# 摘要

重复体识别问题是现代生物信息学中基因组分析的一个基本研究课题。通过识别重复体可以发现基因组的进化规则和许多疾病的遗传规律。许多转位子重复体序列作为可编码区域重复出现在基因组序列中，识别这些重复体对基因组解码起到了非常重要的作用。虽然现在已经存在多种算法解决重复体识别问题，但是这些算法在很多方面还不够完善。针对当前存在的问题，本文提出了一种基于种子序列的方法来求解重复体识别问题。

本文提出了两个重复体识别算法 RepeatSearcher 和 GSRSearcher，这两个算法的共同点在于都是基于对种子序列扩展的识别算法。RepeatSearcher 算法的核心是对包含种子的序列通过双序列局部比对构建多序列局部比对，结合限定范围的空位罚分策略，通过比对得分值扩展调和序列，同时扩展每一个重复体序列。这种方法的优点在于在扩展调和序列的同时可以确定每一个重复体序列的精确边界。构建多序列局部比对在很大程度上防止了基于高分相似对算法的边界不精确性。GSRSearcher 算法继承了算法 RepeatSearcher 基于种子序列扩展的特点，结合 Gibbs 采样统计方法，综合考虑了基因组中背景碱基对结果的影响，使识别出来的重复体家族序列更加精确。通过概率统计策略的 GSRSearcher 算法收敛速度明显比通过比对的算法 RepeatSearcher 更合理，而且可以判断出重复体序列的精确边界。

本文最后使用这两个算法测试了 12 种哺乳动物的部分基因组序列，将实验结果和重复体数据库 RepBase 以及当前流行的算法 RECON 的结果进行了比较，结果表明：本文提出的算法在大部分情况下均优于 RECON 算法的结果，是一种高效的重复体识别算法。

关键词：生物信息学 重复体识别 调和序列 精确边界 种子序列

# Abstract

Repeat Identification is the most common fundamental subject of genome analysis in modern bioinformatics. Through repeat identification, the roles in genome evolution and inheritance of diseases can be found. Many transposons and retrotransposons which contain coding regions exist in genome sequences. Identification of these repeats is important to decode genome. Although a lot of algorithms were proposed to solve this problem, but there is not an optimal algorithm of repeat identification. For current flaws we present a novel kind of algorithm for repeat identification based on seed sequences.

Two methods, RepeatSearcher and GSRSearcher, were proposed in this paper, which based on extension of seed sequences. Using sequences which include seed, RepeatSearcher translates local pair-wise alignment into multiple sequences alignment, combining gapped penalty in limited area. This algorithm extends consensus sequence according score of alignment, and at the same time extends every repeat sequence. In this way, the accurate boundaries of repeat sequences can be conformed when extending consensus sequence. Multiple sequences alignment greatly avoid the imprecise of high score pairs. GSRSearcher inherits the way of seed's extension and make use of statistical function of Gibbs Sampling. Considering infection of background in genome, the repeat family sequences which were identified will be more accurate. Using probability statistical policy, the speed of convergence in GSRSearcher is more reasonable then the speed of convergence in RepeatSearcher and can judges the boundaries of repeat sequence exactly.

In the end, the report tests twelve kinds of genome sequences of mammal on RepeatSearcher and GSRSearcher, and then compares the output with RepBase and the output of RECON. The result shows that our algorithm is better than RECON and is an effect algorithm.



**Keyword**: **bioinformatics repeat identification consensus sequence accurate boundaries seed sequence**

# 创新性声明

秉承学校严谨的学风和优良的科学道德，本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切法律责任。

本人签名：白帆　　日期 2007.1.26

# 关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属西安电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律署名单位为西安电子科技大学。(保密的论文在解密后遵守此规定)

本学位论文属于保密，在____年解密后适用本授权书。

本人签名：白帆　　日期 2007.1.26

导师签名：霍红卫　　日期 2007.1.26

# 第一章 绪论

## 1.1 引言

DNA 双螺旋结构的发现开创了分子生物学时代，从此生物学家可以从分子水平上研究生物的生长发育等现象。从二十世纪九十年代起，分子生物学研究发生了翻天覆地的变化。许多物种全基因测序的完成和高通量实验技术的发展，使人们可以从基因组水平研究细胞的各种生命过程。

1986 年，著名的生物学家，诺贝尔奖获得者 Renato Dulbecco 在 Science 上首次提出了“人类基因组计划”(Human Genomic Project, 简称 HGP)[1][2]。“人类基因组计划”于 1990 年 10 月正式启动。HGP 旨在阐明人类基因组 30 亿个碱基序列对的序列[3]，发现所有人类基因并搞清楚其在染色体上的位置，破译人类全部遗传信息，使人类第一次在分子水平上全面地认识了自我。在短短的十多年时间里，人们不但提前完成了整个人类基因组的全部测序，同时还完成了许多其它模式生物的全基因组测序，如 40 多种微生物，线虫，拟南芥等。计划的内容也从最初的结构基因组学，发展产生了功能基因组学，蛋白质组学，转录组学等等。重复体序列在基因组中决定了基因组的功能，因此功能基因组学的发展很大程度上推动了重复体序列识别问题的研究。

早在 20 世纪 60 年代中期，科学家们就发现了许多基因组包括大量高度重复的 DNA 序列分支(Reassociation Kinetics Experiments , C-Value Paradox)。这些序列分支也就是这里提到的重复体序列，它们后来被大致归类为如下五类[4][5][6]：

(1) Simple Repeats。Simple Repeats 是几个 DNA 碱基(一般为 1 到 5 个碱基)集合的复制品，比如 A,CA,CGG 等。

(2) Tandem Repeats。Tandem Repeats 的长度一般在 100 到 200，重复出现在染色体的着丝点(Centromere)和端粒(Telomere)上，而且高频率的以某种明显的规律出现在一个很小的范围内。

(3) Segmental Duplications。Segmental Duplications 经常被拷贝到基因组中距离比较远的位置，其长度比较长，一般在 1 万到 30 万之间。

(4) Interspersed Repeats。Interspersed Repeats 包括 LINES(long interspersed nuclear elements)和 SINES(short interspersed nuclear elements)。一般作为转录(Transcription)序列或基因组中跳跃子(Transposon)存在。

(5) Retroposed Pseudogenes。Retroposed Pseudogenes 在基因组序列中不起什么作用但是又大量存在，一般被称为伪基因序列，也许有某些还没

有被发现的作用需要进一步研究证实。

随着当前大量新基因组序列的发现，重复体识别问题已经成为当今生物信息学基因组分析中首当其冲的问题之一，在真核生物基因中重复体DNA占据了非常重要的地位，比如约20%的Caenorhabdits elegans 与 Caenorhabditis briggsae的基因组[7]和约50%的人类基因组[8]已经被识别为重复体DNA序列。

基因组中的重复体序列给基因组的测序和分析带来了很多问题。大量重复体序列的出现经常使序列装配变得很混乱，高度重复和保守的序列使这种情况更为严重。低重复度的序列也会影响到序列的装配，尤其是在整个基因组序列上进行鸟枪法测序[9]的时候。当一个基因组被装配完成之后，重复体序列将呈现出基因组中未知的而且非常具有重要生物学意义的功能规则。不同的重复体家族有着不同的功能，比如转位子的功能就是使移动元素围绕着基因组中心旋转，而且很大程度上挈带着从其它类型基因中难以获得的编码信息。还有一些重复体序列在基因组中大量存在，但是他们并不起任何作用，也有可能是它们所起的作用还没有被人们所发现。在许多真核细胞中的着丝点上大部分都被重复体序列所占据[10]，而且在染色体的两端调节聚合反应序列和亚调节聚合反应序列可以扩展出成千上万的重复序列。由于某些未知的原因，这些重复体序列同时也大量存在于基因组其它位置。根据上面所述的种种原因，可以得知重复体序列的识别和归类在基因组序列的装配和分析中起着非常重要的作用。而且由于重复体在很大程度上决定了基因组的进化方向[11]，并且在实际生物信息学的应用中重复体也被用来确定一个基因组的家族关系，所以重复体的识别是分析新基因组序列的一个重要组成部分。当今随着新基因组序列数目的不断增多，在新的大规模基因组中识别重复体的需求已经迫在眉睫[12]。

## 1.2 重复体识别问题的发展和现状

自从上世纪 60 年代基因组中重复的基因组序列片断被发现和生物信息学研究的大规模展开，重复体识别问题的研究得到了巨大的进步。解决重复体识别问题的各种方法也相继地被提出来。

在基因组中识别重复体家族序列是一个很有挑战性的算法问题，当前在基因组 DNA 序列分析中存在很多种识别重复体的方法，比如识别重复出现的子串或串联子串(tandem repeats)。知名的 RepeatMasker 算法[13]使用已知的重复体序列数据库，并应用串匹配算法 cross_match 或者 Smith-Waterman[14]从一个新的基因组序列中寻找和数据库中序列相匹配的序列片断，最后根据数据库搜索结果判定基因组中的片断属于哪一个重复体家族。RepeatMasker 虽然算法速度很慢，但是它所识别出来的结果被誉为“黄金结果[15]”。所以这种算法现在一直还在广泛使用。

算法 MaskerAid[16](Bedell JA., 2000)也使用的同样的方法来识别重复体，但是速度更快，因为 MaskerAid 使用启发式算法 WU-BLAST[17]（Gish, http://blast.wustl.edu）数据库搜索程序代替了 cross_match 程序和 Smith-Waterman 程序。由于算法效率的问题大部分算法都限定了输入序列的最大长度，而且限定的这些长度值远远小于真核生物的染色体长度。针对这些问题出现了一些基于后缀树系统的重复体识别算法，这些算法很大程度上解决了输入长度限制的问题，比如类似 MUMmer[18]的 RepeatMatch[19]算法和 REPuter 算法[20][21]。这两个算法是非常有效的识别工具，它们可以在完整的真核生物基因组(10-100Mb)中寻找到所有相同重复体。虽然这些算法的输出精确地给出了这些完全相同的重复体在基因组中的具体位置，但是却没有提供任何这些重复体序列的结构信息。REPuter 算法虽然提供了一个用来产生重复体图形的可视化工具，而且尽管这个功能对于识别重复体的位置非常重要，但是它还是不能提供基因组中所有相同或者相似重复体的整体信息。使用算法 REPuter 和 RepeatMatch 测试一个完整的细菌基因组，检测它们的输出可以发现许多相同的重复体序列是相似重复体的复制品。对于任何完整的基因组序列，完全手工的识别和描述重复体序列之间的相似性是非常复杂的。Agarwal 和 States 在 1993 年提出了使用单聚类方法[22]解决重复体识别的问题，这个方法开始先将序列相似对中重叠的序列片断链接成一个更长的片断，然后通过链接后的片断之间的相似性比对将这些片断归类到相应的重复体家族中。如果基因组中所有的重复体序列都是以全长存在的，而且之间都有规律的间隔，那么单链聚类算法可以得到很好的结果。但是如果大部分重复体序列存在缺失或者重叠度很高，那么单链聚类方法的运行结果是很不理想的。针对单链方法的这些问题，RepeatFinder[23]算法和 RECON[15](Bao and Eddy, 2002)算法扩展和提高了单链聚类方法，这两个算法在高分相似对的基础上构建相似片断集合，然后在集合序列中通过双序列比对构造多序列比对提取最长调和序列，并通过构造元素集合和比对图归类重复体家族。这类算法存在的问题在于其本身基于高分相似对实现，所以从算法的宗旨上来说重复体序列的边界只是取最长的匹配边界，而且对于所有的片断构造聚类集合和连接图，导致算法计算量太大。算法 RepeatGluer[24]类似于 RECON 算法，将目标序列或者基因组与自身相比较来识别不同位置上面的局部比对结果，不同之处只是在归类重复体家族的方法。另一个值得关注的算法是 PILER[25]，算法以其敏感性为代价增加了其对各种重复体的特异性。

虽然现在存在各种各样的重复体识别算法，但是还没有一种能非常完善的解决重复体识别问题的算法。因此重复体识别问题的研究还在继续，需要更高效的算法来解决。

## 1.3 本文的研究工作

本文旨在对重复体识别中的一些关键问题进行研究与探讨，重点在于研究重复体识别算法中如何确定重复体序列的精确边界和提取具有高度代表性的重复体家族调和序列，并合理地平衡重复体序列长度和频率的关系，提高算法对于大规模基因组序列的运算速度，降低算法的运算复杂度。

本文首先介绍了重复体识别问题相关的理论知识。分类介绍了当前比较流行的重复体识别算法，具体分析了各类型算法的优缺点，说明了当前在重复体识别中存在的问题。

接着本文针对当前重复体识别算法存在的问题提出了基于高频率种子序列识别重复体的 RepeatSearcher 和 GSRSearcher 算法。RepeatSearcher 算法是基于典型数据库搜索算法 BLAST 的高效重复体识别算法，算法通过构造多序列比对合理的判断出了每条重复体序列的精确边界，并且通过相似度扩展得到了重复体家族的调和序列。算法采用限制空间的空位插入方法很好地提高了算法的精确性而且降低了算法的时间复杂度。GSRSearcher 算法继承了种子序列扩展的方法，结合概率统计模型提高了重复体序列边界判断的精确性，而且在限定的范围内采用采集得分位点的方式估计调和序列扩展的候选碱基，不但使算法敏感性更高，而且运行速度更快。

最后本文在真实的哺乳动物基因组序列中测试了这两种算法，并拿实验结果和当前流行的算法 RECON 以及重复体数据库 RepBase[26]中的家族序列做了比较分析，验证了这两个算法的高效性。在文章的最后提出了算法可以改进的地方和下一步的研究方向。

## 1.4 本文各章节安排

本文详细而全面的介绍了生物信息学中重复体识别问题的基本概念，针对当前国际上流行的重复体识别算法进行了细致分析和深入研究，并且提出了一种新的重复体识别算法。

论文内容具体安排如下：

第一章 主要介绍重复体识别问题的背景知识，重复体识别的意义和研究现状，最后对本文所做工作进行了简要的介绍。

第二章 主要介绍重复体识别中涉及到的一些基本概念和问题，包括：重复体识别问题的形式化描述，重复体识别中涉及到的局部序列比对，替换矩阵，评价标准等。然后对重复体识别问题进行了分类，列出了每类算法的优缺点，并针对

每类算法举例说明了各类算法的特点和存在问题。

第三章 在对数据库搜索算法 BLAST[27]分析的基础上，提出了基于 BLAST 算法具有精确边界的重复体识别算法。然后和当前比较流行的算法在真实基因组序列上的测试结果作了比较分析。

第四章 根据第三章提出的基于种子序列扩展识别重复体序列的方法，结合 Gibbs 采样算法[28][29]的统计理论，进一步研究了种子序列的扩展方法，提出了基于概率统计的重复体识别算法。算法最后通过对人类 X 染色体，大鼠 X 染色体，小鼠 X 染色体等 12 种哺乳动物基因组序列进行了测试，并和当前算法的识别结果作了比较分析。

第五章 对整篇文章进行了总结，分析了方法中的不足，并指出了今后的研究方向。

# 第二章 生物信息学重复体识别

## 2.1 重复体识别概述

重复体识别是生物信息学中基因组分析最常见的问题之一，通过重复体识别可以将基因组中相似的重复体序列归类到各自的家族中。重复体识别主要有两种方法，一种是基本的人工识别方法，再一种是基于算法的识别方法。重复体识别的理论基础来源于生物进化学说，生物学中许多证据表明：相异的 DNA 序列或者蛋白质序列很可能源自于同一祖先，在进化过程中经过序列内残基取代，残基或序列片断的缺失，以及序列重组等遗传变异过程演化。通过重复体识别，可以很明显的看出这些重复体家族序列在进化过程中非常保守，而且也表明了这些序列的结构和功能在进化中对物种的决定起了非常重要的作用。因此，重复体识别可以用在基因组家族的分析，进化树的构建等方面的研究。

重复体识别需要判断多个序列片断之间是否具有足够的相似性，以此来判别这些序列片断是否属于同一个家族，也就是所谓的同源。在这里需要区分一下相似性和同源性的概念，两者虽然在某种程度上具有一致性，但它们是完全不同的两个概念。相似性是指一种很直接的数量关系，比如部分相同或相似的百分比或其它一些合适的度量，而同源性是指从一些数据中推导出的两个基因在进化过程中曾经具有共同祖先，属于质的判断。基因之间要么同源，要么不同源，而不是象相似性那样具有大或小的数量关系。

研究重复体识别问题很大程度上是为了研究基因组的进化问题，几乎所有的重复体识别方法都希望能精确的建立重复体家族数据库。每一个重复体家族中的重复体序列都来源于同一个祖先，但是并不能得知这个祖先序列的具体结构，除非能从古代化石中得到这个家族的序列，但是这种概率微乎其微，所以算法必须从当前的现存物种中探求真相。从祖先序列到现在的家族序列发生了各种变化，比如基因的取代，插入和删除，而这些重复体家族序列可以通过相互比对产生一个家族调和序列，在一定程度上调和序列代表了这个家族的祖先序列。家族中的重复体序列和家族的调和序列相互比对时可以明显地表明每个位置上的碱基在进化过程中所发生的变化。下面是一组重复体序列和其调和序列的例子。

重复体序列： S1 = CGGATATACCGG,
S2 = CGGTGATAGCGG
S3 = CGGTACTAACGG

S4 = CGGCGGTAACGG

S5 = CGGCCCTAACGG

---

调和序列： Q = CGGNGATAACGG

序列 S1 到 S5 表示一个重复体家族的 5 个序列，这 5 个序列通过比对得到和这 5 个序列都很相似的调和序列 Q，这里 Q 可以作为这个重复体家族的代表序列。红色字体表示这个位置上的碱基在进化过程中发生了变异，N 表示这个位置的碱基是不确定的。

### 2.1.1 重复体识别问题

根据上面对重复体识别问题的概述，可以将重复体识别问题描述为如下六元组：

$$RI = (\Sigma,\ S,\ O,\ R,\ [Q,M],\ F)$$

当取为DNA序列时Σ={A, T, C, G, -}，而当是蛋白质序列时Σ={ A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y, -}，"-"表示空位符号。$S$ 是给定的基因组序列，$S = \{S_i \mid i=1,2,\ldots,n\}$，$S_i = (c_{i1}, c_{i2}, \ldots, c_{il})$，其中 $l$ 为序列的长度，$c_{ij}$ 为序列 $s_i$ 中的第 $j$ 个残基。$O$ 为在判定重复体时所做的操作集合，$O$ = {insert_gap, delete_gap}，包括一般的插入和删除操作。$R$ 表示识别出来的重复体结果，$R = \{R_i \mid i=1,2,\ldots,m\}$，$R_i$ 代表给定序列中所有的重复体家族，$R_i = \{r_{ij} \mid j=1,2,\ldots,k\}$，$r_{ij}$ 代表重复体家族 $R_i$ 中第 j 个序列。这是算法需要输出的主要结果。第 5 个元素可以为 $Q$ 或者 $M$，$Q$ 表示家族序列中的调和序列，$M$ 表示使用模体(motif)表示的家族序列。最后一个元素 $F$ 为重复体识别的具体方法，是重复体识别中的核心问题，也是本文涉及的重点。

重复体识别问题总的来说需要解决三个难点问题：首先基因组中包含重复体序列的位置预先是未知的，再者重复体序列的边界是未知的，最后许多重复体序列因为变异只剩下整条重复体的一部分。所以重复体识别过程也就是解决这三个问题的过程。

空位处理是针对序列进化过程中可能发生的插入和缺失而设计的。插入和缺失可能只涉及 1 个或 2 个残基，也可能是整个功能域。引入空位不仅要考虑到空位总数，还必须考虑连续空位的数量。所以必须对引入空位做适当的处理，故引入空位罚分的概念。

## 2.1.2 局部序列比对及其相关问题

序列比对的数学模型大体可以分为两类，一类从全长序列出发，考虑序列的整体相似性，即全局比对；另一类考虑序列部分区域的相似性，即局部比对。局部相似性比对的生物学基础是基因组序列包含许多短的功能序列和蛋白质功能位点，它们往往是由较短的序列片段组成的，这些位置的序列具有相当大的保守性，尽管在序列的其它部位可能有插入、删除或突变。此时，局部相似性比对往往比整体比对具有更高的灵敏度，其结果更具生物学意义。根据重复体在基因组的功能上的特点和其保守性，局部序列比对方法可以很好的识别出这些保守的重复体序列，所以在当前的重复体识别算法中，大部分都依赖于局部序列比对方法来判断重复体的相似性。

局部序列比对针对的是基因组中序列片断之间的比对，可以是双序列局部比对，也可以是多序列局部比对。双序列比对主要以数据库搜索算法 BLAST 为代表，多序列局部比对暂时还没有好的方法。不管是双序列比对还是多序列比对都涉及到替换矩阵和空位罚分这两个重要概念，下面就简要介绍这两个方面。

(1)替换矩阵：

在序列比对中，算法可以用替换矩阵来增强弱势比对的敏感性。很显然，在相关 DNA 序列或蛋白质序列之间，某些碱基或者氨基酸可以很容易地相互取代而不用改变它们的生理生化性质。在计算比对分值时，相同的碱基或氨基酸打分会高于替代的碱基或氨基酸，而保守的替代打分高于非保守的替代打分。换句话说，就是设计了一系列的分值，而且，在比对非常相近的序列以及差异极大的序列时会设计出不同系统的分值。考虑到这些因素，使用替换矩阵会极为有利，在这个矩阵中，任何碱基或氨基酸配对的分值会一目了然。

比如下面 2 条氨基酸序列的比对情况，如果将各残基按相同的统计计数处理，则 2 种比对(a 和 b)的得分将是相等的(9 个残基中的 5 个匹配)：

```
(a)  T TYGAPPWCS          (b)  TTYGAPPWCS
      TGYAPPPWS                 TGYAPPPWS
      *   ***   *               *  *  ***
```

但是比对 a 是一些普通的残基(A,P,S 和 T)保持一致，而比对 b 则是一些相对稀有残基(W-色氨酸，Y-酪氨酸)相一致。所以需要一个更为科学的方法来反映匹配残基间的生物学和化学关系。

在比对中，C-C 匹配相对比 S-S 匹配更重要些，因为半胱氨酸(C)是具有非常特殊性质的相对稀有氨基酸，而丝氨酸(S)则相对普通。同样 D-E 匹配应取正值，因为这两个氨基酸具有相同的化学性质，在两条比对的蛋白质序列中能起到相同

的功用。但是 V-K 匹配则应被罚分，因为这两个氨基酸毫不相似，不可能在两条序列中起到一样的作用。

替换矩阵包括了在比对中各种匹配方式如何赋分的信息，故替换矩阵又常被称为打分矩阵(scoring matrices)。

用于 DNA 序列比对的替换矩阵相对比较直观。以下是一个常被使用的替换矩阵：

表 2-1：一个简单的 DNA 替代矩阵

| | A | C | G | T |
|---|---|---|---|---|
| A | 0.9 | -0.1 | -0.1 | -0.1 |
| C | -0.1 | 0.9 | -0.1 | -0.1 |
| G | -0.1 | -0.1 | 0.9 | -0.1 |
| T | -0.1 | -0.1 | -0.1 | 0.9 |

矩阵中每一个匹配的碱基对均得 0.9 分，每个不匹配的碱基被罚 0.1 分，这样下面一个比对的得分应为 4.3(5*0.9+2*(-0.1)):

GCGCCTC
GCGGGTC
* * *  * *

用于蛋白质的替换矩阵要复杂一些，因为没有一个替换矩阵可以适用各种情况。第一个广泛使用的最优矩阵建立在进化的点突变模型上(PAM[30])。一个 PAM 就是一个进化的变异单位即 1%的氨基酸改变，这并不意味着经过 100 次 PAM 后，每个氨基酸都发生变化，因为其中一些位置可能会经过多次改变，甚至可能变回到原先的氨基酸，因此另外一些氨基酸可能不发生改变。如果这些变化是随机的，那么每一种可能的取代频率仅仅取决于不同氨基酸出现的频率(称为背景频率)。然而，在相关蛋白质中，已经发现的替代频率（称为目标频率）大大地倾向于那些不影响蛋白质功能的替代，换句话说，这些点突变已经被进化所接受。Dayhoff 同合作者们第一次使用了 log-odd 处理，在这种处理中，矩阵中的替代分值同目标频率与背景频率的比值的自然对数成比例。为了评估目标频率，人们用非常相近的序列（比对时不需要替代矩阵）来收集对应于一个 PAM 的突变频率，然后将数据外推至 250 个 PAM。虽然 Dayhoff 等人只发表了 PAM250，但潜在的突变数据可以外推至其它 PAM 值，产生一组矩阵，在比较差异极大的序列时，通常在较高的 PAM 值处得到最佳结果，比如在 PAM200 到 250 之间，较低值的 PAM 矩阵一般使用于高度相似的序列，其关系如图 2-1 所示。BLOSUM[31]替换矩阵和 PAM 矩阵的构建方法基本上相似。

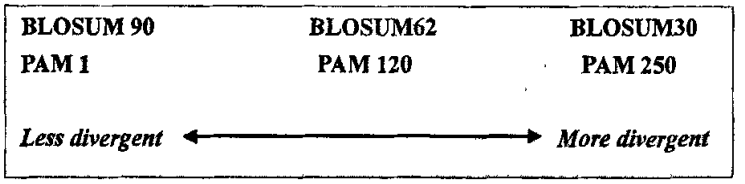



图 2-1: 替换矩阵选择和序列相似度的关系

(2)空位罚分[32]

所谓空位罚分，就是指对于在序列比对过程中引入的每个空位都要按照某种规则扣除一定的分值作为惩罚，从而尽可能的逼近现实生物进化规律。

假定，初始输入序列为 $s_1$ 和 $s_2$，比对后的输出序列为 $s_{1O}$ 和 $s_{2O}$，且比对后的序列长度为 $L$，常用的空位罚分规则有三种：

**a.常量空位罚分**

这是一种最简单的罚分策略，即对插入比对序列的每个空位都赋予一个常数量的罚分 $W_g$，整个比对的空位罚分就是插入的全部空位数 $R_g$ 的罚分之和，即 $W_g \times R_g$。

这种罚分策略的优点是简单，不会增加额外的时间复杂度。但是其缺点也是显而易见的。因为在实际的生物分子进化中，基因中不同位点的突变概率是不同的，对每个空位使用相同的常量罚分显然是不能准确刻画序列比对的生物意义。

**b.恒定空位罚分**

这种罚分策略是从整体上处理插入的空位，即序列中插入的相连空格被看作一个整体进行罚分。这样每个空位的罚分 $W_g$ 都是与其长度无关的。具体计算如下：

- 用 $\sigma$ 表示匹配或者不匹配的得分值，$\forall x \sigma(x,-) = \sigma(-,x) = 0$；
- 则整个比对的得分计算公式为公式(2-1)：

$$\sum_{i=1}^{L} \sigma(s_{1O}[i], s_{2O}[i]) + W_g \times gaps \qquad \text{公式(2-1)}$$

其中 gaps 表示空位的数目。

恒定空位罚分虽然可以避免常量空位罚分将罚分单纯依赖于空位长度的缺点，但是却忽略了空位长度对比对结果的影响，可能造成插入过多空格导致割裂整个比对片段。因此更合适的罚分策略应当同时考虑引入空位长度的影响，但又要避免单纯的长度依赖的罚分。

**c. 仿射空位罚分**

仿射型的空位罚分策略则把整个空位罚分分成两个部分进行罚分：一部分称为开放空位罚分(gap opening penalty)，另一部分称为扩展空位罚分(gap extension penalty)。对于长度为 $q$ 的空位，$W_g$ 为开放空位罚分值，$W_s$ 为扩展空位罚分值，

则总的罚分值 $W$ 的计算公式为： $W = W_g + q \times W_s$，这样整个比对的得分计算公式为公式 2-2：

$$\sum_{i=1}^{L} \sigma(s_{1O}[i], s_{2O}[i]) + W_g \times gaps + W_s \times spaces \qquad \text{公式(2-2)}$$

其中 gaps 表示空位的数目，spaces 表示空格的数目。

仿射型的空位罚分策略能够比较准确的反映真实的生物基因变异规律，即基因产生插入和变异是很罕见的，但当它们一旦发生，就会影响到一系列附近的残基。

### 2.1.3 调和序列

调和序列(consensus sequence)是重复体家族的一种代表序列。一个家族的调和序列，就是一个捕获了家族多数成员的共同特征的序列。通过调和序列可以很方便的确定一个基因家族的特征，也可以很方便的总结出一个序列家族中的公共元素，所以找出$N$个序列的调和序列就变得很重要。后来找调和序列问题被证明是一个NP完全问题[33] (Day and McMorris. , 1993)。

目前，专门用于描述找调和序列的文献非常的少，Gusfield在1997年提供了一个关于这个问题的简明但十分有意义的讨论[34]。在这里简单的提及二种存在的算法，第一种是众所周知的(不属于任何个人或团体)的算法，即是首先对这一序列族进行多序列比对，然后在比对结果的每一列根据出现的概率权值来挑出共同的元素。第二种方法与多序列比对中的渐进比对问题[35]很相似，然而它只是在序列族中选一条序列作为侯选调和序列。

本文第三章和第四章的重复体识别算法给出了调和序列的一种求法，通过识别基因组中重复体的过程求得调和序列。算法通过逐步向两边扩展种子序列得到调和序列，并结合多序列比对信息，判断调和序列下一步扩展的残基，根据目标函数值的变化判定扩展过程是否结束。这个过程的数学描述如下：

假设$A$为具有有限个字符的字母表，$|A|$为字母表中不同字符的个数；假设$\Sigma$为由字母表中的字符组成的所有序列的全集，$s = \{s_1, s_2, \ldots, s_q\} \subset \Sigma$ 是一个$q \geq 2$的序列集，且序列集$s$就是输入序列集； $n_i$为每一序列$s_i$ $(i = 1, \ldots, q)$的长度。本文提出来的算法所描述的基本思想就是在空间$\Sigma$中通过逐步扩展调和序列最终生成调和序列$Q$，也就是在$\Sigma$中的重复体序列中找一条与相应重复体家族序列集具有最大比对得分之和的序列。

定义：假设$\Sigma$是由字母表A中的字符组成的所有序列的全集，给定一个$\Sigma$中的序列族$s = \{s_1, s_2, \ldots, s_q\}$，一条任意序列$s'$(这一序列不一定是$s$中的)和一双序列计分函数$SCORE$：$\Sigma * \Sigma \rightarrow R$， $s'$与$S$的得分总和为：

$$E(s') = \sum_{i=1}^{q} SCORE(s_i, s') \qquad \text{公式(2-3)}$$

如果，对于所有$s \in \Sigma$有E($s'$)≥E($s$)，则序列为$s'$为输入序列族s的一致性调和序列$Q$。

### 2.1.4 重复体识别结果的评判标准

当前在重复体识别结果评判上还没有一个统一的标准。已有的重复体识别算法大部分是把自己识别出来的结果通过 RepeatMasker 算法在相应的重复体数据库（比如 RepBase）中进行匹配，根据识别出来重复体家族的数目和可以匹配出来的重复体序列多少来判定算法的效果。之所以这么做是因为 RepeatMasker 算法匹配出来的结果被誉为“黄金标准”(Bao and Eddy，2002)。

调和序列或者模体表示一个重复体家族，所以也可以通过分析调和序列或者模体与重复体家族序来列判别识别算法的效果。这里列出 2 种常用方法：

(1) 相对熵(Relative entropy, *RE*)[36]

设调和序列或者模体序列的长度为 k，RE 值可定义为下：

$$RE = \sum_{j=1}^{k} \sum_{c \in E} p_{c,j} \log_2 \frac{p_{c,j}}{b_c} \qquad \text{公式(2-4)}$$

$p_{c,j}$代表碱基 c 在位置 j 在重复体序列中出现的频率，$b_c$代表碱基 c 的背景频率。*RE* 的值越大表明算法敏感度越高。

(2) 敏感性测量[37]

假设真阳性(true positives，*TP*)表示属于识别出来的重复体家族而且和家族调和序列或者模体匹配的序列，真阴性(true negatives，*TN*)表示不属于识别出来的重复体家族而且不和家族的调和序列或者模体匹配的序列，假阴性(false negatives，*FN*)表示属于识别出来的重复体家族但是不和家族调和序列或者模体匹配的序列，假阳性(false positives，*FP*)表示不属于重复体家族但是和家族调和序列或者模体匹配的序列。

敏感性可以表示为： $TP / (TP + FN)$

特异性可以表示为： $TN / (TN + FP)$

综合起来两者可以表示为：

$$\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FN)(TN+FP)}} \qquad \text{公式(2-5)}$$

## 2.2 重复体识别算法

目前，进行重复体识别的算法很多，这些算法主要分为三类：基于已知重复体数据库的重复体识别算法，基于序列比对的重复体识别算法和其它的启发式重复体识别算法。这些算法各自有各自的有特点。下面给出几种具有代表性的算法来说明当前的重复体识别算法。

### 2.2.1 基于已知重复体数据库的重复体识别算法

基于已知重复体数据库的重复体识别算法是指在已经建立好的重复体数据库中寻找与请求被识别序列的相似序列，以此来判别请求被识别序列是属于哪个重复体家族。这种方法的识别结果是非常精确的，一般作为其它算法参考值。其中算法 RepeatMasker 为代表算法。

**RepeatMasker 算法：**

RepeatMasker 是覆盖分散的重复体序列和低复杂度 DNA 序列的程序。程序的输出为一些已被标识的重复体序列。当前人类基因组 DNA 序列平均 50%已经被 RepeatMasker 所识别。RepeatMasker 程序通过 cross_match 或者 Smith-Waterman 作为它的数据库比较策略。这里主要介绍一下 Smith-Waterman 序列匹配策略。

Smith-Waterman 描述了一种查找具有最高相似性片断的算法。对于序列 $A=(a_1,a_2,\ldots,a_m)$和 $B=(b_1,b_2,\ldots,b_n)$，$H_{ij}$ 被定义为以 $a_i$ 和 $b_j$ 碱基对结束的片段(亚序列)的相似性值。算法通过递推关系来确定 $H$ 值，$H$ 的初始值为：

$$H_{i0}=0,\quad 0\leqslant i\leqslant n,\qquad H_{0j}=0,\quad 0\leqslant j\leqslant m$$

相似性计算中包括 2 个统计量：碱基对（序列因子）$a_i,b_j$ 的相似性值 $S(a_i,b_j)$ 和空位权重 $w_k=v+uk$(k 为空位长度)。Smith-Waterman 算法可以给出 2 条序列的最大似然度。以 $a_i,b_j$ 碱基对结束的片断可以由以 $a_{i-1}$ 和 $b_{j-1}$ 结束片段增加碱基（因子）来获得，或者 $a_i$ 可以删除 $k$ 长度的碱基片断，$b_j$ 可删除 $l$ 长度碱基片断。具体算法如下：

$$P_{i,j}=\max(H_{i-1,j}-w_1,P_{i-1,j}-u)$$

$$Q_{i,j}=\max(H_{i,j-1}-w_1,P_{i,j-1}-u)$$

$$H_{i,j}=\max\begin{cases}H_{i-1,j-1}+S(a_i,b_j)\\ P_{ij}=\max_{1\le k\le i}(H_{i-k,j}-w_k)\\ Q_{ij}=\max_{1\le l\le j}(H_{i,j-1}-w_l)\\ 0\end{cases},\quad (1\le i\le m,1\le j\le n)$$

其中 $P_{0,0}=P_{0,j}=Q_{0,0}=Q_{i,0}=0$ 公式(2-6)

该算法可以确保具有最大 $H_{ij}$ 值的序列片断是最优的。从($a_i$,$b_j$)为起点，向后追踪 $H_{ij}$ 矩阵，直到到达某一负值。对于具有最大似然度片断以外部分的差异性不会影响到该片断的 $H$ 值。

## 2.2.2 基于序列比对的重复体识别算法

当前大部分重复体识别算法都是基于高分相似对，比如通过WU-BLAST(Gish,http://blast.wustl.edu)或者 REPuter(Kurtz and Schleiermacher,1999; Kurtz et al., 2000)算法用双序列比对方法产生重复体相似对集合来识别重复体家族。再一种改进算法是通过构造多序列比对识别重复体，典型代表为 RECON 算法。

这里主要介绍基于双序列比对产生的高分相似对(high score pairwist，HSP)链接重复体家族的算法 Single Linkage Clustering Algorithm[38]和通过 HSP 提取多序列比对信息来链接重复家族的算法 RECON。

**(1) Single Linkage Clustering Algorithm**

在讲算法之前，这里说明几个相关名词。元素(element)是指一个重复体 $S_n(s_k,e_k)$ 的独立重复片断，映像(image)是指参与比对的所有子序列，同位体(syntopic)指同一个元素的两个映像。元素是需要推导的生物序列实体，映像是从基因组序列 $S_n$ 中通过比对观察出来的子序列。因为重复体是重复体出现的，所以一个元素可以对应多个映像。

算法以 BLAST 等算法产生的高分相似对为基础，然后通过单链算法链接这些高分相似对成重复体元素，并归类到相应的家族中。BLAST 算法在第三章中提到，这里不再列出。

Single Linkage Clustering Algorithm 过程：

1. 从基因组序列{$S_n$}中得到高分相似对序列。
2. 从得到的高分相似对或映像中定义元素集合{ $S_n(s_k,e_k)$}：
   a. 构造图 G(V,E)，V 代表所有的映像，E 表示映像之间的同位体关系。如果两个映像的重叠度超过一定阈值则认为它们是同位体。
   b. 寻找图 G 中所有被连接的部分[39]。
   c. 对于每一对连接的映像定义一个元素 $S_n(s_k,e_k)$作为最短的覆盖所有映像片断。
3. 根据元素序列的相似性将元素序列归类到相应重复体家族中：
   a. 构造图 H(V',E')，V'表示所有的元素，E'表示两个元素相似(如果两个元素在步骤 1 中构成了比对则将这两个元素连接起来)。
   b. 寻找图 H 中所有相连的部分。

c. 对于相邻接的元素定义一个家族，其包括所有这些相连的元素。

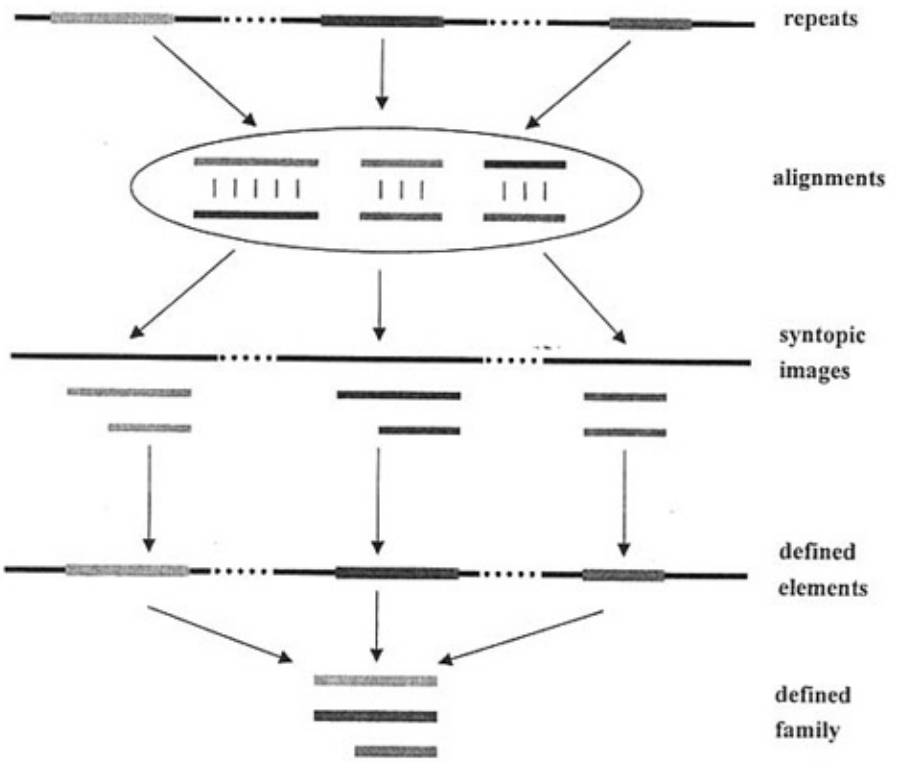



图 2-1：Single Linkage Clustering Algorithms 过程

如果所有的重复体序列都是全长的，而且没有重叠那么使用 Single Linkage Clustering Algorithms 算法可以得到非常好的结果，但是如果元素有重叠，那么仅仅依靠相似对的算法就很难区分重复体的边界。

**(2) RECON 算法**

RECON 算法对单链聚类方法作了改进，通过高分相似对抽取多序列比对信息而很好的处理了元素重叠的问题。通过设定阈值来衡量两个元素之间的保守区域和两个元素序列的比值，以此判断两个元素是否属于同一个重复体家族。

RECON 算法主过程：

1. 从基因组序列$\{S_n\}$中得到高分相似对序列。
2. 从得到的高分相似对序列中定义元素。
   a. 使用 Single Linkage Clustering Algorithms 算法第 2 步定义元素。
   b. 在映像完成端点选择，元素重新评估和刷新过程后，每一个被定义

的元素需要被重新计算。

c. 如果一个定义的元素被认为是组合元素而且被分割，那么与这个元素比对的所有元素都必须被重新评估，这个过程直到所有定义的元素稳定为止。

3. 根据序列的相似性归类元素到相应的重复体家族中。

a. 通过家族关系决定过程和边评估重建过程决定元素和家族的关系并将其转化为图 H(V',E')，如图 2-2。

b. 寻找图 H 中实线相连的元素，然后将这些元素归为一个家族。

RECON 算法中包括了 4 个优化过程：映射结束点选择规则，元素重新评估和刷新过程，家族关系确定过程和边重估构建家族图过程。下面是这 4 个过程基本步骤：

1. 映射结束点选择规则：

这个规则通过考虑两个元素之间比对或非比对序列的长度和位置来过滤被丢失的映像，具体如下：

a. 针对每一对组成比对的元素寻找所有最大的比对组，在这个比对组中所有的比对都是这两个元素组成的全局比对的一部分(不一定最优)。这个过程通过端点代表比对，边表示两个端点上面的比对是这两个元素全局比对的一部分。

b. 对上面找到的每一个组根据它们的相似度对其排序，排除那些序列在比对之外的组或者在组中两个相邻的比对之间而到两者之间的距离都小于阈值 cutoff 的组。如果没有被排除，则规定这个组的得分为组内所有比对分值之和。阈值 cutoff 需满足如下条件：长度小于这个阈值 cutoff 的序列都可以用成对比对工具通过比对的随机扩展产生。经过上面过程后，如果剩下的组多余 1 个，则取出分值最高的一个组，而其它的组都可以抛弃掉。剩下这个组中映像的端点被采集出来并作下一步分析。

2. 元素重新评估和刷新过程：

在这里算法通过评估上面规则所产生的映像终端集合来刷新元素的定义。

a. 选择一个长度阈值 cutoff，使其满足长度小于这个阈值的序列都可以用成对比对工具随机扩展产生。

b. 在产生的元素上面滑动一个长度为 cutoff 的窗口，在每个滑动窗口中，如果最左边的端点还没有被归类则将其加入到一个聚集中，如果端点和这个聚集中的其它成员距离小于一定值则将其加入到这个聚集中，如果已经没有可以加入到当前聚集中的端点则开始一个新的聚

集直到窗口中所有端点被归类。

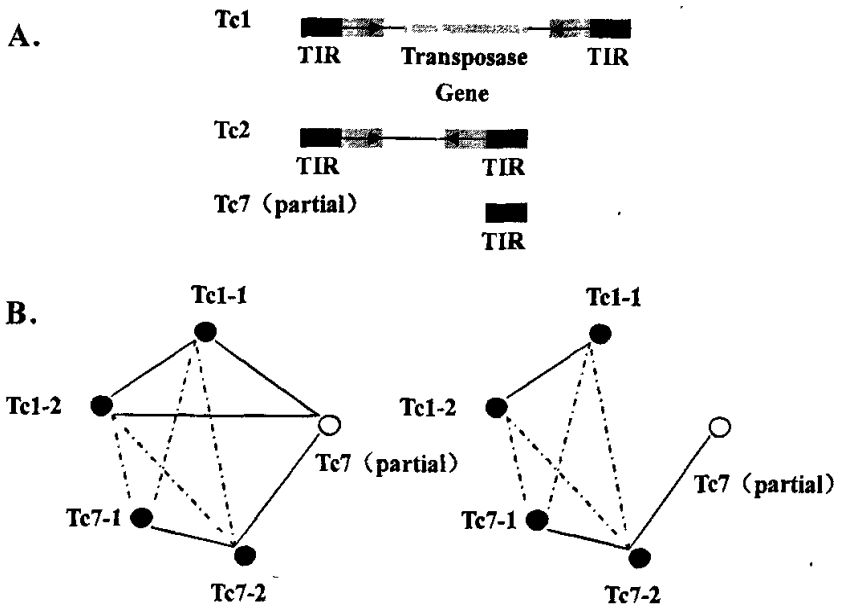



图 2-2：构建重复体家族过程

c. 针对上面找到的每一个聚类，设 n 表示聚类中终端的个数，c 表示 n 个终端的位置平均值，m 表示给定元素扩展 c 个位置的映像数目。如果 n/m 大于特定阈值则 c 被认为是一个集合点。
d. 如果没有找到集合点则使用原始的元素定义。
e. 如果找到了集合点则在这个集合点分割元素和它的比对，丢弃原来定义的元素和分割以后长度小于阈值 cutoff 的新元素，然后重新定义元素。这个过程刷新了已经定义的元素。
f. 如果一个原始的元素被分割为多个元素则初始的元素定义为一个组。

3. 家族关系确定过程：

这个过程确定两个参与比对的元素是否属于同一个家族，并且比较比对序列和元素的长度。

a. 针对给定的一对组成比对的元素，寻找均是以这两个元素进行全局比对且为集合一部分的最大比对组(不一定最优)。
b. 上面找到的每一个组的总长度为这个组中所有比对的长度之和。这些组之间的最大长度视为这两个元素的比对长度。

c. 如果这两个元素的最大长度大于特定的长度则这两个元素属于同一个重复体家族，否则不属于。

4. 通过边重估构建家族图过程：

a. 每一个定义的元素表示图中一个端点。

b. 如果两个元素通过家族关系确定过程判断出来属于同一个家族，则通过实线将这两个元素连接起来，如果两个元素做了比对但是不属于同一个家族则用虚线将两者连接起来，如果两个元素没有比对则之间不用连接。

c. 对于每一个端点 v，它的实线连接的边需要重新评估(如图 2-2)。假设 N(v)表示直接和端点 v 通过实线连接的端点集合，如果 N(v)中节点之间有虚线连接的结点，对于 N(v)中的所有任意节点 v'，如果 v 和 v' 比对值小于一定的阈值或者两者不是直接相关的元素则删除 v 和 v' 之间的边。通过这个过程，图中每一个节点都得到了更新。

d. 删除图中所有的虚线边。

上面几个过程从集合的角度通过集合内部所有映像的比对信息重新定义了元素的标准，使得识别出来的元素更加准确。这个算法已经被广泛采用来构建重复体数据库。

### 2.2.3 其它重复体识别算法

除了基于序列相似对重复体识别算法以外，还有许多其它的识别算法，这些算法主要以种子（seed）序列为起点并结合种子的频率来识别重复体。种子序列是通过扫描基因组序列找到的重复度非常高的短序列，可以理解它们在基因组中是非常保守的序列。结合种子长度和频率的方法识别重复体是一个新的方向，这里主要介绍一个具有代表性的算法[40]：

假设（$A_1$，$A_2$，…，$A_m$）是基因组序列 S 中序列 A 出现的位置列表，B 是 A 的子串，（$B_1$，$B_2$，…，$B_k$）表示 B 在 S 中出现的位置列表。因为 B 是 A 的子串，所以每一个 A 都有一个子串 B，但是 B 并一定都在 A 中，有可能在 A 之外其它的地方。如果 B 在 A 中出现的位置是 s，则可以得出 A[s+1，s+|B|] = B。这个是算法的基本贴入点。接下来算法提出了几个基本的定义：

定义 1：一个子串 S 称为非平凡子串，当且仅当 S 的长度|S|大于等于特定的阈值 $l$。

定义 2：A 和 B 为 S 的子串，如果 k 等于 m 而且 $B_i$ 在 $A_i$ 中的偏移都是 s(i=1，2，…，m)，则称 B 是 A 的子重复体。

定义 3：A 是一个最长的非平凡子串，如果 A 是一个 S 中相等基本重复体必

须满足如下条件：A 在基因组序列 S 中出现 $f_m$ 次，而且每一个 A 中的非平凡子串是 A 的一个子重复体。

定义 4：假设最短的有序编辑序列（$e_1$，$e_2$，…，$e_d$）将 A 转换为 A'，对（$e_i$，$e_{i+1}$，…，$e_j$），$1 \leqslant i < j \leqslant d$，转换 A 的子串 B 为 A'的子串 B'，则称 B'是 A 引起的 B 映射。

这里编辑距离[41]是指将 A 通过插入，删除和取代操作编辑为 A'的最少操作次数。如果编辑距离 $D(A,A')/|A| \leqslant \varepsilon$，则称 A 和 A'的匹配度为 ε。

定义 5：如果子串 A 和子串 B 满足如下 3 个条件，则称 B 是 A 的子重复体串。

(1) B 是 A 的一个子串。

(2) 每一个由 A 生成的 B 的映射都是 B 的 ε 拷贝。

(3) S 中 B 的 ε 拷贝的数目等于 A 生成的 B 的映射的数目。

* 子重复体串不应该被认为是重复体串，因为它不是独立存在的。

推论：一个至少重复 2 次的非平凡子串 A 是一个相等重复体串，当且仅当最长非平凡子串 A 的所有 l-mer 子序列的频率和 A 的频率相等。

根据该推论，算法只需要比较子串的频率就可以抛弃大部分非候选子串。鉴于一般的基因组序列规模是非常大，所以为了提高算法的效率，l-mer 子串的存储必须使用高效的数据结构，比如后缀树或者哈希表。

定义 6：如果子串 A 在基因组序列 S 中至少存在 $f_m$ 个 ε 拷贝而且 A 的每一个非平凡子序列是 A 的一个子重复体串，那么 S 中最大长度的非平凡子串 A 是似然重复体序列。

下面给出两个例子分别说明相等重复体识别和似然重复体问题。输入基因组序列 S = cACGTGagACGTGgcaACGTG。假设种子长度为 2，则种子 AC,CG,GT,TG 的出现频率都是 3，AC 的匹配列表为(2，9，17)，CG 为(3，10，18)，GT 为(4，11，19)，TG 为(5，12，20)。它们的匹配列表之间的偏移都相等，而且根据定义每一个种子都是子串 ACGTG 的子重复体，所以 ACGTG 是相等重复体。

当输入基因组序列 S = cACGTGagACGAGgcaACGTG 时，如果 ε 设定为 2，因为子串 ACGTG 和 ACGAG 之间的编辑距离为 1，所以 ACGAG 算是一个满足条件的 ε 拷贝，结合各个种子序列的匹配列表可以得出 ACGTG 和 ACGAG 都是同一个家族的重复体序列。

相等重复体串是指完全相同的重复体片断，通过推论 5 给出了相等重复体的推算方法。似然重复体是指那些重复体之间相似度在 ε 以上的重复体序列，定义 6 结合前面的定义也给出了相应的定义。这个算法通过简单的种子频率扫描和频率对比实现了重复体的识别，是一个简单有效的算法。

## 2.3 本章小结

本章首先对重复体识别问题进行了形式化描述，然后详细介绍了重复体识别涉及到的一些相关问题，如：局部序列比对，替换矩阵，空位罚分，重复体识别结果评价标准等。

本章接着对现有重复体识别算法进行了分类。对其中应用最广泛的基于局部序列比对的代表算法 Single Linkage Clustering Algorithm 和 RECON 算法作了详细的介绍，并对比了两者的优缺点。然后对近来出现的一种基于种子频率寻找重复体的识别算法作了重点阐述。

通过分析，可以看出当前的重复体识别算法均有其优缺点，本文就是在研究现有重复体识别算法的基础上，在下面两章提出了一种新的重复体识别算法：基于 BLAST 的重复体识别算法，并通过概率统计分析对其作了过程和结果的优化。

# 第三章 基于 BLAST 的重复体识别算法

通过基因组中的种子(Seed)序列来识别重复体家族序列的方法已经成为近几年重复体识别领域里面一个重要突破点。本文通过研究分析 BLAST 算法中利用种子序列来进行局部序列比对和逐步扩展比对序列的方法，将双序列局部比对扩展为多序列局部比对，并逐步扩展判别出来的调和序列和重复体家族中每一个重复体序列，判定每一个重复体序列的精确边界。

## 3.1 BLAST 算法简介

BLAST 的全称是 Basic local alignment search tool(局部相似性基本查询工具)。它是由美国国立生物技术信息中心（NCBI）开发的一个基于序列相似性的数据库搜索程序。Blast 是一个序列相似性搜索的程序包，其中包含了很多个独立的程序，这些程序是根据查询的对象和数据库的不同来定义的。Blast 算法的基本思想是通过产生数量更少的但质量更好的增强点来提高速度，也就是扫描序列中出现频率大于一定域值的种子片断来进行数据库搜索，这样不但准确率更高而且速度会更快。由于局部比对的限制条件，在大多数情况下比对被分解为若干个明显的 HSP(High score Pairs，两个比对值最高的序列片断)。HSP 值的大小说明了比对的两个序列相似的程度。Blast 算法是建立在严格的统计学基础之上的，它集中于发现具有较高的相似性的局部比对，但是基本的 Blast 算法不支持在序列中插入空位，Blast2[42]在 Blast 算法的基础上实现了空位的插入。

### 3.1.1 BLAST 算法过程

算法从一个输入 DNA 序列 $S_q$ 和比对对象数据库开始，先从数据库中取出一条序列 $S_i(1\leqslant i\leqslant n)$，然后把 $S_i$ 分解为长度 W 的片断 $S_{i,j}(1\leqslant j\leqslant|S_i|)$。在输入序列 $S_q$ 中寻找出和 W 片断 $S_{i,j}$ 完全相等的片断 $S_{q,k}$，记录下来匹配的位置 k(word hits)。然后将匹配的片断 $S_{q,k}$ 根据打分矩阵逐个和 $S_{i,j}$ 进行比对，并逐渐根据相似度把 $S_{q,k}$ 和 $S_{i,j}$ 向左右两边延伸，每延伸一个字节则计算一次局部比对的 HSP 值，记录下来最高分值位置，得到最大 HSP 值而且其值在一定范围内不再增加为止，这次比对结束，这个过程也是局部寻优过程。依次针对每一个 W 片断 $S_{i,j}(1\leqslant j\leqslant|S_i|)$ 进行上述局部寻优过程并得到最大 HSP 值点，这样就依次找到了所有共享相同片断的重复体序列，最后将这些重复体序列组成最高分值片断群。图 3-1 展示了

BLAST 算法的 3 个步骤。

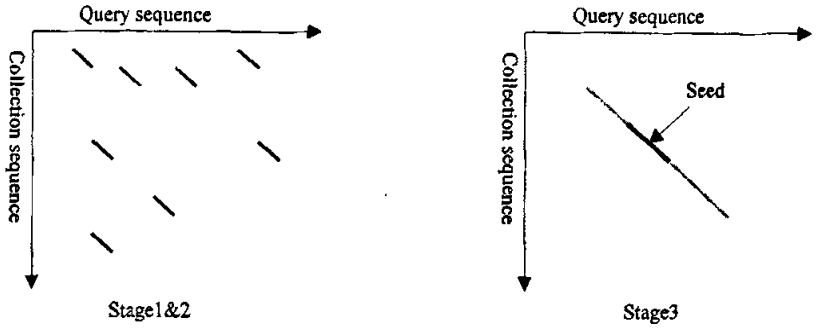



图 3-1：(Stage1&2)寻找短的高分命中序列　(Stage3)扩展短的高分命中序列

BLAST 算法的比对过程基于基本的双序列局部比对方法，在查询序列和集合序列之间寻找最长的相似对。算法通过设置比对分值来确定相似对的边界。基本 BLAST 算法不支持空位的插入，这样保证了算法有较高的运算速度，但却损失了结果的精度。

## 3.2 基于 BLAST 的 RepeatSearcher 重复体识别算法

### 3.2.1 算法的引入

RepeatSearcher 算法使用 BLAST 算法在图 3-1 中的 3 个主要步骤：分割一个基因组序列 S 为 l-mer 的种子序列(相同而且长度为 *l* 的高频率序列片断)，记录下来这些种子在序列 S 中的位置，然后比对具有相同种子的序列，并逐步向两边扩展直到得到最大比对分值。这样就可以计算出夹杂着一些伪重复体元素的序列。怎么精确的判断出每一条重复体序列的精确边界和找出家族的调和序列将是算法需要解决的问题。

针对重复体识别问题，RepeatSearcher 对 BLAST 基本算法进行了扩展，改双序列局部比对为多序列局部比对，在 BLAST 算法中加入空位罚分机制。鉴于局部比对问题的特点，算法限制了空位的插入范围，这样在加快了比对速度同时也避免了 BLAST 算法不支持空位带来的不精确性。空位限制的范围大小会影响到比对的速度，越小的范围比对速度越快。

## 3.2.2 算法的描述

### 3.2.2.1 目标函数的定义

首先定义多序列局部比对的目标函数。设 $S_1,...,S_n$ 表示 DNA 序列中包含相同 l-mer 种子(长度为 l 的高频率子串)的子序列，这些子序列都通过 l-mer 种子向左右做了相似性扩展。$S_1,...,S_n$ 的子串 $R_1,...,R_n$ 代表每一个具有精确边界的重复体序列。Q 表示调和序列，它使得公式 3-1 取最大值。这里 $A(Q;S_{1,...,}S_n)$是多序列匹配优先比对的目标函数，|Q|表示调和序列 Q 的长度，c 表示重复体要求的最小出现频率，μ表示匹配得分。由此可知 $R_1,...,R_n$ 是 $S_1,...,S_n$ 比对时目标函数 $A(Q;S_1,...,S_n)$ 取得最优值时的子串。如公式 3-1 所示，目标 $A(Q;S_1,...,S_n)$等于 Q 分别和 $S_1...S_n$ 比对得分函数最大值之和减去调节因子 $c\mu|Q|$。

$$A(Q;S_1,...,S_n)=\left[\sum_k \max\{s(Q,S_k),0\}\right]-c\mu|Q| \qquad \text{公式(3-1)}$$

在一个重复体家族中，重复体家族核心序列被这个家族中的重复体子串共享，这些重复体子串各自有各自的边界。算法通过目标函数 $A(Q;S_1,...,S_n)$来严格的确定每一个重复体子串的边界，当目标函数取最大值时，每一条重复体序列的当前计算位置即为这条序列的边界位置，也就是算法要寻找的重复体序列终点，这样就找到了重复体的精确边界。通过公式 3-1 可以看出计算目标函数的过程就是计算比对得分函数的过程，下一节将给出比对得分函数 $s(Q, S_k)$的计算方法。

### 3.2.2.2 比对得分函数计算

计算比对得分函数 $s(Q, S_k)$有 3 种计算方法，图 2 列出了几条 DNA 序列使用这 3 种方法的结果，红色粗体的序列片断表示重复体序列共享的片断。

(a) 包含部分重复体的序列集合

TAGCACCTTAGGGCGTCTCG**CAACGTCTGC**CCCACGAACGTTAATCAGTAA

GATTATCATGAAGCGCTTCG**CAACGTCTGC**AGCTGTCCAGACCGCTGTCA

TATATCCGGTAATCGCCCCG**CAACGTCTGC**TAACGGGCGTACGGTCGAAT

TGACCTGCTCAGGAGCCTTG**CAACGTCTGC**TCGCGGATGTGTATGCACGC

ATCCATGCTCGGTATGAATC**CAACGTCTGC**TCATGGACATCTCATGACGT

(b) 取最长相似对计算出的调和序列[43](家族中所有序列两两比对的最长片断)

AGCACCTTAGGGCGTCTCG**CAACGTCTGC**CCACGAACGTTAATCA

(c) 使用完全匹配比对计算出的调和序列(所有家族序列的公共片断)

CAACGTCTGC

(d) 使用匹配优先比对算法计算出的调和序列

AGGCGCCTCGCAACGTCTGCTCACGGACGT

图 3-2: 通过不同的比对策略得到的调和序列

匹配优先算法使得调和序列每次取 $A(Q;S_1,\ldots,S_n)$最大值时的元素。比对得分函数 $s(Q, S_k)$等于 $S_k$ 和 Q 的局部比对分数最大值。根据图 2 所示，通过匹配优先算法调和序列的精确边界被计算出来，而不只是简单的取比对的最大长度或最高出现频率的片断。所以 RepeatSearcher 算法使用匹配优先比对策略作为比对算法。定义匹配优先算法的比对得分函数为：

$$f(i,0)=\max(-\gamma i,-p),\quad f(0,j)=0$$

$$f(i,j)=\max\begin{cases}f(i-1,j-1)+\mu_{i,j}\\ f(i-1,j)-\gamma\\ f(i,j-1)-\gamma\\ -p\end{cases}$$

$$s(Q,S)=\max\begin{cases}f(i,j) & if(i=|Q|)\\ f(i,j)-p & if(i<|Q|)\end{cases}$$ 公式(3-2)

$\mu_{ij}$ 表示 $Q_i$ 和 $S_j$ 的比对得分，$\gamma$ 表示一个固定空位罚分值，$p$ 表示序列 $S_i$ 不完全匹配罚分值。

表 3-1 列出了一个使用这 3 种方法的实例结果。假设一个重复体总共有 1000 个重复片断，其中有 500 个片断字节长度为 100，450 个片断长度为 150，50 个片断长度为 200。根据这些信息可以知道调和序列的长度候选值为 100，150 和 200。这里取最简单的打分策略，假设比对规则为匹配加 1 分，不匹配减 1 分，不完全匹配罚 20 分。表 3-1 中红色粗体标识出最优得分。从表中分数可以看出使用匹配优先策略计算出来的调和序列长度体现了非常显著的重复体部分。

表 3-1：三种比对方法的得分，红色粗体数字代表最优分值

| A(Q, S) | Q=100b | Q=150b | Q=200b |
|---|---|---|---|
| 局部序列比对 | 100000 | 125000 | **127500** |
| 完全匹配比对 | **100000** | 75000 | 68500 |
| 匹配优先比对 | 100000 | **115000** | 108500 |

### 3.2.2.3 比对过程及其优化

上面两节定义了多序列局部比对的目标函数，而且给出了比对得分函数的计算方法，接下来逐步优化多序列局部比对过程。

最容易想到的方法就是使用 n 维动态规划算法来计算 n 序列比对，但是时间复杂度和空间复杂度分别为 $O(L^n 2^n)$和 $O(L^n)$(L 为序列平均长度)，这在基因组的计算上是不可能实现的。算法扩展第 2 节提到的启发式算法 BLAST 来优化多序列局部比对过程。通过 Blast 算法搜索出来一个高频率出现的 l-mer 种子，$S_1,\ldots,S_n$ 是共享这个种子的序列片断，这些序列两边已经超出了每个重复体的真正边界。令 $Q_0$ 等于 l-mer 种子，然后贪婪的向左右扩展 $Q_0$，每次扩展一个碱基，每次扩展的碱基必须使目标函数 $A(Q; S_1,\ldots,S_n)$取最大值。$Q_0$ 在第 k 次扩展时，针对每一个碱基 N 计算 $A(Q_kN; S_1,\ldots,S_n)(N\in\{A,T,C,G\})$，$Q_kN$ 表示 $Q_k$ 扩展一个碱基 N($Q_k$ 的左边或右边)的序列。$Q_{k+1}$ 表示 $Q_kN$ 使得目标函数 $A(Q_kN; S_1,\ldots,S_n)$取得最大值时的 k+1 次循环的结果。结合公式 3-2 可以看出第 k+1 次循环的输入就是第 k 次循环的输出。算法每次将 $Q_0$ 扩展一个字节，为了限制运行时间，算法定义了一个阈值 I，当 Q 扩展了 I 次后还没有提高目标函数 $A(Q;S_1,\ldots,S_n)$的值就停止扩展过程。扩展方向默认从右边开始扩展，直到 $A(Q;S_1,\ldots,S_n)$的值不再增长时则扩展结束。

为了在局部序列比对时支持空位的插入，算法在扩展每个碱基的同时参考其周围一定范围内(-b 到 b)的碱基。首先假设当前序列 $S_i$ 已经不可以扩展，根据公式 3-2 使用比对分值减去不完全匹配罚分，记录下这个分值。在-b 到 b 范围内，先拿调和序列 Q 的候选碱基 N 和空位做比较，空位个数从 1 到与当前碱基 N 匹配的位置和-b 的差值，记录下来比对最大分值。然后拿调和序列 Q 的候选碱基与序列 $S_i$ 的当前碱基做比较，根据匹配得分计算出比对值。最后比较这几个值，取出一个最大比对值。于是比对得分函数可以修改为公式 3-3：

$$s(Q_kN,S_i)=\max_{f\in\{-b,b\}} s_f(Q_kN,S_i) \qquad \text{公式(3-3)}$$

使用上述方法计算出每一个 $S_i(1\leqslant i\leqslant n)$和 Q 最大比对分值 $s(Q,S_i)$。根据这些比对分值可以确定下来每一个 $S_i$ 的精确边界。这里使用目标函数的最大值的比对作为选择调和序列当前位置碱基的标准。这样的比对方法避免了 BLAST 算法不支持空位的问题，增加了比对结果的精确性。

根据上述过程，给定一个基因组，从中提取出一组高频率的 l-mer 种子，算法通过扩展这些种子得到代表一个重复体家族的调和序列。算法必须使用上述过程去计算每一个高频率的l-mer序列来识别给定基因组中所有的重复体家族序列。但是计算每一个高频率的 l-mer 序列是非常花费时间，而且是多余的。因为一个重复体家族会由于变异产生大量的高频率 l-mer 序列，其数目很可能大于重复体

的长度。为了解决这个问题，算法在每次计算出一个重复体家族的调和序列后记录下来调和序列的所有 l-mer 种子，然后判断原序列中的 l-mer 种子是否在调和序列中出现，假设出现的次数为 x，那么就将原序列中相应的种子频率减去 x，这样就刷新了原序列种子频率。

### 3.2.2.4 算法实现

本算法的主要核心是把种子向两边扩展的过程，Extend过程每次向左或向右扩展一个字节，计算每个碱基在这个位置上的最大比对值，然后取值最大的那个碱基将作为这个位置上的候选碱基，将其插入到调和序列中。假如在向右扩展时，如果$S_k$和Q比对时比对值一直没有提高则将比对值取最大值的点作为$S_k$的右边的边界，同样方法也可以确定下来$S_k$左边的边界。由此可以在比对过程中确定每个l-mer种子周围的边界。计算出来一个重复体家族中的调和序列后，为了避免重复查找算法可以根据这个调和序列刷新l-mer种子表中种子频率。

算法主过程伪代码如下：

**Algorithm RepeatSearcher(S, l, m, b)**

**Input:** *string S of length n, seed length l, minimum repeat frequency m, banding constraint b*

**Output:** *every repeat family's consensus sequence*

```
1.  l-mer_table ← Build_l-mer_Table(S, l, m)
2.  Q : consensus sequence
3.  l-mer_seed ← most frequency seed in l-mer_table   /*seed's hash table*/
4.  IF NOT any l-mer_seed satisfy the condition THEN exit
5.  Extend_Right(S, l-mer_seed, b, Q)
6.  Extend_Left(S, l-mer_seed, b, Q)
7.  IF( Q.right – Q.left ≥ GOODLENGTH )
8.        write Q into the output file : Q's   length longer than   GOODLENGTH
9.        Updata_l-mer_Table(Q, l-mer_table)
10.       GOTO step 3
11. ELSE
12.       Updata_l-mer_Table(Q, l-mer_table);
13.       GOTO step 3
14. the output file is our result which list every repeat family's consensus sequence
```

过程 Build_l-mer_Table(S, l, m)扫描一遍输入的基因组序列，判断基因组序列中所有长度为 *l* 的序列片断出现的频率和位置，然后在 hash 表中以种子序列为关键字记录这些序列片断的频率和位置。这些长度为 *l* 的序列片断就是算法的候选

l-mer 种子。种子刷新过程 Updata_l-mer_Table(Q, l-mer_table)根据上一节最后提到的刷新方法刷新种子列表中种子频率。

比对分值是扩展过程的主要依据。它可以通过多重循环来实现，比对分值 $s(Q_kN; S_i)$可以很快的通过上一次的计算结果 $s(Q_k; S_i)$计算出来。令比对的偏移量为第 k 次循环插入的数目减去删除的数目，接着计算和存储每一个偏移 f（$f\in[-b, b]$）的比对分数 $s_f(Q_k; S_i)$，这里同时借助了有限制的动态规划思想让 b 作为比对带宽的约束值，空位的插入只在这个范围内实现。$s_f(Q_kN; S_i)(f\in[-b,b])$可以使用上一次循环的结果按照公式 3-3 计算出来。在程序计算过程中相应的减小 b 值可以提高算法的计算速度。下面给出了计算目标函数扩展种子序列过程的伪代码：

**Subroutine Extend_Right(S, l-mer_seed, b, Q)**

**Input:** *string S of length n, selected l-mer seed, banding constraint b,*

**Output:** *consensus sequence which have extended right*

```
1.  initial scores[0→1][0→n-1][-b→+b]
2.  initial master[2L+l]    /*score the consensus sequence , L denote how far to extend*/
3.  FOR y ← L + 1   TO   2L + 1    /* determine master[y] and score[y%2][][] */
4.      FOR a ← 0 TO 3             /*(0→3) equal (A,T,C,G)*/
5.          FOR n ← 0 TO N-1   /*N is the number of seed's occurrence */
6.              FOR offset ← -b TO +b
7.                  bestscore = max_offset(s(Q_y a,S_n))
8.              bestscore_a += bestscore
9.          best_a = a max the bestscore_a
10.     master[y] = best_a
11.     IF the total score have not increased after 200 intervals THEN break
12. Q = master
```

子过程Extend_Left(S, l-mer_seed, b, Q)和子过程Extend_Right(S, l-mer_seed, b, Q)的算法过程相似，其区别只是种子的扩展方向不同，y是从L到1。

### 3.2.3 算法分析

算法主过程 RepeatSearcher(S, l, m, b)的时间复杂度依赖于子过程 Extend_Right 和 Extend_Left 的时间复杂度。子过程 Extend_Right 是一个多重循环过程，第 1 层循环从种子的右边第一个位置开始直到得到最大目标函数值或者到最长允许的比对长度为止，第 2 层循环对每个碱基循环计算，这里以 DNA 序列为对象，所以循环次数为 4。第 3 层循环的循环次数为当前种子的频率，最后一

次循环计算次数为 2b，2b 为算法的计算窗口大小，从当前比对位置向左右偏移 b。从上面分析可以看出扩展过程运算次数为 L×N×2b×4，时间复杂度为 O(L×N)。Extend_Left 的时间复杂度和 Extend_Right 相同。创建 l-mer 种子列表的过程是一个线性的过程，顺序扫描一遍输入序列就可以创建完成种子列表。刷新种子频率过程的循环次数是种子列表中频率大于重复体最低频率的种子数 $C_s$ 乘以调和序列 Q 中种子的个数 $C_q$，所以这个过程的时间复杂度为 $O(C_sC_q)$。刷新种子列表过程可以大大减少主过程的循环次数和扩展过程的循环次数。综上所述可以分析出算法的时间复杂度为 O(MN)。

算法在运行过程中存储种子信息到 hash 表中，这个表的大小和输入序列中种子的重复度有关系，最坏情况下表的大小将是种子长度 $l$ 乘以序列长度 N，最好情况下表大小为 1。扩展过程需要存储每次比对的分值，因为第 k 次的运算结果可以作为第 k+1 次的输入，所以算法的比对分值的存储空间大小为种子类型数|S|乘以每个种子出现的频率 $F_{seed}$ 再乘以总的偏移 2b，最后乘上种子可以扩展的最大长度 L，总空间为 $2bL|S|F_{seed}$。存储调和序列的空间大小为 2L+$l$（L 为左右运行扩展的最大长度，$l$ 为种子的长度）。最后需要存储的是基因组序列。根据上面的分析，算法的空间复杂度为 O(M+N)（M 由种子扩展长度 L，计算偏移 b，种子类型数目|S|和种子频率决定）。

### 3.2.4 实验结果及性能分析

程序 RepeatSearcher 在 FreeBSD 环境上通过 C 语言实现。Bao and Eddy 建议种子序列长度 $l = \lceil \log_4 N+1 \rceil$，N 为输入基因组的长度。这个值在多次试验中都已经经过了验证，实验取种子长度 $l$ 为 15。因为 RECON 算法标准输出只包含重复体出现次数超过 10 次的序列，所以 l-mer 种子频率阈值和重复体频率阈值均为 10。为了简单，两个元素匹配则得 1 分，不匹配得-1 分，插入一个空位得-5 分，一个序列不完全匹配则得-20 分。折中算法速度和精度，比对带宽约束值 b 设为 5。为了限制算法的运行时间，在向两边扩展种子时如果扩展了 200 次而没有提高目标函数 $A(Q;S_1,\ldots,S_n)$的值就停止扩展过程。因为几乎所有重复体家族的调和序列长度在 50 到 10000 之间(Bao and Eddy.,2002)，所以结果中不包含调和序列长度小于 50 的重复体。这一章只是简单的测试了算法的效果，更具体的测试见下章。本实验测试机器为 Intel 至强 3.66GHz, 8GB 内存的服务器。

#### 3.2.4.1 实验结果

实验使用108M的C.briggsae[44]基因组序列作为测试对象。同时使用RECON算法和RepeatSearcher算法来测试这个基因组序列，计算出基因组中重复体家族的数

目和各家族中的重复体序列大小，并记录下来算法运行时间。然后拿这两个算法的运行结果和RepBase数据库中相应基因组重复体家族的数目及家族中重复体序列的大小做对比，如表3-2。因为RECON算法的输出只包含重复体出现次数超过10次的重复体，所以RepeatSearcher算法把输出结果中循环次数小于10的重复体序列去掉。

表3-2: RepeatSearcher，RECON的结果中和RepBase中的家族个数和总大小，以及两个算法运行时间

| 测试方法 | 重复体家族个数 | 重复体家族大小 (Kb) | 运行时间(M) |
|---|---|---|---|
| **RepeatSearcher** | 1391 | 665.6 | 340 |
| **RECON** | 723 | 440.32 | 3370 |
| **RepBase** | 594 | 368.64 | --- |

RepeatMasker是在生物信息学中非常有名的重复体分析检测程序，它可以在基因组序列中找到一个重复体序列的所有相近的家族序列。实验使用RepeatMasker在C.briggsae基因组序列上分析检测算法RECON和RepeatSearcher的运行结果，表3-3列出了分析检测结果。

表3-3：RepeatMasker在C.briggsae基因组上分析RepeatSearcher和RECON输出的结果

| 大小(Mb) | **RepeatSearcher** 覆盖的大小 | **RepeatSearcher** 没有覆盖的大小 | 总大小 |
|---|---|---|---|
| **RECON** 覆盖的大小 | 22.9 | 2.1 | 25.0 |
| **RECON** 没有覆盖的大小 | 4.7 | 78.7 | 83.4 |
| 总大小 | 27.6 | 80.8 | 108.4 |

### 3.2.4.2 结果分析

基于表3-2和3-3的试验结果，这里分别从重复体识别结果和运行时间两个方面对试验结果进行分析。

重复体识别结果。RepeatSearcher算法在C.briggsae基因组中识别出来了1391个重复体家族，远远多于RECON的识别结果723个重复体家族，而RepBase中已经注册的重复体数目为594。从使用RepeatMasker重复体分析工具分析两个算法的结果表3-3可以看出RepeatSearcher总共覆盖了27.6Mb的重复体序列，而RECON覆盖了25Mb，两者可以覆盖的共同部分为22.9Mb，有4.7Mb被RepeatSearcher算法覆盖而被算法RECON遗漏，2.1Mb被RECON覆盖而被RepeatSearcher遗漏，RepeatSearcher的结果明显优于RECON的结果。根据算法的特点和结果分析，这些区别在于重复体家族之间的区别和重复体家族序列的边界区别。从总体上来看， RepeatSearcher

算法的灵明度远远高于RECON，而且RepeatSearcher算法找出的每一个重复体家族序列的边界更精确。下一章对算法RE值的分析可以很好地说明这一点。

运行时间。RECON算法直接对整个基因组计算将会导致算法无法运转下去，所以这里将整个基因组分割为大小为3M(测试大小)大小的序列，然后逐个计算，并将总时间计算出来。从表3-2可以看出，RepeatSearcher和RECON的运行时间为340分钟和3370分钟。从运行时间上RepeatSearcher算法远远优于RECON算法，这也证明了算法在时间复杂度上的改进。下一章算法将在如何提高算法的精度上做改进，并将引入概率统计学方法来进一步提高算法的效率。

## 3.3 本章小结

本章根据当前重复体识别算法中存在的问题提出了一种基于BLAST算法具有精确边界的重复体快速识别算法RepeatSearcher。算法在计算多序列局部比对的同时定义了每一个重复体序列的边界条件，保证了每一个重复体序列具有精确的边界。算法结合限制范围的空位插入方法，既提高了运行速度也增加了结果的精确性。在识别重复体的同时算法计算出来了重复体家族的调和序列。试验结果表明RepeatSearcher算法的运算速度相对其它算法得到了很大的提高而且也保证了非常高的精确性，是一种高效的重复体识别算法。下一章将结合概率统计方法继续对RepeatSearcher进行了改进，并用RepeatSearcher对12组哺乳动物基因组序列进行了测试。

# 第四章 基于概率统计的重复体识别算法

当前的重复体识别算法大部分都是以序列比对为中心判断多个序列是否属于同一个家族，第三章提到的 RepeatSearcher 算法是通过双序列比对构造多序列比对来识别重复体。这一章将参照 Gibbs 采样算法结合种子序列通过统计分析来识别重复体家族序列，更精确的评估了重复体序列的边界。算法 GSRSearcher 不但提高了重复体识别的精度，同时也提高了运算速度。

## 4.1 Gibbs 采样算法

Gibbs采样算法是一种特殊的马尔可夫链蒙特卡罗方法（Markov Chain Monte Carlo，MCMC），该算法最早是用于蛋白质序列中的模体(motif)序列识别，其基本原理是通过随机采样不断更新模体和在各序列中出现的位置以优化目标函数，当满足一定的迭代终止条件时就得到了最终的候选模体。后来Gibbs采样被整合进贝叶斯模型并应用于多序列比对，获得了较好的结果。

### 4.1.1 Gibbs 采样算法数学背景

Gibbs 采样初始需要一个参数 $\Phi$ 作为算法的开始点，一个向量集 $\theta$，一个观察值 $Y$，算法从这些初始参数中找到一个集合分布。采样必须有一个初始的起点 $(\theta_1^{(0)}, \theta_2^{(0)},\ldots, \theta_D^{(0)}, \Phi^{(0)})$,然后重复执行下面四个过程：

(1) Sample $\theta_1^{(i+1)}$ from $p(\theta_1 \mid \theta_2^{(i)},\ldots, \theta_D^{(i)}, \Phi^{(i)}, Y)$

(2) Sample $\theta_2^{(i+1)}$ from $p(\theta_2 \mid \theta_1^{(i+1)}, \theta_3^{(i)},\ldots, \Phi^{(i)}, Y)$

(3) Sample $\theta_D^{(i+1)}$ from $p(\theta_D \mid \theta_1^{(i+1)},\ldots, \theta_{(D-1)}^{(i+1)}, \Phi^{(i)}, Y)$

(4) Sample $\Phi^{(i+1)}$ from $p(\Phi \mid \theta_1^{(i+1)},\ldots, \theta_D^{(i+1)}, Y)$

向量 $\theta^{(0)}, \theta^{(1)},\ldots, \theta^{(t)}$表示马尔可夫链实现的表示，其中 $\theta'$到 $\theta$ 的转移概率定义如下：

$$K(\theta',\theta) = p(\theta_1|\theta_2',\ldots, \theta_D', \Phi', Y) \times p(\theta_2|\theta_1, \theta_3',\ldots, \theta_D', \Phi', Y) \times\ldots\times$$
$$p(\theta_D|\theta 1,\ldots, \theta(D\text{-}1), \Phi', Y) \qquad \text{公式(4-1)}$$

从上面过程可以看出当 i 趋于∞时，$(\theta_1^{(i)},\ldots, \theta_D^{(i)}, \Phi^{(i)})$的联合分布从几何上趋于 $p(\theta_1,\ldots, \theta_D, \Phi|Y)$。

### 4.1.2 多序列比对中的 Gibbs 采样算法

Gibbs 采样算法在生物信息学中主要应用是检测和比对蛋白质序列或者 DNA 序列中的保守区域，而算法不依赖于任何先验知识。算法避免了其它算法(如 EM[45])陷入局部最优的问题，而且保证了算法的速度和敏感度。

(1) 算法参量

N：序列的个数

$S_1 \ldots S_N$：序列集合

W：各序列中模体的长度

J：符号表中残基的个数，如果是核酸序列时 J=4，如果是蛋白质序列则 J=20。

$c_{i,j,k}$：模体 k 位置 i 上观察到的残基 j 的个数。j 属于 1...J，i 属于 0...W。$c_{0,j}$ 表示背景中残基 j 的数目。在最简单的情况下只有一个模体时 k 等于 1。

$q_{i,j}$：在模体中第 i 个位置上面残基 j 出现的频率。i 属于 0...W。$q_{0,j}$ 表示残基 j 的背景频率。

$a_k$：序列中模体起始位置向两，k 属于 1...N。

$b_j$：每一个残基的伪次数，根据贝叶斯统计[46]规则消除次数为 0 的情况。

B：所有残基伪次数的和，$B = \sum_j b_j$

(2) 算法过程

算法是一个循环迭代的过程，主要分为如下几个步骤：初始化，预测刷新过程和采样过程，下面分别描述这几个过程。

在初始化过程，输入给定了初始序列，有了这些序列表示残基背景出现次数的参数 $c_{0,j}$ 可以很容易计算出来。而所有的 $c_{i,j}(i \neq 0)$ 被初始化为 0。在每一个输入序列中随机或者按照某种规律选择一个模体的起始位置 $a_k$，根据上面的这些参数的设定 $q_{i,j}$ 可以被计算出来。计算公式如下：

$$q_{i,j} = \frac{c_{i,j} + b_j}{N - 1 + B} \qquad \text{公式(4-2)}$$

$$q_{0,j} = \frac{c_{0,j} + b_j}{\sum_{k=1}^{J} c_{0,k} + B} \qquad \text{公式(4-3)}$$

在预测刷新过程，随机的选择一个序列并把这个序列中的模体放在背景序列中，然后刷新碱基出现次数。比如 N 个序列中的序列 $S_z$ 被选中，则将 $S_z$ 中的模

体放到背景中，残基出现的次数 $c_{i,j}$ 和残基的频率 $q_{i,j}$ 也随之被刷新。序列 $S_z$ 的选择可以是随机的也可以是有特定顺序的。

采样过程中算法从预测刷新过程选择出来的序列 $S_z$ 中重新选择一个模体位置。$S_z$ 中所有长度为 W 的序列都有可能，这个位置通过对其中所有长度为 W 的序列通过权值比较来判断。对于每一个长度为 W 的片断，权值计算如下式：

$$A_x = Q_x / P_x \qquad \text{公式(4-4)}$$

$$Q_x = \prod_{i=1}^{W} q_{i,r_i} \qquad \text{公式(4-5)}$$

$$P_x = \prod_{i=1}^{W} q_{0,r_i} \qquad \text{公式(4-6)}$$

其中 $Q_x$ 表示片断 x 为模体的概率，$P_x$ 是根据公式 2 计算出来的背景残基的频率。$r_i$ 表示 x 片断上面第 i 个位置上面的残基。当对于每一个长度为 w 的序列片断算出了 $A_x$ 值，算法根据这些值的权重从中随机选取一个新的位置 $a_z$。选定的这个新模体具有更高的权重，因此更有可能是算法需要寻找的片断。因为这个过程是随机过程，所以不一定选择到的是最高权重的序列，这样也避免陷入局部最大。

对所有序列迭代使用预测刷新过程和采样过程，最后可以得到一个可能的比对结果。对于这个比对算法可以计算出来最大后验概率值（maximum aposteriori probability ，MAP），计算方法如下：

$$F = \sum_{i=1}^{W} \sum_{j=1}^{J} c_{i,j} \log \frac{q_{i,j}}{q_{0,j}} \qquad \text{公式(4-7)}$$

算法的最终目标就是最大化 F。下面是算法的伪代码：

```
1.  globalMaxAlignmentProb = 0
2.  For Iteration = 1 TO N:
3.      Initialize Random alignment
4.      LocalMaxAlignmentProb = 0;
5.      WHILE(not in local maximum and innerloop < MAXLOOP) DO
6.          FOR each sequence DO
7.              Predictive Update
8.              Sample
9.          calculate AlignmentProb
10.         IF(AlignmentProb > localMaxAlignmentProb)
11.             localMaxAlignmentProb = AlignmentProb;
```

```
12.                 not in local maximum = TRUE;
13.             innerloop++;
14.         IF(localMaxAlignmentProb == globalMaxAlignmentProb)
15.             exit → max found twice
16.         ELSE IF(localMaxAlignmentProb > globalMaxAlignmentProb)
17.             globalMaxAlignmentProb = localMaxAlignmentProb
```

## 4.2 GSRSearcher 算法的引入

上一章中的 RepeatSearcher 算法通过种子的扩展识别出来了每一个重复体序列的具体位置，最终求出了重复体家族的调和序列。在确定调和序列每一个位置上碱基的过程中限定了碱基比对的范围，在这个范围内判断序列的插入，删除和取代。家族中每一个重复体结束点和调和序列扩展终点的判断主要依赖于当前已经比对的序列，而没有考虑整个基因组序列空间中背景序列的影响，这势必会影响到确定终点的精确性。针对这个问题，这一章提出了基于 Gibbs 采样算法的扩展过程。

上一节列出了用于多序列比对的 Gibbs 采样算法。算法目标是找一个固定长度的模体，这个模体在每一个序列中仅出现一次，模体的选择必须参考背景残基出现的概率。因为初始时模体在每一个序列中的起始位置是未知的，所以算法的开始时就必须随机地在每一个输入序列中选择一个模体序列的开始位置作为算法的初始起点，然后通过刷新和采样过程循环计算得到结果。在重复体识别问题中，初始的每一个种子序列位置已经知道，而需要知道的是种子序列扩展过程中的终点位置和调和序列扩展的结束点位置。本章的算法将结合背景频率构建 PSSM(Position Specific Scoring Matrix)[47][48]来增加识别的精确性，同时在 PSSM 的构建中增加空位的插入。

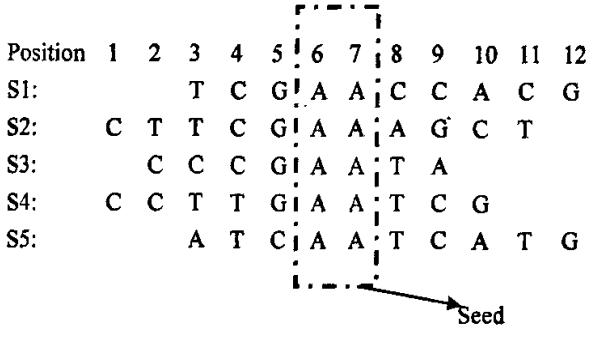
## 4.3 GSRSearcher 算法描述

GSRSearcher 算法继承了第三章 RepeatSearcher 中的以种子序列扩展为核心的思想，扩展过程结合的 Gibbs 采样算法的背景考察策略。对于重复体识别问题，需要解决的问题重点在于调和序列扩展的目标函数，重复体序列扩展的分值计算方法和调和序列碱基的选择策略。下面几节将展开讨论这几个方面。

### 4.3.1 目标函数的定义

算法首先构建扩展矩阵，矩阵的横坐标表示重复体的位置，纵坐标表示基因组中含有的所有的碱基的种类并加上空位。矩阵中的值表示这个位置上的碱基出现的频率。因为每条家族中重复体的长度都不一定相同，所以矩阵中的横坐标的长度跨度为N条序列中左右扩展最长的位置，没有扩展的地方用空位填充。空位在初始序列中是不存在的，这里设定空位相对背景的概率为 1，这样空位取对数后为 0，不会影响到目标函数的结果。计算碱基在某一位置上的频率时，需要根据公式 4-2 考虑到碱基的伪数目以防止碱基频率为 0 的情况。示例中简单的设置A,T,C,G 碱基伪频率各为 1。

(a) 包含相同种子的重复体序列




(b) (a)中序列的频率矩阵

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

(c) 根据频率矩阵(b)算出来的初始概率矩阵

| P(S) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 0.11 | 0.11 | 0.22 | 0.11 | 0.11 | 0.66 | 0.66 | 0.22 | 0.22 | 0.33 | 0.11 | 0.11 |
| T | 0.11 | 0.22 | 0.44 | 0.33 | 0.11 | 0.11 | 0.11 | 0.44 | 0.11 | 0.11 | 0.33 | 0.11 |
| C | 0.33 | 0.33 | 0.22 | 0.44 | 0.22 | 0.11 | 0.11 | 0.22 | 0.44 | 0.22 | 0.22 | 0.11 |
| G | 0.11 | 0.11 | 0.11 | 0.11 | 0.55 | 0.11 | 0.11 | 0.11 | 0.22 | 0.22 | 0.11 | 0.33 |
| - | 0.33 | 0.22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.11 | 0.22 | 0.33 |

图 4-1:构建概率矩阵

上图中的频率表 P(S)只是显示了本身看到的碱基的频率，但是并没有考虑到背景 P(M)给这些位置碱基带来的影响。在随机的背景频率下面碱基出现的概率应为 P(S|M)。为了计算得分算法给 P(S|M)取对数，当得分为 0 时表示序列 S 在背景 M 中出现的概率和在重复体中的概率相等，得分小于 0 则表示 S 在背景 M 中的概率大于在重复体的概率，得分大于 0 表示 S 在重复体 R 中的概率大于在背景 M 中的概率。这里假设各个碱基的背景频率为 0.25，那么上图中的频率表将转换为下面的表。因为空位在背景频率中是不出现的，所以略去空位的行。

表 4-1：图 4-1(a)中序列的碱基位置概率对数矩阵

| lnP(S\|M) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | -0.82 | -0.82 | -0.13 | -0.82 | -0.82 | 0.97 | 0.97 | -0.13 | -0.13 | 0.28 | -0.82 | -0.82 |
| T | -0.82 | -0.13 | 0.57 | 0.28 | -0.82 | -0.82 | -0.82 | 0.57 | -0.82 | -0.82 | 0.28 | -0.82 |
| C | 0.28 | 0.28 | -0.13 | 0.57 | -0.13 | -0.82 | -0.82 | -0.13 | 0.57 | -0.13 | -0.13 | -0.82 |
| G | -0.82 | -0.82 | -0.82 | -0.82 | 0.79 | -0.82 | -0.82 | -0.82 | -0.13 | -0.13 | -0.82 | 0.28 |

表 4-1 中给出了每一个碱基在各个位置上面作为重复体序列的得分，根据这些得分值，算法取目标函数为这些位置上面碱基得分之和 A：

$$A=\sum_{k=1}^{N}\sum_{i=1}^{|R_k|}\sum_{j=1}^{J}\log\frac{p_{k,i,j}}{p_{0,j}} \qquad \text{公式(4-8)}$$

其中 N 表示拥有共同种子序列的序列条数，$|R_k|$代表第 k 个序列的长度，J 表示碱基的种类数目，$p_{k,i,j}$为重复体序列 $S_k$ 上在序列矩阵中第 i 个位置上作为碱基 j 的概率。将序列矩阵转化为 PSSM 矩阵后可以将公式 4-8 转换为公式 4-9，公式 4-9 类似于 Gibbs 采样的目标函数，区别是在这里 W′ 表示重复体序列片断 1 到 N 跨度最长的值，概率 $p_{i,j}$ 的计算考虑了用空位填充序列不对齐的问题。

$$A=\sum_{i=1}^{W'}\sum_{j=1}^{J}C_{i,j}\log\frac{p_{i,j}}{p_{0,j}} \qquad \text{公式(4-9)}$$

$C_{i,j}$ 表示碱基 j 在位置 i 上面的频率。算法的目标就是最大化 A，当 A 值在调和序列的扩展过程中开始递减时扩展过程结束，算法结束。这时候每一条重复体的位置就是当前重复体的边界，同时也确定了调和序列的最大长度。

### 4.3.2 种子序列的扩展过程

上节介绍了算法的目标函数，解决了调和序列的扩展范围。这个范围确定之

后算法需要考虑的下一个问题就是如何扩展，每一次扩展时以什么样的标准来选择候选碱基，这里需要一个合理的比较算法才能达到很好的效果。还有一个问题就是每一条种子序列的扩展到什么时候结束，即每一条重复体序列边界的定位需要一个判定函数来衡量，而这个函数必须反映每一条重复体序列在当前样本序列中的权值。

重复体序列一般是基因组序列中的片段序列，它们在全局上是没有什么特征，而它们在局部中却有很高的相似度。鉴于这一点，算法在 N 条种子序列扩展时限定了相对于当前位置的一个计算范围[-b,b]。式 4-10 给出了确定当前位置 $l$ 上面碱基的分数，其中 N'表示当前参与扩展的重复体序列数目，这个数目是随着扩

$$s(Q_{l-1}\cdot L)=\max_{j\in\{A,T,C,G\}}\sum_{i=1}^{N'}\left(\frac{1}{N'}-\frac{k_{i,j}}{N'\times b\times\eta}\right),\quad 0\le k\le b \qquad \text{公式(4-10)}$$

展的进行而逐步减少的，因为部分重复体的边界已经确定下来而无需进一步扩展。算法判断位置 i 上面每一个碱基的概率，初始在 i 位置上碱基 j 出现的概率为 1/N'，然后判断碱基 j 在每一个扩展序列中出现的位置相对 i 的偏移 k，这个偏移也就是需要插入空位的个数，这个过程限定在范围[-b,b]之间，所以 0≤k≤b，k 取值为碱基 j 最小的频率。初始概率减去空位带来的罚分概率则是这个位置上碱基 j 出现的概率，然后将每条扩展序列上面的概率相加起来则得到了碱基 j 在 N'条序列中在当前位置出现的概率。η 在这里表示罚分概率的调节因子，以防止罚分概率过小或者过大的情况。

按照上面提出的扩展过程，算法可以确定调和序列每一个位置上概率最大的碱基。调和序列上面的碱基确定以后，所有扩展序列就可以构建出来 PSSM 矩阵，每条扩展序列上面碱基都有各自的出现频率，算法根据这些频率计算这条扩展序列是否需要进行进一步扩展。式 4-11 根据式 4-4 给出了一条重复体序列的得分，

$$R_x=\prod_{i=1}^{W'}q_{i,r_i}\Big/\prod_{j=1}^{W'}q_{0,r_i} \qquad \text{公式(4-11)}$$

当扩展过程达到一定次数而没有增加 $R_x$ 的最大值时则停止这条序列的扩展。x 表示重复体序列得分最大值的点，$r_i$ 是当前位置上面的碱基。当扩展种子序列右边边界时，W'是从种子左边界到位置 x 的宽度，当扩展种子序列左边时，W'是从当前位置 x 到种子右边边界的位置。因为在扩展一边时背景序列只考虑种子一边的序列。种子序列扩展方向的先后对算法结果没有影响。

### 4.3.3 算法实现

GSRSearcher算法的主要过程和第三章的RepeatSearcher算法相似，因为两者都

是从种子序列开始，并逐步向两边扩展种子序列直到得到最优值。而两者之间的区别在于对评分的计算。后者主要通过比对来打分而前者主要通过前景和背景的概率之比来给序列打分。GSRSearcher算法在调和序列每扩展一个碱基就刷新一次PSSM矩阵，刷新完成后判断全局扩展结束条件和每一条扩展序列的结束条件，确定结束点。类似于RepeatSearcher算法，每次扩展完成一个家族以后需要根据求出来的调和序列刷新其它家族的l-mer种子列表中相应的频率。

算法主过程伪代码如下：

**Algorithm GSRSearcher(S, l, m, b, L)**

**Input:** *string S of length n, seed length l, minimum repeat frequency m, banding constraint b max extention length L*

**Output:** *every repeat family's consensus sequence*

```
1.  l-mer_table ← Build_l-mer_Table(S, l, m) /*seed's hash table*/
2.  Q : consensus sequence
3.  l-mer_seed ← most frequency seed in l-mer_table
4.  IF NOT any l-mer_seed satisfy the condition THEN EXIT
5.  Extend_Right(S, l-mer_seed, b, Q, L)
6.  Extend_Left(S, l-mer_seed, b, Q, L)
7.  IF( Q.right – Q.left ≥ GOODLENGTH )
8.          write Q into the output file : Q's  length longer than  GOODLENGTH
9.          Updata_l-mer_Table(Q, l-mer_table)
10.         GOTO line 3
11. ELSE
12.         Updata_l-mer_Table(Q, l-mer_table);
13.         GOTO line 3
14. the output file is our result which list every repeat family's consensus sequence
```

扩展分值 $s(Q_{(L-1)}.L)$是判断扩展过程一个主要的依据，它主要用来判断调和序列扩展的结束点和每条重复体序列的结束点。扩展过程需要维护一个碱基的频率矩阵，矩阵是一个 5×L 的矩阵，L 表示重复体最长可以扩展的大小。每次计算时不需要更新矩阵中的所有列，而只需更新到扩展到的列为止。扩展过程先根据公式 4-10 确定当前调和序列位置上面的碱基，然后根据结果刷新频率矩阵判断扩展过程是否需要结束，最后判断每一个参与扩展的序列是否已经达到了最优结束点，如果已经得到最优点则不再需要参与下一步扩展。左右扩展过程计算方法相同。

扩展过程伪代码如下：

**Subroutine Extend_Right(S, l-mer_seed, b, Q, L)**

**Input:** *string S of length n, selected l-mer seed, banding constraint b,max extention length L*

**Output:** *consensus sequence which have extended right*

```
1.  initial p[0→4][0→L-1] /*initial p according the input sequence*/
2.  initial master[2L+l]   /*score the consensus sequence , L denote how far to extend*/
3.  FOR y ← L+l  TO  2L+l      /* determine master[y] and p[][] */
4.      FOR a ← 0 TO 3          /*(0→3) equal (A,T,C,G)*/
5.          FOR n ← 0 TO N-1    /*N is the number of seed's occurrence */
6.              FOR offset ← -b TO +b
7.                  k = min offset from y to a   /*look for the min offset*/
8.                  score = s(Q(i-1),L)          /*compute a's score*/
9.              bestscore_a += score
10.         best_a = a max the bestscore_a
11.     master[y] = best_a          /*conform the element at position y*/
12.     stop = UpdatePSSM(p, S, y) /*Updata PSSM and judge the stop condition and   */
13.                                /*judge whether any sequence should stop extension*/
14.     IF stop = TRUE THEN break
15. Q = master
```

PSSM 矩阵的刷新过程在调和序列每扩展一个位置以后进行，判断扩展过程是否应该结束。矩阵的刷新过程先统计碱基的频率，然后计算概率，最后根据公式 4-9 判断得分值，如果得分值不再增加则停止扩展过程。这个过程在目标函数定义中进行了阐述。过程 UpdatePSSM(p, S, y)不但完成这个过程而且在更新完成矩阵后判断每条重复体的终点。

矩阵刷新过程伪代码：

**Subroutine UpdatePSSM(p, S, y)**

**Input:** *the Matrix of probability p, sequences of extention S, current position y*

**Output:** *the updated matrix p and whether to extend continually*

```
1.  IF y EQUAL L THEN RETURN TRUE
2.  update C[0→4][0→L-1]   /* create matrix C in the first run */
3.  compute p[0→4][0→L-1]
4.  compute F       /*according to formula 4-9*/
5.  IF F continually decrease after extended 200 repeats THEN RETURN TRUE
6.  FOR  i ← 1  TO  |S|
7.      compute Ry of Si    /*according to formula 4-11*/
8.      IF Ry continually decrease after extend 50 repeats THEN remove Si
9.  IF |S| <> 0 THEN RETURN FALSE   /*still have any sequence*/
10. ELSE RETURN TRUE
```

## 4.4 算法分析

GSRSearcher 算法主过程由扩展过程 Extend 和种子列表刷新过程组成，主过程的时间复杂度主要依赖于扩展的时间复杂度。过程 Extend_Right(S, l-mer_seed, b, Q, L)中的 3 到 11 行确定调和序列当前位置上面的碱基。第一个循环从 L+*l* 到 2L+*l*，长度 L 为最长可扩展的长度，这是根据生物学重复体一般长度预先设定的常数值。第二层循环表示 4 个碱基，第三层循环 n 表示扩展序列集合 S 中拥有序列的条数，最后一个循环是需要计算候选碱基得分的范围。根据这些值可以确定 3 到 11 行的计算次数为 L×4×N×2×b，b 的取值一般比较小，所以时间复杂度可以写为 O(LN)。过程 UpdatePSSM(p, S, y)的时间复杂度由两部分组成，一部分计算 PSSM 矩阵，一部分是判断每一个当前重复体的得分。PSSM 矩阵的规模为 5×L，所以这个过程的计算次数为 5L。N 个当前重复体需要计算 N 次，根据这两部分的计算量可以推算出来过程 UpdatePSSM(p, S, y)的时间复杂度为 O(L+N)。结合第三章中对刷新种子列表过程时间复杂度的由此可以得出算法总的复杂度为 O(MN)。虽然 GSRSearcher 算法和 RepeatSearcher 算法在总的时间复杂度上是相同的，但是 GSRSearcher 算法在判定最优比对时省去了针对每个偏移的计算过程，所以总的速度上优于算法 RepeatSearcher。

和 RepeatSearcher 算法相似，GSRSearcher 算法也是将种子列表存储到 hash 表中，hash 表的大小和种子的重复度有关，最坏情况可以到种子序列长度 *l* 乘以基因组序列长度 N，最好时只需要 *l*。扩展过程中需要存储 PSSM 矩阵并在其上进行运算，前面已经提到 PSSM 矩阵的规模为 5×L，存储相同种子序列需要空间大小为 N×L。判断每一条扩展序列过程都需要一个存储空间存储当前的得分值，这里需要 N 个存储空间，N 代表家族种子的频率。最后存储调和序列和输入基因组序列需要长度为 2L+*l* 的扩展存储空间和长度为 N 的基因组空间。这样算法的总的空间复杂度为 O(M+N)(M 由扩展长度 L 和种子序列种类数目和种子频率决定)。和 RepeatSearcher 算法相比这里少了计算每个偏移得分的存储空间，所以 GSRSearcher 的空间复杂度优于 RepeatSearcher。

## 4.5 实验结果及性能分析

算法 GSRSearcher 在 FreeBSD 环境上通过 C 语言实现，种子序列长度依照 RepeatSearcher 算法的值取$\lceil \log_4 N+1 \rceil$。因为 RECON 算法标准输出只包含重复体出现次数超过 10 次的序列，所以 l-mer 种子频率阈值和重复体频率阈值均为 10。比对带宽约束值 b 设定为 5。为了限制算法运行时间，在向两边扩展种子时如果

超过了 200 次而没有提高目标函数则停止扩展过程，扩展结束。限定每一个重复体序列在扩展 50 次之后还没有提高重复体得分函数则这条种子序列扩展结束。重复体最大扩展范围设定为 10000，算法在结果中丢弃了长度小于 50 的重复体序列。计算重复体序列中碱基频率时，设定每种碱基的伪频率为各种碱基在输入序列中出现的频率，这样伪频率也呈现了背景的频率。实验测试机器为 Intel 至强 3.66GHz，8GB 内存的服务器。

为了验证算法的性能，进行了两方面的实验。首先从 RepBase 数据库中抽取了 12 组重复体家族序列，验证算法在识别重复体家族时的准确性。这 12 组基因组序列在 2004 年被录入 NISC 数据库中。这里使用 RECON 算法，第三章的 RepeatSearcher 算法，GSRSearcher 算法的结果和 RepBase 中已经注册的重复体家族序列标准结果做比较。因为 RECON 算法直接计算特大的基因组时一次计算耗时太长，所以先将大基因组分成小段，同时运行，然后再对所有结果合并。最后从这 12 组基因组中随机抽取了 Cow、Dog、Horse 和 Pig 四个家族基因组序列来对几个算法的结果进行分析，测量算法确定边界的精确性。

### 4.5.1 实验结果

表 4-2：用于测试的 12 组基因组序列 (Gi 是对基因组序列名称的缩写)

| Genome | Human X(chr)(G1) | Mouse X(chr)(G2) | Rat X(chr)(G3) | Cat (G4) | Cow(G5) | Dog(G6) |
|---|---|---|---|---|---|---|
| Size | 153.7M | 160.6M | 160.8M | 138.5K | 148.1K | 151K |
| **Genome** | **Horse(G7)** | **Pig(G8)** | **Rabbit(G9)** | **Vervet(G10)** | **Gorilla(G11)** | **Chimp(G12)** |
| Size | 152.3K | 112.8K | 168.8K | 157.5K | 181.5K | 141.3K |

在每一个基因组下应用算法 RECON，RepeatSearcher 和 GSRSearcher。通过算法运行时间，识别出来重复体家族的个数和 RepeatMasker 在重复体数据库中覆盖出来的重复体序列大小来判断这几个算法的运行效率，并且拿算法识别出来的结果和 RepBase 数据库当前标准作了比较。表 4-3 给出了几个算法针对不同输入序列的运行时间，表 4-4 给出各算法结果通过 RepeatMasker 在 RepBase 数据库中覆盖出来的重复体序列大小。图 4-2 描绘了各算法识别出来的重复体序列家族数目。

表 4-3：12 组基因组序列应用于各算法的运行时间

| 运行时间(M) | G1 | G2 | G3 | G4 | G5 | G6 | G7 | G8 | G9 | G10 | G11 | G12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **RECON** | 6350 | 6783 | 6803 | 32.3 | 40.5 | 45.4 | 46 | 28.9 | 50.2 | 49.6 | 59.1 | 35.2 |
| **RepeatSearcher** | 480 | 489 | 492 | 6.5 | 7.4 | 7.8 | 7.8 | 5.5 | 8.3 | 8.0 | 9.1 | 6.6 |
| **GSRSearcher** | 352 | 374 | 378 | 4.9 | 5.4 | 5.7 | 5.8 | 4.7 | 6.8 | 6.5 | 6.9 | 5.1 |

表 4-4：识别的重复体家族序列的大小

| 重复体大小(K) | G1 | G2 | G3 | G4 | G5 | G6 | G7 | G8 | G9 | G10 | G11 | G12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **RECON** | 829.33 | 1249.53 | 475.12 | 9.55 | 19.6 | 3.74 | 16.82 | 4.61 | 13.29 | 9.25 | 9.71 | 9.52 |
| **RepeatSearcher** | 643.28 | 1214.31 | 529.09 | 11.05 | 15.41 | 3.03 | 16.7 | 5.98 | 13.8 | 8.99 | 10.3 | 9.12 |
| **GSRSearcher** | 655.36 | 1239.04 | 522.24 | 11.23 | 17.9 | 4.51 | 16.82 | 6.13 | 14.67 | 10.6 | 11.8 | 10.04 |
| **RepBase** | 993.28 | 501.76 | 501.76 | 9.24 | 11.88 | 2.63 | 7.23 | 4.8 | 16.2 | 7.95 | 6.6 | 8.86 |

表 4-5: RepeatMasker 对各算法识别出来重复体覆盖结果

| 覆盖结果(K) | G1 | G2 | G3 | G4 | G5 | G6 | G7 | G8 | G9 | G10 | G11 | G12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **RECON** | 35260.2 | 25770.3 | 18392.3 | 20.1 | 45.3 | 7.4 | 24.9 | 12.6 | 36.1 | 17.1 | 15.9 | 15.4 |
| **RepeatSearcher** | 31821.3 | 22950.1 | 20381.6 | 31.8 | 40.7 | 6.8 | 24.6 | 13.8 | 38.6 | 15.7 | 17.6 | 15.2 |
| **GSRSearcher** | 32940.8 | 24236.8 | 20995.2 | 35.1 | 42.8 | 7.6 | 25.3 | 14.4 | 42.1 | 19.1 | 19.1 | 17.5 |

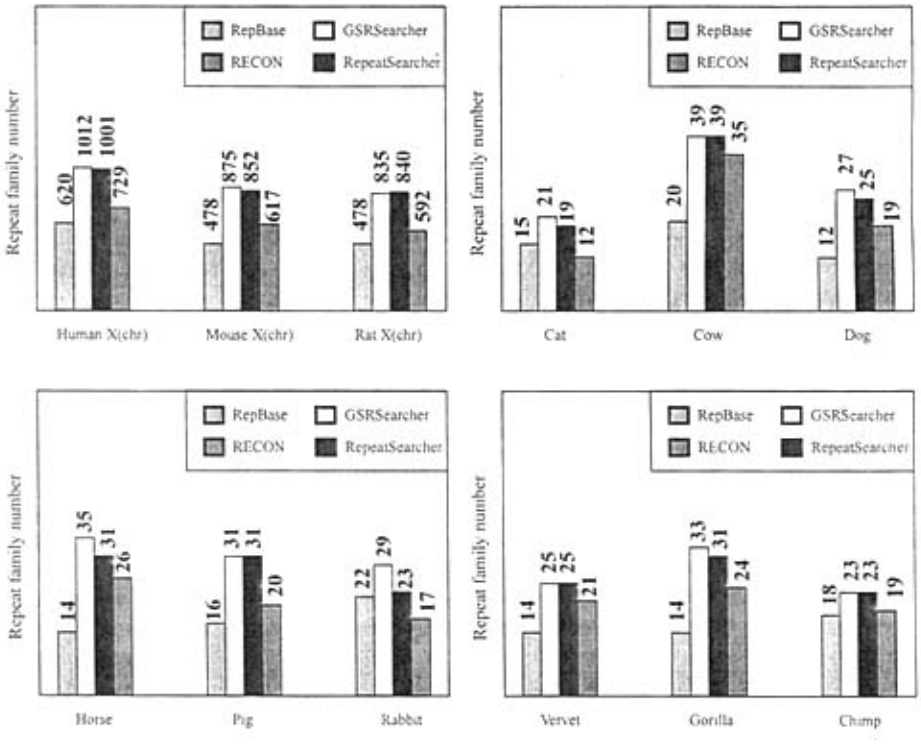



图 4-2: 算法 RECON,RepeatSearcher,GSRSearcher 识别的重复体数目和 RepBase 中的数目

实验从这 12 组结果中选取了 Cow、Dog、Horse 和 Pig 四个基因组序列的结果，针对算法 RECON, RepeatSearcher 和 GSRSearcher 识别出来结果中的 15 条重

复体家族调和序列计算出来了第二章提到的 RE 值。通过比较 RE 值可以很容易的看出算法敏感性。如图 4-3 所示。

### 4.5.2 结果分析

从表4-3的结果中可以看出算法RepeatSearcher和GSRSearcher的运算时间远远优于算法RECON的运行时间，而且基本上和输入序列的大小成线性比例的。而RECON算法在针对大型基因组序列时必须分割分析才能计算下去。GSRSearcher

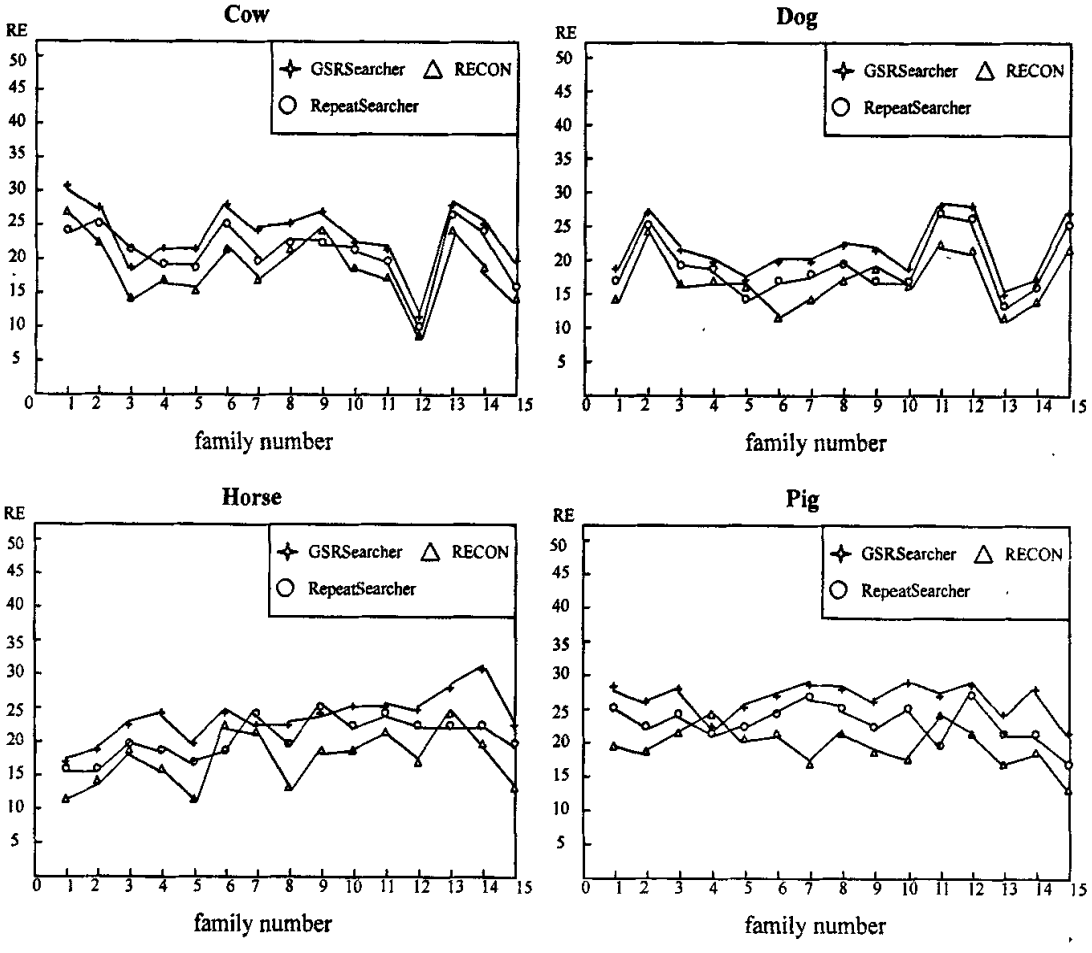



图 4-3: 算法 RECON、RepeatSearcher、GSRSearcher 针对 Cow、Dog、Horse 和 Pig 基因组识别结果的 RE 值对比

相对RepeatSearcher提高了局部比对时的时间，所以运行速度快于RepeatSearcher算法，这个从运行结果中可以很明显的看出来。

图4-3给出了各算法针对每组基因组序列识别结果的比较，可以看出RepeatSearcher和GSRSearcher算法识别出来的重复体家族数目均大于RECON算法

的识别数目。因为前两种算法都是基于同一种子序列识别算法，区别只是在种子序列频率刷新时产生的差异。从表4-4给出的算法识别结果大小和表4-5给出的通过RepeatMasker工具针对重复体家族序列RepBase上面覆盖出来的结果来看，RepeatSearcher和GSRSearcher算法大部分情况下都优于RECON算法。RepeatSearcher算法大部分情况下面在判断重复体家族序列边界时没有考虑全局背景影响，所以导致它的识别结果一般少于GSRSearcher算法。

从图 4-3 的四组基因组序列中重复体家族序列的 RE 值可以看出本文算法的 RE 值明显大于算法 RECON 结果的 RE 值。因为种子频率列表刷新和边界精确度不同的原因，算法 GSRSearcher 的 RE 值优于 RepeatSearcher。

## 4.6 本章小结

本章算法是在第三章提出的基于高度保守的种子序列并通过计分比对扩展调和序列和识别重复体边界的算法基础上提出来的。算法结合 PSSM 矩阵的概率统计方法对调和序列的扩展过程和每条重复体序列的扩展过程进行了概率评估，并且参考了背景碱基对重复体序列的影响，使输出结果精确度更高。同时算法使用在比对范围内采集得分位点的方法代替全比对降低了算法的时间复杂度和空间复杂度，提高了算法的效率。

算法可以在并行环境上同时对多个家族的重复体进行扩展处理，这样将大大减少算法的运行时间。算法下一步的改进将是对其做并行化处理和提高算法对各种类型重复体的特异性。

# 第五章 结论与展望

随着人类基因组计划及其它的模式生物基因组计划的实施和研究的进一步深入，各种新的基因组序列不断被发现，基因组数据库也随之成倍的增长，海量的数据需要高效的算法来进行处理。对这些新发现的基因组序列进行分析并对其进行归类成为当前最为紧迫的任务之一。重复体识别就是对这些新发现的基因组进行分析的一个主要方法，通过重复体识别方法可以将基因组中存在的家族进行分类并将不同的重复体序列归类到各自的家族中。这对识别重复体家族的功能和预测家族的结构提供了非常重要的信息。虽然当前已经存在了很多重复体家族识别算法，比如基于已知的重复体数据库识别方法和基于最高相似对的方法，但是这些方法都有各自的不足，所以还需要改进现有的方法和探索新的识别方法。

本文首先介绍了生物信息学的研究内容和当前进展，并且介绍了重复体识别问题在基因组分析中的重要性。接着用数学方法定义了重复体识别问题，介绍了重复体分类。着重描述了重复体识别问题中牵扯到的局部序列比对，调和序列和识别算法结果的评判标准。将当前存在算法分为基于已知的重复体数据库算法，基于序列比对的识别算法和其它的识别算法，并各自举例说明了各种算法的特点。最后重点介绍了当前流行的基于高分相似对的 Single Linkage Clustering Algorithms 和 RECON 算法。

接着本文提出了两个基于基因组高频率种子序列扩展的重复体识别算法：RepeatSearcher 和 GSRSearcher。RepeatSearcher 算法也延续了比对的策略，通过两两局部序列比对构建多序列比对，逐步扩展调和序列和重复体序列来定义重复体家族各序列的精确边界并求出家族的调和序列。GSRSearcher 算法同样延续了高频率种子序列扩展的方法，但是扩展过程通过概率统计来判别，加上背景概率对重复体的影响，再结合空位罚分的策略不但使算法的精度有所提高而且降低了算法的运行时间。最后算法通过在人类 X 染色体，大鼠 X 染色体和小鼠 X 染色体[49][50][51]等 12 种哺乳动物基因组序列上进行了测试，结果表明本文提出的基于种子序列的算法是非常高效的算法。

虽然本文提出的算法在精度上和时间上都有所提高，但是算法还存在许多需要改进的地方。这两个算法都是可以通过并行化大大提高算法的运行时间。下一步的工作可以通过并行化改进这两个算法的运行效率，缩短运行时间，同时研究算法对各种类型重复体的特异性，提高算法灵敏度。

# 致谢

本论文是在我的导师霍红卫教授的悉心指导和严格要求下完成的，从论文的选题，资料的搜集，疑难问题的解答和论文的成型无不浸透着霍老师的心血和汗水。衷心感谢恩师霍红卫老师三年来在生活与学习中给予的无微不至的关怀，她严谨的治学态度，渊博的学识，丰富的实践经验和在科研中高瞻远瞩不断创新的精神使我终生受益。值此论文付梓之际，谨向霍老师表示我深深地敬意和由衷的感谢！

感谢我的师兄师姐：肖志伟、江裕民、陈昊、明华、康芬、林秋利，在生活中他们给我以关心和鼓励，在学习中，与他们的讨论，总能给我以指点。

感谢我的同门在学习和生活中的帮助。他们是：方义、程力行、姜飞翔、郭海涛、刘淼。

同时感谢我的师弟师妹：何华、王小武、崔强、谢巧娈、鱼桂娟，感谢他们在学习上对我的帮助。

特别感谢我的父母。他们无私的支持和鼓励，以及对我各方面的支持，才能使我有动力，有信心顺利完成我的学业。

最后，在此感谢所有帮助和支持过我的朋友们，谢谢你们。

# 参考文献

[1] 李越中，闫章才，高陪基。基因组研究与生物信息学. 2001. 山东大学出版社。

[2] 钟扬，张亮，赵琼. 简明生物信息学. 2001. 高等教育出版社。

[3] 李衍达，孙之荣. 生物信息学. 2000. 清华大学出版社。

[4] Doolittle, W.F. and Sapienza, C. Selfish genes, the phenotype paradigm and genome evolution. 1980. Nature 284: 601-603.

[5] Orgel, L.E. and Crick, F.H, Selfish DNA: The ultimate parasite. 1980. Nature 284:604-607.

[6] McClintock, B. The significance of responses of the genome to challenge. 1984 Science 226:792-801.

[7] Stein,L.D., Bao,Z., Blasiar,D., Blumenthal,T., Brent,M.R., Chen,N., Chinwalla,A., Clarke,L., Clee,C., Coghlan,A. et al. The genome sequence of C.briggsae: a platform for comparative genomics. 2003 PLoS Biol., 1, E45.

[8] International Human Genome Consortium. Initial sequencing and analysis of the human genome. 2001 Nature, 409: 860-921.

[9] Adams MD, celniker SE, Holt RA, Evans CA, Gocayne JD, Amanatides PG, Scherer SE, Li PW, Hoskins RA, Galle RF, etal: The genome sequence of Drosophila melanogaster,. 2000 Science, 287: 2185-2195.

[10] The Arabidopsis Genome Initiative: Analysis of the genome sequence of the flowering plant Arabidopsis thaliana. 2000 Nature, 408:796-815.

[11] Kazazian, H.H., Jr Mobile elements: drivers of genome evolution. 2004 Science, 303, 1626-1632.

[12] A.L. Price, N.C. Jones, and P.A.Pevzner. De novo identification of repeat families in large genomes. 2005 In Proc. Of the 13th International Conference on Intelligent Systems for Molecular Biology (ISMB'05), page To appear, Detroit, Michigan, AAAI press, Menlo Park. CA.

[13] Smit and P. Green. REPEATMASKER. Avaliable at http://www.repeatmasker.org/.

[14] Smith, T.F. and Waterman, M.S. Identification of common molecular subsequences. 1981 J.Mol.Biol. 147, 195-197.

[15] Bao, Z. and Eddy, S.R. Automated de novo identification of repeat sequence families in sequenced genomes. 2002 Genome Res., 12, 1269–1276.

[16]Bedell JA, Korf I, Gish W., Masker Aid: a performance. enhancement to RepeatMasker. 2000 Bioinfomatics, 16:1040-1041.

[17]Gish. WU-BLAST Avaliable at http://blast.wustl.edu/.

[18]Adams MD, celniker SE, Holt RA, Evans CA, Gocayne JD, Amanatides PG, Scherer SE, Li PW, Hoskins RA, Galle RF, etal: The genome sequence of Drosophila melanogaster, 2000 Science, 287: 2185-2195.

[19]Delcher AL, Kasif S, Fleischmann RD, Peterson J,White O,Salzberg., SL: Alignment of whole genomes. 1999 Nucleic Acids Res., 27: 2369-2376.

[20]Kurtz,S. Ohlebusch E, Schleiermacher C, Stoye J, Giegerich R. Computation and visualization of degenerate repeats in complete genomes. 2000 In Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology. Menlo Park, CA: AAAIPress, 228-238.

[21]Kurtz, S. and Schleiermacher, C. REPuter: fast computation of maximal repeats in complete genomes,. 1999 Bioinfrmatics, 15, 426-427.

[22]Agarwal, P. and States, D.J. The Repeat Pattern Toolkit (RPT): analyzing the structure and evolution of the C.elegans genome. 1994 In Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology(ISMB-94),AAAI Press, Stanford, CA,pp.1-9.

[23]Volfovsky, N., Haas, B.J. and Salzberg, S.L. A clustering method for repeat analysis in DNA sequences. 2001 Genome Biol., 2, RESEARCH0027.

[24]Pevzner, P.A., Tang, H. and tesler, G. De novo repeat classification and fragment assembly. 2004 Genome Res., 14, 1786-1796.

[25]Edgar, R.C. and Myers, E.W. PILER: identification and classification of genomic repeats. 2005 Appear in Proceedings of the Thirteenth International Conference on Intelligent Systems for Molecular Biology (ISMB-05), Detroit, Michigan.

[26]Jurka,J. RepBase Update: a database and an electronic journal of repetitive elements. 2000 Trends Genet. 9, 418-420.

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

[28]Lawrence CE, Altschul SF, Boguski MS, Liu JS, Neuwald AF, Wootton JC. . Detection subtle sequence signals: A Gibbs sampling strategy for multiple alignment. 1993 Science 262:208-214.

[29]Liu JS, Neuwald AF, Lawrence CE. .Bayesian models for multiple local sequence alignment and Gibbs sampling strategies. 1995 Journal of the American Statistical Association 90, 432:1156-1171.

[30]Dayhoff, M.O., Schwartz, R.M. and orcucc, B.C. In Atlas of Protein Sequence and Structure, 1978 vol.5, suppl.3 (Dayhoff, M.O., Ed.). Pp345-352. NBRF, Washington.

[31]Hcnikoff, S. and Henikooff, J.G. 1992 Proc. Natl. Acad. Sci. USA 89, 10915-10919.

[32]樊龙江. 生物信息札记. 2001. 浙江：浙江大学生物信息学研究所.

[33]Wiliam H.E.day, F.R.McMorris. Analysing molecular sequences using consensus. New Zealand Journal of Botany, 1993, Vol. 31:211-218.

[34]D.Gusfield. Algorithms on Strings, Trees, and Sequences: 1997 Computer Science and Computational Biology. Cambridge University Press,.

[35]Julie D.Thompson, Desmond G.Higgins and Toby J.Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalities and weight matrix choice. 1994. Nucleic Acids Research, Vol 22. No. 22 4673-4680.

[36]Tompa, M. An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. 1999 In Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology (ISMB), pages 262-271.

[37]Brazma, A., Jonassen, I., Eidhammer, I., and Gilbert, D. Approaches to the automatic discovery of patterns in biosequences. 1998 Journal of Computational Biology, 5(2):279-305.

[38]Natalia Volfovsky, Brian J Haas and Steven L Salzberg., A clustering method for repeat analysis in DNA sequences. 2001 Genome Biology.

[39]Skiena, S.S. The algorithm design manual. 1997. Telos/Springer-Verlagg, New York.

[40]Jie Zheng, Stefano Lonardi. Discovery of Repetitive Pattrens in DNA with Accurate Boundaries. 2005 Proc. of IEEE International Symposium on BioInformatics and BioEngineering (BIBE'05), pp. 105-112, Minneapolis, Minnesota, USA, 2005.

[41]E.Vidal, A. Marzal, and P.Aibar. Fast computation of normalized edit distances. 1995 IEEE Trans. on PAMI, 17:899-902, 1995.

[42]Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W. & Lipman, D. J. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. 1997. Nucleic Acids Res 25, 3389-3402

[43]Waterman, M.S. Introduction to Computational Biology: Sequences, Maps and

Genomes. 1995 Chapman and Hall/CRC Press, Boca Raton, FL.

[44] Stein,L.D., Bao,Z., Blasiar,D., Blumenthal,T., Brent,M.R., Chen,N., Chinwalla,A., Clarke,L., Clee,C., Coghlan,A. et al. The genome sequence of C.briggsae: a platform for comparative genomics. 2003. PLoS Biol., 1, E45.

[45] Lawrence, C.E. and Reilly, A.A. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. 1990 Proteins, 7(1):41-51.

[46] Liu JS, Neuwald AF, Lawrence CE. .Bayesian models for multiple local sequence alignment and Gibbs sampling strategies. 1995 Journal of the American Statistical Association 90, 432:1156-1171.

[47] Henikoff S and Henikoff J., Position-based sequence weights. J 1994 Molecular Biology, 243: 574-578.

[48] Heinikoff J and Henikoff S., Using substitution probabilities to improve position-specific scoring matrices. 1996 CABIOS, 12:135-143.

[49] Bourque, G., Pevzner, P.A. and Tesler, G. Reconstructing the genomic architecture of ancestral mammals: lessons from human, mouse, and rat genomes. 2004 Genome Res., 14, 507-516.

[50] Wasserman, W.W., Palumbo, M., Thompson, W., Fickett, J.W. and Lawrence, C.E. Human-mouse genome comparisons to locate regulatory sites. 2000 Nature Genet.26, 225-228.

[51] Rat Genome Sequencing Project Consortium. Genome sequence of the Brown Norway rat yields insights into mammalian evolution. 2004 Nature 420,493-521.

# 研究成果

白帆，霍红卫，《一种具有精确边界的重复体识别算法》，西安电子科技大学2006年学术年会