# 摘 要

生物信息学是综合运用生物学、数学、物理学、信息科学以及计算机科学等诸多学科的理论方法的崭新交叉学科，它是整个生命科学发展的重要组成部分，已成为生命科学研究的前沿。生物信息学的核心是基因组信息学，包括基因组信息的获取、处理、存储、分配和解释，即以计算机、网络为工具，用数学、物理等学科的理论、方法和技术研究生物大分子，主要包括脱氧核糖核酸（DNA）和蛋白质（Protein）的序列、结构和功能。基因组信息学的关键是"读懂"基因组的核苷酸顺序，即全部基因在染色体上的确切位置以及各DNA 片段的功能；同时在发现了新基因信息之后进行蛋白质空间结构模拟和预测。这对于人类疾病基因的发现、基因与蛋白质的表达与功能研究、合理化药物设计等方面都有着重要的意义.

在生物信息学中，生物序列的比对、拼接和基因表达数据的聚类是一些最重要的操作。通过序列比对可以发现生物序列中的一些功能、结构以及进化信息。序列拼接是基因组测序阶段研究的最基本、最重要的问题，它处理全基因组随机测序得到的小片断序列，根据他们间的重叠关系用计算机将它们拼接以期得到目标序列的一个或多个较长的连续段。对基因表达数据的双向聚类用相似性度量函数确定基因的相似程度，将基因分组。通过聚类分析，将共调控基因聚集成类，分析和识别同类基因所共同拥有的转录调控元件。

由于以上的生物信息学问题具有数据量多、计算量大的特点，对计算机的处理速度等性能要求较高。而这些问题大部分本质上是组合优化问题，不象向量运算那样具有有规则的数据结构和相关关系，因而对其进行并行化有很大的难度。本文对生物信息学领域中的一些问题的并行计算进行了深入的研究，提出了相应的并行算法，通过试验取得了很好的效果。

我们提出一种快速的最长公共子序列算法FAST_LCS，该算法通过对字符串建立相应的同字符后续表，随后对于相应的初始同字符对，并行地在该表中逐层地搜索其后继同字符对，得到所有的后继同字符对及相应的层次值。最后由最大层次值的同字符对进行回溯，依次求得其所有前驱同字符对，最后得到相应的比对

结果。这种基于同字符后续表的算法同样可以用到多序列的最长公共子串问题中。算法使用了剪枝和跳跃技术，提高了处理速度。与其它经典的LCS算法相比，不但能够取得准确的结果，而且在速度、效率上有了很大的提高。

我们深入研究了生物序列的拼接问题，提出了一种高效的并行算法。该算法提出了后缀索引的概念以代替后缀树，首先对所有的序列片段建立后缀索引；然后对所有的序列片段，在所有其他序列片段的后缀索引中查找匹配度最高且最长的后缀,再以序列片段为顶点，以它们的前、后缀匹配长度为边上的权来建立带权有向图；使用并行蚁群算法寻找最长的哈密尔顿回路；最后根据回路得出拼接方案。该算法以后缀索引代替后缀树，大大减少了计算量，有利于并行计算。算法利用蚁群算法解决 TSP 问题的优势，减少了优化时间，与其它类似的算法相比，取得的结果准确，在速度、效率上有了很大的提高。

我们还对基因表达数据进行双向聚类的问题进行了深入的研究，提出了一种进行双向聚类的并行算法。该算法根据数据集合的大小对于双向聚类质量的反单调性，由最小的数据集合开始逐步添加行或列，最终找到所有满足条件的聚类。该算法处理速度快，聚类质量高，性能明显优于其他类似算法。

本文的研究工作将并行处理技术应用到生物信息学的研究之之中，对生物信息学中的一些问题提出的高效的并行算法，并且都在并行计算机深腾1800上用MPI(C绑定)编程运行，都使用生物信息的标准数据库中的测试数据进行试验。取得的试验结果表明，这些算法不但处理速度快，而且结果质量高，说明并行计算机是生物信息学研究中的有力工具，有关问题的高效并行算法的开发，会有力促进生物信息学的研究。

# Abstract

Bioinformatics is a new comprehensive cross discipline involving biology, mathematis, physics, informatics and computer science. It plays an important role in the development of the life science and become the frontier of life science research. The core issue in bioinformatics is genome informatics which includes the obtaining, processing, storing, assigning and explaining of the genome information. Using computers and network as tools, based on the mathematical and physical theory, methods and technology, genome informatics studies the biopolymers include the sequences, structures and functions of DNA and protein. The key issue in genome informatics is to understand the meaning of the order in nucleotide sequences, namely, to understand the exact locations of the genes in the chromosome and the functions of the DNA segments. Furthermore, it simulates and predicts the secondary and tertiary structure of protein using the genome information it discovered. These are very important in the research on disease gene of human being, the expression and function of gene and protein and the designing of pharmacy.

Alignment and splicing of biosequences, clustering of gene expressing data are the important tasks in bioinformatics. Biosequences alignment plays an essential role in sequence analysis, reconstruction of phylogenetic trees, detecting regions of significant sequence similarity in collections of primary sequences, and predicting the secondary and tertiary structure. Splicing of biosequences is the most important and essential task in the stage of genome sequencing. It assembles the small segments obtained by the genome sequencing into one or more longer and continuous objective sequences. The biclustering of the gene expressing data is to identify a subset of genes whose expressing levels rise and fall coherently with a subset of experimental conditions. By the biclustering, the genes with identical regulatory gene are classified into one cluster,

it is helpful to analysis and identify the presumptive regulatory sites of the genes belongs to the same cluster.

In solving those bioinformatical problems, high performance computer is required since they consumes large amount of computation time and memory space. Unlike the vector computation where the data structure and dependencies are regular which make it easy to be parallelized, those bioinformatical problems are essentially combinatory optimization problems which are difficult to be processed in parallel. In this paper, we deeply research on parallel processing of those bioinformatic problems, several parallel algorithms are presented and satisfied experimental results are obtained.

First, an fast algorithm for LCS problem named FAST_LCS is presented. The algorithm first seeks the successors of the initial identical character pairs according to a successor table to obtain all the identical pairs and their levels. Then by tracing back from the identical character pair at the largest level, the result of LCS can be obtained. The algorithm can also be used in solving multiple sequences alignment. In the process of seeking identical pairs, techniques of pruning and skipping are used to speedup the process. Experimental result shows that our algorithm can get exactly correct result and is faster and more efficient than other LCS algorithms.

Second, we also deeply study the problem of biosequences splicing and present an efficient parallel algorithm. In the algorithm the concept of suffix index was presented instead of suffix tree. The algorithm first constructs a suffix index for the segments of the gene sequences, then for each gene segment $i$ searches in the suffix index of all other segments , to find the longest matching suffix in segment $j$ . Suppose the length of such match is $l_{ij}$, a digraph is built with vertexes representing the segments and $l_{ij}$ representing the weight of the edge linking segment $i$ and $j$. Then the longest Hamilton circle is computed by parallel ant colony optimization. At last, the scheme of splicing is obtained according to the Hamilton circle. Since the algorithm

uses suffix index instead of suffix tree, computation load is largely reduced and it is more suitable for parallel processing. Since the algorithm exploits the strong optimization ability of ant colony optimization in TSP solving to find the longest Hamilton circle, it reduces large amount of optimization time. Compared with other similar algorithms, our algorithm can obtain more accurate results and has higher computational speed and efficiency.

Furthermore, after studying the problem of gene expressing data analysis, a parallel biclustering algorithm is also presented. Based on the anti-monotones property of the quality of the data sets with their sizes, the algorithm starts from the data sets containing of every two rows and every two columns of the data matrix, and gets the final biclusters by gradually adding columns and rows on the data sets. Both the theory analysis and experimental results show our algorithm has superiority our other similar algorithms in terms of processing speed and quality of clustering and efficiency.

The research work involving this paper applies the technique of parallel processing on bioinformatic research, all the parallel algorithms are coded using MPI (C bonding) and tested on parallel computer Shenteng-1800. All the experiments use benchmark data sets randomly selected from the standard bioinformatics data base. Experimental results show our algorithms can not only get higher processing speed, but also higher quality of results. It demonstrates the strong computational ability of parallel computers in applications for bioinformatic problems. The development of efficient parallel algorithms for the boinformatic problems will greatly promote the research of bioinformatics.

# 第一章 引　言

## 1.1 生物信息学

当前人类基因组研究已进入信息提取和数据分析阶段，这是基因组研究的转折点和关键时刻，即生物信息学[1]发挥重要作用的阶段。到1999年12月15日发布的第115 版为止，GenBank 中的DNA 碱基数目已达46亿5千万，DNA序列数目达到535万；其中EST序列超过339万条；UniGene的数目已达到7万个;已有25个模式生物的完整基因组被测序完成，另外的70个模式生物基因组正在测序当中；到2000年1月28 日为止，人类基因组已有16%的序列完成测定，另外37.7%的序列已经初步完成；同时功能基因组和蛋白质组的大量数据已开始涌现。如何分析这些数据，从中获得生物结构、功能的相关信息是基因组研究取得成果的决定性步骤。

生物信息学是在此背景下发展起来的综合运用生物学、数学、物理学、信息科学以及计算机科学等诸多学科的理论方法的崭新交叉学科。生物信息学是内涵非常丰富的学科，其核心是基因组信息学，包括基因组信息的获取、处理、存储、分配和解释。基因组信息学的关键是"读懂"基因组的核苷酸顺序，即全部基因在染色体上的确切位置以及各DNA片段的功能；同时在发现了新基因信息之后进行蛋白质空间结构模拟和预测，然后依据特定蛋白质的功能进行药物设计。了解基因表达的调控机理也是生物信息学的重要内容，根据生物分子在基因调控中的作用，描述人类疾病的诊断、治疗内在规律。它的研究目标是揭示基因组信息结构的复杂性及遗传语言的根本规律，解释生命的遗传语言。生物信息学已成为整个生命科学发展的重要组成部分，成为生命科学研究的前沿。

在国外，生物信息学的研究起步较早。美国在 20 世纪 60 年代就开始建立用手工搜索的蛋白质数据库。1979 年美国洛斯阿拉莫斯国家实验室开始建立核酸序列数据库 GenBank，现在由 1988 年成立的美国国家生物信息中心(NCBI)管理和维护。1982 年欧洲分子生物学实验室的 EMBL 数据库开始提供服务，随后又建立了欧洲分子生物学网(EMBNet)。1994 年开始 EMBL 数据库由建在英国剑桥的欧洲生物信息研究所(EBI)管理。1984 年日本着手建立国家级核酸数据库 DDBJ, 1987 年正式对外服务。目前绝大部分核酸和蛋白质数据由美国，欧洲和日本三家产生，以

上三家共同组成了 DDBJ/EMBL/GenBank 国际核酸序列数据库，24 小时交换数据，同步更新[2-8]。其他国家如德国，法国，意大利，澳大利亚，丹麦，以色列等，在分享网络资源的同时，还纷纷建立自己的生物信息中心，为本国的科研服务。

我国对生物信息学的研究始于 20 世纪末,但已显露出蓬勃发展的势头,许多科研单位已经开始或准备开始从事这方面的研究工作。1999 年 3 月，清华大学生物信息研究所，国家人类基因组北方研究中心和北京生物技术和新医药产业促进中心共同举办了“北方生物信息学学术研讨会”。1999 年 4 月，北京大学举办了“国家生物信息学讲习班”。2000 年 11 月，中国科学院和华大基因中心举办了“北京生物信息学研讨会”。目前北京大学生物信息中心建立了 EMBL 70 多种分子生物信息镜像系统和数据库，并提供数据检索服务,有些数据库可以每日更新。在复旦大学遗传学研究所，为克隆新基因而建立的一整套生物信息系统也已初具规模。中科院上海生化所、生物物理等在结构生物学和基因预测研究方面也有相当的基础。中科院计算所作为我国计算机科学的顶尖机构，利用自身优势，也开始在生物信息方面投入大量的人力物力，从事相关的研究。中国科学院上海生命科学研究所建立了我国核酸序列公共数据库。广州中山大学生物信息中心开通了法国巴斯德亚洲信息网。总的来说，虽然国内在生物信息学上的研究尚处于起步阶段，但我们有理由相信，我国的生物信息学在 21 世纪会有巨大的飞跃。

生物信息学的研究重点是从核酸和蛋白质序列出发，分析序列中表达的结构和功能的生物信息[9]。从信息学的角度来看，生物分子是生物信息的载体，如 DNA 核苷酸序列对蛋白质氨基酸序列进行编码，蛋白质序列决定蛋白质结构，而蛋白质结构又决定蛋白质的功能。归根到底，DNA 序列包含了最基本的生物信息。生物信息学是生命科学和自然科学研究的重大前沿领域之一，它在人类疾病基因发现、基因与蛋白质的表达与功能研究、合理化药物设计等方面都有着关键的作用。

### 1.1.1 DNA

DNA 是安全地构建在我们细胞中的只读信息，它只能被读取，不能被写入。而基因是 DNA 分子上具有遗传效应的特定核苷酸序列的总称，基因信息就储存在 DNA 中[10-13]。

DNA 是由脱氧核苷酸组成的双链，两条链缠绕在一起形成双螺旋结构，螺旋中的两条链平行且方向相反，我们称其中一条链的方向为 5′→3′，而另一条链的方向为 3′→5′。

DNA 的单链是由重复的基本单元—脱氧核苷酸组成的骨架，脱氧核苷酸由一个称为脱氧核糖的糖分子和磷酸、碱基组成。与骨架中碳原子相连的分子为碱基（base），如图 1.1 所示。不同的碱基决定了不同的核苷酸。在 DNA 分子中包含有 4 种碱基，分别是腺嘌呤、鸟嘌呤、胞嘧啶和胸腺嘧啶，分别用字母 A、G、C、T 表示。

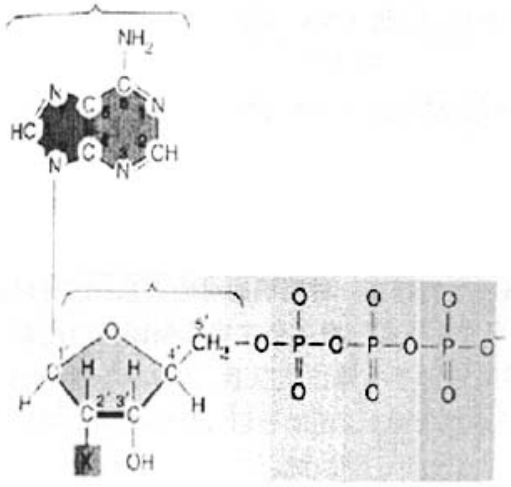


图 1.1 核苷酸结构

DNA 分子是双链结构，两条链沿着同一根轴平行盘绕，形成右手双螺旋结构。双链结合的机制是一条链的碱基与另一条链的碱基配对，碱基 A 始终与碱基 T 配对，碱基 C 始终与碱基 G 配对，因此将它们称为互补碱基对，它是特异的和稳定的，其结构模型见图 1.2。我们可以将 DNA 视为字符序列，每一个字符代表一个碱基，将一串字符置于另一串字符之上来表示双链 DNA。

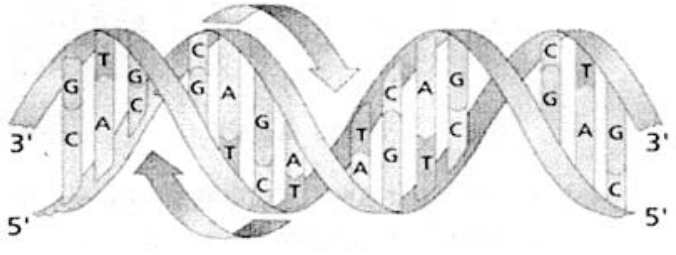


图 1.2 DNA 双螺旋结构模型

组成 DNA 分子的碱基虽然只有四种，它们的配对方式也只有 A 与 T，C 与 G 两种，但是，由于碱基可以任何顺序排列，构成了 DNA 分子的多样性。例如，

某 DNA 分子的一条多核苷酸链有 100 个不同的碱基组成，它们的可能排列方式就是 $4^{100}$。

上文提到的碱基配对的特异性是 DNA 精确复制的基础。细胞分裂时，通过 DNA 准确地自我复制（self-replication），亲代细胞所含的遗传信息就原原本本地传送到子代细胞。DNA 在复制过程中碱基间的氢键断裂，双链解开，以每条链分别作为模板合成新链。因此 DNA 双螺旋结构模型对遗传的分子机理产生了深远的影响。

DNA 是遗传信息的载体，DNA 序列上存储有蛋白质氨基酸序列的编码信息、基因表达调控的信息以及遗传信息，所以，DNA 序列包含着最基本的生命信息。遗传信息的载体主要是 DNA(少数情况下核糖核酸 RNA 也可以充当遗传信息载体)，控制生物体性状的基因实际上是一系列 DNA 片段。基因控制着蛋白质的合成，基因序列和蛋白质序列存在一种明确的对应关系，这种对应关系称为“遗传密码”。

### 1.1.2 蛋白质

蛋白质是构成生物体的最直接的元素，生物体之间的差异是直接由蛋白质的不同所造成的。蛋白质几乎参与所有的生命活动，生物体的生长、发育、繁殖、遗传等生命活动都离不开蛋白质，它是各种生命活动的物质基础。蛋白质包括很多种，结构蛋白是组织的构成单元，酶是化学反应的催化剂，蛋白质的其他功能还包括氧气运输和抗体防御等。

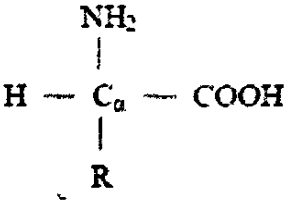



图 1.3 氨基酸通式

各种蛋白质水解后都产生氨基酸，所以氨基酸（amino acid, aa）是组成蛋白质的基本单位。每个氨基酸有 1 个中心碳原子，记为 C，连接 1 个氢原子(H)，1 个氨基(NH2)，1 个羧基(COOH)和 1 个侧链(R)，正是侧链决定了氨基酸间的差异。侧链可以是简单的氢原子，也可以是复杂的两个碳环，如图 1.3 所示。

自然界存在的绝大多数蛋白质分子中的氨基酸有 20 种，这些氨基酸之间可以

相互形成化学键，构成一个以牢固的氨基酸链为基础的复杂的三维结构体，即成为蛋白质分子。表 1.1 列出了最常见的 20 种氨基酸：

表 1.1 氨基酸代码

| 符号 | 意义 | 符号 | 意义 | 符号 | 意义 |
|---|---|---|---|---|---|
| A | 丙氨酸 | I | 异亮氨酸 | R | 精氨酸 |
| C | 半胱氨酸 | K | 赖氨酸 | S | 丝氨酸 |
| D | 天冬氨酸( | L | 亮氨酸 | T | 苏氨酸 |
| E | 谷氨酸 | M | 甲硫氨酸 | V | 缬氨酸 |
| F | 苯丙氨酸 | N | 天冬酰胺 | W | 色氨酸 |
| G | 甘氨酸 | P | 脯氨酸 | Y | 酪氨酸 |
| H | 组氨酸 | Q | 谷氨酰胺 | X | 任意氨酸 |

因此，我们可以将蛋白质同样看作字符序列，每一个字符代表一个氨基酸。这种字符序列称为蛋白质的一级结构，可是蛋白质并不仅仅是氨基酸分子的线性序列，实际上蛋白质在三维空间中折叠，形成如图 1.4 所示的二级、三级和四级结构。

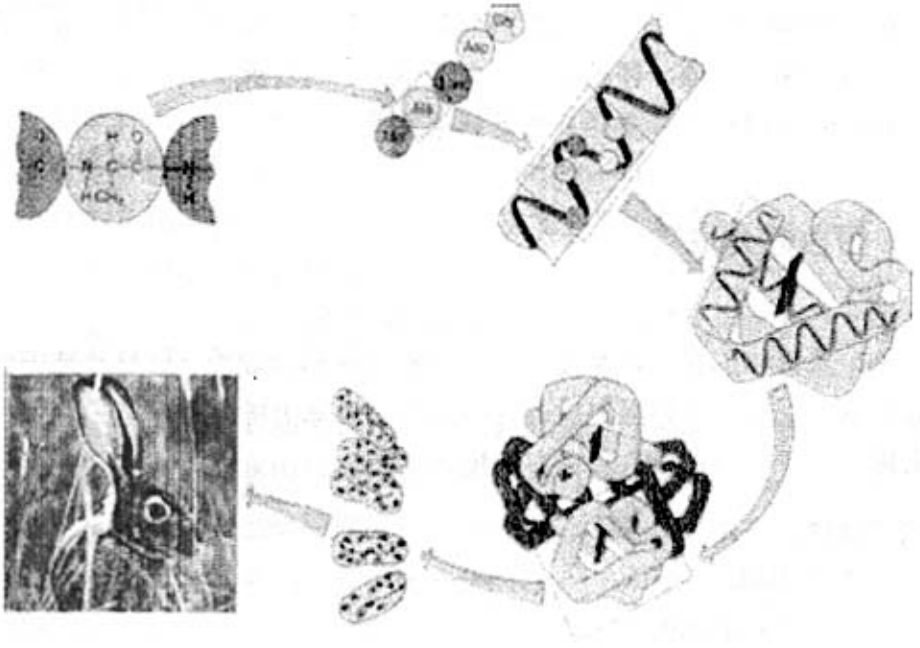


图 1.4 蛋白质的各级结构

蛋白质分子在生物体内执行着各项重要任务，如生物化学反应的催化、营养的输运、抗体防御、信号的识别与传递等。要了解蛋白质的功能必须首先分析蛋白质的结构，因为蛋白质功能取决于蛋白质的空间结构，而蛋白质的空间结构取决于蛋

白质序列，蛋白质的结构信息隐含在蛋白质序列之中。

### 1.1.3 RNA

RNA（核糖核酸）也是有核苷酸合成的链式分子，在化学结构上与 DNA 有所不同，RNA 的核苷酸由磷酸、戊糖和碱基组成，但 RNA 的碱基是腺嘌呤（A）、鸟嘌呤（G）、包嘧啶（C）和尿嘧啶（U）。RNA 分为信使 RNA、核糖体 RNA 和转运 RNA 等。各类 RNA 分子中与遗传信息传递关系密切的是 mRNA。mRNA 经过剪切修饰后，即可作为合成蛋白质的模版。RNA 在细胞外不稳定，一般以 mRNA 为模版，反转录得到互补的 cDNA 以及双链 DNA，进行基因克隆和测序工作，进而得到其序列。

RNA 与 DNA 非常类似，但是在 RNA 的 4 种碱基中使用 U 代替 DNA 中的 T，RNA 中的核糖代替了 DNA 的脱氧核糖。

RNA 同时具有某些 DNA 和蛋白质的特性。因为和 DNA 一样由核苷酸序列组成，它与 DNA 具有相同的信息存储能力。另一方面，RNA 能形成三维结构的能力使得它具有和蛋白质一样的特性。由于 RNA 的双重功能，人们猜想，生命可能起源于 RNA，而 DNA 和蛋白质都是后来进化而来的。

### 1.1.4 蛋白质合成

每个有机体的蛋白质都是由一部分基因编码合成的。信息从基因的核苷酸序列中被提取出，用来指导蛋白质合成的过程对地球上的所有生物是相同的，分子生物学家称之为“中心法则”如图1.5 所示。

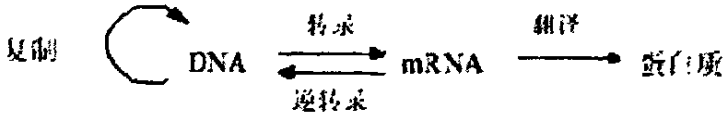



图1.5 分子生物学的中心法则

DNA 编码合成蛋白质的过程包括了“转录”和“翻译”两个重要步骤。为了表达 DNA 上的基因，mRNA 读取 DNA 上的遗传信息，这个过程称作转录，RNA 聚合酶能够催化这个反应。DNA 上的遗传信息有内含子和外显子，所以最初转录

形成的RNA需分裂以除去内含子，除去内含子的成熟的mRNA再翻译合成蛋白质。

由此可见，一切的物种核酸或蛋白质序列看作由 4 个或 20 个元素组成的字母表中选出的字母序列，如：{ATGTCCAACG}，{GSSKYPRETT}分别表示一条核酸序列和一条蛋白质序列。生物信息就是成千上万条以字符序列形式存储核酸或蛋白质序列，并以某些特定格式存放在各类生物数据库中。DNA 上核酸的特定序列决定了生物体结构和功能(包括蛋白质的种类、结构和功能)，并以其半保留复制机制，保证世世代代准确地传递下去。

蛋白质结构和功能都是由核酸根据三联体密码决定的，并在细胞内合成，它参与生物的一切生命活动。因此，蛋白质决定了一个生物是什么和做什么，核酸则负责编码产生蛋白质所必要的信息，并把这种信息传给后代。

作为信息的载体，DNA序列和蛋白质序列都包含有进化信息。通过对相似蛋白质序列的比较，可以发现分子进化的过程；通过对不同种属的同源蛋白质序列的比较，可以分析蛋白质之间的种系发生关系，推测它们共同的祖先。

### 1.1.5 生物序列比对

有了完整基因组，人类对自身的认识就更为细致、更为精确，但是测序基因组后，还必须对各种核酸和蛋白质序列进行序列分析，目的是了解这个序列在生物体中充当了怎样的角色。例如，DNA序列中的重复片段、编码区、启动子、内含子、外显子、转录调控因子、结合位点等。

我们可以通过对序列之间相似性比较来推断不同物种之间的进化关系。如果两个序列具有足够的相似性，则可以认为两者具有同源性，那么它们的生物性状会存在很大的相似性，如果我们知道其中一个物种的基因序列所决定的生物功能信息，就可以推断另一个物种的基因序列所决定的生物功能信息。

因此，将未知序列同已知序列进行比较分析，进而了解未知序列的生物信息的方法已成为一种强有力的研究手段，这使得我们可以从核酸以及氨基酸的层次上去分析序列的相同点和不同点，从而推测它们的结构、功能以及进化上的关系。最常用的比较方法是序列比对。

序列比对的目的是求出给定的序列对的之间距离，从而为诸如DNA分类，聚类，蛋白质的二级结构预测和生物进化树的创建打下基础，提供了一个相似性度量的基本工具[14]。最常见的比对是蛋白质序列之间或核酸序列之间的两两比对，通过比较两个序列之间的相似区域和保守性位点，可以寻找二者可能的分子进化关系。进一

步的比对是将多个蛋白质或核酸同时进行比较，寻找这些有进化关系的序列之间共同的保守区域、位点和特征，从而探索导致它们产生共同功能的序列模式。此外，还可以把蛋白质序列与核酸序列相比来探索核酸序列可能的表达框架，把蛋白质序列与具有三维结构信息的蛋白质相比，从而获得蛋白质折叠类型的信息。

实际上序列比对就是运用某种特定的数学模型或算法，找出两个或多个序列之间的最大匹配碱基或残基数。早期的序列比对是全局的序列比较，但由于蛋白质具有的模块性质，可能由于外显子的交换而产生新蛋白质，因此局部比对会更加合理。通常用打分矩阵描述序列两两比对，两条序列分别作为矩阵的两维，矩阵点记录两个维上对应的两个残基的相似性分数，分数越高则说明两个残基越相似。因此，序列比对问题变成在矩阵里寻找最佳比对路径，目前最有效的方法是Needleman-Wunch动态规划算法，在此基础上又改良产生了 Smith-Waterman算法和SIM算法。

从一次性参加比对的序列的数目考虑，序列比对可分为双序列比对和多序列比对。双序列比对是为了找出两个序列之间的最大相似性匹配，用于对两条序列进行同源性分析，是多序列比对和数据库搜索的基础。动态规划算法是最为经典的双序列比对算法。

多序列比对可以用来区分一组序列之间的差异，或者描述一组序列之间的相似性关系，以便了解一个基因家族的共同特征，以及定量估计序列间的关系，由此推断它们在进化中的亲缘关系。例如，某些在生物学上有重要意义的相似性只能通过将多个序列对比排列起来才能识别。同样，只有在多序列比对之后，才能发现与结构域或功能相关的保守序列片段。对于一系列同源蛋白质，人们希望研究隐含在蛋白质序列中的系统发育的关系，以便更好地理解这些蛋白质的进化。在实际研究中，生物学家并不是仅仅分析单个蛋白质，而是更着重于研究蛋白质之间的关系，研究一个家族中的相关蛋白质，研究相关蛋白质序列中的保守区域，进而分析蛋白质的结构和功能。序列两两比对往往不能满足这样的需要，难以发现多个序列的共性，必须同时比对多条同源序列。但是，多序列比对的计算量非常大，传统的动态规划算法在三个以上的序列比对当中很难实现，一般采用渐进式算法或迭代算法。

序列比对的基础是找出序列间的最长公共子序列。最长公共子序列((Longest Conunon Subsequence)LCS)是将两个给定字符串分别删去零个或多个字符后得到的长度最长的相同字符序列。例如，字符串 abcabcabb 与 bcacacbb 的最长公共子序列为 bcacabb。在人类基因组计划的快速进展过程中，DNA 和蛋白质序列数据库的规模正呈指数增加[15]。伴随着序列数据库的增长，三维结构数据库也在不断增长，于是，在精确度不受影响的前提下，如何提高序列 LCS 问题的处理速度和效率成了一

项重要的课题。

目前人们对于最长公共子序列问题，已经作了大量的研究工作。著名的 LCS 算法 Smith－Waterman 算法于 1981 年提出，它由 Needleman－Wunsch 算法演化而来，该算法能保证解的正确性。Aho[16]等人证明了最长公共子序列问题的串行算法的时间复杂度的下界为 $O(mn)$[17]，使用动态规划设计的算法可以达到时间和空间的复杂度为 $O(mn)$，Mayers 和 Miller[18]使用 Hirschberg[19]提出的技巧在时间复杂度不变的前提下将空间约减到 $O(m+n)$。研究表明，若对该问题采用并行化的算法[20-22]，可以大大地加快问题求解速度。所以，近年来对该问题的并行化的研究也引起研究者的注意，在 CREW－PARM 模型上，Aggarwal[23] 和 Apostolico 等人[24]独立地提出了一个 $O(\log m \log n)$ 时间，使用 $\frac{mn}{\log m}$ 个处理机，Mi Lu 等人[25]设计了两个并行算法：一个使用 $\frac{mn}{\log m}$ 台处理机，时间复杂度为 $O(\log^2 n+\log m)$，另一个使用 $\frac{mn}{\log^2 m \log\log m}$ 台处理机，时间复杂度为 $O(\log^2 m \log\log m)$；在 CRCW-PRAM 上 Apostolico 等人[24]给出了时间为 $O(\log n(\log\log m)^2)$ 的使用 $\frac{mn}{\log\log m}$ 个处理机的算法。在 Systolic 阵列上，Robert 等[26]使用 $m(m+1)$ 个单元，计算时间为 $n+5m$；Chang 等[27]使用 $mn$ 个单元，计算时间为 $4n+2m$；Luce 等[28]使用 $\frac{m(m+1)}{2}$ 个单元，计算时间为 $n+3m+q$；Freschi 等[29]基于 Run-length-encoded 给出了使用 $M+N$ 个单元，计算时间为 $O(m+n)$；这里有 $m \le n$，$q$ 为最长公共子序列的长度，$M$ 和 $N$ 分别为两个序列在 Run-length-encoded 的编码长度。

对于多序列的最长公共子串问题，其复杂程度更高。虽然 Smith-Waterman 算法可以对两个序列找到最优解，但如果将这个算法直接延伸应用到多序列最长公共子串的问题上来，时空开销将达到 $O(2^N-1)(\prod_{i=1}^{N}|S_i|)$，其中 $N$ 为序列的个数，$|S_i|$ 为第 $i$ 个序列的长度。所以算法虽然精确且简单，但在多重序列比对问题中却不实用。尽管后来很多人对此算法作了改进，如基于 Carrillo 和 Lipman 的算法[30]的 MSA[31]，基于分治策略和 MSA 的 DCA[32]，基于优化 DCA 算法的 OMA[33]等等，多序列比对的启发式计算方法通常在适度牺牲正确性的基础上来提高计算效率。ClustalW[34]是应用最为广泛的多重序列比对软件，它是高效的渐进方法的代表。ClustalW 对 Feng 和 Doolittle[35]提出的算法作了一系列的改进，使得结果的正确性

得到进一步提高，所以在生物信息学中得到了广泛的应用。可是这些算法处理的序列长度和数量仍受到很大限制。

### 1.1.6 序列拼接

基因组计划的目标是获得所研究的生物的全基因组序列，而序列拼接是基因组测序阶段生物信息学研究的最基本、最重要的问题。众所周知，生物的基因组是指该生物所有遗传物质的总和，绝大部分基因组由DNA(脱氧核糖核酸)组成。DNA是由核苷酸单体构成的线性、无分支的多聚分子。核苷酸由碱基区分，DNA中，碱基分别是腺嘌呤(Adenine)、胞嘧啶(Cytosine)、鸟嘌呤(Guanine)和胸腺嘧啶(Thymine)，分别用字母A, C, G, T表示。基因组测序就是要确定DNA分子的碱基序列。

对于完整基因组自上而下的测序过程一般包括三个步骤：（1）建立克隆的物理图谱，如酵母人工染色体YAC（Yeast Artificial Chromosome）克隆、细菌人工染色体BAC（Bacterial Artificial Chromosome）克隆等；（2）利用鸟枪法（Shotgun Strategy）测定每个克隆的序列；（3）注释。当得到一段DNA 序列之后，可以利用序列分析工具，通过与数据库序列的比较，得到与该序列相关的信息，如基因、调控元件、重复区域等，进而对序列的生物学特性进行注释。人类基因组计划（HGP）采用的就是这种策略。Venter 提出的战略构想正好与目前的人类基因组计划相反，即首先是测序，然后才是在测序的基础上作图。Venter 把这种战略称为“全基因组随机测序”也称为“全基因组鸟枪战略”(whole genome shotgun strategy)。

在大规模DNA 测序中，目标DNA分子的长度可达上百万个bp。现在还不能直接测定整个分子的序列，然而，可以得到待测序列的一系列序列片段。序列片段是DNA 双螺旋中的一条链的子序列（或子串）。这些序列片段覆盖待测序列，并且序列片段之间也存在着相互覆盖或者重叠。在一般情况下，对于一个特定的片段，我们不知道它是属于正向链还是属于反向链，也不知道该片段相对于起点的位置。另外，这样的序列片段中还可能隐含错误的信息。序列片段的长度范围300—1000 bp，而目标序列的长度范围是30000—1000000 bp，总的片段数目可达上千个。DNA序列片段拼接（sequence assembly，又称序列拼接）的任务就是根据基因组测序得到的上千万个小片段序列通过比对再正确拼接起来。如果能够得到DNA一条链的序列，那么根据互补原则，另一条链的序列也就得到了。当前世界范围内的主要测序中心以及重要的测序工程都普遍采用了鸟枪(Shotgun)测序法。它根据目前可以用测序仪直接测出序列的长度水平，将较长的DNA序列的多条克隆随机打断成很短的片

断，再通过测序仪精确地将这些小的片断序列一一测出，最后根据这些小片断序列间的重叠关系用计算机将它们进行拼接，以期得到目标序列的一个或多个较长的连续段。

目前用于各种大型测序工程中的拼接软件很多，最广泛使用的是美国Washington大学的Phil Green实验室开发的Phred-Phrap-Consed软件(Ewing et al.,1998; Ewing and Green, 1998; gordon et al.,1998)[36]。该软件覆盖了基因组测序的全过程，为基因组测序提供完整的计算机解决方案。在它的协助下，包括HGP等几十个规模不同的测序工程都完成了最后的片段拼接。这套软件不仅适用于大规模测序，也适用于一般实验室应用。Phil Green小组一直在不断地完善和改进该套系统。

美国基因组研究所(The Institute of Genome Research)于1995年前后研制了TIGR 拼接软件-TIGR ASSEMBLER TIGR EDITOR[37]，他们试图通过简化拼接过程中的序列比对( sequence alignment)来节约时间，取得了一定的效果。

2001年7月，California大学的Pavel A. Pevzner等[38]将序列拼接问题转化为一个寻找Euler路径的问题，并在此基础上研制了名为EULER的拼接软件。

此外，还有很多其他比较有特点的拼接软件，如GAP[38-39]、CAP3/4[40-41]、Seqman II[38]、SLIC[38][42]等，某些在实际的测序工程中也得到了应用，但范围都不是很广泛。

国内在片段拼接方面的研究主要开始于近几年，目前主要集中在北京大学和中科院北京华人基因研究中心，后者承担和完成了HGP国际大合作中1%的测序工作并在国际上首次独立进行在中国广泛种植的水稻( Oryza sative L. ssp. Indica)全基因组的测序拼接工作，并取得成功。

近年来，为了进一步提高拼接速度，人们开始研究序列拼接的并行处理，这方面国外较著名的是SPSOFT[43](Southwest Parallel Software)(http://www.spsoft.com)，它实现了Phrap程序的SMP机并行版本，主要是利用多线程技术加速其中具有并行性的部分，获得较好的效率，最近，它完成了Linux环境下的并行Phrap。在国内，中科院计算技术研究所与华大基因研究中心合作，基于曙光3000超级计算机系统，开发了Phrap的并行算法，实现了Phrap的并行化[44]。

总之，目前关于序列拼接问题的研究可以归为三点:第一，序列拼接问题的研究集中在提高拼接速度和准确度上。第二，就提高序列拼接速度问题而言，单机上的拼接算法研究已经相当广泛深刻，机群系统上的拼接算法研究刚刚开始。第三，测序所得原始数据的出现是随机的且总体上的相关性比较大，而现有拼接算法的计算局部性较差，并行难度大，并且研究下处于起步阶段。

### 1.1.7 基因表达数据的双向聚类

DNA微阵列技术是继DNA重组技术、PCR扩增技术之后的又一重大生物技术。基于微阵列实验可以测定出。某一特定生物过程(如细胞分裂周期的不同时期)中所有基因表达情况,或不同样本(如来自不同组织病理诊断的肿瘤标本的细胞)中的所有基因表达情况[45-48]。这些基因表达数据中蕴含着基因活动的信息，可以反映细胞当前的生理状态。例如细胞是处于正常还是恶化状态、药物对肿瘤细胞是否有效等。基因表达数据的分析可以获取基因功能和基因表达调控信息。

基因表达数据主要来自于两个方面，一是基因芯片，这是最主要的表达数据来源,利用基因芯片技术可以大规模并行获取基因转录结果mRNA 的数据(Schena Eet al，1995）。表达系列分析SAGE 和差异显示（Kozian and Kirschbaum，1999）、蛋白质芯片等是快速检测蛋白质及其含量的另一类技术。

聚类分析是模式识别中一种非常有吸引力的方法，特别适用于模式分类数不知道的情况。从机器学习的角度来看，有两种基本的聚类分析（Kaufman 1990），即所谓有教师聚类和无教师聚类。在有教师聚类中，对于每一类有一个参考模式，对于一个未分类的向量，通过计算选择一个最接近的参考模式，并将该向量归入该参考模式所对应的类，这实际上是一个分类问题。而真正的聚类分析是一种无教师学习（或无监督学习），没有关于聚类的先验知识，需要聚类算法根据样本之间的距离或者相似程度进行自动分类。

基因表达数据聚类分析一般包括以下几个步骤：（1）确定基因表达的数据；（2）计算相似性矩阵，各个矩阵元素代表两个基因的表达是否相似；（3）选择算法进行聚类分析；（4）显示分析结果。

在一种基因芯片上往往含有成百上千个基因探针，一次可以同时检测大量基因的表达。利用同一种芯片在不同条件下（不同时间，不同细胞，不同外界作用）进行基因表达实验，搜集表达数据，将原始数据放在一起，形成一个数据表格。表格的每一行代表一个基因，是一个基因在不同实验条件下表达的“快照”，而每一列则代表各个基因在同一种实验条件下的表达水平。从数学形式上来看，表格的一行数据就是一个向量，常称其为一个基因的表达模式，而表格本身就相当于一个矩阵。聚类分析就是将这些向量按照相似程度进行归类。

对数据进行聚类分析之前，必须将包含在基因表达矩阵中的数据进行相似程度分析，并且对分析结果进行量化。通常情况下，相似往往被赋于一个较大的量化的值，而不相似则由一个较小的量化的值来表示。在实际计算中，往往以距离代替相

似的概念，相似性度量被转化为两个基因表达模式之间的距离。距离越小，表达模式越相近，反之，则表达模式差异大。所有的距离值的集合可以构成一个距离矩阵。有两种方式计算距离矩阵：（1）表达矩阵的任意两行数据之间的距离可用以确定具有相同表达方式的基因簇；（2）表达矩阵的任意两列数据之间的距离可以确定基因表达实验条件的差别。

目前常用的聚类方法是基于所有属性比较的聚类，用相似度量函数确定相似程度，将对象进行类别的划分。在这些聚类中，把对象分成了若干类，但对有些情况不能满足，例如：某些属性对有些对象不起作用，在这些情况下，需要考虑与对象相关的属性。如果将对象在不同属性下的取值看作一个矩阵，基于此矩阵，根据对象和属性同时聚类，这样可以找出其中满足条件的各个小矩阵，即由对象子集和属性子集组成的聚类，这个过程称为双向聚类[37-42]。在对基因表达矩阵的双向聚类中，就是找出这样的一组基因及试验条件的子集，使得这组基因在这些试验条件具有相同的变化趋势。Hartigan在1973年最早的介绍了"同时聚类"这个概念[46]，之后，Mirkin在文献[47]提出了类似的概念，比如："盒式聚类"，"双向聚类"的概念。文献[49]对基因表达数据的聚类中引用了双聚类概念，从原始矩阵中依次产生满足条件的*δ-bicluster*，从中分析相似的基因表达行为，用参数H判定是否为聚类，采用添加/删除行/列的方式求出一个聚类，按照同样的方法得到*k*个聚类。文献[50]的判定方法跟文献[49]类似，在删除的时候考虑行的平均值与所有元素的平均值是否大于给定的阈值。文献[51]对双向聚类的算法进一步改进，提出FLOC算法，第一阶段随机产生给定数目的聚类，第二阶段再对这些随机产生的*k*个聚类改进为满足条件的聚类。添加/删除任意的行/列，通过计算新的聚类的行列相乘得到的偏差值，判断是否满足条件阈值，从而得到初始聚类的扩展聚类。但FLOC算法首先产生的初始聚类是随机的，很不精确；其次反复的偏差值的计算以及迭代才能找到近似的各个满足条件要求的聚类，这样既影响了准确率又影响了效率。

目前已有的双向聚类的算法有如下几类：1. 行列交叉叠代方法：如 G. Getz 等人[52] 的 Coupled Two-Way Clustering 方法(CTWC)，Chun Tang 等人[53] 的 Interrelated Two-Way Clustering (ITWC) 方法。这类方法以所有行、列对的聚类为基础，逐步分层聚类，不断识别"稳定"的行/列聚类，直到取得满意的结果。2. 分治策略方法：如 Hartigan [54]等人 的 Block Clustering。 该方法首先将行和列按照它们的均值排序，将其中最好的行或列分解，以降低块间的落差。重复这样行列分解过程，直到取得满意的 $K$ 个块。 这种方法速度快，但如果初始的分解如果不恰当，有可能丢失好的聚类。3. 贪心叠代搜索方法：如 Cheung & Church 等人的 $\delta$-bicluster 方法和 Jiong Yang 等人[51][55]的 FLOC 方法、Yuval Klugar 等人的 Spectral

方法[56]、Amir Ben-Dor等人[57]的*OPSMs*方法等。4. 穷举方法，如文献[58]提出基于模式的新型聚类模型Pcluster、Sungroh Yoon[59]也是用相似的方法求取聚类、如Tanay[60]等人的Statistical-Algorithmic Method for Bicluster Analysis ( SAMBA)算法。5. 分布参数确认（Distribution Parameter Identification）法：如Lazzeroni and Owen 等人的*Plaid Models*方法[61]以及Qizheng Sheng,等人的*Gibbs*方法[62]、Eran Segal等人的*PRMs*方法[63-64]等。

## 1.2 并行处理和并行算法

随着科学技术的快速发展，对计算机的处理速度提出了愈来愈高的要求。诸如天气预报、地震分析以及生物信息学中的一系列问题等富有挑战性的应用问题，使得人们需要比串行计算机所能提供的能力更强的计算能力。克服这种限制的一种方法是增加处理器和其他部件的运算速度，以使他们能提供应用所需的更强大的计算能力。然而计算机单机技术的发展受着诸如光速与物理尺寸的限制，正是这种计算机单机技术发展的有限性与科学工程计算的无限性之间的矛盾决定了计算机发展必然走上多机并行的道路。由于采用大规模的并行，使得计算机的计算速度大大提高，反过来，科学技术需要求解的课题规模越大、问题越复杂，并行的潜力越大，这也要求高性能计算机走并行的道路。 随着计算机技术和大规模集成电路技术的发展,近十年来,并行处理技术得到了迅速的发展。一方面，人们在不断研究、探索并行处理的新技术，不断研制出性能价格比高的并行计算机；另一方面，又力求使各项并行处理技术得以应用，为解决国民经济、国防建设和科技发展中的重大、复杂问题提供有力的工具。

### 1.2.1 并行计算机

并行计算机是指由两个或两个以上的处理机连接起来可以并发操作的计算机。这样的计算机有时也称为多处理机。并行计算机主要可分为两大类：（1）SIMD（单指令流多数据流）并行机，各个处理机在同一时刻执行相同的指令，但数据不同；（2）MIMD（多指令流多数据流）并行机，各个处理机在同一时刻可执行不同的指令，处理的数据也可以不同。MIMD多处理机又可以分为：共享存储多处理机、分布

式存储多处理机和分布共享存储并行机。

当今的并行计算机主要以MIMD多处理机为主，其中有SMP（共享存储多机系统）和MPP（大规模并行处理系统）外，还有向量并行机（PVP，或称VPP），以及工作站机群（NOW，或称COW）。PVP与SMP的主要不同是CPU，后者是标准的RISC芯片，而前者是各个厂家自行研制的向量处理机。PVP承袭了向量机的优势，技术成熟、效率高，可将互连网络的路由器插入其中，方便地进行各种优化组合设计。但是，PVP的向量CPU与已经形成上千万生产规模的RISC相比，在规模效应和性价比上相差很大。另外，RISC的生命力在于不断吸收超级计算机领域的最新技术，目前的目标之一就是增加向量处理的功能。如果能达到这个目标，SMP与PVP就完全统一了。此外，工作站机群NOW是一种新兴的并行计算机的类别，具有“单一系统形象（Single System Image）”的工作站群集，特别是同构的，其本质与MPP没有差别。

不同的并行计算机各有特点，但它们也有区别于其它计算机的共性。其中最重要的就是可扩展性（Scalability）和可编程性（Programmability）这一对共生而又矛盾的特性。可扩展性是并行计算机最大的优势，可简单定义为“在确定的应用背景下，计算机系统的性能要随处理机数的增加而线性增长”。可扩展性包括规模可扩展、时间可扩展和问题可扩展几个方面。规模可扩展的要点是均衡，均衡的目的是防止瓶颈的发生，“三T”表达了当今均衡的指标。时间可扩展也称换代可扩展，主要指体系不受限于芯片、器件、工艺等。问题可扩展指格点增加时，系统能适应问题规模的扩大；而当问题的粒度加大时，效率能相应提高。现有的MPP，如Inetl的Paragon、IBM的SP/2、国内的曙光4000A，一般认为是可以扩展的。

可编程性是在并行机发展的过程中，伴随可扩展性而产生的新概念。并行计算机有共享与分布式两种存储结构，所以操作系统进程间的通信（IPC）也有两种不同的机制：共享变量和信息传递。机器结构和操作系统的这些区别又影响到支撑软件和应用软件的编程模式。SMP的程序设计仍以传统的高级语言为基础，系统提供自动并行识别或增加并行语言成分。而MPP必须建立另一种消息传递机制，如PVM、MPI等，在程序中显式地写出信息的发送和接收，这导致了应用软件编写的困难，给用户增加了很多负担。因此，并行计算机软件系统的开发，是并行计算机应用中重点要解决的问题。

### 1.2.2 并行算法

并行算法是一些可同时执行的进程的集合，这些进程相互作用和协调，以完成对一个问题的求解。设计并行算法的目的是让并行机的众多处理机充分发挥作用，以达到对问题的快速有效求解。并行算法对并行机的依赖性很强，在一台并行机上有效的算法在别的不同结构的并行机上可能效果并不好，因此，设计者必须了解并行机的结构才能设计出好的并行算法。

并行算法的目标是尽可能减少时间复杂性，通常这是通过增加空间复杂性(如增加空间的维数及增加处理器的台数)来实现的。所以，并行算法树采用截然不同的"浅而宽"结构，即每个时刻可容纳的计算量相应增加，使整个算法的步数尽可能减少，或者说，通过增加每个时刻步的算法复杂性来减少整体的时间复杂性。这样，就达到了把时间复杂性转化为空间复杂性的目的。

并行算法可以从不同的角度加以分类。从并行算法处理问题的类型不同可以划分为数值算法和非数值算法。数值并行算法主要为数值计算而设计的并行算法；非数值并行算法，如排序、归并、查找、图论问题，组合优化问题的算法以及为符号计算而设计的并行算法。

从并行算法规则所定义的并行运算之间的相互关系可以把并行算法分为同步算法和异步算法。同步并行算法（synchronized algorithm），是指某些进程必须等待其他进程的一种并行算法，要求所有进程必须在一个给定时刻同步。SIMD以及共享存储型MIMD并行机上通常运行同步并行算法；异步并行算法（asynchronized algorithm），是指诸进程执行相对独立、不要互相等待的一类算法。其主要特征是在计算的整个过程中都不需要等待，而是根据当前的最新信息决定进程的继续或终止。这种算法通常是针对分布式存储的MIMD并行机设计的。

根据设计和实现算法所依赖的并行机类型不同又可以分为SIMD算法、MIMD算法、分布式算法。分布式算法（distributed algorithm），是指由包括网络在内的通信链路连接的多结点机或计算机群协同完成某个计算任务的算法。

### 1.2.3 并行计算模型

所谓计算模型，是算法设计者进行理论分析时所依据的计算机模型。冯·诺依曼机是理想的串行计算模型。由于并行机在飞速发展之中，尚未定型，故目前尚没有所谓的通用并行计算模型。

使用并行计算机解决一个应用问题时，并行算法的设计是非常重要的。而并行算法最终要成为一个由程序实现的结构依赖的算法，就特别需要一个抽象的并行计算机结构作为研究高效的结构依赖性算法的基础，以保证并行算法适应于广泛的并行计算机结构，并能够依照抽象的结构分析并行算法的效率，以及指导与并行机结构相匹配的并行算法的设计。并行计算模型就是为并行算法的设计、分析而研究出的并行计算机的抽象结构。因此，并行计算模型并非某种具体的并行计算机，而是某一类并行计算机的抽象。从更广意义上来说，并行计算模型为并行计算提供了硬件和软件设计者的界面。在这一界面的约定下，并行系统的硬件设计者和软件设计者可以开发硬件结构和软件（包括算法、操作系统、语言和软件设计工具等），设计对并行性的支持机构，从而提高系统的性能。当前，人们将并行计算机的某一些特征抽象出来，形成了各种特定的并行计算理论模型，以便于并行算法的设计与理论分析。并行机的特征有：消息包的长度或延迟时间、消息包传递的开销、处理器连续传递消息的最小间隔（或通信的带宽）、处理器个数等。由诸如此类的参数构成各种特定的并行计算模型。

从并行计算机的发展史来看，并行计算机的体系结构多种多样，因此构造一个合理的并行计算机模型是很困难的。在过去的 30 多年中，人们对不同类型的并行计算机给出了多种并行计算模型。常用的并行计算模型有PRAM 模型及其改进模型、BSP 模型、VLSI模型和$C^3$模型等等。1993 年美国伯克利大学 David Culler 等人提出了 LogP 模型，这个模型比较适合 MPP 和机群系统。

### 1.2.4 并行算法的设计

并行算法的设计，不仅是提高并行计算机的使用效率的关键，而且往往能找到改进现有串行算法的新途径。并行算法的研究是研制高效并行计算机软件的基础。并行算法设计的可供选择的技术路线有两条：一条是在现有的串行算法基础上作并行化；另一条是直接从所要解决的问题出发，面向并行系统研制高效率的并行算法。

虽然并行算法研究还不是太成熟，但并行算法的设计依然是有章可循的，例如对非数值问题，划分法、平衡树法、倍增法/指针跳跃法、流水线法、破对称法等都是常用的设计并行算法的方法。对数值问题，人们还可以根据问题的特性采用“Divide and Conquer”（分而治之）的方法。

采用“Divide and Conquer”方法可以比较顺利地达到“负载平衡”及减少通信等要求，对于处理器较少的小规模并行，不失是一个有效的方法。然而随着处理器的

增多，对于整体有内在联系的计算问题，每个处理器中所包含的整体信息急剧减少，这时的并行度仍可提高，但整体效率却随之下降。经过研究和探索，人们对"Divide and Conquer"的方法提出了补充原则，目前对此没有统一的名称，有的学者称之为"Divide and Conquer +global Corrections"（分而治之+整体修正）。具体地说，一个高效率的中、大规模并行算法主要由两部分组成：即分而治之与整体修正（或全局控制），前者是并行程序执行的主要部分，占工作量的很大部分；后者是必要的信息交流，起到加速作用。尽管后者工作量占的比例很小，却是决定并行算法效率高低的关键部分。

设计一个高效的并行算法的过程比较复杂，很难归结为一个单一化的过程，往往几经反复才能达到要求。而设计出一个具有良好的并发性和可扩展性的并行算法，可分为以下几步,即任务划分（Partitioning）、通信(Communication)分析、任务组合（Agglomeration）和处理器映射(Mapping)，简称为 PCAM 设计过程，它是一种设计方法学，其基本要点是：首先尽量开拓算法的并发性和满足算法的可扩展性；然后着重优化算法的通信成本和全局执行时间，同时通过必要的整个过程的反复回溯，以期最终达到一个满意的设计选择。

### 1.2.5 并行算法的编程实现

并行算法最终总是要在并行计算机上实现的，实现一个并行算法就需要一个适当的编程环境，或者说，需要有一个并行程序设计语言及相应的编译、调试、性能分析工具等。针对并行系统，一般而言，程序并行性分为控制并行性和数据并行性。控制并行性是指多个不同操作可同时进行，数据并行性是指对不同数据同时执行同一操作。并行程序设计模型（Parallel Program Model）是一种程序抽象的集合，它给程序员提供了一幅计算机硬件/软件系统透明的简图，程序员利用这个模型就可以为向量处理机、多计算机和工作站机群等设计并行程序。目前主要有隐式并行、数据并行、消息传递和共享变量四种编程模型。

本论文所有的并行算法都是在国产并行计算机深腾 1800 上实现的。深腾 1800 是一个机群结构，采用的是消息传递的编程模型。在消息传递模型中，各个并行执行的进程之间通过消息传递来交换信息、协调步伐、控制执行。消息可以是指令、数据、同步信号或中断信号等。在消息传递的并行程序中，用户必须明确地为进程分配数据和负载，它比较适合开发大粒度的并行性，这些程序是多线程的和异步的，要求显示同步以确保正确地执行顺序。然而这些程序均有其分开的地址空间。消息

传递一般是面向分布式内存的，但是它也可适用于共享内存的并行机。消息传递模型比数据并行模型灵活，两种广泛使用的标准库 PVM 和 MPI 使消息传递程序大大地增强了可移植性。消息传递为编程者提供了更灵活的控制手段和表达并行的方法，一些用数据并行方法很难表达的并行算法，都可以用消息传递模型来实现。灵活性和控制手段的多样化，是消息传递并行程序能提供高的执行效率的重要原因。

本文实现并行算法使用的是 MPI(C 语言绑定)，它是 1994 年发布的一种消息传递接口，是目前最重要的并行编程工具。它实际上是一个消息传递函数库的标准说明，共有上百个函数调用接口。在 FORTRAN77 和 C 语言中可以对这些函数进行调用。MPI 是一个复杂的系统，它包含了 129 个函数（根据 1994 年发布的 MPI 标准）。事实上，1997 年修订的标准，称之为 MPI-2，已超过 200 个函数，目前最常用的也有约 30 个。然而我们可以只使用其中的 6 个最基本的函数就能编写一个完整的 MPI 程序去求解问题。这 6 个基本函数，包括启动和结束计算，识别进程以及发送与接收消息等。由于 MPI 具有移植性好、易用性、有完备的异步通信功能、有正式和详细的精确定义等多种优点，而且有多种不同的免费、高效、实用的实现版本，几乎所有的并行计算机都提供对它的支持。因此，在短短几年内 MPI 已成为消息传递并行编程模式的标准。

## 1.3 生物信息处理的并行算法的研究

由于生物信息数据的规模极其巨大，因此国内外都开展了生物信息处理算法并行化方向的研究。主要有NCBI的BLAST机群系统版本、PHRAP程序的SMP机器并行版本，以及在硬件基础上并行化工作，IBM还研制了专门用于基因组数据处理的超级计算机。

NCBI的BLAST系统用于提供网络序列检索服务，在任何时候可能会有大量的用户提交序列检索的请求，NCBI的机群系统版本主要是采用负载均衡系统实现对多用户请求任务在机群的多个节点间的分配，这种系统可以大大提高BLAST检索网站的任务吞吐率。PHRAP程序主要用于大规模的序列拼接过程中，PHRAP程序在数据量比较大的情况下非常耗时间，PHRAP程序的SMP机器并行版本主要是利用多线程技术加速其中具有并行性的部分，获得更好的时间效率。比如SPSOFT(southwest parallel software http://www.spsoft.com)，它实现了在普通计算机上提供高性能和高处理量的生物信息软件。其最新提供支持linux下的并行化的Phrap, SWAT, CrossMatch，并且支持SP和PowerPC上的IBM AIX 系统。并行的Phrap

在单cpu上能快一倍，并且随着cpu的增多性能会更好。Cross_Match在单cpu上能快30%，在4个cpu机器上快3倍。

国外还开展了特殊生物信息处理中算法的研究以及在硬件基础上的并行化方向的研究，主要是研究生物信息学中的一些关键的算法，研究其中的可并行性，然后将其固化到硬件芯片中，从而提高整个计算系统的性能。比如DeCypher生物信息处理加速器，它提供了可配置计算的技术（configurable computing See http://www.timelogic.com ），它的目的即是通过添加硬件来达到加速。可配置系统的特点是通过软件来控制硬件，使其具有自动从连线的能力，形成新的功能。从而复杂算法的内循环可以在一个时钟周期中完成，而通常则需要上千的时钟执行。可配置计算机利用FPGA (Field Programmable Gate Array)集成电路，集成电路的逻辑功能是动态的安排。集成电路用软件动态的控制，为每一个计算单元分配尽量少的资源，最大化地提高计算的并行度和速度。DeCypher系统包括一个或多个加速的计算管道，由作业调度软件的统一管理。每一个计算管道集成了多个cpu，可扩展的FPGA加速阵列以及本地RAM和磁盘缓存。标准的DeCypher服务可达到每秒6万亿次的Smith-Waterman letter-pair比对以及每秒250万亿次的letter-pair比对。

IBM耗资1亿美元研制了一套代号为"蓝色基因"(Blue Gene)的超级计算机，通过对各个蛋白质分子聚合到一起的多种力量加以测量，来研究人类蛋白质分子的折叠方式。第一台“蓝色基因L”计算机安装在美国的劳伦斯-利弗摩尔国家实验室，设计速度为每秒360万亿次，采用IBM的Power系列处理器，每个处理器都拥有两个内核，其中一个用于处理数据，另外一个用来通信，但两个内核都可以执行计算任务。"蓝色基因"采用一种称为SMASH的全新体系结构，可以在简化指令的基础上实现800万个线程并行处理的能力，并能做到自稳定、自适应和自修复。整套系统由64个6英尺高的机柜互联而成，每个机柜配置8块主板、每块主板上有64个芯片、每个芯片上包含32个处理器。

在国内，中科院计算技术研究所与华大测序中心合作，基于曙光3000超级计算机系统，开发了Balst，Phrap，Smith-Waterman的并行算法，并应用于华大测序中心的数据处理流程中

## 1.4 论文的主要贡献

由于生物信息学问题具有计算量大的特点，对计算机的处理速度要求高。而这

些问题大部分本质上是组合优化问题，对其并行化的难度较大。本文对生物信息学领域中的一些问题的并行计算进行了深入的研究，取得了很好的效果。主要的贡献体现在如下几个方面：

1. 提出一种快速的最长公共子序列算法FAST_LCS，该算法首先对字符串建立相应的同字符后续表，随后对于相应的初始同字符对，并行地在该表中逐层地搜索其后继同字符对，得到所有的后继同字符对及相应的层次值。最后由最大层次值的同字符对进行回溯，依次求得其所有前驱同字符对，最后得到相应的比对结果。这种基于同字符后续表的算法同样可以用到多序列的最长公共子串问题中。算法使用了剪枝和跳跃技术，提高了处理速度。与其它经典的LCS算法相比，不但能够取得准确的结果，而且在速度、效率上有了很大的提高。

2. 对生物序列拼接问题提出了高效的并行算法。该算法提出了后缀索引的概念，首先对所有的序列片段建立后缀索引；然后对所有的序列片段 $i$，在所有其他序列片段 $j$ 的后缀索引中查找匹配度最高且最长的后缀，匹配长度记为 $l_{ij}$，再根据 $l_{ij}$ 建立有向图；在该有向图中，顶点代表序列片段，顶点 $ij$ 间的边上的权值为序列片段前、后缀之间的匹配长度 $l_{ij}$。再使用蚁群算法寻找最长的哈密尔顿回路；根据回路得出拼接结果。该算法以后缀索引代替后缀树，大大减少了计算量，有利于并行决算。算法还利用蚁群算法解决 TSP 问题的优势，减少了优化时间。与其它类似的算法相比，取得的结果准确，在速度、效率上有了很大的提高。

3. 提出了一种对基因表达数据进行双向聚类的并行算法。该算法根据数据集合双向聚类质量的反单调性，由最小的数据集合开始逐步添加行或列，最终找到所有满足条件的聚类。该算法处理速度快，聚类质量高，性能明显优于其他类似算法。

4. 对本文对生物信息学中的一些问题所提出的高效的并行算法已经在并行计算机深腾 1800 上用 MPI(C 绑定)编程运行，都使用生物信息的标准数据库中的测试数据进行试验，取得的试验结果表明，这些算法不但处理速度快，而且结果质量高。

## 1.5 论文的组织

论文的以下章节内容组织如下：

第二章在介绍与分析生物序列比对问题的研究现状的基础上，提出了同字符对和同字符后续表的概念，进而提出了一种快速的最长公共子序列并行算法 FAST_LCS，介绍了我们所提出的剪枝和跳跃技术。

第三章分析了目前国内外对生物序列拼接问题的研究现状，对该问题提出了高效的并行算法。在该算法中，我们提出了后缀索引的概念，用来查找匹配度最高且最长的后缀，进而建立有向图；使用蚁群算法寻找最长的哈密尔顿回路以找出拼接。

第四章 在介绍与分析基因表达数据双向聚类问题研究现状的基础上，提出了一种对基因表达数据进行双向聚类的并行算法。我们根据数据集合双向聚类质量的反单调性，由最小的数据集合开始逐步添加行或列，并行找到所有满足条件的聚类。

最后，第五章是论文的总结和对生物信息学并行计算研究领域的展望。

# 第二章 序列比对问题的并行计算

生物序列的比对是生物信息学中最基本、最重要的操作，在序列分析、基因识别、蛋白质结构预测、生物进化树的构建等领域中有着广泛的应用。通过序列比对可以发现生物序列中的一些功能、结构以及进化信息。本章在介绍与分析生物序列比对问题的研究现状的基础上，提出了同字符对和同字符后续表的概念，进而提出了一种快速的最长公共子序列并行算法 FAST_LCS，介绍了我们所提出的剪枝和跳跃技术。

## 2.1 生物序列比对问题的研究回顾

### 2.1.1 双序列比对

序列比对的理论基础是进化学说，如果两个序列之间具有足够的相似性，我们就可以推测二者可能有共同的进化祖先，可以认为它们是由同一祖先经过序列内残基的替换、残基或序列片段的缺失、以及序列重组等遗传变异过程分别演化而来的。序列相似和序列同源是不同的概念，序列之间的相似程度是可以量化的参数，而序列是否同源需要有进化事实的验证。如果两个序列有显著的相似性，要确定二者具有共同的进化历史，进而认为二者有近似的结构和功能还需要更多实验和信息的支持。通过大量实验和序列比对的分析，一般认为蛋白质的结构和功能比序列具有更大的保守性，因此粗略地说，如果序列之间的相似性超过 30%，它们就很可能是同源的[5]。

序列比对是序列分析和数据库搜索的基础，是计算机工具运用于生物学领域最重要、最基本的操作，是生物信息学的基础，在蛋白质结构预测、种系发生树创建、生物进化研究等领域中有着非常广泛的应用。序列比对的目的是求出给定的序列之间的相似性及差异性，是提供序列之间相似性度量的基本工具。

在进行序列两两比对时，有两方面因素会直接影响序列相似性分值：替代矩阵和空位罚分。

在进行序列比对的过程中，我们可以对字符进行统一的处理，即匹配与不匹配，但在实际情况中，尤其对于蛋白质来说，不同字符之间的替代所得到的相似性分值得分是不一样的。某些氨基酸可以很容易地相互取代而不用改变它们的理化性质。例如，考虑这样两条蛋白质序列，其中一条在某一位置上是丙氨酸，如果该位点被替换成另一个较小且疏水的氨基酸，比如缬氨酸，那么对蛋白质功能的影响可能较小;如果被替换成较大且带电的残基，比如赖氨酸，那么对蛋白功能的影响可能就要比前者大。直观地讲，比较保守的替换比起较随机替换更可能维持蛋白质的功能，且更不容易被淘汰。所以有必要提出一种不同于稀疏矩阵的打分矩阵来表达这种不同，这就是提出替代矩阵(也可称为打分矩阵)的理由。在打分矩阵中，详细地列出各种字符替换的得分，从而使得序列之间的相似度计算更为合理。在比较蛋白质时，我们可以用打分矩阵来增强序列比对的敏感性。打分矩阵是序列比较的基础，选择不同的打分矩阵将得到不同的比较结果，而了解打分矩阵的理论依据将有助于在实际应用中选择合适的打分矩阵。

目前国际上常用的替代矩阵有 PAM 和 BLOSUM 等，它们来源于不同的构建方法和不同的参数选择，包括 PAM250、BLOSUM62、BLOSUM90、BLOSUM30 等。对于不同的对象可以采用不同的替代矩阵以获得更多信息，例如对同源性较高的序列可以采用 BLOSUM90 矩阵，而对同源性较低的序列可采用 BLOSUM30 矩阵。

在序列比对时插入的空位可以单个不连续的形式插入，也可以大片连续的空位插入，但也不能无限制的插入空格，这样会打乱序列，使序列的排列面目全非。所以为了保证序列的相似性，引入空位罚分。

实际上，空位罚分主要是是为了补偿插入和缺失对序列相似性的影响，由于没有什么合适的理论模型能很好地描述空位问题，因此空位罚分缺乏理论依据而更多的带有主观特色。 一般的处理方法是用两个罚分值，给空位的第一个空格分配空位罚分 $W_g$，称为“空位设置罚分”，表示新增一个空位的成本，给后面的每个空格分配罚分 $W_s$，称为“空位扩展罚分”，表示空位延伸一个空格的成本；对于长度等于 1 的空位，就给这个空格分配“空位设置罚分”。对于具体的比对问题，采用不同的罚分方法会取得不同的效果。

对于比对计算产生的分值，到底多大才能说明两个序列是同源的，对此有统计学方法加以说明，主要的思想是把具有相同长度的随机序列进行比对，把分值与最初的比对分值相比，看看比对结果是否具有显著性。相关的参数 E 代表随机比对分值不低于实际比对分值的概率。

比对的结果具有足够的统计学显著性，这样就排除了由于偶然的因素产生高比对得分的可能。对于严格的比对，必须 E 值低于一定阈值才能说明比对的结果具有

足够的统计学显著性，这样就排除了由于偶然的因素产生高比对得分的可能。

### 2.1.1.1 问题描述

序列比对实际上就是运用某种特定的数学模型或算法，找出两个或多个序列之间的最大匹配碱基或残基数。

将 DNA 序列和氨基酸序列抽象成为字符序列，我们就可以用数学方式来描述序列比对问题了。从数学的角度来看，序列比对实质上是一个优化问题[16]。

假设现在有两条生物序列：

$$S_1 = s_{11}s_{12}...s_{1l_1}$$

$$S_2 = s_{21}s_{22}...s_{2l_2}$$

其中 $s_{ij} \in R, j=1,2,\cdots l_i, i=1,2$。这里 $R$ 为核苷酸或氨基酸字符集，对于 DNA 序列 $R=\{A,T,C,G\}$；对于蛋白质序列， $R$ 包含了 20 个字符，每个字符代表一种氨基酸。

在长期进化过程中，有些核苷酸残基相对保守，而有些则可能发生变异，如 T 变成 G，C 变成 A 等。另外，少数残基会发生缺失或插入现象。因此，在比较两条相关序列时会出现中断现象，这就产生了“间隙”问题，间隙用字符“-”表示。我们记 $R' = R \cup \{-\}$。

序列 $S_1$ ,$S_2$ 的一个比对 就是要在 $S_1$ 及 $S_2$ 中的适当的位置插入一些间隙，得到 $S_1'$及 $S_2'$，使得 $S_1'$、$S_2'$具有相等的长度，并且不允许 $S_1', S_2'$ 中对应的列同时为“-”。

所有比对方法都是将序列间残基的相似与不相似转换成数值后进行比较，即给相同或相似的残基对赋以正分，对不同的配对进行罚分。同时，对于相似度很高的序列来说，基因序列发生插入/删除的情况是较少的，所以对于以寻求序列间最大相似度为目的序列比对来说，应当尽量减少空位的数目。因此，除了对替换进行罚分以外（替换降低了相似程度），对插入空位也要进行罚分。然后再计算总得分。所以，为了评价一个比对质量的优劣，需要定义一个计算比对得分的适当的目标函数。一种常用的目标函数定义如下：

$$Score(Align) = \sum_{r=1}^{L} \sigma(s_{1r}', s_{2r}')$$

其中函数 $\sigma(x,y)(x,y \in R')$是一计分函数，表示残基对 $(x,y)$ 比较时的得分。

例如，假设有两条序列分别为：

$$S_1 = \text{“}ATGGCTATG\text{”}$$

$$S_2 = \text{“}ATGCGTAGT\text{”}$$

设计分函数σ（$x,y$）用如下表格表示：

| y \ x | A | T | G | C | - |
|---|---|---|---|---|---|
| A | 2 | -1 | -1 | -1 | -2 |
| T | -1 | 2 | -1 | -1 | -2 |
| G | -1 | -1 | 2 | -1 | -2 |
| C | -1 | -1 | -1 | 2 | -2 |
| - | -2 | -2 | -2 | -2 | 0 |

它们的一种比对结果是：

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 序列 S1'： | A | T | G | G | C | T | A | - | T | G | A |
| 序列 S2'： | A | T | G | C | G | T | A | G | T | - | G |
| | 2 | 2 | 2 | -1 | -1 | 2 | 2 | -2 | 2 | -2 | -1 |

*Score* (*Align*)=2+2+2-1-1+2+2-2+2-2-1=5

序列比对的任务就是在序列之间的所有比对当中，找到取得最大比对得分的最优比对。一般来说，最优比对就是两个序列匹配（相同）字符数目最多的情况。

这样，具有生物学意义的双序列比对问题就成为了如下的优化问题:

$$\max_{Align\in\Gamma} Score(Align)$$

其中Γ表示序列 $S_1, S_2$ 的所有可能比对的集合。

### 2.1.1.2 研究回顾

Needleman-Wunsch算法[65]是双序列比对的经典算法，其使用的是动态规划的基本思想。对于长度分别为$m$和$n$的两个序列$A$和$B$，构造矩阵$T$，矩阵$T$中的最后一个元素$T[m][n]$即对应于最优比对的得分，而最优比对本身则可以通过回溯算法得到。该算法的时间和空间复杂度均为$O(mn)$。

Smith-Waterman对Needleman-Wunsch算法稍加改动，使其可以计算局部最优比对[66]，其所需的时间和空间复杂度仍是$O(mn)$。Mayers和Miller[18]使用Hirschberg[19]提出的技巧在时间复杂度不变的前提下将空间约减到$O(m+n)$。M. Crochemore等人

对上述经典算法加以改进，提出了一个可以在$O(n^2/\log n)$时间内实现的双序列比对算法[67]。其主要思路是对序列进行压缩编码，由此将序列分为若干段，从而将比对所构造的矩阵分为若干块来计算。后面的块的计算可以利用前面的块的结果在常数时间内计算得出。

除了利用矩阵来计算序列比对外，还有两种常用于序列分析的工具是后缀阵列Suffix Array[68]和后缀树[69]。AVID[70]是一个双序列全局比对算法，首先，用后缀树找出所有的最大匹配子序列，并在其中选择所有不重叠，不交叉的序列作为锚点。然后用锚点作为最后比对的一部分，在锚点之间的序列部分则递归的用此算法进行比对。

由于生物数据的信息量极大，序列比对的计算需要耗费大量的时间。由于并行算法可以大大地加快问题求解速度，近年来对该问题并行化的研究也引起研究者的注意。

在CREW－PARM模型上，Aggarwal[23]和Apostolico等人[24]独立地提出了一个$O(\log m \log n)$时间、使用$\frac{mn}{\log m}$个处理机的并行算法；Mi Lu等人[25]设计了两个并行算法：一个使用$\frac{mn}{\log m}$台处理机，时间复杂度为$O(\log^2 n+\log m)$；另一个使用$\frac{mn}{\log^2 m \log\log m}$台处理机，时间复杂度为$O(\log^2 m \log\log m)$。

在CRCW-PRAM上Apostolico等人[24]给出了时间为$O(\log n(\log\log m)^2)$的使用$\frac{mn}{\log\log m}$个处理机的算法。

在Systolic阵列上，Robert等人[26]提出了使用$m(m+1)$个单元，计算时间为$n+5m$的并行算法；Chang等人[27]提出了使用$mn$个单元，计算时间为$4n+2m$的并行算法；Luce等人[28]的并行算法使用了$\frac{m(m+1)}{2}$个单元，计算时间为$n+3m+q$，这里$q$为最长公共子序列的长度；Freschi等人[29]基于Run length encoded技术给出了使用$M+N$个单元，计算时间为$O(m+n)$的算法，这里$m \le n$，$q$为最长公共子序列的长度，$M$和$N$分别为两个序列在Run-length-encoded的编码长度。

### 2.1.1.3 Needleman and Wunsch 算法

下面介绍经典的序列比对算法，即Needleman -Wunsch算法。该算法是序列比

对研究领域中的奠基石。后来出现的很多两两序列比对算法、多序列比对算法、以及序列比对的并行算法都是在该算法的基础上提出来的。Needleman -Wunsch算法使用的是动态规划的基本思想，即把一个问题分解成为计算量合理的子问题，然后使用这些子问题的结果来计算最终答案。

算法构造一个$(m+1)\times(n+1)$的二维数组 $T$，数组中的元素 $T[i, j]$（$0\le i\le m, 0\le j\le n$）记录了"$A_1,A_2,\ldots,A_i$"与 "$B_1,B_2,\ldots B_j$" 的最优比对分值。

$T[0,0]$初始化为 0，$T[0,j]$和 $T[i,0]$分别初始化为：

$$T[0,j]=\sum_{k=1}^{j} f('-',B_k) \tag{2.1}$$

$$T[i,0]=\sum_{k=1}^{i} f(A_k,'-') \tag{2.2}$$

当对非空序列"$A_1,A_2,\ldots,A_i$"与 "$B_1,B_2,\ldots B_j$" 进行比对时，字符 $A_i$ 及 $B_j$ 的比较有三种情况：

(1) $A_i$ 与'—'比对；

(2) '—'与 $B_j$ 比对；

(3) $A_i$ 与 $B_j$ 比对。

故有：

$$T[i,j]=\max\begin{cases}T[i-1,j]+f(A_i,'-')\\ T[i,j-1]+f('-',B_j)\\ T[i-1,j-1]+f(A_i,B_j)\end{cases} \tag{2.3}$$

对矩阵 $T$ 中的元素我们可以按行从上到下，每行从左到右依次计算出来。$T[m,n]$即为最优比对的 *sim* 分值，整个矩阵的计算可在 $O(mn)$时间及空间内完成。

最优比对的分值计算出来以后，寻找最优比对可以看作一个回溯过程。例如，在 $T[i,j]$处往前回溯，就取决于公式 2.3 的三个侯选项中哪几个贡献了最大值，如果是第一项，就从 $T[i,j]$回溯至 $T[i-1,j]$，如果是第二项，就从 $T[i,j]$回溯至 $T[i,j-1]$，以此类推。回溯过程从 $T[m,n]$开始，至 $T[0,0]$结束。因为公式 2.3 中可能有一项或几项的值都是最大值，所以最优路径有可能不止一条。回溯过程的时间复杂度为 $O(m+n)$，空间复杂度为 $O(mn)$。

设序列 $A$="acbcdb"，序列 $B$="cadbd"，用 Needleman -Wunsch 算法求得其比对的过程见图 2.1。

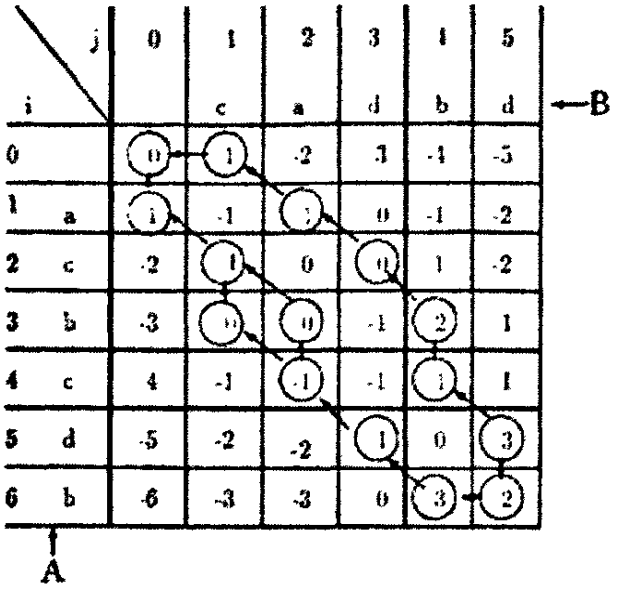



图 2.1 Needleman-Wunsch 算法

### 2.1.1.4 Smith-Waterman 算法

1981 年，Smith 和 Waterman 提出了一种寻找并比较序列中具有局部相似性的区域的算法。它也是一种基于矩阵的算法，而且也运用了回溯法建立允许空位插入的比对。多年来，Smith-Waterman 算法一直是序列局部比对算法基础，许多算法都是基于这一算法开发和改进的。

Smith-Waterman 算法相对于 Needleman-Wunsch 算法有一定的变化，首先是第 0 行和第 0 列的元素分值都赋为 0，可以把这些单元理解为序列片段的起始端，其长度为 0。在递归关系中 $T[i,j]$的值是公式(2.3)中三个值和 0 之间的最大值。如果 0 是最大值，表示局部相似片段在这一点不再延伸，即片段的末端。

初始条件：　$T[i,0]=0$; $T[0,j]=0$; $(0\leqslant i\leqslant m, 0\leqslant j\leqslant n)$

递归关系：
$$T[i,j]=\max\begin{cases}T[i-1,j]+f(Ai,'-')\\T[i,j-1]+f('-',Bj)\\T[i-1,j-1]+f(Ai,Bj)\\0\end{cases}$$

序列的最优匹配片段从矩阵中最高分值元素(位置不固定在矩阵右下角)开始回溯来获取，次优片段从次高分值元素回溯来获取。

Smith-Waterman 算法也要计算所有的矩阵元素和$(m+1)*(n+1)$个存储单元，因此

它的时间和空间复杂度同 Needleman-Wunsch 算法一样都是 O(*mn*)。

### 2.1.2 多序列比对

多序列比对就是把两条以上可能有系统进化关系的序列进行比对的方法，实际上是双序列比对问题的一般化推广。目前对多序列比对的研究还在不断深入，现有的大多数算法都基于渐进的比对思想，在序列两两比对的基础上逐步优化多序列比对的结果。进行多序列比对后可以对比对结果进行进一步处理，例如构建序列模式的 profile，将序列聚类构建分子进化树等等。可以通过比对所得到的多个相关蛋白质的相似性(同源性)，了解其在进化上亲缘关系的远近，推断分子起源和进化规律等。由相似性还可以研究多个序列中的保守区域，就可以猜测这些区域对蛋白质结构、功能的重要性，从而进行分子设计。

多序列比对的前提是所有待比对序列都有一个共同的祖先，由于进化过程中的局部变异而产生了各个不同的物种，其中局部变异包括插入、删除、替换三种情况。在理想情况下，正确的多序列比对结果应该能够真实的反映序列的进化过程。

#### 2.1.2.1 问题描述

对于多序列比对问题，传统方法所采用的表示模型是行-列模型，即对于输入的多个序列插入空位并排列比对，使其达到相同的长度。对于 $N$ 个序列 $S_1,\ldots,S_n$，其多序列比对是一个新的序列集 $S'=(S_1',\ldots,S_N')$，$S'$中的所有序列长度相同，并且每一个序列 $S_i'$由 $S_i$ 插入空位'－'得到。如果将各序列在垂直方向排列起来，则可以根据每一列观察各序列中字符的对应关系。图 2.2 是 6 个蛋白质序列片断基于行-列模型的多序列比对。

```
SRC_RSVP     -FPIKWTAPEAALY---GRFTIKSDVWSFGILLTELTTKGRVPYPGMVNR-EVLDQVERG
YES_AVISY    -FPIKWTAPEAALY---GRFTIKSDVWSFGILLTELVTKGRVPYPGMVNR-EVLEQVERG
ABL_MLVAB    -FPIKWTAPESLAY---NKFSIKSDVWAFGVLLWEIATYGMSPYPGIDLS-QVYELLEKD
FES_FSVGA    QVPVKWTAPEALNY---GRYSSESDVWSFGILLWETFSLGASPYPNLSNQ-QTREFVEKG
FPS_FUJSV    QIPVKWTAPEALNY---GWYSSESDVWSFGILLWEAFSLGAVPYANLSNQ-QTREAIEQG
KRAF_MSV36   TGSVLWMAPEVIRMQDDNPFSFQSDVYSYGIVLYELMA-GELPYAHINNRDQIIFMVGRG
```

图 2.2 多序列比对

当得到多序列比对后，需要对比对的质量进行评价，*SP*（Sum of Pairs）模型是评价比对优劣的最常用模型。设得分函数具有可加性，多序列比对的得分是各列得

分之和，对于某一列字符的得分可用以下公式进行计算，即某一列字符的*SP*得分为一列中所有字符对得分之和：

$$SP\_Score(c_1,c_2,...,c_N)=\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}f(c_i,c_j) \tag{2.4}$$

其中$c_i$表示该列中的第$i$个字符，$f(c_i,c_j)$表示字符$c_i$和字符$c_j$比较所得分值。具体计算时，可以先对多序列比对的每一列进行计算，然后将各列得分相加，也可以先计算所有两两序列比对的得分，然后再将得分相加。这两种计算在$f('-','-')=0$这一条件成立下等价。

多序列比对的目标是：在计分机制确定的情况下，寻找使得比对得分最高的多序列之间的最优比对。可以证明，利用 *SP* 模型寻找最优多重序列比对是一个 NP-完全问题。

除了常用的行-列模型外，POA[71]提出了用有向无环图来表示多序列比对，开辟了多序列比对问题中的一个新的研究领域。

ABA[72]提出用 A-Bruijn 图作为多序列比对的表达模型，该模型能够对包括重复域和交错域的蛋白质序列进行较灵活的比对。

要获得给定的多个基因或蛋白质序列之间的一个正确的比对是一个困难的计算问题，其困难在于两个方面：一是如何根据包括结构信息在内的生物学意义对给定比对打分，即如何获得一个完美的目标函数(Objective Function 简称 *OF*)；二是在目标函数确定的情况下，如何求得分值最高的最优比对。前者要依据生物学的知识和实际问题的需要来决定。假设已经求得的目标函数相当完美且简单，后者也将是一个非常困难的计算问题。

### 2.1.2.2 研究回顾

虽然Needleman -Wunsch 的算法可以找到两个序列的最优解，但如果将这个算法直接延伸应用到多序列比对的问题上来却是不切实际的，因为算法的时空开销达到$O(2^N-1)(\prod_{i=1}^{N}|S_i|)$，其中$N$为序列的个数，$|S_i|$为第$i$个序列的长度。尽管后来很多人对此算法作了改进，如基于Carrillo-Lipman的算法[30]的MSA[31]，基于分治策略和MSA的DCA[32]，优化DCA算法的OMA[33]等等，可是这些算法处理的序列长度和数量仍受到很大限制。

多序列比对的启发式计算方法通常在适度牺牲正确性的基础上来提高计算效

率。ClustalW [34]是应用最为广泛的多重序列比对软件，它是高效的渐进方法的代表。由于对 Feng-Doolittle 提出的算法[35]作了一系列的改进，使得结果的正确性得到进一步提高，所以 ClustalW 在生物信息学中得到了广泛的应用。

但是渐进方法是以序列的两两比对为基础的，两两比对的局部结果并没有考虑其他序列所提供的信息，所以在多重序列比对中往往并不最优，这样会对多重序列比对带来错误信息，并且这些错误信息不能在后期阶段被纠正。

除渐进方法以外，常用到的方法有迭代优化方法[73-76]、基于一致性（consistency-based）的方法[77-80]、基于隐马尔可夫模型的方法[81-83]等。

迭代方法的思想是对已经求得的中间比对进行迭代优化，从而提高解的正确性。比如 Gotoh 提出的 Prrp，其运用的是非随机迭代方法，而基于遗传算法的多重序列比对方法 SAGA[84]以及基于模拟退火方法的多重比对方法[85]运用的是随机迭代的策略。

Dialign 系列和 Align-m 用一致性策略来解决多重序列比对问题，对于局部的相似性具有较高的敏感性。还有一种方法是基于隐马尔可夫决策方法为蛋白质家族训练建立模型，从而识别出家族的模式特征，然后用序列与模型的比较来判断序列是否属于这个家族，其实际上是基于统计的一种方法。

另外还有基于快速傅里叶变换的方法 MAFFT[86]、基于块的多序列比对算法[87]、基于禁忌搜索的多序列比对方法[88]等等。除此以外，有些算法采用将几种策略集成在一起的方法来提高算法的效率，比如 T-Coffee[89]、MUSCLE[90]、PROBCONS[91]等，关于多重序列比对的相关算法及其测评可参阅文献[92-94]。

针对LCS问题，我们提出一种快速的最长公共子序列的算法FAST_LCS。该算法通过对两条字符串建立相应的同字符后续表，随后对于相应的初始同字符对，并行地在该表中逐层地搜索其后继同字符对，得到所有的后继同字符对及相应的层次值。最后由最大层次值的同字符对进行回溯，依次求得其所有前驱同字符对，最后得到相应的比对结果。对于长度为$n$和$m$的两序列$X$, $Y$，该串行算法所需的内存为$\max\{4*(n+1)+4*(m+1),L\}$，这里$L$指初始同字符对的个数。而并行算法的时间复杂度为$O(|\mathrm{LCS}(X,Y)|)$，这里$|\mathrm{LCS}(X,Y)|$指序列$X$, $Y$的最长公共子串的长度。这种基于同字符后续表的算法同样可以用到多序列的最长公共子串问题中，对于$n$个序列$X_1$, $X_2$, … , $X_n$, 该算法的并行复杂度为$O(|\mathrm{LCS}(X_1, X_2, \ldots, X_n)|)$，该复杂度与序列个数$n$无关。我们对tigr数据库中的基因序列在MPP并行处理机深腾1800上进行的实验结果证明，本文算法与其它经典的LCS算法相比，不但能够取得准确的结果，而且在速度、效率上有了很大的提高。以下几节就详细介绍我们所提出的算法及其基本概念、基本操作。

## 2.2 同字符后续表及其同字符对

设欲比对的生物序列分别为 $X=(x_1, x_2, \cdots\cdots, x_n)$，$Y=(y_1, y_2, \cdots, y_m)$，其中，$x_i$，$y_i \in \{A, C, G, T\}$。在我们的算法 FAST_LCS 中，为了找出这两条串的最长公共子序列，我们首先要对这两条序列建立对应的同字符后续表。我们定义 $A$=CH(1)，$C$=CH(2)，$G$=CH(3)，$T$=CH(4)，将 $X$, $Y$ 的同字符后续表分别记为 $TX$, $TY$，这里，$TX$ 为 4*($n$+1)，$TY$ 为 4*($m$+1)的二维数组。我们对 $TX(i,j)$ 定义如下：

**定义 2.1.** 对序列 $X=(x_1, x_2, \ldots\ldots, x_n)$，其同字符后续表 TX 定义为：

$$\mathrm{TX}(i,j)=\begin{cases}\min\{k \mid k \in SX(i,j)\} & SX(i,j)\neq\phi \\ - & \text{otherwise}\end{cases} \tag{2.5}$$

其中 SX $(i,j)=\{k \mid x_k=\mathrm{CH}(i), k>j\}$, $i=1,2,3,4, j=0,1,\ldots n$. “－”表示无定义. 由此定义可知，若 $TX(i,j)$ 不为“－”，它表示 $X$ 第 $j$ 个位置上下一个为 CH($i$)的字符的位置；若 $TX(i,j)$ 为“－”则表示 $X$ 序列在第 $j$ 个字符后无等于 CH($i$)的字符。

例 2.1 设 $X$ = “TGCATA”，$Y$ = “ATCTGAT” 则 $TX$ 及 $TY$ 分别为：

TX:

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| 1 | A | 4 | 4 | 4 | 4 | 6 | 6 | – |
| 2 | C | 3 | 3 | 3 | – | – | – | – |
| 3 | G | 2 | 2 | – | – | – | – | – |
| 4 | T | 1 | 5 | 5 | 5 | 5 | – | – |

TY:

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | A | 1 | 6 | 6 | 6 | 6 | 6 | – | – |
| 2 | C | 3 | 3 | 3 | – | – | – | – | – |
| 3 | G | 5 | 5 | 5 | 5 | 5 | – | – | – |
| 4 | T | 2 | 2 | 4 | 4 | 7 | 7 | 7 | – |

**定义 2.2.** 在 $X$,$Y$ 中，若有 $x_i=y_j$，则记 $(i,j)$ 为一同字符对。所有同字符对的集合记为 $S(X,Y)$。

**定义 2.3.** 若 $(i,j)$，$(k,l)$ 皆为同字符对，且有 $i<k, j<l$ 则称$(i,j)$为 $(k,l)$ 的一个前驱，或称 $(k,l)$ 为 $(i,j)$ 的一个后继，记为 $(i,j)<(k,l)$。

**定义 2.4.** 若设集合 $P(i,j)=\{(r,s) \mid (i,j)<(r,s), (r,s)\in S(x,y)\}$为 $(i,j)$ 的所有后继同字符对集合，若有 $(k,l)\in P(i,j)$，且不存在 $(k',l')\in P(i,j)$，使得：$(k',l')<(k,l)$, 则称 $(k,l)$ 为 $(i,j)$ 的直接后继，记为 $(i,j)\prec(k,l)$。

**定义 2.5.** 设 $(i,j)\in S(x,y)$，且不存在$(k,l)\in S(x,y)$，使得 $(k,l)<(i,j)$, 则称 $(i,j)$ 为初始同字符对，我们定义初始同字符对的层次为 1。

**定义 2.6.** 对任意的同字符对 $(i,j)\in S(x,y)$，它的层次号 $level(i,j)$ 定义为：

$$level(i,j)=\begin{cases}1 & \text{if}(i,j)\text{为一初始同字符对}\\ \max\{level(k,l)+1\mid(k,l)<(i,j)\} & \text{otherwise}\end{cases} \tag{2.6}$$

由以上定义，我们不难得出以下引理：

**引理 2.1.** 记 $X,Y$ 最长公共子序列的长度为 LCS($X$, $Y$)，则 |LCS($X$, $Y$)|=max{level ($i$,$j$) | （$i$,$j$） $\in S(X,Y)$} 。

**证明：** 设 $X,Y$ 的一个最长公共子序列所对应的同等字符对依次为（$x_{i1}, y_{j1}$）,($x_{i2}$, $y_{j2}$).....($x_{ir}$,$y_{jr}$)，这里 $r$=LCS（$X,Y$）。根据定义 2.5 可知，这里，（$i_1, j_1$）必然为初始同字符对，且由定义 2.4 和定义 2.6，必然有关系（$i_k, j_k$）$\prec$ ($i_{k+1}, j_{k+1}$)，$k$=1,2,...,$r$-1，则（$x_{ik}, y_{jk}$）的层次 *level* 即为 $k$，$r$ 则为最大的层次 *level*，即 $r$= max{level ($i$,$j$) | （$i$,$j$） $\in S$ ($X,Y$)}。如果不是，则存在正整数 $r'>r$，且有初始同字符对：($x_{i1'}$, $y_{j1'}$)$\prec$($x_{i2'}$, $y_{j2'}$)$\prec$.....$\prec$($x_{ir'}$, $y_{jr'}$)。其长度为 $r'>r$，这与 $r$=|LCS（$X,Y$）|的假设相矛盾。

**证毕。**

## 2.3 产生后继及剪枝的操作

在我们的算法中，首先对所有初始同字符对利用同字符后继表产生其所有的直接后继，然后再对这些后继并行地产生其所有直接后继，重复这样的操作，直至不能继续产生后继为止。因此，由同字符对产生其所有直接后继，是本算法的一个基本操作。对某同字符对（$i$,$j$）$\in S(x,y)$，由（$i$,$j$）产生其所有后继同字符对的操作可表示为：

$$(i,j)\rightarrow\{(\mathrm{TX}(k,i),\mathrm{TY}(k,j))\mid k=1,2,3,4,\mathrm{TX}(k,i)\neq\text{'-'}\text{and TY}(k,j)\neq\text{'-'}\} \tag{2.7}$$

即对 *TX* 的第 $i$ 列、*TY* 的第 $j$ 列相应元素进行配成对偶，例如对例 2.1 中的同字符对（2, 5），上述操作可表示为：

$$(2\ \ 5)\rightarrow\begin{bmatrix}4&6\\3&-\\-&-\\5&7\end{bmatrix}\rightarrow\begin{Bmatrix}(4&6)\\(3&-)\\(-&-)\\(5&7)\end{Bmatrix}\rightarrow\begin{Bmatrix}(4&6)\\(5&7)\end{Bmatrix}$$

由于（3, —）及（—, —）不表示一个同字符对，表示这一支搜索不能得出所需的结果，故应舍去。产生的结果说明（2, 5）的后继为（4, 6），（5, 7）。应该说明的是，所产生的不一定都是（$i$,$j$）的直接后继。例如（5, 7）不是（2, 5）的直接后继，因为有(2, 5) $\prec$ (4, 6) $\prec$ (5, 7)。

**引理 2.2** 对任一同字符对（$i$, $j$），用上述方法可以产生其所有的后继。

**证明**：根据（2.7）知，由（$i,j$）可以产生所有直接后继（$TX(k,i),TY(k,j)$），$k$=1,2,3,4；又根据（1）可知，$TX(k,i)$是$SX(i,j)$中的最小者，即在序列$X$中，位于字符$x_i$之后且距离$x_i$最近的第$k$种同字符CH($k$)。类似地，$TY(k,j)$是指在序列$Y$中，寻找位于字符$y_j$之后且距离$y_j$最近的第$k$种同字符。又因为$k$=1,2,3,4，即包含了所有可能的字符配对，所以一次后继产生操作可以得到（$i,j$）的所有直接后继。同理，对它的每一个直接后继可再进行后继产生操作得到其每一个直接后继的所有的直接后继。依此类推，可对同字符对（$i,j$）产生其所有的后继。

证毕。

由引理2.2的证明易知$(TX(k,0),TY(k,0))$, $k$=1,2,3,4,为所有初始同字符对。且根据引理2.2，由这些初始同字符对，我们可以得到所有同字符对及其层次值。在利用上述方法产生后继同字符对的过程中,我们可进行剪枝操作，对某些明显不能得出最长公共子序列的同字符对删除，以缩小搜索空间、加快搜索速度。

**定理2.1.** 如果在同一层上所产生的同字符对$(i,j)$和$(k,l)$中，有$(k,l)>(i,j)$，则剪去$(k,l)$不影响算法得到最优解。

**证明**：设在上一层产生$(k,l)$的同字符对为$(k_1,l_1)$，产生$(i,j)$的同字符对为$(i_1,j_1)$，又设经由$(k_1,l_1)$及$(k,l)$所产生的公共子序列为$a_1a_2...a_ma_{m+1}...a_r$，其中$a_m$对应于$(k_1,l_1)$，$a_{m+1}$对应于$(k,l)$。同样的，我们设经由$(i_1,j_1)$及$(i,j)$所产生的公共序列为$b_1b_2...b_mb_{m+1}...b_s...b_q$，其中$b_m$对应于$(i_1,j_1)$，$b_{m+1}$对应于$(i,j)$。由于$(k,l)>(i,j)$,根据引理2.2，$(k,l)$必然在$(i,j)$后被产生，则必存在$b_s$,$(m+1<s<q)$,使得$b_s$对应于$(k,l)$，由于$a_ma_{m+1}...a_r$及$b_sb_{s+1}...b_q$都是由同字符对$(k,l)$进行若干次后继操作产生的局部最长公共子序列，因为有“$a_ma_{m+1}...a_r$”＝“$b_sb_{s+1}...b_q$”，即有$q-s=r-m$，即：$q=r+(s-m)$。因为$s>m$，故有$q>r$。这说明了，经由$(k,l)$在第$m$层上所产生的子序列“$a_ma_{m+1}...a_r$”必不是最长公共子序列,我们可以在第$m$层将其剪去。

证毕。

由定理2.1可知，整个剪枝过程可以将所有冗余的同字符对剪去。在每一层产生同字符对后，我们在同一层上观察所有新产生的同字符对，若有同字符对$(i,j)$和$(k,l)$，满足$(k,l)\prec(i,j)$，则将$(i,j)$剪去。例如，在例2.1中同字符对(2,5)的后继(4,6)和(5,7)在同一层中，且$(4,6)\prec(5,7)$，根据定理2.1，我们可以将(5,7)剪去。

又如对于例2.1中的同字符对(1,2)，上述操作可表示为：

$$(1\ 2)\rightarrow\begin{bmatrix}4&6\\3&3\\2&5\\5&4\end{bmatrix}\rightarrow\begin{Bmatrix}(4\ 6)\\(3\ 3)\\(2\ 5)\\(5\ 4)\end{Bmatrix}\rightarrow\begin{Bmatrix}3&3\\2&5\end{Bmatrix}$$

即产生（1, 2）的 4 个后继：（4, 6），（3, 3），（2, 5），（5, 4）， 因为(3, 3)<（4, 6）并且（3,3）<(5,4)，根据定理 2.1，我们可以将(4, 6)和(5,4)剪去。随后我们在得到（3, 3），（2, 5）的两组后继同字符对后，也要进行剪枝操作：

$$\begin{Bmatrix}(3\ 3)\\(2\ 5)\end{Bmatrix}\rightarrow\left\{\begin{bmatrix}(4&6)\\(5&4)\end{bmatrix}\begin{bmatrix}(4&6)\\(5&7)\end{bmatrix}\right\}\rightarrow\begin{Bmatrix}(4\ 6)\\(5\ 4)\end{Bmatrix}$$

对于任一同字符对（$i,j$），将该同字符对在本层上与本组及其它组的同字符对作比较,若有$(k, l)<(i,j)$，则将（$i,j$）剪去。例如（5,7）是(4,6) 的后继，可将其剪去。这样，在每一轮产生后继的操作中，我们逐个剪去同一层中的一些同字符对，直到不能继续剪枝再进行下一轮产生后继的操作。

类似于定理 2. 1，还有一些有用的剪枝操作，可用以进一步缩小搜索空间。这些剪枝操作基于如下一些定理和推论。

**定理 2. 2.** 若在同一层中有同字符对$(i_1, j)$及$(i_2, j)$，其中 $i_1<i_2$, 则$(i_2, j)$可以剪去。

**证明**：设$(i_1, j)$的后继依次为：$(l_2,\ j_2)\prec(l_3, j_3)\prec\cdots\prec(l_r, j_r)$ 即子串 “” $x_{i1}\,x_{i1+1}\cdots x_n$” 和 “ $y_j\,y_{j+1}\cdots y_m$ ” 的最长公共子串长度为 $r$。再设$(i_2, j)$的后继同字符对依次为：$(k_2, j_2')\prec(k_3, j_3')\prec\cdots\prec(k_q, j_q')$ 因为 $i_1<i_2<k_2, j<j_2'$；$(k_2, j_2')$为$(i_1, j)$的一个后继，即必然存在 $s$（$2<s<r$）使 $(l_s,\ j_s)=(k_2,\ j_2')$，进而可推得：$(l_{s+1},\ j_{s+1}')=(k_3, j_3')$ ；$(l_{s+2},\ j_{s+2}')=(k_4, j_4')$，…，$(l_r, j_r)=(k_q, j_q')$。即可得 $r\text{-}s=q\text{-}2$，即 $r\text{-}q=s\text{-}2\geqslant 0, r\geqslant q$ ，由于经由$(i_2, j)$产生最长公共子串长度不大于经由$(i_1, j)$所产生的最长公共子串长度，故可以剪去$(i_2, j)$。

证毕。

将定理 2. 2 推广，不难推出如下的推论：

**推论 2. 1.** 若在同一层中，有同字符对$(i_1, j)$，$(i_2,\ j)$，…，$(i_r, j)$，其中 $i_1<i_2<\cdots<i_r$，则（$i_2,\ j$），…，$(i_r, j)$ 可以剪去。

## 2.4 产生后继的跳跃操作

在产生后继同字符对的过程中，我们可以对一些明显可确定其后面的最长公共子序列的同字符对进行跳跃操作，跳过那些最长公共子序列所对应的同字符对，以加快搜索速度。为此，我们首先定义序列的同字符增量表，以反映在字符串某一位置上，下一个同字符与当前字符的相对位置偏移。

**定义 2.7.** 对序列 $X=(x_1,x_2,\ldots,x_n)$，其同字符增量表 $SX$ 是一个 $4*n$ 的数组，定义为：

$$SX(i,j)=\begin{cases} TX(i,j)-j & \text{if } TX(i,j)\neq\text{'-'} \\ \infty & \text{otherwise} \end{cases}$$

显然，$SX$ 的元素 $SX(i,j)$若不为∞，则表示在 $x_j$ 后面的第一个为 CH($i$)的字符相对于 $x_j$ 的位移。由 $SX$ 的定义可知，它可由 TX 直接导出。

**例 2.2** 设 $X=$ "TGCATA"，则 $SX$ 为：

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| 1 | A | 4 | 3 | 2 | 1 | 2 | 1 | ∞ |
| 2 | C | 3 | 2 | 1 | ∞ | ∞ | ∞ | ∞ |
| 3 | G | 2 | 1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 4 | T | 1 | 4 | 3 | 2 | 1 | ∞ | ∞ |

**引理 2.3.** 在同字符增量表的每一列元素中，如果它们不全为∞，则至少有一个元素为 1。

**证明：** 在 $SX$ 的第 $j$ 列不全为∞，说明 $x_j$ 不是 $X$ 的最后一个字符，存在 $x_{j+1}$，设 $x_{j+1}$ =CH（$i$），则 $SX(i,j)=1$。

证毕。

事实上，$SX$ 第 $j$ 列中的 "1" 所在的行，反映了 $x_{j+1}$ 是什么字符。由同字符增量表，我们可以在产生后继过程中对连续的同字符实现跳跃技术。

**定理 2.3.** 设 $X$ 中有子串 "$x_{i+1}x_{i+2}\ldots x_{i+t}$" 与 $Y$ 中的子串 "$y_{j+1}y_{j+2}\ldots y_{j+t}$" 完全相同，且该子串中仅包含 CH($i_1$),CH($i_2$),...CH($i_w$)这 $w$ 个字符，$w$=1,2,3,4. 则由第 $l$ 层同字符对$(i,j)$由字符 CH($i_1$),CH($i_2$),...CH($i_w$)产生的层号为 $l$+1 的后继同字符对可直接由层号为 $l+t$ 的同字符对$(i+t,j+t)$所代替。

**证明：** 设对于某 $i$ 值，$i$=1,2...$w$, 同字符对$(i,j)$由 CH($i$)所产生的后继为$(i+p, j+p)$. 又设在 "$x_{i+1}x_{i+2}\ldots x_{i+t}$" 与 "$y_{j+1}y_{j+2}\ldots y_{j+t}$" 中包含 CH($i$)的字符设为 $x_{i+k1}x_{i+k2}\ldots x_{i+kq}$.这里 $1\leqslant k_1\leqslant\ldots\leqslant k_q\leqslant t$，因为 $q\leqslant t$，则它所含的后继为$(i+k_1,j+k_1)$ $(i+k_2,j+k_2)\ldots(i+k_q,j+k_q)$.又设$(i+k_q,j+k_q)$的后继为$(i+s, j+s)$,这里 $s>t$，故$(i+s, j+s)$的层号

为 $l+q+1$.由于$(i+t,j+t)$是一个同字符对，且层号为 $l+t$.显然$(i+s, j+s)$也是$(i+t,j+t)$的关于字符CH($i$)的后继层号为$l+t+1 \geq l+q+1$.因此可以直接跳至层号为$l+t+1$的同字符对$(i+t,j+t)$。

证毕。

**例** 2.3 设 $X$= “GACGTA”,$Y$= “CGACGT”,则同字符增量表 $SX$、$SY$，同字符后续表 $TX$、$TY$ 分别为：

SX:

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| 1 | A | 2 | 1 | 4 | 3 | 2 | 1 | ∞ |
| 2 | C | 3 | 2 | 1 | ∞ | ∞ | ∞ | ∞ |
| 3 | G | 1 | 3 | 2 | 1 | ∞ | ∞ | ∞ |
| 4 | T | 5 | 4 | 3 | 2 | 1 | ∞ | ∞ |

SY:

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| 1 | A | 3 | 2 | 1 | ∞ | ∞ | ∞ | ∞ |
| 2 | C | 1 | 3 | 2 | 1 | ∞ | ∞ | ∞ |
| 3 | G | 2 | 1 | 3 | 2 | 1 | ∞ | ∞ |
| 4 | T | 6 | 5 | 4 | 3 | 2 | 1 | ∞ |

TX:

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| 1 | A | 2 | 2 | 6 | 6 | 6 | 6 | - |
| 2 | C | 3 | 3 | 3 | - | - | - | - |
| 3 | G | 1 | 4 | 4 | 4 | - | - | - |
| 4 | T | 5 | 5 | 5 | 5 | 5 | - | - |

TY:

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | A | 3 | 3 | 3 | - | - | - | - | - |
| 2 | C | 1 | 4 | 4 | 4 | - | - | - | - |
| 3 | G | 2 | 2 | 5 | 5 | 5 | - | - | - |
| 4 | T | 6 | 6 | 6 | 6 | 6 | 6 | - | - |

由于 $X$ 串中存在子串 “GACGT” 与 $Y$ 中的子串 “GACGT”完全相同，因此我们可以对其实施跳跃技术。例如对于上例中的同字符对(1,2),我们首先得到它关于 $SX$、$SY$ 的后继同字符对：

$$(1,2)\rightarrow\begin{bmatrix}1&1\\2&2\\3&3\\4&4\end{bmatrix}\rightarrow\begin{Bmatrix}(1&1)\\(2&2)\\(3&3)\\(4&4)\end{Bmatrix}$$

由此可知，无论是在串 $X$ 还是串 $Y$ 中，从当前同字符 “G” 开始，同时向后跳跃一个字符便得到 “A”，同时向后跳跃两个字符便为 “C”，而同时向后跳跃三个字符则为 “G”，同时向后跳跃四个字符则为 “T”，这表示从当前字符开始我们可以连续向后跳跃四个字符。即在得到(1,2)关于 $TX$,$TY$ 的后继同字符对：

$$(1,2)\rightarrow\begin{bmatrix}2&3\\3&4\\4&5\\5&6\end{bmatrix}\rightarrow\begin{Bmatrix}(2&3)\\(3&4)\\(4&5)\\(5&6)\end{Bmatrix}$$

随后，同字符对（1,2）可以直接向后跳跃四个字符，得到直接后继（5,6），相应的

层次号也随之加 4。而与（1,2）处于同一层的其它同字符对则仍然按原来的方法并行的产生其所有直接后继。

**定理 2.4.** 设在对序列 $X,Y$ 产生后继过程中有层号为 $l$ 的同字符对 $(i,j)$，且 $TX$ 及 $TY$ 的第 $i$、$j$ 列分别为 $G_i=(g_1,g_2,g_3,g_4)^T$, $H_j=(h_1,h_2,h_3,h_4)^T$,$SX$ 及 $SY$ 的第 $i$、$j$ 列分别为：$R_i=(r_1, r_2, r_3, r_4)^T$ , $S_j=(s_1,s_2,s_3,s_4)^T$,又设 $R_i$ 中的元素由小到大依次为 $r_{i1}, r_{i2}, r_{i3}, r_{i4}$, $S_j$ 中的元素由小到大依次为 $s_{j1},s_{j2},s_{j3},s_{j4}$。

(1) 若 $i_1=j_1$，设 $k=\min(r_{i2}, s_{j2})$,则由同字符对 $(i,j)$ 所产生的层号为 $l+1$ 的同字符对 $(g_{i1},h_{j1})$ 可以被层号为 $l+k-1$ 的$(i+k-1, j+k-1)$的同字符对所代替。

(2) 若 $i_1=j_1$，且存在正整数 $w\in[2, 4]$, 使得对于所有 $k\leqslant w$，有 $r_{ik}=s_{jk}$,$i_k=j_k$,且 $r_{ik}=r_{i(k-1)}+1$,则由同字符对$(i,j)$所产生的层号为 $l+1$ 的同字符对$(g_{i1},h_{j1})$、$(g_{i2},h_{j2})$…$(g_{i(w-1)},h_{j(w-1)})$ 可以被层号为 $l+r_{iw}$ 的 $(g_{iw},h_{jw})$ 所代替。

**证明：** (1) 由引理 2.3 易知 $r_{i1}=s_{j1}=1$,$x_{i+ri1}=x_{i+1}=\mathrm{CH}(i_1)$,$y_{j+sj1}=y_{j+1}=\mathrm{CH}(j_1)$, 且 $g_{i1}=i+1$,$h_{j1}=j+1$,又由 $i_1=j_1$,说明 $x_{i+1}$ 和 $y_{j+1}$ 是相等的字符 $\mathrm{CH}(i_1)$. 且因为 $r_{i2}$ 为 $r_{i1}$ 以后的最小元素，表示下一个为 $\mathrm{CH}(i_1)$的字符相对位移，说明在 $x_{i+1}$ 之后有 $r_{i2}-r_{i1}=r_{i2}-1$ 个相同的字符 $\mathrm{CH}(i_1)$. 同理可知在 $y_{j+1}$ 之后有 $s_{j2}-1$ 个相同的字符 $\mathrm{CH}(i_1)$，因为 $k=\min(r_{i2},s_{j2})$,则说明 "$x_{i+1}, x_{i+2},\cdots, x_{i+k}$" = "$y_{j+1}, y_{j+2},\cdots, y_{j+k}$" ,则由引理 7 可知，由第 $l$ 层同字符对$(i,j)$所产生的关于的第 $l+1$ 层后继同字符对 $(g_{i1},h_{j1})$，可以由第 $l+k-1$ 层的$(g_{i1}+k-1,h_{j1}+k-1)$的同字符对所代替。

(2) 设关于 $\mathrm{CH}(i_1)\ \mathrm{CH}(i_2)\ \cdots\ \mathrm{CH}(i_w)$的后继同字符对 $(g_{i1},h_{j1})$、$(g_{i2},h_{j2})$ … $(g_{i(w-1)},h_{j(w-1)})$,由条件知,从 $x_{i+1}$ 至 $x_{i+ri2-1}$ 全为 $\mathrm{CH}(i_1)$, 从 $y_{i+1}$ 至 $y_{i+ri2-1}$ 全为 $\mathrm{CH}(i_1)$, $x_{i+ri2}$ 及 $y_{i+ri2}$ 全为 $\mathrm{CH}(i_2)$, … $x_{i+riw}$ 及 $y_{i+riw}$ 全为 $\mathrm{CH}(i_w)$,则子串 $x_{i+1}\ldots x_{i+riw}$ 与 $y_{i+1}\ldots y_{i+riw}$ 完全一致。由引理 7 可知如果 $X$中有子串 $x_{i+1}x_{i+2}\ldots x_{i+t}$ 及 $Y$中的子串 $y_{j+1}y_{j+2}\ldots y_{j+t}$ 完全一致，且它们中仅包含 $\mathrm{CH}(i_1)\ \mathrm{CH}(i_2)\ \cdots\ \mathrm{CH}(i_w)$这几个字符 $w\in[1,4]$，则由第 $l$ 层同字符对$(i,j)$由字符 $\mathrm{CH}(i_1)\ \cdots\ \mathrm{CH}(i_w)$所产生的层号为 $l+1$ 的后继同字符对 $(g_{i1},h_{j1})$、$(g_{i2},h_{j2})$ … $(g_{i(w-1)},h_{j(w-1)})$ 可以被层号为 $l+r_{iw}-1$ 的 $(g_{iw},h_{jw})$ 所代替。

证毕。

**例 2.4** 设 $X$= "GAATCGGACGGT",$Y$= "GAAGGCGACGCT",则同字符增量表 $SX$、$SY$，同字符后续表 $TX$、$TY$ 分别为：

$SX$:

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | 2 | 1 | 1 | 5 | 4 | 3 | 2 | 1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | C | 5 | 4 | 3 | 2 | 1 | 4 | 3 | 2 | 1 | 3 | 2 | 1 | ∞ |
| 3 | G | 1 | 5 | 4 | 3 | 2 | 1 | 1 | 3 | 2 | 1 | 1 | ∞ | ∞ |
| 4 | T | 4 | 3 | 2 | 1 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | ∞ |

*SY*:

| *i* | CH(*i*) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | 2 | 1 | 1 | 5 | 4 | 3 | 2 | 1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | C | 6 | 5 | 4 | 3 | 2 | 1 | 3 | 2 | 1 | 2 | 1 | ∞ | ∞ |
| 3 | G | 1 | 3 | 2 | 1 | 1 | 2 | 1 | 3 | 2 | 1 | ∞ | ∞ | ∞ |
| 4 | T | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | ∞ |

*TX*:

| *i* | CH(*i*) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | 2 | 2 | 3 | 8 | 8 | 8 | 8 | 8 | – | – | – | – | – |
| 2 | C | 5 | 5 | 5 | 5 | 5 | 9 | 9 | 9 | 9 | – | – | – | – |
| 3 | G | 1 | 6 | 6 | 6 | 6 | 6 | 7 | 10 | 10 | 10 | 11 | – | – |
| 4 | T | 4 | 4 | 4 | 4 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | – |

*TY*:

| *i* | CH(*i*) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | A | 2 | 2 | 3 | 8 | 8 | 8 | 8 | 8 | – | – | – | – | – |
| 2 | C | 6 | 6 | 6 | 6 | 6 | 6 | 9 | 9 | 9 | 11 | 11 | – | – |
| 3 | G | 1 | 4 | 4 | 4 | 5 | 7 | 7 | 10 | 10 | 10 | – | – | – |
| 4 | T | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | – |

对于同字符对(1,1)，我们分别得到它关于 *SX,SY* 以及 *TX,TY* 的后继同字符对：

$$(1,1)\rightarrow\begin{bmatrix}1&1\\4&5\\5&3\\3&11\end{bmatrix}\rightarrow\begin{Bmatrix}(1&1)\\(4&5)\\(5&3)\\(3&11)\end{Bmatrix}$$

$$(1,1)\rightarrow\begin{bmatrix}2&2\\5&6\\6&4\\4&12\end{bmatrix}\rightarrow\begin{Bmatrix}(2&2)\\(5&6)\\(6&4)\\(4&12)\end{Bmatrix}$$

由此，我们很容易得到 $k=\min(r_{t2}, s_{t2})=3$,根据定理 2.4 中的（1），由同字符对(1,1)所产生的层号为 2 的同字符对(2,2)可以被层号为 3 的同字符对(3,3)所代替。

$$(7,7)\rightarrow\begin{bmatrix}1&1\\2&2\\3&3\\5&5\end{bmatrix}\rightarrow\begin{Bmatrix}(1&1)\\(2&2)\\(3&3)\\(5&5)\end{Bmatrix}$$

$$(7,7)\rightarrow\begin{bmatrix}8 & 8\\ 9 & 9\\ 10 & 10\\ 12 & 12\end{bmatrix}\rightarrow\begin{Bmatrix}(8 & 8)\\ (9 & 9)\\ (10 & 10)\\ (12 & 12)\end{Bmatrix}$$

同样的，对于同字符对(7,7)，它满足定理 2.4 中（2）的条件，且 $w$=3，则由(7,7)所产生的关于 *TX,TY* 的层号为 2 的同字符对(8,8)，(9,9)可以被层号为 4 的同字符对(10,10)所代替。

## 2.5 算法框架及复杂性分析

综上所述，本文基于同字符对及剪枝和跳跃技术的并行最长公共子序列算法 FAST_LCS 分为两个阶段：搜索同字符对阶段与回溯得到最长公共子序列阶段。第一个阶段是从初始同字符对开始，利用同字符后续表不断搜索同字符对的后继的过程。在此过程中，我们采用了剪枝技术，去掉明显不能得出最优解的搜索分支。同时采用跳跃技术，跳过两序列中完全相同的两段子序列，以降低搜索空间，提高搜索速度。对于搜索到的同字符对，我们设立一个同字符对表 *pairs*，表中的每一项由四元组(*k, i, j, level, pred, state*)构成，分别表示记录字符，同字符对（*i,j*），层次号，直接前驱以及 当前的状态。表中的每一个记录都包含有两种状态，对于尚未进行搜索后继操作的同字符对，我们将它们置为 *active* 状态，否则为 *inactive* 状态。在每一轮搜索中，可以并行地对所有处于 *active* 状态的同字符对同时搜索其后继。搜索过程直至表中不存在 *active* 的同字符对为止。回溯阶段则由表中层数最大的同字符对开始，根据其前驱 *pred* 由表进行回溯直到遇到初始同字符对为止，可得到最长公共子序列。由于表中可能存在多个层数最大的同字符对，因此对于那些同字符对，我们可以并行的进行回溯，从而可以一次得到若干个最长公共子序列。因此，算法框架如下：

**算法 FAST_LCS(*X,Y*)**

**输入**：长度分别为 *m,n* 的序列 *X,Y*；

**输出**：*X,Y* 的最长公共子序列 LCS(*X,Y*)；

**Begin**

1. 构造 *TX,TY* 表；
2. 找出所有的初始同字符对：(*TX*(*k*,0),*TY*(*k*,0)), *k*=1,2,3,4；*level*=1；
3. 将所有初始同字符对(*k*,*TX*(*k*, 0),*TY*(*k*, 0), 1, $\phi$, *active*), *k*=1,2,3,4 加入到表 *pairs*

中。

/*对所有初始同字符对，赋予层号 1，*pred*= $\phi$，*state=active*/

4. **while** 表 *pairs* 中有处于 *active* 状态的项 **do**

4.1 对对表 *pairs* 中所有层号为 *level* 的同字符对进行剪枝，将所有冗余的同字符对从表中去掉

4.2 对表 *pairs* 中所有层号为 *level* 的活动同字符对(*k, i, j, level, pred, active*)并行地进行：

4.2.1 使用产生后继操作和跳跃操作产生(*k, i, j, level, pred, active*)的所有后继集合 (*k', g, h, level*+1, *k, active*)，再将该集合插入到表 *pairs* 中；

4.2.2 将(*k, i, j, level, pred, active*)的状态置为 *inactive*.

4.3 *level*= *level*+1;

**end while**

5. 计算 *r*= 表 *pairs* 中的最大层次值；

6. 对于表中的所有同字符对(*k, i, j, r, l, inactive*)并行地进行：

6.1 *pred* = *l*; *LCS*(*r*) = $x_i$.

6.2 当 *pred* $\neq \phi$ 时

6.2.1 从表 *pairs* 中得到其前驱(*pred, g, h, r', l', inactive*)

6.2.2 *pred* = *l'*; *LCS*(*r'*) = $x_g$.

**End**

设 *X,Y* 中同字符对的个数为 *L*，由于本算法要对 *X,Y* 中所有同字符对至少进行一次产生后继操作。由于剪枝技术，对于每一个同字符对不可能重复进行这样的操作，因此算法 FAST_LCS(*X,Y*)串行执行的过程即为对所有同字符对进行一次产生后继操作的过程，因此算法的时间复杂度为 *O*(*L*)。由于表 *pairs* 记录所需处理的同字符对，其所占的存储空间为 *O*(*L*)，同字符后续表 *TX,TY* 的存储空间分别为 4*(*n*+1)及 4*(*m*+1)。本文算法所需的存储复杂度为 max{4*(*n*+1)+4*(*m*+1),*L*}。在并行计算中，由于对每一层的计算是并行的，可以在 *O*(1)时间内完成，因此并行计算的时间就等于同字符对的最大层次值。由引理 2.1 可知，*X,Y* 的最长公共子序列长度为 |LCS(*X,Y*)| 即最大层次值也等于|LCS(*X,Y*)|，所以并行计算的时间复杂度为 *O*(|LCS(*X,Y*)|)。

## 2.6 扩展至多序列的最长公共子序列问题

我们很容易将算法 FAST_LCS 扩展为多序列的 LCS 问题。设欲匹配的字串分

别为 $X_1, X_2, \ldots, X_n$，其中 $X_i=(x_{i1}, x_{i2}, \ldots, x_{i,n_i})$，$n_i$ 是 $X_i$ 的长度，$x_i \in \{A,C,G,T\}$。与双序列类似，为了找到多序列的最长公共子序列，我们首先也要对 $X_1, X_2, \ldots, X_n$ 分别作一个同字符后续表，分别记为 $TX_1, TX_2, TX_3 \ldots TX_n$ 其中，$TX_s$ 为 $4*(n_i+1)$的二维数组。$TX_s(i, j)$ 为：$X_s$ 中第 $j$ 个位置上下一个为 $i$ 的字符；$s$=1, 2…n。类似于双序列中的同字符对，我们对多序列 $X_1, X_2, \ldots, X_n$ 定义同字符组为：若有 $x_{1,i_1}=x_{2,i_2}=\ldots=x_{n,i_n}$，则记$(i_1,i_2,\ldots,i_n)$为一同字符组。相应地，我们定义同字符组的前驱，后继和层次的概念。

与双序列的情形类似，在多序列 LCS 问题的并行算法中，我们利用后续表可以并行地产生所有初始同字符组的所有直接后继，随后又可以并行地产生这些后继的所有直接后继。如此重复这些产生直接后继的操作直至不能再产生任何后继为止。对某同字符组$(i_1, i_2, \ldots, i_n) \in S(X_1, X_2, \ldots, X_n)$，由$(i_1, i_2, \ldots, i_n)$产生其后继同字符组的操作，可表示为：

$$(i_1,i_2,\ldots,i_n) \rightarrow \{(TX_1(k,i_1), TX_2(k,i_2), \ldots, TX_n(k,i_n)) \mid k=1,2,3,4, TX_j(k,i_j) \neq -1, j=1,2,\ldots,n\} \quad (2.8)$$

由(2.8)我们可以看出该操作将 $TX_j(j=1,2,\ldots,n)$的 $i_j$ 列组合在一起就得到了$(i_1, i_2, \ldots, i_n)$的后继同字符组。

**例 2.5**　设 $n$=3, 三个序列分别为 $X_1$ ="TGCATA"，$X_2$ ="ATCTGAT"，$X_3$ = "CTGATTC"．则后续表 $TX_1$，$TX_2$ 和 $TX_3$ 分别为:

$TX_1$:

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| 1 | A | 4 | 4 | 4 | 4 | 6 | 6 | – |
| 2 | C | 3 | 3 | 3 | – | – | – | – |
| 3 | G | 2 | 2 | – | – | – | – | – |
| 4 | T | 1 | 5 | 5 | 5 | 5 | – | – |

$TX_2$:

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | A | 1 | 6 | 6 | 6 | 6 | 6 | – | – |
| 2 | C | 3 | 3 | 3 | – | – | – | – | – |
| 3 | G | 5 | 5 | 5 | 5 | 5 | – | – | – |
| 4 | T | 2 | 2 | 4 | 4 | 7 | 7 | 7 | – |

$TX_3$:

| $i$ | CH($i$) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | A | 4 | 4 | 4 | 4 | – | – | – | – |
| 2 | C | 1 | 7 | 7 | 7 | 7 | 7 | 7 | – |
| 3 | G | 3 | 3 | 3 | – | – | – | – | – |
| 4 | T | 2 | 2 | 5 | 5 | 5 | 6 | – | – |

我们将这三张表的第 0 列组合在一起就得到了初始同字符组：

$$(0,0,0)\rightarrow\begin{bmatrix}4&1&4\\3&3&1\\2&5&3\\1&2&2\end{bmatrix}=\begin{Bmatrix}(4,1,4)\\(3,3,1)\\(2,5,3)\\(1,2,2)\end{Bmatrix}$$

随后，对于(1,2,2)，我们继续产生其后继：

$$(1,2,2)\rightarrow\begin{bmatrix}4&6&4\\3&3&7\\2&5&3\\5&4&5\end{bmatrix}=\begin{Bmatrix}(4,6,4)\\(3,3,7)\\(2,5,3)\\(5,4,5)\end{Bmatrix}$$

由此，我们得到(1,2,2)的四个后继：(4,6,4), (3,3,7), (2,5,3) 和 (5,4,5)。

**引理 2.4.** 对同字符组$(i_1, i_2, \dots, i_n)$, 用上述方法可以产生所有的后继。

引理 2.4 的证明类似于引理 2.2。

由引理 2.4 可知，在每一层上，通过产生后继操作，我们可以得到所有同字符组的后继。该操作由初始同字符组开始可以产生所有后继同字符组。在产生后继的过程中，我们同样可以利用剪枝技术将那些明显不能得出最长公共子序列的同字符组删除。也可以采用跳跃技术越过那些相同的子序列，以缩小搜索空间、加快搜索速度。由双序列的 LCS 问题，我们很容易得到多序列关于剪枝和跳跃技术的类似定理和推论。

**定理 2.5.** 如果在同一层上有同字符组$(i_1, i_2, \dots, i_n)$ 和$(j_1, j_2, \dots, j_n)$，且$(j_1, j_2, \dots, j_n) > (i_1, i_2, \dots, i_n)$,则剪去$(j_1, j_2, \dots, j_n)$不影响算法得到最优解。

定理 2.5 的证明类似于定理 2.1。由定理 2.5 我们可以剪去所有冗余的同字符组。

例如在例 2.5 中，在得到(1,2,2)的后继后，同上所述,因为$(2,5,3)\prec(4,6,4)$，且(2,5,3)和(4,6,4)在同一层上，我们可以将(4,6,4)剪去。又如对于初始同字符组(0,0,0)的四个后继：(4,1,4), (3,3,1), (2,5,3)和(1,2,2)。他们都处于同一层，且有$(1,2,2)\prec(2,5,3)$，则我们可以剪去(2,5,3)。

**定理 2.6.** 如果在同一层上有同字符组$(i_1, i_2, \dots, i_n)$ 和 $(j_1, j_2, \dots, j_n)$，且存在 $r \in [1,n]$，满足 $i_k<j_k$, $(k=1,2,\dots,r)$, 且 $i_k=j_k$ $(k=r+1,\dots,n)$,则剪去$(j_1, j_2, \dots, j_n)$不影响算法得到最优解。

由定理 2.6，我们可以得到以下推论：

**推论 2.2.** 若同字符组$(i_{j1}, i_{j2}, \dots, i_{jn})$, $j=1,2,\dots,s$ 在同一层上，且存在 $r\in[1,n]$, 满足 $i_{1k}<i_{jk}$,$(j=2,3,\dots,n,\ k=1,2,\dots,r)$, 且 $i_{1k}=i_{jk}$ $(j=2,3,\dots,r,\ k=r+1,\dots,n)$, 则 $(i_{j1}, i_{j2}, \dots, i_{jn})$, $j=2,\dots,s$ 可被剪去。

设序列 $X_1, X_2, \dots, X_n$ 的同字符组的个数为 $L$。由于本算法要对所有同字符组至

少进行一次产生后继操作，因此本算法对于序列 $X_1, X_2, \ldots, X_n$ 的串行时间复杂度为 $O(L)$。我们使用表 *tuples* 记录所需处理的同字符对，其所占的存储空间为 $O(L)$，同字符后续表 $TX_j$, $(j=1,2,\ldots,n)$的存储空间为 $4\sum_{i=1}^{n}(n_i+1)$。因此本文算法所需的存储复杂度为 $\max\{4\sum_{i=1}^{n}(n_i+1), L\}$。在并行计算中，由于可以将每一个同字符组分配到一个处理器上进行处理，因此对于同字符组的处理可以并行执行。因此每一层的处理可以在 $O(1)$时间内完成，而并行计算的时间就等于同字符组的最大层次值即序列 $X_1, X_2, \ldots, X_n$ 最长公共子序列的长度。所以多序列并行计算的时间复杂度为 $O(|\mathrm{LCS}(X_1, X_2, \ldots, X_n)|)$。

必须指出，在多序列 LCS 问题的大部分算法中，时间复杂度与序列个数之间有着很大的关系。例如，我们用 Smith－Waterman 算法来解决多序列的 LCS 问题，其时间复杂度为 $O(2^n-1)\prod_{i=1}^{n}n_i$,这里 $n$ 是指序列的个数。显然当 $n$ 较大时，该算法在实际上无法完成。而我们算法的串行时间复杂度为 $O(L)$，并行时间复杂度为 $O(|\mathrm{LCS}(X_1, X_2, \ldots, X_n)|)$，都与序列个数 $n$ 无关。这也就说明对于处理大规模序列的 LCS 问题，我们的算法更为高效。

## 2.7 实验结果及分析

### 2.7.1 双序列串行计算结果

我们从 tigr[95]数据库中抽取了稻谷基因序列对本文算法 FAST_LCS 进行了实验，并将本文算法分别与目前应用最广泛的 LCS 算法 Smith－Waterman 算法 [96-97] 及 FASTA 算法[98-99]在速度上进行了比较，由于本文算法与 Smith－Waterman 算法都可以得到精确解，因此我们将 FAST_LCS 算法与 Smith－Waterman 算法在速度上作了比较，同样的，我们在计算时间相同的前提下，也将本文算法与 FASTA 算法在精度上作了比较。

表 2.1 给出了对于不同长度的基因组序列，本文算法 FAST_LCS 与 Smith－Waterman 算法在计算速度上的比较。由于对一对序列进行测试所需的时间非常小，因此在算法的速度上很难作出比较。为方便起见，我们将长度相似的若干对序列作

为一组进行实验。我们分别使用这两种算法对五组序列对进行了测试，每一组都包含有 100 对序列，测试总时间如表 2.1 所示：

表 2.1. 本文算法 FAST_LCS 与 Smith－Waterman 算法在计算速度上的比较

| 序列名称 | 长度 $l$ | 序列对个数 | FAST_LCS 的时间 (S) | | S-W algorithm 的时间(S) | |
|---|---|---|---|---|---|---|
| | | | 总时间 | 平均时间 | 总时间 | 平均时间 |
| gi\|21466196～gi\|21466195<br>…<br>gi\|21466168～gi\|21466167<br>gi\|21466166～gi\|30250556<br>gi\|30230255～gi\|30230254<br>gi\|30229613～gi\|30229612<br>…<br>gi\|30229449～gi\|30229448 | $0 \leqslant l \leqslant 50$ | 100 | 0.49 | 0.0049 | 1.09 | 0.0109 |
| gi\|30229047～gi\|30229046<br>…<br>gi\|30229001～gi\|30229000<br>gi\|30228999～gi\|30228998<br>…<br>gi\|30228849～gi\|30228848 | $50 \leqslant l \leqslant 100$ | 100 | 5.88 | 0.0588 | 11.55 | 0.1155 |
| gi\|30229447～gi\|30229446<br>…<br>gi\|30229249～gi\|30229248 | $10 \leqslant l \leqslant 150$ | 100 | 29.41 | 0.2941 | 65.95 | 0.6595 |
| gi\|30228846～gi\|30228845<br>…<br>gi\|30228648～gi\|30228647 | $15 \leqslant l \leqslant 200$ | 100 | 94.11 | 0.9411 | 172.213 | 1.7213 |

| | | | | | | |
|---|---|---|---|---|---|---|
| gi\|30229247～<br>gi\|30229246<br>…<br>gi\|30229049～<br>gi\|30229048 | $20 \leqslant l \leqslant 250$ | 100 | 230.51 | 2.3051 | 425.16 | 4.2516 |

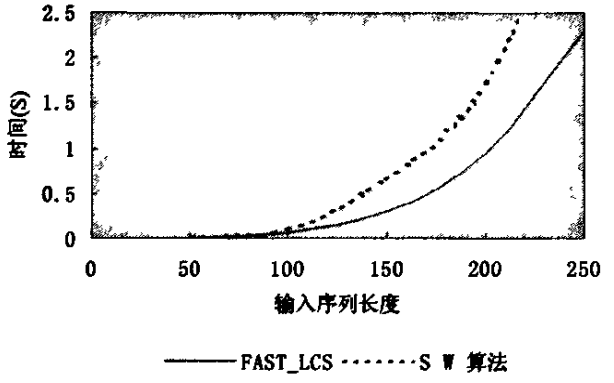



图 2.3 本文算法 FAST_LCS 与 Smith－Waterman 算法在计算时间上的比较

由表 2.1 和图 2.3 可以看出，对于不同长度的序列，本文算法要明显快于 Smith-Waterman 算法。在输入序列长度小于 150 时，速度变化较为缓慢，而在序列长度大于 150 之后，速度变化异常迅速，本文算法的速度大大超过 Smith-Waterman 算法。由此可见，对于长度列的 LCS 问题，本文算法和 Smith-Waterman 算法虽然都能得精确的解，但本文算法的速度更快、效率更高。

同样地，我们用本文算法与 FASTA 算法在计算时间相同的情况下，对计算结果精度进行了比较，这里精度是指：

$$\text{精度} = \frac{\text{算法求得的公共子序列地长度}}{\text{正确匹配中的最长公共子序列长度}}$$

两种算法的精度的比较结果如图 2.4 所示。由图 2.4 可见，不管输入序列长度如何增加，本文算法都能够取得准确的结果，而 FASTA 算法则随着序列长度的增加，精度明显下降。因此本文算法的精度要明显优于 FASTA 算法。

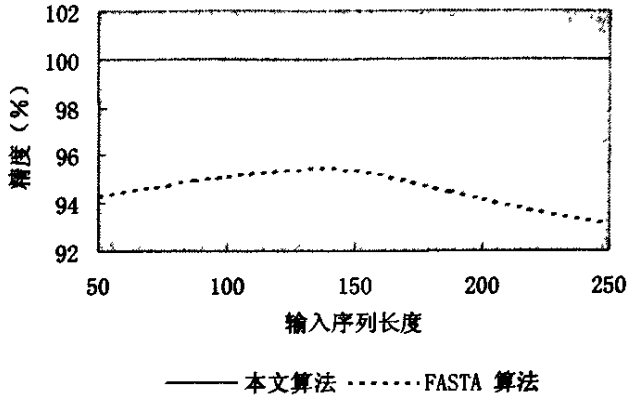



图 2.4 本文算法 FAST_LCS 与 FASTA 算法在精度上的比较

### 2.7.2 多序列串行计算结果

我们将本文算法 FAST_LCS 在多序列上做了实验，并将它与目前应用比较普遍的多序列算法 CLUSTAL-W[34]做了比较。图 2.5 和图 2.6 表示了 FAST_LCS 与 CLUSTAL-W 关于计算时间的比较。表 2.2 列出了这两个算法处理长度为 50 的多序列分别需要的计算时间。我们分别对 5 个含有不同序列个数的序列集做了测试。

由图 2.5 和表 2.2 可以看出，在处理含不同序列个数的序列集时，本文算法 FAST_LCS 要比 CLUSTAL-W 算法快得多。尤其是在序列个数大于 5 时，本文算法的计算时间要远远小于 CLUSTAL-W 算法。

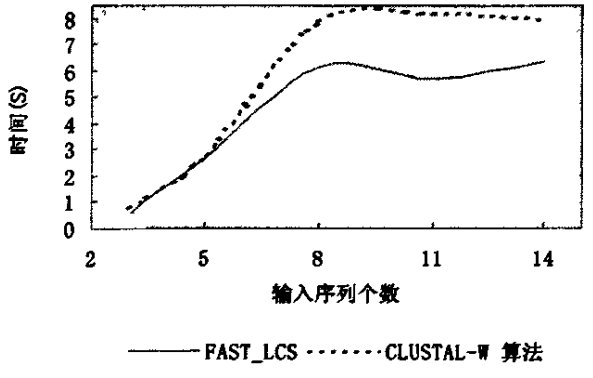



图 2.5 FAST_LCS 与 CLUSTAL-W 在含不同个数序列集上的计算时间的比较

表 2.2 FAST_LCS 与 CLUSTAL-W 在含不同个数序列集上的计算时间的比较

| 序列名称 | 序列个数 | FAST_LCS 的时间 (S) | Clustal-W 的时间 (S) |
|---|---|---|---|
| gi\|21466194 … gi\|21466196 | 3 | 0.609 | 0.804 |
| gi\|21466192 … gi\|21466196 | 5 | 2.656 | 2.732 |
| gi\|21466189 … gi\|21466196 | 8 | 6.14 | 7.91 |
| gi\|21466186 … gi\|21466196 | 11 | 5.71 | 8.20 |
| gi\|21466183 … gi\|21466196 | 14 | 6.34 | 8.49 |

表 2.3 列出了这两个算法测试 5 个不同长度的序列集所需的计算时间。图 2.6 列出了两个算法关于计算时间的比较。由表 2.3 和图 2.6 我们可以看出，对于不同长度的序列集，本文算法 FAST_LCS 要明显快于 CLUSTAL-W 算法。

表 2.3 FAST_LCS 与 CLUSTAL-W 在不同长度序列集上的计算时间的比较

| Algorithm | The length of input sequences | | | | |
|---|---|---|---|---|---|
| | 20 | 30 | 50 | 60 | 80 |
| Time of FAST_LCS (S) | 0.109 | 0.391 | 2.656 | 3.516 | 6.166 |
| Time of Clustal-W (S) | 0.312 | 1.053 | 2.732 | 3.612 | 5.992 |

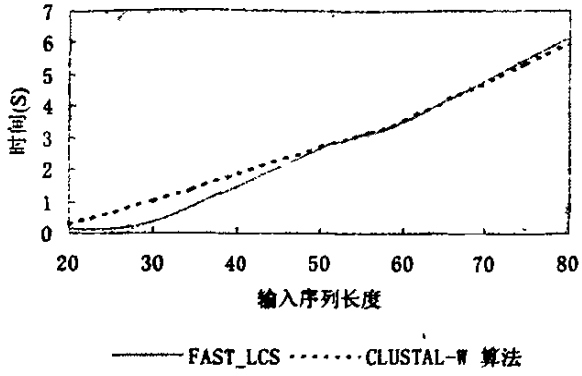



图 2.6 FAST_LCS 与 CLUSTAL-W 在不同长度序列集上的计算时间的比较

同样的，我们也将本文算法 FAST_LCS 与 CLUSTAL_W 算法在精度上做了比较。图 2.7 是 FAST_LCS 与 CLUSTAL-W 在对含不同序列个数的序列集进行实验得到的关于计算精度的比较。而图 2.8 则表示的是这两个算法对不同长度的序列进行实验得到的计算精度的比较。由这两张图我们可以知道，无论序列的长度或序列个数如何增加，我们的算法始终都能保证得到精确解。而 CLUSTAL-W 算法的计算精度则随着序列长度或序列个数的增加而逐渐下降，因此本文算法的精度要远远高于 CLUSTAL-W 算法。

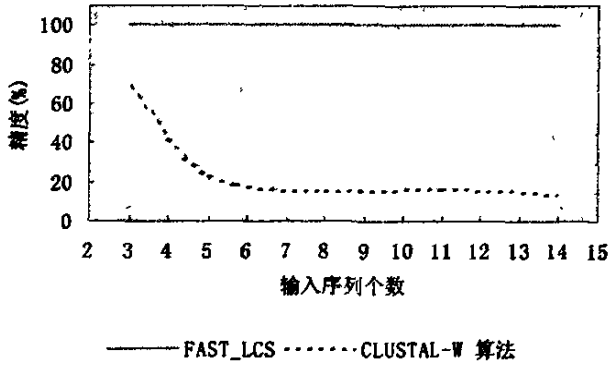



图 2.7 FAST_LCS 与 CLUSTAL-W 在含不同个数序列集上的计算时间的比较

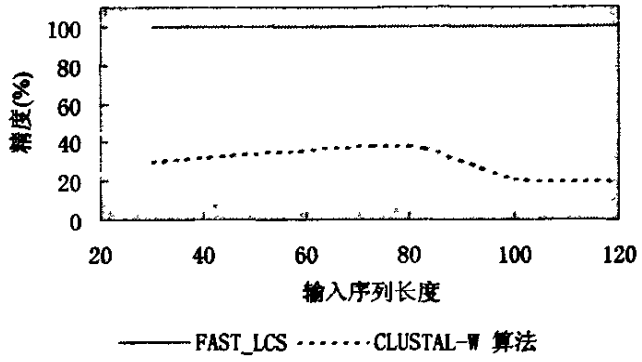



图 2.8 FAST_LCS 与 CLUSTAL-W 在不同长度序列集上的计算精度的比较

### 2.7.3 双序列并行计算结果

我们同样也从 tigr[95]数据库中抽取了稻谷基因序列对本文算法 FAST_LCS 在 MPP 处理机深腾 1800 上采用 MPI 绑定 C 语言编程对该并行算法实现。在并行算法中，我们将处于 *active* 的同字符对分配给若干个处理机，随后每个处理机并行的执行后继产生操作。我们使用不同个数的处理器进行实验，结果如图 2.9 所示。我们分别测试了三对基因序列。表 2.4 列出了序列的名称，长度和计算时间。由图 2.9 和表 2.4 可知对于给定长度的生物序列，随着处理器个数的增加，计算速度会有大大的提高。但由于各个处理器间的额外通信将会导致算法时间的增加，因此算法的加速比不可能严格的随着处理器的增加而线性的增长。但这是符合并行处理的加速比性能的 Amdahl 定律的。

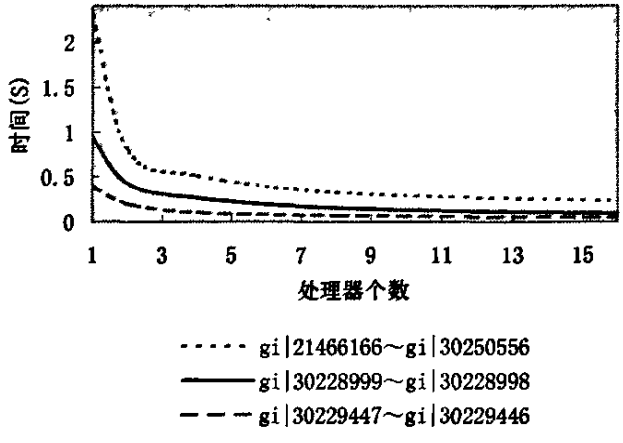



图 2.9 并行计算的时间随处理机个数增加的变化情况

### 2.7.4 多序列并行计算结果

我们使用不同个数的处理器对多序列进行实验，结果如图 3.0 所示。3.0 分别表示 3 条、5 条和 8 条基因序列的并行计算时间随处理机个数增加的变化情形。由图 3.0 可知，对于给定长度的生物序列，随着处理器个数的增加，计算速度会有大大的提高。但是算法的加速比不能严格地随着处理器的增加而线性地增长，这也是符合并行处理的加速比性能的 Amdahl 定律的。

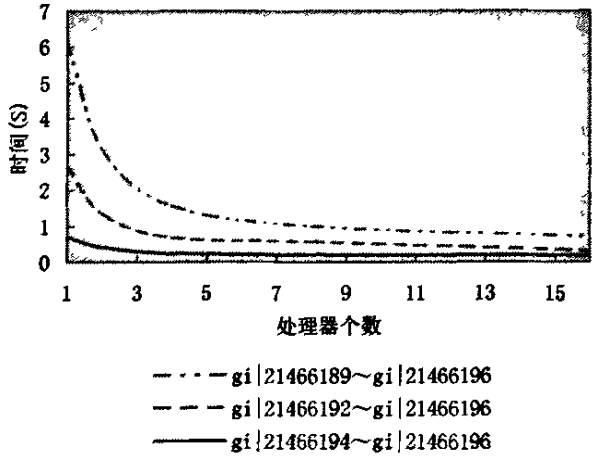



图 3.0 多序列的并行计算时间随处理机个数增加的变化情况

表 2.4 多序列的并行计算时间随处理机个数增加的变化情况

| 序列名称 | 序列条数 | 使用不同处理器的计算时间 | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 4 | 6 | 8 |
| gi\|21466189 ~ gi\|21466196 | 8 | 6.14 | 3.1093 | 1.5849 | 1.0063 | 0.7154 |
| gi\|21466192 ~ gi\|21466196 | 5 | 2.656 | 1.3579 | 0.6921 | 0.5707 | 0.3173 |
| gi\|21466194 ~ gi\|21466196 | 3 | 0.709 | 0.4214 | 0.25382 | 0.20445 | 0.19042 |

# 第三章　生物序列的拼接的并行计算

生物基因测序是生物信息学一个重要的研究领域，基因组计划的目标是获得所研究的生物的全基因组序列，而序列拼接是基因组测序阶段生物信息学研究的最基本、最重要的问题。本章介绍并分析了目前国内外对生物序列拼接问题的研究现状，对该问题提出了高效的并行算法。在该算法中，我们提出了后缀索引的概念，用来查找匹配度最高且最长的后缀，进而建立有向图；使用蚁群算法寻找该有向图的最长的哈密尔顿回路以找出拼接。

## 3.1 生物序列拼接问题的研究回顾

生物基因测序[100-103]是生物信息学一个重要的研究领域，提高生物序列拼接的精度和速度对于生物基因测序有着重大意义。生物的 DNA 由 A,C,G,T 四种碱基组成。由于 DNA 包含的碱基数量极其巨大,无法采用某种手段一次测出，因此，在生物学上普遍采用将较长的序列做多个克隆(通常是 5 到 10 个,即采用 5 到 10 倍的覆盖率),将这些克隆用酶随机切割成小片断,然后由测序反应测出小片断上各个位置上的碱基并通过自动测序仪直观地读出 A,C,G,T 的序列。这些序列经过电脑加工、检查质量,用序列拼装程序将相互重叠的序列拼接起来。如果中间有“空洞”(gap),还要将这些“空洞”用各种技术“补”起来,最后形成一个大片断克隆的完整序列。如图 3.1 所示：

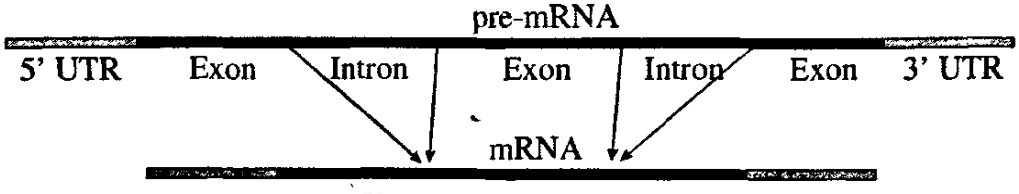



图 3.1 生物序列的拼接

### 3.1.1 序列拼接问题的描述

序列片段拼接的定义如下：给定一组取自特定字母表的字符串集合*F*，寻找一

个最短的字符串$s$，使得$F$ 中的每一个字符串都是$s$ 的一个连续子串。这里，集合$F$中的字符串相当于待拼接的序列片段，而$s$ 则是序列片段拼接的结果。

假设有下列4 个DNA 序列片段ACCGT、CGTGC、TTAC 和TACCGT，并且已知目标序列的长度约为10，如图3.2（a）所示，则可以按图3.2（b）所示的方式拼接这4 个片段。

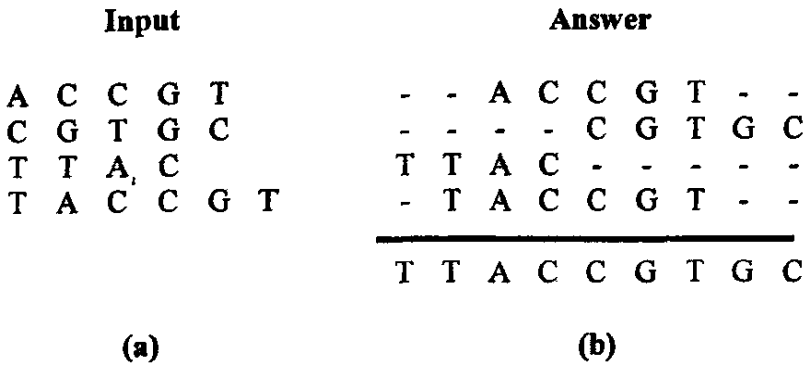



图3.2 序列片段的拼接

将输入的序列片段进行两两比对，但是，这里序列比对的目标与基本的两两比对问题有所不同，现在的目标是寻找一个序列的尾端（后缀）与另一个序列的前端（前缀）相同（或者非常相似）的部分。这实际上是一种局部序列比对，也就是将一个序列的后缀与另一个序列的前缀进行对比，忽略两端的空白字符。指导片段拼接的因素就是片段之间的覆盖。所谓片段之间的覆盖，是指一个片段的末端与另一个片段的前端相同（或相似）的部分。通过各个片段之间的覆盖，可以将所有片段连接起来。这实际上相当于将每个片段进行相对定位，得到各片段的布局，逐步确定目标序列。这也可以看成是序列片段的多重比对。图3.2（b）横线下面的是拼接的目标序列，也就是序列拼接的结果。对于每一列，提取出现频率最大的一个字符。

上面的例子是一种理想的情况，而实际上，拼接问题非常复杂。除了序列片段很长之外，还有4个主要问题：

第一个问题是碱基标识错误，如在序列片段中出现的碱基替换、插入和删除。在图3.3 中，待拼接的片段有4个，而与上面的例子相比较，在第4个片段上有一个A到G的替换。在现实中，测序碱基标识错误发生的频率大约为0.1%，发生在3’端的可能性比较大。从图3.3 中的例子看出，在一定程度上可以对有标识错误的片段进行拼接。但是，计算机必须能够处理这些错误，应具有容错功能，在进行子串比较时不一定要求完全匹配，而只要子串达到一定的相似度即可。

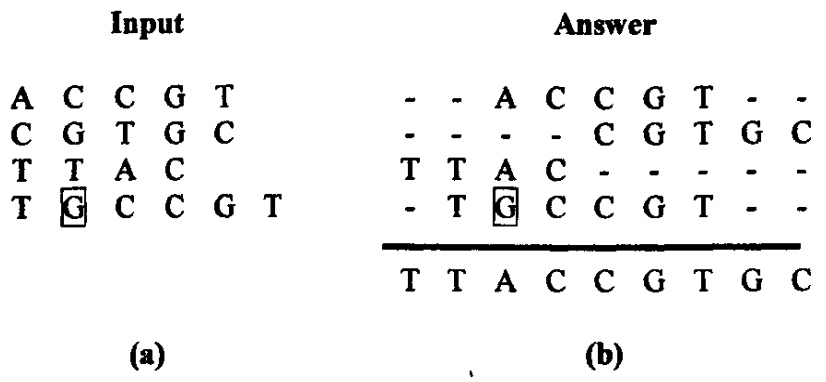



图3.3 具有碱基标识错误的片段拼接

序列拼接的第二个问题是不知道片段的方向。一个片段可能来自于目标DNA的某一条链，但是我们不知道它究竟来自于哪一条链。如果一个片段是一条链的子串（子序列），那么根据互补原则，该片段的反向互补片段是另一条链的子串。于是，对于一条输入的片段，在进行拼接时，既可以用其本身，也可以用其反向互补片段，如图3.4 所示。在进行片段拼接时，应能够选择正确方向上的片段。

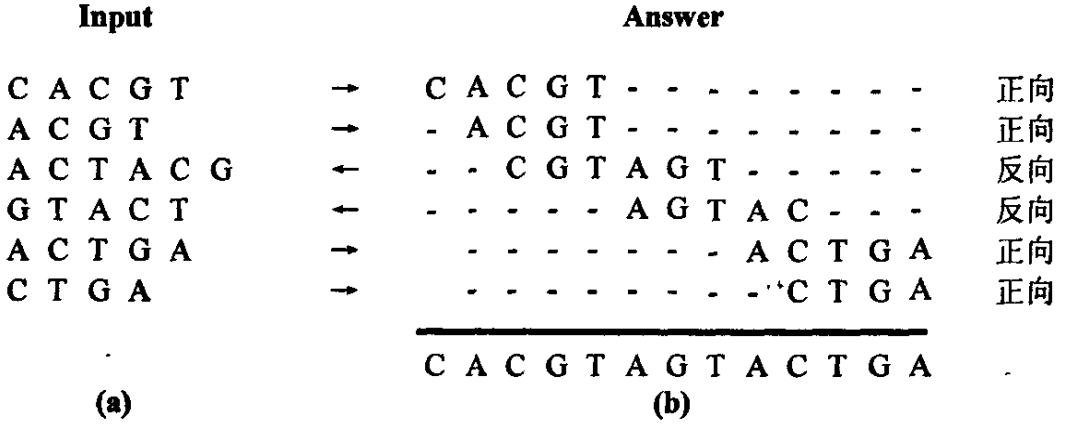



图 3.4　利用不同方向的片段进行拼接

第三个问题是存在重复区域。重复区域是目标序列中多次出现的子序列。短的重复，即包含于一个片段的重复，对于片段拼接没有太大的问题。问题是有的重复区域太长，超过片段的边界，这会使片段拼接变得不确定。在如图3.5 中，被重复序列$X$ 所包围的两个区域B和C，其位置可以互换，从而形成由于重复序列$X$ 引起两种不同的拼接，而这两种拼接都满足各片段之间的约束条件。

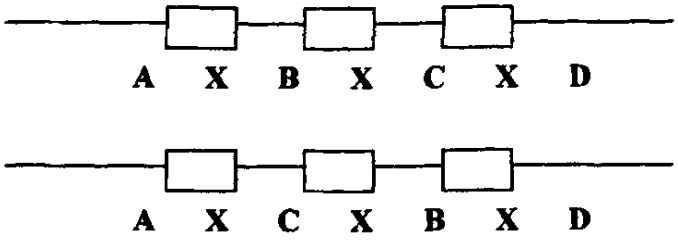



图 3.5 由于重复序列引起的拼接不确定性

第四个问题是缺少覆盖。令目标序列上某个位点的覆盖强度为覆盖此位点的片段的个数。这个定义非常简单，但是无法计算，因为我们不知道每个片段的确切位置。当然，可以计算各位点的平均覆盖强度，即加和所有片段的长度，再除以目标序列长度的估计值。如果对于目标序列上某个位点，覆盖该点的序列片段个数为0，那么，就没有相关的序列信息来重建该点附近的目标序列，如图3.6 所示。在这种情况下，所能做的就是对每段连续被覆盖区域分别进行重建。被分开的连续区域称为连续交叠群。

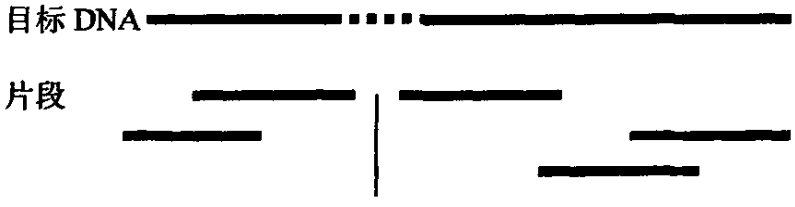



图 3.6 连续交叠群示意

### 3.1.2 序列拼接的目标序列的模型

序列片段拼接过程一般包括三个步骤。首先进行序列片段的两两比较，确定可能的片段之间的覆盖（或者重叠）；在此基础上，确定所有片段统一的覆盖模式，即确定各个序列片段的相对位置；最后确定片段拼接结果，即确定目标序列。

确定所有片段统一的覆盖模式是序列片段拼接过程中最困难的一步，一旦确定了各个序列片段的相对位置，将很容易得到目标序列。下面介绍根据序列片段拼接目标序列的三种模型，即最短公共超串模型、重建模型以及多重连续区模型。

（1）最短公共超串模型

所谓最短公共超串问题就是给定一个字符串集合$F$，求出一个最短的字符串$s$，使得对于所有属于$F$的字符串$f$，$s$是$f$ 的超串（或者$f$是$s$的子串）。例如，设$F$={ACT,

CTA, AGT}，则$s$=ACTAGT是$F$的公共超串，因为$s$包含了$F$中所有的片段（作为子串）。同时也可以证明$s$是包含$F$中各片段的最短公共超串。

就序列片段拼接而言，集合$F$对应于各序列片段，而$s$则代表目标序列。注意，由于$s$必须是各片段严格的超串，因此不允许片段的实验误差，另外，各片段的方向必须是已知的。当然，这是假设的理想情况，在实际工作中这些先决条件很难同时满足。

严格求解最短公共超串计算量较大，在具体实现时，往往采用近似求解算法。

（2）重建模型

重建模型考虑到片段的误差和未知方向的问题。在这种模型中，对于判别子串的序列比较不需要完全匹配，而是根据相似程度进行处理，采用近似子串的概念。这里使用编辑距离，对于任何插入、删除或替换的距离值（代价）均设置为1，而字符匹配的距离值为0。这里的距离称为子串编辑距离，并以$d_s$表示。假设$f$、$g$ 是代表两条序列的字符串，$f$ 作为$g$ 的近似子串的代价为：

$$d_s(f,g)=\min d_s\in S(g)(f,s)$$

其中，$S(g)$代表$g$所有子串的集合，$d$为编辑距离。注意，字串编辑距离是不对称的，即$d_s(f,g)\neq d_s(g,f)$。

设$\xi$是一个介于0和1之间的数，称串$f$是在误差水平$\xi$下$s$的近似子串，如果

$$d_s(f,s)\leq\xi|f|$$

这里$|f|$是$f$的长度。上述不等式表明对于所有的$f$，允许有一个平均的误差。

重建模型可表示如下：给定一个字符串集合$F$，求一个最短的字符串$s$，使得对于所有属于$F$的字符串$f$，下式成立：

$$\min(d_s(f,s),d_s(f',s))\leq\xi|f|$$

其中，$f'$是$f$ 的反向互补串。重建模型主要思想是寻找一个尽可能短的串$s$，使$f$或其反向互补串成为$s$的近似子串。

该模型可以同时处理序列误差、序列方向问题，但是不能处理目标序列中的重复区域问题和缺少覆盖的问题。

（3）多重连续区模型

首先考虑没有误差的多重连续区模型。给定一个序列片段的集合，考察其相应的多重比对，该比对必须是每一列仅包含一种字符。同时，在处理片段方向问题时，要求每个片段本身或者其反向互补片段两者之一处于多重比对中。

设这样的多重比对为$\alpha$，将每一列进行编号，从1 到$|\alpha|$（$|\alpha|$代表多重比对列的个数）。按照这种编号，令每个片段$f$ 的左端点所对应的编号是$l(f)$，而右端点对应的编号是$r(f)$，于是$|f|=r(f)-l(f)+1$。设$f$ 和$g$ 分别为两个序列片段，如果区间

$[l(f)\cdots r(f)]$和区间$[l(g)\cdots r(g)]$相交，则称$f$ 和$g$ 在多重比对$\alpha$中相互覆盖，而非空的相交部分$[l(f)\cdots r(f)]\cap[l(g)\cdots r(g)]$称为$f$ 和$g$ 之间的覆盖。覆盖的长度等于交叠部分的长度。

最弱连接是一个所有连接中交叠部分最小的连接。另外，称一个多重序列比对是$t$-contig，如果其最弱连接的交叠长度至少为$t$。如果能够根据序列片段集合$F$ 构造一个$t$-contig，称F允许一个$t$-contig。

多重连续区模型的形式化描述如下：给定一个片段集合$F$和一个整数$t$（$\geq 0$），将$F$分割为最小数目的子集$C_i$，$1\leq i\leq k$，每个$C_i$允许一个$t$-contig。

在基本的多重连续区模型的基础上，考虑允许误差的多重连续区模型。其形式定义如下：给定一个片段集合$F$、一个整数$t$（$\geq 0$）和一个介于0和1之间的误差值$\xi$，将$F$分割为最小数目的子集$C_i$，$1\leq i\leq k$，每个$C_i$在$\xi$的误差范围内允许一个$t$-contig。

### 3.1.3 生物序列拼接问题的算法研究

由于序列拼接问题[104-105]的复杂性以及在生物信息学中的重要性，人们对这一问题进行了深入的研究，并提出了许多算法。总的来说，序列拼接算法可以大致分成两类：一类是利用 Hamiltonian path 的方法，另一类是利用 Eulerian path 的方法。两类算法都是采用了基于图论的近似方法，其主要思想是根据片段间的重叠信息构造某种类型的图，然后利用基于图论的一些算法得到结果，下面几节将对这个问题加以讨论。

#### 3.1.3.1 基于 Hamilton 路径方法的拼接算法

基于 Hamilton 路径方法的拼接算法基本思想是：首先将所有的片段构成一个有向图 $G$，每个片段看成一个结点，如果两个片段之间存在有重叠，那么在相应的结点之间就存在有一条边。然后通过寻找经过每个片段一次且仅一次的一条路径，就将序列拼接问题转化成 Hamilton 路径问题。这种方法可以分为如下三步：

1）找出序列片段间的重叠信息；

2）将存在重叠的片段组合起来，形成一个连续交叠群结构；

3）根据片段中每个碱基的质量值，在连续交叠群结构中寻找一条最终序列，称作“Consensus”序列。

这一类算法主要有 Phrap[36],TIGR[37],CAP3/CAP4[40-41],GigAssemble[44]等，它们都遵循"Overlap-Layout -Consensus"[106-109]三步曲，但是都有一些细微的差别和特点。Phrap 是由美国华盛顿大学分子生物技术学院的 Phil Green 和 Brent Ewing 开发的软件包。我们以 Phrap 为例，详细描述此类算法的三个步骤：

(1) Overlap：如果两个序列片段有重叠区域，那么它们之间必定有一定长度的精确匹配区域，称为 Match。

因此算法首先寻找存在长度至少为某个阈值的 Match 的所有相关的片段对，对每个片段对来说，一个 Match 就在 Smith－Waterman 比对矩阵上定义了一条对角线，将这条对角线扩展成矩阵上的一个带状区域，对这个带状区域运行 Smith－Waterman 算法，如果 Smith－Waterman 得分超过某个阈值，则认为是一个重叠区域。Phrap 寻找 Match 的算法是：首先建立一个指针集合，包含指向每个片段中的每个字符位置的指针，然后按照字母序列对集合中指针排序，最后在相邻的指针中寻找 Match。

(2) Layout：在这一步中，Phrap 首先对重叠区域进行打分，衡量两个片段之间的不同是真不同，还是由于测序错误造成的，然后将所有重叠区域按照该打分值由高到低进行排序。

该步操作中，Phrap 算法定义了一种称为连续交叠群的结构来表示一组片段的组合关系。连续交叠群包含一个片段集合 $F$，这些片段最终将形成一条模糊的序列 $S$（因为某些重叠并不是完全匹配的），而每一条片段相对于 $S$ 起始位置都有一个偏移量。片段在进行组合时，每个片段先形成一个单独的连续交叠群，当发现某两个片段 $f_i$ 和 $f_j$ 存在有重叠，即可以形成一个实际的拼接时，Phrap 算法将 Contig($f_i$)和 Contig($f_j$)进行合并，形成一个新的结构 Contig($f_i$, $f_j$)。我们用 $F(f_i)$表示 contig($f_i$)中的片段集合，用 $S(f_i)$表示 $F(f_i)$所形成的序列。

在将 Contig($f_i$)和 Contig($f_j$)进行合并形成 Contig($f_i$, $f_j$)时，首先需要对原结构 Contig($f_i$)和 Contig($f_j$)的片段集合 $F_{fi}$ 和 $F_{fj}$ 进行并操作形成 Contig($f_i$, $f_j$)中的片段集合 $F(f_i, f_j)$，对原结构所表达的序列 $S_{fi}$ 和 $S_{fj}$ 通过合并重叠形成新的序列 $S(f_i, f_j)$，并重新计算 Contig($f_i$, $f_j$)中每条片段相对于 $S(f_i, f_j)$的偏移位置，最后还需要删除原来的结构 Contig($f_i$)和 Contig($f_j$)。这样，Phrap 算法将存在重叠的片段组合起来形成一个连续交叠群结构。

(3) Consensus：将各个片段的高质量部分拼接起来构成最终的一致性序列。在将存在重叠的片段形成一种组合关系后，Phrap 利用一种称作"Mosaic"的方法将连续交叠群结构中一些高质量的序列片段连接起来形成一条一致性序列。使用这种方法需要先在连续交叠群结构上构造一个加权有向图 $G$，将求解一致性序列问题转

化为在 $G$ 中寻找一条从起始节点到终止节点的权重最大的路径。加权有向图 G 的具体构造方法如下：

1）节点：在连续交叠群结构的片段集合中，定义偏移量最小的片段的最左端碱基字符为图的起始节点，偏移量最大的片段的最右端碱基字符为图的终止节点，从起始节点开始每间隔若干个碱基字符定义一个图的节点。

2）边：在同一条片段的两个相邻节点之间定义一条单向边，边的方向与起始节点到终止节点的方向相同，该边的权重为两个节点间所有碱基字符所对应的质量值之和。在两条片段重叠区域的相对应节点间定义一条权重为 0 双向边。

Phrap 采用 Tarjan[110]算法来寻找权值最大的路径，虽然该算法的时间复杂度和空间复杂度均为问题规模的线性函数，但是由于加权有向图 $G$ 对内存的需求较大，尤其在处理大规模数据时该步的运行时间以及对内存的需求量都是非常庞大的。因此，如何提高 Phrap 的运行时间以及减少程序对内存空间的需求就显得尤为重要。

### 3.1.3.2 基于 Euler 路径方法的拼接算法

此类拼接算法是 2001 年由 California 大学计算机系的 Pavel A. Pevzner[111-112]等提出的，他们认为传统的“重叠—排列—生成共有序列”的思维模式导致了将拼接问题抽象为 NP-完全的 Hamilton 路径问题，他们从另一种从来没有在实际测序工程中应用但是直接导致了基因芯片工业的杂交测序方法 SBH（Sequencing by Hybridization）中得到启示，提出了容易计算的在 *de Bruujin* 图中寻找 Euler 超路径的拼接方法，并用研制的软件得到了差错和重复匹配错误均优于 Phrap 的结果[39]。

以下介绍此类拼接算法的基本思想。

1. 对每个片段 $f_i$，设它的长度为 $n_i$，则将 $f_i$ 转化为一个含有 $n_i+l-1$ 个相互重叠 $l$-子串的集合 $F(f_i)$。即，从 $f_i$ 的第一个字母起，依次取以此字母开头的 $l$-子串，这样相邻两次取到的 $l$-子串至少有长度为 $l-1$ 的重叠区域。

2. 依次将每个 $F(f_i)$合并进 *de Bruujin* 图 G，其中 G 的节点为每个 $l$-子串，边集为每两个有长为 $l-1$ 的重叠区域的 $l$-子串有序对。于是，每个 $F(f_i)$可以看作是依次通过某个 $l$-子串的路径，并且不同的 $F(f_i)$、$F(f_j)$可通过他们所含的公共节点和节点间的边相联系。

这样，拼接问题就转化为 Euler 超路径问题，即给定一个图 $G$ 及此图中一些路径的集合 ，给出图 $G$ 的一个 Euler 路径 $P$（称为 Euler 超路径），使得它将 $F$ 中的每一条路径都作为自己的子路径。

为了求得这样的 Euler 路径，可以对系统<$G, F$>进行一系列“等价”的 $T$ 变换：

$$<G, F> \xrightarrow{T} <G_1, F_1> \xrightarrow{T} <G_2, F_2> \xrightarrow{T} \cdots \xrightarrow{T} <G_k, F_k>$$

其中对 $T$ 变换的最基本要求就是变换前后两个系统的 Euler 超路径相同。最终得到的变换结果<$G_k, F_k$>就提供了所需要的拼接结果。

该类拼接算法以一种新颖的观点解决片段拼接问题，在剔除重复子序列及矫正片段数据中错误方面有其独到之处，这一类拼接算法最著名的是 EULER。EULER[113-115]将拼接问题归结为寻找 Eulerian 超路径问题，主要特点有：1）不尝试各个片段之间的重叠区域，没有查找重叠区域的步骤；2）将片段切割成规整的小片段。

EULER 算法流程：

**Step1**：

1）将所有的片段切割成小片段（即 $k$-子串），以排除片段中的错误，获得准确度较高的片段。思路是：如果知道该片段的原始序列 $G$，那么可直接使用片段和 $G$ 进行比较。但是我们并不知道 $G$，那么可以使用 $G$ 的 $k$-子串集合来代替它。但是我们也并不知道 $G_k$，所以可以使用 $G_k$ 的近似来代替。我们设定一个阈值 $m$，将所有出现在至少 $m$（阈值）个片段中的 $k$-子串形成的集合称为 $G_k$ 的近似。

2）将每个片段和 $G_k$ 的近似进行比对，寻求片段的最小改变，使得片段的所有 $k$-子串在 $G_k$ 的近似中。

**Step2**：构造 de Bruujin 图。

结点：每个 $k$-子串是结点 。

边： 结点 $v$ 和 $u$ 有一条边，当前仅当 $v$ 的尾部和 $u$ 的头部相同。

图的构造结果是：每个片段表示成一条路径，每个重复表示成一个多入口、多出口的单一链，但是不知道出入口之间的对应关系。

**Step3**：通过图的等价变换.，在 de Bruujin 图中寻找一条 Eulerian 路径，使得能够覆盖所有的路径。

解决 DNA 序列拼接问题的关键点在于如何提高拼接速度和准确度。事实上，影响拼接速度的主要因素是片段（Read）数目巨大、片段比对操作非常耗时，影响拼接准确度的主要因素是测序错误和重复序列（Repeat）的存在。目前的两类拼接算法都使用了图这一复杂的数据结构记录拼接的中间信息，无论是存储开销还是时间开销都是非常大的。另外，两者还需要保留大量的临时和过渡数据，特别是第二类拼接算法，在将片段分割成更小的串后以及进行图变换时，存储的要求将数倍于第一类拼接算法。在时间开销上，虽然 Euler 路径的计算复杂性低，但其图节点数

百倍于 Hamilton 图，抵消了一部分这方面的优势。

在基于 Hamilton 路径的拼接算法中，人们把每个片段看成一个图的顶点，两个片段有重叠就连一条边，所有片段根据重叠信息排列，算法的目标是寻求一条合理的 Hamilton 路径。但是，这种方法存在的最大缺陷是无法处理 DNA 序列中存在的重复序列问题，也就是说对于两个相似的片段(overlap)算法无法判断它们是来自基因组中的同一区域还是来自位于基因组中不同的重复区域。这样就可能使拼接后的序列缺失某一段重复区域，甚至可能将不同染色体的两个片段错误地连接在一起。在基于 Euler 路径方法的拼接算法中，人们把每个片段看成一个路径，许多相同的路径组成了 de Bruujin 图的边，算法的目标是寻求一条合理的 Euler 路径。但是，由于 de Bruujin 图的存储占用了较大的内存空间，软件的应用规模受到了较大的限制。

为了克服上述算法的缺点，提高序列拼接问题的处理速度，我们提出了该问题的高效并行算法 Ant-Splicing。该算法采用基于 Hamilton 路径的拼接方法。在算法中，为了得到片段对之间前、后缀重叠部分的长度，我们提出了后缀索引的概念来取代后缀树，使得算法易于并行，且节约存储，在寻找权值最大的 Hamilton 路径时，我们利用蚁群算法解决 TSP 问题的优化功能，并将蚁群算法并行化，取得了满意的结果。以下几节将详细介绍算法 Ant-Splicing 中提出的概念、方法及基本思想。

## 3.2 算法的思路及框架

对于给定的序列片段集合 $F$，我们的算法步骤如下：

**Algorithm Ant_Splicing($F$)**

**输入：** 片段序列的集合 $F$；

**输出：** 拼接后的序列 $S$；

**Begin**

1. 对每一序列片段的后缀按照字典序进行排序，建立后缀索引，每一个片段的所有后缀使用一个处理器；
2. 对所有的序列 $i$，使用序列 $j$ 在序列的 $i$ 的后缀索引中通过二分查找得到与其相对精确匹配的后缀在后缀索引中的位置；
3. 在该位置的前后区域分别各取 $T$ 条后缀索引，随后通过比对，查找到匹配度最高且最长的后缀索引，长度记为 $l_{ij}$ ；
4. 以各片段为顶点，各片段间前后缀最大重叠长度 $l_{ij}$ 为对应边上的权，建成有向图；
5. 使用蚁群算法找出最大的哈密尔顿回路；

6. 根据该哈密尔顿路径得到拼接后的序列 $S$;

**End**

接下来我们介绍一下该算法中需要解决的四个问题。

一、对序列建立后缀索引:设对序列 $X=(x_1, x_2, \cdots, x_n)$ 使用 $P$ 个 $PE$,设 $P=4^k$,即 $k=log_4 p$;取各个后缀序列 $s_1, s_2, \cdots s_n$ 的前 $k$ 个字符编码($k\in[0, p-1]$),按编码分配至各处理机,在处理机内按字典序排序形成后缀索引。

二、对序列 $i$,设分配 $P$ 个 $PE$,将后缀索引复制到 $P$ 个 $PE$ 中,将其他的字串 $x_j$ 分配至 $P$ 个处理机中。对每一个序列 $x_i$,在 $x_i$ 的后缀索引中作二分查找,找到一个序列后缀 $s_i$ 使得 $s_i$ 为 $x_i$ 的前缀,且$|s_i|$最大,随后将处在 $s_i$ 前后为 $T$ 的区域内的所有序列后缀,通过比对,找到得分最高的序列,记为 $l_{ij}$,若无,则 $l_{ij}=0$。

三、在片段的前、后缀的比较时,精确匹配往往会缩短匹配长度,我们要在匹配精度与长度之间取得平衡。因此在 $T$ 区域对前后缀的比较需采用比对方法,而传统的比对方法需要大量的计算时间和存储时间。因此,我们需要一个快速的、近似估计序列比对的方法。为此,我们提出了一种四次扫描的方法,在线性时间里得出序列的近似比对。

四、利用蚁群算法求最长的哈密尔顿路径时,需要对蚁群算法进行并行化。在并行化过程中需要将蚂蚁群体分成若干个独立的子群体,分配到多个处理机上去独立地进行。在这里,就要解决如何控制、管理子群体之间的信息交换,确定它们之间的通讯方式。还要解决如何确定各个蚂蚁子群体之间的信息交换周期。为此,我们提出了一个并行策略,使并行蚁群算法保持很好的收敛性,同时具有较高加速比及计算效率。

## 3.3 生成后缀索引的并行算法

在序列拼接中,为了查找某一个序列片段的前缀是否是另一个序列片段的后缀,我们需要对各个序列的后缀进行分析,分析的常用工具是后缀树。后缀树是传统字符串领域的一种非常重要的数据结构,由于其高效的索引能力和易于构造的特点,它逐渐成为生物信息领域的主要研究工具。

### 3.3.1 后缀树概念

后缀树是用树的存储结构来存储字符串，从而能够实现快速的字符串检索，检索的时间复杂度和被检索的字符串长度之间具有特定的关系。

对于一个由 $m$ 个字符构成的字符串 $S$，其后缀树 $T$ 是一棵有 $m$ 个叶子结点的树。树中的每个中间结点至少有两个儿子结点，每条边标记一个非空字符串，并且是 $S$ 的子串。任意两条开始于同一个结点的边，它们标记的是不同字符串。最重要的特性是从根结点沿着边一直到叶结点，可以通过边上标记的字符串，得到叶结点对应的后缀字符串，从而通过每个叶结点可以得到 $S$ 的所有后缀字符串。$T$ 具有如下性质：

1. 每条边标记了 $S$ 的子序列。
2. 每个中间结点至少有两个子结点。
3. 叶结点的个数为 $m$。
4. 每个叶结点对应一条 $S$ 的后缀字符串。

**例 3.1** 串“ababbaaba #”的后缀树：

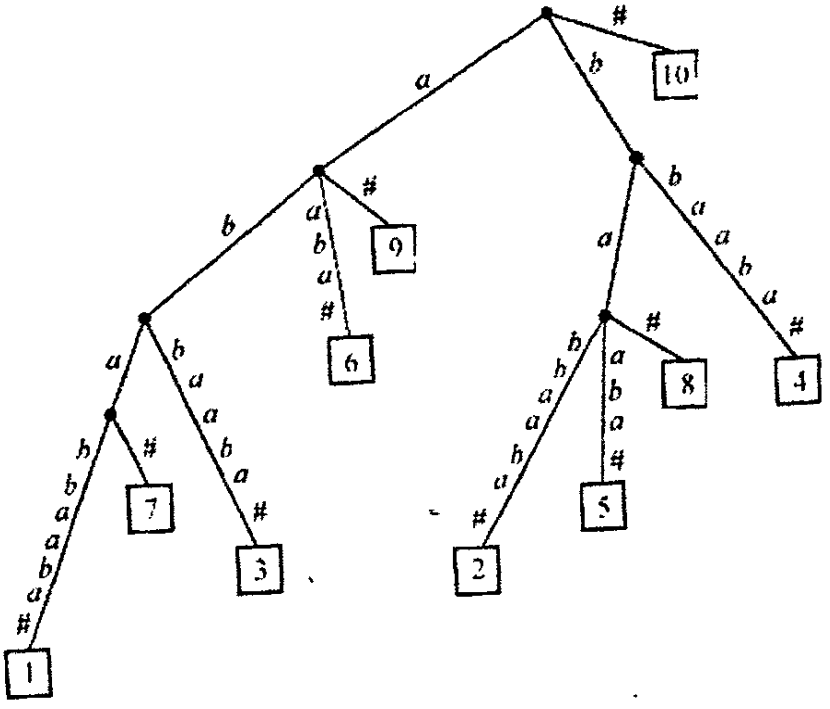



图 3.7“ababbaaba #”的后缀树

### 3.3.2 后缀树算法及其并行性问题

构造后缀树所需要的时间和空间都是与字符串长度有一定关系的,这也是后缀树得到广泛应用的原因之一。经典的后缀树构造算法有两种,分别是 MCC 算法[116]和 UKK 算法[117]。

虽然后缀树可以快速的进行字符串搜索,但是,面对最大长度可达 6 Gbps(basepairs)的生物序列[118-119],传统的后缀树构造算法受到了极大的限制。其原因在于:一方面,许多经典的后缀树构造算法都是基于主存的算法,它们对主存空间的要求是数十倍于字符串的长度。很显然,面对这种超长的生物序列,这些基于主存的算法必然要受到主存大小的限制。另一方面,因为这些生物序列的长度非常长,因此构造其后缀树所消耗的时间也很长。例如,使用文献[120]中的算法,构造长度为 20.5 Mbps 的 DNA 序列的后缀树所需要的时间是 34 h。由此可见,当字符串的长度很长时,构造其后缀树的时间和空间开销是令人无法忍受的。

并行技术是解决这些问题的很好途径。目前,对并行后缀树的研究集中于算法并行方面,而不是结构并行方面。Apostolico[24]等人,Sahinalp 和 Vishkin[121]及 Hariharen[122]分别提出了几种并行算法,但这些算法都不适用于生物信息领域。这是因为:①现有的几种并行后缀树构造算法都是基于 PRAM 模型[123]的,PRAM(parallel random accessmachine)模型是一种研究并行技术的理论模型,它假设处理机的个数是不受限制的,而且假设有无限大的公共存储器,并假设这些存储器可共读或共写,这对目前大部分多处理机系统是不现实的。因此,实际应用这些算法时必然涉及到处理机及存储的重新分配问题,而且算法的性能也会因为处理机个数的减少而下降;②虽然这些算法缩短了后缀树的构造时间,但是它们对空间的需求并没有降低,因而没有解决主存限制的问题,因此这些算法仍然不适用于生物信息领域。

### 3.3.3 后缀索引及其操作

为了克服上述困难,我们提出了后缀索引的概念。

**定义 3.1** 设对字母表$\Sigma=\{A, C, G, T\}$上的序列 $X=(x_1, x_2, \cdots, x_n)$,其后缀序列依次为 $s_1$, $s_2$, $\cdots s_n$,定义字母表的一种大小顺序,如 A<C<G<T,在此顺序下对 $s_1$, $s_2$, $\cdots s_n$ 进行排序后得到 $s_1'$,$s_2'$,...,$s_n'$,线性表 I= ($s_1'$,$s_2'$,...,$s_n'$)为 $X$ 的后缀索引。

**例 3.2** 设有序列 $X$= "atattaata",它的后缀索引 I 为:

| | |
|---|---|
| 1 | a |
| 2 | aata |
| 3 | ata |
| 4 | atattaata |
| 5 | attaata |
| 6 | ta |
| 7 | taata |
| 8 | tattaata |
| 9 | ttaata |

设具有 $n$ 个序列片段，$p$ 个处理机（$n \geq p$），我们将 $n$ 个片段分配至 $p$ 个处理机中，每个 $p$ 个处理机处理$\lceil n/p \rceil$个片段，建立它们的后缀索引，并存储在该处理机中。然后，在计算前后缀重叠长度阶段，可将各片段 $X_j$ 依次广播至其它处理机之中，在其它片段的后缀索引中作二分查找，找到与 $X_j$ 最精确匹配的后缀 $S_b$，为了寻找精度与长度的最佳组合，我们定义一个阈值 $T$，在 $S_b$ 的位置的前、后邻域取 $T$ 个片段与 $X_j$ 的某一前缀作比对。我们取其中比对得分最高且最长的后缀作为重叠部分。综上所述，设共有 $n$ 个片段，使用 $p$ 个处理机，$m=\lceil n/p \rceil$，算法的 Ant_Splicing($F$)的(1)、(2)、(3)步骤的描述如下：

**For** $i$=1 to $p$ **pardo**
  **For** $j$=1 to $m$ **do**
    对 $X_{(i-1)*m+j}$ 建立后缀索引；
  **end for** $j$
 **For** $j$=1 to $m$ **do**
  向其它所有处理机广播 $X_{(i-1)*m+j}$
**end for** $j$
 **For** $j$=1 to $n$ **do**
   If $(j<(i-1)m+1)$ and $j>im$ then
     接收其它处理机广播来的 $X_j$
    **end if**
  **end for** $j$
 **For** $j$=1 to $m$ **do**
   **For** $k$=1 to $n$ **do**
     **If** $k \neq (i-1)m+j$ **then**

在 $X_{(i-1)*m+j}$ 的后缀序列中对 $X_k$ 作二分查找，设 $X_{(i-1)*m+j}$ 的后缀序列中的后缀依次为 $s_1$，$s_2$，…$s_l$，对 $X_k$ 查找的最精确匹配为 $S_b$。

max=0；$max_l$=0；

**For** $r$=$b$-$T$ to $b$+$T$ **do**

取 $X_k$ 的长度为$|S_r|$的前缀，记为 $X_k$（$r$）；

求 $S_r$ 与 $X_k$（$r$）的比对得分 Score($S_r$, $X_k$（$r$）)；

**If** Score($S_r$, $X_k$（$r$）) > max **then**

max= Score($S_r$, $X_k$（$r$）)；

$max_l$=$r$；

**end if**

**end for** $r$

$l_{(i-1)*m+j,\ k}$=max；

**end for** $k$

**end for** $j$

### 3.3.4 双序列比对的近似估计算法

在上述算法中，要求计算 $S_r$ 与 $X_k$（$r$）的比对得分，这需要进行双序列的比对。如果使用动态规划的方法，设有两个序列的长度分别为 $m$、$n$，则其计算代价是 $O(mn)$。

为了减少计算代价，可以用近似的方法估计两个序列的相似程度。因此，我们提出一个计算两序列 $X$ 和 $Y$ 比对的得分的近似的算法 SE（Score-Estimate）。由于在多序列比对的 SP（逐对加和）模型中，整个多序列比对的得分等于所有双序列比对得分的和，我们可以 SE 算法为基础近似计算多序列比对的得分。

SE 算法的计算分为 4 步，分别记为 Left-Upper、Right-Upper、Left-Lower 和 Right-Lower。其中每一步代表着对 $X$ 和 $Y$ 的一次扫描。例如，Left-Upper 是从序列 $X$ 的第一个字符 $X_0$ 出发，在 $Y$ 中从左向右依次寻找与 $X_0$ 匹配的第一个字符 $Y_j$。然后从 $Y_{j+1}$ 出发，在 $X_0$ 后面从左向右寻找第一个与 $Y_{j+1}$ 匹配的字符 $X_i$，再从其后一个字符 $X_{i+1}$ 出发，寻找 $Y_{j+1}$ 后面第一个与 $X_{i+1}$ 匹配的字符。此过程重复进行直到找遍整个序列 $X$ 或序列 $Y$。每找到一个匹配字符，计数器 count1 加 1。每次在 $Y$(或 $X$)的尚未扫描部分中从左向右寻找与 $X_{i+1}$（或 $Y_{j+1}$）匹配的字符时，如果找不到这样的匹配的字符，算法则改为在 $Y$(或 $X$)的尚未扫描部分中寻找与下一字符 $X_{i+2}$（或 $Y_{j+2}$）匹配的字符。算法 Left-Upper 描述如下：

```
Algorithm Left-Upper(X, Y, n, m, count1)
Begin
    i=0; j=-1; count1=0;
    While (i<n) and (j<m) do
        k:=j+1; found=false;
        Repeat
            While X_i<> Y_k do k=k+1;
                If k=m then
                    j=j+1; k=j+1
                else
                    count1=count1+1;   j=k; found=true
                end if
        Until found;
        k:=i+1; found=false;
        Repeat
            While Y_j<> X_k do k=k+1;
                If k=n then
                    i=i+1; k=i+1
                else
                    count1=count1 +1; i=k; found=true
                end if
            Until found;
        End while
End.
```

其他 3 步的算法 Right-Upper、Left-Lower 和 Right-Lower 的描述与 Left-Upper 类似，仅在开始点和扫描方向不同。SE 算法取这 4 次扫描所得到的 *count*1 至 *count*4 中的最大值作为序列 *X* 和 *Y* 比对得分的近似值。SE 算法描述如下：

```
Algorithm SE(X,Y)
Begin
    Left-Upper(X, Y, n, m, count1);
    Left-Lower(X, Y, n, m, count2);
    Right-Upper(X, Y, n, m, count3);
    Right-Lower(X, Y, n, m, count4);
    Return [max(count1, count2, count3, count4)]
End
```

最后，返回值就代表序列 $X$ 和 $Y$ 之间相似程度的近似值。SE 算法可以在 $O(m+n)$ 时间内近似计算出两个序列之间的相似性得分。

**例 3.3** 设序列 $X$=“ACTGCTA”，$Y$=“TCGATACT”，下图显示了 SE 算法的 4 个步骤。

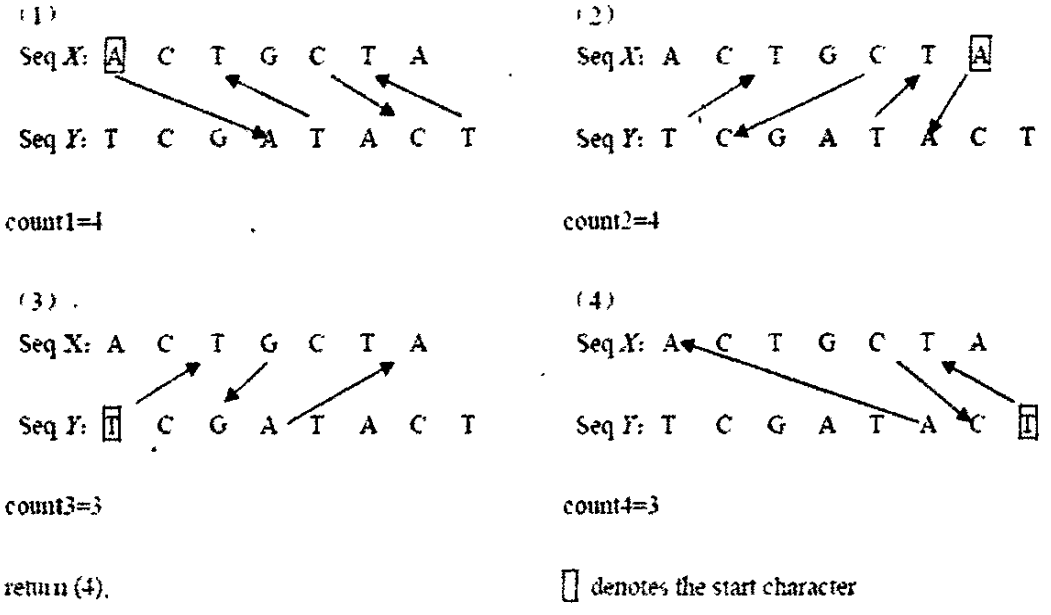
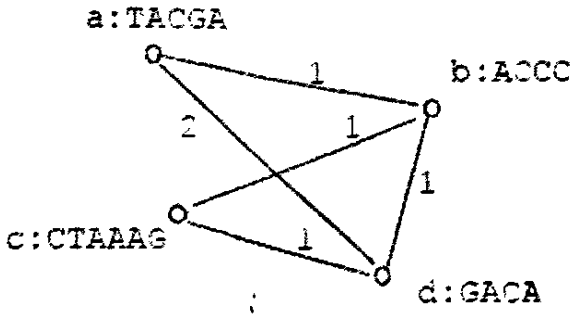



图 3.8 SE 算法示意图

### 3.3.5 建立片段集合的覆盖图

序列片段集合 $F$ 的覆盖图 OM（$F$）是一个有向图，图中的各个顶点代表 $F$ 中的一个字符串。如果 $f \in F$、$g \in F$，并且 $f$ 的 $t$ 个字符的后缀与 $g$ 的 $t$ 个字符的前缀相同，则图中存在一条权值为 $t$ 的有向边。

**例 3.4** 假设序列片段集合 $F$={a:TACGA, b:ACCC, c:CTAAAG, d:GACA}，其覆盖图如 3.9 所示：




(a) F 的覆盖图

```
T A C G A - - - - - - - - - - - -
- - - - A C C C - - - - - - - - -
- - - - - - - C T A A A G - - - -
- - - - - - - - - - - - G A C A
```

(b) 路径 $P_1$=a-b-c-d 对应的拼接结果

(长度 $l$=16)

```
T A C G A - - - - - - - - - - - -
- - - G A C A - - - - - - - - - -
- - - - - - A C C C - - - - - - -
- - - - - - - - - - C T A A A G
```

(c) 路径 $P_2$=a-d-b-c 对应的拼接结果

(长度 $l$=15)

图 3.9 覆盖多图示意

在覆盖图中的一个哈密尔顿路径代表了一个拼接的结果。在图3.9(b)及图3.9(c)中表示了两个不同路径：$P_1$=a-b-c-d及$P_2$=a-d-b-c所对应的不同的拼接结果。对于覆盖图中的一条路径$P_1$ 可以按下述方法构造序列的拼接：如果路径中的某一条边 $e=(f, g)$ 的权值为$t$，则$f$ 和$g$可以被拼接起来，即$f$末端的$t$个字符与$g$前端的$t$个字符相重叠。按照上述方法顺序处理路径上的每一条边，就可得到一个拼接方案。

设 $P$ 是覆盖图中的一条哈密尔顿路径，该路径上$|F|$-1 条边，将根据 $P$ 所得到的拼接结果记为 $S(P)$，则 $F$ 的总长度$||F||=\sum_{i=1}^{i=n}|X_i|$、哈密尔顿路径权值之和 $W(P)=\sum_{e\in P}t(e)$ 以及拼接结果的长度关系如下：

$$||F||=W(P)+|S(P)|$$

即
$$|S(P)|=||F||-W(P)。$$

由于$||F||$为常数，则根据上式可以看出，在$W(P)$取最大时，$S(P)$的值最小，即求最短的公共超串等价于求覆盖图中的最大的哈密尔顿路径。

在求得覆盖图后，如何找出其中最长的哈密尔顿路径呢？我们使用蚁群算法。这是因为蚁群算法对 TSP 问题有很强的求解能力，而 TSP 问题本质上是求最短哈密尔顿回路，与求最长哈密尔顿路径问题十分类似。

## 3.4 用并行蚁群算法求最长哈密尔顿路径

### 3.4.1 蚁群算法的基本原理

蚁群算法(Ant colony algorithm，简称 ACA)是一种新型的模拟进化算法，它是由意大利学者 M.Dorigo 等人受到自然界中真实蚁群集体行为的研究成果的启发而首先提出来的[124-126]。他们充分利用蚁群搜索食物的过程与旅行商(TSP)问题之间的相似性，通过人工蚂蚁搜索食物的过程中个体之间的信息交流[127]与相互协作找到从蚁巢到信物源的最短路径的原理解决了 TSP 问题[128]，取得了较好的求解效果。

蚁群算法可以有效求解一些复杂优化问题，包括调度问题[129]、指派问题[130-131]、图着色问题[132]、聚类问题[133]、网络路由问题[134]，双序列比对问题[135]等，并在大规模集成电路设计，电信路由控制方面表现出较好的性能。

在蚂蚁集体觅食时，当一工蚁发现食物后，释放出一种挥发性激素作为示踪标记，然后回巢告知同伴，它在巢内来回迅速跑动并用前腿和触角轻拍同伴，当同伴得到信息后，成百上千的工蚁涌向食物直至食物被取食完为止。

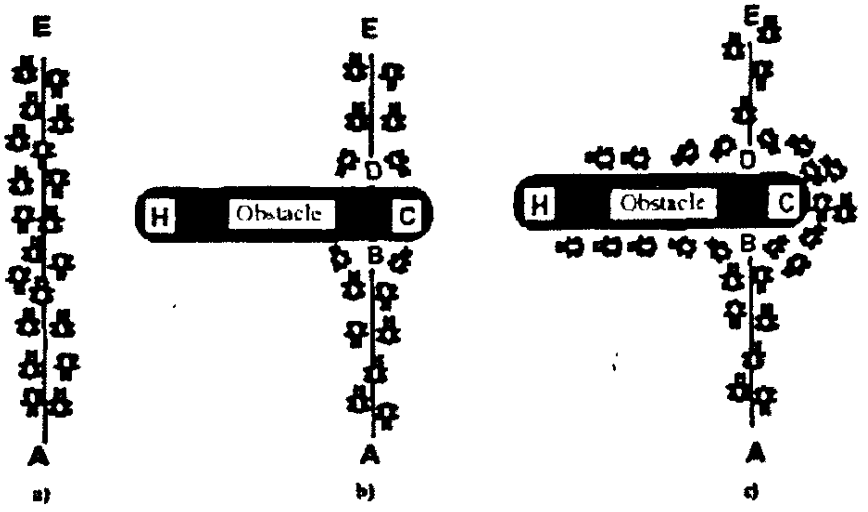



图 3.10 蚂蚁寻找最短路径的基本原理

图 3.10 说明了蚂蚁如何发现最短路径的基本原理。由于中间存在一个障碍物，从蚂蚁巢穴 A 到食物源 E 有两条路径。由于右边的路径较短，蚂蚁在走这条路径时，从巢穴到食物源再返回巢穴所经历的时间就短，从而相同时间内，蚂蚁在这条路径上所留下的信息素就较多。而后面的蚂蚁要根据前面走过的蚂蚁所留下的信息素的多少选择其要走的路径，路径上的信息素越多，蚂蚁选择这条路径的概率就越大。这样，距离较短的路径上信息量很快得到了强化，其优势也很快被蚁群所发现。所以，如图 3.10 所示，虽然刚开始时蚂蚁是按同等概率选择路径，但经过一段时间后，蚂蚁能够很快的重新找到最优路径。

蚂蚁群体的集体觅食行为实际上构成了一种学习信息的正反馈机制，蚂蚁之间通过这种信息交流与相互协作寻求从巢穴到食物源之间的最短路径。

蚁群算法是受上述真实的蚂蚁在觅食过程中的群体合作行为而提出的，它的很多观点都来源于真实蚁群，因此它们都包括下列几项[136]：

(1)存在一个群体中个体相互交流通信的机制，这里通常表现为信息量迹；

真实蚂蚁和人工蚂蚁都存在一种机制改变它当前所处的环境：真实蚂蚁在经过的路径上留下化学刺激物—信息量(pheromone)，人工蚂蚁在它们经过的路径上改变了路径上存储的数字信息,这个信息记录了蚂蚁当前的和历史的解的性能状态,而且能够被经过的其他人工蚂蚁读写。类似的,我们称这种数字信息为人工信息量。在蚁群优化算法中蚂蚁进行交流协作的方式就是当前路径上的信息量。这种交流方式在收集可利用的知识上占据着重要的位置,其重要的作用在于它改变了当前蚂蚁所经过的路径周围的环境,同时也像一个函数似的改变了整个蚁群所存储的历史信息。通常,在蚂蚁优化算法中有一个挥发机制,它像真实的信息量挥发一样随着时间的推移

改变路径上的人工信息量。挥发现象使得蚁群可以逐渐的忘却历史遗留信息，这样就可以使选路不局限于过去蚂蚁的路径选择经验。

(2)群体中每个个体所记录的当前遍历序列;

人工蚂蚁和真实蚂蚁都要完成一个相同的任务：寻找一个从源节点(巢穴)到目的节点(食物源)的最短路径,它们都不具有跳跃性,都只能在相邻节点之间一步一步移动直至选择完所有城市得到一个遍历序列,为了能在多次寻路过程中找到最短路径则应该记录当前移动序列。

(3)利用当前信息进行路径选择的随机选择策略：

人工蚁群算法中人工蚂蚁从一个节点移动到下一个节点的求解方法是利用概率选择策略实现的,概率选择策略只利用当前的信息去预测未来的情况,而不能利用未来的信息,因此,该选择策略利用的都是当前信息。

在从真实蚂蚁的行为中获得启发构造蚁群算法的过程中人工蚂蚁还具备了真实蚂蚁不具有的一些特性：

（1） 人工蚂蚁存在于一个离散的空间中,它们的移动是一个状态到另一个状态的转换;

（2）人工蚂蚁具有一个记忆了它本身过去行为的内在状态;

（3）人工蚂蚁更新信息量的时机是随不同问题而变化的,不反映真实蚁群的行为。如：有的问题中人工蚂蚁在产生一个解后改变信息量,有的问题中则蚂蚁每作出一步选择就更改信息量,但无论哪种方法,信息量的更新并不是随时可以进行的;

（4）为了改善系统的性能,人工蚁群算法中可以增加一些性能,如：预测未来、局部优化、回退等,这些行为在真实蚂蚁中是不存在的。在很多应用中人工蚂蚁可以在局部优化过程中相互交换信息,还有一些蚁群算法实现了简单预测。

### 3.4.2 蚁群算法的基本框架

人工蚁群可以模仿真实蚁群进行信息素的交流、信息素的更新、路径的选择，所以，人工蚁群可以模拟真实蚁群进行现实问题的求解。

蚁群的集体行为表现出了一种信息正反馈机制：某一路径上走过的蚂蚁越多，则后来的蚂蚁选择该路径的概率就越大，蚂蚁个体之间就是通过这种信息的交流搜索食物，并最终沿着最短路径行进。蚁群算法中的人工蚂蚁寻找最优路径的方法就

是根据真实蚂蚁寻找最优路径的方法提出的，即让人工蚂蚁根据路径上的相当于信息素的数字信息量的强度选择路径，并在所经过的路径上留下相当于信息素的数字信息量。随着时间的推移，最优路径上的数字信息量将积累的越来越大，从而被选择的概率也越来越大，最终所有人工蚂蚁将趋向于选择该路径。这种模拟蚁群搜索食物的过程与著名的旅行商问题非常相似，因而最初人工蚁群算法被提出应用于求解旅行商问题[137]。

下面以 TSP 为例说明蚁群算法的基本实现模型。

TSP问题是一个著名的NP-hard问题。给定$n$个城市的集合及城市之间的环游花费，找到一条经过每个城市一次且回到起点的最小花费的环游。若将每个顶点看成是图上的节点，花费为连接两点的边上的权，则TSP问题就是在一个具有$n$个节点的完全图上找一条花费最小的哈密尔顿回路。

设有$n$个城市集$C=(1,2,\cdots n)$，任意两个城市$i,j$之间的距离为$d_{ij}$，求一条经过每个城市仅一次的路径$\pi=(\pi(1),\pi(2),\cdots\pi(n))$，目标是使得$\sum_{i=1}^{n-1}d_{\pi(i)\pi(i+1)}+d_{\pi(n)\pi(1)}$最小。

$b_i(t)$表示$t$时刻位于城市$i$的蚂蚁的个数，$m=\sum_{i=1}^{n}b_i(t)$为蚂蚁的总数。$\tau_{ij}(t)$表示$t$时刻边$ij$上的信息素量，$\tau_{ij}(0)=\tau_0(\tau_0$为常数)。

随着时间的推移，新的信息素加进来，旧的信息素挥发掉，$1-\rho$表示信息素的挥发快慢。当所有蚂蚁完成一次旅行后，各条边上的信息素按下式调整：

$$\tau_{ij}(t+n)=\rho\tau_{ij}(t)+\Delta\tau_{ij} \tag{3.1}$$

其中：

$$\Delta\tau_{ij}=\sum_{k=1}^{n}\Delta\tau_{ij}^{k} \tag{3.2}$$

$\Delta\tau_{ij}$表示本次周游中路径$ij$上的信息素增量，初始时刻，$\Delta\tau_{ij}=0$。$\Delta\tau_{ij}^{k}$表示第$k$只蚂蚁在周游过程中释放在边$ij$上的信息素。

$$\Delta\tau_{ij}^{k}=\begin{cases}\frac{Q}{L_k} & \text{若第}k\text{只蚂蚁在本次周游中经过边}ij\\ 0 & else\end{cases} \tag{3.3}$$

$Q$为常数，$L_k$表示本次周游第$k$只蚂蚁所形成的回路的长度。蚂蚁在周游时，向哪个城市转移由转移概率$p_{ij}^{k}$决定。

$$p_{ij}^{k}=\begin{cases}\dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{s\in allowed_k}\tau_{is}^{\alpha}\eta_{is}^{\beta}} & j\in allowed_k\\ 0 & else\end{cases} \tag{3.4}$$

其中 $allowed_k=\{0,1,\cdots n-1\}-tabu_k$ 表示蚂蚁 $k$ 当前能选择的城市集合，$tabu_k$ 为禁忌表，它记录蚂蚁 $k$ 已路过的城市，用来说明人工蚂蚁的记忆性。$\eta_{ij}$ 是某种启发信息。在 TSP 问题中，$\eta_{ij}=d(c_i,c_j)^{-1}$。$\alpha,\beta$ 体现了信息素和启发信息对蚂蚁决策的影响。

综合以上所述，用蚁群算法求解 TSP 问题的算法框架如图 3.11 所示：

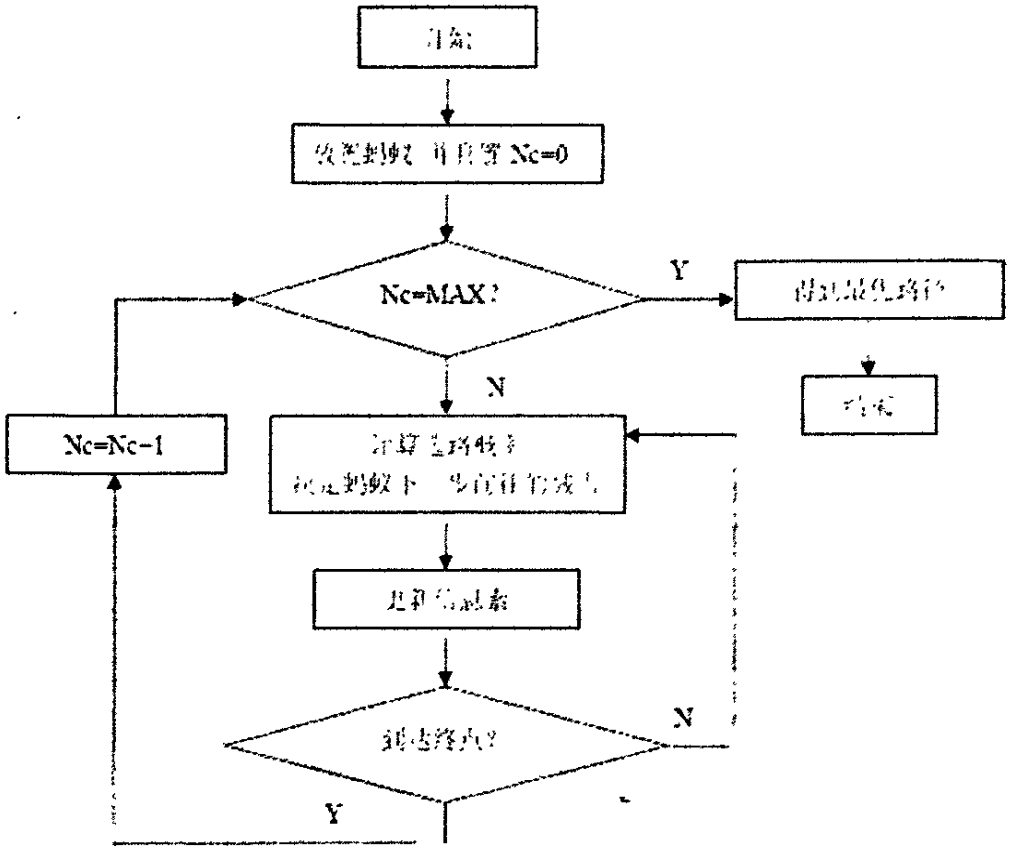



图 3.11 基本蚁群算法求解TSP问题

蚁群算法基本的过程是，$m$ 只蚂蚁同时从某个城市出发，根据转移概率选择下一次旅行的城市，已去过的城市放入 $tabu_k$ 中，一次循环完成后，根据信息素更新公式更新每条边上的信息素，反复重复上述过程，直到满足迭代终止条件。

可见，蚁群算法[138]是一种随机搜索算法，类似于其他的模拟进化算法，通过

候选解组成的群体的进化过程来寻求最优解。进化过程包含两个基本阶段：适应阶段和协作阶段。在适应阶段，各个候选解根据积累的信息不断调整自身结构；在协作阶段，候选解之间通过信息交流，期望产生性能更好的解。

### 3.4.3 应用蚁群算法求解最长哈密尔顿路径问题

蚁群算法在解决 TSP 问题中显示出强有力的优化功能，求权值最大的哈密尔顿路径问题与 TSP 问题是十分相似的。因此，我们完全可以利用蚁群算法来进行求解。下面是我们的求解最长哈密尔顿路径的蚁群算法的具体描述。

设覆盖图 $G$ 中有 $n$ 个顶点，代表 $n$ 个待拼接的片段，顶点 $v_i$ 与 $v_j$ 之间的有向边$(i,j)$上的权为 $t_{ij}$，表示相应的片段之间前后缀的重叠部分长度。为了寻找 G 中的权最大的哈密尔顿路径，我们使用 $m$ 个人工蚂蚁，随机地分散在各个顶点上，边$(i,j)$上的信息素 $\tau_{ij}$ 的初值可以取一个常数：

$$\tau_0 = \frac{1}{n(n-1)} \sum_{j=i+1}^{n} \sum_{i=1}^{n-1} t_{ij}$$

边$(i,j)$上的启发式信息 $\eta_{ij}$ 可以取 $\eta_{ij} = t_{ij}$。

所有蚂蚁在覆盖图上移动以形成各自的哈密尔顿路径。在 $t$ 时刻，位于 $v_i$ 的第 $k$ 只蚂蚁选择 $v_j$ 的概率 $p_{ij}^{k}(t)$ 由下式决定：

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}}{\sum_{r \in allowed_k} \tau_{ir}^{\alpha}(t)\eta_{ir}^{\beta}} & j \in allowed_k \\ 0 & otherwise \end{cases}$$

其中 $allowed_k$ 表示蚂蚁当前能选择的顶点的集合，$\alpha, \beta$ 为信息素和启发信息对蚂蚁选择的影响因子。

在所有蚂蚁进行了一轮周游，形成了一个哈密尔顿路径以后，各边上的信息素按下式调整：

$$\tau_{ij}(t+1) = \rho \tau_{ij}(t) + \Delta \tau_{ij}$$

$\Delta \tau_{ij}$ 表示本次周游中路径$(i,j)$上的信息素增量，由下式表示：

$$\Delta \tau_{ij} = \sum_{k=1}^{m} \Delta \tau_{ij}^{k}$$

$\Delta \tau_{ij}^{k}$ 表示第 k 只蚂蚁在本次周游过程中释放在边$(i,j)$上的信息素：

$$\Delta\tau_{ij}^{k}=\begin{cases}QL_k & \text{若第}k\text{只蚂蚁在本次周游中经过边}(i,j)\\ 0 & otherwie\end{cases}$$

其中，$Q$ 为常数，$L_k$ 表示本次周游第 k 只蚂蚁所形成的哈密尔顿路径的长度。由上面的公式可以看出，对于边$(i,j)$，经过的蚂蚁越多，这些蚂蚁所形成的路径越长，该边上的信息素增加越大。重复上述的周游过程，每次迭代中记下最好的解，直至满足迭代终止条件。在我们的算法中，迭代终止条件为达到一个固定的迭代次数或在若干代中的最优解，没有变化为止。

### 3.4.4 蚁群算法的并行实现

上述求最长哈密尔顿路径的蚁群算法需要并行执行。 我们对蚁群算法的并行化的基本思想如下：

把 $M$ 只蚂蚁分成 $P$ 个子群体，每个处理机上分配一个子群体，然后各处理机上的子群体独立地搜索最优解。为了防止某个处理机上的子群体陷入局部最优，在满足既定的条件（如时间间隔、相隔代数等条件）时，处理机之间进行信息交换。信息交换时每个处理机不是随机地选择某个处理机与其进行信息交流，而是采用自适应的方法来动态地决定与哪个处理机交流信息。这样每个处理机能够根据自身解的情况，从其他处理机中吸收到最有利于自己进化的优良信息素。另外，处理机之间信息交流的周期不是固定的，而是根据解的多样性来自适应地调节。

在上述并行化过程中，处理机间信息交流策略、信息交流的周期是影响算法收敛速度、解的质量、加速比和计算效率的关键因素。

#### 3.4.4.1 选择处理机进行信息交流的策略

在我们的处理机选择信息交流对象的策略中，每个处理机不再是选择地理上的邻接处理机或是随机地选择处理机来进行信息交流，而是根据处理机上得到的解的平均适应度来自适应地选择与之交换的处理机。这样，在信息交换时可以为每个处理机提供指导信息，使它向着最有利于优化的方向进行搜索。我们让每个处理机选择一个与其最优解差异最大的处理机进行信息交流。

为此，我们定义 $dis(i,j)$ 为处理机 $i$ 和处理机 $j$ 之间的距离，其计算公式为：

$$dis(i,j)=\frac{1}{n(n-1)}\sum_{l=k+1}^{n}\sum_{k=1}^{n-1}x(i,j,k,l)$$

这里，$x(i,j,k,l)=\left|\tau_{kl}^{(i)}-\tau_{kl}^{(j)}\right|$，$\tau_{kl}^{(i)}$和$\tau_{kl}^{(j)}$分别为处理机 $i$ 和处理机 $j$ 的信息素矩阵的$(k,l)$元素。由此公式可以看出，处理机间的距离越大则两者的最优解在解空间的位置就相隔较远，它们的相似程度就越低。为了使尽可能不相似的处理机之间进行配对，我们按如下的策略对各个处理机选择配对：对 $i$=1，2，…，$P$，若处理机 $i$ 尚未选择配对，则它的配对处理机 $j$ 由下式确立：

$$j=\arg\max_{\substack{1\le k\le P\\ k\notin tabu}}\{dis(i,k)\}$$

处理机 $i$ 确定了与它进行信息交换的处理机 $j$ 后，便对其信息素矩阵进行更新：

$$\tau(u,v)=(1-\lambda)\cdot\tau(u,v)+\lambda\cdot[\Delta\tau(u,v)+\Delta\tau'(u,v)]$$

其中：

$$\Delta\tau(u,v)=\begin{cases}Q_1L(i) & if\text{处理机i的最优解经过边(u,v)}\\ 0 & otherwise\end{cases}$$

$$\Delta\tau'(u,v)=\begin{cases}Q_2L(j) & if\text{处理机}j\text{的最优解经过边(u,v)}\\ 0 & otherwise\end{cases}$$

公式中的$\lambda\in(0,1)$,它表示处理机 $i$ 上的信息消逝程度，$Q_1$ $Q_2$ 为一正的常数，$L(i)$、$L(j)$分别为处理机 $i$ 和处理机 $j$ 中获得最优解的蚂蚁遍历的路径长度。从上述两式可以看出，解的质量较高的解所经过的路径上的信息量更新的程度就很大，这样做既可以强化适应度高的路径，又可以保证解的多样性。

基于距离选择的信息交换方式，是根据处理机最优解之间的距离为每个处理机选择另一个适应度高、与其距离较远的处理机 $j$ 和它进行信息交换，然后利用处理机 $j$ 的最优解来更新信息素矩阵，既可以使处理机 $i$ 向着问题的最优解的方向进化，又可以保持解的多样性，使得信息素矩阵能够有效地引导搜索继续进行，避免某些路径上的信息量过于集中而发生早熟现象。

### 3.4.4.2 进行信息交流周期的调节

处理机之间的信息交换的目的是使得优质解传播给其他处理机，当某个处理机上的蚁群逐渐进入收敛状态，它可以根据其他处理机上的优质解来摆脱局部最优。通过调节交换周期使算法收敛于全局最优或在全局搜索空间中开辟新的解空间。当解的多样性较差，即各个解很相似时，要减小交换周期，使得处理机之间进行较频繁的信息交换，从而提高解的多样性，增强算法的搜索能力。而在解的多样性已经

很强时，则要增大交换周期，从一方面减少通信开销，另一方面可保持各处理机的进化环境的相对稳定，加速其收敛。

对信息交流的周期的调节，我们采取一种自适应地调节信息交流周期的方法，即交流周期并不是固定的，而是根据解的分布情况而变化，这样有利于在解的多样性和算法的收敛速度两个方面达到平衡。算法根据整体多样性来调整通信周期，使得当整体多样性较差时，通信周期可以适当地减小，以让优质解在处理机之间得以迅速传播，以改善处理机的进化环境。整体的多样性好时，通信周期可以得到增加，从而适当地减少通信开销。这样可以使算法不仅具有较强的全局收敛性，而且有更快的寻优速度、更高的效率。

为了根据解的多样性来自适应的调节交换周期，我们需要对各个处理机的解的多样性进行度量。设某处理机 $i$ 上有 $N_i$ 个蚂蚁，在当前循环中每个蚂蚁的适应度分别为 $f(i,1), f(i,2), \cdots f(i,N_i)$，我们用 $div$ 来表示所有处理机的整体多样性程度：

$$div=\frac{1}{p}\sum_{i=1}^{p}\frac{1}{N_i}\sum_{j=1}^{N_i}\left|f(i,j)-\overline{f}(i)\right|,$$

这里 $\overline{f}(i)$ 为第 $i$ 个处理机上所有蚂蚁的平均适应度，$div$ 的值越大表示整体多样性越好。算法根据整体多样性来调整通信周期 $gap$：

$$gap=k\left[1+e^{k_1\,div}\right]$$

其中，$k$、$k_1$ 为正的常数。从上式可以看出，当 $div$ 的值较小时，整体的多样性差，通信周期则可以适当地减小，使得优质解在处理机之间得以迅速传播，改善处理机的进化环境，且多样性越差，通信周期越小。随着 $div$ 值的增大，整体的多样性越来越好，通信周期也可以得到增加，从而适当的减少了通信开销。本文设 $k$=16，$k_1$=0.5。

### 3.4.4.3 并行蚁群算法框架的描述

我们对上述的自适应并行蚁群算法的框架进行描述，其中 $N$ 是最长哈密尔顿路径问题中顶点的个数，$m$ 是每个处理机上蚂蚁的个数，$NCMAX$ 是总的迭代次数，$p$ 为处理机个数 。

**Algorithm**　Adaptive parallel Ant Colony Algorithm

**Begin**

*m=N/p;*

```
For  mid= 0 to p-1  pardo      /*p 个处理机并行参加计算*/
   初始化信息素矩阵和时间间隔 gap;
    i=0;
 while (i<NCMAX)
   i=i+gap;
  /*每个处理机中的 m 个蚂蚁作为一个子群体独立的运行蚁群算法*/
  For j=0  to gap do
     for k= 1 to m do    (m 个蚂蚁)
         for  l= 1 to n do  (n 个顶点)
            蚂蚁 k 寻找下一个顶点
          Endfor l
     Endfor  k
    计算各个蚂蚁所得解的长度，记下最优者;
    局部更新信息素矩阵;
   Endfor  j
    /*循环间隔完成，处理机之间进行信息交换*/
    计算本处理机解的多样性，把最优解和局部多样性发送到 0 号处理机
    if mid= 0 then
       为各个处理机确定交流对象和计算新的交换周期 gap;
       通知各个处理机进行信息交流
    Endif
    接受 0 号处理机发来的交流对象信息和新的交换周期 gap;
    和交流对象进行信息交换
   Endwhile
 Endfor  mid
End
```

## 3.5 实验结果及分析

我们从爱荷华州立大学[139]网站上获取了人类染色体20的真实数据集，并对此数据集在MPP处理机深腾1800上采用MPI(绑定C语言)编程对我们的并行算法实现。结果显示，我们提出的基于蚁群算法的序列拼接方法可以高速、高效地完成拼接。

实验显示，一般情况下，算法在2000代以内即可收敛得到较优解。我们分别抽取了2500、6400以及9900个片段进行拼接实验，图3.12表示了该算法的计算时间随

节点数的变化情况（包括输入、输出时间），图3.13表示的是算法的加速比性能比较，而图3.14则表示的算法的效率。

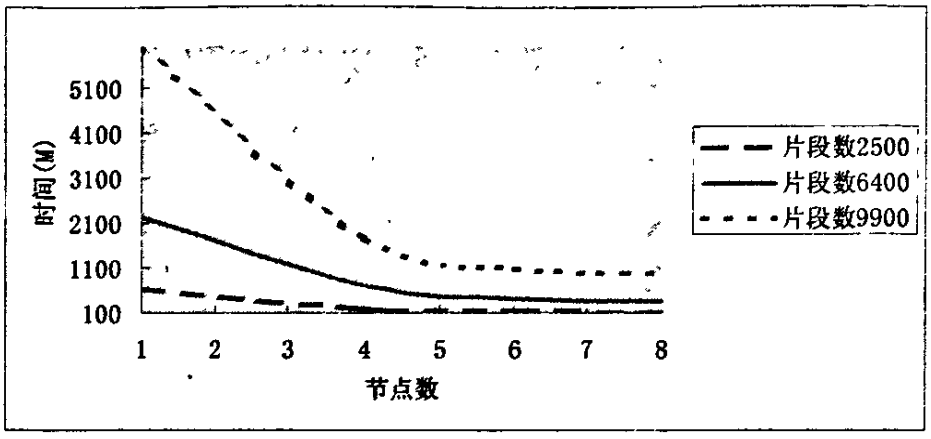



图 3.12 算法计算时间与处理节点数的关系

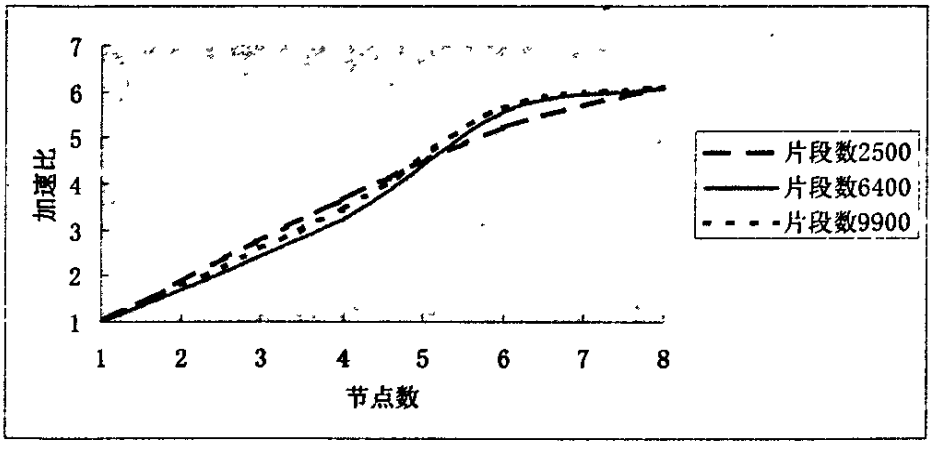



图 3.13 算法加速比与处理节点数的关系

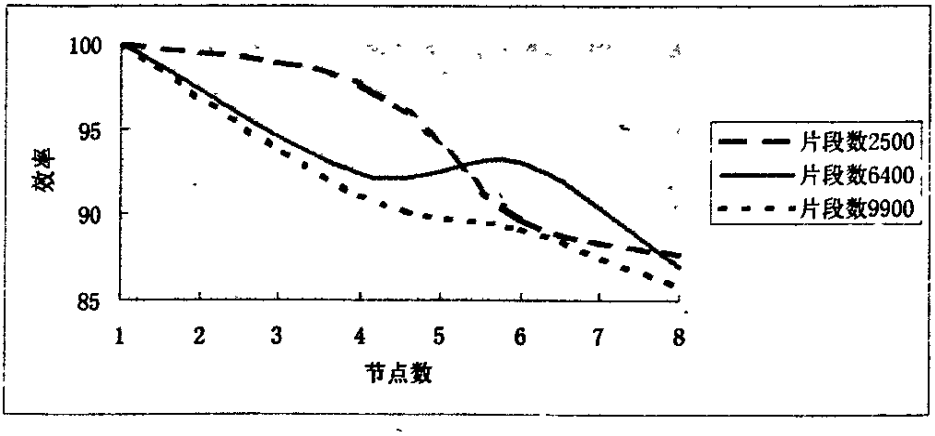



图 3.14 算法效率与处理节点数的关系

由以上结果可以看出，当处理机节点数增加后，算法的加速比会大大加快，算法的计算时间大大降低，但是当节点数增加到5以后，计算速度的增长逐渐变缓，这是由于算法的加速比不能严格地随着处理器的增加而线性增长，这也是符合并行处理的加速比性能的Amdahl定律的。

我们使用上述的测试数据，对本文算法 Ant-Splicing 与同样基于 Hamilton 路径方法的 Phrap[140]算法在拼接质量上进行了比较。比较的结果如图 3.15 所示。在图中，每一“◆”表示一次测试，即一组片段的拼接。横坐标方向表示该组中含有的片段的个数，纵坐标方向表示我们的算法 Ant-Splicing 与 Phrap 相比，在拼接长度上所减少的百分比，即：

$$\frac{\text{length(Phrap)} - \text{length(Ant-Splicing)}}{\text{length(Phrap)}}$$

其中，length（Phrap）、length（Ant-Splicing）分别为两个算法结果的长度。

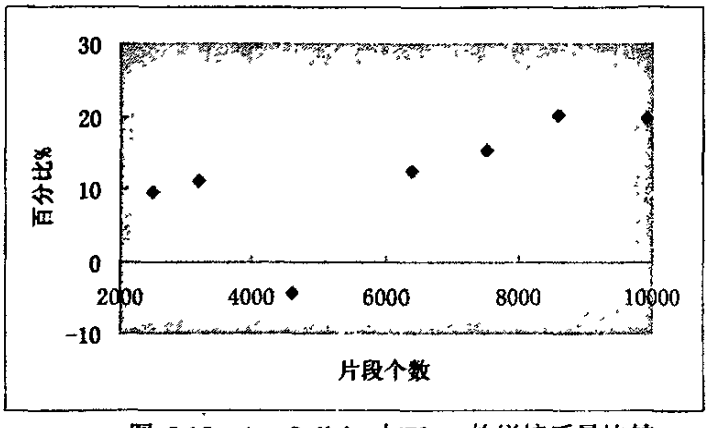



图 3.15 Ant-Splicing与Phrap的拼接质量比较

由图 3.15 可以看出，Ant-Splicing 所得到的拼接结果的长度在大部分情况下要比 Phrap 要短到 10%左右，特别是在片段个数较多的情况下，Ant-Splicing 拼接结果的长度比 Phrap 缩短比例要更大。这说明了我们的算法通过并行化在提高了处理速度的同时，可以保证较好的拼接质量。

# 第四章　基因表达数据的并行双向聚类算法

基因表达数据的双向聚类问题是生物信息学中的一个重要的问题，通过对基因在各种不同实验条件下的表达数据进行双向聚类，可以分析和识别同类基因所共同拥有的基因功能以及转录调控元件。在本章中，我们对基因表达数据进行双向聚类的问题进行了深入的研究，提出了一种并行算法。该算法根据数据集合的大小对双向聚类质量的反单调性，由最小的数据集合开始逐步添加行或列，最终找到所有满足条件的聚类。该算法处理速度快，聚类质量高，性能明显优于其它同类算法。

## 4.1 基因表达数据的双向聚类问题的研究回顾

基因表达调控是分子生物学目前研究的一个重点，基因转录水平上的调控是最重要的环节。随着人类基因组计划的顺利实施，人类和其它模式生物基因组测序工作不久就能顺利完成，这为研究人类基因及基因表达打下了坚实的基础。而基因芯片技术的迅速发展，已使大规模检测基因转录水平、研究基因表达时空规律、分析基因之间的相互作用关系成为现实。

与基因表达调控有关的信息包括基因组DNA 序列、转录因子、调控元件、基因表达数据等。作为基因转录调控信息的载体，基因上游区域中的转录调控元件（regulatory element）在基因转录过程中起着重要的作用。基因调控物质即转录因子通过与调控元件的相互作用来调节基因的转录、控制基因的表达。生物信息学研究人员一直在研究转录调控元件的识别方法，国际上已经出现一些调控元件的分析和识别算法，并取得一些好的结果，可以识别已知的转录调控元件。如果通过分析得知一类基因受到相同蛋白质调控因子的作用，则可以认为这些基因具有共同的转录调控元件，并在这样的假设之下分析共同的转录调控元件。

基因转录调控信息隐藏在基因组序列中，基因表达数据代表基因转录调控的结果，是转录调控信息的实际体现。如果能将基因表达数据与基因调控区域的核酸序列结合起来，综合分析，可望发现基因转录调控信息，揭示基因调控信息组成的规律。通过分析基因表达数据，在基因组中寻找共调控基因，即表达水平上调或下调趋势一致的基因，这些基因具有相同的转录调控信息。然后通过信息学的方法，分析隐藏在基因组序列中的转录调控信息。具体说就是通过聚类分析，将共调控基因

聚集成类，分析和识别同类基因所共同拥有的转录调控元件。这里，转录调控元件是转录调控信息的载体。

基因表达数据分析中所采用的传统的方法是聚类，用相似性度量函数确定基因的相似程度，从而将基因分组。它的前提是：条件样本属同一类，所有条件的变化，对基因的表达几乎没有影响。聚类分析是模式识别中一种非常有吸引力的方法，特别适用于模式分类数不知道的情况。目前，用于基因表达数据的分析方法可以分为有监督和无监督两大类。有监督的方法认为所有表达谱的附加信息都已知(例如基因的功能组、样本的类标签)，并在此基础上构建分类器。无监督的方法假定先验知识很少甚至没有先验知识。这种方法的目标是发现具有统计意义的功能基因组及相关的样本聚类。层次聚类[141]、*K*-平均值法(*K*-mean)[142-143]，模拟退火法[144]和自组织图映射[145](Self-organizing map)是目前应用于基因表达矩阵的常用无监督聚类算法。

当前基因表达数据的大部分研究聚焦于有监督的分析,无监督的方法相对较少,无参数的方法更是不多见,而当领域知识不完备或者很难获取时,无监督的方法却又显得非常重要。

在很多情况下，对基因的聚类要找出在一部分实验条件下表达水平上调或下调一致的基因组，这就是双向聚类问题。 在双向聚类中，不但要将基因进行聚类，而且要同时考虑实验条件的变化。双向聚类所处理的对象是用二维矩阵表示的基因表达数据。 矩阵的行表示基因，每一行表示一个基因的表达模式，列表示不同时间点上或不同实验条件下的样本，每一列表示此样本中所有基因的相对表达水平。基因表达矩阵具有一个显著的特点,既可通过比较表达矩阵的行来分析基因的表达谱,又可通过比较表达矩阵的列来比较样本的表达谱。这种特点,使得我们可以同时在基因表达矩阵的行和列两个方向上进行聚类分析，得到由对象子集和属性子集组成的聚类。

目前已有的双向聚类的算法有如下几类：

1. 行列交叉叠代方法：如G. Getz 等人[52] 的 Coupled Two-Way Clustering 方法(CTWC)，该方法以所有行、列对的聚类为基础，逐步分层聚类，不断识别“稳定”的行/列聚类，直到取得满意的结果。 又如Chun Tang等人[53] 的Interrelated Two-Way Clustering (ITWC)首先将行聚类成若干个组，然后对于每一行组聚类成两个组，再进一步合并行和列的聚类，找出行/列聚类中差距最大的一些聚类对，在这些聚类对中保留最好的1/3个行。重复这样的过程，直到取得满意的结果。

2. 分治策略方法：如 Hartigan [54]等人 的 Block Clustering。 该方法首先将行和列按照它们的均值排序，将其中最好的行或列分解，以降低块间的落差。重复这

样行列分解过程，直到取得满意的 $K$ 个块。这种方法速度快，但如果初始的分解如果不恰当，有可能丢失好的聚类。

3. 贪心叠代搜索方法：如 Cheung & Church 等人的 $\delta$-bicluster 方法和 Jiong Yang 等人[55-56]的 FLOC 方法、Yuval Klugar 等人的 Spectral 方法[57]、Amir Ben-Dor 等人[58]的 *OPSMs* 方法等。Cheng and Church[49]最早将双向聚类概念应用到基因表达数据分析中，他们提出的 δ -biclusters 方法首先从原始矩阵中产生一个满足条件的 *δ -bicluster*。对源数据删除或者添加行、列后的数据元素，若其判定值 $H$ 满足 $H \le \delta$，则为一个聚类。再对源数据进行同样操作，直到找到 $K$ 个聚类。文献[146]方法类似于文献[49]，只是在删除的时候考虑行平均值及所有元素的平均值是否大于给定阈值。Jiong Yang[55-56]提出 FLOC 算法，先随机产生 $K$ 个聚类，再对这些随机产生的 $K$ 个聚类，添加或者删除行、列，改进为满足 $H \le \delta$ 的聚类。但以上方法均存在结果不确定的问题。

4. 穷举方法，如文献[59]提出基于模式的新型聚类模型Pcluster。该方法先对行两两比对，求出对象中所有两行组合的列方向上最大维的聚类。再对列两两比对，求出属性中所有两列组合在行方向上最大维的聚集。再对这些模式进行剪枝处理，产生满足条件的聚类。Sungroh Yoon[60]也是用相似的方法求取聚类。然而，这类方法要求分别对基因和实验条件两两比对。计算量大，时间复杂度高。在模式的剪枝中，有些聚类被移除，所以结果仍然不确定。又如 Tanay[61] 等人的 Statistical-Algorithmic Method for Bicluster Analysis ( SAMBA)算法首先产生一个二分图，然后找出其中的权重最大的子图，继而求出$K$个权重最大的二分子图，对应所求的聚类。

5. 分布参数确认（Distribution Parameter Identification）法：如 Lazzeroni and Owen 等人的 *Plaid Models* 方法[62]以及 Qizheng Sheng,等人的 *Gibbs* 方法[63] 、Eran Segal 等人的 *PRMs* 方法[64-65]等。Plaid Model 方法从一个聚类开始，逐个增加聚类的个数。在由 $K$-1 个聚类增加到 $K$ 个聚类时，要使得类元素平方差的总和最小。表 4.1 列出了各种双向聚类算法的方法的类别和聚类方式的比较。

表4.1　各种双向聚类算法比较

| 算法名称 | 聚类方式 | 所用的方法 |
|---|---|---|
| *Block Clustering* [54] | 一次发现一个类 | 分治策略方法 |
| *δ -biclusters* [49] | 一次发现一个类 | 贪心迭代搜索方法 |
| *FLOC* [55-56] | 同时发现 | 贪心迭代搜索方法 |
| *pClusters* [59] | 同时发现 | 穷举方法 |

| | | |
|---|---|---|
| *Plaid Models* [62] | 一次发现一个类 | 分布参数确认方法 |
| *PRMs* [64-65] | 同时发现 | 分布参数确认方法 |
| *CTWC* [52] | 一次发现一个类 | 行列交叉迭代方法 |
| *ITWC* [53] | 一次发现一个类的集合 | 行列交叉迭代方法 |
| *DCC* [147] | 同时发现 | 行列交叉迭代方法 |
| *δ-Patterns* [148] | 同时发现 | 贪心迭代搜索方法 |
| *Spectral* [57] | 同时发现 | 贪心迭代搜索方法 |
| *Gibbs* [60] | 一次发现一个类 | 分布参数确认方法 |
| *OPSMs* [58] | 一次发现一个类 | 贪心迭代搜索方法 |
| *SAMBA* [61] | 同时发现 | 穷举方法 |
| *xMOTIFs* [149] | 同时发现 | 贪心迭代搜索方法 |
| *OP-Clusters* [150] | 同时发现 | 穷举方法 |

在本章中，我们提出一种对基因表达数据进行双向聚类的并行算法。该算法用子矩阵的行、列元素的差值来衡量聚类的质量，使数据集合的大小对于聚类质量具有反单调性。算法从最小的数据集合，即2*2的子矩阵开始逐步增加行或列，最终找到所有的聚类。以下几节将介绍算法所涉及的概念及基本原理、算法的框架。

## 4.2 双向聚类的并行算法

### 4.2.1 双向聚类基本概念

基因表达数据可用矩阵 $A=(a_{ij})_{m\times n}$ 表示：$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$，其中包含 $n$ 个基因对象，$m$ 个样本，每个元素 $a_{ij}$ 表示第 $i$ 个基因在第 $j$ 个条件下的表达水平值，行向量 $x_i=(x_{i1},x_{i2},\ldots,x_{im})$代表基因 $i$ 在 $m$ 个条件下的表达水平，称为基因 $i$ 的表达谱，列向量 $x_j=(x_{1j},x_{2j},\ldots,x_{nj})^T$ 代表某一条件下的各基因的表达水平。

假设有三组数据$a_1$=(1,2,4,5)，$a_2$=(2,3,5,6)，$a_3$= (4,5,7,10)，图4.1示出了它们的变化趋势曲线。由图中，我们可以看到三组数据所标示的三条曲线在前三个点变化趋势一致。传统聚类中，根据距离的相似性度量函数判定，这三个对象不会被划分到同一聚类中。然而，在双向聚类算法中，这三个对象因为变化的趋势是一致的，可以把它们作为一类。

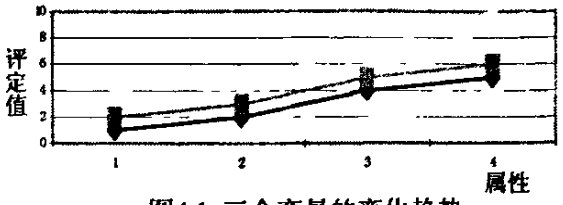



图4.1 三个变量的变化趋势

图4.2是用基因表达矩阵表示的几种双向聚类。在距阵中，行表示基因对象，列表示样本属性，表格中的元素表示某个基因在某种样本上的表达水平。图中的(a)中的所有元素都是一个常数，(b)的各行皆为常数，(c)的各列皆为常数，他们都代表一个双向聚类。在(d)中，所有行对的对应元素之间的差是一个定值，故它也代表一个双向聚类。例如，我们任意取出$r_1$=1, $r_2$=2这样两行，将这两行位于同一列的元素值相减，得到一组差值：1，1，1，1，这组值是相等的。对于其他任意两行，都可以得到同样的结果，这就说明，(d)也形成一个双向聚类。在(e)中，所有行对的对应元素之间的比值是一个定值，故它也可以看成是一个双向聚类。

对于(a)到(d)的情形，我们可以通过求所有行对中对应元素的差值，来判定数据块是否为聚类。对于(e)的情形，我们只要对各个元素取对数，就可化为上述情形来判别。

| | | | |
|---|---|---|---|
| 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 |
| 1.0 | 1.0 | 1.0 | 1.0 |

(a)

| | | | |
|---|---|---|---|
| 1.0 | 1.0 | 1.0 | 1.0 |
| 2.0 | 2.0 | 2.0 | 2.0 |
| 3.0 | 3.0 | 3.0 | 3.0 |
| 4.0 | 4.0 | 4.0 | 4.0 |

(b)

| | | | |
|---|---|---|---|
| 1.0 | 2.0 | 3.0 | 4.0 |
| 1.0 | 2.0 | 3.0 | 4.0 |
| 1.0 | 2.0 | 3.0 | 4.0 |
| 1.0 | 2.0 | 3.0 | 4.0 |

(c)

| | | | |
|---|---|---|---|
| 1.0 | 2.0 | 5.0 | 0.0 |
| 2.0 | 3.0 | 6.0 | 1.0 |
| 4.0 | 5.0 | 8.0 | 3.0 |
| 5.0 | 6.0 | 9.0 | 4.0 |

(d)

| | | | |
|---|---|---|---|
| 1.0 | 2.0 | 0.5 | 1.5 |
| 2.0 | 4.0 | 1.0 | 3.0 |
| 4.0 | 8.0 | 2.0 | 6.0 |
| 3.0 | 6.0 | 1.5 | 4.5 |

(e)

图4.2 双向聚类模型

## 4.2.2 子矩阵的差值及其性质

通过上述双向聚类的定义可以看出，在一个双向聚类中，两行对应元素的差值是判别一个子矩阵是否为双向聚类的重要依据。为了在数据矩阵中找出所有行对的差值满足条件的数据块，我们对这组差值用如下的定义表示：

**定义 4.1**：将数据矩阵中的行子集 $I$ 和列子集 $J$ 所构成的子矩阵记为$\langle I, J\rangle$。对$\langle I, J\rangle$中的某一固定列 $j$，任意 $i_1, i_2$ 两行对应元素差值用 $\mathrm{div}_r(i_1, i_2, j)$表示：$\mathrm{div}_r(i_1, i_2, j) = a_{i1j} - a_{i2j}$。对于 $J$ 中所有列，任意 $i_1, i_2$ 两行元素的差值记为 $\mathrm{div}_r(i_1, i_2, J) = \max_{j_1, j_2 \in J} |\mathrm{div}_r(i_1, i_2, j_1) - \mathrm{div}_r(i_1, i_2, j_2)|$。整个子矩阵$\langle I, J\rangle$元素的差值为 $\mathrm{div}_r(I, J) = \max_{i_1, i_2 \in I} div_r(i_1, i_2, J)$。

从列的角度，我们也可以作出如下类似的定义：

**定义 4.2**：对于子矩阵$\langle I, J\rangle$中的某一固定行 $i$，任意 $j_1, j_2$ 两列对应元素差值用 $\mathrm{div}_c(i, j_1, j_2)$表示：$\mathrm{div}_c(i, j_1, j_2) = a_{ij1} - a_{ij2}$。对于 I 中的所有行，任意两列 $j_1, j_2$ 元素的差值为 $\mathrm{div}_c(I, j_1, j_2) = \max_{i_1 \in I} |\mathrm{div}_c(i_1, j_1, j_2) - \mathrm{div}_c(i_2, j_1, j_2)|$。整个子矩阵$\langle I, J\rangle$元素的差值为 $\mathrm{div}_c(I, J) = \max_{j_1, j_2 \in J} \mathrm{div}_c(I, j_1, j_2)$。

**引理 4.1**：在表达矩阵的子矩阵$\langle I, J\rangle$中，有 $\mathrm{div}_r(I, J) = \mathrm{div}_c(I, J)$。

**证明**：对任意两行 $i_1$，$i_2 \in I$，任意两列 $j_1$，$j_2 \in J$，我们有

$$\begin{aligned}
&\mathrm{div}_r(i_1, i_2, j_1) - \mathrm{div}_r(i_1, i_2, j_2) \\
&= (a_{i1j1} - a_{i2j1}) - (a_{i1j2} - a_{i2j2}) \\
&= (a_{i1j1} - a_{i1j2}) - (a_{i2j1} - a_{i2j2}) \\
&= \mathrm{div}_c(i_1, j_1, j_2) - \mathrm{div}_c(i_2, j_1, j_2)
\end{aligned}$$

而 $\mathrm{div}_r(I, J) = \max_{i_1, i_2 \in I} div_r(i_1, i_2, J)$。

$$\begin{aligned}
&= \max_{i_1, i_2 \in I} \max_{j_1, j_2 \in J} |div_r(i_1, i_2, j_1) - div_r(i_1, i_2, j_2)| \\
&= \max_{i_1, i_2 \in I} \max_{j_1, j_2 \in J} |div_c(i_1, i_1, j_2) - div_c(i_2, i_1, j_2)| \\
&= \max_{j_1, j_2 \in J} \max_{i_1, i_2 \in I} |div_c(i_1, j_1, j_2) - div_c(i_2, j_1, j_2)| \\
&= \max_{j_1, j_2 \in J} div_c(I_1, j_1, j_2) = div_c(I, J)
\end{aligned}$$

证毕。

根据上述引理，我们可以将 $div_c(I, J)$ 及 $div_r(I, J)$ 统一记为 $div(I, J)$。

对于表达矩阵中的子矩阵$\langle I, J\rangle$，如果要形成一个双向聚类，必须有 $div(I, J) \approx 0$。如图 4.2 中的(a)到(d)，它们的 div 值全为 0。但在实际应用中，象这样的使所有行对之间的差值全为 0 的情况很少见。在实际的基因表达数据中，要找到完全使 div 为 0 的双向聚类是不现实的。因此，可以定义一个允许误差 $\delta$，对于子矩阵$\langle I, J\rangle$，如果 $div(I, J) < \delta$，就可以认为它是一个双向聚类。

**引理 4.2**：设 $I_1 \subset I$, 则 $\mathrm{div}(I_1, J) \leqslant \mathrm{div}(I, J)$。

**证明**：设 $I=I_1 \cup I_2$, 且 $I_1 \cap I_2 \neq \varnothing$则

$$\begin{aligned}\mathrm{div}(I, J) &= \max_{j_1, j_2 \in J} \mathrm{div}_c(I, j_1, j_2) = \max_{j_1, j_2 \in J} \mathrm{div}_c(I_1 \cup I_2, j_1, j_2) \\ &= \max_{j_1, j_2 \in J} (\max(\mathrm{div}_c(I_1, j_1, j_2), \mathrm{div}_c(I_2, j_1, j_2))) \\ &= \max\{\max_{j_1, j_2 \in J} \mathrm{div}_c(I_1, j_1, j_2), \max_{j_1, j_2 \in J} \mathrm{div}_c(I_2, j_1, j_2)\} \\ &= \max(\mathrm{div}(I_1, J), \mathrm{div}(I_2, J)) \geq \mathrm{div}(I_1, J)\end{aligned}$$

证毕。

**引理 4.3**：若 $J_1 \subset J$, 则 $\mathrm{div}(I, J_1) \leqslant \mathrm{div}(I, J)$。

**引理 4.4**：若 $I_1 \subset I$, $J_1 \subset J$, 则 $\mathrm{div}(I_1, J_1) \leqslant \mathrm{div}(I, J)$。

引理 4.3，4.4 的证明与引理 4.2 的证明类似。

**定理 4.1**：

(1)在数据矩阵中，若有行集合 $I_1 \subset I$ 以及列集合 $J$, 如果$\langle I_1, J\rangle$不能构成一个聚类，则$\langle I, J\rangle$也不能构成一个聚类。

(2)在数据矩阵中，若有行集合 $I$ 以及列集合 $J_1 \subset J$, 如果$\langle I, J_1\rangle$不能构成一个聚类，则$\langle I, J\rangle$也不能构成一个聚类。

(3)在数据矩阵中，若有行集合 $I_1 \subset I$, 列集合 $J_1 \subset J$, 若$\langle I_1, J_1\rangle$不能构成一个聚类，则$\langle I, J\rangle$也不能构成一个聚类。

**证明**：(1)因 $I_1 \subset I$，由引理 4.2 知 $\mathrm{div}(I_1, J) \leqslant \mathrm{div}(I, J)$；又因$\langle I_1, J\rangle$不能构成一个聚类，则有 $\mathrm{div}(I_1, J) \geqslant \delta$，因而有 $\mathrm{div}(I, J) \geqslant \delta$，即$\langle I, J\rangle$也不能构成一个聚类。

对(2)、(3)的证明与(1)类似。

证毕。

以上的引理说明了如果$\langle I, J\rangle$的一个子阵不能构成聚类，$\langle I, J\rangle$也必不能构成聚类，即上述定义的差值 div 对于子矩阵的大小满足反单调性。利用这样的性质，我们的算法采用从最小的子矩阵，即 2*2 子矩阵开始，逐步扩大的方法来寻找聚类。算法首先判别这些子矩阵是否构成聚类，如果是聚类，则在它的基础上进行行、列的扩展，以得到更大的聚类。如果某个子矩阵不能构成聚类，则没有必要对其扩展。因为根据反单调性，对其任何扩展不可能得到聚类。这样可以删除这个子矩阵，以缩小搜索空间，减少计算量。

### 4.2.3 互斥集合及其性质

在我们的算法中，为了进一步减少在扩展过程中的计算量，我们记录在某一个

行集合 $I$（列集合 $J$）下明显不能与列 $j$（行 $i$）构成聚类的列（行）集合，在以后的行集合 $I$（列集合 $J$）下的，包含列 $j$（行 $i$）的列（行）扩展中，则不能考虑这些集合中的列（行）。为此，我们定义如下的行（列）互斥集合。

**定义 4.3**：对于行集合 $I$，以及某列 $j$，集合 $ME(I,j)=\{k|\mathrm{div}_C(I, j, k)\geq\delta\}$ 称为对于行集合 $I$ 的列 $j$ 的互斥列集合；对于列集合 $J$，以及某行 $I$，集合 $ME(J,i)=\{l|\mathrm{div}_R(i, l, J)\geq\delta\}$ 称为对于列集合 $J$ 的行 $i$ 的互斥行集合。

**引理 4.5**：①设 $J_1\subset J_2$，则 $ME(J_1, i)\subset ME(J_2, i)$

②设 $I_1\subset I_2$，则 $ME(I_1, j)\subset ME(I_2, j)$

**证明**：①设 $i_1\in ME(J_1, i)$ 即有 $\mathrm{div}_R(i, i_1, J_1)\geq\delta$，记 $I=\{i, i_1\}$ 即有 $\mathrm{div}(I, J_1)\geq\delta$，因为 $J_1\subset J_2$，所以有 $\mathrm{div}(I, J_1)\leq\mathrm{div}(I, J_2)$，故有 $\mathrm{div}(I, J_2)\geq\delta$，即 $\mathrm{div}_R(i, i_1, J_2)\geq\delta$。即说明 $i_1\in ME(J_2, i)$，因此有 $ME(J_1, i)\subset ME(J_2, i)$

②的证明与①类似。

**证毕。**

**定理 4.2**：①$ME(J_1, i)\cup ME(J_2, i)\subseteq ME(J_1\cup J_2, i)$

②$ME(I_1, j)\cup ME(I_2, j)\subseteq ME(I_1\cup I_2, j)$

**证明**：因为 $J_1\subset (J_1\cup J_2)$，由引理 4.5 知 $ME(J_1, i)\subset ME(J_1\cup J, i)$，同理可知 $ME(J_2, i)\subset ME(J_1\cup J, i)$，因此有：$ME(J_1, i)\cup ME(J_2, i)\subseteq ME(J_1\cup J_2, i)$

②的证明与①类似。

**证毕。**

**定义 4.4**：对于数据矩阵中，若有子矩阵 $<I, J>$ 及 $<I', J'>$，若有 $I\subset I'$ 且 $J\subset J'$，我们称子矩阵 $<I', J'>$ 包含了 $<I, J>$，记为 $<I, J>\subset<I', J'>$。

如果 $<I', J'>$ 为一个聚类，我们可以知道它所包含的所有子矩阵皆为聚类，我们在求聚类的实际应用中感兴趣的仅仅是最大的聚类，它的定义如下：

**定义 4.5**：在数据矩阵中，如果一个子矩阵 $<I, J>$ 是一个聚类，而对任何 $<I', J'>$，若 $<I, J>\subset<I', J'>$，则 $<I', J'>$ 不是聚类，则称 $<I, J>$ 为一个最大的聚类。

我们对基因表达矩阵进行双向聚类，就是对已知阈值 $\delta$，求得其所有的最大聚类。

### 4.2.4 算法的基本思想

我们的算法利用聚类的反单调性，从数据矩阵的最小子矩阵（即 2*2 的子阵）开始，利用阈值 $\delta$，逐个判别它们是否构成聚类。如果能构成聚类，说明有可能在其基础上构建更大的聚类；如果不能构成聚类，根据反单调性，它不可能继续扩展

成聚类，则将其删除而不继续处理。

对于一个聚类 $R$，如果对其添加一行（列）能构成一个更大的聚类，我们称此过程为对聚类 $R$ 的行（列）扩展，我们称聚类 $R$ 为可扩展的。对于不可扩展的聚类，它就是一个最大的聚类，算法对其进行保存。对于可扩展的聚类，在对其进行所有的可能的行、列扩展以后，算法对其进行删除，对其扩展所产生的聚类再进一步进行扩展操作。

上述的扩展过程是从 2*2 的子矩阵开始逐层进行的，为了便于识别与处理，我们对每一个所产生的聚类定义一个层号。

**定义 4.6**：对于某一形成聚类的子矩阵$<I, J>$，$<I, J>$的层号 level$(I, J)$定义为：

$$\text{level}(I,J)=\begin{cases}1 & |I|=2 \text{ 且 } |J|=2\\ \text{level}(I',J')+1 & (I,J) \text{由} (I',J') \text{扩展而成}\end{cases}$$

在对一个聚类$<I, J>$进行扩展的时候，为了避免丢失可能产生的聚类，对它进行行扩展和列扩展的过程只能分别进行，不能同时进行。例如，在如下的数据矩阵 A 中：

$$A=\begin{bmatrix}1&2&3&4\\2&3&4&5\\3&4&5&6\\4&5&6&9\end{bmatrix}$$

设δ=1，$I$={1，2，3}，$J$={1，2，3},因为 $\text{div}(I,J)=0<\delta$，$<I,J>$构成一个聚类，在对$<I,J>$进行扩展时，如果我们同时将 $I$ 扩展至 $I'=\{1,2,3,4\}$，$J'=\{1,2,3,4\}$，由于 $\text{div}(I',J')=3>\delta$,不能构成聚类。但如果我们单独对$<I,J>$进行行扩展得到$<I',J>$，由于 $\text{div}(I',J)=0<\delta$，$<I',J>$构成一个聚类。同样地，如果我们单独对$<I,J>$的列进行扩展得到$<I,J'>$，由于 $\text{div}(I,J')=0<\delta$，$<I,J'>$也构成一个聚类，而对$<I,J>$同时进行行、列扩展，这两个聚类就会失去。因此，在算法中，我们同时使用两个表 $R$ 及 $C$ 来分别记录准备进行行、列扩展的可扩展聚类。在算法中，每当由第 $i$ 层的聚类$<I,J>$扩展成一个 $i$+1 层的聚类$<I',J'>$时，无论$<I',J'>$是由$<I,J>$进行行或列扩展而成，$<I',J'>$都要同时被存入行扩展表 $R$ 及列扩展表 $C$ 之中。

算法在处理第 $i$ 层可扩展聚类时，对 $R$、$C$ 中所有第 $i$ 层所有可扩展聚类进行所有可能的行、列扩展，所产生出的新的聚类为第 $i$+1 层聚类，被同时存入 $R$、$C$ 表之中。在第 $i$ 层处理结束时，如果第 $i$ 层中的某一聚类$<I,J>$已被扩展而产生了新的聚类，则$<I,J>$不是一个最大聚类，可以被删去。如果它并没有被扩展成任何新的聚类，则该聚类为一个最大聚类，仍然保存在 $R$ 或 $C$ 之中。因此在第 $i$ 层处理结束时，$R$ 及 $C$ 中所存的第 $i$ 层聚类全部为最大聚类，它们是不可扩展的。

为了减少在对聚类扩展过程中的计算量，我们在对<$I$,$J$>进行行扩展时，仅考虑那些与$I$中的行不互斥的行。设$I=\{i_1,i_2,...,i_r\}$，我们仅考虑集合$S-\bigcup_{k=1}^{R}\mathrm{ME}(J,i_k)$中的行。类似地，设$J=\{j_1,j_2,...,j_c\}$。对<$I$,$J$>进行列扩展时，仅考虑集合$T-\bigcup_{k=1}^{C}\mathrm{ME}(I,j_k)$中的列。对ME的计算可以在算法逐层对聚类进行扩展的过程中相应地逐层进行。在算法开始时，我们对所有ME集合赋以初值∅。然后由2*2的不能构成聚类的子矩阵中获得互斥信息：设$I=<i_1,i_2>$，$J=<j_1,j_2>$若<$I$,$J$>不能构成聚类，则有：$\mathrm{ME}(I,j_1)=\mathrm{ME}(I,j_1)\bigcup\{j_2\},\mathrm{ME}(I,j_2)=\mathrm{ME}(I,j_2)\bigcup\{j_1\},\mathrm{ME}(J,i_1)=\mathrm{ME}(J,i_1)\bigcup\{i_2\}$，$\mathrm{ME}(J,i_2)=\mathrm{ME}(J,i_2)\bigcup\{i_1\}$。对在以后的每层的扩展操作中所得到的新聚类<$I$,$J$>，对$\mathrm{ME}(I,j)$，$j\in J$及$\mathrm{ME}(J,i)$，$i\in I$的计算可以根据定理4.2用下式来计算：

$$\mathrm{ME}(I,j)=\bigcup_{\tilde{I}\in I(|\tilde{I}|=2)}\mathrm{ME}(\tilde{I},j),\mathrm{ME}(J,i)=\bigcup_{\tilde{J}\in J(|\tilde{J}|=2)}\mathrm{ME}(\tilde{J},i)。$$

## 4.3 算法的描述

综上所述，我们算法的框架描述如下：

**Algorithm bicluster(*Y*);**

**Input**: 基因表达矩阵 *Y*[*n*][*m*];

**Output**: *Y* 中所有双向聚类；

**begin**

1. 对矩阵 *Y*[*n*][*m*]计算阈值 δ；
2. **For** *Y*[*n*][*m*]的所有 2×2 的子矩阵<$I$,$J$>=<{$i_1$, $i_2$}, {$j_1$,$j_2$}> **pardo**
3. **if** div($I$,$J$)< δ **then**

将<$I$,$J$>同时加入行扩展集合 *R* 和列扩展集合 *C* 中,定义其层号为 1

**else**

**ME**($I$,$j_1$)={$j_2$}; **ME**($I$,$j_2$)={$j_1$}; **ME**($J$,$i_1$)={$i_2$}; **ME**($J$,$i_2$)={$i_1$};

**endif**

**endfor**

4. $i$=1;
5. **Repeat**
6. **For** *R* 的所有 $i$ 层的子矩阵<$I$,$J$> **pardo** Extend(*R*,*I*,*J*, $i$);
7. **For** *C* 的所有 $i$ 层的子矩阵<$I$,$J$> **pardo** Extend(*C*,*I*,*J*, $i$);

8.　$i=i+1$
9.　**Until** 集合的 $i$ 层元素中无可扩展的子矩阵;
10.　**For**　$C$ 中所有的子矩阵$<I,J>$ **pardo**
11.　**For**　$R$ 中所有的子矩阵$<I',J>$ **pardo**
12.　If $<I,J>\subset <I',J>$ then 在 $C$ 中去除$<I,J>$;
13.　If $<I',J>\subset <I,J>$ then 在 $C$ 中去除$<I',J>$;
14.　**Endfor**
15.　**endfor**
16.　输出 $R$ 和 $C$ 中所有的子矩阵
**end**

算法的第 6 行的过程 Extend$(R, I,J, i)$ 对 $R$ 中层号为 $i$ 的子矩阵$<I,J>$进行行扩展，将可扩展的子矩阵$<I',J>$同时加入行扩展集合 $R$ 和列扩展集合 $C$ 中,定义其层号为 $i+1$; 如果$<I,J>$被扩展过，则将其从 $R$ 中删除； 类似地，算法的第 7 行的过程 Extend$(C,I,J, i)$ 对 $C$ 中层号为 $i$ 的子矩阵$<I,J>$进行行扩展，将可扩展的子矩阵$<I,J'>$同时加入行扩展集合 $R$ 和列扩展集合 $C$ 中,定义其层号为 $i+1$; 如果$<I,J>$被扩展过，则将其从 $C$ 中删除。

算法 Extend$(R, I,J, i)$ 的描述如下,算法中分别用 $S$、$T$ 表示 $Y$ 中所有行、列号的集合:

**Algorithm** Extend$(R, I,J, i)$;

**Input**: 子矩阵$<I,J>$, 行扩展集合 $R$, 列扩展集合 $C$,当前层号 $i$; 互斥表的集合 ME;

**Output**: 修改后的行扩展集合 $R$, 列扩展集合 $C$, 修改后的互斥表的集合 ME;

**Begin**
1. 设 $I=\{i_1,i_2,\cdots,i_k\}$，记 $I'=S-(\mathrm{ME}(J,i_1)\cup \mathrm{ME}(J,i_2)\cup\cdots\cup \mathrm{ME}(J,i_k))$;
2. *ext*=false;
3. **for**　$l=i_1,i_2,\cdots,i_k$ **do**
4.　　**for**　$I'$ 中所有大于 $l$ 的行号 $t$　**do**
5.　　**if** div$(I\cup\{t\},J)<\delta$ **then**
6.　　*ext*=true;
7.　　Insert $(I\cup\{t\},J,\ i+1,R)$;
8.　　Insert $(I\cup\{t\},J,\ i+1,C)$;
9.　　**endif**
10.　**endfor**
11.　**endfor** $l$
12.　**if** *ext*=true **then** 将子矩阵$<I,J>$从 $R$ 中删除 **endif**;

**end**

算法 Extend(C,*I,J, i*) 和算法 Extend(*R, I,J, i*) 的描述是类似的。

算法 Extend(*R, I,J, i*) 的第 7 行的过程 Insert ( $I\cup\{t\}$,*J*， *i* +1,*R*)是将新产生的聚类 <$I\cup\{t\}$,J>插入到 *R* 中，并建立相应的互斥表。算法 Extend(*R, I,J, i*) 的第 8 行的过程 Insert ( $I\cup\{t\}$,*J*， *i* +1,*C*)是将新产生的聚类<$I\cup\{t\}$,*J*>插入到 *C* 中，并建立相应的互斥表。

算法 Insert ( *I*,*J*， *i*,*C*)的描述如下：

**Algorithm** Insert ( *I*,*J*， *i*, *C*);

**Input**: 子矩阵<*I*,*J*>及其层号 *i*, 列扩展集合 *C*，互斥表的集合 ME;

**Output**: 修改后的列扩展集合 *C*, 修改后的互斥表的集合 ME;

**Begin**

1. 在 *C* 中寻找子矩阵<*I*,*J*>;
2. **if** *C* 中无<*I*,*J*>**then**

   在 *C* 中插入<*I*,*J*>,其层号为 *i*;

   **for** *T* 中所有行号 *j* **do**

   建立互斥表 ME（*I*,*j*）$=\bigcup_{\tilde{I}\subset I} \mathrm{ME}(\tilde{I}, j)$ (*j*=1,2,...,*m*)
3. **endfor**
4. **endif**

**end**

算法 Insert ( *I, J*， *i, R*)和算法 Insert ( *I, J*， *i, C*)的描述是类似的。

## 4.4 实验结果及分析

为了对本文算法 bicluster 进行评价，我们对它应用人类基因表达数据[151]进行实验，并将它和 $\delta$-*Bicluste*[51]以及 Pcluster[152]算法在效率和准确性方面进行比较。实验表明了与其他类似算法相比，本文方法具有较快的处理速度和较高的聚类质量，能更快速的找到比其他算法更多符合条件要求的数据。

首先我们对本文算法进行串行执行，与其它两种算法进行性能比较。在我们的算法及Pcluster中，皆取 $\delta$=6，最低行数限制*Nr*=5，最低列数*Nc*=30，在算法中，令 *k*=50。我们取矩阵元素为500行*10列，本文算法找到了50个聚类，所用时间为9.1秒，$\delta$-*Bicluster*所用时间为12秒，Pcluster算法所用时间为10秒。取数据为1000*17

时，本文算法所用时间为35.12秒，*δ-Bicluster*所用时间为50秒，Pcluster所用时间为40秒。分别取不同个数的行和列，各类算法求得相同聚类所需要的时间比较，如表4.2和图4.3所示。

表4.2 本文算法与其他算法执行相同聚类的时间比较（s）

| 各类算法 | 500*10 | 500*15 | 1000*15 | 1000*17 | 1000*19 |
|---|---|---|---|---|---|
| *δ-Bicluster*[51] | 12 | 20 | 35 | 50 | 65 |
| Pcluster[154] | 10 | 18 | 30 | 40 | 52 |
| 本文算法 bicluster | 9.1 | 16.89 | 28.05 | 35.12 | 49.74 |

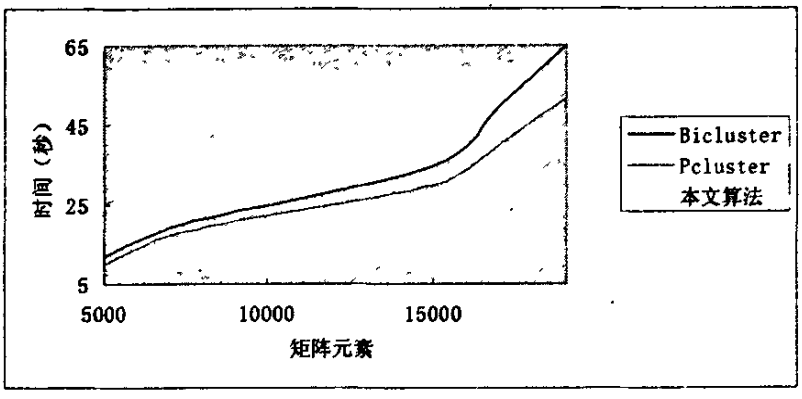



图 4.3 本文算法与其他算法执行相同聚类的时间比较

由表 4.2 和图 4.3 可以看出，本文算法与其他两种算法相比，在相同条件下，其串行聚类的速度也比其他算法更快。

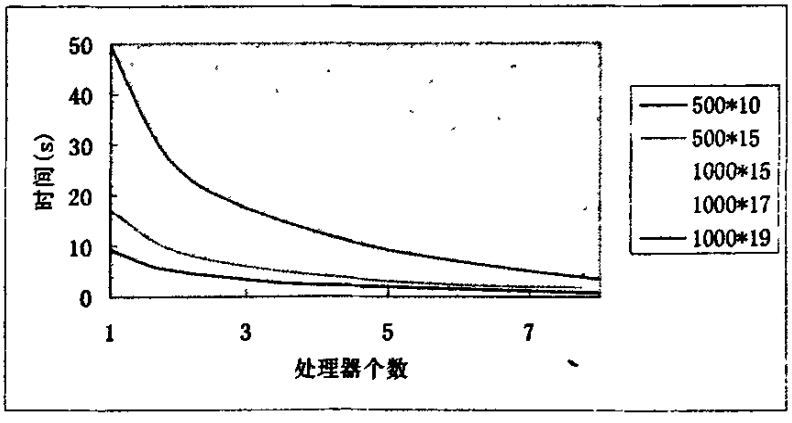



图 4.4 本文算法的计算时间随处理器个数的变化情况

图 4.4 示出了本文算法的计算时间随处理器个数的变化情况，从图中我们很容易看出，当处理机节点数增加后，算法的加速比会大大加快，算法的计算时间大大降低，但是当节点数增加到 7 以后，计算速度的增长逐渐变缓，这是由于处理机之间的通讯时间和其它额外开销，使得算法的加速比不能严格地随着处理器的增加而线性增长，这也是符合并行处理的加速比性能的 Amdahl 定律的。

我们再对各种算法的聚类结果的质量作一比较。图4.5表示的是我们的算法bicluster在10个基因和50个基因表达条件下的一组聚类结果。其中粗线为Pcluster算法中丢失的记录，点线为 $\delta$ -*Bicluster*算法中丢失的记录。由该图我们可以看出，本文的算法bicluster可以比其它类似算法找到更多符合条件的聚类。

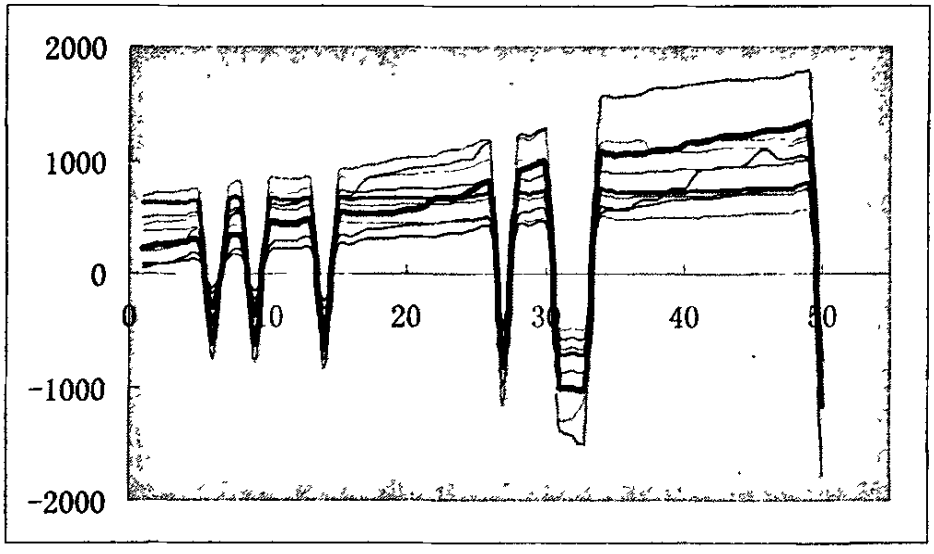



图 4.5 本文算法找到的聚类

# 第五章　结论与展望

## 5.1 总结

生物信息学是整个生命科学发展的重要组成部分，它的核心是基因组信息学，包括基因组信息的获取、处理、存储、分配和解释。在基因组信息学中，生物序列的比对、拼接和基因表达数据的聚类是最关键、最重要的操作。由于这些问题具有计算量大的特点，对计算机的处理速度要求高. 而这些问题大部分本质上是组合优化问题,不象向量运算那样具有有规则的数据结构和相关关系，因而研究他们的并行化有很大的意义。

本文对生物信息学领域中的一些问题的并行计算进行的深入的研究,提出了相应的并行算法，通过试验取得了很好的效果。

本文的研究工作和所取得的主要贡献及创新之处如下：

1. 我们提出一种快速的最长公共子序列算法FAST_LCS，该算法通过对字符串建立相应的同字符后续表，随后对于相应的初始同字符对，并行地在该表中逐层地搜索其后继同字符对，得到所有的后继同字符对及相应的层次值。最后由最大层次值的同字符对进行回溯，依次求得其所有前驱同字符对，最后得到相应的比对结果。这种基于同字符后续表的算法同样可以用到多序列的最长公共子串问题中。算法使用了剪枝和跳跃技术，提高了处理速度。与其它经典的LCS算法相比，不但能够取得准确的结果，而且在速度、效率上有了很大的提高。

2. 我们深入研究了生物序列的拼接问题，提出了一种高效的并行算法。该算法提出了后缀索引的概念以代替后缀树，首先对所有的序列片段建立后缀索引;然后对所有的序列片段 $i$,在所有其他序列片段 $j$ 的后缀索引中查找匹配度最高且最长的后缀,匹配长度记为 $l_{ij}$, 再根据 $l_{ij}$ 建立有向图; 然后使用并行蚁群算法寻找最长的哈密尔顿路径;最后根据路径得出拼接方案。该算法以后缀索引代替后缀树，大大减少了计算量，有利于并行决算。算法利用蚁群算法解决 TSP 问题的优势，减少了优化时间。与其它类似的算法相比，取得的结果准确，在速度、效率上有了很大的提高。

3. 我们还对基因表达数据进行双向聚类问题进行了深入的研究，提出了一种进行双向聚类的并行算法。该算法根据数据集合的大小对于双向聚类质量的反单调

性，由最小的数据集合开始逐步添加行或列，最终找到所有满足条件的聚类。该算法处理速度快，聚类质量高，性能明显优于其他类似算法。

本文的研究工作将并行处理技术应用到生物信息学的研究之之中，对生物信息学中的一些问题提出的高效的并行算法都在并行计算机深腾1800上用MPI(C绑定)编程运行，都使用生物信息的标准数据库中的测试数据进行试验，取得的试验结果表明，这些算法不但处理速度快，而且结果质量高，说明并行计算机是生物信息学的研究中的有力工具，有关问题的高效并行算法的开发，会有力促进生物信息学的研究。

## 5.2 研究展望

由于生物信息数据的规模极其巨大，开展生物信息处理算法并行化方向的研究、对有关问题的高效并行算法的开发，会有力促进生物信息学的研究。在我们今后的工作中，我们将在如下两个方面进行继续的深入研究：

1. 将上述算法继续优化，以提高它们的性能，如速度、效率、可扩展性等，使之实用化；

2. 对更多的生物信息学的问题进行研究，设计其高效的并行算法，如系统发生树问题、蛋白质二级、三级结构预测问题等。

3. 将我们的研究工作和生物学科的一些具体研究课题结合起来，以超级计算机为有力工具，解决实际研究的问题。

# 参考文献

[1] 张春霆. 生物信息学的现状与发展. 世界科技研究与发展，2000, 22(6):17-20。

[2] 张阳德. 生物信息学. 北京:科学出版社，2004.

[3] 张成岗，贺福初. 生物信息学方法与实践. 北京:科学出版社，2002

[4] 王哲. 生物信息学概论. 西安:第四军医大学出版社，2002

[5] 郝柏林，张淑誉. 生物信息学手册. 上海:上海科学技术出版社，2000

[6] T.K. Attwood and D.J. Parry-smith. Introduction to Bioinformatics. 罗静初等译. 生物信息学概轮. 北京大学出版社 .2002年4月. 131页.

[7] Minoru Kanehisa. 后基因组信息学，孙之荣等译. 北京二清华大学出版社，2002

[8] Kanehisa, Post-genome Informatics, Oxford Univ Press, 2000, 6-9.

[9] 史忠植。知识发现[M]。北京:清华大学出版社，2001-9: 36-47。

[10] J. Setubal and J. Meidanis, Introduction to computational molecular biology, Science publishing company, Beijing, (2003)

[11] Runsheng Chen, Bioinformatics, Acta biophysica Sinica, (1999),16(1):5-12

[12] 生物信息学基础，孙啸，陆祖宏，谢建明编著，清华大学出版社，(2005)

[13] Tompa, M. 2000. Lecture notes on biological sequence analysis. Technical Report (2000), Department of Computer Science and Engineering, University of Washington, Seattle.

[14] 生物信息学教程，东南大学吴健雄实验室，http://www.lmbe.seu.edu.cn/chenyuan/xsun/bioinfomatics/Web/Index.html

[15] Waterman M S. Introduction to Computational Biology, Maps, Sequences and Genomes. London: Chapman & Hall, 1998.

[16] A.Aho, D.Hirschberg, and J. Ullman, Bounds on the Complexity of the Longest Common Subsequence Problem, J. Assoc. Comput. Mach., vol. 23, no. 1, pp. 1-12, Jan. 1976.

[17] O.Gotoh, An improved algorithm for matching biological sequences, J. Molec. Biol. 162 (1982) 705-708.

[18] E.W.Mayers, W. Miller, Optimal Alignment in Linear Space, Comput. Appl. Biosci. 4(1) (1998) 11-17.

[19] D.S.Hirschberg, A Linear Space Algorithm for Computing Maximal Common

Subsequences, Commun. ACM 18 (6) (1975) 341-343.

[20] Edmiston E W, Core N G, Saltz J H, et al. Parallel processing of biological sequence comparison algorithms. International Journal of Parallel Programming, 1988, 17(3): 259-275.

[21] Lander E. Protein sequence comparison on a data parallel computer. In: Proceedings of the 1988 International Conference on Parallel Processing, 1988. 257-263.

[22] Galper A R, Brutlag D L. Parallel similarity search and alignment with the dynamic programming method. Technical Report, California: Stanford University, 1990.

[23] A. Aggarwal and J. Park, Notes on Searching in Multidimensional Monotone Arrays, Proc. 29th Ann. IEEE Symp. Foundations of Comput. Sci. 1988, pp. 497-512

[24] A. Apostolico, M. Atallah, L. Larmore , and S. Mcfaddin, Efficient Parallel Algorithms for String Editing and Related Problems, SIAM J. Computing, vol. 19, pp. 968-988, Oct. 1990.

[25] M. Lu, H. Lin, Parallel Algorithms for the Longest Common Subsequence Problem, IEEE Transaction on Parallel and Distributed System, vol 5. No. 8, August 1994.

[26] Y. Robert, M. Tchuente, A Sytolic Array for the Longest Common Subsequence Problem, Inform. Process. Lett.21 (1985) 191 – 198.

[27] J. H. Chang, O.H. Ibarra, M.A. Pallis, Parallel Parsing on a one-way array of finite-state machines, IEEE Trans. Computers C-36 (1987) 64-75.

[28] G. Luce, J.F. Myoupo, Systolic-based Parallel Architecture for the Longest Common Subsequences Problem.

[29]V.Freschi, A.Bogliolo, Longest Common Subsequence between Run-length-encoded Strings : a New Algorithm with Improved Parallelism.

[30] Carrillo H, Lipman DJ: The multiple sequence alignment problem in biology.SIAM J. Appl. Math. 48, 1073-1082 (1988).

[31] Lipman DJ, Altschul SF, Kececioglu JD: A tool for multiple sequence alignment. Proc.Natl. Acad. Sci. USA 86, 4412-4415 (1989).

[32] Stoye J, Moulton V, Dress AW: DCA: an efficient implementation of the divide-andconquer approach to simultaneous multiple sequence alignment. Comput. Appl. Biosci.13(6), 625-6 (1997).

[33] Reinert K, Stoye J, Will T: An iterative method for faster sum-of-pair multiple equence alignment. Bioinformatics 16(9),808-814 (2000).

[34] Thompson, JD, Higgins, DG and Gibson, TJ (1994) CLUSTAL W: improving

the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. Nucleic Acids Research,1994, vol.22,No.22.4673-4680.

[35] Feng D-F, Doolittle RF: Progressive sequence alignment as a prerequisite to correct phylogenetic trees. J. Mol. Evol. 25,351-360 (1987)

[36]P.Green.Documentationforphrap,http//bozeman.mbt.washington.edu/phrap.docs /phrap.htm1,2003.07.10.

[37] G.G. Sutton, O. White, M..D. Admas and A.R. Kerlavage. TIGR assembler: a new tool for assembling large shotgun sequencing projects. Genome Science and Technology, 1995, 1:9-19.

[38] 张博锋. 全基因组DNA测序中的片段拼接方法及其并行处理. 长沙:国防科技大学研究生院，2002.11.

[39] P. A. Pevzner, Haixu Tang and M. S. Waterman. A New Approach to Fragment Assembly in DNA Sequencing.The 5th Annual International Conference on Computational Molecular Biology (RECOMB2001)，Montreal, Canada, April, 2001.

[40] Xiaoqiu Huang, Glen Herrmannsfeldt, Ted Jones, Jun Qian,Samuel L. Rash, Charles P.mith and Cecilie Boysen. CAP4-Paracel's DNA Sequence Assembly Program.http:/lwww.paracel.com, 2000.9.

[41] X. Huang and A. Madan. CAP3:A DNA Sequence Assembly Program. Genome Research, 1990, 9:868—877.

[42] J. Kececioglu, Y Jun, Separating repeats in DNA sequencing assembly. Proceedings of the 5th ACM Conference on Computational Molecular Biology, 2001.

[43]中科院计算所生物信息学实验室. 生物信息、处理并行算法的研究. http://www.bioinfo.org.cn/biology/biodevelope.htm，2003.11.15.

[44] 张法，刘志勇，等. 生物序列拼接算法一PHRAP的并行化研究. 成都:第七届全国并行计算年会，2002.8.

[45] Jiawei Han, Micheline Kamber. 数据挖掘概念与技术[M]. 北京：机械工业出版社。

[46] Hartigan, J.A. Direct clustering of a data matrix[J]. JASA 67:123-129. 1972.

[47] Mirkin, B.Mathematical Classiffication and Clustering. Dordrecht: Kluwer. 1996.

[48] Pavel Berkhin. Survey of clustering data mining techniques. Accrue Software, Inc.2002

[49] Yizong Cheng and George M. Church. Biclustering of expression data[J]. In

Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology (ISMB'00), pages 93–103, 2000.

[50] Ya Zhang, Hongyuan Zha. Clustering of Time-Course Gene Expression Data[J]. Pages 1-2, 2004.

[51] Jiong Yang, Wei Wang, Haixun Wang, and Philip Yu. Enhanced biclustering on expression data[C]. In Proceedings of the 3rd IEEE Conference on Bioinformatics and Bioengineering, pages 321–327, 2003.

[52] G. Getz, E. Levine, and E. Domany. Coupled two-way clustering analysis of gene microarray data. In *Proceedings of the Natural Academy of Sciences USA*, pages 12079–12084, 2000.

[53] Chun Tang, Li Zhang, Idon Zhang, and Murali Ramanathan. Interrelated two-way clustering: an unsupervised approach for gene expression data analysis. In *Proceedings of the 2nd IEEE International Symposium on Bioinformatics and Bioengineering*, pages 41–48, 2001. INESC-ID TEC. REP. 1/2004, JAN 2004 31

[54] J. A. Hartigan. Direct clustering of a data matrix. *Journal of the American Statistical Association (JASA)*, 67(337):123–129, 1972.

[55] Jiong Yang, Wei Wang, Haixun Wang, and Philip Yu. Æ-clusters: Capturing subspace correlation in a large data set. In *Proceedings of the 18th IEEE International Conference on Data Engineering*, pages 517–528, 2002.

[56]Yuval Klugar, Ronen Basri, Joseph T. Chang, and Mark Gerstein. Spectral biclustering of microarray data: coclustering genes and conditions. In *Genome Research*, volume 13, pages 703–716, 2003.

[57] Amir Ben-Dor, Benny Chor, Richard Karp, and Zohar Yakhini. Discovering local structure in gene expression data: The order–preserving submatrix problem. In *Proceedings of the 6th International Conference on Computacional Biology (RECOMB'02)*, pages 49–57, 2002.

[58] Haixun Wang, Wei Wang, Jiong Yang, and Philip S. Yu. Clustering by pattern similarity in large data sets. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 394–405, 2002.

[59] Sungroh Yoon, Giovanni De Micheli. An Application of Zero-suppressed Binary Decision Diagrams to Clustering Analysis of DNA Microarray Data. In Proceedings of the 26th Annual International Conference of the IEEE EMBS. 2004, pages 2925-2928.

[60] Amos Tanay, Roded Sharan, and Ron Shamir. Discovering statistically significant biclusters in gene expression data. In *Bioinformatics*, volume 18 (Suppl. 1), pages S136–S144, 2002.

[61] Laura Lazzeroni and Art Owen. Plaid models for gene expression data. Technical report, Stanford University, 2000.

[62] Qizheng Sheng, Yves Moreau, and Bart De Moor. Biclustering micrarray data by gibbs sampling. In *Bioinformatics*, volume 19 (Suppl. 2), pages ii196–ii205, 2003.

[63] Eran Segal, Ben Taskar, Audrey Gasch, Nir Friedman, and Daphne Koller. Rich probabilistic models for gene expression. In *Bioinformatics*, volume 17 (Suppl. 1), pages S243–S252, 2001.

[64] Eran Segal, Ben Taskar, Audrey Gasch, Nir Friedman, and Daphne Koller. Decomposing gene expression into cellular processes. In *Proceedings of the Pacific Symposium on Biocomputing*, volume 8, pages 89–100, 2003.

[65] Needleman, S. and Wunsch, C. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* (1970) 48:443--453.

[66]Smith T F, Waterman M S. Identification of common molecular subsequence[J], J.Mol.Biol.(1981)147:195-197

[67] M. Crochemore, G. M. Landau, and M. Ziv-Ukelson, A sub-quadratic sequence alignment algorithm for unrestricted cost matrices, *SIAM J. Comput.* (2003) 32, 5: 1654--1673.

[68] U.Manber and G.Myers. Suffix arrays:A new method for on-lime string searches. SIAM Journal on Computing, (1993) 22(5):935-948.

[69] Gusfield, D. Algorithms on strings, trees, and sequences: Computer science and computational biology. Cambridge University Press, Cambridge, UK. (1997).

[70] Bray N., Dubchak I, Pachter L. AVID: A Global Alignment Program, Genome Res. (2003) Jan;13(1):97-102.

[71] Christopher Lee, Catherine Grasso and Mark F. Sharlow. Multiple sequence alignment using partial order graphs. Bioinformatics, (2002) Vol 18 no. 3, 452-464

[72] Benjamin Raphael, Degui Zhi, Haixu Tang and Pavel Pevzner, A novel method for multiple alignment of sequences with repeated and shuffled elements, Genome Research (2004)14:2336-2346.

[73] Gotoh O: Significant Improvement in Accuracy of Multiple Protein Sequence Alignments by Iterative Refinements as Assessed by Reference to Structural Alignments. J. Mol. Biol. (1996)264(4), 823-838.

[74] Berger MP, Munson PJ: A novel randomized iterative strategy for aligning multiple protein sequences. Comput. Appl. Biosci. (1991)7: 479-484.

[75] Wang Y, Li KB An adaptive and iterative algorithm for refining multiple

sequence alignment Comp. Biol. Chem. (2004)28, 141-148

[76] Jonathan D. Moss and Colin G. Johnson, An ant colony algorithm for multiple sequence alignment in bioinformatics, 2003, Computer Science.

[77] B. Morgenstern and T. Werner DIALIGN: Finding local similarities by multiple sequence alignment. Bioinformatics (1997)14, 290-294.

[78] Morgenstern B DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. Bioinformatics, (1999) vol.15, no. 3, pp. 211-218(8).

[79] Morgenstern B, Dress A, Wener T: Multiple DNA and protein sequence based on segment-to-segment comparison. Proc. Natl. Acad. Sci. USA (1996)93: 12098-12103.

[80] Ivo Van Walle, Ignace Lasters and Lode Wyns, Align-m—a new algorithm for multiple alignment of highly divergent sequences. Bioinformatics.( 2004)Vol. 20 no. 9, pages 1428–1435

[81] A.Krogh. An introduction to hidden markov models for biological sequences. In: S. L. Salzberg, D. B. Searls and S. Kasif. Computational Methods in Molecular Biology. Elsevier, (1998): 45-63

[82] Rasmussen TK and Krink T, Improved Hidden Markov Model training for multiple sequence alignment by a particle swarm optimization evolutionary algorithm hybrid. Biosystems. (2003) Nov;72(1-2):5-17.

[83] Kevin karplus and Birong Hu, Evaluation of protein multiple alignments by SAM-T99 using the BaliBASE multiple alignment test set, Bioinformatics. (2001) vol.17.No.8.713-720

[84] Notredame C, Higgins DG: SAGA:sequence alignment by genetic algorithm. Nucleic Acids Res. (1996)24, 1515-1524.

[85] Kim J, Pramanik S, Chung MJ: Multiple Sequence Alignment using Simulated Annealing. Comp. Applic. Biosci. (1994)10(4), 419-426.

[86] Kazutaka Katoh , Kazuharu Misawa1 , Kei-ichi Kuma and Takashi Miyata MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. Nucleic Acids Res. (2002) Jul 15; 30(14):3059-66

[87] P.Zhao and Tao Jiang J. A heuristic algorithm for multiple sequence alignment based on blocks. Combinatorial Optimization (2001) Mar 5(1):95–115.

[88] Tariq Riaz, Yi Wang, Kuo-Bin Li.Multiple Sequence Alignment Using. Tabu Search. In Proc. Second Asia-Pacific Bioinformatics Conference (APBC2004)

[89] Notredame C, Higgins DG, Heringa J T-Coffee: A novel method for fast and accurate multiple sequence alignment. J Mol Biol. (2000) Sep 8;302(1):205-217

[90] Edgar RC. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinformatics. (2004) Aug 19;5(1):113.

[91]Do.CB,Brudno.M.,and Batzoglou, S. ProbCons: Probabilistic.consistency-based multiple alignment of amino acid sequences. In. Proceedings of the Thirteenth National Conference on Artificial. Intelligence,( 2004) pp. 703–708.

[92] C. Notredame, Recent progresses in multiple sequence. alignment: a survey, Pharmacogenomics (2002)3(1). 131–144

[93] Thompson, JD, Plewniak, F. and Poch, O. A comprehensive comparison of multiple sequence alignment programs. Nucleic Acids Research, (1999)27: 2682 - 2690.

[94] Lassmann,T. and Sonnhammer,EL Quality assessment of multiple alignment programs. FEBS Lett. (2002), 529, 126–130

[95] http://www.tigr.org/tdb/benchmark

[96]Smith T.F.,Waterman M.S.Identification of common molecular subsequence.Journal of Molecular Biology,1990,215:403-410.

[97] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J., Basic local alignment search tool, *J. Mol. Biol.*, 215:403-410, 1990.

[98] http://alpha10.bioch.virginia.edu/fasta_www/cgi/

[99] http://www.ebi.ac.uk/services/

[100]Baxevanis AD, Francis BF. 1998. Bioinformatics: A practical guide to the analysis of gnes and proteins. John Wiley & Sons, New York.

[101]Bishop and Rawlings (eds). 1987. Nucleic acid and protein sequence analysis: A practical approach. IRL Press, Oxford.

[102]Setubal JC, Meidanis J. 1997. Introduction to Computational Molecular Biology. PWS PUBLISHING COMPANY.

[103]Venter JC, Adams MD, Sutton GG, Kerlavage AR, Smith HO, Hunkapiller M. 1998. Shotgun sequencing of the human genome. Science , 280(5369):1540-1542.

[104]Bonfield JK, Smith Kf, Staden R. 1995. A new DNA sequence assembly program. Nucleic Acids Res., 23(24): 4992-4999.

[105]Idury RM, Waterman MS. 1995. A new algorithm for DNA sequence assembly. J Comput Biol.,2(2):291-306.

[106] Batzogou S., Jaffe D., Stanley K., Butler J., Gnerre S., Manceli E., Berger B., Mesirov J.P., Lander E.S., ARACHNE: A whole genome shotgun assembler. Genome Research, 2002, 12:177~189.

[107] Huang X., Nadan A., CAP3: A DNA sequence assembly program. Genome Research, 1999, 9:868~878.

[108]Huang X., An improved sequence assembly program. Genomics, 1999,33:21~31.

[109]Sutton G.G., White O., Adams M.D., Kerlavage A.R., TIGE assembler: A new tool for assembling large shotgun sequencing projects, Genome Science&Technology, 1995,1(1):9~19.

[110] 涂俐兰，王能超，陈莹，梅启鹏. 生物序列拼接及其算法. 生命科学研究，2003,7(2):79-82.

[111] Pevzner P.A., Tang Haixu, Waterman M.S., An Eulerian path approach to DNA fragment assembly,PNAS,2001,&9748~9753.

[112] Pevzner P.A., Tang Haixu, Waterman M.S., A new approach to fragment assembly in DNA sequencing, In Proceedings of the RECOMB,2001,256~267.

[113] EULER Software. http://nbcr.sdsc.edu/euler/arch2.htm, 2003.7.

[114] EULER Portal User Guide. http://nbcr.sdsc.edu/euler/about.htm, 2003.7.

[115] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. www.pnas.orgycgivdoiy10.1073ypnas.2001.7

[116]McCreight E M. A space-economical suffix tree construction algorithm[J].Journal of ACM, 1976,23(2):262-272.

[117]Ukkonen E. On-line construction of suffix trees [ J].Algorithmica, 1995,14(3):249-260.

[118] The human genome project[EB/OL]. http: // www.nhgri.nih.gov/HGP/, 1990-10-01.

[119] Benson D A, Karsch-Mizrachi I, Lipman D J,et al.Genbank[J].Nucleic Acids Research, 2000,28(1):15-18.

[120] Hunt E, Atkinson M P, Irving R W. A database index to large biological sequences[A]. 27th VLDB[C]. Roma:Morgan Kaufmann, 2001.139-148.

[121] Sahinalp S J, Vishkin U. Symmetry breaking for suffix tree construction [A]. 26th ACM Symposium on Theory of Computing[C]. Montreal: ACM, 1994.300-309.

[122] Hariharan R. Optimal parallel suffix tree construction[A].26th ACM Symposium on Theory of Computing[ C].Montreal: ACM, 1994.290-299.

[123] JaJa J.An introduction to parallel algorithms[M]. New York: Addison-Wesley Publishing Company, 1992.1.

[124] K. Li, Y. Pan, and S. Q. Zheng, "Fast and Processor Efficient Parallel Matrix Multiplication Algorithms on a Linear Array with a Reconfigurable Pipelined Bus System", *IEEE Transactions on Parallel and Distributed Systems*, (1998)

August Vol. 9, No. 8, pp. 705-720.

[125] Dorigo M., Maniezzo V., Colorni A. Ant system: Optimization by a colony of coorperating agents [J]. IEEE Transactions on Systems, Man, and Cybernetics-Part B, (1996), 26(1): 29-41.

[126]Colorni A, Dorigo M, Maniezzo V. Distributed optimization by ant colonies[C].Paris, France:Proc of the First EuropeanConf on Artificial Life, (1991).134-142.

[127]李秋霞, 贺达汉, 长有德, 刘丽丹. 蚂蚁取食行为研究概况. 宁夏农学院报 2000, 21(2):94-97。

[128] Dorigo, M., Gambardella, L.M. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computaing, (1997),1(1):53-56.

[129] Gutjahr J.W. A Graph-based Ant System and its convergence. Future Generation Computer Systems, (2000),16(8):873-888.

[130] Talbi E.G, Roux O, Fonlupt C, Robillard D.Parallel Ant Colonies for the quadratic assignment problem. Future Generation Computer Systems, (2001),17(4): 441-449.

[131] Maniezzo V., Carbonaro A. An ANTS heuristic for the frequency assignment problem, Future Generation Computer Systems, (2000), 16(8):927-935.

[132] D CostaandAHertz. Ants can color graph~EJ. Journal of the 0peational Research Society, (1997), 48: 295—305.

[133] Ling Chen, Xiao-Hua Xu, Yi-Xin Chen, An Adaptive Ant Colony Clustering Algorithm, Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, (2004) August 26-29.

[134] DI CARO G, DORIGO M. AntNet : A mobile agents approach to adaptive routing [ R] . Belgium: UniversitéLibre de Bruxelles , (1997).

[135] 梁栋，霍红卫. 自适应蚁群算法在序列比对中的应用. 计算机仿真, (2005), 22(1):100-106.

[136]Marco Dorigo, Gianni Di Caro. Ant algorithms for discrete optimization. Artificial Life, 1999,5(3):137-172

[137] 孔令军，张兴华，陈建国. 基本蚁群算法及其改进. 北华大学学报(自然科学版) (2004)12:572-574

[138] 胡娟, 王常青, 韩伟, 全智. 蚁群算法及其实现方法研究. 计算机仿真, (2004)7:110-114

[139]Human Chromosome 20 sequences[OL]. http://seq.cs.iastate.edu, 2005.

[140]http://www.phrap.org

[141]JAIN AK, MURTYMN, FLYN PJ. DataClustering: A Review[J].ACM Computing Surveys, 1999, 31(3): 264-323.

[142]HARTIGAN JA. ClusteringAlgorithms[R]. JohnW iley and Sons,New York. 1975, 99: 351.

[143]MACQUEEN J. Some Methods for Classification and Analysis of MultivariateObservation[A]. Proc, 5th Berkeley Symp[C]. Math, Statics, Prob, 1967, 1: 281-297.

[144]JM P. VAN LAARHOVEN EH L. AARTS. SimulatedAnnealing:Theory and Applications[M]. D. ReidelPublishingCompany, Bos-ton, 1987.

[145] KOHONEN T. Self-Organization and AssociativeMemory[A]. vol-ume 8 of Springer Series in Information Sciences[C]. Springer-Verlag, Berlin, 1989, 312.

[146] Zonghong Zhang, Alvin Teo. Mining Deterministic Biclusters in Gene Expression Data. In Proceedings of the Fourth IEEE Symposium on Bioinformatics and Bioengineering (BIBE'04), pages 2173-2180,2004

[147] Stanislav Busygin, Gerrit Jacobsen, and Ewald Kramer. Double conjugated clustering applied o leukemia microarray data. In *Proceedings of the 2nd SIAM International Conference on Data Mining, Workshop on Clustering High Dimensional Data*, 2002.

[148] Andrea Califano, Gustavo Stolovitzky, and Yunai Tu. Analysis of gene expression microarays for phenotype classification. In *Proceedings of the International Conference on Computacional Molecular Biology*, pages 75–85, 2000.

[149] T. M. Murali and Simon Kasif. Extracting conserved gene expression motifs from gene expression data. In *Proceedings of the Pacific Symposium on Biocomputing*, volume 8, pages 77–88, 2003.

[150] Jinze Liu and Wei Wang. Op-cluster: Clustering by tendency in high dimensional space. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pages 187–194, 2003.

[151] http://arep.med.harvard.edu/biclustering/

[152] H.Wang, W.Wang, J.Yang, and P.Yu . Clustering by Pattern Similarity in Large Data Sets[J]. In ACM SIGMOD, Madison, Wisconsin, 2002.

# 致谢

本论文是在我的导师陈崚教授的指导下完成的，在我三年的硕士生学习期间，陈教授不仅在学习上给予悉心指导，而且在生活上给予我关怀和照顾，在此向他表示衷心的感谢。另外，陈教授认真负责的工作态度、严谨的治学精神、渊博的知识让我感到由衷的钦佩，也必定会让我在以后的学习、工作中受益非浅。同样衷心感谢师母李秀兰老师对我学习和生活上的关心和照顾。

感谢殷新春教授、沈洁教授、李斌副教授、李云副教授、杨云副教授、胡孔法副教授、陈宏建老师对我学习上的帮助；感谢陶春明老师、谢晓悦老师、林雪美老师对我生活和学习的关心和帮助；感谢所有教导过我的老师们，没有他们的辛勤耕耘，就没有我今天的进步，感谢他们对我的指导和帮助；感谢与我一起度过三年硕士研究生学习生活的蔡俊杰、于广建、顾颀、唐小丽、郭静、黄志艳等同学；感谢同一课题组的邹凌君、何萍、李拓、刘佳佳、徐涛等师弟师妹，谢谢他们对我的关照和帮助；感谢全体计算机系的师生和工作人员，使我在这儿度过愉快的学习和研究生活。

在此，我特别感谢我的父母，感谢他们给予我无私的关怀、默默的鼓励和无微不至的照顾，他们的支持是我精神的支柱和力量的源泉，是我不断取得进步的动力！

感谢和祝福所有关心我的人！

二零零七年四月

于扬州大学信息工程学院

# 在学期间科研情况

## 一、发表的学术论文

1. Chen Y, Wan A, Liu W: A fast parallel algorithm for finding the longest common sequence of multiple biosequences. BMC Bioinformatics 2006, ( 4): S4.
2. Wei Liu, Yixin Chen, Ling Chen, Ling Qin, A Fast Parallel Longest Common Subsequence Algorithm Based on Pruning Rules, Proceeding of International MultiSymposiums on Computer and Computational Science (IMSCCS-06),Symposion of Computations in Bioinformatics and Bioscience(SCCS06),IEEE Press,2006,pp179-187
3. Wei Liu, Ling Chen, A Parallel Algorithm for Longest Common Substring of Multiple Biosequences, DCABES 2006 PROCEEDINGS,Volume I pp13-17
4. Wei Liu, Ling Chen, A Parallel Algorithm for Solving LCS of Multiple Bioseqences, Proceedings of the fifth International Conference on Machine Learning and Cybernetics, (ICMLC2006),pp4316-4319
5. Yixin Chen, Yi Pan, Juan Chen, Wei Liu, Ling Chen, Multiple Sequence Alignment by Ant Colony Optimization and Divide-and-conquer, Proceedings of International Workshop on Bioinformatics Research and Applications, UK, May 2006
6. Yixin Chen, Yi Pan, Juan Chen, Wei Liu, Ling Chen, Partitioned optimization algorithms for multiple sequence alignment, Proceedings of Second IEEE Workshop on High Performance Computing in Medicine and Biology (HiPCoMB-2006), April 2006.
7. 刘维，陈峻，最长公共子序列的快速算法及其并行实现，计算机应用. Vol. 26 NO. 6 June 2006, P1422-1427
8. 刘维，陈峻，陈娟，多重生物序列的最长公共子序列的并行算法，计算机科学，2006 Vol. 33 No. 7，p72－75
9. 刘维，陈峻，基于剪枝跳跃技术的最长公共子序列算法，计算机科学，2006 Vol. 33 No. 8，p322－324
10. 陈娟，刘维，陈峻，LARPBS 上的快速可扩放的序列比对算法，计算机科学，2006 Vol. 33 No. 7，p338－341

## 二、参与的科研项目

[1] 国家自然科学基金项目，“蚁群算法并行化、收敛性和新型模型研究”，60473012

[2] 国家自然科学基金项目 “蚁群优化算法的理论基础研究”， 项目批准号：60673060

[3] 国家科技攻关计划项目子课题，“重大环境污染事故危险数据库建立研究”，题号：003BA614A－14

[4] 国家科技基础条件平台工作项目，环境高危数据库建设，项目编号：2994DKA20310

[5] 南京大学国家软件新技术重点实验室基金项目，“蚁群算法的加速收敛及并行化计算”

[6] 江苏省自然科学基金项目 “蚁群算法并行化、新模型和收敛性研究”， 项目批准号：BK2005047

[7] 江苏省教育厅自然科学基金项目 “可重构的光总线并行计算模型的研究”，02KJB520009

## 三、参加学术会议情况

1. First International Multi-Symposiums on Computer and Computational Sciences(IMSCCS/06),Hangzhou,Zhejiang,China 20-24, June 2006
2. Fifth International Conference on Machine Learning and Cybernetics,Dalian,13-16 August 2006
3. The fifth International Conference on Distributed Computing and Applications for business, engineering and sciences （ DCABES2006 ） 2006.10, Hangzhou, China
4. 2005 年全国 BASICS 会议，2005.8，江苏扬州
5. 2006 年全国理论计算机科学学术年会，2006.8，吉林长春
6. 全国计算机体系结构学术年会（ACA’06），2006.8，四川成都