

# 海量高维数据相似性搜索研究

作者姓名 冯小康

指导教师姓名、职称 崔江涛 教授

申请学位类别 工学博士



学校代码 10701  
分 类 号 TP391

学 号 1403110189  
密 级 公开

# 西安电子科技大学

## 博士学位论文

### 海量高维数据相似性搜索研究

作者姓名：冯小康

一级学科：计算机科学与技术

二级学科（研究方向）：计算机系统结构

学位类别：工学博士

指导教师姓名、职称：崔江涛 教授

学 院：计算机科学与技术学院

提交日期：2020年6月



# **Similarity search on large-scale high-dimensional data**

A Dissertation submitted to  
**XIDIAN UNIVERSITY**  
in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy  
in Computer Architecture

By  
Feng Xiaokang  
Supervisor: Cui Jiangtao Title:Professor  
June 2020



西安电子科技大学  
学位论文独创性（或创新性）声明

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同事对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

学位论文若有不实之处，本人承担一切法律责任。

本人签名: 冯小康      日期: 2020.6.15

西安电子科技大学  
关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权属于西安电子科技大学。学校有权保留送交论文的复印件，允许查阅、借阅论文；学校可以公布论文的全部或部分内容，允许采用影印、缩印或其它复制手段保存论文。同时本人保证，结合学位论文研究成果完成的论文、发明专利等成果，署名单位为西安电子科技大学。

保密的学位论文在\_\_\_\_年解密后适用本授权书。

本人签名: 冯小康      导师签名: 徐江伟  
日期: 2020.6.15      日期: 2020.6.15



## 摘要

高维相似性搜索是实现海量非结构化数据有效处理与分析的一个关键基础问题，在数据库、信息检索、机器学习、推荐系统等相关领域具有广泛应用。相似性搜索是指从给定数据集中搜索和指定查询数据最相似数据的过程。实际场景中，根据相似性度量方式和对精度要求的不同，问题可分为多种形式。本文主要研究高维欧氏空间中的三个重要相似性搜索问题：精确最近邻搜索、近似最近邻搜索和近似最远邻搜索。

有效的相似性搜索需同时满足实时性和准确性两方面的要求，受非结构化大数据海量、高维特性的限制，各个方向的研究面临巨大挑战：(1) 精确最近邻搜索的关键是通过构造更紧致的距离下界提高无关数据过滤比例，现有最紧致的距离下界是一种数据自适应的距离下界，但是该方法存在距离下界计算复杂度高、聚类内部剪枝不精细的问题，限制了搜索速度的进一步提升；(2) 对于近似最近邻搜索，现有研究基于局部敏感哈希技术提出了很多先进的外存搜索方案，但是存在随机I/O普遍、候选点局部分布改善不充分的问题，I/O效率依旧较低；(3) 对于近似最远邻搜索，现有最先进的哈希方法和启发式方法都无法充分适应最远邻分布特性，搜索精度和效率无法有效提升。

本文首先深入分析了现有方案问题的成因，在此基础上设计了更加高效的索引结构与搜索算法，进一步提升海量高维数据相似性搜索的性能。主要研究工作如下：

(1) 对于高维精确最近邻搜索，提出两种优化方案克服现有方法的两个局限。第一，基于随机投影技术构造了一种快速估计方法，能够快速确定最优距离下界所在的边界，保证距离下界紧致性的同时提升计算速度；第二，为聚类内部数据点设计了一种距离下界，实现在聚类内部过滤无关数据。实验结果表明，新方法能够比现有方法降低20%的I/O开销和30%的CPU响应时间，有效提升精确最近邻搜索速度。

(2) 对于高维近似最近邻搜索I/O效率低的问题，首先，提出一种基于最优排序的局部敏感哈希索引方法改善近邻候选点的局部分布。通过引入空间填充曲线为复合哈希键值建立线序并排序，能够使近邻候选点更多地聚集在局部磁盘页面从而实现用快速的顺序I/O加载。为获得最好的局部分布，对多种常用空间曲线进行了量化分析，发现一类“邻域优先遍历”特性的曲线能够比基本排序方案产生更好的候选点局部分布，且排序性能更加稳定。经过对比，选取一种最优的“邻域优先遍历”曲线构建了新的排序方案。实验表明，新方案能进一步提升近似最近邻搜索的精度和I/O效率，实用性也得到增强。

(3) 然后，从提升单个磁盘页面的候选数据容量入手提出一种基于有序短判别码的局部敏感哈希索引方法，进一步降低高维近似最近邻搜索的I/O开销。现有方法普遍使用原始数据作为最近邻候选点，受限于原始数据较大的物理尺寸，搜索性能难以充分提升。短判别码具有体量小、相似度保持能力高的特点，本方法使用短判别码代替原始数据作为近邻候选数据，可以大幅提升单个磁盘页面能够容纳的候选数据数量。结合上述最优排序方法对短判别码重新排列，能够实现在磁盘局部用更少的I/O加载到足够多高质量近邻候选编码，保证搜索精度。在与现有先进方案的比较中，本方案能够取得最好的搜索精度，同时I/O成本和空间开销显著降低。

(4) 对于高维近似最远邻搜索，通过一系列实验观察，发现最远邻的分布具有一种“距离聚集效应”，是现有哈希方法性能局限的主要原因；另外发现不同数据集内最远邻点数量存在很大差异，与现有启发式方法在不同数据集上的性能差异有很强的相关性。基于以上观察，提出一种数据集自适应的最远邻搜索策略。首先改善了一种数据集难度快速评估方法，将数据集按照最远邻搜索难度分为三个档次。之后，针对不同难度的数据集适应性地改进了现有算法并提出了两个新的最远邻搜索算法。最后，将难度快速评估方法与三个最远邻搜索算法组合构造出一种数据集自适应的最远邻搜索方案。实验结果表明，新方案能够在不同数据集上取得最优搜索性能，且具有很高的实用性。

**关键词：**海量高维数据，维度灾难，高维索引，相似性搜索，最近邻搜索，最远邻搜索，距离下界，I/O性能瓶颈

## ABSTRACT

High-dimensional similarity search is a key fundamental component for effective processing and analysis of large-scale unstructured data, and has wide applications in database, information retrieval, machine learning, recommendation system and many other related areas. Given a query, similarity search refers to the process of searching the most similar data from a given data set to the query. In practical applications, this problem can be divided into various forms according to the similarity function and the different requirements on accuracy. This thesis mainly focuses on three important similarity search problems in high-dimensional Euclidean space: exact nearest neighbor (NN) search, approximate nearest neighbor (ANN) search and approximate furthest neighbor (AFN) search.

Effective similarity search needs to meet the requirements of both instantaneity and accuracy. Limited from the large scale and high dimensionality of massive unstructured data, current research in all the above three areas face great challenges. (1) The key of promoting exact NN search is to construct tighter distance lower bounds to lift the filtration rate of irrelevant data. Currently, the tightest one is a kind of data adaptive lower bound. However, it suffers from a high lower bound computation complexity and inefficient pruning inside clusters during the NN search, which draw strict limits on lifting the search speed. (2) For ANN search, a lot of advanced external solutions have been proposed based on locality sensitive hashing (LSH), however, they still suffer from unsatisfactory I/O efficiency due to large number of random I/O consumption or inadequate improvement on the local distribution of NN candidates. (3) Hashing methods and heuristic methods are the most advanced solutions for AFN search. However, neither of them can well adapt to the distribution of furthest neighbors to effectively improve the accuracy and efficiency of AFN search.

In this thesis, we first conduct in-depth explorations on the causes of the above issues, and then design more efficient index structures and search algorithms to further improve the performance of large-scale high-dimensional similarity search. The main contributions of this thesis are summarized as follows:

(1) For high-dimensional exact NN search, we design two new methods to address the limitations of the current method. Firstly, we design a fast estimation method to quickly determine the hyperplane bound providing the best lower bound of each cluster. In this way, we can ensure the compactness of the lower bounds as well as accelerating the lower bound computing. Secondly, we construct lower bounds for the member points of each cluster, which

can help to filter irrelevant points inside clusters. Experimental results show that our new methods can effectively speedup the exact NN search. Compared with existing solutions, the I/O consumption and the CPU response time can be reduced by 20% and 30%, respectively.

(2) In order to improve the I/O efficiency of high-dimensional ANN search, we firstly propose a novel LSH index called Optimal Order LSH (O2LSH) to improve the local distribution of NN candidates. By introducing space-filling curves to sort the compound LSH keys and rearrange the original data accordingly, NN candidates can be stored on the same or adjacent disk pages so that we can load them via sequential I/O operations. To enquire the best local distribution, a thorough quantitative analysis is conducted on several common space-filling curves. The results show that the “neighborhood-first-traverse” curves can generate better local distribution than the row-wise curve conducted in the basic sorting-based method. By comparison, we choose the best “neighborhood-first-traverse” curve for O2LSH. Based on that, O2LSH further improves both the I/O efficiency and accuracy of ANN search and exhibits better practicality according to the experimental results.

(3) Secondly, we propose a new LSH scheme SortingCodes-LSH(SC-LSH) to further reduce the I/O cost of ANN search by enlarging the capacity of NN candidates of each disk page. Most existing ANN solutions use original data points as NN candidates, the relatively large physical size of original data would draw strict limitations on the improvement of ANN search performance. In SC-LSH, we use discriminative short codes instead of original data as NN candidates. Benefited from the high space efficiency and good similarity-preserving ability of discriminative short codes, SC-LSH can greatly increase the number of NN candidates held in a single disk page. Then we introduce the above sorting method into SC-LSH to rearrange the storage of the codes. In this way, we can access sufficient candidate codes to guarantee the ANN search accuracy via much less sequential I/Os. Experimental evaluations show that SC-LSH can achieve the best search accuracy compared with the state-of-the-arts and also significantly reduce the I/O consumption and space consumption.

(4) For high-dimensional AFN search, we conduct a series of observations on the distribution of furthest neighbor points, and find that there exist a “distance aggregation phenomenon” in the distribution of furthest neighbors, which we claim is the main cause of the limitation of existing hashing methods. Also, we find that the numbers of furthest neighbor points vary a lot in different data sets, which has a strong correlation with the performance differences of existing heuristic methods on different data sets. Based on these findings, we propose a data set adaptive strategy for AFN search. Firstly, by improving an AFN search difficulty evaluating approach, we divide the data sets into three difficulty levels. Then we improve an

## ABSTRACT

---

existing AFN search algorithm and design two new algorithms to do AFN search on different difficulty levels of data sets. Combining the improved difficulty evaluating approach with the three algorithms, we propose a data set adaptive AFN search solution. Experimental results show that it can achieve the best AFN search performance on different data sets and exhibits good practicability.

**Keywords:** large-scale high-dimensional data, curse of dimensionality, high-dimensional index, similarity search, nearest neighbor search, furthest neighbor search, distance lower bound, I/O performance bottleneck



## 插图索引

图 1.1 从图像检索到最近邻搜索 . . . . .	2
图 3.1 基于分隔超平面构建的距离下界 . . . . .	19
图 3.2 基于距离下界过滤无关数据 . . . . .	24
图 3.3 比较HB和HB+方法的选择能力 . . . . .	25
图 3.4 变化 $m$ 对HB+性能的影响 . . . . .	27
图 3.5 HB方法和HB+方法的性能比较 . . . . .	29
图 3.6 不同高维精确最近邻搜索方法的性能比较 . . . . .	30
图 4.1 现有外存索引中候选点分布对比 . . . . .	33
图 4.2 几种典型的空间填充曲线 . . . . .	35
图 4.3 二维空间近邻分布对比（一） . . . . .	36
图 4.4 二维空间近邻分布对比（二） . . . . .	37
图 4.5 空间曲线近邻搜索量化对比 . . . . .	39
图 4.6 O2LSH的一个哈希表 . . . . .	42
图 4.7 O2LSH与SK-LSH近邻局部分布对比 . . . . .	43
图 4.8 O2LSH与SK-LSH性能对比 . . . . .	46
图 4.9 ANN搜索ratio对比 . . . . .	50
图 4.10 ANN搜索I/O开销对比 . . . . .	50
图 4.11 ANN搜索平均性能对比 . . . . .	51
图 4.12 空间开销对比 . . . . .	51
图 5.1 通过查距离表计算AQD . . . . .	58
图 5.2 不同候选数据加载方式对比 . . . . .	60
图 5.3 索引构建 . . . . .	61
图 5.4 ratio的比较 . . . . .	66
图 5.5 I/O开销的比较 . . . . .	66
图 5.6 ART的比较 . . . . .	66
图 5.7 空间开销比较（可视化展示） . . . . .	68
图 6.1 新的评估标准下现有方法的性能 . . . . .	72
图 6.2 最近邻与最远邻距离比 . . . . .	74

图 6.3	远邻共享率观察 . . . . .	81
图 6.4	最远邻搜索性能对比（容易数据集） . . . . .	87
图 6.5	最远邻搜索性能对比（中等难度数据集） . . . . .	87
图 6.6	最远邻搜索性能对比（困难数据集） . . . . .	88

## 表格索引

表 3.1	本章所使用的符号及其含义 . . . . .	18
表 3.2	不同方法在各数据集上的参数设置 . . . . .	28
表 4.1	近邻分布统计 . . . . .	36
表 4.2	Gray曲线与Hilbert曲线优势次数比较 . . . . .	40
表 4.3	实验数据集 . . . . .	45
表 4.4	不同ANN搜索方法在各数据集上的参数选取 . . . . .	49
表 4.5	ANN搜索平均ratio (数值结果) . . . . .	49
表 4.6	ANN搜索平均I/O (数值结果) . . . . .	51
表 4.7	总空间开销相对比例 . . . . .	52
表 4.8	索引结构空间开销相对比例 . . . . .	52
表 5.1	参数设置 . . . . .	65
表 5.2	QALSH与SC-LSH的I/O开销比较 . . . . .	67
表 5.3	空间开销比较 (数值结果) . . . . .	68
表 6.1	数据集内的最近邻数量 . . . . .	74
表 6.2	远邻搜索难度系数评估 . . . . .	77
表 6.3	RQALSH*(All)在Gist100K上的调参结果 . . . . .	78
表 6.4	RQALSH*(All)在Color上的调参结果 . . . . .	79
表 6.5	RQALSH*(All)在Trevi上的调参结果 . . . . .	79
表 6.6	RQALSH*(All)在Audio上的调参结果 . . . . .	79
表 6.7	RQALSH*(All)在中等和困难数据集上的调参结果 . . . . .	80
表 6.8	数据集信息 . . . . .	84
表 6.9	部分方法通过调参确定的参数值 . . . . .	86



## 目录

摘要 .....	I
ABSTRACT .....	III
插图索引 .....	VII
表格索引 .....	IX
第一章 绪论 .....	1
1.1 研究背景 .....	1
1.2 问题定义及应用 .....	2
1.3 现有相似性搜索存在的问题 .....	4
1.4 研究内容与贡献 .....	5
1.5 论文组织结构 .....	6
第二章 相关研究工作 .....	7
2.1 高维精确最近邻搜索 .....	7
2.1.1 距离下界 .....	7
2.1.2 基于降维的方法 .....	8
2.1.3 基于量化的方法 .....	9
2.1.4 基于嵌入的方法 .....	10
2.2 高维近似最近邻搜索 .....	10
2.2.1 外存搜索面临的挑战 .....	10
2.2.2 局部敏感哈希LSH .....	11
2.2.3 基于LSH的外存近似最近邻搜索方案 .....	11
2.3 高维近似最远邻搜索 .....	13
2.3.1 早期研究 .....	13
2.3.2 现有基于哈希的方法 .....	14
2.3.3 现有启发式方法 .....	15
第三章 基于聚类的高维精确最近邻搜索下界优化方法 .....	17
3.1 概述 .....	17
3.2 对现有HB方法的分析 .....	18
3.2.1 基于分隔超平面的距离下界 .....	19
3.2.2 更好地定位分隔超平面 .....	20
3.2.3 HB方法的局限 .....	21
3.3 基于聚类技术的下界优化方法HB+ .....	21

---

3.3.1	加速计算距离下界 .....	21
3.3.2	聚类内部高效剪枝 .....	23
3.3.3	新的最近邻搜索算法 .....	24
3.3.4	复杂度分析 .....	26
3.4	实验结果与分析 .....	26
3.4.1	实验设置 .....	27
3.4.2	降维数 $m$ 的影响 .....	27
3.4.3	HB与HB+的比较 .....	28
3.4.4	同其他先进方法的比较 .....	28
3.5	本章小结 .....	30
<b>第四章</b>	<b>针对近似最近邻搜索的基于最优排序的局部敏感哈希索引 .....</b>	<b>31</b>
4.1	概述 .....	31
4.2	基于空间曲线进一步改善I/O性能 .....	33
4.2.1	现有外存索引的I/O性能 .....	33
4.2.2	空间曲线及其特性 .....	34
4.2.3	最优空间线序 .....	35
4.3	O2LSH .....	40
4.3.1	线序定义 .....	40
4.3.2	构建索引结构 .....	41
4.3.3	最近邻搜索算法 .....	42
4.3.4	算法有效性分析 .....	43
4.3.5	算法复杂度分析 .....	44
4.4	实验结果与分析 .....	44
4.4.1	实验设置 .....	45
4.4.2	O2LSH与SK-LSH性能对比 .....	45
4.4.3	ANN搜索综合对比 .....	48
4.5	本章小结 .....	52
<b>第五章</b>	<b>针对近似最近邻搜索的基于有序短判别码的局部敏感哈希索引 .....</b>	<b>55</b>
5.1	概述 .....	55
5.2	一种先进的短判别码技术 .....	57
5.3	SC-LSH .....	58
5.3.1	方法概述 .....	58
5.3.2	有效性分析 .....	59
5.3.3	构建索引结构 .....	61

## 目录

---

5.3.4 最近邻搜索算法 .....	62
5.3.5 复杂度分析 .....	63
5.4 实验结果与分析 .....	64
5.4.1 实验设置 .....	64
5.4.2 ANN搜索性能对比 .....	65
5.5 本章小结 .....	68
<b>第六章 一种数据集自适应的高维最远邻搜索策略 .....</b>	<b>69</b>
6.1 概述 .....	69
6.2 现有方法的性能 .....	71
6.2.1 重新评估现有方法 .....	71
6.2.2 现有方法性能欠缺的原因 .....	73
6.3 数据集自适应的最远邻搜索方案 .....	75
6.3.1 数据集难度评估 .....	76
6.3.2 容易等级数据集上的最远邻搜索 .....	77
6.3.3 中等难度数据集：基于远邻共享的多中心方法 .....	78
6.3.4 困难数据集：基于近邻图的最远邻搜索方法 .....	82
6.4 实验结果与分析 .....	83
6.4.1 实验设置 .....	83
6.4.2 容易数据集上的比较 .....	86
6.4.3 中等难度数据集上的比较 .....	86
6.4.4 困难数据集上的比较 .....	88
6.4.5 实验小结 .....	88
6.5 本章小结 .....	89
<b>第七章 总结与展望 .....</b>	<b>91</b>
7.1 研究总结 .....	91
7.2 未来展望 .....	92
<b>参考文献 .....</b>	<b>95</b>
<b>致谢 .....</b>	<b>105</b>
<b>攻博期间取得的研究成果和参与的科研项目 .....</b>	<b>107</b>



# 第一章 绪论

## 1.1 研究背景

当前信息化社会，大数据已成为国家的重要战略资源，并正在渗透到经济发展、国家安全、社会稳定和人民生活的诸多方面<sup>[1]</sup>。据国际数据公司IDC预计，2025年全球每年被创建、采集或复制的数据总量（也称数据圈，Datasphere）将从2018年的33ZB（ $10^{21}$ ）增长到175ZB，增长5倍以上。其中，中国数据总量增长最为迅速，预计到2025年将占全球数据圈的27.8%，成为全球最大的数据圈<sup>[2]</sup>。大数据中主体多为半结构化和非结构化数据（如文本、图像、音频和视频数据等），占总采集量的85%以上<sup>[3]</sup>，蕴含巨大价值。不过非结构化大数据海量、异构和混杂等特性以及爆炸式的增长速度也给信息存储、计算以及面向各种应用的数据处理技术带来了前所未有的挑战。

为充分挖掘利用非结构化大数据的价值，需要能够对海量非结构化数据进行高效地处理与分析，相似性搜索作为其中的一个关键基础问题，具有重要研究意义。相似性搜索是指从给定数据集合中搜索出与指定查询数据最相似数据的过程。原始非结构化数据由于结构上的复杂性难以直接计算，常用手段是为其提取特征向量（一般具有很高的维度，几百到几千不等），将问题转化为海量高维特征集合上的相似性搜索问题。

图1.1以图像数据为例展示了这一过程。可以看到，为了从大规模图像集合中搜索出和查询图像相似的图像，原始图像集合和查询图像都要先经过特征提取，然后再执行某种形式的相似性搜索。在图1.1的场景中，执行的是最近邻搜索。在其他应用场景下，根据相似性度量方式的不同，相似性搜索问题也会有不同形式。另外，根据实际场景对搜索精度要求的不同，相似性搜索也会分为精确搜索和近似搜索两类。本文着重研究高维欧氏空间中的三个重要相似性搜索问题：精确最近邻搜索、近似最近邻搜索和近似最远邻搜索。

进行相似性搜索最基本的方案是线性扫描（Linear Scan），即计算查询对象与所有数据对象之间的相似性，返回满足特定查询请求的数据对象。虽然是线性复杂度，但是受非结构化数据海量、高维（或模糊高维）等特性的巨大挑战，该方法会变得十分耗时：一方面，实际数据海量的特性使得算法需要计算很多次相似性；另一方面，数据的高维特性使得每一次相似性计算都十分耗时。为满足实际应用实时性的搜索要求，需要进一步探索适合海量、高维特性的高效数据索引与搜索算法。

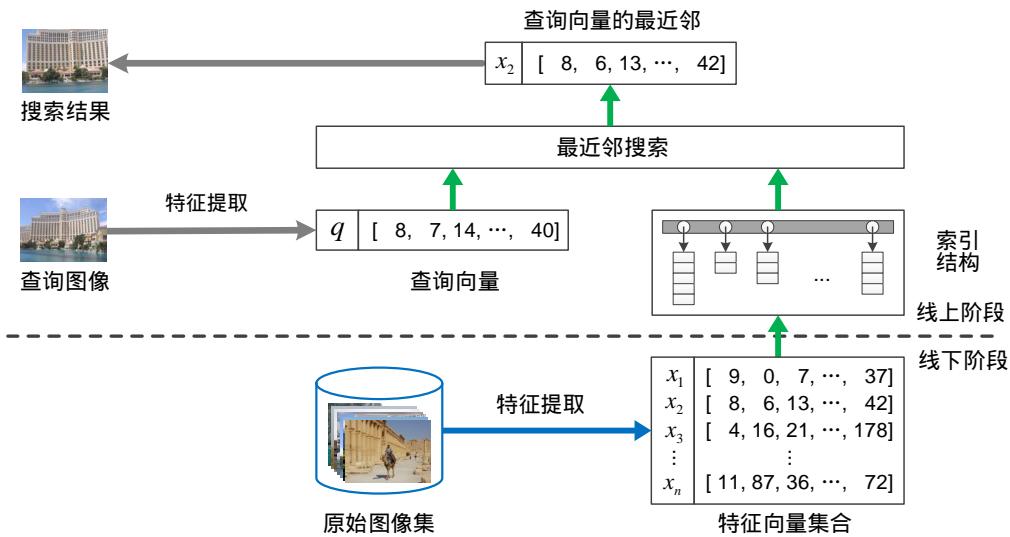


图 1.1 从图像检索到最近邻搜索

## 1.2 问题定义及应用

高维最近邻 (Nearest Neighbor, 简称NN) 搜索的定义如下：给定一个包含 $n$ 个 $d$ 维特征向量的数据集 $D \in \mathcal{R}^d$ , 令 $\|\cdot\|$ 代表两个点间的欧氏距离, 对于查询点 $q \in \mathcal{R}^d$ , 最近邻搜索是指从 $D$ 中寻找特征向量 $x^*$ , 使得

$$x^* = \arg \min_{x \in D} \|q, x\| \quad (1-1)$$

查询点 $q$ 通常由用户指定。

以上问题也被称作精确最近邻搜索, 因为返回的是距离 $q$ 真正最近的数据点 $x^*$ 。有些问题场景允许不返回精确结果, 或者追求精确结果对实际应用的准确率影响不大, 相似性搜索可以变成近似最近邻 (Approximate Nearest Neighbor, 简称ANN) 搜索问题, 即从数据集 $D$ 中搜索返回数据点 $x$ ,  $x$ 与 $q$ 的距离尽可能近但不一定非得是 $x^*$ 。还有一些情况, 用户要求返回前 $k$ 个与 $q$ 距离最小的数据点, 问题会变成 $k$ -最近邻 ( $k$ -Nearest Neighbor, 简称 $k$ -NN) 搜索。

高维最近邻搜索是一个拥有广泛应用需求的基础性问题, 其应用领域包括但不限于多媒体数据库<sup>[4-6]</sup>、信息检索<sup>[7]</sup>、机器学习<sup>[8]</sup>、模式识别<sup>[9-12]</sup>、推荐系统<sup>[13-15]</sup>等等<sup>[16, 17]</sup>。比如, 各大搜索引擎的图像搜索功能、淘宝的搜图购等, 底层都可以归约为高维图像特征向量上的最近邻搜索问题; 机器学习领域,  $k$ -NN搜索是基于实例的学习 (instance-based learning) 方法中的一种<sup>[8]</sup>; 在模式识别中,  $k$ -NN作为一种“非参数”的算法, 能够用于解决分类和回归问题<sup>[9]</sup>; 在数据挖掘领域, 数据点到其第 $k$ -最近邻的距离可被用作一种估计局部密度的方式, 是异常检测领域中的一个很流行的异常点评分指标<sup>[10, 11]</sup>。

高维最远邻搜索是高维最近邻搜索的对偶问题，形式化的定义如下：给定一个包含 $n$ 个 $d$ 维特征向量的数据集 $D \in \mathcal{R}^d$ ，对于用户给定的查询点 $\mathbf{q} \in \mathcal{R}^d$ ，最远邻搜索是指从 $D$ 中寻找数据点 $\mathbf{x}^*$ ，使得

$$\mathbf{x}^* = \arg \max_{\mathbf{x} \in D} \|\mathbf{q}, \mathbf{x}\| \quad (1-2)$$

类似地，根据对搜索精度要求的不同，也可以分为精确最远邻搜索和近似最远邻（Approximate Furthest Neighbor，简称AFN）搜索两种问题形式。如果用户要求返回前 $k$ 个与 $\mathbf{q}$ 距离最大的数据点，问题会变成 $k$ -最远邻（ $k$ -Furthest Neighbor，简称 $k$ -FN）搜索。

高维最远邻搜索主要有三方面研究意义。第一，它是很多计算机重要问题的子问题。比如最大生成树（Maximum Spanning Tree）或计算数据集直径问题<sup>[18]</sup>、全连接聚类（Complete Linkage Clustering）问题<sup>[19]</sup>以及非线性降维（Non-Linear Dimensionality Reduction）问题<sup>[20]</sup>等。

第二，最远邻搜索在一些实际应用中具有重要作用。比如在推荐领域，基于最远邻搜索构建的推荐方法能够提供更加多样的推荐结果，弥补传统基于近邻搜索缺乏新意的问题<sup>[21, 22]</sup>。具体地，在基于协同过滤的推荐系统中，为了给用户 $u$ 推荐新的项（item），传统做法是寻找和 $u$ 最相似的一些用户，将他们喜欢的项推荐给 $u$ 。相似用户会有很多相似的喜好。但问题是，有时候新推荐的项用户 $u$ 已经知道了，只是没有对其打分而已，那么在 $u$ 看来，这份推荐就缺乏新意。Alan Said等人基于最远邻搜索构造了一种新的推荐逻辑：先寻找和用户 $u$ 最不相似的用户 $v$ ，将 $v$ 不喜欢的项推荐给 $u$ 。实验表明，这种方式确实能够找到很多新的有效推荐（即用户原来不知道却很喜欢的项），获得了很好的用户满意度。

最远邻搜索还可用于改善聚类和异常检测效果。文献<sup>[23]</sup>将最远邻搜索和最近邻搜索结合构造了一种新的聚类和异常检测方法，既能降低计算复杂度又能提升聚类和异常检测的精度。另外，在分布式虚拟资源管理系统中，最远邻搜索能够寻找资源需求互补的工作任务（workload），将这些任务一起打包分配到同一个虚拟机能够避免资源冲突<sup>[24]</sup>，不过这个问题场景是否属于高维搜索范畴跟描述虚拟机资源所使用的特征总量有关。

第三，高维远邻搜索提供了一个新的角度理解高维数据。一直以来，高维数据的不易理解性是相似性搜索研究一个主要的障碍。人们对于低维空间能够直观地感受和理解，这为在低维空间进行相似性搜索研究提供了有力的指导。但是很多研究<sup>[25, 26]</sup>发现，相似性搜索在高维空间会遇到维度灾难问题，传统低维索引技术在高维空间并不高效，必须专门针对高维数据设计适合的索引结构，因此探寻高维数据内部的规律变得十分重要。

最远邻搜索在研究过程中发现了很多高维数据内部新的现象，对于进一步理解高维数据有很大帮助。比如，现有性能最好的最远邻搜索方法DrusillaSelect和RQALSH\*，其核心思想都是基于Curtin等人观察到的最远邻与大的数据模长之间具有强相关性的现象<sup>[27, 28]</sup>。本文在第六章对最远邻搜索的研究中也发现了一些新的现象。其中，观察到最远邻的分布具有一种“距离聚集效应”，能够解释现有哈希方法性能不足的原因；观察到一些数据集存在“邻近点共享远邻”的现象，基于此构造了性能更好的最远邻搜索方法。

### 1.3 现有相似性搜索存在的问题

高维相似性搜索最基本的方案是线性扫描，但是计算量太大，难以满足实时性要求。为提升搜索性能，现有研究大都遵循一种“过滤+精炼”（Filter and Refine）的算法设计策略，将搜索过程分为无关数据过滤和候选点精炼两个阶段分别进行突破。但是随着数据规模越来越大，搜索精度要求越来越高，现有方案在各个环节凸显出越来越多的局限。在进一步提升搜索性能方面，各个研究方向面临的主要挑战如下：

- 高维精确最近邻搜索除了要克服数据高维、海量特性带来的挑战，还要满足100%搜索精度的严格要求，其性能提升的关键是构造更加紧致的距离下界提升过滤能力，从而提升搜索速度。传统方法基于超矩形或者超球边界构造的距离下界<sup>[26, 29–31]</sup>，因为不够紧致，过滤效率较低。超平面边界方法(Hyperplane Bound, HB)<sup>[32]</sup>基于聚类技术构造了一种数据自适应的距离下界，是现有最紧致的距离下界。不过该方法需要 $O(K^2)$ 的复杂度计算距离下界（ $K$ 是聚类个数），开销较大。此外，HB只能过滤无关聚类，无法快速过滤聚类内部的无关数据。以上这两点都限制了搜索速度的进一步提升。
- 对于高维近似最近邻搜索，局部敏感哈希（Locality Sensitive Hashing，简称LSH）<sup>[33–35]</sup>因为具有出色的误差保证和较高的计算效率是目前使用最广泛的技术。现有研究基于LSH技术提出了很多先进方案，如LSB-Forest<sup>[36]</sup>、C2LSH<sup>[37]</sup>、SRS<sup>[38]</sup>以及QALSH<sup>[39]</sup>等，但存在一个共同问题：这些方案确立的近邻候选点大都随机分布在磁盘上，为了保证查询精度，需要耗费大量昂贵的随机I/O才能加载到足够多近邻候选点，容易使候选点加载环节成为整个搜索过程的速度瓶颈。为解决这一问题，有研究者提出基于排序的LSH索引方案SK-LSH<sup>[40]</sup>，通过引入空间填充曲线引导数据重新排列，可以使近邻候选点更多地分布在局部，从而实现用更加快速的顺序I/O加载近邻候选点。但是该方案所使用的row-wise曲线并不能充分改善近邻候选点的局部分布。整体上，现有近似最近邻搜索的I/O效

率依旧较低。

- 对于高维近似最近邻搜索，现有最先进的方法主要包括随机哈希方法（包括QDAFN<sup>[41, 42]</sup> 和RQALSH<sup>[43, 44]</sup>）和启发式方法（包括DrusillaSelect<sup>[27, 28]</sup> 和RQALSH\*<sup>[43, 44]</sup>）两类。不过这两类方法都无法充分适应最近邻的分布特性。随机哈希方法是一类数据无关的方法，其主要原理是基于数据点间的距离计算出的概率判断最近邻候选点质量。但我们的观察发现，最近邻点在距离上比较接近，会使得随机哈希方法需要付出更多代价才能准确辨别最近邻。启发式方法是一类数据依赖的方法，基于Curtin等人发现的最近邻与大的模长有很强相关性的规律设计的搜索方案，能够以很少的计算量获得很高的精度。但我们的观察发现，不同数据集内部最近邻点数量差异很大，该规律只适合最近邻点很少的数据集，对于最近邻较多的数据集，现有启发式方法取得好的搜索性能依旧很难。

## 1.4 研究内容与贡献

本文在国家自然科学基金等项目的资助下研究进一步提升海量高维数据相似性搜索的性能。首先对现有研究存在的问题进行了原因探究，在此基础上研究设计了更加高效的相似性搜索方案，主要研究内容与贡献如下：

(1) 对于高维精确最近邻搜索，分别研究了一种距离下界快速计算方法和聚类内部高效剪枝机制克服HB方法上述的两个局限。首先，基于随机投影技术设计了一种距离快速估计方法，配合一种聚类边界部分选择策略，能够快速确定最优距离下界所在的聚类边界，在保证距离下界紧致性的同时提升距离下界计算速度；之后，为聚类内部成员点构造了一种距离下界，能够帮助在聚类内部更加精细地剪枝无关数据，提升过滤比例。将以上二者结合提出了一种新的精确最近邻搜索方案HB+，实验结果表明新方法能够有效提升精确最近邻搜索速度。相对于现有方案，能够降低20%的I/O开销和30%的CPU响应时间。

(2) 对于高维近似最近邻搜索，首先，提出一种基于最优排序的局部敏感哈希索引O2LSH进一步增强近邻候选点的局部分布质量。不同于基本排序方案SK-LSH，该方案寻求能够产生更好近邻局部分布的曲线。为此，对典型空间曲线进行了量化分析，发现：1) 基本排序方案使用的row-wise曲线具有“维度优先遍历”的特性，容易导致最近邻搜索的多种局限；2) 另一类“邻域优先遍历”特性的曲线能够产生更好的候选点局部分布，且排序性能更加稳定。经过对比，选取出一种最优的“邻域优先遍历”曲线构建了新的排序方案。实验表明，新方案能够进一步提升近似最近邻搜索的I/O效率和精度，且实用性进一步增强。

(3) 然后，对现有近似最近邻搜索方案各个环节的有效性进行了全面考察，发现

现有方案使用原始点作为候选数据会降低最近邻搜索多个环节的效率。原始点较高的维度一方面会限制排序方案I/O性能的提升上限；另一方面，昂贵的原始距离也会增加候选点精炼环节的计算开销。本文提出一种基于有序短判别码的新LSH索引方案SC-LSH。新方案使用体量更小的短判别码代替原始数据作为近邻候选数据，能够大幅提升单个磁盘页面所容纳的候选数据数量；同时，短判别码较高的相似度保持能力能够很好地保证候选点精炼环节的精度。实验表明，新方案仅用少量顺序I/O即可加载到充分多的候选数据，同现有先进方案相比，能够取得最好的搜索精度，同时I/O开销和空间消耗分别仅为其他方案的10%-30%和3%-20%。

(4) 对于高维近似最远邻搜索，本文首先进一步研究了最远邻的分布特性。通过实验观察，发现最远邻的分布具有一种“距离聚集效应”，是现有哈希方法性能局限的主要原因；另外发现不同数据集内最远邻点数量存在很大差异，与现有启发式方法在不同数据集上的不同性能表现有很强的相关性。基于以上发现，本文提出一种数据集自适应的最远邻搜索策略，提倡针对不同数据集内部的最远邻分布特性设计搜索算法。按照该策略，首先将数据集按照最远邻分布特性划分成容易、居中、困难三个难度档次。然后，针对不同难度的数据集分别适应性地改良了现有算法并设计了两个新的最远邻搜索算法。最后，将三个最远邻搜索算法与一个数据集难度快速评估方法结合组成了一个数据集自适应的最远邻搜索方案。实验结果表明，该方案能够很好地适应不同数据集上的最远邻分布特性，在各种数据集上能够取得最好的搜索性能，且具有很好的实用性。

## 1.5 论文组织结构

本文第一章介绍本文的研究背景、研究内容和主要贡献等。

第二章分别对精确最近邻搜索、近似最近邻搜索和近似最远邻搜索的国内外研究现状进行梳理和总结。

第三章介绍对精确最近邻搜索的研究。针对HB方法的两个局限，分别提出一种距离下界加速计算算法和聚类内部剪枝策略，进一步提升精确近邻搜索速度。

第四章和第五章介绍本文对近似最近邻搜索的研究。分别从增强近邻候选点的局部分布质量和提高单个磁盘容纳候选数据的数量入手提出了两个方法，进一步提升近似最近邻搜索的I/O效率以及其他性能。

第六章研究近似最远邻搜索，在深入分析最远邻分布特性的基础上提出一种数据集自适应的最远邻搜索方案。研究了不同数据集上进行最远邻搜索的难度，并适应性地改良或提出新的搜索算法，进一步提升了最远邻搜索性能。

第七章对本文工作进行总结，分析本文工作的不足并指出下一步研究方向。

## 第二章 相关研究工作

最近邻搜索是相似性搜索领域中一个很有代表性的问题，一直被广泛关注。经过几十年的研究，最近邻搜索在精确搜索和近似搜索两方面都取得了很大进展。一方面，有力地推动了很多实际问题的解决；另一方面，也促进了很多其他形式相似性搜索问题的研究，如最近对搜索、最远对搜索、最大内积搜索等。作为最近邻搜索的对偶问题，最远邻搜索也有很大的研究和应用价值。早期工作借鉴了很多最近邻搜索领域的思想，近期工作发现了一些新的高维数据分布规律，能够帮助研究者进一步理解高维数据。

本章详细介绍高维精确最近邻搜索、高维近似最近邻搜索和高维近似最远邻搜索三个问题的研究现状，分别见第2.1节、第2.2节和第2.3节。每个部分会先介绍一些必要的预备知识，之后再详细介绍具体研究进展。

### 2.1 高维精确最近邻搜索

高维精确最近邻搜索主要面临三大挑战，除了非结构化数据高维、海量特性的限制，还要满足100%精度的严格要求，具有很高的研究难度。早期工作提出了很多基于数据分区<sup>[29–31, 45, 46]</sup>或空间分区<sup>[47, 48]</sup>的树形索引结构，很好地解决了低维空间中的精确最近邻搜索问题<sup>[25]</sup>。不过随着维度的升高，这些树形方法的性能会随着维数的增加而急速下降，在维度超过某个阈值（10维左右）时甚至不如线性扫描<sup>[25, 26]</sup>。这一“维度灾难”（curse of dimensionality）现象让研究者意识到高维空间和低维空间有很大的不同，开始重点关注能够有效适应高维空间特性的索引结构和搜索算法。

#### 2.1.1 距离下界

精确最近邻搜索的主要研究目标是提升搜索速度，经过多年探索，研究者们凝练出一种“过滤+精炼”（Filter and Refine）的算法设计策略。该策略将最近邻搜索分为两个环节：1) 过滤阶段，通过某种方式快速过滤大量无关数据（即距离查询点较远的点），留下一个体量较小的候选集；2) 精炼阶段：通过一些更加精细的计算（通常也更加耗时，比如计算原始距离）从候选集中找出真正的最近邻。对于精确最近邻搜索来说，这一策略的关键是提升无关点的过滤比例，既要快速又要准确，通常通过构造更加紧致的距离下界（lower bound）来实现。

距离下界是指算法为快速评估数据点间的原始距离而构造的一种估计距离，这种估计距离要小于等于原始距离，否则难以保证搜索的准确性。在过滤过程中，距

离下界一般通过如下方式发挥作用：假设算法目前为查询点 $\mathbf{q}$ 找到的最好最近邻是 $\mathbf{x}_{nn}$ ，如果 $\mathbf{q}$ 与下一个数据点 $\mathbf{x}_i$ 的距离下界 $LB(\mathbf{q}, \mathbf{x}_i) > \|\mathbf{q}, \mathbf{x}_{nn}\|$ ，说明 $\mathbf{x}_i$ 不可能比 $\mathbf{x}_{nn}$ 更靠近 $\mathbf{q}$ ，则 $\mathbf{x}_i$ 可以被安全过滤而不用被加载计算原始距离。如果所有数据点是按距离下界升序排列的，则搜索算法可以在此时停止。好的距离下界应该在满足以上两点要求的基础上尽可能的接近原始距离，也就是所谓的“更加紧致”，下界越紧致，过滤效果越好。继续以上面的场景为例，假设 $\mathbf{q}$ 到当前 $\mathbf{x}_{nn}$ 的距离是10，下一个待验证数据 $\mathbf{x}_i$ 到 $\mathbf{q}$ 的原始距离是15。算法A1计算出的 $\mathbf{x}_i$ 的下界是5，则算法A1无法过滤掉 $\mathbf{x}_i$ ；算法A2计算出的 $\mathbf{x}_i$ 的下界是12，则算法A2可以成功过滤 $\mathbf{x}_i$ 。这就是紧致下界的优勢。

在上述原则的指导下，现有研究围绕降维、量化和嵌入等手段提出了很多高维精确最近邻搜索方法，取得了很大进展。不同方法的区别主要体现在构建距离下界所采用的方式不同。

### 2.1.2 基于降维的方法

降维（dimensionality reduction）是克服维度灾难最直接的一个思路<sup>[49]</sup>。通过将数据从高维降到低维空间进行索引，能够有效降低处理的复杂度<sup>[50]</sup>。主分量分析（Principal Component Analysis，简称PCA）是一种常用的线性降维手段，能够将数据集主要的信息浓缩到少量维度之中，通过舍弃能量较低的维度即可实现降维。由于PCA的距离保持特性，在剩余维度计算的距离恰好满足距离下界的两个要求，从而可以被用于快速过滤无关点。基于该思路主要提出了两种精确最近邻搜索方法<sup>[51]</sup>：全局降维（Global Dimensionality Reduction，简称GDR）和局部降维（Local Dimensionality Reduction，简称LDR）。GDR方法将数据集整体直接降低到少量维度之中，如果这些维度能够保留数据集的大部分信息，则在低维空间计算的距离就能够很好地逼近原始距离<sup>[50]</sup>，这种情况在数据集整体上具有很强相关性时会出现。但Chakrabarti等人指出，较强的全局相关性在实际应用中很少见，相对而言，数据的局部相关性更为普遍。因此，他们提出一种LDR方法<sup>[51]</sup>，首先对数据集进行分区，然后在每个分区内单独进行降维。

还有一种特殊的降维手段叫一维映射（one-dimensional mapping），通过将原始数据映射到一维数值进行最近邻搜索。通常，数据点的一维映射值之差可以直接作为距离下界，而一维数值天然的可排序性使得可以用高效的树形索引结构（如B+树）进行索引<sup>[52]</sup>，从而实现近邻候选点的快速定位。iDistance<sup>[53]</sup>是一维映射的一个代表方法，该方法将数据集进行分区并为每个分区选取一个参考点（reference point），然后根据数据点到参考点的距离将其映射到一维。iDistance的主要问题是性能对参考点的选择很敏感，并且需要消耗较多随机I/O操作<sup>[54]</sup>。

### 2.1.3 基于量化的方法

基于量化（quantization）的方法主要思路是将原始数据转化为一种物理尺寸更小的数据进行表示，从而降低处理负担，加快处理速度。根据转化方式的不同可以分为标量量化和矢量量化两种。标量量化的代表是向量近似文件（Vector Approximation file，简称VA-file）方法<sup>[26]</sup>。该方法通过为所有维度划分区间将整个高维空间划分成非常多的超矩形胞腔（hyper-rectangular cell），然后将数据集中的每个点量化为其所落入的胞腔，每个胞腔可以用一个固定长度的比特串表示，尺寸比原始数据小很多。这种量化方式十分简单，但在高维空间产生了很好的效果。举例，假设每个维度被划分为4个区间（每个区间可以用2个比特表示），在d维空间，总共将产生 $4^d$ 个胞腔。假设某数据集包含 $n=1,000,000$ 个数据点，在 $d=2$ 维空间，划分出的胞腔数远远小于数据点数量，即 $4^2 \ll n$ ，量化后的数据难以通过胞腔相互区分；但如果在高维空间，比如 $d=128$ ，胞腔数将远大于数据集规模，即 $4^{128} \gg n$ 。在数据点均匀分布的情况下，两个点落入相同胞腔的概率几乎为零，这意味着，可以使用量化后的胞腔数据对数据点进行区分。此外，这种量化方式带来的数据压缩比也很可观。假设原始数据每一维分量是一个4字节的整数，量化之后每一维只用2个比特即可表示，压缩了16倍。

VA-file利用超矩形胞腔规整的形状可以十分方便地计算距离下界，此外，基于超矩形胞腔还能计算出数据点之间距离的上界（即不小于原始距离的估计距离）。有了距离上下界可以实现不用原始距离也能排除无关数据。举例，如果查询点 $\mathbf{q}$ 到一个数据点 $\mathbf{x}$ 的距离上界小于到另一个数据点 $\mathbf{y}$ 的距离下界，即 $UB(\mathbf{q}, \mathbf{x}) < LB(\mathbf{q}, \mathbf{y})$ ，则 $\mathbf{y}$ 不会成为 $\mathbf{q}$ 的最近邻，可以直接被过滤掉。VA-file使用线性扫描的方式执行最近邻搜索，利用超矩形胞腔的上述性质，能够快速进行无关点过滤和候选点精炼，加速扫描过程。

VA-file提出之后有多个工作对其进行了扩展和改进<sup>[6]</sup>。Berchtold等人将树结构与向量近似文件结合提出一种IQ-tree<sup>[55]</sup>方法，能够进一步提升加快搜索。Ferhatosmanoglu等人认为VA-file中关于数据均匀分布的假设并不实际，他们提出一种VA<sup>+</sup>-file方法主张先将原始数据转换到PCA空间再进行量化，能够进一步提升量化精度<sup>[56]</sup>。Cui等人认为，在数据集规模较大时，VA-file线性的复杂度依然会比较耗时，此外，VA-file还存在随机I/O开销较多的问题。他们提出一种PCVA方法，结合主分量分析技术进行一维投影，能够实现在磁盘局部用顺序I/O快速加载向量近似数据，进一步提升最近邻搜索的I/O效率和搜索速度<sup>[57]</sup>。

矢量量化（vector quantization）是另一种量化技术，能够将一个d维原始向量 $\mathbf{x}$ 量化到一个d维的码字(codeword)向量 $\mathbf{c}_i$ 。 $\mathbf{c}_i$ 通常来自一个集合 $C = \{c_i | c_i \in \mathcal{R}^d, 1 \leq i \leq K\}$ ，也称量化码书（codebook）<sup>[58]</sup>，K是码书的容量。Ramaswamy等人基于矢量量

化技术提出一种精确最近邻搜索方法HB（Hyperplane Bound）<sup>[32]</sup>。不同于VA-file的标量量化，HB基于矢量量化能够利用到不同维度间的相关性取得更高的量化精度。具体地，HB使用Kmeans进行聚类，利用聚类间的分隔超平面构建了一种自适应的距离下界，要比传统超球的边界（如iDistance）和超矩形的边界（如VA-file）更加贴合数据的实际分布，因而更加紧致。不过HB方法存在下界计算复杂度高和聚类内部过滤不充分的问题，限制了搜索速度的提升。本文针对HB的这两个问题，提出了两个新的措施针对性地进行克服，进一步提升了搜索速度。详细见本文第三章的工作。

### 2.1.4 基于嵌入的方法

Hwang等人在CVPR’12上提出一种新的距离下界技术，非线性嵌入（nonlinear embedding）方法<sup>[59]</sup>。他们通过一种特殊的方式将原始向量嵌入到一个低维空间，并且证明两个向量在低维嵌入空间的欧氏距离要严格小于等于原始距离，因此可以用作距离下界。基于该技术，可以在低维嵌入空间进行快速的无关点过滤，加速最近邻搜索。具体地，该方法首先将一个原始向量平均分成若干子向量，然后为每个子向量计算其分量的均值和标准差，最后将所有子向量的均值和标准差拼接在一起作为嵌入向量。不过我们的研究显示，该方法的I/O性能受 $k$ （即要求返回的最近邻个数）的影响很大。 $k=1$ （即搜索最近邻）时性能很好， $k=100$ 时，I/O开销变得很大。详细见第三章的实验分析。Li等人在DASFAA’18提出了一种复合嵌入（compounded embedding）方法<sup>[60]</sup>。通过将原始向量分为两部分，分别进行非线性嵌入和线性嵌入，构造出了更加紧致的距离下界，进一步提升了最近邻搜索速度。

## 2.2 高维近似最近邻搜索

不同于精确最近邻搜索，高维近似最近邻（Approximate Nearest Neighbor，简称ANN）搜索允许在一定的精度误差范围内寻求近似解，一定程度上能够换取搜索速度的提升。常用的一个ANN搜索问题定义如下：给定一个包含 $n$ 个 $d$ 维特征向量的数据集 $D \subset \mathcal{R}^d$ ，令 $\|\cdot\|$ 代表两个点间的欧氏距离。对于查询点 $\mathbf{q} \in \mathcal{R}^d$ ，ANN查询是指从 $D$ 中搜索到一点 $\mathbf{x}$ ，使得 $\|\mathbf{q}, \mathbf{x}\| \leq c\|\mathbf{q}, \mathbf{x}^*\|$ 。其中 $\mathbf{x}^*$ 是 $\mathbf{q}$ 在 $D$ 中的真实最近邻， $c > 1$ 是近似搜索的比率。

### 2.2.1 外存搜索面临的挑战

现有ANN搜索主要面临非结构化数据海量、高维特性带来的挑战。高维的挑战前面已经介绍。海量特性的主要影响是容易引发I/O性能瓶颈。因为海量数据集一般体量巨大，难以放入内存，外存存储成为更加合理的选择。但是内外存间存在巨

大的速度差距，使得二者之间的通信（即I/O）变得十分昂贵，如果通信次数过多，或者外存访问方式不合理，会使得这一环节成为整个近邻搜索最耗时的部分，称之为I/O性能瓶颈<sup>[61]</sup>。

现有外存ANN搜索方案主要包括局部敏感哈希（Locality Sensitive Hashing, LSH）<sup>[33-35]</sup>系列技术、乘积量化（Product Quantization, PQ）<sup>[62-66]</sup>技术以及近邻图（Proximity Graph或Nearest Neighbor Graph, 简称NNG）<sup>[16, 17, 67-71]</sup>技术等。其中，LSH由于具有出色的误差保证和较高的计算效率受到越来越多的关注，是目前解决ANN搜索使用最为广泛的技术。本文重点关注基于LSH技术进一步提升外存ANN搜索的性能，包括搜索精度和效率。接下来将详细分析和介绍这方面的研究工作<sup>1</sup>。

### 2.2.2 局部敏感哈希LSH

LSH技术最早由Indyk和Motiwani等人提出并应用于ANN搜索<sup>[33-35]</sup>，基本定义为：令 $B(\mathbf{q}, r)$ 表示 $d$ 维空间中以 $\mathbf{q}$ 为中心点，半径为 $r$ 的超球。给定近似比率 $c$ 和两个概率值 $p_1, p_2$ ，其中 $p_1 > p_2$ ，对于 $\mathcal{R}^d$ 空间中任意两点 $\mathbf{x}, \mathbf{q}$ ，如果函数族 $H = \{h : \mathcal{R}^d \rightarrow Z\}$ 满足两个条件，则称 $H$ 是 $(r, cr, p_1, p_2)$ -敏感的：1) 若 $\mathbf{x} \in B(\mathbf{q}, r)$ ，则 $Pr[h(\mathbf{q}) = h(\mathbf{x})] \geq p_1$ ；2) 若 $\mathbf{x} \notin B(\mathbf{q}, cr)$ ，则 $Pr[h(\mathbf{q}) = h(\mathbf{x})] \leq p_2$ 。式2-1是一种欧氏空间中常用的LSH函数：

$$h(\mathbf{x}) = \frac{\lfloor \mathbf{a} \cdot \mathbf{x} + b \rfloor}{W} \quad (2-1)$$

首先生成一个 $d$ 维向量 $\mathbf{a}$ ，每个分量独立服从标准正态分布 $N(0, 1)$ 随机产生，将 $\mathbf{a}$ 所在直线等分成宽度为 $W$ 的区段（interval）。对于数据集 $D$ 中任意一点 $\mathbf{x}$ ，将其往 $\mathbf{a}$ 上投影，并沿着 $\mathbf{a}$ 的方向偏移 $b$ 的距离，投影点最终落入区段的编号即作为 $\mathbf{x}$ 最终的哈希值 $h(\mathbf{x})$ 。其中， $b$ 是一个随机数，服从 $[0, W)$ 上的均匀分布。对于式2-1，相距 $s$ 的两个点 $\mathbf{x}_1, \mathbf{x}_2$ 发生碰撞（落入相同区段）的概率可用如下公式计算：

$$\begin{aligned} p(s) &= \Pr[h(\mathbf{x}_1) = h(\mathbf{x}_2)] \\ &= \int_0^W \frac{1}{s} f_2\left(\frac{t}{s}\right) \left(1 - \frac{t}{W}\right) dt \end{aligned} \quad (2-2)$$

其中 $f_2(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ 。该公式表明，桶宽 $W$ 固定时， $\mathbf{x}_1, \mathbf{x}_2$ 发生碰撞的概率随二者之间距离 $s$ 的增加而单调递减。因此称函数 $h(\mathbf{x})$ 是 $(r, cr, p_1, p_2)$ -敏感的哈希函数，这里 $p_1 = p(r), p_2 = p(cr)$ 。

### 2.2.3 基于LSH的外存近似最近邻搜索方案

LSH函数具有距离保持的特性，能够让原始空间中邻近的点以较大的概率碰撞，原始距离越大，碰撞概率越小。在单个LSH函数上，取与查询点 $\mathbf{q}$ 落入相同区段的点作为近邻候选点，可以成功过滤掉很多无关数据点。但一些距离 $\mathbf{q}$ 较远的点也能

<sup>1</sup>关于ANN搜索其他方向的研究，可以参考Li等人所做的实验综述工作<sup>[16, 17]</sup>。

与之碰撞，这些点被称作假阳性点（False Positive，简称FP），会降低搜索的准确率。为提升过滤能力，通常从 $H$ 中随机生成 $m$ 个LSH函数组合成一个复合哈希函数 $G(\cdot) = (h_1(\cdot), h_2(\cdot), \dots, h_m(\cdot))$ 进行联合过滤。对于数据点 $\mathbf{x}$ ,  $G(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_m(\mathbf{x}))$ 被称作 $\mathbf{x}$ 的复合哈希键值。如果 $\mathbf{x} \in B(\mathbf{q}, r)$ , 则 $Pr[G(\mathbf{x}) = G(\mathbf{q})] \geq p_1^m$ , 如果 $\mathbf{x} \notin B(\mathbf{q}, cr)$ , 则 $Pr[G(\mathbf{x}) = G(\mathbf{q})] \leq p_2^m$ 。因此 $G(\cdot)$ 是 $(r, cr, p_1^m, p_2^m)$ -敏感的。

复合哈希函数能有效解决假阳性问题，但也会产生另外一个问题，假阴性现象：随着哈希函数的增多，一些原本较近的点可能在部分哈希函数上无法发生碰撞而被错误地排除掉，称为假阴性点（False Negative，简称FN）。假阴性点增多会降低近邻的召回率，同样会降低查询精度。通常解决办法是，建立 $L$ 组复合哈希函数，多个复合函数间用“或”的关系，即数据点只要在一个复合函数上发生碰撞就被认作候选点。通常，数据集连同其在一组复合哈希函数上的键值集合构成一个哈希表，多组复合哈希函数意味着多个哈希表的存储，如果所需构建的哈希表过多，则空间开销会被显著增大。

基于以上近邻候选点机制发展出了很多ANN搜索方案，搜索精度和效率不断提升。不过，早期的LSH方案<sup>[34, 72, 73]</sup>主要面向内存，且需要构建大量哈希表，这在实际海量高维数据集上并不实际。LSB-Forest<sup>[36]</sup>是第一个面向外存的LSH索引结构，通过将复合哈希键值转化为一个二进制的z-order值，实现在一个B树索引上适应多种搜索半径，极大地降低了索引规模。C2LSH<sup>[37]</sup>采用动态碰撞计数的方法筛选候选点，根据两个点复合哈希键值间的碰撞次数来衡量它们距离的远近，取得了更高的搜索精度。SRS<sup>[38]</sup>致力于建立tiny级的索引结构，该方法借助原始距离与投影空间距离的对应关系，实现在少量（一般少于10个）哈希函数上进行候选点的初步筛选，虽然I/O开销不是很大，但精度损失会比较多<sup>[74]</sup>。QALSH<sup>[39]</sup>指出以前方法的LSH映射机制存在“无视查询点”（query oblivious）的问题，查询点通常不位于碰撞区间的中心，会使得相同质量的候选点与查询点发生碰撞的机会有可能不相同。该方法设计了一种能够感知查询点位置的LSH映射机制，沿用了C2LSH的碰撞计数机制，进一步提升了近邻搜索性能。

现有外存LSH索引技术在解决ANN查询问题时有着各自的优势，但存在一个共同问题：加载候选点会消耗较多的随机I/O操作。众所周知，在外存存储结构中随机I/O读取十分耗时，且一次随机I/O通常只能加载到一个近邻候选点，使得ANN搜索的I/O利用率较低。针对这一问题，Liu等人提出一种基于键值排序的LSH方案SK-LSH<sup>[40]</sup>方法，通过引入空间填充曲线为复合哈希值建立线序关系能够让更多近邻候选点分布在局部，从而实现用顺序I/O加载近邻候选点，改善了ANN搜索的I/O性能。

## 2.3 高维近似最远邻搜索

同最近邻搜索类似，最远邻搜索也分为精确搜索和近似搜索两种情况。对于精确最远邻搜索，低维空间中已经有很好的方法，但是低维空间的有效方法难以扩展到高维，而高维精确最远邻搜索本身则具有十分大的难度。现有突破主要集中在高维近似最远邻搜索问题上。

### 2.3.1 早期研究

二维空间的精确最远邻搜索可以使用最远点维诺图技术（Furthest-point Voronoi Diagram）解决，只消耗线性复杂度的空间和对数复杂度的搜索时间<sup>[75]</sup>。但是随着维度的增大，维诺图的空间开销会呈指数级增长，在高维空间很不实用。高维空间精确最远邻搜索的难度可以通过如下方法规约：给定数据集合  $D \subset \{-1, 1\}^d$  和一个查询向量  $\mathbf{q} \in \{-1, 1\}^d$ ，欧氏空间中的， $-\mathbf{q}$  在  $D$  中的最远邻将会和  $\mathbf{q}$  具有最小汉明距离。也就是说， $d$  维实数空间中的精确最远邻搜索至少和  $d$  维汉明空间中的精确最近邻搜索一样难，而后者在高维空间则是一个十分困难的问题<sup>[41, 76]</sup>。

同近似最近邻搜索一样，近似最远邻搜索由于放松了精度限制，也有利于克服维度灾难问题。常用的问题定义是一种  $c$ -近似最远邻搜索问题：给定一个包含  $n$  个  $d$  维特征向量的数据集  $D \in \mathcal{R}^d$ ，令  $\|\cdot\|$  代表两个点间的欧氏距离，对于查询点  $\mathbf{q} \in \mathcal{R}^d$ ， $\mathbf{x}^*$  是  $\mathbf{q}$  在  $D$  中的真实最远邻， $c$ -近似最远邻搜索（简称  $c$ -AFN）是指从  $D$  中寻找数据点  $\mathbf{x}$ ，使得

$$\|\mathbf{q}, \mathbf{x}\| \geq \|\mathbf{q}, \mathbf{x}^*\|/c \quad (2-3)$$

其中  $c > 1$  是近似搜索比率。

高维近似最远邻搜索的研究大致可以分为两个阶段。早期的工作主要包括 Agarwal 等人提出的方法<sup>[77]</sup> 和 Bespamyatnikh 提出的方法<sup>[78]</sup>，但这两个方法查询时间都随维度  $d$  呈指数递增关系，耗时较高。 Indyk 在 SODA'03 上提出了第一个较为实用 AFN 搜索算法<sup>[79]</sup>。自此以后，越来越多实用的 AFN 算法相继被提出，包括 Pagh 等人提出的 QDAFN 方法<sup>[41, 42]</sup>、Curtin 等人提出的 DrusillaSelect 方法<sup>[27, 28]</sup>，以及 Huang 等人提出的 RQALSH 和 RQALSH\* 方法<sup>[43, 44]</sup>。这些方法从技术本质上可以分为两类。一类是基于随机哈希的方法（random hashing based methods，简称哈希方法），另一类是基于数据分布的启发式方法（heuristic methods，简称启发式方法）。

关于近似最远邻搜索的相关工作，Sivertsen 的博士论文<sup>[80]</sup> 和 Huang 的博士论文<sup>[81]</sup> 从时间顺序的角度进行了很好的梳理，是十分值得推荐的参考。本章换一个角度，从技术本质的不同对现有工作进行分类介绍和分析。

### 2.3.2 现有基于哈希的方法

基于哈希的方法主要使用一种基于点积的随机投影哈希技术：首先随机生成一个 $d$ 维随机投影向量 $\mathbf{a}$ ,  $\mathbf{a}$ 的每个分量根据标准正态分布 $N(0, 1)$ 随机生成, 对于数据点 $\mathbf{x} \in R^d$ , 将其在 $\mathbf{a}$ 上的投影值 $\mathbf{a} \cdot \mathbf{x}$ 作为哈希值。这种哈希技术有两大优点, 一是计算快速, 二是便于进行理论分析。基于这种随机哈希技术一共发展出了三个方案, 它们都能提供理论保证。

Indyk的方法<sup>[79]</sup>是第一个基于哈希的方法, 克服了以往最远邻搜索时间对维度指数依赖的问题。该方法使用多个随机投影哈希构建索引结构, 然后将最远邻搜索划分为两个阶段来解决。首先, 定义了一个近似最远邻搜索的中间问题版本 $(R, c)$ -AFN搜索: 给定搜索半径 $R$ , 如果存在数据点到 $\mathbf{q}$ 的距离大于 $R$ , 算法需要返回一个距离 $\mathbf{q}$ 至少 $R/c$ 的点; 否则, 任意返回一个点。Indyk的索引结构能够直接支持 $(R, c)$ -AFN搜索。接着, 结合二分搜索 (binary search), Indyk能够将一个 $(c + \delta)$ -AFN问题规约到一系列 $(R, c)$ -AFN搜索来解决, 从而实现近似最远邻搜索, 这里 $\delta$ 是一个很小的常量。

Pagh等人发现多随机投影哈希能直接用于解决 $c$ -AFN问题, 不需要经过中间问题。他们在Indyk方法索引结构的基础上提出了一个更加简单直接的近似最远邻搜索方案QDAFN (Query Dependent Approximate Furthest Neighbor)<sup>[41, 42]</sup>。基本原理是根据投影差值大小判断最远邻候选点质量: 如果两个点的投影值差距很大, 它们很有可能在原始空间距离很远。基于此, QDAFN将那些在投影方向上与查询点投影值差距大的点作为最远邻候选点优先验证。具体地, 在索引阶段, QDAFN首先生成 $L$ 个随机投影哈希对数据集 $D$ 进行投影, 然后在每个投影方向上对投影值进行排序, 并将投影值最大的 $M$ 个数据点保存。查询时, 重复如下过程最多 $M$ 次: 在 $L$ 个投影方向之间同时比较, 优先将与 $\mathbf{q}$ 具有最大投影值差距的数据点作为最远邻候选点, 加载并验证。QDAFN是一个有理论保证的方法, 能够自动设置参数,  $L = 2n^{1/c^2}$ ,  $M = 1 + e^2 L \log^{c^2/2-1/3} n$ 。在该取值下, 对于任意 $c > 1$ , QDAFN能以至少0.72的概率返回 $c$ -AFN。不过, 参照Pagh等人在<sup>[42]</sup>的实验设置, 也可以按照需要手动设置 $L, M$ 的取值。

Huang等人在ICDE’17和TKDE’17上提出了第三个哈希方法<sup>[43, 44]</sup>, 一种反转会引导的局部敏感哈希方法 (Reverse Query-Aware LSH, 简称RQALSH)。具体地, Huang等人调整了原始LSH函数的定义以适应最远邻搜索。他们将概率值由原来的随数据点距离增大单调递减变成单调递增, 新的LSH函数称为反转LSH函数 (Reverse LSH, 简称RLSH), 概率值被称为分离概率。RQALSH也需要构建多个随机哈希索引。查询时, 算法会从大到小缩小分离半径 $R$ , 在每个分离半径下统计每个数据点与查询点的分离次数, 作为衡量最远邻候选点质量的依据。当一个数据点与查询点的

分离次数超过给定阈值时，会被加载并验证。与以往方案不同的是，RQALSH第一次考虑了外存环境下的最远邻搜索，分别给出了外存和内存版本的算法实现。为了实现外存高效管理，RQALSH使用B+树管理每个投影函数上的哈希值。此外，RQALSH基于虚拟重哈希技术，只存储一套多索引哈希结构就能支持不同大小的分离半径，能够节省空间开销。

综合来看，现有哈希方法一个共同点是利用随机哈希函数上的投影差距寻找最远邻候选点，投影差距本质上是对原始距离的一种近似，所以现有哈希方法本质上是一种基于距离的方法。这使得哈希方法具有一个潜在的局限：如果存在较多点与查询点的距离十分接近，将增加哈希方法辨别的难度。我们在第六章对远邻点的距离分布进行了观察，发现距离查询点较远的点具有一种“距离聚集效应”，即对于给定的查询点，存在很多点在距离上与查询点到最远邻的距离十分接近，这就使得哈希方法需要付出更多代价才能有效鉴别出真实最远邻。

### 2.3.3 现有启发式方法

启发式最远邻搜索主要包含两个方法，DrusillaSelect和RQALSH\*。它们都是基于Curtin等人对最远邻分布的一个实验观察发展而来。Curtin等人观察发现<sup>2</sup>，数据集中的数据点能否成为最远邻与它的模长有很大关系，在很多情况下数据集中的最远邻都具有很大的模长<sup>3</sup>，或者说，最远邻都是分布在数据集外围的一些点<sup>[81]</sup>。基于这一发现，Curtin等人提出了他们的最远邻搜索算法DrusillaSelect。核心思想十分简单：离线阶段从数据集中选出少量模长较大的点构成一个公共的最远邻候选集，搜索时只遍历这个候选集即可。这一思路看起来比较简单，但是效果却很显著。根据Huang等人的评估（见文献<sup>[81]</sup>表4-5），在Trevi数据集<sup>4</sup>上搜索前10最远邻，DrusillaSelect只消耗30个I/O可以取得1.0187的ratio准确率<sup>5</sup>。

决定DrusillaSelect性能的关键是候选点的选取。算法首先会选择一些模长较大的点作为投影向量，然后根据一个预先定义的打分机制选出一些被投影向量“很好地代表”（“well-represented”）的点加入候选集，此处“被投影向量‘很好地代表’”可以理解成某种程度上非常靠近投影向量。关于如何选到最好的候选点，Curtin等人发现很难从理论上给出指导，因此DrusillaSelect是一个启发式的方法，通过自由调节两个参数确定最优性能：一个是投影向量个数 $L$ ，一个是每个投影向量上选出的候选点个数 $M$ ，因此，总的候选点数量为 $LM$ 。此外，Curtin等人认为质量好的最远邻候选点应该是比较分散的。因此他们给算法又设置了一个角度阈值：每次在当前投影向

<sup>2</sup>详细的实验过程见文献<sup>[28]</sup>第5节。

<sup>3</sup>确切地说，是中心模长，即以数据集质心为原点计算出的模长，如无特殊说明，本文后续出现的模长均指代中心模长。

<sup>4</sup>一个包含99,000个4,096维图像特征的数据集。

<sup>5</sup>一种衡量最远邻搜索准确率的指标，详细定义见第六章实验设计部分。

量附近选出候选点之后需要隔开一定的角度选择新的投影向量，所有与当前投影方向夹角小于该阈值的点都会被舍弃，Curtin等人推荐的角度是 $\pi/8$ 。

Huang等人提出的启发式方法RQALSH\*在DrusillaSelect框架的基础上进行了两方面改进。第一，他们为候选集建立了索引。Huang等人认为，当候选集体量很大时直接遍历并不高效，因此他们设置了一个阈值：如果候选集体量小于该阈值则直接遍历；否则，将为候选集建立一个RQALSH索引来提升搜索效率。Huang等人给出的一个参考阈值是100。第二，他们调整了候选点的选取原则。相对于Curtin等人，Huang等人更加青睐模长大的点，认为大的模长是成为最近邻候选点很重要的因素。为此，他们不再设置角度阈值，因为隔开一定的角度选取新投影向量会丢失一些模长大的点；此外，他们修改了打分机制，将原先DrusillaSelect定义的基于绝对值误差的打分方式改为基于平方误差，这样同等情况下也能优先保留模长更大的点。

现有启发式方法主要有两方面的局限。第一，算法的有效性主要建立在Curtin等人发现的最近邻与大模长有很强相关性的现象之上。但是本文的观察发现，该现象的适用范围是有限的。在最近邻点很少的数据集上能够取得很好的效果，但是在最近邻较多的数据集上适应性并不是很好。第二，现有方法的调参效率不高。DrusillaSelect和RQALSH\*都有两个自由参数，但是没有给出设置最优参数的指导原则。实际部署时通常使用网格搜索（Grid Search）的调参方式确定最优参数，比较耗时，影响算法的实用性。

## 第三章 基于聚类的高维精确最近邻搜索下界优化方法

精确最近邻搜索由于要满足100%搜索精度的要求，很难避免原始距离的计算。高维空间原始距离的计算十分耗时，因此相关研究十分重视提升算法对无关数据的过滤能力。近年来，国内外学者提出了很多最近邻搜索方案，本质上都是通过设计更加紧致的距离下界获得更高的过滤能力。基于聚类技术构造的自适应距离下界由于利用了维度间的相关性，能够获得比传统方法更加紧致的距离下界，在最近邻搜索时过滤更多无关数据。但该方案存在距离下界计算复杂度高、聚类内部过滤不充分的问题，严重限制了最近邻搜索速度的提升。本章在现有基于聚类的距离下界的基础上，研究了新的方法针对性地克服该方案的局限，以进一步提升最近邻搜索性能。

### 3.1 概述

高维最近邻（Nearest Neighbor，简称NN）搜索是很多非结构化数据应用领域的一个基础性关键问题。受非结构化数据海量、高维特性的挑战，线性扫描方法难以提供令人满意的速度，有效的高维最近邻搜索通常需要设计高效的索引结构和搜索算法。高维索引与精确最近邻搜索算法已经有几十年的研究历史。早期工作提出了很多基于树形结构的方法<sup>[29–31, 45–48]</sup>，但是只在低维空间有效，在高维空间会遇到维度灾难问题<sup>[25, 26]</sup>。之后，研究者基于降维<sup>[51, 53]</sup>、量化<sup>[26, 32, 55–57]</sup>和嵌入<sup>[59]</sup>等手段提出了针对高维空间特性的索引结构和搜索算法，取得了很大进展。这些方法遵循一种“过滤+精炼”(Filter & Refine)的算法设计策略，核心是构造紧致的上下界距离进行有效过滤，不同方法能力的差距主要体现在构造上下界的方式不同。传统上，很多方法<sup>[26, 29–31, 45, 53]</sup>基于超矩形或者超球构造距离下界，由于不能有效适应高维数据的分布，很难获得较好的紧致性。Ramaswamy等人提出的HB(Hyperplane Bound)方法<sup>[32]</sup>，利用数据维度间的相关性信息构造了一种数据自适应的距离下界，能够获得比超矩形、超球边界更加紧致的距离下界，是目前过滤能力最高的高维精确最近邻搜索方法。

本章对HB方法进行了深入分析，发现HB方法存在两个局限严重限制了近邻搜索性能的有效提升。第一，HB存在距离下界计算复杂度高的问题。使用HB方法在线搜索最近邻时，需要首先计算出查询点到所有聚类的距离下界，但这部分的计算复杂度很高，是 $O(K^2)$ ，其中 $K$ 是聚类个数。由于HB方法需要增加聚类个数获得更加紧致的距离下界，因为增加聚类个数能够减小聚类粒度，从而让距离下界更接近

原始距离。这就导致HB方法在获得紧致下界和提升计算速度之间形成了一个矛盾。第二，HB方法无法在聚类内部进行高效的无关数据过滤。HB方法提供的距离下界只能用于无关聚类的剪枝，但对于聚类内部存在的无关数据，HB没有提供任何剪枝方法。如果一个聚类确定被加载，算法将加载该聚类的所有成员点并计算昂贵的原始距离寻找最近邻，这也给最近邻搜索带来很大的负担。

针对HB方法的两个局限，本章提出一种新的最近邻搜索方法HB+，进一步提升高维精确最近邻搜索性能。首先，基于随机投影技术和部分选择策略设计了一种快速估计方法，加速距离下界的计算。之后，为聚类内部成员点构造了一种距离下界，实现聚类内部无关数据的过滤。在三个高维视觉特征数据集上进行了综合的实验评估，结果显示新的HB+方法能够比HB方法减少近20%的I/O开销和30%以上的CPU运算开销。同时，与其他先进方法（包括VA-file<sup>[26]</sup>、iDistance<sup>[53]</sup>和FNN<sup>[59]</sup>）相比，HB+也取得了最好的最近邻搜索性能。

本章余下内容组织如下：首先在第3.2节概要地介绍现有HB方法的关键技术并分析其局限。然后在第3.3节设计新的方法克服HB方法的局限，并给出新的最近邻搜索算法。第3.4节进行实验评估和分析。最后在第3.5节总结本章工作。

## 3.2 对现有HB方法的分析

本节首先对现有HB方法的关键技术进行一些概要介绍，然后讨论分析HB的局限，从而引出本章的研究内容。表3.1中汇总了本章将要使用到的符号及其含义。需要特别指出的是，本章使用欧氏距离度量特征向量之间的相似性。

表 3.1 本章所使用的符号及其含义

符号	含义
$D$	数据集
$K$	聚类数量
$H(\mathbf{n}, b)$	一个超平面
$\mathbf{q}$	查询点
$\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2, \dots$	数据集中的点
$\{C_i\}_{i=1}^K$	为数据集产生的 $K$ 个聚类
$\{\mathbf{c}_i\}_{i=1}^K$	$K$ 个聚类的中心
$H_{ij}$	聚类 $C_i, C_j$ 之间的超平面边界
$d(\mathbf{x}, H)$	数据点 $\mathbf{x}$ 与超平面 $H$ 的距离
$d(H, C)$	聚类 $C$ 与其超平面边界 $H$ 的最小距离
$LB(\mathbf{x}, C)$	数据点 $\mathbf{x}$ 到一个聚类的距离下界
$k$	最近邻搜索要返回的近邻数量
$\ \mathbf{x}_1, \mathbf{x}_2\ $	数据点 $\mathbf{x}_1, \mathbf{x}_2$ 的欧氏距离

### 3.2.1 基于分隔超平面的距离下界

HB方法是一种基于聚类的方法。首先使用KMeans技术为数据集生成 $K$ 个聚类，然后基于聚类间的超平面边界构造查询点到某个聚类的距离下界。在 $d$ 维空间，一个超平面可以用如下方式表示：

$$H(\mathbf{n}, b) = \{\mathbf{x}^T \mathbf{n} + b = 0 \mid \mathbf{x} \in \mathcal{R}^d\} \quad (3-1)$$

其中， $\mathbf{n}$ 是超平面的法向量， $b$ 是一个实数。为简便起见，本章剩余部分使用 $H$ 表示一个超平面。数据点 $\mathbf{x}$ 和超平面 $H$ 之间的距离可以通过如下方式计算：

$$d(\mathbf{x}, H) = \left| \frac{\mathbf{x}^T \mathbf{n} + b}{\|\mathbf{n}\|} \right| \quad (3-2)$$

图3.1展示了HB构造的一个距离下界。对于查询点 $\mathbf{q}$ 和一个聚类 $C_i$ ， $\mathbf{q}$ 到聚类 $C_i$ 的距离下界 $LB(\mathbf{q}, C_i) = d(\mathbf{q}, H) + d(\mathbf{x}, H)$ 。其中 $\mathbf{x} \in C_i$ 是聚类 $C_i$ 中距离边界 $H$ 最近的点。需要说明的是，只有分隔超平面（separating hyperplane）才能产生距离下界，即查询点和一个聚类分隔在其两侧的超平面，如图3.1所示。

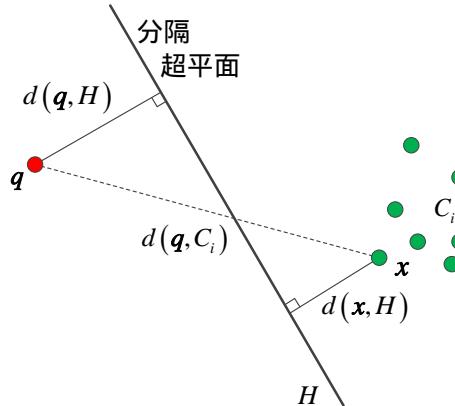


图 3.1 基于分隔超平面构建的距离下界

通常一个聚类和查询点间会有多个分隔超平面，因而可以构造多个距离下界。为提升紧致性，HB取所有下界<sup>1</sup>中最大的那个，可以用如下方式表示：

$$LB(\mathbf{q}, C) = \max_{H \in H_{sep}} \{d(\mathbf{q}, H) + \min_{\mathbf{x} \in C} d(\mathbf{x}, H)\} \quad (3-3)$$

HB进一步对式3-3进行了放缩，最终计算下界的方式如下：

$$LB(\mathbf{q}, C) = \max_{H \in H_{sep}} d(\mathbf{q}, H) + \min_{\mathbf{x} \in C} \min_{H \in H_c} d(\mathbf{x}, H) \quad (3-4)$$

<sup>1</sup>为了表述方便，如无特殊说明，下文中所有“下界”都指查询点和对应聚类之间的距离下界。

$H_{sep}$ 代表 $\mathbf{q}$ 和 $C$ 之间的所有分隔超平面,  $H_c$ 表示 $C$ 的所有超平面边界。等号右侧的第一部分表示 $\mathbf{q}$ 与 $C$ 的所有分隔超平面中的最大距离, 只能在线计算。等号右边的第二部分表示聚类 $C$ 中所有数据点到其所有超平面边界的最小距离, 可以看成是 $C$ 内部所有数据点与其所有边界之间的一个“内部间隔”(inner gap), 本章用一个新的符号表示:

$$IG_c = \min_{x \in C} \min_{H \in H_c} d(x, H) \quad (3-5)$$

很明显, 每个聚类的内部间隔都可以在离线阶段计算并预先存储, 而且只占用很小的空间(即 $O(Kd)$ )。因此, 下界计算的主要成本在计算查询点与所有分隔超平面之间距离的部分。

### 3.2.2 更好地定位分隔超平面

假设数据集被划分为 $K$ 个聚类 $\{C_i\}_{i=1}^K$ , 聚类中心分别为 $\{\mathbf{c}_i\}_{i=1}^K$ 。根据<sup>[32]</sup>中的定义, 两个聚类 $C_i, C_j$  ( $1 \leq i, j \leq K$  且  $i \neq j$ ) 之间的超平面 $H_{ij}$ 可以用如下方式计算:

$$H_{ij} = H(-2(\mathbf{c}_i - \mathbf{c}_j), \|\mathbf{c}_i\|^2 - \|\mathbf{c}_j\|^2) \quad (3-6)$$

虽然每个聚类周围有 $K - 1$ 个超平面边界, 但是, 并非所有边界都是分隔超平面。**HB**方法指出, 对于聚类 $C_i$ , 只有满足如下条件的边界才能将查询点 $\mathbf{q}$ 和其自身分隔在两边:

$$H_{sep}(\mathbf{q}, C_i) = \{H_{ij} \mid \|\mathbf{q}, \mathbf{c}_i\| > \|\mathbf{q}, \mathbf{c}_j\|\} \quad (3-7)$$

这个式子告诉我们,  $C_i$ 的一个边界 $H_{ij}$ 是否成为分隔超平面和 $\mathbf{q}$ 到聚类 $C_i, C_j$ 中心的距离有关, 只有当 $\mathbf{q}$ 和另一个聚类的中心 $\mathbf{c}_j$ 的距离小于 $\mathbf{q}$ 和自己中心的距离时, 才能形成一个分隔超平面。

根据式3-7, **HB**给出了一个快速确定每个聚类分隔超平面的方法, 具体如下。给定查询点 $\mathbf{q}$ , 首先计算 $\mathbf{q}$ 到所有 $K$ 个距离之间的距离, 存储在 $d_{qc}[\cdot]$ 中。将距离升序排序, 得到一个聚类的序列 $o_c[\cdot]$ 的一个序列。假设某聚类 $C$ 在 $o_c[\cdot]$ 中的序号是 $r$  ( $1 \leq r \leq K$ ), 根据式3-7, 只有序号小于 $r$ 的聚类才能产生 $C$ 的分隔超平面。

由此可以估计**HB**方法中每一个聚类分隔超平面的平均数量。按照 $o_c[\cdot]$ 中的顺序, 每个聚类对应的分隔超平面的数目依次为 $0, 1, 2, \dots, K - 1$ 。其中, 距离 $\mathbf{q}$ 最远的聚类有 $K - 1$ 个分隔超平面, 距离 $\mathbf{q}$ 最近的聚类(即 $\mathbf{q}$ 落入的聚类)没有分隔超平面。分隔超平面的总数是 $K(K - 1)/2$ , 平均一个聚类有 $(K - 1)/2$ 个分隔超平面。也就是说, 要确定 $\mathbf{q}$ 到某个聚类的距离下界, 平均需要计算约 $0.5K$ 个 $\mathbf{q}$ 到分隔超平面的距离。

### 3.2.3 HB方法的局限

HB方法进行 $k$ -最近邻（简称 $k$ -NN）搜索的算法过程大致如下。给定查询点 $\mathbf{q}$ ，首先计算 $\mathbf{q}$ 与所有聚类的距离下界。然后按照下界升序的顺序依次访问每个聚类。每个聚类执行如下判断决定是否被加载：如果当前 $k$ -最近邻半径（即 $\mathbf{q}$ 到当前找到的第 $k$ 最近邻的距离）大于等于该聚类的距离下界，则加载该聚类的所有成员点进行精炼；否则，意味着 $k$ -NN被找到，搜索终止。

HB的搜索算法主要有两个局限。第一，需要计算非常多的查询点到超平面的距离（即式3-2）。在搜索过程中，HB方法需要先确定所有聚类的访问顺序，为此需要计算查询点到所有聚类的距离下界。距离下界的计算主体是查询点到分隔超平面的距离。根据第3.2.2节的分析，总共有 $K(K - 1)/2$ 次计算，这是很大的计算量。而且复杂度很高，为 $O(K^2)$ ，会限制HB方法通过生成较多聚类获得更加紧致的距离下界。因为聚类越多，每个聚类的规模相对越小，构造的距离下界更接近查询点到聚类内部成员点的原始距离。

第二，HB无法支持聚类内部无关点的高效剪枝。在确定一个聚类需要被访问之后，HB会将该聚类的所有成员加载到内存中，通过计算昂贵的原始距离寻找更好的 $k$ -最近邻。然而，聚类内部一般有很多成员点（尤其在聚类个数被限制导致聚类粒度较大的情况下），而其中只有少部分能够更新最近邻列表，大部分都是无关点，这种方式会产生很多不必要的I/O开销和计算开销。

## 3.3 基于聚类技术的下界优化方法HB+

本节提出一种新的方法HB+，针对性地克服HB方法的两个局限，进一步提升最近邻搜索性能。

### 3.3.1 加速计算距离下界

由前文可知，距离下界只和分隔超平面有关，本节从两个方面入手加速距离下界的计算。第一，结合随机投影技术快速估计查询点到分隔超平面边界的距离；第二，提出一种部分选择策略，快速确定距离查询点最远的分隔超平面。接下来详细介绍这两个思路。

对于第一点，结合式3-2、式3-6和式3-7，可以发现查询点到一个超平面边界的距离和两种其他距离有关：

$$d(\mathbf{q}, H_{ij}) = \frac{\|\mathbf{q}, \mathbf{c}_i\|^2 - \|\mathbf{q}, \mathbf{c}_j\|^2}{2\|\mathbf{c}_i, \mathbf{c}_j\|} \quad (3-8)$$

一种是查询点到所有聚类中心的距离，一共有 $K$ 个，需要在线计算。另一种是每一对聚类中心之间的距离，共有 $K(K - 1)/2$ 个，数量较多。为节省存储空间，本章

和HB的做法一样，选择在线计算这部分距离。为了加速计算，本章参考机器学习领域的做法，引入随机投影技术（Random Projection）<sup>[82-84]</sup>构建一种聚类中心距离快速估计方法。

首先将 $K$ 个聚类中心表示为一个 $K \times d$ 的矩阵 $A$ ，并根据如下概率分布生成一个 $d \times m$ 大小的随机矩阵 $R$ ：

$$r_{ij} = \begin{cases} \sqrt{3} & \text{with prob. } 1/6 \\ 0 & \text{with prob. } 2/3 \\ -\sqrt{3} & \text{with prob. } 1/6 \end{cases} \quad (3-9)$$

矩阵 $R$ 的每个元素 $r_{ij}$ 独立地按照式3-9随机生成<sup>[82]</sup>。然后，按照如下方式对 $A$ 降维：

$$B = \frac{1}{\sqrt{m}} AR \quad (3-10)$$

降维之后，原先 $d$ 维空间中的聚类中心会被转化为 $m$ 维空间中的点。

式3-9和式3-10是Achlioptas构造的一种随机投影方法<sup>[82]</sup>，其中式3-9的具体含义为，以1/6的概率生成 $\sqrt{3}$ ，1/6的概率生成 $-\sqrt{3}$ ，2/3的概率生成0。这是一个期望为0，方差为1的概率分布。Achlioptas证明按照这一方法对数据进行降维能够满足Johnson - Lindenstrauss定理<sup>[85]</sup>，使得在降维之后的聚类中心上计算的距离能够很好地保持原始聚类中心间的相似性。因此，本章使用降维之后计算的聚类中心距离代替式3-8中原始聚类中心间的距离，以提升距离下界的计算速度。

需要说明的是，在降维过程中， $m$ 是一个很重要的参数，同时影响着下界的紧致性和计算的效率。 $m$ 越大，降维后保留的信息越多，降维后计算的距离更接近原始距离，但也会消耗更多计算开销。在实际实验中， $m$ 通常取很小的值（比数据集维度 $d$ 小很多）就能取得很好的效果，本章将在3.4.2中详细分析 $m$ 的设置。

对于第二点，由式3-4可知，查询点到一个聚类的距离下界实际上只由一个分隔超平面决定，即距离查询点最远的那个分隔超平面。若能为每个聚类快速确定距离查询点最远的分隔超平面，则可以加速距离下界的计算。实际上，根据第一点设计的方法能够快速估计查询点到分隔超平面的距离，从而找到拥有最大距离的分隔超平面。但是由于存在误差，本章设置了一个两步走的策略：先根据第一点的方法选出多个拥有最大估计距离的分隔超平面，然后计算原始距离选出真正最远的分隔超平面，从而整体上降低距离下界的计算量。

具体地，本节设置参数 $\alpha$ 代表每个聚类可以选择的分隔超平面比例（ $0 < \alpha \leq 1$ ）， $\alpha K$ 即是每个聚类能够保留的分隔超平面个数。实验表明，随机投影技术的相似性保持能力非常好，以至于 $\alpha$ 设置很小的值<sup>2</sup>也能准确地找出真正最远的分隔超平面。

将以上两个思路结合设计了新的下界计算算法，见算法1。首先计算查询点 $q$ 与

<sup>2</sup>远小于聚类的平均分隔超平面比例0.5，见第3.2.2节的分析。

---

**Algorithm 1** LowerBound

---

```

Require:  $\{c_i\}_{i=1}^K, q, \alpha, K, IG_c[\cdot]$ 
Ensure:  $LB[\cdot]$ 

1:  $d_{qc}[\cdot] \leftarrow \text{dist}(q, \{c_i\}_{i=1}^K);$ 
2:  $\{d_{qc}^{sort}[\cdot], o[\cdot]\} \leftarrow \text{sort}(d_{qc}[\cdot], \text{'ascend'});$ 
3:  $T \leftarrow \alpha \cdot K;$ 
4:  $LB[o[1]] \leftarrow 0;$ 
5: for  $i \leftarrow 2 : K$  do
6:   // 确定分隔超平面
7:    $H_{sep}[\cdot] \leftarrow \phi$ 
8:    $i_c \leftarrow o[i];$ 
9:   for  $j \leftarrow 1 : i - 1$  do
10:    add  $H_{i_c, o[j]}$  to  $H_{sep}[\cdot];$ 
11:   end for
12:   // 根据估计距离选择最多  $T$  个分隔超平面
13:   if  $i - 1 > T$  then
14:      $d_{qH}[\cdot] \leftarrow \text{estimate\_dist}(q, H_{sep}[\cdot]);$ 
15:      $\{d_{qH}^{sort}[\cdot], o_H[\cdot]\} \leftarrow \text{sort}(d_{qH}[\cdot], \text{'descend'});$ 
16:      $H_{sep}[\cdot] \leftarrow H_{sep}[o_H[1 : T]];$ 
17:   end if
18:   // 基于原始距离计算距离下界
19:    $LB[i_c] \leftarrow \max_{H \in H_{sep}[\cdot]} d(q, H) + IG_c[i_c];$ 
20: end for
21: return  $LB[\cdot];$ 

```

---

所有聚类中心之间的距离，并按升序排序，得到所有聚类的有序列表 $o[\cdot]$ （第1-2行）。在此基础上，结合式3-7可以快速确定与每个聚类形成分隔超平面的聚类（第9-11行）。其中， $q$ 和其最近的聚类（即 $o[1]$ ）之间不会有分隔超平面，算法将其下界设置为0（第4行）。随后，算法利用上述快速估计方法为每个聚类最多选择 $T = \alpha K$ 个分隔超平面（第13-17行）。通过计算查询点到这些分隔超平面的原始距离找出最终的距离下界。如第3.2.1节所述，每个聚类的内部间隔都是离线阶段预先计算并存储的（第19行）。

### 3.3.2 聚类内部高效剪枝

关于HB的第二个局限，可以结合图3.2更直观地进行说明。图3.2是一个搜索最近邻的例子（即 $k = 1$ ）， $r$ 是当前最近邻半径。当前聚类由于距离下界小于 $r$ 需要被加载。可以看到，理想情况下，只有点 $x_1$ 会更新最近邻，但HB会加载并验证该聚类中的所有点。这一问题的原因是HB的距离下界不够精细。根据式3-4可知，HB的距离下界严格小于等于查询点到一个聚类内部任何点的距离。这意味着，HB距离下界的判断粒度只到聚类这一层，无法用作聚类内部点的距离下界，因此无法用于过滤成员点。

本节对HB距离下界的定义进行了扩展，使得能够在聚类内部继续过滤无关数

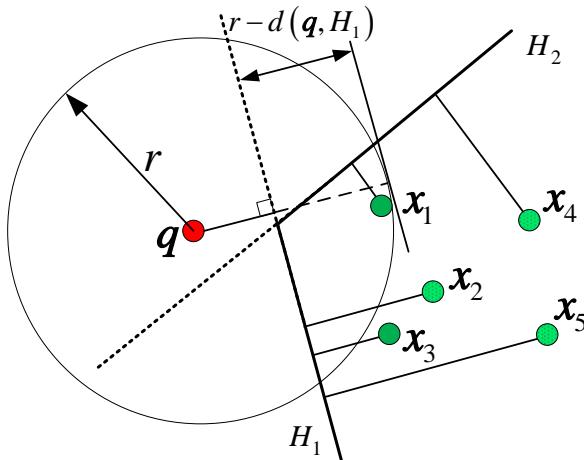


图 3.2 基于距离下界过滤无关数据

据。具体地，首先为聚类中的每个成员点定义了一种“内部间隔”：

$$IG_p(\mathbf{x}) = \min_{H \in H_c} d(\mathbf{x}, H) \quad (3-11)$$

这里， $\mathbf{x}$ 表示聚类 $C$ 中的一个成员点， $H_c$ 表示 $C$ 的所有超平面边界。因此，成员点的内部间隔实际上就是 $\mathbf{x}$ 到 $C$ 所有边界距离的最小值。在图3.2中就是连接每个数据点及其最近超平面边界的实线。基于新的内部间隔，可以获得查询点 $q$ 到一个成员点 $\mathbf{x}$ 的距离下界：

$$LB(\mathbf{q}, \mathbf{x}) = \max_{H \in H_{sep}} d(\mathbf{q}, H) + IG_p(\mathbf{x}) \leq \|\mathbf{q}, \mathbf{x}\| \quad (3-12)$$

### 3.3.3 新的最近邻搜索算法

本节将第3.3.1节和第3.3.2的两个方法结合，提出一种新的高维精确最近邻搜索算法HB+。详细的 $k$ -最近邻搜索流程见算法2。整体上，HB+的算法流程和HB一样。先计算距离下界，然后根据距离下界访问聚类，区别主要有两点。第一，使用了本章新设计的算法1加速聚类距离下界的计算（见第2行）；第二，应用新定义的数据点的距离下界细化了聚类内部的无关数据过滤（第8-14行）。

需要说明的是，按照新的距离下界定义，每个成员点都会获得一个内部间隔。离线阶段，每个聚类内部的成员点会按照内部间隔递增的顺序存储。在线搜索时，如果算法决定访问某一聚类，也会按照相应的顺序逐个加载和验证成员点。如果某个成员点的距离下界大于当前搜索半径，则该成员点及其之后的所有成员点都可以被过滤，从而实现对聚类内部无关点更加充分的过滤（第9-13行）。还以图3.2为例，按照内部间隔，该聚类成员点的存储及访问顺序是 $\mathbf{x}_1, \mathbf{x}_3, \mathbf{x}_2, \mathbf{x}_4, \mathbf{x}_5$ 。由于数据点 $\mathbf{x}_2$ 的距离下界 $d(\mathbf{q}, H_1) + IG_p(\mathbf{x}_2) > r$ ，因此 $\mathbf{x}_2$ 及其之后的 $\mathbf{x}_4, \mathbf{x}_5$ 都会被过滤。

图3.3展示了增加聚类内部剪枝机制后带来的效果。在Corel, Aerial和WT三个高维数据集上进行10-NN搜索（数据集的详细信息见第3.4.1节）比较HB和HB+算法的

**Algorithm 2**  $k$ -NNSearch

---

**Require:**  $\{\mathbf{c}_i\}_{i=1}^K, IG_c[\cdot], IG_p[\cdot], \alpha, k, \mathbf{q}$

**Ensure:**  $kNNs[\cdot] // kNNs$  是一个容量为  $k$  的最大堆，其中的每个元素是一个(点, 距离)对，元素按照距离值的大小被维持

- 1: 将  $kNNs[\cdot]$  中的每个元素初始化成  $\langle NULL, MAXREAL \rangle$ ,  $kNNs.count \leftarrow 0$ ;
- 2:  $LB[\cdot] \leftarrow \text{lowerbound}(\{\mathbf{c}_i\}_{i=1}^K, \mathbf{q}, \alpha, K, IG_c[\cdot])$ ;
- 3:  $\{LB^{sort}[\cdot], o[\cdot]\} \leftarrow \text{sort}(LB, \text{'ascend'})$ ;
- 4:  $i \leftarrow 1$ ;
- 5: **while**  $i \leq K$  and  $LB^{sort}[i] < kNNs[1].distance$  **do**
- 6:    $i_c \leftarrow o[i]$ ;
- 7:    $Candidate_p[\cdot] \leftarrow \phi$ ;
- 8:   **for** point  $x$  in the  $i_c$ -th cluster **do**
- 9:     **if**  $kNNs[1].distance \geq LB[i_c] - IG_c[i_c] + IG_p(x)$  **then**
- 10:       从外存中加载数据点  $x$  并将其加入候选集  $Candidate_p[\cdot]$ ;
- 11:     **else**
- 12:       break;
- 13:     **end if**
- 14:   **end for**
- 15:   **for** point  $x$  in  $Candidate_p[\cdot]$  **do**
- 16:     **if**  $kNNs.count < k$  or  $\|\mathbf{q}, x\| < kNNs[1].distance$  **then**
- 17:       将  $\langle x, \|\mathbf{q}, x\| \rangle$  插入最大堆  $kNNs[\cdot]$ ;
- 18:     **end if**
- 19:   **end for**
- 20:    $i \leftarrow i + 1$ ;
- 21: **end while**
- 22: **return**  $kNNs[\cdot]$ ;

---

选择能力。所谓的选择能力，指算法在最近邻搜索过程中访问的原始数据占整个数据集的比例。实验调整分隔超平面比例系数  $\alpha$  和聚类个数  $K$  以查看性能变化趋势。

根据图3.3可以看到，与HB相比，HB+在搜索过程中需要访问的点数明显减少，意味着本节设计的聚类内部剪枝机制发挥了作用。此外，当  $\alpha$  大于 0.06 时，增加  $\alpha$  将不会明显地提升选择比例，说明此时已经能够稳定地找到最好的分隔超平面，不用再增加更多分隔超平面。这个比例远小于HB平均需要计算的分隔超平面比例0.5（见第3.2.2节），意味着HB+节省了很多下界计算量。

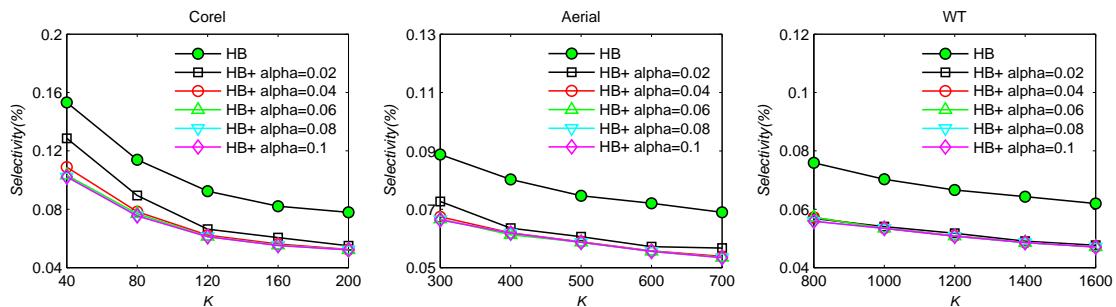


图 3.3 比较HB和HB+方法的选择能力

### 3.3.4 复杂度分析

**时间复杂度.** HB+进行最近邻搜索可以分为两个阶段：1) 计算距离下界; 2) 按照距离下界递增的顺序依次加载聚类成员点精炼最近邻。

在距离下界计算阶段，HB+与HB方法的过程都分为两部分。第一部分两个方法的计算任务一样，都需要计算查询点与所有聚类中心的距离并排序，计算复杂度是 $O(K \log K)$ 。第二部分两个方法存在不同。HB方法需要计算查询点到每一个聚类的所有分隔超平面的原始距离（即式3-8所描述的距离）。而HB+方法则不同，对于分隔超平面个数较少（不超过 $\alpha K$ ）的聚类，HB+与HB方法一样，直接计算查询点与该聚类所有分隔超平面的原始距离；对于分隔超平面个数较多（超过 $\alpha K$ ）的聚类，HB+方法会先计算查询点与该聚类所有分隔超平面的估计距离，快速定位距离查询点最远的 $\alpha K$ 个边界，再计算这些边界到查询点的原始距离。因此，HB+方法距离下界的计算开销主要由 $\alpha$ 和 $m$ 两个参数控制。 $m$ 越小，估计一次查询点到一个超平面边界距离的速度越快； $\alpha$ 越小，使用快速估计的方式计算到查询点距离的超平面边界的数量越多。

对于第二阶段，假设HB+与HB两个方法生成的聚类结果相同，如果HB+方法能够成功为每个聚类保留最远分隔边界<sup>3</sup>，则对于每一个数据点，HB+方法构造的距离下界一定大于等于HB方法提供的距离下界，从而有更大的可能过滤更多无关数据，减少计算量。

**空间复杂度.** HB+方法的空间开销主要来自五部分数据的存储：1)  $K$ 个聚类中心，每一个聚类中心是一个 $d$ 维向量；2)  $K$ 个聚类内部间隔；3) 所有数据点的内部间隔值；4) 所有数据点所归属的聚类编号列表；5) 同一个聚类的所有数据点按照内部间隔值递增的顺序依次存放。因此HB+的索引结构共占据 $K(d + 1) + n(d + 2)$ 个机器字空间。

**I/O开销.** 由于难以预知HB+方法的近邻搜索过程会在何时终止，因此难以给出具体的I/O开销量。不过，可以说明的是，HB+方法的I/O开销主要来自聚类成员点的加载环节。其中，每加载一个新的聚类消耗一次随机I/O。同一个聚类内部成员点的加载通过顺序I/O即可完成，因为成员点存放的顺序即是搜索时依次验证的顺序。

## 3.4 实验结果与分析

本节设计实验全面评估HB+方法的性能。首先介绍性能评估的指标和数据集的详细信息。然后，在第3.4.2节观察 $m$ 对HB+下界紧致性的影响，为在不同数据集上设置最优 $m$ 值提供参考。第3.4.3节对比HB与HB+方法。最后，在第3.4.4节，将HB+方

<sup>3</sup>即一个聚类所有分隔边界中距离查询点最远的分隔边界。

法和现有最先进的精确最近邻搜索算法进行性能比较（包括VA-file, iDistance和FNN方法）。所有算法使用C语言实现，在一台Linux主机上运行（处理器：Intel®Core™ 2 Quad CPU 2.83GHz，内存：4GB，操作系统：Ubuntu 12.04 LTS）。

### 3.4.1 实验设置

本章主要关注外存环境中的精确 $k$ -NN搜索。受内外存通信巨大速度差异的影响，I/O开销是影响外存最近邻搜索速度的重要因素<sup>[61]</sup>。为详细评估算法性能，本节将最近邻搜索性能分为两个方面进行考察：I/O开销（I/O cost）和CPU响应时间（CPU response time）。对于I/O开销，参考文献<sup>[51]</sup>将随机I/O开销（ $IO_r$ ）与顺序I/O开销（ $IO_s$ ）按照如下方式合并： $IO_r + IO_s/10$ 。CPU响应时间是指从算法线上进行最近邻搜索的全部时间减去I/O操作消耗的时间。

除非另有说明，本节实验考察的都是搜索前10最近邻的性能，即 $k=10$ 。实验在三个代表性的高维视觉特征数据集上进行：Corel<sup>4</sup>，Aerial<sup>5</sup>和WT<sup>6</sup>。其中，Corel共包含68,040个32维颜色直方图特征向量；Aerial由275,465个60维纹理特征向量组成；WT由269,648个128维小波纹理组成。对于每个数据集，随机选择200个数据点构成查询点集，每性能取所有查询点的平均值作为最终结果。

### 3.4.2 降维数 $m$ 的影响

本节观察 $m$ 的变化对HB+性能的影响， $m$ 是HB+快速估计距离下界时需要降低到的维数。实验分别为 $m$ 和 $K$ 设置了一组值观察I/O开销的变化。根据上一节的分析， $\alpha$ 设置为0.06。图3.4描述了在三个数据集上的实验结果。

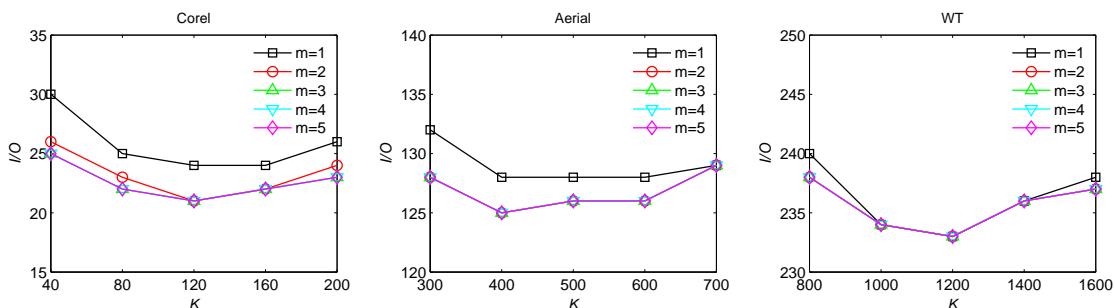


图 3.4 变化 $m$ 对HB+性能的影响

如图3.4所示，当 $m$ 很小（ $m=1$ ）时，HB+普遍会消耗较多I/O。略微增大 $m$ 之后，HB+的I/O开销会下降，在 $m=3$ 左右时达到最佳，此后继续增大 $m$ ，I/O开销保持不变。这说明 $m = 1$ 时降维之后计算的聚类中心距离相对于原始距离有较大失真，使得难以

<sup>4</sup><http://archive.ics.uci.edu/ml/datasets/Corel+Image+Features/>

<sup>5</sup><http://vision.ece.ucsb.edu/download.html>

<sup>6</sup><http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm>

成功找出和查询点拥有最大距离的边界，从而无法获得最紧致的距离下界，I/O开销增大。增大 $m$ 能够减小这种失真，从而更容易找出和查询点拥有最大距离的边界，提升下界的紧致性，I/O开销也被降低。整体上，每个数据集上最佳的 $m$ 值都很小，远小于数据维度 $d$ ，这将显著降低式3-8中分母部分的计算量。根据图3.4中的结果，在之后的实验中，Corel、Aerial和WT数据集上的 $m$ 分别固定为3、2和2。

### 3.4.3 HB与HB+的比较

本小节比较HB和HB+这两个基于聚类的方法的性能。对于HB，实验变化聚类个数 $K$ ；对于HB+，实验变化 $K$ 和 $\alpha$ 。在所有三个数据集上比较了I/O开销和CPU响应时间，结果见图3.5。

图3.5(a)比较了HB和HB+的I/O开销。可以看到，当 $\alpha \geq 0.06$ 时，HB+的I/O开销在所有数据集上都明显低于HB，说明本章设计的聚类内部剪枝机制发挥了很好的效果，可以明显提高I/O性能。此外，每个数据集上都出现了一个 $\alpha$ 阈值，超过此阈值HB+的I/O开销变化很小。这为设置适合的 $\alpha$ 参数值提供了很好的参考。在图3.5(b)中，实验将Corel、Aerial和WT上的 $\alpha$ 值分别设置为0.08、0.06、0.04，发现HB+分别比HB能够减少24.9%，18.6%和18.9%的I/O开销。

图3.5(c)比较了HB和HB+之间的CPU响应时间。可以看到， $\alpha$ 会显著影响CPU响应时间。 $\alpha$ 越小，计算速度越快。不过，由于本章将I/O开销作为性能的主要考量， $\alpha$ 的取值主要参考I/O性能。根据上一段中对 $\alpha$ 的设置，实验发现，在Corel、Aerial和WT上，HB+分别比HB能够降低17.6%、23.0%和56.7%的CPU响应时间。

综上，本章设计的距离下界加速方法和聚类内部剪枝机制均产生了可观的效果，使得HB+在I/O性能和CPU性能方面相对于HB均获得了提升。

### 3.4.4 同其他先进方法的比较

表 3.2 不同方法在各数据集上的参数设置

方法	数据集与参数设置		
	Corel	Aerial	WT
HB	$K = 120$	$K = 500$	$K = 1200$
HB+	$K = 120, m = 3, \alpha = 0.08$	$K = 500, m = 2, \alpha = 0.06$	$K = 1200, m = 2, \alpha = 0.04$
iDistance	$N_{ref} = 64$	$N_{ref} = 64$	$N_{ref} = 64$
VA-File	$N_{bit} = 8$	$N_{bit} = 8$	$N_{bit} = 8$
FNN	(none)	(none)	(none)

本节将基于聚类的方法（HB与HB+）与其他先进 $k$ -NN搜索方法（包括VA-file、iDistance和FNN）进行性能比较。不同方法具有不同的参数。其中，HB和HB+都需要设置聚类个数 $K$ ，实验让两个方法在各个数据集上取相同 $K$ 值便于比较。HB+的另外两个参数 $m$ 和 $\alpha$ 按照之前的实验观察设置。iDistance的参数是参考点数量 $N_{ref}$ 。

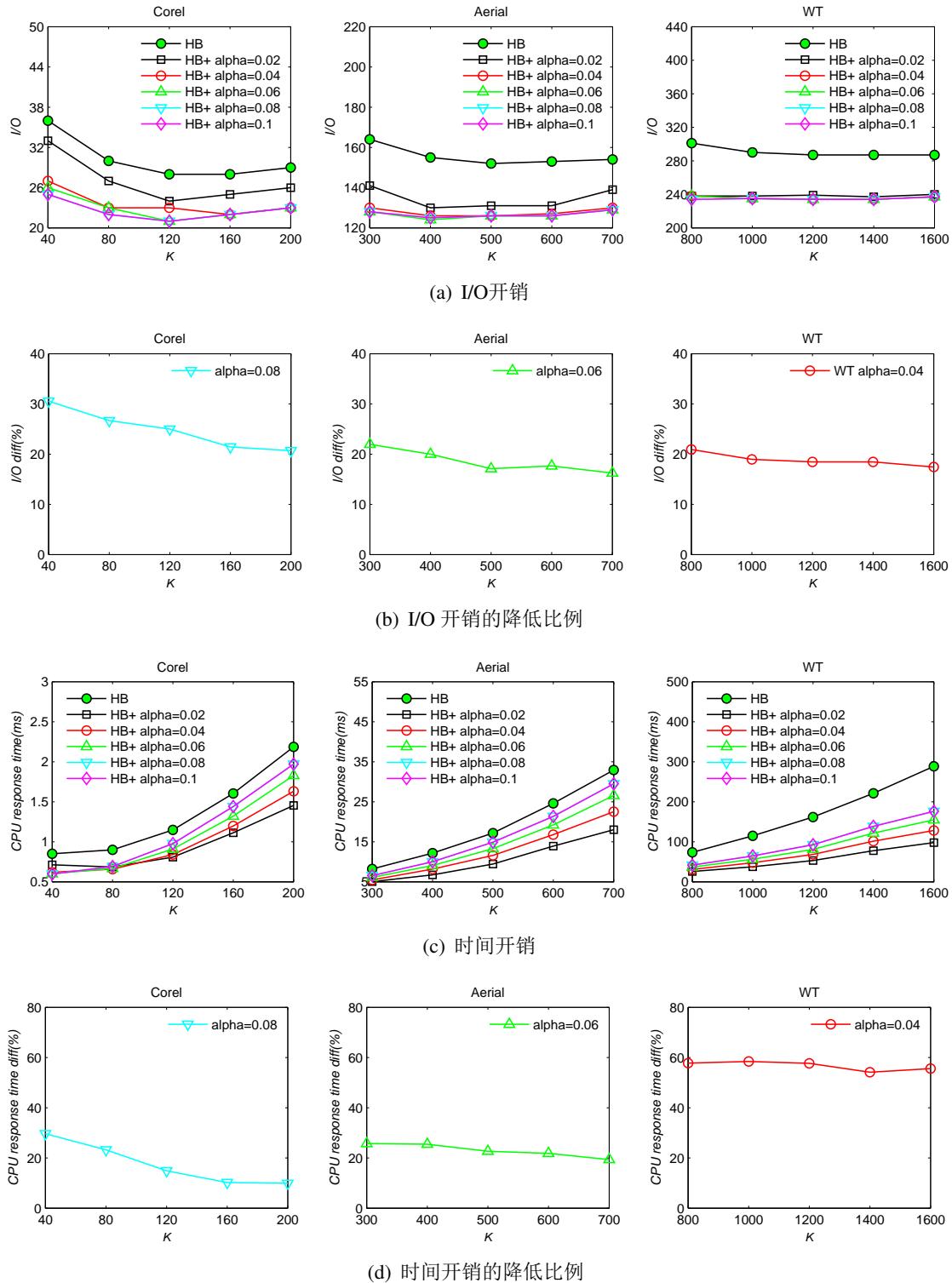


图 3.5 HB方法和HB+方法的性能比较

VA-file的参数是每个维度分配的比特数 $N_{bit}$ 。对于FNN，根据<sup>[59]</sup>中的算法2运行，无需进行参数设置。实验为每个方法设置合适的参数值让其取得最好性能，性能参考的主要标准是I/O开销，表3.2列出了各个方法的参数设置。参数确定之后，实验让所有方法在三个数据集上分别执行10-NN、40-NN、70-NN和100-NN搜索，比较这些方

法的I/O开销和CPU响应时间，结果见图3.6。

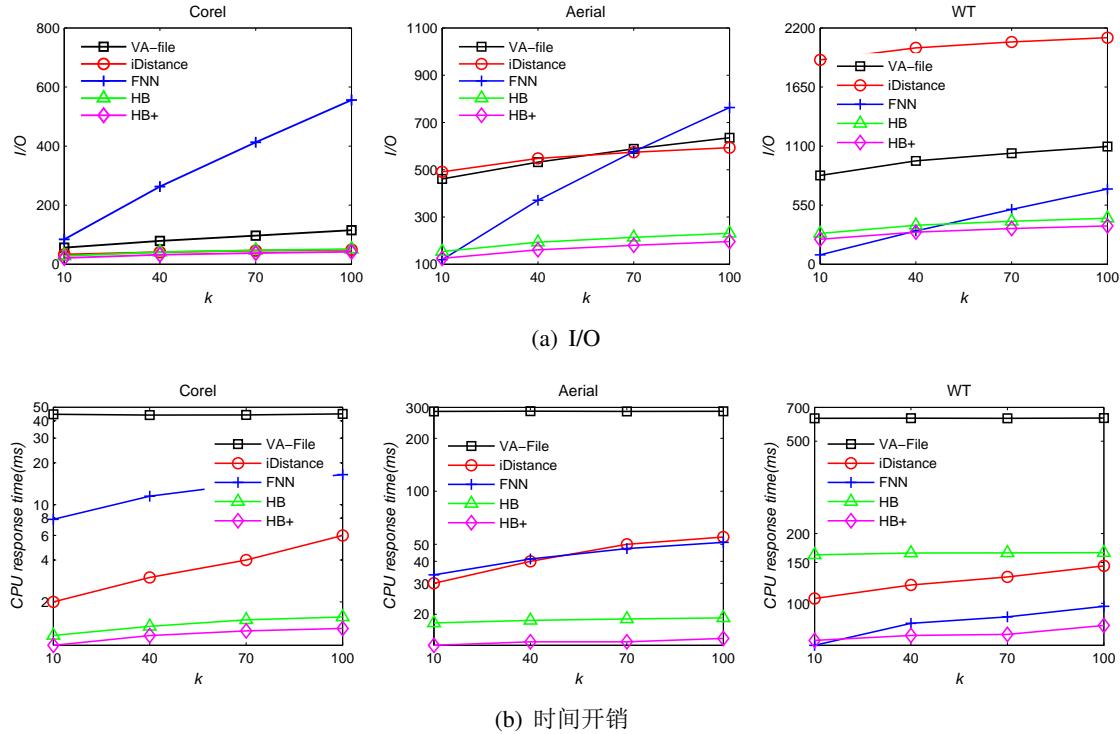


图 3.6 不同高维精确最近邻搜索方法的性能比较

图3.6显示，两个基于聚类的方法HB和HB+在三个数据集上拥有最少的I/O开销，且相比于其他方法有明显地降低。本章提出的HB+方法整体上在三个数据集中具有最少的I/O开销和CPU响应时间。这表明基于聚类构建的自适应距离下界具有出色的过滤能力，而本章提出的两个优化方法帮助HB+进一步提升了最近邻搜索的性能。FNN方法在k较小时也展现了很好的I/O性能和计算速度，但是它对k的稳定性不是很好。图3.6(a)显示，FNN的I/O开销会随k的增大快速增长。相比较而言，HB+方法则不太受k的影响，当k增大时，I/O开销和CPU响应时间都稳定地保持在很低的水平。

### 3.5 本章小结

本章提出了一种基于聚类的下界优化方法进一步提升高维精确最近邻搜索速度。HB方法基于聚类技术构造了现有最紧致的距离下界，但是在下界计算和聚类内部剪枝环节存在两方面局限，严重限制了最近邻搜索速度的提升。针对这些局限，本章针对性地设计了两个改进方法进行克服。一方面，基于随机投影技术和部分选择策略设计了一种距离下界加速计算方法；另一方面，为聚类成员点设计了一种距离下界，实现了聚类内部无关数据的过滤。实验表明，以上两个措施能进一步加速高维精确最近邻搜索，相对于现有先进方法在I/O性能和计算效率上均具有优越性。

## 第四章 针对近似最近邻搜索的基于最优排序的局部敏感哈希索引

本章针对外存环境中海量高维数据近似最近邻搜索面临的维度灾难和I/O性能瓶颈难题，研究一种基于最优排序的局部敏感哈希（Locality-Sensitive Hashing, LSH）索引方案，O2LSH(Optimal Order LSH)。通过引入空间填充曲线为复合哈希键值建立线序并排序，使近邻候选点更多地分布在相同或相邻磁盘页面，可以实现用少量顺序I/O加载到足够多的候选点。对多种常用空间曲线技术进行了量化分析，发现：1) 基本排序方案SK-LSH使用的row-wise曲线具有“维度优先遍历”的特性，容易对近似最近邻搜索造成多种局限；2) 另一类“邻域优先遍历”特性的曲线能够产生更好的候选点局部分布，且排序性能更加稳定。通过对比，选取了一种最优的“邻域优先遍历”曲线构造线序，能够最大程度地改善近邻候选点的局部分布，进一步提升磁盘访问效率和查询精度。在多个真实多媒体数据集上进行了对比实验，证实了O2LSH相对于先进LSH方案（包括C2LSH、SK-LSH、SRS以及QALSH）在查询精度和I/O效率上的优越性。特别地，O2LSH克服了基本排序方案SK-LSH对LSH关键参数的敏感性，算法实用性进一步提升。

### 4.1 概述

最近邻（Nearest Neighbor, NN）搜索广泛应用于文本信息检索、搜索引擎、基于图像内容的信息搜索、数据重复性检测等领域<sup>[7]</sup>。在这些领域，数据的海量、高维特性给快速、有效的最近邻搜索提出了巨大的挑战。一方面，高维空间中的维数灾难难以克服，越来越多的研究者开始关注近似最近邻（Approximate Nearest Neighbor, ANN）搜索算法，即在允许的范围内通过寻求近似解换取搜索速度的提升；另一方面，海量数据集一般体量巨大，难以放入内存，外存存储成为更加合理的选择。但是内外存间存在巨大的速度差距，使得二者之间的通信（即I/O）变得十分昂贵，如果通信次数过多，或者外存访问方式不合理，会使得这一环节成为整个近邻搜索最耗时的部分，称之为I/O性能瓶颈<sup>[61]</sup>。

现有外存ANN搜索方案主要包括局部敏感哈希（Locality Sensitive Hashing, LSH）<sup>[33-35]</sup>系列技术、乘积量化（Product Quantization, PQ）<sup>[62, 65]</sup>技术以及近邻图（Nearest Neighbor Graph, NNG）<sup>[67, 69, 86]</sup>技术等。其中，LSH由于具有出色的误差保证和较高的计算效率受到越来越多的关注，是目前解决ANN搜索使用最为广泛的技术。LSH是一种特殊的哈希函数，具有距离保持的特性，能够将原始空间中邻近的点以较大的

概率映射到相同的哈希桶（称二者发生碰撞）。在实际搜索时，通过构建复合LSH函数（compound LSH function）以及建立多索引哈希表能够提升相似点的碰撞概率和不相似点不碰撞的概率，从而减少近邻点的误报（False Positive，简称FP）和漏报（False Negative，简称FN）现象。目前先进的外存LSH方案主要有LSB-Forest<sup>[36]</sup>、C2LSH<sup>[37]</sup>、SRS<sup>[38]</sup>以及QALSH<sup>[39]</sup>等。其搜索算法一般分为两个阶段：1) 先在内存中通过快速过滤无关数据确立近邻候选点；2) 从外存读入候选点原始数据，计算真实距离进而精炼出最近邻结果。

现有外存LSH索引技术在解决ANN搜索问题时有着各自的优势，但存在一个共同问题：这些方案确立的候选点大都随机分布在磁盘上。而为了保证搜索精度，一般需要从外存中加载足够多候选点，这就导致这些方案的候选点加载耗费大量昂贵的随机I/O操作，成为整个近邻搜索的速度瓶颈。

为改善ANN搜索的I/O效率，有研究者提出一种基于排序的LSH索引方案SK-LSH<sup>[40]</sup>。核心思想是，利用row-wise空间填充曲线（简称空间曲线）为复合哈希键值构造一种空间线序（简称线序）并对其排序。原始数据按照新的顺序在磁盘上重新排列可以让更多近邻候选点存放在相同或者相邻磁盘页面上，使得近邻候选点加载更多地通过顺序I/O完成。而且，由于一个磁盘页面通常能容纳多个候选点，单次I/O因此可以加载到更多候选点，整体I/O次数也被降低。

然而，基本的排序方案没有深入考察row-wise曲线下的近邻候选点局部分布质量，在实际ANN搜索时存在三方面局限：1) row-wise线序在局部排列时易产生较多近邻点漏报，降低了搜索精度；2) 线序的局部排列效果受LSH关键参数（主要是哈希函数桶宽W）变化敏感，依赖较大的桶宽，但需要大量的调参才能确定最优参数值，影响算法实用性；3) 较大的桶宽会引入更多近邻干扰点，需要更多哈希函数才能充分过滤，增加了存储和计算开销。实际应用中存在多种空间曲线技术<sup>[87]</sup>，这些曲线具有不同的特性，可能对近邻候选点的局部分布产生不同影响。由于候选点的局部分布质量是决定排序方案近邻搜索性能的关键，因此，本章围绕不同空间曲线特性对近邻候选点局部分布的影响展开深入研究，期望进一步提升ANN搜索性能。主要贡献如下：

- 对四种典型空间曲线下的近邻分布质量开展量化分析和对比，发现基本方案中所使用的曲线具有一种“维度优先遍历”的特性，是造成其局限的主要原因；
- 通过对比，发现“邻域优先遍历”特性的曲线能产生更好的近邻分布且性能更加稳定。通过比较，选取一种排序效果最优的“邻域优先遍历”曲线设计了一种新的LSH索引方案（Optimal Order LSH，简称O2LSH），能够最大程度地改善近邻候选点的局部分布，进一步提升ANN搜索性能；

- 建立索引结构并设计相应的ANN搜索算法，在多个真实数据集上进行了对比实验，证实了O2LSH相对于其他先进LSH方案（包括C2LSH、SK-LSH、SRS和QALSH）在ANN搜索性能上的优越性。并且新方案克服了基本方案的几个局限，不再对哈希关键参数敏感，算法实用性得以提升。

本章接下来内容组织如下：第4.2节集中讨论I/O性能的改善方法，通过对不同曲线下的近邻分布质量展开研究，寻找最优线序，进一步改善I/O性能；第4.3节给出本章新方法O2LSH的索引构建和搜索算法；第4.4节进行实验对比和分析；第4.5节进行本章小结。

## 4.2 基于空间曲线进一步改善I/O性能

### 4.2.1 现有外存索引的I/O性能

现有外存LSH索引技术使用随机I/O加载近邻候选点的问题可以通过图4.1(a)进一步说明。从图中可以看到，这些方法所确立的近邻候选点一般随机分布在磁盘上，在确立候选点之后会直接将候选点从外存中读入以进行精炼。这就意味着每一个候选点的加载平均会消耗一次昂贵的随机I/O操作。由于在过滤环节无法精确地确立候选点，为了保证近邻搜索的精度，大多数情况下都需要加载足够多的候选点。最终造成这类方法在候选点加载环节十分耗时，容易形成速度瓶颈。此外，一个磁盘页面通常可以容纳多个原始数据，而一次随机I/O操作平均只能获得一个候选点，I/O利用率也比较低。

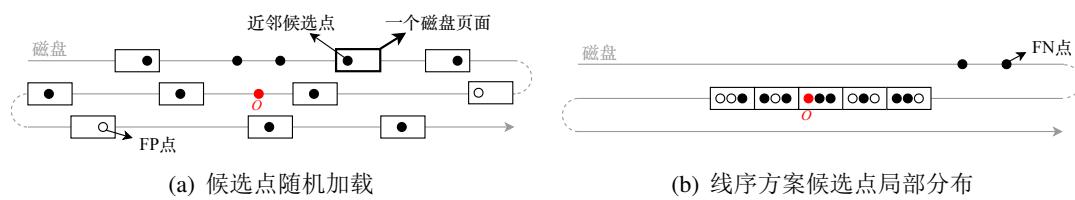


图 4.1 现有外存索引中候选点分布对比

由于I/O操作主要发生在候选点加载之时。相应地，改善I/O性能的途径主要有两种：1) 降低I/O次数，主要通过提升过滤能力以减少所要加载的候选点来实现；2) 优化I/O方式，减少昂贵的随机I/O访问，更多地使用顺序I/O，并且提升单次I/O操作的候选点加载量。对于第一种，Tang等人提出一种缓存候选点紧凑编码的技术<sup>[88]</sup>。在候选点加载前利用紧凑编码进行上下界判断，能够较早地将一些无关点排除。不过本质上其候选点仍旧是随机加载，存在加载效率低的问题。

Liu等人提出基于排序的LSH索引方案SK-LSH<sup>[40]</sup>，借助空间填充曲线技术实现候选点的局部分布可以同时实现以上两个目标。如图4.1(b)，数据按照空间线序重新

排列之后，邻近的点会更多地排列在连续的磁盘页面，这样就可以采用连续的I/O来读取。并且单次的磁盘访问平均能够加载到更多最近邻候选点，从而在保证搜索精度的前提下，降低磁盘访问次数。

### 4.2.2 空间曲线及其特性

空间填充曲线是一种比较成熟的空间降维技术，能够将多维空间映射到一维<sup>[87, 89]</sup>。其填充过程为：将多维空间划分成超立方体网格，按照某种规则不重复地遍历所有网格并递增地为所遍历网格分配一个独特的编号，编号递增的顺序就是曲线为多维空间规定的一维空间线序。空间曲线的一个特性是，曲线上编号邻近的网格在实际空间中往往也是邻近的，如果数据依照空间线序在磁盘上存放，就会实现让很多邻近点在磁盘局部聚集存放。

空间曲线的这种特性使得其很适合作为多维数据索引并应用于相似性数据搜索<sup>[36, 40, 87, 89–91]</sup>。其中，LSB-Forest是最早将空间曲线技术引入LSH索引的方法，因为复合LSH键值也是一种多维数据。不过该方法关注的重点是如何借助z-order空间曲线技术实现近邻搜索半径自适应扩充，并没有深入考虑空间曲线对I/O效率的改善，因此LSB-Forest在进行近邻搜索时仍旧需要耗费大量的随机I/O。

SK-LSH率先指出了空间曲线下近邻候选点的局部分布对改善I/O效率的意义，提出了第一个完整的基于排序的LSH索引框架，实现了I/O效率的提升。但是SK-LSH没有深入考虑曲线特性的不同对近邻点局部分布的影响，选择了一种较为简单的row-wise曲线构造空间线序，导致近邻搜索算法依旧存在多种局限（见本章第4.1节）。

本节进一步探究使用空间曲线技术改善LSH索引的性能。我们对多维索引和相似性搜索领域的相关工作进行了考察，汇总了几类最为常用的空间曲线技术，分别是row-wise曲线、z-order曲线、Gray曲线和Hilbert曲线，见图4.2。这些曲线在特性存在较大的差异，有可能对LSH的索引性能产生不同的影响，本节选取这几类曲线作为代表作进一步的研究。

直观地，能够看到row-wise曲线与其他曲线在填充特性上的不同。最明显的区别体现在曲线填充时优先级的分配上。row-wise曲线倾向于为不同维度设定不同的填充优先级，如图4.2(a)，X维度的填充优先级要大于Y维度。因此，曲线会率先填充高优先级的X维度，完成之后进入次高优先级的Y维度上移动一个网格，然后返回高优先级的X维度继续填充。假设二维空间中某一网格的坐标是 $(x, y)$ ，那么它在曲线上的编号就是 $x + y \cdot I$ ，其中 $I$ 代表Y维度上的网格总数量。不妨称这种填充特性为“维度优先遍历”(Dimension-First-Traverse，简称DFT)特性。

另外几类曲线则呈现出一种从邻域空间逐步向外扩充的填充趋势，即曲线总是率先填充一个小的邻域，完成后，再扩大到一个较大的邻域继续填充。在从局部逐

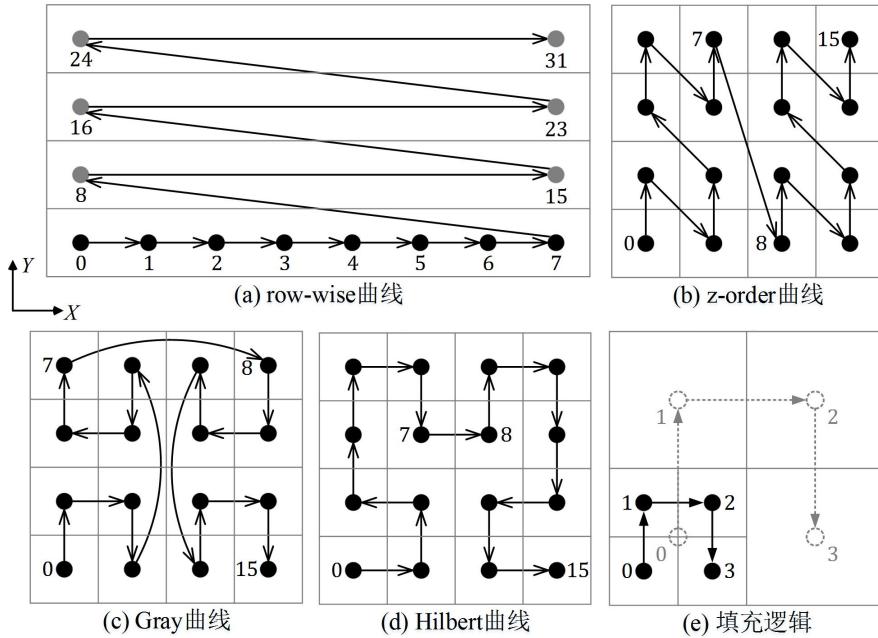


图 4.2 几种典型的空间填充曲线

渐向外扩充的过程中，曲线的填充呈现出明显的递归性。以图4.2(c)中的Gray曲线为例，图4.2(e)显示的是Gray曲线的两阶填充逻辑，可以看到，曲线在左下角 $2 \times 2$ 小网格里的填充逻辑和将整个 $4 \times 4$ 网格看作一个 $2 \times 2$ 大网格的填充逻辑是一致的。同样，z-order曲线和Hilbert曲线也呈现出类似的填充规律。不妨称这种填充特性为“邻域优先遍历”（Neighborhood-First-Traverse，简称NFT）特性。

### 4.2.3 最优空间线序

曲线填充特性的不同可能对近邻候选点的局部分布产生不同影响，为便于比较，不妨定义一种曲线上的近邻候选点局部分布质量评价标准：一条空间填充曲线，如果能在相同的局部范围内汇集更多近邻点，就称该曲线能够产生更好的近邻候选点局部分布（简称局部分布）。该评价标准有两个关键点，一个是“局部范围”，一个是“更多近邻点”。前者用来保证近邻候选点的加载采用顺序I/O来完成，后者用来反映候选点局部分布的质量。这两者必须同时满足才能称曲线产生了更好的局部分布。

直观上看，NFT曲线能够比DFT曲线产生更好的近邻分布。因为近邻点一般分布在查询点周围的邻域，NFT曲线优先遍历邻域，有可能更早地“捕获”近邻点；但是对于DFT曲线，这取决于近邻点的位置：若近邻点落在高优先级维度上，就会被很快捕获，否则则很难聚集在局部。而且后面这种情况在高维空间会更加频繁，因为大部分维度都是较低优先级的维度。

### 4.2.3.1 直观对比

为更清晰地说明这一点，首先选取row-wise曲线和Hilbert曲线分别代表DFT和NFT曲线在二维空间作一些简单对比。本节对比了两个示例，分别见如图4.3和图4.4。其中， $O$ 是查询点， $\{A, B, C, D\}$ 是其真实前4最近邻， $F$ 是一个距离 $O$ 比较远的点。灰色和黑色圆点是空间中部分网格中心，其中黑色圆点突出显示几个待观察点归属的网格。定义两点在曲线上的距离为二者沿着曲线相距的网格数。每个示例图中前两个子图是原始空间中的图示，第三个子图是将原始空间和映射到1维空间后的图示。

先看第一个示例。如图4.3(c)，可以直观地看到在这一示例中经过Hilbert曲线映射后的近邻点分布要比经过row-wise曲线映射后的更加集中。

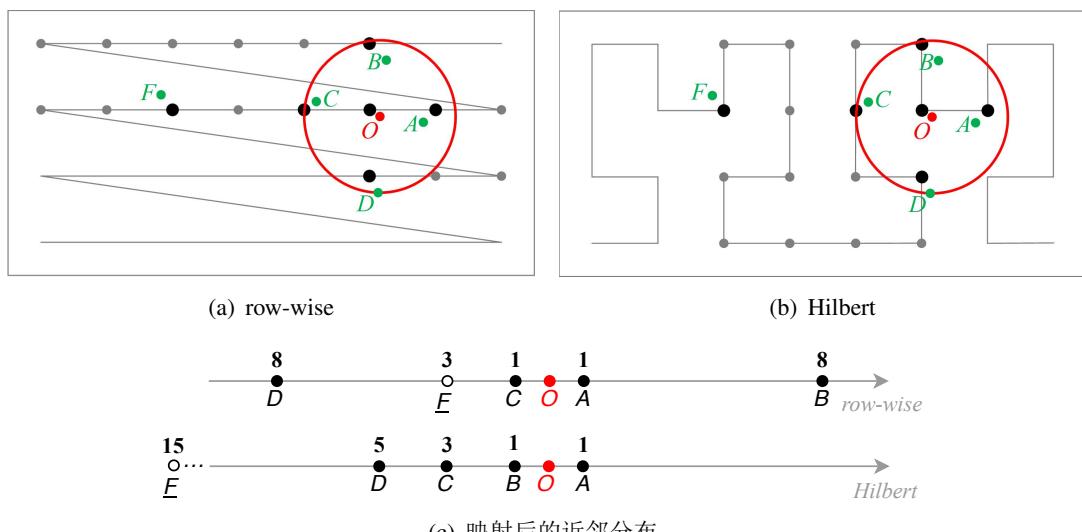


图 4.3 二维空间近邻分布对比（一）

表4.1列举了 $\{A, B, C, D, F\}$ 几个点在曲线上到 $O$ 的距离。根据平均距离和标准差可以看到，Hilbert曲线上四个近邻点的局部分布要优于row-wise曲线（平均距离更小，分布更加集中）。究其原因，在于row-wise曲线没有较早地捕获 $B, D$ 两点。正如之前所预料的，由于 $B, D$ 两点相对于 $O$ 落在了较低优先级的维度上，需要等待曲线完整地遍历一次高优先级的维度之后才会被捕获，因此二者在曲线上距离 $O$ 都比较远。对于Hilbert曲线，由于邻近的空间总是会被率先遍历，因此位于 $O$ 邻域的4个近邻都能很快地被捕获。

表 4.1 近邻分布统计

曲线	$O$ 与 $\{A, B, C, D, F\}$ 距离	近邻平均距离与标准差
row-wise	{1, 8, 1, 8, 3}	4.5, 3.0
Hilbert	{1, 1, 3, 5, 15}	<b>2.5, 1.4</b>

如果给近邻搜索限定一个加载范围 $R$ ，表示只将距离查询点 $R$ 个网格之内的点作为近邻候选点。那么，当 $R=3$ 时，row-wise曲线上的近邻召回率是50%，Hilbert曲

线上是75%； $R=5$ 时，row-wise曲线的召回率还是50%，Hilbert则能够返回全部近邻。直到 $R=8$ 时，row-wise曲线才能返回全部近邻，在此之前， $B, D$ 两点会一直被漏报。这种近邻易漏报的问题就是row-wise曲线在近邻搜索时的第一个局限，是由于曲线维度优先遍历的特性造成的。

此外，维度优先遍历的特性还会产生近邻误报问题。图4.3(a)中，原本距离 $O$ 比较远的 $F$ 点由于落在了高优先级的维度上，使得其在row-wise曲线上距离 $O$ 只差3个网格。这就意味着，只要 $R \geq 3$ ，row-wise会一直把 $F$ 点当作近邻候选点返回，成为近邻误报。而在图4.3(b)中，Hilbert曲线由于邻域优先遍历的特性，会先于 $F$ 之前捕获处在局部小邻域的4个近邻点，而后扩充到更大的邻域捕获 $F$ 点，从而可以更好地避免这种问题的发生。

不过，NFT曲线并非总是能够完整地召回同一邻域范围中的近邻。由于NFT曲线需要递归地对空间进行划分，并优先遍历某一个子空间，这就造成子空间之间存在遍历优先级的不同，如果某一查询点的邻域跨越了多个子空间。则其近邻点就有可能获得差距很大的线序。如图4.4(b)和图4.4(c)， $O$ 的近邻 $A$ 点由于与其他三个近邻点落入了不同子空间之内（由虚线分隔），其所获得的线序就与另外三个点产生了很大差距。从而无法再映射后和其他点聚集在局部，很容易成为FN点。而对于row-wise曲线，近邻点映射之后的分布几乎没有变化。如果取近邻加载范围 $R=3$ ，则row-wise依旧会产生较多的FN点（即 $B, D$ ），并引入FP点 $F$ ；而Hilbert曲线则会引入一个FN点（即 $A$ ），不过依旧能很好地规避FP点（即 $F$ ）。

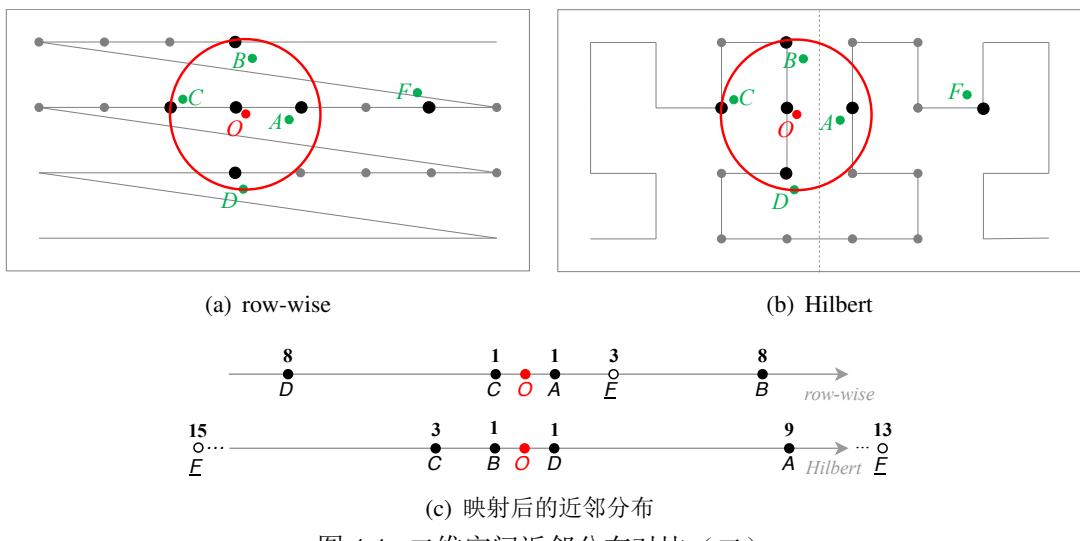


图 4.4 二维空间近邻分布对比（二）

因此，直观上看，两种曲线映射都存在一定的优势和劣势区间。DFT曲线能够较好地召回高优先级维度上的近邻点，但是也容易在高优先级维度上引入FP点，而处在较低优先级维度上的近邻则比较容易形成FN点；NFT曲线由于优先遍历邻域，

能够较好的避免非邻域中的点成为FP点的情况，且对于处在曲线遍历邻域中的近邻也能够较好的召回。但是对于跨越了多个曲线填充时划分的子空间的近邻邻域，则有可能形成FN点。

#### 4.2.3.2 量化分析

为了更加清楚地考察两类曲线在优势和劣势区间上的具体差距，特别是在多维空间（10到20维），本节作进一步的量化分析。由于目前难以在多维空间对曲线性能做出理论上的分析，本节选择在生成的一系列数据集上进行量化观察。

**数据集维度设定.** 需要说明的是，生成数据集的维度参照的是实际排序方案中复合哈希键值的维度而非原始数据的维度。因为基于排序的LSH方案并不直接在原始空间进行曲线映射，而是将原始数据经过复合LSH函数投影得到复合哈希值，在复合哈希值所在多维空间进行映射并排序。这么做，一是因为原始数据往往具有很高的维度，直接映射会有很大计算开销；二是因为一些研究表明大多数真实高维数据集的本征维度通常远远小于其原始维度<sup>[92]</sup>。凭借LSH函数出色的距离保持能力，复合哈希值能充分反映原始数据间的相似性，因此对复合哈希值排序可以近似等价于对原始数据排序。

**数据集生成.** 本节一共生成了两组观察数据集，分别是均匀分布数据和高斯分布数据。每一组数据集包含{2,10,20}三个维度的观察数据集。每个数据集含有200,000个基本数据点和200个查询点，所有维度的坐标范围限定在[0,1024]。

**性能考察指标.** 实验过程如下，先将每个数据集所在的空间划分为宽度为W的网格，然后选择一种空间曲线遍历所有网格进行填充。这样，每个数据点都会落入一个网格从而被映射到一个线序值，将数据点按照线序的顺序重新排列存放在磁盘。近邻搜索过程类似图4.3和图4.4中的过程，即给定候选点范围，计算近邻召回率。不过，由于高维空间变得稀疏，候选点范围R不再设定为曲线距离而是设置为数量。即给定查询点 $q$ ，从曲线上 $q$ 落在的线序值开始，左右各加载 $R$ 个点作为近邻候选点。最后从这 $2R$ 个候选点中找出 $k$ 个最近邻，计算其中真实 $k$ 最近邻的比例作为近邻召回率，以此衡量该曲线上的局部分布质量。实验中，取 $R=500$ ,  $k=10$ ，改变W以变化空间划分粒度，对比结果见图4.5。

通过观察，可以发现：1) row-wise曲线对W的变化十分敏感，而几类NFT曲线的性能对W的变化则比较稳定，且一直优于DFT曲线的性能；2) row-wise曲线比较“依赖”大的桶宽：在桶宽较小时，row-wise曲线的召回率同其他几类NFT曲线存在很大差距，只有在W足够大（W=512）时才会接近其他几类NFT曲线；3) 在2维空间中，当W较小时，几类曲线的表现正常（即符合前两点规律），但是当W增大到一定程度后，所有曲线的性能会急剧下降，如图4.5(a)中W=32，图4.5(d)中W=16。

关于前两点，之前提到，row-wise曲线容易在局部错失很多近邻点，形成近邻

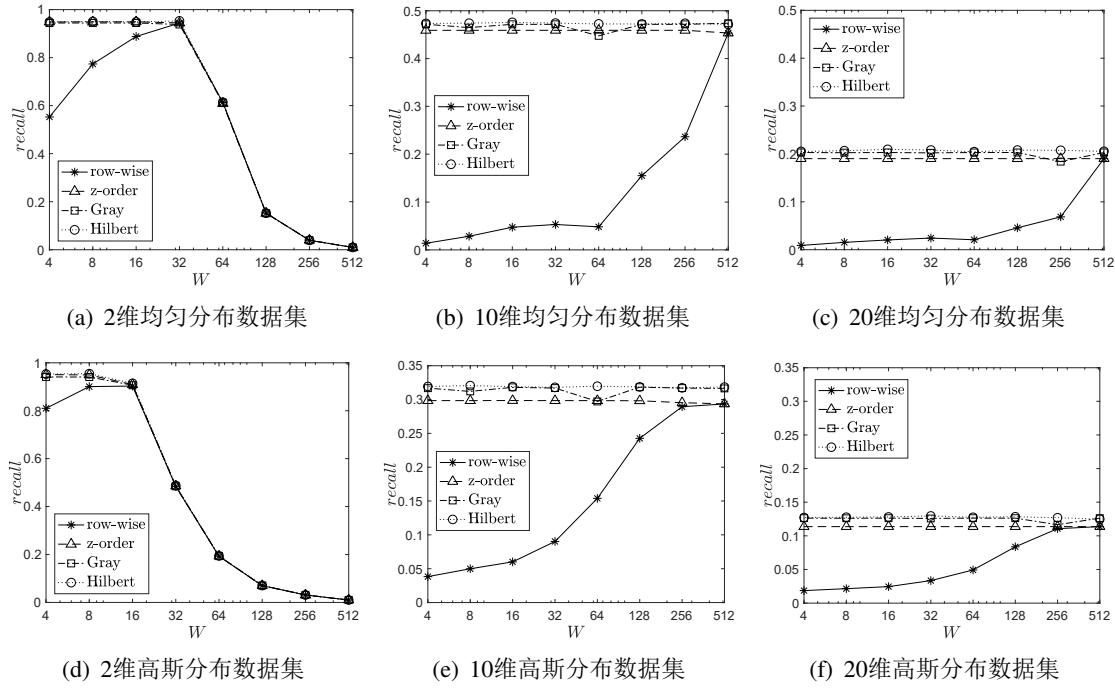


图 4.5 空间曲线近邻搜索量化对比

漏报，而设置较大的桶宽能够帮助缓解这一问题。因为较大的桶宽能够更早地将这些漏报点捕获在同一个网格。如图4.3(a)和图4.4(a)， $B, D$ 两点在当前较小的桶宽下成为了漏报点，如果将网格的宽度扩大一倍，则有可能一开始就将A,B,C,D划分在和O同一个网格，从而被很快加载。然而根据LSH函数的性质，桶宽增大会使得两个较远点的碰撞概率增大，导致候选集中混入更多假阳性点，容易降低近邻搜索精度。为了充分过滤这些干扰点以保证搜索精度，需要增加LSH函数以提升过滤强度，然而这么做会使得索引和计算开销被增大。这就是之前提到的基本排序方案的第三个局限，我们将在本章第4.4节分析 $W$ 对近邻搜索的影响中对此作进一步的讨论。

相比之下，几类NFT曲线的性能则一直比较稳定，且一直优于DFT曲线的性能，这种情况在多维空间尤为明显。这说明了在多维空间中，邻域优先遍历的特性能够比维度优先遍历更好的进行FP点的过滤，同时减少FN点的产生，从而使得近邻搜索时，局部邻域内能够聚集更多高质量近邻点，从而具有更高的近邻召回率。

此外，几类NFT曲线的近邻分布质量基本不受网格宽度 $W$ 变化的影响，也就不会产生DFT曲线的上述局限性。由于网格宽度在下一步设计索引结构时对应着LSH的桶宽，是索引结构中一个十分重要的参数。NFT曲线的这一特性会为克服基本排序方案对LSH关键参数的敏感性以进一步提升搜索算法的实用性奠定基础。

第三点反映了使用空间曲线实现高效近邻搜索的前提：对数据空间充分划分。只有空间中划分出足够多的网格，在使用曲线填充后，线序值才会形成足够的区分力。否则，近邻点和干扰点混在一起，难以有效区分，近邻搜索性能就会下降，这

种情况主要出现在网格粒度较大的时候。举例，图4.5(a)中，当 $W=64$ 时，每个维度最多划分出 $1024/64 = 16$ 个区段，整个空间则只有 $16^2=256$ 个网格，相对于200,000的数据总量，区分力相对来讲是不够的。不过，这一问题会随着维度的升高而逐渐消失，因而，在几个更高维度的数据集上，几类NFT曲线的性能都没有随 $W$ 的变化而有明显的波动。

数据的不同特性对曲线的以上三点趋势没有产生较大的影响。相对于DFT曲线与NFT曲线较大的性能差距，几类NFT曲线之间的性能则比较接近，尤其是Hilbert曲线和Gray曲线。整体上看，Hilbert曲线的性能更好一些。不过，图4.5中的结果是重复生成数据集10次后的平均结果，为了更加细致地比较Hilbert曲线和Gray曲线的优劣，我们又详细统计了这两种曲线在10次实验中8种不同 $W$ 取值下各自取得最优性能的次数（见表4.2），结果显示Hilbert曲线占优次数更多。因此，本章提出一种新的基于排序的LSH索引方案（Optimal Order LSH，O2LSH），利用Hilbert曲线构造线序为复合哈希键值排序，实现近邻候选点的最优排序<sup>1</sup>，从而最大程度地改善磁盘访问效率，同时提升最近邻搜索的准确率。

表 4.2 Gray曲线与Hilbert曲线优势次数比较

曲线占优次数	均匀分布数据		高斯分布数据	
	10维	20维	10维	20维
Gray曲线	35	32	31	28
Hilbert曲线	45	47	49	52

## 4.3 O2LSH

本节详细介绍O2LSH中线序的定义、索引结构的构建以及ANN搜索的过程。

### 4.3.1 线序定义

更优线序的构建是O2LSH进行索引与搜索的核心。为便于描述，首先给出一些必要的定义。

**定义 1. (原始数据点的线序值)** 给定 $d$ 维数据点 $\mathbf{p} \in \mathcal{R}^d$ ，其在复合哈希函数 $G$ 下的哈希值 $K = C(\mathbf{p}) = \{k_i\}_{i=1}^m$ 是一个 $m$ 元组。构造 $m$ 维空间的Hilbert曲线<sup>[93]</sup>将 $K$ 在曲线上映射，将得到一个整数值，称其为 $\mathbf{p}$ 的线序值，记为 $O_p$ 。

<sup>1</sup>需要说明的是，本章凡出现类似“最优曲线”或者“最优排序”的声明，其所指的范围均仅限于本章所考察的四类曲线。但实际上，空间曲线技术还有很多种，不排除在剩余的曲线中存在某个会在本问题上展现更好的性能。这也是本章后续工作中正在继续跟进的工作。目前看，剩余的曲线中，一部分与本文中考察的曲线在本质上类似（如snake曲线）；还有一些曲线（如Sierpiński曲线以及分形领域的很多曲线）较少被应用于多维索引与相似性搜索相关问题中，对于其能否有效部署在多维空间还需要进一步的研究和考察。

**定义 2. (线序值最长公共前缀)** 数据点 $p$ 的线序值 $O_p$ 可以表示成一个二进制序列 $O_p = o_1o_2\dots o_U$ , 其中 $U$ 是二进制串的长度。我们用 $\text{pref}(O, l) = (o_1o_2\dots o_l)$ 表示 $O_p$ 的长为 $l$ 的前缀, 其中 $0 \leq l \leq U$ 。对于两个给定线序值 $O_1 = (o_{1,1}o_{1,2}\dots o_{1,U})$ 和 $O_2 = (o_{2,1}o_{2,2}\dots o_{2,U})$ , 如果有 $\text{pref}(O_1, l) = \text{pref}(O_2, l)$ 成立, 且 $\text{pref}(O_1, l+1) \neq \text{pref}(O_2, l+1)$ , 则称 $\text{pref}(O_1, l)$ 是二者的最长公共前缀, 最长公共前缀长度 (*Length of Longest Common Prefix*, 简称LLCP) 为:

$$\text{LLCP}(O_1, O_2) = l$$

如果 $\text{pref}(O_1, U) = \text{pref}(O_2, U)$ , 则称 $O_1, O_2$ 相等,  $\text{LLCP}(O_1, O_2) = U$ 。

根据线序值最长公共前缀的定义可以进一步定义O2LSH所建立的线序关系:

**定义 3. (线序关系)** 给定两个线序值 $O_1, O_2$ , 定义它们之间的偏序关系如下:

$$\begin{cases} O_1 <_O O_2, & l < U \text{ 且 } o_{1,l+1} < o_{2,l+1} \\ O_1 =_O O_2, & l = U \\ O_2 <_O O_1, & l < U \text{ 且 } o_{1,l+1} > o_{2,l+1} \end{cases}$$

其中,  $l = \text{LLCP}(O_1, O_2)$ 。

### 4.3.2 构建索引结构

线序定义好之后, 可以进行O2LSH索引结构的构建, 过程如下:

- (1) 将数据集 $\mathbf{D}$ 中每个数据点在一个复合哈希函数上投影, 将得到的复合哈希键值在Hilbert曲线上映射, 得到所有点的线序值集合 $\mathbf{O}$ ;
- (2) 按照定义3中定义的线序对 $\mathbf{O}$ 中的线序值排序, 得到排序后的线序值集合 $\mathbf{O}'$ ;
- (3) 依照 $\mathbf{O}'$ 中的顺序重新排列原始数据, 得到排序后的数据集 $\mathbf{D}'$ , 将 $\mathbf{D}'$ 存放到磁盘上, 如图4.6;
- (4) 假定一个磁盘页面的容量是 $B$ 个机器字, 则一个页面最多能容纳 $V = \lfloor B/d \rfloor$ 个数据点, 整个数据集 $\mathbf{D}'$ 将占据 $\eta = \lceil n/V \rceil$ 个磁盘页面。对于每个页面, 提取第一个和最后一个数据点的线序值 $\alpha$ 和 $\beta$ 为代表。为所有代表线序值建立B+树进行索引, 即可有效索引所有原始数据 $\mathbf{D}'$ 。由B+树和磁盘中存放的 $\mathbf{D}'$ 共同构成O2LSH的一个哈希表, 如图4.6所示。
- (5) 重复(1)-(4)的步骤建立 $L$ 个哈希表 $T = \{T_i\}_{i=1}^L$ , 完成O2LSH的多索引哈希表的构建。

可以看到, O2LSH的索引结构主要由两部分组成: 一是依照新的空间线序在磁盘上重新排列的原始数据, 二是由代表哈希键值组成的B+树索引。索引结构最大的特点主要有两方面: 1) 使用更加优秀的线序对数据点重新排列, 近邻点的局部分布

质量更高；2) 以磁盘页面为单位进行组织与索引。我们知道，磁盘中的数据通常是以页面为单位存储和访问的，传统的LSH外存索引采用随机读取的方式加载候选点，虽然每一次I/O能读到一个页面的数据，但是其中一般只有一个点是所需要的，这就存在I/O利用率低的问题。O2LSH的索引结构以页面为单位组织和索引数据，将来候选点加载时也是以页面为单位读取，得益于更好的局部分布，一次I/O操作就能够获得更多高质量候选点，I/O效率得以提升。此外，从索引构建的第(4)步可以看到，B+树并不直接在全部复合哈希键值上构建，而是以每个数据页面提取的代表哈希键值为基础。这样做，能够保证B+树索引的效果，也能进一步降低B+树索引的规模，从而能够最大程度地降低B+树的高度，提升近邻搜索时的搜索效率。

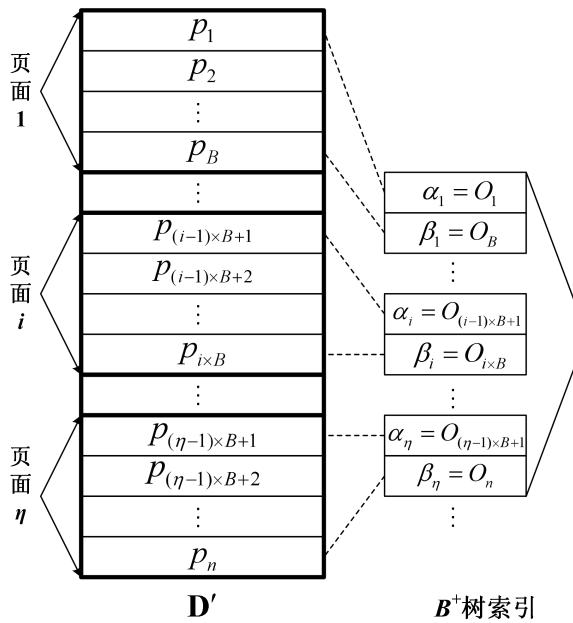


图 4.6 O2LSH的一个哈希表

### 4.3.3 最近邻搜索算法

O2LSH进行 $k$ -近似最近邻搜索( $k$ -ANN)的算法流程见算法1。为便于控制搜索性能，算法的终止条件设置为搜索时允许加载的数据页面数量 $N_P$ 。搜索过程大体可分为两个阶段：1) 候选页面确定阶段；2) 候选点加载与最近邻验证阶段。第一个阶段的任务是找出“质量”最好的 $N_P$ 个起始页面。这需要定义查询点 $q$ 到一个数据页面的距离，作为衡量一个数据页面“质量”好坏的标准。

**定义 4. (查询点到数据页面的距离)** 给定查询点 $q$ 和一个磁盘数据页面 $P$ ， $O_q$ 是查询点的线序值， $\alpha$ 和 $\beta$ 是页面 $P$ 的代表线序值( $\alpha \leq_O \beta$ )，定义 $q$ 到页面 $P$ 的距离如下：

$$DIST(q, P) = \begin{cases} dist(O_q, \alpha), & O_q \leq_O \alpha \leq_O \beta \\ 0, & \alpha \leq_O O_q \leq_O \beta \\ dist(O_q, \beta), & \alpha \leq_O \beta \leq_O O_q \end{cases}$$

其中 $dist(,)$ 表示两个线序值间的距离。例如两个线序值 $O_1, O_2$ , 其计算方式为 $dist(O_1, O_2)=U-LLCP(O_1, O_2)$ 。

根据这一定义, 可以进行最近邻搜索的初始页面定位和迭代搜索, 详细算法如下:

#### 算法1. $k$ -ANN搜索

输入: $\{T_i\}_{i=1}^L, \mathbf{q}, k, N_P$

输出:  $k$ -ANN

- (1) (初始化) 初始化候选页面集合 $P_C = \phi$ ;
- (2) (定位初始页面) 在每一个哈希表 $T_i$ 上定位到距离 $\mathbf{q}$ 最近的页面, 记为 $T_i$ 的左页面 $P_{iL}$ , 其右侧页面记为 $T_i$ 的右页面 $P_{iR}$ ;
- (3) (迭代确定候选页面) 根据定义4, 计算出当前 $P_C$ 中距离 $\mathbf{q}$ 最近的一个页面 $P^*$ 。若 $P^*$ 来自某哈希表 $T_i$ 的左页面 $P_{iL}$ , 则更新 $P_{iL}$ 指向 $P^*$ 的左侧页面; 否则, 更新 $P_{iR}$ 指向 $P^*$ 的右侧页面。重复此步骤 $N_P$ 次;
- (4) (加载候选点, 验证 $k$ 最近邻) 对每个哈希表 $T_i$ , 沿磁盘顺序加载从 $P_{iL}$ 到 $P_{iR}$ 间的所有页面。计算 $\mathbf{q}$ 与所有加载点的距离, 将最近的 $k$ 个作为 $k$ -ANN结果返回。

#### 4.3.4 算法有效性分析

O2LSH与SK-LSH都是基于排序的LSH索引方案, 二者在索引与搜索机制上十分类似。最大的不同, 在于O2LSH使用了NFT特性的曲线, 能够进一步改善近邻候选点的局部分布, 使得O2LSH在近邻搜索时能够加载到更多高质量的近邻点, 从而获得比SK-LSH更好的搜索性能。

结合图4.7更加清楚地说明这一点。图4.7描绘了这两种方法在索引构建好之后数据点在磁盘中的分布情况。图中一个方框代表一个磁盘页面, 可以容纳多个数据点。对于给定的查询点 $O$ , 所有的黑色实心点是其真实 $k$ 近邻。搜索时, 首先在B+树中搜索确立 $O$ 所对应的磁盘页面, 即起始页面(算法1步骤(2)), 然后执行一个双向搜索确立一系列候选磁盘页面(算法1步骤(3)), 最后从这些候选页面中加载数据点进行精炼(算法1步骤(4))。

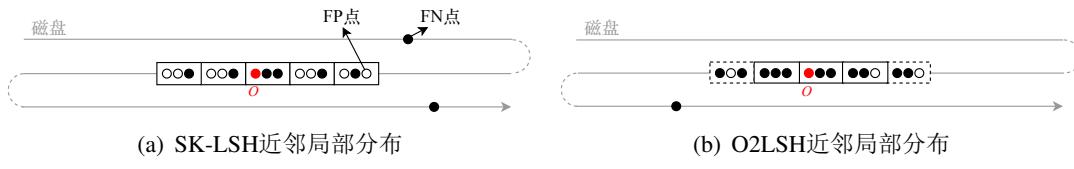


图 4.7 O2LSH与SK-LSH近邻局部分布对比

对于SK-LSH, row-wise曲线DFT的特性会给其所确立的候选页面中引入更多FP点, 并丢失更多真实近邻点(即FN点), 近邻搜索的准确率因此随之下降。而对于O2LSH,

NFT特性的曲线能更好地过滤FP点并丢失更少的真实近邻。最终的结果，O2LSH能够产生更好的近邻局部分布，即在相同（甚至更少）的局部页面中汇集更多近邻点（如图4.7），从而可以用相同（甚至更少）的I/O开销加载到这些高质量候选点，且取得更高的近邻搜索精度。根据第4.4节的近邻搜索实验结果，O2LSH确实在I/O性能和近邻搜索准确率上同时取得对SK-LSH领先。

### 4.3.5 算法复杂度分析

**空间复杂度.** O2LSH的空间开销主要包括三部分。1)  $L$ 组复合哈希函数的参数，占据 $Lmd$ 的空间；2)  $L$ 组有序的数据ID列表，占据 $nL$ 的空间；3)  $L$ 个哈希表，每个哈希表包含一个B+树和一个原始数据集。假设一个磁盘页面的物理大小是 $B$ 个机器字，则一个B+树的空间开销约为 $2ndm/B$ ，其中 $n$ 是数据集规模， $m$ 是复合哈希键值的维度。而一个原始数据集占据的空间大小是 $nd$ 。因此，O2LSH总的空间开销是 $L(md + n + nd(1 + 2m/B))$ 。通常情况下 $2m/B \ll 1$ ，总的空间开销可以化简为 $O(L(md + nd))$ 。由此，可以看到O2LSH索引中最主要的空间开销是原始数据。

**时间复杂度.** O2LSH的时间开销主要包括两部分：1) 在 $L$ 个B+树中搜索定位起始页面的时间；2) 加载并处理 $N_P$ 个候选磁盘页面的时间。

第一部分中，总共需要处理 $LE$ 个页面，其中 $E$ 表示每个B+树的高度。在O2LSH中，每个B+树通过索引 $\eta = \lceil n/V \rceil \approx nd/B$ 个页面的代表哈希键值来索引全部原始数据，B+树的序 $b = \lfloor B/m \rfloor$ ，因此 $E \approx \log_{\frac{B}{m}} \frac{2nd}{B}$ 。处理一个页面需要 $O(B)$ 的时间，因此，第一部分时间复杂度是 $O(LBE)$ 。由于一个磁盘通常能容纳非常多的复合哈希键值，即 $m \ll B$ ，也就是说 $b$ 的值很大，因此O2LSH中B+树的高度不会很大，因此整个第一部分不会耗费太多时间。

第二部分的耗时包含两个方面，一个是加载 $N_P$ 个页面所耗费的I/O时间，另一部分是计算全部候选点的原始距离进行近邻点精炼的时间。这两部分中，主要影响因素都是 $N_P$ 。由于I/O开销和高维距离计算都十分耗时。实际部署时，我们一般会将 $N_P$ 设置为比较小的值，从而让O2LSH的第二部分也不会耗费太多时间。

**I/O开销.** 根据时间开销的分析，可以直观地看出O2LSH的I/O开销，共包括两部分。一个是在 $L$ 个B+树中搜索定位初始页面所需要的磁盘访问，一个是对 $N_P$ 个候选页面的访问，总共是 $LE + N_P$ 。

## 4.4 实验结果与分析

本节设计实验评估O2LSH的ANN搜索性能，并与几类最先进的LSH索引技术，包括C2LSH、SK-LSH、SRS以及QALSH，进行综合的性能对比。实验在一台惠普工作站上进行，CPU主频3.6GHz，内存64GB，硬盘2TB，操作系统为Ubuntu 14.04

LTS，代码采用C/C++编写，编译器为g++ 4.6.3。

#### 4.4.1 实验设置

选取了6个公开的真实多媒体特征数据集进行对比实验。数据类型涵盖文本、图像和音频数据。详细信息见表4.3，其中 $n$ 表示数据集包含的特征向量个数， $d$ 表示特征向量维度。实验中，Mnist和Gist1M数据集的磁盘页面大小设置为16KB，其余数据集为4KB。

表 4.3 实验数据集

名称	$n$	$d$	说明
WT <sup>2</sup>	269,648	128	小波纹理图像特征
Sift1M <sup>3</sup>	1,000,000	128	图像SIFT特征
Audio <sup>4</sup>	53,387	192	音频特征
Glove200d <sup>5</sup>	1,192,514	200	twitter文本词频特征
Mnist <sup>6</sup>	69,000	784	手写字体图像特征
Gist1M <sup>7</sup>	1,000,000	960	全局图像特征

从三个方面衡量算法在外存环境下进行ANN搜索的性能：ratio、I/O开销和空间开销。

**ratio：**评估ANN搜索的准确率。给定查询点 $\mathbf{q}$ ，令 $o_1^*, o_2^*, \dots, o_k^*$ 依次代表 $\mathbf{q}$ 的前 $k$ 个真实最近邻，令 $o_1, o_2, \dots, o_k$ 依次代表搜索算法返回的 $k$ 最近邻。则该查询点 $\mathbf{q}$ 的ratio的计算如下：

$$ratio(\mathbf{q}) = \frac{1}{k} \sum_{i=1}^k \frac{\|o_i, \mathbf{q}\|}{\|o_i^*, \mathbf{q}\|} \quad (4-1)$$

**I/O开销：**算法进行ANN搜索过程中访问磁盘页面的次数，是衡量算法进行外存ANN搜索效率的一个重要指标。

**空间开销：**指算法进行ANN搜索所需数据总的存储开销，一般包括数据集和索引两部分。

每个数据集随机选取200个点作为查询点，取所有查询点的平均性能作为最终性能。

#### 4.4.2 O2LSH与SK-LSH性能对比

本章的O2LSH和基本排序方案SK-LSH都是基于空间线序的LSH索引技术，二者具有直接的竞争关系。本小节专门对这两个方法进行详细的ANN搜索对比。在四

<sup>2</sup><http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm>

<sup>3</sup><http://corpus-texmex.irisa.fr/>

<sup>4</sup><http://www.cs.princeton.edu/cass/audio.tar.gz>

<sup>5</sup><http://nlp.stanford.edu/projects/glove/>

<sup>6</sup><http://yann.lecun.com/exdb/mnist/>

<sup>7</sup><http://corpus-texmex.irisa.fr/>

个不同参数（LSH函数桶宽 $W$ 、复合哈希函数中LSH函数个数 $m$ 、ANN搜索终止条件 $N_p$ 和哈希表数目 $L$ ）变化下评估两种方法进行 $k=10$ 近邻搜索的性能。图4.8列出的是在Sift1M数据集上的对比结果。

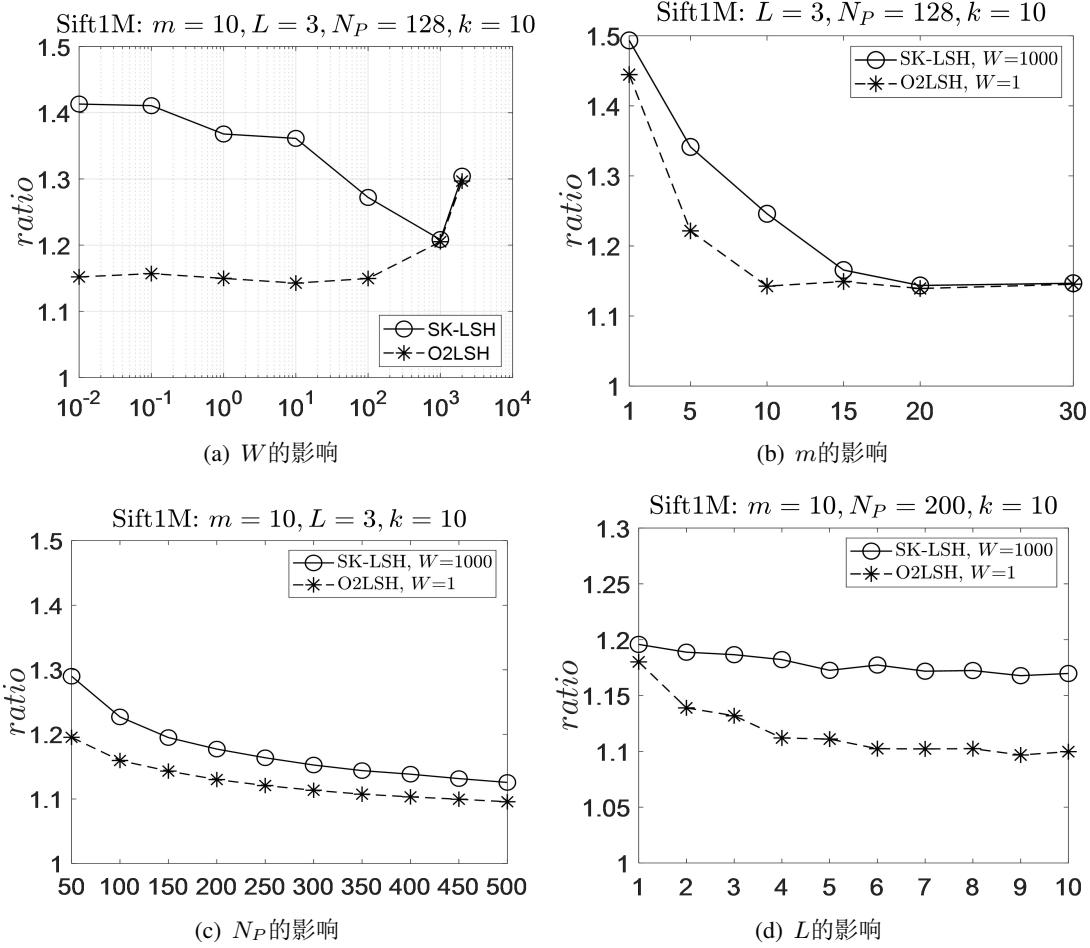


图 4.8 O2LSH与SK-LSH性能对比

**$W$ 的影响.** 图4.8(a)中两种方法的LSH函数桶宽 $W$ 都以10倍的速率变化。可以看到，SK-LSH的准确率对桶宽变化十分敏感： $W$ 偏小或偏大时ratio都比较差，只有在设置合适的桶宽（图中为 $W=1000$ ）时才能取得最好的准确率；而O2LSH的准确率则相对比较稳定，在 $W$ 很大的变动范围内（从 $10^{-2} \sim 10^2$ 跨4个数量级的范围）ratio几乎没有波动，一直处于最优值，且其最优值优于SK-LSH的最优ratio。但是在 $W$ 取1,000和更大数值时，ratio开始变差，出现和SK-LSH的性能重合的现象。

正如前面所分析的，SK-LSH对 $W$ 比较敏感是因为DFT特性让row-wise曲线在局部容易错失近邻点，设置较大的桶宽一定程度上能弥补这一问题。但根据图4.8(a)，两种曲线的桶宽都不宜设置过大。图4.8(a)中，我们看到SK-LSH曲线的准确率在 $W > 1000$ 之后没有进一步改善，而是出现拐点；O2LSH在 $W > 1000$ 之后准确率也急速变差。其原因都是桶宽设置过大使得单个哈希函数上划分不出足够多的桶，使得整个

空间无法被充分划分造成哈希值混淆在一起，近邻难以区分，最终降低准确率。这和图4.5中出现的第三点现象是同一个原因，反映了应用空间曲线实现高效近邻搜索的前提：对数据空间充分划分。为了便于在实际应用中将基于排序的方法应用于不同的数据集，我们给出一种评估空间划分程度的方法。

首先计算不同数据集在LSH函数上投影的平均范围，算法如下：

**算法2.** 计算数据集的LSH平均投影范围

**输入:** 数据集 $D$

**输出:** 平均投影范围 $R_H$

- (1) 随机生成足够多的LSH随机向量 $a$ ；
- (2) 将数据集 $D$ 在所有随机向量上投影，计算每个向量上的投影值范围；
- (3) 所有投影值范围取平均即作为 $R_H$ 返回；

我们在Sift1M数据集上共生成1,000个LSH随机向量，根据这一算法可以计算得到Sift1M数据集在LSH函数上的平均投影范围，约为3,067。这就意味着， $W=1,000$ 时，每个维度平均能划分约 $[R_H/W] = \lceil 3067/1000 \rceil = 4$ 个桶，整个空间约能产生1,048,576（即 $4^{10}$ ）个网格，稍微大于Sift1M数据集的规模（即1,000,000）；而当 $W=2,000$ 时，每个维度平均只能划分 $[3067/2000] = 2$ 个桶，整个空间只有约1,024个网格，网格数量远远小于数据集规模，这种情况下很难保证对数据集进行充分划分，因此不管采用何种空间曲线，哈希值都极容易混淆在一起难以区分。而反过来，当 $W=100$ 时，整个空间约能产生 $30^{10}$ 个网格，空间被充分地划分，因此O2LSH能够在此情况下产生较好的ANN搜索效果（如图4.8(a)）。

SK-LSH对桶宽敏感且青睐较大桶宽除了会遇到空间划分不充分的问题，还会产生其他两个局限：1)  $W$ 的最优取值范围很小，使得SK-LSH面对新数据集时需要进行大量的调参工作才能确立最优性能，算法实用性受到很大影响，这是第一章提到的SK-LSH的第二个局限；2) 较大的桶宽在更早地捕获更多近邻点的同时也会引入更多假阳性点，为了对这些新的干扰点进行充分过滤，需要增加LSH函数个数，最终导致LSH函数开销增大，见图4.8(b)中 $m$ 变化时的分析。

**$m$ 的影响.** 图4.8(b)中，SK-LSH的桶宽定为其在图4.8(a)中取得最优ratio时的桶宽，即 $W=1000$ 。对于O2LSH，由于其对 $W$ 取值不太敏感， $W$ 取1。

可以看到，两个方法的ratio一开始都随着 $m$ 值的增大而迅速下降，说明增加哈希函数确实能过滤掉更多假阳性点。但当 $m$ 增大到一定程度，再增加 $m$ ，ratio基本趋于平稳。说明存在一定的哈希函数数量能够充分过滤掉假阳性点，没有必要一味增加哈希函数。而且更多的哈希函数意味着更多的存储和计算开销，因此最优的 $m$ 值便取在ratio刚开始趋于平稳时的值。按照图4.8(b)中的结果，SK-LSH的最优 $m$ 值约为20，O2LSH则在 $m=10$ 时取得最好ratio。O2LSH使用一半的哈希函数开销取得了

和SK-LSH相当甚至更优的搜索准确率。

正如之前提到的，产生这种区别的主要原因是来自两种方法曲线特性上的不同。SK-LSH采用维度优先遍历的线序，容易错失更多的近邻点，需要用更大的桶宽来“捕捉”。然而较大的桶宽会导致桶内混入更多假阳性点，需要增多哈希函数来过滤掉；相应地，O2LSH中邻域优先遍历的线序能够更早地将更多近邻点索引到，混入的假阳性点很少，因此只需要较少的哈希函数进行过滤。

$N_P$ 的影响. 图4.8(c)中横坐标改变的是搜索的终止条件，即允许加载的候选页面个数 $N_P$ 。可以看到，加载更多候选点，两类方法的准确率都有显著提升。因为 $N_P$ 增大意味着局部搜索时的范围增大，能够考察更多候选点，因此有机会召回更多最近邻，提升搜索准确率。不过， $N_P$ 增大也会增加算法的I/O开销和时间开销。实际搜索时，需要根据需求进行准确率和效率的权衡。

$L$ 的影响. 同 $N_P$ 类似，更多的哈希表一定程度上也都能提升搜索准确率。因为更多哈希表能相互弥补丢失的近邻，有助于克服假阴性问题。但是，更多的哈希表意味着更大的空间开销。因此需要权衡搜索精度与效率来选取最合适的 $L$ 值。本章一般构建 $L=3$ 个哈希表。

#### 4.4.3 ANN搜索综合对比

本小节，我们将O2LSH与目前几类最先进的LSH方法（包括C2LSH、SK-LSH、SRS和QALSH）进行综合的ANN搜索性能对比。由于这几类算法都证实了对LSB-Forest的优越性，此处不再对LSB-Forest进行对比。

##### 4.4.3.1 参数设置

各个算法以取得最好的精度与效率平衡为标准选取参数，详细参数设置见表4.4。其中，对于C2LSH，我们使用其高效率版本（即 $Ct=1$ ），近似比率 $c$ 统一取3；QALSH的近似比率统一取5或6；对于SRS，参考原论文推荐，近似比率 $c$ 和哈希函数个数 $m$ 分别取4和6。对于SK-LSH和O2LSH，为深入对比二者的线序索引性能，我们让二者建立同等规模的哈希索引结构，即哈希表数目相同（即 $L=3$ ）、每个哈希表中复合哈希函数包含的LSH函数个数相同（即 $m=10$ ）。 $W$ 的变化不会影响索引开销，但是会影响搜索精度，因此需要设置到最优值以获取最优精度。为此，SK-LSH需要对不同数据集进行调参以确立最优的参数取值；而O2LSH对桶宽不敏感，最优值取值范围比较宽，只需要确保哈希映射后的空间被充分划分即可。为此，我们根据算法2统计出各个数据集的平均投影范围 $R_H$ ，将 $R_H$ 统一缩小1000倍作为最终的 $W$ 值。各个数据集经过统计得到的 $R_H$ 值和两个方法最终确立的 $W$ 值见表4.4。最后， $N_P$ 兼顾精度和I/O开销设置一个合适的值。

实验在全部6个数据集上实施，考察各个方法ANN搜索的ratio、I/O以及空间开

表 4.4 不同ANN搜索方法在各数据集上的参数选取

数据集 ( $R_H$ ) \ 方法	C2LSH ( $C_t, c, m$ )	QALSH ( $c, m$ )	SRS ( $c, m$ )	SK-LSH ( $W, m, L, N_P$ )	O2LSH ( $W, m, L, N_P$ )
WT (65.7)	(1, 3, 230)	(6, 19)	(4, 6)	(1, 10, 3, 256)	(0.06, 10, 3, 128)
Sift1M (3,067)	(1, 3, 250)	(5, 21)	(4, 6)	(1000, 10, 3, 500)	(3, 10, 3, 350)
Audio (10.5)	(1, 3, 204)	(5, 16)	(4, 6)	(3, 10, 3, 200)	(0.01, 10, 3, 170)
Glove200d (80)	(1, 3, 252)	(5, 21)	(4, 6)	(15, 10, 3, 500)	(0.08, 10, 3, 400)
Mnist (15,176)	(1, 3, 208)	(5, 17)	(4, 6)	(2000, 10, 3, 250)	(15, 10, 3, 200)
Gist1M (23.3)	(1, 3, 250)	(4, 26)	(4, 6)	(2.5, 10, 3, 300)	(0.02, 10, 3, 250)

销，结果分别见图4.9，图4.10和图4.12。

#### 4.4.3.2 综合对比ratio和I/O

综合图4.9的ratio性能和图4.10的I/O性能进行观察。可以发现，SK-LSH以较少的I/O开销取得了较为领先的搜索准确率，反映了基于排序的LSH思想同时取得较好的ANN搜索精度和I/O效率的基本能力；QALSH的搜索精度仅次于O2LSH和SK-LSH，在部分数据集上甚至超越SK-LSH（WT数据集），但无一例外地，QALSH是所有方法中I/O开销最大的；SRS方法的I/O性能仅次于O2LSH和SK-LSH，但其搜索精度距离二者却有很大差距；C2LSH的I/O性能对维度变化不是特别敏感，但是受数据集规模变化影响较明显：在较小规模的数据集（Audio和Mnist）上I/O性能匹敌甚至超过O2LSH，但是在大规模数据集（Sift1M、Glove200d以及Gist1M）上I/O开销迅速增长。

最后，如图4.9和图4.10，O2LSH几乎在所有数据集上都能以最少的I/O开销取得最好的搜索准确率，领先于基本排序方案和其他先进方案，且领先性不受数据集维度和数据集规模的影响。

此外，我们还统计了所有方法的平均ratio和平均I/O，数值结果分别见表4.5和表4.6，可视化对比结果见图4.11。可以看到O2LSH能同时取得最好的准确率和I/O效率。其中，相对于基本方案SK-LSH，O2LSH在准确率全面提升的情况下，I/O开销降低了14%~48%不等。实验结果证实了本章提出的基于最优线序的索引方案相对于基本排序方案和其他方案的优越性，确实能够有效改善近邻候选点的局部分布，实现在相同（甚至更小）范围内维护更多高质量近邻候选点，最终同时提升I/O效率和搜索精度。

表 4.5 ANN搜索平均ratio（数值结果）

ratio	WT	Audio	Sift1M	Glove200d	Mnist	Gist1M
C2LSH	1.268953	1.24457	1.29723	1.241162	1.303211	1.236579
QALSH	1.118804	1.137593	1.180986	1.200379	1.191545	1.152471
SRS	1.245954	1.245954	1.250638	1.18517	1.245954	1.175362
SK-LSH	1.131237	1.115522	1.148983	1.138871	1.149203	1.138134
O2LSH	<b>1.09878</b>	<b>1.104771</b>	<b>1.124887</b>	<b>1.133074</b>	<b>1.129808</b>	<b>1.129679</b>

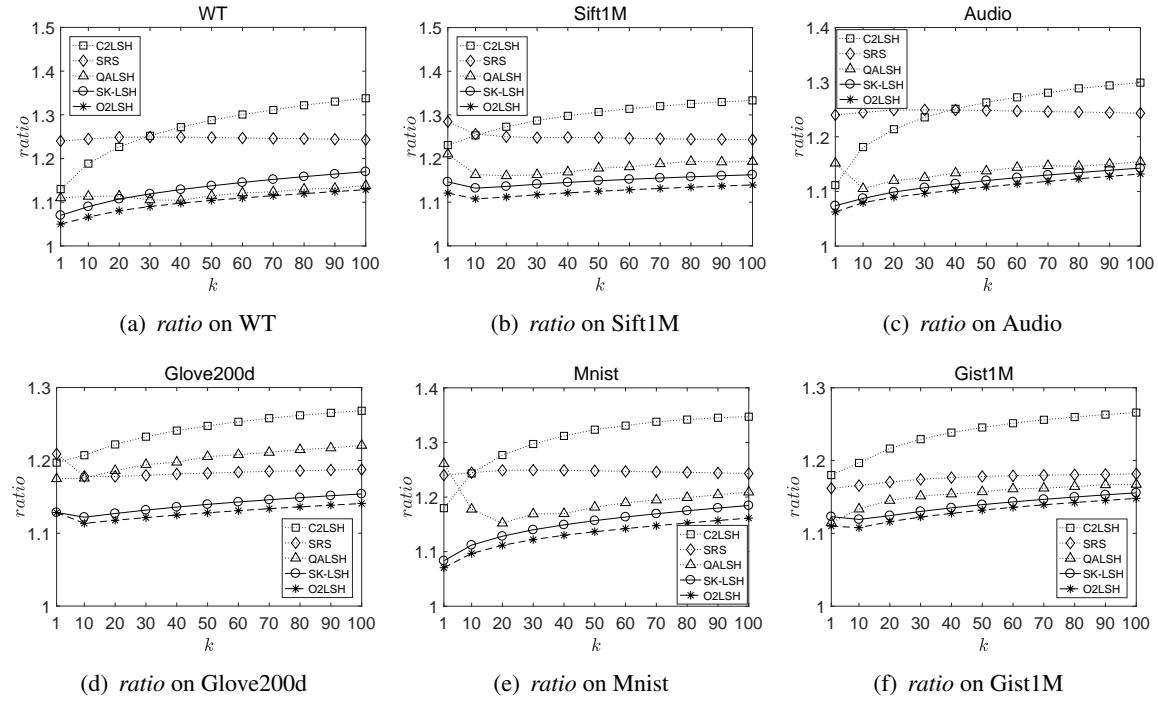


图 4.9 ANN 搜索ratio 对比

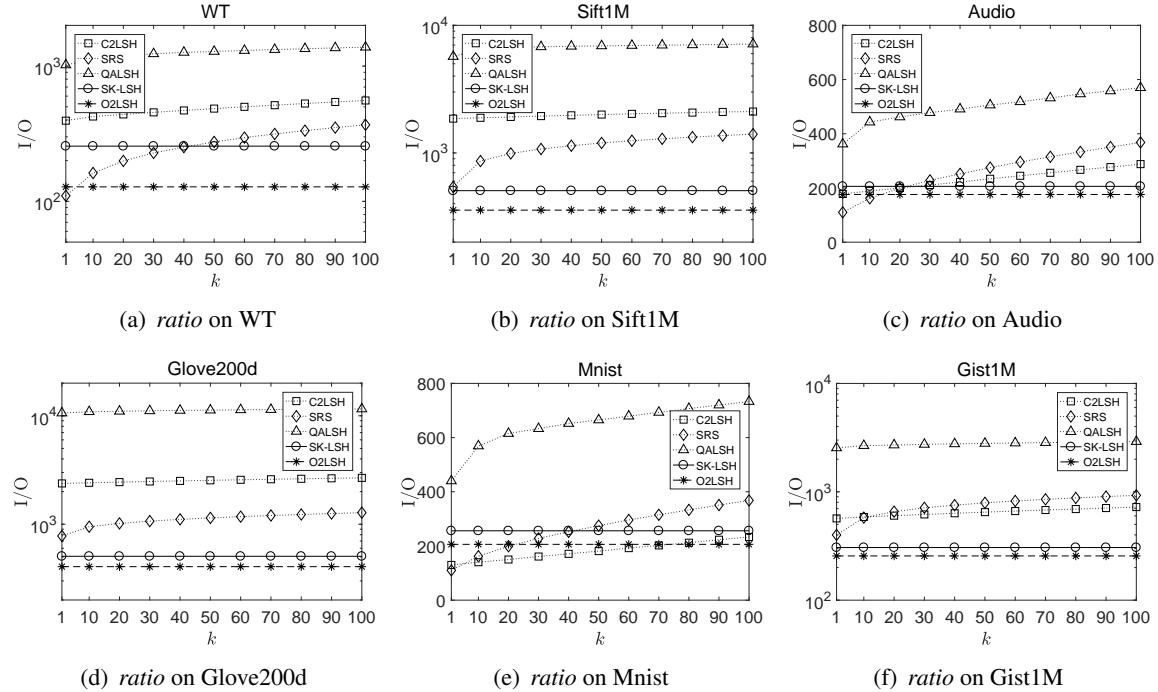


图 4.10 ANN 搜索I/O 开销对比

#### 4.4.3.3 空间开销

本节我们比较各个近邻搜索方法取得以上近邻搜索性能所付出的空间开销。我们使用了两种空间开销指标，一个是搜索算法的总空间开销，一个是搜索算法索引结构的空间开销。统计结果分别见图4.12(a)和图4.12(b)。

表 4.6 ANN 搜索平均 I/O (数值结果)

<b>IO</b>	<b>WT</b>	<b>Audio</b>	<b>Sift1M</b>	<b>Glove200d</b>	<b>Mnist</b>	<b>Gist1M</b>
C2LSH	483	233	1992	2534	181	648
QALSH	1256	497	6781	11218	646	2776
SRS	263	263	1131	1109	263	752
SK-LSH	262	206	506	506	256	306
O2LSH	<b>134</b>	<b>176</b>	<b>356</b>	<b>406</b>	<b>206</b>	<b>256</b>
(SK-O2)/SK	48.85%	14.56%	29.64%	19.76%	19.53%	16.34%

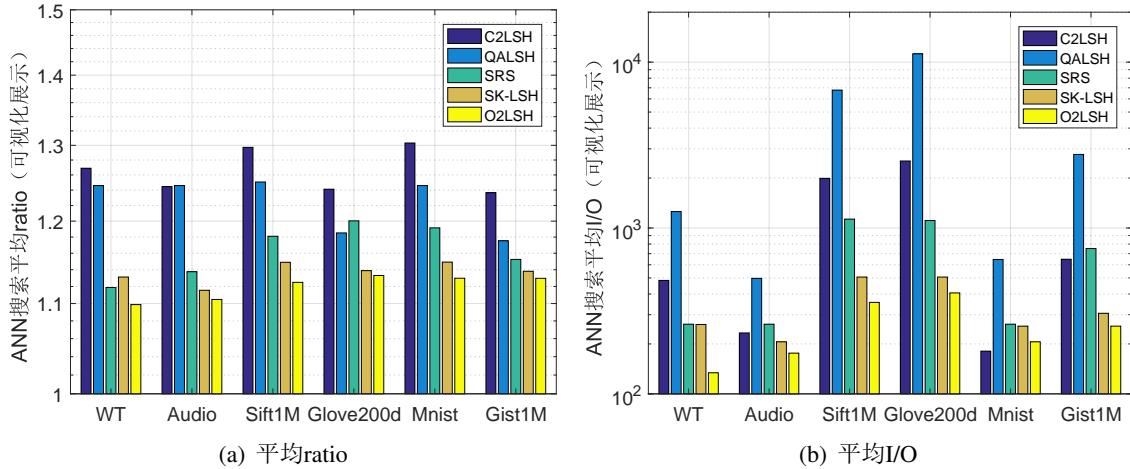


图 4.11 ANN 搜索平均性能对比

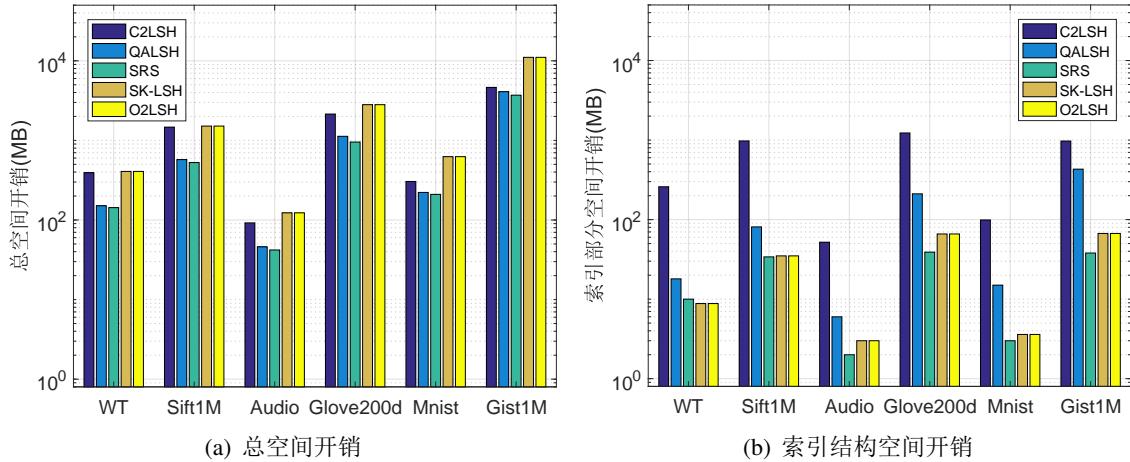


图 4.12 空间开销对比

从图4.12(a)可以看到，SRS总的空间开销最少，其次是QALSH和C2LSH，SK-LSH和O2LSH产生了最大的空间开销。为便于分析，我们将原始空间开销值进行了转换，转换成一种相对空间开销比例，即拿算法实际的空间开销比上原始数据集的物理大小得出一个比例值，结果见表4.7和表4.8。

从表4.7可以明显地看到，对所有这些LSH外存索引方法来说，空间开销中最主要的成分是原始数据的存储。在这一点上，SRS最为明显，其次是QALSH。这两种方法都只存储一份原始数据，因此二者的相对空间开销比例都是稍稍大于1的值。

表 4.7 总空间开销相对比例

数据集 (大小/MB)	C2LSH	QALSH	SRS	SK-LSH	O2LSH
WT (133)	2.95	1.14	<b>1.08</b>	<u>3.07</u>	<u>3.07</u>
Sift1M (493)	2.98	1.16	<b>1.07</b>	<u>3.07</u>	<u>3.07</u>
Audio (40)	2.3	1.15	<b>1.05</b>	<u>3.08</u>	<u>3.08</u>
Glove200d (915)	2.34	1.23	<b>1.04</b>	<u>3.07</u>	<u>3.07</u>
Mnist (207)	1.48	1.07	<b>1.01</b>	<u>3.02</u>	<u>3.02</u>
Gist1M (3,665)	1.27	1.12	<b>1.01</b>	<u>3.02</u>	<u>3.02</u>

表 4.8 索引结构空间开销相对比例

数据集 (大小/MB)	C2LSH	QALSH	SRS	SK-LSH	O2LSH
WT (133)	<u>1.95</u>	0.14	<b>0.08</b>	<b>0.07</b>	<b>0.07</b>
Sift1M (493)	<u>1.98</u>	0.16	<b>0.07</b>	<b>0.07</b>	<b>0.07</b>
Audio (40)	<u>1.30</u>	0.15	<b>0.05</b>	0.08	0.08
Glove200d (915)	<u>1.34</u>	0.23	<b>0.04</b>	0.07	0.07
Mnist (207)	<u>0.48</u>	0.07	<b>0.01</b>	<b>0.02</b>	<b>0.02</b>
Gist1M (3,665)	<u>0.27</u>	0.12	<b>0.01</b>	<b>0.02</b>	<b>0.02</b>

C2LSH在不同数据集上则出现了一定的波动，虽然也是只用存储一份原始数据，但是C2LSH会生成比较多的哈希函数，其个数有时会接近甚至超过原始数据集维度，造成其空间开销也比较高。不过由于C2LSH对维度较好的可扩展性，随着维度的升高，哈希表所占的空间比例相对数据集大小会下降，此时C2LSH的相对空间开销会降下来。

而SK-LSH和O2LSH都是多哈希表结构，需要存储多份原始数据，因此总的空间开销会比较大。从表4.7中可以看到，二者的相对空间开销值都是大于3的数值，这和设置的哈希表个数 $L$ 的取值有关系：O2LSH和SK-LSH在索引构建时所设置的哈希表个数 $L$ 都是3。另外可以看到，由于O2LSH与SK-LSH相比最主要的区别在于改善了线序，二者使用的索引结构在架构上是类似的，因此二者几乎具有相同规模的空间开销。

不过，如果从总的空间开销中减去原始数据，只观察索引部分的大小，如图4.12(b)和表4.8，会发现O2LSH和SK-LSH的索引（即B+树部分）开销显著减少，超越C2LSH和QALSH，仅次于tiny级的SRS方法。

## 4.5 本章小结

为进一步提升外存环境下的高维ANN搜索的性能，本章研究了一种基于最优排序的LSH索引方案O2LSH。一方面，基于LSH技术构造索引结构有效克服维度灾难问题；另一方面，通过引入空间填充曲线技术对复合LSH键值进行排序能够增强近邻候选点分布的局部性，提升I/O效率。为获得最优的局部分布，对典型空间填充曲线的特性进行了深入分析。发现：1) 基本排序方案使用的row-wise曲线具有DFT特性，会给ANN搜索带来多个局限；2) NFT特性的曲线能够产生更好的局部分布，且

性能更稳定。经过对比，选出一种排序效果最好的NFT曲线作为O2LSH的线序。实验结果证实，O2LSH能够进一步提升ANN搜索的I/O效率和精度，性能优于基本排序方案和其他先进LSH索引。其中，相对于基本排序方案，能够减少对关键参数变化的敏感性，是一种更加高效、实用的ANN搜索方法。

下一章将在本章工作的基础上，从另一个角度继续解决I/O效率低的问题，进一步提升外存高维ANN搜索性能。



## 第五章 针对近似最近邻搜索的基于有序短判别码的局部敏感哈希索引

上一章提出的最优排序方案从增强近邻候选点局部分布的角度研究了外存ANN搜索I/O效率的改善。不过，由于使用原始数据作为候选点，这一方法的性能提升受到很大限制。因为高维原始数据具有较大的物理尺寸，单个磁盘页面能够容纳的原始数据数量十分有限，即使优化了局部分布，单次I/O也不能加载到非常多的近邻候选点，因此整体上仍旧需要较多I/O操作。本章设计了一种基于有序短判别码的LSH方法克服这一局限。考察了一种空间尺寸小同时具有较高相似度保持能力的短判别码，乘积量化编码。使用这种编码代替原始点作为候选数据能够大大提升单个磁盘页面中的候选数据数量。结合上一章的最优排序方案调整短判别码的存储顺序，能够让更多相似数据的编码聚集在局部磁盘页面从而仅用少量顺序I/O即可加载，最终进一步提升ANN搜索的I/O效率。

### 5.1 概述

在高维ANN搜索领域，哈希是一种十分重要的技术手段<sup>[94–101]</sup>。通过将原始数据转换为一种低维表达（low-dimensional representation）或者二进制短编码，能够获得很高的存储和计算效率。根据Wang等人的综述<sup>[94]</sup>，在ANN搜索领域主要存在两种形式的哈希技术。一种将相似数据映射到相同哈希桶，并据此构建哈希表索引数据，能够发展出很高效的索引结构；另一种将原始数据映射为物理尺寸很小的短编码，根据短编码计算出的距离能够快速而准确地估计原始距离。

第一种哈希的典型代表是局部敏感哈希技术（Locality Sensitive Hashing, LSH）<sup>[33–35]</sup>，在ANN搜索过程中能够快速过滤无关数据，取得次线性时间复杂度，有效克服维度灾难问题<sup>[88]</sup>。在上一章的概述中已经提到，现有研究基于LSH技术发展了很多先进外存ANN搜索方案，包括LSB-Forest<sup>[36]</sup>、C2LSH<sup>[37]</sup>、SRS<sup>[38]</sup>以及QALSH<sup>[39]</sup>等。但是存在随机I/O开销较大的问题，容易遇到I/O性能瓶颈。

基于排序的方案（包括SK-LSH<sup>[40]</sup>和上一章提出的O2LSH）利用空间曲线技术改善近邻候选点的局部分布，实现了用更快的顺序I/O加载近邻候选点，ANN搜索的I/O效率得以提升。但是由于使用原始数据作为候选点，性能提升会受到很大限制。因为在高维空间，原始数据一般具有很大的体量，而一个磁盘页面的容量一般是固定的，这就导致一个磁盘页面能够容纳的候选点十分有限。举例，假设一个物理磁盘页面的大小是4KB，对于常用的128维SIFT特征<sup>[102]</sup>，一个磁盘页面最多只能

容纳8个原始特征点<sup>1</sup>。而且维度越高，能够容纳的原始数据越少。这种情况下，即使对近邻候选点的局部分布进行了充分的优化，单次I/O能够加载到的近邻候选点也十分有限。如果想要加载充分多的近邻候选点保证ANN搜索的精度，整体上需要的I/O操作依旧较多。

造成这一问题的主要原因是使用原始数据点作为近邻候选点，这也是现有外存LSH方法的普遍选择。因为在精炼环节需要足够高的精度将最近邻从候选集中区分出来，原始数据能够为近邻精炼提供最准确的参考，即原始距离。但问题是，原始数据不一定是近邻候选数据最好的选择。

受非结构化数据海量、高维特性的挑战，一个有效的外存ANN搜索方案通常需要在三个环节都具有很好的表现。第一是无关点过滤阶段或候选点定位阶段，主要任务是快速过滤和查询点不相关的数据，确立一组高质量的近邻候选点；第二是候选点加载阶段，受内外存间较大的通讯代价（即I/O）的影响，候选点加载的效率也十分重要；第三是近邻精炼阶段，主要任务是从加载到的候选数据中快速准确地鉴别出最终的最近邻结果。对于一个高效的外存ANN方案来说，好的候选数据需要满足三方面要求：能够被高效存储和加载、距离计算快、距离精度足以鉴别最近邻。对照这个标准，原始数据并不是一个很好的选择。第一，存储不高效。为了使用原始数据进行近邻精炼，所有现有外存LSH方案都需要在索引结构中维护至少一份原始数据集，这其实会占用很大的空间；第二，加载不高效。上面已经提到，对于基于排序的方法，高维原始数据会严重限制I/O性能的提升；第三，原始距离计算十分耗时。

对于使用原始点作为近邻候选数据的方案来说，合理的做法是提升第一环节的过滤能力。通过提升过滤精度，降低候选集规模，可以降低之后所有环节的负担。实际上，这也是现有一些外存LSH方案的努力方向，比如C2LSH和QALSH。不过，根据本章的实验评估，目前还没有取得令人满意的性能。

本章发现一种更好的近邻候选数据选项，也就是上文提到的第二种哈希——短判别码（discriminative short code）技术<sup>[94, 96]</sup>。这种技术通过将原始距离嵌入短编码，能够在保持对原始距离估计精度的同时提供很高的存储和计算效率。不过，短判别码自身通常不具有很好的可索引性，一般用于基于内存的ANN搜索方案中。本章发现这两种哈希具有很好的互补性，因此将二者结合提出一种新的外存ANN搜索方法，基于有序短判别码的LSH方法（SortingCodes-LSH，简称SC-LSH），进一步提升ANN搜索性能。具体地，首先选取一种优秀的短判别码，乘积量化编码（Product Quantization，简称PQ）<sup>[62, 63]</sup>，代替原始数据进行存储和近邻精炼，提升单个磁盘页面中的候选数据数量。之后，利用上一章的最优排序方法重新组织数据编码，让更

<sup>1</sup>本章默认原始数据的每一维分量都是一个4字节大小的整数/浮点数。

多相似数据的PQ编码排列在局部。如此，仅用少量顺序I/O即可加载到充分多的候选数据，进一步提升ANN搜索的I/O效率。实验结果表明，与现有最先进LSH方法相比，SC-LSH可以在ANN搜索中取得最好的性能。其中，在大规模的*GIST 1M*数据集上，相比于现有方法，能够在取得最好搜索精度的同时降低90%的I/O开销和超过97%的空间开销。

本章余下的内容组织如下。第5.2节介绍一种先进的短判别码技术。第5.3节详细介绍本章的SC-LSH方法，包括方法设计、有效性分析以及索引构建和最近邻搜索算法。第5.4节进行实验评估和分析。最后，第5.5节总结本章工作。

## 5.2 一种先进的短判别码技术

短判别码技术<sup>[96, 97, 103–105]</sup>在基于内存的快速最近邻搜索领域有很多研究工作。这些工作最主要的研究努力在于建立一种有效的编码方式，要有很高的空间效率，支持快速的距离计算，并能很好地保持原始空间的相似性。短判别码研究中一个杰出代表是乘积量化技术（Product Quantization，简称PQ）<sup>[62, 63]</sup>。这是一种矢量量化技术的发展，基于对子空间进行矢量量化并通过笛卡尔积进行结合的方式，能够生成大量的量化中心，以非常高的精度将原始数据量化为紧凑编码。

一个PQ量化器 $\pi_{k^*}^{\mathcal{M}}$ 包括两个关键参数 $\mathcal{M}, k^*$ ，量化过程分为三个步骤。第一步，将原始 $d$ 维空间分解为 $\mathcal{M}$ 个 $\mathcal{S}$ 维的子空间， $d = \mathcal{M}\mathcal{S}$ 。由此，每个原始数据 $\mathbf{p} \in D$ 可以看成由 $\mathcal{M}$ 个 $\mathcal{S}$ 维子向量 $\mathbf{u}_i(\mathbf{p}) (1 \leq i \leq \mathcal{M})$ 连接而成：

$$\mathbf{p} = \underbrace{\mathbf{p}^1, \dots, \mathbf{p}^{\mathcal{S}}}_{u_1(\mathbf{p})}, \dots, \underbrace{\mathbf{p}^{d-\mathcal{S}+1}, \dots, \mathbf{p}^d}_{u_{\mathcal{M}}(\mathbf{p})}$$

第二步，对于每个子空间，执行矢量量化器 $\pi_i(\cdot)$ （如K-Means方法）获得一个包含 $k^*$ 个子码字（sub-codewords）的子码书（sub-codebook） $\mathbf{C}_i$ 。

所有子码书的笛卡尔积（Cartesian product）构成整个数据空间的码书 $\mathbf{C} = \mathbf{C}_1 \times \dots \times \mathbf{C}_{\mathcal{M}}$ 。 $\mathbf{C}$ 一共包含 $(k^*)^{\mathcal{M}}$ 个码字，每个码字都是一个 $d$ 维向量，可以作为一个矢量量化中心。PQ使用这些大量的量化中心能够对一个原始数据 $\mathbf{p}$ 进行精细的量化：

$$\begin{aligned} \mathbf{p} \rightarrow \pi(\mathbf{p}) &= (\pi_1(\mathbf{u}_1(\mathbf{p})), \dots, \pi_{\mathcal{M}}(\mathbf{u}_{\mathcal{M}}(\mathbf{p}))) \\ &= (\mathbf{C}_1[I_1], \dots, \mathbf{C}_{\mathcal{M}}[I_{\mathcal{M}}]) \end{aligned}$$

这里， $I_i = \arg \min_{0 \leq j < k^*} \|\mathbf{u}_i(\mathbf{p}), \mathbf{C}_{ij}\|$ 代表在第 $i$ 个子空间中，子向量 $\mathbf{u}_i(\mathbf{p})$ 被量化到的子码字的编号。所有 $\mathcal{M}$ 个编号的集合作为 $\mathbf{p}$ 的PQ编码 $Q(\mathbf{p}) = (I_1, \dots, I_{\mathcal{M}})$ 。实际使用时， $k^*$ 通常被设置为2的指数，令 $\mathcal{U} = \log_2^{k^*}$ ，则每个PQ编码可以用一个 $\mathcal{U}\mathcal{M}$ 比特长的二进制串表示，占据 $s = \mathcal{U}\mathcal{M}/32$ 的机器字（machine word）空间。

给定查询点 $\mathbf{q}$ 和数据点 $\mathbf{p}$ , 可以计算一种非对称量化距离 (Asymmetric Quantizer Distance, 简称AQD) 估计二者之间的原始距离:

$$\|\mathbf{q}, \mathbf{p}\|^2 \approx \|\mathbf{q}, \pi(\mathbf{p})\|^2 = \sum_{i=1}^{\mathcal{M}} \|\mathbf{u}_i(\mathbf{q}), \mathbf{C}_i[I_i]\| \quad (5-1)$$

所谓“非对称”是指 $\mathbf{p}$ 使用的是其PQ编码 $Q(\mathbf{p}) = (I_1, \dots, I_{\mathcal{M}})$ , 而 $\mathbf{q}$ 不用被量化。

在实际使用时, 可以通过查距离表的方式快速计算 $\mathbf{q}$ 与大量PQ编码之间的AQD。首先创建一个距离表, 该表存储 $\mathbf{q}$ 的每一个分量 $\mathbf{u}_i(\mathbf{q})$ 到该子空间中所有子码字的距离的平方。如果要计算 $\mathbf{q}$ 到数据点 $\mathbf{p}$ 的AQD, 只需要根据 $\mathbf{p}$ 的PQ编码到距离表中查表求和即可。图5.1描绘了一个AQD计算的查表过程, 其中 $\mathcal{M} = 8, k^* = 256$ 生成64比特的PQ代码。由于只涉及查表和加法运算, AQD的计算十分快速。

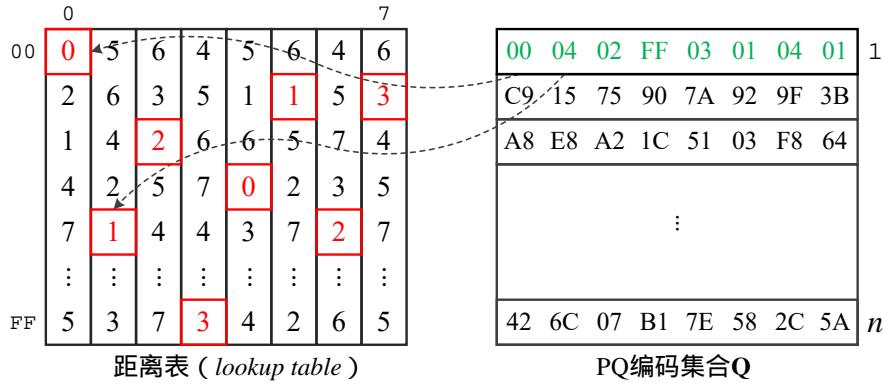


图 5.1 通过查距离表计算AQD

### 5.3 SC-LSH

#### 5.3.1 方法概述

概括地说, SC-LSH的主要思路是通过加强外存ANN搜索后两个环节 (即候选数据加载和近邻精炼) 的效率来提升搜索性能。其关键是找到一种比原始点更好的候选数据。按照之前在第5.1节概述部分的分析, 一个好的候选数据应该满足三方面特性: 能够被高效存储和加载、距离计算快、精度足以鉴别最近邻。PQ编码能够比原始数据更好地满足这三方面需求。

首先, PQ编码有很高的相似度保持能力。同标量量化和矢量量化一样, PQ技术也是一种量化技术, 其相似度保持能力的高低主要取决于量化精度。PQ技术可以看作是矢量量化技术的发展。本文第三章介绍过矢量量化技术, 一般而言, 对于矢量量化, 量化中心越多, 量化精度越高。PQ能够产生比传统矢量量化多得多的量化中心。根据文献<sup>[62, 63]</sup>在基于内存的近邻搜索评估实验, 通常64比特长的PQ编码可以在

大多数典型高维数据集上提供足够的精度区分最近邻。64比特长PQ编码的常用配置是 $\mathcal{M} = 8, k^* = 256$ ，如此，一共可以产生 $256^8 = 2^{64}$ 个量化中心，远远超过传统矢量量化技术能够支持的量化中心规模。这种规模也远大于目前实际特征数据集的规模，因此能够对原始数据进行充分量化。类似的情况在VA-file的介绍中也出现过。VA-file基于标量量化产生了远大于数据集规模的量化胞腔，能够获得紧致的距离下界，高效过滤无关数据。

高空间效率和计算效率比较容易理解。如上一节所述，每个PQ编码的物理尺寸是 $s = \mathcal{U}\mathcal{M}/32$ 个机器字，64比特长的PQ编码 $s = 2$ 。而AQD的计算只有 $\mathcal{M}$ 次查表和 $\mathcal{M} - 1$ 次求和操作， $\mathcal{M}$ 是划分的子空间个数，通常是比 $d$ 很小的值。

基于以上分析，SC-LSH选择使用PQ编码代替原始数据作为候选数据进行存储和精炼。假设一个磁盘页面的容量是 $B$ 个机器字，替换之后，一个磁盘页面将能够容纳 $V$ 个PQ编码：

$$V = \lfloor B/s \rfloor \quad (5-2)$$

由于 $s \ll d$ ，这意味着磁盘页面能够容纳的候选数据数量大大提升。还以128维SIFT特征为例，一个4KB大小的磁盘页面最多能容纳8个原始特征点，如果使用64比特长的PQ编码，可以容纳512个。

不过此时SC-LSH还不具有很好的候选数据加载效率。因为PQ编码默认是按照数据集原始的顺序存放，这种情况下，查询点的近邻候选编码一般随机分布在磁盘中。SC-LSH的下一步是使用O2LSH中的排序方法对PQ编码重新排列。具体做法为：先引入空间填充曲线对复合哈希键值排序，按照新的顺序重新组织PQ编码。如此可以让相似数据的PQ编码聚集在磁盘局部，从而可以用顺序I/O加载。而且，受益于PQ编码较小的物理尺寸，SC-LSH最终能在每次I/O操作中加载到更多近邻候选数据，从而整体上降低需要的I/O操作数量，进一步提升ANN搜索的I/O效率。

### 5.3.2 有效性分析

为更加清楚地说明SC-LSH对I/O效率的提升，结合图5.2对不同方法的候选数据加载方式进行对比分析。

图5.2一共描述了三种近邻候选数据加载方式。在每种方式的描述中使用一个有向线条代表磁盘存储顺序，因为磁盘在逻辑上可抽象为一个一维存储设备。每个粗边框大矩形代表一个磁盘页面，也是磁盘上的标准读/写单元。图5.2(a)和图5.2(b)中的小矩形代表一个原始数据点。图5.2(c)中的小正方形代表尺寸更小的PQ编码。

图5.2(a)描述了现有大多数外存LSH方法（包括LSB-forest<sup>[36]</sup>、C2LSH<sup>[37]</sup>、QALSH<sup>[39]</sup>以及SRS<sup>[38]</sup>等）在ANN搜索过程中的候选点加载方式，可以称之为随机加载。因为这些方法确立的候选点一般随机分布在磁盘上，只能通过随机I/O操作（即随机磁

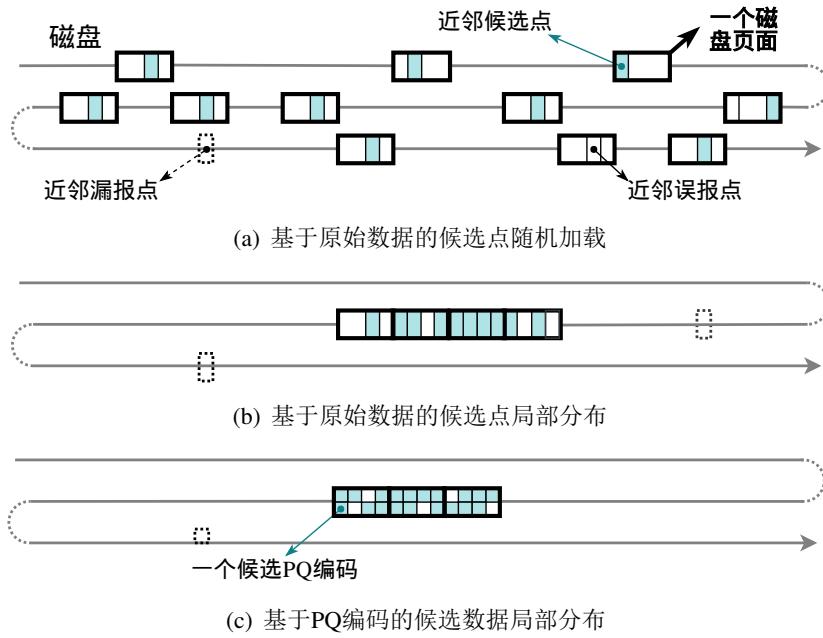


图 5.2 不同候选数据加载方式对比

盘页面读取) 来加载。缺点是比较明显的。一方面, 随机I/O是一种比较耗时的磁盘读取方式, 而算法为了保证ANN搜索的准确性, 通常需要加载足够多的近邻候选点, 这会使得候选点加载环节容易成为整个近邻搜索过程中的速度瓶颈。从另一个角度讲, 这种方式的I/O利用率也比较低, 因为一个磁盘页面通常能容纳多个原始数据, 而随机加载的方式平均一次I/O只能加载到一个近邻候选点。

基于排序的LSH方法通过引入空间填充曲线对候选点重新排列有助于解决上述随机I/O开销过多的问题。如图5.2(b)所示, 通过为复合哈希键值建立线序并指导原始数据重新排列, 能够让更多近邻候选点聚集在磁盘局部(即相同或相邻磁盘页面), 从而可以使用速度更快的顺序I/O操作加载近邻候选点。此外, 由于局部磁盘页面汇集了更多近邻候选点, 单次I/O操作能够加载到更多候选点, 总的I/O需求量也被降低。SK-LSH<sup>[40]</sup>和上一章研究的SC-LSH方法都采用了这一思路。区别在于, SC-LSH基于实验分析找到了比SK-LSH的row-wise曲线排序性能更好的曲线, 能够最大程度地改善近邻候选点的局部分布, 从而在相同I/O访问中命中更多最近邻。

然而, 原始高维数据较大的体量给图5.2(b)中这种方式的性能提升带来了严格的限制。如上所述, 每个磁盘页面最多可以容纳 $\lfloor B/d \rfloor$ 个数据点, 由于磁盘页面大小 $B$ 通常是一个常数, 随着维度的提升, 一个磁盘页面能够容纳的近邻候选点越来越少, 为了加载到足够多的近邻候选点保证近邻搜索精度, 需要访问更多的磁盘页面。

SC-LSH使用空间更加高效的PQ编码作为近邻候选数据能够克服这一局限。正如图5.2(b)与图5.2(c)中图示的对比, 在SC-LSH的索引结构中, 每个磁盘页面现在可

以容纳的候选数据数量大大提升，因为PQ编码的物理尺寸 $s$ 一般远远小于 $d$ 。因此，SC-LSH可以用更少的I/O操作加载到足够多的候选数据，从而进一步提升I/O效率。

### 5.3.3 构建索引结构

给定数据集 $\mathbf{D} \subset \mathcal{R}^d$ ，SC-LSH的索引构造主要包括两个阶段：数据预处理阶段和哈希表构建阶段，图5.3(a)和图5.3(b)分别展示了两个阶段的直观过程。

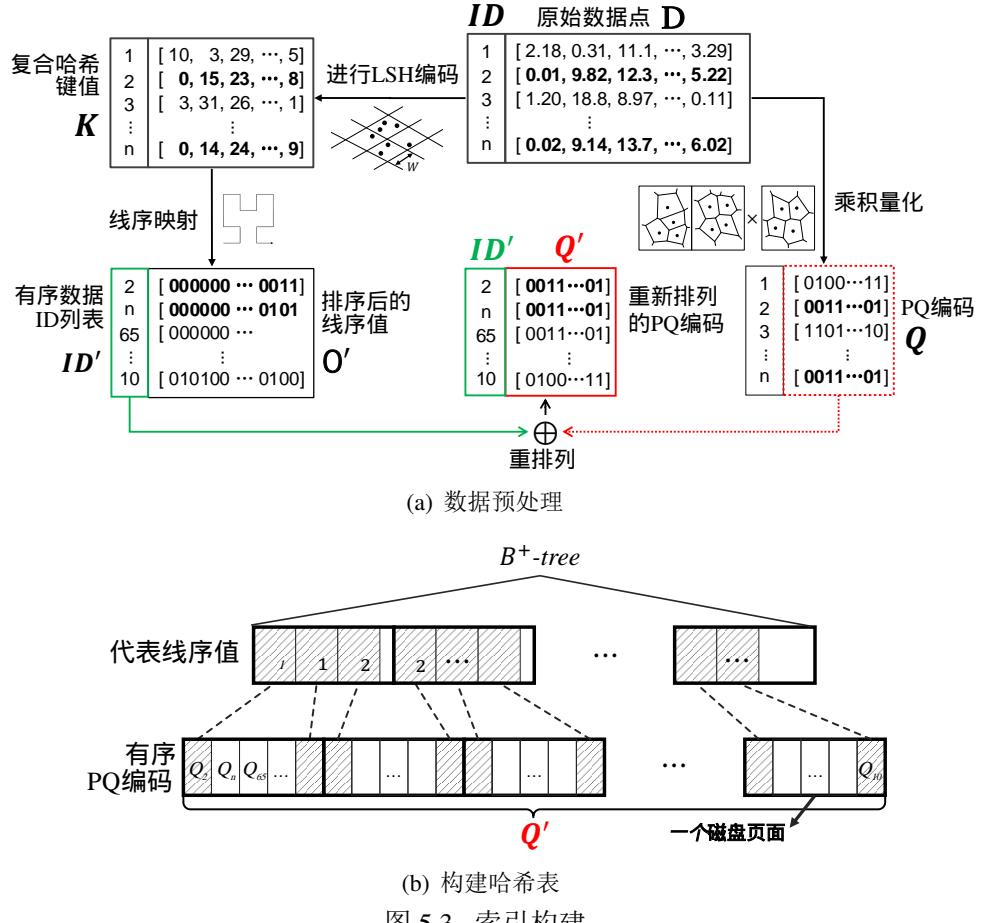


图 5.3 索引构建

#### 5.3.3.1 数据预处理

数据预处理分为四步。

- (1) 建立一个包含 $m$ 个LSH函数的复合哈希函数 $\mathbf{G}$ ，将数据集 $\mathbf{D}$ 在 $\mathbf{G}$ 上投影得到一组复合哈希键值 $\mathbf{K}$ ，见图5.3(a)的左上部分；
- (2) 按照O2LSH定义的线序对 $\mathbf{K}$ 进行映射，得到一组线序值 $\mathbf{O}$ ，将线序值升序排列，得到一组有序的线序值 $\mathbf{O}'$ 以及新的数据顺序 $\mathbf{ID}'$ ，见图5.3(a)的左侧部分。其中 $\mathbf{ID} = \{1, 2, \dots, n\}$ 是数据集中每个特征向量的原始序号；
- (3) 构建一个PQ量化器 $\pi_{k^*}^{\mathcal{M}}$ ，将 $\mathbf{D}$ 中的所有原始量化为一组PQ编码 $\mathbf{Q}$ ，见图5.3(a)的右侧部分；

(4) 最后, 按照**ID'**中的顺序对**Q**重新排列, 得到排序后的PQ编码集合**Q'**, 见图5.3(a)的下侧。

### 5.3.3.2 构建哈希表

将所有排序后的PQ编码**Q'**连续地存储磁盘上, 并建立一个B+对其进行索引, 构成SC-LSH的一个哈希表, 如图5.3(b)所示。

B+树构建的过程如下。对每一个存储PQ编码的磁盘页面, 选取第一个和最后一个PQ编码的线序值(分别用 $\alpha, \beta$ 表示)作为该页面的代表线序值。将所有代表线序值汇总为一个集合**O<sup>R</sup>**中, 并为其建立一个B+树, 将所有叶子层的线序值指向其来自的磁盘页面, 从而索引所有PQ编码。

SC-LSH多次执行以上两个阶段, 一共建立L个哈希表索引。因此, SC-LSH的索引结构一共包含L个B+树, L组数据ID列表和L组排序后的PQ编码。需要注意的是, 原始的数据点及其复合哈希键值只出现在索引构建的中间过程, 最终的索引结构并不需要维护这些数据。

### 5.3.4 最近邻搜索算法

SC-LSH的k-近似最近邻搜索(k-ANN)过程同O2LSH的过程很类似, 见算法1。为便于控制搜索性能, 算法的终止条件设置为允许加载的PQ编码页面数量 $N_P$ 。搜索过程大体可分为两个阶段: 1) 候选页面确定阶段; 2) 候选数据加载与最近邻验证阶段。为了能够在第一阶段定位到“质量”最好的 $N_P$ 个起始页面, 需要定义查询点 $q$ 到一个数据页面的距离, 此处可以使用O2LSH中对应的定义4。因为这一距离只跟查询点以及磁盘页面的代表线序值有关, 跟磁盘页面内部存储的数据形式没有关系。

#### 算法1. k-ANN搜索

输入: $\{T_i\}_{i=1}^L, q, k, N_P, k^*, \mathcal{M}, \{\mathbf{C}_i\}_{i=1}^M$

输出: k-ANN

(1) (初始化) 初始化候选页面集合 $P_C = \emptyset$ ;

(2) (定位初始页面) 在每一个哈希表 $T_i$ 上定位到距离 $q$ 最近的页面, 记为 $T_i$ 的左页面 $P_{iL}$ , 其右侧页面记为 $T_i$ 的右页面 $P_{iR}$ ;

(3) (迭代确定候选页面) 计算出当前 $P_C$ 中距离 $q$ 最近的一个页面 $P^*$ 。若 $P^*$ 来自某哈希表 $T_i$ 的左页面 $P_{iL}$ , 则更新 $P_{iL}$ 指向 $P^*$ 的左侧页面; 否则, 更新 $P_{iR}$ 指向 $P^*$ 的右侧页面。重复此步骤 $N_P$ 次;

(4) (加载候选数据, 验证k最近邻) 对每个哈希表 $T_i$ , 沿磁盘顺序加载从 $P_{iL}$ 到 $P_{iR}$ 间的所有页面中的PQ编码, 计算 $q$ 与所有候选PQ编码的AQD, 将AQD最小的k个作为k-ANN结果返回。

### 5.3.5 复杂度分析

**空间复杂度.** 根据式5-2, 一个磁盘页面最多可以容纳 $V = \lfloor B/s \rfloor$ 个PQ编码。因此, 数据集全部PQ编码将占据 $\lceil n/V \rceil$ 的磁盘页面。每个磁盘页面选取两个代表线序值, 则总的代表线序值数量为:

$$R = 2\lceil n/V \rceil \approx 2ns/B \quad (5-3)$$

SC-LSH的空间开销主要包括三个部分: 1)  $L$ 组复合哈希函数, 占用 $Lmd$ 的空间; 2)  $L$ 组有序的数据ID列表, 占用 $nL$ 的空间; 3)  $L$ 个哈希表, 每个哈希表包含一组代表线序值(占用 $Rm$ 的空间)、一组PQ编码(占据 $ns$ 的空间)和一组PQ编码码字(占据 $dk^*$ 的空间)。因此, SC-LSH的总空间消耗为 $L(md + dk^* + n + ns(2m/B + 1))$ 。实际上 $2m \ll B$ 通常成立, 因此总的空间占用为 $O(L(md + dk^* + ns))$ 。

对比第四章第4.3.5节O2LSH的空间开销 $O(L(md + nd))$ , 可以看到SC-LSH不再存储原始数据, 但需要存储PQ编码和对应的一组PQ编码码字。在实际情况中,  $s \ll d, k^* \ll n$ , 使得后两种数据的空间开销远远小于原始数据集占据的空间。因此SC-LSH在空间开销上会显著低于O2LSH的空间开销。

**时间复杂度.** SC-LSH的时间开销主要包含两部分: 1) 在 $L$ 个哈希表中定位 $N_P$ 个候选页面并加载候选数据; 2) 在候选数据中精炼最近邻。在哈希表中定位候选页面的时间跟哈希表中B+树的高度 $E$ 有关。假设一个B+树的分支因子(也称序, order)  $b = \lfloor B/m \rfloor$ , 则B+树的高度为:

$$E = \log_b R \approx \log_{\frac{B}{m}} \frac{2ns}{B}. \quad (5-4)$$

一般 $B \gg m, B \gg s$ , 意味着B+树的分支因子很大, 同时B+树中要维护的代表哈希键值较少, 因此综合式5-3和式5-4, B+树的高度会很小。处理一个页面需要花费 $O(B)$ 的时间, 因此, SC-LSH第一部分的总时间成本为 $O(LBE + BN_P)$ 。

根据第四章第4.3.5节的分析, 在O2LSH中, 一个B+树的高度约为 $\log_{\frac{B}{m}} \frac{2ns}{B}$ 。对比式5-4可以看到, 两个方法的B+树的分支因子是相同的, 但SC-LSH总的代表哈希键值数量更少, 这意味着SC-LSH中B+树的高度比O2LSHB+树的高度更低, 所以这部分时间开销, SC-LSH更少一些。

第二阶段的时间开销共包含两部分, 1) 计算查询点到所有PQ编码子码字距离的平方构建距离表, 时间开销是 $O(dk^*)$ ; 2) 计算查询点与所有候选PQ编码的AQD。由于每个页面最多可容纳约 $B/s$ 个PQ编码,  $N_P$ 个磁盘页面意味着SC-LSH最多需要进行 $BN_P/s$ 次复杂度为 $O(\mathcal{M})$ 的距离查表操作。

对比O2LSH,  $N_P$ 个磁盘页面最多可以加载到约 $BN_P/d$ 个原始数据, 由于 $s \ll d$ , 因此O2LSH加载到的候选数据远远小于SC-LSH能够加载到的候选数据。但是O2LSH进

行近邻精炼时每一个候选数据计算的是复杂度为 $O(d)$ 的欧氏距离，一般 $\mathcal{M} \ll d$ 。因此，综合来看，很难比较O2LSH和SC-LSH在近邻精炼的时间开销，需要结合实际实验进行对比。

**I/O开销** I/O开销包含两部分。首先在 $L$ 个B+树上定位起始页面，然后加载 $N_P$ 个数据页面，因此，总的I/O开销是 $LE + N_P$ 。根据上文在时间复杂度中的分析，SC-LSH中B+树的高度要比O2LSH中B+树的高度低，因此在设置相同 $L$ 和 $N_P$ 的情况下，SC-LSH消耗的I/O比O2LSH要少。

实际实验中发现，SC-LSH能在更少的页面中加载到足够多高质量候选数据以提供与O2LSH相当甚至更好的查询精度，因此SC-LSH相比于O2LSH能够进一步节省I/O开销。

## 5.4 实验结果与分析

本节设计实验评估SC-LSH方法的有效性，并与现有最先进的LSH方案进行对比。

### 5.4.1 实验设置

实验选取了三个公开的真实世界多媒体特征数据集对SC-LSH的ANN搜索性能进行评估，各个数据集详细信息如下。

- **Audio**<sup>2</sup>. 包含54,387个192维的音频特征，从DARPA TIMIT音频语音数据库中提取得到。本节将 $B$ 设置为4KB。
- **Sift**<sup>3</sup>. 包含1,000,000个128维的图像SIFT特征。本节将 $B$ 设置为4KB。
- **Gist**<sup>4</sup>. 包含1,000,000个960维的GIST特征（全局图像特征）<sup>[106]</sup>。 $B$ 设置为16KB。

从以下四个指标衡量一个ANN搜索算法的性能。

- **ratio**. 用于衡量ANN搜索的准确率。给定查询点 $\mathbf{q}$ ，令 $\mathbf{o}_1^*, \mathbf{o}_2^*, \dots, \mathbf{o}_k^*$ 依次代表 $\mathbf{q}$ 的前 $k$ 个真实最近邻， $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_k$ 依次代表搜索算法返回的 $k$ 最近邻，则该查询点 $\mathbf{q}$ 的ratio的计算如下：

$$ratio(\mathbf{q}) = \frac{1}{k} \sum_{i=1}^k \frac{\|\mathbf{o}_i, \mathbf{q}\|}{\|\mathbf{o}_i^*, \mathbf{q}\|} \quad (5-5)$$

<sup>2</sup><http://www.cs.princeton.edu/cass/audio.tar.gz>

<sup>3</sup><http://corpus-texmex.irisa.fr/>

<sup>4</sup><http://corpus-texmex.irisa.fr/>

根据定义可知，ratio值越小，准确率越高。

- **ART(Average Response Time).** 指算法进行一次ANN搜索的平均耗时。
- **I/O开销.** 指算法在ANN搜索过程中访问的磁盘页面数量。
- **空间开销.** 指算法为支持ANN搜索需要维持的所有数据占用的空间。

对于每个数据集，随机抽取200个数据点作为查询点，取所有查询点的平均性能作为最终性能。

### 5.4.2 ANN搜索性能对比

将SC-LSH与现有最先进的LSH方案进行了ANN搜索性能对比，对比方法包括C2LSH<sup>[37]</sup>、SK-LSH<sup>[40]</sup>、SRS<sup>[38]</sup>、QALSH<sup>[39]</sup>以及上一章设计的O2LSH方法。

#### 5.4.2.1 参数设置

为了进行公平的比较，每个方法以取得最好的准确率和效率平衡（trade-off）作为标准设置参数，详细参数值见表5.1。

表 5.1 参数设置

数据集	C2LSH $c, C_t$	QALSH $c$	SRS $c, m$	SK-LSH $m, W, L$	O2LSH $m, W, L$	SC-LSH $m, W, L$
<b>Audio</b>	3, 1	3	4, 6	10, 3, 3	10, 0.01, 3	10, 0.01, 3
<b>Sift</b>	3, 1	3	4, 6	10, 1000, 3	10, 3, 3	10, 3, 3
<b>Gist</b>	3, 1	3	4, 6	10, 2.5, 3	10, 0.02, 3	10, 0.02, 3

C2LSH和QALSH两个算法十分相似，二者都使用动态碰撞计数机制确立候选点，且都分别有一个侧重效率的算法版本和一个侧重准确率的算法版本。实验比较时让这两个算法具有不同的性能侧重，让C2LSH更加侧重效率（设置参数 $C_t = 1$ ），而让QALSH更侧重精度；对于SRS，实验部署其SRS-12版本，同时按照<sup>[38]</sup>中的建议在所有数据集上令 $c = 4, m = 6$ ；对于三个基于排序的方法（即SK-LSH、O2LSH和SC-LSH），为便于比较，设置相同的 $m, L$ 值。对于桶宽 $W$ ，SK-LSH经过调参之后选取最优值，SC-LSH和O2LSH部署相同的曲线，可以根据第四章的算法2快速确定 $W$ 值。最终，实验通过变化待返回近邻个数 $k$ 以观察所有ANN搜索方法的性能变化，结果见图5.4、图5.5、图5.6和表5.3。

#### 5.4.2.2 准确率与I/O效率

结合图5.4和图5.5可以看到，本章提出的SC-LSH方法在所有参与评估的数据集上都同时取得了最佳的准确率和I/O效率，这证实了SC-LSH方法的有效性。一方面意味着SC-LSH可以通过很少的磁盘访问（比如，Gist上只访问26次磁盘页面）命中

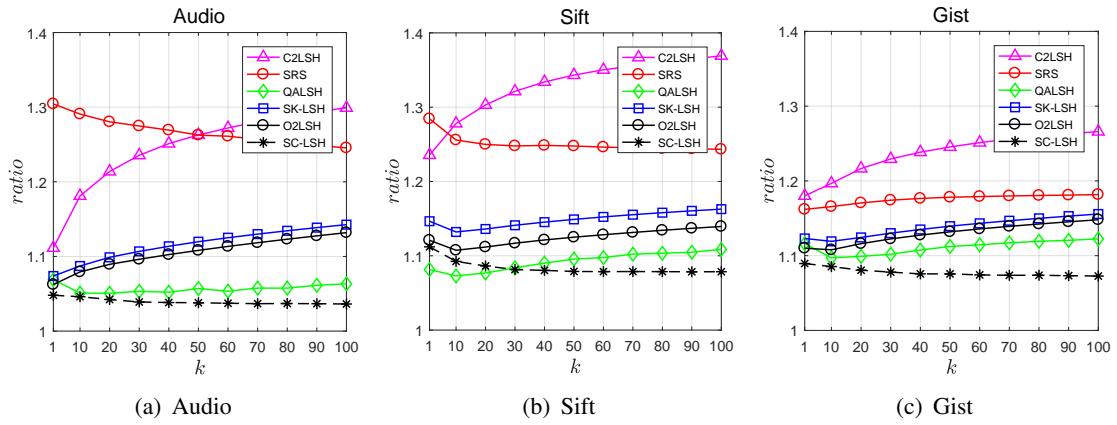


图 5.4 ratio的比较

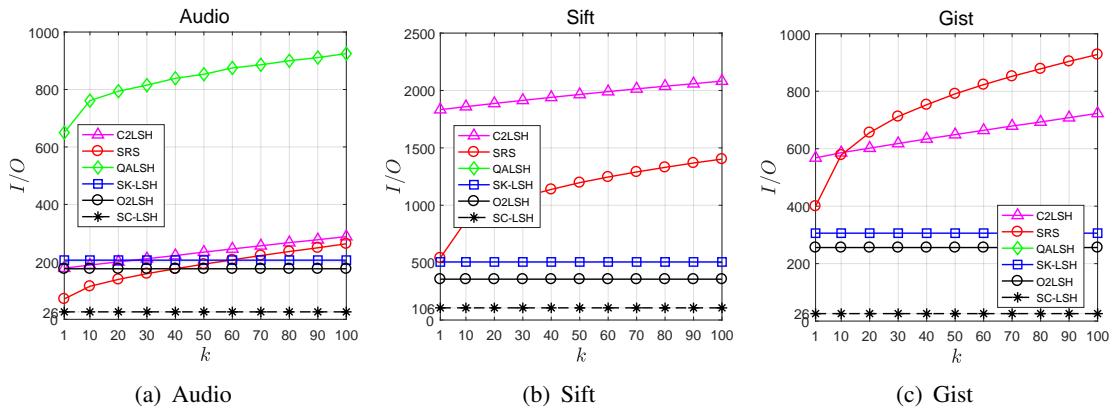


图 5.5 I/O开销的比较

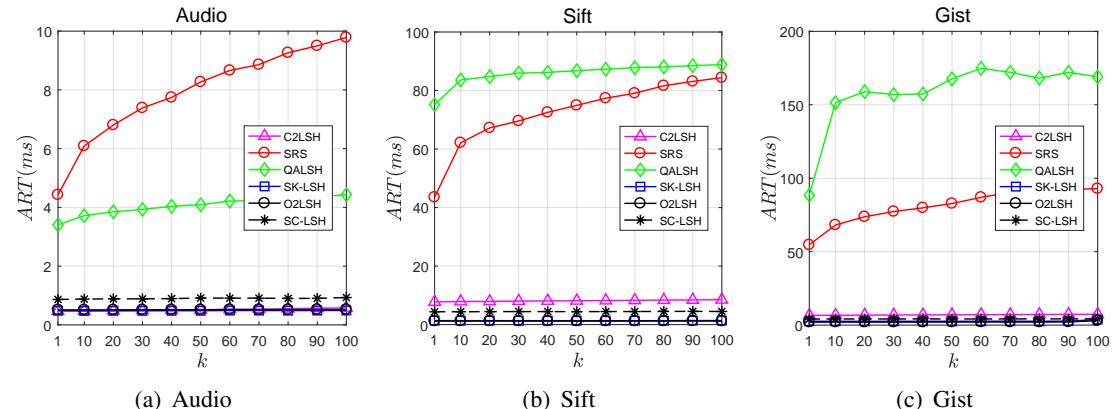


图 5.6 ART的比较

足够多的最近邻，另一方面也表明PQ编码的相似性保持能力能够胜任最近邻精炼的任务。

QALSH整体上取得了次优的精度。有时甚至比SC-LSH更好，比如，在Audio和Sift数据集上当 $k$ 较小时。这一现象的主要原因是PQ编码中存在的量化失真。不同于QALSH使用原始距离鉴别候选集中的最近邻，SC-LSH使用AQD。AQD是一种近似距离，相对于原始距离存在一定的误差，且这种误差对于越近的数据点影响越大，比如 $k$ 较

小时的最近邻；越远的数据点受误差的影响越小，比如 $k$ 较大时的最近邻。所以SC-LSH的ratio会呈现出随 $k$ 的增大而降低的趋势。

虽然QALSH取得了次优的搜索精度，但是消耗的I/O操作却是所有方法中最多的。由于QALSH在一些较大规模数据集（如Sift和Gist）上的I/O开销很大，本节没有在图5.5中显示其性能曲线，而是在表5.2中列出了它的具体开销，以更加清晰地展示。从表5.2可以看到，在相对较小规模的Audio数据集上，QALSH消耗的I/O分别是SC-LSH的25~35倍；而在两个较大规模的数据集Sift和Gist上，则分别增长到112~133和140~166倍。

O2LSH和SK-LSH整体上具有次优的I/O性能，并且在精度和I/O效率之间取得了较好的平衡。总体上，O2LSH比SK-LSH性能更好。但是，受原始数据的限制，二者在每次I/O访问期间命中的最近邻候选数据比SC-LSH要少得多，因此在准确率和I/O效率方面都落后于SC-LSH。具体地，SC-LSH在三个数据集上分别比SK-LSH节省了87%，78%和95%的I/O开销，分别比O2LSH节省了85%，70%和90%的I/O开销。

表 5.2 QALSH与SC-LSH的I/O开销比较

$k$	Audio			Sift			Gist		
	QALSH	SC-LSH	Q/S	QALSH	SC-LSH	Q/S	QALSH	SC-LSH	Q/S
1	650	26	25	11901	106	112	3659	26	140
10	761	26	29	13298	106	125	4052	26	155
40	839	26	32	13753	106	129	4192	26	161
70	886	26	34	13986	106	131	4268	26	164
100	925	26	35	14174	106	133	4332	26	166

#### 5.4.2.3 时间开销

从图5.6中可以看到，SC-LSH、O2LSH、SK-LSH和高效版本的C2LSH时间开销都很小。SC-LSH有时比O2LSH和SK-LSH的时间开销要多一些，比如Audio和Sift数据集。这是因为尽管单个AQD计算比单个原始欧氏距离计算更快速，但是SC-LSH计算的AQD次数要比O2LSH和SK-LSH的距离次数多很多。例如，在Audio数据集上，SK-LSH访问200个磁盘页面可以获得大约1,000个点，而SC-LSH访问20个磁盘页面能够加载到近10,000个PQ编码，Sift数据集上也是类似的原因。

#### 5.4.2.4 空间开销

表5.3和图5.7对比了各个方法的空间开销。其中表5.3列出了各个算法的具体空间开销值，并且注明了每个原始数据集占用的物理空间大小。图5.7是更为直观的对比，每个柱子的高度代表的是一个算法的空间开销与该数据集大小的比值。

可以看到，对于现有先进LSH方法来说，原始数据集的维护已成为整体空间开销的主体。其中，O2LSH和SK-LSH因为是多哈希表结构( $L=3$ )，需要维护多份原始数据集，空间开销最大。C2LSH、QALSH和SRS都只需要维护一份原始数据，所

以空间开销更小一些。其中，SRS由于使用的LSH投影函数最少，在所有使用原始数据作为候选数据的方法中具有最小的空间开销。SC-LSH由于存储更加节省物理空间的PQ编码代替原始数据，具有最小的空间消耗，且同其他方法具有显著差距。具体的，在三个数据集上，分别仅为SRS方法的9.5%，19.2%和2.8%。

表 5.3 空间开销比较（数值结果）

方法	Audio (39.3MB)	Sift (492.01MB)	Gist (3.58GB)
C2LSH	135MB	1.43GB	4.55GB
QALSH	49MB	635MB	3.96GB
SRS	42MB	537MB	3.71GB
SK-LSH	121MB	1.47GB	10.80GB
O2LSH	121MB	1.47GB	10.80GB
SC-LSH	<b>4MB</b>	<b>103MB</b>	<b>103MB</b>

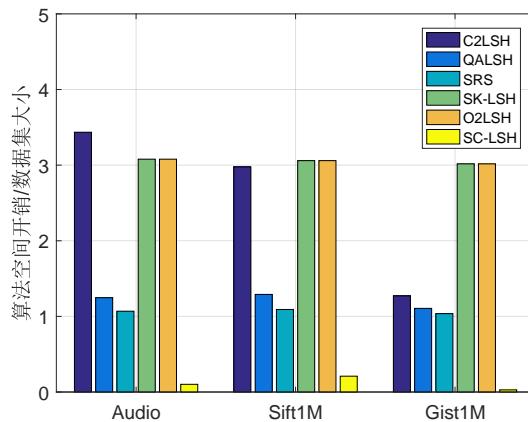


图 5.7 空间开销比较（可视化展示）

## 5.5 本章小结

受限于非结构化数据海量、高维特性的困难，外存ANN算法应该加强各个环节的有效性。本章首先考察了原始数据对外存ANN搜索各个环节的影响，发现原始数据并不是一个好的近邻候选数据选项。原始数据较高的维度，一方面会带来较大的存储和计算开销，另一方面也会阻碍I/O效率的提升。之后，本章分析了两类先进哈希技术，发现二者有很好的互补性。基于此，结合短判别码和最优排序方案设计了一种新的外存LSH索引SC-LSH。受益于短判别码较高的空间效率，一方面能大幅减少SC-LSH索引结构的空间消耗，另一方面能使得单个磁盘页面所容纳的候选数据数量显著提升，最终实现用更少的顺序I/O操作加载到足够多近邻候选数据。实验结果表明，与现有使用原始数据作为近邻候选数据的方法相比，SC-LSH能够同时取得最好的搜索精度和I/O效率，证实了本章方法的有效性，同时也表明短判别码能够提供足够高的鉴别精度胜任近邻精炼的任务。

## 第六章 一种数据集自适应的高维最远邻搜索策略

作为高维最近邻搜索的对偶问题，高维最远邻搜索提供了一个新的角度认识高维数据，同时也是很多实际应用以及其他重要问题的子问题，具有很高的研究价值。早期工作受最近邻搜索领域的启发提出了很多基于随机哈希的方法；近期工作发现最远邻点分布具有一些明显的规律，基于此提出了若干启发式方法，取得了更好的搜索性能。本章对最远邻的分布特性进行了更加深入的观察，发现最远邻的分布具有一种“距离聚集效应”，会严重限制哈希方法的搜索性能；而启发式方法所发现的规律只适用于最远邻点较少的数据集，实际数据集中最远邻点数量差异很大，对于最远邻较多的数据集，现有启发式方法取得好的搜索性能依旧比较困难。

本章提出一种数据集自适应的算法设计策略，主张根据不同数据集上最远邻的分布特点针对性地设计最远邻搜索算法。按照该思路，首先将数据集划分成三个最远邻搜索难度等级，然后针对不同难度的数据集分别改进了现有算法并提出两个新的搜索算法。最后，将这些最远邻搜索算法与一个数据集难度快速评估方法结合提出一种数据集自适应的最远邻搜索方案，能够为不同数据集分配最合适的数据集，进一步提升最远邻搜索性能。

### 6.1 概述

在高维相似性搜索领域，最远邻搜索也是一个十分重要的问题。精确的高维最远邻搜索具有非常大的难度<sup>[41, 76]</sup>，现有研究主要集中在高维近似最远邻（Approximate Furthest Neighbor，简称AFN）搜索问题。截至目前，研究者们共提出了四个较为实用的方法，从技术本质上可以分为基于哈希的方法和启发式方法两类。基于哈希的方法是数据无关（data independent）的，通过随机构建局部敏感哈希对数据进行投影，利用数据点与查询点投影哈希值间的差距作为最远邻候选点的判断依据。这一类方法主要包括QDAFN和RQALSH，二者的区别主要是利用投影差值的方式不同。其中，QDAFN选择将那些与查询点拥有最大投影差距的点优先作为最远邻候选点；RQALSH则通过统计查询点与数据点在所有哈希函数上的分离次数<sup>1</sup>，将分离次数多的点优先作为最远邻候选点。

启发式方法是数据依赖（data dependent）的，主要包括DrusillaSelect和RQALSH\*。它们都基于Curtin等人发现的一个最远邻分布规律。具体地，Curtin等人发现许多数据集中只有少部分点成为最远邻，而且这些点往往具有很大的模长<sup>2</sup>。基于该发

<sup>1</sup>给定哈希函数，若查询点与数据点没有被映射到相同哈希桶内，则称二者在该哈希映射中发生了“分离”。

<sup>2</sup>实际上是“中心模长”，即将数据集中心平移到质心之后对数据点计算的模长，为便于表述，后续将直接简

现, Curtin等人提出了第一个启发式的最远邻搜索算法DrusillaSelect: 在索引阶段, 按照某种方式选出少量模长较大的点构成一个公共最远邻候选集合, 搜索最远邻的时候, 只遍历这个候选集。DrusillaSelect的思路非常简单, 但是取得了很好的搜索效果。实验发现, 在Trevi数据集上搜索前10最远邻, 验证不到30个点能够取得1.0275的ratio准确率<sup>3[44]</sup>。Huang等人在DrusillaSelect的基础上进行了三点改进, 提出了一个新的启发式算法RQALSH\*。第一, 当最远邻候选集规模较大时, 为最远邻候选集建立一个RQALSH索引, 以避免线性扫描, 提升搜索效率; 第二, 取消了DrusillaSelect在选取候选点时设置的一个角度参数; 第三, 修改了最远邻候选点的评判标准。其中, 后两点的目的都是为了更好地将模长大的点选为最远邻候选点。

本章对现有方案的搜索性能进行了重新评估, 发现有方案在基于ratio准确率的评价标准下能够取得很好的表现, 不过在更加严格的基于precision准确率的性能评价标准下还有很多性能欠缺。一方面, 基于哈希方法的性能普遍低于启发式方法, 很难同时取得高搜索准确率和高速度。另一方面, 启发式方法的性能随数据集变动很大, 在一些数据集上能较“容易”地取得好的搜索性能, 一些数据集上则比较“困难”。

本章通过一些实验观察探究了以上问题的原因。首先, 发现最远邻的分布具有一种“距离聚集效应”是造成现有哈希方法性能局限的主要原因。然后, 发现不同数据集内最远邻数量存在很大差异, 与启发式方法的不同表现有很强的相关性: 在最远邻很少的数据集上, Curtin等人发现的规律能很好地适用, 现有启发式方法普遍也能取得很好的性能; 但是对于最远邻较多的数据集, 大模长与最远邻不再有很强的相关性, 现有方法也很难同时取得高搜索准确率和高速度。

综合以上现象, 本章提出一种数据集自适应的算法设计策略进一步提升最远邻搜索性能。该策略主张根据不同数据集内的最远邻分布特点针对性地设计最远邻搜索算法。按照该思路, 首先考察了一种能够快速评估数据集难度的方法, 将数据集分为容易、中等和困难三个等级。然后, 针对不同难度的数据集分别适应性地改进了现有方案并设计了两个新的最远邻搜索算法: 对于容易的数据集, 优化了现有启发式方法的调参方式, 使其更加实用; 对于居中难度数据集, 设计了一种多中心的最远邻搜索算法; 对于困难数据集, 设计了一种多中心与近邻图结合的搜索算法。最后, 将三个最远邻搜索算法与数据集难度快速评估方法结合构造出一个数据集自适应的最远邻搜索方案。在8个真实世界高维数据集上进行了评估, 结果表明, 该方案能为不同数据集匹配最合适的算法, 取得最好的最远邻搜索性能, 且具有很高的实用性。

---

写为模长。

<sup>3</sup>关于ratio的详细定义可见本章第6.4节。

本章余下内容组织如下：首先在第6.2节对现有方法重新评估，并分析现有方法性能欠缺的原因。之后在第6.3节提出一种数据集自适应的最远邻搜索方案。第6.4节设计实验评估新方案的性能。最后，在第6.5节总结本章工作。

## 6.2 现有方法的性能

最远邻搜索虽然是最近邻搜索的对偶问题，但相关研究表明，现有最远邻搜索方法普遍能取得很高的ratio准确率。例如，文献<sup>[44]</sup>显示现有最远邻搜索方法普遍能够取得低于1.04的ratio，这在最近邻搜索中是不太常见的现象，表明现有最远邻搜索研究已取得非常大的突破。不过，ratio并不是衡量相似性搜索准确率最严格的标准，为了更好地了解现有方法的水平，本节在一个更加严格的标准下重新评估了现有最远邻搜索方法的性能。

### 6.2.1 重新评估现有方法

新的评价标准是“precision-speedup”，详细定义可见第6.4节。其中，precision是一种比ratio更加严格的相似性搜索准确率衡量指标。不同于ratio基于距离评估搜索结果的准确率，precision衡量的是搜索结果中找到真实结果的比例，因此比ratio更接近精确搜索的要求。speedup是指待评价算法的搜索速度相对于线性扫描整个数据集的加速比。需要说明的是，由于最远邻搜索整体上比最近邻搜索开销更小，本章的实验评估主要面向内存环境，重点考察现有方法搜索精度和运行速度两个方面的性能。

图6.1显示了在8个高维数据集上的评估结果，数据集详细信息可见第6.4节。在进行分析之前有两点需要说明。第一，实验中将RQALSH\*方法分为两个版本进行评估，一种是全部遍历候选集的版本，称之为RQALSH\*(All)；一种是为候选集建立索引结构的版本，称之为RQALSH\*(Index)。这么做是为了将哈希方法和启发式方法充分地分开进行比较，因为RQALSH\*(All)是一个纯粹启发式的方法，RQALSH\*(Index)则是启发式与哈希技术混合的方法。第二，对于启发式方法，图中只展示了RQALSH\*(All)方法的性能，没有展示DrusillaSelect方法的性能，这是因为二者都需要通过调参确立最优性能。而本章通过调参观察，发现两个方法的最优性能都出现在 $M = 1$ 时，这种情况下二者实际上成为了同一个方法，即直接选取模长最大的一些点作为最远邻候选点（详细见本章第6.3.2节的研究）。

在precision-speedup性能图中，越靠近右上角表示算法性能越好。对于图6.1中的评估结果，有三点值得注意。第一，哈希方法（包括QDAFN和RQALSH）的性能普遍低于启发式方法，很难同时取得高搜索准确率和高搜索速度，而且哈希方法的性能曲线经常有较大波动；第二，启发式方法在不同数据集上的性能有明显差异，

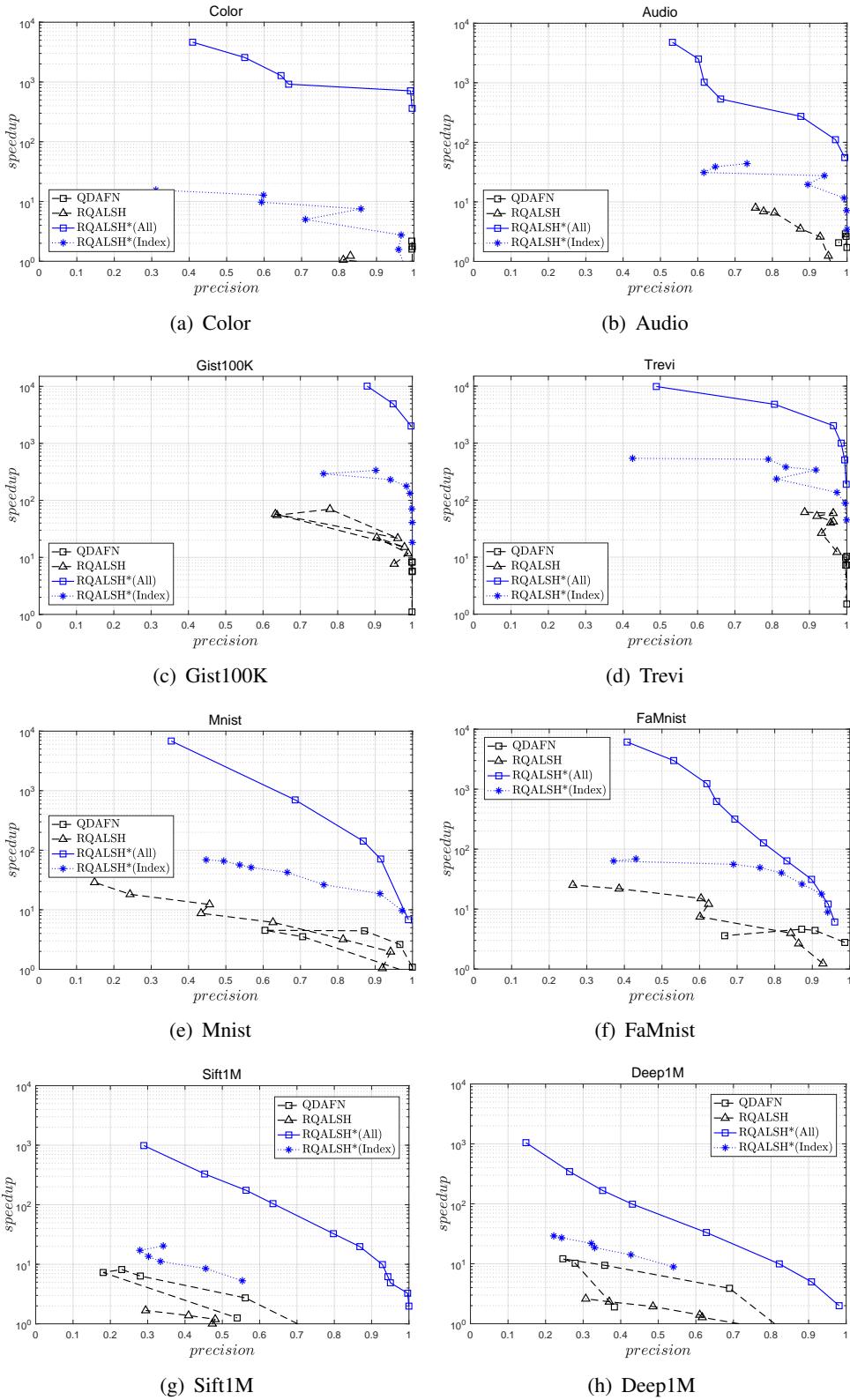


图 6.1 新的评估标准下现有方法的性能

一些数据集上较“容易”取得好的准确率和速度，一些数据集似乎比较“困难”。如果以取得90%左右准确率时的加速比划分，大致可以分为三种表现：1) 在Color、

Gist100K和Trevi数据集上，现有启发式方法加速比能达到1000左右或更高；在Mnist和FaMnist数据集上，加速比降低到30~100之间；在Sift1M和Deep1M数据集上速度最慢，加速比只有5~10左右。第三，没有部署哈希索引的RQALSH\*(All)方法比有索引的RQALSH\*(Index)性能更好。

## 6.2.2 现有方法性能欠缺的原因

上一节的评估表明现有最远邻搜索方法在更严格的评估标准下性能还有一定欠缺。本节通过一些实验观察探究这些性能欠缺的原因。

### 6.2.2.1 远邻的“距离聚集效应”

首先，设计了一个距离分布观察实验观察最远邻的分布。具体地，给定数据集 $D$ ，计算并比较两种距离比，一种称之为“最远邻距离比” $fnratio_k$ ，一种称之为“最近邻距离比” $nnratio_k$ 。最远邻距离比定义如下：

$$fnratio_k = \frac{\|\mathbf{x}, \mathbf{fn}_1\|}{\|\mathbf{x}, \mathbf{fn}_k\|} \quad (6-1)$$

这里， $\mathbf{fn}_1, \mathbf{fn}_k$ 分别代表数据点 $\mathbf{x}$ 在 $D$ 中的最远邻和第 $k$ 最远邻。最远邻距离比反映的是 $\mathbf{x}$ 到其第 $k$ 最远邻的距离与 $\mathbf{x}$ 到其最远邻的距离间的差距。最近邻距离比的定义如下：

$$nnratio_k = \frac{\|\mathbf{x}, \mathbf{nn}_k\|}{\|\mathbf{x}, \mathbf{nn}_1\|} \quad (6-2)$$

其中， $\mathbf{nn}_1, \mathbf{nn}_k$ 分别表示数据点 $\mathbf{x}$ 在 $D$ 中的最近邻和第 $k$ 最近邻。类似地，最近邻距离比反映的是 $\mathbf{x}$ 到其第 $k$ 最近邻的距离与 $\mathbf{x}$ 到其最近邻的距离间的差距。

实验具体实施如下：首先在数据集 $D$ 上执行全远邻搜索，即将 $D$ 中的每一个数据点作为查询点在 $D$ 中搜索最远邻，观察所有点前1000最远邻与其最远邻的距离比，对最近邻重复以上过程。实验在8个数据集上都进行了观察，所有数据集呈现出相似的现象，图6.2选取其中4个数据集作为代表呈现结果。

在图6.2中，每个数据集有两个距离比分布图，左侧是最近邻距离比的分布，右侧是最远邻距离比的分布。可以看到，所有数据集上，数据点的最远邻距离比比最近邻距离比更小。以Mnist数据集为例， $k = 100$ 时最远邻距离比的中位值约为1.1，最大值约为1.2；同样是 $k = 100$ ，最近邻距离比的中位值超过1.4，并且有很多超过1.8。这一现象说明，相比于 $k$ -最近邻的分布，数据点 $x$ 的 $k$ -最远邻中有更多点到 $x$ 的距离非常接近 $x$ 到其最远邻的距离。可以称这种现象为最远邻的“距离聚集效应”（简称“聚集效应”）。

“聚集效应”可以解释哈希方法为什么存在性能欠缺：现有哈希方法本质上是一种基于概率的方法，概率是基于数据点间的距离计算的。 $k$ -最远邻在距离上更靠

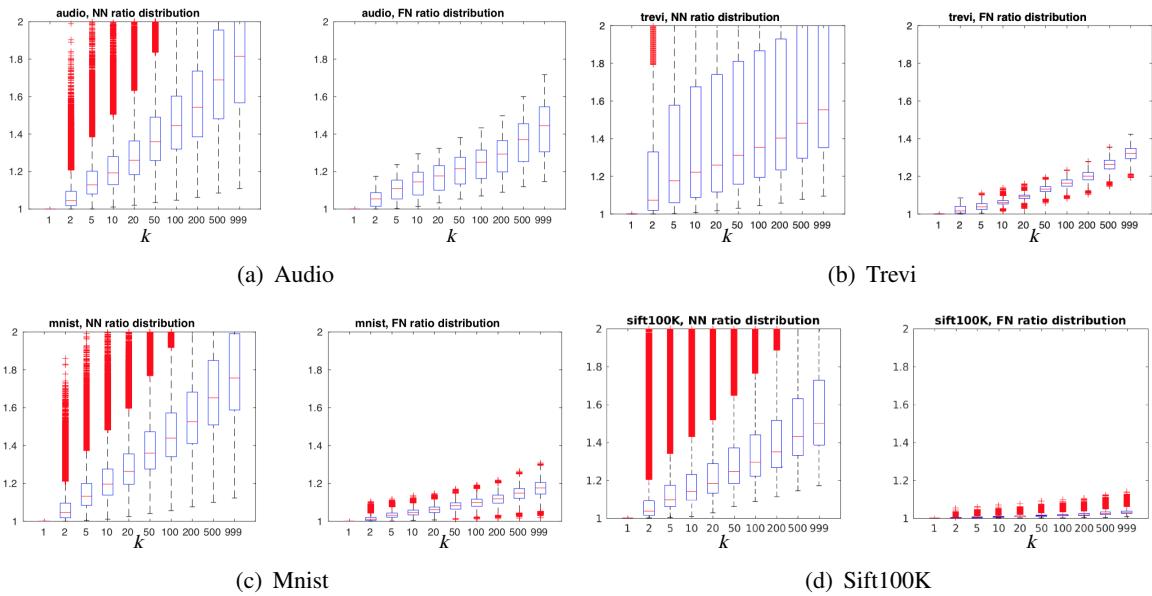


图 6.2 最近邻与最远邻距离比

近最远邻，会增大哈希方法辨别真实最远邻的难度。要想保证足够的辨别能力，通常需要增加哈希函数，其后果就是计算开销增大。

#### 6.2.2.2 数据集内部最远邻数量的差异

基于上一节的全远邻搜索，我们进一步统计了数据集内部最远邻的数量，统计结果见表6.1。

表 6.1 数据集内的最远邻数量

数据集	$n$	最近邻数量	最近邻占数据集比例(%)
Color	67040	9	0.13
Audio	53387	46	0.86
Gist100K	100000	15	0.15
Trevi	99000	42	0.42
FaMnist	60000	181	3.02
Mnist	69000	948	13.74
sift100K	100000	2485	24.85
deep100K	100000	12997	129.97

表6.1中,  $n$ 表示一个数据集成员点的总数量。从表中可以看到, 不同数据集内的最远邻数量存在很大差异。有些数据集内只有很少的数据点成为最远邻, 如Color, Audio, Gist100K和Trevi, 最远邻点占整个数据集的比例不到1%。有些数据集则比较多, 如Mnist, FaMnist, Sift100K和Deep100K。并且, 结合图6.1我们发现, 现有启发式方法的最远邻搜索性能与数据集内最远邻数量有很强的相关性: 在最远邻很少的数据集上, 启发式算法的性能都非常好, 最远邻多的数据集上最远邻搜索性能则随之下降。

经过分析，我们发现这一现象跟现有启发式方法的机理有关。现有启发式方法

选择为数据集提取一个最小的最远邻候选集，查询时通过遍历这一候选集搜索最远邻。当数据集内最远邻很少时，这种方法会很高效。但当数据集内存在很多最远邻点时，会出现效率和精度间的矛盾：如果仍旧维持一个最小的候选集，就无法有效覆盖全部最远邻点，准确率下降；如果扩大候选集的规模，容纳更多最远邻点，那么线序遍历候选集的开销就随之增大。最严重的，比如Deep100K数据集中有超过12,000个最远邻点，全部遍历会产生很大的时间开销。

对于候选集规模较大造成的问题，Huang等人给出的方案为候选集建立索引。具体地，他们设置了一个阈值，当候选集规模超过该阈值时，算法为整个候选集建立一个RQALSH索引，这一方法即上文提到的RQALSH\*(Index)方法。我们在图6.1中也评估了这个方法，可以看到RQALSH\*(Index)的性能并没有超过直接全部遍历候选集的RQALSH\*(All)方法。经过分析，我们认为这和第6.2.2.1节提到的聚集效应有关。因为现有启发式方法的基本逻辑是选择模长大的数据点作为最远邻候选点，一定程度上能够降低待索引数据的规模，但这种方式并不能够针对性地克服聚集效应。因此，RQALSH方法要想在候选集中充分过滤无关数据依旧需要付出很大代价。

总结现有方法的表现，主要有两个特点。第一，哈希方法的最远邻搜索性能受聚集效应限制很大。第二，启发式方法的性能在不同数据集上有很大差异，数据集内最远邻点的数量能够反映远邻搜索的难易：在最远邻较少的数据集上很容易能达到好的性能，但是在一些最远邻较多的数据集上，性能有明显下降，最远邻搜索变得困难。

### 6.3 数据集自适应的最远邻搜索方案

本节提出一种数据集自适应的算法设计策略进一步提升不同数据集上的最远邻搜索性能。该策略主张根据不同数据集上最远邻的分布特点针对性地设计最远邻搜索算法。按照该思路，首先考察了一种数据集难度量化评估方法，将数据集分为容易、中等和困难三个难度等级。然后，针对不同难度的数据集，分别改进了现有算法并提出了两个新的算法进行最远邻搜索。

对于容易等级的数据集，发现现有启发式方法（包括DrusillaSelect和RQALSH\*）能够取得非常好的效果，但存在调参难的问题，本节提出一种更加实用的调参方式，详细见第6.3.2节；对于中等难度的数据集，通过实验观察发现一种“邻近点共享远邻”现象，基于此提出一种多中心的最远邻搜索算法，见第6.3.3节；对于困难的数据集，设计了一种近邻图与多中心方法相结合的最远邻搜索算法，详细见第6.3.4节。最后，将数据集难度估计方法与三种最远邻搜索算法结合，提出一种数据集自适应的最远邻搜索方案。在实际应用中，先自动评估数据集的难度，根据评估结果部署合适的最远邻搜索算法，可以在不同数据集上取得最好的最远邻搜索性能。

### 6.3.1 数据集难度评估

关于数据集存在不同最远邻搜索难度的现象，我们首先在第6.2.1节的性能评估中得到了直观的认识，然后在第6.2.2节的全远邻搜索实验中发现这种难度差异跟数据集内部最远邻点数量有很大联系。实际上，Curtin等人也研究过数据集的难度问题，他们从概率的角度给出了一种解释<sup>[28]</sup>。具体地，Curtin等人指出，如果数据集中只有少量点成为其它点的最远邻，意味着最远邻比较容易预测，所以比较容易搜索；反之，如果数据集中有很多点成为最远邻，意味着最远邻难以预测，所以最远邻搜索比较困难。进一步地，他们发现这种最远邻数量与可预测性的对应关系跟信息熵的概念十分类似，因此，他们提出了一种基于信息熵理论评估数据集最远邻搜索难度的方法。

具体地，给定查询集 $Q$ 和数据集 $D$ ，首先在 $D$ 上搜索 $Q$ 中所有查询点的最远邻，得到最远邻集合 $FN(Q)$ ，然后计算 $FN(Q)$ 中每一个最远邻点 $\mathbf{x}$ 在 $FN(Q)$ 中出现的频率：

$$Pr(\mathbf{x}) = \frac{count(\mathbf{x}, FN(Q))}{|Q|} \quad (6-3)$$

将这些频率代入信息熵公式计算出如下系数：

$$hardness(Q, D) = \sum_{\mathbf{x} \in FN(Q)} -Pr(\mathbf{x}) \log_2(Pr(\mathbf{x})) \quad (6-4)$$

Curtin等人使用这一系数评估 $Q$ 在 $D$ 上搜索最远邻的难度： $hardness(Q, D)$ 值越小，表示最远邻搜索难度越小； $hardness(Q, D)$ 值越大，表示最远邻搜索难度越大。可以通过如下方式进行理解。在数据集 $D$ 上搜索 $Q$ 的最远邻，最容易的情况是所有查询点的最远邻都相同（用 $\mathbf{x}^*$ 表示），此时 $Pr(\mathbf{x}^*) = 1$ ，难度系数为0；最难的情况是所有查询点的最远邻都不相同，这种情况下，每个查询点的最远邻都很难预测，此时 $FN(Q)$ 中共有 $|Q|$ 个最远邻点，难度系数为 $\log_2 |Q|$ ，是最大的难度系数。

在Curtin等人的模型中，最远邻搜索的难度和查询集 $Q$ 、数据集 $D$ 的分布都有关系，而二者有可能来自不同的分布，因此他们的难度系数拥有 $Q$ 和 $D$ 两个参数。这是一个泛化能力很强的模型，但是缺点也很明显：只能用于事后解释，无法用于预测，因为只有在查询集确定之后才能计算。

我们调整了Curtin等人的假设，通过牺牲一定的泛化能力，获得一个能够在事前预测远邻搜索难度的方法。具体地，我们假定查询点一般来自和数据集相同或类似的分布。因此，可以在查询集到来之前先从数据集中随机选取一个子集模拟查询集，代入Curtin等人的公式评估数据集难度。

按照这种方法，我们对多个实际的高维特征集进行了评估<sup>4</sup>，结果见表6.2。我们

<sup>4</sup>数据集详细信息见第6.4节的介绍。

对比了两种随机选取点规模下的计算结果。一种从数据集中随机选取1000个模拟查询点，计算更快速。另一种，我们执行全远邻搜索，将整个数据集作为查询集计算难度系数。

表 6.2 远邻搜索难度系数评估

数据集	$n$	难度系数, 最远邻数量 (全远邻搜索)	难度系数, 最远邻数量 (随机抽样1000)	难度等级
Gist100K	100,000	1.424, 15	1.392, 9	容易
Color	67040	1.4857, 9	1.56, 6	
Trevi	99,000	1.7179, 42	1.656, 15	
Audio	53,387	1.7626, 46	1.736, 15	
FaMnist	60,000	3.185, 181	3.073, 44	中等
Mnist	69,000	4.8351, 948	4.714, 147	
Sift100K	100,000	8.2893, 2485	7.557, 344	困难
Sift1M	1,000,000	9.7249, 11466	8.371, 484	
Deep100K	100,000	11.4872, 12997	9.3086, 738	
Deep1M	1,000,000	13.5008, 86582	9.599, 861	

从评估结果中可以看出，两种方式计算出的难度系数相差并不大，都能比较清晰地将所有数据集划分成三个层次。具体地，Color, Audio, Gist和Trevi的难度系数比较接近，数值最小，在1~3之间；Mnist和FaMnist的难度系数比较接近，数值比上一组更大一些；Sift和Deep拥有最大的难度系数，而且与上一组拉开了明显的差距。可以看到，这一划分结果和第6.2.1节反映的数据集难度很吻合，说明这一系数能够比较准确地反映数据集的最远邻搜索难度。因此，我们选定这一方法作为数据集难度自动评估的方法，实际应用中随机抽取部分数据点作为查询点加快评估速度。具体地，结合表6.2，我们将数据集最远邻搜索的难度大致划分为三个等级：第一等级，难度系数0~3，作为容易级别；第二等级，难度系数3~6，作为中等（或比较困难）级别；第三等级，难度系数6以上，作为困难级别。

### 6.3.2 容易等级数据集上的最远邻搜索

对于容易等级的数据集，现有启发式方法性能已经足够好，我们的方案将继续沿用。从第6.2.1节的评估中可以看到，现有启发式方法在Color、Audio、Gist和Trevi上都能够同时取得高搜索精度和高搜索速度。另外，从第6.2.2.2节的全远邻观察中可以看到，这些数据集内部的最远邻点非常少，即使采用线序遍历候选集的方式也不会耗费太多时间。可以说，这些数据集上的最远邻搜索性能已经没有很大的提升空间。所以，对于容易的数据集，我们的方案将沿用现有启发式方法。具体地，根据第6.2.1节的比较，现有启发式方法性能最好的是RQALSH\*(All)这一直接遍历候选集的版本。

然后，包括RQALSH\*(All)在内的现有启发式方法存在调参耗时的问题。因为这些方法都包含两个参数：随机投影个数 $L$ 和每个随机投影上选出的候选点数量 $M$ ，但

是没有给出设置最优参数的指导原则。实际部署时只能通过网格搜索（grid search）的调参方式寻找最优参数值，即对两个参数分别设置一组独立的参数值，遍历所有组合寻找最佳性能。这是一种比较耗时的调参方式，比较影响算法的实用性。

我们对RQALSH\*(All)方法进行了大量的调参观察，发现在大多数情况下， $M = 1$ 就能达到最好的性能，这意味着RQALSH\*(All)可以只设置一个参数。

调参实验设计如下：给定数据集，实验改变候选点总量 $N_C$ ，对于每一个 $N_C$ 值，将其分解为一系列 $L$ 和 $M$ 的乘积，其中 $L$ 和 $M$ 的取值都在 $[1, N_C]$ 间变化。对每一个 $L, M$ 组合运行一次算法，实验观察在同等 $N_C$ 下取得最好性能（以准确率为主）时的 $L, M$ 参数数值。

表6.3、6.4、6.5和6.6分别展示了在多个容易等级数据集上的调参结果，数据集根据表6.2中的难度系数顺序排列。以表6.4展示的Color的结果为例，表格总体上分为左右两部分。左侧为 $M = 1$ 时的性能，其中加粗的参数表示在 $M = 1$ 时取得了当前 $N_C$ 数量下最好的准确率；未加粗的表示在 $M = 1$ 时没有取得最高的准确率（即例外），右侧会列出当前 $N_C$ 取得最高准确率时的参数。

可以看到，在所有数据集上最远邻搜索的准确率都随候选集规模的增大而提高。在同等候选集规模下， $M = 1$ 能够在大部分情况下取得最优准确率。即使有些情况 $M = 1$ 的性能不是最好，但是和最优性能的差距非常小（最大的准确率差距为1.48%，出现在Trevi数据集上 $N_C = 10$ 时）。此外， $M = 1$ 取得的最好准确率一般都是较高的准确率。综上，将 $M$ 设置为1可以取得近似最优的性能。因此，在我们的最远邻搜索方案中，对于容易的数据集，我们部署RQALSH\*(All)算法，并且只设置一个参数，即要选取的候选点总数量 $N_C$ ，这将大大减少算法的调参工作量。实际上，这意味着候选点选择过程简化为直接从数据集中选取 $N_C$ 个模长最大的点<sup>5</sup>。假设需要搜索 $k$ -近似最远邻，这一方法总的时间开销将是 $O(N_C \log k)$ 。

表 6.3 RQALSH\*(All)在Gist100K上的调参结果

$M = 1$ 时的准确率			最高准确率		
$L, M$	<i>precision</i>	<i>speedup</i>	$L, M$	<i>precision</i>	<i>speedup</i>
<b>10, 1</b>	<b>0.8790</b>	10110	-	-	-
<b>20, 1</b>	<b>0.9486</b>	4938	-	-	-
<b>50, 1</b>	<b>0.9967</b>	2021	-	-	-

### 6.3.3 中等难度数据集：基于远邻共享的多中心方法

#### 6.3.3.1 远邻共享现象

现有的启发式方法（包括DrusillaSelect和RQALSH\*）可以看作是一种基于“单中心”的方法：这些方法计算所有数据点的中心模长，并选择模长最大的点作为候

<sup>5</sup>这也符合DrusillaSelect到RQALSH\*的算法改进方向，见本文第二章第2.3.3节的分析。

表 6.4 RQALSH\*(All)在Color上的调参结果

$M = 1$ 时的准确率			最高准确率		
$L, M$	<i>precision</i>	<i>speedup</i>	$L, M$	<i>precision</i>	<i>speedup</i>
<b>10, 1</b>	<b>0.4091</b>	4627	-	-	-
<b>20, 1</b>	<b>0.5480</b>	2567	-	-	-
<b>50, 1</b>	<b>0.6451</b>	1278	<b>25, 2</b>	<b>0.6451</b>	1241
100, 1	0.9906	711	<b>50, 2</b>	<b>0.9943</b>	703
<b>200, 1</b>	<b>0.9951</b>	362	-	-	-

表 6.5 RQALSH\*(All)在Trevi上的调参结果

$M = 1$ 时的准确率			最高准确率		
$L, M$	<i>precision</i>	<i>speedup</i>	$L, M$	<i>precision</i>	<i>speedup</i>
10, 1	0.4893	9857	<b>5, 2</b>	<b>0.5041</b>	9946
20, 1	0.8061	4810	<b>10, 2</b>	<b>0.8073</b>	4988
50, 1	0.9641	2025	<b>25, 2</b>	<b>0.9685</b>	2026
<b>100, 1</b>	<b>0.9849</b>	1001	-	-	-
<b>200, 1</b>	<b>0.9943</b>	509	-	-	-
<b>500, 1</b>	<b>0.9984</b>	190	-	-	-

表 6.6 RQALSH\*(All)在Audio上的调参结果

$M = 1$ 时的准确率			最高准确率		
$L, M$	<i>precision</i>	<i>speedup</i>	$L, M$	<i>precision</i>	<i>speedup</i>
10, 1	0.5321	4780	<b>2, 5</b>	<b>0.5347</b>	4897
<b>20, 1</b>	<b>0.6019</b>	2505	-	-	-
50, 1	0.6173	1020	<b>2, 25</b>	<b>0.6311</b>	1037
100, 1	0.6614	533	<b>2, 50</b>	<b>0.6704</b>	525
<b>200, 1</b>	<b>0.8763</b>	273	-	-	-
<b>500, 1</b>	<b>0.969</b>	111	-	-	-
<b>1000, 1</b>	<b>0.9939</b>	55	-	-	-

选点。换一个角度看，这些候选点其实是数据集中心 $O$ 的最远邻。搜索时，算法遍历候选集寻找最远邻，相当于默认数据集中心 $O$ 会和查询点共享很多最远邻。在一些容易的数据集上，这种方法仅计算少量的候选点就能取得很高的准确率（见上一节的观察），说明， $O$ 和查询点的最远邻共享率很高， $O$ 可以很好地代表查询点。

但是随着数据集难度的增加，这一方法的性能有明显下降。我们在一些中等和困难数据集上也进行了上一节所述的调参实验，结果见表6.7。结果显示，在所有情况下， $M = 1$ 都能够取得同等候选集规模下的最好性能<sup>6</sup>。但是整体上在这些数据集上取得较好准确率时付出的效率代价都比较高。从表6.7可以看到，为了搜索查询点的前10最远邻，Mnist需要在候选集中维持1,000个点才能取得近91%的准确率，Sift100K上则需要维持5,000个才能取得89%的准确率。这表明以数据集中心 $O$ 为代表选取的最远邻候选点不再具有较好的代表性。换句话说，有些查询点的最远邻不一定具有很大的中心模长。

为了寻找具有更好代表能力的数据点，本节提出一个假设：在高维空间，两个数据点间的最远邻共享率与它们的距离有关。然后设计实验验证这一假设。具体地，

<sup>6</sup>由于没有例外情况，表6.7中将两个数据集放在一起呈现。

表 6.7 RQALSH\*(All)在中等和困难数据集上的调参结果

Mnist			Sift100K		
$L, M$	<i>precision</i>	<i>speedup</i>	$L, M$	<i>precision</i>	<i>speedup</i>
<b>10, 1</b>	<b>0.3537</b>	6823	<b>1000, 1</b>	<b>0.5736</b>	101
<b>100, 1</b>	<b>0.6857</b>	700	<b>2000, 1</b>	<b>0.7214</b>	52
<b>500, 1</b>	<b>0.8680</b>	142	<b>5000, 1</b>	<b>0.8901</b>	21
<b>1000, 1</b>	<b>0.9148</b>	71	<b>10000, 1</b>	<b>0.9589</b>	10
<b>10000, 1</b>	<b>0.989</b>	7	<b>20000, 1</b>	<b>0.9875</b>	5

给定数据集  $D$ , 首先随机选择  $R$  个点作为代表点。然后, 对于成员点  $\mathbf{x} \in D$ , 计算  $\mathbf{x}$  与所有代表点的距离, 并将代表点按照距离升序排列, 如此每个代表点会获得一个排序号  $i, 1 \leq i \leq R$ 。检查  $\mathbf{x}$  与每一个代表点是否拥有相同最远邻, 是计 1, 否计 0。一共从  $D$  中随机选择  $S$  个成员点重复上述过程, 统计所有样本点与其第  $i$  近代表点共享最远邻的比例。实验观察随着  $i$  的增大, 最远邻共享率的变化情况。

实验中  $R = 100, S = 10,000$ , 一共观察了 4 个数据集, 结果见图 6.3。可以看到, 数据点与代表点的最远邻共享率和它们之间的距离有关。距离越近, 共享率越大。比如 Mnist 数据集, 样本点与其最近代表点(即  $i=1$ )的平均共享率约为 50%, FaMnist 上平均高达 70%。可以称这一现象为“邻近点最远邻共享”现象(简称“远邻共享”现象)。

### 6.3.3.2 基于多中心的远邻搜索算法

基于远邻共享现象, 我们提出一种多中心的最远邻搜索方案 MultiCentroid 进一步提升更加困难数据集上的性能。索引构建与搜索算法的详细流程分别见算法 3 和算法 4。

索引构建的主要任务是生成一个远邻表: 给定数据集  $D$ , 首先, 按照某种方式为  $D$  生成  $K$  个代表点, 然后为每个代表点在  $D$  中搜索  $g$  个最远邻。将代表点及其最远邻保存为一个远邻表。代表点的生成方式可以有很多种, 我们推荐使用 KMeans 聚类算法。

---

#### Algorithm 3 MultiCentroid.Indexing

---

**Require:** 数据集  $D$ , 代表点数量  $K$ , 每个代表点需要维护的远邻数量  $g$

**Ensure:** 代表点集合  $C$ , 代表点的远邻表  $FNTable$

- 1:  $C \leftarrow \text{GenerateCentroids}(K)$ ; // 生成  $K$  个代表点
  - 2:  $FNlist = \phi$ ; // 初始化远邻列表
  - 3: // 为每一个代表点  $c_i \in C$  搜索  $g$  个最远邻点;
  - 4: **for**  $i = 1$  to  $K$  **do**
  - 5:      $FNTable[i] = \text{LinearFNSearch}(D, c_i, g)$ ;
  - 6: **end for**
  - 7: **return**  $C, FNTable$ ;
- 

最远邻搜索的主要过程是使用远邻表定位候选点: 给定查询点  $\mathbf{q}$ , 首先找到距离  $\mathbf{q}$  最近的  $w$  个代表点, 通过查远邻表将它们的最远邻合并并进行精炼, 将最远的  $k$  个

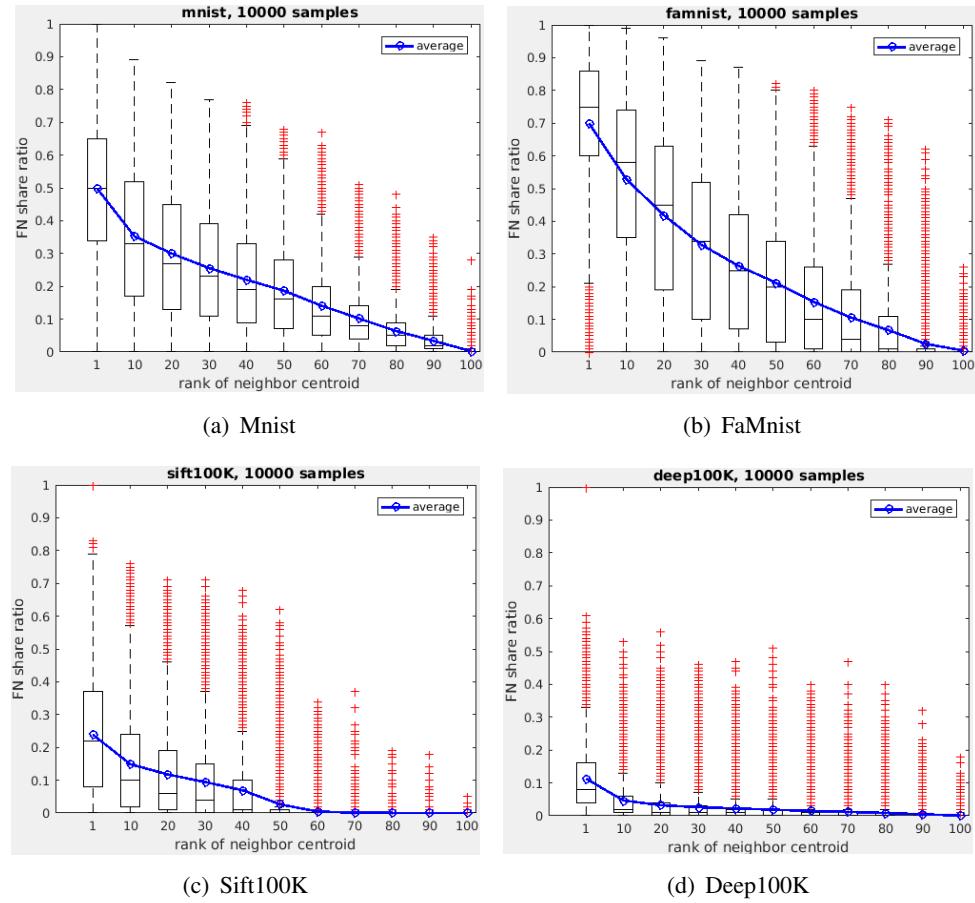


图 6.3 远邻共享率观察

点作为最远邻搜索结果返回。

---

**Algorithm 4** MultiCentroid.FNSearch

**Require:** 查询点 $q$ , 数据集 $D$ , 待搜索远邻个数 $k$ , 代表点集合 $C$ , 远邻表 $FNTable$

**Ensure:** 最远邻 $kFN[]$

- 1: 从 $C$ 中查找距离 $q$ 最近的 $w$ 个代表点;
  - 2: 合并全部 $w$ 个最近代表点的远邻列表, 将其中距离 $q$ 最远的 $k$ 个点作为搜索结果 $kFN[]$ ;
  - 3: **return**  $kFN[]$ ;
- 

可以看到, 算法设计的主要思路遵照上述的实验发现。根据图6.3, 由于最近的代表点具有最大的最远邻共享率, 因此搜索算法选择最近的几个代表点定位最远邻。另外, 图6.3也显示只使用最近的代表点大部分情况下无法100%涵盖全部最远邻, 所以搜索算法设置了 $w$ 参数, 通过访问多个最靠近的代表点提高对最远邻的覆盖率。假设需要搜索 $k$ -近似最远邻, MultiCentroid方法首先从 $K$ 个中心点中确定 $w$ 个距离查询点最近的代表点(算法4第1行), 然后从这些代表点的最远邻中搜索距离查询点最远的 $k$ 个点(算法4第2行), 因此总的时间开销是 $O(K \log w + wg \log k)$ .

### 6.3.4 困难数据集：基于近邻图的最远邻搜索方法

不幸的是，最远邻共享率的高低似乎也受数据集难度的影响。从图6.3中可以看到，困难数据集上的最远邻共享率普遍没有中等难度数据集上那么高。比如Sift和Deep数据集，数据点和最近的代表点（即 $i = 1$ ）之间分别平均只有24%和12%的共享率，这意味着仅仅依靠多中心的方法可能无法在这些数据集上达到非常高的搜索精度。对于这些非常困难的数据集，本节提出一种近邻图与多中心相结合的最远邻搜索算法。

近邻图（Nearest Neighbor Graph或者Proximity Graph）同哈希技术一样，也是最近邻搜索领域一个非常重要的技术方向。近邻图通过将每一个数据点与其近邻点连接，可以为原始数据集构造一种图结构。其好处是能够利用数据点间的邻近关系构建明确的搜索路径。在近邻图上进行最近邻搜索通常是由一些初始点出发反复执行以下过程：加载当前点的近邻列表，检查其中是否存在更加靠近查询点的点，如果是，则将当前点移动到最靠近查询点的近邻（称之为向最近邻收敛）；否则，停止搜索。

在近邻图上进行最近邻搜索一个非常大的困难是图的连通性问题。因为高维空间的复杂性，很难构造一个近邻图保证总是存在能够收敛到最近邻的路径，大部分情况下近邻图只能提供一个向最近邻收敛的路径片段，使搜索停在一个局部最优解<sup>7</sup>。为了弥补这一不足，一个常用的手段是优化初始搜索点：结合一种辅助方法快速定位一些比较靠近查询点的数据点，从这些点出发在图上搜索，可以缩短图上搜索路径的长度，降低停在局部最优解的可能性，提升最近邻搜索的成功率。

本质上，近邻图是一种能够提供收敛路径的数据结构，沿着这些路径可以实现逐步向目标点靠近。对于最近邻搜索，目标点是查询点，通过逐步向查询点靠近从而找到最近邻。如果调整收敛方向，近邻图也可以用于最远邻搜索。具体地，通过将上述过程中向“靠近”查询点的方向移动改为向“远离”查询点的方向移动，可以实现一步步向远邻收敛。

不过，连通性的问题依然存在，因为依旧是在高维空间搜索，对此，我们提出将近邻图与上一节的多中心方法相结合的思路。我们发现，基于多中心的算法即使不能够成功搜索到真实最远邻，也可以定位到很多距离查询点非常远的点，这些点对于图上搜索来说是很好的初始点。

最终，本节提出一种多中心与近邻图结合的方法进一步提升困难数据集上的最远邻搜索性能，为便于表述，算法命名为Multi+Graph。其中，多中心方法的部署参考上一节。对于近邻图的选择，本章默认使用KGraph技术<sup>[68]</sup>。因为本章的重点是验证Multi+Graph方案的可行性，因此选取了KGraph这一比较先进且被广泛使用的近邻

<sup>7</sup>即虽然没有到达真实最近邻，但是由于当前点无法提供更加靠近查询点的近邻而被迫停止。

图技术。不过，近年来学术界也提出了很多新的近邻图技术<sup>[17, 71]</sup>，本章尚未对此进行详细的考察，暂时无法给出定论。

选定近邻图之后即可部署Multi+Graph算法。其中，索引的构建过程分两步：首先参照算法3建立一个多中心索引，然后调用KGraph索引构建的方法建立一个近邻图。最远邻搜索的过程见算法5。有两点需要说明：1) 第一步的多中心方法需要搜索 $P$ 个最远邻作为初始点（第1-2行），因为KGraph算法要维护一个容量为 $P$ （一般是一个大于等于 $k$ 的值）的优先队列，一般要求初始点能将填满这个优先队列；2) 第4行提到的在KGraph进行最远邻搜索(KGraph.FNSearch)，可以通过对KGraph现有最近邻搜索算法进行一些简单调整来实现。假设需要搜索 $k$ -近似最远邻，Multi+Graph方法第一部分的时间开销可以参照上一节对MultiCentroid方法的分析，是 $O(K \log w + wg \log P)$ ；第二部分是在近邻图上搜索的过程，假设在近邻图上需要经过 $\tau$ 次迭代才能停止，近邻图上平均每个节点所维持的最近邻与反向最近邻的总数量是 $g'$ ，则Multi+Graph方法在近邻图上搜索的时间开销是 $O(\tau g' \log P)$ 。

---

**Algorithm 5** Multi+Graph.FNSearch

---

**Require:**  $q, D, k, C, FNTable$ , 近邻图 $G$ , 近邻图优先队列容量 $P$

**Ensure:**  $kFN[]$

- 1: // 使用多中心方法搜索出 $P$ 个远邻作为初始点
  - 2:  $seeds[] = \text{MultiCentroid.FNSearch}(q, D, P, C, FNTable);$
  - 3: // 在近邻图上搜索最远邻
  - 4:  $kFN[] = \text{KGraph.FNSearch}(G, P, k, seeds[]);$
  - 5: **return**  $kFN[];$
- 

## 6.4 实验结果与分析

本节设计实验评估本章提出的新最远邻搜索方案的性能。实验中所有代码使用C/C++编写，编译器为g++4.6.3。实验在一台惠普PC上进行，CPU主频2.4GHz，内存8GB，硬盘1TB，操作系统为Ubuntu 14.04 LTS。

### 6.4.1 实验设置

实验在8个公开的真实高维特征数据集上进行，详细信息见表6.8。表中 $n$ 表示数据集包含的特征向量个数， $d$ 表示特征向量维度。其中，对于规模较大的数据集(Sift1M和Deep1M)，会随机抽取100,000个点构成一个小数据集用于快速评估，分别命名为Sift100K和Deep100K。

表 6.8 数据集信息

数据集名称	$n$	$d$	数据来源
Color <sup>8</sup>	67,040	32	图像颜色直方图特征
Audio <sup>9</sup>	54,387	192	音频特征
Sift1M <sup>10</sup>	1,000,000	128	SIFT图像特征
Deep1M <sup>11</sup>	1,000,000	256	深度神经网络生成的图像特征
Mnist <sup>12</sup>	69,000	784	数字手写体图像
FaMnist <sup>13</sup>	60,000	784	Fashion Mnist, 时尚商品图像
Gist100K <sup>14</sup>	100,000	960	全局图像特征
Trevi <sup>15</sup>	99,000	4,096	旅行景点图像

由于本章主要关注内存环境中的近似最远邻搜索，每个算法考察准确率和速度两方面的性能。准确率使用  $precision$  进行衡量，指最远邻搜索的精准率，一些文章也称为召回率（recall）。假定查询点  $\mathbf{q}$  的真实最远邻是  $X^* = \{\mathbf{o}_i^* | 1 \leq i \leq k\}$ ，算法返回的最远邻结果是  $X = \{\mathbf{o}_i | 1 \leq i \leq k\}$ ， $precision$  指算法返回的结果中真实  $k$  最远邻所占的比例：

$$precision = \frac{|X \cap X^*|}{k} \quad (6-5)$$

搜索速度使用  $speedup$  进行衡量，指算法进行最远邻搜索相对于线性扫描的加速比：

$$speedup = \frac{t_{BF}}{t_A} \quad (6-6)$$

其中， $t_{BF}$  指线性扫描进行一次最远邻搜索的时间， $t_A$  指待评估算法 A 搜索一次最远邻的时间。

另外，由于现有近似最远邻搜索方案使用  $ratio$  度量准确率，为便于理解，此处也给出相关定义：

$$ratio(\mathbf{q}) = \frac{1}{k} \sum_{i=1}^k \frac{\|\mathbf{o}_i^*, \mathbf{q}\|}{\|\mathbf{o}_i, \mathbf{q}\|} \quad (6-7)$$

可以看到， $ratio$  是一种根据距离衡量搜索结果准确率的指标。

如无特殊说明，实验所观察的都是搜索  $k=10$  近似最远邻的性能。对于每个数据集，随机抽取 1,000 个点构成查询集评估算法，取所有查询点的平均性能作为最终性能。实验中，将本章提出的方法与哈希和启发式两类方法都进行比较，各个方法详细参数设置如下。

<sup>8</sup><http://archive.ics.uci.edu/ml/datasets/Corel+Image+Features/>

<sup>9</sup><http://www.cs.princeton.edu/cass/audio.tar.gz>

<sup>10</sup><http://corpus-texmex.irisa.fr/>

<sup>11</sup>[https://yadi.sk/d/l\\_yaFVqchJmoc](https://yadi.sk/d/l_yaFVqchJmoc)

<sup>12</sup><http://yann.lecun.com/exdb/mnist/>

<sup>13</sup><https://www.kaggle.com/zalando-research/fashionmnist>

<sup>14</sup><http://corpus-texmex.irisa.fr/>

<sup>15</sup><http://phototour.cs.washington.edu/patches/default.htm>

- **QDAFN.** QDAFN<sup>[41, 42]</sup>主要有三个参数：远邻搜索近似比率 $c$ 、随机投影哈希数量 $L$ 和候选点数量 $M$ 。由于QDAFN能够根据 $c$ 自动决定 $L, M$ 值，实验中将变化 $c$ 观察QDAFN的性能变化。
- **RQALSH.** RQALSH<sup>[43, 44]</sup>RQALSH\*<sup>[43, 44]</sup>是Huang等人提出的基于哈希的最远邻搜索方法，主要有五个参数：远邻搜索近似比率 $c$ 、随机投影个数 $m$ 、频繁碰撞次数阈值 $l$ 、出错概率 $\delta$ 以及搜索允许加载的误报点百分比 $\beta$ 。本节主要通过改变 $c$ 观察RQALSH的性能变化，给定 $c$ 可以自动确定 $l, m$ 取值，不过 $\delta, \beta$ 需要手动设置。我们根据原文献意见将 $\delta$ 设定为0.49。对于 $\beta$ ，原文因为考虑的是外存环境中的最远邻搜索，需要控制I/O开销，因此建议了一个较小的值 $100/n$ 。由于本章关注内存环境中的最远邻搜索，相比较于RQALSH原文不用考虑I/O开销，可以适当地增大 $\beta$ 值。本节在每个数据集比较了多个 $\beta$ 值的性能，选取性能最好的作为最终参数值，详细见表6.9。
- **RQALSH\*.** RQALSH\*<sup>[43, 44]</sup>是Huang等人提出的启发式最远邻搜索方法，本节将其细化为两个版本进行对比。一个是直接遍历所有候选集搜索远邻的版本RQALSH\*(All)，根据第6.3.2节的调参观察，这一版本只用设置一个参数 $N_C$ ；另一个是为候选集建立索引的版本RQALSH\*(Index)，同时包含RQALSH和RQALSH\*(All)两个方法的参数。根据原文建议，我们将是否建立索引的阈值设置为100，将候选集规模 $N_C$ 设置为较大的值 $n/10$ 更好地考察为候选集建立索引的效果。对于 $\beta$ ，实验通过比较 $100/n$ 和0.01两个值时的性能优劣来确定，见表6.9。最后设置 $\delta = 0.49$ ，通过改变 $c$ 观察本方法性能变化。
- **MultiCentroid.** MultiCentroid是本章为中等难度提出的基于多中心的最远邻搜索方法，有三个参数：代表点个数 $K$ 、每个代表点维护的最远邻个数 $g$ ，搜索时允许访问的最近代表点个数 $w$ 。实验中，根据数据集的难度设置合适的 $K$ 和 $g$ 值，总体上二者的值都不是很大，详细见表6.9。 $K, g$ 确定后即可建立索引构建远邻表，实验通过改变 $w$ 观察这一方法的性能变化。
- **Multi+Graph.** Multi+Graph是本章为困难数据集设计的最远邻搜索方法。其中， $K, g$ 的值和MultiCentroid方法保持一致， $w$ 的值固定为10，实验中通过改变KGraph优先队列的容量 $P$ 来观察该方法的性能变化。

在所有数据集上都进行了性能比较，接下来按照数据集的难度等级分别进行对比分析，结果分别见图6.4、图6.5和图6.6。

表 6.9 部分方法通过调参确定的参数值

数据集	RQALSH $\beta$	RQA*(Index) $\beta$	MultiCentroid $K, g$	Multi+Graph $K, g$
Gist100K	1000/n	100/n	100, 10	-
Color	0.01	0.01	100, 10	-
Trevi	1000/n	0.01	100, 10	-
Audio	1000/n	0.01	100, 10	-
Mnist	0.01	0.01	100, 100	100, 100
FaMnist	0.01	0.01	100, 100	100, 100
Sift1M	0.1	0.01	100, 1000	100, 1000
Deep1M	0.1	0.01	200, 500	200, 500

### 6.4.2 容易数据集上的比较

图6.4显示，在容易级别的数据集上（包括Gist100K, Color, Trevi和Audio），RQALSH\*(All)大部分情况下取得了最好的性能。其中在Gist100K、Color和Trevi数据集上，取得了所有方法中最好的性能；而Audio数据集有些例外，在精度大约75%以前速度最快，之后被本章提出的多中心算法超过，不过尽管如此，在90%精度的时候也能取得200左右的很高的加速比。

实际上，根据实验结果，在这些较为容易的数据集上，即使取得的最远邻搜索准确率非常高，算法所计算的数据点也很少。比如，Color数据集上，计算80个点能达到89%的准确率；Gist100K数据集上，计算20个点能达到94.8%的精度；Trevi数据集上，50个点能达到96.4%的精度；Audio数据集上稍多一些，达到92.7%的精度需要计算300个点。这些现象说明，现有启发式方法“单中心”的候选点选取策略能够有效地覆盖这些数据集内部的最远邻，是这一难度级别上最适合的方法。

### 6.4.3 中等难度数据集上的比较

在Mnist和FaMnist这两个中等难度的数据集上，情况发生了变化。图6.5显示，多中心的远邻搜索方法MultiCentroid取得了最好性能，多中心与图结合的方法Multi+Graph次之，现有单中心的启发式方法又次之。多中心方法的优势是能够同时取得很高的准确率和加速比。而多中心与图结合后准确率比较高，但是速度却比前者慢。

现有启发式方法与多中心方法的性能差距则相对比较大。在Mnist数据集上，单中心方法在取得91.5%的精度时计算了1000个点，加速比是71，而多中心方法取得94.7%的精度时计算了约250个点，加速比为231；在FaMnist数据集上，单中心方法在取得90.0%的精度时计算了2000个点，加速比31，而多中心方法取得97.1%的精度时只计算了约200个点，加速比为260。这说明，在这两个数据集上，最远邻与大的模长不再具有必然的联系：最远邻不一定具有大的模长。现有启发式方法依据模长大小挑选候选点，如果选择的点少，会漏掉很多最远邻；如果选择的点多，则会掺入很多干扰点，其结果就是难以兼顾高精度和高速度。

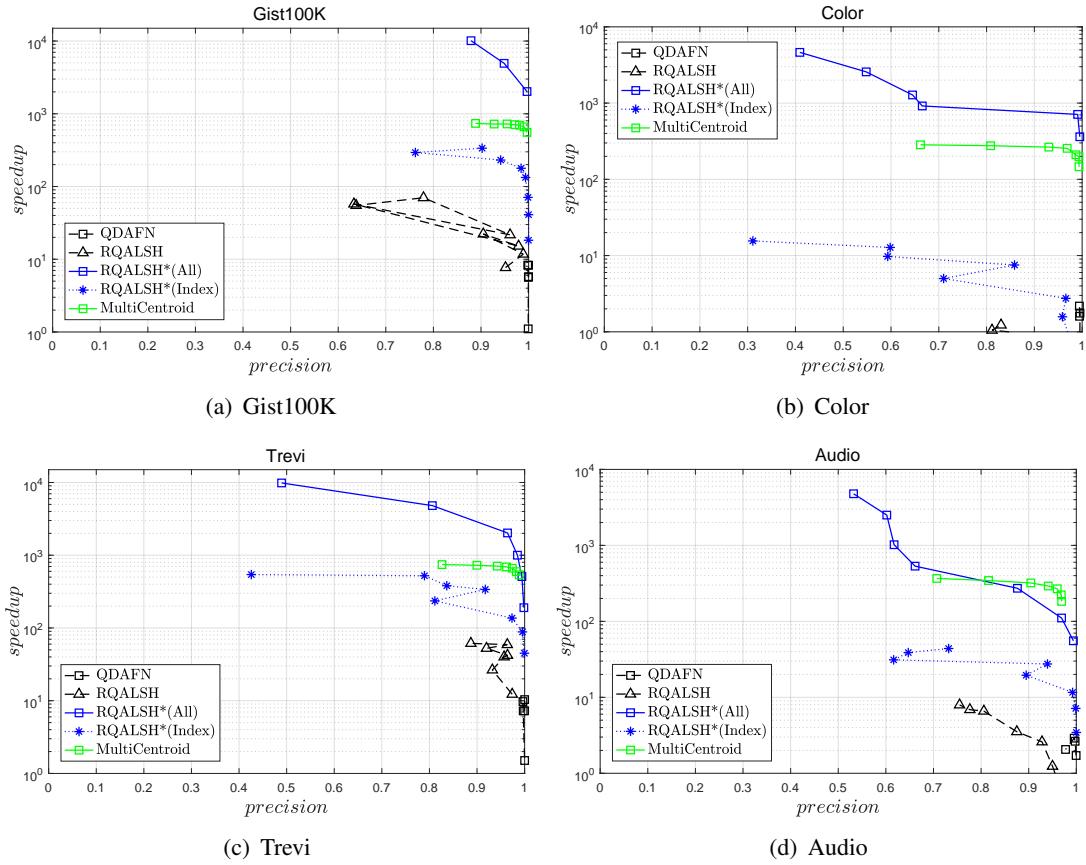


图 6.4 最远邻搜索性能对比（容易数据集）

同时，这也验证了第6.3.3节提到的远邻共享原则的有效性。第6.3.3节的观察告诉我们，数据点之间最远邻的共享率随二者的距离增加而递减。单中心方案只提供了数据集中心一种代表点，如果查询点与数据集中心距离很远，二者的最远邻共享率就比较低，会拉低搜索精度；多中心方案能够为查询点提供距离更近的代表点，具有更高的最远邻共享率，因此能够取得更高的搜索精度。

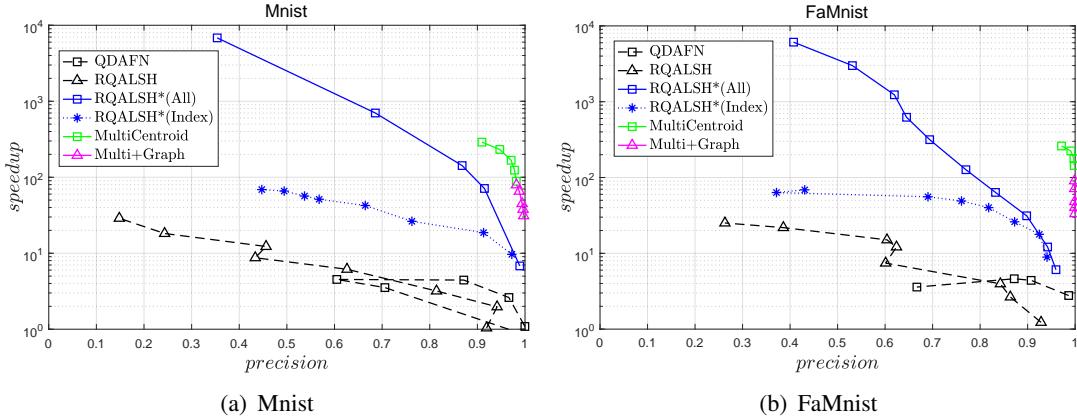


图 6.5 最远邻搜索性能对比（中等难度数据集）

#### 6.4.4 困难数据集上的比较

Sift和Deep被我们评定为困难数据集，在这两个数据集上，不同方法的表现又发生了变化。从图6.6中可以看到，多中心与图结合的方法Multi+Graph整体上取得了最好的性能，然后是多中心方法MultiCentroid和现有启发式方法RQALSH\*(All)。其中，Multi+Graph方法整体上精度比较高，而且在精度很高的时候速度也最快。Sift数据集上，达到94.6%的精度加速比是64，Deep数据集上达到93.0%的精度加速比是45。与之对比，RQALSH\*(All)在Sift数据集上达到92.9%的精度时加速比约为10，Deep数据集上达到90.8%的精度时加速比是5。

多中心的方法，在精度不太高的时候具有很快的速度，但是在高精度区域，性能快速下降。这是因为在多中心的方法中，我们调整的是允许使用的最近代表点数量 $w$ 。从图6.3中可以看到，当 $w$ 较小时，算法访问的是距离查询点较近的代表点，与查询点具有较高的最远邻共享率，因此能够带来较为不错的收益，最远邻搜索性能比现有启发式方法要好；但是当 $w$ 变大时，查询点与代表点越来越远，二者之间的最远邻共享率越来越小。此时虽然提供了更多候选点，但是候选点的质量很低，因此虽然计算开销增大，精度却没有相应的提升。

不过，虽然Multi+Graph方法取得了最好的性能，但它的基础是多中心方法奠定的。多中心方法自身虽然无法将搜索精度提升到很高的水平，但它的搜索结果对于近邻图是非常好的初始点。在此基础上，近邻图上的搜索节省了很多开销，同时降低了停在局部最优的几率，因此才能进一步提升最远邻搜索的准确率。

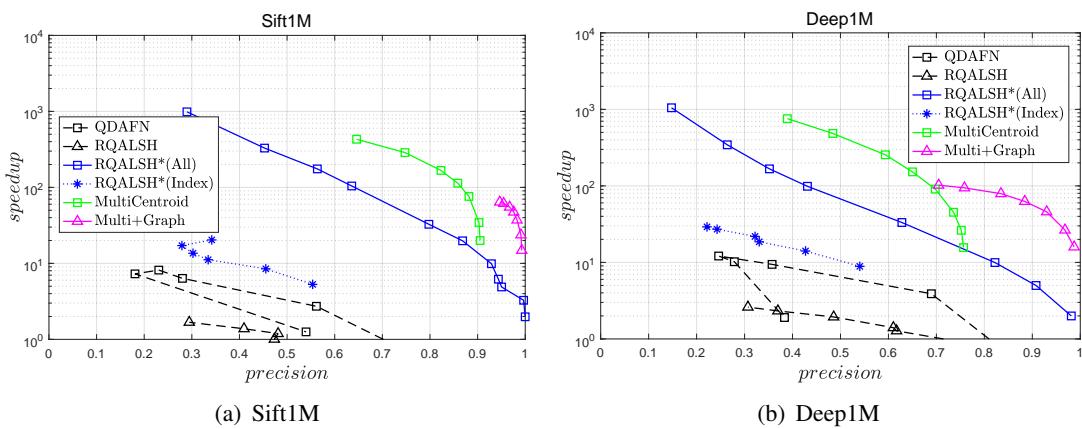


图 6.6 最远邻搜索性能对比（困难数据集）

#### 6.4.5 实验小结

综合来看，本节实验评估证实了数据集自适应的算法设计策略的合理性，即应该根据数据集的难度分配合适的算法。对于容易的数据集，现有启发式方法能够取得很好的性能，是最合适的方法；对于中等难度的数据集，多中心的方法能更好地

定位最远邻；对于最困难的数据集，需要在多中心定位的基础上使用近邻图进一步搜索。

分配合适的算法还有另外一层含义：算法并不是越复杂越好。多中心的方法比单中心方法复杂，但是在图6.4的比较中，多中心方法的性能却低于单中心方法。因为这些数据集内部成为最远邻的点非常少，不需要构造更复杂的多中心索引；类似地，对于图6.5中中等难度的数据集，基于最近的多个代表点能够很准确地定位最远邻，不需要在近邻图上做进一步的收敛。

## 6.5 本章小结

本章通过一系列实验观察进一步研究了高维空间中的近似最远邻搜索问题。现有方法能够取得很高的ratio准确率，但是在更加严格的基础上还存在较大性能欠缺。经过观察，发现最远邻的分布具有一种距离聚集效应，是造成现有哈希方法性能局限的主要原因；同时发现不同数据集内部最远邻数量存在很大差异，而现有启发式方法搜索最远邻的难度会随着数据集内部最远邻数量的增加而增大。

基于以上发现，本章提出一种数据集自适应的最远邻搜索新方案，根据数据集难度针对性地设计最远邻搜索算法以进一步提升搜索性能。首先考察了一种数据集难度快速评估方法，将数据集划分为容易、中等难度和困难三个等级。然后针对每个难度等级分别改进或设计了新的搜索算法。实验结果表明，这一自适应的策略能够为数据集分配最合适的搜索算法，在不同数据集上取得最好的最远邻搜索性能。此外，本章对现有启发式方法进行的调参优化以及自适应的搜索方案在实际应用中也将具有很好的实用性。



## 第七章 总结与展望

为充分挖掘利用非结构化大数据的价值，需要能够对海量非结构化数据进行高效处理与分析，相似性搜索是其中必不可少的一环。本文以最近邻搜索问题为代表，从精确搜索、近似搜索及其对偶问题——最远邻搜索三个角度展开研究，深入探索了适合海量、高维背景下的索引结构与搜索机制，为实现实时、准确的非结构化数据相似性搜索提供有力支撑。

### 7.1 研究总结

通过本文研究，可以总结出一个更加细化的最近/最远邻搜索方法设计策略。以最近邻搜索为例，该标准可被称作“候选点快速定位+近邻高效精炼”，将最近邻搜索的设计分为五个方面进行细化考量：无关点快速过滤、候选点全面覆盖、候选点高效加载、近邻准确判别和近邻快速收敛。这是一个从现有“过滤+精炼”策略之上发展出的更加严格的标准，为了更好地应对海量、高维特性的挑战，主要从两方面进行了细化。第一，分别对过滤和精炼环节从准确和快速两方面提出了明确要求；第二，在过滤和精炼环节之外加入了候选点加载环节以应对数据海量特性可能会引发的I/O性能瓶颈问题。为了更好地应对海量、高维带来的严峻挑战，最近/最远邻搜索算法需要加强各个方面的有效性。本文遵循这一标准对现有最近/最远邻搜索方法的优劣势进行了全面的评估，同时更好地指导了新搜索方法的设计，主要工作和结论如下。

(1) 对于高维精确最近邻搜索，现有基于聚类技术的HB方法的主要问题是无法进行快速的无关点过滤和快速的最近邻精炼。为此，提出两个优化方法针对性地解决了这些问题。一方面，基于随机投影技术和部分选择策略设计了一种距离下界加速计算方法，提升了无关数据过滤的速度；另一方面，为聚类内部成员点构造的距离下界进一步提升了整个搜索过程的无关数据过滤比例以及候选点加载效率。

(2) 对于高维近似最近邻搜索，现有方案基于LSH技术能够构建高效的索引实现近邻候选点的快速定位，但是随机I/O开销大的问题使得候选点加载环节耗时较高，容易成为整个近邻搜索的速度瓶颈。本文研究了不同特性空间曲线对候选点局部分布的影响，提出了一种基于最优排序的LSH索引方案O2LSH，通过最大程度地改善近邻候选点的局部分布，提升了近邻加载环节的I/O效率。

(3) 进一步地，对照新的算法设计标准，本文发现现有方法使用原始点作为近邻候选数据降低了多个环节的效率：一方面原始点较大的物理尺寸会限制排序方

案I/O性能的提升上限；另一方面，昂贵的原始距离也会增加近邻精炼环节的计算开销。经过综合对比，本文发现短判别码比原始点更加适合作为近邻候选数据，因此提出一种基于有序短判别码的新LSH索引方法SC-LSH。实验表明，新方法能够显著提升磁盘局部单个页面中的候选数据数量，从而仅用少量顺序I/O即可加载到充分多的候选数据，全面提升了候选点加载环节和近邻精炼环节的效率。

(4) 对于高维近似最远邻搜索，首先通过实验观察发现了最远邻分布具有一种距离聚集效应，会增大哈希方法过滤无关点的负担。而启发式方法所依赖的大模长与最远邻具有较强相关性的现象只适用于最远邻点很少的数据集，对于最远邻较多的数据集，则很难高效定位最远邻候选点。

基于以上发现，提出一种数据集自适应的远邻搜索方案，针对不同特性的数据集分别设计了远邻搜索方法。对于容易的数据集，现有启发式方法能够取得最好的性能，本文对其调参方法进行了改进，使其更加实用；在居中难度数据集上发现邻近点有很高的远邻共享率，基于此设计了一种多中心的远邻搜索方案，能够更加准确地定位最远邻；对于困难数据集，发现邻近点的远邻共享率并不显著使得多中心方法无法全面覆盖最远邻，因此设计了一种多中心与近邻图结合的搜索方案，多中心方法能够快速定位到一些高质量的起始搜索点，近邻图能够提供明确的搜索路径，二者结合能够更加高效地向最远邻收敛。最终本文提出的方案能够为不同数据集分配最合适的搜索算法，取得最好的远邻搜索性能。

## 7.2 未来展望

虽然本文在最近邻搜索和最远邻搜索问题上取得了一些进展，但非结构化数据海量、高维特性的挑战依然严峻。数据总量依旧在持续地快速增长，高维也依然没有被充分理解。高斯奖获得者David L. Donoho教授曾将“维度灾难”问题列为21世纪的一大数学挑战<sup>[107]</sup>。本文只是在前人工作的基础上，通过一些观察发现了一些规律，并据此提出了一些改进方法。对于相似性搜索本文认为还有很多问题可以进一步研究：

(1) 迁移到其他相似性搜索问题：除了最近邻和最远邻搜索，相似性搜索领域还有很多其他重要问题，比如最近对/最远对搜索、最大内积搜索、查询超平面的最近邻/最远邻等等。它们同本文研究的这两个问题有很大的共性，比如也会划分精确、近似搜索版本以及存在对偶问题。最主要的，这些问题也要面对非结构化数据带来的海量、高维特性的挑战。本文在海量、高维背景下为最近邻搜索和最远邻搜索凝练出了一个经验策略，未来会尝试将其应用于其他问题领域，帮助更加全面的评估现有研究或指导新研究。

(2) 适应数据分布特性的精确最近邻搜索：在精确最近邻搜索研究中，虽然基于

聚类的方法能够构造出数据自适应的距离下界，但是实际特征数据的分布特性有很大差异，有些如果不适合聚类，则有可能不能取得很好的效果，需要进一步研究。实际上，现有精确最近邻搜索围绕不同技术手段提出了很多方法，如降维、向量近似、矢量量化以及嵌入等。这些方法是否有各自适合的场景，能否根据数据集自身特性自动挑选最适合的方案，可以作进一步尝试。

(3) 近期有一些工作<sup>[108, 109]</sup>通过将相似性搜索中的一些关键环节转化为学习问题并借助机器学习技术进行优化，进一步提升了搜索性能，为相似性搜索的未来研究提供了新的启发。本文所设计的方法也存在类似的可优化之处，例如O2LSH和SCLSH方法的搜索终止条件（即允许算法能够加载的磁盘页面个数）目前都是手动设定的。对于不同查询点，能否借助学习技术自适应地决定终止条件，是一个值得尝试的点。

(4) 最远邻分布成因探究：在最远邻搜索研究过程中发现，有相当数量的数据集，虽然规模很大、维度很高，但是却在一个比较简单的方法下取得了很好的搜索性能。这些数据集的共同点是内部最远邻点很少，且都具有很大的模长。对于这一现象的形成原因，目前尚不清楚，是否跟数据集的生成方式有关？是否能够应用于其他形式的最远邻搜索问题，如查询超平面的最远邻等等？是下一步一个很好的探索方向。



## 参考文献

- [1] 国务院. 国务院关于印发促进大数据发展行动纲要的通知[EB/OL]. (2015-08-31)[2015-09-05]. [http://www.gov.cn/zhengce/content/2015-09/05/content\\_10137.htm](http://www.gov.cn/zhengce/content/2015-09/05/content_10137.htm).
- [2] REINSEL D, 武连峰, GANTZ J F, et al. IDC: 2025年中国将拥有全球最大的数据圈[R]. Framingham : IDC, 2019.
- [3] 李国杰. 大数据研究的科学价值[J]. 中国计算机学会通讯, 2013, 9(2) : 8 – 15.
- [4] SMEULDERS A W M, WORRING M, SANTINI S, et al. Content-Based Image Retrieval at the End of the Early Years[J]. IEEE Trans. Pattern Anal. Mach. Intell., 2000, 22(12) : 1349 – 1380.
- [5] AL-OMARI F A, AL-JARRAH M A. Query by image and video content: a colored-based stochastic model approach[J]. Data Knowl. Eng., 2005, 52(3) : 313 – 332.
- [6] 崔江涛. 高维索引技术中向量近似方法研究[D]. 西安 : 西安电子科技大学, 2005.
- [7] 袁培森, 沙朝锋, 王晓玲, et al. 一种基于学习的高维数据c-近似最近邻搜索算法[J]. 软件学报, 2012, 23(8) : 2018 – 2031.
- [8] MITCHELL T M. Machine Learning[M]. Sydney : McGraw-Hill Education, 1997.
- [9] ALTMAN N S. An introduction to kernel and nearest-neighbor nonparametric regression[J]. The American Statistician, 1992, 46(3) : 175 – 185.
- [10] RAMASWAMY S, RASTOGI R, SHIM K. Efficient Algorithms for Mining Outliers from Large Data Sets[C] // Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA. 2000 : 427 – 438.
- [11] CAMPOS G O, ZIMEK A, SANDER J, et al. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study[J]. Data Min. Knowl. Discov., 2016, 30(4) : 891 – 927.
- [12] LESKOVEC J, RAJARAMAN A, ULLMAN J D. Mining of Massive Datasets, 2nd Ed[M]. University of Cambridge : Cambridge University Press, 2014.
- [13] BELLOGÍN A, PARAPAR J. Using graph partitioning techniques for neighbour selection in user-based collaborative filtering[C] // Sixth ACM Conference on Recommender Systems, RecSys '12, Dublin, Ireland, September 9-13, 2012. 2012 : 213 – 216.
- [14] BOURKE S, MCCARTHY K, SMYTH B. Power to the people: exploring neighbourhood formations in social recommender system[C] // Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011. 2011 : 337 – 340.
- [15] CANDILLIER L, JACK K, FESSANT F, et al. State-of-the-art recommender systems[G] // Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling. Pennsylvania : IGI Global, 2009 : 1 – 22.

- [16] LI W, ZHANG Y, SUN Y, et al. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement (v1.0)[J]. CoRR, 2016, abs/1610.02455.
- [17] Li W, Zhang Y, Sun Y, et al. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement[J]. IEEE Transactions on Knowledge and Data Engineering, 2019 : 1 – 1.
- [18] AGARWAL P K, MATOUŠEK J, SURI S. Farthest Neighbors, Maximum Spanning Trees and Related Problems in Higher Dimensions[J]. Comput. Geom., 1991, 1 : 189 – 201.
- [19] VEENMAN C J, REINDERS M J T, BACKER E. A Maximum Variance Cluster Algorithm[J]. IEEE Trans. Pattern Anal. Mach. Intell., 2002, 24(9) : 1273 – 1280.
- [20] VASILOGLOU N, GRAY A G, ANDERSON D V. Scalable semidefinite manifold learning[C] // 2008 IEEE Workshop on Machine Learning for Signal Processing. 2008 : 368 – 373.
- [21] SAID A, KILLE B, JAIN B J, et al. Increasing diversity through furthest neighbor-based recommendation[J]. Proceedings of the WSDM, 2012, 12.
- [22] SAID A, FIELDS B, JAIN B J, et al. User-centric evaluation of a k-furthest neighbor collaborative filtering recommender algorithm[C] // Proceedings of the 2013 conference on Computer supported cooperative work. 2013 : 1399 – 1408.
- [23] FAROUGHI A, JAVIDAN R. CANF: Clustering and anomaly detection method using nearest and farthest neighbor[J]. Future Gener. Comput. Syst., 2018, 89 : 166 – 177.
- [24] BOUGET S, BROMBERG Y, TAÏANI F, et al. Scalable Anti-KNN: Decentralized Computation of k-Furthest-Neighbor Graphs with HyFN[C] // Distributed Applications and Interoperable Systems - 17th IFIP WG 6.1 International Conference, DAIS 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings. 2017 : 101 – 114.
- [25] BÖHM C, BERCHTOLD S, KEIM D A. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases[J]. ACM Comput. Surv., 2001, 33(3) : 322 – 373.
- [26] WEBER R, SCHEK H-J, BLOTT S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces[C] // VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA : Vol 98. 1998 : 194 – 205.
- [27] CURTIN R R, GARDNER A B. Fast Approximate Furthest Neighbors with Data-Dependent Candidate Selection[C] // Similarity Search and Applications - 9th International Conference, SISAP 2016, Tokyo, Japan, October 24-26, 2016. Proceedings. 2016 : 221 – 235.
- [28] CURTIN R R, ECHAUZ J, GARDNER A B. Exploiting the structure of furthest neighbor search

- for fast approximate results[J]. Inf. Syst., 2019, 80 : 124–135.
- [29] GUTTMAN A. R-Trees: A Dynamic Index Structure for Spatial Searching[C] // SIGMOD'84, Proceedings of Annual Meeting, June 18-21, 1984. Boston, Massachusetts, USA : ACM Press, 1984 : 47–57.
- [30] BECKMANN N, KRIEGEL H, SCHNEIDER R, et al. The R\*-Tree: An Efficient and Robust Access Method for Points and Rectangles[C] // Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, May 23-25, 1990. Atlantic City, NJ, USA : ACM Press, 1990 : 322–331.
- [31] WHITE D A, JAIN R C. Similarity Indexing with the SS-tree[C] // Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996. New Orleans, Louisiana, USA : IEEE Computer Society, 1996 : 516–523.
- [32] RAMASWAMY S, ROSE K. Adaptive Cluster Distance Bounding for High-Dimensional Indexing[J]. IEEE Trans. Knowl. Data Eng., 2011, 23(6) : 815–830.
- [33] INDYK P, MOTWANI R. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality[C] // Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, May 23-26, 1998. Dallas, Texas, USA : ACM, 1998 : 604–613.
- [34] DATAR M, IMMORLICA N, INDYK P, et al. Locality-sensitive hashing scheme based on p-stable distributions[C] // Proceedings of the 20th ACM Symposium on Computational Geometry, June 8-11, 2004. Brooklyn, New York, USA : ACM, 2004 : 253–262.
- [35] GIONIS A, INDYK P, MOTWANI R. Similarity Search in High Dimensions via Hashing[C] // VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999. Edinburgh, Scotland, UK : Morgan Kaufmann, 1999 : 518–529.
- [36] TAO Y, YI K, SHENG C, et al. Quality and efficiency in high dimensional nearest neighbor search[C] // Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, June 29 - July 2, 2009. Providence, Rhode Island, USA : ACM, 2009 : 563–576.
- [37] GAN J, FENG J, FANG Q, et al. Locality-sensitive hashing scheme based on dynamic collision counting[C] // Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, May 20-24, 2012. Scottsdale, AZ, USA : ACM, 2012 : 541–552.
- [38] SUN Y, WANG W, QIN J, et al. SRS: Solving c-Approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index[J]. PVLDB, 2014, 8(1) : 1–12.
- [39] HUANG Q, FENG J, ZHANG Y, et al. Query-aware locality-sensitive hashing for approximate nearest neighbor search[J]. PVLDB, 2015, 9(1) : 1–12.
- [40] LIU Y, CUI J, HUANG Z, et al. SK-LSH: An Efficient Index Structure for Approximate Nearest Neighbor Search[J]. PVLDB, 2014, 7(9) : 745–756.

- [41] PAGH R, SILVESTRI F, SIVERTSEN J, et al. Approximate Furthest Neighbor in High Dimensions[C] // Similarity Search and Applications - 8th International Conference, SISAP 2015, Glasgow, UK, October 12-14, 2015, Proceedings. 2015 : 3 – 14.
- [42] PAGH R, SILVESTRI F, SIVERTSEN J, et al. Approximate furthest neighbor with application to annulus query[J]. Inf. Syst., 2017, 64 : 152 – 162.
- [43] HUANG Q, FENG J, FANG Q. Reverse Query-Aware Locality-Sensitive Hashing for High-Dimensional Furthest Neighbor Search[C] // 33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017. 2017 : 167 – 170.
- [44] HUANG Q, FENG J, FANG Q, et al. Two Efficient Hashing Schemes for High-Dimensional Furthest Neighbor Search[J]. IEEE Trans. Knowl. Data Eng., 2017, 29(12) : 2772 – 2785.
- [45] KATAYAMA N, SATOH S. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries[C] // SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997. Tucson, Arizona, USA : ACM Press, 1997 : 369 – 380.
- [46] BERCHTOLD S, KEIM D A, KRIEGEL H. The X-tree: An Index Structure for High-Dimensional Data[C] // VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996. Mumbai (Bombay), India : Morgan Kaufmann, 1996 : 28 – 39.
- [47] ROBINSON J T. The K-D-B-Tree: A Search Structure For Large Multidimensional Dynamic Indexes[C] // Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, April 29 - May 1, 1981. Ann Arbor, Michigan, USA : ACM Press, 1981 : 10 – 18.
- [48] BENTLEY J L. Multidimensional binary search trees used for associative searching[J]. Communications of the ACM, 1975, 18(9) : 509 – 517.
- [49] FRIEDMAN M, KANDEL A. Series in Machine Perception and Artificial Intelligence, Vol 32 : Introduction to Pattern Recognition - Statistical, Structural, Neural and Fuzzy Logic Approaches[M]. Singapore : WorldScientific, 1999.
- [50] FALOUTSOS C, BARBER R, FLICKNER M, et al. Efficient and Effective Querying by Image Content[J]. J. Intell. Inf. Syst., 1994, 3(3/4) : 231 – 262.
- [51] CHAKRABARTI K, MEHROTRA S. Local Dimensionality Reduction: A New Approach to Indexing High Dimensional Spaces[C] // VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000. Cairo, Egypt : Morgan Kaufmann, 2000 : 89 – 100.
- [52] FALOUTSOS C. Advances in Database Systems, Vol 3 : Searching Multimedia Databases by Content[M]. Maryland : Kluwer, 1996.
- [53] JAGADISH H V, OOI B C, TAN K-L, et al. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search[J]. ACM Trans. Database Syst., 2005, 30(2) : 364 – 397.

- [54] CUI J, HUANG Z, WANG B, et al. Near-Optimal Partial Linear Scan for Nearest Neighbor Search in High-Dimensional Space[C] // Database Systems for Advanced Applications, 18th International Conference, DASFAA 2013, Wuhan, China, April 22-25, 2013. Proceedings, Part I. 2013 : 101 – 115.
- [55] BERCHTOLD S, BÖHM C, JAGADISH H V, et al. Independent Quantization: An Index Compression Technique for High-Dimensional Data Spaces[C] // Proceedings of the 16th International Conference on Data Engineering, February 28 - March 3, 2000. San Diego, California, USA : IEEE Computer Society, 2000 : 577 – 588.
- [56] FERHATOSMANOGLU H, TUNCEL E, AGRAWAL D, et al. Vector Approximation based Indexing for Non-uniform High Dimensional Data Sets[C] // Proceedings of the 2000 ACM CIKM International Conference on Information and Knowledge Management, November 6-11, 2000. McLean, VA, USA : ACM, 2000 : 202 – 209.
- [57] CUI J, ZHOU S, SUN J. Efficient high-dimensional indexing by sorting principal component[J]. Pattern Recogn. Lett., 2007, 28(16) : 2412 – 2418.
- [58] GRAY R M. Vector quantization[J]. ASSP Magazine, IEEE, 1984, 1(2) : 4 – 29.
- [59] HWANG Y, HAN B, AHN H. A fast nearest neighbor search algorithm by nonlinear embedding[C] // 2012 IEEE Conference on Computer Vision and Pattern Recognition, June 16-21, 2012. Providence, RI, USA : IEEE Computer Society, 2012 : 3053 – 3060.
- [60] LI M, ZHANG Y, SUN Y, et al. An Efficient Exact Nearest Neighbor Search by Compounded Embedding[C] // Lecture Notes in Computer Science, Vol 10827 : Database Systems for Advanced Applications - 23rd International Conference, DASFAA 2018, May 21-24, 2018, Proceedings, Part I. Gold Coast, QLD, Australia : Springer, 2018 : 37 – 54.
- [61] VITTER J S. Algorithms and data structures for external memory[J]. Foundations and Trends® in Theoretical Computer Science, 2008, 2(4) : 305 – 474.
- [62] JEGOU H, DOUZE M, SCHMID. C. Product quantization for nearest neighbor search[J]. IEEE Trans. Pattern Anal. Mach. Intell., 2011, 33(1) : 117 – 128.
- [63] NOROUZI M, FLEET D J. Cartesian K-Means[C] // 2013 IEEE Conference on Computer Vision and Pattern Recognition, June 23-28, 2013. Portland, OR, USA : IEEE Computer Society, 2013 : 3017 – 3024.
- [64] KALANTIDIS Y, AVRITHIS Y. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search[C] // 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014. 2014 : 2329 – 2336.
- [65] WANG J, WANG J, SONG J, et al. Optimized Cartesian K-Means[J]. IEEE Trans. Knowl. Data Eng., 2015, 27(1) : 180 – 192.

- [66] JOHNSON J, DOUZE M, JÉGOU H. Billion-scale similarity search with GPUs[J]. IEEE Transactions on Big Data, 2019.
- [67] HAJEBI K, ABBASI-YADKORI Y, SHAHBAZI H, et al. Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph[C] // IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, July 16-22, 2011. Barcelona, Catalonia, Spain : IJCAI/AAAI, 2011 : 1312 – 1317.
- [68] DONG W, CHARIKAR M, LI K. Efficient k-nearest neighbor graph construction for generic similarity measures[C] // Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011. 2011 : 577 – 586.
- [69] MALKOV Y, PONOMARENKO A, LOGVINOV A, et al. Approximate nearest neighbor algorithm based on navigable small world graphs[J]. Inf. Syst., 2014, 45 : 61 – 68.
- [70] MALKOV Y A, YASHUNIN D A. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs[J]. CoRR, 2016, abs/1603.09320.
- [71] FU C, XIANG C, WANG C, et al. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph[J]. PVLDB, 2019, 12(5) : 461 – 474.
- [72] PANIGRAHY R. Entropy based nearest neighbor search in high dimensions[C] // Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, January 22-26, 2006. Miami, Florida, USA : ACM Press, 2006 : 1186 – 1195.
- [73] LV Q, JOSEPHSON W, WANG Z, et al. Multi-Probe LSH: Efficient Indexing for High-Dimensional Similarity Search[C] // Proceedings of the 33rd International Conference on Very Large Data Bases, September 23-27, 2007. University of Vienna, Austria : ACM, 2007 : 950 – 961.
- [74] 王磊. 基于邻域碰撞计数的局部敏感哈希索引[D]. 西安: 西安电子科技大学, 2018.
- [75] de BERG M, CHEONG O, van KREVELD M J, et al. Computational geometry: algorithms and applications, 3rd Edition[M]. Berlin : Springer, 2008.
- [76] INDYK P. Dimensionality reduction techniques for proximity problems[C] // Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA. 2000 : 371 – 378.
- [77] AGARWAL P K, MATOUVSKE J, SURI S. Farthest Neighbors, Maximum Spanning Trees and Related Problems in Higher Dimensions[J]. Comput. Geom., 1991, 1 : 189 – 201.
- [78] BESPAMYATNIKH S. Dynamic Algorithms for Approximate Neighbor Searching[C] // Proceedings of the 8th Canadian Conference on Computational Geometry, Carleton University, Ottawa, Canada, August 12-15, 1996. 1996 : 252 – 257.
- [79] INDYK P. Better algorithms for high-dimensional proximity problems via asymmetric embed-

- dings[C] // Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA. 2003 : 539 – 545.
- [80] SIVERTSEN J. Similarity Problems in High Dimensions[D]. Denmark : IT University of Copenhagen, 2017.
- [81] 黄强. 高维欧氏空间中的近似相似性检索[D]. 广州 : 中山大学, 2017.
- [82] ACHLIOPHAS D. Database-friendly random projections:Johnson-Lindenstrauss with binary coins[J]. Journal of Computer and System Sciences, 2003, 66(4) : 671 – 687.
- [83] BINGHAM E, MANNILA H. Random projection in dimensionality reduction: applications to image and text data[C] // Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, August 26-29, 2001. San Francisco, CA, USA : ACM, 2001 : 245 – 250.
- [84] LI P, HASTIE T, CHURCH K W. Very sparse random projections[C] // Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 20-23, 2006. Philadelphia, PA, USA : ACM, 2006 : 287 – 296.
- [85] JOHNSON W B, LINDENSTRAUSS J. Extensions of Lipschitz mappings into a Hilbert space[J]. Contemporary mathematics, 1984, 26(189-206) : 1.
- [86] DONG W, CHARIKAR M, LI K. Efficient k-nearest neighbor graph construction for generic similarity measures[C] // Proceedings of the 20th International Conference on World Wide Web, WWW 2011, March 28 - April 1, 2011. Hyderabad, India : ACM, 2011 : 577 – 586.
- [87] GAEDE V, GÜNTHER O. Multidimensional access methods[J]. ACM Comput. Surv., 1998, 30(2) : 170 – 231.
- [88] TANG B, MAN L Y, HUA K A. Exploit Every Bit: Effective Caching for High-Dimensional Nearest Neighbor Search[J]. IEEE Trans. Knowl. Data Eng., 2016, 28(5) : 1175 – 1188.
- [89] ASANO T, RANJAN D, ROOS T, et al. Space-Filling Curves and Their Use in the Design of Geometric Data Structures[J]. Theor. Comput. Sci., 1997, 181(1) : 3 – 15.
- [90] LIAO S, LÓPEZ M A, LEUTENEGGER S T. High Dimensional Similarity Search With Space Filling Curves[C] // Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany. 2001 : 615 – 622.
- [91] VALLE E, CORD M, PHILIPP-FOLIGUET S. High-dimensional descriptor indexing for large multimedia databases[C] // Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008. 2008 : 739 – 748.
- [92] AMSALEG L, CHELLY O, FURON T, et al. Estimating Local Intrinsic Dimensionality[C] // Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and

- Data Mining, Sydney, NSW, Australia, August 10-13, 2015. 2015 : 29 – 38.
- [93] SKILLING J. Programming the Hilbert curve[C] // AIP Conference Proceedings : Vol 707. 2004 : 381 – 387.
- [94] WANG J, SHEN H T, SONG J, et al. Hashing for Similarity Search: A Survey[J]. CoRR, 2014, abs/1408.2927.
- [95] HE S, YE G, HU M, et al. Learning binary codes with local and inner data structure[J]. Neurocomputing, 2018, 282 : 32 – 41.
- [96] LUO X, NIE L, HE X, et al. Fast Scalable Supervised Hashing[C] // The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, July 08-12, 2018. Ann Arbor, MI, USA : ACM, 2018 : 735 – 744.
- [97] ZHANG P-F, LI C-X, LIU M-Y, et al. Semi-relaxation supervised hashing for cross-modal retrieval[C] // Proceedings of the 2017 ACM on Multimedia Conference. 2017 : 1762 – 1770.
- [98] SHEN F, ZHOU X, YANG Y, et al. A Fast Optimization Method for General Binary Code Learning[J]. IEEE Transactions on Image Processing, 2016, 25(12) : 5610 – 5621.
- [99] SHEN F, YANG Y, LIU L, et al. Asymmetric Binary Coding for Image Search[J]. IEEE Transactions on Multimedia, 2017, 19(9) : 2022 – 2032.
- [100] SHEN F, XU Y, LIU L, et al. Unsupervised deep hashing with similarity-adaptive and discrete optimization[J]. IEEE transactions on pattern analysis and machine intelligence, 2018.
- [101] LI Z, NIE F, CHANG X, et al. Beyond trace ratio: weighted harmonic mean of trace ratios for multiclass discriminant analysis[J]. IEEE Transactions on Knowledge and Data Engineering, 2017, 29(10) : 2100 – 2110.
- [102] LOWE D G. Distinctive Image Features from Scale-Invariant Keypoints[J]. International Journal of Computer Vision, 2004, 60(2) : 91 – 110.
- [103] WEISS Y, TORRALBA A, FERGUS R. Spectral Hashing[C] // Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008. 2008 : 1753 – 1760.
- [104] GONG Y, LAZEBNIK S. Iterative quantization: A procrustean approach to learning binary codes[C] // The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, 20-25 June 2011. Colorado Springs, CO, USA : IEEE Computer Society, 2011 : 817 – 824.
- [105] PARK Y, CAFARELLA M J, MOZAFARI B. Neighbor-Sensitive Hashing[J]. PVLDB, 2015, 9(3) : 144 – 155.
- [106] OLIVA A, TORRALBA A. Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope[J]. International Journal of Computer Vision, 2001, 42(3) : 145 – 175.

## 参考文献

---

- [107] DONOHO D L, OTHERS. High-dimensional data analysis: The curses and blessings of dimensionality[J]. AMS math challenges lecture, 2000, 1(2000): 32.
- [108] LI C, ZHANG M, ANDERSEN D G, et al. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination[C] // Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, June 14-19, 2020. online conference [Portland, OR, USA]: ACM, 2020: 2539 – 2554.
- [109] LI M, ZHANG Y, SUN Y, et al. I/O Efficient Approximate Nearest Neighbour Search based on Learned Functions[C] // 2020 IEEE 36th International Conference on Data Engineering (ICDE). 2020: 289 – 300.



## 致谢

感谢我的导师崔江涛教授。感谢老师当初带我结识高维索引这个问题，从此结下不解之缘，深入领略了一次钻研科研的过程；感谢老师营造的宽松氛围，让我能潜心问题，自由探索；感谢这个问题带给我的那些宁静与专注时刻，思之忘我，暗涌欣愉；感谢老师关键时刻的指点和提醒，在我受挫时让我及时摆正心态，重拾前行的勇气；感谢老师的严格要求，让我有机会领略更高处的学术风景。值此论文完稿之际，衷心感谢老师的指导和谆谆教诲！

感谢李辉教授的指导和教诲。李老师的專業总是令我叹服，很惊叹怎么有如此什么都懂的老师。每次遇到难题都被老师拆解得头头是道，受益匪浅。很有幸蒙受李老师的指导。感谢新南威尔士大学王炜教授。“炜者，光彩鲜明”，老师对于科研的热情令人深受鼓舞，很喜欢与老师一起激情的讨论。感谢老师的批判和鼓励，引导我思考困难的问题。谢谢老师用《师说》、《卖油翁》回答我对于科研的困惑，微言大义，启发良多。很有幸接受您的指导。

感谢带我入门高维索引的阿帆师兄，喜欢和师兄一起讨论问题，听师兄讲故事。感谢对我工作生活细致指点的彭师兄，对我找工作比我还着急的师兄和嫂子。感谢与夏老师的相遇，希望你生活幸福，工作顺利。感谢与李冰师兄的相遇，也是后知后觉师兄朴实的外表下掩藏着多大的霸气。感谢与亦方师兄的相遇，谢谢师兄对我未来科研方向的建议，希望有一天能学会师兄那样的细致成熟。

感谢我的研究生舍友和同学，史雨、杨阳、泽民、晓斌、王叔叔、王凌霄和月姐，有些事情分别后才知道珍惜，感谢有你们陪伴的最初几年。感谢与杰哥的相遇，非常感谢杰哥对我出国的启发、鼓励和帮助。感谢新南的舍友，星宇、长政和任飞。

感谢实验室所有的师兄师姐师弟师妹，书不尽言，感谢与你们每一位的相遇，与你们交流、合作、共同学习。感谢我的博士同学，小朱、侯老师、何老师、王蒙师兄。读博漫漫，感谢能有你们相伴，一起切心交流，互相关心。一直记得侯老师对我一些重要困惑的指点，非常感谢你。

感谢我的家人，一直以来对我的默默支持、殷殷关切和担忧。感谢你们对我的包容，很抱歉一直任性，没有早点让你们放下心来。感谢我亲爱的女朋友，在我身心最低谷的时刻陪我一同走过，感谢你一直以来的支持和包容。

感谢留学基金委的资助，一年虽短却收获甚多：语言自信心的提升、一段深刻铭记的教诲、国外校友、同胞的风采和真诚相待以及一段难忘的友谊。

感谢读博过程中我所有论文的评审、答辩老师。感谢你们的无私奉献、认真的指点和宝贵的建议。

感谢在西电和新南遇到的所有球友，很开心能与你们一起投入这项共同喜爱的运动。过程中每一次的挥洒、舒展、不屈和忘我地表达，点点滴滴都汇入我信心和健康的河流，成为我继续前进的动力。

感谢那些不知从何处而来，曾经光顾我脑海的灵感。

最后，感谢一路走来我的那些“心药”，无数迷茫、低落之时很有幸遇到那些文字和言语，仿佛心中重新渗出一泓清泉，滋润出新的可能。

At the sight of beauty, the soul grows wings.

## 攻博期间取得的研究成果和参与的科研项目

### 一、 学术论文

- [1] FENG X, CUI J, LIU Y, LI H, et al. Effective optimizations of cluster-based nearest neighbor search in high-dimensional space[J/OL]. Multimedia Systems, 2017, 23(1): 139-153. (第一作者) (和本文第三章相关) (SCI源刊, 已发表, 检索号: 000393759100014)
- [2] 冯小康, 彭延国, 崔江涛, 刘英帆, 李辉. 基于最优排序的局部敏感哈希索引[J/OL]. 计算机学报, 2019. (2018中国数据库学术会议“萨师煊优秀学生论文(Best Student Papers)”奖、“最佳论文提名”奖) (第一作者) (和本文第四章相关) (EI源刊, 已录用)
- [3] FENG X, CUI J, LI H, LIU Y, et al. An efficient LSH indexing on discriminative short codes for high-dimensional nearest neighbors[J/OL]. Multimedia Tools and Applications, 2019, 78(17): 24407 - 24429. (第一作者) (和本文第五章相关) (SCI源刊, 已发表, 检索号: 000482419900033)

### 二、 授权专利

- [1] 崔江涛, 冯小康, 刘畅, 侯勇超, 蔡洋. 一种索引生成方法、数据检索方法和装置: 中国, ZP201611170581.9[P]. 2020-5-12.

### 三、 参与科研项目及获奖

- [1] 国家自然科学基金面上项目, 基于哈希的海量高维数据近似最近邻查询研究, 2015年1月-2018年12月, 已结题, 负责外存哈希索引设计与实现.
- [2] 西安电子科技大学研究生创新基金, 海量非结构化数据近邻搜索关键技术研究, 2018年5月-2019年5月, 已结题, 主持.

