东北大学秦皇岛分校

Northeastern University at Qinhuangdao

毕业论文

L-MinHash：一种蛋白质同源性搜索的高效算法

院　　别	计算机与通信工程学院
专业名称	计算机科学与技术
班级学号	**1803 - 20188117**
学生姓名	项溢馨
指导教师	王和兴

2022 年 6 月

郑 重 声 明

本人呈交的学位论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确的方式标明。本学位论文的知识产权归属于培养单位。

本人签名：　　日期：

L-MinHash：一种蛋白质同源性搜索的高效算法

摘　　要

同源蛋白质序列检索是蛋白质生物信息学的关键性基础问题。蛋白质同源性通常可以根据蛋白质序列的相似性推断。因此将同源蛋白质序列检索抽象为数学问题就是：在一系列字符串中，找出相似度高于某个阈值的全部字符串对。现阶段该问题的难点在于序列的数量规模过于庞大，采用序列间两两比较的朴素算法不可行。

本文提出了一种无需对所有序列进行两两比对而找出高相似度序列对的新算法。鉴于序列相似性通常采用编辑相似度进行度量，本文首先证明了序列间的 Jaccard 相似度是编辑相似度的上界。由于 Jaccard 相似度可以利用随机算法快速求解，因此可用于对低相似度的序列对进行快速初筛，使序列间编辑相似度的精确计算只在其更相似的子集上进行。基于此结论，本文利用最小哈希和局部敏感哈希设计了 L-MinHash 算法，实现了 Jaccard 相似度的快速估计，并基于相似度估计值对全部序列进行了聚类。考虑到高相似度序列对仅有可能在聚类集合内出现，新算法通过避开低相似度序列之间无意义比较的方式，大幅提升了同源序列检索的算法性能。

本文使用 C++语言实现了新算法并进行对照实验，完成了参数的初步调整优化，同时也复现了同源蛋白质序列检索的经典算法和最新算法。本文分别在模拟数据和真实数据上对不同算法的性能进行了测试，实验结果验证了新算法具有出色的性能优势，值得进行深入研究。

关键词：蛋白质同源性搜索，计算生物学算法，最小哈希算法，局部敏感哈希算法

L-MinHash: An Efficient Algorithm for Protein Homology Search

Abstract

Homologous protein sequence search is a key fundamental problem in protein bioinformatics. Protein homology is usually inferred from the similarity of protein sequences. Therefore, abstracting homologous protein sequence search as a mathematical problem is to find all pairs of strings with similarity higher than a certain threshold in a series of strings. The difficulty of the problem at this stage is that the number of sequences is so large in size that it is infeasible to use the plain algorithm of multiple comparison between sequences.

In this paper, we propose a new algorithm to find high similarity sequence pairs without multiple comparison of all sequences. Considering that sequence similarity is usually measured by using editorial similarity, this paper first proves that Jaccard similarity between sequences is an upper bound for editorial similarity. Since Jaccard similarity can be solved quickly with stochastic algorithms, it can be used for fast initial screening of sequence pairs with low similarity, so that the exact calculation of edit similarity between sequences is performed only on their more similar subsets. Based on this conclusion, this paper designs an L-MinHash algorithm utilizing Min Hash and Local Sensitive Hash to achieve fast estimation of Jaccard similarity and clusters all sequences based upon the similarity estimates. In consideration of the fact that high similarity sequence pairs are only possible within the clustering sets, the new algorithm substantially improves the algorithmic performance of homologous sequence search by avoiding meaningless comparisons between low similarity sequences.

For this paper, we implemented the new algorithm in C++ and conducted controlled experiments to complete the initial adjustment and optimization of parameters, and also reproduced the classical and latest algorithms for homologous protein sequence search. In addition, the performance of different algorithms is tested on simulated data and real data, and the experimental results verify that the new algorithm has excellent performance advantages and is worthy of further study.

Key words: Protein Homology Search, Algorithms in computational biology, MinHash, Locality-sensitive hashing

目　　录

1 绪　论

1.1 研究背景

2000 年 6 月 26 日，生物学和医学经历了历史性的巨变。英国首相布莱尔和美国总统克林顿宣布完成人类基因组草案。当时的报道宣称“科学家破解了人类生命的遗传密码”。人类基因组计划的进展以及多种生物基因测序工作的完成，标志着现代生物学的一个新开端。其中大多数生物学和生物医学的研究将以“基于序列”的方式进行。大量 DNA 序列和蛋白质序列已被测定，人类跨入了蛋白质时代。

人类基因组序列只是已知的许多完整基因组序列中的一种。广泛分布在生命树分支中的生物体的基因组序列，给了我们一种地球上所有生命都具有强大的统一性的感觉。人类基因组本质上是信息。计算机对于序列的确定以及生物学和医学应用都是必不可少的。计算机不仅提供了处理和存储数据的原始能力，还提供了实现结果所需的复杂数学方法。生物学和计算机科学的结合创造了生物信息学。

生命科学中，同源性 (homology) 是指由一个共同祖先演化而来的多个结构间所具有的共性特征。1843 年，欧文首次在生物学中定义了同源性，并将其定义为“在各种形式和功能下的同一组织 (organ) ”[1]。这个定义中没有提到共同祖先，因为欧文所处的时代是达尔文和孟德尔之前的时代。因此欧文对同源性的定义强调结构和位置，而不是祖先，但“共同祖先”一词正是对这种相同的结构和位置意思的精炼[2]。

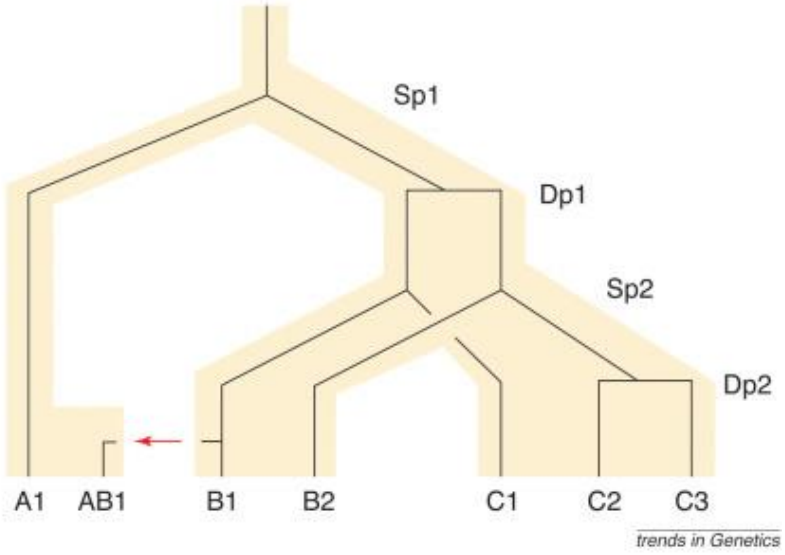

图 1.1　直系同源，旁系同源和外同源示意图[2]

如图 1.1 所示，一个基因系的理想化进化是从一个祖先群体中的一个共同祖先开始

的，最终演化为三个标记为 A、B 和 C 的群体。有两个物种形成 (speciation) 事件 Sp1 和 Sp2，都发生在以倒置 Y 表示的连接处。还有两个基因复制 (gene-duplication) 事件 Dp1 和 Dp2，用水平线段表示。两个共同祖先位于 Y 连接上方的基因是直系同源 (orthologous) 的；两个共同祖先位于水平线段连接处的基因是旁系同源 (paralogous) 的。例如，C2 和 C3 彼此旁系同源，但与 B2 是直系同源的。图中红色箭头表示 B1 基因从物种 B 转移到物种 A。因此，AB1 基因与所有其他六个基因都是外同源的。这三种同源关系都具有对称性和不可传递性。

同源蛋白质序列检索是任何基于序列的蛋白质研究的第一步，也是计算量最大的步骤。蛋白质的同源性常常通过序列的相似性 (sequence similarity) 来判定，相似性一般由检测序列和目标序列之间百分比一致性 (percent identity) 来表示。当两个序列的相似性超过偶然预期时（即超过两个随机序列的期望相似性时），则可推断其同源性。因为序列高度相似最简单合理的解释是：这两个序列不是独立出现的，而是从一个共同祖先演化而来。共同祖先解释了高度相似性，反之亦然，高度相似性则意味两个序列极有可能拥有共同祖先[3]。如图 1.2 所示，CaGre2 与其 6 种同源蛋白质具有相似的一级结构，仅在红色虚线圆圈和红色箭头处存在差异。

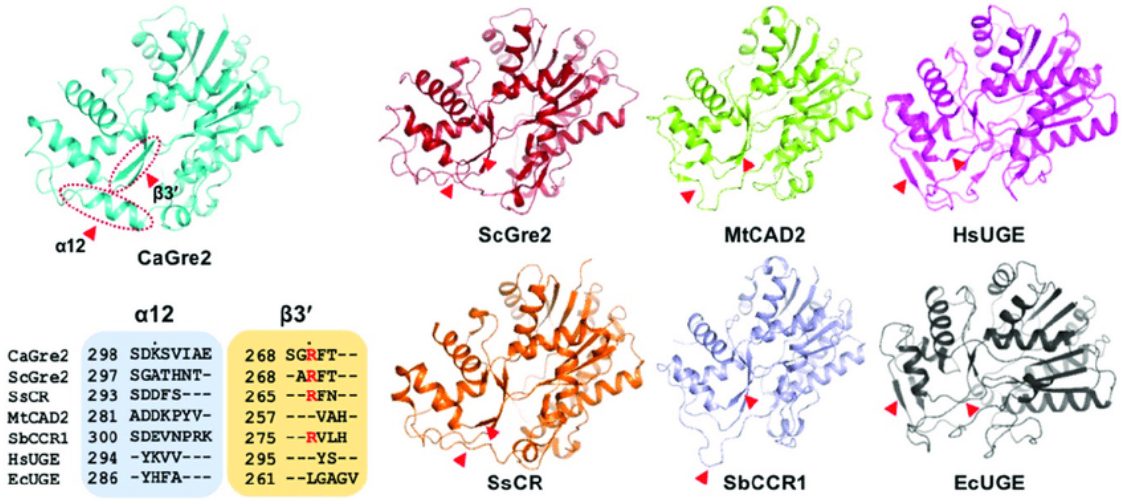

图 1.2 CaGre2与同源蛋白的结构比较[4]

然而，同源性和相似性的概念不能完全等价。因为同源序列并不总是具有显著的序列相似性，有数以千计的同源蛋白质序列相似性并不显著[5]。但当相似性搜索发现具有统计显著相似性时，却可以推断这两个序列是同源的。因此序列相似性和同源性在逻辑意义上讲应当是充分而非必要的。

1.2 研究意义与内容

蛋白质同源性搜索，是生物信息学中许多算法的核心。多数计算机程序根据蛋白质的氨基酸序列决定其三维结构，从而决定其功能特性的基本原理，计算这些蛋白质的结构。第一步通常是数据库筛选已知结构的相关蛋白质，结构预测的问题将被简化为预测序列变化对结构的影响，并且目标结构将通过同源建模的方法进行预测。如果没有发现已知结构的同源蛋白质，那么结构预测必须完全从头开始。

例如，由 DeepMind 团队研发的 AlphaFold 算法就能够根据一个蛋白质的氨基酸序列来确定其三维结构，破解了蛋白质折叠预测的世纪难题，对于准确认识蛋白质功能，甚至整个二十一世纪生命科学的发展，有着至关重要的意义。AlphaFold 算法会先在已有蛋白质序列和结构数据库里面寻找目标蛋白质的同源蛋白，作为神经网络输入[6]。AlphaFold 算法的预测精度依赖于输入的同源蛋白数量和相似性，以及同源蛋白是否已经有实验结构。由此可见，同源蛋白质序列检索是非常关键的基础问题。

同源序列检索的研究目标是在海量序列中挖掘出相似性满足一定要求的序列集合，作为下游蛋白质结构预测与功能分析任务的关键输入。同源蛋白质序列检索抽象为数学问题便是：在超大数据规模的字符串中，检索出全部相似性高于某个阈值的字符串对，并且通常情况下相似性采用 Levenshtein 距离（亦称作编辑距离）度量[7]。

生物信息学数据的一个明显特点是数据量非常大且更新速度迅速。到目前为止，核苷酸序列数据库包含16×10^9个碱基，这仅相当于五个人类基因组序列的数量。大分子结构数据库包含 16000 个条目，即蛋白质的完整三维坐标信息，而其平均长度约为 400 个残基。由于测序技术的飞速发展，被测序蛋白质的数量已经达到亿级，并且仍在迅速增加[8]。

因此现阶段该问题的研究难点在于序列数量规模过于庞大（超过 10^9 条），序列间两两比较的朴素算法的 $O(n^2)$ 时间复杂度是不可行的，更何况两个序列间计算 Levenshtein 距离还需要更多计算。因此，在合理时间内找到全部同源蛋白质序列是本文要解决的关键问题，也是难点所在。

综上，如何将序列比对算法与生物信息学中的寻找同源蛋白质序列问题的特性相结合，进一步提高序列比对算法在求解同源蛋白质序列问题时的性能，已成为了一个值得关注的研究方向。围绕这一前沿研究领域，本文尝试开发一种快速算法来实现同源蛋白质序列的快速检索。

1.3 论文组织结构

第 1 章绪论，阐明了本文的研究背景，并论述了本文做出的主要研究意义。最后对于本文的工作方向和内容进行了简明扼要的概括和整理。

第 2 章是蛋白质同源性搜索问题的综述。首先对蛋白质同源性搜索问题相关的基础术语进行了介绍，并采用形式化的语言具体定义了同源性搜索问题，用编辑相似度来替换同源性概念，便于之后的模型建模与分析。接着概括了该领域国内外主要的研究成果，介绍了领域内比较经典的解决方案及其优化方向。最后提出了蛋白质同源性搜索问题目前面临的挑战。

第 3 章介绍了蛋白质同源性搜索算法的总体设计。首先提出了算法总体的设计思路和预期目标，阐述了算法的总体架构，介绍了构成算法的模块。由于超长的序列长度导致无法采用动态规划思想求解序列对的编辑相似度；超大的序列规模导致无法用序列间两两对比的暴力算法进行序列比对，所以给出了基于 Jaccard 相似度的快速过滤算法和面向大规模集群的 L-MinHash 聚类算法，并设计了其中的函数关系。最后对经典模型 Order Min Hash 的复现流程进行了简单介绍。

第 4 章是全文的核心章节，旨在详述本文提出的蛋白质同源性搜索的高效算法 L-MinHash，即如何从计算编辑相似度的近似解和快速过滤实现序列规模的降维两方面进行优化。首先从最长公共子序列 LCS 问题入手，引出 Jaccard 相似度，证明 Jaccard 相似度是编辑相似度的上界，并利用该上界实现低同源性序列的快速过滤。接着寻找问题规模降维的可行性，提出了基于 Locality-sensitive hashing 的 L-MinHash 算法，证明 L-MinHash 是局部敏感的，可以用于数据的聚类。最后，给出了经典模型 Order Min Hash 的复现细节。

第 5 章属于实验环节，首先对数据集与将要进行的实验任务进行介绍，接着详述了实验设计中各类参数，方便其他研究者复现。再将实验结果与朴素算法的结果进行比对分析，检测算法的可信程度和实现效率。并将本文提出的 L-MinHash 算法进行多次独立实验。其结果与 Order Min Hash 模型结果进行对比，取得了较高的性能。最终对模型的各项参数进行优化，并进行多组对照实验，找到最优的参数。实验证明，当 k=4，l=2，L=300，p=19260817，pp=500 时，模型能取得比较理想的效果。

结论是正文的最后部分，将在此对全文的工作进行总结，并对未来进一步研究工作的发展提出思路和展望。

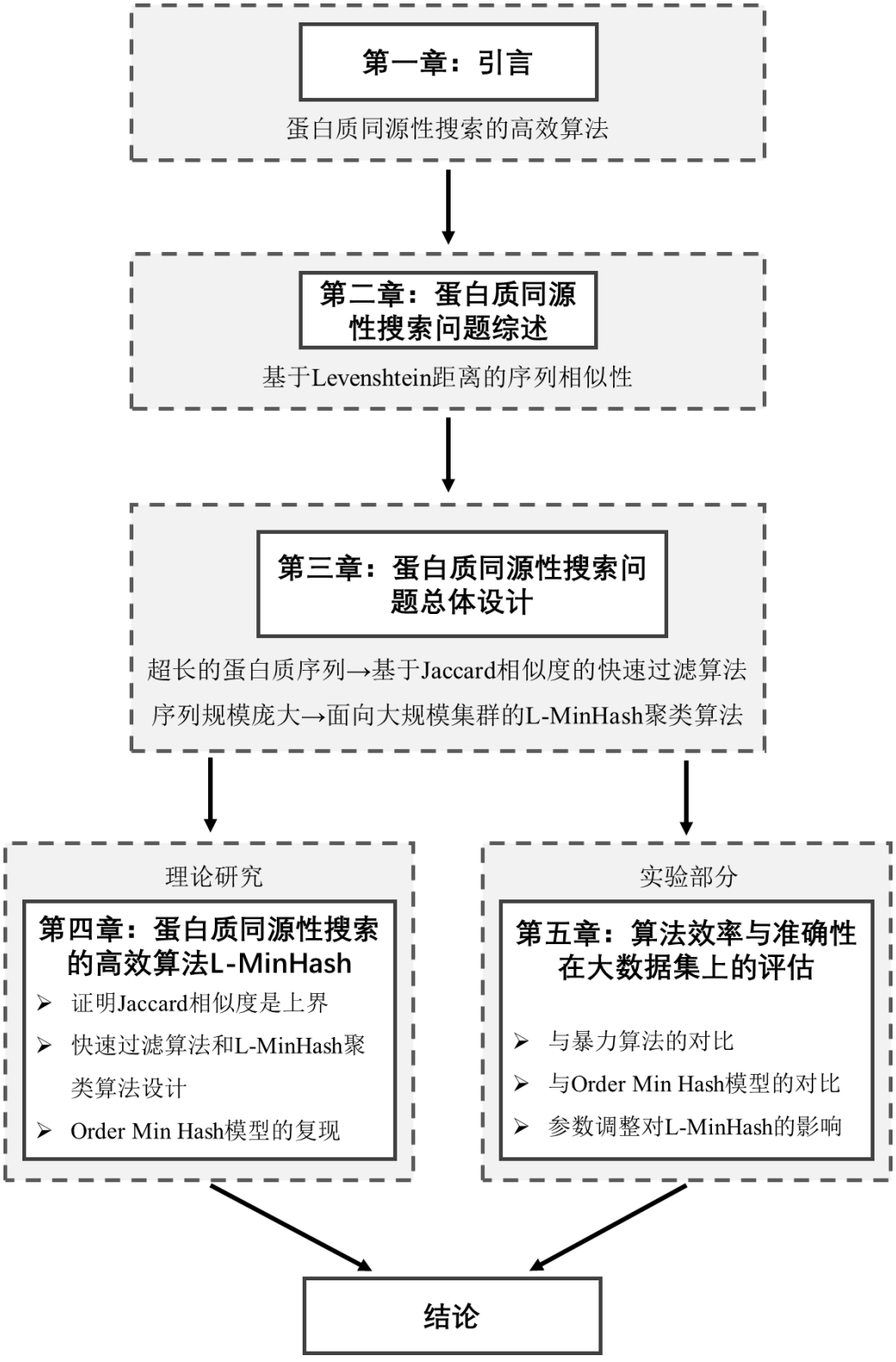

图 1.1　论文各章节的关系图

2 蛋白质同源性搜索问题综述

2.1 基本术语介绍

2.1.1 脱氧核糖核酸 DNA

DNA 是一种聚合物，由两条多核苷酸链组成。它们相互缠绕，形成一个双螺旋结构，携带遗传指令。每个核苷酸由四种含氮核碱基，脱氧核糖和磷酸基团组成。在 DNA 分子中，脱氧核糖和磷酸是稳定的，而碱基是可变的。

DNA 分子中的碱基分为嘌呤和嘧啶。嘌呤包括腺嘌呤 (adenine) 和鸟嘌呤 (guanine) ，嘧啶包括胞嘧啶 (cytosine) 和胸腺嘧啶 (thymine) ，还有一种碱基是尿嘧啶 (uracil) 存在于 mRNA 中[9]。图 2.1 列出了五种碱基的分子结构。在 DNA 分子中，腺嘌呤，鸟嘌呤，胞嘧啶和胸腺嘧啶分别用字母 A、G、C 和 T 表示。

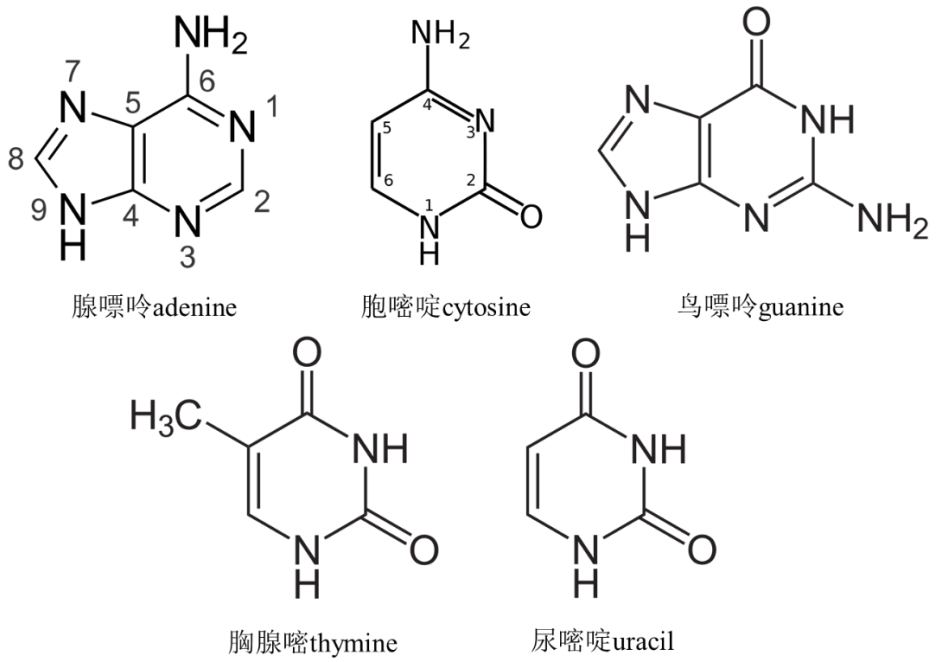

图 2.1　碱基的分子结构

2.1.2 氨基酸 Amino acids

氨基酸是组成蛋白质的基本单位，赋予蛋白质特定的分子结构形态。如图 2.2 所示，氨基酸由一个氨基、一个羧基、一个氢原子和一个 R 基连在同一个中心 C 原子上组成。

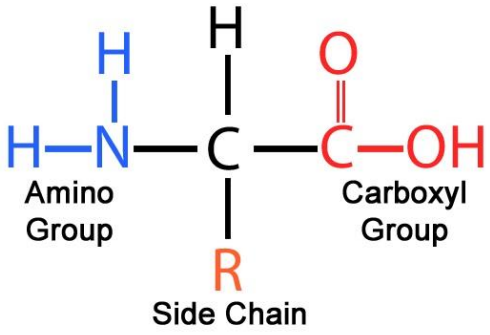

图 2.2　氨基酸的基本结构

氨基酸中的 R 基可以是不同侧链基团，对应的理化性质也不同。基因中每三个碱基序列组合表示一个氨基酸，例如 AUG 是蛋氨酸的序列，因此可能的密码子总数为 64 个。但是由于基因中存在一些冗余，一些氨基酸可以对应多个序列组合。

蛋白质氨基酸，即标准氨基酸，共 22 种。其中 20 种是常见的，见于所有生物体中。这 20 种氨基酸的结构和化学性质不同，所以能够组成功能各异的蛋白质分子。

2.1.3 蛋白质 Protein

蛋白质由氨基酸组装而成。每种蛋白质都有自己独特的氨基酸序列，由编码该蛋白质基因的碱基序列决定。蛋白质的一级结构即氨基酸序列。蛋白质是生物大分子，分子量一般在 8000 以上。按照氨基酸的平均分子量 110 计算，蛋白质所含氨基酸的数目为约为 73 个。图 2.3 展示了蛋白质是怎样由碱基通过转录组合构成的。

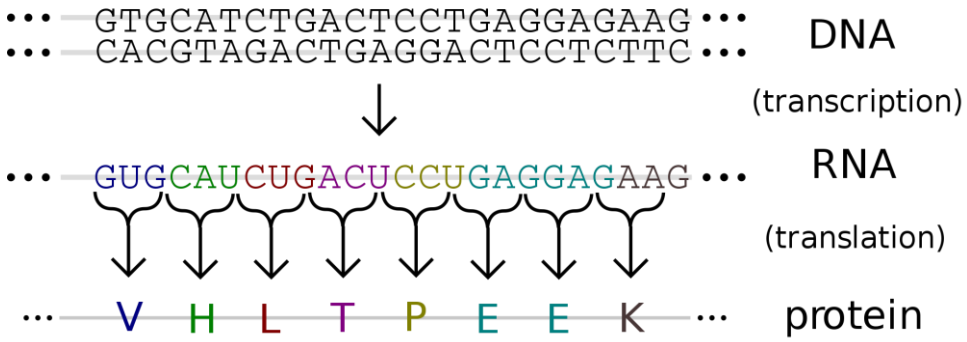

图 2.3　碱基与蛋白质序列关系

2.1.4 同源性 Homology

在生物学中，同源性是指不同类群中的一对结构或基因之间由于拥有共同的祖先而产生的相似性。蛋白质的序列同源性同样根据共同祖先来定义。由于物种形成事件（直

系同源），片段重复事件（旁系同源）和基因横向转移事件（异源同源）的作用，两个不同的 DNA 片段可从共同祖先演化而来。

蛋白质之间的同源性一般由氨基酸序列的相似性推断。因为两个蛋白质之间具有同源性，因此它们的氨基酸序列高度相似，并且具有相似的结构和功能。

2.1.5 k-mer

在生物信息学中，k-mers 是包含在生物序列中的长度为 k 的子串[10]。所以，一条生物序列就是很多个不同的 k-mer 的集合。

例如，序列为 AHFUWFU。令 k=2，那么这条序列中所有的 2-mer 组成的集合为{AH, HF, FU, UW, WE}。需要注意的是，FU 在序列中出现了两次，但是在集合中只能出现一次。这是因为集合中不能包含相同的元素，但元素的权值会进行相应的增加。

2.1.6 哈希函数 Hash function

哈希函数是可将任意大小的数据映射到固定大小值的一类函数。哈希函数及其相关的数据结构——散列表 (hash table) 用于数据存储和检索，以便每次检索在较小且几乎恒定的时间内访问数据。

Hash 需要的存储空间量仅略大于数据本身所需的总空间。当实际存储的关键字数目比全部的可能关键字总数要小时，采用散列表就成为直接数组寻址的一种有效替代。因为散列表使用一个长度与实际存储的关键字数目成反比的数组来存储。散列是一种时空复杂度高效的数据访问形式，避免了列表和结构化树的非恒定访问时间，以及直接访问的指数级存储要求。

当关键字的全域 U 比较小时，直接寻址是一种简单而有效的技术。如图 2.4 所示，假设动态集合中每个元素都取自于全域 U={0, 1, …, m-1}中的一个关键字，且集合中任意两个元素都不具有相同的关键字。我们用直接寻址表 (direct-address table) 表示动态集合，记为 T[0..M-1]。其中每个位置都称为一个槽 (slot) ，对应全域 U 中的一个关键字。槽 k 指向集合中一个关键字为 k 的元素。T[k]=NIL 代表该集合中没有关键词为 k 的元素。

直接寻址技术具有如下缺点：

（1）如果全域 U 很大，空间复杂度O(U)将不可接受，超出标准计算机内存；

（2）如果实际需要存储的关键字集合|K| ≪ |U|，分配给 U 的大部分空间实际都是

NIL 的，造成空间浪费。

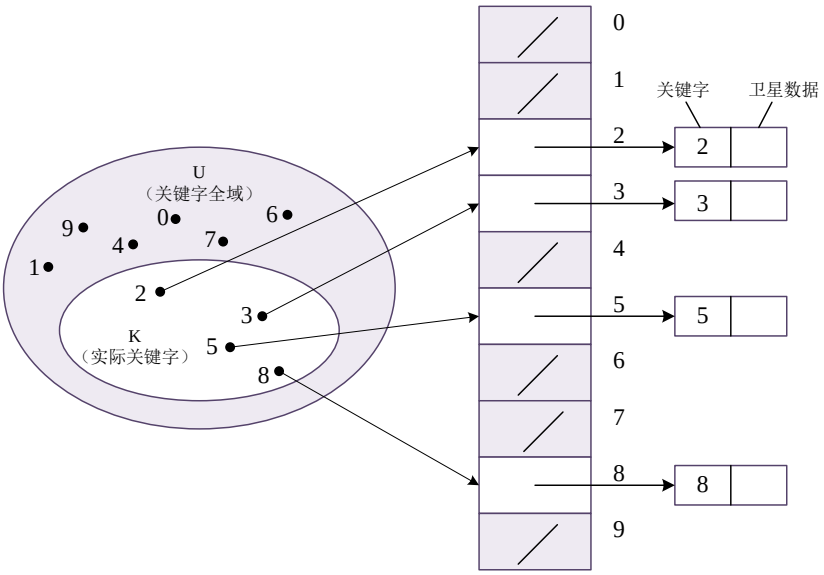

图 2.4　直接寻址表[11]

当|K| ≪ |U|时，散列表需要的存储空间远小于直接寻址表。散列表的空间复杂度可以降至O(|K|)，同时查找一个元素的效率仍能得到保持，只需要O(1)的时间。

在直接寻址方式下，具有关键字 k 的元素被存放在槽 k 中。在散列方式下，该元素存放在槽 h(k)中，即利用散列函数 h，由关键字 k 计算出槽的位置。函数 h 将关键字的全域 U 映射到散列表 T[0..m-1]的槽位上：

$$h: U \to \{0, 1, \ldots, m-1\} \quad (m \ll |U|) \tag{2.1}$$

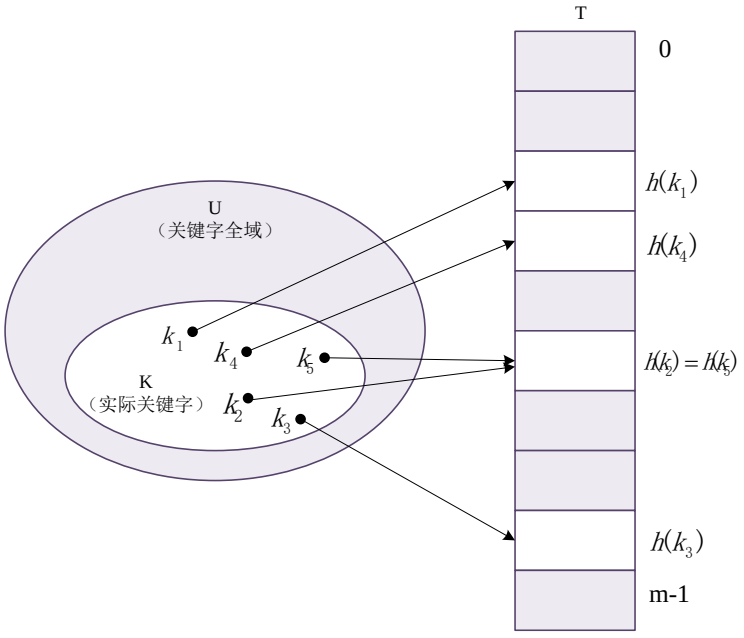

图 2.5　散列表[11]

图 2.5 描述了这种映射关系。我们可以说一个具有关键词 k 的元素被散列到槽 h(k)上，也可以说 h(k)是关键词 k 的散列值。散列函数大大降低了对存储空间的需求。

然而，散列会产生一个新的问题：两个不同的关键字有可能映射至同一个槽中。这种情况也被称为冲突 (collision) 。

理想的解决方案是找到一种可行且有效的方法规避所有冲突。但由于$|U| > m$，所以至少存在两个关键字，它们的散列值相同，被划分在同一个槽中。因此完全避免冲突是无法实现的。一方面，我们可以通过精心设计散列函数来尽量降低冲突发生概率；另一方面，当冲突不可避免发生时，我们仍需要一种解决冲突的办法。

2.1.7 全域散列 universal hashing

如果针对某个特定的散列函数去选择散列关键字，增加冲突可能性，是非常容易实现的。事实上可以将 n 个关键字全部散列在同一个槽中，平均检索时间就从常数级别$O(1)$变为了线性$O(n)$。任意一个特定的散列函数都存在如上所述的致命缺点。

唯一有效的改进方法就是采用全域散列 (universal hashing) ：随机选择散列函数，使之独立于要存储的关键字。不论被给予什么样的关键词集合，全域散列都能保证其平均性能。

设$\mathcal{H}$是一组有限散列函数，它将给定的关键词全域 U 映射到{0, 1, …, m-1}中。一个函数组如果满足：对每一对不同的关键字$k,l \in U$，$h(k) = h(l)$的散列函数$h \in \mathcal{H}$的个数至多为$|\mathcal{H}|/m$，就被称为全域的 (universal)。要而言之，如果从$\mathcal{H}$中随机选择一个散列函数，当关键字$k \neq l$时，两者发生冲突的概率小于等于 1/m，恰好就是从集合{0, 1, …, m-1}中独立地随机选择 h(k)和 h(l)时发生冲突的概率。

我们日常使用的全域散列函数类构造非常简单。首先需要选择一个大素数 p (p≥m) ，使得每一个可能的关键字 k 都落在[0, p-1]内。设Z_p表示集合{0, 1, …, p-1}，Z_p^*表示集合{1, 2, …, p-1}。

对于任何$a \in Z_p^*$和任何$b \in Z_p$，定义散列函数h_{ab}。利用一次线性变换，再进行模 p 和模 m 的归约，有

$$h_{ab}(k) = \big((ak + b) \ mod \ p\big) \ mod \ m \tag{2.2}$$

例如，当 p=17，m=6 时，$h_{3,4}(8) = 5$。

依据生日悖论 (Birthday problem) ，假设存在 n 个人，他们的生日是[1,365]中的随机整数。当$n > 365$时，一定存在两个人生日相同；当$n \leq 365$时，n 个人的生日都不相同

的概率$p_n = \frac{P_{365}^n}{365^n}$。当 n = 23 时，上述结果约为 0.46，即有超过 50% 的概率有人生日相同。而在哈希构造中，可以解释为当检验次数超过$\sqrt{p}$，就会有较大概率发生错误。因此要求 p 足够大。

考虑 p 是合数的情况。假设$N = g * n$，$M = g * m$，N 和 M 存在最大公因数 g。$N \bmod M = r$可以转换成$N = Mq + r$，即$gn = gmq + r$。其中 q 是商，r 是余数。看起来 r 的取值范围仍是{0, 1, 2, …, M-1}。但$gn = gmq + r$仍可以继续转换为$n = mq + r/g$。因为 n 和 mq 都是整数，所以 r 一定能整除 g。而 r/g 的取值范围{0, 1, 2, …, m}={0, 1, 2, …, M/g}，因此 r 的实际取值范围是{0, k, 2*k, 3*k, …, m*k}，缩小了 k 倍。因此还要求 p 是个足够大的质数。

所有这样的散列函数构成的函数族为

$$\mathcal{H}_{pm} = \{h_{ab} : a \in Z_p^*, b \in Z_p\} \tag{2.3}$$

每一个散列函数h_{ab}都将Z_p映射到Z_m。这一类散列函数具有一个良好的性质，即输出范围的大小 m 是任意的，不局限于大质数。由于对 a 来说有 p-1 种选择，对 b 来说有 p 种选择，所以$\mathcal{H}_{pm}$中包含 p(p-1)个散列函数。

以下给出散列函数族$\mathcal{H}_{pm}$是全域的证明。

考虑Z_p中的两个不同关键字 k 和 l，即$k \neq l$。对于某一个给定的散列函数h_{ab}，设

$$r = (ak + b) \bmod p \tag{2.4}$$

$$s = (al + b) \bmod p \tag{2.5}$$

因为$r - s \equiv a(k - l) \pmod p$，而根据已知$a > 0$，$k - l \neq 0$，$a(k - l) \neq 0$。p 为质数，与合数$a(k - l)$互质，因此$a(k - l) \pmod p \neq 0$，得$r \neq s$。在模 p 层次上，计算任何$h_{ab} \in \mathcal{H}_{pm}$，不同的输入 k 和 l 一定会被映射到不同的值 r 和 s，不会产生冲突。而给定 r 和 s 后，也可以解出 a 和 b 的表达式：

$$a = \left((r - s)((k - l)^{-1} \bmod p)\right) \bmod p \tag{2.6}$$

$$b = (r - ak) \bmod p \tag{2.7}$$

其中$(k - l)^{-1}$是$(k - l)$在模 p 意义下的逆元。因为除法取余运算中不具有分配律，需要把除法运算转换为乘法计算，即乘逆元取余。因此，在数对$(a, b)(a \neq 0)$与数对$(r, s)(r \neq s)$之间存在着一一对应的关系。所以，对于任意给定的输入对 k 和 l，如果从$Z_p^* \times Z_p$中均匀地随机选择(a, b)，则结果数对(r, s)在模 p 意义下，就等可能地为任何不同的数对值。

综上所述，当 r 和 s 为模 p 下随机选择的不同的值时，不同的关键字 k 和 l 发生冲突的概率等于$\Pr\{r \equiv s(\mathrm{mod}\ m)\}$。对于某个给定的 r 值，s 的可能取值空间为 p-1，其中满足$s \neq r$且$s \equiv r(\mathrm{mod}\ m)$的 s 值的数目至多为：

$$\lceil p/m \rceil - 1 \leq \left(\frac{p+m-1}{m}\right) - 1 = (p-1)/m \tag{2.8}$$

因此，s 与 r 发生冲突的概率至多为$((p-1)/m)/(p-1) = 1/m$。所以，对于任何不同的数对$k,l \in Z_p$，有

$$\Pr\{h_{ab}(k) = h_{ab}(l)\} \leq 1/m \tag{2.9}$$

得以证明$\mathcal{H}_{pm}$是全域的。

2.2 蛋白质同源性搜索问题定义

输入两条蛋白质序列：WLVAKRKOTAXHZHXKKPHLWYADZPHVSOKTXZELPPG 和 VFTRGPGFPBPXTGKPGGXPULAPGLAAGPMBL。直接对他们从头进行序列匹配，字符匹配数非常少，这两条序列似乎并没有什么相似之处。但如果将第二条序列后移 2 位，就可以发现它们的相似性增加。如果进一步在第二条序列中间部分插入一位空格，就可以发现这两条序列原来还存在更多的相似之处。

图 2.6　蛋白质序列比对

上面是两条序列相似性的一种定性表示方法。为了说明两条序列的相似程度，还需要定量计算。而可用于量化两条序列相似程度有两个方向。一是相似度，可以定义为两条序列相关的函数。其值越大，表示两条序列越相似；与相似度对应的另一个概念是两条序列之间的距离，与两条序列的相似度呈负相关关系。在大多数情况下，相似度和距离可以交互使用，并且距离越大，相似度越小，反之亦然。但一般而言，相似度使用得

较多，并且灵活多变。

相异度 (dissimilarity) 可以被定义为一个函数d: $\mathcal{U} \times \mathcal{U} \rightarrow [0,1]$，表示在全集$\mathcal{U}$内两个对象之间的距离。$d$必须符合一些基本条件，即：

（1）非负性：$d(\mathrm{x},\mathrm{y}) \geq 0$；

（2）对称性：$d(\mathrm{x},\mathrm{y}) = d(\mathrm{y},\mathrm{x})$；

（3）同一性：$d(\mathrm{x},\mathrm{x}) = 0$。当$d(\mathrm{x},\mathrm{y}) = 0$时，意味着 x=y；

（4）三角不等式：$d(\mathrm{x},\mathrm{z}) + d(\mathrm{y},\mathrm{z}) \geq d(\mathrm{x},\mathrm{y})$。

因此相异度是一种取值在[0,1]区间的归一化距离。相似度 (similarity) 定义为函数s，那么$1 - s$就是相异度。因此，相异度可以定义相似度，反之亦然。

最简单的距离就是海明距离 (Hamming distance, HD) 。海明距离仅针对两条长度相等的序列而言，其值等于序列对应位置字符不同的个数。例如，图 2.7 表示了 3 组序列海明距离的计算结果。

```
a=    EHX      SHHDB      AHBCRYEGFHF
b=    EXX      WIWDJ      ABCRYEQGFHF
-------------------------------------------------------------------
HD =   1          4                6
```

图 2.7　海明距离的计算

对于蛋白质序列而言，海明距离的计算不具有灵活性。蛋白质序列往往具有不同的长度，无法确定对齐位置，因此序列各位置上的字符难以找到真正的对应关系。虽然两条序列的其他部分相同，但位置的移动会导致海明距离变化迅速。例如，图 2.7 中最右边的序列对的海明距离为 6。仅从海明距离来看，序列对的相似度不超过 50%，因此两条序列差别很大。但如果从序列 a 中删除 H，从序列 b 中删除 Q，则两条序列都变成 ABCRYEGFHF。这说明两条序列仅仅相差两个字符。因此直接运用海明距离来衡量两条蛋白质序列的相似程度会产生较大误差，是不合理的。

为了解决上述字符插入和删除的问题，我们引入字符“编辑操作” (Edit Operation) 的概念，可以通过编辑操作将一个序列转化为一个新序列。需要用一个新的字符“-”代表空位 (Space) ，并定义以下字符的编辑操作：

Match (a, a) — 字符匹配；

Delete (a, -) — 从第一条序列删除一个字符，或在第二条序列相应的位置插入一个空位字符；

Replace (a, b) — 以第二条序列中的字符 b 替换第一条序列中的字符 a；

Insert (-, a) — 在第一条序列插入空位字符，或删除第二条序列中的对应字符 b。

很显然，在比较两条序列 s 和 t 时，在 s 中的一个删除操作等价于在 t 中对应位置上的一个插入操作，反之亦然。需要注意的是，两个空位字符不能匹配，因为这样的操作没有意义。

给定两个字符序列 a 和 b，分别用$|a|$和$|b|$表示序列的长度。编辑距离 (Levenshtein distance) 定义为

$$\mathrm{lev(a,b)}=\begin{cases}|a| & if\ \ |b|=0,\\ |b| & if\ \ |a|=0,\\ \mathrm{lev}\big(\mathrm{tail(a)},\mathrm{tail(b)}\big) & if\ \ a[0]=b[0],\\ 1+min\begin{cases}\mathrm{lev(tail(a),b)}\\ \mathrm{lev(a,tail(b))}\\ \mathrm{lev(tail(a),tail(b))}\end{cases} & otherwise\end{cases} \tag{2.10}$$

tail(x)代表字符序列 x 除了第一个字符外的其余字符序列。最小值函数中的三种情况分别对应从 a 到 b 匹配的删除、插入和替换。

编辑相异度 (edit dissimilarity/ normalized edit distance) $E_d(a,b)$代表将 a 转换成 b 所需的编辑距离除以 a、b 中较长序列的长度，见公式 2.11。由此可以得到本文讨论的蛋白质同源性搜索问题的关键，编辑相似度$E_s(a,b)$。

$$E_d(a,b)=E_d(b,a)=\frac{lev(a,b)}{\max(|a|,|b|)} \tag{2.11}$$

$$E_s(a,b)=1-E_d(a,b) \tag{2.12}$$

蛋白质同源性搜索问题就可以被定义为在超大数据规模的字符串中，检索出全部编辑相似性高于某个阈值的字符串对。

2.3 国内外研究现状及挑战

序列比对是计算生物学中的关键基础问题。通过序列比对可以阐明序列间的相似程度，从而推断目标之间的亲缘关系和结构关系[12]。序列比对问题可以分为双序列比对和多序列比对。双序列比对问题目前已有近乎完美的解决方案，最基础的序列比对算法就是基于动态规划思想的 Smith-Waterman 算法[13]和 Needleman-Wunsch 算法[14]。在此基础上利用启发式策略加速诞生的 FASTA[15][16]和 BLAST[17]软件目前广泛应用在生物医学的前沿领域，比如目前世界大流行的新型冠状病毒变异株的确定就是通过 BLAST 完成的。多序列比对是双序列比对的扩展，但是多序列比对问题就不再有多项式复杂度的

算法，并且已经被证明为 NP 完全问题[18]，因此解决方案都是近似算法或启发式算法，其中较流行的软件是 CLUSTALW[19]。

Needleman-Wunsch 算法是最先将动态规划应用在生物序列比对问题上的，也称作优化匹配算法或整体序列比较法。Needleman-Wunsch 算法能找到两个序列的全局最优比对，但无法用于寻找高度相似的局部区域。该算法将两个序列之间每个位置的比对情况分为匹配 (match) ，失配 (mismatch) 和增删 (indel, i.e. Insertion or Deletion) 三类，并赋予相应分值。动态规划的思路为：令 F_{ij} 表示第一个序列的前 i 个字符与第二个序列的前 j 个字符已经完成比对时的最大分值，于是动态规划的转移方程为

$$F_{ij} = max\{F_{i-1,j-1} + S(A_i, B_j),\ F_{i,j-1} + d,\ F_{i-1,j} + d\ \} \tag{2.13}$$

其中 A_i 表示第一个序列的第 i 个字符， B_j 表示第二个序列的第 j 个字符，$S(A_i, B_j)$ 表示 A_i 和 B_j 匹配或失配情况下的分值， d 表示增删（即在 A_i 或 B_j 相应位置插空）的分值。显然，对于两个长度分别为 m 和 n 的序列，Needleman-Wunsch 算法的时间复杂度为 $O(mn)$ 。

后续有很多工作尝试对 Needleman-Wunsch 算法进行优化，比如运用经典的“四个俄罗斯人”（Method of Four Russians）算法可将复杂度降至$O(mn/log_n)$[20][21]。Rashed 等人则提出了基于机器学习的算法优化策略，其核心思想是将包括多层感知机、支持向量机、决策树、随机梯度下降法和 XGBoost 分类器在内的 15 种机器学习方法对输入数据进行预处理，利用数据分布的特征和并行技术对算法进行了加速[22]。

Smith-Waterman 算法是 Needleman-Wunsch 算法的变体，其目的不再是寻找全局最优的序列比对，而是寻找两个序列中具有高相似度的局部片段。而为了实现局部序列比对这一目标，Smith-Waterman 算法只需在 Needleman-Wunsch 算法的基础上做简单修改，只需保证状态的价值函数不再出现负值即可[13]。具体来说，就是在状态转移方程中，对可能出现负值的转移情况，要将目标状态的价值重置为 0，即定义 $H_{i,j}$ 表示第一个序列的前 i 个字符与第二个序列的前 j 个字符的最大局部比对得分，初始状态和状态转移方程如下：

$$H_{i,0} = H_{0,j} = 0, \quad (0 \le i \le n, 0 \le j \le m) \tag{2.14}$$

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + S(A_i, B_j) \\ H_{i,j-1} + d \\ H_{i-1,j} + d \\ 0 \end{cases} \tag{2.15}$$

最终，最优局部比对的分值应为 $\max_{0\le i\le n,0\le j\le m} H_{i,j}$ ，具体的比对结果可以通过回溯得到。

就算法的时间复杂度而言，Smith-Waterman 算法与 Needleman-Wunsch 算法一样，也是 $O(mn)$ 。FASTA 工具就是以 Smith-Waterman 算法为核心研发的，其对于基础算法进行了并行化，使比对速度提升了 10-20 倍[23]。不仅如此，FASTA 工具通过搜索和匹配 *k*-mers（长度为 k 的子序列，对蛋白质序列而言，一般取 $k=2$）来实现近似最优比对。于是，虽然 FASTA 工具核心算法的最坏时间复杂度仍为 $O(mn)$，但平均时间复杂度可以减小到 $O(mn/20^k)$ [24]。FASTA 工具的算法复杂性取决于 *k*-mers 的长度：*k*-mers 越长，算法速度越快。但是使用 *k*-mers 加速后的算法无法保证找到两个蛋白质序列之间的最优比对，因为可能会错过较小的相似区域导致比对错位。很明显，FASTA 工具的算法需要在速度和精度之间权衡。

此外，通过 FPGA[25]、GPU[26]、SIMD[27]和 Cell Broadband Engine（Cell 宽带引擎）[28]等硬件或指令级的加速，都可以很好地提升 Smith-Waterman 算法在实际运行时的性能，即使最坏情况下仍是 $O(mn)$ 级别的时间复杂度。

尽管有许多算法和工程上的优化，但两个序列之间的比对计算仍是两个输入序列长度的乘积。在实际应用中，如果需要对大规模序列对进行两两比较，这种级别的时间复杂度依然很高，甚至不可接受。鉴于此，最近部分新方法尝试使用哈希算法如 MinHash 和 Locality-sensitive hashing (LSH)，来避开直接的序列间比较，以便更快速地检测出具有高对齐概率的序列。典型的有 Mash[29]，Mashmap[30]和 MHap[31]等序列比对工具。

从国内外研究现状的分析中，我们不难发现蛋白质同源性搜索问题主要存在以下两个方面的挑战：

（1）超长的蛋白质序列

蛋白质是一种相对分子质量很大，变化范围也很大的生物分子，分子量范围可从几千到百万以上。因此蛋白质序列长度也为几十到上万。

传统计算 Levenshtein 距离的算法时间复杂度为$O(mn)$，m 和 n 分别表示进行比对的两条序列的长度。考虑其使用动态规划思想，无法简单通过改造算法结构来实现时间复杂度的降维。虽然可以通过$O(\max(m,n))$的时间查询对应位置相同的字符并进行删除的预处理，缩小$O(mn)$中的 m 和 n，使时间复杂度降为$O(m'n')$ $(m'\le m, n'\le n)$ 。但由于蛋白质序列的特殊性，对应位置字符相同的概率非常小，使得这种优化发挥空间的余地很小。

从 Needleman-Wunsch 算法到 Smith-Waterman 算法的序列比对优化历程，也不过仅仅是针对字符串匹配的细节做出了一系列调整，并没有从本质上解决两个序列之间的比对计算仍是两个输入序列长度的乘积的问题。

因此以往在计算相似度中比较流行的基于比对的计算方法，在处理如此长度的数据时其性能已经远远达不到需求。如何解决蛋白质同源性搜索问题中超长序列之间的相似度计算问题是本文中重点研究内容之一，基于 Jaccard 相似度的解决思路将在 3.2.2 中提及。

（2）蛋白质序列规模庞大

另外在蛋白质同源性搜索问题中，面临的另一个问题就是序列条数比较多。

若采用朴素算法进行两两序列比对，时间复杂度 $O(l^2)$ ，l 表示字符序列的数量。因此总的序列比对时间复杂度近似为 $O(l^2mn)$ ，是绝对无法接受的。所以我们需要找寻一种数据聚类方式，比如本文在 3.2.3 中介绍的 L-MinHash 聚类方法，对同源概率较高的序列对进行快速分类处理，使 l 的规模迅速缩小，从而令序列比对的时间复杂度降为 $O(l'^2)$ $(l' \ll l)$ 。并且最后能通过并行处理的方式，使得程序在多组测试数据上同时运行。

2.4 本章小结

本章首先对蛋白质同源性搜索问题相关的基础术语进行了介绍，并采用形式化的语言具体定义了同源性搜索问题，用编辑相似度来替换同源性概念，便于之后的模型建模与分析。接着概括了该领域国内外主要的研究成果，介绍了领域内比较经典的解决方案及其优化方向。最后提出了蛋白质同源性搜索问题目前面临的挑战。

3 蛋白质同源性搜索问题总体设计

3.1 设计思路与总体目标

3.1.1 算法设计思路

本文提出了一种无需对所有序列进行两两比对而找出高相似度序列对的新算法。鉴于序列相似性通常采用编辑相似度进行度量，本文首先证明了序列间的 Jaccard 相似度是编辑相似度的上界。由于 Jaccard 相似度可以利用随机算法快速求解，因此可用于对低相似度的序列对进行快速初筛，使序列间编辑相似度的精确计算只在其更相似的子集上进行。

基于此结论，本文利用最小哈希和局部敏感哈希设计了 L-MinHash 算法，实现了 Jaccard 相似度的快速估计，并基于相似度估计值对全部序列进行了聚类。考虑到高相似度序列对仅有可能在聚类集合内出现，新算法通过避开低相似度序列之间无意义比较的方式，大幅提升了同源序列检索的算法性能。

3.1.2 算法预期目标

本文实现的是蛋白质同源性搜索的高效算法，预期目标主要有以下几点：

（1）支持读取未定大小的蛋白质序列文件，实现高效的 I/O 框架。

（2）能将具有高相似度的序列进行聚类。

（3）能对聚类后的序列进行相似度计算。

（4）支持调整参数来控制算法的精度。

3.2 算法总体架构

本文主要针对蛋白质同源性搜索这一核心问题，系统地从算法设计、数据处理优化、I/O 等多个角度，设计并实现基于 MinHash 和 LSH 的 L-MinHash 算法。如图 3.1 所示，算法主要由基于 Jaccard 相似度的快速过滤方法和面向大规模集群的 L-MinHash 聚类算法构成。

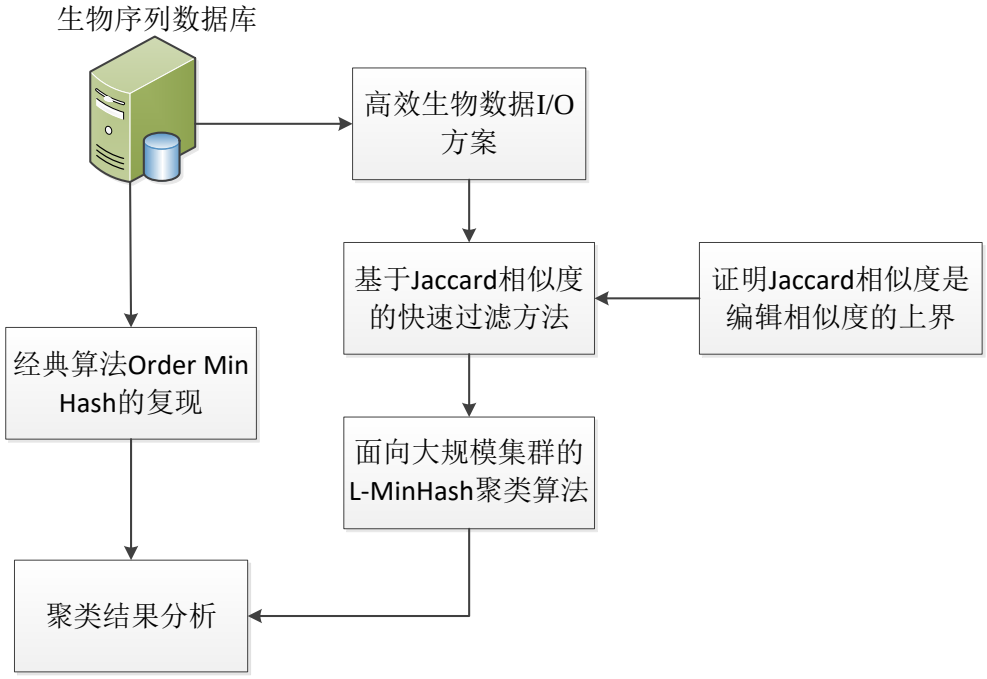

图 3.1　技术路线图

3.3 面向大规模生物序列数据的高性能 I/O 框架

为了解决大规模蛋白质同源性搜索问题，本文首先需要解决的是如何读取大规模生物序列数据。这是后续序列数据处理的前提条件，也是需要本文首先解决的性能热点。

为此本文首先提出了用 string_view 取代 string。因为我们只需“观测”字符串，而不需要对字符串进行修改，这恰恰是 string_view 相较于 string 的区别。如表 3.1 所示，采用 string_view 可以不产生额外的内存分配，加快读取速度，大大提高了整体性能。

而后使用 ifstream 流读入 FASTA 文件，并开启 GCC 优化，取得了比较不错的读取效果。具体处理代码可见 5.1。而因为同源蛋白质的稀缺性，所以输出规模并不是很大，无需进行额外优化。

表 3.1　std::string与std::string_view性能对比

	std::string	std::string_view
Heap Allocation	Yes.	No.
Ownership Semantics	Owns its contents.	Non-owning (pointer + length) .
Copying	Expensive.	Cheap.
Passing style	By reference.	By value.
Element Mutability	Allowed.	Not allowed.

3.4 基于 Jaccard 相似度的快速过滤算法

因为无法找到精确计算编辑相似度的高效算法，我们只能退而求其次寻找一种编辑相似度的估计算法。对编辑相似度进行估计，要求我们找到编辑相似度的上界，从而通过上界的计算过滤掉编辑相似度较低的序列对，实现问题规模的降维。

本文引入最长公共子序列长度作为中间变量，证明了 Jaccard 相似度可以作为编辑相似度的上界，用来实现低编辑相似度序列的快速过滤。具体证明步骤可见 4.1.3。基于 k-mer 的 Jaccard 相似度计算仍需要 $O(mn)$ 的时间。但是 Jaccard 相似度可以采用 MinHash 算法来近似计算，从而实现用整体时间复杂度为 $O(l^2(m+n))$ 的比对算法来代替 $O(l^2mn)$ 的朴素算法。

然而，快速过滤算法在数据处理阶段的 $O(l^2(m+n))$ 的时间复杂度仍是不可接受的。 $O(l^2)$ 来源于序列的两两比对， $O(m+n)$ 来源于使用 MinHash 来计算集合之间的交集和并集。因此，找到一种聚类算法来对 l 进行降维，是提升快速过滤算法速度的关键。

3.5 面向大规模集群的 L-MinHash 聚类算法

基于 Order Min Hash 模型的启发，本文设计了一种名为 L-MinHash 的聚类算法，大大缩减 l 的值域，最后将快速过滤算法中数据处理的时间复杂度降为 $O\big(l'^2(m+n)\big)(l' \ll l)$ 。

L-MinHash 是基于 LSH 方法的。如图 3.2 所示，LSH 方法具有能使 2 个相似度很高的数据以较高的概率映射成同一个哈希值，而 2 个相似度很低的数据以极低的概率映射成同一个哈希值的特性。因此 LSH 方法能高效处理海量高维数据的聚类和最近邻搜索问题。

本文提出的 L-MinHash 聚类算法的优越性主要体现在一下两个方面：相较于基于 MinHash 的 LSH 而言，L-MinHash 对于每个哈希函数都选择了 L 个最小哈希值，降低了哈希函数的碰撞概率，使得序列的聚类更加分散，只有真正具有高相似度的序列才会被聚在一起；而相较于 Order Min Hash 模型，L-MinHash 不考虑 k-mer 相对顺序的影响，进一步优化算法效率，并且增加了哈希函数的碰撞概率，使得序列的聚类更加集中，使更多有较大概率相似的序列对进入待检验队列。

因此，L-MinHash 聚类算法其实是一种 MinHash 与 Order Min Hash 的中间方案，试

图在两者之间寻找算法效率与准确性的平衡。所以，L-MinHash 聚类算法的性能非常依赖于参数设置。本文还对不同参数设置下算法的效率与准确性进行了测试，具体可见5.3.2。

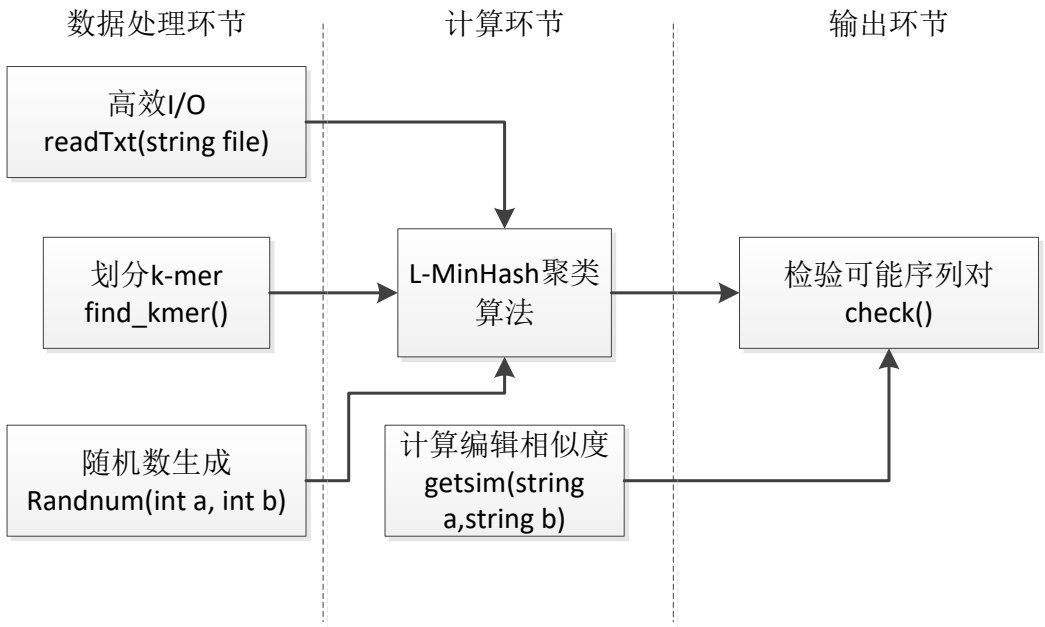

图 3.2　L-MinHash聚类函数关系图

如图 3.2 所示，L-MinHash 聚类算法可以分为三个环节。第一个环节进行数据的预处理工作，包括三块：读入函数 readTxt(string file)，划分 k-mer 的函数 find_kmer()以及随机数生成函数 Randnum(int a, int b)。其中 readTxt(string file)函数就是 3.3 提到的高性能 I/O 框架。Randnum(int a, int b)函数用来生成一个值域在[a, b]上的随机数，作为 MinHash 的参数。

第二个环节是计算环节，这是整个算法的核心。由 L-MinHash 聚类算法和计算编辑相似度的 getsim(string a, string b)函数组成。其中 L-MinHash 聚类算法是通过 Jaccard 相似度的近似估计来实现数据过滤，而 getsim 函数则是具体计算过滤后的序列之间的编辑相似度。

第三个环节是输出环节，仅包含检验可能的序列对的 check()函数。check()函数通过调用 getsim 函数对过滤后的序列对之间进行相似度检验，符合相似度阈值的序列对将被输出。

3.6 经典算法 Order Min Hash 的复现

此外，本文还使用 C++语言对 Order Min Hash 模型进行了复现。

Marçais 等人提出了 Order Min Hash 模型，是一种基于 k-mers 的有序 MinHash[32]。他们的结果虽然包括证明哈希对位置敏感的边界；然而，Order Min Hash 不会对近点 $(||p-q|| \leq R)$ 碰撞概率和远点 $(||p-q|| \geq cR)$ 碰撞概率之间的差距做出任何最坏情况的保证，因此无法获得相似性搜索边界。

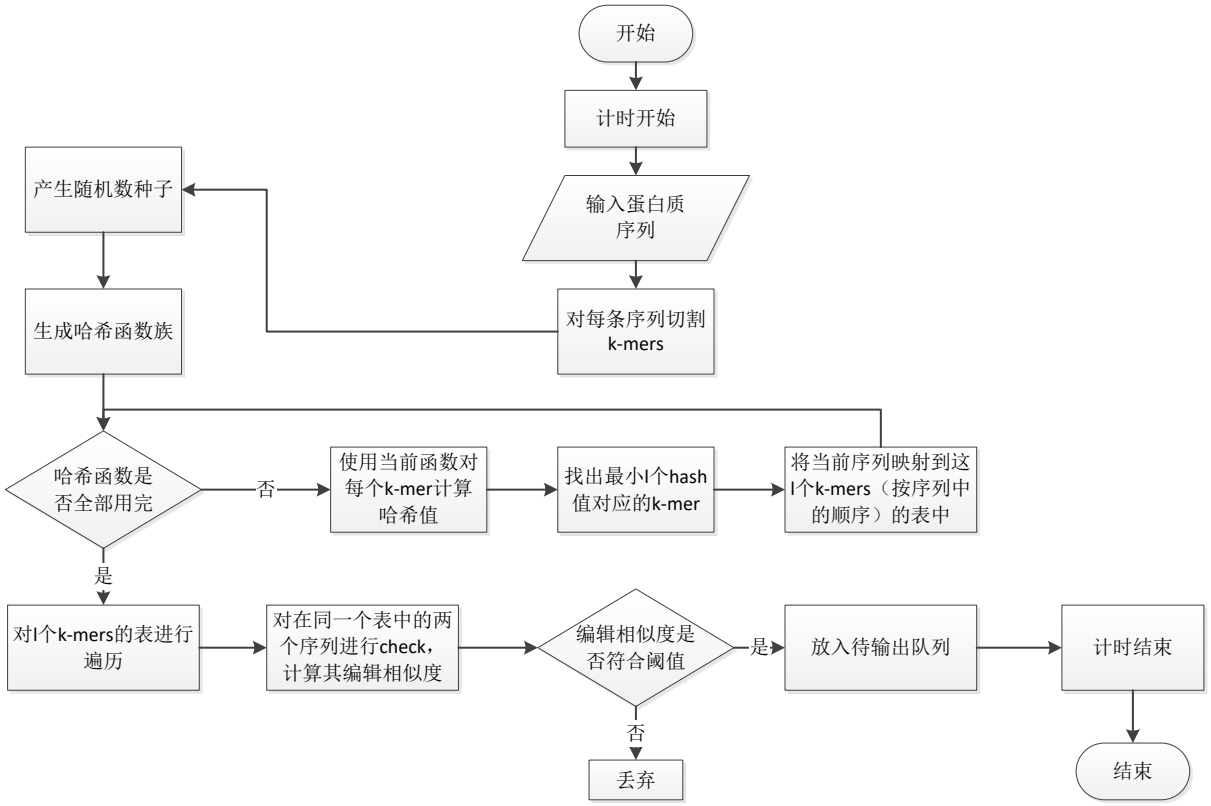

图 3.3　Order Min Hash算法流程图

Order Min Hash 模型复现的难点在于如何科学设计 MinHash 函数，使得相似的 k-mer 有较大概率被散列到一起。并且对于模型中存在的多种映射关系，该如何进行处理。本文的算法复现首先对程序中关键的 k-mer 进行划分，之后利用标准模板库 STL 中的 unordered_map 和 map 作为 LSH Table 的实现方式。最后利用 set< pair < int, int > >的方式存储待检验序列对。具体算法设计和代码见 4.4。

本文在 5.3.1 中还实现了将 L-MinHash 算法与 Order Min Hash 模型的比较，验证了 L-MinHash 算法的优越性。

3.7 本章小节

本章介绍了蛋白质同源性搜索算法的总体设计。首先提出了算法总体的设计思路和预期目标，阐述了算法的总体架构，介绍了构成算法的模块。由于超长的序列长度导致

无法采用动态规划思想求解序列对的编辑相似度；超大的序列规模导致无法用序列间两两对比的暴力算法进行序列比对，所以给出了基于 Jaccard 相似度的快速过滤算法和面向大规模集群的 L-MinHash 聚类算法，并设计了其中的函数关系。最后对经典模型 Order Min Hash 的复现流程进行了简单介绍。

4 蛋白质同源性搜索的高效算法 L-MinHash

4.1 编辑相似度上界E_s的确定

因为无法找到精确计算编辑相似度的高效算法，我们只能退而求其次寻找一种编辑相似度的估计算法。对编辑相似度进行估计，要求我们找到编辑相似度的上界，从而通过上界的计算过滤掉编辑相似度较低的序列对，实现问题规模的降维。

我们首先引入最长公共子序列问题。

4.1.1 最长公共子序列问题

给定一个序列X = ⟨x_1, x_2, … , x_m⟩，另一个序列Z = ⟨z_1, z_2, … , z_k⟩满足如下条件时称为 X 的子序列 (subsequence)，即存在一个严格递增的的 X 的下标序列⟨i_1, i_2, … , i_k⟩，对所有j=1, 2, ..., k，满足$x_{i_j} = z_j$。例如，Z = ⟨B, C, D, B⟩是X = ⟨A, B, C, B, D, A, B⟩的子序列，对应的下标序列为⟨2，3，5，7⟩。

给定两个序列 X 和 Y，如果 Z 既是 X 的子序列，也是 Y 的子序列，我们称它是 X 和 Y 的公共子序列 (common subsequence) [11]。例如，如果X = ⟨A, B, C, B, D, A, B⟩，Y = ⟨B, D, C, A, B, A⟩，那么序列 ⟨B, C, A⟩ 就是 X 和 Y 的公共子序列，但并非最长公共子序列 (longest common subsequence, LCS) 。因为 ⟨B, C, B, A⟩ 也是 X 和 Y 的公共子序列，但其长度为 4，大于 ⟨B, C, A⟩ 的长度。⟨B, C, B, A⟩ 是 X 和 Y 的最长公共子序列，⟨B, D, A, B⟩ 也是，因为 X 和 Y 不存在长度大于等于 5 的公共子序列。

基于 LCS 也可以定义两个序列的相似度new_s。LCS 的长度越长，则两个序列的相似度越大。

LCS 从某种意义上来说，可以看作增删代价取 0，改操作换成相同奖励的 Levenshtein 距离。因为求编辑距离过程中，两个序列不需要进行编辑的对应字符一定包含于它们的 LCS 中，所以两个序列的 LCS 长度一定大于编辑距离下两个序列重合的字符数目。因此可以得到

$$\begin{aligned} E_s(a,b) &= 1 - E_d(a,b) \\ &= \frac{\max(|a|,\ |b|) - lev(a,\ b)}{\max(|a|,\ |b|)} \\ &\le \frac{|LCS(a,b)|}{\max(|a|,|b|)} = new_s \end{aligned} \tag{4.1}$$

求解序列对的编辑距离本质上是离散上的最优化问题。Levenshtein 距离的计算对全

域进行检索，使得算法可以跳出极值点，收敛到全局最优解。而基于 LCS 的编辑距离计算，由于 LCS 限制了最优化问题的求解域，求解函数只能在一定范围内迭代，逼近局部最优解，即极小值。

而计算两个序列的 LCS 长度，仍是基于动态规划思想。具体的转移方程可以表述为

$$\mathrm{LCS}(X_i, Y_j) = \begin{cases} \emptyset & if\ \ i = 0\ \ or\ \ j = 0 \\ \mathrm{LCS}(X_{i-1}, Y_{j-1})\hat{}x_i & if\ \ i,j > 0\ \ and\ \ x_i = y_j \\ \max\left(LCS(X_{i-1}, Y_j), LCS(X_i, Y_{j-1})\right) & if\ \ i,j > 0\ \ and\ \ x_i \neq y_j \end{cases} \tag{4.2}$$

而这种方法虽然可以得到编辑相似度的上界，却并没有缩减时间复杂度，仍需要$O(mn)$的时间复杂度。因此我们还需要探索新的上界表示方法。

4.1.2 Jaccard 相似度

Jaccard 相似度 (Jaccard index) 是用于衡量数据集的相似性、相异性和距离的统计量。测量两个数据集之间的 Jaccard 相似度是将所有数据集共有的特征数除以属性数的结果[33]。

$$\mathrm{J}_s(\mathrm{A}, \mathrm{B}) = \frac{|A \cap B|}{|A \cup B|} \tag{4.3}$$

由于计算序列相似度需要考虑同一个元素出现的次数，我们采用多重集合 (multi-sets 或 weighted sets) 来代替集合。具体来说，多重集合 A 被定义为一个索引函数χ_A：当集合元素对应的全集$\mathcal{U} \to \mathbb{N}$，A 中的元素 x 的出现次数（出现次数为 0 的元素不在集合中记录）。两个多重集合的交集的索引函数是函数的最小值，而对于并集来说，则是最大值。所以我们引入加权 Jaccard 相似度的概念，见式 4.4。

$$\mathrm{J}_s(\mathrm{A}, \mathrm{B}) = \frac{\sum_{x \in \mathcal{U}} \min(\chi_A(x), \chi_B(x))}{\sum_{x \in \mathcal{U}} \max(\chi_A(x), \chi_B(x))} \tag{4.4}$$

Jaccard 相似度是一种索引函数只取值{0,1}的特殊加权 Jaccard 相似度。为了方便描述，在之后的文章中，Jaccard 相似度特指加权 Jaccard 相似度。

4.1.3 Jaccard 相似度是编辑相似度的上界

Jaccard 相似度是编辑相似度的上界。证明如下：

存在序列 a 和 b，集合 A、B 分别与之对应。LCS 是一种包含集合相对位置信息的交集，而$\mathrm{A} \cap \mathrm{B}$不要求位置信息，重合率会更高。因此$|LCS| \leq |A \cap B|$。例如，序列 a=ABC，序列 b=ACB，a 和 b 的 LCS 长度只能是 2，但是交集大小是 3。因为 BC 的倒置会影响 LCS 中的基于位置关系的转移方程，但并不影响集合元素数量关系。

假设$|a| \geq |b|$，则$\max(|a|,|b|) = |a|$。$|A \cup B| = (|A| + |B| - |A \cap B|) \geq |A| = |a|$。所以有

$$\frac{|A \cap B|}{|A \cup B|} \geq \frac{|LCS(a,b)|}{\max(|a|,|b|)} \geq E_s(a,b) \tag{4.5}$$

利用 Jaccard 相似度可以实现低编辑相似度序列的快速过滤。对于序列S_1, S_2，存在两个常数θ和α $(\theta, \alpha \epsilon [0,1],\ \theta \geq \alpha)$，只有当 Jaccard 相似度$J_s(S_1,S_2) < \alpha$，才有编辑相似度$E_s(S_1,S_2) < \theta$的情况存在。我们考虑它的逆否命题：若$E_s(S_1,S_2) < \theta$的情况不存在，即任意$E_s(S_1,S_2) \geq \theta$，则由式 4.5 可推导得$J_s(S_1,S_2) \geq E_s(S_1,S_2) \geq \theta$。因为$\theta \geq \alpha$，所以$J_s(S_1,S_2) \geq \alpha$。从而原命题成立。

因此使用阈值为α的$J_s(S_1,S_2)$对序列进行过滤时，虽然仍存在假阳性 (false positive, FP) 的情况，过滤后的序列中依然有$E_s(S_1,S_2) < \theta$的对；但是假阴性 (false negative, FN) 可以被完全消除，即所有$E_s(S_1,S_2) \geq \theta$的序列对都被保留。

4.2 基于 Jaccard 相似度的快速过滤算法设计

4.2.1 Jaccard 相似度的计算问题

尽管用 k-mer 的方式来表示每条序列，然后再通过判断每条序列中 k-mer 集合的重叠率，就可以得出序列对的 Jaccard 相似度。但是，一条序列得到的 mer 集合的元素个数并不少。换而言之，长度为 L 的序列有 L-k+1 的 k-mers。k 的取值影响相似度度量的准确性以及内存大小。k 取值过小：特别是当 k=1 时，k-mer 退化为单字符，不携带任何序列位置信息，大大降低了相似度的可靠性。k 取值过大：假定 k=5，那么每个 mer 中就会包含 5 个字符，序列需要的内存空间大概是原序列大小的 5 倍。因为原序列中的每个字符（除去开头和结尾的字符有特殊性）都会出现在 5 个 mer 中。以 k-mer 的方式来存储序列会消耗大量的内存。

因此，我们需要把上面的 k-mer 集合替换成规模小很多的散列值集合。这样，通过比较两个序列的散列值集合的相似度，就可以估计 k-mer 的相似度。散列值会损失部分精度，因此这样得到的相似度只是原来相似度的近似值。

4.2.2 最小哈希 MinHash

MinHash 是一种快速估计两个集合的相似程度，即 Jaccard 相似度$J_s(S_1,S_2)$，而无需直接计算集合之间交集和并集的技术。

给定一个全集$\mathcal{U}$和它的子集$\mathcal{S} \subseteq \mathcal{U}$，MinHash 定义如下：假设存在包含 D 个散列函

数（或随机排列）的函数族 $\{\hbar_d\}_{d=1}^{D}$，$\hbar(x)$ 表示一个将$\mathcal{U}$上的关键字x映射到$\mathcal{S}$上的散列函数。$\hbar_{min}(\mathcal{U})=\mathrm{y}$ 被定义为如果 $\mathrm{y}\in\mathcal{U}$，且与$\mathcal{U}$的所有元素中比较 $\hbar(y)$ 具有最小散列值，这也被称作$\mathcal{S}$的一个 MinHash。

假设集合$\mathrm{S}=\{s_1,s_2,s_5,s_7,s_{10}\}$被散列成$\{4,6,2,1,8\}$，那么$\hbar_{min}(\mathrm{S})=s_7$。

将$\hbar_{min}$应用到集合 A 和 B 上，假设不存在哈希冲突，$\hbar_{min}(A)=\hbar_{min}(B)$当且仅当在$\mathrm{A}\cup\mathrm{B}$的所有元素中，具有最小哈希值的元素在$\mathrm{A}\cap\mathrm{B}$中。这种情况发生的概率恰好等于 Jaccard 相似度。因此，

$$\Pr[\hbar_{min}(A)=\hbar_{min}(B)]=\mathrm{J}_s(\mathrm{A},\mathrm{B}) \tag{4.6}$$

一个散列函数很容易受到特殊数据的影响。为了获得无偏相似度估计，我们进行多个独立的随机排列：

$$\widehat{\mathrm{J}_s}(\mathrm{A},\mathrm{B})=\frac{\sum_{d=1}^{D}\mathrm{if}(\hbar_{d_{min}}(A)=\hbar_{d_{min}}(B))}{D} \tag{4.7}$$

如果 state 为真，$\mathrm{if(state)}=1$；否则$\mathrm{if(state)}=0$。当$\mathrm{D}\to\infty$的时候，$\widehat{\mathrm{J}_s}(\mathrm{A},\mathrm{B})\to\mathrm{J}_s(\mathrm{A},\mathrm{B})$。

而由 2.1.6 分析可知，哈希冲突无法避免，但如果我们在 MinHash 中使用全域散列函数族，可以最大程度减少这种冲突，使得冲突概率不高于$1/|\mathcal{S}|$。因此，使用 MinHash 来衡量 Jaccard 相似度只是一种近似估计，其准确度受到 k-mer 的参数选择以及散列函数族的选择等影响。

假设我们在函数族中使用了 k 个随机选择的哈希函数，满足$\hbar_{min}(A)=\hbar_{min}(B)$的函数个数为 y。$\mathrm{y/k}$可以视为$\mathrm{J}_s(\mathrm{A},B)$的无偏估计量。根据切诺夫界 (Chernoff bound)，误差期望$\mathrm{E}(|\mathrm{y/k}-\mathrm{J}_s(\mathrm{A},B)|)=\mathrm{O}(1/\sqrt{k})$。因此如果 k=400 时，预期误差仅为 0.05。

例如，存在$S_1=\{a,d,e\}$，$S_2=\{c,e\}$，全集$\mathcal{U}=\{\mathrm{a,b,c,d,e}\}$。集合可以表示为表 4.1 的形式。

表 4.1　集合表示

行号	元素	S_1	S_2	类别
1	a	1	0	Y
2	b	0	0	Z
3	c	0	1	Y
4	d	1	0	Y
5	e	1	1	X

其中，列表示集合，行表示相应的元素，对应的值为 1 表示某个集合中有某个值，0 则代表没有。S_1和S_2的每一行元素可以按取值是否相等，分为以下三类：

（1）X 类，两者值均为 1。如表 4.1 中第 5 行，两个集合都具有元素 e；

（2）Y 类，两者取值不相同，一个为 0，另一个为 1；如表 4.1 中第 1 行，表示S_1中具有元素 a，而S_2中没有；

（3）Z 类，两者值均为 0。如表 4.1 中第 2 行，表示S_1和S_2中都没有元素 b。

设哈希函数$h_1 = (i+1)\%5$，其中 i 代表行号。h_1作用于集合S_1和S_2，可以得到表 4.2 的结果表示。

表 4.2　h_1下的集合表示

行号	元素	S_1	S_2	类别
1	e	1	1	X
2	a	1	0	Y
3	b	0	0	Z
4	c	0	1	Y
5	d	1	0	Y
MinHash		e	e	

此时，$h_{min}(S_1)$=e，$h_{min}(S_2)$=e，即S_1和S_2的 MinHash 值都是 e。

所有标记为 Z 类的行都可以被忽略，因为 Z 对应的元素完全可以从全集$\mathcal{U}$中删除，对两个集合的相似度计算不产生任何影响。我们讨论在哈希函数h_1均匀分布的情况下，由于h_1将原始行号均匀分布到新的行号，那么就可以认为在新行号排列下，任意一行出现 X 类的情况的概率为$|X|/(|X|+|Y|)$。所以，第一个出现 X 类行的行出现 X 类的概率也为$P = |X|/(|X|+|Y|) = J_s(X,Y)$。

继续假设存在哈希函数$h_2 = (i-1)\%5$，那么可以得到表 4.3。

表 4.3　h_2下的集合表示

行号	元素	S_1	S_2	类别
1	b	0	0	Z
2	c	0	1	Y
3	d	1	0	Y
4	e	1	1	X
5	a	1	0	Y
MinHash		d	c	

哈希函数族中存在 h 个散列函数 $(h \ll \mathcal{U})$，因此我们需要为每个集合都计算 h 次 Minhash 值，而后用这 h 个 Minhash 值组成一个摘要表，来表示当前集合，如表 4.4 所示。

表 4.4 $\hbar_1$和$\hbar_2$下的MinHash摘要

哈希函数	S_1	S_2
$\hbar_1 = (i+1)\%5$	e	e
$\hbar_2 = (i-1)\%5$	d	c

令 M 表示 Minhash 摘要中集合对应行相等的次数，即函数族中满足两个集合最小散列值相等的函数个数。表 4.4 中，M=1，因为在哈希函数$\hbar_1$散列下，S_1和S_2的 MinHash 值相同，而在$\hbar_2$下则不同。那么可以发现，

$$\mathrm{M} \sim \mathrm{B}(\mathrm{h}, \mathrm{J}_s(S_1, S_2)) \tag{4.8}$$

M 符合次数为 h，概率为$\mathrm{J}_s(S_1, S_2)$的二项分布，而期望$\mathrm{E}(\mathrm{M}) = \mathrm{h} * \mathrm{J}_s(S_1, S_2) = 2 * (1/4) = 0.5$。也就是说，每运用 2 个散列函数计算 Minhash 摘要时，可以期望有 0.5 个元素对应相等。综上所述，MinHash 在对原集合进行散列的基础上，保证了集合的相似度不受破坏。

4.3 面向大规模集群的 L-MinHash 聚类算法设计

4.3.1 局部敏感哈希 Locality-Sensitive Hashing

局部敏感哈希 （Locality-Sensitive Hashing, LSH） 是一种可以以较高概率将相似的输入项散列到相同的槽中的算法技术。

与传统的哈希算法理念不同，LSH 将产生 Hash 冲突的概率最大化，而并非极力避免这种冲突。LSH 能使 2 个相似度很高的数据以较高的概率映射成同一个哈希值，而 2 个相似度很低的数据以极低的概率映射成同一个哈希值的特性。因此 LSH 能高效处理海量高维数据的聚类和最近邻搜索问题。

我们提出一种问题场景。存在长度为 d 的两个碱基序列S_1和S_2，$\mathrm{r} < \mathrm{d}$，如果S_1和S_2之间相差不超过 r 个字符，就称S_1和S_2相似。给定一个序列 G ($|\mathrm{G}| = \mathrm{n} \gg \mathrm{d}$) ，对于每一个询问的长度为 d 的序列，在 G 中找出相似的子串。

如果我们采用直接比较的方式，时间复杂度为$\mathrm{O}(\mathrm{nd})$。我们需要找到一种时间复杂度与 d 相关（关于 d 的多项式）而与 n 无关的算法。

在预处理阶段，将 G 中所有长度为 d 的子串通过随机选择的局部敏感哈希 (LSH) $h_1, h_2, \ldots, h_l$分别散列到哈希表$A_1, A_2, \ldots, A_l$的集合中。对于每一个长度为 d 的询问串 q，查找A_1中的$h_1(q)$，A_2中的$h_2(q)$，…，A_l中的$h_\mathrm{l}(q)$，令并集$\mathrm{Q} = A_1[h_1(q)] \cup A_2[h_2(q)] \cup$

$\ldots \cup A_{\mathrm{l}}[h_{\mathrm{l}}(q)]$。那么最后只需要将 Q 中的每一项与 q 进行相似度计算，就能完成对 n 的降阶。

我们定义一种局部敏感哈希函数h_{i}，如果对于一个与询问串 q 相似的串 g，h_{i}存在 p 的概率返回 g。

$$\Pr\left(\mathrm{g} \in h_{\mathrm{i}}(q)\right) = p \tag{4.9}$$

$$\Pr\left(h_{\mathrm{i}}(q) = h_{\mathrm{i}}(g)\right) = p \tag{4.10}$$

由于存在哈希函数族$h_1, h_2, \ldots, h_l$，因此 g 被找到的概率为

$$\begin{aligned}\Pr(\mathrm{g} \in \mathrm{Q}) &= 1 - \Pr(\mathrm{g} \notin \mathrm{Q}) \\ &= 1 - \textstyle\sum_{i=1}^{l} \Pr\left(\mathrm{g} \notin h_{\mathrm{i}}(q)\right) = 1 - (1-p)^l\end{aligned} \tag{4.11}$$

假设 p=0.1，我们能得到如下关系表 4.5。

表 4.5　p=0.1时，l与函数结果的关系

l (p=0.1)	$1-(1-p)^l$
1	0.1
2	0.19
4	0.344
8	0.570
16	0.815
32	0.966

分析可知，随着l的小幅度增大，g 被找到的概率会无限趋近 1。

当$\mathrm{n} = |\mathrm{G}|$的时候，算法的空间复杂度显然为$\mathrm{O}(l\mathrm{n})$。在预处理阶段，需要对每一个子串计算一遍散列值，因此时间复杂度为$\mathrm{O}(l\mathrm{n})$；而在查找阶段，运用 MinHash 估计聚类后数据的 Jaccard 相似度，时间复杂度为$\mathrm{O}(l\mathrm{d} + |\mathrm{Q}|\mathrm{d})$。

对于解决前文提出的问题，我们可以设计这样一个 LSH 函数，对于 G 中每个长度为 d 的子串 s，从$\{1 \ldots \mathrm{d}\}$均匀随机选择$k\ (k < d)$ 个置换$i_1, i_2, \ldots, i_k$。定义 LSH 函数$\mathrm{H} = \{\mathrm{h}: \sum^d \to \sum^k\}$，其中$\sum = \{a, c, g, t\}$，那么

$$\mathrm{h(s)} = < \mathrm{s}[i_1], \mathrm{s}[i_2], \ldots, \mathrm{s}[i_k] > \tag{4.12}$$

例如，当 k=2，$i_1 = 2$，$i_2 = 5$时，h(“acgtacgt”) = “ca”。

每一位 i 在集合中都有 d 种选择，因此总共有d^k个不同的置换函数 h。

当$\mathrm{i} = 1 \ldots l$，随机选择一个 LSH 函数h_1，对于每个符合条件的子串的起始位置 v，将 $(h_i(\mathrm{G}[\mathrm{v} \ldots \mathrm{v} + \mathrm{d} - 1]), \mathrm{v})$ 放入散列表A_i中。为了方便接下来的多组询问，在预处理阶段我们将每个散列表A_i中的元素按照h_i值的大小进行排序。因此这部分总时间复杂度近

似 $O(lnd\log n)$ 。在询问阶段，对于$\mathrm{i}=1\ldots l$，计算出$h_i(q)$，在每个A_i中二分查找与$h_i(q)$相同的值对应的位置信息 v，最后只需要比较序列 q 和$\mathrm{G}[\mathrm{v}\ldots\mathrm{v}+\mathrm{d}-1]$的相似度。时间复杂度为 $O(ld\log n)$ 。

因为函数 h 具有以较高的概率将相似的序列散列到同一个槽中的性质。 如果序列s_1和s_2相似（即序列间差异的位数小于 r），有

$$\Pr\big(h(s_1)=h(s_2)\big)\geq(1-\frac{r}{d})^k \tag{4.13}$$

假阴性情况（本来相似的序列对在所有散列表中都不在一个槽中）发生的概率f_n为

$$f_n\leq[1-(1-\frac{r}{d})^k]^l \tag{4.14}$$

对于阈值ρ_{f_n}，有$f_n\leq\rho_{f_n}$。因此敏感度就可以定义为$1-\rho_{f_n}$，这通常是提前设置的。也就是说，如果要求 95%的相似序列对被找到，ρ_{f_n}就需要调整为 0.05。

对于一个固定的值l，我们需要找到一个 k 满足

$$f_n\leq[1-(1-\frac{r}{d})^k]^l\leq 1-\rho_{f_n} \tag{4.15}$$

假设每个散列函数的假阳性概率为f_p。LSH 函数会返回一组包括真解和假阳性解的结果。因为要对所有假阳性结果检验来筛除不满足的解，f_p对运行时间的影响为$\mathrm{O}\big(lnf_p\big)$。

假设一个随机生成的长度为 d 的 DNA 序列 s 满足$f_a=f_c=f_g=f_t=0.25$的频率分布，那么有$f_p=\Pr\big(h(s)=h(\mathrm{q})\big)=0.25^k$。所以$\mathrm{O}\big(lnf_p\big)=\mathrm{O}(ln0.25^k)$，因此我们需要尽可能地选择较大的参数 k。若 d=100，r=20，将l视为固定的值，求解 k，有

$$[1-(1-\frac{20}{100})^k]^l=(1-0.8^k)^l\leq 0.05 \tag{4.16}$$

$$1-e^{\frac{\ln 0.05}{l}}\leq 0.8^k \tag{4.17}$$

$$k\leq\frac{\ln(1-e^{\frac{\ln 0.05}{l}})}{\ln 0.8} \tag{4.18}$$

当l取不同的数值时，对应的更为直观的结果可见表 4.6。

表 4.6　l, k与lf_p的对应关系

l	$\mathrm{k}\leq$	$l\times 0.25^k$
20	8.8	9.53E-05
30	10.5	1.34E-05
40	11.8	3.23E-06
50	12.7	1.06E-06
60	13.5	4.21E-07
80	14.8	9.77E-08
100	15.8	3.13E-08

虽然使用 LSH 不能保证找到解决方案，但找到解决方案的概率较高。这个概率可以通过增加 LSH 函数的数量来提升。

4.2.2 中提出的 MinHash 方法，仍然需要遍历所有的集合对，才能挖掘所有相似的集合对，无法对 $O(n^2)$ 的复杂度进行优化。接下来将以 LSH 方法解决集合间两两比对的难题进行举例说明。

现在有 5 个集合，对应的 Minhash 摘要见表 4.7。

表 4.7　集合的Minhash摘要

	S_1	S_2	S_3	S_4	S_5
区间1	b	b	a	b	a
	c	c	a	c	b
	d	b	a	d	c
区间2	a	e	b	e	d
	b	d	c	f	e
	e	a	d	g	a
区间3	d	c	a	h	b
	a	a	b	b	a
	d	e	a	b	e
区间4	d	a	a	c	b
	b	a	c	b	a
	d	e	a	b	e

如上的集合摘要采用了 12 个不同的哈希函数散列，之后将结果分成了 B = 4 个区间。根据（4）中的分析，任意两个集合 (S_1, S_2) 对应的 Minhash 值相等的概率 r = $J_s(S_1, S_2)$。在区间 1 中，$\Pr(S_1 = S_2) = r^3$。因此如果$J_s(S_1, S_2)$足够大，S_1和S_2在区间 1 内的三个最小哈希值就可能完全一致，S_1和S_2可以被认定为相同。也就是说，$\Pr(S_1 \neq S_2) = 1 - r^3$。

现在有 4 个区间，其他区间与第一个区间等价，所以有$\Pr(S_1 \neq S_2) = (1 - r^3)^4$。那么至少存在一个区间上$S_1 = S_2$的概率$\Pr(S_1 = S_2) = 1 - (1 - r^3)^4$。函数图像如下。

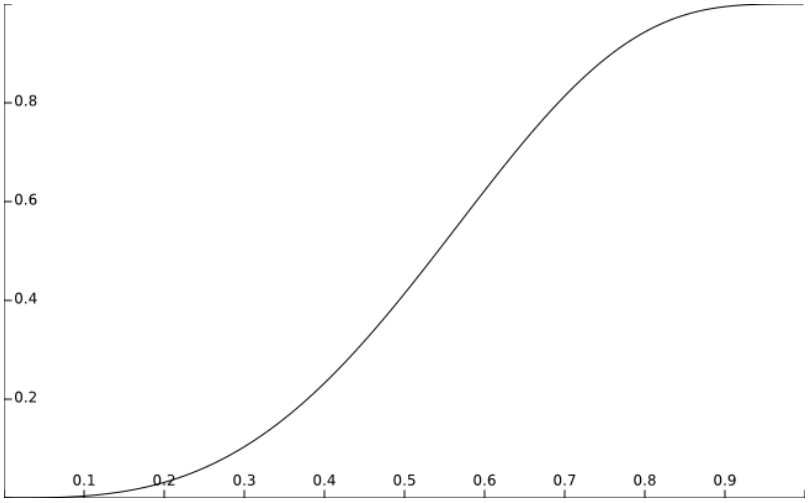

图 4.1　4个区间，12个哈希函数下的冲突概率函数

如果令总区间个数为 B，每个区间内的行数为 C，那么上面的公式可以表示为 Pr（B 个区间中至少有一个区间中两个集合相等）=$1-(1-r^{C})^{B}$。

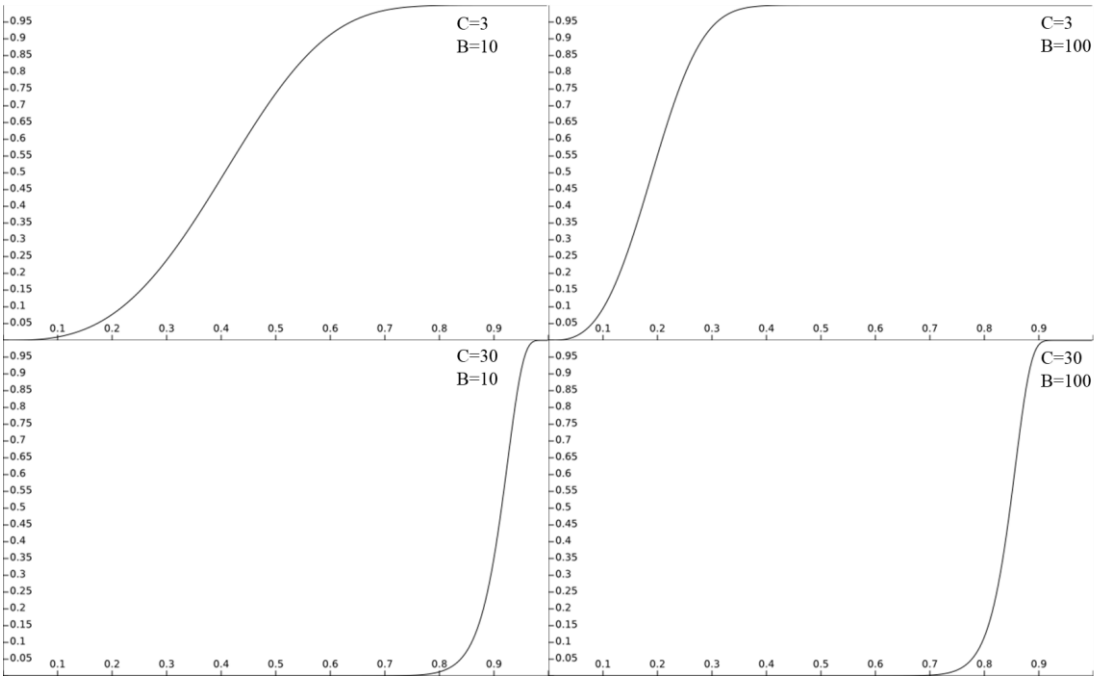

图 4.2　不同B和C的值对冲突概率函数的影响

令 r = 0.4，C=3，B = 100。上述公式计算的概率为 0.9986585。这表明两个 Jaccard 相似度为 0.4 的集合在至少一个区间内冲撞的概率达到了 99.9%。因此我们只需要选取合适的 B 和 C，和一个冲撞率很低的 hash 函数，就可以使相似的集合至少在一个区间内冲撞，这样也就达成了将相似的集合聚类的目的。因为 B 和 C 都是常量，所以 LSH 的时间复杂度是线性O(n)的。由于聚到一起的集合相比于整体较少，所以在这小范围内

互相比较的时间开销也可以计算为常量，那么总体的计算时间也是O(n)。

在上文基础上，我们引入离散空间上的广义局部敏感哈希 (Generalized Locality Sensitive Hashing) 的定义。假设有一个在 d 维空间$\mathbb{R}^d$上，包含 n 个数据点$p = (p_1, \dots, p_d)$的集合 P。对于任意两点 p 和 q，它们之间的距离定义为

$$||p-q||_s = \left(\sum_{i=1}^{d}|p_i - q_i|^s\right)^{\frac{1}{s}} \tag{4.19}$$

对于任意的 S>0，这个距离函数被称为标准化l_s。如果$||p-q||_s \le R$，则称 p 是 q 的 R 近邻 (R-near neighbor) 。如图 3.5，p_1，p_2，p_3都是 q 的 R 近邻，p_4则不是。

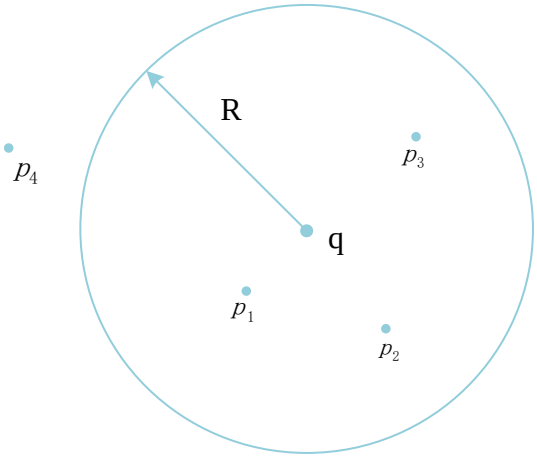

图 4.3　离散空间上的R近邻

从几何学的角度理解 LSH，本质就是一种投影。首先根据散列函数进行散列操作，这个函数在二维空间上可以表示为一条直线，使得空间上相邻的点投影在这条直线的同一段区间里，即散列到同一个桶中。这种 LSH 和连续函数 mod k 的散列方式是有本质区别的，无法进行精准的归类，让所有相邻的点都被散列在一个桶中，而不相邻的点也无法保证一定不在同一个桶中。但是 LSH 在一定程度上可以区分查询点的相近点和较远点。

上述最近邻 (near neighbor, NN) 问题可以扩展到近似最近邻(Approximate near neighbor, ANN) 问题[34] ，也被称为 Randomized c-approximate R-near neighbor ((c,R)-NN)。

给定点集P $\subseteq \mathbb{R}^d$，查询点q $\in \mathbb{R}^d$，查询范围 R>0 和近似因子 c>1，(c, r)-NN 问题的输出如下：若存在$p \in$ P，满足$||p-q||_s \le$ R，则输出某点$p' \in$ P，满足$||p'-q||_s \le$ cR。

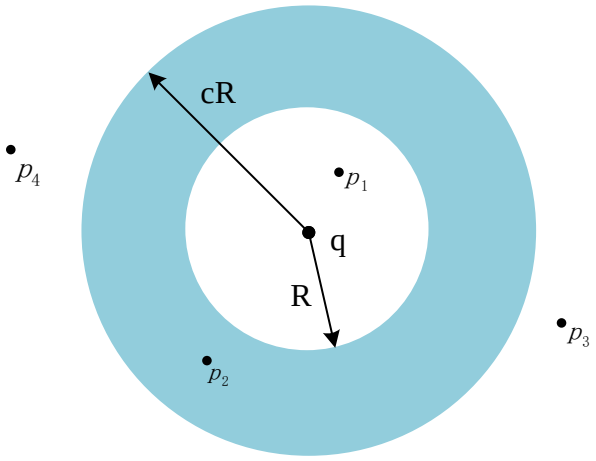

图 4.4　离散空间上的ANN

我们需要让在相近的 R 近邻（图 4.4 中白色部分）进行 hash 冲突的概率更大，而 cR 近邻（图 4.4 中蓝色部分）则越小。但是原始的 LSH 则会使得 R 近邻和 cR 近邻都尽可能的大，会导致查找的结果很多时候并不能得到一个较好的回馈。

构造一个数据结构 LSH，对于任意查询$q \in \mathbb{R}^d$，如果在 P 中存在 q 的 R 近邻，能以 1-δ (δ>0) 的概率给出 P 中 q 的 cR 近邻。跟连续空间上的 LSH 相似，如果 LSH 函数的规模扩大一倍，这个概率将从 1-δ上升到 $1-\delta^2$。当我们对离散空间进行放缩，使 R=1 时，ANN 问题就变成了 c 近似最近邻问题 (c-approximate near neighbor problem, c-NN) 。

由此，我们可以正式地定义局部敏感哈希。如果对于任意两点p, q，从 H 中均匀随机选择一个函数 g，有

（1）如果$||p-q|| \le R$，$\Pr(g(p)=g(q)) \ge P_1$；

（2）如果$||p-q|| \ge cR$，$\Pr(g(p)=g(q)) \le P_2$；

（3）$0 < P_2 < P_1 < 1$，

则称哈希函数族 H 是局部敏感的，也称 Gapped LSH。P_1和P_2间的距离可能非常小，需要额外添加步骤放大两者的间隙。

我们选择l个函数$h_1, h_2, \dots, h_l$，其中$h_j(q) = \left(g_{j1}(q), \dots, g_{jk}(q)\right)$，而$g_{ji}$是从 H 中均匀随机选择的函数 $(1 \le j \le l,\ 1 \le i \le k)$ 。使用这些函数将 P 转换到l个散列表中，之后对于每一组询问q，查找这些散列表以提取数据点进行验证。如果$||p-q|| \le R$，$\Pr\left(h_j(p)=h_j(q)\right) \ge {P_1}^k$；如果$||p-q|| \ge cR$，$\Pr\left(h_j(p)=h_j(q)\right) \le {P_2}^k$。

对于一个 c-NN 问题，我们先找到 $L = tl$ 个点验证，然后中止。定义事件 E1 为假阳性的数量少于 $L = tl$ 个数据点（即总哈希冲突次数$< tl$），E2 为以 1-θ的概率找到一

个解决方案。我们需要根据给定的t和θ，找到最佳参数l和k。

$||p-q|| \geq cR$碰撞的概率$\Pr\left(h_j(p)=h_j(q)\right) \leq {P_2}^k = \frac{1}{n}$（n 表示数据点的数量），因此可以推导得$k = -\frac{\ln n}{\ln P_2}$。

q 在散列表 a 中发生冲突的期望 E(#collisions with q in a table) ≤ 1 ，所以l个散列表中 q 发生冲突的期望 E(total #collisions with q in l tables) ≤ l 。根据马尔可夫不等式 (Markov Inequality) 可知，Y 为仅假设在非负值上的随机变量，对于任意t ∈ $\mathbb{R}^+$，都有$\Pr(Y \geq t) \leq \frac{E(Y)}{t}$。因此，

$$\Pr(\text{total \#collisions} \geq tl) \leq \frac{l}{tl} = \frac{1}{t} \tag{4.20}$$

$$\Pr(< tl \text{ collisions}) \geq 1 - \frac{1}{t} \tag{4.21}$$

如果存在 NN（即$||p-q|| \leq R$），找到一个 ANN 的概率为

$$\begin{aligned} &\Pr\left(h_1(p)=h_1(q) \vee \ldots \vee h_l(p)=h_l(q)\right) \\ &= 1 - \Pr\left(h_1(p) \neq h_1(q) \wedge \ldots \wedge h_l(p) \neq h_l(q)\right) \\ &\geq 1 - \left(1-{P_1}^k\right)^l \approx 1 - e^{-l{P_1}^k} = 1 - \theta \end{aligned} \tag{4.22}$$

在$\theta = e^{-l{P_1}^k}$，${P_2}^k = \frac{1}{n}$的条件下，

$$l = -\ln\theta / {P_1}^k = -\ln\theta / n^{-\frac{\ln P_1}{\ln P_2}} = -\ln\theta \times n^{\frac{\ln P_2}{\ln P_1}} = O(n^\rho)(\rho = \frac{\ln P_2}{\ln P_1} < 1) \tag{4.23}$$

根据上述分析，可知

$$\Pr(E1 \cap E2) \geq 1 - (1 - \Pr(E1)) - (1 - \Pr(E2)) = 1 - (\frac{1}{t} + \theta) \tag{4.24}$$

令$\delta = (\frac{1}{t} + \theta)$，事件 E1 和 E2 同时为真的概率就是$\Pr(E1 \cap E2) \geq 1 - \delta$。$\Pr(E1 \cap E2) = 1 - \delta$当且仅当所有假阳性情况都被找到，$\Pr(E1) = 1$。算法空间复杂度$O(dn + nl) = O(dn + n^{1+\rho})$，搜索部分的时间复杂度$O(dl) = O(dn^\rho)$。

对于一个 R 近邻问题，我们在l个散列表中搜索询问串 q 的哈希值，合并所有产生哈希冲突的项，对它们进行验证。关于假阳性和敏感度部分的分析与 c-NN 相同，空间复杂度也是一样的。而在搜索部分的时间复杂度上，需要额外考虑解个数的期望，E(#false positives + #occurrences of solutions)=$O(dl) + O\left(d \times {P_1}^k \times l \times \#solutions\right)$。由于${P_1}^k \times l = n^{-\rho} \times n^\rho = 1$，所以化简得$O(dn^\rho + d \times \#solutions)$。

对于一个长度为 d 的子串，

(a) 如果$||p-q|| \leq R$，$\Pr\left(g(p)=g(q)\right) \geq \frac{d-r}{d} = P_1$；

(b) 如果$||p-q|| \geq cR$，$\Pr(g(p)=g(q)) \leq \frac{d-cr}{d} = P_2$；

所以$\rho = \frac{\ln P_2}{\ln P_1} = \frac{\ln 1-\frac{r}{d}}{\ln 1-\frac{cr}{d}} \leq \frac{1}{c}$，$l = O(n^{\rho}) = O\left(n^{\frac{1}{c}}\right)$。

4.3.2 L-MinHash 是一种关于($s1$,$s2$,$p1$,$p2$)敏感的 LSH

与 MinHash 类似，L-MinHash 是通过在 k-mers 上使用置换来随机选择的。为了降低哈希冲突概率，我们并非只选取 1 个最小的哈希值，而是l个。取l个散列值最小的 k-mers，按照散列值值从小到大记录 k-mers。

此外，L-MinHash 必须处理重复的 k-mers。同一 k-mer 的两个副本出现在序列中的不同位置，将其视作同一集合元素会损失精度。因此我们通过在 k-mers 后附加“出现次数”，使其唯一。

更准确地说，对于长度$|S| = n$的字符串 S，考虑 k-mers 对及其出现次数的集合$\mathcal{M}_k^w(S)$。如果序列 S 中有 x 个 m 的副本，那么集合$\mathcal{M}_k^w(S)$中有 x 对关于 m 的元素，形如$(m, 0), \dots, (m, x-1)$。副本的出现次数表示在序列 S 中，该特定副本的左侧序列中存在的其它副本 m 的数量。也就是说，如果 m 是 S 中位置在 i 处的 k-mer（即$m = S[i:k]$），它的出现次数是$|\{j \in [i]|S[j:k] = m\}|$。该集合是关于字符串 S 的 k-mer 的“多重集合”，或者叫 k-mers 的“加权集合”，其中出现次数就是 k-mer 的权重（因此有上标 w）。我们把 k-mer 的 (m, i) 对和出现数称为“对应”的 k-mer。

$\sum^k \times [n]$上的一个置换 π 定义了两个函数 $\mathfrak{h}_{l,\pi}^w$ 和 $\mathfrak{h}_{l,\pi}$。$\mathfrak{h}_{l,\pi}^w(S) = ((m_1, o_1), \dots, (m_l, o_l))$ 是一个长度为 l，由项 $\mathcal{M}_k^w(S)$ 组成的向量。根据置换π，对(m_i, o_i)是最小的 l 个 $\mathcal{M}_k^w(S)$ 中的项。

向量 $\mathfrak{h}_{l,\pi}(S) = (m_1, \dots, m_l)$ 只包含来自 $\mathfrak{h}_{l,\pi}^w(S)$ 的 k-mers。L-MinHash 方法定义为哈希函数集 $\mathcal{H}_{k,l} = \{\mathfrak{h}_{l,\pi} | \sum^k \times [n]$*上的置换* $\pi\}$ 上的均匀分布。

对于极端情况，当$l = n - k + 1$时，向量包含覆盖整个序列 S 的所有加权 k-mers。在这种情况下，L-MinHash 值的相等意味着序列的完全一致。

在另一种极端情况下，当$l = 1$时，向量仅包含一个 k-mer，L-MinHash 退化成 MinHash。

两个序列的加权 Jaccard 相似度 $J^w(S_1, S_2)$ 是其 k-mer 的加权 Jaccard。由于 k-mers 在 $\mathcal{M}_k^w(S)$ 中的出现次数是唯一的，因此加权 Jaccard 相似度等效定义为$J^w(S_1, S_2) = J(\mathcal{M}_k^w(S_1), \mathcal{M}_k^w(S_2))$。

设$\mathcal{H}$是定义在集合$\mathcal{U}$（全集）上的散列函数族。当

$$\mathrm{s(x,y)} \geq s_1 \Rightarrow Pr_{\mathfrak{h}\in\mathcal{H}}[\mathfrak{h}(x)=\mathfrak{h}(y)] \geq p_1, \tag{4.25}$$

$$\mathrm{s(x,y)} \leq s_2 \Rightarrow Pr_{\mathfrak{h}\in\mathcal{H}}[\mathfrak{h}(x)=\mathfrak{h}(y)] \leq p_2, \tag{4.26}$$

则称集合$\mathcal{H}$上的概率分布对相似度$\mathcal{s}$关于(s_1,s_2,p_1,p_2)敏感的。其中$s_1 \geq s_2$，$p_1 \geq p_2$。如果存在一组散列函数的分布关于 (s_1,s_2,p_1,p_2) 敏感，则允许使用 Gapped LSH 对相似序列进行聚类。在上面的定义中，具体概率取决于$\mathcal{H}$中对任意 x, y$\in \mathcal{U}$ 构造的哈希函数的选择。在 Gapped LSH 中，相似元素之间哈希冲突的概率上升 $(\geq p_1)$ ，而对于不同元素，哈希冲突的概率较小 $(\leq p_2)$ 。

因为$\ell = 1$时，L-MinHash 等价于 MinHash；$l = \mathrm{n-k+1}$时，哈希又失去意义。因此在下文的讨论中，我们规定$l \in [2, \mathrm{n-k}]$。因此，我们需要证明对于任意 $\ell \in [2, \mathrm{n-k}]$，$1 > \mathrm{s_1} \geq \mathrm{s_2} > 0$，存在函数 $\mathrm{p^1_{n,k,l}}(\cdot)$ 和 $\mathrm{p^2_{n,k,l}}(\cdot)$，L-MinHash 在编辑距离上对 $\left(\mathrm{s_1, s_2, p^1_{n,k,l}}(\cdot), \mathrm{p^2_{n,k,l}}(\cdot)\right)$ 敏感。

（1）$\mathrm{s(x,y)} \geq s_1 \Rightarrow Pr_{\mathfrak{h}\in\mathcal{H}}[\mathfrak{h}(x)=\mathfrak{h}(y)] \geq p_1$

假设$\mathrm{S_1}$和$\mathrm{S_2}$是两个长度为 n 的序列，每个序列中 k-mer 的数量n_k=n-k+1。当 n 非正(n≤0) 时，二次项系数$\binom{\mathrm{n}}{\mathrm{k}}$=0，表示从空集中选择元素的方案数为 0。

令$E_s(\mathrm{S_1,S_2}) \geq \mathrm{s_1}$，则有$E_d(\mathrm{S_1,S_2}) = 1 - E_s(\mathrm{S_1,S_2}) \leq 1 - \mathrm{s_1}$。因此编辑距离$\leq n(1-\mathrm{s_1})$。因为一位不匹配或是编辑操作最多影响 k 个 k-mers，所以可以推出能匹配上的的 k-mer 的数量最少为$n_k - kn(1-\mathrm{s_1})$。

当两个序列中所有没匹配上的 k-mers 都互不相同时，并集$\mathcal{M}_k^w(S_1) \cup \mathcal{M}_k^w(S_2)$的大小达到最大。所以有$|\mathcal{M}_k^w(S_1) \cup \mathcal{M}_k^w(S_2)| \leq n_k + kn(1-\mathrm{s_1})$。

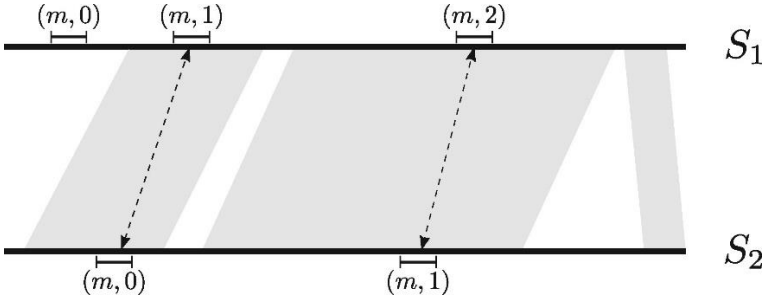

图 4.5 $\mathrm{S_1}$和$\mathrm{S_2}$匹配示意图[32]

我们可以根据匹配中“对应的”的 k-mer 的数量来估计哈希冲突的概率。一个“对应的”的 k-mer 由 m 和它的出现次数组成。匹配成功的 k-mer 对的出现次数可能不一致（如图 4.5 所示）。对于每一个 m 存在这种出现次数不一致的匹配，意味着匹配不成功

的 k-mer 中存在包含 m 的对（如图 4.5 中S_1中的(m,0)）。

设S_1和S_2中出现次数不同的 k-mers 的数量为 x，因此最少可以产生 x+k-1 位不匹配。而因为$E_s(S_1,S_2) \geq s_1$，不匹配的位数最多为$2n(1-s_1)$（只采取增删操作）。所以，$x+k-1 \leq 2n(1-s_1)$，可得$x \leq 2n(1-s_1)-k+1$。而序列加权 k-mer 交集大小可以视作 m 相同的 k-mer 数量减去 m 相同 o 不同的 k-mer 数量。

$$\begin{aligned}|\mathcal{M}_k^w(S_1) \cap \mathcal{M}_k^w(S_2)| &\geq n_k - kn(1-s_1) - x \\ &\geq n-k+1-kn(1-s_1)-2n(1-s_1)+k-1 \\ &= n-n(k+2)(1-s_1)\end{aligned} \tag{4.27}$$

通过置换π，$\mathcal{M}_k^w(S_1) \cup \mathcal{M}_k^w(S_2)$中的每一项都有相同的概率被选入最小的$l$项中。因此，当$E_s(S_1,S_2) \geq s_1$时，产生一次哈希冲突的概率

$$\begin{aligned}Pr\left[\mathfrak{h}_{l,\pi}(S_1) = \mathfrak{h}_{l,\pi}(S_2)\right] &\geq Pr\left[\mathfrak{h}_{l,\pi}^w(S_1) = \mathfrak{h}_{l,\pi}^w(S_2)\right] \\ &\geq \frac{\binom{n-n(k+2)(1-s_1)}{l}}{\binom{n_k+kn(1-s_1)}{l}}\end{aligned} \tag{4.28}$$

这里找到了式 4.25 中s_1和p_1的关系，因此 L-MinHash 是满足式 4.25 的。

（2）$s(x,y) \leq s_2 \Rightarrow Pr_{\mathfrak{h}\in\mathcal{H}}[\mathfrak{h}(x)=\mathfrak{h}(y)] \leq p_2$

因为$\mathfrak{h}_{l,\pi}^w(S)$中所有元素都是唯一的，所以我们可以使用集合$\{\mathfrak{h}_{l,\pi}^w(S)\}$表示向量$\mathfrak{h}_{l,\pi}^w(S)$中的所有元素。令事件 C 表示$\{\mathfrak{h}_{l,\pi}^w(S_1)\}=\{\mathfrak{h}_{l,\pi}^w(S_2)\}$。

假设 m=$|\mathcal{M}_k^w(S_1) \cap \mathcal{M}_k^w(S_2)|$，相当于两个序列加权 k-mer 集交集的大小。当且仅当通过置换π后，l个最小“对应的”k-mers 全部来自$\mathcal{M}_k^w(S_1) \cap \mathcal{M}_k^w(S_2)$，事件 C 成立。因此，

$$Pr\left[\mathfrak{h}_{l,\pi}^w(S_1) = \mathfrak{h}_{l,\pi}^w(S_2)\right] = \frac{\binom{|\mathcal{M}_k^w(S_1)\cap\mathcal{M}_k^w(S_2)|}{l}}{\binom{|\mathcal{M}_k^w(S_1)\cup\mathcal{M}_k^w(S_2)|}{l}} \leq \frac{\binom{m}{l}}{\binom{n_k}{l}} \tag{4.29}$$

现在我们考虑在$\mathcal{M}_k^w(S_1) \cap \mathcal{M}_k^w(S_2)$中的$\mathcal{M}_k^w(S_1)$和$\mathcal{M}_k^w(S_2)$分别按照它们在$S_1$和$S_2$中的出现顺序列出，长度都是 m。在条件 C 下，事件$\mathfrak{h}_{l,\pi}(S_1) = \mathfrak{h}_{l,\pi}(S_2)$相当于通过哈希函数$\mathfrak{h}_{l,\pi}^w$在$\mathcal{M}_k^w(S_1)$和$\mathcal{M}_k^w(S_2)$中选择了长度为$l$的公共子序列。由于集合中的元素没有重复，因此$\mathcal{M}_k^w(S_1)$和$\mathcal{M}_k^w(S_2)$中的公共子序列 (Common Subsequence , CS) 问题等价于在长度为 m 的整数序列中寻找递增子序列 (Increasing Subsequence, IS) 的问题。

$$Pr\left[\mathfrak{h}_{l,\pi}(S_1) = \mathfrak{h}_{l,\pi}(S_2) | (C)\right] \leq \max_{\pi\in[m]!} Pr[pick\ IS\ of\ length\ l\ in\ \pi([m])] \tag{4.30}$$

其中 $[m]!$ 代表 $[m]$ 的全排列。

当$i,n,l\in\mathbb{N}$，$n\geq i\geq l$，对于任意序列长度为 n 且最长上升子序列 (Longest Increasing Subsequence, LIS) 长度为 i 的序列来说，长度为 l 的递增子序列的最大数量为

$$\binom{n}{i}^{l}\binom{i}{l} \tag{4.31}$$

并且这是个紧密边界 (tight bound) 。证明如下。

首先需要引入一种排序算法的概念，称为耐心排序 (patience sorting) 。给出一叠上标数字的纸牌，打乱顺序后一张一张放入堆栈中。只有当现在放入的纸牌上的数字小于某一堆栈栈顶数字时，它才能被放入当前栈顶；如果不存在这样的堆栈，就在右侧新建一个堆栈。要求在所有纸牌放置完成后，最小化桌面上的堆栈的数量。例如，有 5 张纸牌，牌上的数字打乱后分别为 7，2，8，1 和 3。

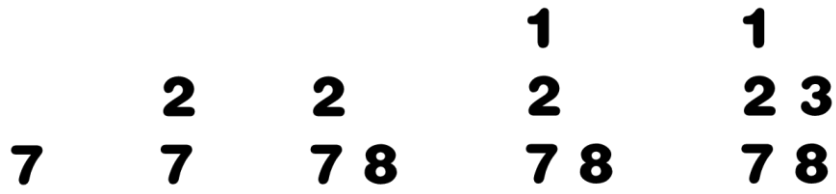

图 4.6　耐心排序示意图

贪心的做法被证明是可行的。即尽可能把牌往靠左的堆栈上放，这样得到的所有堆栈的栈顶组成的序列就是其中一个最长上升子序列。牌堆具有如下两个属性：

(a) 每个堆栈的栈顶组成的序列是递增的。假设按照贪心思想放置牌，会出现栈顶序列不递增的情况，且第一次出现在放置牌 X 时。那么牌 X 左侧的栈顶元素 Y＞X，按照贪心思想，X 至少可以被放置在 Y 所在的栈顶，与假设的前提相悖，所以假设不成立。

(b) 堆栈的数量等于 LIS 的长度。原始牌组的任意上升子序列中，不可能存在两张牌在同一个堆栈。假设存在上升子序列 a，其中两个元素被放置在同一个堆栈中。因为堆栈中元素从栈底到栈顶是单调递减的，所以两个元素中较小的元素一定比较大元素后被放入堆栈中，a 不可能是上升子序列。因此，任意上升子序列最多从每堆中选取一个元素。

当从每个堆栈中取一个元素，且组成上升序列时，这个序列必然是最长上升子序列。因为原序列所有元素都在堆栈中。假设有比这个更长的最长上升子序列 b，说明存在两

个或以上元素来自同一堆栈。而由上述分析，b 就不可能是上升序列。

设$i' \in [l, i]$，有一个长度为 n 的序列 S，其中 LIS 长度为i'。经过耐心排序后，每个堆栈的高度组成向量$\mathrm{s} = (s_0, \dots, s_{i'-1})$。S 中长度为$l$的上升子序列数量的上界

$$\mathrm{g(s)} = \sum_{\substack{A \subset [i'] \\ |A| = l}} \prod_{j \in A} s_j \tag{4.32}$$

因为从每个堆栈中取一个元素不一定就能组成上升序列，所以g(s)是一个比较宽松的上界。

当$s_0 = \cdots = s_{i'-1} = n/i'$时，g(s)取得最大值。集合$\mathrm{C} = \left\{\mathrm{s} = (s_0, \dots, s_{i'-1}) | \sum_j s_j = n\right\}$是一个紧集 (compact set) 。有界定理表明连续函数在紧集上是有界的，并且在集合上的某些点取得最大值与最小值。因此g(s)在 C 上存在最大值。假设在 s 中，存在不是所有s_j都相等。那么我们不失一般性地设$s_{i'-1}$和$s_{i'-2}$不相等。令$\alpha = (s_{i'-1} + s_{i'-2})/2$，向量$\mathrm{s}' = (s_0, \dots, s_{i'-3}, \alpha, \alpha)$。

$$\rho(\mathrm{x}) = \sum_{\substack{A \subset [i'-2] \\ |A| = x}} \prod_{j \in A} s_j \tag{4.33}$$

然后我们从$\mathrm{g}'(\mathrm{s})$分解出既不包含$s_{i'-1}$，又不包含$s_{i'-2}$的项（=$\rho(l)$），包含$s_{i'-1}$或$s_{i'-2}$其中一个的项（=$2\alpha\rho(l-1)$），包含$s_{i'-1}$和$s_{i'-2}$的项（=$\alpha^2\rho(l-1)$）。因为 n 个正数的算术平均数不小于它们的几何平均数。当且仅当这 n 个正数都相等时，它们的算术平均数和几何平均数的值相等。所以有$\alpha^2 > s_{i'-1}s_{i'-2}$。

$$\begin{aligned} \mathrm{g}'(\mathrm{s}) &= \rho(l) + 2\alpha\rho(l-1) + \alpha^2\rho(l-1) \\ &> \rho(l) + (s_{i'-1} + s_{i'-2})\rho(l-1) + s_{i'-1}s_{i'-2}\rho(l-1) \\ &= \mathrm{g(s)} \end{aligned} \tag{4.34}$$

所以当s中包含两个值不同的元素时，g(s)就无法达到最大值。只有当$s_0 = \cdots = s_{i'-1} = n/i'$时，g(s)取得最大值。

$$\max_{s \in C} \mathrm{g(s)} = \sum_{\substack{A \subset [i] \\ |A| = l}} \left(\frac{n}{i'}\right)^l = \left(\frac{n}{i'}\right)^l \binom{i'}{l} \triangleq m_{n,l}(i') \tag{4.35}$$

函数$m_{n,l}(\cdot)$是一个增函数。当$i' = i$的时候，函数取得最大值。

我们考虑这样一个序列$\mathrm{S}(i, n)$，其中 n 能被 i 整除。序列具体形式如下。

$$\underbrace{\left(\frac{n}{i} - 1\right) \dots 0}_{block\ 1} \underbrace{\left(\frac{2n}{i} - 1\right) \dots \left(\frac{n}{i}\right)}_{block\ 2} \dots \underbrace{(n-1) \dots \left(n - \frac{n}{i}\right)}_{block\ i} \tag{4.36}$$

每个块的长度为 n/i ，每个块中的数字按降序排列，但块的开头按升序排列。块 j 由一个上升序列 $(jn/i - 1) \dots ((j-1)n/i)$ 组成。

当耐心排序算法应用于序列 $S(i,n)$ 时，堆栈从下到上、从左到右逐个填充，并且具有相同的 n/i 的高度。因此，从每个堆栈中任意选择一个元素都是 $S(i,n)$ 的有效上升子序列，并获得等式 4.35 中的界。所以式 4.31 得以证明。

我们最终需要证明式 4.26。即找到一个p_2，使得$E_s(S_1,S_2) \le s_2 \Rightarrow Pr\left[\mathfrak{h}_{l,\pi}(S_1)=\mathfrak{h}_{l,\pi}(S_2)\right] \le p_2$。依照上文的分析，

$$\begin{aligned} &Pr\left[\mathfrak{h}_{l,\pi}(S_1)=\mathfrak{h}_{l,\pi}(S_2)\right] \\ &= Pr\left[\mathfrak{h}_{l,\pi}(S_1)=\mathfrak{h}_{l,\pi}(S_2) \middle| \left\{\mathfrak{h}^w_{l,\pi}(S_1)\right\}=\left\{\mathfrak{h}^w_{l,\pi}(S_2)\right\}\right] \\ &\quad \cdot Pr\left[\left\{\mathfrak{h}^w_{l,\pi}(S_1)\right\}=\left\{\mathfrak{h}^w_{l,\pi}(S_2)\right\}\right] \end{aligned} \tag{4.37}$$

根据式 4.29 和式 4.30，以及式 4.31，最后可以推出

$$Pr\left[\mathfrak{h}_{l,\pi}(S_1)=\mathfrak{h}_{l,\pi}(S_2)\right] \le \max_{\substack{l\le m \\ m\le n_k}} \frac{\left(\frac{m}{L}\right)^l \binom{L}{l}}{\binom{m}{l}} \frac{\binom{m}{l}}{\binom{n_k}{l}} \le \frac{\left(\frac{n_k}{L}\right)^l \binom{L}{l}}{\binom{n_k}{l}} \tag{4.38}$$

其中 L 表示$\mathcal{M}_k^w(S_1)$和$\mathcal{M}_k^w(S_2)$的最长公共子序列的长度。式 4.38 右侧的式子是关于 L 的增函数。当 $L < l$ 时，式子等价于 0；当 $L = n - k + 1$ 时，式子等价于 1。

给定$E_s(S_1,S_2) \le s_2$的条件，有$s_2 \ge E_s(S_1,S_2) \ge (L+k-1)/n$。$L \le ns_2 - k + 1$，将式 4.38 中的 L 用$ns_2 - k + 1$替换，得

$$Pr\left[\mathfrak{h}_{l,\pi}(S_1)=\mathfrak{h}_{l,\pi}(S_2)\right] \le \frac{\left(\frac{n_k}{ns_2-k+1}\right)^l \binom{ns_2-k+1}{l}}{\binom{n_k}{l}} \tag{4.39}$$

综上，当$p_2 = \frac{\left(\frac{n_k}{ns_2-k+1}\right)^l \binom{ns_2-k+1}{l}}{\binom{n_k}{l}}$时，$E_s(S_1,S_2) \le s_2 \Rightarrow Pr\left[\mathfrak{h}_{l,\pi}(S_1)=\mathfrak{h}_{l,\pi}(S_2)\right] \le p_2$。找到了式 4.26 中$s_2$和$p_2$的关系，因此 L-MinHash 是满足式 4.26 的。

所以，L-MinHash 是关于$\left(s_1, s_2, p^1_{n,k,l}(s_1), p^2_{n,k,l}(s_2)\right)$敏感的，可以被用于相似序列的聚类。

4.4 Order Min Hash 模型的复现

Order Min Hash 上的 LSH 要求必须对字符串的 k-mer 内容和相对顺序都敏感，但对于 k-mers 在字符串中的绝对位置相对不敏感。这引出了下面的定义。与 minHash 类似，Order Min Hash 是通过在 k-mers 上使用置换来随机选择的。为了保留有关的相对顺序的信息，Order Min Hash 并非只选取 1 个最小的哈希值，而是l个。取l个最小的 k-mers 的散列值，按照它们在初始序列中出现的顺序（而不是随机置换所定义的顺序）记录。

作为预处理步骤，需要把单个蛋白质序列划分为 k-mer。并根据 OMH 算法的特点，

最终“对应的”k-mer 包括 k-mer 和它当前在序列中的出现次数。

算法 4.1　k-mer 分割算法

输入：长度为 L 的蛋白质序列 S，k-mer 的长度 k
输出：输入序列 S 的全部 k-mer 加权集合 V

```
1: V ← [ ]                          //初始化集合
2: for i = 1 to L-k+1               //每条序列拥有的 k-mer 数量为 L-k+1
3:    s ← S[i..i+k-1]
4:    V ← V ∪ [s, H[s]]
5:    H[s] = H[s]+1
6: end for
7: return V
```

算法 4.1 中，重点在于对 k-mer 出现次数的统计，可以用 c++中内置的 unordered_map 关键词实现。序列 S 中 k-mer 的截取，也可以采用字符串 string 的内置函数 substr，基于滑动窗口算法思想进行一遍线性扫描。加权集合 V 的元素类型可以设置为 pair<string, int>，方便对元素进行打包。

接下来，就是 OMH 算法的具体实现部分。

算法 4.2　Order Min Hash 算法

输入：n 组蛋白质序列 S[]，k-mer 的长度 k，散列函数模数 p，全域散列模数 pp
输出：聚类后存在同源性可能的序列对集合 V

```
1: V ← [ ]
2: let A[1..L] be a new array       //A 存放散列函数的基数
3: for i=1 to L
4:    A[i] = RANDOM(1, p-1)         //生成 L 组散列函数
5: end for
6: for ii = 1 to L
7:    for i = 1 to n
8:       m ← |k_mer[i]|             //m 是第 i 个序列 k-mer 的数量
9:       for j = 1 to m
10:         x ← k_mer[i][j]
11:         val = 1                 //val 用来计算 k-mer 的散列值
12:         for jj = 1 to k
13:            val = ( val * A[ii] + x[jj] ) % p
14:         end for
15:         hash[j] = val % pp
```

```
16:     end for
17:     select hash[1..m] corresponding to the smallest l values
        //将最小的 l 个值放置在 hash 序列最左边，l个值内部保持原有序列排序
18:     for j = 1 to l
19:       v ← v ∪ hash[j]
20:     end for
21:     set[v] ← set[v] ∪ i      //对包含l个值和相对位置信息的列表做一个散列
22:   end for
23:   for i = 1 to |set[ ]|
24:     for j = 1 to |set[i]|
25:       for jj = j + 1 to |set[i]|
26:         V ← V ∪ ( set[i][j] , set[i][jj] )      //散列相同的两个序列可能同源
27:       end for
28:     end for
29:   end for
30: end for
31: return V
```

简化 OMH 算法的实现，需要对 c++的标准模板库 (Standard Template Library, STL) 具有一定掌握。尤其是将其中 l 个 k-mer 组的序列表映射到一个集合上，基础数据结构较难实现。以下给出本文的 OMH 函数具体实现过程。

代码清单 4.1　OMH 函数

```
vector< pair<string, int> > k_mer[MAXN];
unordered_map<string, int> mp, mpp;
map< vector<string>, int > hash_table, emptyh;
vector<int> hashset[MAXN], emptyhs;
set< pair<int, int> > preset;
void omh() {
    for(int ii = 0; ii < L; ii ++) {                    //L组hash函数
        hash_table = emptyh;    num = 1;
        for(int i = 0; i <= num; i ++)    hashset[i] = emptyhs;
        for(int i = 0; i < n; i ++) {
            int m = k_mer[i].size();
            for(int j = 0; j < m; j ++) {
                string x = k_mer[i][j].first;    long long h = 1;
                for(int jj = 0; jj < k; jj ++)    h = (h * hasha[ii] + x[jj]) % p;
```

```
                hashval[j].val = h % pp;    hashval[j].no = j;
            }
            //找出最小l个hash值对应的k-mer
            sort(hashval, hashval + m, cmp);    sort(hashval, hashval + l, cmp2);
            vector<string> v;
            for(int j = 0; j < l; j ++)    v.push_back(k_mer[i][hashval[j].no].first);
            if(hash_table[v] == 0)    hash_table[v] = num ++;
            hashset[hash_table[v]].push_back(i);
        }
        for(int j = 1; j < num; j ++) {
            int ans = hashset[j].size();
            for(int jj = 0; jj < ans; jj ++)
                for(int jjj = jj + 1; jjj < ans; jjj ++)
                    preset.insert ( make_pair(hashset[j][jj], hashset[j][jjj] ) ) ;
        }
    }
}
```

在代码清单 4.1 中，将 L 个存储 l 个 k-mer 序列的散列表巧妙地转换成了，先将 l 个 k-mer 序列映射成某个 hash 值，再将 l 个 k-mer 序列对应的原始序列放入 hash 值对应的集合中。最后将这些集合里的所有序列两两配对，存入目标集合中。

而因为既要选出 l 个最小哈希值，又要保持选出的 k-mer 值相对顺序不改变，我们采用两遍排序，但两组排序的数据范围并不相同。第一遍按照 hash 值对结构体 hashval[0..m-1]进行排序，第二遍按照记录的原始位置对结构体 hashval[0..l-1]进行排序，以符合算法要求。

最后，只需要将待检验集合中的序列对两两进行相似度计算，就能得到最后的结果。相似度计算采用动态规划的思想求解编辑距离，如算法 4.3 所示。

算法 4.3　动态规划求解编辑相似度

输入：长度分别为 l_1, l_2 的蛋白质序列 S_1 和 S_2

输出：输入序列 S_1 和 S_2 的编辑相似度 s

1: let dp[0..l_1][0..l_2] be a new array
2: **for** i = 0 **to** l_1-1
3: dp[i][0] = i
4: **end for**

```
for i = 0 to l_2-1
    dp[0][i] = i
end for
for i = 1 to l_1
    for j = 1 to l_2
        if S_1[i] = S_2[j]
            dp[i][j] = dp[i-1][j-1]
        else dp[i][j] = 1 + min{ dp[i-1][j], dp[i][j-1], dp[i-1][j-1] }
    end for
end for
s = 1 – dp[l_1][l_2] / max(l_1, l_2)
return s
```

4.5 本章小结

本章旨在详述本文提出的蛋白质同源性搜索的高效算法 L-MinHash，即如何从计算编辑相似度的近似解和快速过滤实现序列规模的降维两方面进行优化。首先从最长公共子序列 LCS 问题入手，引出 Jaccard 相似度，证明 Jaccard 相似度是编辑相似度的上界，并利用该上界实现低同源性序列的快速过滤。接着寻找问题规模降维的可行性，提出了基于 Locality-sensitive hashing 的 L-MinHash 算法，证明 L-MinHash 是局部敏感的，可以用于数据的聚类。最后，给出了经典模型 Order Min Hash 的复现细节。

5 算法效率与准确性在大数据集上的评估

5.1 数据集与任务介绍

FASTQ 和 FASTA 是存储生物序列的两种常用数据格式[35]。本文所研究的蛋白质同源性搜索问题中所处理的就是其中 FASTA 的数据。

FASTA 是一种在生物信息学领域广泛应用的、用单个大写英文字母来表示核苷酸序列或氨基酸（蛋白质）序列的文本文件格式。FASTA 格式主要是由 William R. Pearson 和 David J. Lipman 等人共同发明的[36]。所以其也可以被称之为 Pearson 格式。在 FASTA 中，一条序列由一行序列标记和后续的一行或多行序列信息组成。其中序列标记使用‘>’作为起始符号，接着是序列的名称，然后是序列的注释信息，换行之后是序列信息。为了保证 FASTA 文件的可读性，序列每行不超过 120 个字符，通常不超过 80 个字符。

图 5.1 为一个简单的 FASTA 文件示例。

```
>43FE01DC-D266-11EB-AD85-FBA22131C556
DGPSGIIWPOUXKXXBKWOGPMGTOPKLWZGKWPHMETKUKWHLWWJPUBATOWPBMKTTJKSTOHWWSIULPXHGGLOLSIGB
>43FE1B54-D266-11EB-ADF9-030D101C36E9
FBBHTFJSVMHJJJSTWTVXHUCMJUTTGHXWUBKXAJNJTBXJGUJTGAJTWXTKXXWSBHLSNAS
>43FE333C-D266-11EB-AE6C-7B98043C15DE
GPVFJCTGKJKWWTHGAMBXIIGTVVPGIJLSWMTZGMOJOHJGJGGTGBXISWKBHWTWWTGTHKZICTSIXMJBI
```

图 5.1 FASTA文件示例

需要注意的是，在 FASTA 格式中，没有对单条序列的长度区间进行限制。但与核苷酸序列碱基数跨度巨大不同，蛋白质序列最长仅含几千个氨基酸，因此在读取时还是较为方便的。一般来说，氨基酸序列包含了 23 个字母和 3 个特殊字符，见表 5.1。为了方便进行数据处理，我们只将其看作 26 个字母的排列组合。实际在具体应用中，需要考虑 B、J、X 和 Z 四个字母的特殊性。

表 5.1 FASTA支持的氨基酸编码

编码	含义	编码	含义	编码	含义	编码	含义
A	丙氨酸	H	组氨酸	O	吡咯赖氨酸	V	缬氨酸
B	D或N	I	异亮氨酸	P	脯氨酸	W	色氨酸
C	半胱氨酸	J	L或I	Q	谷氨酰胺	X	任何氨基酸
D	天冬氨酸	K	赖氨酸	R	精氨酸	Y	酪氨酸
E	谷氨酸	L	亮氨酸	S	丝氨酸	Z	E或Q
F	苯丙氨酸	M	蛋氨酸	T	苏氨酸	*	翻译停止
G	甘氨酸	N	天冬酰胺	U	硒代半胱氨酸	-	不定长间隙

本文测试数据集来源于 protein data bank[37][38]，其中包含 90 个大小为 1.33GB 的 FASTA 文件，每个文件包含约为 1E7 条蛋白质序列，每条蛋白质序列平均约含 200 个字符。而由于同源蛋白质的稀缺性，最终找到的序列对个数应该极少。因此，在 I/O 优化的时候，重点考虑读取速度的优化。代码清单 5.1 列举了本文做出的一些实际优化，包括 GCC 优化，ifstream 读入和 string_view 替换。

代码清单 5.1　I/O 优化

```
#pragma GCC diagnostic error "-std=c++11"
#pragma GCC optimize("-fdelete-null-pointer-checks,inline-functions-called-once,-funsafe-loop-optimizations,-fexpensive-optimizations,-foptimize-sibling-calls,-ftree-switch-conversion,-finline-small-functions,inline-small-functions,-frerun-cse-after-loop,-fhoist-adjacent-loads,-findirect-inlining,-freorder-functions,no-stack-protector,-fpartial-inlining,-fsched-interblock,-fcse-follow-jumps,-fcse-skip-blocks,-falign-functions,-fstrict-overflow,-fstrict-aliasing,-fschedule-insns2,-ftree-tail-merge,inline-functions,-fschedule-insns,-freorder-blocks,-fwhole-program,-funroll-loops,-fthread-jumps,-fcrossjumping,-fcaller-saves,-fdevirtualize,-falign-labels,-falign-loops,-falign-jumps,unroll-loops,-fsched-spec,-ffast-math,Ofast,inline,-fgcse,-fgcse-lm,-fipa-sra,-ftree-pre,-ftree-vrp,-fpeephole2",3)
#pragma GCC target("avx","sse2")

void readTxt(string file)
{
    ifstream fin;
    fin.open(file.data());
    assert(fin.is_open());
    while(getline(fin,s2[n]))
    {
        std::string_view str {s2[n]};
        s[n++]=str;
    }
}
```

我们需要对一组给定规模的蛋白质序列输入，输出其中具有同源性概率的序列对。在实验中，本文使用了两种传统的指标，效率和准确性来对用于测试这些数据集的不同方法进行评估。

效率采用程序从开始到结束的运行时间来评估，运行时间越短，算法的效率就越好。而准确性的评估则是与朴素算法的运行结果进行比较。朴素算法虽然耗时巨大，但可以

保证结果的 100%准确率，从而可以得到比较算法的 FP 率和 FN 率。

5.2 实验设计

本文实验包含两部分，第一部分是在小数据集下，将本文模型同朴素算法与 OMH 模型的搜索结果进行对比；第二部分是对本文模型涉及的参数进行调整，并对细节进行优化，在大数据集下测试模型的可靠性。

在第一部分中，由于朴素算法的时间复杂度过高，所以我们将数据范围缩减至 1E4，以期在可接受的运行时间内，对三种模型的效率与准确性进行对比。为了使结果更具有说服力，我们将 OMH 模型和本文模型的参数设置相同，并选择两组比较合理的参数分别比对，具体见表 5.2。

表 5.2　实验参数设置

参数	意义	值1	值2
k	k-mer的k大小	3	4
l	选取的最小hash个数	3	2
L	hash函数个数	300	500
p	hash函数模数	19260817	19260817
pp	全域散列的最后模数	100	300

测试数据从真实数据随机选取产生。考虑到 1E4 的数据范围较小，可能无法找到编辑相似度较大的序列对。为了方便结果的比对，我们往里加入了 5 条在真实序列基础上进行人工编辑操作的序列，并把同源性的阈值设置为 0.5。选用 Windows mingw 平台进行模拟。

在第二部分中，为了方便比较，我们仍采用第一部分的测试数据集，在 Windows mingw 平台完成对照实验。最后将调整好参数的模型部署在 Linux 中，应用于真实的 90 组大数据集中，得到结果。

5.3 实验结果与分析

5.3.1 三种模型的效率与准确性的对比

因为朴素模型无需对参数进行调整，因此我们先对其进行测试。在 1E4 的数据集上耗时 10588.1 秒，得到了 19 组编辑相似度大于 0.5 的蛋白质序列对。

接下来我们用 10 组独立实验，分别对两组参数下的 OMH 模型和本文模型进行测试。结果如表 5.3 所示。测试结果中第一列代表找到的解的组数，第二列代表程序耗时。

表 5.3　1E4数据集下的测试结果

实验编号	OMH+参数值1		改进模型+参数值1		OMH+参数值2		改进模型+参数值2	
1	11	89.220	10	56.524	16	138.463	17	94.194
2	12	86.554	9	56.283	16	139.230	16	93.654
3	14	88.449	8	57.003	17	138.129	17	99.766
4	13	89.723	10	56.847	17	138.895	17	94.846
5	16	86.876	10	56.415	17	138.444	17	94.249
6	12	88.171	10	56.151	17	138.847	17	94.599
7	14	89.991	13	56.569	17	138.145	17	93.514
8	9	88.776	11	56.167	17	138.650	17	93.517
9	11	88.987	10	56.500	17	139.750	17	93.848
10	14	86.255	11	56.190	17	139.350	17	93.668

在第一组参数值下，OMH 模型平均能找到 12.6 个解，占解集的 66.32%，总体方差 3.64，平均用时 88.30 秒；改进模型平均能找到 10.2 个解，占解集的 53.68%，总体方差 1.56，平均用时 56.46 秒。也就是说，OMH 模型在平均准确性上优于改进模型，但存在求解不稳定的情况，但改进模型相较于 OMH 模型提速约 36.06%。

第二组参数值中，OMH 模型平均能找到 16.8 个解，占解集的 88.42%，总体方差 0.16，平均用时 138.79 秒；改进模型平均能找到 16.9 个解，占解集的 88.95%，总体方差 0.09，平均用时 94.59 秒。改进模型相较于 OMH 模型提速约 31.85%，并且两个模型都达到了很好的求解效果，不仅准确性较高，而且算法稳定性也比较好。

从算法效率来说，OMH 模型和改进模型相较于朴素算法，提升都是非常显著的。改进模型在 OMH 模型的基础上对时间复杂度进行了进一步的优化，缩小了算法时间复杂度的常数，对于大数据的改进效果会更加明显。在一些适当参数下，两者都能达到较高的准确率，但 OMH 模型的平均准确率会更好一些。

5.3.2 参数调整对 L-MinHash 模型的影响

实验需要调整的参数如表 5.2 所示。我们将 k=4，l=2，L=300，p=19260817，pp=100 的一组参数设置为对照组，再针对多个具体参数分别设置不同的实验组。具体结果可见表 5.4。

表 5.4　不同参数的对照实验结果

k	l	L	p	pp	解的组数	耗时
4	2	300	19260817	100	13	53.258
					15	54.330
3	2	300	19260817	100	16	419.586
					17	424.830
5	2	300	19260817	100	12	53.036
					12	53.369
6	2	300	19260817	100	9	54.727
					8	55.631
4	1	300	19260817	100	17	1373.520
					/	/
4	3	300	19260817	100	7	50.765
					6	52.973
4	2	500	19260817	100	16	91.857
					15	92.265
4	2	700	19260817	100	16	142.035
					17	149.776
4	2	300	12289	100	14	59.917
					14	59.133
4	2	300	1610612741	100	16	59.697
					16	58.088
4	2	300	19260817	300	17	59.440
					16	60.358
4	2	300	19260817	500	17	61.777
					17	61.133
4	2	300	19260817	700	15	59.799
					17	62.099

显而易见的，这五个参数对结果的影响都是显著的。具体表现在：

k 的减小意味着初筛更加宽松，能使更多序列对进入候选区域，因此会带来效率的降低和准确度的上升，增大则刚好相反。k=4 是一个比较理想的参数值，当 k 减小为 3，运行时间就扩大为原来的 8 倍左右，时间成本太大。

l=1 时，改进模型就彻底变成 MinHash，运行时间约为对照组的 26 倍，效果不理想。而l=3 时，准确率急速下降到 50%以下，在天然数据中很难保证能搜寻出解。因此l=2 也是一个相对较好的参数。

L 表示哈希函数的组数。解的组数一定是一个关于 L 的增函数，并且在 L 达到一个值后，解的组数近似于一条平滑直线。具体数值的选取应该在运行时间允许的情况下，尽量选择大一些的值。

p 的选择对结果是有一定影响的。但由于 p 的不连续性，只能说选取一个靠近2^{31}的

大素数效果都还是不错的，并且带来的效率上的影响也比较小。

pp 在一定范围内的增加可以提升准确率，但超过后又会因为哈希冲突的减少而导致准确率的下降。pp=500 是一个比较理想的参数值。

5.4 本章小结

本章属于实验环节，首先对数据集与将要进行的实验任务进行介绍，接着详述了实验设计中各类参数，方便其他研究者复现。再将实验结果与朴素算法的结果进行比对分析，检测算法的可信程度和实现效率。并将本篇提出的模型算法进行多次独立实验。其结果与该领域最新模型结果进行对比，取得了较高的性能。最终对模型的各项参数进行优化，并进行多组对照实验，找到最优的参数。实验证明，当 k=4，l=2，L=300，p=19260817，pp=500 时，模型能取得比较理想的效果。

结　　论

针对蛋白质同源性搜索问题，即具有编辑相似性的序列对搜索问题，本文在生物大数据聚类分析的基础上，提出了基于 MinHash 算法和 LSH 算法的高效近似 Jaccard 相似度估计算法 L-MinHash。区别于基础的 MinHash 算法，L-MinHash 在 Order Min Hash 模型的启发下，用最小 l 个哈希值代替最小哈希值，提高了程序的运行效率。并且，通过对 Order Min Hash 模型的改进，进一步平衡效率与准确率。同时，各参数的调整是灵活的，可以基于具体数据集的需要进行修改。

在实验结果中，我们统一以 C++语言复现了朴素算法模型和 OMH 模型，取得了与参考文献相近的结果，表明了我们实验的设计与模拟环境具有相当高的可信度。最终，以相同的数据处理，运行环境，I/O 框架和训练流程，对本文提出的蛋白质同源性搜索问题进行了多次实验。实验结果表明，我们的算法模型在 1E4 的小数据集上取得了较好的性能，效率提升迅速。

在生物数据规模快速增长的背景下，当前对蛋白质同源性搜索问题主要聚焦在聚类分析以缩小需要分析的集合大小，即主要对序列进行哈希分类。而本文提出的基于 MinHash 算法和 LSH 算法的高效近似 Jaccard 相似度估计算法在这个思路的基础上，做了进一步的分析论证，验证了这个方向的理论正确性，取得了一定程度的性能提升，或许值得更加深入地研究。