

毕业论文

**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：

蛋白质同源性搜索的高效算法

摘 要

检索同源蛋白质序列是蛋白质生物信息学的基本问题，也是信息量最大的步骤之一。蛋白质的同源性常常通过序列的相似性来判定。同源蛋白质序列检索抽象为数学问题就是，在超大数据量的字符串中，检索出编辑相似度高于某个阈值的字符串对。现阶段该问题的研究难点在于序列数目过于庞大，序列间进行两两比较的朴素算法不可行。有力的数学模型和高效的算法设计成为蛋白质同源性搜索中的关键。本文基于聚类方法提供的降维思路，进行了相关的算法开发和计算分析的工作。

Jaccard相似度作为编辑相似度的上界，可以用来实现低编辑相似度序列的快速过滤。虽然过滤后仍存在假阳性的情况，但是保证不会出现假阴性的情况。MinHash方法是一种快速估计两个集合的Jaccard相似度而无需直接计算集合交集和并集的技术。利用MinHash方法可以优化序列对之间的相似度计算，但仍未解决序列间需要进行两两比对的问题。近年来，该领域主要聚焦采用聚类的方法进行序列规模的降维。局部敏感哈希就是一种可以以较高概率将相似的输入项散列到相同的类中的算法技术。

本文提出一种基于MinHash方法和局部敏感哈希方法的高效近似Jaccard相似度估计算法。该方法是对Order Min Hash算法的改进，吸收了OMH模型中关于 个最小哈希值的思想，能很大程度优化时间复杂度。

在实验部分，我们统一以C++语言复现了朴素算法模型和OMH模型，与本文的改进模型进行比较和分析。 最后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。本文所提出的算法模型的性能总体优于OMH模型，并取得了比当前大多数算法更好的性能。

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

**An Efficient Algorithm for Protein Homology Search**

**Abstract**

The search for homologous protein sequences is a fundamental problem in protein bioinformatics and one of the most informative steps. Homology of proteins is often judged by sequence similarity. The abstraction of homologous protein sequence searching as a mathematical problem is to find pairs of strings with edit similarity greater than a certain threshold in a very large amount of data. The difficulty of this problem at this stage of research is that the number of sequences is too large and the plain algorithm for multiple comparison between sequences is not feasible. Powerful mathematical models and efficient algorithms become the key to protein homology search. In this paper, based on the idea of dimensionality reduction supplied by clustering methods, we develop some novel algorithms and per­form data analysis.

Jaccard similarity is used as an upper bound for edit similarity which can be applied to achieve fast filtering of low edit similarity sequences. Although there are still false positives after filtering, it is guaranteed that there will be no false negatives. MinHash method is a technique to quickly estimate the Jaccard similarity of two sets without directly computing the set intersection and union. Using MinHash method can optimize the similarity calculation between sequence pairs, but it still does not solve the need for multiple comparison between sequences. In recent years, the field has mainly focused on sequence-scale dimensionality reduction with clustering methods. Locally sensitive hashing is an algorithmic technique that can hash similar input items into the same class with high probability.

In this paper, we propose an efficient approximate Jaccard similarity estimation algorithm based on MinHash method and locally sensitive hashing method. This method is an improvement of Order Min Hash algorithm, which absorbs the idea of OMH model about the smallest hash values and can optimize the time complexity to a great extent.

In the experimental part, we reproduce the plain algorithm model and OMH model in C++ uniformly , and compare and analyze them with the improved model in this paper. Finally, we optimize the parameters of the model and conduct several groups of controlled experiments to find the optimal parameters. The proposed algorithmic model in this paper outperforms the OMH model in general and achieves better performance than most current algorithms.

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

目 录

[1 绪 论 3](#_Toc103709532)

[1.1 引言 3](#_Toc103709533)

[1.2 研究背景 3](#_Toc103709534)

[1.2.1 国内外研究现状 4](#_Toc103709535)

[1.2.2 Needleman-Wunsch算法及其优化 4](#_Toc103709536)

[1.2.3 Smith-Waterman算法及其优化 5](#_Toc103709537)

[1.3 研究意义与内容 5](#_Toc103709538)

[1.4 论文组织结构 6](#_Toc103709539)

[2 蛋白质同源性搜索问题综述 9](#_Toc103709540)

[2.1 基本术语介绍 9](#_Toc103709541)

[2.2 蛋白质同源性搜索问题定义 11](#_Toc103709542)

[2.3 总体思路 13](#_Toc103709543)

[2.4 本章小结 14](#_Toc103709544)

[3 编辑距离下的同源序列高效检索算法 15](#_Toc103709545)

[3.1 编辑相似度上界的确定 15](#_Toc103709546)

[3.2.1 最长公共子序列问题 15](#_Toc103709547)

[3.2.2 Jaccard相似度 16](#_Toc103709548)

[3.2 快速过滤算法的时间复杂度分析和优化 17](#_Toc103709549)

[3.3 Order Min Hash模型介绍及复现 17](#_Toc103709550)

[3.3.1 术语介绍 17](#_Toc103709551)

[3.3.2 Order Min Hash模型 32](#_Toc103709552)

[3.3.3 模型复现 38](#_Toc103709553)

[3.4 基于MinHash算法和LSH算法的高效近似Jaccard相似度估计算法 41](#_Toc103709554)

[3.5 本章小结 41](#_Toc103709555)

[4 算法效率与准确性在大数据集上的评估 43](#_Toc103709556)

[4.1 数据集与任务介绍 43](#_Toc103709557)

[4.2 实验设计 44](#_Toc103709558)

[4.3 实验结果与分析 45](#_Toc103709559)

[4.3.1 三种模型的效率与准确性的对比 45](#_Toc103709560)

[4.3.2 参数调整对改进模型的影响 46](#_Toc103709561)

[4.4 本章小结 47](#_Toc103709562)

[结 论 49](#_Toc103709563)

[致 谢 50](#_Toc103709564)

[参考文献 52](#_Toc103709565)

[附 录 54](#_Toc103709566)

[附录A 54](#_Toc103709567)

# 1 绪 论

## 1.1 引言

如果两个或多个结构由一个共同的祖先演化而来，则称其同源 (Homology) 。在生物信息学中，同源主要是指序列上的同源，用来说明两个或多个蛋白质或DNA序列具有相同的祖先。

## 1.2 研究背景

检索同源蛋白质序列是蛋白质生物信息学的基本问题，通常是任何基于序列的蛋白质研究的第一步，也是信息量最大的步骤之一。

例如，由DeepMind团队开发的人工智能软件系统AlphaFold能够根据一个蛋白质的氨基酸序列来确定它的3D结构，对于准确认识蛋白质功能，破解蛋白质折叠难题有着至关重要的意义。AlphaFold会先在已有的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，构成神经网络的输入[1]。AlphaFold的预测精度依赖于同源蛋白的数量和相似性，以及同源蛋白是否已经有实验结构。

蛋白质的同源性常常通过序列的相似性 (Sequence similarity) 来判定，相似性一般用检测序列和目标序列之间序列一致性 (Percent identity) 来表示。当两个序列的相似性超过偶然预期时，我们推断同源性。对序列过度相似的最简单解释是，这两个序列不是独立出现的，而是来自一个共同的祖先。共同祖先解释了过度的相似性（其他的解释要求相似的结构独立出现）。因此，过度的相似性意味着共同的祖先[2]。

然而，同源性和相似性的概念不能等价。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比对并不显著。但当相似性搜索发现具有统计意义的匹配时，我们可以推断这两个序列是同源的。

所以同源检索的研究目标是在海量蛋白质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就是：在超大数据量的字符串中，检索出相似度高于某个阈值的字符串对，并且一般情况下相似度用Levenshtein距离（编辑距离）度量。

由于测序技术的飞速发展，被测序蛋白质的数量在迅速增加。现阶段该问题的研究难点在于序列数目过于庞大（超过），序列间两两比较的朴素算法的时间复杂度是不可行的，更何况两个序列间计算Levenshtein距离还需要更多的计算。因此，在合理的时间内找到同源蛋白质序列是本文要解决的关键问题，也是难点所在。

1.2.1 国内外研究现状

序列比对是生物信息学最重要的方法之一。通过序列比对可以阐明序列之间的相似性程度，从而确定目标之间的亲缘关系和结构关系[3]。序列比对也可分为双序列比对和多序列比对。双序列比对技术较为成熟。主流的基础序列比对算法有 Smith-Waterman算法[4]和 Needleman-Wunsch 算法[5]等。通过启发式提速，被广泛使用的双序列比对软件有 FASTA和 BLAST 软件。多序列比对是双序列比对的扩展，更为复杂，目前尚无比较有效的算法，其中较流行的软件是 CLUSTALW。

尽管有许多算法和工程上的改进，但两个序列之间的序列比对或编辑距离的计算仍是输入序列长度的平方级别，这在实践中仍然耗时巨大，甚至不可接受。基于此，近来部分算法依靠哈希来降维，以便更快速地检测具有高对齐概率的序列。MinHash和Locality-sensitive hashing (LSH) 方法被广泛运用在Mash[6]，Mashmap[7]和MHap[8]等序列比对工具中。

1.2.2 Needleman-Wunsch算法及其优化

Needleman-Wunsch (NW) 算法是将动态规划应用于生物序列比较的最早期的几个实例之一，也被称为优化匹配算法和整体序列比较法。NW算法能找到两个序列的最佳比对，是一种全局比对技术，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的对应情况分为Match，Mismatch和Indel（Insertion或Deletion）三类，并为之分配相应的得分。原始的动态转移方程为

(1.1)

表示第一个序列的第i位，表示第二个序列的第i位，表示从和更新到和的匹配得分情况。对应和 Match或Mismatch的得分，对应Indel（即在或相应位置插空匹配）的得分。

该算法对于计算两个长度分别为m和n的序列的时间复杂度为，运用Method of Four Russians算法进行优化可将复杂度降为[9][10]。

Rashed等人提出了基于机器学习的NW算法优化，将包括多层感知机 (MLP) 、支持向量机 (SVM) 、决策树、随机梯度下降法 (SGD) 和XGBoost分类器在内的15种机器学习方法运用在原始数据的分类上，快速并行实现NW算法[11]。

1.2.3 Smith-Waterman算法及其优化

Smith-Waterman (SW) 算法是NW算法的变体。该算法的目的不是进行全序列的比对，而是找出两个序列中具有高相似度的片段。相较于NW算法而言，SW算法更新阶段不存在负数得分情况，这使得局部序列比对成为可能。任何位置的分数低于0，意味着此前的序列不具备相似性。将之设置为0，达到了“重设”的效果，使得从此位置开始的比对不受之前位置的影响。动态转移方程可以表示为

(1.2)

(1.3)

SW算法的时间复杂度与NW算法相类似，也是平方级别。FASTA运用了SW算法，并将之并行化，使比较速度提升了10-20倍[12]。FASTA算法通过搜索和匹配k-tuples（长度为k的子序列）来近似最佳匹配。对于蛋白质序列来说，这些k-tuples的长度往往为2。虽然FASTA算法的最坏时间复杂度仍为，但平均时间复杂度可以缩小到[13]。因此，FASTA算法的复杂性取决于k-tuples的大小：k-tuples越大，算法的速度越快。尽管FASTA算法比之前任何的算法都要快，但它无法保证找到两个蛋白质序列之间的最佳比对。因为它使用k-tuples来加速，可能会错过较小的相似区域，从而可能使两个蛋白质错位配对。此外，限制动态规划搜索区域的范围可能会产生次优对齐。很明显，FASTA算法在速度和精度之间需要权衡。

此外，通过FPGA[14]、GPU[15]、SIMD[16]和Cell Broadband Engine（Cell宽带引擎）[17]的加速，都很好地提升了SW算法的性能，但在最坏情况下仍是平方级别的复杂度。

1.3 研究意义与内容

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。

生物信息学数据的一个明显特点是数据量非常大。目前，核苷酸序列数据库包含个碱基，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，平均长度约为400个残基。不仅单个数据库很大，而且它们的规模正在以非常高的速度增长[18]。

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。

因此，蛋白质同源性搜索，即序列相似性度量，是生物信息学中许多算法的核心。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。

1.4 论文组织结构

第1章绪论，阐明了本文的研究背景，概括了该领域国内外主要的研究成果，阐明了本文做出的主要贡献，最后对于本文的工作内容跟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。首先对同源性相关的生物学基础概念进行了介绍，并使用形式化语言对同源性搜索问题进行了定义，用编辑 (Levenshtein) 相似度来狭义化同源性概念，便于之后的模型建模与分析。最后，从如何度量相似性的角度，概括表述了解决蛋白质同源性搜索问题的总体思路，即从快速过滤实现序列规模的降维和计算Levenshtein距离的近似解两方面进行优化。

第3章旨在详述本文提出的蛋白质同源性搜索的高效算法。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。接着从算法时间复杂度角度进行论证，寻找问题规模降维的可行性。之后，引入Order Min Hash模型，先对文本相似度算法和散列函数交叉领域的术语做了详细介绍，分析MinHash算法和Locality-sensitive hashing算法的原理，并具体计算了概率和期望。对于Order Min Hash模型的理念，给出了基于C++的实现方法。最后，基于MinHash算法和Locality-sensitive hashing算法，对Order Min Hash模型进行修改，给出Jaccard相似度的高效近似估计算法。

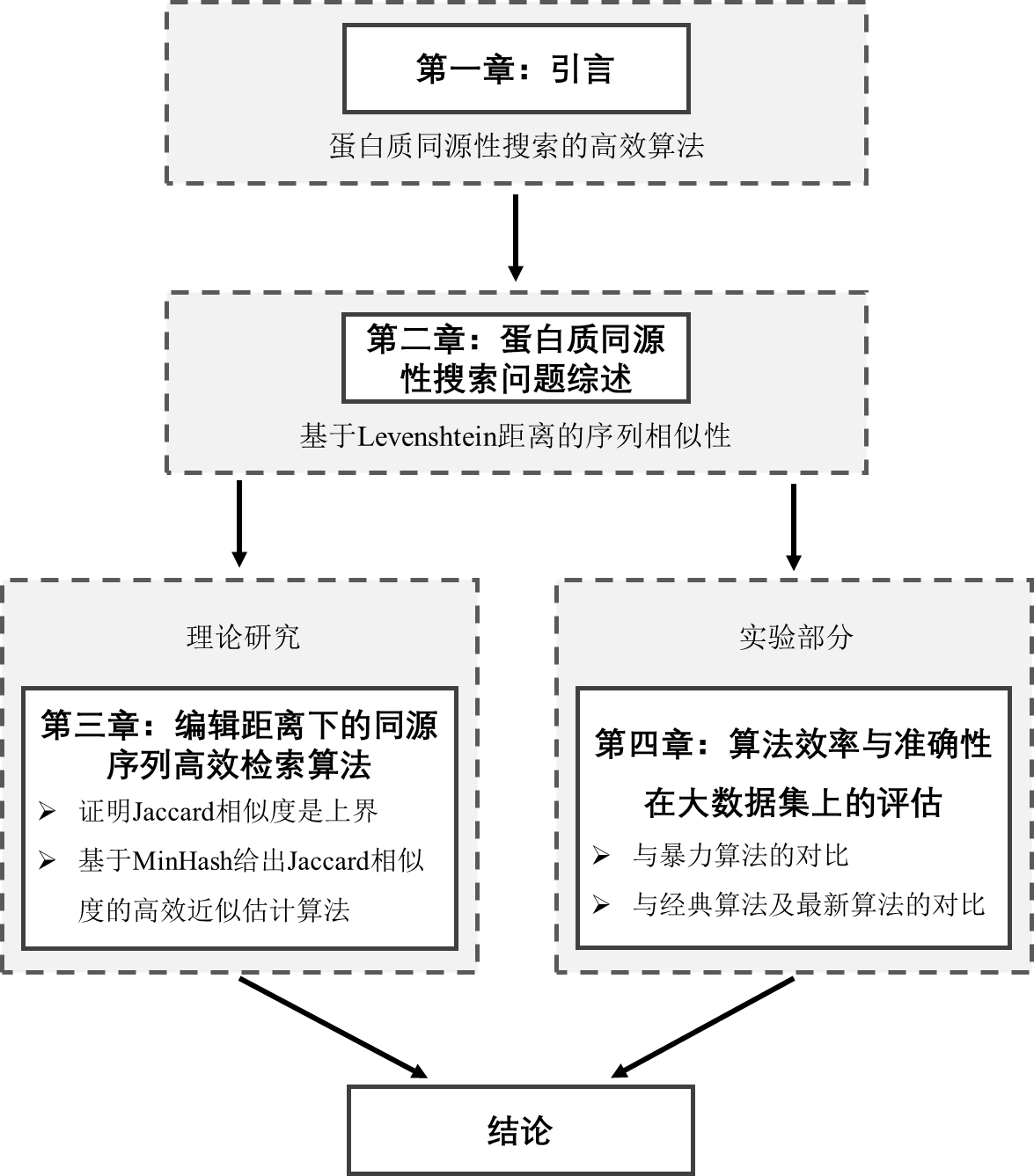


图 1.1 论文各章节的关系图

第4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，包括实验环境、数据集预处理过程、模型参数设置与具体的训练流程，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，=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。

# 2 蛋白质同源性搜索问题综述

同源性搜索的核心是要找到相似性较高的序列对，其中主要包含两个方面的难题需要解决。一方面是如何度量出这种相似性，另一方面是如何降低需要进行相似性分析的数据规模。

2.1 基本术语介绍

（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**错误!未找到引用源。**。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。

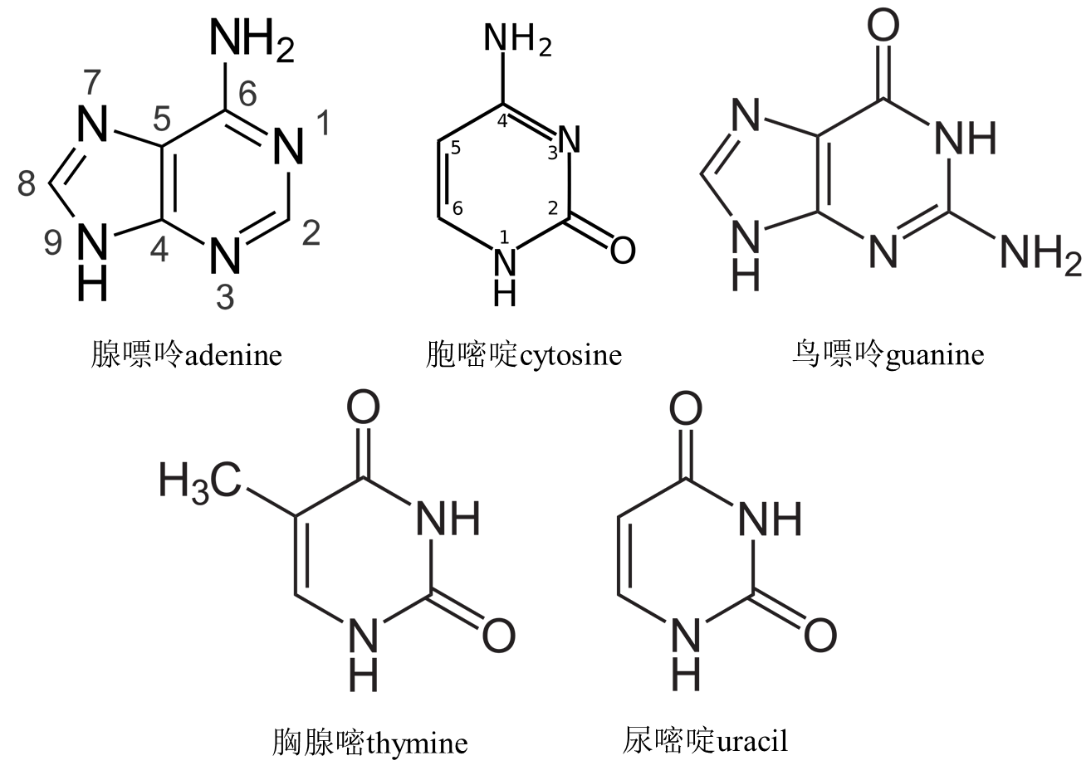


图 2.1 碱基的分子结构

（2）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。

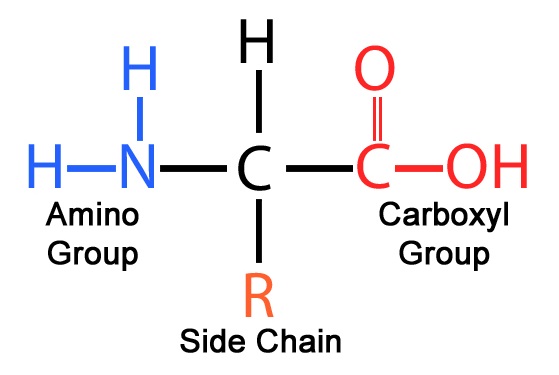


图 2.2 氨基酸的基本结构

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。由于这 20 种氨基酸的结构和化学性质不同，形成了功能各异的蛋白质分子。

（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。

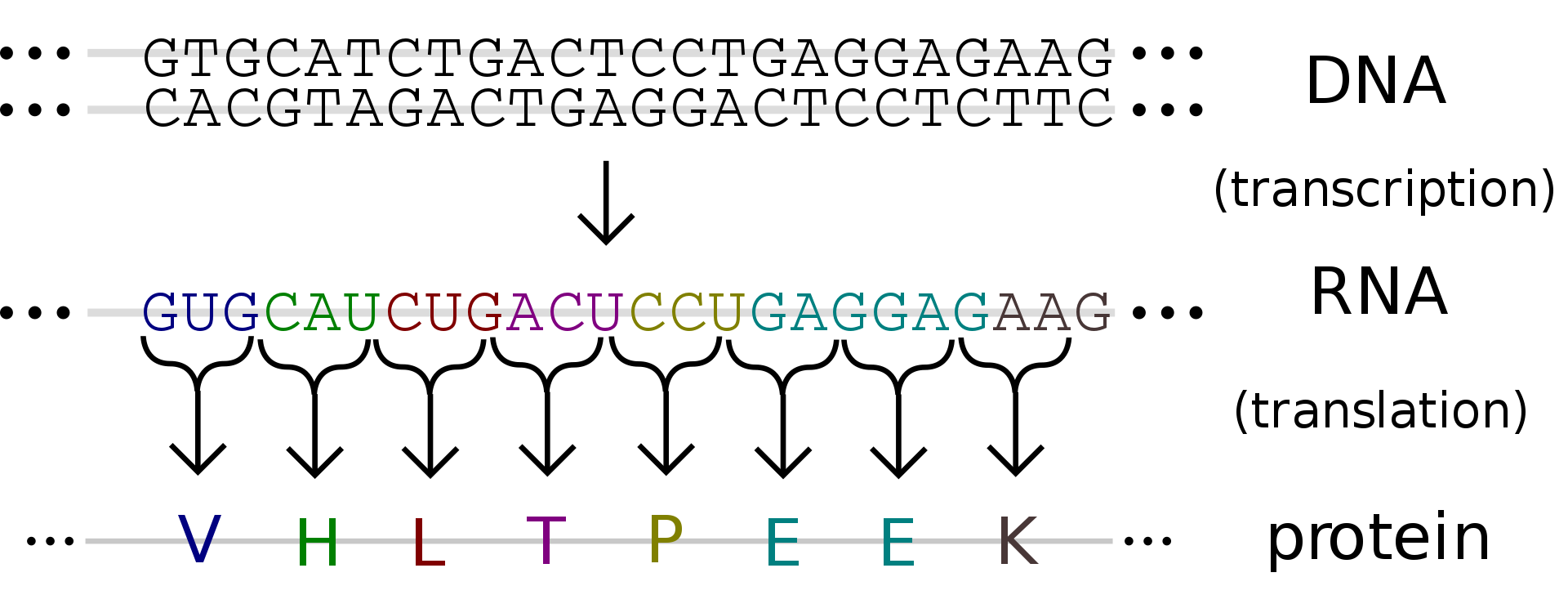


图 2.3 碱基与蛋白质序列关系

（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。

2.2 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的话，匹配数非常少，这两条序列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插入一位空格，就会发现原来这两条序列有更多的相似之处。

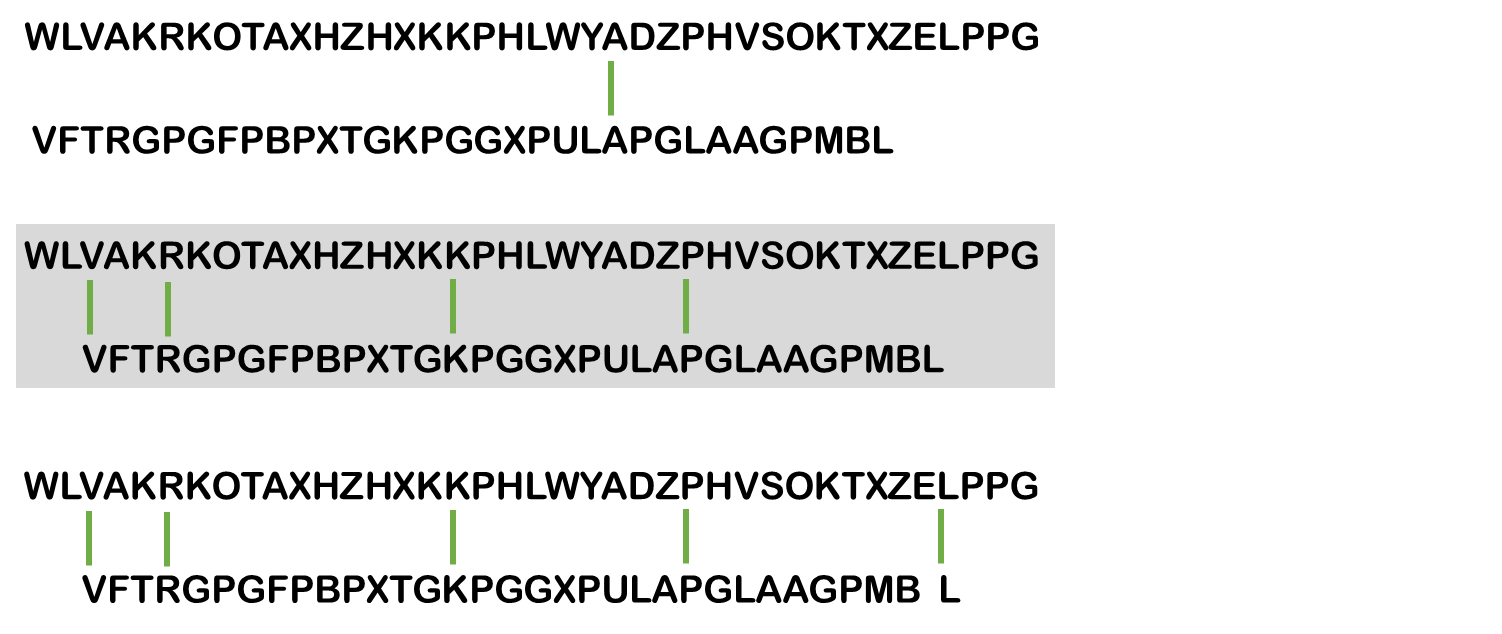


图 2.4 蛋白质序列比对

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。有两种方法可用于量化两条序列的相似程度：一为相似度，它是两条序列的函数，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，距离越大，则两条序列的相似度就越小。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，表示在全集内两个对象之间的距离。必须符合一些基本条件，即：

（1）非负性：；

（2）对称性：；

（3）同一性：。当时，意味着x=y；

（4）三角不等式：。

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，那么就是相异度。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) 。对于两条长度相等的序列，海明距离等于对应位置字符不同的个数。例如，图2.5表示了3组序列海明距离的计算结果。

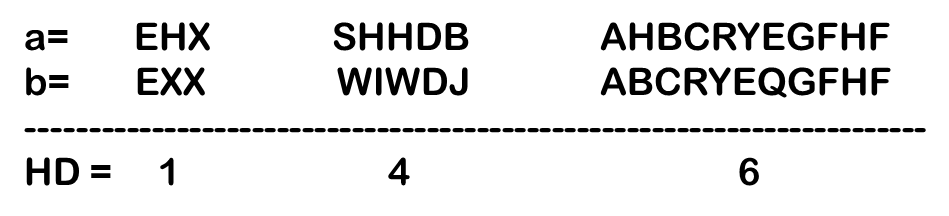


图 2.5 海明距离的计算

使用距离来计算不够灵活。因为序列可能具有不同的长度，两条序列中各位置上的字符并不一定是真正的对应关系。例如，在实际情况中，可能会发生像删除或插入一个氨基酸这样的错误。虽然两条序列的其他部分相同，但由于位置的移动导致海明距离的失真。就图2.5中最右边的情况，海明距离为6。简单地从海明距离来看，两条序列差别很大（整个序列的长度只有11），但是，如果从a中删除H，从b中删除Q，则两条序列都成为ABCRYEGFHF，这说明两条序列仅仅相差两个字符。实际上，在许多情况下，直接运用海明距离来衡量两条序列的相似程度是不合理的。

为了解决字符插入和删除问题，引入字符“编辑操作” (Edit Operation) 的概念。通过编辑操作将一个序列转化为一个新序列。用一个新的字符“-”代表空位 (Space) ，并定义下述字符编辑操作：

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空白字符；

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，分别用和表示序列的长度。编辑距离 (Levenshtein distance) 定义为

(2.1)

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) 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2.2。由此可以得到本文讨论的蛋白质同源性搜索问题的核心，编辑相似度。

(2.2)

(2.3)

2.3 总体思路

若采用朴素算法进行两两序列比对，时间复杂度，表示字符序列的数量。

而传统计算Levenshtein距离的算法时间复杂度为，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，缩小中的m和n，使时间复杂度降为 () 。但由于蛋白质序列的特殊性，使得这种优化发挥空间的余地很小。

因此总的序列比对时间复杂度近似为，是绝对无法接受的。整个改进算法最主要的提升仍然在对的降维上。因此我们需要找寻一种数据清洗方式，对同源概率非常低的序列对进行快速过滤，使的规模迅速缩小。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。

而对于相似度的计算过程进行降维处理，既然无法通过算法层面的优化来计算准确的Levenshtein距离，我们可以尝试寻找新的相似度计算方法去逼近Levenshtein距离的上下界，以可以接受的精确度误差来近似编辑相似度。

2.4 本章小结

本章首先对同源性相关的生物学基础概念进行了介绍，并使用形式化语言对同源性搜索问题进行了定义，用编辑相似度来狭义化同源性概念，便于之后的模型建模与分析。最后，从如何度量相似性的角度，概括表述了解决蛋白质同源性搜索问题的总体思路，即从快速过滤实现序列规模的降维和计算Levenshtein距离的近似解两方面进行优化。

# 3 编辑距离下的同源序列高效检索算法

因为无法避免同源序列两两比对的步骤，需要实现时间复杂度的降维只能从缩小需要比对的序列的规模入手，找寻一种低相似度序列的快速过滤算法。

3.1 编辑相似度上界的确定

### 3.2.1 最长公共子序列问题

给定一个序列，另一个序列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，对所有，满足。例如，是的子序列，对应的下标序列为。

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) 。例如，如果，，那么序列就是X和Y的公共子序列，但并非最长公共子序列 (longest common subsequence, LCS) 。因为也是X和Y的公共子序列，但其长度为4，大于的长度。是X和Y的最长公共子序列，也是，因为X和Y不存在长度大于等于5的公共子序列。[20]

基于LCS也可以定义两个序列的相似度。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

(3.1)

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

(3.2)

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，仍需要的时间复杂度。因此我们还需要探索新的上界表示方法。

### 3.2.2 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[21]。

(3.3)

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：当集合元素对应的全集，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3.4。

(3.4)

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。

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，而不要求位置信息，重合率会更高。因此。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，则。。所以有

(3.5)

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，存在两个常数和 ) ，只有当Jaccard相似度，才有编辑相似度的情况存在。我们考虑它的逆否命题：若的情况不存在，即任意，则由式3.5可推导得。因为，所以。从而原命题成立。

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

3.2 快速过滤算法的时间复杂度分析和优化

Jaccard相似度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。的朴素算法，可以采用的过滤算法来大大缩减的值域，最后将时间复杂度降为。

然而，快速过滤算法的时间复杂度仍是不可接受的。来源于序列的两两比对，来源于使用hash来计算集合之间的交并集。因此，找到一种聚类算法来对进行降维，并且运用兼顾效率与准确性的hash算法，是提升快速过滤算法速度的关键。

## 3.3 Order Min Hash模型介绍及复现

### 3.3.1 术语介绍

（1）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。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。

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，就可以得出序列的相似度。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。

（2）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。



图 3.1 直接寻址表[20]

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3.1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：

(a) 如果全域U很大，空间复杂度将不可接受，超出标准计算机内存；

(b) 如果实际需要存储的关键字集合，分配给U的大部分空间实际都是NIL的，造成空间浪费。

当时，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，同时查找一个元素的效率仍能得到保持，只需要的时间。

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：

(3.6)

图3.2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。



图 3.2 散列表[20]

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，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。

（3）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变为了线性。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。

设是一组有限散列函数，它将给定的关键词全域U映射到中。一个函数组如果满足：对每一对不同的关键字，的散列函数的个数至多为，就被称为全域的 (universal)。要而言之，如果从中随机选择一个散列函数，当关键字时，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()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。设表示集合，表示集合。

对于任何和任何，定义散列函数。利用一次线性变换，再进行模p和模m的归约，有

(3.7)

例如，当p=17，m=6时，。

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。当时，一定存在两个人生日相同；当时，n个人的生日都不相同的概率。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，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和M存在最大公因数g。可以转换成，即。其中q是商，r是余数。看起来r的取值范围仍是。但仍可以继续转换为。因为n和mq都是整数，所以r一定能整除g。而的取值范围，因此r的实际取值范围是，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

(3.8)

每一个散列函数都将映射到。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，所以中包含p(p-1)个散列函数。

以下给出散列函数族是全域的证明。

考虑中的两个不同关键字k和l，即。对于某一个给定的散列函数，设

(3.9)

(3.10)

因为，而根据已知，，。p为质数，与合数互质，因此，得。在模p层次上，计算任何，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：

(3.11)

(3.12)

其中是在模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，在数对与数对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，如果从中均匀地随机选择，则结果数对在模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。对于某个给定的r值，s的可能取值空间为p-1，其中满足且的s值的数目至多为：

(3.13)

因此，s与r发生冲突的概率至多为。所以，对于任何不同的数对，有

(3.14)

得以证明是全域的。

（4）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，而无需直接计算集合之间交集和并集的技术。

给定一个全集和它的子集，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 ， 表示一个将上的关键字映射到上的散列函数。 被定义为如果 ，且与的所有元素中比较 具有最小散列值，这也被称作的一个MinHash。

假设集合被散列成，那么。

将应用到集合A和B上，假设不存在哈希冲突，当且仅当在的所有元素中，具有最小哈希值的元素在中。这种情况发生的概率恰好等于Jaccard相似度。因此，

(3.15)

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：

(3.16)

如果state为真，；否则。当的时候，。

而由（3）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。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，满足的函数个数为y。可以视为的无偏估计量。根据切诺夫界 (Chernoff bound) ，误差期望。因此如果k=400时，预期误差仅为0.05。

例如，存在，，全集。集合可以表示为表3.1的形式。

表 3.1 集合表示

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 行号 | 元素 |  |  | 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。和的每一行元素可以按取值是否相等，分为以下三类：

(a) X类，两者值均为1。如表3.1中第5行，两个集合都具有元素e；

(b) Y类，两者取值不相同，一个为0，另一个为1；如表3.1中第1行，表示中具有元素a，而中没有；

(c) Z类，两者值均为0。如表3.1中第2行，表示和中都没有元素b。

设哈希函数，其中i代表行号。作用于集合和，可以得到表3.2的结果表示。

表 3.2 下的集合表示

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 行号 | 元素 |  |  | 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，=e，=e，即和的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均匀分布的情况下，由于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。所以，第一个出现X类行的行出现X类的概率也为。

继续假设存在哈希函数，那么可以得到表3.3。

表 3.3 下的集合表示

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 行号 | 元素 |  |  | 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 () ，因此我们需要为每个集合计算h次Minhash值，用这h个Minhash值组成一个摘要，来表示当前集合，如表3.4所示。

表 3.4 和下的MinHash摘要

|  |  |  |
| --- | --- | --- |
| 哈希函数 |  |  |
|  | e | e |
|  | 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3.4中，M=1，因为在哈希函数散列下，和的MinHash值相同，而在下则不同。那么可以发现，

(3.17)

M符合次数为h，概率为的二项分布，而期望。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。

（5）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的两个碱基序列和，，如果和之间相差不超过r个字符，就称和相似。给定一个序列G () ，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。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) 分别散列到哈希表的集合中。对于每一个长度为d的询问串q，查找中的，中的，…，中的，令并集。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，如果对于一个与询问串q相似的串g，存在p的概率返回g。

(3.18)

(3.19)

由于存在哈希函数族，因此g被找到的概率为

(3.20)

假设p=0.1，我们能得到如下关系表3.5。

表 3.5 p=0.1时，与函数结果的关系

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

分析可知，随着的小幅度增大，g被找到的概率会无限趋近1。

当的时候，算法的空间复杂度显然为。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；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。

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，从均匀随机选择 个置换。定义LSH函数，其中，那么

(3.21)

例如，当k=2，，时，。

每一位i在集合中都有d种选择，因此总共有个不同的置换函数h。

当，随机选择一个LSH函数，对于每个符合条件的子串的起始位置v，将 放入散列表中。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中的元素按照值的大小进行排序。因此这部分总时间复杂度近似 。在询问阶段，对于，计算出，在每个中二分查找与相同的值对应的位置信息v，最后只需要比较序列q和的相似度。时间复杂度为 。

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。 如果序列和相似（即序列间差异的位数小于r），有

(3.22)

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）发生的概率为

(3.23)

对于阈值，有。因此敏感度就可以定义为，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，就需要调整为0.05。

对于一个固定的值，我们需要找到一个k满足

(3.24)

设每个散列函数的假阳性概率为。LSH函数会返回一组包括真解和假阳性解的结果。因为需要对所有假阳性结果进行检验来筛除，对运行时间的影响为。

假设一个随机生成的长度为d的DNA序列s满足的频率分布，那么有。所以，因此我们需要尽可能地选择较大的参数k。若d=100，r=20，将视为固定的值，求解k，有

(3.25)

(3.26)

(3.27)

当取不同的数值时，对应的更为直观的结果可见表3.6。

表 3.6 与的对应关系

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 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）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 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3.7。

表 3.7 集合的Minhash摘要

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 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 (, ) 对应的Minhash值相等的概率r = 。在区间1中，。因此如果足够大，和在区间1内的三个最小哈希值就可能完全一致，和可以被认定为相同。也就是说，。

现在有4个区间，其他区间与第一个区间等价，所以有。那么至少存在一个区间上的概率。函数图像如下。

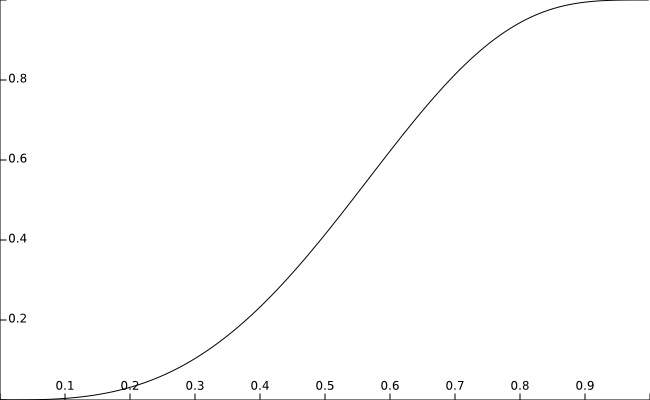


图 3.3 4个区间，12个哈希函数下的冲突概率函数

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）=。

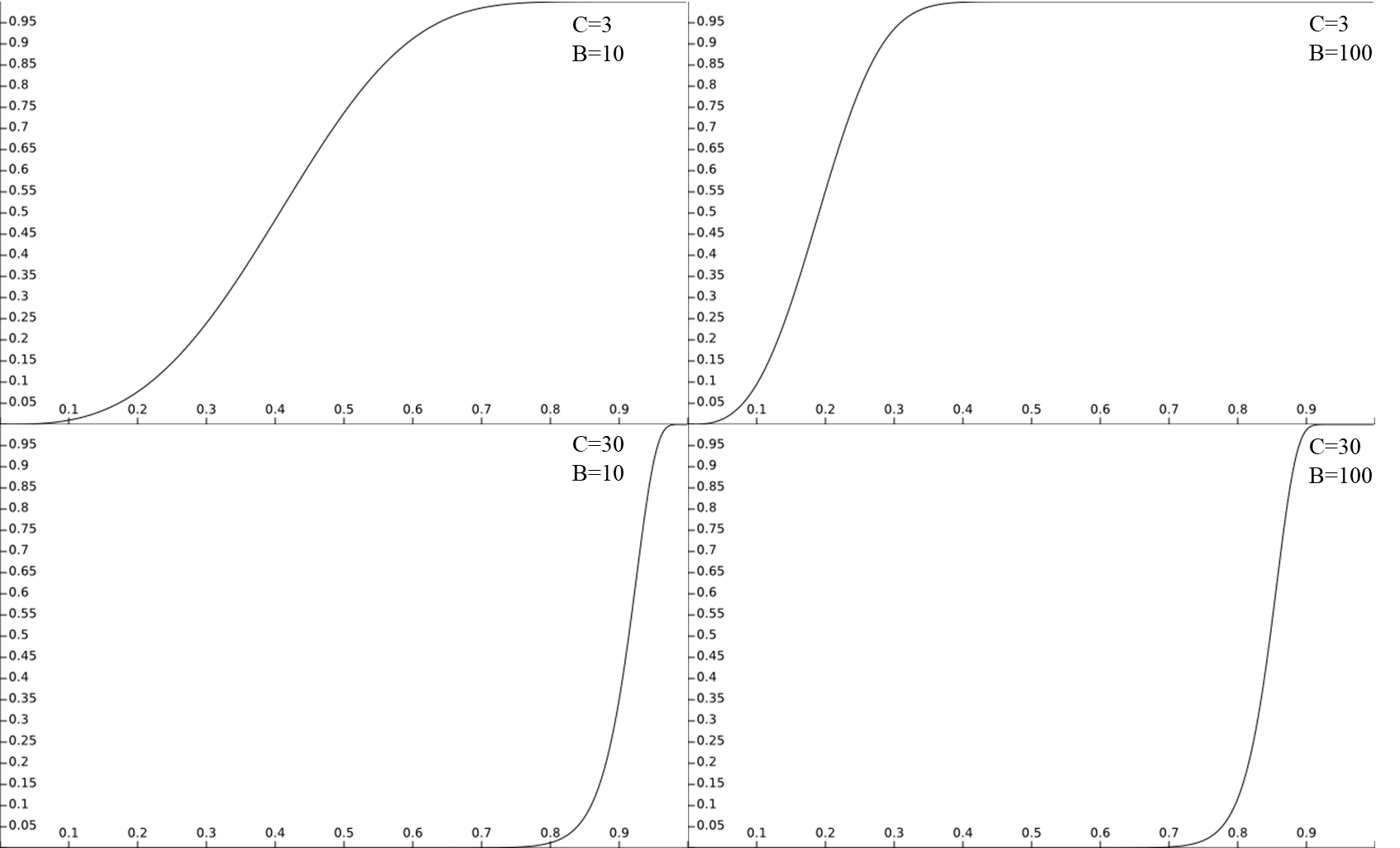


图 3.4 不同B和C的值对冲突概率函数的影响

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。

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维空间上，包含n个数据点的集合P。对于任意两点p和q，它们之间的距离定义为

(3.28)

对于任意的S>0，这个距离函数被称为标准化。如果，则称p是q的R近邻 (R-near neighbor) 。如图3.5，，，都是q的R近邻，则不是。



图 3.5 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[22] ，也被称为Randomized c-approximate R-near neighbor ((c,R)-NN)。

给定点集，查询点，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，满足，则输出某点，满足。



图 3.6 离散空间上的ANN

我们需要让在相近的R近邻（图3.6中白色部分）进行hash冲突的概率更大，而cR近邻（图3.6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，如果在P中存在q的R近邻，能以1- (>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-上升到1-。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，从H中均匀随机选择一个函数g，有

(a) 如果，；

(b) 如果，；

(c) ，

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

我们选择个函数，其中，而是从H中均匀随机选择的函数 () 。使用这些函数将P转换到个散列表中，之后对于每一组询问，查找这些散列表以提取数据点进行验证。如果，；如果，。

对于一个c-NN问题，我们先找到 个点验证，然后中止。定义事件E1为假阳性的数量少于 个数据点（即总哈希冲突次数），E2为以1-的概率找到一个解决方案。我们需要根据给定的和，找到最佳参数和。

碰撞的概率（n表示数据点的数量），因此可以推导得。

q在散列表a中发生冲突的期望 ，所以个散列表中q发生冲突的期望 。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，都有。因此，

(3.29)

(3.30)

如果存在NN（即），找到一个ANN的概率为

(3.31)

在，的条件下，

(3.32)

根据上述分析，可知

(3.33)

令，事件E1和E2同时为真的概率就是。当且仅当所有假阳性情况都被找到，。算法空间复杂度，搜索部分的时间复杂度。

对于一个R近邻问题，我们在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，=。 由于，所以化简得。

对于一个长度为d的子串，

(a) 如果，；

(b) 如果，；

所以，。

### 3.3.2 Order Min Hash模型

设是定义在集合（全集）上的散列函数族。当

(3.34)

(3.35)

则称集合上的概率分布对相似度关于敏感的。其中，。如果存在一组散列函数的分布关于 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，具体概率取决于中对任意 x, y 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 () ，而对于不同元素，哈希冲突的概率较小 () 。

编辑相似度上的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，k-mers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，我们并非只选取1个最小的哈希值，而是个。取个最小的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。

此外，该方法必须处理重复的k-mers。同一k-mer的两个副本出现在序列中的不同位置，区分这两个副本对于k-mer之间的相对顺序很重要。我们通过在k-mers后附加“出现次数”，使其唯一。

更准确地说，对于长度的字符串S，考虑k-mers对及其出现次数的集合。如果序列S中有x个m的副本，那么集合中有x对元素，形如。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），它的出现次数是。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 ）。我们把k-mer的 (m, i) 对和出现数称为“对应”的k-mer。

上的一个置换定义了两个函数和。是一个长度为 ，由项 组成的向量。具体要求如下：

* 根据置换，对是最小的 个 中的项；
* 按k-mer在序列S中出现的相对顺序，在向量中列出这些对。也就是说，如果 ， 且 ，那么就有。

向量 只包含来自 的k-mers，并以如上顺序排列。OMH方法定义为哈希函数集 上的均匀分布[23]。

对于极端情况，当时，向量包含覆盖整个序列S的所有加权k-mers。在这种情况下，OMH值的相等意味着序列的完全一致。

在另一种极端情况下，当时，向量仅包含一个k-mer，因此不存在相对顺序信息。在这种情况下，只会考虑和之间的k-mer内容相似性，即MinHash。

两个序列的加权Jaccard相似度 是其k-mer的加权Jaccard。由于k-mers在 中的出现次数是唯一的，因此加权Jaccard相似度等效定义为。

因为时，OMH等价于MinHash；时，哈希又失去意义。因此在下文的讨论中，我们规定。因此，我们需要证明对于任意 ，，存在函数 和 ，OMH在编辑距离上对 敏感。

假设和是两个长度为n的序列，每个序列中k-mer的数量=n-k+1。当n非正(n0) 时，二次项系数=0，表示从空集中选择元素的方案数为0。

令，则有。因此编辑距离。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。

当两个序列中所有没匹配上的k-mers都互不相同时，并集的大小达到最大。所以有。

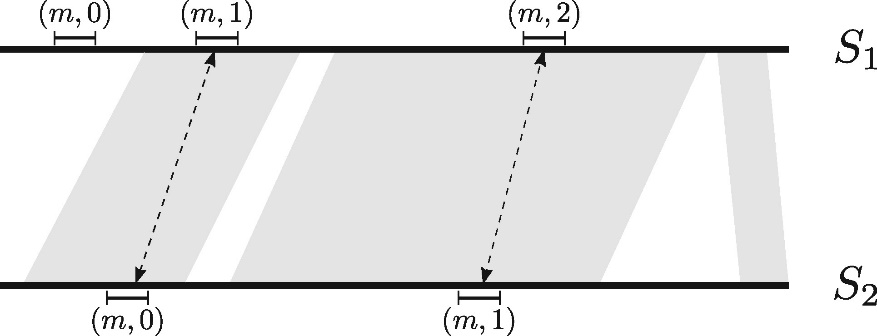


图 3.7 和匹配示意图[23]

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3.7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3.7中中的(m,0)）。因此，任意置换只要满足(m,0)在最小的个“对应的”的k-mer中，就不会导致哈希冲突。

设和中出现次数不同的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，不匹配的位数最多为（只采取增删操作）。所以，，可得。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。

(3.36)

通过置换，中的每一项都有相同的概率被选入最小的项中。因此，当时，产生一次哈希冲突的概率

(3.37)

这里找到了式3.34中和的关系，因此OMH是满足式3.34的。

因为中所有元素都是唯一的，所以我们可以使用集合表示向量中的所有元素。令事件C表示=。在事件C发生的条件下，可以得到。因为 是 只使用k-mer的特殊情况，所以永远成立。而由于=，加权向量 中的内容是相同的。k-mers是按照它们在各自序列中出现的顺序列出，同时对于相同的k-mer，也可以视作按照它们的出现次数列出，所以未加权向量 的相等意味着加权向量的相等，也是一定成立的。

令m=，相当于两个序列加权k-mer集交集的大小。当且仅当通过置换后，个最小“对应的”k-mers全部来自，事件C成立。因此，

(3.38)

现在我们考虑在中的和分别按照它们在和中的出现顺序列出，长度都是m。在条件C下，事件相当于通过哈希函数在和中选择了长度为的公共子序列。由于集合中的元素没有重复，因此和中的公共子序列 (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。

(3.39)

其中 代表 的全排列。

当，，对于任意序列长度为 且最长上升子序列 (Longest Increasing Subsequence, LIS) 长度为 的序列来说，长度为 的递增子序列的最大数量为

(3.40)

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。

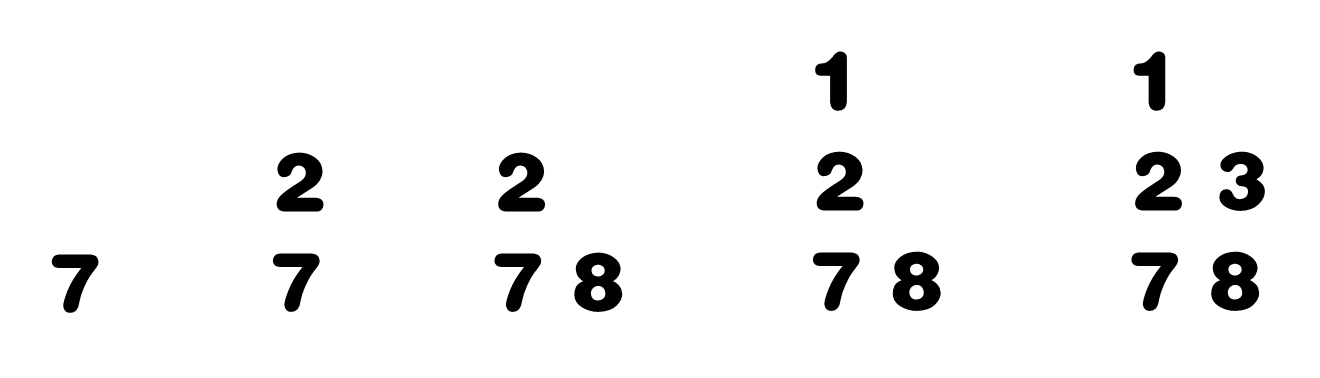


图 3.8 耐心排序示意图

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，有一个长度为n的序列S，其中LIS长度为。经过耐心排序后，每个堆栈的高度组成向量。S中长度为的上升子序列数量的上界

(3.41)

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

当时，取得最大值。集合是一个紧集 (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。因此在C上存在最大值。假设在s中，存在不是所有都相等。那么我们不失一般性地设和不相等。令，向量。

(3.42)

然后我们从分解出既不包含，又不包含的项（=），包含或其中一个的项（=），包含和的项（=）。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。

(3.43)

所以当中包含两个值不同的元素时，就无法达到最大值。只有当时，取得最大值。

(3.44)

函数是一个增函数。当的时候，函数取得最大值。

我们考虑这样一个序列，其中 能被 整除。序列具体形式如下。

(3.45)

每个块的长度为 ，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 组成。

当耐心排序算法应用于序列 时，堆栈从下到上、从左到右逐个填充，并且具有相同的 的高度。因此，从每个堆栈中任意选择一个元素都是 的有效上升子序列，并获得等式3.44中的界。所以式3.40得以证明。

我们最终需要证明式3.35。即找到一个，使得。依照上文的分析，

(3.46)

根据式3.38和式3.39，以及式3.40，最后可以推出

(3.47)

其中L表示和的最长公共子序列的长度。式3.48右侧的式子是关于L的增函数。当 时，式子等价于0；当 时，式子等价于1。

给定的条件，有。，将式3.48中的L用替换，得

(3.48)

综上，当时，。找到了式3.35中和的关系，因此OMH是满足式3.35的。所以，OMH是关于敏感的，可以被用于相似序列的聚类。

### 3.3.3 模型复现

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 3.1 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 𝑘­mer 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3.1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 3.2 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 //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，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**  24: **for** j = 1 **to**  25: **for** jj = j + 1 **to**  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 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。以下给出我的OMH函数具体实现过程。

代码清单 3.1 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3.1中，将L个存储 个k-mer序列的散列表巧妙地转换成了，先将 个k-mer序列映射成某个hash值，再将 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 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..-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3.3所示。

算法 3.3 动态规划求解编辑相似度

|  |
| --- |
| **输入**：长度分别为 的蛋白质序列 和  **输出**：输入序列 和 的编辑相似度 s  1:( let dp[0..][0..] be a new array  2: (**for** i = 0 **to** -1  3: dp[i][0] = i  4: (**end for**  5:( **for** i = 0 **to** -1  6: dp[0][i] = i  7: (**end for**  8: (**for** i = 1 **to**  9:( **for** j = 1 **to**  10: **if** [i] = [j]  11: dp[i][j] = dp[i-1][j-1]  12: **else** dp[i][j] = 1 + min{ dp[i-1][j], dp[i][j-1], dp[i-1][j-1] }  13: ( **end for**  14: (**end for**  15: (s = 1 – dp[][] / max(, )  15: (**return** s |

## 3.4 基于MinHash算法和LSH算法的高效近似Jaccard相似度估计算法

对OMH算法进行分析，我们不难发现，因为OMH的提出是用来对碱基序列的相似度进行聚类的，所以它的一些实现在蛋白质序列上并不实用。具体来说，因为碱基序列只包含ACGT四种字符，假设k-mer中的k为3，也就只能产生 种k-mer。如果序列长度比较长，一个序列就非常容易产生多个相同的k-mer。所以OMH中提出的包含出现相对位置的“对应的”k-mer就很巧妙地解决了加权集合的问题。

但是蛋白质序列中含有26个字符，假设k-mer中的k为3，就能产生 种k-mer。这个数量级是非常恐怖的，意味着一个长度为L的序列中出现多个相同的k-mer的概率会比碱基序列低很多。

因此OMH关于相对位置的考虑就显得不太必要。将OMH算法进行改造，使得哈希冲突率在可接受范围内上升，期望能找到效率和准确性的更优平衡点。

本文模型吸收OMH模型关于 个最小哈希值的思想，一定程度上会同时降低TP率和FP率，但是能很大程度优化时间复杂度。最后回归LSH算法，仅按照 个最小哈希值进行聚类，改进了OMH模型哈希冲突率较低的特点。

3.5 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。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。接着从算法时间复杂度角度进行论证，寻找问题规模降维的可行性。之后，引入Order Min Hash模型，先对文本相似度算法和散列函数交叉领域的术语做了详细介绍，分析MinHash算法和Locality-sensitive hashing算法的原理，并具体计算了概率和期望。对于Order Min Hash模型的理念，给出了基于C++的实现方法。最后，基于MinHash算法和Locality-sensitive hashing算法，对Order Min Hash模型进行修改，给出Jaccard相似度的高效近似估计算法。

# 4 算法效率与准确性在大数据集上的评估

4.1 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[24]。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[25]。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4.1为一个简单的FASTA文件示例。

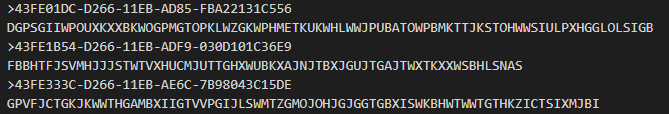


图 4.1 FASTA文件示例

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4.1。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 4.1 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。。。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。表4.2和表4.3展示了一些常见的读取方法和函数在各大平台的测试时间。

表 4.2 读取文件方案在各大平台运行时间

|  |  |  |  |
| --- | --- | --- | --- |
| 方法/平台/时间(s) | Linux gcc | Windows mingw | Windows VC2008 |
| scanf | 2.010 | 3.704 | 3.425 |
| cin | 6.380 | 64.003 | 19.208 |
| cin取消流同步 | 2.050 | 6.004 | 19.616 |
| fread | 0.290 | 0.241 | 0.304 |
| read | 0.290 | 0.398 | 不支持 |
| mmap | 0.250 | 不支持 | 不支持 |
| Pascal read | 2.160 | 4.668 |  |

表 4.3 读取字符串函数的运行效率

|  |  |
| --- | --- |
| 读取函数 | 所用时间 |
| gets | 72ms |
| fgets | 76ms |
| scanf | 960ms |
| getline | 2189ms |
| cin | 2275ms |

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。朴素算法可以保证结果的100%准确率，从而可以找到比较算法的FP率和FN率。

4.2 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4.4。测试数据由真实数据随机产生。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，为了方便结果比对，我们往里加入了5条在真实序列基础上进行编辑操作的序列，并把同源性的阈值设置为0.5。选用的平台是Windows mingw，读入均采用getline函数，不进行I/O优化。

表 4.4 实验参数设置

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 意义 | 值1 | 值2 |
| k | k-mer的k大小 | 3 | 4 |
|  | 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 |

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。

4.3 实验结果与分析

### 4.3.1 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4.5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 4.5 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。

### 4.3.2 参数调整对改进模型的影响

实验需要调整的参数如表4.4所示。我们将k=4，=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4.6。

表 4.6 不同参数的对照实验结果

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k |  | | 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 |

续表 4.6 不同参数的对照实验结果

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k |  | | L | p | | pp | 解的组数 | 耗时 |
| 4 | 2 | 300 | | 19260817 | 100 | | 13 | 53.258 |
| 15 | 54.330 |
| 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。

=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=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=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。

4.4 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，包括实验环境、数据集预处理过程、模型参数设置与具体的训练流程，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，=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。区别于基础的MinHash算法，该模型在Order Min Hash模型的启发下，用最小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。

# 致 谢

行至此篇，论文已然接近尾声，也伴随着我的大学四年迎来终章。东秦四年，开启了我通向计算机广阔世界的大门，使我找寻到了为之不懈奋斗的方向。和ACM俱乐部朝夕与共的三年又十个月，不仅给我的代码能力和算法水平带来质的飞跃，也锤炼了我自信自强的品质，更加从容坦率地迈向人生新篇章。

由衷地感谢王和兴老师在科研上的悉心指导以及在生活中给予的关心。作为东秦ACM的指导老师，王老师把东秦ACM竞赛带上了一个新高度，使得我一入学就有机会亲临全国赛事。同时，王老师带我开拓视野，走向更大的舞台。我不仅能够参与到多项科技大会工作，同时有机会站在河北省大学生程序设计竞赛的开幕式进行发言。这些珍贵经历，内化于心，外化于行。对于王老师的指导和关怀，我感激涕零，铭感五内。

感谢南开大学的周建宇老师提供的蛋白质同源性搜索的课题。周老师既是温柔亲切的老师，也是可靠可爱的朋友。在我进行毕业设计这段时间，他给予了充分的指导与帮助。被坚定地选择，本身就是这半年孤独之路最温暖的事。享受学习，而不是被迫学习；日常多看即使与自己研究领域无关的算法文章；不做无意义的卷，不跟别人比，淡泊明志，宁静致远。从周老师身上看到了很多己身弗如甚远的品质，督促我戒骄戒躁，精益求精。我希望未来能够跟随周老师展开更为深入的生物信息学方面的研究，雏凤清声，相辅相成。深藏于殷殷雪底的种子在世事沉浮中大概会有悄然绽放的那一天吧。

感谢东秦的方淼老师、张旭老师、陈晶晶老师，ACM的队友刘泽铭、卢佳星、喻军宇，建模竞赛的队友齐浩君、时睿佳、蒋雪涵，以及包括张紫闻、邓楠、刘婉乔、冯世龙在内的ACM俱乐部的小伙伴们，帮助我在科研竞赛上入门并快速成长，一直保持热爱，用一往无前的孤勇和睥睨山河的狂狷去翻越一座又一座的险峰。

感恩母校台州中学。“文理兼修，彰显特长”，这样的教学理念使我的学习之路自律且自由。感谢从双林南路220号一直陪伴我成长的郭旖倩同学和金婉盈同学。感谢6039的漂亮姑娘们，特别是我的莹宝、婷宝、宁宝，大家都会有光明的未来。

焉得谖草，言树之背。22年的养育，我的父母是我漫漫人生路的定海针。感谢他们以爱意的浇灌，深草中的小刺头才能如愿成长。

致谢删减数番，还是决定在最后，感谢自己的那份初心。如果没有那份想成为Hacker的梦想，如果没有《燕归来熙》描绘的黑客世界对我的致命吸引，我可能会去追逐自己的另一个梦想，那份关于 “梨花院落溶溶月，柳絮池塘淡淡风”的恬静。某种意义上，写代码恰似写诗，思绪倾涌，指尖跃动。虽然这四年的旅途看似离我的初心越来越远，但我坚信，人生海海，山山而川，我们终将重逢。

每一岁都有新的欢喜，都是最好的年岁。希望未来我还能在学习中感受到现下这份极致的快乐，也能去实现很多没来得及实现的愿望。虽说“来日方长”会显得虚无缥缈，但是还是想和所有我爱的和爱我的人说，我们来日方长。也希望那个时候，我能如愿成为一个温柔且强大的人。

终军弱冠，遇钟期奏流水潺潺；宗悫长风，逢杨意志青云皎皎。

何其幸运，得诸君相知相识，引为同侪。纸短情长，聊表谢忱。

# 参考文献

1. Jumper J , Evans R , Pritzel A , et al. Highly accurate protein structure prediction with AlphaFold[J]. Nature, 2021:1-11.
2. William R , Pearson. An introduction to sequence similarity ("homology") searching.[J]. Current protocols in bioinformatics / editoral board, Andreas D. Baxevanis. [et al.], 2013.
3. 冯冬. 微生物同源蛋白质序列相似性分析[D]. 河北工业大学, 2015.
4. Smith T F , Waterman M S . Identification of common molecular subsequences.[J]. Journal of Molecular Biology, 1981, 147(1):195-197.
5. S. B. Needleman and C. D. Wunsch, Needleman-Wunsch Algurithm for sequence similarity searches. Mol. Biol., 1970, 48: 443 - 453.
6. BD Ondov, Treangen T J , Melsted P , et al. Mash: Fast genome and metagenome distance estimation using MinHash[J]. Genome Biology, 2016, 17(1).
7. Jain C , Dilthey A , Koren S , et al. A Fast Approximate Algorithm for Mapping Long Reads to Large Reference Databases[C]// International Conference on Research in Computational Molecular Biology. 2017.
8. Pearson, William R . Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms.[J]. Genomics, 1991, 11(3):635-650.
9. Sung W K . Algorithms in bioinformatics. A practical introduction[J]. Journal of Polymer ence Part A Polymer Chemistry, 2010.
10. Masek W J , Paterson M S . A faster algorithm computing string edit distances[J]. Journal of Computer and System ences, 1980, 20( 1):18-31.
11. Rashed A E E D, Amer H M, El-Seddek M, et al. Sequence Alignment Using Machine Learning-Based Needleman–Wunsch Algorithm[J]. IEEE Access, 2021, 9: 109522-109535.
12. Wozniak A. Using video-oriented instructions to speed up sequence comparison[J]. Bioinformatics, 1997, 13(2): 145-150.
13. Chan A. An analysis of pairwise sequence alignment algorithm complexities: Needleman-wunsch, smith-waterman, fasta, blast and gapped blast[J]. 2007.
14. Vermij E P. Genetic sequence alignment on a supercomputing platform[J]. 2011.
15. Manavski S A, Valle G. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment[J]. BMC bioinformatics, 2008, 9(2): 1-9.
16. Rognes T. Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation[J]. BMC bioinformatics, 2011, 12(1): 1-11.
17. Sarje A, Aluru S. Parallel biological sequence alignments on the cell broadband engine[C]//2008 IEEE International Symposium on Parallel and Distributed Processing. IEEE, 2008: 1-11.
18. Lesk A. Introduction to bioinformatics[M]. Oxford university press, 2019.
19. 展永. 生物物理学[M]. 北京:科学出版社，2011:189-206.
20. 殷建平, 徐云, 王刚, 等. 算法导论 (原书)[M]. 计算机教育, 2013 (10): 51-51.
21. Niwattanakul S, Singthongchai J, Naenudorn E, et al. Using of Jaccard coefficient for keywords similarity[C]//Proceedings of the international multiconference of engineers and computer scientists. 2013, 1(6): 380-384.
22. 马恒钊. 若干近邻问题的亚线性算法设计与分析[D].哈尔滨工业大学,2021.DOI:10.27061/d.cnki.ghgdu.2021.000449.
23. Marçais G, DeBlasio D, Pandey P, et al. Locality-sensitive hashing for the edit distance[J]. Bioinformatics, 2019, 35(14): i127-i135.
24. 殷泽坤. 大规模超长生物序列聚类分析[D].山东大学,2020.DOI:10.27272/d.cnki.gshdu.2020.006010.
25. Lipman D J, Pearson W R. Rapid and sensitive protein similarity searches[J]. Science, 1985, 227(4693): 1435-1441.

# 附 录

## 附录A

**Locality-sensitive hashing for the edit distance**

Introduction

Measuring sequence similarity is the core of many algorithms in computational biology. For example, in the overlap–layout–consensus paradigm to assemble genomes, the first overlap step consists of aligning the reads against one another to determine which pairs have a significant alignment (an overlap). In meta-genomics, sequencing reads, or longer sequences created from these reads, are aligned against known genomes, or against one another to cluster the sequences, to determine the constituent species of the sample. Sequence similarity is also at the heart of the many general sequence aligners, either genome to genome [e.g. MUMmer4, LASTZ] or reads to genome [e.g. Bowtie2, BWA], that are used in countless pipelines in bioinformatics.

Despite many algorithmic and engineering improvements [e.g.implementation on SIMD and GPU], computing the sequence alignment or edit distance between two sequences takes approximately quadratic time in the length of the input sequences, which remains computationally expensive in practice. Given that the edit distance is likely not computable in strong subquadratic time, most aligners rely on heuristics to more quickly detect sequences with a high probability of having an alignment.

Recent aligners, such as Mash, Mashmap, or overlappers such as MHap, use a method called ‘locality-sensitive hashing’ (LSH) to reduce the amount of work necessary.The procedure is a dimensionality reduction method and works in two steps. First, the sequences (or part of the sequences) are summarized into sketches that are much smaller than the original sequences while preserving important information to estimate how similar two sequences are. Second, by directly comparing those sketches (with no need to refer to the original sequences) or by using these sketches as keys into hash tables, the software finds pairs of sequences that are likely to be similar. A more thorough, and computationally expensive, alignment procedure may then be used on the candidate pairs to refine the actual alignments.

In an LSH method, the distance between sketches is used as a first approximation for the distance between the sequences. That is, with high probability, two sequences which are very similar must have sketches which are similar, and conversely dissimilar sequences have dissimilar sketches. More precise definition of these concepts is given in Section 2.

Instead of using an LSH for the edit distance or an alignment score, in practice sequence alignment programs use the minHash LSH for the Jaccard similarity or an LSH for the Hamming distance as a proxy for the edit distance. Although these two techniques have proven themselves useful in practice, they suffer from one major flaw: neither the Jaccard similarity nor the Hamming similarity directly corresponds to the edit distance (see Section 2.2 for examples). In fact, it is possible to find sequences that are indistinguishable according to the Jaccard similarity, but have large edit distance. Similarly, with the Hamming distance, there exist sequences with very low edit distance that are completely dissimilar according to the Hamming similarity.

Depending on the problem and the software implementation, the cases above can lead to false negatives (an alignment is missed) and a decrease in precision, or false positives (a nonexistent potential alignment reported) and extra computational work. An LSH method for edit distance instead of the proxy Jaccard or Hamming similarities would reduce both of these issues.

Although multiple definitions are possible for sequence similarity(or distance), in this study, we focus on the edit distance (a.k.a. Levenshtein distance, Levenshtein, 1966), which is the number of operations (mismatch, insertion, deletion) needed to transform a string into another one.

Two methods that are LSH for the edit distance have been described previously. Bar-Yossef et al. (2004) propose a sketch that can distinguish, with some probability, between sequences with edit distance t from sequences with edit distance , where n is the length of the sequences, for any . They use an indirect method to obtain an LSH for the edit distance: first they embed the edit distance space into a Hamming space with low distortion, and second, apply an LSH on the Hamming space. That is, the input sequence is first transformed into a bit vector of high dimension, then sketching for the Hamming distance is applied to obtain an LSH for the edit distance.

Similarly, Ostrovsky and Rabani (2007) propose a two-step method, where the edit distance space is first embedded into an space with low distortion, then a sketching algorithm for the is used to obtain an LSH for the edit distance. This method can distinguish between sequences with edit distance t and edit distance , for some constant c.

We propose a simpler and direct method that is an LSH for the edit distance. Our method is an extension to the minHash method. We call our method OMH for Order Min Hash, and it can be seen as a correction of the minHash method. The probability of hash collision in the OMH method is the product of two probabilities. The first is the probability to select a k-mer from the set of common k-mers between the two sequences. This probability is similar to minHash that estimates the Jaccard similarity between the k-mer contents of two sequences. However, there is one key difference: the minHash method estimates the Jaccard similarity which treats sequences as sets of k-mers, and the number of occurrences of each k-mer in the sequences is ignored, whereas OMH estimates the weighted Jaccard, where the number of occurrences of a k-mer in a sequence is significant, i.e. the weighted Jaccard works with multi-sets. The second probability is the likelihood that the common k-mers appear in the same relative order in the two sequences. Therefore, OMH is sensitive not only to the k-mer content of the sequences but also to the order of the k-mers in the sequences.

The sketch proposed for OMH is only slightly bigger than the sketch for minHash while maintaining significantly more information about the similarity of two sequences. In addition to providing an estimate for the edit distance between two sequences, it also provides an estimate of the k-mer content similarity (the weighted Jaccard) and how similar the relative order is between the common k-mers of the two sequences.

Section 2 summarizes the notation used though out and main results. Detailed proofs of the results are given in Section 3. Section 4 discusses some practical consideration on the implementation of the sketches.

Main results

**Concepts and definitions**

*Similarity and dissimilarity.* A dissimilarity is a function : that indicates the distance between two elements in the universe . satisfies the triangle inequality and means that . In other words, a dissimilarity is a normalized distance. A similarity is a function such that is a dissimilarity. Hence, a dissimilarity defines a similarity and vice versa. We will therefore use either of the terms ‘edit dissimilarity’ or ‘edit similarity’.

Given two strings of length n (where is the alphabet of size ), the Hamming dissimilarity is the number of indices at which and differ divided by : ( denotes the set ). The edit dissimilarity (a.k.a. normalized edit distance) is the minimum number of indels (short for insertion or deletion) and mismatches necessary to transform into , divided by . Given two sets A and B, the Jaccard similarity is .

*Gapped LSH.* Let H be a set of hash functions defined on a set (the universe). A probability distribution on the set is called -sensitive for the similarity when

(1)

(2)

where and . A similarity admits a gapped LSH scheme if there exists a distribution on a set of hash functions that is -sensitive. In the definition above, the probability is taken over the choice of the hash function in and the implications hold for any choice of x and y. In a gapped LSH, the probability of a hash collision is increased () between similar elements, and less likely ( ) for dissimilar elements.

In the following, the probabilities are always taken over the choice of the hashing function, even though we may omit the ‘’ subscript.

*LSH.* An LSH for a similarity is a family of hash functions that is -sensitive for any . Equivalently, the family of hash functions satisfies . In practice a gapped LSH is typically used to put elements into a hash table where there is high likelihood of a collision, whereas a full LSH can be used as a direct estimator of the underlying measurement.

*minHash sketch.* Let the universe be a family of sets on the ground set X (i.e. ). The minHash LSH for the Jaccard similarity is defined as the uniform distribution on the set . That is, the hash function selects the smallest element of the set A according to some ordering of the elements of the ground set X. This family of hash functions is -sensitive for any value of , or equivalently .

*LSH for Hamming similarity.* The Hamming similarity between two sequences with same length n is the proportion of positions which are equal: . For the Hamming similarity, the uniform distribution on satisfies .

*String k-mer set.* For a sequence S, the set of its constituent k-mers is , where is the substring of length k starting at index i. By extension, the Jaccard similarity between two sequences is the Jaccard similarity between their k-mer sets: .

*Weighted Jaccard.* The weighted Jaccard similarity on multi-sets (or weighted sets) is defined similarly to the Jaccard similarity on sets, where the intersection and union take the multiplicity of the elements into account. More precisely, a multi-set A is defined by an index function , where gives the multiplicity of x in A (zero if not present in A). The index function of the intersection of two multi-sets is the minimum of the index functions, and for the union it is the maximum. Then, the weighted Jaccard is defined by

.

This is a direct extension to the set definitions, where the index function takes values in {0, 1}.

**Jaccard and Hamming similarities differ from edit similarity**

Similarly to the definition of the LSH, we say that a similarity is a -proxy for the similarity if

(3)

(4)

That is high similarity for implies high similarity for , and the converse. Because of the similar structure between the definitions of sensitivity and proxy, if is not a proxy for (for any non-trivial choice of parameters ), then an LSH for f1 is not an LSH for .

We show here that neither the Hamming similarity nor the Jaccard similarity is a good proxy for the edit dissimilarity. More precisely, only one of the implications above is satisfied.

*Jaccard similarity differs from edit similarity.* A low Jaccard similarity does imply a low edit similarity (Equation 4). On the other hand, consider the sequence that has 0s followed by k 1s, and with k 0s followed by 1s (k fixed, n arbitrarily large). The k-mer sets of and are identical, hence , while the edit similarity is . These sequences are indistinguishable according to the Jaccard similarity while having arbitrarily small edit similarity (Equation 3 not satisfied).

*Weighted Jaccard similarity differs from edit similarity.* Consider two de Bruijn sequences: sequences of length containing every k-mer exactly once (van Aardenne-Ehrenfest and de Bruijn, 1951). There is a very, very large number of such sequences , and although any two such sequences have exactly the same k-mer content, they might otherwise have a very low edit similarity. Both the Jaccard and weighted Jaccard similarities fail to distinguish between de Bruijn sequences, regardless of their mutual edit dissimilarity.

For example, the two sequences 1111011001010000111 and 0000101001111011000 of length 19 each contain exactly the 16 possible 4-mers; hence, their Jaccard and weighted Jaccard similarities are 1. Their edit similarity is only . By comparison, two random binary sequences of length 19 have an average edit similarity of and an average Jaccard similarity of . In other words, these two de Bruijn sequences are much more dissimilar than two random sequences despite having a perfect Jaccard similarity.

More generally, both Jaccard and weighted Jaccard similarities treat sequences as bags of k-mers. The information on relative order of these k-mers within the sequence is ignored, although it is of great importance for the edit similarity. In contrast, an OMH sketch does retain some information on the order of the k-mers in the original sequence. In the case of the two de Bruijn sequences above, the proportion of pairs of k-mers that are in the same relative order in the two sequences is 0.4. The expected similarity between the OMH sketches of these sequences is also equal to 0.4.

*Hamming similarity differs from edit similarity.* A high Hamming similarity does imply a high edit similarity (Equation 3). The opposite is not true however. Consider the sequences of length n, and . These sequences have a Hamming similarity of 0 and an edit similarity of (two indels). That is, these sequences are as dissimilar as possible according to the Hamming dissimilarity, but an arbitrarily high edit similarity (Equation 4 not satisfied).

The Hamming similarity is very sensitive to the absolute position in the string. A single shift between two sequences has a large impact on the Hamming similarity but only a unit cost for the edit similarity. An OMH sketch on the other hand only contains relative order between k-mers and is indifferent to changes in absolute position.

**LSH for the edit similarity**

An LSH for the edit similarity must be sensitive to the k-mer content of the strings and the relative order of these k-mers, but relatively insensitive to the absolute position of the k-mers in the string. This motivates the definition below. Similarly to the minHash, k-mers are selected at random by using a permutation on the k-mers. Additionally, to preserve information about relative order, k-mers are selected at once and recorded in the order they appear in the sequence (rather than the order defined by the permutation).

Additionally, the method must handle repeated k-mers. Two copies of the same k-mer occur at different positions in the sequence, and it is important for the relative ordering between k-mers to distinguish between these two copies. We make k-mers unique by appending to them their ‘occurrence number’.

More precisely, for a string S of length , consider the set of the pairs of the k-mers and their occurrence number. If there are x copies of m in sequence S, then the x pairs are in the set , and the occurrence number denotes the number of other copies of m that are in the sequence S to the left of this particular copy. That is, if m is the k-mer at position i in S (i.e. ), then its occurrence number is . This set is the ‘multi-set’ of the k-mer content of string S, or the ‘weighted set’ of k-mers where the number of occurrences is the weight of the k-mer (hence the superscript). We call a pair (m, i) of a k-mer and an occurrence number a ‘uniquified’ k-mer.

A permutation of defines two functions and . is a vector of length of elements of such that:

1. the pairs are the smallest elements of according to ,
2. the pairs are listed in the vector in the order in which the k-mer appears in the sequence S. That is, if , and , then .

The vector contains only the k-mers from , in the same order. The OMH method is defined as the uniform distribution on the set of hash functions .

For extreme cases, where , the vector contains overlapping k-mers that cover the entire sequence S. In that case, equality of the hash values implies strict equality of the sequences.

At the other extreme, where , the vectors contain only one k-mer and no relative order information is preserved. In that case, only the k-mer content similarity between and matters.

The weighted Jaccard similarity of two sequences is the weighted Jaccard of their k-mer content (seen as multi-set). Because k-mers were made unique by their occurrence number in , the weighted Jaccard similarity is equivalently defined as .

THEOREM 1. *When , OMH is an LSH for the weighted Jaccard similarity:*

(5)

PROOF. This proof is similar to that of minHash and the Jaccard similarity. Because every uniquified k-mer in has the same probability of being selected, the probability of having a hash collision is the same as selected a k-mer from the intersection where the probability of picking a k-mer is weighted by its maximum occurrence number.

As we shall see in Section 4.4, the weighted Jaccard similarity contains approximately the same information as the Jaccard similarity with respect to the edit similarity.

For the general case , we shall prove the following theorem in Section 3 that OMH is a gapped LSH for the edit dissimilarity.

*THEOREM 2. For any and any , there exist functions and such that OMH is -sensitive for the edit distance.*

The actual functions and are explicitly defined in Section 3, but they may not be easily expressed with elementary functions in general.

中文译文A

局部敏感哈希在编辑距离上的应用

背景

序列相似性度量是计算生物学中许多算法的核心。例如，在组装基因组的overlap-layout-consensus（重叠-排列-生成一致序列）策略中，第一步的重叠操作中包括将序列对彼此对齐，以确定哪些序列对具有显著对齐的特征（重叠）。在宏基因组学中，测序读取的序列（或者从读取中创建的更长序列），与已知基因组对齐，或彼此间进行对齐，可以对序列进行聚类，从而确定样本的组成物种。序列相似性度量也是许多综合序列比对器的核心。无论是基因组-基因组（如MUMmer4和LASTZ），还是序列-基因组（如Bowtie2和BWA），都在生物信息学的无数管道工具 (pipeline) 中使用。

尽管有许多算法和工程对此改进（例如，在SIMD和GPU上实现），但序列比对（计算两个序列之间的编辑距离）仍大约需要平方级别的时间复杂度，在实践中仍然很低效。考虑到编辑距离在强次二次时间内可能无法计算，大部分比对器依靠启发式算法以更快速地检测出具有高对齐概率的序列。

最新的一些比对器（如Mash和Mashmap）或重叠器（如MHap）使用了一种被称为“位置敏感哈希”（LSH）的方法来减少必要的工作量。该过程是一种降维方法，分两步进行。 首先，将序列（或部分序列）分成比原始序列小得多的sketches，同时保留重要信息以估计两个序列的相似程度。其次，通过直接比较这些sketches（而无需参考原始序列），或者将这些sketches映射成散列表的关键字，软件可以从中找到可能相似的序列对。然后，在这些候选对的基础上，使用更彻底且计算成本更高的比对过程来优化实际结果。

在LSH方法中，sketches之间的距离用作序列之间距离的第一近似值。换而言之，两个非常相似的序列很大可能有相似的sketches；相反，不相似的序列有不同的sketches。第2节将给出了这些概念更精确的定义。

在日常实践中，序列比对程序使用minHash LSH来表示Jaccard相似度，或者使用LSH表示汉明距离，从而替代编辑距离，而不是使用LSH直接表示编辑距离。尽管前两种方法在实践中已经被证明是有效的，但它们有一个主要缺陷：Jaccard相似度和Hamming相似度都不直接对应于编辑距离（示例见第2节）。事实上，可以找到Jaccard相似度接近但编辑距离较大的序列。类似地，存在编辑距离非常小的序列对，但汉明相似度很低的情况。

根据具体问题和软件实现的不同，上述情况可能会导致假阴性情况（false negatives，相似度高的序列对被丢弃），带来精度的降低；或假阳性情况（false positives，比对结果相近的序列对实际相似度不高），增加额外的计算工作。相较于用LSH方法计算Jaccard或Hamming相似度来近似，用LSH直接计算编辑距离可以减少上述两个问题。

虽然序列相似度（或距离）可能有多种定义，但在本研究中，我们关注编辑距离（又称Levenshtein距离），即将一个字符串转换为另一个字符串所需的操作（不匹配、插入、删除）数。

前文已经描述了两种应用于编辑距离的LSH方法。Bar Yossef等人在2004年提出了一种sketch，它能够以一定的概率从编辑距离不小于的序列中区分编辑距离不大于t的序列。其中n是序列的长度，。他们使用一种间接方法来获得编辑距离的LSH：首先，他们将编辑距离空间嵌入到低失真的汉明空间中，然后，在汉明空间上应用LSH。也就是说，首先将输入序列转换为高维的位向量，接着对汉明距离进行计算，以获得编辑距离的LSH。

类似地，Ostrovsky和Rabani在2007年提出了一种二步法。首先将编辑距离空间嵌入到低失真的空间中，然后在上使用素描算法获得编辑距离的LSH。该方法可以区分编辑距离不大于t的序列和编辑距离不小于的序列，c为常数。

我们提出了一种简单而直接的方法，即编辑距离的LSH。我们的方法对minHash进行了扩展。我们称我们的方法为Order Min Hash (OMH) ，它可以被看作是对minHash方法的修正。OMH方法中产生哈希冲突的概率是以下两种概率的乘积：第一种是从两个序列的公共k-mers集合中选择同一个k-mer的概率。这种概率类似于minHash，用于估计两个序列的k-mer集合的Jaccard相似度。但是，这两种方法有一个关键的区别：minHash方法估计Jaccard相似度，是将序列视为k-mers集合，并且忽略序列中每个k-mer的出现次数；而OMH估计相似度则采用加权Jaccard方法， k-mer的出现次数也变得很重要，即加权Jaccard可用于多重集合。第二种概率是公共k-mers在两个序列中以相同的相对顺序出现的可能性。因此，OMH不仅对序列中k-mer的数目敏感，而且对序列中k-mers出现的顺序也敏感。

OMH提出的sketch仅略大于minHash的sketch，同时还保留了更多关于两个序列相似度的信息。除了提供两个序列之间编辑距离的估计之外，OMH还提供了k-mer内容相似性（加权Jaccard）的估计值，以及两个序列的公共k-mers在序列中相对顺序的相似程度。

文章第2节总结了使用的符号和主要结果。第3节给出了结果的详细证明。第4节讨论了sketch实施的一些实际考虑。

主要结果

**概念和定义**

*相似度与相异度。*相异度可以表示为函数：，即全集中的两个元素之间的距离。满足三角形不等式，且意味着。换而言之，相异度是一种归一化距离。相似度可以表示为函数，那么就代表了相异度。因此，相异度定义了相似度，反之亦然。我们紧接着将使用“编辑相异度”或“编辑相似度”这两个术语。

给定两个长度为n的字符串（其中是大小的字母集），汉明相异度是用和相异的位数除以长度n：（表示集合）。编辑相异度（又称归一化编辑距离）是将转换为所需的最小操作（插入或删除）数和不匹配数除以n。给定两个集合A和B，Jaccard相似度为。

*间隙LSH。*设H是定义在集合（全集）上的一组散列函数。当

(1)

(2)

则称集合上的概率分布对相似度关于敏感的。其中，。如果存在一组散列函数的分布关于敏感，则相似性允许有间隙的LSH方案。在上面的定义中，具体概率取决于中对任意x, y构造的哈希函数的选择。在有间隙的LSH中，相似元素之间哈希冲突的概率上升()，而对于不同元素，哈希冲突的概率较小( )。

下文中，我们选择散列函数时总是会考虑冲突概率，尽管可能会导致忽略 ‘’ 的条件。

*LSH。*表示相似度的LSH是对任意，都存在敏感的函数族。也就是说，哈希函数族满足。在实践中，有间隙的LSH通常用于将元素放入散列表中，这样产生冲突的可能性很高，而完整的LSH可以用作基础度量的直接估计。

*最小哈希素描*。假设全集是基础集X（即）的集合族。计算Jaccard相似度的minHash LSH定义为在集合上的均匀分布。换而言之，哈希函数根据基础集X中元素的某种随机排列，选择出集合A的最小元素。这个哈希函数族是对任意，都存在敏感。也就是。

*汉明相似度的LSH。*长度都为n的两个序列之间的汉明相似度是相同位置上具有相同字符的比例：。对于汉明相似度而言，上的均匀分布满足。

*字符串k-mer集。*对于一个序列S，组成它的k-mer集合定义为，其中是起始位置在下标i处，长度为k的子串。引申开来，两个序列之间的Jaccard相似度就是他们k-mer集合之间的Jaccard相似度：。

*加权Jaccard*。多重集合（或加权集合）上的加权Jaccard相似度的定义类似于普通集合上的Jaccard相似度，但计算交集和并集时考虑了集合元素的多重性。更准确地说，多重集合A被定义在一个权值函数中，考虑了A中元素x的多重性（如果x在A中不存在，则为零）。两个多重集合的交集大小是权值函数的最小值，而对于并集来说，则是最大值。因此，加权Jaccard相似度被定义为

.

这是对集合定义的直接扩展，其中权值函数取值在{0，1}中。

**Jaccard和Hamming的相似度与编辑相似度的区别**

与LSH的定义类似，当

(3)

(4)

我们说相似度是相似度对于的代替。也就是说，的高相似度意味着的高相似度，对于低相似度也是如此。由于灵敏度和代替之间存在结构上的相似，如果不是的代替（对于参数的任何非平凡选择），则的LSH不是的LSH。

我们在此表明，无论是汉明相似度还是Jaccard相似度都不能很好地代替编辑相异度。更准确地说，两者都只满足了上述条件之一。

*Jaccard相似度不同于编辑相似度。*低Jaccard相似度确实意味着低编辑相似度（满足式4）。另一方面，考虑序列，即由前n-k个0和后k个1组成；序列由前k个0和后n-k个1组成（k为固定常数，n可以是任意数）。和的k-mer集是相同的，因此，而编辑相似度不大于。根据Jaccard相似度，这些序列不可区分，但它们的编辑相似度却可以任意小（不满足等式3）。

*加权Jaccard相似度不同于编辑相似度。*考虑两个de Bruijn序列：恰好包含每个k-mer一次的长度为的序列（由van Aardenne Ehrenfest和de Bruijn在1951年提出）。这类序列的数量多达。尽管任何两个这样的序列具有完全相同的k-mer的多重集合，但它们的编辑相似度可能非常低。无论序列的编辑相异度多大，Jaccard相似度和加权Jaccard相似度都无法区分两个de Bruijn序列。

例如，长度为19的两个序列1111011001010000111和000010100111011000恰好都包含相同的16个互异的4-mers。因此，它们的Jaccard相似度和加权Jaccard相似度为1。他们的编辑相似度只约等于0.37。两个长度为19的随机二进制序列的平均编辑相似度约为0.62，平均Jaccard相似度约为0.36。换句话说，尽管这两个de Bruijn序列在Jaccard相似度上十分相近，但仍然比随机序列更不相似。

更一般地说，Jaccard相似度和加权Jaccard相似度都将序列视为k-mer的集合。尽管这对编辑相似度非常重要，但却忽略了序列中k-mers的相对顺序信息。相比之下，OMH sketch保留了一些关于原始序列中k-mers顺序的信息。对于上述两个de Bruijn序列来说，两个序列中具有相同相对顺序的k-mer对的比例为0.4。这些序列的OMH sketch之间的预期相似性也等于0.4。

*汉明相似度不同于编辑相似度。*高汉明相似度确实意味着高编辑相似度（满足式3）。反之则不成立。考虑长度为n的两个序列，，。这两个序列的汉明相似度为0，但编辑相似度不小于（只需要在序列头尾各进行一次操作）。也就是说，根据汉明相异度，两个序列完全不同，但编辑相似度却可以任意高（不满足式4）。

汉明相似度对字符串中的绝对位置非常敏感。两个序列之间的单次移位会对汉明相似度造成很大影响，但对于编辑相似度来说，仅需要单次操作的开销。而OMH sketch只包含k-mers之间的相对顺序，不受绝对位置的变化影响。

**编辑相似度的LSH**

编辑相似度上的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，k-mers是通过在k-mers上使用置换来随机选择的。此外，为了保留有关的相对顺序的信息， 个k-mers被随即选中，并按照它们在序列中出现的顺序（而不是随机置换所定义的顺序）记录。

此外，该方法必须处理重复的k-mers。同一k-mer的两个副本出现在序列中的不同位置，区分这两个副本对于k-mer之间的相对顺序很重要。我们通过在k-mers后附加“出现次数”，使其唯一。

更准确地说，对于长度的字符串S，考虑k-mers对及其出现次数的集合。如果序列S中有x个m的副本，那么集合中有x对元素，形如。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），它的出现次数是。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（因此为上标 ）。我们把k-mer的 (m, i) 对和出现数称为“对应”的k-mer。

上的一个置换定义了两个函数 和 。是一个长度为 ，由项 组成的向量。具体要求如下：

1. 根据置换，对是 个 中第 小的项；
2. 按k-mer在序列S中出现的顺序，在向量中列出这些对。也就是说，如果 ， 且 ，那么就有。

向量 只包含来自 的k-mers，并以相同的顺序排列。OMH方法定义为哈希函数集 上的均匀分布。

对于极端情况，当时，向量包含覆盖整个序列S的所有加权k-mers。在这种情况下，哈希值的相等意味着序列的完全一致。

在另一种极端情况下，当时，向量仅包含一个k-mer，因此不存在相对顺序信息。在这种情况下，只会考虑和之间的k-mer内容相似性。

两个序列的加权Jaccard相似度 是其k-mer（被视为多重集合）的加权Jaccard。由于k-mers在 中的出现次数是唯一的，因此加权Jaccard相似度等效定义为。

*定理1：当时, OMH 是加权Jaccard相似度的LSH:*

(5)

证明：这个证明类似于用minHash近似Jaccard相似度。因为 中每个“对应”的k-mer有相同的概率被选择，发生一次散列冲突的概率与从并集中选择一个k-mer的概率相同，其中选择某个k-mer的概率由其最大出现次数加权。

正如我们将在第4.4节中看到的，加权Jaccard相似度包含的信息与Jaccard相似度在编辑相似度上大致相同。

对于一般情况，我们将在第3节证明以下定理，即OMH是计算编辑相异度的有间隙LSH。

*定理 2：对于任意 , , 存在函数 和 ， OMH在编辑距离上对 敏感。*

实际函数和 在第3节中有明确定义，但它们通常不容易用初等函数表示。