代　号 10701　　学　号 1021121286

分　类 TP301.6　　密　级 公开

题（中、英文）目 局部敏感哈希算法的研究

Research on the Locality Sensitive Hashing

作　者　姓　名 史世泽　　**指导教师姓名、职务** 霍红卫 教授

学　科　门　类 工学　　**学科、专业** 计算机软件与理论

提交论文日期 二〇一三年三月

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。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切的法律责任。

本人签名：_____ 日期_____

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单位属西安电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律署名单位为西安电子科技大学。

（保密的论文在解密后遵守此规定）

本学位论文属于保密，在___年解密后适用本授权书。

本人签名：_____ 日期_____

导师签名：_____ 日期_____

摘要

高维数据的近邻搜索是许多应用研究的一个基础问题，它需要依赖于有效的数据结构和算法。本文主要研究了局部敏感哈希算法并对其进行了改进。

在本文中，我们首先阐述了局部敏感哈希算法的基本思想，并介绍了几个具体的局部敏感哈希算法：基于汉明距离的局部敏感哈希算法，基于 P 稳定分布的局部敏感哈希算法，多探寻的局部敏感哈希算法。注意到上述算法中回收数据的处理依赖于对数据的距离的计算和排序，这需要花费大量时间。针对此，本文提出了一种对于局部敏感哈希算法的改进方法，其主要思想是使用数据的出现次数来近似数据的距离，从而从回收数据中获得结果。在该算法中，我们在数据的回收过程中统计数据的出现次数，然后将数据按其出现次数分类，并按其出现次数的大小依次获得结果。

最后，本文给出了两个评估局部敏感哈希算法的两个标准参数，并分别使用实际数据和模拟数据对算法进行了测试，证明了该方法能有效的减少算法的运行时间，提高算法的时间效率。

关键词：高维数据　近邻搜索　局部敏感哈希

Abstract

The nearest neighbor search for high-dimension data is a fundamental problem in many application domains, it needs to rely on efficient data structures and algorithms. This paper studies the locality sensitive hashing algorithm and improve it.

In this paper, we first explain the basic idea of the locality sensitive hashing algorithm, and introduce several specific locality sensitive hashing algorithm: locality sensitive hashing algorithm based on the Hamming distance, locality sensitive hashing algorithm based on P-stable distributions, multi-probe locality sensitive hashing algorithm. Note that the algorithms described above, final data processing depends on computing the distance of the candidate data and ranking them by distance, which takes a lot of time. Thus , in this paper, we present a improved method for locality sensitive hashing , the main idea is to use the number of data occurrences to approximate the distance of the data, and we use it to get the answer from the candidate data. In this algorithm, we count the occurrences of the data in the data retrieve process, then we make the data classification by their number of occurrences , and we use it to get the answer from the retrieve data.

Finally, we give two standard parameters to evaluate the locality sensitive hashing algorithm ,and we use a set of actual data and a set of analog data to test the algorithms , it proved that the method can effectively reduce the running time of the algorithm and improve the time efficiency of the algorithm.

Keyword: High-dimension data Nearest neighbor search
Locality sensitive hashing

目录

第一章 绪论.....1
1.1 研究意义及背景.....1
1.2 国内外研究现状.....2
1.3 本文的组织结构.....3
第二章 局部敏感哈希算法概述.....5
2.1 基本方法.....5
2.2 距离度量.....6
2.3 局部敏感哈希函数库.....8
2.4 本章小节.....9
第三章 几个常用的局部敏感哈希算法介绍.....11
3.1 基于汉明距离的局部敏感哈希算法.....11
3.1.1 算法描述.....11
3.1.2 算法分析.....14
3.2 基于 P 稳定分布的局部敏感哈希算法.....16
3.2.1 P 稳定分布.....16
3.2.2 算法描述.....17
3.3 多探寻的局部敏感哈希算法.....18
3.3.1 基于熵的局部敏感哈希算法.....18
3.3.2 算法概述.....19
3.3.3 步进式的探寻.....20
3.3.4 一个更精确的序列产生方法.....21
3.3.5 扰动序列的构造优化.....24
3.4 本章小结.....25
第四章 对局部敏感哈希算法的改进.....27
4.1 改进的动机.....27
4.2 算法描述.....28
4.2.1 哈希表的构造.....28
4.2.2 数据回收.....29
4.2.2 数据处理.....31
4.3 本章小节.....32
第五章 实验.....33
5.1 实验数据.....33

5.2 评价标准.....33
5.3 实施细节.....34
5.4 实验结果.....34
5.5 本章小结.....40
第六章 总结与展望.....41
致谢.....43
参考文献.....45

第一章 绪论

1.1 研究意义及背景

随着计算机技术的飞速发展以及互联网的不断普及，信息技术极大的影响了人们的生活方式。互联网使得信息的获取，传播以及积累的速度和规模达到了前所未有的地步，实现了世界信息的共享与交互，使人们的工作和生活得到了极大的方便。随着因特网的不断建设和完善，网络带宽的不断增加，现实世界与互联网不断的互相融合与发展，互联网应用的不断增加与完善，网络的信息数据日益庞大。过去数十年，人类所拥有的知识和信息每 5 年翻一番，据有资料统计，近三十年来人类所产生的的信息已经超过了过去 5000 年的信息数据之和，而预计到 2020 年，每三个月，信息和知识就要翻一番，我们把这个称为“信息爆炸”。而从如此海量的数据中如何获得自己所需要的信息，这成为一个很大的问题。

进入 21 世纪，谷歌和百度等搜索引擎迅速成为互联网的新星，因为它们能迅速的从海量的互联网文本信息数据中准确检索出我们所需要的文本数据信息。但是，随着互联网应用的不断扩展，人们对于搜索的需求已经不仅仅满足于对文本信息的数据，也包括图像，音频等特征信息丰富的数据。对于这些数据的处理迅速成为了现在研究的热点，而对于这些特征信息丰富的数据转化成的高维数据的处理也成为了一个被关注的问题。

对于数据搜索问题，KNN 问题是一个被广泛研究的问题。KNN 问题即是指 K 最近邻节点问题，用来寻找一个查询点附近的 K 个最邻近的点，是许多应用领域中研究的一个基础问题，例如：数据挖掘[1]，机器学习[2]，信息检索[3]，多媒体数据[4]，生物信息学[5]，重复文件检查[6,7]等等。在向量模型中，这些特征丰富的数据对象可以表示为 R^d 空间里的一个点(d 是指空间向量的维度，即数据特征所表示向量的维度)，这些数据对象可以表示为几十维至几千维的数据。一些在 R^d 空间上的距离衡量方法可以用来表示这些数据对象的相似性，因此，这些数据的相似性可以与空间上的点之间的距离相关联。

K 最近邻节点问题已经被广泛的研究了几十年，具体问题可以被定义如下：给定一个数据点的集合 S，对于一个查询点 q，返回一个包含 k 个数据的集合 T，该集合满足以下性质，如果 $a \in T, b \in S$ 且 b 不属于 T，则有 $F(q,a) \leqslant F(q,b)$，F 是指根据数据性质定义的距离，例如欧几里德距离。

对于 K 最近邻问题，一个简单的方法是计算 R^d 空间内所有的点与点 q 的距离，并选出其中距离最小的 k 个点。显然，这个方法将会花费巨大的时间，且仅能适用于数据数量较小能够在主存中运行的情况(当数据量较大时，数据只能存储在磁

盘中，因此数据的计算用时和 I/O 读取用时都将是极其庞大的)。然而，随着计算机技术的发展和应用的广泛增多，如今，数据集合的规模变的越来越庞大，需要处理的数据也变的越发庞大，从 GB 级别到 TB 级别，这意味着要处理数以亿计的几百维乃至于几千维的高维数据。对于如此大的数据规模，必须要依靠有效的数据索引结构来处理这么庞大的数据，来更快的处理数据，在恰当的时间内返回数据结果。

对于如此大规模的数据，必须依赖于有效的数据结构支撑。在局部敏感哈希算法之前，提出了基于树的空间划分方法，能够有效的应用于低维数据之上，例如，R-Tree[8],KD-Tree[9],SR-Tree[10]以及 cover-Tree[11]，都能返回准确的结果，但对于高维数据，他们不能提供一个令人满意的时间效率，当数据的维度超过 10 时，有资料显示[12]，这些依赖与空间划分的数据索引结构方法的时间效率将会急剧下降，甚至不强与强力线性扫描方法，这被称为“维度灾难”。

基于此，P.Indyk 和 R.Motwani 提出了局部敏感哈希方法(LSH)，来解决“维度灾难”，他们基于这样一个理念，即近似的搜索结果能提供一个令人满意的搜索效果。因此，它使用近似的 K 最近邻问题来替代精确的 K 最近邻问题。近似的 K 最近邻问题定义如下：

假设对于查询点 q,其 KNN 问题的精确的查询结果的最大距离为 r(r = max(Dis(p,q)),Dis()是距离函数，q 是查询点，p 是 KNN 问题的精确查询结果)，所以，在高维空间中，K 最近邻问题的 k 个返回结果在查询点 q 相距为 r 的范围内,对于近似的 K 最近邻节点问题，我们希望返回 k 个与查询点 q 相距(1+ ε) * r 内的数据点。在定义中， ε > 0，且 ε 的数值越小，则搜索的质量越好，当 ε = 0，即近似的 K 最近邻节点问题等价于精确的 K 最近邻节点问题。在文[13]中,作者提出了我们所熟知的局部敏感哈希算法，并显示了它能很好的解决近似的 K 最近邻节点问题。

1.2 国内外研究现状

局部敏感哈希算法已经有了十几年的历史，在高维数据搜索理论以及应用方面引起了广泛的关注和研究。

在文[13]中，首次提出了局部敏感哈希算法，详细阐述了局部敏感哈希算法的思想以及理论证明，并于基于空间划分的高维数据搜索方法做了比较，解决了其存在的“维度灾难”问题。

1999 年，P.Indyk 和 R.Motwani 提出了一个基于汉明距离的局部敏感哈希方法，该方法[14]将高维数据中的向量坐标都转换为正整数，并将正整数再转换为 0,1 编

码，从而通过汉明距离来表示两个点间的距离。在前述算法中，在特殊情况下，在一个 n 个点的数据集合中，最近邻搜索的最坏性能可能达到 O($dn^{1/\varepsilon}$)，而在这个算法中，显著改进了其算法性能，使其的搜索时间可以达到 O($dn^{1/(1+\varepsilon)}$),其中，ε > 0,是衡量近似最近邻问题的近似性的一个系数。

2004 年，在文[15]中，提出了一个利用 P 稳定分布性质的局部敏感哈希方法，其原理是利用 P 稳定分布性质，将相近的点以高可能性聚集在一起。该方法可以应用于任何 L_p 距离上，其中 $p\in(0,2]$, L_p 距离即欧几里德距离，L_2 即我们常用的“距离”，即 d([x1,x2,…,xn],[y1,y2,…,yn]) = $((x1-y1)^2+(x2-y2)^2+\ldots+(xn-yn)^2)^{1/2}$。与前述算法相比，该算法可以直接应用于高维数据的实数坐标上，无须将他们转换成其他数据。

在此基础上，许多研究者专注于研究提高哈希表的利用率，提高局部敏感哈希算法的效率。在相关的理论研究中[16]，Panigrahy 提出并论证了一个基于熵的局部敏感哈希方法，它通过随机产生查询点附近的扰动点，并查询这些扰动点的近邻数据点，返回查询点的数据结果以及这些数据结果的集合，从而提高哈希表的利用率，减少哈希表的数量，节省数据的存储空间。在此基础上，Qin Lv 和 William Josephson 等人，探索了合适的改进方法，提出了多探寻的局部敏感哈希算法，它[17]通过设计合适的扰动序列产生查询点附近的扰动点而不是随机的产生扰动点来改进方法，在文中，它提出了一个步进式的扰动序列生成方法以及一个针对 P 稳定分布的更精确的扰动序列生成方法。

局部敏感哈希算法及其变种被广泛的应用于许多领域的计算问题中，相当多的研究者针对具体问题对局部敏感哈希算法进行了深入的研究，包括文本聚类[18],计算生物学[19,20],计算机视觉[21],药物设计计算[22]以及语言计算统计[23]。在国内，局部敏感哈希算法也被广泛的应用于图像搜索以及音频搜索等领域中。例如，在文[24]中，使用基于 P 稳定分布的局部敏感哈希算法来进行音频检索，在文[25,26]中，局部敏感哈希算法被用来处理视频以及图片。

1.3 本文的组织结构

本文的组织结构如下：

第一章：绪论部分。该部分介绍了 K 最近邻节点问题以及近似 K 最近邻节点问题，介绍了局部敏感哈希算法产生的背景及意义，并介绍了国内外对于局部敏感哈希算法的研究情况。

第二章：局部敏感哈希算法概述。本章主要概括讨论了局部敏感哈希算法的基本思想以及其相关的基础知识，首先介绍了局部敏感哈希算法的基本思想和方

法，并给出了一个简明的算法；其次介绍了与局部敏感哈希算法紧密相关的各种距离度量，最后，给出了一个局部敏感哈希函数族的函数库，介绍了基于不同距离下的一些具有局部敏感特性的哈希函数族。

第三章：几个常用的局部敏感哈希算法介绍。本章具体介绍了三个不同的局部敏感哈希算法。首先，我们介绍了两个基本的局部敏感哈希算法，它们设计了不同的局部敏感哈希函数族；然后，我们介绍了一个在基本的局部敏感哈希算法之上的改进方法，它通过在哈希表中生成扰动序列来查询更多的高成功性的哈希桶来提高哈希表的利用率，从而减少算法的空间需求。

第四章：对局部敏感哈希算法的改进。在本章中，我们介绍了对局部敏感哈希算法的一个改进。我们首先介绍了这一改进的动机以及改进的所依赖的性质与方法，然后，我们具体阐述了这个方法对于前述章节的三个局部敏感哈希算法进行改进后的算法。

第五章：实验。本章主要介绍了算法的实验，首先我们介绍了算法的实验数据，以及对算法的评价标准的两个标准参数和实验细节。然后，我们显示了在不同情况下算法的实验结果，实验结果显示，基于统计的改进算法能改善基本的局部敏感哈希算法的时间效率。

第六章：总结与展望部分。该部分概括了全文，对本文的所有工作进行了总结，并且对进一步的工作方向进行了展望。

第二章 局部敏感哈希算法概述

2.1 基本方法

局部敏感哈希算法的基本思想是将两个点 p 和 q 冲突的可能性与其距离紧密相连，即两个点的距离越近，它们冲突的可能性越高，两个点的距离越远，它们冲突的可能性则越低。

局部敏感哈希算法依赖于局部敏感哈希函数族，定义 H 是一个由 R^d 映射到集合 U 的哈希函数族，对于任意两个点 p 和 q,对于任意的从哈希函数族 H 中随机选择的一个哈希函数 h,我们分析 h(p) = h(q)的可能性。如果满足以下两个性质，我们则将函数族 H 称为局部敏感的：

定义 2.1 局部敏感哈希 ： 如果对于 R^d 空间内的任意两个点 p 和 q，满足以下两个性质，则称函数族 H 是(R,cR,P_1,P_2)敏感的：

If $||p-q|| \leqslant R$ then $Pr_H(h(p) = h(q)) \geqslant P_1$ 式(2-1)

If $||p-q|| \geqslant cR$ then $Pr_H(h(p) = h(q)) \leqslant P_2$ 式(2-2)

为了使局部敏感哈希函数有效，我们使 $c > 1$, $P_1 > P_2$ 。

为了说明这个概念，考虑下面的这个例子。假设数据点是二进制的，即每个坐标都是 0 或 1。另外，假设两个点 p 和 q 的距离由汉明距离来确定，在这个例子中，我们可以使用一个简单的局部敏感哈希函数族 H，它包含输入点映射到点的一个坐标的全部映射，即 H 包含所有函数 h_i(从$\{0,1\}^d$ 映射到{0,1}),例如 $h_i(p) = p_i$,随机的从函数族 H 中选择一个函数 h 即意味着随机从点的坐标中随机选择一个坐标。

现在我们来考察函数族 H 的局部敏感哈希特性，观察到概率 $Pr_H(h(p) = h(q))$ 等价于 p 和 q 的坐标的相同个数，因此，$P_1 = 1 - R/d$, $P_2 = 1 - cR/d$,因为参数 c 大于 1，所以 $P_1 > P_2$ 。

局部敏感哈希函数族 H 可以被用来设计一个有效的算法来解决近似最近邻搜索问题。但是，对于函数族 H 而言，一个显著的问题是两个概率 P_1 和 P_2 之间的差距太小而不能直接应用。因此，需要一个处理方法来扩大这两者之间的差距，下面描述了这个方法。

给定一个哈希函数族 H 拥有定义 2.1 中的(R,cR,P_1,P_2)敏感特性，我们通过使用多个哈希函数来扩大高可能性 P_1 和低可能性 P_2 之间的差距。特别的，对于参数 k 和 L，我们选择 L 个函数 $g_j(q) = (h_{1,j}(q), h_{2,j}(q),\ldots h_{k,j}(q))$, $h_{t,j}()(1 \leqslant t \leqslant k , 1 \leqslant j \leqslant L)$是独立的从函数族 H 中随机选取的。这些是我们在哈希数据时所实际使用的函数。

该数据结构用来把集合中的每一个点 p 哈希到桶 $g_j(p)$中，j = 1,…,L ，因为桶的整体数量可能庞大，所以我们通过标准哈希来保存那些非空的桶。通过这个方

法，数据结构只使用 O(nL)个数据单元，注意桶中只存储指向数据点的指针，而并不是存储数据本身。

处理一个查询点 q,我们通过扫描 $g_1(q), g_2(q),\ldots, g_L(q)$,来回收存储在它们桶中的数据，在回收这些数据点后，我们计算这些数据点与查询点 q 之间的距离，返回符合满足需要的数据点。以下是两个可行的收集扫描策略：

1. 在找到 w 个数据(包括重复)后中止搜索过程。
2. 持续搜索每个桶中的每一个数据，不需要任何一个附加参数。

下面简单列出了算法过程，以供参考：

Preprocessing:

1. Choose L functions g_j , by setting $g_j = (h_{1,j}, h_{2,j},\ldots h_{k,j})$ where $h_{1,j}, h_{2,j},\ldots h_{k,j}$ are chosen at random from the LSH family H.
2. Construct L hash tables, where,for each j = 1,…,L ,the j hash table contains the dataser points hashed using the function g_j.

Query algorithm for a query point q:

1. For each j = 1,…,L

a) Rrtrieve the points from the bucket $g_j(q)$ in the j hash table

b) For each of the retrieved point ,compute the distance from q to it , and report the point if it is a correct answer

c) (optional) Stop as soon as the number of reported points is more than w

2.2 距离度量

设计针对不同距离度量的局部敏感哈希函数族是设计局部敏感哈希算法的关键，下面介绍几种常用的距离度量。

首先我们来确定一个距离度量的定义：

假设我们有一个点的集合，称为空间。一个在该空间内的距离度量函数 d(x,y)处理两个点来产生一个实数，并满足下列性质：

1. d(x,y) > 0 (没有负的距离)
2. d(x,y) = 0 当且仅当 x = y(距离是正的，除了一个点于它本身的距离)
3. d(x,y) = d(y,x) (距离是对称的)
4. d(x,y) ≤ d(x,z) + d(z,y) (三角不等式)

三角不等式是最复杂的条件。它直观的说明，从点 x 到 y，我们不可能通过某个特定的点 z 来减少距离。

下面介绍常用的距离：

2.2.1 欧几里德距离

最著名的距离，也是我们脑海中通常所想的“距离”。一个 n 维欧几里德空间是由包含 n 个实数向量的点的坐标来确定的，在这个空间中，最常用的距离我们定义为 L_2 模式，定义如下

$$d([x1,x2,\ldots,xn],[y1,y2,\ldots,yn]) = ((x1-y1)^2+(x2-y2)^2+\ldots+(xn-yn)^2)^{1/2} \quad \text{式(2-3)}$$

也就是说，将每一维的距离平方后，然后将它们相加在开方。

2.2.2 jaccard 距离

两个集合 S 和 T 的 jaccard 相似性被定义为$|S \cap T|/|S \cup T|$,也就是说，两个集合的交于两个集合的并之比，我们定义这个相似性为 SIM(S,T)，两个集合的距离定义为 d(S,T) = 1 – SIM(S,T),例如，在图 2.1 中，我们可以看到两个集合 S 和 T,有三个元素在它们的交集中，而它们的并集中总共有八个元素，因此，它们的距离为：d(S,T) = 1 – SIM(S,T) = 5/8 。

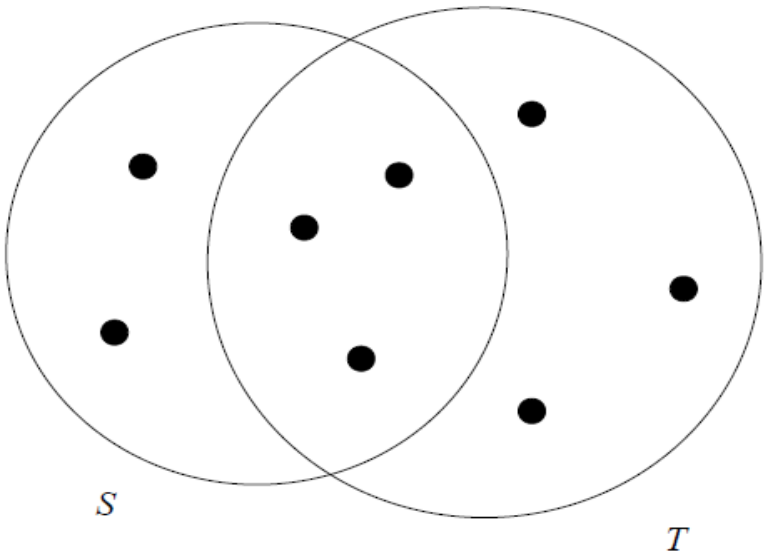

图 2.1 距离为 5/8 的两个集合

2.2.3 余弦距离

余弦距离在有维度的空间内有效，包括欧几里德空间和离散的欧几里德空间(空间内的点的向量坐标都是由整数或者二进制数(0 或 1)构成)。在这样一个空间里，空间点可以被视为一个向量。我们不区分一个向量与一个向量的倍数之间的关系，它们被视为同一个向量。于是两个点之间的余弦距离被视为由这两个点形成的向量之间的角度。不管这个空间是几维的，这个角度在 0 度于 180 度之间。

我们可以首先计算角的余弦值，然后通过这个余弦值再求出我们所需要的余弦距离。给定两个向量 x 和 y，角的余弦值为点集 x.y 除以 x 和 y 的 L_2 长度(即它们与原点之间的欧几里德距离)，回顾两个向量的点绩为：

$$[x_1,x_2,\ldots,x_n].[y_1,y_2,\ldots,y_n] = \sum_{i=1}^{n} x_i y_i \qquad \text{式(2-4)}$$

举例如下：给出两个向量 x = [1,-2,1]和 y = [2,1,1],点绩 x.y 为 1*2+2*1+(-1)*1=3, 两个向量的 L_2 长度均为$\sqrt{6}$，于是，x 和 y 的余弦距离为 1/2，所以，它们之间的角度为 60 度，也就是说它们的余弦距离为 60 度。

2.2.4 编辑距离

这个距离在点是字符串时使用，编剧距离即是对于两个字符串 $x = x_1x_2\ldots x_n$ 和 $y = y_1y_2\ldots y_n$,对它们做删除和插入单个字符操作使得两个字符串相等的最小操作次数。

举例如下，对于两个字符串 x = abcde 和 y = acfdeg ,为了使 x 与 y 相等，做如下操作：

1．删除 b

2．在 c 之后插入 f

3．在 e 之后插入 g

确信没有更少的操作使的两个序列相等，因此，x 和 y 的距离 d(x,y) = 3

2.2.5 汉明距离

给定一个向量空间，汉明距离被定义为两个向量之间它们坐标不同的数目。可以清楚的看到，汉明距离是非负的，当它等于零时，两个向量相等。其三角不等式也可以很容易证明：对于 x 和 z 它们有 m 个不同，y 和 z 它们有 n 个不同,则 x 和 y 的不同数不可能超过 m+n 个。汉明距离常常用于二进制向量空间(也就是坐标由 0 或 1 构成)中，举例如下：两个向量 10101 和 11110 之间的汉明距离是 3，因为，两个向量在第 2,4,5 个位置不同，在第 1,3 个位置相同。

2.3 局部敏感哈希函数库

局部敏感哈希算法研究至今，一些局部敏感哈希函数族已经被发现，我们在这章节里简单的概括一下它们。对于每一个函数族，我们呈现它们的哈希函数族，它们从哈希函数族随机选择哈希函数的过程以及它们的局部敏感特性。

2.3.1 汉明距离

对于$\{0,1\}^d$空间的二进制向量，Indyk 和 Motwani 在文[14]中提出了局部敏感哈希函数 $h_i(p) = p_i$, i 为从{1,…,d}中随机选择的一个索引。

同样可以很清楚的看到这个哈希函数族可以在汉明距离下直接应用于 M-ary 向量(即向量的坐标在{1,…,M}中)，一个简单的操作可以使得这个哈希函数族可以在 L_1 距离下处理 M-ary 向量。考虑$\{1,\ldots,M\}^d$空间内的任意点 p，一个简单的处理过程是把点 p 的一个坐标转换为一个二进制字符串 Unary(p)，具体方法是将每一个

坐标 p_i 替代为一个前面为 p_i 个 1 后面紧跟 M-p_i 个 0 的二进制串。可以直观的看到，对于任意两个 M-ary 向量 p 和 q,它们在 Unary(p)和 Unary(q)之间的汉明距离等价于 p 和 q 之间的 L_1 距离。要注意到的一点是，这种做法当且仅当在 M 是一个相当小的数字时有效。

2.3.2 L_1 距离

一个在 R^d 中作用于 L_1 距离下的更加直接的局部敏感哈希函数族在文[27]中被介绍，给定一个实数 w 远大于 R,然后强加一个随机移动于每一个宽度为 w 的单元格，每一个单元格定义一个桶。更明确的，选择随机实数 $s_1,s_2,\ldots,s_d \in [0,w)$,然后定义局部敏感哈希函数族为 $h_{s1,\ldots sd} = (\lfloor (x_1-s_1)/w \rfloor,\ldots,\lfloor (x_d-s_d)/w \rfloor)$ 。

2.3.3 Ls 距离

对于欧几里德空间，文[15]提出了以下局部敏感哈希函数族，选择一个从 R^d 空间到一维线段上的随机投影，在给予一个随机值的移动，该值为 $b \in [0,w)$,然后将它们截为宽度为 w 的线段区域,将其形式化为，$h_{r,b} = \lfloor (r.x + b)/w \rfloor$,映射向量 $r \in R^d$ 的每一个坐标由高斯分布中随机选取。

这种方法可以泛化为任何 Ls 距离，对于任意 $s \in [0,2)$,这个方法被证明是有效的。对于任意 s,我们必须从 s 稳定分布中随机选取变量构成向量 r，并使用该向量使用上述方式对高维数据进行降维处理。

2.3.4 jaccard 距离

正如前文中所介绍，jaccard 距离被用来衡量两个集合 $A,B \subset U$ 的相似性(例如，两个有单词组合成的文档)，不同于汉明距离，jaccard 距离用来衡量两个集合的相似性,文[28,29]的作者利用了 jaccard 距离，提出下面这个局部敏感哈希函数族，被称为 Min-hash。从背景集合 U 中选择一个随机排列，然后定义 $h_P(A) = \min\{P(a) | a \in A\}$,不难证明，冲突的可能性 $Pr_p[h_P(A) = h_P(B)] = s(A,B)$。文[30]中介绍了关于这个哈希函数的进一步理论发展。

2.3.5 余弦距离

对于余弦距离，Charikar 定义了下列一个局部敏感哈希函数族[31]，随机从 R^d 空间中选择一个单位向量 u,然后定义 $h_u(p) = sign(u.p)$。这个哈希函数可以被视使用一个随机选择的超平面将空间划分为两半。这里，设余弦的距离为 k,则冲突的可能性为：$Pr_u[h_u(p) = h_u(q)] = 1 - k/180$ 。

2.4 本章小节

本章主要概括讨论了局部敏感哈希算法的基本思想以及其相关的基础知识，首先介绍了局部敏感哈希算法的基本思想和方法，并给出了一个简明的算法；其

次介绍了与局部敏感哈希算法紧密相关的各种距离度量，最后，给出了一个局部敏感哈希函数族的函数库，介绍了基于不同距离下的一些具有局部敏感特性的哈希函数族。

第三章 几个常用的局部敏感哈希算法介绍

在上一章中，介绍了局部敏感哈希算法的基本思想，在本章中，将介绍几个常用的局部敏感哈希算法。

首先，我们会介绍在汉明距离下的局部敏感哈希算法，然后将介绍基于 P 稳定分布的局部敏感哈希算法，最后，我们将介绍一个针对局部敏感哈希算法性能改进的方法，即多探寻的局部敏感哈希算法。

3.1 基于汉明距离的局部敏感哈希算法

基于汉明距离的局部敏感哈希算法[14]由 Indyk 和 Motwani 在 1999 年提出，与首次提出的局部敏感哈希算法相比，该算法大大提高了局部敏感哈希算法的时间性能，它主要思想是通过将点的坐标转换为 01 序列从而将汉明距离与 L_1 距离统一起来。

在这章节里，我们使用 L_p^d 来定义在 L_p 规范下的欧几里德空间 R^d，例如，一个向量（$x_1,x_2,\ldots,x_k$）的长度被定义为（$|x_1|^p + |x_2|^p+\ldots+|x_k|^p$）$^{1/p}$,另外，$dp(p,q)=\|p-q\|_p$ 定义了 p,q 在 L_p^d 中的距离。我们使用 H^d 来定义 d 维的汉明距离空间，也就是说，在标准的汉明距离度量下的有 d 个二进制向量的空间。我们使用 $d_H(p,q)$来定义汉明距离，也就是 p,q 中的位不同的个数。

3.1.1 算法描述

因为基于汉明距离的局部哈希算法只能作用于非负整数向量上以及将与 L_1 距离关联，所以我们首先对数据做出如下两个不失一般性的假定：

1：距离被定义为 L_1 规范。

2：在 P 中所有点的坐标均为正整数。

第一个假定不是很严苛，有大量的研究人员证明，通常情况下，使用 L_2 规范与使用 L_1 规范相比，在近似搜索中,性能上没有显著的进步甚至差别。例如，在 web 检索的实验中，发现使用 L_1 规范与 L_2 规范返回近似的结果(甚至 L_1 规范的结果比 L_2 结果略好)。再如在文[32]中，他们对大量实际数据进行检测，对于在 L_1 距离下的与所给点的一定距离范围内的数据点集合与 L_2 距离下的集合仅有 3%的不同，也就是说，在大多数情况下，两个情况下的返回结果并无不同。更多的，如果所要应用的数据没有上述的特性，我们可以通过缩放以及旋转数据区域来使得相对于 L_2 距离而言我们仅损失很小的精度。我们来关注第二个假定，将所有的坐标转换为非负整数。在空间 R^d 中可以通过平移原点的方式使得所有的坐标为正，

接下来，我们可以通过将所有坐标乘以一个较大的数并四舍五入使得每个坐标都为整数，我们可以通过调整相乘的数，使的这一变化所产生的误差相当小。通过上述步骤，我们可以发现两个点的最近距离为 1 且都为非负整数。

下面我们将介绍具体的局部敏感哈希算法：

假设 c 为所有点的坐标中的最大值，对于点集 P 中的每一个数据点 p，我们将将坐标 p（$x_1,x_2,\ldots,x_d$）转化为汉明空间 H^{c*d} 中的二进制向量

$$v(p) = Unary_c(x_1)\ldots\ldots Unary_c(x_d) \quad \text{式(3-1)}$$

其中 $Unary_c(x)$表示为 x 个 1 后紧跟 c-x 个的 0 的 c 维二进制向量。

我们可以发现：对于任意一对点 p,q，它们的坐标在{1,2，…，c}中取值,有

$$d_L(p,q)=d_H(v(p),v(q)) \quad \text{式(3-2)}$$

也就是说，将点 p,q 化为如上的二进制向量后，它们的汉明距离与原本的 L_1 距离相等。因此，我们可以通过在 H^{c*d} 中使用汉明距离来解决近似最近邻问题。但是，我们要强调我们不整个处理数据的二进制向量，因为 c 可能是极大的而导致花费巨大，事实上这个算法的运行时间并不依赖与 c，二进制向量的表示给我们提供了一个很方便的方法。

我们定义哈希函数如下：给定一个整数 L(即所设置的哈希表数量)，从{1,2,…,c*d}中选择 L 个子集 $I_1,I_2,\ldots,I_L$,我们定义 p|I 为向量 p 在坐标集 I 上的映射，也就是说我们根据每个子集 I 所给出的坐标来选出 p 上的坐标点并把他们连接在一起。定义 $g_j(p) = p|I_j$，在处理过程中，我们存储 P 上的每一个点 p 到桶 $g_j(p),j=1,2,\ldots L$。桶的数量可能很多，我们通过传统的哈希方法将它们重新排列。因此我们使用两层的哈希函数：LSH 函数将点 p 映射到桶 $g_j(p)$,一个标准的哈希函数将这些桶映射到一个 M 大小的哈希表中。哈希表中的桶的尺寸大小被定义为 B，对算法分析而言，我们假设哈希表被定义为链表，即当一个桶被数据填满后，我们再分配一个桶链接在他后面存储数据，在实际中，我们不使用链表，而是使用一个更简单的方法，当一个桶被数据索引存满后，当有新的数据要存储时，我们将不再添加，我们假设它已经被其它哈希表的桶存储。桶的大小 B，点集 P 的大小 n，哈希表的大小 M，它们之间的关系按照下面的等式关联：

$$M = \alpha \frac{n}{B} \quad \text{式(3-3)}$$

α 是个内存利用率参数，也就是说，点的数量与哈希表的实际可放数量之比。图 3.1 示意了局部敏感哈希算法的哈希过程，它显示一个点通过函数 $g_i,i=1,\ldots,L$, 将点集 P 中的所有点 p 哈希到 L 个表中。

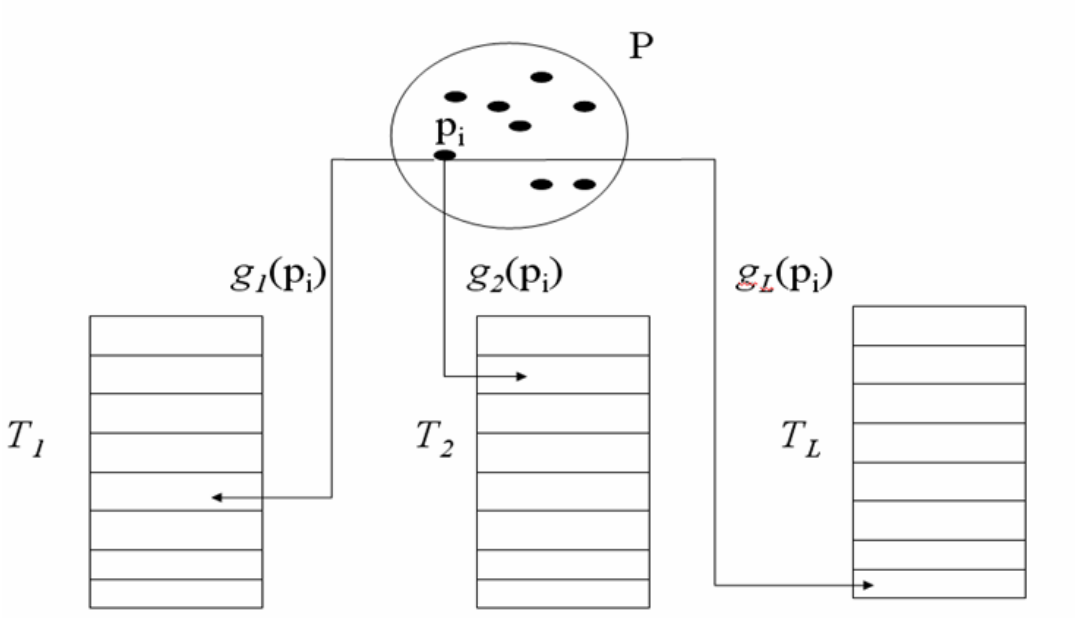

图 3.1 局部敏感哈希函数示意图

处理一个查询点 p，我们搜查索引 $g_1(q),g_2(q),\ldots,g_L(q)$直到找到 c.L 个点或者查询完所有 L 个表，$p_1$，…,$p_t$ 为查询到的点，显然，算法的磁盘读取次数以索引表的数量为上限，通常情况下，等于索引表数量 L。对于近似 K-NNS 问题，我们返回前 K 个距离最近的点，或者对于 t<k,我们返回不足 K 个点。

我们还需要特别指定子集 I_j 的选择，对于 j=1,2,…,L,I_j 包含 k 个从{1，…,C*d}个中随机选择的数，合适的 k 值应使得一个与点 q“相近”的点 p 的冲突的概率变大，而使一个与点 q“远离”的点 p 的冲突的概率减小，k 的选择将在下面指定。

在通过局部敏感哈希算法哈希之后，我们还需要标准的哈希函数将数据哈希到哈希表的桶中。

我们用前文所述的局部敏感哈希函数来产生一组向量$\{v_1,v_2,\ldots,v_k\}$,然后我们使用一个常规的哈希将数据哈希到表，哈希过程如下：

$$H(v_1,v_2,\ldots,v_k) = (a_1*v_1 + a_2*v_2 + \ldots + a_k*v_k) \bmod M \qquad \text{式(3-4)}$$

其中 M 为哈希表的长度，$a_1,\ldots,a_k$ 为随机从[1,M]中选择的数，这个函数可以仅使用 2k-1 个操作完成，并能满足我们的随机性要求。

尽管在该方法中主要关注 I/O 的复杂性，但是同样值得指出局部敏感哈希函数从 d 维空间映射到汉明空间时可以进行更加有效的计算。设 p 是数据集中的任意一点，定义 p'为经过映射后的数据，I 为坐标的集合，而我们想要的数据为经过映射后的 $p'|I$，对于 i = 1,2…,d，I|i 定义为 p 点的第 i 个坐标，注意到，I|i 为一系列的 01 串的长度及模式是固定的，也就是说，由每个坐标的 01 串构成(一串 1 后再连接一串 0)，我们把每个坐标的 01 串记为 o_i，i=1,…,d,给定一个选取的位置 w，在 I|i 中选取位置为 w 的 01 的任务等价于 u=w mod C 跟一个给定值(p 的第⌊w/C⌋

个坐标值)做比较,如果 u 大于该值，则选取的值为 0，反之，则选取的值为 1，这个过程所使用的时间为一个常数。因此函数总的计算时间等于 O(k)(k 为所选取的哈希函数个数)。

下面简单描述了算法的处理过程以及查询过程：

局部敏感哈希算法的数据处理存储过程

Algorithm Preprocessing

Input A set of points P, L(哈希表的数量)

Output Hash tables T_i, i = 1,…,L

1 Foreach i = 1,…,L

2 Initialize hash table T_i by generating a random hash function $g_i()$

3 Foreach i=1,…,L

4 Foreach j=1,…,n

5 Store point p_j on bucket $g_i(p_j)$ of hash table T_i

局部敏感哈希算法的查询过程

Algorithm Approximate Nearest Neighbor Query

Input A query point q , K(所要返回的近邻数据数目)

Access To hash tables T_i , i=1,…,L (由处理过程所生成的哈希表)

Output K(or less) appr. Nearest neighbors

1 Let S is a empty set

2 Foreach i=1,…,L

3 S = S∪{points found in $g_i(q)$ bucket of table T_i}

4 Return the K nearest neighbors of q found in set S

/*Can be found by main memory linear search*/

3.1.2 算法分析

LSH 算法的原理是两个点 p,q 之间冲突的可能性与它们的距离相关，也就是说，距离越近，它们冲突的可能性越大，距离越远，它们冲突的可能性越小。这个原理被形式化定义如下：以 D(.,.)来表示集合 S 中点的距离函数，对于任何属于 S 的 p 而言，定义 B（p,r）来表示集合 S 中与点 p 距离小于等于 r 的点。

注意到如果 D(.,.)是汉明距离函数 $d_H(.,.)$,则选择一个坐标的映射的函数族是局部敏感的，更具体的：

令 S 为 $H^{d'}$(d’为 C*d,$H^{d'}$为 d’维汉明空间)及 D（p,q）=d_H(p,q),p 和 q 为汉明空

间上的点，对于任意 r, ε 大于 0，可知哈希函数族 H = {$h_i = b_i$,i = 1,2,…,d',b_i 为汉明空间上的一个点}是（r,r(1+ ε),1-r/d',1-r(1+ ε)/d'）是局部敏感的。

我们现在可以把前述的算法归纳为一个抽象的局部敏感哈希算法，同时，这个方法同样适用于其它局部敏感哈希函数。这个归纳是很简单的，我们把函数 g 定义成如下形式：

$$g_i(p) = \{h_{i1}(p),h_{i2}(p),\ldots,h_{ik}(p)\} \quad \text{式(3-5)}$$

函数 $h_{i1}(p),\ldots,h_{ik}(p)$为随机从哈希函数族 H 中选择，如前所述，我们选择 L 个这样的函数 $g_1,g_{2,\ldots},g_L$,在这种情况下，当函数族 H 被使用，也就是,每个函数为选择空间 01 坐标中的一位，$g_i(p)$的结果基本上相当于 p|Ij 。

现在我们说明这个 LSH 算法可以用来解决(r, ε)-近邻问题：确定是否存在一个点 p 距离在 r1 = r of q 内，如果存在，算法要求我们返回一个与点 q 距离在 r(1+ ε) 内的点 p'。具体的，我们讨论通过选择合适的 k,L 参数来使用局部敏感哈希算法来解决这个问题。然后我们将说明如何使用这个算法来解决在不知道 r 的情况下的近似最近邻问题。

定义 P'为所有距离 d(q,p') > r2 的点 p'的集合，我们注意到这个 LSH 算法正确的解决(r, ε)-近邻问题，通过保持一下两个性质：

T1:如果存在 p*使得 p* ∈B(q,r1),那么存在 j 使得 $g_j(p^*) = g_j(q)$,j =1,2,…,L

T2:点 q 所指向的块仅包含属于 P'的点的块的数量总数小于 cL

假设 H 是一个(r1,r2,p1,p2)局部敏感哈希函数族，定义 ρ =(ln(1/p1))/(ln(1/p2)), 局部敏感哈希算法的正确性可以由以下定理得出：

定理 1：设 k=$\log_{1/p2}$(n/B),L=$(n/B)^{\rho}$,可以使得保持性质 T1,T2 的概率的可能性至少为 1/2-1/e≥0.132

证明：设保持性质 T1 的概率为 P1，保持性质 T2 的概率为 P2，我们将显示 P1 和 P2 都很大，我们假设存在一个 p*与点 q 的距离小于 r，k=$\log_{1/p2}$(n/B) ，对于 p'∈P-B(q,r2),g(p')=g(q)的概率最多为 p_2^k =B/n,由 g_j 分配的仅包含 P'中的点的块的数量的期望不会超过 2，所以对于所有 g_j，j=1,…,L，这种块的数量的期望不会超过 2L,因此，根据马尔科夫不等式，这个数量超过 4L 的概率小于 1/2,如果我们选择 c=4，则保持性质 T2 的概率 P2 小于 1/2。

现在我们注意保持 T1 性质的概率 P1，也就是 $g_j(p^*) = g_j(q)$的概率，显然，它的边界下线为：

$$p_1^k = p_1^{\log_{1/p1}(n/B)} = (n/B)^{-\frac{\log 1/p1}{\log 1/p2}} = (n/B)^{-\rho} \quad \text{式(3-6)}$$

令 L=$(n/B)^{\rho}$,我们可以得到对于所有 j=1,2,…,l, gj(p*) ≠gj(q)至多为 1/e,所以保持 T1 性质的概率 P1≥1-1/e 。

所以保持性质 T1，T2 的概率至少为 1-((1-P1)+(1-P2))≥1/2-1/e,定理得到论证。

接下来，我们讨论在不知道 r 的情况下的近似最近邻问题。首先，我们注意到我们可以根据前述算法通过不同的 r 值来构造一些数据结构来解决问题。特别的，我们可以扩展 r 为 r_0,$r_0(1+\varepsilon)$，$r_0(1+\varepsilon)^2$,…,r_{max} ，r_0 和 r_{max} 查询点与数据集合中的点的最近以及最远的距离。但是，我们注意到，不同 r 的数量将增加算法的运行时间以及存储空间。另一方面，我们注意到在实际选择中，只选择一个 r 值也足以能够产生足够好的质量答案。这个在文[39]中被解释，它观察到一个查询点与数据集之间的距离分布不依赖与特定的查询点，而是依赖于数据集的内在属性。在这种分布不变的假设下，一个相同的参数 r 适用于大多数的情况的查询。所以，在实际情况中，可以使用一个固定的 r 值，同样可以使用固定的 k 和 L 值。

3.2 基于 P 稳定分布的局部敏感哈希算法

前面我们介绍了基于汉明距离的局部敏感哈希算法，但它不能直接作用于实数，只能作用于非负整数。在这节中，我们介绍一个基于 P 稳定分布的局部敏感哈希算法[15]，它能直接作用于实数，且它可以作用在任意 L_P 距离下，$P\in(0,2]$。

3.2.1 P 稳定分布

稳定分布被定义如下：一组独立同分布的变量它们的归一化之和也被限制为该分布的变量。一个著名的稳定分布是高斯分布，同时，该稳定分布的存在也是相当广泛的，例如，还有 heavy-tail 分布等。具体定义如下：

稳定分布：对于一个在实数集 R 上的分布 D，如果存在 $P\geqslant 0$,对任何 n 个实数 $v_1,v_2,\ldots,v_n$ 和 n 个满足 D 分布的变量 $X_1,X_2,\ldots,X_n$ 随机变量$\sum_i(v_i.X_i)$和$(\sum_i|v_i|^p)^{1/p}.X$ 有相同的分布，其中 X 是服从 D 分布的一个随机变量，则称 D 为一个稳定分布。

根据已有资料可以知道，对于 $P\in(0,2]$都存在稳定分布，特别的

P=1,即 1 稳定分布，有柯西分布，概率密度函数为：

$$c(x)=\frac{1}{\pi}\frac{1}{1+x^2} \qquad \text{式(3-7)}$$

P=2，即 2 稳定分布，有高斯(标准)分布，概率密度函数为：

$$g(x)=\frac{1}{\sqrt{2\pi}}e^{-x^2/2} \qquad \text{式(3-8)}$$

我们注意到，很多情况下，尽管缺乏严格定义的密度与分布函数，但我们可以使用一个实际的方法，如文[33]显示，我们可以使用在[0,1]均匀分布的两个独立变量来产生 P 稳定分布变量。

稳定分布已经被广泛的应用在许多领域，在计算机科学中，Indyk 首次提出用稳定分布来处理高维特征向量，并且已经在许多基于内容的图像、视频检索中取

得很大成功。利用 P 稳定分布的特性可以设计有效的哈希函数族来处理高维特征向量，并能在保证数据间的距离相对不变的情况下，对高维数据进行降维。其主要方法是，产生一个 d 维的随机向量 a，随机变量 a 中的每一维都是随机的，独立的从 P 稳定分布中产生，对于一个 d 维的特征向量 v，根据稳定分布性质，随机变量 a .v 具有和$(\Sigma_i|v_i|^p)^{1/p}$.X 一样的分布（其中 X 是一个满足 P 稳定分布的随机变量），因此我们可以用 a.v 对向量 v 来进行降维，并可以用它来估算$||v||^p$,很容易看出这个过程是可以线性组合的，也就是说，$a(v_1-v_2) = a.v_1 – a.v_2$。

3.2.2 算法描述

在基于P稳定分布的局部敏感哈希算法中，算法将使用一个稍微不同的方法，并不直接利用点绩a.v代表v去估算$||v||^p$,而是使用它对每一个特征向量v赋予一个哈希值。很明显，该哈希函数族是局部敏感的，因为对于两个向量v_1和v_2，如果它们之间的距离很近($||v_1-v_2||$较小)，它们将以很高的概率发生碰撞(哈希值相同)，如果它们之间的距离很远，则冲突的概率将会很小。点绩a.v将每一个高维向量映射到一条实数线上。根据P稳定分布,两个向量v_1和v_2的映射距离$a.v_1-a.v_2$与$||v_1-v_2||^p$.X（其中随机变量X满足P稳定分布）是同分布的,a.v将高维向量v 映射到实数集R，如果将实轴以宽度r进行等分，并对每一段区域从左到右进行分段标号，如果a.v落在哪个区域，就将此区域标号作为哈希值赋给它，很明显，使用这种方法构造的局部敏感哈希函数在降维的同时能有效的保持两个高维数据之间的距离。

因此，每个局部敏感哈希函数形式化定义为如下格式,$h_{a,b}(v) : R^d \rightarrow N$，映射一个高维数据到一个整数集,其中a为一个d 维特征向量，每一维都是独立的，随机的从P稳定分布中选取的一个变量，b 为从[0，r]范围内随机选取的一个变量,对于给定的a, b，我们设置哈希函数：

$$h_{a,b}(v) = \left\lfloor \frac{a.v + b}{r} \right\rfloor \qquad \text{式(3-9)}$$

接下来我们计算两个向量v_1,v_2在从局部敏感哈希函数族中随机选择的一个函数下的冲突的概率，使用$f_p(t)$来定义P稳定分布的绝对值的概率密度函数，对于两个向量v_1,v_2，我们设$c=||v1-v2||^p$,对于一个随机从P稳定分布中独立选择的向量a, $a.v_1-a.v_2$与cX的分布相同(其中X是服从P稳定分布的一个随机变量)，b是一个独立从[0,r]中选择的随机变量，可以很容易的看出：

$$p(c) = \Pr_{a,b}\left[h_{a,b}(v_1) = h_{a,b}(v_2)\right] = \int_0^r \frac{1}{c} f_p(\frac{t}{c})(1-\frac{t}{r})dt \qquad \text{式(3-10)}$$

对于一个固定的参数 r,冲突的可能性随着 $c=||v1-v2||^p$ 的减小而增大。因此，根据上一节的定义，该哈希函数族使(r_1,r_2,p_1,p_2)-敏感的，$p_1=p(1),p_2=p(c),r_2/r_1=c$。因此，据上一节所述，该局部敏感哈希函数族可以用来解决近似最近邻问题。

经过一组局部敏感哈希函数哈希之后，我们得到一组哈希值 $h_1,h_2,\ldots,h_k$,将他们通过两个哈希函数存入到哈希表中：

$$H_1 = ((a_1*h_1+\ldots a_k*h_k)\bmod C) \bmod T \quad \text{式(3-11)}$$

$$H_2 = (b_1*h_1+\ldots b_k*h_k) \bmod C \quad \text{式(3-12)}$$

其中，a_i,b_i 为随机选取的整数，T 为哈希表长度，一般设置为 n,C 为一个大素数，在 32 位机器上可以设置为 2^{32}-1。我们通过 H_1 将数据哈希到哈希表中后，使用 H_2 通过链表将不同的值链接起来。

对于查询点 q,我们首先使用局部敏感哈希函数获得一组哈希值，然后我们使用 H_1 获得哈希表的位置，再计算 H_2 值，我们通过查询该位置的链表，获得与点 q 的 H_2 值相同值的点。最后，通过查询 L 个表，我们获得一组回收的点，通过对距离排序，我们获得 K(或者少于 K)个近邻点。

3.3 多探寻的局部敏感哈希算法

上述两小节我们介绍了两个局部敏感哈希算法，它们都设计了不同的局部敏感哈希函数族。在这一节中，我们将介绍一个多探寻的局部敏感哈希算法，它不设计新的局部敏感哈希函数族，它从另一个角度改进现有的局部敏感哈希算法。因为在上述算法中，每一个哈希表存储全部的数据，而且在一个算法中，我们往往需要数十个至数千个不等的哈希表，也就是说，局部敏感哈希算法的空间需求较高。在多探寻的局部敏感哈希算法[17]中，它通过在哈希表中回收更多的桶的数据点从而提高哈希表的利用率，从而减少局部敏感哈希算法的空间需求。

3.3.1 基于熵的局部敏感哈希算法

Panigrahy 提出了一个基于熵的局部敏感哈希方法，它使用基本的局部敏感哈希方法构造哈希表，但是使用一个不同的查询过程。假设我们知道一个近邻 p 于查询点 q 的距离 R_p。理论上，对于每一个哈希桶，我们可以计算 p 在那个哈希桶的可能性(称这个为哈希桶的成功概率)。给定所述信息，我们可以查询那些成功概率高的哈希桶，但是，计算哈希桶的成功概率的任务量相当大。Panigrahy 提出了一个简单的方法来寻找这些成功概率高的桶。每一次，在于查询点 q 距离 R_p 的范围内随机产生一个点 p',然后查询 p'所对应的哈希桶，这些桶具有很高的成功概率。通过多次使用这个简单的方法来保证那些成功概率高的哈希桶被探寻到。

但是，这个方法有几个缺点：每次查询的过程是低效的，因为产生扰动点并计算它们的哈希值是比较慢的，另一方面，它必然会产生重复的桶，特别的，成功概率高的桶会被查询多次，它的大量计算时间被浪费了。尽管可以记住先前已

经查询过的哈希桶，但是，当有很多的查询同时发生时，总的开销还是很高。另外，成功概率较低的桶也会被查询到，而这不是我们所需要的。另一个缺点是这个方法需要知道距离 R_p，它需要一个依赖于数据的方法来得到。而且当 R_p 较小，不能产生足够的扰动点来获得足够的哈希桶。而当 R_p 较大，它需要查询很多的扰动点来获得足够好的查询结果。

为了解决这些缺点，下文将提出一个多探寻的局部敏感哈希方法，它提供一个更加系统的方法来获得那些成功概率高的桶。

3.3.2 算法概述

多探寻的局部敏感哈希算法的核心思想是使用一组仔细准备的序列去查找多个很有可能含有查询点近邻的哈希桶。根据局部敏感哈希算法的性质，我们知道如果一个数据点靠近查询点 q，但并不跟 q 哈希到一个桶中，那么它很有可能在一个“邻近”桶中(也就是说，两个哈希桶的一组哈希值仅有些许不同)，因此，我们的目标是找到这些“邻近”的哈希桶，来增加找到与查询点 q 邻近的点的机会。

我们定义一个扰动序列向量 $\Delta = \{\delta_1, \ldots, \delta_k\}$,给定一个查询点 q，基本的局部敏感哈希方法查询哈希桶 $g(q) = \{h_1(q),\ldots,h_k(q)\}$,当我们应用扰动序列 Δ，我们将探询桶 $g(q)+\Delta$。

假定我们使用基于 P 稳定分布的局部敏感哈希函数 $h_{a,b}(v)$，因此，如果我们选择的 r 足够大，邻近的数据点将以很高的概率落在相同或者邻近的值上(也就是说，相差不超过 1)，因此，我们约束 Δ 的取值范围为{-1,0,1}。

每一个扰动向量将直接作用于查询对象的哈希值，所以，与基于熵的局部敏感哈希方法相比，节省了计算扰动点的哈希值的时间。更多的，我们将设计一组扰动序列作用于哈希值的集合使得查询一个哈希桶的次数不会超过一次。

图 3.1 显示了多探寻的局部敏感哈希方法的工作原理，在图中，$g_i(q)$是在第 i 个表中的查询点 q 的哈希值，$(\Delta_1, \Delta_2, \ldots)$是一组探寻序列，$g_i(q)+\Delta_1$ 是应用扰动向量 Δ_1 后产生的新的哈希值，它指向表中的一个新的哈希桶，通过使用多个扰动向量，我们可以获得多个与 q 指向的哈希桶“邻近”的桶，这些桶中可能含有与 q 邻近的元素。在下面小节中，我们将主要讨论如何产生一组扰动序列。

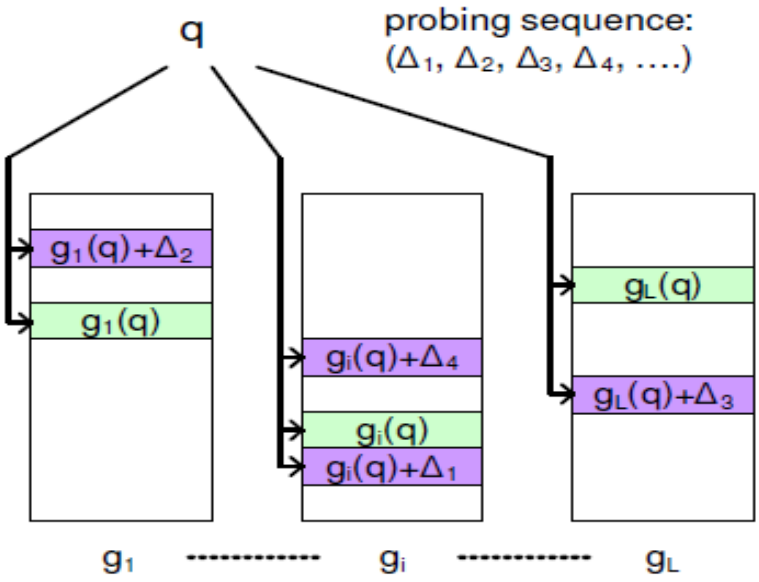

图 3.2 多探寻的局部敏感哈希方法示意

3.3.3 步进式的探寻

定义一个 n 步的扰动序列 Δ 有 n 个非零坐标。这个扰动序列对应于查询一个哈希桶(它与查询点所在的哈希桶的一组哈希值有 n 个不同)。根据局部敏感哈希算法的性质，只有一步距离的哈希桶(也就是说，在查询点的一组哈希值中，仅有一个不同)比二步距离的哈希桶更有可能包含查询点的邻近的数据点。

上述就是步进式探寻的方法来源，我们首先检查一步距离的桶，然后是二步距离的桶，直到我们找到足够的哈希桶。而在大多数情况下，一个 2 步内的步进式探寻足以保证我们可以找到大部分成功概率较高的桶。对于一个含有 L 个哈希表以及 K 个敏感哈希函数的局部敏感哈希算法，n 步的查询桶的总数为 $L \times C_K^n \times 2^n$，而 s 步内的查询哈希桶的总数为：

$$L \times \sum_{n=1}^{s} C_K^n \times 2^n \qquad \text{式(3-13)}$$

对于基于 P 稳定分布的局部敏感算法，我们可以对哈希值做+1 或者-1 扰动，注意到，对于基于汉明距离的局部敏感哈希算法，它的哈希值为 0 或 1，也就是说，在它等于 0 时，它不能做-1 扰动，而它在等于 1 时，不能做+1 扰动。我们使用一个简单的方法来替代它，当我们选定一个哈希值时，我们将它做 01 变换，也就是说，若它为 0，则将它变为 1，若它为 1，则将它变为 0。

3.3.4 一个更精确的序列产生方法

在步进式的探寻方法中，查询点 q 的所有哈希值被同等的对待，也就是说，每个哈希值的被扰动的概率是一样的。在这节中，我们介绍一个针对基于 P 稳定分布的局部敏感哈希算法的一个方法，它相较于步进式的探寻方法，能更精确的产生一组扰动序列。

首先，我们关注基于 P 稳定分布的局部敏感哈希算法的哈希值产生方法。注意到每个哈希函数 $h_{a,b}(q)$ 将查询点 q 映射到一条直线上，这条直线以 r 为宽度从左到右划分为不同的区域，得到的哈希值就是点 q 所要映射到的区域，一个与点 q 相邻近的点 p 可能落在与点 q 一样或相邻的区域，事实上，点 p 落在点 q 右边(左边)的区域取决于点 q 落下的位置与它落下的区域的右边界(左边界)的距离，所以，k 个哈希函数中查询点 q 在它映射到的区域的位置对于决定扰动向量是很有用的。接下来，我们将利用上述信息(即查询点 q 在它所在的区域的位置信息)来设计一个精确的方法产生扰动序列。

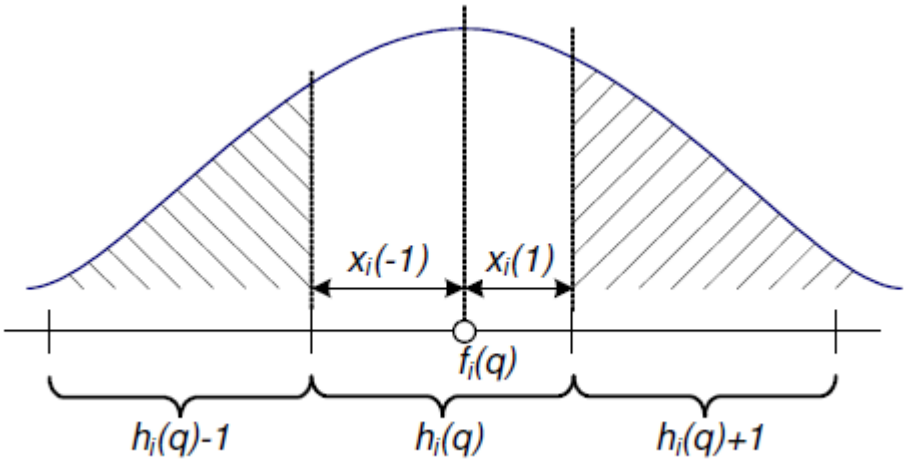

图3.3 查询点q的近邻落点概率

图3.2说明了查询点q的近邻元素落在邻近区域的概率。这里$f_i(q) = a_i.q + b_i$是在使用第i个哈希函数后，查询点q在直线上的映射，$h_i(q)$ 是查询点q哈希到的区域，对于 $\delta \in \{-1,1\}$,我们定义$x_i(\delta)$为查询点q到区域$h_i(q)+\delta$边界的距离，也就是说，$x_i(-1)= f_i(q)- h_i(q)*r, x_i(1) = r – x_i(-1)$,方便起见，我们定义$x_i(0)=0$。对于任何一个固定的查询点p，$f_i(p)- f_i(q)$是一个服从高斯分布的随机变量，这个随机变量的方差与点p和点q的距离成比例，我们假设选择r足够大使得我们感兴趣的点p以足够高的概率落在区域$h_i(q)$，$h_i(q)-1$，$h_i(q)+1$上，注意到高斯随机变量的概率密度函数为e^{-x^2/σ^2}，所以，点p落在区域$h_i(q)+\delta$的概率可以估计为：

$$\Pr[h_i(p) = h_i(q) + \delta] \approx e^{-C \cdot x_i(\delta)^2} \quad \text{式(3-14)}$$

C 是一个依赖于点 p 和点 q 之间的距离的常量。

我们现在来估计扰动向量 Δ 成功(找到与查询点 q 相近的点 p)的可能性：

$$\Pr[g(p)=g(q)+\Delta]=\prod_{i=1}^{K}\Pr[h_i(p)=h_i(q)+\delta_i]$$
$$\approx\prod_{i=1}^{K}e^{-C.x_i(\delta)^2}=e^{-C.\sum_i x_i((\delta_i)^2)} \quad \text{式(3-15)}$$

这表明，我们可以讲扰动序列成功的可能性相关与：

$$score(\Delta)=\sum_{i=1}^{K}x_i\left(\delta_i\right)^2 \quad \text{式(3-16)}$$

也就是说，得分越低的扰动向量所产生的新的哈希桶找到与查询点相近的点的概率越高，得分越高的扰动向量所产生的新的哈希桶找到与查询点相近的点的概率越低。注意到，扰动序列函数的分数仅依赖于查询点 q，所以我们可以直接依据查询点 q 中获得扰动向量，并按其得分从小到大来产生扰动向量。

上面我们阐述了这个方法的具体思想，下面我们将依据于前文所述的信息来具体描述这个扰动序列的产生过程。

一个很不好的生成扰动序列的方法是生成所有的序列，计算它们的得分并对其进行排序。但是，可知总共有 $L*(2^K-1)$个扰动序列，而我们只希望准确的使用它们其中分数最小的一部分，所以生成所有的扰动序列是极浪费的。下面描述了一种按得分的升序来产生扰动向量的更加有效的方法。

首先，我们注意到扰动向量 Δ 的得分仅依赖于那些 Δ 中的非零坐标(因为对于 $\delta=0, x_i(\delta)=0$)，我们预计得分低的扰动向量仅含有一些非零坐标。在扰动序列生成中，我们把向量的非零坐标表示为一组(i,δ_i)对，一个(i,δ)对表示在查询点 q 的第 i 个哈希值上加上 δ 。

给定一个查询点 q 和一个哈希表对应的哈希函数 $h_1, i=1,2,\ldots,K$,对应于一个单一的哈希表，我们首先计算 $x_i(\delta), i=1,2,\ldots,K$, $\delta\in\{-1,1\}$,我们将这 2K 个元素从小到大进行排序，并定义 z_j 为该序列中的第 j 个元素。定义 $\pi_j=(i,\delta)$如果 $z_j=x_i(\delta)$, 这表示值 $x_i(\delta)$是在序列中第 j 个小的值。注意到 $x_i(1)+x_i(-1)=r$，如果 $\pi_j=(i,\delta)$，那么 $\pi_{2K+1-j}=(i,-\delta)$。现在我们把扰动向量表示成集合{1,2，…,2K}的子集，相当于一个扰动集。对于每一个这样的扰动集 A，相应的扰动向量为根据扰动集合 A 从集合$\{\pi_j | j\in\{1,\ldots,2K\}\}$中获得 $\{\pi_j | j\in A\}$的扰动坐标及值。每一个扰动集 A 有相应的得分 score(A)为属于集合 A 的 z_j 值的平方和，这与其对应的扰动向量的得分相等。因此给定一个排好序的(i,δ_i)对的队列 π 以及值 z_j，$j=1,\ldots,2K$,产生扰动向量的问题可以等价为从集合{1,2，…,2K}按其得分从小到大产生扰动集。

我们定义了在扰动集上的两个操作：

shift(A):这个操作将用元素 1+max(A)来替代集合 A 中的元素 max(A)，例如，shift({1,3,4})={1,3,5};

expand(A):这个操作扩充集合 A，将元素 1+max(A)添加到集合 A 中，例如，shift({1,3,4})={1,3,4,5}

通过这两个操作，我们显示我们如何产生 T 个扰动集的算法，算法具体描述如下：

Algorithm Generate T perturbation sets

1 $A_0 = \{1\}$

2 minHeap_insert(A_0,score(A_0))

3 for i = 1 to T do

4 repeat

5 A_i = minHeap_extractMin()

6 A_s = shift(A_i)

7 minHeap_insert(A_s,score(A_s))

8 A_e = expand(A_i)

9 minHeap_insert(A_e,score(A_e))

10 until valid(A_i)

11 output A_i

12 end for

算法显示了我们如何产生 T 个扰动集的过程，一个最小堆用来维护候选的扰动集(其父母集合的得分不会大于其子集合)，堆最小初始化为{1}，每一次我们得到最顶端的集合 A_i 并将 shift(A_i)和 expand(A_i)加入堆中。只有有效的顶点集合 A_i 才会被输出，注意到，对于任何 j=1,2,…,K, π_j 和 π_{2K+1-j} 表示在同一个坐标上做相反的操作。所以在有效的扰动集中最多有{j,2K+1-j}中的元素的一个，更明显的，扰动集中有大于 2k 的元素也是无效的。图 3.2 显示了扰动集的产生过程：

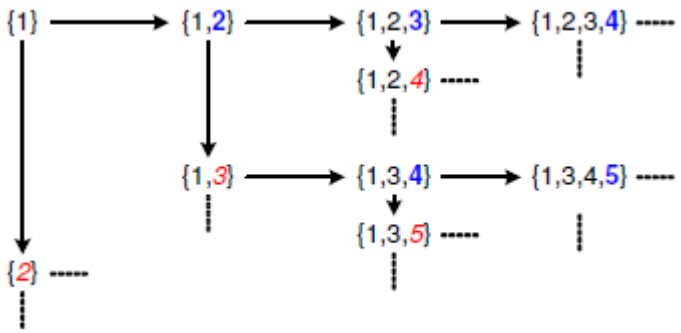

图 3.3 扰动集的产生过程

注意到下面两个关于 shift 和 expand 操作的性质对于保证上述产生扰动集过程

的正确性很重要。

1：对于任意的扰动集 A，集合 shift(A)和 expand(A)的得分都比 A 的得分高。

2：对于一个任意的扰动集 A，有一个单一的 shift 和 expand 过程从{1}开始到直到生成 A。

基于上面两个性质，很容易得出下面两个正确的性质：

性质 1：该产生过程正确的按集合的得分从小到大产生了所有有效的扰动集。

性质 2：在任何时候，堆中的元素数量总是比一轮操作前的元素数量多一个。

为了简化描述，我们只是描述了在一个单一的哈希表上的扰动序列的产生过程。事实上，我们必须对所有的 L 个哈希表产生扰动序列，对于每一个哈希表，我们维护一个排好序的(i, δ_i)对的队列以及值 z_j。但是我们可以只使用一个最小堆来同时对每个哈希表产生扰动集。每一个在堆中的坐标扰动集合都于一个哈希表 t，t = 1,…,L 相关联。我们初始化 L 份集合{1}的拷贝，并将对这些集合的操作分别于不同的哈希表 t 相关联。

3.3.5 扰动序列的构造优化

上一节所阐述的扰动序列构造方法通过维护一个堆以及在查询时查询堆来生成扰动向量。我们现在来描述一种方法来在查询时维护和查询这个堆的开销。为了做到这些，我们通过预先计算一组序列来替代在查询时对堆的查询以及插入，从而减少扰动序列的生成时间。

注意到扰动序列的生成可以被分为两个部分：(1) 按顺序生成扰动集合，(2) 将每个扰动集合映射为一个扰动向量。第一个部分需要值 z_j 而第二个部分需要一个映射 π 将集合{1,…,2K}映射到(i, δ)对。这些都是基于查询点 q 的函数。

稍后我们将会解释，事实上我们可以预先精确的知道值 z_j 的分布并对于每个 j 计算 $E[(z_j)^2]$,所以，我们可以做如下优化：我们通过这个期望来近似值$(z_j)^2$。使用这个近似，我们可以事先按顺序计算扰动集合(因为集合的分数是值$(z_j)^2$ 的函数)。这个生成的过程可以于上一节描述的过程一样，但是使用值 $E[(z_j)^2]$来替代它们真实的值。这个可以独立于查询点 q 完成。在查询时，我们根据查询点 q 来计算映射 π (每个哈希表是不同的)。这些映射将预先准备好的每个扰动集合转换为扰动向量。这个预先处理的过程减少了在查询时动态生成扰动集合的时间。

为了完成这个方法，我们还需要解释如何来获得 $E[(z_j)^2]$。回顾值 z_j 是排列好的值 $x_i(\delta)$中的一个值。注意到 $x_i(\delta)$时在[0,r]中均匀分布，另外 $x_i(-1)+ x_i(1) = r$, 因为 K 个哈希函数中的每一个都是独立选择的，对于 $j \neq i$,值 $x_i(\delta)$独立于值 $x_j(\delta)$。值 $z_j(j=1,\ldots,K)$的联合分布如下所述：随机从均匀分布[0, r/2]中选择 K 个数，z_j 是这个集合中第 j 个大的数。这是一个已经被很好研究过的分布，使用已知的关于这个

分布的知识，对于 j∈{1,…,K},我们可以得到：

$$E[z_j^2]=\frac{j(j+1)}{4(K+1)(K+2)}r^2 \qquad \text{式(3-17)}$$

对于 j∈{K+1,…,2K},我们可以得到：

$$E[z_j^2]=(1-\frac{2K+1-j}{K+1}+\frac{(2K+1-j)(2K+2-j)}{4(K+1)(K+2)})r^2 \qquad \text{式(3-18)}$$

正如先前所述，这些值可以用来预先计算扰动集合的顺序。

3.4 本章小结

本章具体介绍了三个不同的局部敏感哈希算法。首先，我们介绍了两个基本的局部敏感哈希算法，它们设计了不同的局部敏感哈希函数族，从而完成将数据哈希到哈希表的过程；然后，我们介绍了一个在基本的局部敏感哈希算法之上的改进方法，它通过在哈希表中生成扰动序列来查询更多的高成功性的哈希桶来提高哈希表的利用率，从而减少算法的空间需求。

第四章 对局部敏感哈希算法的改进

上一章中，我们介绍了两个基本的局部敏感哈希算法，通过设计有效的局部敏感哈希函数使算法发生作用，另外，我们介绍了对于一个基本的局部敏感哈希算法改进的多探寻的方法，它通过在同一个哈希表中，查找与查询点 q 所哈希的桶的“邻近”哈希桶并回收其中的数据，从而提高哈希表的利用率。在这一章中，我们关注回收数据的方法以及对于回收数据的处理方法，设计一个新的数据回收方法以及一个处理回收数据(返回查询结果)的方法，从而改善局部敏感哈希算法的性能，提高算法的时间效率。

4.1 改进的动机

回顾前文所述的基本的局部敏感哈希算法，我们从一个(r_1,r_2,p_1,p_2)敏感哈希函数族中随机选取 k 个函数$\{h_1,h_2,\ldots,h_k\}$构成一个函数将带点从 d 维空间映射到哈希表中。它以较高的概率将相近的点映射到一个桶中，但是，这并不代表其它相距较远的点不能进入到该桶中，也并不代表所有与某点相近的点都能哈希到该桶中。所以我们需要 L(或许从几十到几千)个表来使我们可以从表中回收足够的数据点以保证我们可以获得令人满意的结果。特别的，对于多探寻的局部敏感哈希算法，它通过在一个哈希表中查找更多的桶来减少哈希表的数量，但这不代表它减少了回收的数据点，恰恰相反，对于在一个哈希表中查找那些与查询点 q 所映射的桶相“邻近”的桶，它们于查询点 q 所直接映射的桶相比，它们仅能以次高的概率发现查询点 q 的近邻数据，也就是说，为了保证获得令人满意的数据结果，我们需要比基本的局部敏感哈希算法查询更多的桶，回收更多的数据点。然后，在获取足够的数据点之后，我们计算每个点与查询点的距离，并按它们的距离进行排序，然后返回 K 个数据结果。注意到我们回收的数据点数量相当大，这就意味着我们需要花费相当多的时间来处理回收的数据点(包括对数据点与查询点间的距离计算以及将数据点按距离进行排序)。在下面，我们讨论一种新的对回收数据处理的方法。我们改进回收数据的方法，并使用一种新的标准来衡量数据，从而改善这一过程的时间效率。

注意到局部敏感哈希算法的一个关键性质。给定两个数据点 x 和 y,以及一个查询点 q，我们假设 $Dis(x,q) = r_1$, $Dis(y,q) = r_2$, $r_1 < r_2$,对于任意一个哈希函数 h_i,我们都有 $p(r_1) > p(r_2)$,也就是说与点 y 相比，点 x 更容易与查询点 q 发生冲突。给定一个哈希表的联合函数 $g(p) = \{h_1(p),h_2(p),\ldots,h_k(p)\}$,在哈希表中，数据点 x 与查询点 q 在同一个哈希桶中的概率为$(p(r_1))^k$, 同样，数据点 y 与查询点在同一个哈希桶中的概率为$(p(r_2))^k$,根据数学性质，通过调整 k 的大小，我们可以扩大 $p(r_1)$和 $p(r_2)$

的差距，使得$(p(r_1))^k$ 远远大于 $(p(r_2))^k$。例如，若设 $p(r_1) = 0.95$，$p(r_2) = 0.8$，令 k=30, 我们可以得到$(p(r_1))^{30} = 0.21$, $(p(r_2))^{30} = 0.001$,也就是说，在一个局部敏感哈希算法的哈希表中，与查询点相近的数据点出现在于查询点一样的桶中的可能性远远大于与查询点相距较远的点。注意到，对于多探寻的局部敏感哈希算法，这一特性同样成立。因此，给定 L 个局部敏感哈希算法的哈希表，我们可以预期与查询点相距较近的点在 L 个表的回收数据集中将会出现更多次数。所以，我们可以根据数据点的出现次数来对它们排序来近似它们真实的排序(数据点的真实排序通过计算数据点与查询点的距离，并依据距离对它们进行排序，正如前文所述，这一过程的时间花费相当大)。

注意到，目前的局部敏感哈希算法并没有在数据回收过程中依据数据的出现信息对重复的数据以及其它较不可能的数据进行排除。在下文中，我们使用这一信息来改善数据回收的过程。并在数据回收之后，提出使用新的方法来获得所需要的数据结果。在第五章中，通过实验数据结果，我们将显示在保持数据搜索质量的情况下，我们极大的改善了数据的搜索时间。

4.2 算法描述

下面我们具体阐述改进的算法。在算法中，主要有三个步骤：(1)哈希表的构造，这个步骤与前文所提到的局部敏感哈希算法相同(例如基于汉明距离的局部敏感哈希算法，多探寻的局部敏感哈希算法等)；(2)数据回收，这一步也基本与前文所提到的算法相同，但是在数据存入回收表(即我们存储回收数据的空间)的时候，我们引入数据的出现次数来对数据进行处理;(3)数据处理，与前文所述算法不同，在这一步骤中，我们不使用距离来衡量并得出搜索结果，我们引入数据点出现次数来近似数据与查询点的距离，从而获得数据的查询结果。具体阐述如下：

4.2.1 哈希表的构造

对于我们前文所述的三个局部敏感哈希算法而言，哈希表的构造方法主要有两种：(1)基于汉明距离的局部敏感哈希算法，在这个算法中，我们设置敏感哈希函数族 H 为$\{h_i = i \mid i \in H^{C*d}\}$,即从数据点的汉明空间中随机选择一位，对于每一个哈希表，我们引入 $g_i() = \{h_1,h_2,\ldots,h_k\}$，i=1,…,L,来构造哈希表，存储数据索引；(2)对于基于 P 稳定分布的局部敏感哈希算法和多探寻的局部敏感哈希算法而言，我们设置它们的哈希表构造方法是一样的，它们都使用同一个敏感哈希函数族，设计哈希函数为：

$$h_{a,b}(v)=\left\lfloor \frac{a.v+b}{r} \right\rfloor \quad \text{式(4-1)}$$

a 为一个 d 维的向量(每一维的坐标都独立从 P 稳定分布中产生，b 为[0,r]的一个随机变量)，我们通过 a 和 b 的变动得到这个哈希函数族 H，同样的，对于每一个哈希表，我们设置 $g_i() = \{h_1,h_2,\ldots,h_k\}$，i=1,…,L,来构造哈希表，存储数据索引，但是注意到哈希数值较大，我们引入链表来存储数据，方法具体如下：

经过一组局部敏感哈希函数哈希之后，我们得到一组哈希值 $h_1,h_2,\ldots,h_k$,将他们通过两个哈希函数存入到哈希表中：

$H_1 = ((a_1*h_1+\ldots a_k*h_k) \bmod C) \bmod T$ 式(4-2)

$H_2 = (b_1*h_1+\ldots b_k*h_k) \bmod C$ 式(4-3)

其中，a_i,b_i 为随机选取的整数，T 为哈希表长度，一般设置为 n,C 为一个大素数，在 32 位机器上可以设置为 2^{32}-1。我们通过 H_1 将数据哈希到哈希表中后，使用 H_2 通过链表将不同的值链接起来。

4.2.2 数据回收

给定一个经过局部敏感哈希算法处理的 L 个哈希表，我们要获得查询点 q 附近的数据点。对于基于汉明距离的局部敏感哈希算法和基于 P 稳定分布的局部敏感哈希算法而言，它们的数据回收过程是一样的：对于每一个哈希表，根据与该表相对应的局部敏感哈希函数，我们获得查询点 q 所在的哈希桶，在我们收集数据的同时，我们记录每个数据的出现次数，具体的，对于一个没有出现过的数据，我们将它加入回收的数据表并把它的出现次数设为 1，对于一个已经出现过的数据，我们把它的出现次数加 1。算法的伪代码描述如下：

```
Alrotithm Data Collect One(q)
Input: query object q(需要查询的点 q)
Access To hash tables T_i , i=1,…,L (由处理过程所生成的哈希表)
Output: a hash table P which contains candidate objects and their occurrence count
1    make P is an empty hash table
2    for i = 0 to L-1 do
3       A = getBucket(q,i)
4       for j = 0 to A.length-1 do
5          if P.containsId(A(j)) then
6                count = p.getValue(A(j))
7                count = count +1
8                P.update(A(j),count)
9          else
```

```
10                  P.put(A(j),1)
11              end if
12          end for(j)
13  end for(i)
```

对于一个多探寻的局部敏感哈希算法而言，给定一个经过局部敏感哈希算法处理的 L 个哈希表，我们要获得查询点 q 附近的数据点的过程如下：对于每一个哈希表，我们假设我们在每一个哈希表中要查询 M 个桶的数据，我们开始收集 M 个哈希桶的过程：首先我们根据每个哈希表所对应的哈希函数收集查询点 q 本身所映射的桶，然后我们再依据前文算法，给 q 以一个得分从小到大的扰动向量 v，然后收集扰动后的点 q'所映射的桶，直到我们在哈希表中收集到足够多哈希桶的数据点，注意到查询点 q 本身也可以被视为经过一组{0,…,0}扰动向量扰动，因此我们可以统一收集查询点 q 以及扰动后的查询点 q'的哈希桶的过程(通过在扰动向量的首部添加一组{0,…,0}向量)，算法的伪代码描述如下：

Alrotithm Data Collect Two(q)

Input: query object q(需要查询的点 q)，a set of perturbation v(一组扰动向量)

Access To hash tables T_i , i=1,…,L (由处理过程所生成的哈希表)

Output: a hash table P which contains candidate objects and their occurrence count

B[0,…,L-1] = {0,…,0} (B 是一个数组，用来记录每个哈希表中所查询的哈希桶)

```
1   make P is an empty hash table
2   for i = 0 to L-1 do
3       t = 0
4       while(B[i] < M) do
5           q' =q + v[t]
6           A = getBucket(q',i)
7           for j = 0 to A.length-1 do
8               if P.containsId(A(j)) then
9                   count = p.getValue(A(j))
10                  count = count +1
11                  P.update(A(j),count)
12              else
13                  P.put(A(j),1)
14              end if
15          end for(j)
16          B[i] = B[i] +1
```

```
17            t = t + 1
18        end while
19    end for(i)
```

4.2.2 数据处理

在经过数据回收后，我们获得一个数据点的集合，同时我们也获得每个数据点的出现次数，我们将根据这一信息来近似数据点与查询点的距离，获得最终的数据结果。注意到，不论对于哪一个版本的局部敏感哈希算法，每一个数据点在一个哈希表中最多出现一次，所以，对于 L 个哈希表，数据点最多只能出现 L 次，也就是说，数据点的出现次数最多为 L。根据这一信息，我们设置数组 S[L],它存储 L 个链表的指针，分别指向出现次数为 1,…,L 的数据点的链表，显然，这个过程可以在 O(n)的时间内进行完毕(n 为回收的数据个数)，根据这个数组，我们对回收的数据集合 P 进行处理，可以得到依据数据出现次数分类的 L 个链表。然后，我们可以对数据点进行处理，对于 K 最近邻查询问题，我们可以从高出现次数(从 L 开始)数据的链表回收起，依次查询出现次数从高到低的链表，直到获得 K 个最近邻数据。显然，这一过程可以在 O(K)的时间内处理完毕。所以，整个算法的处理过程时间为 O(n+K),这一过程的算法如下所述：

```
Algorithm Data Process
Input: candidate set table P,the size of query result K
Output: a set of Kobjects as the KNN query result
S[1,…,L] = sort(P) (S 为 L 个依据数据出现次数而分别存储的链表)
1    make R is an empty set
2    for i = L-1 to 0 do
3        if R.size>= K then
4            break
5        end if
6        for j= 0 to B[i].length-1 do
7            R.put(B[i][j])
8            if R.size >= K then
9                Break;
10           end if
11       end for(j)
12   end for(i)
13   return R
```

4.3 本章小节

在本章中，我们介绍了对局部敏感哈希算法的一个改进。我们首先介绍了这一改进的动机以及改进的所依赖的性质与方法，然后，我们具体阐述了这个方法对于前述章节的三个局部敏感哈希算法进行改进后的算法，显然，我们可以直观的看到这一方法改善了对于上述三个局部敏感哈希算法的时间效率。

第五章 实验

在这一章节里，我们首先将介绍实验相关信息，包括实验数据，算法实验性能的评价标准，实现的算法版本以及其它一些相关细节。然后我们将呈现实验结果并对实验结果做一定分析。

5.1 实验数据

在实验中，我们将使用两组高维数据，一组提取图像特征得到的高维数据以及一组模拟高维数据，具体介绍如下：

SIFT 数据：

这个数据集[34]是由一个在线图像数据集经过尺度不变特征变换方式提取得到的一组向量集合。SIFT 数据关心的是图像的不变特征，因此特征匹配时仅考虑特征点的 128 维特征描述向量，而忽视位置横坐标，位置纵坐标，尺度和方向四个参数。由上所述，SIFT 数据向量为 128 维，并且把每一维向量都表示为 0 至 255 的整数。在我们的实验中，我们从该数据集中选取 10 万个该数据向量。

模拟数据：

对于算法，我们常用随机生成的模拟数据进行测试。对于 d 维向量，对于每一维，我们可以独立的从一个区域中随机选择一个数作为它的坐标。但注意到，对于如此选择的 d 维向量，在高维向量空间中，给定一个查询点，我们会发现大量的数据点集中在于查询点相同的距离的位置。这种情况并不能正确反映真实世界的数据情况，我们通过在点附近插入它的近邻点来改善这种情况，模拟真实世界的数据。具体做法如下：

1 产生一组随机产生的高维数据，他的每一维独立的从[a,b]中随机选取。

2 对于每一个产生的数据产生其附近的数据，具体方法如下：给定一个总的变异量（在一定范围内随机），然后随机选择数据的一位给它一个变异，重复这个过程，直到对每位的变异的总和超过设定的总的变异量。

通过上述方法，我们产生一组包括 10 万个该数据向量的数据集，每一个数据向量的维度为 64 维。

5.2 评价标准

一个近似搜索方法的性能可以由三个方面来评价：搜索质量，搜索速度以及空间需求。理想的，一个近似搜索方法应该能够使用很少的时间以及很少的空间来达到高质量的搜索。

搜索质量可以用 recall(回收成功率)来评价，给定一个查询点 q，设 I(q)为理想的答案集合(也就是说，真实的查询点 q 的 K 个近邻)，A(q)为我们执行算法后获得的答案集合，我们可以定义 recall 为 A(q)与 I(q)的交于 I(q)之比。

在理想的情况下，recall 的值为 1，也就是说，所有的 K 个最近邻数据点都被返回了。在通常情况下，recall 的值小于 1，对于搜索质量而言，recall 的值越接近于 1，则返回的数据质量越好，反之，则返回的数据质量越差。

更多的，搜索质量还可以被定义为 error ratio,它是由我们搜索方法得到的 K 个最近邻点与查询点 q 的距离之和与真实的 K 个查询点 q 的最近邻点与查询点 q 的距离之和之比。具体定义如下：

$$error\ ratio = \frac{\sum_{i=1}^{d} Dis(q, p_i')}{\sum_{i=1}^{d} Dis(q, p_i)} \qquad \text{式(5-1)}$$

其中，p_i'是我们执行算法后获得的 K 个最近邻，而 p 则是真实的查询点 q 的 K 个最近邻。可以看到，对于 error ratio,它的值越小，它的搜索质量越好。当它的值为 1 时，我们所获得 K 个最近邻数据也就是真实的 K 个最近邻数据。

与 recall 相比，当查询点 q 附近的数据点较密集(远大于 K)且它们与查询点 q 的距离相差不大时，error ratio 更能真实的反映搜索质量。而当查询点 q 附近数据点较稀少且它们与查询点相距较远时，recall 能更直观的反映出搜索质量。

在本文实验中，我们将同时使用这两种评价标准来对搜索质量进行评价。对于时间效率，天然的我们可以用搜索时间来进行评价。我们将在实验中显示算法的运行时间以及搜索的质量结果。

5.3 实施细节

在本文实验中，我们实现 6 个不同的局部敏感哈希算法版本并使用它们对实验数据进行哈希索引表构造并进行搜索测试，6 个不同版本的算法具体罗列于下：(1) 基于汉明距离的局部敏感哈希算法(基本的)；(2) 基于汉明距离的局部敏感哈希算法(改进算法);(3) 基于 P 稳定分布的局部敏感哈希算法(基本的)；(4) 基于 P 稳定分布的局部敏感哈希算法(改进算法)；(5) 多探寻的局部敏感哈希算法(基本的);(6) 多探寻的局部敏感哈希算法(改进算法)。

本文的实验在惠普的 Elite 7100MT 机器上执行，机型配置为处理器为四核 2.93GHZ,内存为 4G，操作系统为 windows XP。

5.4 实验结果

在这一节中，我们将显示基于汉明距离的局部敏感哈希算法，基于 P 稳定分布

的局部敏感哈希算法，多探寻的局部敏感哈希算法以及上述三个算法的改进版本在我们前文所述的两个数据集上运行的结果。我们对每个数据集的 10 万个数据构建哈希表，并选择其中 1000 个数据点对其进行查询并获得其每个数据点的 K=10 个近邻，我们根据其运行结果获得其每个数据的平均查询时间，并计算查询的 error ratio 和 recall 值，这将在我们下面的图表中反映。

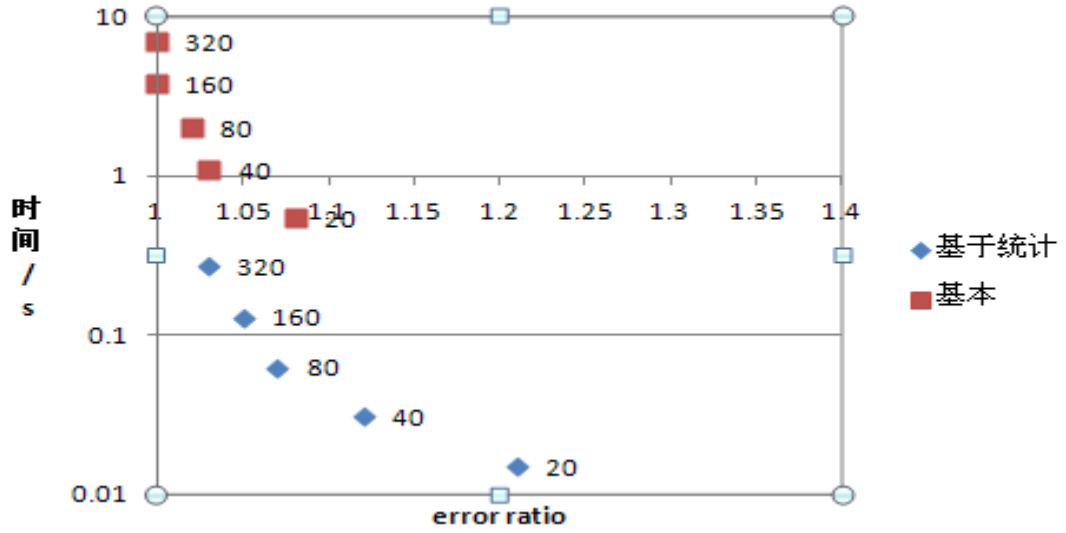

图 5.1 基于汉明距离的算法的运行结果(SIFT 数据,error ratio)

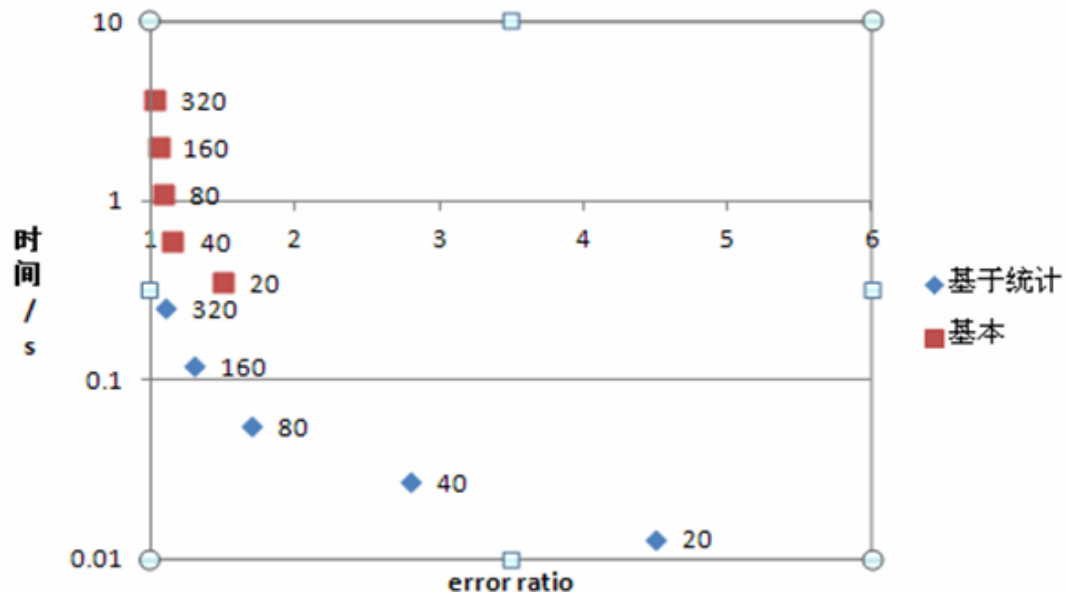

图 5.2 基于汉明距离的算法的运行结果(模拟数据,error ratio)

图 5.1，图 5.2 显示了基于汉明距离的局部敏感哈希算法在 SIFT 数据集和模拟数据集上的运行结果，我们首先根据算法构建 320 个哈希表，并依次设置几次实验搜索哈希表的数量分别为 20,40,80,160,320。我们通过增加检索的哈希表的数量来改善算法的搜索质量，显然增加哈希表的检索数量会增加的算法运行时间。观察图表可得：在检索哈希表数量的数目相同的情况下，我们的改进算法的时间远

远小于其基本的算法(在略损失搜索质量的情况下)，同样可以发现，我们的改进算法可以通过增加哈希表的检索数量来达到与其基本的算法一样的搜索质量，其运行时间仍小于其基本算法的运行时间。

注意图 5.2 的 error ratio 的值远大于图 5.1，这个是因为我们的模拟数据除了植入的近邻之外，其它数据与其近邻相比，其距离较大。在下面，我们将显示两组数据的 recall 值，我们会发现，它们的值相差不大。

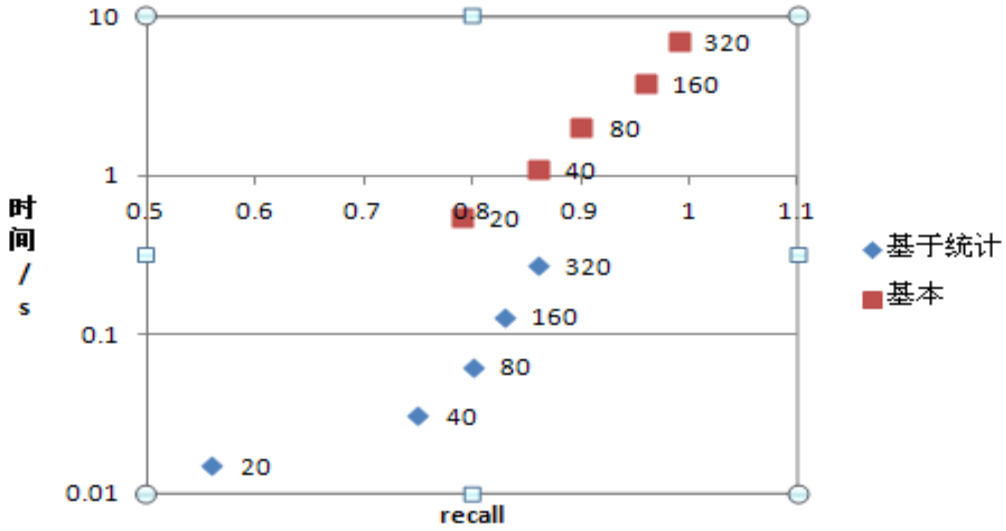

图 5.3 基于汉明距离的算法的运行结果(SIFT 数据,recall)

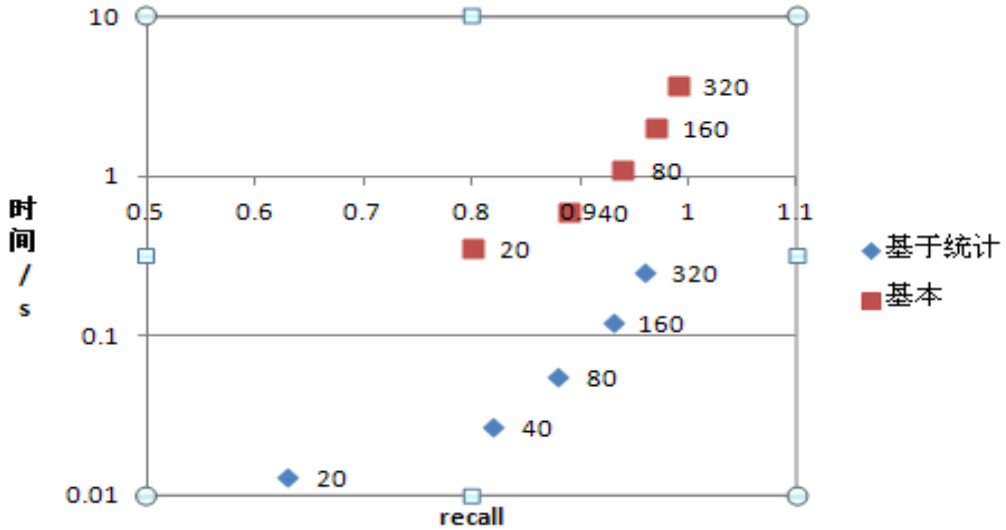

图 5.4 基于汉明距离的算法的运行结果(模拟数据,recall)

图 5.3，图 5.4 显示了基于汉明距离的局部敏感哈希算法在 SIFT 数据集和模拟集上的运行结果，它显示了在增加哈希表的检索数量的情况下，recall 与时间的关系。正如前文所述，改进后的算法速度远快与基本的算法。

下面我们将显示基于 P 稳定分布的局部敏感哈希算法运行结果，其运行结果图

示如下：

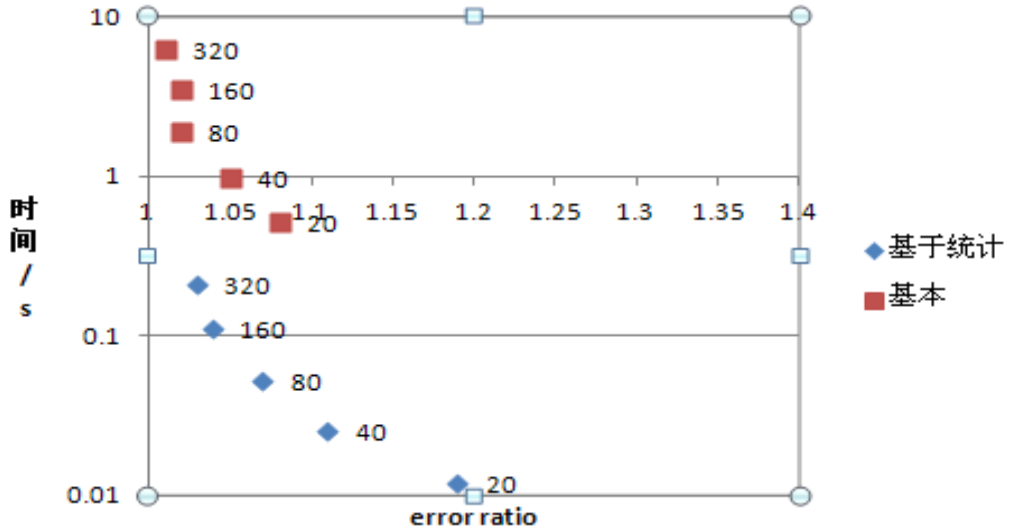

图 5.5 基于 P 稳定分布的算法的运行结果(SIFT 数据,error ratio)

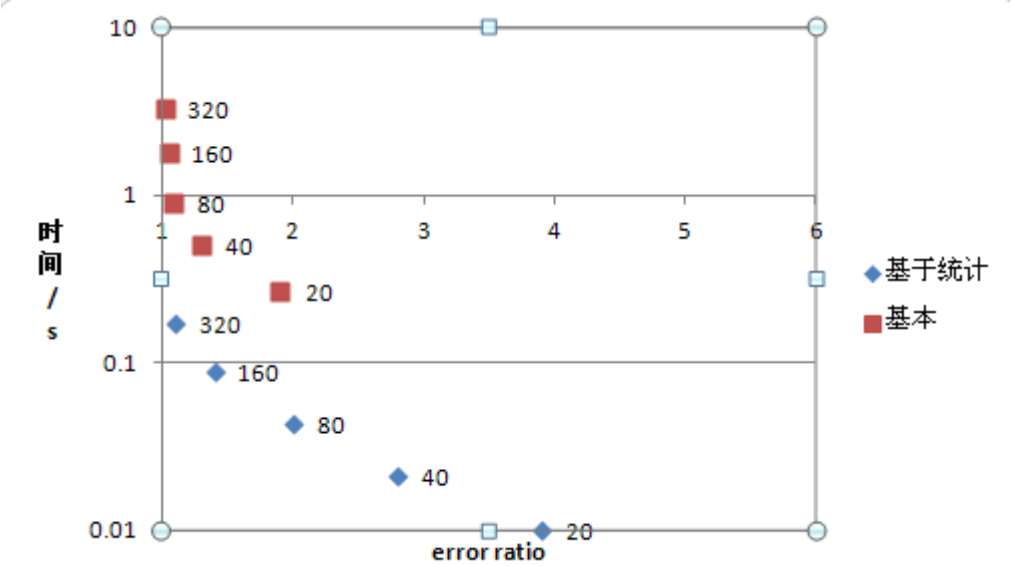

图 5.6 基于 P 稳定分布的算法的运行结果(模拟数据,error ratio)

图 5.5，图 5.6 显示了基于 P 稳定分布的局部敏感哈希算法在 SIFT 数据集和模拟数据集上的运行结果，同前文所述，我们首先根据算法构建 320 个哈希表，并依次设置几次实验搜索哈希表的数量为 20,40,80,160,320。同样观察图表可得：在检索哈希表数量的数目相同的情况下，我们的改进算法的时间远远小于其基本的算法(在略损失搜索质量的情况下)，同样可以发现，我们的改进算法可以增加哈希表的检索数量来达到与基本的算法一样的搜索质量，其运行时间仍小于其基本算法的运行时间。

图 5.7，图 5.8 显示了基于 P 稳定分布的局部敏感哈希算法在 SIFT 数据集和模拟集上的运行结果，它显示了在增加哈希表的检索数量的情况下，recall 与时间的

关系。它显示了我们改进的算法改善了算法的时间效率。

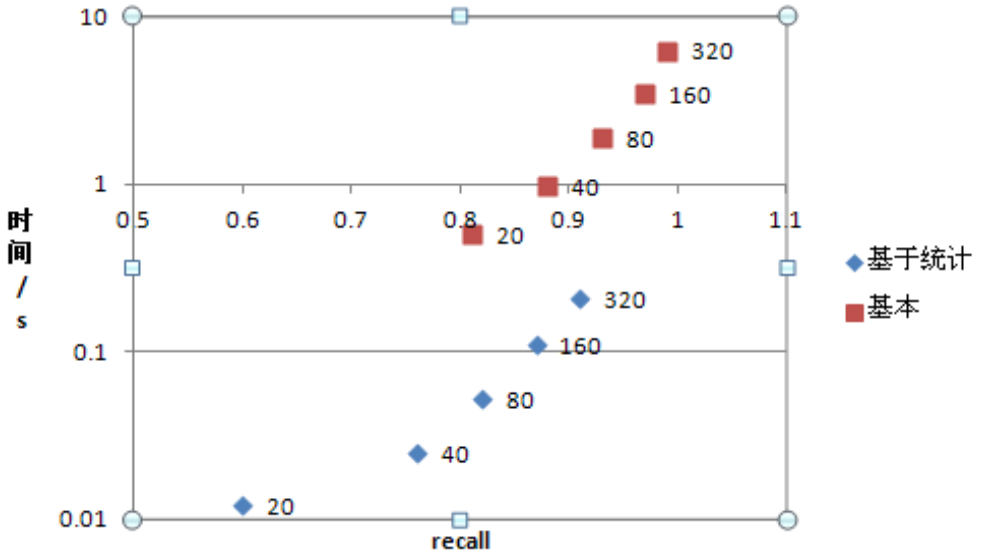

图 5.7 基于 P 稳定分布的算法的运行结果(SIFT 数据,recall)

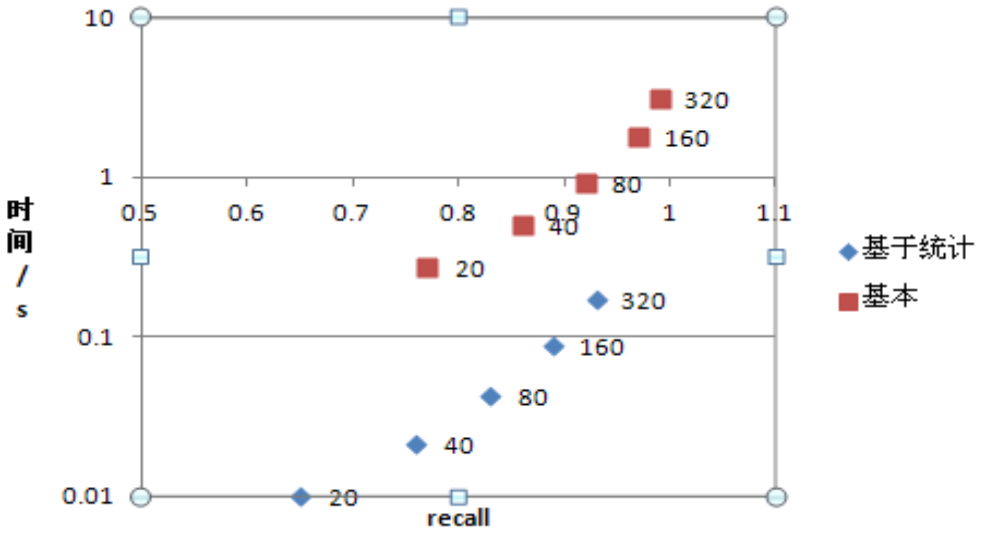

图 5.8 基于 P 稳定分布的算法的运行结果(模拟数据,recall)

下面我们显示多探寻的的局部敏感哈希算法在 SIFT 数据集和模拟数据集上的运行结果，我们首先根据算法构建 80 个哈希表，并在每个哈希表中设置多探寻的哈希桶数为 20，依次设置几次实验搜索哈希表的数量为 5,10,20,40,80。观察图 5.9，图 5.10，图 5.11，图 5.12 可得：在检索哈希表数量的数目相同的情况下，我们的改进算法的时间远远小于其基本的算法(在略损失搜索质量的情况下)，同样可以发现，我们的改进算法可以增加哈希表的检索数量来达到与基本的算法一样的搜索质量，其运行时间仍小于其基本算法的运行时间。

其中，图 5.9，图 5.10 显示了 error ratio 与时间的关系，图 5.11，图 5.12 显示

了 recall 与时间的关系，具体图表如下：

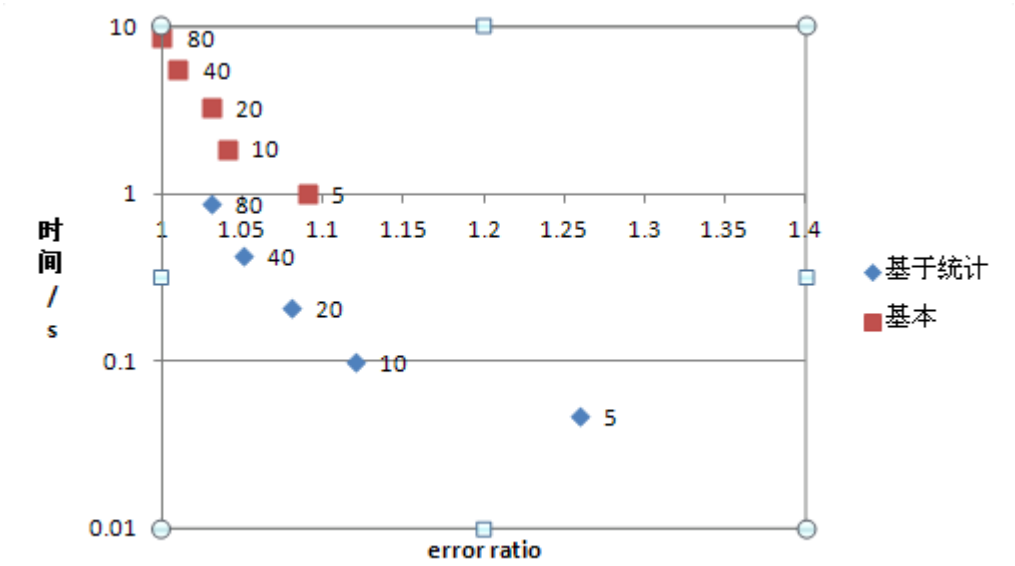

图 5.9 多探寻算法的运行结果(SIFT 数据,error ratio)

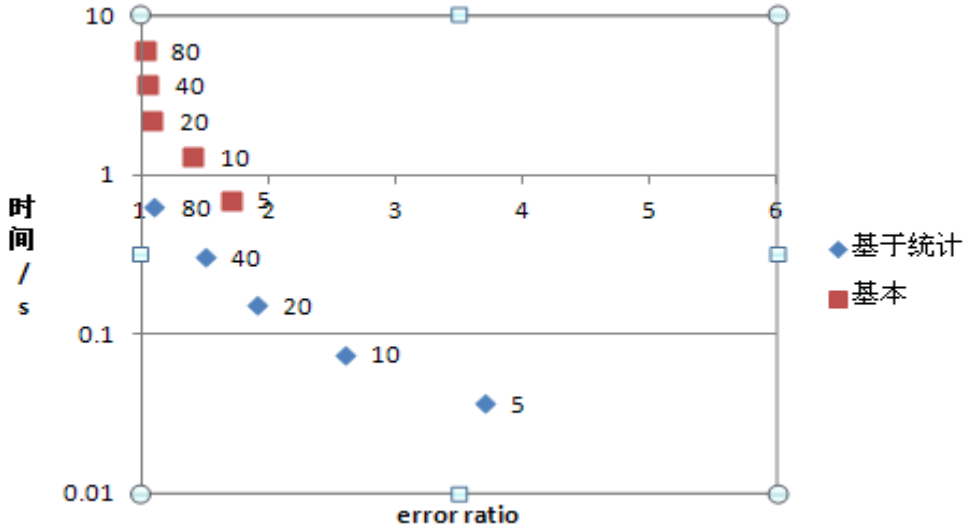

图 5.10 多探寻算法的运行结果(SIFT 数据,error ratio)

图 5.9,图 5.10 显示了多探寻的局部敏感哈希算法的两个版本在两组数据上随着哈希表的检索数量变化，其 error ratio 与时间的变化情况，可以发现，我们的基于统计的改进算法仍然有效。

图 5.11,图 5.12 显示了多探寻的局部敏感哈希算法的两个版本在两组数据上随着哈希表的检索数量变化，其 recall 与时间的变化情况，具体变化情况如下所示，同样，基于统计的改进算法有更高的时间效率。

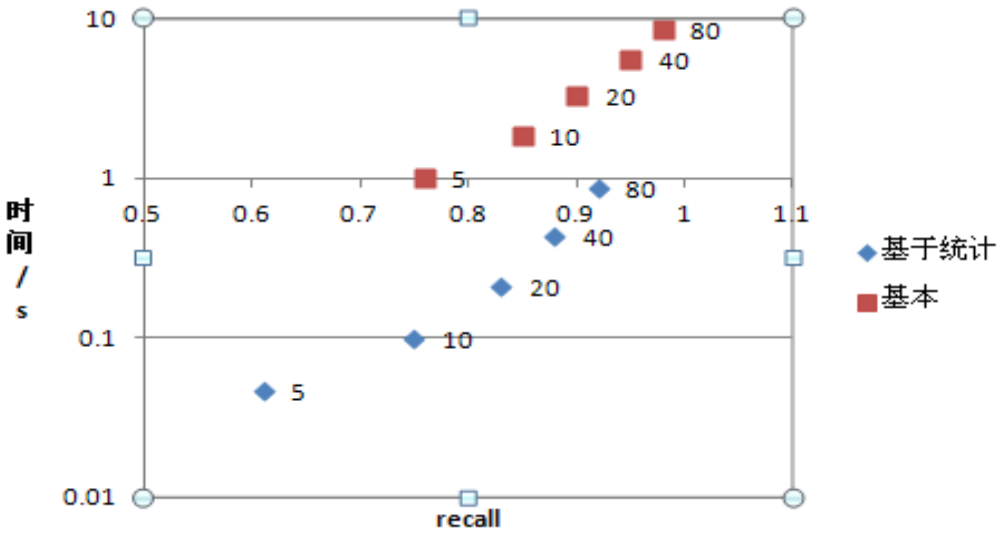

图 5.11 多探寻算法的运行结果(SIFT 数据,recall)

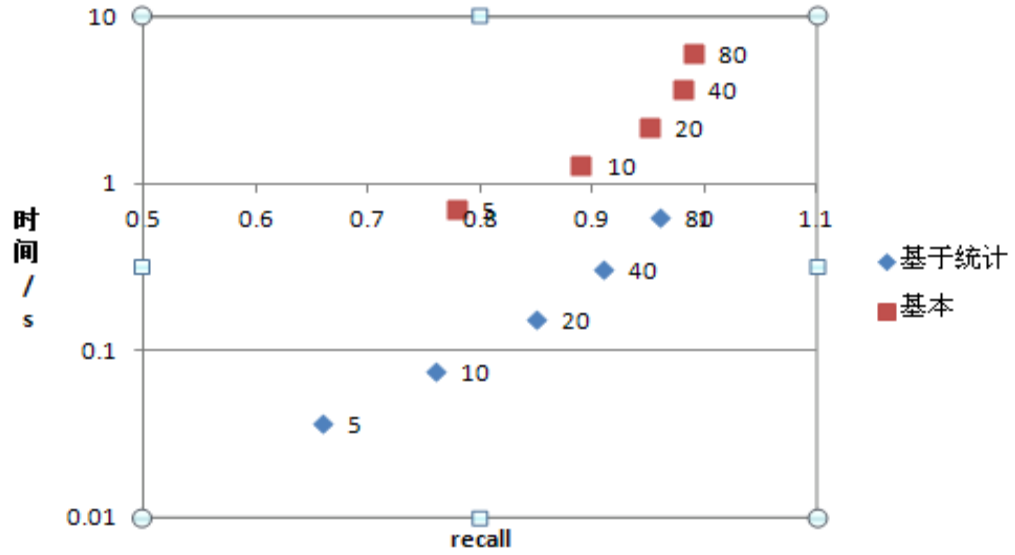

图 5.12 多探选算法的运行结果(模拟数据,recall)

5.5 本章小结

本章主要介绍了局部敏感哈希算法的实验，首先我们介绍了算法的实验数据，以及对算法的评价标准和实验细节。然后，我们显示了三个不同版本的局部敏感哈希算法及其改进后的算法在两组高维数据上(一组提取图像特征得到的高维数据，一组模拟的高维数据)的实验结果，实验结果显示，对于不同版本的局部敏感哈希算法，基于统计的改进算法都能改善基本的局部敏感哈希算法的时间效率。

第六章 总结与展望

随着计算机技术的发展，特征丰富的数据越来越多，对这些特征丰富的数据的搜索成为一个相当重要的问题。对于这些特征丰富的数据转换而成的高维数据，局部敏感哈希算法是一个相当有效的索引构建和搜索方法。自提出后，不仅引起了对算法本身广泛的研究，同时，算法也被广泛研究应用在具体问题上。本文针对几个被广泛使用的局部敏感哈希算法版本，提出了一种改进方法，改善了算法的时间效率。

本文围绕局部敏感哈希算法所做的主要工作可以总结为以下几个方面：

分析了 K 最近邻查询问题的研究现状，阐述了 K 最近邻查询问题面对大规模数据以及高维数据的挑战，总结了局部敏感哈希算法发展的历史以及现在的研究现状。

阐述了局部敏感哈希算法的基本原理，并详细介绍了局部敏感哈希算法所依赖的各种距离衡量标准，然后给出了一个根据各种距离衡量标准的局部敏感哈希函数库，其中定义了依据不同距离的局部敏感哈希函数族。

研究了几个被广泛研究和使用的局部敏感哈希算法版本，分别是基于汉明距离的局部敏感哈希算法，基于 P 稳定分布的局部敏感哈希算法和多探寻的局部敏感哈希算法。详细阐述了这几个版本的局部敏感哈希算法的原理性质以及它们的算法过程。

分析了局部敏感哈希算法的性质，因为最后在获得数据结果的过程中，依据数据点的距离并对它们进行排序将会花费大量时间来计算距离和排序，我们提出了一种对于局部敏感哈希算法的改进方法，它使用哈希表中数据点的出现次数来近似数据点与查询点的距离来获得最终的 K 个数据结果。基于前文研究的几个局部敏感哈希算法，我们分别使用该方法对它们进行了改进。

最后，我们介绍了对局部敏感哈希算法的评价标准和实验数据，并使用一组实际数据和一组模拟数据对上述三个基本的局部敏感哈希算法和它们的改进算法做了测试和比较。通过实验，我们证明了改进的算法能提高局部敏感哈希算法的时间效率。

本文的主要思想是利用哈希表中数据点的出现次数来近似数据点与查询点的距离来对从哈希表中回收的数据进行排序，从而节省了计算距离的时间以及减少了排序的时间，也因此，改善了局部敏感哈希算法的时间效率。但是，也因此，也略降低了在回收同等数量的数据情况下的搜索的质量。因此，如何在这种情况下，提高搜索质量也是一个值得关注的问题。

局部敏感哈希算法还是一个诞生时间比较短的算法，因此，还可以对这个算

法进行许多有效的研究。比如，研究发现新的局部敏感哈希函数族来更好的处理高维数据，也可以针对具体问题分析其转换而成的高维数据的特点提出针对性的局部敏感哈希函数族；同样的，我们还可以改善数据的回收过程以及处理过程，使得我们可以提高哈希表的利用率。

致谢

时光匆匆而过，转眼间两年半的研究生生涯就快结束了。这段最后的学生时光将是我生命中一段宝贵而又难忘的经历，回首我在西安电子科技大学计算机学院学习的这两年半时间，往事历历在目。在此，谨向在这个期间关心帮助过我的人士表示衷心的谢意。

首先，感谢我的硕士生导师霍红卫教授。在我的研究生阶段，霍老师给予了我细心的关怀和认真的指导，在论文的选题，资料收集和答疑的各个阶段都给了我极大的帮助。霍老师扎实的科研态度，永不止步的学习精神，谦虚随和的优秀品德都深深的感染和激励着我，让我受益匪浅。在此向霍老师表达我最诚挚的谢意。

感谢我的舍友张鹏，刘勇，蒋星，一起生活了两年半，他们给了我无私的帮助和关怀；感谢我的研究生同学李超，黄超，陈行海，张宏灏，他们对我学习和生活上的无私帮助让我收获了很多。

最后，我要特别感谢我的家人，是你们在背后默默无闻的支持和鼓励使我能够在学校专心完成学业。

参考文献

[1] Trevor Hastie and Robert Tibshirani. Discriminant adaptive nearest neighbor classification.IEEE Trans. Pattern Anal. Mach. Intell. 1996,6,18.607–616.

[2] Scott Cost and Steven Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. Mach. Learn. 1993,1,10. 57–78.

[3] Scott Deerwester, Susan T. Dumais, GeorgeW. Furnas, Thomas K. L, and Richard Harshman. Indexing by latent semantic analysis. J. Am. Soc. Information Science.1990.391–407.

[4] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. J. Intell. Inf. Syst. 1994,4.231–262.

[5] Haiquan Li, Xinbin Dai, and Xuechun Zhao. A nearest neighbor approach for automated transporter prediction and categorization from protein sequences. Bioinformatics.2008.1129–1136.

[6] Jenq-Haur Wang and Hung-Chi Chang. Exploiting sentence-level features for near-duplicate document detection. In Proceedings of the 5th Asia Information Retrieval Symposium on Information Retrieval Technology.2009,8 .205–217.

[7] Jack G. Conrad, Xi S. Guo, and Cindy P. Schriber. Online duplicate document detection:signature reliability in a dynamic retrieval environment. In Proceedings of the twelfth international conference on Information and knowledge management CIKM ’03.2003. 443–452.

[8] A. Guttman. R-trees: A dynamic index structure for spatial searching. In Proc. of ACM Conf. on Management of Data (SIGMOD). 1984. 47–57.

[9] J. L. Bentley. K-D trees for semi-dynamic point sets.In Proc. of the 6th ACM Symposium on Computational Geometry (SCG).1990. 187–197.

[10] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In Proc. of ACM Intl. Conf. on Management of Data (SIGMOD).1997. 369–380.

[11] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In Proc. of the 23rd Intl.Conf. on Machine Learning.2006. 97–104.

[12] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity search methods in high dimensional spaces. In Proc. of the 24th Intl. Conf. on Very Large Data Bases (VLDB).1998. 194–205.

[13] Indyk, P. and Motwani, R. Approximate nearest neighbor:Towards removing the curse of dimensionality. In Proceedings of the Symposium on Theory of Computing.1998.351–360

[14] Gionis, A., Indyk, P., and Motwani, R. Similarity search in high dimensions via hashing. In Proceedings of the International Conference on Very Large Databases. 1999.25–38

[15] Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. Locality sensitive hashing scheme based on p-stable distributions.In Proceedings of the ACM Symposium on Computational Geometry.2004.23–36

[16] Panigrahy, R. Entropy-based nearest neighbor algorithm in high dimensions. In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms.2006.155–163.

[17] Qin Lv,William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In VLDB Conference. 2007.950–961.

[18] Haveliwala, T., Gionis, A., and Indyk, P. Scalable techniques for clustering the web. WebDB Workshop.2000.178–188

[19] Buhler, J. and Tompa, M.. Finding motifs using random projections. In Proceedings of the Annual International Conference on Computational Molecular Biology (RECOMB1).2001.132–143.

[20] Califano, A. and Rigoutsos, I. Flash: A fast look-up algorithm for string homology. In Proceedings of the IEE Conference on Computer Vision and Pattern Recognition (CVPR).1993.57–68

[21] Indyk, P. Nearest neighbors in high-dimensional spaces. In Handbook of Discrete and Computational Geometry. CRC Press.2003.153–159

[22] Dutta, D., Guha, R., Jurs, C., and Chen, T. Scalable partitioning and exploration of chemical spaces using geometric hashing.J. Chem. Inf. Model. 2006.34–43

[23] Ravichandran, D., Pantel, P., and Hovy, E. Randomized algorithms and nlp: Using locality sensitive hash functions for high speed noun clustering. In Proceedings of the Annual Meeting of the Association of Computational Linguistics.2005.67–75

[24] 李淼，孙荣昆，韩纪庆. 基于 p-稳定分布局部敏感哈希地址的鲁棒音频检索方法. 信号处理.2012,3.368–375

[25] 肖永良，夏利民.基于局部多核支持向量机的视频镜头边界检测.信息与控制. 2001,3.381–386.

[26] 王金德，李晓燕，寿黎但.面向近重复图像匹配的 SIFT 特征裁剪算法.计算机辅助设计与图形学学报.2011,6.1043–1049.

[27] Andoni, A. and Indyk, P. Efficient algorithms for substring near neighbor problem. In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms. 2006.1203–1212.

[28] Broder, A., Glassman, S., Manasse, M., and Zweig, G. . Syntactic clustering of the web. In Proceedings of the 6th International World Wide Web Conference. 1997.391–404.

[29] Broder, A. On the resemblance and containment of documents. In Proceedings of Compression and Complexity of Sequences.1997. 21–29.

[30] Broder, A., Charikar, M., Frieze, A., and Mitzenmacher, M. Min-wise independent permutations. J. Comput. Sys. Sci.1998.112–118.

[31] Charikar, M. .Similarity estimation techniques from rounding.In Proceedings of the Symposium on Theory of Computing.2002.256–265

[32] J.R.Smith Integrated Spatial and Feature Image Systems: Retrieval ,Analysis and Compression,Ph.D.thesis.Columbia University, 1997.75–93

[33] J. M. Chambers, C. L. Mallows, and B. W. Stuck. A method for simulating stable random variables. J. Amer. Statist.Assoc.1976.340–344.

[34] Caltech Computational Vision Lab. Caltech image data set. http://www.vision.caltech.edu/Image-Datasets/Caltech101/.