final success possibility is $\gamma \cdot P_{NT} + (1-\gamma) \cdot P_{ET}$, where P_{NT} and P_{ET} are the success possibility of NT and ET separately. Our goal is to choose a set of parameters so that the final success possibility δ is a constant value. First, we just suppose that the normal termination condition and the early-termination condition share the false possibility and prove the correctness of the algorithm. Furthermore, we discuss what is the real proportion of false probability for the two termination conditions. Below, we present the key idea of our proof, followed by the details.

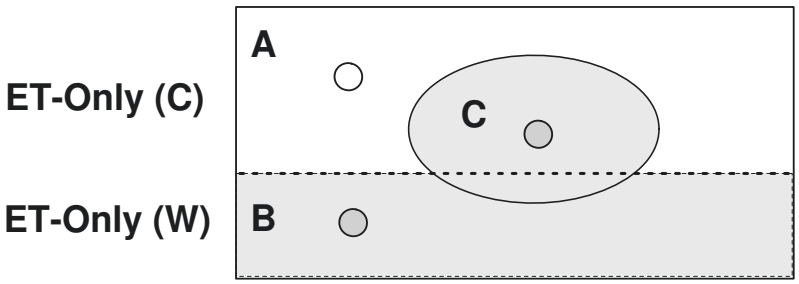


Figure 3.5 : The sample space $\mathcal{S}$

(1) Outline of the Proof. Let $\mathcal{S}$ denote the sample space resulting from the m hashing functions, and each sample is an instance of the projections for the objects and the query. We first partition $\mathcal{S}$ into two disjoint two sets A and B. Suppose we only consider the early termination condition in Algorithm 3, denoted by `ET-Only`,

the instance in A means that `ET-Only` will terminate with a correct o_{min}. Other instances belong to the set B. Due to the use of normal termination condition in Algorithm3, some instances in A may be wrongly terminated by `NT`. Let C denote the set of instances in which an instance[†] is wrongly terminated by `NT`; that is, there are βn "bad" candidate objects and early termination condition is not satisfied. Note that we say a candidate is "bad" if its distance is larger than cR^*. Note that if both conditions are satisfied at the same time, the `ET-Only` algorithm is not interrupted by `NT` because o_{min} will be returned anyway.

Let $P(B)$ and $P(C)$ denote the probability mass of the samples in B and C, respectively. If we have $P(B) + P(C) \leq 1 - \delta$, then Algorithm 3 will return the correct c-ANN object with probability at least δ.

(2) Parameter Settings. The desired accuracy in this thesis is defined by c and δ, where δ is the success possibility and c is the approximate ratio. Same as other LSH papers, we set $\delta = 1/2 - 1/e$. Note that δ actually can be boosted by repeating the procedure and taking the median of the results. For the initial bucket width w, we set w to 3.5. Then we can calculate the probability p_1 and p_2 following the Lemma 1 with $R = 1$ and $R = c$, respectively. According to Lemma 2, it is immediate that the hash function is (R, cR, p_1, p_2)-sensitive regarding the search radius r with $R = \frac{2r}{w}$.

In this thesis, let $\delta = 1/2 - \delta'$ (So when $\delta = 1/2 - 1/e, \delta' = 1/e$), we set $\beta = 0.01$ 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 $m = \lceil \frac{ln\frac{1}{\delta'}}{2(p_1-p_2)^2}(1 + \sqrt{\frac{ln\frac{2}{\beta}}{ln\frac{1}{\delta'}}})^2 \rceil$. For early termination, we set $P_E = \frac{1-\delta}{2}$ and the calculation of λ will be introduced in the following part.

(3) Computation of $P(B)$ for early termination only. Now we show that, if we only use early termination condition in Algorithm 3, we have $P(B) \leq P_E$. Based

[†]Here, it is not necessary that the instance belongs to A.

on the Lemma 1, it is immediate that the probability of hitting the anchored bucket will monotonically decrease when the distance between object and the query grows, which is formally described as follows.

Lemma 3: Given two objects o_1 and o_2 where o_1 is closer to q than o_2, with distance R_1 and R_2 respectively, we have that $\eta(R_1, w) \geq \eta(R_2, w)$, recall that $\eta(R, w)$ denote the probability that an object o with distance R falls in the anchored bucket with width w.

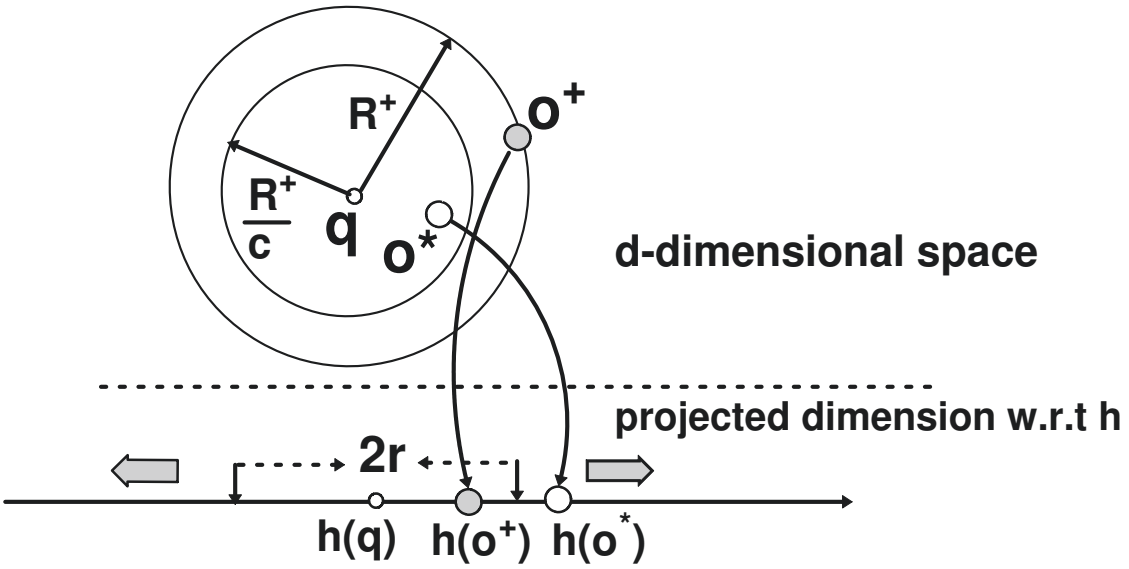


Figure 3.6 : Motivation of early termination

As shown in Fig. 3.6, o^+ denotes the closest candidate object seen so far with distance R^+. If we simply return o^+ as the c-ANN object, the result is *incorrect* if $R^* \leq \frac{R^+}{c}$ (i.e., o^* falls in the ball $B(q, \frac{R^+}{c})$ *and* o^* is not a candidate object (e.g., $|h(o^*) - h(q)| > r$ for $m = 1$). Thus, for the given r, m, c, and the probability P_E, we can come up with a distance R^+ such that, with probability $P_{ET} = 1 - P_E$, an object o with distance $\frac{R^+}{c}$ will become a candidate w.r.t search radius r. Specifically, for the given P_E and m, we can find a probability p such that $P_E = 1 - \sum_{i=\alpha m}^{m} C_m^i p^i (1 - p)^{(m-i)}$. That is, for an object with distance $\frac{R^+}{c}$, it will have probability p to hit an

anchored bucket w.r.t r on a single hash function, and hence become a candidate (i.e., hit at least αm times) with probability $1 - P_E$ on m independent 2-stable hashing functions. With the given p and r, we can calculate a distance $R^+ = \lambda r$ such that $\eta(\frac{\lambda r}{c}, 2r) = p$. Particularly, we have $\lambda = \frac{c}{N^{-1}(\frac{p+1}{2})}$ and $N^{-1}(\cdot)$ is the inverse function of the cumulative distribution function of the standard normal distribution. Note that the calculation of λ is not related to the search radius r, and hence λ can be pre-computed based on P_E, c, and m.

Then we have the following lemma showing that $P(B) \leq P_E$.

Lemma 4: The Algorithm 3 with early termination only will return an *incorrect* c-ANN with probability at most P_E; that is, we have $P(B) \leq P_E$;

Proof 2: According to the Lemma 3 and the calculation of R^+, if the early termination condition is satisfied (i.e., $R_{min} \leq R^+$), the probability that the returned o_{min} is not a c-ANN is at most P_E. That is, $P(B) \leq P_E$.

(4) Computation of $P(C)$. In this part, we show that $P(C) \leq \frac{1-\delta}{2}$. Recall that for each instance in C, Algorithm 3 is terminated by normal termination condition with an incorrect o_{min} meanwhile the early termination condition is not satisfied; that is, we have: (1) βn "bad" candidates; and (2) $R_{min} > R^+$.

Lemma 5: With probability at most $\frac{1-\delta}{2}$, Algorithm 3 is terminated by normal termination condition with an *incorrect* c-ANN; that is, we have $P(C) \leq \frac{1-\delta}{2}$.

Proof 3: Given that `ET` condition is not satisfied at search radius r, which implies that none of the objects within ball $B(q, R^+)$ are chosen as candidate. Now the algorithm is stopped by `NT` condition, which means there have already been βn candidate objects outside of the ball $B(q, R^+)$. According to the setting of P_E, m and the calculation of R^+, we have $R^+ > cR$. Thus, all these candidate objects are also outside of ball $B(q, cR)$.

Given an object outside of ball $B(q, cR)$, we use p_3 to denote the probability that o is chosen as a candidate (i.e., $\Pr(cn(o) \geq \alpha m)$) where $cn(o)$ is number of hits of the anchored bucket with width $2r$. According to the Chernoff bound, we have $p_3 < exp(-2(\alpha - p_2)^2 m)$ because our LSH function is (R, cR, p_1, p_2)-sensitive for any R according to Lemma 2. Let X denote the number of candidate objects outside of the ball $B(q, cR)$ (i.e., number of false positives). According to the Markov Inequality, we have $P_N = Pr(X \geq \beta n) \leq \frac{n \cdot p_3}{\beta n} = \frac{p_3}{\beta}$. Since $\frac{p_3}{\beta}$ can be calculated with given parameters, the probability of early terminate with βn candidates is bounded by $\frac{1-\delta}{2}$. Therefore, $P(C)$ is bounded by P_N because the probability of "visiting βn *bad* candidates and `ET` not satisfied" is bounded by that of "visiting βn candidates and `ET` not satisfied".

(5) Correctness of Algorithm. Based on the above analysis, the correctness of Algorithm 3 follows since $P(B) + P(C) \leq 1 - \delta$.

Theorem 1: When Algorithm 3 terminates with object o_{min}, we have $\| o_{min}, q \| \leq cR^*$ with probability at least δ, where R^* is the distance of NN object o^* to q in $\mathcal{R}^d$ space.

(6) Discussion of of P_B and P_C. We suppose P_B and P_C have a same value to simplify the proof, and in this section, we discuss about whether it is possible allowing P_B to be a larger value so that the early-termination condition can be more efficient. In figure 3.7, we can get the final success possibility $\delta = P_{ET} \cdot (1-\gamma) + P_{NT} \cdot \gamma$ where P_{ET}, P_{NT}, γ demonstrate the success possibility of ET ($P_{ET} = 1 - P_B$), the success possibility of NT ($P_{NT} = 1 - P_C$), and the possibility that the algorithm is terminated by NT separately. If we can prove that γ is a smaller value than 50% (the value we set in the proof above, which means ET and NT divide the false possibility equally), P_B could be set to a larger value and the total success possibility won't be changed. Here we discuss the bond of γ.

With the given p which effects the success possibility of the new early termination condition P_{ET}, we can compute the stop possibility of the normal termination (NT) γ as following:

When we are doing the c-ANN search, if the search is terminated by ET, it will satisfy that $R_{min} \leq \lambda r$ where $\lambda = \frac{c}{N^{-1}(\frac{p+1}{2})}$. Therefore, if it is stopped by the normal termination condition, we will have $R_{min} > \lambda r$ for all the candidate which have been assessed for at least αm times. Let p_{can} denotes the possibility that a point o, whose distance to q is slightly greater than λr, has been found as a candidate, we have $p_{can} = \sum_{i=\alpha m}^{m} C_m^i p^i (1-p)^{m-i}$. The normal termination condition will work if and only if all of the βn candidates found don't satisfy the new early-termination condition, which means all the candidates o have a distance to q larger than λr. It is easily to get γ, the possibility that NT terminates the algorithm, is $p_{can}^{\beta n}$.

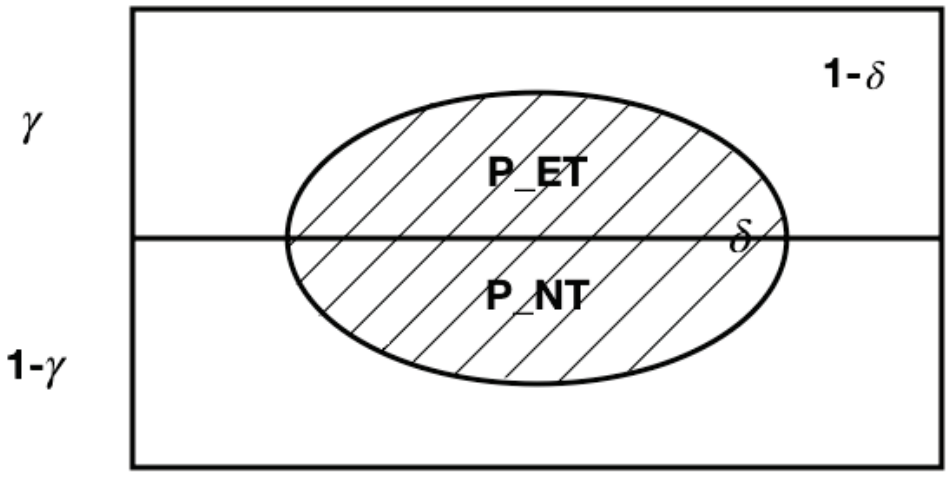


Figure 3.7 : Success possibility δ

Because the success possibility for normal termination condition only is at least

δ, which has bee proved before, we have

$$\begin{aligned}(1-\gamma)P_{ET}+\gamma\cdot\delta &\geq \delta \\ (1-\gamma)P_{ET} &\geq \delta-\gamma\cdot\delta \\ (1-\gamma)P_{ET} &\geq (1-\gamma)\delta\end{aligned} \tag{3.1}$$

Just as we mentioned above, the relation ship between p and P_{ET} is $P_{ET} = p_{can} = \sum_{i=\alpha m}^{m} C_m^i p^i (1-p)^{m-i}$, so from the above possibilities, because $0 < \gamma < 1$, we will get a total success probability greater than δ as long as choosing a proper p to have $P_{ET} > \delta$. The theoretical guarantee still holds.

Compared with the formal setting $P_E = \frac{1-\delta}{2}$, the new P'_E has a lower bound of $1 - P_{ET} \leq 1 - \delta$, which is a larger value of P_E. A greater P_E leas to a greater R^+, so the algorithm can stop earlier without breaking the theoretical guarantee.

Comparison with C2LSH and QALSH.

Although our technique follows the framework of C2LSH and QALSH, we show that the new early-termination framework and the incremental search strategy underpinned by the above analysis can bring us the following benefits.

(1) More promising early termination technique. C2LSH and QALSH use the traditional early termination condition where the algorithm can be terminated with $R_{min} \leq cR$. This decision is *safe* (i.e., deterministic) because it ensures to return a correct c-ANN in their algorithms, and hence they do not need to consider its correlation with normal termination condition. But the safe early termination condition also causes extra success possibility to the theoretical bond. Our strategy is much more aggressive as we have $R^+ > cR$, and hence can significantly reduce the number of candidate objects visited in practice, at the cost of a more complicated proof to handle correlation between two termination conditions.

(2) Better approximate ratio. As shown in [37], C2LSH and QALSH return a

c^2-approximate NN if the approximate ratio is set to c in their algorithms. This is because their bucket width grows to wc^k during the search where k is an integer, and $R_{min} \leq cR$ and $R \leq cR^*$ when their search algorithms terminate, where R^* is the NN distance and R is the corresponding ball radius in $\mathcal{R}^d$ for the search radius r. This implies that the value of c has to be set to $\sqrt{c}$ to achieve c-approximate in their algorithms. Thanks to our incremental search strategy and the novel early termination condition, Algorithm 3 and algorithm 2 can directly use c as the parameter to achieve c-approximate NN with success probability at least δ. Thus, under the same theoretical guarantee, the number of hashing functions required by EI-LSH is much smaller than that of C2LSH and QALSH.

3.6 Performance Studies

In this section, we conduct comprehensive experiments to demonstrate the I/O efficiency of our proposed algorithm, and compare the performance with two state-of-the-art I/O efficient c-ANN search algorithms using a variety of metrics.

3.6.1 Experiment Setup

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

Benchmark methods. SRS [82] and QALSH [37] are two state-of-the-art I/O efficient c-ANN search algorithms. In [37], QALSH has demonstrated superior I/O performance compared with C2LSH [28] and LSB-tree [83], hence we exclude C2LSH and LSB-tree from the performance evaluation. Besides, we also compare EI-LSH and I-LSH to show the effect of our early termination framework.

- `QALSH` uses a set of single LSH functions for c-ANN search. It requires m B-trees to store the index. The source code of QALSH is provided by the authors.

Note that, in their implementation, authors only keep the minimal hash value for each B+ tree leaf-node, together with all object IDs in the leaf-node. This can significantly reduce the index size.

- `SRS` has several variants. In the performance evaluation, we use the I/O efficient version with theoretical guarantee, where objects are indexed by an m-dimensional R-tree based on their hash values. The source code of SRS is public available at `https://github.com/DBWangGroupUNSW/SRS`.

- `I-LSH` is our incremental search LSH algorithm proposed in this thesis without the new early termination condition (Algorithm 2).

- `EI-LSH` is our incremental search LSH algorithm proposed in this thesis (Algorithm 3 in Section 3.4) using the new early termination condition.

Same as the previous work, 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. We use four million-scale datasets in the experiments, which are downloaded from a recent nearest neighbor search benchmark[‡]. Since all of the three methods are available to deal with real numbers, we didn't scale up the data.

- `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.

- `Glove` contains 1.1 million 100-dimensional word feature vectors extracted from Tweets.

[‡]`https://github.com/DBWangGroupUNSW/nns_benchmark/tree/master/data`

- `Sift` consists of 1 million 128-dimensional SIFT vectors
- `Audio` is a 192-dimensional dataset with 50000 points
- `Tiny_800m` is the full tiny dataset containing 800 million 384-dimensional vectors.

Evaluation Metrics. Our main focus is the I/O efficiency of the c-ANN algorithms under the same theoretical guarantee. Following the convention, 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 costs between random and sequential I/O. Recall that all I/O reads in SRS are random ones, while both random and sequential I/Os are invoked in QALSH and I-LSH. We also evaluate the accuracy of the search results by reporting the ratios of returned results compared with the ground truth answers.

Parameter Setting. To compare the methods fairly, the success possibility δ is set to $\frac{1}{2} - \frac{1}{e}$ for all algorithms. The default approximate ratio c is set to 4. Note that QALSH is c^2 approximate method and hence need to sets 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. Specifically, QALSH sets $\beta = \frac{100}{n}$, so at most $100 + k - 1$ candidate objects will be evaluated. As suggested, its initial bucket size is set to 2.72 and the number of required hash functions m is calculated accordingly. In SRS, we use a constant value of m for a given c, following its settings in [82]. For I-LSH, the parameters are set according to the specification of the parameter setting in Section 3.5. The page size B is set to $8,192$ bytes for all the datasets and algorithms.

All of the experiments were conducted on a PC with intel(R) Xeon(R) CPU E3-1231 v3 with 3.04GHz, 8 cores and 16G main memory. All of the source codes were

implemented in C++ and compiled with g++ -with -O3 flag. We evaluate three algorithms by studying the performance after averaging the 200 queries. To compare the performance fairly, we use the same random vectors to project the datasets.

3.6.2 Evaluate Index Size

We list the index size for all the datasets we have used in experiments under the default settings (e.g., $c = 4$ and $\delta = \frac{1}{2} - \frac{1}{e}$). Table 3.2 shows the index sizes for QALSH,I-LSH and SRS and the number of hash function (m) used for the desired theoretical guarantee. It is reported that SRS uses the smallest size of index. This is because a small m value is required and SRS only keeps one data entry for each object. I-LSH uses much less number of hash functions compared with QALSH, mainly because QALSH is a c^2-approximation algorithm. Index sizes of the three methods are independent of the dimensionality d, and grow linearly with the dataset size n.

Table 3.2 : Index size

dataset (d)	Glove (100)	Sift (128)	Tiny (384)	Million Song (420)	Audio (192)
m	84	83	93	83	60
QALSH (MB)	444	394	2150	391	32
m	16	16	16	16	16
I-LSH (MB)	165	149	735	157	12
m	6	6	6	6	6
SRS (MB)	46	43	254	46	2.4

3.6.3 Evaluate Index Building time

Another important evaluation matrix is the efficiency to build indexes for the LSH methods. Although the indexes are only built once for each dataset, if building index takes too much time compared with other algorithms, it can also be a shortage. To compare the algorithms fairly, all the three algorithms read data from text files instead of binary files. We list the index building time for all the datasets we have used in the experiments under the default settings. Table 3.3 shows the indexing time cost for QALSH, SRS and I-LSH. For list-based LSH algorithms, the index time is mainly dependent on the number of hash functions. According to our proof, I-LSH needs less hash functions than QALSH to hold the same theoretical guarantee, and building procedures of the two algorithms are similar. So for all the datasets, I-LSH only cost about 25% time to build index compared with QALSH. For the tree-based LSH methods, there is some different reasons to be considered. Although m is less than 10 in SRS , the algorithm has to do the insertions on a m-dimensional tree, which is a more comprehensive operation compared with insertion on a B+ tree. SRS has the smallest size of index, but the slowest index building speed.

Table 3.3 : Index time

dataset	tiny	Audio	Glove	Million Song	Sift
SRS(s)	1777	9.42	568	421	490
QALSH(s)	386	2.71	67	153	56
I-LSH (s)	89	1.1	13	30	11

3.6.4 Evaluate I/O costs

In this section, we evaluate the goodness of an I/O efficient c-ANN algorithm and the most essential criteria is to use as less I/O costs as possible under the same theoretical guarantee for the given approximate ratio c and success probability δ. Three algorithms have rigorous theoretical analysis to ensure the desired theoretical guarantee, and it turns out that all queries in the experiments are answered **correctly** regarding the given approximate ratio c. Fig 3.8 reports the I/O costs of three algorithms on four datasets where k varies from 1 to 100 and the approximate ratio c is set to 4 (default value) or 2 respectively. Besides, to prove that our new early termination framework can decrease the I/O cost dramatically, we also conduct experiments for EI-LSH and I-LSH only to demonstrate the results more clearly because both of the two methods use a much smaller I/O compared with SRS and QALSH. From figure 3.8, we have the following observation.

- I-LSH has the best I/O performance on four datasets under all settings, especially on the million-scale and billion-scale datasets.

- When $c = 4$ (i.e., 4-ANN), 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. The performance gap between SRS and I-LSH is not very significant on Sift data where the dimensionality is 128. On Tiny and Million song datasets with dimensionality 384 and 420, respectively, I-LSH outperforms SRS by a big margin.

- When the approximate ratio c is set to 2, it is not surprising that the higher demand of the accuracy leads to much more expensive I/O costs for three algorithms. As shown in Fig.6 (c)-(f), SRS is more sensitive to the decrease of

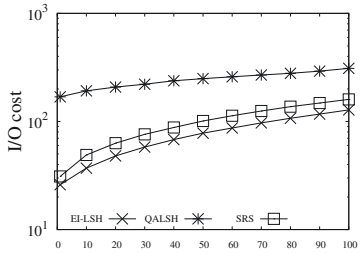
c (i.e., increase of the accuracy required) compared with QALSH and I-LSH. Although SRS consistently outperforms QALSH when $c = 4$, it consumes more I/O costs when $c = 2$. This is because the performance of exact NN search on R-tree is very sensitive to the dimensionality, and m of SRS is set to 15 for $c = 2$. Although the number of B+ trees also increases to a large number, e.g., $m = 60$ and $m = 351$ for I-LSH and QALSH respectively on Tiny dataset, their performance is less sensitive to the increase of approximate ratio compared to SRS.

- When k increases, the I/O costs also grow steadily for three algorithms. It is noticed that, k doesn't significantly affect the I/O costs when the dataset size is large, because the dominant I/O costs of three algorithms on large dataset are to find the first candidate.

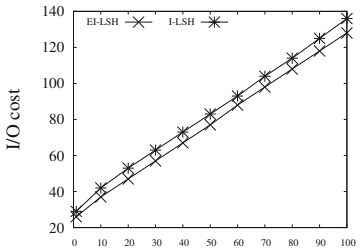
- When only considering I-LSH and EI-LSH, I-LSH can reduce the I/O cost on all the datasets under the same theoretical guarantee because it has a more strict bond of the success possibility. If k grows to a large value, the I/O cost difference between I-LSH and EI-LSH becomes more obvious.

3.6.5 Evaluate Accuracy

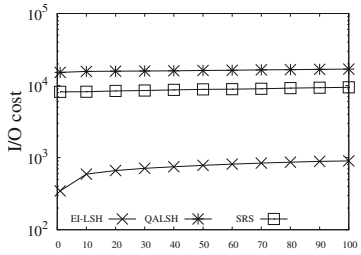
Our experiments show that all three algorithms are quite pessimistic. Given a very low success probability $\delta = \frac{1}{2} - \frac{1}{e} = 0.132$, all queries of three algorithms are answered correctly with ratio smaller than 2.0 for the required ratio $c = 4.0$. Given the fact that all of the three algorithms rely on the same hash function, and the key differences are the search strategy and the indexing method. Suppose the same number of hash functions are deployed for three algorithms. Intuitively, the more candidate objects accessed by an algorithm, the better ratio it should be able to achieve given a reasonable good search heuristic. The key challenge is how to stop



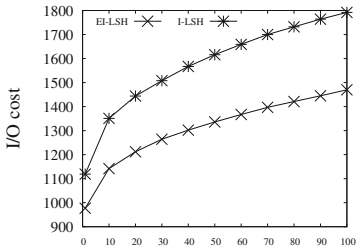
(a) Audio



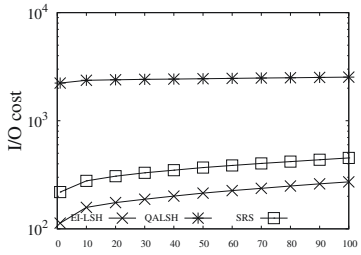
(b) Audio, EI-LSH vs. I-LSH



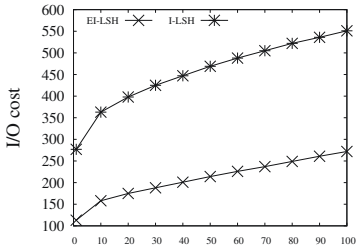
(c) Tiny



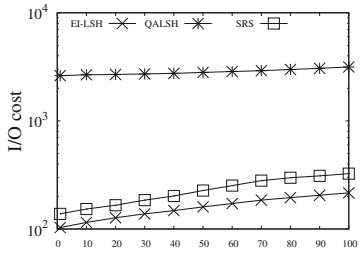
(d) Tiny, EI-LSH vs. I-LSH



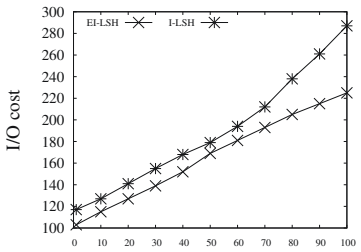
(e) Glove



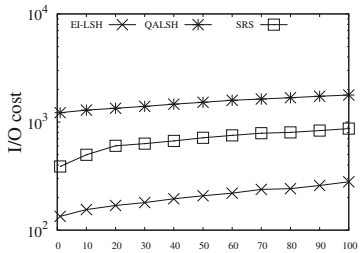
(f) Glove, EI-LSH vs. I-LSH



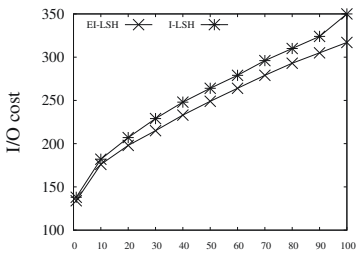
(g) Sift



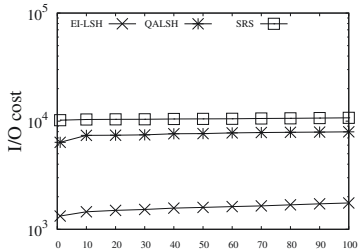
(h) Sift, EI-LSH vs. I-LSH



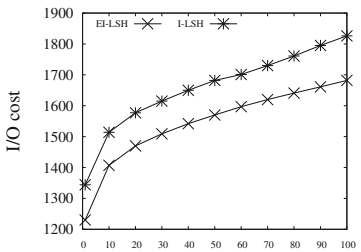
(i) MillionSong



(j) MillionSong, EI-LSH vs. I-LSH



(k) Sift, $c = 2$



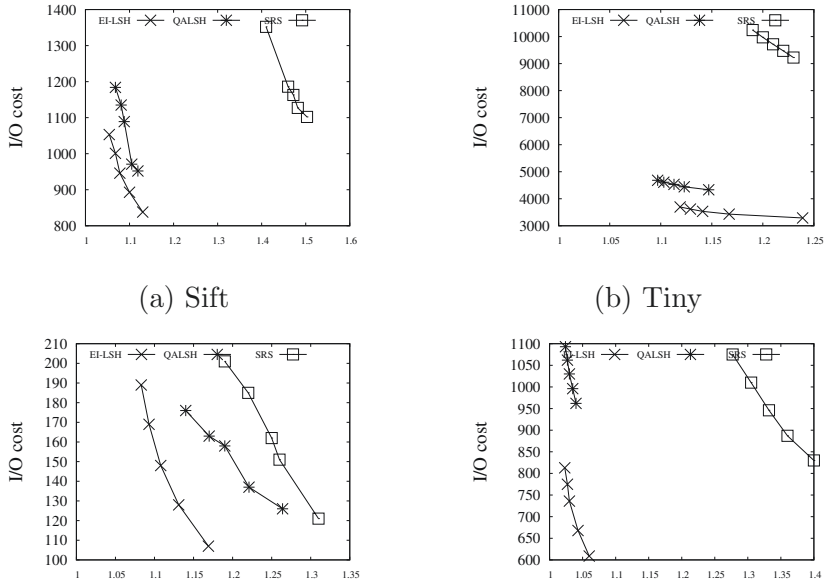
(l) Sift, $c = 2$, EI-LSH vs. I-LSH

as early as possible under the theoretical guarantee. As expected, given the same c and δ, the ratio of QALSH is much better than that of SRS and I-LSH because QALSH terminates much later.

Instead of just reporting the ratio vs. k, in Fig 3.9, we report the trade-off between I/O costs and average approximate ratio of the queries for three algorithms on Sift, Tiny, Audio and MillionSong datasets by enforcing the algorithm to explore some objects (i.e., more I/O costs incurred) even the early termination condition has been satisfied. It is shown that, under the same I/O costs, I-LSH can achieve lower ratio. Similarly, under the same ratio, I-LSH uses much less I/O costs. The performance of SRS is not competitive in this set of experiments mainly because of the penalty of random I/O assesses and the R+ tree property. Another observation from this set of experiments is that, SRS is harder to return a high quality result even it is forced to visit more points. From the figures, the ratio decreases slowly when the I/O cost increasing. If there is a requirement of accuracy of the result, SRS could not well support it.

3.6.6 Effect of the approximate ratio c

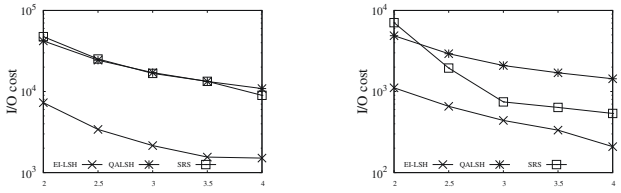
A good c-ANN algorithm should support different approximate ratio, especially when c is a small value. We conduct experiments on Tiny and Million Song datasets where c varying from 2.0 to 4.0, and evaluating the I/O costs. The results are reported in Fig 3.10. Same as Section 3.6.4, the difference between EI-LSH and I-LSH are shown separately. As expected, the performance of three algorithms degrades against the decrease of the approximate ratio. It is shown that the performance of SRS is most sensitive to the change of the approximate ratio c. This is because the higher requirement of accuracy leads to a larger m, and hence deteriorates the performance of the exact NN search on R-tree in SRS.



(a) Sift (b) Tiny

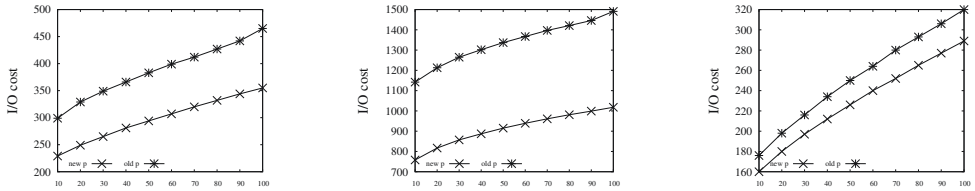
(c) Audio (d) MillionSong

Figure 3.9 : I/O costs vs. ratio



(a) Tiny (diff. c) (b) MillionSong (diff. c)

Figure 3.10 : I/O costs vs. c



(a) Glove (diff. P_{ET}) (b) Tiny (diff. P_{ET}) (c) MillionSong (diff. P_{ET})

Figure 3.11 : I/O costs vs. P_{ET}

3.6.7 Effect of the value of P_{ET}

As we discussed in Section 3.5, as long as the value of P_{ET} is greater than δ, the theoretical guarantee will still hold. We conduct experiments to illustrate the effect of adopting a stricter theoretical bond to the I/O cost. In the default setting, P_{ET} is set to be $1 - \frac{1-\delta}{2} = 1/2 + \delta/2$ in I-LSH and in this experiment, $P_{ET} - new$ is set to be $\delta + 0.001$. When $c = 4$, λ should be 8.7 and 22.3, separately. The results are reported in Fig 3.11. The 'old p' line demonstrates the former setting of P_{ET} and the 'new p' line means the new setting of P_{ET}. The effect of P_{ET} is more tremendous on a higher dimensional dataset or on a larger dataset. It is because that on a larger dataset or a higher dimensional dataset there is less possibility to find k-c-ANN within k candidates only and therefore, a small P_{ET} can terminate the algorithm earlier and reduce the I/O cost. On the million-scale datasets, the tiny dataset is the largest one and shows the most obvious difference between the two P_{ET}. For any k in the figure, the I/O cost of new p is at least 400 less than the old p.

3.6.8 Large dataset

Another important feature of a c-ANN algorithm is the scalability. We run our algorithm on the large dataset tiny with different scales to evaluate the performance on non-trivial dataset. Since QALSH's indexing time cost is unfordable, we only

compare SRS and our algorithm.

The following table shows the index size of I-LSH and SRS for different size of the tiny dataset.

Table 3.4 : Index size

size of dataset (million)	10	30	50	80
m	16	16	16	16
`I-LSH (GB)`	1.3	3.8	6.4	10.1
m	6	6	6	6
`SRS (GB)`	0.34	1.1	1.9	3

We change the dataset size from 10 million to 80 million to show the I/O performance of our algorithm. The value of k is set to 50, and all the other settings are same with other experiments. To better shows the I/O efficiency of our ET framework and our incremental search strategy, we use the most strict bond of the success possibility where $P_{ET} = \delta + 0.001$. Under this setting, in most cases, the first k candidates found by I-LSH could satisfy the early-termination condition. Figure 3.12 shows the I/O cost for different size of the tiny dataset. When the size grows to 80 million, the I/O cost of I-LSH to solve the c-k-ANN problem is less than 6000, which is a tiny value considering the huge dataset size. Comparing with SRS, I-LSH use less I/O over all the experiments.

3.6.9 Summary

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

- Under the same theoretical guarantee, I/O performance of EI-LSH consistently outperforms QALSH and SRS under all settings. This is because: (1) EI-LSH

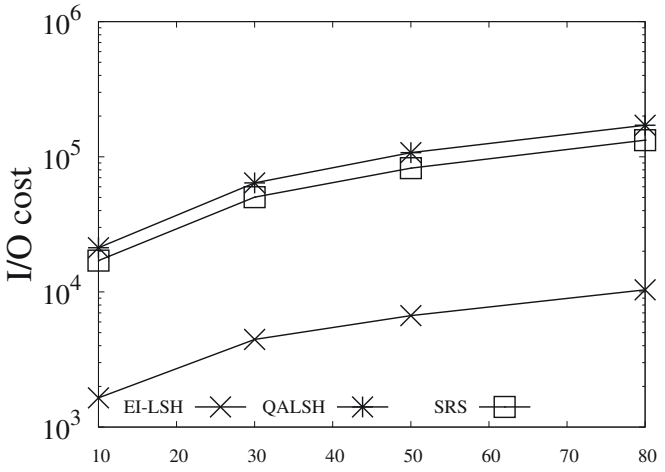


Figure 3.12 : I/O cost on difference dataset size ($k = 50$)

adopts a natural incremental search strategy, which enables us to achieve c-approximation (unlike the c^2 approximation of QALSH); (2) EI-LSH uses a much aggressive early termination technique, which can significantly reduce I/O costs without sacrificing the theoretical guarantee; and (3) EI-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 search on the multi-dimensional index including R-tree. Moreover, the random I/O incurred by R-tree is also a limit of SRS.

- The performance of QALSH is not competitive on the I/O efficiency under the same theoretical guarantee. This is mainly because: (1) the c^2 approximate algorithm resulting from its bucket expansion search strategy; and (2) the conservative early termination strategy may miss many opportunities to stop the algorithm without breaking the theoretical guarantee.

- I-LSH and EI-LSH require a same size of index and same parameter settings to hold a given theoretical guarantee, but EI-LSH adopts a more aggressive

early-termination condition and can stop earlier than I-LSH.

- The value of P_{ET} has a dramatical effect on the I/O efficiency of EI-LSH when the dataset is enormous. Because in most cases, our algorithm is stopped by the early-termination condition, and on extremely large dataset a smaller P_{ET} can reduce the candidate set size.

3.7 Conclusion

To deal with the fundamental similarity search on large scale high dimensional data, in this thesis we investigate the problem of c-ANN search and propose an aggressive early-termination condition, which can be used on most list-based LSH algorithms to reduce the I/O cost. Besides, we also develop an I/O efficient incremental LSH method, namely I-LSH. We show that, by applying a natural incremental search strategy and the new early termination technique, I-LSH can significantly enhance the I/O efficiency under the same theoretical guarantee. By using the more aggressive early-termination strategy, EI-LSH can improve I/O performance further compared with I-LSH. 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. The experiment on large scale dataset shows that our algorithm also performs well on non-trivial datasets.

Chapter 4

Approximate Maximum Inner Product Search

4.1 Overview

In this chapter, we select a set of state-of-the-art Approximate MIPS algorithms and conduct comprehensive experiments to evaluate their performance in terms of both accuracy and efficiency. First we introduce the background of this problem in section 4.2, then we briefly introduce the algorithms are category them into two categories in section 4.4 and section 4.3. The experiment studies are given in section 4.5.

4.2 Background

In this section, we first define MIPS problem formally, and describe the scope of algorithms we selected in this paper, then we introduce how we divide the algorithms into three groups and the idea of each group.

4.2.1 Problem definition

Given a dataset D with n points in d-dimensional space, inner product of two points o_1 and o_2 is defined as:

Definition 3: **Inner Product**

Let $\langle o_1, o_2 \rangle$ denotes the inner product of points o_1 and o_2,

$$\langle o_1, o_2 \rangle = \|o_1\|\|o_2\| cos\beta$$

where $\|o\| = \sqrt{\sum_{i=1}^{d} o_i^2}$ means the l_2 norm of point o, and β is the angle between o_1 and o_2.

The Maximum Inner Product is defined as follows.

Definition 4: **Maximum Inner Product Search**

Given a query point q, the maximum inner product of q is the point $o^* \in D$ which satisfies $\langle o^*, q\rangle > \langle o, q\rangle$ where o is any point also in dataset D.

Considering the time and space consumption, most MIPS algorithm aim to find the approximate solution of this problem. To measure the result accuracy, recall, ratio and precision are commonly adopted in many papers.

4.2.2 Algorithm scope

MIPS is a fundamental problem and plays an important role in various areas such as database, recommendation systems and computer visions. There are numerous papers published to propose a novel improvement of current solution from different aspects. It is impossible, and also unfair to compare all of the algorithms together. So in this thesis, we only choose several state-of-the-art algorithms which also satisfies the following requirements.

No dependence on hardware. It is a popular implementation to use GPU or multiple CPUs, multiple threads to speed up similarity search but not all of the algorithm use such techniques. In this thesis, we disabled the hardware-related optimisation for all algorithms to do a fair comparing.

In-memory algorithm. Similarity search in high-dimensional space can be done in internal memory or external memory. For internal memory algorithms, running time is the most important part of algorithm efficiency while for external memory algorithms, the primary goal is to reduce I/O cost. Since I/O operations cost much more time than CPU operation, in this thesis, we only compare algorithms based on internal memory, and use running time to evaluate the efficiency.

Exact Maximum Inner Product as the ground truth. Some methods require

a label for each data point in the input, and also use the label to evaluate result accuracy (especially in some recommendation system papers). Since we hope to use a same set of datasets in our experiments, we only use algorithms which use exact k maximum inner product points as the ground truth.

4.2.3 Categories

The algorithms are divided into two categories: Approximate MIPS algorithm with theoretical guarantee and without theoretical guarantee. Some of the algorithms are not designed for MIPS specifically, just like Annoy [25], which supports several distance such as Euclidean distance, angular distance and so on.

The algorithms with theoretical guarantee are all based on LSH scheme. LSH is short for Locality-Sensitive Hashing, a well-known solution for kNN search in high-dimensional space. LSH-based algorithms transform the whole data space to a new one so that kNN algorithm could work. Due to the property of LSH, most LSH-based algorithms provide a theoretical guarantee for results, which will be explained in next section. This category contains [38, 66, 80, 93].

The other category contains graph-based MIPS algorithms and Tree-based MIPS algorithms, which map the dataset to a graph or divide the dataset to sub-spaces, then use graph related algorithm or tree related algorithm to achieve query processing. ip-nsw [64], ip-nsw+ [54] and Annoy [25] all belong to this group.

4.3 Approximate MIPS algorithms with theoretical guarantee

Theoretical guarantee means that all algorithms belong this category could provide a guarantee for their result accuracy. Norm-ranging LSH [93] guarantees that the result will be greater than cS_0 with a possibility greater than $1 - \delta$ where c and δ are constants smaller than 1 and S_0 is a pre-defined parameter. The other

algorithms provide a guarantee that the result will greater than cR^* with a constant possibility, where R^* is the real MIPS of query q.

The state-of-the-art Approximate MIPS algorithm with theoretical guarantee all adopt LSH scheme. Locality-Sensitive Hashing is used in approximate nearest neighbour search in high-dimensional space. It uses a set of hash functions to map high-dimensional data objects to a low-dimensional vector and keeps distance relationship among points to some extent. In this section, we first describe some basic information of LSH, then introduce how to use LSH to solve MIPS problem, and the details of algorithms belonging to this category.

4.3.1 Locality-Sensitive Hashing

Locality-Sensitive Hashing is first proposed by Indyk [40] in 1998 to solve c-ANN problem in binary hamming space, and extended to Euclidean space by Datar et al. [22]. The most significant part of LSH is LSH function.

Definition 5: **LSH function**

Given a Locality-Sensitive hash function $h(\cdot)$, it is (r_1, r_2, p_1, p_2)-sensitive if and only if for any two points x and y, there exists two distance thresholds r_1 and r_2 and two possibility thresholds p_1 and p_2, which satisfy:

$$\begin{cases} \Pr[h(x) = h(y)] \geq p_1 & \text{, if } f(x, y) < r_1 \\ \Pr[h(x) = h(y)] \leq p_2 & \text{, if } f(x, y) > r_2 \end{cases}$$

From definition 5, if the distance between any two points o_1 and o_2 increases, the possibility of $h(o_1) = h(o_2)$ will decrease accordingly. In Euclidean space, the Gaussian/normal distribution, which is also called 2-stable distribution, is used to generate LSH functions.

4.3.2 From LSH to MIPS

LSH can well support kNN search in high-dimensional space, however, inner product is not a metric, LSH can't be simply applied on MIPS problem because there is no symmetric nor asymmetric LSH in the entire space [66]. A feasible solution is to convert MIPS to NN search by transforming the whole dataset and query set: $D : \mathcal{R}^d \rightarrow \mathcal{R}^{d'}$, and $Q : \mathcal{R}^d \rightarrow \mathcal{R}^{d'}$ where $d' > d$. After transforming the two sets into new data space, using LSH to do kNN search and the result will be returned as MIPS result.

There are two errors of using LSH to solve MIPS problem. The first appears when converting dataset and query set to a new data space. Most asymmetric transformations require to add a large constant to the Euclidean distance $\|D(o) - Q(q)\|$ after transformation, which will cause distortion error for the next step NN search. Suppose in a dense dataset D and a query point q, if every point $o \in D$ has a very small distance to q, after adding the large constant in transformation, every point can be NN of q. It is obvious that the final result can be extremely bad.

The second error happens when processing NN search. LSH is able to offer a theoretical guarantee of its result, which means, given an approximation ratio c, LSH guarantees that the result can be c-NN of query q with a constant possibility regardless of the data distribution, but the recall is not always very competitive comparing with other NN methods.

As a result, the primary challenge of LSH-based MIPS algorithm is to minimise the two errors meanwhile try to achieve a good performance on time consumption. We select H2-ALSH [38], L2-ALSH [79], norm-range LSH [93] and Sign-ALSH [80] in the thesis. Except for norm-range LSH, all the other algorithms provide theoretical guarantee to the result.

4.3.3 H2ALSH

H2ALSH is the most recent MIPS algorithm with theoretical guarantee, which was published in 2018 by Qiang H et. al. It put forwards a novel transformation method to reduce the error occurred in data space converting and doesn't need any assumptions on dataset and query set. Based on their previous NN paper QALSH [37], they use QALSH to retrieve top-k nearest neighbour in the new data space, meanwhile keeps the theoretical guarantee.

H2-ALSH utilized a new asymmetric transformation method, which is called Query Normalized First (QNF) transformation, to reduce Approximate MIPS problem to c-ANN problem. The vector transformations of dataset D and query set Q are defined as follows:

$$D'(o) = [o_1, o_2, ..., o_d, \sqrt{M^2 - \|o\|^2}]$$

$$Q'(q) = [\lambda q_1, \lambda q_2, ..., \lambda q_d, 0], \text{where } \lambda = \frac{M}{\|q\|} \text{ and } M = max_{o \in D}\|o\|$$

The QNF transformation maps the d-dimensional dataset and query set to a $d+1$-dimensional data space without transformation error because $\lambda = \frac{M}{\|q\|}$ is a fixed constant.

There are two steps in H2-ALSH. The first step is to pre-process the dataset. H2-ALSH first sort the data points in increasing order according to their norms $\|o\|$ and then divides them into different sets, then compute M_i for each set and applies QNF to transform the whole dataset to $\mathcal{R}^{d+1}$ space. In the new data space, QALSH is used to build indexes for each subset if there are a large amount of points in the subset, which consists of multiple B+ trees.

The next step of H2-ALSH is query processing. It computes an upper-bond ub for each subset according to M_i and $\|q\|$, which means the norm of q. If ub is greater than a pre-defined threshold, q will be mapped to $\mathcal{R}^{d+1}$ space and use QALSH to

find kNN for this subset, or linear scan will be used on the subset when the subset size is small, which means it doesn't need a index in the first step. When every subset has been precessed, the union set of all results is MIPS result.

4.3.4 L2-ALSH

L2-ALSH is another important LSH-based MIPS algorithm, which was published on NIPS in 2014. The key idea of L2-ALSH is that using different hash functions in pre-processing step and querying step but keep LSH's property, unlike the classical LSH solving NNS problem generally requires using a same hash function in the two steps.

There are two assumptions for L2-ALSH on dataset and query set:

- All of the data points are located in a unit sphere, which means $\|o\| \leq U < 1$ satisfied for all points in D, where U is a constant value less than 1.
- The norms of all query points in Q are 1.

The assumptions are very strict so that In most real-world dataset, they can't always hold, then we have to rescale the dataset D or query set Q. ALSH transforms dataset and query set from $\mathcal{R}^d$ to $\mathcal{R}^{d+m}$:

$$D(o) = [o, \|o\|^2, \|o\|^4, ..., \|o\|^{2^m}]$$

$$Q(q) = [q, 1/2, 1/2, ..., 1/2]$$

After transformation, LSH and its variants could be applied for c-NN search. In L2-ALSH, the most classical LSH was used [22]. There are L hash tables used in LSH, and each hash table contains k LSH functions. For every hash function, all of the data points are divided into different hash buckets according to their hash values. In query processing step, if a point o is in all of the k hash buckets with

q in a hash table, the o can be a candidate of MIPS. The algorithms will look for enough candidates in the L tables and then return top-k of those candidates.

4.3.5 Sign-ALSH

Bothe of H2-ALSH and L2-ALSH are converting MIPS problem into Nearest Neighbour search problem, but Sign-ALSH is different. Sign-ALSH is the improvement version of L2-ALSH, but it uses signed random projection to convert approximate MIPS problem into correlation similarity problem. The two assumptions for L2-ALSH are still required in Sign-ALSH. This algorithms also adopts asymmetric transformation on dataset $\mathcal{R}^d \rightarrow \mathcal{R}^{d+m}$and query set $\mathcal{R}^d \rightarrow \mathcal{R}^{d+m}$ which is defined as follows:

$$D(o) = [o_1, o_2, ..., o_d, 1/2 - \|o\|^2, 1/2 - \|o\|^4, ..., 1/2 - \|o\|^{2^m}]$$

$$Q(q) = [q_1, q_2, ..., q_d, 0, ..., 0]$$

4.3.6 Norm-range LSH

Norm-range LSH is an improvement of simple-LSH. It is also based on simple-LSH space transformation, but divide data points into disjoint sub-datasets thus reducing the distortion error in the data transformation. There are also two steps in norm-range LSH: index and query processing. Query processing is similar to other LSH-based algorithms, here we mainly introduce index building step.

In the index building step, norm-range LSH first partitions dataset D into m subsets and then builds index for each sub-dataset $\mathcal{S}_j$ after normalising each sub-dataset using $U_| = max_{x \in S_j}\|x\|$. Unlike the three algorithms above, norm-range LSH uses a symmetric transformation to transform MIPS to angular similarity search problem as follows:

$$P(x) = [x, \sqrt{1 - \|x\|^2}]$$

$$P(q)^{\mathsf{T}}P(x) = [q, 0]^{\mathsf{T}}[x, \sqrt{1 - \|x\|^2}] = q^{\mathsf{T}}x$$

The querying step, Norm-ranging LSH conducts MIPS using multi-probe on every sub-dataset then selects the largest candidates among all the sub-datasets as Approximate MIPS result.

4.4 Approximate MIPS algorithms without theoretical guarantee

4.4.1 Graph-based algorithms

Navigable small world graph [49, 13] is network with a diameter which is dramatically smaller than its size and is bounded by a polynomial in $logN$ where N is the graph size (number of vertexes in graph). In a Navigable Small World graph, in most cases, any two nodes can always be connected by a very short path. It has been used to solve k-approximate nearest neighbour search problem [59, 60]. A NSW graph is built by continuously inserting nodes randomly and connecting them to their m nearest neighbours in current graph using bidirectionally edge. An advantage of NSW is that the construction procedure can be parallelized without losing accuracy. However, the performance of NSW is not stable. In some low-dimensional dataset, it will degrade and lose to tree-based algorithms.

In this section, we introduce three representative NSW-based algorithms: ip-NSW and ip-NSW$^+$.

ip-NSW

ip-NSW is based on the authors' previous work HNSW [62] so first we briefly introduce HNSW.

HNSW improves performance of NSW by separating the links according to their length scale into multiple layers, and then applies searching in a multi-layer graph.

The algorithm starts from upper-layer (with longer links) until reaching a local minimum, and then it goes to a lower layer to continue searching from the current node. It is achievable to limit every node's maximum connection in all the layers to be a constant, therefore the complexity of routing is logarithmic. The key idea of HNSW is very similar to another well-known data structure called probabilistic skip list structure [74].

Index construction. The index of HNSW is a multi-layer graph, whose lower layers contain more points and higher layers have fewer points. The ground layer includes all the data points in a dataset. HNSW adopts a similar method to insert elements with NSW. For each data point, first HNSW computes its layer $l = \lfloor -ln(unif(0..1)) \cdot m_L \rfloor$ where m_L is a pre-defined constant and $unif(0..1)$ means randomly choosing a value from a uniform distribution, then the new point is added to all the layers from level $l+1$ layer down to level 0. There are two phases of the insertation. First, from layer $l+1$ down to layer L, where L is the level of current enter point, and also is the top level of current graph, the algorithm only search for the closet point to the new point in each layer by greedily traversing the graph, which will be used as the next enter point. The second phase is from layer L down to ground layer. The algorithm requires M nearest neighbours of the new point on each layer, where M is a value affected by a pre-defined parameter ef, and connects them with the new point. The major difference between this two phases is that ef is always 1 in the first phase.

Query processing. Compared with index building, searching is much simpler. The searching procedure is basically same as inserting a new point into the graph with $l = 0$. Unlike LSH-based algorithms, HNSW doesn't provide theoretical guarantee to result quality, but it support multiple distance metrics such as Hamming distance, Euclidean distance, angular distance and so on.

ip-NSW ip-NSW extends HNSW to support inner product search. It defines a novel kind graph called s-Delaunay graph for a similarity function $s(x, y)$.

Definition 6: A s-Delaunay graph $G_s(V, E)$ is a graph whose vertices set V is the dataset X and every edge (i, j) in edges set E connecting two vertices i and j if the corresponding s-Voronoi cells R_i and R_j are adjacent.

Definition 7: Given a dataset X, the s-Voronoi cell $R - i$ is a set

$$R_i = x \in R^d | s(x, x_i) > s(x, x_j) \forall i \neq j$$

where x_i and x_j are all elements in set X, and $s(x, y)$ means the distance between x and y regardless of the distance metric.

Since it is infeasible to use exact s-Delaunay graph to solve MIPS problem in high-dimensional space, ip-NSW uses approximate s-Delaunay graph and processes queries using the method proposed in HNSW [58].

***ip-NSW*$^+$**

ip-NSW$^+$ analysed the reason that why ip-NSW has a remarkable performance, and then further improves ip-NSW by introducing a novel angular proximity graph. In ip-NSW$^+$, the author found that there is a strong norm bias in ip-NSW which can avoid calculation on small norm items. However, when the elements with large norm bias are not MIPS result, ip-NSW will waste a large amount of time to compute distance of these elements. ip-NSW$^+$ solves this problem by proposing a new rule: the MIPS neighbor of an angular neighbor is probably a MIPS neighbor, instead of the old one adopted in ip-NSW: the MIPS neighbor of another MIPS neighbor is likely to be a MIPS neighbor. The index building and query processing procedure are given as follows.

Index Construction.ip-NSW$^+$ builds two graphs A_s and G_s, where A_s is an angular NSW graph and G_s is an inner product NSW graph, and the points are inserted

into both of the two graphs in a random order. For each point o, it is first inserted into the angular graph and then o is inserted into G_s, and the neighbors of o in the inner product graph will be found using query processing procedure.

Query processing. Given a point q, first, ip-NSW$^+$ finds top-k' angular neighbors of point q and then for each of these neighbors v, the algorithm add every point connected with v in G_s into a candidate set C. Then, the algorithm adopts the classical graph walk in NSW with set C to return top-k inner product neighbor of q.

4.4.2 Tree-based algorithms

Another method to solve Approximate MIPS problem is tree-based space partition method, which is also popular in nearest neighbour search. In this section we introduce an algorithm used in commercial recommendation system which is called Annoy [25] and it has been used in spotify.com.

Index construction. Annoy builds index by conducting multiple hierarchical k-means trees where $k = 2$ in this algorithm. Each tree is constructed by recursively dividing the data points into two parts. The following is detail procedure of data partition. For each iteration:

- Annoy executes a clustering algorithm on a set of samples from input data set, and then find two centres (since it is 2-means tree).
- Partition the input data points into two parts and each part is a sub-tree of the current node. Each centre can define a hyperplane according to distance and the hyperplane is used to decide which sub-tree a point should belong to.

After recursively construction of each sub-tree, the index construction is completed.

Query processing. The query processing is achieved by traveling each 2-means tree from root. Every root is pushed to a maximum priority queue with a very

large value to ensure root can always be first popped out from the queue. At each iteration, if q belongs to node n_i, Annoy calculates a key for n_i which equals to the minimum of n_i's parent's key value and the distance to the hyperplane. If q doesn't fall in node n_i, the key value of n_i will be the minimum of n_i's parent's key value, and the distance to the hyperplane times -1. Annoy always select the node with larger key value to continue searching.

4.5 Experiments

In this section, we conduct a set of experiments to evaluate all algorithms mentioned above.

4.5.1 Experiment settings

Datasets and query sets

We conduct experiments on 6 real-world datasets which have been generally used on existing works covered a wide range of applications including text, video and image. All fo the datasets are in high-dimensional space where dimensionality is greater than 100. Below, we introduces the datasets which are used in the experiments.

- Audio contains about 0.05 million 192-dimensional audio feature vectors from DARPA TIMIT dataset.
- Enron is collected from a set of emails. The vectors have 1369 dimensions.
- Glove is a dataset extracted from Twitter, which contains 1.2 million points in 100-dimensional space.
- Sift has 1 million SIFT vectors in 128-dimensional space.
- MillionSong is another audio feature vector set which consists of about one million music tracks in 420 dimensions.

- Tiny5M is a sample of Tiny80M dataset which contains visual descriptors of the tiny images. The dimensionality of Tiny is 384.

For each dataset, we randomly choose 100 points from it as the query point and average their performance in the final report. The value of k is varied from 1 to 100 with default value 40.

Benchmarks

We choose 9 representative MIPS algorithms from two categories. All the source code are got from the author's website. The algorithms are all implemented in C++. To fairly compare them, we made some necessary changes such as multi-thread technique on the code to get a fair result.

MIPS method with theoretical guarantee. This type contains all LSH-based algorithms including H2-ALSH, L2-ALSH, Norm-range LSH and sign-ALSH. The unique point of these algorithms is they all provide a theoretical guarantee to the result quality, which means, given a approximate ratio c, they guarantee that the result is a c-maximum inner product of the query point q.

MIPS method without theoretical guarantee. This type contains 2 graph-based methods, ip-NSW, ip-NEW$^+$ and one tree-based method Annoy. The algorithms in this category generally are faster than algorithms with theoretical guarantee, but the performance is affected by data distribution.

In our experiments, all C++ source codes are compiled using G++ 4.7 in Linux environment with Intel Xeon 8 core CPU at 2.9GHz and 128G memory.

4.5.2 Evaluation Metrics

The result quality is measured by the following metics:

- Running time-k. Since all methods are internal memory algorithms, running

time can well represents efficiency of an algorithm.

- Recall-k. Recall is used to evaluate the overall quality of results.
- Recall-Running time. It is evident that visiting more points can have a better recall performance but costing longer time, so compared recall-running time can well reflect which algorithm has a good performance on both accuracy and efficiency.
- Recall-speedup. Since exact MIPS result can be got by running brute-force linear scan algorithm, speedup of an algorithm A is defined as $\frac{T_{BF}}{T_A}$ where T_{BF} represents time consumption of brute-force algorithm and T_A means average running time of algorithm A. Comparing recall-speedup is a fair method to evaluate efficiency and accuracy from another aspect: time.
- Fixed Recall-Speedup. Given a specific recall value, we compare the speedup of each algorithm over various datasets.
- Index size. All of the algorithms required to construct indexes before query processing, therefore index size should also be considered.
- Index time.

All reported results are averaged over all 100 query points in the query sets by running multiple times.

4.5.3 Parameter settings.

We adopted default settings of each algorithm:

- Annoy. The number of trees m to build index is set to 100.
- ip-NSW and ip-NSW$^+$. M is set to 32 and ef is set to 100.

- H2-ALSH. The approximation ratio c is set to 0.5.
- L2-ALSH, Sign-ALSH. U is set to 0.83
- Norm-ranging LSH. Hash code length is set to 16.

4.5.4 Comparison within each category

In this section, we compare algorithm using evaluation metrics given in section 4.5.2 within each category only. The purpose of this round of evaluation is to select the algorithm in this group with best performance as representative to be compared in the second round.

MIPS algorithm with theoretical guarantee

Table 4.1 and table 4.2 describe the index size and index time of each algorithm. Norm-ranging LSH and H2-ALSH have a smalleer index size and index building time compared with L2-ALSH and Sign-ALSH. The reason is that H2-ALSH only conduct index for sub-dataset whose size is greater than a given threshold so that the index size can be reduced dramatically. All of the algorithms are implemented in memory so the index time is basically increased when index size is growing up. So H2-ALSH is also the method with shortest index building time. And the index size of Norm-ranging LSH depends on the hash code length which is set to 16 for all datasets.

Figure 4.1 shows recall-k of these LSH-based algorithms where k is varied from 20 to 100. From the figure, Norm-range LSH and H2-ALSH behave much better than the others. The majority reason is that the data space transformation of H2-ALSH doesn't cause distortion error, and in query-processing step, if the sub-dataset only contains a few points, H2-ALSH directly adopts linear search on the set, which leads to a higher accuracy result. Norm-ranging LSH also doesn't cause

Table 4.1 : Index size

dataset (d))	Sift (128)	Enron (1396)	Song (420)	Audio(192)	Glove(100)	Tiny(384)
H2-ALSH (MB)	11.39	0.6	10.96	0.61	12.47	57
L2-ALSH (MB)	316	24	313	13	352	1773
Sign-ALSH (MB)	1953	185	1933	106	2148	1324
Norm-range LSH (MB)	3.82	0.38	3.82	0.19	4.58	19.1

too much distortion error when transforming data since it divides the whole dataset into sub-dataset as well.

Figure 4.2 gives information of running time-k. Considering efficiency only, it is difficult to choose the best method. L2-ALSH is the worst algorithm on almost all the datasets, except Audio. H2-ALSH usually has short running time when k is small, but the running time increases dramatically when k rises. Norm-range LSH has a quite stable running time consumption over all the six datasets but it is not the best one on most datasets. Sign-LSH also doesn't cost more time when k grows, but it is the worst algorithm on Audio dataset. Overall, to evaluate efficiency, running time vs. k is not convinced enough.

From the above two figures, it is difficult to simply choose the most efficient algorithm, and H2-ALSH and norm-ranging LSH achieve high accuracy, so we choose norm-ranging LSH and H2-ALSH to compare their recall-running time, and the results are shown in figure 4.3. In norm-range LSH, we change the probed item

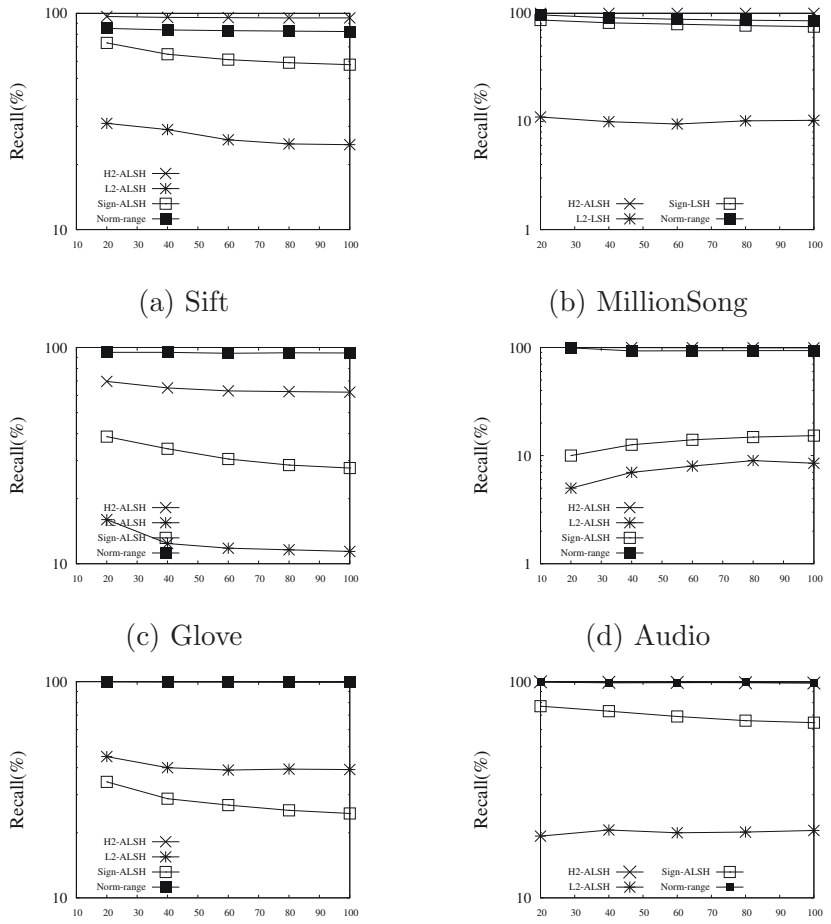
Table 4.2 : Index time

dataset (d))	Sift (128)	Enron (1396)	Song (420)	Audio(192)	Glove(100)	Tiny(384)
`H2-ALSH (s)`	15.73	5.62	31.74	0.9272	12.85	168
`L2-ALSH (s)`	28.68	12.29	62.58	1.34	27.62	366
`Sign-ALSH (s)`	68.26	75	231.49	5.66	56.72	1046
`Norm-range LSH (s)`	3.47	2.5	8.23	0.25	2.56	45.2

number to get different recall and running time, and in H2-ALSH, we change the candidate set size upper bound βn in QALSH to get the results. k is set to 20 in this experiment. From the figure, except for Audio dataset, norm-ranging LSH always reach a higher recall with less time compared with H2ALSH. So norm-range LSH is selected as the best algorithm within this category to be compared in the next round.

MIPS algorithm without theoretical guarantee

This category contains ip-NSW, ip-NSW$^+$ and Annoy. Table 4.4 and 4.3 compares the index size and index construction time of these algorithms. Index construction time of ip-NSW and ip-NSW$^+$ is affected by dataset. ip-NSW$^+$ requires one more graph in its index but each single graph is smaller than ip-NSW. On sift and glove datasets, ip-NSW$^+$ costs more time to build index but on the others, ip-NSW is faster. Comparing index size,



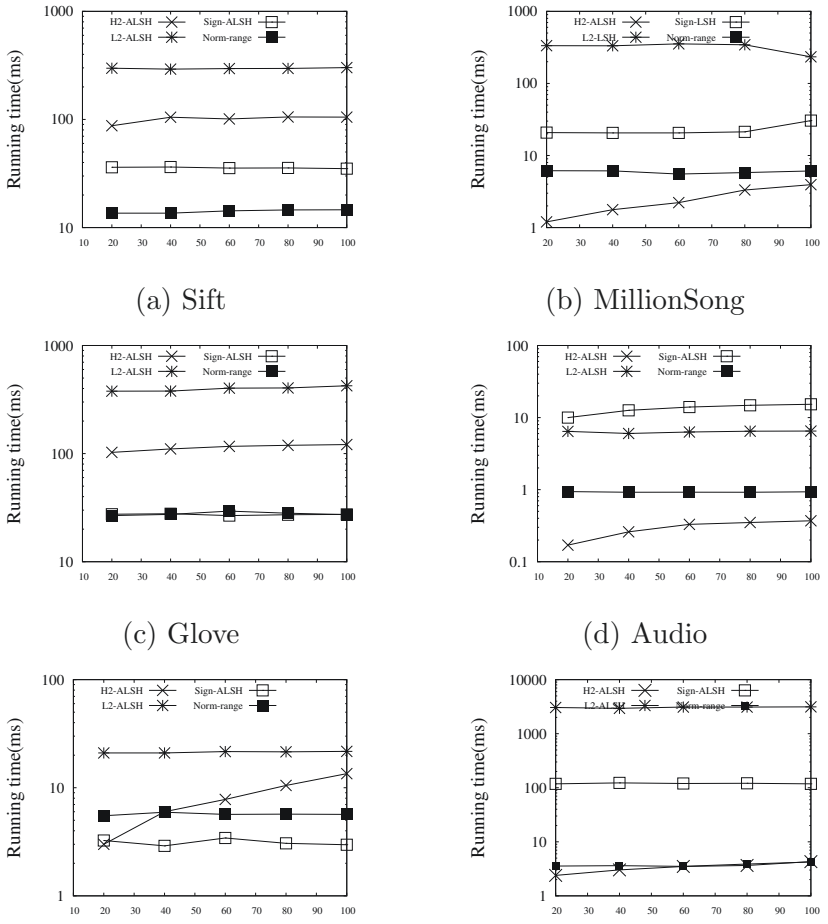
(a) Sift (b) MillionSong

(c) Glove (d) Audio

(e) Enron (f) Tiny

Figure 4.1 : recall varying k

Figure 4.4 shows running time vs. k of these three algorithms. Since running time is actually affected by visited item number but not k, we set the expected visited item number as k+100 and show the tendency. It is quite clear that ip-NSW$^+$ costs least time among these three methods and Annoy is the slowest one. Annoy cost even more than 10 times of running time compared with ip-NSW and ip-NSW$^+$. Since Annoy is tree-based algorithm and it partitioned data into sub-spaces, when dimensionality is a large value, time consumption of Annoy will increase fast. ip-NSW$^+$ is specifically designed for Approximate MIPS problem based on NSW graph so that it has the best performance. Only on Audio dataset, ip-NSW has a similar performance with ip-NSW$^+$ since audio is a small dataset with lower dimensionality.



(a) Sift (b) MillionSong

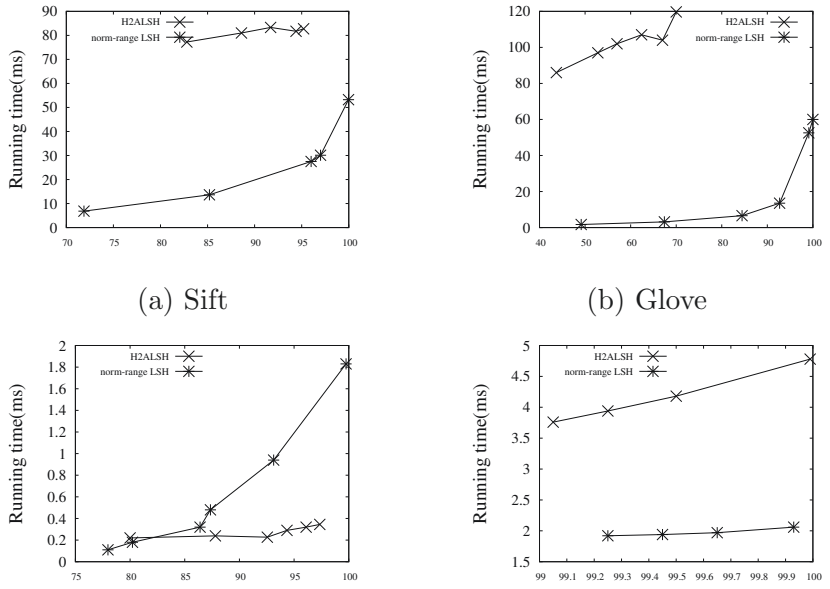
(c) Glove (d) Audio

(e) Enron (f) Tiny

Figure 4.2 : running time varying k

The running time difference between ip-NSW and ip-NSW$^+$ becomes larger then dataset size is increasing or dimensionality is a large value.

Figure 4.5 is the result of recall vs. k experiment. ip-NSW$^+$ also beat Annoy and ip-NSW on this evaluation metric. It is obvious that Annoy is not a good choice for MIPS problem. On some easy dataset such as Audio, recall of Annoy is always below 10% while ip-nsw and ip-nsw$^+$ can both hit at leat 90%. On million-scale datasets, Annoy can be even worse. It totally cannot compete with the other two methods. Over the six datasets, ip-NSW has a comparable recall with ip-NSW$^+$ when $k = 20$, but it drops down dramatically when k becomes larger. It shows that



(a) Sift (b) Glove

(c) Audio (d) Enron

Figure 4.3 : running time vs. recall

ip-NSW requires to check more points than ip-NSW$^+$ to reach similar recall.

Since time vs. k cannot well present which algorithm can perform a higher recall in a shorter time, we conduct experiments of recall vs. running time for ip-nsw and ip-nsw$^+$. It has been proved above that Annoy is not competitive with these two algorithms, we don't consider it in this part. Figure 4.6 shows the result. $k = 20$ in this experiment.

From the figure, we notice that to return a higher quality result, ip-nsw$^+$ doesn't have to visit more elements because its running time doesn't increase very sharply, while ip-nsw has to sacrifice more time consumption to achieve a more accurate result. On million song dataset and tiny dataset, ip-nsw even can't reach ip-nsw$^+$'s recall when it takes more than 5 times of running time. So ip-NSW$^+$ could return a higher quality Approximate MIPS result using less time.

Among the three algorithms, ip-NSW$^+$ is the best one considering running time

Table 4.3 : Index time

dataset (d))	Sift (128)	Enron (1396)	Song (420)	Audio(192)	Glove(100)	Tiny(384)
`ip-NSW`$^+$ `(s)`	444	67.69	342	18.22	766	3126
`ip-NSW (s)`	25.6	660.492	1502.67	59.29	207	10692
`Annoy (s)`	1953	185	36688	257.44	2148	561231

and recall. So it is chosen as the representative to be evaluated in the next round.

4.5.5 Second round comparing

In this section, we compare the two best algorithms from the two categories above: Norm-range LSH and ip-NSW$^+$. Norm-range LSH represents Approximate MIPS algorithms based on LSH-scheme and ip-NSW$^+$ is the best one based on NSW graph. In this section, we evaluate the two algorithms more comprehensively.

Recall vs. Running Time Generally, checking more points means higher time consumption and better quality of result. This experiment is conducted to show which method can achieve higher recall with less time. We set $k = 40$ and let the ip-NSW$^+$ and norm-range LSH visit different size of points to compare their running time and recall, all the other settings are using the default setting provided in their papers. Fig 4.7 shows that norm-ranging LSH requires longer time to attain same recall as ip-nsw$^+$. The recall-running time curse shows that norm-ranging LSH's recall increases sharply from 10% to about 70%, but to reach more than 90% recall, the running time is exponentially growing.

Table 4.4 : Index size

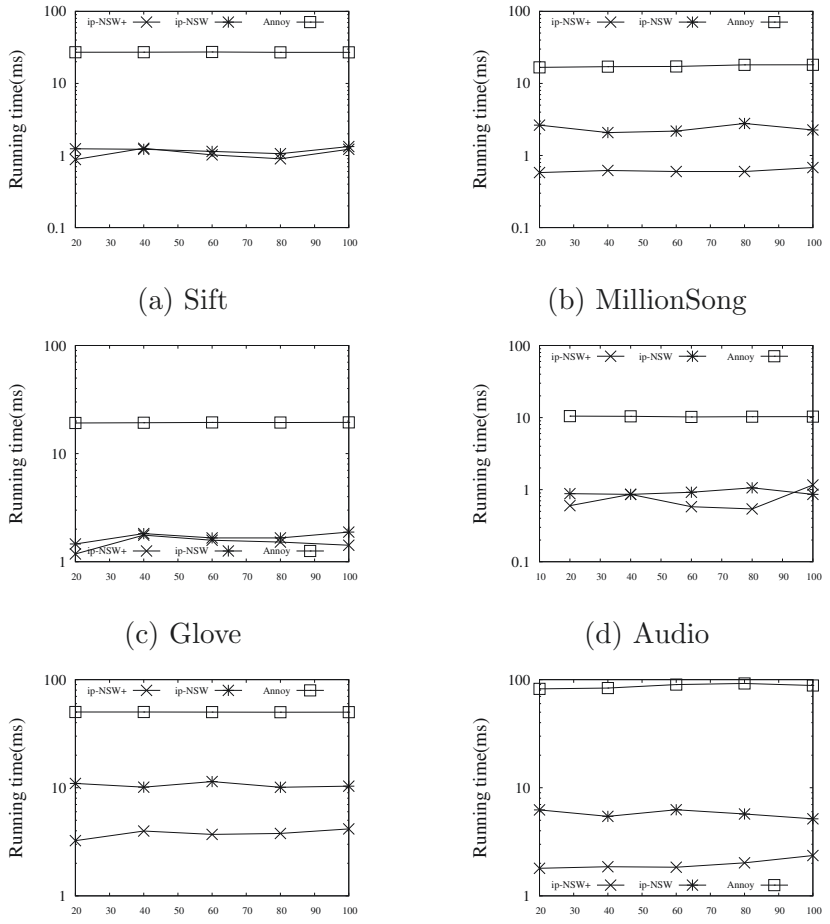
dataset (d))	Sift (128)	Enron (1396)	Song (420)	Audio(192)	Glove(100)	Tiny(384)
`ip-NSW`$^+$ `(MB)`	873.4	555.3	2048	61.3	837.6	10002
`ip-NSW` `(MB)`	788.3	547.2	1945.6	59.29	743.9	9318
`Annoy` `(MB)`	14950	2764	23859	301.9	3122	26502

Recall vs. Speedup

Speedup can well represent an algorithm's efficiency. From figure 4.8, speedup of norm-ranging is able to reach more than 100 on sift, glove and enron, but as a trade-off, the corresponding recall is very poor, which is even less than 10%. ip-NSW$^+$ stably achieves a recall higher than 90% on sift, million song, enron, and higher than 60% on glove, while its speedup is also competitive. Overall, ip-nsw$^+$ has a dramatically higher recall than norm-ranging LSH when speedup is same.

Speedup vs. fixed recall value

Given a recall which is greater than 95%, $k = 40$, figure 4.9 compares the speedup of norm-ranging LSH and ip-nsw$^+$ over 4 datasets. norm-ranging LSH apperantly has a lower speedup compared with ip-NSW$^+$. Except for enron, speedup of ip-NSW$^+$ is almost more than ten times higher than speedup of norm-ranging LSH.



(a) Sift (b) MillionSong

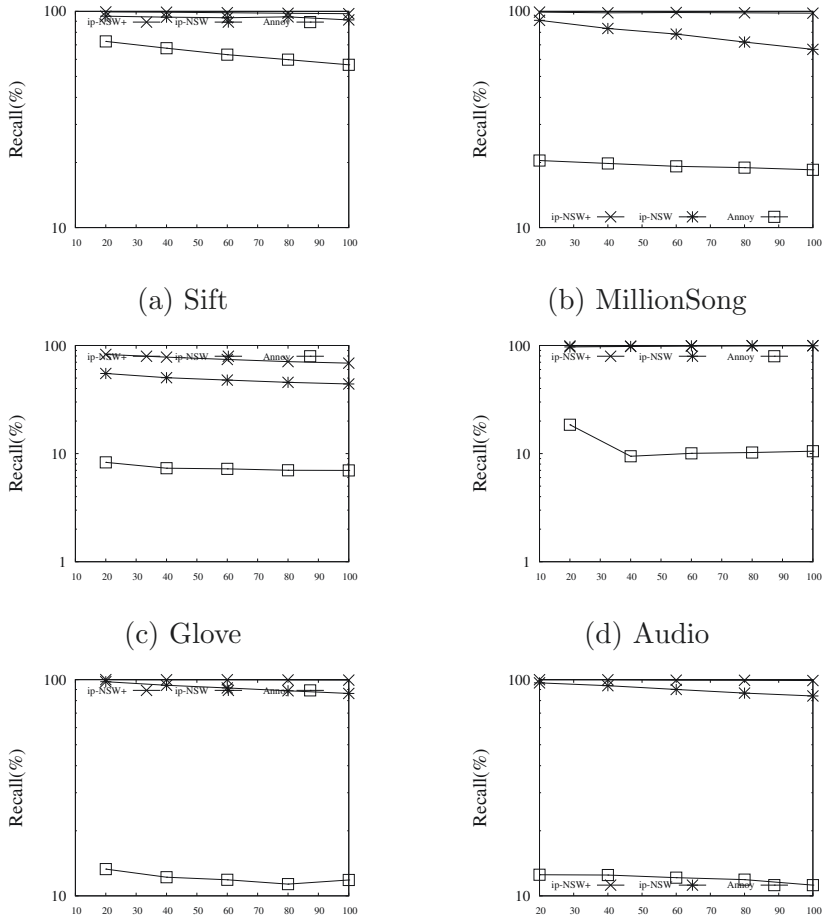
(c) Glove (d) Audio

(e) Enron (f) Tiny

Figure 4.4 : running time varying k

4.5.6 Conclusion

According to the experiments, LSH-based methods can provide a guarantee of its result quality but the trade-off is longer processing time, and graph-based algorithms generally require a larger index size and longer index construction time, but after building index, the query processing step.is much faster and the candidate accuracy is pretty good although theoretically the result is possible to be much smaller than real MIPS. Annoy, a tree-based nearest neighbour search algorithm which support multiple distance metrics, doesn't have a good performance in the experiments. Recall of Annoy is substantially lower than all the other algorithms chosen in this



(a) Sift (b) MillionSong

(c) Glove (d) Audio

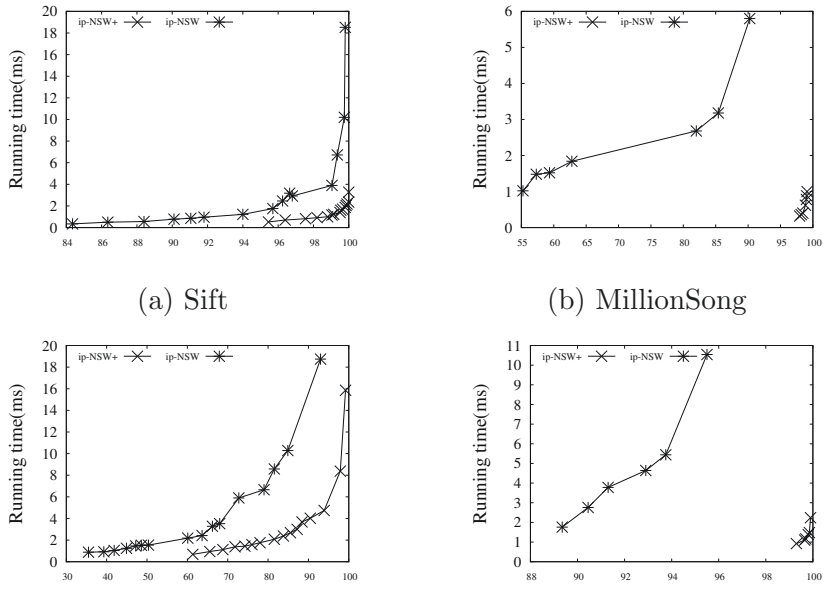
(e) Enron (f) Tiny

Figure 4.5 : recall varying k

thesis but Annoy is a popular solution for approximate nearest neighbour problem, which shows that it is not a suitable algorithm for Approximate MIPS problem. ip-NSW$^+$ is the best algorithm considering query processing step, while its index size is larger than other algorithms except for Annoy.

Below are some suggestions according to our experimental observations.

- If resources including both CPU and memory are enough for off-line index construction, ip-NSW$^+$ is the best choice for Approximate MIPS problem. After building two graphs, ip-NSW$^+$ can quickly process Approximate MIPS query and its recall is nearly to 100% on most datasets. It also supports multi-



(a) Sift (b) MillionSong

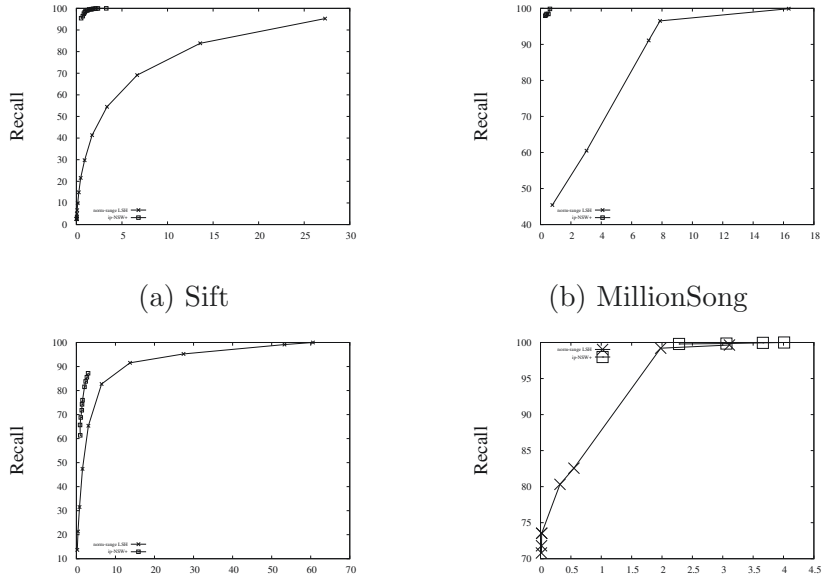
(c) Glove (d) Tiny

Figure 4.6 : recall vs. time

thread in index construction step, so the indexing procedure can be even faster if CPU resource is sufficient.

- If a guarantee of accuracy is required, norm-ranging LSH and H2-ALSH are both considerable. These two algorithm both offer a kind of theoretical guarantee for their result quality. Norm-ranging LSH costs a longer index construction time, but it processes queries faster and can reach higher recall with shorter time, while H2-ALSH provides a more strict theoretical guarantee. It guarantees that the result is always no less than cr^* where r^* is the real maximum inner product to the query with a constant possibility.

- Annoy shouldn't be considered as a Approximate MIPS algorithm. It takes the longest index building time, and longest query processing time, but achieves the worst recall over all the datasets.



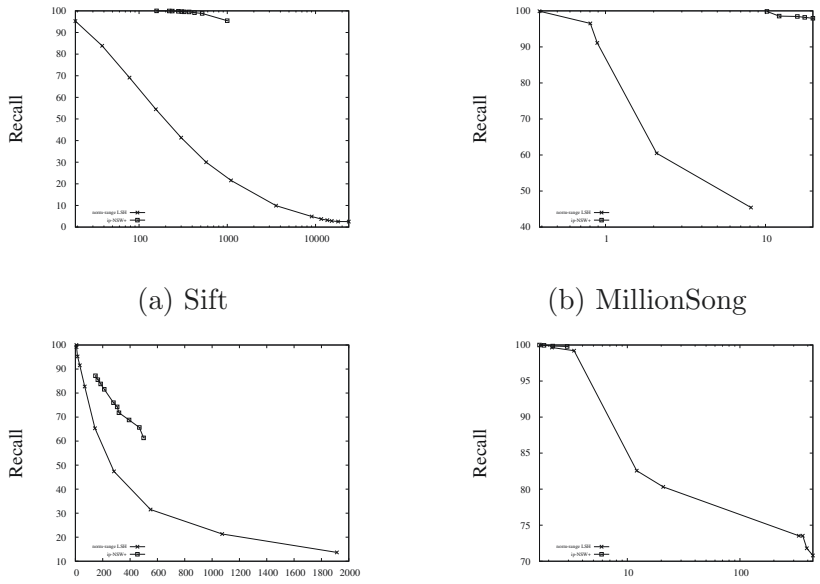
(a) Sift (b) MillionSong

(c) Glove (d) Enron

Figure 4.7 : recall vs. time

4.6 Summary

MIPS is a fundamental and important query processing problem and has been widely used. It is not practical to ask for exact maximum inner product in real scenarios, and the common solution is to try to retrieve k Approximate MIPS by visiting a limited number of points. In this thesis, we selected several state-of-the-art Approximate MIPS algorithms to evaluate their performance comprehensively and also gave recommendations for users.



(a) Sift (b) MillionSong

(c) Glove (d) Enron

Figure 4.8 : recall vs. speedup

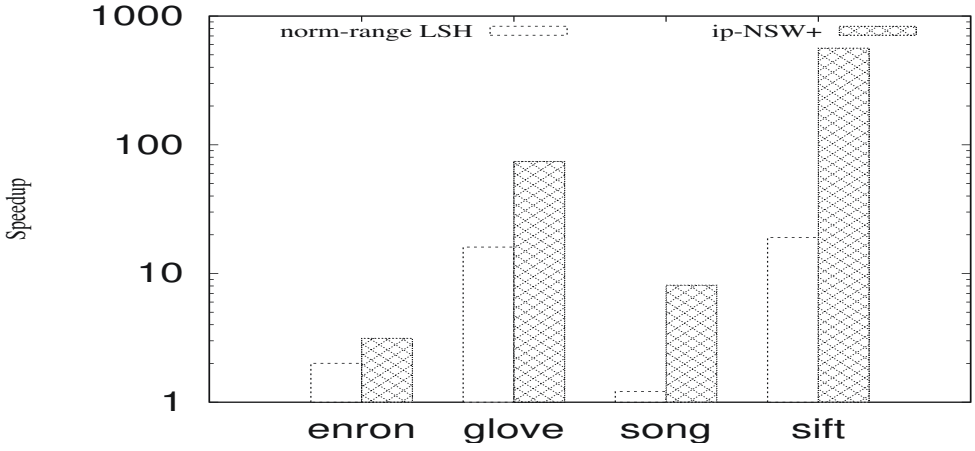


Figure 4.9 : Recall $\geq$ 95%

Chapter 5

Approximate Furthest Neighbour Search

5.1 Overview

In this chapter, we introduce an I/O efficient algorithm RI-LSH to solve c-AFN problem in high-dimensional space. This work has been accepted by DASFAA 2020. The rest of this chapter is organized as follows. First we give preliminaries in section 5.3, then introduce our approach in section 5.4. Detailed analysis and experimental results are shown in section 5.5 and section 5.6, respectively.

5.2 Motivation

Locality Sensitive Hashing is also a possible solution for c-AFN problem. As we mentioned above, LSH can map high-dimensional data into a low-dimensional space and likely keep the distance relationship among data points. Using such property, we can also design an algorithm for c-AFN search by modifying LSH to make it suitable for c-AFN problem. The projection distance on a LSH function is the most important information to reflect the distance between two points in the original space. In most LSH schemes, there is a "bucket" with a certain width. If two points o_1 and o_2 fall in the same bucket, we say o_1 and o_2 collide over this function. In the NN problem, the number of collisions of a point o and query point reflects the possibility of o to be a NN points. As an opposite problem of Nearest Neighbor, it is easy to get a conclusion: if o_1 and o_2 don't collide in a LSH function, the Euclidean distance between o_1 and o_2 is likely to be a large value in the original space.

Table 5.1 : Summary of Notations

Notation	**Definition**
n	the number of point objects in the dataset
d	the dimensionality of the dataset
m	the number of hash functions (projected dimensions)
q	the query object
$\parallel o_1, o_2 \parallel$	the Euclidean distance between o_1 and o_2
o^*, R^*	the FN object of q, with distance R^*
o_{max}, R_{max}	c-AFN object returned, with distance R_{max}
c and δ	approximate ratio and success probability
w	the initial bucket width in LSH functions
R and r	$R = \frac{2r}{w}$, radius of ball $B(q, R)$ in $\mathcal{R}^d$ space regarding search radius r

5.3 Preliminary

In this section, we present the formal definition of the problem. The important notations used throughout the thesis are summarized in Table 5.1.

5.3.1 Problem Definition

Given a d-dimensional dataset D with n points, donated by R^d, where d is a large number (e.g., $d \geq 100$), each point o in the dataset has d coordinate values. The coordinate value of o on the i_{th} dimension is denoted as $o[i]$. For a query point q, the Euclidean distance between o and q is denoted by $\parallel o, q \parallel = \sqrt{\sum_{i=1}^{d}(o[i] - q[i])^2}$. The c-approximate furthest neighbor is defined as follows:

Definition 8: **c-approximate furthest neighbor (c-AFN)**. Given a query object q and a d-dimensional dataset D and a furthest neighbor o^* of q whose distance

to q is R^*, a c-approximate furthest neighbor of q is a point $o \in D$ which satisfied $\| o, q \| \geq \frac{R^*}{c}$, where c is the approximation ratio ($c > 1$).

Note that the theoretical guarantee is independent to the distributions of the dataset D and the query q. Thus, our proposed algorithm can guarantee the success probability upon arbitrary data points and query points distributions.

Problem Statement. In this thesis, we focus on proposing an efficient algorithm to solve the c-AFN problem in a high-dimensional Euclidean space with theoretical guarantee, which means given the approximation ratio c, a query point q and the probabilistic threshold δ, the algorithm will return a c-AFN of q with probability at least δ.

5.3.2 LSH family for furthest neighbor

Our algorithm is designed on the LSH scheme, since LSH scheme has already be introduced in chapter 3, in this section, we only describe the modification of LSH for c-AFN problem.

To solve the c-AFN problem, the inequaltion of LSH should be modified:

$$\begin{cases} \Pr_{H\in\mathcal{H}}[H(x) \textit{ separates with } H(y)] \geq p_1 & \text{, if } f(x,y) \geq r_1 \\ \Pr_{H\in\mathcal{H}}[H(x) \textit{ separates with } H(y)] \leq p_2 & \text{, if } f(x,y) \leq r_2 \end{cases}$$

. The definition of "separate" is given in definition 9.

Definition 9: **Separate** Given a constant value w and a query point o, if the projection distance of o and q is larger than $\frac{w}{2}$ ($\| H(o), H(q) \| \geq \frac{w}{2}$), then we say o and q are separated on $H(\cdot)$.

In Figure 5.1, q is the query points and there are two bucket widths R_1 and R_2. For bucket R_1, all the objects are separated with q, while for bucket R_2, all the points are collided with q.

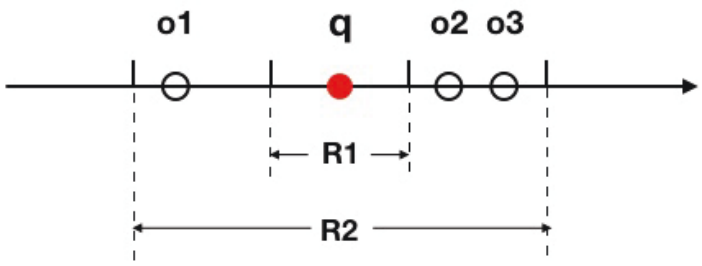


Figure 5.1 : Example for separation

5.3.3 $(c, 1, p_1, p_2)$-sensitive Reverse LSH

Similar as RQALSH, a reverse LSH function can be formally represented by $H_{\vec{a}}(o) = \vec{a} \cdot \vec{o}$ where o is the d-dimensional object and a is a d-dimensional vector for the random projection, whose elements are following the p-stable distribution. To solve the Euclidean space problem, p is set to 2, e.g., the normal distribution.

Let $s = \| o, q \|$, $r_1 = c$ and $r_2 = 1$. We can compute p_1 and p_2. According to the property of p-stable distribution, $(\vec{a} \cdot \vec{o} - \vec{a} \cdot \vec{q})$ has the same distribution with sX where X is a random variable chosen from the normal distribution $\mathcal{N}(0, 1)$. Let $\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, the Probability Density Function (PDF) of $\mathcal{N}(0, 1)$, then the possibility that o and q are separated is:

$$p(s) = 2 \int_{-\infty}^{-\frac{w}{2s}} \phi(x)dx = 1 - \int_{-\frac{w}{2s}}^{\frac{w}{2s}} \phi(x)dx$$

When $r_1 = c$ and $r_2 = 1$, we have $p_1 = p(c)$ and $p_2 = p(1)$. Because $p(s) = 2norm(-\frac{w}{2s})$ where $norm(x)$ is the CDF of $\mathcal{N}(0, 1)$ and is a monotonic increasing function of x, and the value of $-\frac{w}{2s}$ increases when s increases. Thus, $p(s)$ increases monotonically as s increases. So the reverse LSH function is $(c, 1, p_1, p_2)$-sensitive.

5.4 Approach

In this section, we present our RI-LSH algorithm in detail. First, in Section 5.4.1, we briefly introduce our motivation and then introduce our algorithm in Section 5.4.2. Section 5.4.3 shows that our algorithm can also be easily extended to solve top-k c-approximate furthest neighbor problem (c-k-AFN).

5.4.1 Motivation

From section 5.3 we can get that for two points o_1, o_2 and a reverse LSH function h_i, the projection distance $\parallel h_i(o_1), h_i(o_2) \parallel$ reflects the Euclidean distance between o_1 and o_2. If o has a far projection distance on most hash functions, o is likely to be a c-AFN of q. Most LSH based algorithms used a "bucket" to determine the original distance relationship between two points. To expand the radius, the algorithm using buckets has to involve all the points falling into the bucket at one time. While in our method, the key idea is to continuously reduce the radius R and only one point on a single hash function is involved in each iteration.

For example, in Figure 5.2, there are two hash functions h_1 and h_2, and q is located at the origin, i.e. the hash value of q on h_1 and h_2 are both 0. The point who has the longest projection distance to q on h_1 is o_4, and $\parallel h_1(o_4), h_1(q) \parallel = 3$. So point o_4 will be accessed first, and then the algorithm will visit the next furthest point. o_4 on h_2 is the next point to visit, because $\parallel h_2(o_4), h_2(q) \parallel = 3$ and all the other points have a smaller projection distance. The searching area is a square with a decreasing size when there are two hash functions. The points will be accessed one by one according to their projection distances on the functions.

To boost the success possibility, we need a set of reverse LSH functions. Suppose there are m hash functions, and a point o separated with q on all of the functions, o is a c-AFN of q with a high probability. But if we set the separation threshold as

m, it is too strict to have enough candidates. In our scheme, we use a parameter α ($p_2 < \alpha < p_1$). When a point o is separated with q for at least αm times, it could be far from o with a high enough probability. Following the traditional LSH scheme, o will be added to the candidate set S. Only the candidates in S will be calculated its real distance to q. We keep set S as a limited size to guarantee that the I/O performance is much better than linear search.

5.4.2 Approach

In this subsection, we describe our algorithm in details and give the pseudo-code. The setting of parameters will be given in Section 5.5. There are two steps of RI-LSH: indexing and querying. In the indexing step, we project the whole d-dimensional dataset into a m-dimensional space, and store the hash values in B+ trees. In this step, the query point is still unknown. In the querying step, the query point comes. The algorithm searches in the projection space to find the c-AFN result.

Indexing. We adopt the similar method in chapter 3 to build index for RI-LSH algorithm. The difference is, we store min_i and max_i for the minimum hash value and maximum hash value for each hash function.

Query processing. When the query point q comes, we first project q to m hash functions and insert the hash values of q into B+ trees, then the points in the dataset and their hash values will be accessed according to their projection distance to q on the m projected dimensions. Since we are looking for c-AFN of q, the algorithm starts from the furthest point over the m projection dimensions and decreases the projection distance continuously. For example, in Figure 5.2, $m = 2$, and the furthest projection distance to q is $\| h_1(o_1), h_1(q) \|$. So o_1 will be accessed first, and then the second furthest projection distance to q is $\| h_2(o_2), h_2(q) \|$. o_2 is the second object to be visited. A point will be considered as a candidate if it has been accessed

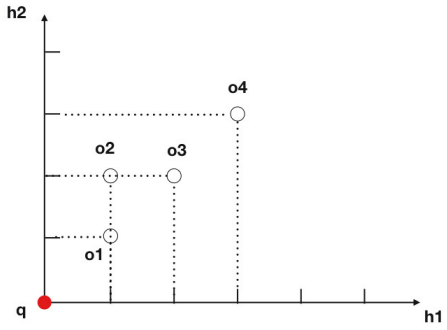


Figure 5.2 : Motivation for RI-LSH

at αm times, where α is a parameter related to c and the initial bucket width. The termination conditions of RI-LSH are similar with I-LSH: (1) if there are βn candidates in the candidate set S, RI-LSH will terminate and return the furthest point in S as the c-AFN fo q, or (2) if there is a point o satisfied that $\parallel o, q \parallel \geq R/c$ (in I-LSH, the condition is $\parallel o, q \parallel \leq cR$) then return o as the result, where R is corresponding l_2 distance of the current searching radius r.

Algorithm 3 shows the details of the query processing step. The input consists of m B+ trees indices, the query point q and an initial bucket width w. Let n_{can} denotes the number of candidates found in the searching procedure. Firstly the query point q will be projected to the m hash functions (line 3) and then the searching start at the furthest point to q over all of the m B+ trees (line 4 to line 3). At each iteration, the next object with largest projected distance to q will be accessed (line 3). We use o and i to denote the object and the projection dimension separately. $r = \parallel h_i(o), h_i(q) \parallel$ is the projection distance between o and q on i. Using r, the corresponding radius R can be calculated (line 3). $cn(o)$ demonstrates the visited times of o. If $cn(o) == \alpha m$ then o will be added to the candidate set S. The real

Algorithm 3: **Reverse 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-AFN object

1 $n_{can} := 0$;

2 Apply m hash functions on q;

3 **while** $n_{can} < \beta n$ **do**

4 $o \leftarrow$ next object with largest projected distance;

5 $i \leftarrow$ the projection dimension o comes from;

6 $r \leftarrow \parallel h_i(o), h_i(q) \parallel$;

7 $R \leftarrow \frac{2r}{w}$;

8 $cn(o) := cn(o) + 1$;

9 **if** $cn(o) == \alpha m$ **then**

10 compute $\parallel o, q \parallel$ and update o_{max} and R_{max};

11 $n_{can} := n_{can} + 1$;

12 **if** $R_{max} \geq R/c$ **then**

13 break;

14 **return** o_{max}

distance between o and q will be computed (line 3). If $\parallel o, q \parallel > R_{max}$ then update o as the current c-AFN result o_{max} and set $R_{max} = \parallel o, q \parallel$ (line 3). The candidate set size n_{can} will be added to 1 (line 3). If the current largest distance R_{max} satisfies the termination condition 2 (3), the algorithm will terminate (line 3) otherwise it continues until the first termination condition is satisfied (line 4). Finally, the point o_{max} will be returned as the c-AFN.

Correctness. Given a query point q, an approximation ratio c and a success pos-

sibility δ, when we choose proper parameters m, α and β, $RI-LSH$ can return a c$-AFN$ with a probability at least δ. The details of the parameters will be described in Section 5.5.

I/O costs. The majority of I/O cost comes from reading data from the B+ trees for reverse incremental search and the computation of real distance between candidates and the query point. We use the number of leaf-node visited to evaluate the I/O cost because all the hash values and data IDs are in the leaf-nodes. To compute the real distance between two points, one random I/O will be invoked to read the original d-dimensional data from the disk. For the reverse incremental search, each time we load l leaves together and cost 1 random I/O and l sequence I/O. Thus the total I/O cost is $O(\frac{s}{n_e} \cdot I_{seq} + \frac{s}{n_e l} \cdot I_{ran} + n_{can} \cdot I_{ran})$, where s is the total iteration times, n_e is the number of entries per page, n_{can} is the number of candidate objects accessed and l_{seq}, l_ran denote the unit cost of sequence I/O and random I/O respectively.

5.4.3 c-k-AFN

Algorithm 3 can be easily extended to solve the c-k-AFN search problem by the following changes: (1) instead of at most βn candidates in the set, we require $\beta n+k-1$ candidates as the termination condition; (2) instead of o_{max}, we maintain k furthest candidate points o_k, and their distances to q will be used for the termination condition test, and (3) there will be k points returned by the algorithm as the result.

5.5 Analysis

In this section, we provide the theoretical guarantee of RI-LSH: given a query point q, approximation ratio c ($c > 1$) and success possibility δ ($0 \leq \delta \leq 1$), our algorithm can return a c-AFN of q with probability at least δ.

Correctness of (R, c)-FN Given a radius R, a reverse LSH family can return a point o satisfies that $\| o, q \| \geq R/c$. First, we prove the correctness when the radius

R is given. Suppose $R = c$, and q is the query point, the reverse LSH family is $(c, 1, p_1, p_2)$-sensitive. There are two termination conditions.

$\mathcal{E}1$: There is a point o which satisfies $\| o, q \| \geq R/c$.

$\mathcal{E}2$: There are βn candidates have been found in the searching procedure.

We use $P_1 = Pr[\mathcal{E}1]$ and $P_2 = Pr[\mathcal{E}2]$ to denote the possibility that our algorithm is terminated by $\mathcal{E}1$ or $\mathcal{E}2$ and returns a correct result, separately.

Proof 4: A point o will be considered as a candidate if $\#collision(o) \geq \alpha m$. For $\forall o \notin B(q, R)$,

$$Pr[\#collision(o) \geq \alpha m] = \sum_{i=\alpha m}^{m} C_m^i p^i (1-p)^{m-i}$$

, where $p = Pr[o \ separates \ with \ q]$. According to C2LSH [28], $Pr[\#collision(o) \geq \alpha m] \geq 1 - exp(-2(p_1 - \alpha)^2 m)$. This is the value of P_1. To compute P_2, we consider for any data object $o \in B(q, R/c)$,

$$Pr[\#collision(o) \geq \alpha m] = \sum_{i=\alpha m}^{m} C_m^i p^i (1-p)^{m-i}$$

, where $p = Pr[o \ separates \ with \ q] \leq p_2 < \alpha$, $j = 1, 2, ..., m$. Similarly, based on Hoeffding's Inequality, the upper bound of $Pr[\#collision(o) \geq \alpha m]$ is $exp(-2(\alpha - p_2)^2 m)$. And based on Markov's Inequality, we have

$$P_2 > 1 - \frac{1}{\beta} \cdot exp(-2(\alpha - p_2)^2 m)$$

. When $m = \lceil max(\frac{1}{2(p_1-\alpha)} ln\frac{1}{\delta}, \frac{1}{2(\alpha-p_2)^2} ln\frac{2}{\beta}) \rceil$, we have $P_1 \geq 1 - \delta$ and $P_2 > \frac{1}{2}$. So the total success possibility is $P_1 + P_2 - 1 > \frac{1}{2} - \delta$.

Reverse Incremental Search The correctness of RI-LSH is depending on the (r_1, r_2, p_1, p_2)-sensitive property of the reverse LSH function. We have proved in chapter 3 section 3.5 to indicate that a $(1, c, p_1, p_2)$-sensitive hash function with bucket width w is (k, ck, p_1, p_2)-sensitive if the bucket width is set to kw. So each bucket width kw has a corresponding $B(q, R)$ in $\mathcal{R}^d$ centered at q with radius $R = k$,

which means for any point $o \notin B(q, R)$, o separates with q (i.e., $\parallel h(o), h(q) \parallel > \frac{kw}{2}$) with possibility at least p_1, and for any point $o \in B(q, R/c)$, o separates with q with possibility at most p_2.

Since k can be any real value greater than 0, kw and ck can be any non-negative real value as well. Let r be the current searching radius, we can always find a k satisfied $ck = r$. So the hash function is always (cr, r, p_1, p_2)-sensitive.

Correctness of algorithm. We have proved the correctness of (R, c)-AN situation, and proved the correctness won't be broken when w changes with the bucket width R. So RI-LSH can return a c-AFN with at least $\frac{1}{2} - \delta$ possibility.

Approximation Ratio. The existing c-AFN algorithm with theoretical guarantee RQALSH achieves a c^2 approximation ratio. Since the number of B-trees required to keep the theoretical guarantee m is related to the approximation ratio c, a c^2-approximate furthest neighbor algorithm will require a larger size of index and a larger I/O cost compared with c-approximate algorithm under the same condition. Our algorithm reduce the approximation ratio to c and can use a much smaller size of index to keep the theoretical guarantee.

Let P_1 denote the condition that if there is a point o falling out of $B(q, R)$, $\parallel h(o), h(q) \parallel \geq r$. Let P_2 denote the condition that the total false positive number is less than βn, which means when the algorithm finds βn candidates, at least one candidate is a c-AFN of q.

Lemma 6: When both of P_1 and P_2 hold, given a query point q, suppose the furthest neighbor of q is o^* and $\parallel o^*, q \parallel = R^*$, rilsh will stop at a radius R which satisfies $R \geq R^*$.

Proof 5: Since P_1 is satisfied, the furthest neighbor of q must have a projection distance larger than r^*, where r is the corresponding projection radius of $B(q, R^*)$, which means $\parallel h(o^*), h(q) \parallel \geq r$. So when all the points falling out of $B(q, R^*)$ have

been checked, there are two possible conditions: (1) There are more than βn points visited, then the algorithm will be terminated by C_1 since P_2 holds. Then we have $R \geq R^*$; (2) Before RI-LSH visits βn candidates, it finds o^* and then return it as the result since o^* falls out of r according to P_1. We still have $R \geq R^*$.

According to lemma 6, we have that under a constant possibility $1/2 - \delta$, the algorithm can always return a c-approximate furthest neighbor of q. So the approximation ratio is c.

5.6 Experiments

In this section, we present results of comprehensive experiments to evaluate the I/O efficiency and accuracy of the proposed technique in the thesis compared with the existing c-AFN algorithms. We choose the state-of-the-art algorithm RQALSH and use the same setting to conduct experiments.

5.6.1 Experiment Setup

In this section, we introduce the experiment settings of our performance evaluation including the chosen benchmark methods, the datasets, the evaluation metrics we use to compare the algorithm fairly and the initial parameter setting.

Benchmark methods. RQALSH is the state-of-the-art c-AFN algorithm with theoretical guarantee.

- `RQALSH` was proposed by Qiang Huang [36] in 2017. The source code is from the author's website https://github.com/HuangQiang/RQALSH .
- `RI-LSH` is the algorithm proposed in this thesis.

Datasets. We conduct experiments on several million-scale real-world high-dimensional datasets.

- `Sift` contains 1 million 128-dimensional SIFT vectors.
- `Tiny` contains 5 million GIST feature vectors in a 384-dimensional space.
- `MillionSong` contains 1 million 420-dimensional data points.

Evaluation Metrics. We evaluate the algorithms using three evaluation metics: I/O cost, running time and I/O/ratio. We also compare the index size between RI-LSH and RQALSH under the same theoretical guarantee. Since the algorithms are built for external memory, we use I/O cost as the primary evaluation metric to evaluate the algorithms. Each random I/O contributes one I/O cost to the I/O costs, and each sequential I/O contributes 0.1 I/O cost. **Parameter Setting.** To fairly compare the algorithms, we set the theoretical guarantee to be $1/2 - 1/e$, which means the algorithm will return a c-AFN with at least δ possibility. And the default ratio c is 4. Since RI-LSH is a c-approximate algorithm but RQALSH is c^2-approximate algorithm, we set c to be 4 for our algorithm and 2 for RQALSH. All the other settings follow the default setting in RQALSH. The success possibility is $\delta = 1/2 - 1/e$ and the required hash function number m is calculated using c, β and δ. In our algorithm, we set $\beta = 0.001$ and when $c = 4$, m is a constant value 16 for all the datasets. The page size B is set to 8192 bytes for all the algorithms and all the datasets.

All the experiments were executed on a PC with intel(R) Xeon(R) CPU E3-1231 v3 with 3.04GHz, 8 cores and 32G memory. The program was implemented in C++ 11. We select 100 query points for each dataset and use the average result to evaluate the algorithms.

5.6.2 Index Size

Using the default approximation ratio $c = 4$, the index sizes of our algorithm and RQALSH are shown in Table 5.2. Since we set m to be a constant value and

changes β to achieve the theoretical guarantee, our algorithm always requires 16 B+ trees when $c = 4$. RQALSH sets β to be a constant and changes m to hold the guarantee, so the index size varies dramatically.

Table 5.2 : Index size

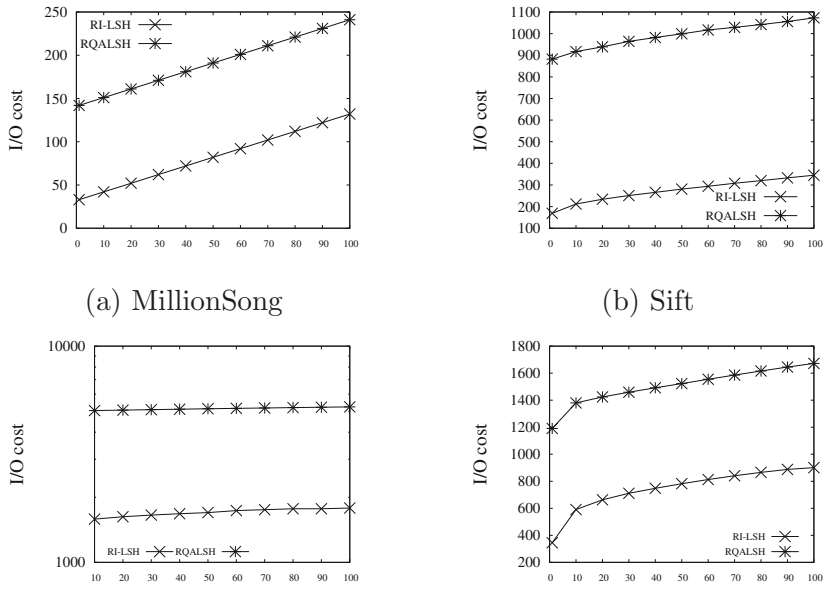
dataset (d))	Sift (128)	Tiny (384)	Song (420)
m	77	86	77
`RQALSH (MB)`	372	2000	391
m	16	16	16
`RI-LSH (MB)`	151	760	155

5.6.3 I/O costs

For a c-AFN algorithm with theoretical guarantee, the best solution to evaluate its I/O efficiency is to calculate the minimum I/O cost required to hold the theoretical guarantee. For the given approximation ratio $c = 4$ and success possibility $\delta = 1/2 - 1/e$, we conduct experiments for the two algorithms over the real-world datasets. The value of k is varying from 1 to 100 and the default value is 30. The experiment results are given in Figure 5.3. From the figures, we have the following observations:

- The I/O consumption of RQALSH is larger than RI-LSH over all the four datasets under the save theoretical guarantee and success possibility. This is because RQALSH is c^2-approximate algorithm while RI-LSH is c-approximate algorithm. When the approximation ratio is same, RQALSH requires much larger hash functions to project the dataset, and causes a larger I/O cost.

- When $c = 2$, the difference between our algorithm and RQALSH becomes larger. RQALSH costs more than 1500 I/O per query on the sift dataset.
- The I/O cost of both RQALSH and RI-LSH increase steadily when k grows. Because to solve the c−k−AFN problem, the algorithm has to visit more points to find more candidates.



(a) MillionSong (b) Sift

(c) Sift, $c = 2$ (d) Tiny

Figure 5.3 : I/O costs varying k

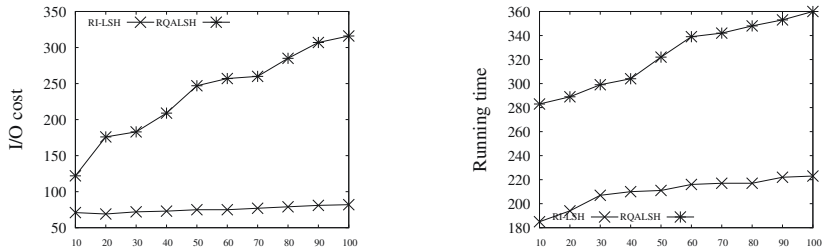
5.6.4 Running time

Another evaluation metric is to compare the running time of the two algorithms. Since both methods use external memory, the I/O time is also included in the total running time. To get a fair and accurate result, we conduct experiments on the same PC and repeat for several times. The experiment results are shown in Figure 5.4. From the experiment result, we get the following conclusion:

- The running time is related to I/O cost. Since RQALSH always costs more I/O than our algorithm, it also has a larger time consumption over all the

datasets. The running time of RQALSH is at least 2 times of the RI-LSH's running time.

- On some easy dataset, like millionSong, the running time of RI-LSH basically doesn't increase when k becomes larger, but QALSH's running cost grows dramatically. The main reason is on easy dataset, the number of candidates finally found won't be much greater than k. Our algorithm requires less B+ trees so that the value of k doesn't affect running time a lot.



(a) MillionSong (b) Sift

Figure 5.4 : Running time varying k

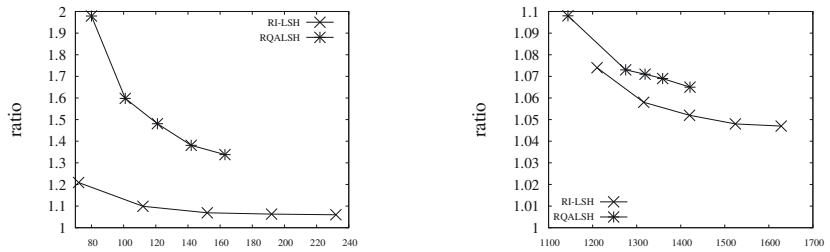
5.6.5 I/O and ratio

To evaluate the accuracy of the two algorithms fairly, we conduct experiment to compare the ratio-I/O of RI-LSH and RQALSH instead of ratio-k. A I/O efficient algorithm means that it can achieve a same ratio using less I/O for the same theoretical guarantee. We set $k = 20$ and modify the termination condition: for a given upper bound of the candidate set size, the algorithm will terminate if the point number in the candidate set has achieved the upper bound. The experimental results are given in Figure 5.5. From the results we have the following observations:

- For all the datasets, RI-LSH can get a higher accuracy result when using the same I/O cost compared with RQALSH. Since the two algorithms use a

similar scheme, the ratio difference is small. Our algorithm uses a continuous searching strategy on each B+ tree, so it can find a better solution earlier than RQALSH.

- Another reason that RI-LSH can get a better I/O-ratio result is our algorithm requires a smaller number of B+ trees to build index. Then in the searching step, it is easier to find a candidate. So using similar I/O cost, our algorithm can visit and verify more points, which leads to a better ratio.



(a) MillionSong (b) Sift

Figure 5.5 : I/O vs. ratio

5.6.6 Summary

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

- Under the same theoretical guarantee, our algorithm can behave better than RQALSH considering both I/O cost and running time. This is because we adopt a continuous searching strategy, which enables us to decrease the approximation ratio from c^2 to c and decreases the hash functions number required to keep the theoretical guarantee.

- When the approximation ratio is a smaller value, RI-LSH's time consumption and I/O consumption grow evidently slower than RQALSH. It can be expected

that when c tends to be 1, RQALSH will require a huge size of index, and also an extremely expensive I/O cost and time cost.

5.7 Conclusion

In this thesis, we proposed a novel I/O efficiency c-AFN method based on the classical LSH scheme. It can return a c-AFN result with a constant possibility regardless of the data distribution. Compared with existing work such as RQALSH, our algorithm uses a much smaller index and cost less I/O and running time to hold the same theoretical guarantee. Besides, RI-LSH discards the 'bucket' in LSH and keep the theoretical guarantee as well. To the best of our knowledge, our algorithm is the first c-AFN method with theoretical guarantee to achieve a c approximation ratio.

Chapter 6

Skyline Nearest Neighbour Search on multi-layer graph

6.1 Overview

In this chapter, we define a new problem on multi-layer graph called Skyline Nearest Neighbour Search and propose two algorithms to solve it. This work has been published on ICDE2019 workshop. Section 6.2 defines the problem, and introduces our approaches in section 6.3. The optimization is given in section 6.4. Section 6.5 shows evaluation results of baseline and optimizations.

6.2 Preliminaries

In this section, we present the problem definition and relevant existing works. Some important notations used throughout the thesis are summarized in Table 6.1.

6.2.1 Problem definition

A multi-layer graph is a set of graphs $\{G_1, G_2, ..., G_l\}$, where l is the number of layers, and G_i is the graph on layer i. All the l graphs contain the same vertex set V. A l-layer graph can be represented by $\mathbb{G} = (V, E_1, E_2, ..., E_l)$, where V is the universal vertex set, and E_i is the edge set of graph G_i. For each vertex $v \in V$, the neighbors of v in graph G_i, denoted as $N(v, i)$, is defined as $N(v, i) = \{u|(v, u) \in E_i\}$. Each edge $e = (u, v, i) \in E_i$ is associated with a positive weight $\phi(u, v, i)$.

Definition 10: **Path** A path on layer j is a sequence of vertices $p = (v_1, v_2, ..., v_k)$

Table 6.1 : Summary of Notations

Notation	**Definition**
$\mathbb{G}(V, E_1, ..., E_l)$	a l-layer graph $\mathbb{G}$ whose vertex set is V and edge set is E_i of layer i
V	the vertex set in the graph
E_i	the edge set of layer i in the graph
l	the number of layers in the graph
$\phi(u, v, i)$	the weight of the edge between vertex u and v on layer i
$dis(u, v, i)$	the shortest distance between u and v on layer i
$Num(u, dis, i)$	for vertex u, the estimated vertex number within the distance dis on layer i
$Max[i]$	the current largest shortest distance has been found on layer i
$N(v, i)$	the vertices which are connected to v on layer i

where $(v_i, v_{i+1}) \in E_j$ for every $1 \leq i < k$. The weight of the path p is denoted as $\phi(p) = \sum_{i=1}^{k-1} \phi(v_i, v_{i+1})$. Given two vertices $s, t \in V$, a shortest path p between s and t is a path whose weight is the smallest among all the paths starting from s and ending at t.

Definition 11: **Shortest distance** The shortest distance between s and t on graph G_i, denoted as $dis(s, t, i)$, is the weight of any shortest path between s and t on layer i.

Skyline nearest neighbor. The nearest neighbor of a vertex in a simple graph has been widely studied. To capture the nearest neighbor information on multi-layer graphs, we first introduce the dominance relationship between two vertices.

Definition 12: **Dominate** Given a graph $\mathbb{G}(V, E_1, ..., E_l)$ and a query vertex v_0, we say v_1 dominates v_2, denoted by $v_1 \prec_{v_0} v_2$, if $\forall dis(v_0, v_1, i) \leq dis(v_0, v_2, i)$ where $i = 1, 2, ..., l$, and $\exists i \in \{1, ..., l\} : dis(v_0, v_1, i) < dis(v_0, v_2, i)$.

For example, given a query vertex $v_0 \in \mathbb{G}(V, E_1, ..., E_2)$ in figure 1.1, $dis(v_0, v_1, 0) = dis(v_0, v_1, 1) = 1$, and $dis(v_0, v_3, 0) = dis(v_0, v_3, 1) = 2$, so $v_1 \prec_{v_0} v_3$.

Given a query vertex v_0 in a multi-layer graph $\mathbb{G}(V, E_1, ..., E_l)$, the skyline nearest neighbors set of v_0 is a vertex set S in which vertices are not dominated by all other vertices in V.

The formal definition of our problem is given as follows:

Definition 13: **Skyline nearest neighbor** Given a multi-layer graph $\mathbb{G}(V, E_1, ..., E_l)$ and a query vertex v_0, the problem is to find all the skyline nearest neighbors of v_0 from $\mathbb{G}$. More formally, given a multi-layer graph $\mathbb{G}$ and a query vertex v_0, we aim to compute a subset S which is defined as

$$S = \{v \in V \mid \nexists v' \in V : v' \prec_{v_0} v\}$$

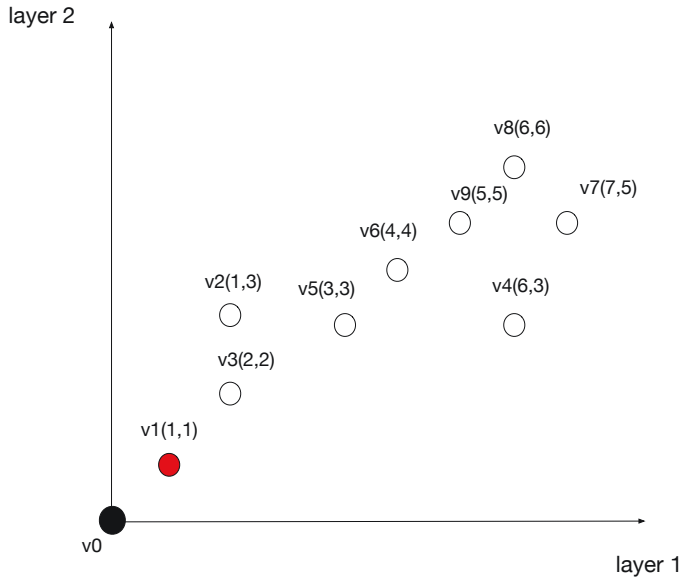


Figure 6.1 : Skyline nearest neighbors of v_0 on graph 1.1

Example 1: We give an example of the skyline nearest neighbors of vertex v_0 in figure 6.1. Graph $\mathbb{G}(V, E_1, E_2)$ contains 10 vertices and two layers. v_0 is the query vertex. According to the definition of skyline nearest neighbor, v_1 is the only object in the skyline set of v_0 because it dominates all the other vertices.

6.3 Approach

In this section, we will briefly introduce the baseline algorithm and theoretically analyze the time complexity. The drawbacks of the baseline and optimizations will be given in section 6.4.

To compute skyline nearest neighbors, the complete coordinates of each vertex, which are the shortest distances to the query vertex v_0 on each layer, are required. Since shortest distance and skyline are both well-solved problems, a simple idea is just calculating the shortest distances for all the vertices on every layer of $\mathbb{G}$ to v_0, then using these values as coordinates to calculate skyline. The pseudocode is given in algorithm 4.

First, line 4 computes the shortest distance between v_0 and v on each layer using Dijkstra's algorithm. Then based on these distances, we use an efficient skyline algorithm SFS [18] to calculate the skyline. It requires $O(l(|E|+|V|log|V|)+T_{SFS})$ to calculate the shortest distances between the query vertex and every vertex in V on all the layers and compute the skyline nearest neighbors, where T_{SFS} denotes the time complexity of the SFS algorithm.

Algorithm 4: **BASE**

Input : $\mathbb{G}(V, E_1, ..., E_l)$: a graph $\mathbb{G}$ which has l layers. The vertex set is V and the edge set of layer i is E_i. v_0: the source vertex

Output : S: a set of skyline nearest neighbors of v_0 in $\mathbb{G}$

1 **for** $i \leftarrow 1$ to l **do**

2 Compute the shortest distance from v_0 to every vertex in V on layer i using Dijkstra's algorithm;

3 $S \leftarrow$SFS(V);

4 **return** S

6.4 Optimizations

6.4.1 An Early-Stop Approach

The baseline is able to return a correct answer for the skyline nearest neighbor query, but it always has to compute shortest distances between every vertex in V to the query vertex on all the layers. The time consumption is expensive when the graph is large. To decrease the time cost, intuitively we hope to decrease the number of vertices needed to be visited. According to the property of Dijkstra's algorithm, we observe lemma 7 which can be useful to stop the shortest distance computation much earlier.

Lemma 7: In the procedure of Dijkstra's algorithm, the first vertex v which is visited on all the layers must be a skyline nearest neighbor.

Proof 6: From the procedure of Dijkstra's algorithm, it is straightforward that for two vertices v_1, v_2, if $dis(v_0, v_1, i) < dis(v_0, v_2, i)$, then v_1 will be visited earlier than v_2 on layer i. A skyline nearest neighbor is definitely won't be dominated by any other vertex, which means, if v is a skyline point, then $\nexists u \in V$ satisfies $\forall i \in \{1, ..., l\} : dis(v_0, u, i) \leq dis(v_0, v, i)$ and $\exists i \in \{1, ..., l\} : dis(v_0, u, i) < dis(v_0, v, i)$. If the first vertex v which is first found on all the layers is not belonging to the skyline of v_0, then there must be a vertex $v' \prec_{v_0} v$, means that $\forall i \in \{1, ..., l\} : dis(v_0, v', i) \leq dis(v_0, v, i)$ and $\exists i \in \{1, ..., l\} : dis(v_0, v', i) < dis(v_0, v, i)$. If v' exists, from the property of Dijkstra's algorithm, it is impossible to visit v on all of the l layers before v', so v' doesn't exist, and v is belonging to the skyline nearest neighbor result of v_0.

Based on lemma 7, it is easy to get lemma 8.

Lemma 8: Given a graph $\mathbb{G}(V, E_1, ..., E_l)$ and a query vertex v_0, when a skyline nearest neighbor v is found, any vertex u which hasn't been accessed on at least one layer is impossible to be a skyline nearest neighbor of v_0.

Proof 7: For two vertices v and u in V, if v and v are both skyline nearest neighbors of query vertex v_0, then $u \nprec_{v_0} v$ and $v \nprec_{v_0} u$, which means $\exists i \in \{1, ..., l\} : dis(v_0, v, i) < dis(v_0, u, i)$ and $\exists j \in \{1, ..., l\} : dis(v_0, u, j) < dis(v_0, v, j)$. If u hasn't been visited on any layer when v is visited for l times, then $\forall i \in \{1, ..., l\} : dis(v_0, v, i) < dis(v_0, u, i)$, so $v \prec_{v_0} u$, u must not be a skyline nearest neighbor of v_0.

The general idea to solve the problem contains two steps: (1) compute the shortest distance, and (2) calculate the skyline. For step 1, using lemma 8, we propose an early-termination condition: during the procedure of computing shortest

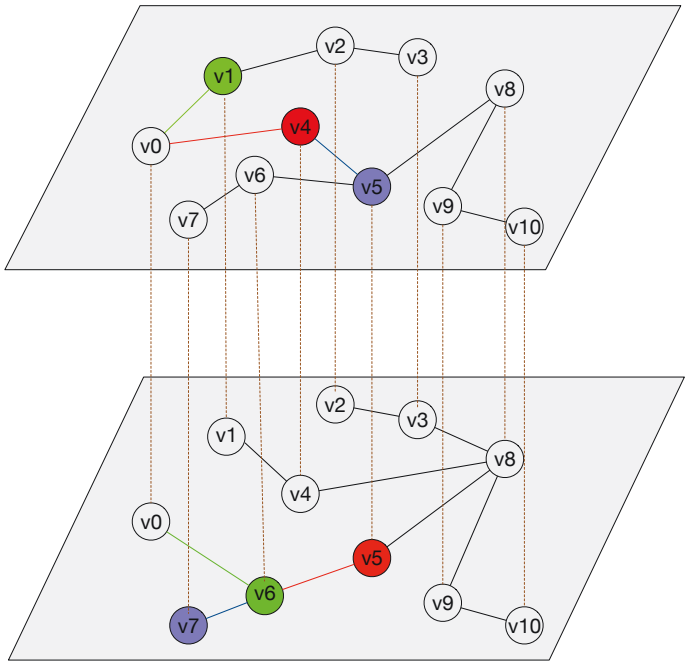


Figure 6.2 : Procedure of optimization

distances from the query vertex to each vertex in graph $\mathbb{G}$, once a vertex v has been visited on all of the l layers, step 1 can be stopped. Using this pruning strategy can stop step 1 much earlier in most cases, meanwhile reduce the candidate size of step 2 to compute skyline. Example 2 shows the procedure.

Example 2: In figure 6.2, there are two layers of an undirected graph $\mathbb{G}(V, E_1, E_2)$. Each edge has the same weight. Suppose the query vertex is v_0. In the first iteration, v_1 and v_6 are visited on layer 1 and layer 2 respectively. Then in the next iteration, v_4 and v_5 are visited on layer 1 and layer 2 repectively. Up to now, there is no vertex which has been visited for 2 times, so the algorithm will continue to the third iteration. In iteration 3, v_5 and v_7 are visited on layer 1 and 2. Since v_5 currently is visited on both layer 1 and layer 2, the early-termination condition is satisfied. Compared with the baseline, using early-termination can reduce the total visited vertices from 20 to 6.

We propose algorithm 5 to improve the performance. To find a vertex satisfied

the early-termination condition, the algorithm visits one vertex in a layer every time, and selects each layer in turn. Different from baseline, we need a set C to store the possible skyline candidates, and SFS will only consider the vertices in set C but not all the vertices in V. The set C is initialized in line 5. Then for each layer in $\mathbb{G}$, the algorithm selects the top element of the priority queue $P[i]$ to visit, adds it to C and updates its neighbors. Once there is a vertex v which has been visited for l times, the step for shortest distance computing will terminate, and the algorithm will compute the skyline of the query vertex.

While before the computation of skyline, we haven't computed the shortest distances for vertices in C on all of the l layers. T. Akiba proposed an effective algorithm [2] called landmark labelling to answer such queries in social network. Using landmark labelling, we can get the complete shortest distances on l layers for all the vertices in C efficiently.

6.4.2 Layer chosen strategy

In algorithm 5, every layer is selected in turn to find a skyline nearest neighbor. To find a skyline vertex of v_0 earlier, another idea is to choose the layer based on the current known shortest distance to v_0 on this layer. In each iteration, the algorithm still only visit one vertex, but the layer chosen rule (line 5 in algorithm 4) is changed: we store the current maximum value of visited vertices' shortest distance to v_0 for each layer, donated as $Max[i]$. For each iteration, the layer i which satisfies $\nexists Max[j] < Max[i]$ where $i, j \in \{1, ..., l\}$ will be selected. In figure 6.3, given a query vertex v_0, using this layer chosen rule can stop visit less vertices. In the first iteration, v_5 is visited on layer 1 and v_1 is visited on layer 2, now $Max[1] = Max[2] = 1$. The algorithm continues on layer 1, and v_3 is visited. Because $Max[1] = 2$ and $Max[2] = 1$, the algorithm will keep choosing layer 2 until v_3 is accessed. Now v_3 has been accessed on all the layers, so Dijkstra can stop. The early-termination

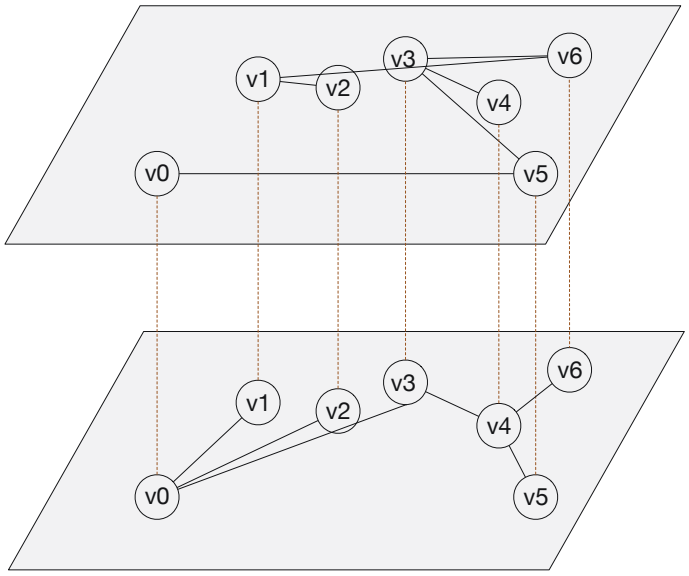


Figure 6.3 : a graph suitable for **SNNS-ET2**

condition could be satisfied when 5 vertices have been visited, and only 4 candidates will be added to C. If using **SNNS-ET**, 6 vertices are needed to be visited. The procedure is as follows:

However, the distance-based strategy is still not suitable for every graph. For example, in figure 6.2, for the query vertex v_0, if the early-termination condition can be activated at v_6, the total visited vertex number won't exceed 8 even in the worst case (because the number is affected by the Dijkstra). Using either of the two orders above is not able to lead to the result. Is there a method to estimate the amount of vertices required to be visited on each layer so that we can stop the algorithm earlier?

The purpose is to reduce the unnecessary visited vertex number. If the algorithm can estimate the vertices number still need to be visited on each layer, and select the layer which has a smallest estimated vertices number, the early-termination condition is more likely to be satisfied earlier.

For example, in a 2-layer graph, suppose the algorithm is currently running on

layer 1 and visiting v on this layer. Let v' denote the latest vertex visited on layer 2, and $U \subset V$ be a vertex set. $\forall u \in U$ satisfied $dis(v_0, u, 2) \geq dis(v_0, v', 2)$ and $dis(v_0, u, 2) \leq dis(v_0, v, 2)$. The size of U is the number of vertices needed to be visited for v to satisfy the early-termination condition.

To estimate the vertices number, we can precompute an index. Given a graph $\mathbb{G}(V, E_1, ..., E_l)$, for $\forall v \in V$, we first compute the shortest distance between v and all the other vertices in G on the l layers. D is a set which contains all the distinct values of the distances. The index can be builded to support such operation: given a vertex $v \in V$, a distance d and a layer $i \in \{1, ..., l\}$, $Num(v, d, i)$ returns the number of vertices u which satisfies $dis(v, u, i) \leq d$ for vertex i in a l-layer graph $\mathbb{G}$.

However, in the worst case, the size of D can be $|V|$. To decrease the index size, if $|D| > k$, where k is a parameter set by the user, we select k elements from D. For each vertex $v_s \in V$, sorting the distances from v_s to all the other vertices in an increasing order for all the layers respectively. Then store all the $dis(v_s, v_j, i)$ as elements in D', where v_j is the $j \cdot |V|/k$th vertex after sorting on layer i. The space consumption for the index is $O(kln)$.

Especially, if the value $j \notin D'$, we use $j' : j' < j \bigwedge \nexists j^* : j > j^* > j'$ instead. The index can well support the estimating function with $O(logk)$ time complexity.

We maintain two values for each layer i: $Max[i]$ and $Limit[i]$. $Max[]$ has the same meaning as it in **SNNS-ET2**, and $Limit[i] = min\{dis(v_0, v, i)\}$, where v satisfies $v \in C \wedge vis[v][i] = false$. At the beginning, the algorithm chooses a layer i using the choose_layer() function, then for layer i, selects the top element v of the priority queue $P[i]$ to visit, adds it to C, updates $Max[i]$ to the current distance. If v is unvisited on another layer j and the $dis(v_0, v, j)$ is smaller than $Limit[j]$, then update $Limit[j]$ to $dis(v_0, v, j)$.

In the choose_layer() function (algorithm 6), the function estimates the number

of vertices which satisfies $Max[i] \leq dis(v_0, v, i) \leq Limit[i]$. The layer which has the smallest estimated number will be returned as the result.

6.5 Performance Studies

This section shows our experimental results on a set of real-world graphs. The parameter k is set as 1000 in our experiments.

6.5.1 Experiment setup

Dataset. There are three real-world graphs. All of the real-world graphs are chosen from SNAP and KONECT.

- math overflow. This is a network of interactions on the stack exchange web site Math Overflow. There are three different types of interactions represented by a directed edge (u, v):

  user u answered user v's question;

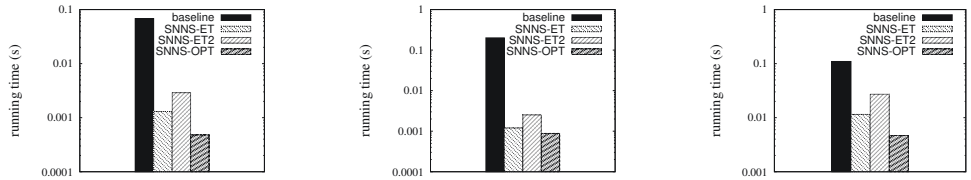
  user u commented on user v's question;

  user u commented on user v's answer;

  There are about 20000 vertices and 500000 edges in the graph.

- soc. This is who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin OTC. It can be modeled as a weighted network with timestamp. For an edge created before 2014, it belongs to layer 1, otherwise it belongs to layer 2. The graph contains about 6000 vertices and 24000 edges.

- College. This dataset is comprised of private messages sent on an online social network at the University of California, Irvine. Users could search the network for others and then initiate conversation based on profile information. An edge (u, v, t) means that user u sent a private message to user v at time t. We model



(a) college (b) soc (c) sx

Figure 6.4 : running time

it as a 2-layer graph considering the timestamp. There are 1899 vertices and 59835 edges.

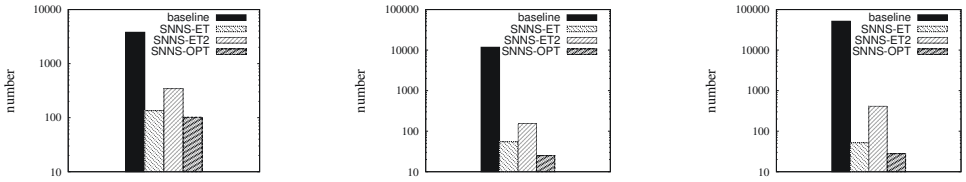
We evaluate the algorithm by studying the performance after averaging the 100 random queries. The query vertices are randomly chosen from the graph.

6.5.2 Running time

The experiment results show that **BASE** is much slower than the other three approachs, especially on large graphs. The reason is it always needs to access almost the entire graph, and all the vertices will be included in the candidate set to compute skyline. The performance on the same graph is effected by the query vertex, but **SNNS-OPT** always utilizes the shortest time. On sx, the running time of **SNNS-OPT** is only about 10% of the cost of **BASE**, and about 50% of the running time of **SNNS-ET** and **SNNS-ET2**. **SNNS-ET2** is slightly slower than **SNNS-ET**, and both of them are faster than **BASE**.

6.5.3 Effectiveness of the optimizations

To prove the optimizations are effective, we conduct experiments to compare the total visited vertices number when Dijkstra is stopped. The results represent that the early-termination condition can reduce the skyline candidate set size significantly. **SNNS-OPT** decreases the visited vertices number most significantly com-



(a) college (b) soc (c) sx

Figure 6.5 : number of visited vertices

pared with **SNNS-ET** and **SNNS-ET2**. The visited vertices number of **SNNS-OPT** is no more than 70% of the number of **SNNS-ET2** and **SNNS-ET**. The differences among these algorithms are also effected by the graph. On sx, **SNNS-ET2** visits about 8 times vertices of **SNNS-ET**.

6.6 Conclusion

To find the nearest neighbors on multi-layer graphs for a given query vertex, we formulate a problem called skyline nearest neighbor search ($SNNS$), and propose a baseline method to answer a query correctly. The optimizations use an early-termination condition, which can generally stop the algorithm in a shorter time. The experiments on several real-world graphs demonstrate that the early-termination condition and the strategy to choose layer can reduce the time consumption significantly.

Algorithm 5: **SNNS-ET**

Input : $\mathbb{G}(V, E_1, ..., E_l)$: a graph G which has l layers, the vertex set is V and the edge set of layer i is E_i. v_0: the source vertex

Output : S: a set of skyline points of v_0 in G

```
1  C ← ∅;
2  for i ← 1 to |V| do
3      sum[v_i] ← 0;
4      for j ← 1 to l do
5          if v_i ≠ v_0 then
6              dis(v_0, v_i, j) ← ∞;
7  for i ← 1 to l do
8      P[i].insert(v_0);
9  while ∃i ∈ {1, ..., l} : P[i] is not empty do
10     if ∃v ∈ V : sum[v] = l then
11         break;
12     for i ← 1 to l do
13         v ← P[i].top();
14         Add v to C;
15         vis[v][i] ←true ;
16         sum[v] ← sum[v] + 1;
17         if sum[v] = l then
18             break;
19         for u ∈ N(v, i) do
20             if !vis[u][i] and dis(v_0, u, i) > dis(v_0, v, i) + (u, v) then
21                 P[i].insert(u);
22 for v in C do
23     for i ← 1 to l do
24         if !vis[v][i] then
25             compute dis(v_0, v, i) using landmark labelling;
26 S ←SFS(C);
```

Algorithm 6: **choose_layer**

Input : $\mathbb{G}(V, E_1, ..., E_l)$: a graph $\mathbb{G}$ which has l layers, the vertex set is V and the edge set of layer i is E_i. v_0: the source vertex $Max[]$: currently the maximum visited distance on each layer $Limit[]$: the minimum unvisited distance on each layer

Output : i: the layer which is chosen in the next iteration

```
1 ans ← ∞; layer ← 0;;
2 for i ← 1 to l do
3     if ans < Num(v_0, Limit[i], i) − Num(v_0, Max[i], i) then
4         ans ← Num(v_0, Limit[i], i) − Num(v_0, Max[i], i);
5         layer ← i;
6 return i
```

Chapter 7

Conclusion

In this thesis, we studied the query processing problem in high-dimensional space as well as graph space considering different distance metrics: Euclidean distance, inner product and skyline on graphs. In this chapter we summarize the contributions we have made and talk about our future research.

7.1 Contributions

In this thesis, we made the following contributions:

- For c-ANN problem, we proposed a new c-approximate nearest neighbor search algorithm and a more aggressive version, namely EI-LSH, seperately, for high-dimensional data, which uses an incremental search strategy on the projected dimensions. We also designed a new early termination technique which can be used by most list-based LSH algorithms to aggressively reduce the number of objects accessed without breaking the theoretical guarantee. In addition, we provided a rigorous analysis to demonstrate the correctness and efficiency of our proposed methods. Furthermore, we performed 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.

- for Approximate MIPS problem, we selected a set of state-of-the-art Approximate MIPS algorithms and conducted comprehensive experiments using real-

world datasets considering a variety of metrics to evaluate their performances. We also gave some practical suggestions.

- For c-AFN problem, we proposed a novel c-AFN algorithm called RI-LSH for high-dimensional data. It uses a continuous searching strategy on each projection dimensions. We proved that our algorithm can return a c-approximate result efficiently and wouldn't break the theoretical guarantee. We conducted extensive performance evaluation against two c-AFN algorithms regarding I/O cost and result accuracy. The results show that our algorithm can achieve the best I/O performance.

- Considering processing problems on graphs, we formulated a new problem which can be applied to discover vertices which are not dominated by others in a multi-layer graph. To the best of our knowledge, the problem has not been formulated before. We developed a baseline algorithm to answer the skyline nearest neighbor queries, and then discussed the early- termination condition to improve the performance. We performed performance evaluation regarding on real-world graphs. The experiment results demonstrate that the optimizations can improve the performance significantly.

7.2 Future work

Even we have already made substantial improvements in this thesis, the similarity search problem is still far from being well solved. Our experimental work on Approximate MIPS problem opens a potential direction for further study. More specifically, LSH is also a popular scheme which is widely applied in Approximate MIPS algorithms and can provide theoretical guarantee for quality. After converting the whole dataset and query data into a new data space, LSH is used to solve NN problem in the new data space. There are already some works using LSH for Ap-

proximate MIPS problem but it is still possible to further improve the performance. For example, proposing new techniques to reduce errors when converting the dataset to a new data space, or adopting better LSH scheme to return results with higher accuracy.

Bibliography

[1] I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck, "Hierarchical hub labelings for shortest paths," in *European Symposium on Algorithms*. Springer, 2012, pp. 24–35.

[2] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 349–360.

[3] T. Akiba, C. Sommer, and K.-i. Kawarabayashi, "Shortest-path queries for complex networks: exploiting low tree-width outside the core," in *Proceedings of the 15th International Conference on Extending Database Technology*. ACM, 2012, pp. 144–155.

[4] K. Aoyama, K. Saito, H. Sawada, and N. Ueda, "Fast approximate similarity search based on degree-reduced neighborhood graphs," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 1055–1063.

[5] A. Arora, S. Sinha, P. Kumar, and A. Bhattacharya, "Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces," *Proceedings of the VLDB Endowment*, vol. 11, no. 8, pp. 906–919, 2018.

[6] A. Babenko and V. Lempitsky, "Additive quantization for extreme vector compression," in *Proceedings of the IEEE Conference on Computer Vision and*

Pattern Recognition, 2014, pp. 931–938.

[7] O. Beaumont, A.-M. Kermarrec, and É. Rivière, "Peer to peer multidimensional overlays: Approximating complex structures," in *International Conference On Principles Of Distributed Systems*. Springer, 2007, pp. 315–328.

[8] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: an efficient and robust access method for points and rectangles," in *Acm Sigmod Record*, vol. 19, no. 2. Acm, 1990, pp. 322–331.

[9] R. Bellman, R. Corporation, and K. M. R. Collection, *Dynamic Programming*, ser. Rand Corporation research study. Princeton University Press, 1957. [Online]. Available: https://books.google.com.au/books?id=wdtoPwAACAAJ

[10] J. L. Bentley, "Multidimensional binary search trees in database applications," *IEEE Transactions on Software Engineering*, no. 4, pp. 333–340, 1979.

[11] E. Bernhardsson, "Annoy at github https://github.com/spotify/annoy," 2015.

[12] B. Boden, S. Günnemann, H. Hoffmann, and T. Seidl, "Mining coherent subgraphs in multi-layer graphs with edge labels," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 1258–1266.

[13] M. Boguna, D. Krioukov, and K. C. Claffy, "Navigability of complex networks," *Nature Physics*, vol. 5, no. 1, pp. 74–80, 2009.

[14] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *Data Engineering, 2001. Proceedings. 17th International Conference on*. IEEE, 2001, pp. 421–430.

[15] L. Chang, J. X. Yu, L. Qin, H. Cheng, and M. Qiao, "The exact distance to destination in undirected world," *The VLDB Journal?The International Journal*

on Very Large Data Bases, vol. 21, no. 6, pp. 869–888, 2012.

[16] E. Chávez and E. S. Tellez, "Navigating k-nearest neighbor graphs to solve nearest neighbor searches," in *Mexican Conference on Pattern Recognition*. Springer, 2010, pp. 270–280.

[17] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *null*. IEEE, 2003, p. 717.

[18] ——, "Skyline with presorting: Theory and optimizations," in *Intelligent Information Processing and Web Mining*. Springer, 2005, pp. 595–604.

[19] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, "Reachability and distance queries via 2-hop labels," *SIAM Journal on Computing*, vol. 32, no. 5, pp. 1338–1355, 2003.

[20] R. R. Curtin, J. R. Cline, N. P. Slagle, W. B. March, P. Ram, N. A. Mehta, and A. G. Gray, "Mlpack: A scalable c++ machine learning library," *Journal of Machine Learning Research*, vol. 14, no. Mar, pp. 801–805, 2013.

[21] R. R. Curtin and A. B. Gardner, "Fast approximate furthest neighbors with data-dependent hashing," *arXiv preprint arXiv:1605.09784*, 2016.

[22] 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.

[23] T. Dean, M. A. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik, "Fast, accurate detection of 100,000 object classes on a single machine," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1814–1821.

[24] W. Dong, M. Charikar, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *WWW*, 2011.

[25] Erikbern, "Annoy," https://github.com/spotify/annoy.

[26] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *Journal of computer and system sciences*, vol. 66, no. 4, pp. 614–656, 2003.

[27] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[28] 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.

[29] J. Gao, H. V. Jagadish, B. C. Ooi, and S. Wang, "Selective hashing: Closing the gap between radius search and k-nn search," in *SIGKDD*, 2015.

[30] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi, "Dsh: data sensitive hashing for high-dimensional k-nnsearch," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 1127–1138.

[31] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2946–2953.

[32] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 12, pp. 2916–2929, 2012.

[33] Y. Gu, Y. Guo, Y. Song, X. Zhou, and G. Yu, "Approximate order-sensitive k-nn queries over correlated high-dimensional data," *IEEE Transactions on Knowledge & Data Engineering*, no. 1, pp. 1–1, 2018.

[34] R. Guo, S. Kumar, K. Choromanski, and D. Simcha, "Quantization based fast inner product search," in *Artificial Intelligence and Statistics*, 2016, pp. 482–490.

[35] S. M. Holland, "Principal components analysis (pca)," *Department of Geology, University of Georgia, Athens, GA*, pp. 30 602–2501, 2008.

[36] Q. Huang, J. Feng, Q. Fang, and W. Ng, "Two efficient hashing schemes for high-dimensional furthest neighbor search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2772–2785, 2017.

[37] 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.

[38] Q. Huang, G. Ma, J. Feng, Q. Fang, and A. K. Tung, "Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1561–1570.

[39] 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.

[40] ——, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 604–613.

[41] P. Jain and A. Kapoor, "Active learning for large multi-class problems," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 762–769.

[42] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[43] 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.

[44] M. Jiang, A. W.-C. Fu, R. C.-W. Wong, and Y. Xu, "Hop doubling label indexing for point-to-point distance querying on scale-free networks," *Proceedings of the VLDB Endowment*, vol. 7, no. 12, pp. 1203–1214, 2014.

[45] R. Jin, N. Ruan, Y. Xiang, and V. Lee, "A highway-centric labeling approach for answering distance queries on large sparse graphs," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 445–456.

[46] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *CoRR*, vol. abs/1702.08734, 2017.

[47] A. Karbasi, S. Ioannidis, and L. Massoulié, "From small-world networks to comparison-based search," *IEEE Transactions on Information Theory*, vol. 61, no. 6, pp. 3056–3074, 2015.

[48] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, 2000, pp. 163–170.

[49] J. M. Kleinberg, "Navigation in a small world," *Nature*, vol. 406, no. 6798, pp. 845–845, 2000.

[50] N. Koenigstein and Y. Koren, "Towards scalable and accurate item-oriented recommendations," in *Proceedings of the 7th ACM conference on Recommender systems*, 2013, pp. 419–422.

[51] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[52] W. Li, Y. Zhang, Y. Sun, W. Wang, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement (v1.0)," *CoRR*, vol. abs/1610.02455, 2016.

[53] Y. Lifshits and S. Zhang, "Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design," in *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2009, pp. 318–326.

[54] J. Liu, X. Yan, X. Dai, Z. Li, J. Cheng, and M.-C. Yang, "Understanding and improving proximity graph based maximum inner product search," *arXiv preprint arXiv:1909.13459*, 2019.

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

[56] 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.

[57] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 950–961.

[58] Y. A. Malkov and D. Yashunin, "Efficient and robust approximate nearest

neighbor search using hierarchical navigable small world graphs," *arXiv preprint arXiv:1603.09320*, 2016.

[59] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces," in *International Conference on Similarity Search and Applications*. Springer, 2012, pp. 132–147.

[60] ——, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Information Systems*, vol. 45, pp. 61–68, 2014.

[61] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *CoRR*, 2016.

[62] ——, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[63] J. Matoušek, "Computing dominances in e n," *Information Processing Letters*, vol. 38, no. 5, pp. 277–278, 1991.

[64] S. Morozov and A. Babenko, "Non-metric similarity graphs for maximum inner product search," in *Advances in Neural Information Processing Systems*, 2018, pp. 4721–4730.

[65] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2227–2240, 2014.

[66] B. Neyshabur and N. Srebro, "On symmetric and asymmetric lshs for inner product search," *arXiv preprint arXiv:1410.5518*, 2014.

[67] D. Ouyang, L. Qin, L. Chang, X. Lin, Y. Zhang, and Q. Zhu, "When hierarchy meets 2-hop-labeling: Efficient shortest distance queries on road networks," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 709–724.

[68] R. Pagh, F. Silvestri, J. Sivertsen, and M. Skala, "Approximate furthest neighbor with application to annulus query," *Information Systems*, vol. 64, pp. 152–162, 2017.

[69] J. Pan and D. Manocha, "Bi-level locality sensitive hashing for k-nearest neighbor computation," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*. IEEE, 2012, pp. 378–389.

[70] R. Panigrahy, "Entropy based nearest neighbor search in high dimensions," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, 2006, pp. 1186–1195.

[71] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 41–82, 2005.

[72] Y. Park, M. J. Cafarella, and B. Mozafari, "Neighbor-sensitive hashing," *PVLDB*, vol. 9, no. 3, pp. 144–155, 2015.

[73] J. Pei, D. Jiang, and A. Zhang, "On mining cross-graph quasi-cliques," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 228–238.

[74] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, 1990.

[75] A. Said, B. Fields, B. J. Jain, and S. Albayrak, "User-centric evaluation of a k-furthest neighbor collaborative filtering recommender algorithm," in *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 2013, pp. 1399–1408.

[76] A. Said, B. Kille, B. J. Jain, and S. Albayrak, "Increasing diversity through furthest neighbor-based recommendation," *Proceedings of the WSDM*, vol. 12, 2012.

[77] Y. Shan, J. Jiao, J. Zhu, and J. Mao, "Recurrent binary embedding for gpu-enabled exhaustive retrieval from billion-scale semantic vectors," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2170–2179.

[78] C. Sheng and Y. Tao, "On finding skylines in external memory," in *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2011, pp. 107–116.

[79] A. Shrivastava and P. Li, "Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips)," in *Advances in Neural Information Processing Systems*, 2014, pp. 2321–2329.

[80] ——, "Improved asymmetric locality sensitive hashing (alsh) for maximum inner product search (mips)," *arXiv preprint arXiv:1410.5410*, 2014.

[81] C. Silpa-Anan and R. I. Hartley, "Optimised kd-trees for fast image descriptor matching," in *CVPR*, 2008.

[82] 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.

[83] 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.

[84] C. Teflioudi and R. Gemulla, "Exact and approximate maximum inner product search with lemp," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 1, pp. 1–49, 2016.

[85] C. Teflioudi, R. Gemulla, and O. Mykytiuk, "Lemp: Fast retrieval of large entries in a matrix product," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 107–122.

[86] J. Wang and S. Li, "Query-driven iterated neighborhood graph search for large scale indexing," in *Proceedings of the 20th ACM international conference on Multimedia*, 2012, pp. 179–188.

[87] J. Wang, T. Zhang, N. Sebe, H. T. Shen *et al.*, "A survey on learning to hash," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 769–790, 2017.

[88] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A survey on learning to hash," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 769–790, 2018.

[89] Y. Wang, A. Shrivastava, and J. Ryu, "Flash: Randomized algorithms accelerated over cpu-gpu for ultra-high dimensional similarity search," *arXiv preprint arXiv:1709.01190*, 2017.

[90] L. Wanqi, W. Hanchen, and Z. Ying, "I-lsh: I/o efficient c-approximate nearest neighbor search in high-dimensional space," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019.

[91] F. Wei, "Tedi: efficient shortest path query answering on graphs," in *Graph Data Management: Techniques and Applications*. IGI Global, 2012, pp. 214–238.

[92] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *NIPS*, 2008, pp. 1753–1760.

[93] X. Yan, J. Li, X. Dai, H. Chen, and J. Cheng, "Norm-ranging lsh for maximum inner product search," in *Advances in Neural Information Processing Systems*, 2018, pp. 2952–2961.

[94] B. Yao, F. Li, and P. Kumar, "Reverse furthest neighbors in spatial databases," in *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 2009, pp. 664–675.

[95] H. Zhang, F. Shen, W. Liu, X. He, H. Luan, and T.-S. Chua, "Discrete collaborative filtering," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2016, pp. 325–334.

[96] J. Zhang, S. Khoram, and J. Li, "Efficient large-scale approximate nearest neighbor search on opencl fpga," in *CVPR*, 2018, pp. 4924–4932.

[97] Y. Zhang, H. Wang, D. Lian, I. W. Tsang, H. Yin, and G. Yang, "Discrete ranking-based matrix factorization with self-paced learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2758–2767.

[98] Y. Zheng, Q. Guo, A. K. Tung, and S. Wu, "Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 2023–2037.

[99] K. Zhou and H. Zha, "Learning binary codes for collaborative filtering," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012, pp. 498–506.