硕士学位论文

基于 P-稳态分布和空间球面网格的位置敏感哈希算法

LOCALITY SENSITIVE HASHING ALGORITHM BASED ON P-STABLE DISTRIBUTION AND SPACE SPHERE LATTICE

陈翔

哈尔滨工业大学

2015 年 12 月

国内图书分类号：TP391.4　　学校代码：10213
国际图书分类号：621.3　　密级：公开

工学硕士学位论文

基于 P-稳态分布和空间球面网格的位置敏感哈希算法

硕士研究生：陈 翔
导　　师：王轩教授
申请学位：工学硕士
学　　科：计算机科学与技术
所在单位：深圳研究生院
答辩日期：2015 年 12 月
授予学位单位：哈尔滨工业大学

Classified Index: TP391.4
U.D.C: 621.3

Dissertation for the Master Degree in Engineering

LOCALITY SENSITIVE HASHING ALGORITHM BASED ON P-STABLE STRIBUTION AND SPACE SPHERE LATTICE

Candidate: Chen Xiang
Supervisor: Prof. Wang Xuan
Academic Degree Applied for: Master Degree in Engineering
Specialty: Computer Science and Technology
Affiliation: Shenzhen Graduate School
Date of Defence: December, 2015
Degree-Conferring-Institution: Harbin Institute of Technology

摘 要

利用数据的相似性对海量数据进行检索是计算机科学中的一个热点研究问题，在多个计算机领域应用广泛。利用数据的相似性进行检索的方法分为两类，最邻近检索和近似最邻近检索。位置敏感哈希算法是一种以概率论和欧几里德几何为理论基础的近似最近邻检索算法。传统的位置敏感哈希算法存在着一些不足，例如传统的位置敏感哈希算法对空间中距离远的数据点的过滤效果有限。而且为了保证算法较高的准确率，需要构建大量的索引表，降低了算法的索引建立效率。为了提出性能更好的解决相似性检索问题的算法，本文在原有的工作上做了以下工作：

研究了传统的位置敏感哈希算法，指出了当查询点与数据点距离远时过滤效果差的缺点，针对这个缺点提出两层结构的位置敏感哈希算法。两层位置敏感哈希算法利用空间球面网格将数据集划分成若干有界的子数据集来提高位置敏感哈希算法对距离远的点的过滤效果，同时对数据点的投影区间进行延伸，降低距离近的点被映射到不同区间的概率。计算了两层位置敏感哈希函数的碰撞概率，并给出了两层位置敏感哈希函数的参数ρ的上限。结果显示两层位置敏感哈希函数在误判率和漏判率低于原有的位置敏感哈希函数。利用 MATLAB 在 CIFAR-10 数据集的 GIST 特征库上对两层位置敏感哈希函数做了对比试验，在准确率和召回率上将两层位置敏感哈希函数与现有的位置敏感哈希函数进行对比。

针对两层位置敏感哈希算法的结构的特点，设计了分布式哈希索引表。两层分布式索引表结构利用两层位置敏感哈希算法分两步对数据集进行哈希的特点，将索引表设计成两层分布式结构。两层分布式索引表可以有效降低算法进行检索时的内存占用率，提高了算法对海量数据进行检索时的检索效率。本课题使用搜狗实验室 2012 版本的数据测试集进行实验，并在检索性能方面与其他海量数据检索算法进行了对比分析。对使用了两层哈希索引表的两层位置敏感哈希算法在检索时间、检索准确率和算法可扩展性方面与单层哈希索引表进行了对比实验。实验证明，在对海量数据进行检索时，本文提出的方法拥有良好的可扩展性，并且具有较高的检索准确率和较快的检索速度。

关键词：相似性检索；近似最近邻检索；位置敏感哈希；空间球面网格；分布式索引

ABSTRACT

Recently, information retrieval based on data similarity has become a hot research direction. Information retrieval based on data similarity is implemented in many field of computer sciences. Using data similarity to retrieve information can be divide into two classes. One is nearest neighbor search, the other is approximate nearest neighbor search. Locality sensitive hashing is an approximate nearest neighbor search algorithm based on probability theory and Euclidean geometry. Yet there is some drawbacks in locality sensitive hashing. For example, locality sensitive hashing has low filtering effect to long distance data point, besides locality sensitive hashing needs to build lots of indexing table to ensure the accuracy rate of the algorithm. To construct better algorithm to solve the similarity search problem, this article present two-level locality sensitive search based on intrinsic locality sensitive hashing. The main research contents are as following:

Studying the intrinsic Locality Sensitive Hashing. When the retrieval point is far from data point, the filtering effect of intrinsic locality sensitive hashing is not good enough. To solve this problem, we used space sphere lattice to partition the data set to improve the filtering effect of the algorithm to long distance point. We scaled the projection segment to lower the probability of short distance point being projected to neighbor segment. Calculated the collision probability of Locality Sensitive Hashing function. Calculate the upper bound of indices ρ. It turn out that Two-Level locality sensitive hashing function is better than the origin Locality Sensitive Hashing. Used MATLAB to do the contrast experiment on the GIST descriptor of CIFAR-10 dataset. Compared the recall rate and precision rate with the existing locality sensitive hashing algorithm.

Improved the distribution indexing table in allusion to the two-level locality sensitive hashing. Two-level distributed indexing table using the characteristic of two-level locality sensitive hashing algorithm that two-level locality sensitive hashing took two step to hashing the data point, design the distribution indexing table into two-levels. Two-level distribution indexing table improved the accuracy rate of locality sensitive algorithm and reduced the rate of in memory occupation. This content used Sougou lab's data set of 2012 version to test the two-level locality sensitive hashing. This content compared two-level locality sensitive hashing with other similarity search algorithm in aspect of retrieval performance. Compared the two-level locality sensitive hashing algorithm used two-level distribution indexing table with algorithm used single level distribution indexing table in aspect of

retrieval time, retrieval accuracy, scale performance. It finds out that the method used in this content have good precision rate and speed rate.

Keywords: similarity search, approximate nearest neighbor search, locality sensitive hash, space sphere lattice, distributed indexing

目　录

第 1 章 绪　论

1.1 课题研究背景及意义

随着信息时代的到来，互联网逐渐形成一个影响巨大的新型媒介系统。互联网上产生了海量的多媒体信息，人们对海量数据进行检索的需求也逐渐增加，因而面向海量数据如何进行快速有效的检索成为了研究热点。

从海量的数据中找到一个或多个与某个数据最相似的数据，在数据量少，数据维度较低时可以采用线性检索的方式进行检索，即对待查找数据与数据库数据进行一一匹配。然而，当数据量大，数据维度高时利用线性检索的方式检索效率很低。为了提高对海量数据的检索效率，研究人员通过建立数据集索引的方法来加快检索过程，通常称这类技术为最近邻检索技术或者近似最近邻检索技术[1]。常见的近邻检索技术有：SR-tree[2]，M-tree[3]，DBM-tree[4]，cover-tree[5]。而位置敏感哈希算法是一种近似最邻近检索技术。

哈希算法通过建立哈希索引表的方式提高索引效率。哈希算法的关键在于选取一个哈希函数，哈希函数将数据集中的数据映射到对应的哈希桶中。当对数据集中的数据进行哈希计算时，不同的数据有可能被映射到相同的哈希桶中，这种情况称为冲突或碰撞。传统的哈希算法通过对数据进行再次哈希，将数据映射到其他空桶中来解决问题。位置敏感哈希算法则是通过位置敏感哈希函数将数据集中距离近的数据点映射到相同的哈希桶中，使距离近的数据点发生碰撞。

位置敏感哈希算法的本质是将原始的数据集分成多个元素个数较少的数据子集，每个数据子集中的数据之间的距离较小，通过检索数据子集来获得近邻数据。位置敏感哈希算法将一个在大的数据集内检索距离近的元素的问题转化为在一个小的子数据集内检索距离近元素的问题。位置敏感哈希算法被应用于多个领域：

（1）**近似重复检测**　　信息时代的到来使得互联网上充斥着大量的近似、重复信息。对于互联网使用者来说，重复的信息是毫无意义的。过多的近似重复信息会影响互联网使用者工作效率，在处理这些信息时会造成存储和资源浪费。位置敏感哈希算法可以有效识别互联网上这些重复的信息。近似重复检测在搜索引擎技术上得到较好的应用[6,7]。

（2）**数据聚类**　　对数据集进行聚类是指将数据集中相似的数据点进行归

类。位置敏感哈希算法可以将数据集中相似度高的数据点归并到相同子集中。因此可以将位置敏感哈希算法应用于数据聚类。有研究学者将位置敏感哈希算法应用于分层聚类，并通过实验验证了位置敏感哈希算法在对数据进行分层聚类时，可以有效提高数据聚类的效率[8,9]。

（3）**图片相似性识别**　　图片相似性识别是一项在大规模图片数据集中检索相似的图片的一项技术。图片相似性是指通过检索图片的内容而不是图片的元数据，例如关键字，标签，文本或描述来检索图片。在图片相似性识别中，需要对图片的特征库进行检索。利用位置敏感哈希算法可以快速判断图片的特征向量之间的相似度。微软的 Bing 搜索引擎当前已经实现了针对图片内容的图片相似性检索[10,11]。

（4）**数字视频指纹技术**　　视频指纹技术通过提取、压缩视频的特征，形成数字视频独一无二的“视频指纹”。视频指纹技术已被证明可以有效的区分和比较数字视频数据。位置敏感哈希算法可以用于视频指纹的分析，通过对数字视频指纹进行位置敏感哈希计算可以快速有效的判断数字视频指纹之间的相似度[12]。

（5）**音频指纹技术**　　音频指纹是一种浓缩的数字摘要，可用于识别音频样本或在音频数据库中快速地找到类似的音频数据。音频指纹可用于具体音乐作品的版权维护。位置敏感哈希算法可以用于快速识别不同音频指纹之间的相似度[13]。

通过位置敏感哈希函数能够构造一个或多个哈希索引表，每个表内的哈希桶中的数据之间是近邻的可能性很大。位置敏感哈希函数的目标是，距离近的数据点经过位置敏感哈希函数计算后，被映射到相同的哈希桶中，而距离远的数据点经过位置敏感哈希函数计算后，被映射到不同的哈希桶中。如果距离近的数据点被投影到了不同的哈希桶内，称之为漏判（False Negative），如果距离远的数据点被投影到相同的哈希桶内，称之为误判（False Positive）。在使用位置敏感哈希算法时，应该使漏判和误判发生的概率尽可能的小。

随着近似近邻性检索技术的发展，研究人员对位置敏感哈希算法提出了一些改进方法，主要有两个途径：一是在哈希索引表中尽可能多的使用位置敏感哈希函数，目的是降低误判发生的概率；二是建立多个哈希索引表，多个哈希索引表可以降低漏判发生的概率。但是这两种途径都会使得哈希索引表占用更多的空间，而且这两种途径并没有从本质上对位置敏感哈希算法进行改进。

降低位置敏感哈希算法在使用过程中误判和漏判发生的概率更为有效的方法是构造一个新的位置敏感哈希函数，相比原有的位置敏感哈希函数，新的位

置敏感哈希函数使得相邻的数据点落到同一个哈希桶中的概率比原有的位置敏感哈希函数更高，不相邻的数据落到相同的哈希桶中的概率比原有的位置敏感哈希函数更低。因此，通过研究更好的位置敏感哈希函数来提高位置敏感哈希算法的性能十分重要。

1.2 海量数据检索国内外研究现状及分析

1.2.1 国外研究现状

最早提出近似最邻近[14]的概念的是 Wilfong 等人。在文献中，他们证明数据容量与近似最邻近的检索时间呈亚线性关系，并提出来一种新的基于哈希索引的思想实现的近似最近邻检索方法，代替以往的基于树型结构的空间划分的方法，并取名为位置敏感哈希（Locality Sensitive Hashing，LSH）。位置敏感哈希算法的核心思想是设计与对应的空间度量相关性强的哈希函数来映射数据点，空间距离较近的点被映射到同一个桶中的概率较高，空间距离较远的点被映射到同一个桶中的概率较低；检索时，哈希函数将待检索的数据点映射到对应的哈希桶中，哈希桶中的点即为被检索到的近邻数据点。与传统的数据结构相比，这种算法的效率更高。

1999 年，Gionis 等人在其论文中提出了汉明空间中的位置敏感哈希算法，用于解决高维数据集中的近似最近邻问题[15]。Gionis 等人将汉明空间中的位置敏感哈希算法与基于树型结构空间划分的检索算法中性能较好的 SR-tree 做比较，结果是在允许少量误差和额外的存储空间的情况下，检索效率得到了很大的提高。Gionis 等人通过实验证明位置敏感哈希函数在高维数和大规模数据下的扩展性很好，并且位置敏感哈希函数的参数可以提前计算，相比于传统的检索算法，位置敏感哈希算法的检索效率有很大提高。

2004 年，Datar 等人提出了基于 P-稳态分布的位置敏感哈希算法[16]，使得位置敏感哈希算法在 P 范数空间中得到了很好的应用。在文献中，作者先是通过理论证明了这种哈希函数的高效性，随后又用实验加以验证。位置敏感哈希算法被成功应用在图像检索，文本分类等领域。

2006 年，Andoni 等人提出了一种近似最优的哈希函数算法[17]，采用超立方体网格代替原始的一维随机映射，使得位置敏感哈希算法在欧式空间的误差达到均方级。位置敏感哈希算法是近似最邻近检索算法的一种，它仅仅是保证返回最邻近数据节点的概率较高，但不能 100%确定返回的一定是最邻近的数据点。通常人们的做法是生成多个位置敏感哈希函数并构建多个哈希索引表。因

此，为了保证高准确率，位置敏感哈希算法需要很大的存储空间，算法的时间复杂度随之增加。

2008 年 Andoni 等人提出了对于高维度数据集的 EMD(Earth Mover Distance)距离 [18]。Andoni 等人通过对数据集进行空间变换，降低原有的数据集的维度来提高算法的检索效率。Andoni 等人在理论上证明了其提出的算法在高维空间的大规模数据集检索中效率较高。

2014 年，Andoni 和 Indyk 等人提出了优化的数据依赖哈希算法[19]。通过使用数据依赖哈希，实现了比经典的位置敏感哈希算法更高的性能。优化的数据依赖位置敏感哈希算法提高了在汉明空间和欧几里德空间中位置敏感哈希算法的性能。与经典的位置敏感哈希算法相比，数据依赖位置敏感哈希算法，在相同的条件下检索准确度比以往的位置敏感哈希算有很大的提升。

Wilfong 等人提出位置敏感哈希算法之后，位置敏感哈希算法在多个领域得到了应用，例如音频检索、社交群发现、图片检索、文本分类等计算机领域。虽然位置敏感哈希算法能对高维数据空间中的检索速度大幅提高，但是它的应用范围局限于汉明空间，当数据点在欧式空间或者其他范数空间时，则需要将数据点映射到汉明空间，限制了算法的准确率和效率[20]。

1.2.2 国内研究现状

从事海量数据检索领域研究的北京邮电大学的王镪等人提出了基于熵的位置敏感哈希算法[21]。王镪等人考虑到实际场景中的数据集分布不均匀，在应用 P-稳态分布位置敏感哈希算法的过程中，基于概率理论的哈希函数在哈希后容易出现有的哈希桶中数据密集而有的哈希桶中数据稀疏的情况。王镪等人通过对初始映射值进行排序，然后为每个哈希桶设置数据点上限来提高算法的检索性能。

南京大学从事信息检索研究的武港山等人提出了一种混合型索引结构，在 HK 树索引的上层，利用层次化的 K-means 树对数据集进行初步划分；然后对 K-means 树划分得到的数据子集上创建 FBLSH 索引，进行进一步过滤。通过空间划分与空间映射相结合，利用两者之间的优点使得近似最近邻检索任务能够更高效的被完成[22]。

复旦大学的袁培森等人针对 Web 数据的海量性及高维等特点，使用位置敏感哈希技术分别在大规模 XML 结构相似性查询、字符串相似性连接和高维数据相似性查询方面进行了研究和算法实现，并在真实数据集上验证了其提出方法的性能效果和可扩展性[23]。

中山大学的甘骏豪等人通过对位置敏感哈希算法增加一个多层次的动态索引树来解决位置敏感哈希算法为了保存碰撞计数器而导致位置敏感哈希算法在常数内存内运行受限的问题。并在真实数据集上验证，证明了其提出的方法对原有的位置敏感哈希算法有所提升[24]。

哈尔滨工程大学的何学文等人通将 KD-tree 与位置敏感哈希算法相结合的方式，对语音文档主题进行分类。在实际应用过程中，何学文等人对算法参数进行了优化，增加 TF-IDF 权重和后验概率下的 TF-IDF 权重，建立语音文档的空间向量模型，达到了很好的效果[25]。

大连理工大学的周建辉等人在对图像检索方法进行研究的过程中，提出了一种平衡半监督的位置敏感哈希算法，通过将图片分成若干小图片，缩短位置敏感哈希算法的哈希编码，并对每张小图片的哈希编码进行加权运算，提高了算法的检索准确度[26]。

浙江大学的林跃等人，研究了密度敏感哈希算法和压缩哈希算法，密度敏感哈希算法在位置敏感哈希算法的基础上，进一步分析数据集的拓扑结构。相比位置敏感哈希，密度敏感哈希算法对数据集的针对性更高。密度敏感哈希算法是将压缩感知和稀疏编码方面理论结合得出的结果。林跃等人通过实验表明其提出的算法比主流的算法有所提高[27]。

国外的研究人员经过长期研究和努力，利用数据相似性进行海量数据检索的技术上领先国内很多，提出了许多性能更好的，更为通用的海量数据检索技术。国内的研究人员在不断的努力之下，针对具体应用对位置敏感哈希算法进行改进，也取得了不错的成绩。经过国内外研究人员的长期研究和努力，对海量数据进行相似性检索的技术得到了很大的发展。但是现有的解决相似性检索问题的算法还存在一定的局限性，研究人员对算法的改进多是针对算法的实际应用场景进行优化，并没有在本质的数据相似性检索问题的解决方法上提出改进。原有的位置敏感哈希算法的不足，例如对数据集中距离远的数据点的过滤效果较差，索引表建立效率较低并没有得到改善。

1.3 本文主要研究内容

正如前面所述，许多国内外的学者在算法的参数，哈希函数的参数选择上进行改进，但是依然难以适应当前的需求。本文在分析位置敏感哈希技术原理的基础上，研究了两层位置敏感哈希算法。

基于上述的研究思路，本文做了如下工作：

（1）分析了近似最邻近检索问题和位置敏感哈希算法，比较位置敏感哈希

方法与传统的哈希方法的不同之处，给出了位置敏感哈希函数的定义。分析了位置敏感哈希算法的框架。

（2）通过分析 P-稳态分布位随机向量投影和空间球面网格，提出了两层位置敏感哈希函数。计算两层位置敏感哈希函数的碰撞概率，从理论上证明两层位置敏感哈希算法的有效性。

（3)在数据集上进行实验，分析两层位置敏感哈希算法的准确率，召回率，运行时间并与其他的解决近似最近邻检索问题的算法进行比较。

（4）分析两层位置敏感哈希索引表特点，提出采用分布式框架对数据集进行处理，设计分布式索引表。在分布式平台上对基于 P-稳态分布和空间球面网格的位置敏感哈希算法进行可扩展性实验和检索效率实验。

1.4 本文的结构

第 1 章介绍对本课题的研究背景及研究意义。介绍当前海量数据相似性检索的国内外现状。

第 2 章研究位置敏感哈希算法，比较位置敏感哈希函数与传统的哈希的不同，给出了位置敏感哈希函数的定义，介绍了数据集的距离度量，并分析了位置敏感哈希的算法框架。

第 3 章针对经典位置敏感哈希函数的不足，研究两层位置敏感哈希函数。利用空间球面网格和 P-稳态分布的随机向量投影设计两层位置敏感哈希函数。通过实验验证使用了两层位置敏感哈希函数的位置敏感哈希算法的准确率和召回率。

第 4 章分析了位置敏感哈希索引表特点，提出分布式索引表的结构。采用 MapReduce 分布式编程框架编程实现分布式哈希索引表的建立和查询。在分布式平台上进行基于空间球面网格和 P-稳态分布的位置敏感哈希算法的相关实验。

第 2 章 位置敏感哈希算法

2.1 引言

位置敏感哈希算法用于解决如何快速的从大规模的高维度的数据集中找到与某个数据最相似的若干个数据的问题。当数据集较小，数据集的维度较低时，可以通过线性检索的方法找到数据集中与某个数据最相似的一个或多个数据。但是当数据集中包含的数据量大，数据维度较高时，采用线性检索方法会非常耗时。因此，为了降低对高维度的大数据集进行检索的时间，可以通过对数据集建立索引来提高检索速度。研究学者们将这类方法称作最近邻检索（Nearest Neighbor Search）或者近似最邻近检索（Approximate Nearest Neighbor Search）。位置敏感哈希算法是近似最邻近检索方法中的一种。本文中的位置敏感哈希函数指位置敏感哈希算法中使用的哈希函数。

2.2 传统哈希与位置敏感哈希比较

在对数据集进行检索的过程中，可以通过对数据集建立哈希索引表的方式来提高检索效率。如图 2-1 所示，传统的哈希方法将原始的数据点通过哈希函数映射到相应的哈希桶内。在对数据点进行哈希的过程中，会发生不同的数据点被映射到同一个哈希桶中的情况。当出现这种情况时，一般通过再次将数据点映射到其他空的哈希桶内或者在哈希桶后设置链表存储碰撞的数据点来解决，这是传统的哈希方法。

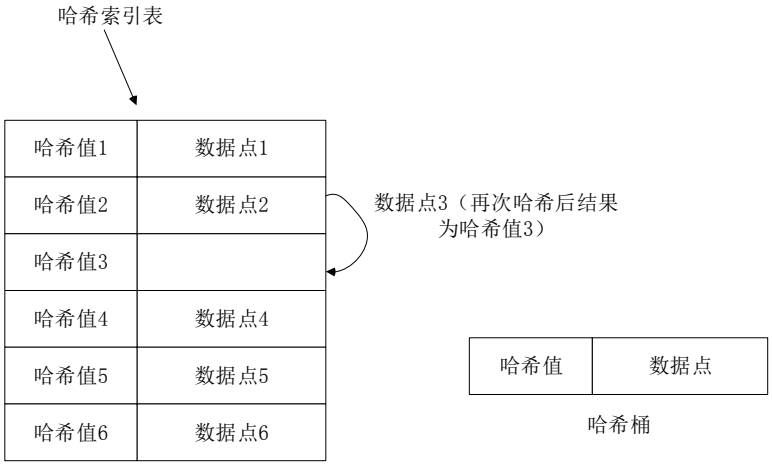

图 2-1 哈希索引表 1

在传统的哈希方法中，选取的哈希函数应该使数据尽可能的哈希成不同的哈希值，减少数据经过哈希后发生碰撞的情况。如公式(2-1)所示，经典的哈希函数就是对数据进行求模。

$$h = x \bmod w \tag{2-1}$$

式中 x——表示数据；

w——为正整数；

h——为哈希值。

位置敏感哈希的基本思想是，将原始的空间中的两个距离近的数据点通过相同的映射变换后，这两个数据在新的空间中依然距离较近的概率很大，而距离远的数据点经过映射变换后距离近的概率很小。也就是说，在对数据集中的数据点进哈希映射后，原先相邻的两个数据点将有较大的概率被哈希到相同的哈希桶内，具有相同的哈希值。当然，也存在原始空间中不相邻的数据点被映射到同一个哈希桶内，不相邻的数据点被映射到同一个哈希桶内的概率较小。如图 2-2 所示，数据集中的数据点经过位置敏感哈希后，将得到一个哈希索引表，原始的数据集被分散到哈希索引表中的哈希桶内，每一个哈希桶中都有若干个数据点。属于同一个哈希桶内的数据就有较大的概率是相邻的。

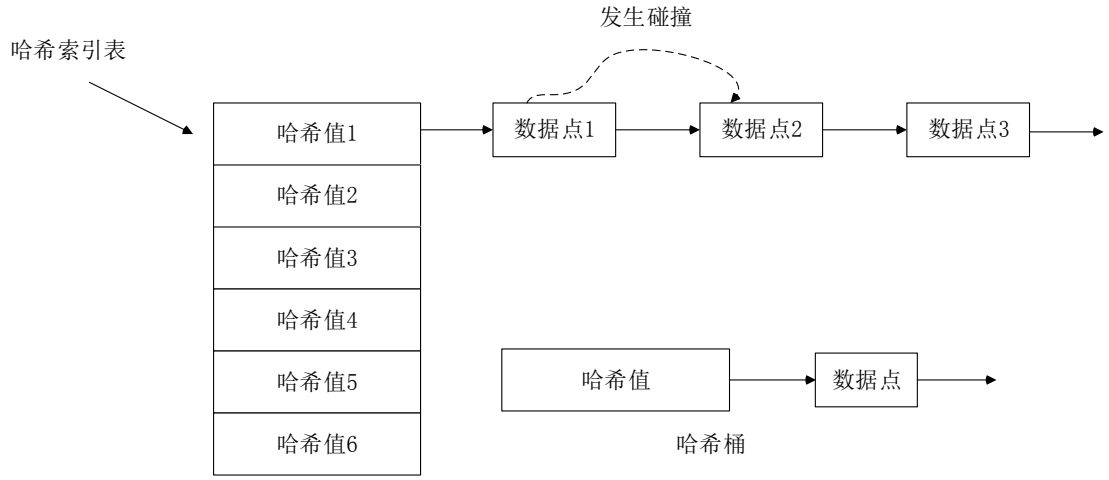

图 2-2 哈希索引表 2

因此，如果可以找到一些特定的哈希函数，使得数据经过哈希映射变换后，原始数据空间中距离近的数据被哈希到同一个哈希桶中，那么在对数据集进行近似最邻近检索时将变得简单。可以将待检索数据进行哈希映射，得到待检索数据的哈希桶编号，然后取出该哈希桶编号对应的哈希桶内的所有的数据，再进行线性检索。也就是说，通过哈希映射，将规模较大的数据集分成多个数据

子集，而每个数据子集中的数据点都是距离相近的且该数据子集中的数据点个数较少。如此，可以将一个在超大数据集内检索邻近数据的问题变成在一个较小的数据子集内检索邻近数据的问题。

通过比较可以发现，位置敏感哈希不同于传统的哈希。传统的哈希在经过哈希映射后发生碰撞的不同数据点之间并无直接联系，而且为了使检索效率更高，所采用的哈希函数应该使数据集中的数据点在经过哈希映射后尽可能的不发生碰撞。而位置敏感哈希的目的则是使数据集中邻近的数据点在经过哈希映射后被映射到相同的哈希桶中。

2.3 位置敏感哈希函数

2.3.1 位置敏感哈希函数定义

如前文所述，不同于传统的哈希函数，位置敏感哈希函数应该使得在原始数据空间中相邻的两个数据点经过哈希后被映射到相同的哈希桶中。因此，位置敏感哈希函数应该满足以下的两个条件：

（1）如果 $d(x,y)\le d_1$，则 $h(x)=h(y)$ 的概率至少为 p_1；

（2）如果 $d(x,y)\ge d_2$，则 $h(x)=h(y)$ 的概率至多为 p_2。

其中 $d(x,y)$ 表示数据点 x 和数据点 y 之间的距离，$h(x)$ 和 $h(y)$ 分别表示对数据点 x 和数据点 y 进行哈希映射，条件中 $d_1\le d_2$ 且 $p_1\ge p_2$。

满足上述两个条件的哈希函数称为 (d_1,d_2,p_1,p_2)-敏感的哈希函数。将通过一个或多个 (d_1,d_2,p_1,p_2)-敏感的哈希函数对原始数据集进行哈希生成一个或多个哈希索引表的过程称为位置敏感哈希。值得注意的是，并不是所有的距离度量都能够找到满足上述两个条件的位置敏感哈希函数。

位置敏感哈希函数可以将数据集中的数据点映射到不同的哈希桶中，每个哈希桶内的数据之间是近邻数据的概率很大。位置敏感哈希函数的目的是为了对数据点进行哈希计算，将近邻的数据映射到相同的哈希桶中，而不相邻的数据映射到不同的哈希桶中。

在位置敏感哈希函数对数据进行哈希时，也有一定的概率使得近邻的数据点被映射到不同的哈希桶中，这种情况称之为漏判。同理，当不近邻的数据点被映射到相同的哈希桶中时，我们称这种情况为误判。对于一个好的位置敏感哈希函数，它的漏判和误判发生的概率应该尽可能的小。

2.3.2 增强位置敏感哈希的方法

对位置敏感哈希函数进行增强，可以降低漏判和误判发生的概率，减少哈希索引表的存储空间。增强位置敏感哈希的方法有很多种，下述的 7 种方法中，前 4 种用于降低漏判和误判 发生的概率，后 3 种用于减少哈希索引表的存储空间。

（1）**设置多个独立的哈希索引表**　　利用多个位置敏感哈希函数创建哈希索引表。每次创建哈希索引表时，选取 k 个哈希函数，重复多次，即可创建多个哈希索引表。多个哈希索引表可以降低漏判发生的概率[28]。

（2）**AND 操作**　　选取 k 个相同类型的位置敏感哈希函数，只有当两个数据点经过 k 个位置敏感哈希函数计算后都发生碰撞，才把两个数据点映射到相同的哈希桶中。只要有一个 位置敏感哈希函数使得两个数据点不发生碰撞，两个数据点就不会被投影到相同的哈希桶中。AND 操作降低漏判发生的概率[29]。

（3）**OR 操作**　　选取 k 个相同类型的位置敏感哈希函数，两个数据经过 k 个位置敏感哈希函数计算，只要有一个位置敏感哈希函数使得两个数据点发生碰撞，就把两个数据点投影到相同的哈希桶中。OR 操作使得近邻数据点发生碰撞的概率提高，而不近邻的数据点发生碰撞的概率保持较小，降低误判发生的概率[30]。

（4）**AND 操作和 OR 操作级联**　　选取多组 k 个相同类型的位置敏感哈希函数，每组内的位置敏感哈希函数采用 OR 操作，每组间的位置敏感哈希函数采用 AND 的操作。AND 操作和 **O**R 操作级联可以同时降低漏判和误判发生的概率[31]。

（5）**求模运算**　　将位置敏感哈希函数得到的哈希值进行求模运算，这样可以使得原先的稀疏的哈希值变的密集，减少哈希索引表中空哈希桶的数量。对位置敏感哈希函数得到的哈希值进行求模运算本质是调整哈希桶存放的位置，不会对哈希桶中的数据造成影响[32]。

（6）**线性变换**　　选取 k 个位置敏感哈希函数，通过 k 个位置敏感哈希函数得到 k 个哈希值，假设为 $h_1,h_2,...,h_k$。新的哈希值可以通过线性变换得到：

$$h_{new}=h_1\cdot r_1+h_2\cdot r_2+...+h_k\cdot r_k \tag{2-2}$$

式中 $r_1,r_2,...r_k$ ——为 k 个随机数。

线性变换同样是调整哈希桶的位置，对哈希桶中的数据不造成影响[33]。

（7）**XOR 异或**　　选取 k 个位置敏感哈希函数，通过 k 个位置敏感哈希函数得到 k 个哈希值，假设为 $h_1,h_2,...,h_k$。新的哈希值可以通过异或运算得到：

$$h_{new} = h_1 \otimes h_2 \otimes ... \otimes h_k \tag{2-3}$$

式中 ⊗——表示异或操作。

同样的，对哈希值进行异或操作将使得哈希索引表中的哈希桶排列更为合理，且不会改变哈希桶中的数据[34]。

2.3.3 距离度量

对于同一个数据集，可以通过定义不同的距离度量来表示数据点之间的距离，从而确定数据点与数据点之间的距离远近。距离度量，也可被称为距离函数，用来定义数据点集中的任意两点的距离的函数。在该小节中，将介绍几种常用的距离度量，并给这些距离度量对应的位置敏感函数。

（1）**欧几里德距离度量** 数据集中两个数据点 x，y 之间的欧几里德距离是连接两个数据点的直线段长度。在卡笛尔坐标系中，如果数据点 $\boldsymbol{x}=(x_1,x_2,\ldots,x_n)$ 和数据点 $\boldsymbol{y}=(y_1,y_2,\ldots,y_n)$ 是数据空间中的两个数据点，则数据点 $\boldsymbol{y}$ 到数据点 $\boldsymbol{x}$ 之间的距离可以用下述公式求得：

$$d(\boldsymbol{x},\boldsymbol{y}) = \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\ldots+(x_n-y_n)^2} = \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2} \tag{2-4}$$

式中 n——表示空间维度；

x_i——表示数据点对应的向量 $\boldsymbol{x}$ 在第 i 个维度上的值；

y_i——表示数据点对应的向量 $\boldsymbol{y}$ 在第 i 个维度上的值[35]。

数据集定义了欧几里德距离度量以后，在原始的数据空间中，位置敏感哈希函数可以是[36]：

$$h_{Euclid}(\boldsymbol{x}) = \left\lfloor \frac{\boldsymbol{x}\cdot\boldsymbol{v}+b}{r} \right\rfloor \tag{2-5}$$

式中 $\boldsymbol{x}$——表示数据点对应的向量；

$\boldsymbol{v}$——n 维向量，每个维度的值都由服从 N(0,1)正态分布的随机变量产生；

b——是随机实数；

r——为正实数。

（2）**汉明距离度量** 对于任意的两个长度为 n 的二进制数 x 和 y，他们之间的汉明距离可以用这两个二进制数 x 和 y 表示。可以用这两个二进制数之间的不同位数衡量他们之间的汉明距离，即：

$$H(x,y) = \sum_{i=1}^{n}(x_i \neq y_i) \tag{2-6}$$

式中 x_i——表示 x 的第 i 位数值；

y_i——表示 y 的第 i 位的数值。

数据集定义了汉明空间以后，在原始的数据空间中，位置敏感哈希函数可以是：

$$h_{ham}(\boldsymbol{x}) = x_i \tag{2-7}$$

式中 $\boldsymbol{x}$——表示数据点对应的向量；

x_i——表示数据点对应的向量 $\boldsymbol{x}$ 的第 i 位上的值[15]。

（3）**Jaccard 距离度量** Jaccard 相似度：对于给定的两个集合 A 和 B，可以用 Jaccard 相似度衡量集合之间的相似度，公式描述如下：

$$Jaccard(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{2-8}$$

式中 $0 \le J(A,B) \le 1$。

Jaccard 相似度也被称为 Jaccard 相关系数，是用来比较两个集合之间的相似度和多样性。Jaccard 相似度衡量有限集合之间的相似度。

Jaccard 距离度量衡量了两个集合的不相似程度，Jaccard 距离的值可以通过 1 减去 Jaccard 相似度计算，或者等于两个集合的并减去两个集合相交得到的结果再除以两个集合的并，用数学语言描述如下：

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \tag{2-9}$$

一个直观的解释是，集合 A 和集合 B 之间的 Jaccard 距离等于集合 A 和集合 B 中仅在集合 A 或集合 B 中存在的元素占集合 A 和集合 B 所包含的所有的元素的比例。Jaccard 距离用来度量有限集合之间的距离[37]。数据集定义了 Jaccard 距离度量以后，在原始的数据空间中，位置敏感哈希函数可以为：

$$h_{Jaccard}(A) = \min\left[h(x)\right] \tag{2-10}$$

式中 $h(x)$——是一个具有很好的均匀性，能够把不同元素映射成不同整数的哈希函数；

x——是集合 A 中的元素[38]。

（4）**Cosine 距离度量** Cosine 距离度量利用数据点对应的空间中向量之间的夹角来定义数据点的之间的，夹角越小，数据点之间的距离越小，夹角越大数据点之间的距离越远。Cosine 距离度量用数学语言描述如下：

$$\cos(\theta) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{|\boldsymbol{x}||\boldsymbol{y}|} \tag{2-11}$$

式中，向量 $\boldsymbol{x}$ 表示数据集中与某一个数据点相对应的某一个向量，向量 $\boldsymbol{y}$ 表示数据中与某一个数据点对应的向量，θ 表示向量 $\boldsymbol{x}$ 与向量 $\boldsymbol{y}$ 之间的夹角。$|\boldsymbol{x}|$ 表示向量 $\boldsymbol{x}$ 的模，$|\boldsymbol{y}|$ 表示向量 $\boldsymbol{y}$ 的模[39]。数据集定义了 Cosine 距离度量以后，原始的数据空间中的位置敏感哈希函数可以为：

$$h_{\cos ine}(\boldsymbol{x}) = sign(\boldsymbol{x} \cdot \boldsymbol{r}) \tag{2-12}$$

式中 $\boldsymbol{x}$ ——表示数据集中与某一个数据点相对应的某一个向量；

$\boldsymbol{r}$ ——为随机向量；

$sign(\cdot)$——表示正弦函数[40]。

上述的是距离度量是较为常见的四种距离度量，在这四种距离度量之外，还有许多其他的距离度量。欧几里德距离度量在当前的许多度量中使用范围较广，也较为直观，因此，本文研究的是在欧几里德距离度量下的位置敏感哈希算法。

2.4 位置敏感哈希算法框架

位置敏感哈希算法在解决近似最近邻检索问题时，可以分为两个步骤，第一步是建立哈希索引表，第二个步骤是通过哈希索引表对待查询数据点进行检索。在位置敏感哈希算法中，由于位置敏感哈希函数只能让邻近的数据点有较高的概率映射到相同的哈希桶中，因此，存在近邻的数据点被映射到不同的哈希桶中的情况，称之为漏判，相反的当不近邻的数据点被映射到相同的哈希桶中时，可以称之为误判。在位置敏感哈希算法中，为了降低漏判与误判发生的概率，我们设置多个哈希索引表，每个哈希索引表对应多个位置敏感哈希函数。

2.4.1 建立索引

如图 2-3 所示，在位置敏感哈希算法中建立哈希索引表需要四个步骤：首先是选取 $k \times L$ 个位置敏感哈希函数 $h_{ij}(x)$，其中 $i = 1,2,..L$， $j = 1,2,...k$；其次，是利用 $k \times L$ 个位置敏感哈希函数生成 L 个映射函数 $g_i(x)$，其中 $i = 1,2,..L$。然后，通过 L 个映射函数计算数据点的 L 个映射值，最后，通过 L 个映射值将数据点插入 L 张哈希索引表中。

位置敏感哈希算法在建立索引后将生成 L 个哈索引表，第 i 个哈希索引表对应 k 个位置敏感哈希函数 $h_{ij}(x)$，$i = 1,2,...L$ 且 $j = 1,2,...k$。如图 2-4 所示，每个哈希索引表对应一个映射函数 $g_i(x)$，每一个映射函数 $g_i(x)$ 采用如下方式构建：

$g_i(x)=(h_{i1}(x),h_{i2}(x),...,h_{ik}(x))$。即 k 个位置敏感哈希函数 $h_{ij}(x)$，其中 $j=1,2,..k$，构成 k 维的向量，形成 $g_i(x)$ 映射函数，其中 $i=1,2,..L$。

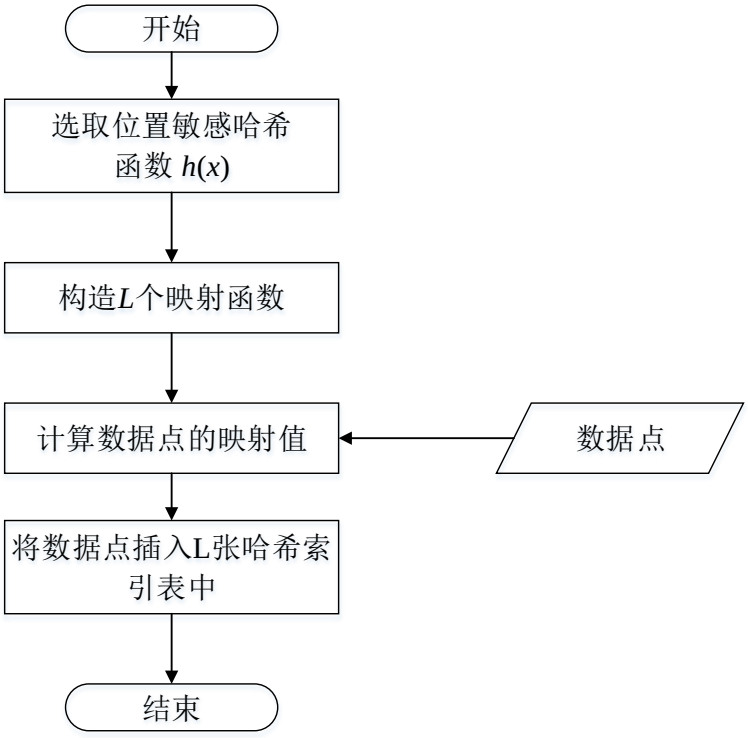

图 2-3 索引表建立流程图

在形成 $g_i(x)$ 映射函数的过程中，可以看出构成 $g_i(x)$ 的 k 个位置敏感哈希函数 $h_{i1}(x),h_{i2}(x),...,h_{ik}(x)$ 都是 AND 关系。在对任意两个数据点进行哈希计算时，只要有一个位置敏感哈希函数得出的值不一致，则这两个数据点不发生碰撞。

数据点经过 $g_i(x)$ 映射函数计算后将得到一个 k 维向量。可以将向量的每一个维度都视为哈希值，通过对 k 个哈希值进行增强操作，例如，对 k 个哈希值进行随机投影或异或操作来获得最终的哈希值。

在哈希索引表的建立过程中，可以只保存非空的哈希桶，这样可以节省存储空间。对于任意的一个数据点 x，经过一个映射函数 $g_i(x)$ 映射，可以把数据点 x 映射成为一个具有 k 个维的空间向量，这个 k 维的空间向量就是数据点 x 的哈希桶编号。对不同的数据点进行映射将形成不同的 k 维向量，存储于不同的哈希桶，从而形成哈希索引表。k 个位置敏感哈希函数 $h_{ij}(x)$ 构造成一个映射

函数 $g_i(x)$ 是为了降低误判发生的概率。位置敏感哈希算法中构造 L 个哈希索引表是为了降低漏判发生的概率。

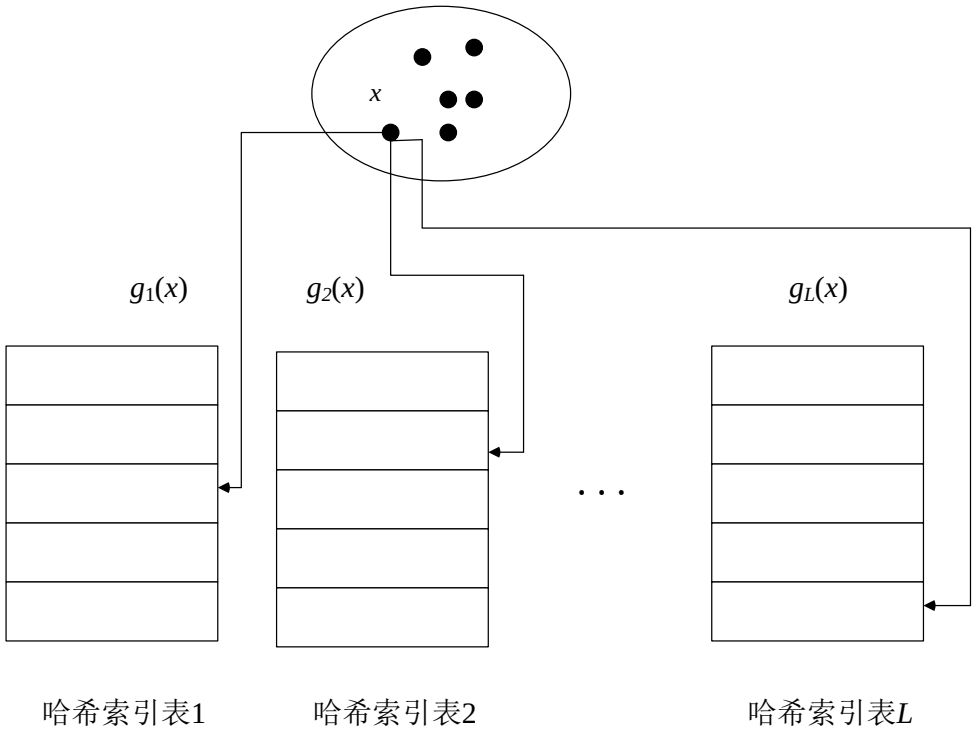

图 2-4 位置敏感哈希索引表建立示意图

2.4.2 检索数据

在高维度空间中的最邻近检索难题是检索效率。如图 2-5 所示，在位置敏感哈希算法中，对数据集进行检索一共分为三个步骤：首先，利用映射函数计算待查询数据点对应的哈希桶编号；其次，将待查询数据点与哈希桶内的数据取出；最后计算与待查询数据点与从哈希桶取出的数据点之间的距离并返回最近邻的数据点。

如图 2-6 所示，在位置敏感哈希算法的检索阶段，对待查询数据点使用 L 个 $g(x)$ 映射函数求出映射值，并把对应的哈希桶中存储的数据点取出，最后计算取出的数据点与待查询数据点之间的距离，返回哈希桶中与待查询数据点距离近的数据点。

由于数据点被哈希映射到那个哈希桶是随机的，因此从对应的哈希桶中返回的数据点的个数并不确定，为了提高位置敏感哈希算法的检索效率，通常当从哈希桶中返回的数据点的个数达到 $3L$ 时，停止返回数据点。

在位置敏感哈希函数中，对于位置敏感哈希函数有一个性能指标参数 ρ。参数 ρ 的定义如公式(2-6)所示。从公式(2-6)中可以看出：当概率 p_1 越大时，表示相邻的数据点被映射到相同的哈希桶中的可能性越大，参数 ρ 越小；当概率 p_2 越小时，表示不相邻的数据点被映射到相同的哈希桶中的可能性越小，参数 ρ 越小。从总体上看，参数 ρ 越小，位置敏感哈希函数的性能要越好。

$$\rho = \frac{\ln\left(1/p_1\right)}{\ln\left(1/p_2\right)} \tag{2-13}$$

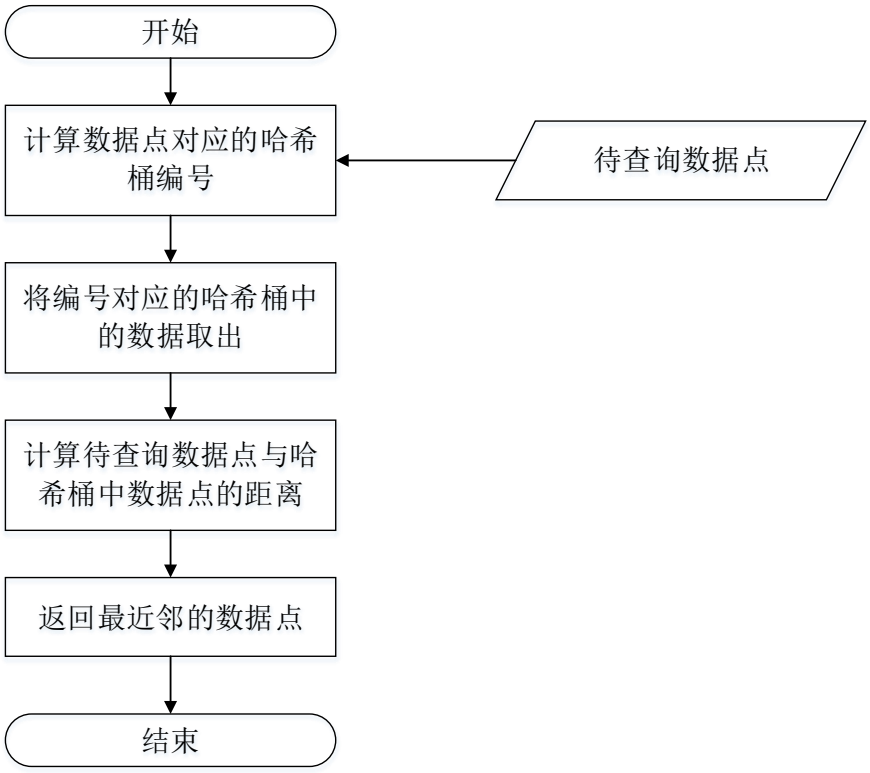

图 2-5 位置敏感哈希算法检索流程图

在这样的位置敏感哈希算法框架下，可以得到位置敏感哈希算法需要 $L \times n \times d$ 个单位空间来存储数据点，同时需要 $n^{1+\rho}$ 个单位空间来存储哈希值，所以算法的空间复杂度为 $O\left(dn + n^{1+\rho}\right)$。因为在位置敏感哈希算法中，进行检索需要先计算数据点点的哈希桶编号，然后再依次计算从哈希桶中取出的数据点与待查询数据点之间的距离，所以位置敏感算法的检索时间依赖于 $O\left(n^{\rho}\right)$ 次的距离计算和 $O\left(n^{\rho}\log_{1/p2}n\right)$ 次的位置敏感哈希函数计算。

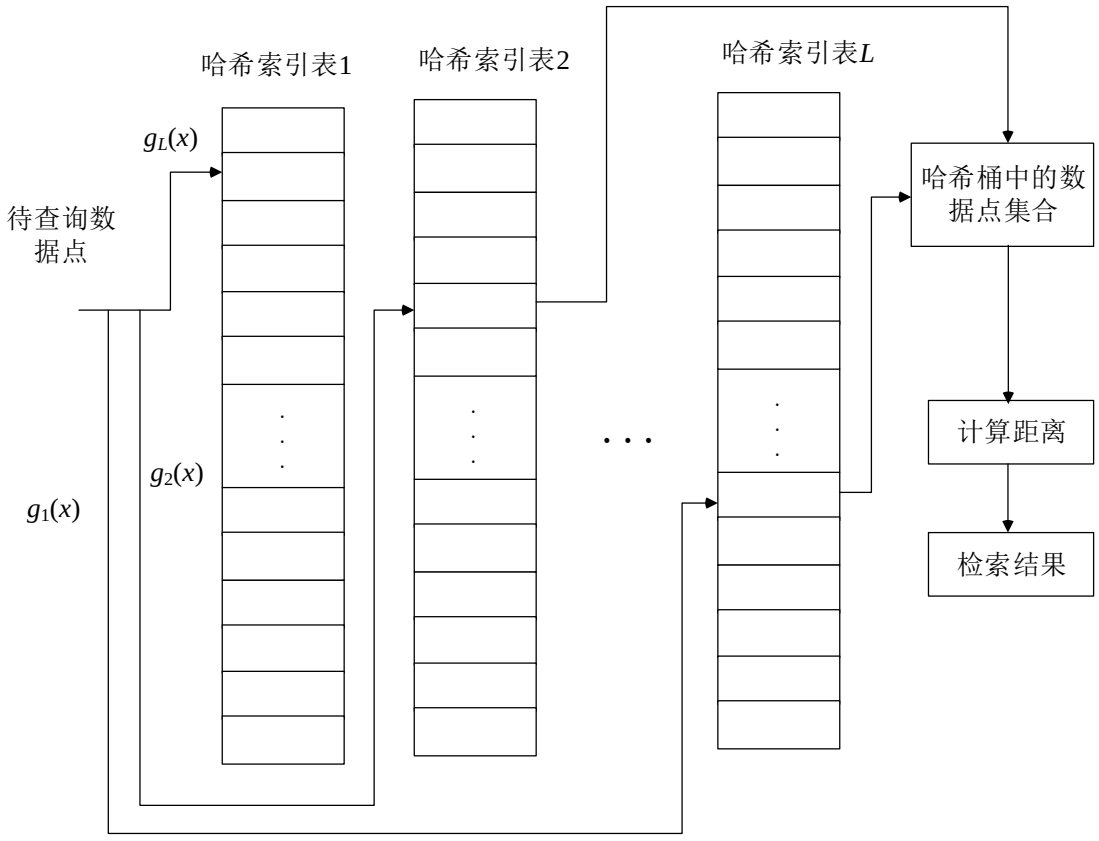

图 2-6 位置敏感哈希算法检索示意图

2.5 本章小结

本章首先介绍了位置敏感哈希并将位置敏感哈希算法与传统的哈希算法进行比较，指出了位置敏感哈希算法与传统的哈希算法在哈希函数选取上的不同。然后，给出了位置敏感哈希函数的定义并介绍了几种常用的距离度量。接着介绍了位置敏感哈希算法的框架，阐述了位置敏感哈希算法建立索引的过程和进行数据检索的过程。

第 3 章 两层位置敏感哈希函数

3.1 引言

本章研究两层位置敏感哈希函数，通过空间球面网格和基于 P-稳态的随机向量投影构造位置敏感哈希函数，计算两层位置敏感哈希函数的碰撞概率。利用数据集测试两层位置敏感哈希算法的准确率及召回率，并将两层位置敏感哈希算法与其他算法进行对比。

位置敏感哈希算法的基本思想是：利用映射的方法，使得数据集中距离近的数据点以较大的概率被哈希到相同的哈希桶中，距离远的数据点被映射为不同的哈希桶中。一个好的位置敏感哈希函数，应该使漏判和误判发生的情况尽可能少，并且可使不相邻的数据点被哈希到相同哈希桶中的概率尽可能的低，即发生的概率也尽可能小。我们使用空间球面网格降低数据集中不邻近的数据点发生误判的概率，然后使用 P-稳态随机向量投影降低漏判发生的概率，最后通过对投影区间进行延伸，降低漏判发生的概率。

3.2 两层位置敏感哈希函数构造

如图 3-1 所示，在研究位置敏感哈希函数的过程中，我们发现，将数据集划分成带有有界直径的子数据集可以使位置敏感哈希函数有效的避免数据空间中距离远的数据点发生误判的情况。

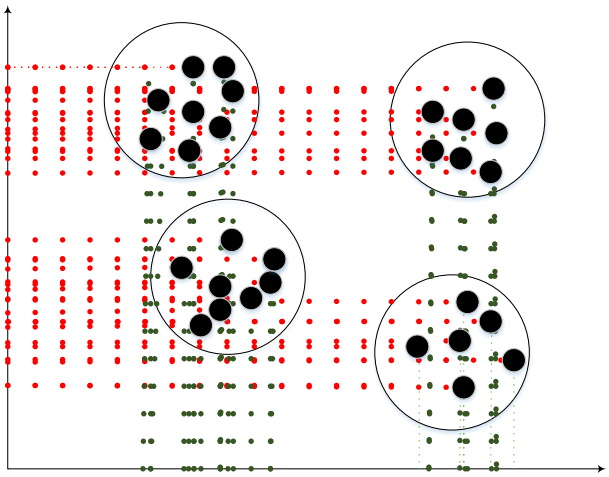

图 3-1 有界子集划分示例图

以二维欧几里德空间为例，将数据集划分成若干个数据子集，约定不同数据子集内的数据点不会被映射成相同的哈希值。如此，可以排除不同数据子集中位置敏感哈希值相同的数据点。

两层位置敏感哈希算法首先使用空间球面网格对数据集进行粗划分，将原数据集划分成若干数据子集。然后再利用随机变量将数据子集投影到随机变量所在的直线上。

3.2.1 空间球面网格划分

对欧几里德空间中的数据集进行球面网格划分最理想的方式是通过一张球体与球体相邻的网格对数据集进行划分。但是在欧几里德空间中，这种理想的网格目前只在 1 维，2 维，3 维，4 维，8 维，24 维的欧几里德空间中发现存在。Leech 网格是 24 维欧几里德空间中的一种球面网格，它使得在其中的球的排列方法，在不重叠的前提下，接近于最稠密。但是将 Leech 网格应用于对欧几里德空间中的数据集进行划分存在着一些不足。例如类似 Leech 的网格目前只被发现在 24 维的欧几里德空间中存在，因此，使用 Leech 网格对欧几里德空间中的数据集进行划分必须先将数据集映射到 24 维欧几里德空间。其次，已知的找出数据点在 Leech 网格中的位置的方法效率并不高。使用 Leech 网格对欧几里德空间中的数据集进行划分反而影响位置敏感哈希函数性能。因此，我们使用多张球体不相邻的空间球面网格对数据集进行划分。

对数据集进行空间球面网格划分一共分为四个步骤。

首先，将数据集通过随机矩阵将数据集从 d 维欧几里德空间映射到 t 维欧几里德空间。选择一个随机矩阵 A，随机矩阵中的每一个元素 A_{ij} 都由服从 N(0,1) 正态分布的随机变量生成的值再乘以伸缩因子 $1/\sqrt{t}$ 得到。将数据集中的每一个数据点乘以随机矩阵即可得到映射到 t 维空间中的数据集:

$$\begin{pmatrix} A_{11} & \cdots & A_{1d} \\ \vdots & & \vdots \\ A_{t1} & \cdots & A_{td} \end{pmatrix} \cdot \left(x_1 \quad \cdots \quad x_d\right)^T = \left(x_1' \quad \cdots \quad x_t'\right)^T \tag{3-1}$$

式中 T——表示矩阵的转置。

其次，生成 t 维空间中的球面网格组。假设 G^t 为一个在 R^t 欧几里德空间中的无限大的球面网格, 网格中的每一个球的半径为 w，且相邻球之间的距离为 $4w\times Z^t$。如图 3-2，图 3-3 所示，二维空间下的空间球面网格球面与球面之间每个球心之间的距离为 $4w$，三维空间下的空间球面网格邻近的空间球面的球心与

球心之间的每个维度上的距离为 $4w$。

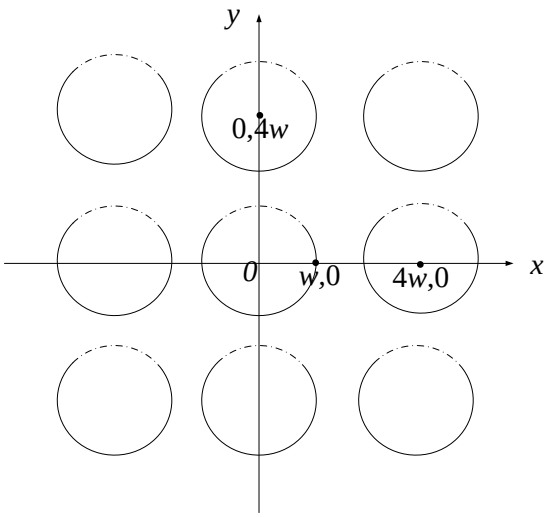

图 3-2 二维空间球面网格示例图

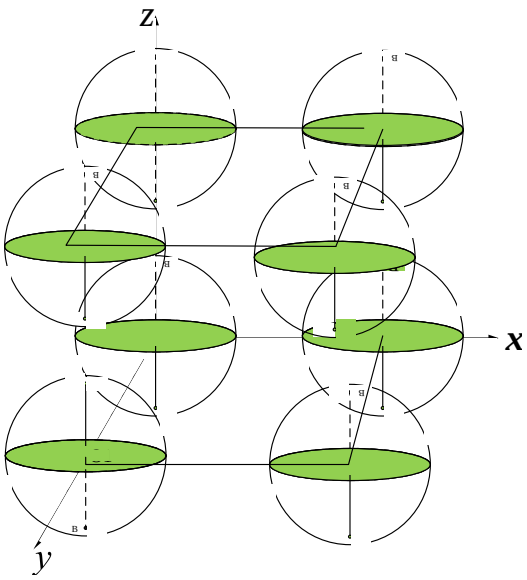

图 3-3 三维空间球面网格

图 3-2 是二维欧几里德空间中的球面网格，图 3-3 是三维欧几里德空间的球面网格。我们约定空间球面网格中任意两个球体的球心距离不小于球体半径的 4 倍。

然后，通过 t 维随机向量对空间球面网格进行 U 次移动，形成 t 维空间球面组。t 维随机向量的每一个维度上的值都由满足 N(0,1)正态分布的随机变量产生。如图 3-4 所示：

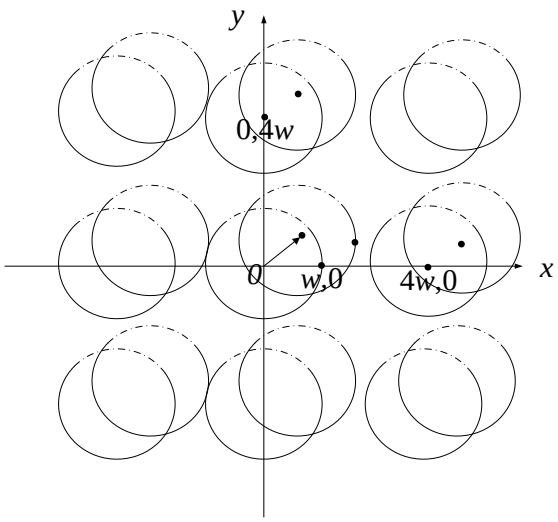

图 3-4 空间球面网格变换示例图

图 3-4 体现的是在二维空间下，初始空间球面网格经过变换后得到的一个空间球面网格。

最后，将包含在 t 维空间球面网格中的单个球面内的数据点划分成为一个数据子集。每个数据点只属于一个空间球面网格，数据点所在的空间球面网格为网格组中最先覆盖数据点的空间球面网格。

在确定数据点所在的空间球面网格后，计算数据点在空间球面网格中的具体的超球面位置。可以通过计算空间球面网格中所有超球面的球心是否包含于以数据点为球心，半径为 w 的球体内。

如图 3-5 所示：

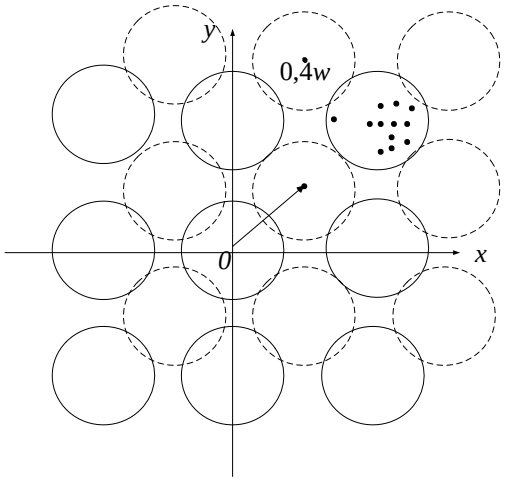

图 3-5 空间球面网格划分后的数据子集示例图

图 3-5 所展示的是经过空间球面划分后的数据子集，数据子集中的数据点都属于最先被实线网格覆盖到。经过划分后，原始的数据集将产生若干类似的数据子集。对数据集进行空间球面网格划分的算法如表 3-1 所示。

表 3-1 空间球面网格划分算法

空间球面网格划分算法

输入：数据点

输出：分区编号

初始化：

（1）对于 $u=1,2,...,U$，选择随机变换 $s_u \in [0,4w]^d$

（2）生成空间球面网格组 $G_u^t = G^t + s_u$

（3）选择随机矩阵 $A \in M_{t,d}$，矩阵中的每个元素 A_{ij} 由伸缩因子 $1/\sqrt{t}$ 乘以服从 N(0,1)正态分布的随机变量生成的值得到

计算划数据点 p 划分后所在的位置：

（1）通过 $p' = Ap$ 将数据点 p 映射到 t 维空间

（2）对于 $u=1,2,...,U$

a）检查 $B(p',w) \cap G_u^t = \phi$ 是否成立，即是否存在 $(x_1,x_2,...,x_t) \in G_u^t$ 使得

$p \in B((x_1,x_2,...,x_t),w)$

b）当找到这样的 $(x_1,x_2,...,x_t)$ 时，停止循环

（3）返回 t 维 0 向量

进行空间球面网格划分的第一个步骤是将数据集从 d 维欧几里德空间映射到 t 维空间，这是为了降低参数 U 的值。我们讨论一下直接在 d 维欧几里德空间直接进行数据划分的情况：

假设 G^d 为一个在 R^d 空间中的无限大的空间球面网格，空间球面网格中每一个球的半径为 w，且相邻球之间的距离为 $\sigma w \cdot Z^d$。对于每一个正数 u，令 G_u^d 表示网格 G^d 在第 u 次移动后的球面网格，即 $G_u^d = G^d + s_u$，其中 $s_u \in [0,\sigma w]^d$。尽可能多的选择 G_u^d，直到空间 R^d 中的点能被网格覆盖。假设需要 U 个网格来覆盖空间中的所有的点。

如图 3-6 所示，二维空间中的空间球面网格组覆盖整个空间的概率等价于空间球面的内接立方体覆盖最接近的四个球心构成的立方体的概率。

通过观察图 3-6 可以推想到在 n 维欧几里德空间中，如果超立方体如果超

立方体$[0,\sigma w]^d$是被覆盖的，则整个空间将被网格组$G_1^d,G_2^d,...,G_{U_d}^d$覆盖。

将超立方体$[0,\sigma w]^d$分割成小的若干个小立方体，并且保证小超立方体被覆盖的概率很高。因此，将超立方体$[0,\sigma w]^d$分割成大小为$\mathrm{w}/\sqrt{\mathrm{d}}\cdot\mathrm{w}/\sqrt{\mathrm{d}}...\mathrm{w}/\sqrt{\mathrm{d}}$的小超立方体。如此的小立方体共有$N=(\sigma w)^d/\left(w/\sqrt{d}\right)^d=\left(\sigma\sqrt{d}\right)^d$个。

假设 x 为小立方体被球面网格G_u^d覆盖的概率，为了使小超立方体被覆盖，则概率 x 等于球体$B\left(0^d+S_u,w\right)$的球心落到小的超立方体中的概率，所以概率为：

$$x\geq\frac{(\sigma w)^d}{\left(w/\sqrt{d}\right)^d}=1/N \tag{3-2}$$

如果x_u表示被任意一个网格覆盖的概率，则$x_u\geq1-(1-x)^U$。可以通过计算至少有一块超立方体没有被覆盖的概率来计算整个大立方体被覆盖的概率。设 P 为整个空间被球面网格覆盖的概率，则：

$$P\geq1-\frac{1}{\left(\sigma\sqrt{d}\right)}\cdot\left(1-\frac{1}{\left(\sigma\sqrt{d}\right)}\right)^U \tag{3-3}$$

式中 d ——表示空间维度；

σ ——表示空间球面网格中影响网格的疏密程度参数；

U ——是空间球面网格数量。

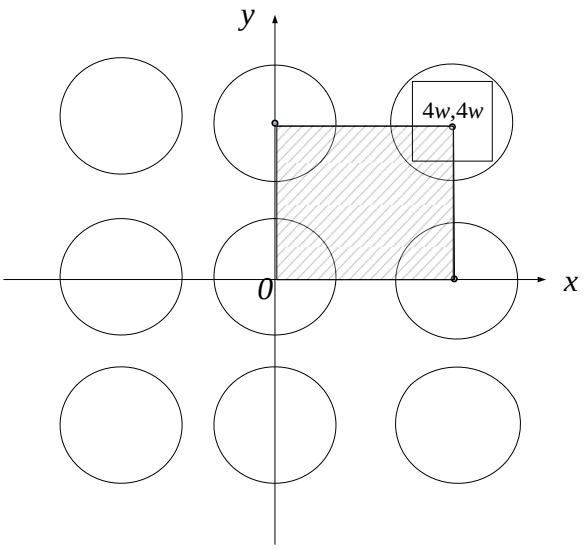

图 3-6 二维空间球面网格示例图

计算结果显示，在 d 维空间下，如果不对数据集进行空间映射，为了保证有较高的概率使得空间球面网格组覆盖整个空间，需要 U 的值更大，即需要有更多的空间球面组。

3.2.2 基于 P-稳态分布的随机向量投影

在对数据集进行空间球面网格划分之后，原有的数据集被划分为若干数据子集，我们对每一个子集分别进行随机向量投影。随机向量投影的目的是将数据子集划分到更小的区间，降低空间中距离较远的数据点在经过位置敏感哈希函数计算后，发生误判的情况。

（1）**P-稳态分布**　　P-稳态分布是一类规范化的概率分布的总和。最经典的 P-稳态分布的例子是正态分布。然而，P-稳态分布涵盖的范围要更广一些，例如柯西分布和 heavy-tailed 分布。P-稳态分布有明确的数学定义，它的数学定义如下：

对于概率分布 D，如果存在一个正常数 p，使得对于任意的 n 个实数 $v_1, v_2, ...v_n$ 和概率分布为 D 的 n 个独立同分布的随机变量 $X_1, X_2..., X_n$，都有：

$$\text{随机变量} \sum_i v_i X_i \text{ 和随机变量} \left(\sum_i |v_i|^p \right)^{1/p} X \text{ 同分布} \tag{3-4}$$

式中　X ——是概率分布为 D 的随机变量;

D ——为服从 P-稳态分布的概率分布。

已经有研究学者证明正态分布是 P-稳态分布，它的 p 值为 2。将两个数据点投影到同一随机向量，利用 P-稳态分布的性质，可以证明在欧几里德空间下，当两个数据点距离近时，有较高的概率两个数据点在随机向量上的投影点距离近，当两个数据点距离远是，有较低的概率两个数据点在随机向量上的投影点距离近。

设向量 $\boldsymbol{a}$ 表示随机向量，由服从正态分布的随机变量生成。向量 $\boldsymbol{v_1}$ 和向量 $\boldsymbol{v_2}$ 分别表示与数据点对应的两个向量，两个数据点映射到随机向量 $\boldsymbol{a}$ 后的距离为 $\boldsymbol{a}(\boldsymbol{v}_1 - \boldsymbol{v}_2)$。由公式(3-4)可知，$\boldsymbol{a}(\boldsymbol{v}_1 - \boldsymbol{v}_2)$ 与 $\|\boldsymbol{v_1} - \boldsymbol{v_2}\|_2 \cdot X$ 同分布，其中 $\|\boldsymbol{v_1} - \boldsymbol{v_2}\|_2$ 等价于数据点在欧几里德空间中的距离，X 是服从正态分布的随机变量，可以看出，当 $\|\boldsymbol{v_1} - \boldsymbol{v_2}\|_2$ 的值变大时，$\boldsymbol{a}(\boldsymbol{v}_1 - \boldsymbol{v}_2)$ 为较大数值的概率更高，当 $\|\boldsymbol{v_1} - \boldsymbol{v_2}\|_2$ 的值变小时，$\boldsymbol{a}(\boldsymbol{v}_1 - \boldsymbol{v}_2)$ 为较大数值的概率更低。

（2）**随机向量投影**　　如图 3-7 所示，在二维空间中对数据点 p 和数据点

q 进行投影。随机向量投影利用高维向量与数据点对应向量的点乘来给每一个数据点赋值。利用随机向量投影赋给的数据点的值是位置敏感的。在 n 维欧几里德空间中如果数据点距离非常接近，则两个数据点在同一个 n 维随机向量上的投影点应该也会非常接近。如果两个数据点的距离非常远，则两个数据点在同一个 n 维随机向量上的投影点距离也会远一些。由于数据点对应的 n 维向量在同一个 n 维随机向量上的投影点可以用实数表示。因此可以通过数据点在同一个 n 维随机向量上的投影对应的实数值来判断数据点之间的距离。

在二维的欧几里德空间下，将数据点 p 和数据点 q 投影到随机向量 $\boldsymbol{a}$ 上，得到投影点 p' 和投影点 q'。如果我们将随机向量 $\boldsymbol{a}$ 所在的直线分成等宽的若干线段，每段长度为 r，对每个线段进行编号，数据点在随机向量 $\boldsymbol{a}$ 上的投影所在的段编号即为随机投影的结果。

通常的，给定一个随机向量 $\boldsymbol{a}$，数据点的随机向量投影可以表示为：

$$f_a(p): R^d \rightarrow N \tag{3-5}$$

在进行随机向量投影时，将经过空间球面网格划分后得到的数据子集投影到整数域。在实际随机向量投影中，我们引入随机实数 b，随机实数 b 是范围在[0,r]内的一个随机数值，对于给定的随机向量 $\boldsymbol{a}$ 和随机实数 b，随机向量投影可以用如下的公式表示：

$$f_{a,b}(p) = \frac{\boldsymbol{a} \cdot \boldsymbol{p} + b}{r} \tag{3-6}$$

为了使随机向量投影的能充分的表现数据点之间的距离远近，我们选取 M 个随机向量和 M 个随机实数。并对同一个数据子集进行 M 次投影。

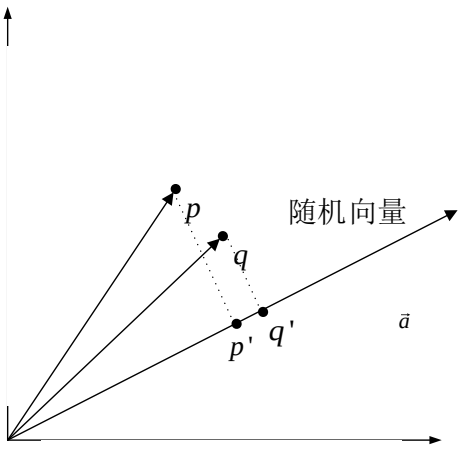

图 3-7 二维空间随机投影示例图

在进行随机向量投影在进行随机向量投影的过程中，由于是将数据点经由

随机向量投影到直线的若干线段上。由于数据在空间中的分布是随机的，因而数据所在的空间球面分区也是随机的，因此，在同一分区中存在着两个距离近的数据点被映射到相邻向量上，而同一区间中随机投影对距离较远的点过滤效果差的问题。如图 3-8 所示，二维空间中对数据子集进行随机投影，数据点 *A* 和数据点 *C* 被投影到一个区间，数据点 *B* 被投影到和数据点 *C*，数据点 *A* 相邻的区间。

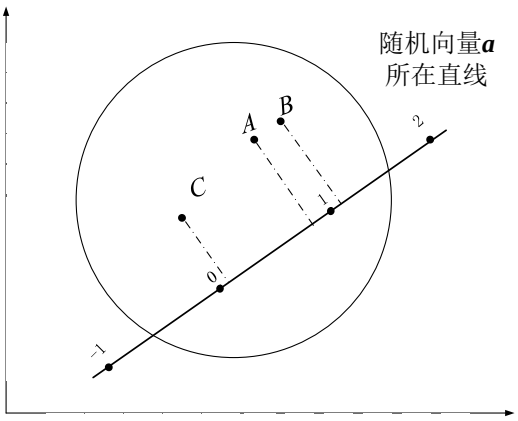

图 3-8 二维空间随机投影示例图

如图 3-9 所示，在对数据点 *A*，数据点 *B*，数据点 *C* 进行随机投影的时候，数据点 *A* 和数据点 *C* 被投影到线段区间[2,3]，数据点 *B* 被投影到线段区间[1,2]。而在欧几里德空间中，数据点 *B* 与数据 *A* 的距离相比数据点 *A* 与数据点 *C* 的距离要更远。因此，我们将随机向量所在直线的线段区间分的更小，同时，对数据点的投影线段区间进行延伸，将数据点投影到数据点原有线段区间的相邻线段区间。

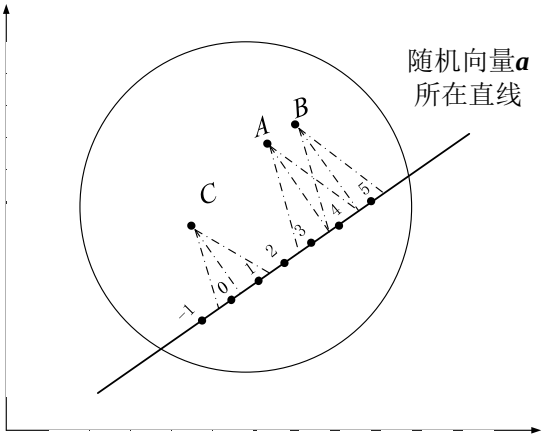

图 3-9 二维空间随机投影延伸示例图

数据点经过投影并进行投影区间延伸。如图 3-9，数据点 A 在进行投影，并经过投影区间延伸后，对应的线段区间分别为线段区间[3,4],线段区间[2,3]和线段区间[4,5]。数据点 B 对应的线段区间分别为线段区间[4,5],线段区间[3,4]和线段区间[5,6]，数据点 C 对应的线段区间分别为线段区间[0,1],线段区间[-1,0]和线段区间[1,2]。数据点 A 和数据点 B 对应的共同线段区间为[3,4]和线段区间[4,5]。对数据子集进行随机向量投影的算法如表 3-2 所示。

表 3-2 随机向量投影算法

随机向量投影算法

输入：划分后的数据子集

输出：哈希函数值

初始化：

对于 $m=1,2,...,M$

（1）选择 t 维随机变换 $\boldsymbol{a}_m$， $\boldsymbol{a}_m$ 的每一个维度由服从 N(0,1)正态分布的随机变量产生

（2）选择随机实数 b_m

计算数据点随机向量投影后的值：

（1）对于 $m=1,2,...,M$

计算 $x_m=\lfloor(\boldsymbol{a}\cdot\boldsymbol{p}+b)/r\rfloor$

（2）返回 $(x_1,x_2,...x_M)$

投影区间进行延伸的目的，是为了降低空间中距离近的数据点被投影到不同线段发生的概率，即对投影区间进行延伸的目的是降低漏判发生的概率。

3.2.3 两层位置敏感哈希函数生成

哈希函数的生成分成两个部分，生成空间球面网格哈希函数和生成 P-稳态分布哈希函数。

首先，是生成空间球面网格哈希函数。第一步是生成空间球面网格组。利用随机函数生成 U 个维度为 t 的随机向量s_u，随机向量的每一个维度的值的范围都为$[0,4w]$，其中 w 是空间球面中的球体的半径；随机向量第二步是生成随机矩阵 A，随机矩阵 A 的作用是将高维空间中的特征点映射到 t 维空间。

$$a_{ij}=\frac{1}{\sqrt{t}}\cdot x_{ij} \tag{3-7}$$

利用随机函数生成d×t个服从正态随机数 $x_{ij}\left(i=1,2,\ldots d,\ j=1,2,\ldots t\right)$，将随机数代入公式4-1可以得到随机矩阵A的元素 a_{ij}；第三步是通过随机矩阵对特征点进行变换，将原先的 R^d 空间中的特征点映射到 R^t 空间：$p'=\mathrm{A}p$。p 表示特征点，p'表示映射后 R^t 空间中的点；第四步是计算映射后的特征点 p'在空间球面组中的位置。

$$r_i=\begin{cases}\dfrac{1}{w}\left(p_i-s_{u_i}\right), if\ \dfrac{1}{w}\left(p_i-s_{ui}\right)\bmod 4=0\\ \dfrac{1}{w}\left(p_i-s_{u_i}\right), if\ \dfrac{1}{w}\left(p_i-s_{ui}\right)\bmod 4=0\\ \infty,\quad \text{the rest of conditions}\end{cases} \tag{3-8}$$

将映射后的 R^t 空间中的特征点代入公式(4-2)得到特征点在网格所在的球心坐标：$\left(r_1,r_2,\ldots,r_t\right)$。在经过计算后，特征点 p 的空间球面网格哈希值 $h_C\left(p\right)=\left(\mathrm{s}_u,r_1,r_2,\ldots,r_t\right)$。

其次，是生成P-稳态分布位置敏感哈希函数。第一步是通过随机函数生成 m 个 t 维向量 $\boldsymbol{v}$，向量的每一个维度服从标准的正态分布。同时通过随机函数生成随机数b，随机数b服从均匀分布。第二步是将是通过随机映射生成哈希值。

$$y_j=\frac{\boldsymbol{v}_j\cdot\boldsymbol{p}'+b}{z} \tag{3-9}$$

式中 z——表示参数，预设为正常数。

将空间 R^t 中的特征点 p'代入公式(4-3)得到特征点的 P-稳态位置敏感哈希值: $h_P\left(p\right)=\left(y_1,y_2,\ldots,y_m\right)$。

经过两层位置敏感哈希函数计算后得到一个向量：

$$h\left(p\right)=\left(h_C\left(p\right),h_P\left(p\right)\right)=\left(s_u,r_1,r_2,\ldots,r_t,y_1,y_2,\ldots,y_m\right) \tag{3-10}$$

在得到最终的哈希值之前，还需要对向量 $\left(s_u,r_1,r_2,\ldots,r_t,y_1,y_2,\ldots,y_m\right)$ 进行编码。对于任意两个向量来说，向量中任意一个维度值的不同都说明两个向量是不同的向量。因此，我们将向量 $\left(s_u,r_1,r_2,\ldots,r_t,y_1,y_2,\ldots,y_m\right)$ 的每一个值都视为一个位置敏感哈希函数值，利用公式(2-4)对向量进行线性变换后求模即可得到最终的哈希值。对向量 $\left(s_u,r_1,r_2,\ldots,r_t,y_1,y_2,\ldots,y_m\right)$ 进行编码的公式如下：

$$h_{final} = s_u \cdot w_1 + r_1 \cdot w_2 + ... + r_t \cdot w_t + y_1 \cdot w_{t+1} + ... y_m \cdot w_{t+m} \tag{3-11}$$

3.3 两层位置敏感哈希函数碰撞概率计算

位置敏感哈希函数的性能可以用参数 ρ 来衡量。该小结我们将计算两层位置敏感哈希函数的参数 ρ。由公式(2-13)可知，计算参数 ρ 的值必须先求两层位置敏感哈希函数的碰撞概率。

假设 d 维空间存在数据点 p 和数据点 q，经过维度变换后，t 维空间中对应的数据点是数据点 p' 和数据点 q'。假设数据点 p 和数据点 q 在 d 维空间的欧几里德距离为 Δ。

假设事件 A 是数据点 p 和数据点 q 经过两层位置敏感哈希函数计算后，发生碰撞的概率；事件 B 是数据点 p 和数据点 q 经过空间球面网格划分后被划分到相同数据子集；事件 C 是数据点 p 和数据点 q 从 d 维空间到 t 维空间的变换后，经过随机向量投影，投影后的值相同；事件 D 是，将数据点变换到 t 维空间后，数据点 p' 和数据点 q' 之间的欧几里德距离为 Δ'；事件 E 是在 t 维空间下，数据点 p' 和数据点 q' 被划分到相同的数据子集；事件 F 是已知数据点 p' 和数据点 q' 之间的欧几里德距离为 Δ'。经过随机向量投影，投影后的值相同。由条件概率公式可得：

$$\begin{cases} P(A) = P(C|B) = P(C) \cdot P(B) \\ P(B) = P(E|D) \\ P(C) = P(F|D) \end{cases} \tag{3-12}$$

（1）**计算事件 *B* 的概率**　首先，计算事件 E 发生的概率。事件 E 等价于 t 维空间下，已知一个数据点在第一次被覆盖进空间球面网格时，另一个点是否也在同一个空间球面网格的同一个超球面内部。因此，事件 E 的概率等于以数据点 p' 和数据点 q' 为圆心的两个球体相交部分的体积与两个球体体积的比。约定 $I(z,w)$ 表示球冠与球的体积比，其中 z 表示球冠离球心的距离，w 表示球体的半径，则有:

$$P(E) = \frac{I(\Delta'/2, w)}{1 - I(\Delta'/2, w)} \tag{3-13}$$

然后，我们讨论事件 D 发生的概率。令 d 维空间数据点 $p = (x_1, x_2, ..., x_d)$，t 维空间下数据点 $p' = (x_1', x_2', ..., x_k', ..., x_t')$。由公式(3-1)可得：

$$x_k' = A_{k1} \cdot x_1 + A_{k2} \cdot x_2 + ... + A_{kd} \cdot x_d \tag{3-14}$$

A_{ij} 是服从 N(0,1)正态分布的随机变量产生的值乘以伸缩因子 $1/\sqrt{t}$ 得到，由正态分布性质可得，随机变量 $A_{kj}\cdot x_j$ 服从 $\mathrm{N}(0, x_j^2/t^2)$ 正态分布。因此随机变量 Δ'^2/Δ^2 的概率密度函数为 $tP_{x^2}(tx)$。其中 $P_{x^2}(x)$ 是卡方分布的概率密度函数。通过对的概率密度函数进行变换，我们可以得到事件 D 的概率密度函数为 $t\Delta\cdot P_{x^2}(t\Delta\cdot x)$，同样 $P_{x^2}(x)$ 是卡方分布的概率密度函数。

最后，我们通过公式(3-4)可以得到事件 B 发生的概率：

$$P(B)=\int_0^{\infty}\frac{I(\sqrt{x},w)}{1-I(\sqrt{x},w)}\cdot t\Delta\cdot P_{x^2}(t\Delta\cdot x)dx \tag{3-15}$$

（2）**计算事件 C 的概率**　　我们讨论事件 F 发生的概率。对空间 R^t 中的数据点 p' 和数据点 q' 进行随机向量投影，得到的投影值为 $\boldsymbol{a}\cdot\boldsymbol{p}+b$ 和 $\boldsymbol{a}\cdot\boldsymbol{q}+b$。对数据点进行随机投影，可以理解为将数据点投影到向量 $\vec{a}$ 所在的直线上。假设数据点 p' 被投影到直线上后的投影点为 M，数据点 q' 被投影到直线上的点为 N。投影后加上常数值 b 可以理解为将数据点在投影后向后移动距离 b，b 为正常数。在投影过后需要对线段区间进行延伸。图 3-10 所示为对线段区间进行延伸的示意图。

将直线分成若干段，每段的长度为 r，当数据点 p' 和数据点 q' 投影到直线上时，将出现三种情况：

假设数据点 p' 和数据点 q' 经过投影后，在 t 维空间对应的投影点之间的距离小于 r，即 $|MN|<r$，则不论随机实数 b 的值为多少，数据点 p' 和数据点 q' 必然发生碰撞。

假设数据点 p' 和数据点 q' 经过投影后，在 t 维空间对应的投影点之间的距离大于 r 小于 2 倍的 r，即 $r<|MN|<2r$，则不论随机实数 b 的值为多少，数据点 p' 和数据点 q' 必然发生碰撞。

假设数据点 p' 和数据点 q' 经过投影后，在 t 维空间对应的投影点之间的距离大于 2 倍的 r 小于 3 倍的 r，即 $r<|MN|<2r$，则不论随机实数 b 的值为多少，数据点 p' 和数据点 q' 必然不发生碰撞。

可以看见，在上述三种情况中，参数 b 的取值范围的长度并不影响数据点 p' 和数据点 q' 经过投影后在 t 维空间的碰撞概率。

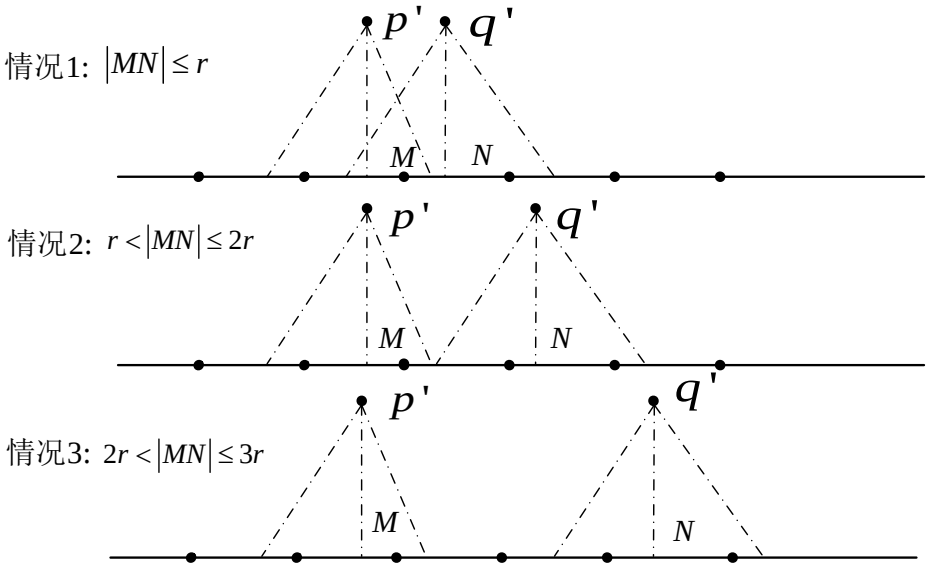

图 3-10 二维空间随机投影示情况图

已知数据点 p' 和数据点 q' 在 t 维空间对应的 t 维向量为 $\boldsymbol{p}$ 和 $\boldsymbol{q}$。设 $f_p(x)$ 表示数据点 p' 和数据点 q' 之间的距离的概率密度函数。对于两个向量 $\boldsymbol{p}$ 和 $\boldsymbol{q}$，令 $\Delta' = \|\boldsymbol{p}-\boldsymbol{q}\|_{\mathrm{P}}$，对于随机向量 $\boldsymbol{a}$，它是由 P-稳态分布的随机变量生成的，由 P-稳态分布的性质可以知道，$\|\boldsymbol{p}\cdot\boldsymbol{a}-\boldsymbol{q}\cdot\boldsymbol{a}\|$ 与 cX 同分布，其中 X 是服从 P-稳态分布的随机变量。由于参数 b 是范围在[0,r]的随机变量，可以得到数据点 p' 和数据点 q' 之间的距离的概率密度函数：

$$f_p(x) = \frac{1}{\Delta}\cdot g(x) \tag{3-16}$$

由随机向量投影的过程，结合公式(3-8)我们可以得到：

$$P(F) = \int_0^{2r} \frac{1}{\Delta'} g\left(\frac{x}{\Delta'}\right) dx \tag{3-17}$$

由我们在（1）中的讨论可以知道，事件 D 的概率密度函数为 $t\Delta\cdot P_{x^2}(t\Delta\cdot x)$。结合公式(3-10)，我们得到事件 C 的发生概率为：

$$P(C) = \int_0^{\infty}\int_0^{2r} \frac{t\Delta}{x} g\left(\frac{y}{x}\right) P_{x^2}(t\Delta\cdot x)\, dxdy \tag{3-18}$$

（3）计算参数 ρ 的上限。

在欧几里德空间中，将数据集的每个维度缩小常数倍数并不会影响数据集的拓扑结构，因此我们可以通过将 r_1 和 r_2 同时缩小 r_1 倍来求取参数 ρ 的上限。令

Δ等于 1，结合公式(3-7)和公式(3-9),得到参数ρ的上限为：

$$\rho \le \frac{1}{c^2} + \frac{1}{2c} + O\left(\frac{\log t}{t^{1/2}}\right) \tag{3-19}$$

计算结果显示，相比原有的基于 P-稳态分布的位置敏感哈希函数，两层位置敏感哈希算法的参数ρ上限高。两层位置敏感哈希函数相对基于 P-稳态位置敏感哈希函数对数据集中距离远的点过滤性能较好，虽然可能存在近距离的数据点被一并过滤的情况，但是总体来讲，两层位置敏感哈希函数相对于 P-稳态位置敏感哈希函数性能要更高。

3.4 对比实验

3.4.1 位置敏感哈希实验程序设计

（1）**图片特征点提取**　　当前的图片特征可以分为两种，全局描述特征和局部描述特征，局部描述特征更多的用来描述图片的局部特征，适用于目标识别，异常行为检测等对图像局部特征要求较高的计算机应用领域。全局描述特征则用来描述一整幅图片的整体信息。由于图片检索对图片的局部描述要求并不高，且更强调整幅图片的全局信息。因此我们使用全局描述特征。在当前，全局描述特征以 GIST 使用范围最广，描述性能最好。

本次实验的目的是检测位置敏感哈希的有效性，因此在给定数据集的情况下，选取的图片特征除了要求对图片描述准确，以维度高的图片特征优先。

（2）**参数优化**　　在 MATLAB 中我们参考 Alexandra 和 Indyk 提供的基于 P-稳态分布的位置敏感哈希的 c++代码实现。在比较不同的哈希编码值时算法的准确率的实验中 LSH 的参数将会有所改变，在其他实验中，LSH 的参数都为默认的最优值[16]。

SpH 哈希函数的参数参考 Jae-Pil Heo 等人的优化过程，对数据集进行降采样，为了保证才后的点基本反映整个数据集的拓扑分布，采用 K-means 进行初始化。在采样的数据点集上确定球半径，使得球体球内与球外数据点数一致，通过迭代计算使得所有的球体的两两交集部分的数据点数的平均值占总样本的四分之一。以最后返回的球心坐标为参数[41]。

3.4.2 实验结果及数据分析

为了验证本文提出的空间球面网格和基于 P-稳态分布的位置敏感哈希算法的性能，在数据集 CIFAR-10 图片数据集上进行实验，CIFAR-10 数据集由 60000

张的彩色图片组成，图片共分十个类，每个类 6000 张彩色图片。从 CIFAR-10 数据集中提取图片的 GIST 特征，并对 GIST 特征库构建位置敏感哈希索引。评价准则是返回若干个最邻近节点时的准确率和召回率。GIST 特征是当前使用范围广泛的图片特征。

对比了两层位置敏感哈希算法，P-稳态位置敏感哈希算法，密度敏感哈希算法。三种算法在不同情况的实际性能。我们在 CIFAR-10 数据集的 GIST 特征库上建立索引结构，然后使用各自查询数据集中的点进行查询，记录下三种算法的准确率和召回率。

（1）**召回率比较**　　如表 3-3 所示，在召回率实验中，两层位置敏感哈希算法，P-稳态位置敏感哈希算法，密度敏感哈希算法都采用 128 位哈希编码，即每个算法的哈希值范围为 0 到 2^{128}。

表 3-3 以样本为变量的召回率曲线

样本数量	两层位置敏感哈希	P-稳态位置敏感哈希	球哈希
10	0.20	0.20	0.20
200	0.61	0.58	0.59
400	0.83	0.79	0.81
800	0.85	0.81	0.83
1000	0.88	0.82	0.85

实验在不同的返回样本数量下的召回率情况下对三种算法进行比对。由于三种位置敏感哈希算法都是基于概率理论的，因此，当检索张索引表后返回的样本数量是随机的。实验中将返回的样本数量取返回的样本中与待查询数据点相似度最高的若干个数据点进行计算。结果显示，两层位置敏感哈希算法在召回率方面高于密度敏感哈希算法和 P-稳态位置敏感哈希算法。

（2）**准确率比较**　　在准确率比较实验中，首先选取 1000 个特征点，通过穷举法找出 1000 个特征点中，与每一个特征点距离最近的 50 个邻近特征点。计算特征点与 50 个邻近特征点的欧几里德距离，并计算与 50 个邻近特征点的平均距离。

我们对算法设置不同的返回样本数量，测试算法的准确率，实验结果如表 3-4 所示。

表 3-4 以样本为变量的准确率表

样本数量	两层位置敏感哈希	P-稳态位置敏感哈希	球哈希
10	0.70	0.60	0.60
200	0.23	0.21	0.22
400	0.15	0.14	0.14
800	0.11	0.09	0.10
1000	0.08	0.06	0.07

如果返回的样本与待查询数据点的距离在平均值以内，则表示返回样本正确，否则表示返回样本错误。由于准确率是返回正确样本数量占所有返回样本的比率，因此，当返回样本数量超过正确的样本数量时，算法的准确率下降。但是从返回的正确样本的数量上看，随着返回样本的数量的增加，返回正确的样本的数量会变多。

在算法准确率的另一个实验中，我们分别以哈希值编码位数和返回样本数量为因变量，测试哈希值编码位数与返回样本数量对两层位置敏感哈希算法，P-稳态位置敏感哈希算法，密度敏感哈希算法的准确率的影响，由于三种算法在进行哈希索引以后，都会逐一计算从哈希桶中返回的数据点与待查询数据点的距离。因此，返回样本越接近待查询的数据点，算法的准确率越高。

如表 3-5 所示，在测试哈希值编码位数对两层位置敏感哈希算法，P-稳态位置敏感哈希算法，密度敏感哈希算法的影响的实验时，发现在编码位数较低的情况下，密度敏感哈希算法具有明显的优势，原因是密度敏感哈希算法是在汉明空间下进行哈希运算，因此在哈希值编码的范围较小时效果较好。

随着编码位数的增加，两层位置敏感哈希函数的平均准确率明显要高于 P-稳态位置敏感哈希算法和密度敏感哈希算法，说明相比于 P-稳态分布位置敏感哈希算法和球哈希算法，两层位置敏感哈希算法在编码位数较高的情况下，其哈希结果更能反映数据集的拓扑结构。

（3）**准确率与召回率曲线**　　如表 3-6 所示，准确率与召回率实验是对算法在召回率基本一致的前提下，比较不同算法的准确率。在相同的准确率情况下，两层位置敏感哈希算法的准确率要更好一些。

在实验中比较了 P-稳态分布位置敏感哈希算法、球哈希算法、两层位置敏感哈希算法在相同召回率前提下的准确率。在相同召回率前提下，准确率越高

说明位置敏感哈希算法将距离近的点哈希到同一个哈希桶中的概率越高，位置敏感哈希函数性能越好。

表 3-5 以哈希编码值为变量的平均准确率表

哈希编码位数	两层位置敏感哈希	P-稳态位置敏感哈希	球哈希
8	0.024	0.022	0.023
16	0.154	0.024	0.025
32	0.223	0.221	0.222
64	0.513	0.412	0.387
128	0.720	0.593	0.531

表 3-6 以召回率为变量的准确率表

算法	召回率	准确率
两层位置敏感哈希算法	0.61	0.65
	0.82	0.42
P-稳态分布哈希算法	0.62	0.47
	0.81	0.31
球哈希算法	0.61	0.41
	0.82	0.22

总的来说，由于位置敏感哈希算法是先通过哈希函数选出于从概率上与待查询数据点相似的候选数据点，然后再对候选数据点逐一计算距离并排序。在建立多个哈希索引表的情况下，候选数据点数量将会增加，而当候选数据点足够多时，查询正确率也会随之增加。相比之下，如果位置敏感哈希算法在准确率大致相同时，算法的召回率越高，证明算法性能更好。

3.5 本章小结

本章研究了两层位置敏感哈希函数。两层位置敏感哈希函数首先利用空间球面网格将数据集中距离远的数据点划分到不同的数据子集，降低误判发生的概率。然后再通过基于 P-稳态分布的随机投影将距离较远的数据点映射到不同

线段，进一步降低误判的发生概率。最后，对投影的线段进行细分和扩展，使得距离近的数据点被映射到相同哈希桶中的概率增加，降低漏判发生的概率。两层位置敏感哈希函数能有效的将数据集中距离近的点映射到同一哈希桶中，使距离远的点映射到不同的哈希桶中。在数据集测试实验中，我们对 CIFAR-10 数据集建立特征库，采用三种算法对特征库建立索引，比较了三种算法的召回率及准确率，证明两层位置敏感哈希在相同准确率的情况下召回率更高，从而看出两层位置敏感哈希算法的性能要更好。

第 4 章 基于 Hadoop 的分布式哈希索引

4.1 引言

本章研究两层位置敏感哈希算法在分布式平台上的性能。在使用位置敏感哈希算法解决海量数据的相似性检索问题时，发现不论是哈希函数的生成、哈希表的存储还是哈希索引表的检索，都存在大量独立且并行的计算。因此，考虑使用分布式平台实现两层位置敏感哈希算法。

4.2 基于 Hadoop 的分布式索引结构

在 Hadoop 集群中，主节点与从节点之间有交互，从节点和从节点之间也存在数据交互。Hadoop 采用的是 MapReduce 计算模型，每一个从节点在工作时都会产生一个 Map 进程和一个 Reduce 进程，Map 进程负责将该从节点的数据进行处理并分发到其他从节点，Reduce 进程负责接受数据并进行数据处理[42]。

4.2.1 单层分布式索引结构

在实现位置敏感哈希算法的查询过程中，每个从节点都存储部分数据并且参与查询计算。分布式集群存在一个主节点，主节点响应查询请求，整合检索结果并返回检索结果。在建立索引的过程中，从节点相互进行交换数据并建立哈希表索引。图 4-1 所示为单层分布式索引的结构。

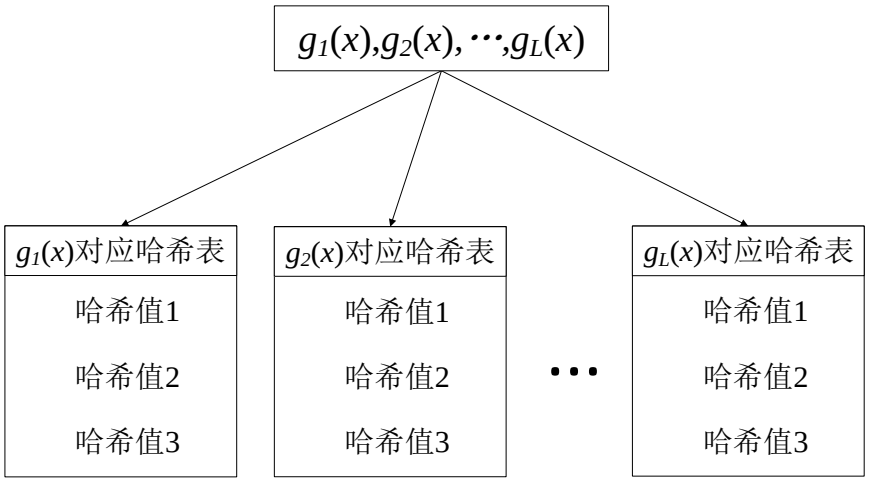

图 4-1 分布式索引结构图

每个哈希索引表对应一个函数$g(x)$，$g(x)$由若干个属于两层位置敏感哈希族的哈希函数$h(x)$组成。

4.2.2 两层分布式索引结构

两层索引结构的分布式哈希索引表是针对两层位置敏感哈希算法特性提出的一个分布式哈希索引结构。两层索引结构的分布式哈希索引表将两层位置敏感哈希算法的分布式哈希索引表分为两个文件存储在多个分布式集群节点。图4-2所示为两层分布式索引表的结构。

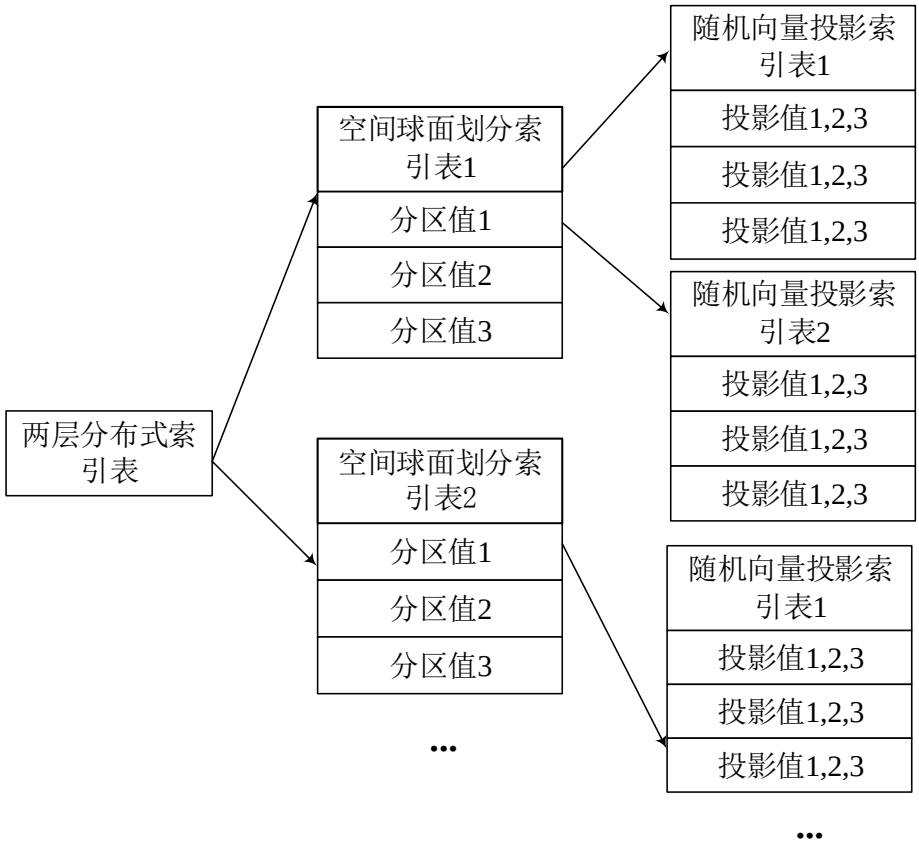

图4-2 分布式索引结构图

两层分布式索引结构一共由两类索引表组成，其中一类是空间球面网格划分索引表，存储的是数据集进行空间球面网格划分后的分区值，另一类是随机向量投影索引表，存储的是数据集进行随机投影后的投影值。在进行检索的时候，先将数据点通过空间球面网格划分确定分区值，然后找到分区值对应的第二层随机向量投影索引表。计算数据点的随机向量投影值，然后返回随机向量

投影索引表中与随机向量索引值对应的内容。

两类索引表都以数据分片的形式存在于分布式集群的每一个节点中。检索时，先对空间球面划分索引表进行 MapReduce 任务，找出数据点的空间球面网格分区值，然后索引到随机向量投影索引表，再次进行 MapReduce 任务，找出相似的数据点。

建立索引表时，分为两个步骤第一个步骤是先通过空间球面网格划分建立空间球面网格哈希索引表，并将数据点位置暂存在空间球面网格哈希索引表上。然后第二个步骤是在建立的空间球面网格哈希索引表上找出数据点，建立随机向量投影索引表。此时，空间球面网格哈希索引表存储数据点分区值以及对应的第二层随机向量投影索引表的位置，随机向量投影索引表存储数据点的随机向量投影值以及对应的位置。

4.3 分布式哈希索引表构造

在该部分，介绍分布式索引表的构造过程。给定的位置敏感哈希参数 k 和参数 L，参数 k 表示每个 $g(x)$ 函数内包含的哈希函数 $h(x)$ 的个数，参数 L 表示位置敏感哈希算法中 $g(x)$ 函数的个数。构造分布式哈希索引结构分为两个步骤：首先，通过 $g(x)$ 函数计算每一个数据点的哈希值；然后，将每一个数据点插入到哈希表中。

位置敏感哈希索引是一种概率索引，为了保证检索的成功率，需要建立多个哈希索引表。多个哈希索引表之间相互独立，在索引过程中每个索引表独立检索，因此采用 MapReduce 模型实现哈希索引表的计算。

MapReduce 是用来实现对海量数据集在集群上进行并行计算和分布式算法的一种编程模型。MapReduce 编程模型中的 Map 线程实现过滤和排序功能，Reduce 线程实现归约功能[43]。

两层结构的分布式索引表采用 MapReduce 分布式编程模型，使用并行的分布式算法实现。用来协助分布式并行算法来实现大规模数据的处理。图 4-3 所示为 MapReduce 分布式编程模型的执行框架。

MapReduce 模型由 Map 线程和 Reduce 线程组成。Map 过程实现数据过滤和数据排序，Reduce 线程实现数据归并。MapReduce 模型通过编组分布式服务器，并行处理任务，管理节点通信和节点间的数据传输来实现数据处理。分布式哈希索引构造过程中执行两个 MapReduce 任务。

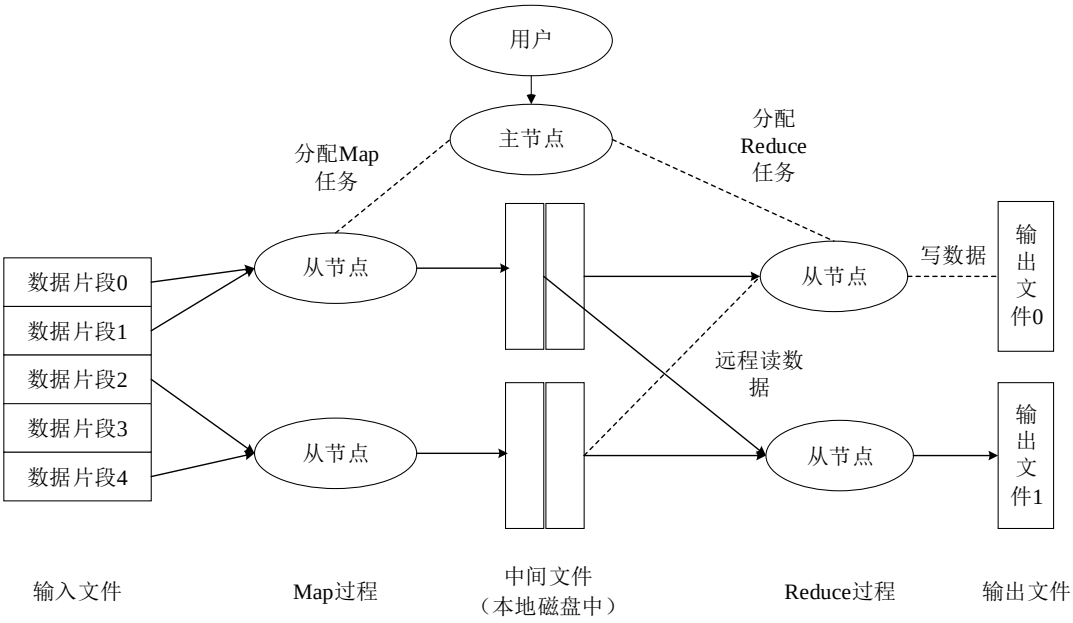

图 4-3 MapReduce 编程模型执行框架图

（1）**MapReduce 任务一**　　在主节点接到一个查询任务后，执行第一个 MapReduce 任务，负责生成位置敏感哈希的部分参数。图 4-4 为 Map 线程和 Reduce 线程的模块结构。

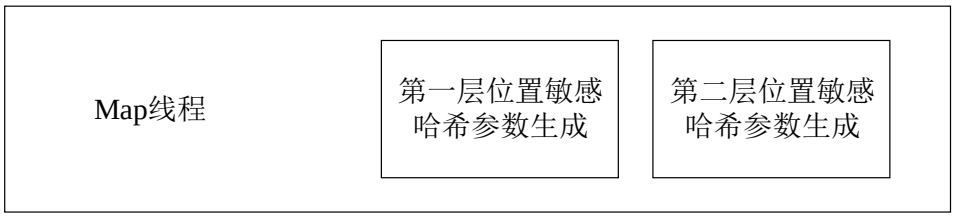

图 4-4 a) Map 线程-LSH 哈希函数生成

图 4-4 b) Reduce 线程-LSH 哈希函数生成

第一个 MapReduce 任务执行过程中，Map 线程生成位置敏感哈希函数的部分参数，并将生成的位置敏感哈希函数参数保存成中间结果。Reduce 线程则对中间结果进行归并，形成位置敏感哈希参数表。

（2）**MapReduce 任务二**　　在第一个 MapReduce 任务结束后，执行第二个 MapReduce 任务，负责特征点哈希值计算和分布式哈希索引表建立，图 4-5

为 Map 线程和 Reduce 线程的模块结构。

图 4-5 a) Map 线程-特征点计算

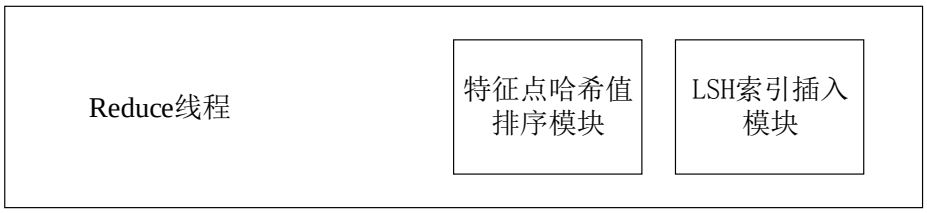

图 4-5 b) Reduce 线程-LSH 索引插入模块

第二个 MapReduce 任务执行过程中，Map 线程计算存储在本地节点中的特征点，形成中间结果。Reduce 线程对中间结果进行归并，得到最后的分布式哈希索引表。

4.4 分布式哈希索引表查询

在用户程序提出查询请求后，主节点执行 MapReduce 任务对分布式哈希索引表进行查找。分布式哈希索引表的结构不同于单机模式的哈希索引表，为了提高处理分布式哈希索引表的时的并行性，提出如下分布式哈希索引表结构。分布式哈希索引表以数据分片的形式存储在集群的不同的节点中，每个数据分片包含索引表编号，哈希值，特征点。在分布式哈希索引表中，索引表编号与 $g(x)$ 函数一一对应，哈希值为特征点经过与索引编号对应的 $g(x)$ 计算后的结果。表 4-1 为分布式哈希索引表的结构。索引表编号和哈希值共同属于索引键，索引键存在且唯一。索引值项则是哈希桶，存储查询后应返回的特征点。哈希索引表在 Hadoop 平台中以单个文件的形式存在，查询使被分成若干数据分片进行分布式查询。

在查询任务中，Map 线程首先，对分布式哈希索引表文件进行数据分片，对每一条索引记录进行筛选，然后对<key,value>键值对进行去重和排序得到中间结果。在 Map 线程中对对<key,value>键值进行排序的是 Shuffle 线程，Shuffle 线程接收 Map 线程的输出结果，在节点中经过排序，输出排序后的结果。Shuffle 线程的输出结果为 MapReduce 线程的中间结果，存储在 HDFS 文件系统中。

Shuffle 线程的输出结果作为 Reduce 线程的输入结果，Reduce 线程对 Shuffle 线程的输出结果进行归约，得到算法输出。

表 4-1 分布式哈希索引表

索引表编号	哈希值	特征点
1	哈希值 1	特征点 1，特征点 2，…
2	哈希值 2	特征点 1，特征点 2，…
3	哈希值 3	特征点 1，特征点 2，…
4	哈希值 4	特征点 1，特征点 2，…
…	…	…

在 Map 线程对索引表进行分片，形成键值对。Map 线程对分布式索引表进行数据分片时，通常以从节点中的数据片段为一个分片。在形成以<key,value>键值对为内容的中间结果的线程中，key 的值包含索引表编号和对应的哈希值。

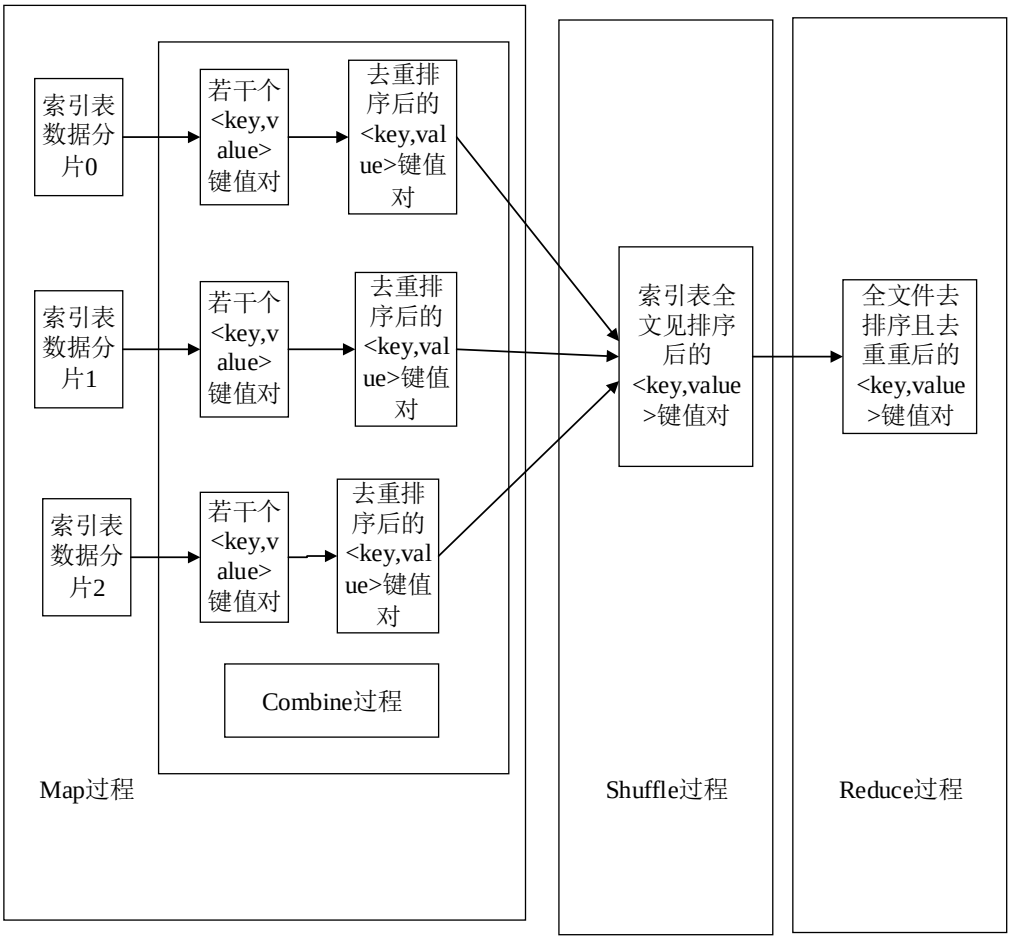

图 4-6 MapReduce 任务数据流

采用 combine 线程对本地节点的键值对先进行去重和排序，然后形成中间结果。中间结果是存储的是部分有序的<key,value>键值对文件。Combine 线程在从节点本地执行，执行后产生的中间结果存储在本地从节点。图 4-6 为分布式哈希索引表的 MapReduce 任务的数据流图。

Combine 线程是 Map 线程的一个子线程，目的是为了减少 Shuffle 线程中的节点间数据交换量，节省带宽，提高 Shuffle 线程的效率。Shuffle 线程对中间结果的全文件进行排序，进一步得到全文件<key,value>键值对有序的中间结果。Reduce 线程对对 Shuffle 线程处理后的中间结果进行归约，去掉中间结果重复的<key,value>键值对，得到最终的查询结果。

4.5 对比实验

4.5.1 系统环境

实验时使用的软硬件系统如下：

（1）硬件环境：处理器为 Intel(R) Xeon(R) CPU E5-2670 @ 2.60GHz，64G 内存；

（2）操作系统：Linux Ubuntu；

（3）编程环境：Java，Eclipse，Hadoop2.6.0

4.5.2 实验数据

为了测试基于 Hadoop 的位置敏感哈希索引，系统采用搜狗实验室 2012 版本的数据测试集，其中收集了包括动物、建筑、人物、风景、机械、运动等不同类别，图片总数达到 2,836,535 张图片，总大小为 269G。在对图片进行检索之前，我们在 Hadoop 分布式平台上对所有图片进行了特征提取，采用时图片全局特征 GIST 特征。GIST 特征具有 1000 个维度，且从全局角度，其对图片的描述更为全面。

4.5.3 实验结果

（1）**运行时间比较**　　我们对两层位置敏感哈希算法、穷举法和传统的位置敏感哈希算法在分布式平台上的运行时间做了对比。穷举法计算待查询特征点与数据集中所有特征点之间的欧式距离，返回数据集中所有与待查询特征点的距离在 R 以内的所有特征点。传统的位置敏感哈希算法通过建立单层位置敏

感哈希索引表，返回数据集中与待查询数据点在欧式空间中距离相近的候选特征点，然后逐一计算候选特征点与待查询特征点之间的欧式距离并返回候选特征点中与待查询特征点距离在 R 以内的特征点。两层位置敏感哈希算法则是通过建立两层位置敏感哈希索引表，通过检索两层位置敏感哈希索引表返回候选特征点，逐一计算候选特征点中与待查询特征点中与待查询特征点的欧式距离在 R 以内的特征点。

实验比较三个算法计算出检索结果需要计算的欧式距离的平均次数和三个算法在单节点中运行所需要的时间。位置敏感哈希算法的运算时间包括索引表的检索时间和距离计算的时间，运行时间实验结果如表 4-2 所示。

表 4-2 运行时间比较

算法	距离计算次数	运算时间
穷举法	2,836,535	1350.3ms
传统位置敏感哈希	54,302	140.1ms
两层位置敏感哈希	34,544	110.8ms

从表 4-2 中可以看出，两层位置敏感哈希算法在距离计算次数和运行时间上要明显优于穷举法。与传统的位置敏感哈希算法相比，两层位置敏感哈希算法距离计算次数减少明显，但是由于需要两次检索哈希索引表，所以在运行时间上要略高于传统的位置敏感哈希算法。

（2）**内存开销比较**　　在穷举法中，将特征点存入特征点文件。检索时，将待查询特征点与特征点文件中的特征点逐一进行欧式距离计算，返回与待查询特征点之间欧式距离小于 R 的所有特征点。传统的位置敏感哈希算法则采用单层分布式索引表。单层位置敏感哈希索引表存储位置敏感哈希值和对应的特征点。

两层位置敏感哈希算法的单层索引表结构。单层分布式哈希索引表的表键存储两层位置敏感哈希值，两层位置敏感哈希值由空间球面网格位置敏感哈希值和 P-稳态位置敏感哈希值共同组成。单层分布式哈希索引表的表值部分则存储特征点。两层位置敏感哈希算法的双层索引表结构。第一层分布式索引表存储特征点的空间球面网格位置敏感哈希值和对应的第二层索引表所在的位置，第二层分布式索引表存储随机向量投影值和对应的特征点所在的位置。

两层位置敏感哈希算法的两层索引以文件的形式存储于 HDFS 文件系统中，在建立两层分布式索引时，先建立第一层索引。计算特征点的空间球面网格分区编号，确定特征点所在的空间球面网格分区。在第一层索引建立完成以后，

第一层索引存储空间球面网格分区的分区编号和特征点所在的位置。第一层分布式索引以数据分片的形式存储于 HDFS 文件系统中。第二次分区建立的时，先计算出特征点的随机向量投影值，建立第二层分布式索引。第二层分布式索引存储特征点的随机向量投影值和特征点所在的位置。两层分布式索引建立完成后，第一层分布式索引存储特征点的空间球面网格分区编号，第二层分布式索引的索引表位置。第二层分布式索引存储特征点的随机投影向量值和特征点所在的位置。

实验结果如图 4-7 所示，在不同数据集规模下，检索特征点时，各个算法的内存开销比较。穷举法由于直接对特征点进行检索，所以内存开销最小。与穷举法不同，传统的位置敏感哈希算法相比单层索引结构的两层位置敏感哈希算法，由于其哈希值位数较少，分布式索引表相对较小，所以内存占用率也要小一些。而两层位置敏感哈希算法将索引分成两层，因此内存占用率最小。

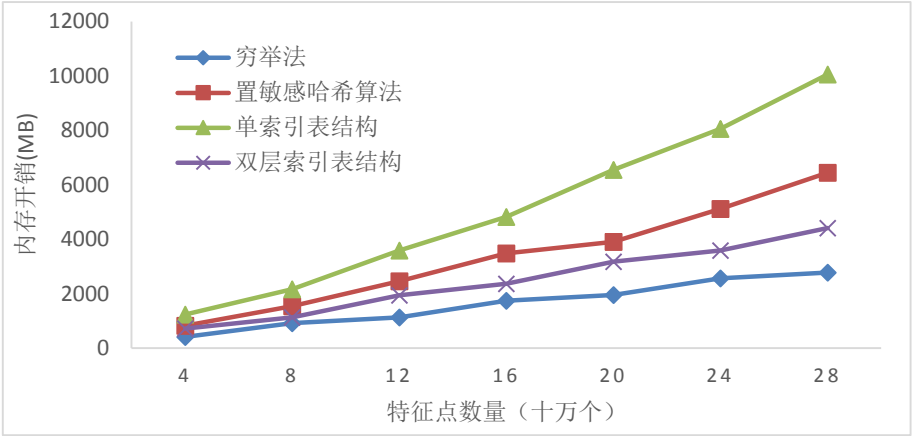

图 4-7 内存开销比较

双层索引表结构的两层位置敏感哈希算法由于将单层分布式索引表分成两层结构，因此在内存占用率上比传统的位置敏感哈希算法和单层索引表的两层位置敏感哈希算法的内存占用率都要小。可以看出，当数据规模足够大时，不同算法之间的内存开销差距将会更大。

（3）**可扩展性实验**　　可扩展性实验讨论在单节点多线程下对两层位置敏感哈希索引表进行检索的情况。我们在单节点下采用两层索引结构的两层位置敏感哈希算法。在建立第一层空间球面网格位置敏感哈希表后，用多个线程建立第二层 P-稳态位置敏感哈希索引表。在检索过程中，先对第一层索引表进行检索，找到第二层索引表所在的位置。然后再通过多线程对第二层索引表进行检索。

在单节点多线程情况下，索引建立时间随着线程数量的增加而下降。当线程数量较少时，由于在对第二层索引表进行检索时是并行操作，因此运行时间下降明显。在线程数量达到一定阈值后，对第一层索引表进行检索占用运行时间比例变大，因此下降趋缓。图 4-8 显示在单个节点下，两层位置哈希算法的索引表建立时间和检索时间随线程数量变化的情况。

其次，讨论在分布式平台上对两层位置敏感哈希算法进行可扩展性实验。如图 4-9 所示，在分布式平台上，保证每个节点处理的数据数量为常数，增加节点的数量，运行时间随线程数量的增加明显减少。

结果显示随着节点数量增加，两层位置敏感哈希的检索运算时间与索引表建立运算时间基本没有变化。在检索和查询过程中，且节点之间数据交互良好。可以看出，两层位置敏感哈希算法在分布式平台上可扩展性良好。

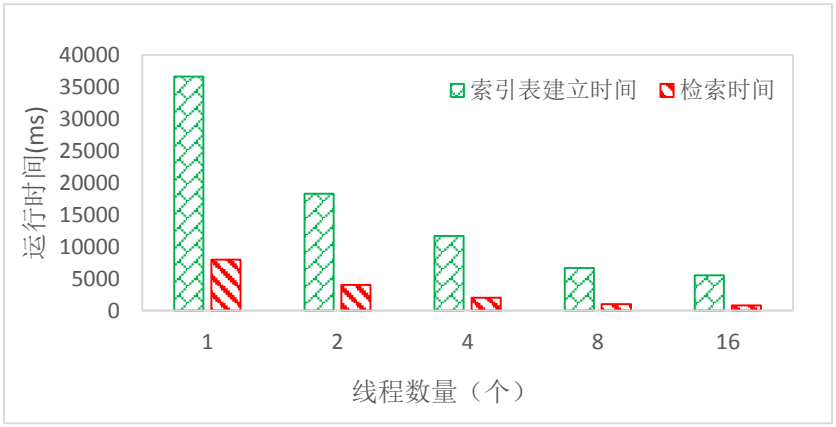

图 4-8 单节点可扩展性实验

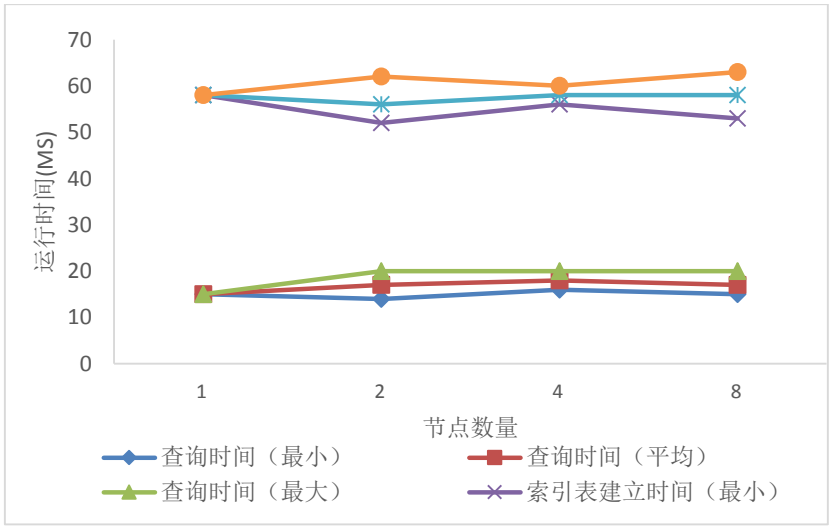

图 4-9 分布式平台可扩展性实验

（4）**延迟和吞吐量实验**　延迟和吞吐量实验测试两层位置敏感哈希算法在分布式平台上对多个待查询特征点进行并行检索时的延迟时间。在实际情况中，更多的是使用分布式平台对数据进行处理。利用多线程对单个特征点进行检索时需要线程之间的同步程度较高，而利用多线程对多个特征点进行检索线程之间同步程度较弱。在实验中，我们通过增加特征点的数量，查看检索时间来判断数据集大小对两层位置敏感哈希算法在分布式平台下检索效率的影响。如图 4-10 所示，在检索的特征点的数量小于 33 个时，使用单层分布式索引表时的检索时间要小于使用两层分布式索引表时的检索时间。在检索的特征点的数量大于 33 时，两层分布式索引表的检索时间要更小。

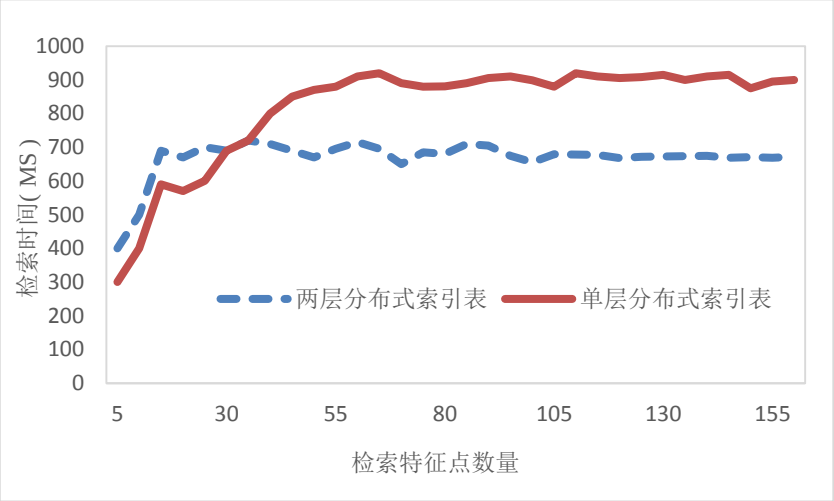

图 4-10 检索时间和每秒检索次数表

实验结果表明在分布式平台上，两层位置敏感哈希算法在数据吞吐量满足一定范围的情况下，数据吞吐量对延迟时间的影响不大。两层索引表结构的两层位置敏感哈希算法相比单层索引结构的两层位置敏感哈希算法，当每秒检索次数在一定阈值以内时，两层索引表结构的延迟时间要长。当每秒检索次数超过一定阈值以后，两层索引结构的两层位置敏感哈希算法延迟时间反而更短。

4.6 本章小结

本章首先分析了位置敏感哈希索引表特点，采用分布式平台处理特征点文件的哈希表建立和索引。然后研究了分布式索引表的结构，采用 MapReduce 分布式编程框架编程实现分布式哈希索引表的建立和查询。最后对两层位置敏感哈希算法做了对比实验，结果显示两层位置敏感哈希算法在分布式平台上表现良好。

结　论

长期以来，利用数据相似性对海量数据进行检索的方法一直是国内外学者的研究热点。对于海量数据的检索，最原始的办法是采用线性检索法，时间复杂度高。后又发展出基于树型结构的检索算法，此算法在检索数据维数较低、数据量比较小的数据集时，对检索效率的提升明显。但是当数据集维数比较高时，数据量达到 TB 级别时，原有的树型结构检索算法的检索效率下降明显。本课题在基于数据相似性的海量数据检索的研究基础上，针对传统的位置敏感哈希算法的不足，提出了空间球面网格划分和基于 P-稳态分布的位置敏感哈希算法，并进行了理论分析和实验，验证了基于 P-稳态分布和空间球面网格划分的位置敏感哈希算法具有较好的准确率和召回率。

本文的主要结论如下：

（1）传统的解决相似性检索问题的位置敏感哈希算法存在一定的局限性，如需要构建大量的哈希索引表；对数据集中的距离近的数据点和距离远的数据点的区分度不够高。

（2）相比经典的位置敏感哈希算法，两层位置敏感哈希算法在数据集中对距离近的数据点和距离远的数据点的区分度要更高。在 CIFAR-10 图片数据集进行的实验表明，两层位置敏感哈希算法在准确率和召回率方面比原有的位置敏感哈希算法要高。

（3）在分布式平台上对两层位置敏感哈希算法进行可扩展性实验和检索效率实验。实验结果显示，两层位置敏感哈希算法在分布式平台上的索引表建立效率和检索效率相比传统的位置敏感哈希算法有明显提高，且两层位置敏感哈希算法在分布式平台上可扩展性更好。

本课题研究的两层位置敏感哈希算法虽然有着很好的性能，但是还不够完善。两层位置敏感哈希算法利用空间球面网格组将数据集划分成若干子数据集，存在着一定的局限性，例如需要多张空间球面网格才能覆盖整个数据空间。划分的精度不够高。为了得到更好的位置敏感哈希算法，下一步的研究方向可以采用如 k-means 等聚类方法对数据集进行划分，进而提高算法对数据集的划分精度。同时，在分布式平台的选择上，可以考虑使用逐渐成熟的 spark 分布式平台，相比基于外存进行操作的 hadoop 平台，spark 分布式平台可以有效提高算法的运行效率。

参考文献

[1] Har-Peled S, Indyk P, Motwani R. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality[J]. Theory of Computing, 2012, 8(1): 321-350.

[2] Tong X, Liu Y, Shi Z, et al. SR-Tree: An Index Structure of Sensor Management System for Spatial Approximate Query[J]. Advanced Materials Research, 2013, 756(9): 885-889.

[3] Ailon N, Chazelle B. Approximate Nearest Neighbors and The Fast Johnson-Lindenstrauss Transform[C]//Proceedings of the 38th Annual ACM symposium on Theory of computing. Seattle: ACM, 2006: 557-563.

[4] Vieira M R, Traina Jr C, Chino F J T, et al. DBM-Tree: A Dynamic Metric Access Method Sensitive to Local Density Data[J]. Journal of Information and Data Management. 2010, 1(1): 111.

[5] Beygelzimer A, Kakade S, Langford J. Cover Trees for Nearest Neighbor[C]// Proceedings of the 23rd international conference on Machine learning. Pennsylvania: ACM, 2006: 97-104.

[6] Karapiperis D, Verykios V S. An LSH-based Blocking Approach with a Homomorphic Matching Technique for Privacy-Preserving Record Linkage[J]. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(4): 909-921.

[7] Rosner F, Hinneburg A, Gleditzsch M. Fast Sampling Word Correlations of High Dimensional Text Data[C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. Arizona: ACM, 2012: 866-866.

[8] Rajaraman A, Ullman J D. Mining of Massive Datasets[M]. Cambridge: Cambridge University Press, 2012:108-111.

[9] Buaba R, Homaifar A, Kihn E. Fast Locality Sensitive Hashing Algorithm For Approximate Nearest Neighbor Search: A Practical Data Mining Approach[D]. North Carolina: North Carolina Agricultural and Technical State University, 2012:13-15.

[10]Dong T, Zhao Z. A novel interactive image retrieval method based on LSH and SVM[C]//Proceedings of 2012 3rd IEEE International Conference on Network Infrastructure and Digital Content. Beijing: IEEE, 2012: 482-486.

[11]Abdelkhalak B, Zouaki H. Randomize Color Signature Employing Locality-Sensitive Hashing [J]. International Journal of Computer Science Issues, 2015, 12(1): 112.

[12]Liu C, Ling H, Zou F, et al. Kernelized Neighborhood Preserving Hashing for Social-Network-Oriented Digital Fingerprints[J]. IEEE Transactions on Information Forensics and Security. 2014, 9(12): 2232-2247.

[13]Han B, Hou Y, Zhao L, et al. A Filtering Method for Audio Fingerprint Based on Multiple Measurements[J]. International Journal of Digital Content Technology and its Applications. 2013, 7(8): 68.

[14]Wilfong G. Nearest Neighbor Problems[J]. International Journal of Computational Geometry & Applications. 1992, 2(04):383-416.

[15]Gionis A, Indyk P, Motwani R. Similarity Search in High Dimensions via Hashing[C]//Proceedings of the 25th International Conference on Very Large Data Bases. San Francisco: VLDB, 1999: 518-529.

[16]Datar M, Immorlica N, Indyk P. Locality-Sensitive Hashing Scheme based on P-Stable Distributions[C]//Proceedings of the 20th Annual Symposium on Computational Geometry. New York: ACM, 2004: 253-262.

[17]Andoni A, Indyk P. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions[C]//Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science. California: IEEE, 2006: 459-468.

[18]Andoni A, Indyk P, Krauthgamer R. Earth Mover Distance over High-Dimensional Spaces[C]//Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms. Philadelphia: ACM, 2008: 343-352.

[19]Andoni A, Indyk P, Nguyen L , et al. Beyond Locality-Sensitive Hashing[C]//Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms. San Diego: ACM, 2014: 1018-1028.

[20]Wang J, Wang J, Yu N , et al. Order Preserving Hashing for Approximate Nearest Neighbor Search[C]//Proceedings of the 21st ACM International Conference on Multimedia. New York: ACM, 2015: 133-142.

[21]王镪. 基于内容的海量音乐检索技术研究[D]. 北京：北京邮电大学学位论文, 2013: 32-35.

[22]蒋凯, 武港山. 基于Web的信息检索技术综述[J]. 计算机工程, 2005, 31(24): 7-9.

[23]袁培森. 基于LSH的Web数据相似性查询研究[D]. 上海：复旦大学学位论文, 2011: 21-24.

[24]甘骏豪. 基于动态碰撞检测的位置敏感哈希[D]. 广州：中山大学学位论文, 2011: 18-20

[25]何学文. 基于LSH的语音文档主题分类研究[D]. 哈尔滨：哈尔滨工程大学学位论文, 2012: 34-37.

[26]周建辉. 基于半监督哈希算法的图像检索方法研究[D]. 大连：大连理工大学学位论文, 2011: 10-12.

[27]林悦. 基于哈希算法的高维数据的最邻近检索[D]. 浙江：浙江大学学位论文，2013: 14-16.

[28]Neylon T. A Locality Sensitive Hash for Real Vectors[C]//Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms. Texas: ACM, 2010: 1179-1189.

[29]Paulevé L, Jégou H, Amsaleg L. Locality Sensitive Hashing: A Comparison of Hash Function Types and Querying Mechanisms[J]. Pattern Recognition Letters. 2010, 31(11): 1348-1358.

[30]Kang B, Jung K. Robust and Efficient Locality Sensitive Hashing for Nearest Neighbor Search in Large Data Sets[C]//Proceedings of The 26th Annual Conference on Neural Information Processing Systems. Nevada: NIPS, 2012: 1-8.

[31]Buaba R, Homaifar A, Kihn E. Optimal Load Factor for Approximate Nearest Neighbor Search under Exact Euclidean Locality Sensitive Hashing[J]. International Journal of Computer Applications. 2013, 69(21): 22-31.

[32]Lee K M. A Projection-Based Locality-Sensitive Hashing Technique for Reducing False Negatives[J]. Applied Mechanics and Materials. 2012, 263(10): 1341-1346.

[33]Ma Y, Wang F, Wu J. Locality Sensitive Hashing Based Service Classification[C]//Proceedings of The 2011 International Conference on Management and Service Science. Wuhan: IEEE, 2011: 1-5.

[34]Lee K M, Lee K M. A Locality Sensitive Hashing Technique for Categorical Data[J]. Applied Mechanics and Materials. 2013, 241(10): 3159-3164.

[35]Xu Z, Xia M. Distance and Similarity Measures for Hesitant Fuzzy Sets[J]. Information Sciences. 2011, 181(11): 2128-2138.

[36]Liberti L, Lavor C, Maculan N, et al. Euclidean Distance Geometry and Applications[J]. SIAM Review. 2014, 56(1): 3-69.

[37]Arasu A, Ganti V, Kaushik R. Efficient Exact Set-Similarity Joins[C]//Proceedings of the 32nd International Conference on Very Large Data Bases. Seoul: VLDB, 2006: 918-929.

[38]Hua Y, Xiao B, Feng D, et al. Bounded LSH for Similarity Search in Peer-to-Peer File Systems[C]//Proceedings of 37th International Conference on

Parallel Processing. 2008. IEEE, 2008: 644-651.

[39]Zhang D, Lu G. Evaluation of Similarity Measurement for Image Retrieval[C]//Proceedings of the 2003 International Conference on Neural Networks and Signal Processing. Nanjing: IEEE, 2003, 2: 928-931.

[40]Bayardo R J, Ma Y, Srikant R. Scaling up all Pairs Similarity Search[C]//Proceedings of the 16th International Conference on World Wide Web. Canada: ACM, 2007: 131-140.

[41]Heo J P, Lee Y, He J, et al. Spherical hashing[C]//Proceedings of 2012 IEEE Conference on Computer Vision and Pattern Recognition. Providence: IEEE, 2012: 2957-2964.

[42]Shvachko K, Kuang H, Radia S, et al. The Hadoop Distributed File System[C]//Proceedings of 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies. Lake Tahoe: IEEE, 2010: 1-10.

[43]Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

哈尔滨工业大学学位论文原创性声明和使用权限

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《基于P-稳态分布和空间球面网格的位置敏感哈希算法》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果，且学位论文中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本学位论文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。

作者签名：陈翔羽 日期：2015 年 12 月 25 日

学位论文使用权限

学位论文是研究生在哈尔滨工业大学攻读学位期间完成的成果，知识产权归属哈尔滨工业大学。学位论文的使用权限如下：

（1）学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，并向国家图书馆报送学位论文；（2）学校可以将学位论文部分或全部内容编入有关数据库进行检索和提供相应阅览服务；（3）研究生毕业后发表与此学位论文研究成果相关的学术论文和其他成果时，应征得导师同意，且第一署名单位为哈尔滨工业大学。

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉学位论文的使用权限，并将遵守有关规定。

作者签名：陈翔羽 日期：2015 年 12 月 25 日

导师签名：王轩 日期：2015 年 12 月 27 日

致　谢

时光如白驹过隙，匆匆而过。细想两年半的研究生生活，丰富多彩，既有欢声笑语际，亦有难过伤心时。

首先要感谢我的导师王轩老师，王老师两年来对我的悉心栽培和谆谆教诲，让我的知识理论水平、综合素质和技能有了很大的提升，更是对为人处事有了更为深刻的理解。在毕业设计的研究过程中，王老师给了我很多的支持跟帮助，对我遇到的一些问题给了我很好的建议，使我能思维开阔，明确了前进的方向。让我记忆深刻的是，王老师虽然公务繁忙，但是在科研和学习上，仍然一丝不苟的对我们进行监督和指导，使我受益良多。

同时感谢实验室的同学和秘书，感谢我的两位舍友，感谢两年半以来你们在学习、科研和生活上对我的无私帮助和照顾，感谢你们陪我走过了两年半美好的研究生生活。感谢我在研究生阶段的各位好朋友，很开心求学道路上认识了你们。

最后，感谢我的父母在我多年的求学路上对的坚定支持，你们二十多年无怨无悔的鼓励、支持和艰辛付出，让我无后顾之忧的学习和生活，在更高的学术平台上发光发热。一路前行，离不开家人、老师、同学和朋友们的帮助和照顾，谢谢你们，让我走得更远，活的更精彩。