第二章　快速比对方法

§2.1　同源序列查询和数据库搜索

为了了解一个新基因的功能，生物学家可以选择生物实验的途径或者同源序列搜索方法。一般来言，生物实验手段得到的结论更为可靠，但是实验手段具有以下困难。第一，一些生物像深海细菌在实验室里难以培植。第二，一些生物例如大猩猩资源稀少，从而实验费用昂贵。第三，由于伦理道德问题，许多实验禁止在人体上进行。由于湿实验方法的这些困难，研究人员通常先将所得到的 DNA 序列和模式生物的基因序列加以比较，并通过模式生物中相似 DNA 序列的已知功能来推断新得到的 DNA 序列的功能。模式生物有大肠杆菌 (*Escherichia coli*)、酵母菌 (*Saccharomyces cerevisiae*)、线虫 (*Caenorhabditis elegans*)、果蝇 (*Drosophila melanogaster*)、拟南芥 (*Arabidopsis thaliana*)、斑马鱼 (*Danio rerio*)、和小鼠 (*Mus musculus*)。目前，对这类模式生物基因组的测序已经完成。通过一个多世纪的生物学研究，模式生物中的大量基因以及它们对应的蛋白质的功能也已经比较清楚。

由于所有生命都是由一个祖先进化而来，DNA 分子无论其来源如何往往在一个甚至几个模式生物基因组里都有高度相似的同源 DNA。两个哺乳动物有 99% 的基因相同。甚至人和果蝇都有 50% 的基因相同！这里，我们称两个 DNA 分子同源是指它们是一个祖先序列的两个后代。由于自然选择的压力，同源 DNA 有相似的序列和类似的生物功能。

理解癌症的机理是同源序列搜索方法获得成功的一个典范。早在 1983 年，发表在科学杂志上的一篇论文报道了有 28 个氨基酸的血小板衍生生长因子 (platelet derived growth factor) 序列。血小板衍生生长因子是刺激细胞生长的蛋白质。将这个蛋白序列和其它蛋白序列比较时，生化学家 Doolittle 意外地发现它和来自猿猴的病毒肿瘤基因 v-sis 的序列几乎一样。这个发现在当时改变了人们对肿瘤形成的认识。今天，人们已经普遍接受癌症是生长因子蛋白质在错误时间表达的结果。

当计算分别具有 m 和 n 个字符的两条序列之间的最优局部比对时，Smith-Waterman 算法需要运行 $O(mn)$ 次算术运算。人和小鼠基因组都有大约 30 亿个核苷酸碱基。使用 Smith-Waterman 算法寻找这两个物种之间的同源序列需要几个月的时间。由于测序技术的发展，DNA 序列的数量正以前所未有的速度增长。目前，NCBI 所拥有的 DNA 和蛋白质序列数据库已有千万条序列，总长度已多达 1.6×10^{14} 个字符。因此，在序列数据库中查询同源序列需要具有线性时间复杂度或者更快的方法。

在这一章的其余小节里，我们将介绍设计快速比对方法所需要的基本知识和常用的同源序列搜索程序。俗话说，世界上没有免费的午餐。为了大幅度提高序列比对的速度，人们不得不牺牲其它一些性能。评估一个同源搜索

程序的性能通常有三个指标：速度 (speed)、灵敏度 (sensitivity) 和选择性 (selectivity)。

假设一个序列，X，在一个含有 k 条序列的数据库，D，中有 a 个同源序列，其它的 $k-a$ 个序列和 X 没有同源关系。一个同源查询程序在 D 里找到 b 个与 X 同源的序列，其中 b' 个是真同源序列。那么，在这个例子中，这个程序的真阳性率是 b'/a；假阳性率是 $(b-b')/(k-a)$。

同源查询程序的灵敏度定义为真阳性率，表示真正的同源序列中有多大的比例可以被程序识别。假阳性率表示一个与查询序列无关的序列被程序认定为非同源序列的几率。同源查询程序的选择性定义为假阳性率的倒数。显然，灵敏度和选择性之间有一种消长关系。一个同源查询程序可以通过输出更少的序列来提高选择性；但同时灵敏度一般也会随之下降。反过来，一个同源查询程序可以通过输出尽可能多的序列来提高其灵敏度，但这样会导致选择性降低。换句话说，设计同源查询程序的关键在于如何平衡其灵敏度和选择性。

要达到灵敏度和选择性之间的最佳平衡点需要使用统计理论定制合适的打分矩阵，还需要将序列比对的得分转化为对实验人员有直观意义的数值，以便使用者能够简单地设置输出结果的得分阈值。关于如何制定更好的打分矩阵，我们在介绍 BLOSUM 矩阵的时候已经有一些初步的了解。而对于比对得分的转化，我们将在本章的第 §2.6.2 节进一步介绍。

与速度进行交换的性能指标一般是灵敏度。直观上讲，对于任何一条 DNA 或蛋白质序列，总有一些和它同源的序列容易被找到；但我们需要花费更长的时间才能找到另外一些同源序列。如果我们不要求一个方法把所有的同源序列都找出 (即放弃一定的灵敏度)，那么我们就有可能通过只识别那些容易被找到的同源序列来大大地提高其速度。当然，在这个速度和灵敏度的取舍过程中，我们希望尽量少地降低灵敏度而同时把速度提的尽量高。

§2.2 序列中的字分布

DNA 分子是由A、G、C、T组成的长链。在分析一个 DNA 分子时，最自然的问题莫过于这四个核苷酸在其中分布的状况。在细胞核里，DNA 以双链形式存在。一个链上的C的碱基和另一个链上的某个G的碱基被氢键结合形成一个碱基对，而A 的碱基和对应的T 的碱基也被氢键结合在一起形成一个碱基对。所以，一个 DNA 分子中C 出现的次数和G 出现的次数总是相等，而A 出现的次数和T 出现的次数也相等。因此，生物学家早就使用G+C 含量来刻划 DNA 分子。一个 DNA 分子的G+C 含量通常用一个百分比来表示，等于一个单链上C 和G 出现的总次数除以这个 DNA 分子中所含的碱基对的个数。例如，下列 DNA 分子：

5′-GAAGGACGCGAAACGCTTTG-3′
3′-CTTCCTGCGCTTTGCGAAAC-5′

含有 20 个碱基对；C和G在每个单链上总共出现 11 次。所以，它的G+C含量是 55%。等价地，其A+G含量则是 1 – 55% = 45%。由于C和G的碱基被三个氢键结合在一起，而A 和T 的碱基被两个氢键结合在一起，DNA 分子的解链温度和它的G+C含量成正相关。在分子生物学的早期还没有发明 DNA 测序技术之前，生物学界正是通过测量一个 DNA 序列的解链温度来估计它的G+C 含量。

基因组的 G+C 含量一般介于 20% 和 75% 之间。酵母菌、大肠杆菌、拟南芥、以及人基因组的G+C含量分别是 38%、36%、35%、和 41%。天蓝色链霉菌 (*Streptomyces coelicolor* A3(2),) 的G+C 含量高达 72%，而镰状疟原虫 (*Plasmodium falciparum*) 的G+C 含量仅有 20%。相对于整个基因组的G+C含量，基因序列的G+C 含量一般来讲略高一些。

一个 DNA 单链上相邻的两个核苷酸称为一个双核苷酸 (dinucleotide)。四个不同的碱基一共组成 16 个双核苷酸：AA，AC，…，TT。一个 DNA 单链分子中双核苷酸的成分被认为和它的空间弯曲形状有关，从而也和基因表达相关。

每三个碱基组成一个密码子。61 个密码子对应 20 个氨基酸；剩余的三个对应终止密码子。一个基因序列中密码子出现的频率决定与其对应的蛋白质的某些生物功能。

生物学家也对更长的碱基字的分布感兴趣。例如，限制性内切酶 AluI 的识别序列是 5′-AGCT-3′. 因此，一个识别序列在一个基因组中的分布决定了酶切实验中收获的酶切片段长度的分布。

为了叙述方便起见，含有 k 个字符的序列段被称为 k- 字。短字的分布还可以用来研究衡量 DNA 序列之间的相似性的非比对方法。在后面的章节里，我们将会看到，选取搜索核也需要考虑字的分布。在考察 DNA 序列里的字分布时，我们经常感兴趣的是是否某个字出现的次数超乎寻常的多或者超乎寻常的少。一个字在统计分布上的特殊性往往意味着它是某种生物功能元素。要回答诸如此类的问题，我们需要对序列建立数学模型。在下面两小节中，我们分别介绍两个常用的 DNA 序列模型。

§2.2.1 DNA 序列的随机模型 I: 一致独立分布

由于 DNA 分子的双链是互补的，我们暂且把 DNA 分子看作一个单链序列。如果我们将 DNA 序列看成是按某种模型产生的随机序列，最简单的模型假设四种核苷酸在各个位置上分布一致而且独立。换句话讲，如果把 DNA 序列上的核苷酸想象成由一部机器一一生成，在产生第 k 个核苷酸时机器的运作和在产生第一个核苷酸时的运作没有什么两样，并且和前面已生成的 $k-1$ 个核苷酸也没有任何关系。

假设 $X = X_1X_2\cdots X_m$ 是一个含 m 个字符的随机序列。X_i 可以是A、G、C、T中的任何一个。换句话说，X_i 是一个随机变量。令 X_i 是A、G、C、T 的概率分别

为 p_A、p_G、p_C、p_T。我们有:

$$p_x \geq 0, \quad \text{满足} p_A + p_G + p_C + p_T = 1, \ x \in \{A, G, C, T\}。$$

$\{p_A, p_G, p_C, p_T\}$ 称为 X_i 的概率分布。在一致独立分布的模型下，所有的 X_i 具有相同的概率分布。

在这本书中，对于一个随机变量 V, 我们用 $\Pr[V = v]$ 表示 V 的值是 v 的概率; 用 $\Pr[E]$ 表示事件 E 发生的概率。

考虑一个 k- 字, $W = w_1 w_2 \cdots w_k$, $k \leqslant m$. 在分析它在一条 DNA 序列中的分布时，我们感兴趣下列问题：

(1) W 在随机序列 X 上某一个特定位置上出现的概率是多少?

(2) W 在随机序列 X 上出现的概率是多少?

(3) W 在随机序列 X 上出现次数的期望值是多少?

首先，我们回答第一个问题。我们称 W 在 X 上的第 i 个位置上出现如果

$$X_{i-k+1} = w_1, \ X_{i-k+2} = w_2, \cdots, X_i = w_k。 \tag{2.1}$$

这里我们以结束的位置作为 W 出现的位置。在这个约定下，因为没有足够的字符，W 不可能在前 $k-1$ 个位置上出现。令 E_i 表示 W 在第 i 个位置上出现这一事件。因为 X_{i-j+1} $(1 \leq j \leq k)$ 是具有相同分布的独立随机变量，并且 $\Pr[X_{i-j+1} = w_{k-j+1}] = p_{w_{k-j+1}}$，我们有:

$$\Pr[E_i] = \begin{cases} 0 & i < k, \\ p_{w_1} p_{w_2} \cdots p_{w_k} & k \leq i \leq m。 \end{cases} \tag{2.2}$$

上述第二个问题的解要复杂的多。首先，我们复习一些有关离散概率的几个基本概念。 考虑一个离散概率空间，$\mathcal{X}$，其中每一个元素 x 都有一个概率 $\Pr[x]$，其值满足 $\sum_{x \in \mathcal{X}} \Pr[x] = 1$。我们不妨假设 $\mathcal{X}$ 是有限空间。它的子集称为事件。一个事件 E 发生的概率等于 $\Pr[E] = \sum_{x \in E} \Pr[x]$。两个事件，$E$ 和 F，共同发生的概率叫做它们的联合概率，记作 $\Pr[EF]$。如果 E 和 F 是两独立事件，$\Pr[EF] = \Pr[E]\Pr[F]$。

W 在 X 上出现的概率可以表示为 $\Pr[\cup_{j=k}^{m} E_j]$。当 $|j - j'| \leq k$, E_j 和 $E_{j'}$ 两事件一般不独立。例如，如果 $W = \mathsf{ACA}$, $\Pr[E_3 E_4] = 0 \neq \Pr[E_3] \times \Pr[E_4]$，隐含 E_3 和 E_4 不独立。因此，W 在 X 上出现的概率和 W 的结构有关，没有一个简单的表达式。但是，我们可以建立一个递归公式来计算这个概率。即使对 $\overbrace{\mathsf{AA}\cdots\mathsf{A}}^{m}$，相应的递归公式也比较繁琐 (Chao & Zhang, 2009, 第 94 页)。我们在此不进行深入的讨论。另外，我们还可以设计一个动态规划算法来计算这个概率 (参考 §2.7.2)。

最后，我们回答第三个问题。定义随机变量 $I_1, I_2, \cdots, I_m$ 如下：

$$I_j = \begin{cases} 1 & \text{如果事件 } E_j \text{ 发生,} \\ 0 & \text{如果事件 } E_j \text{ 不发生。} \end{cases} \tag{2.3}$$

那么，W 在随机序列 X 上出现的次数，N, 是一个随机变量。它等于 0-1 随机变量 $I_j (1 \le j \le m)$ 的和, 即

$$N = I_1 + I_2 + \cdots + I_m。 \tag{2.4}$$

根据期望函数的线性性质，一组随机变量和的期望值等于这些随机变量期望值的和。所以，从公式 (2.2) 得出：

$$E[I_j] = \Pr[I_j = 1] \times 1 + \Pr[I_j = 0] \times 0 = \Pr[E_j] = 0,\ 1 \le j \le k-1,$$
$$E[I_j] = \Pr[E_j] = p_{w_1} p_{w_2} \cdots p_{w_k},\ k \le j \le m,$$

以及

$$E[N] = E[I_1] + E[I_2] + \cdots + E[I_m] = (m-k+1) p_{w_1} p_{w_2} \cdots p_{w_k}。 \tag{2.5}$$

公式 (2.5) 可以用来考察特定短字在基因组里的分布情况。GCTGGTGG叫做 Chi- 序列。在细菌基因组里, Chi- 序列所在位置是同源重组事件多发区域。大肠杆菌 K-12 基因组 (NC_000913.3) 有 4,091,840 个核苷酸, 其中 A、G、C、T 出现的频率分别是 24.11%、27.30%、24.55%、24.04%。在大肠杆菌 K-12 基因组里, Chi- 序列实际出现高达 714 次，虽然它的出现次数的期望值仅是：

$$(4,091,840 - 8 + 1) \times 0.2730^5 \times 0.2455 \times 0.2404^2 \approx 88。$$

和 Chi- 序列相反，字CATG出现次数的期望值是 15,895，但它实际出现次数仅为 1,234。

利用双核苷酸出现的次数，我们还可以检验一致独立序列模型是否适合一条长为 n 的 DNA 序列。假设这条 DNA 序列上核苷酸的频率是 $\{f_A, f_G, f_C, f_T\}$。考虑一个双核苷酸，$D = h_1 h_2$。令 D 在给定的序列中出现的总次数是 O_D。在一致独立的模型下，D 出现的次数的期望值是

$$E_D = (n-1) p_{h_1} p_{h_2}。$$

再令

$$c_D = \begin{cases} 1 + 2p_{h_1} - 3p_{h_1}^2, & \text{如果 } h_1 = h_2, \\ 1 - 3p_{h_1} p_{h_2}, & \text{如果 } h_1 \ne h_2。 \end{cases}$$

如果

$$\frac{(O_D - E_D)^2}{c_D E_D} > 3.84 \tag{2.6}$$

核苷酸的概率分布为 $\{f_A, f_G, f_C, f_T\}$ 的一致独立序列模型不适合描述给定的 DNA 序列。这个统计检验的合理性超出本书的范围，不在此叙述。

§2.2.2 DNA 序列的随机模型 II: 马尔科夫链

下列 DNA 是大肠杆菌 K-12 基因组 (MG1655) 中的一段序列：

```
ATGCGAGTGTTGAAGTTCGGCGGTACATCAGTGGCAAATGCAGAACGTTTTCTGCGTGTTGCCGATATTC
TGGAAAGCAATGCCAGGCAGGGGCAGGTGGCCACCGTCCTCTCTGCCCCCGCCAAAATCACCAACCACCT
GGTGGCGATGATTGAAAAAACCATTAGCGGCCAGGATGCTTTACCCAATATCAGCGATGCCGAACGTATT
TTTGCCGAACTTTTGACGGGACTCGCCGCCGCCCAGCCGGGGTTCCCGCTGGCGCAATTGAAAACTTTCG
TCGATCAGGAATTTGCCCAAATAAAACATGTCCTGCATGGCATTAGTTTGTTGGGGCAGTGCCCGGATAG
CATCAACGCTGCGCTGATTTGCCGTGGCGAGAAAATGTCGATCGCCATTATGGCCGGCGTATTAGAAGCG
CGCGGTCACAACGTTACTGTTATCGATCCGGTCGAAAAACTGCTGGCAGTGGGGCATTACCTCGAATCTA
CCGTCGATATTGCTGAGTCCACCCGCCGTATTGCGGCAAGCCGCATTCCGGCTGATCACATGGTGCTGAT
GGCAGGTTTCACCGCCGGTAATGAAAAAGGCGAACTGGTGGTGCTTGGACGCAACGGTTCCGACTACTCT
GCTGCGGTGCTGGCTGCCTGTTTACGCGCCGATTGTTGCGAGATTTGGACGGACGTTGACGGGGTCTATA
CCTGCGACCCGCGTCAGGTGCCCGATGCGAGGTTGTTGAAGTCGATGTCCTACCAGGAAGCGATGGAGCT
TTCCTACTTCGGCGCTAAAGTTCTTCACCCCCGCACCATTACCCCCATCGCCCAGTTCCAGATCCCTTGC
CTGATTAAAAATACCGGAAATCCTCAAGCACCAGGTACGCTCATTGGTGCCAGCCGTGATGAAGACGAAT
TACCGGTCAAGGGCATTTCCAATCTGAATAACATGGCAATGTTCAGCGTTTCTGGTCCGGGGATGAAAGG
GATGGTCGGCATGGCGGCGCGCGTCTTTGCAGCGATGTCACGCGCCCGTATTTCCGTGGTGCTGATTACG
CAATCATCTTCCGAATACAGCATCAGTTTCTGCGTTCCACAAAGCGACTGTGTGCGAGCTGAACGGGCAA
TGCAGGAAGAGTTCTACCTGGAACTGAAAGAAGGCTTACTGGAGCCGCTGGCAGTGACGGAACGGCTGGC
CATTATCTCGGTGGTAGGTGATGGTATGCGCACCTTGCGTGGGATCTCGGCGAAATTCTTTGCCGCACTG
GCCCGCGCCAATATCAACATTGTCGCCATTGCTCAGGGATCTTCTGAACGCTCAATCTCTGTCGTGGTAA
ATAACGATGATGCGACCACTGGCGTGCGCGTTACTCATCAGATGCTGTTCAATACCGATCAGGTTATCGA
AGTGTTTGTGATTGGCGTCGGTGGCGTTGGCGGTGCGCTGCTGGAGCAACTGAAGCGTCAGCAAAGCTGG
CTGAAGAATAAACATATCGACTTACGTGTCTGCGGTGTTGCCAACTCGAAGGCTCTGCTCACCAATGTAC
ATGGCCTTAATCTGGAAAACTGGCAGGAAGAACTGGCGCAAGCCAAAGAGCCGTTTAATCTCGGGCGCTT
AATTCGCCTCGTGAAAGAATATCATCTGCTGAACCCGGTCATTGTTGACTGCACTTCCAGCCAGGCAGTG
GCGGATCAATATGCCGACTTCCTGCGCGAAGGTTTCCACGTTGTCACGCCGAACAAAAAGGCCAACACCT
CGTCGATGGATTACTACCATCAGTTGCGTTATGCGGCGGAAAAATCGCGGCGTAAATTCCTCTATGACAC
CAACGTTGGGGCTGGATTACCGGTTATTGAGAACCTGCAAAATCTGCTCAATGCAGGTGATGAATTGATG
AAGTTCTCCGGCATTCTTTCTGGTTCGCTTTCTTATATCTTCGGCAAGTTAGACGAAGGCATGAGTTTCT
CCGAGGCGACCACGCTGGCGCGGGAAATGGGTTATACCGAACCGGACCCGCGAGATGATCTTTCTGGTAT
GGATGTGGCGCGTAAACTATTGATTCTCGCTCGTGAAACGGGACGTGAACTGGAGCTGGCGGATATTGAA
ATTGAACCTGTGCTGCCCGCAGAGTTTAACGCCGAGGGTGATGTTGCCGCTTTTATGGCGAATCTGTCAC
AACTCGACGATCTCTTTGCCGCGCGCGTGGCGAAGGCCCGTGATGAAGGAAAAGTTTTGCGCTATGTTGG
CAATATTGATGAAGATGGCGTCTGCCGCGTGAAGATTGCCGAAGTGGATGGTAATGATCCGCTGTTCAAA
GTGAAAAATGGCGAAAACGCCCTGGCCTTCTATAGCCACTATTATCAGCCGCTGCCGTTGGTACTGCGCG
GATATGGTGCGGGCAATGACGTTACAGCTGCCGGTGTCTTTGCTGATCTGCTACGTACCCTCTCATGGAA
GTTAGGAGTCTGA
```

这段 DNA 序列含有 2463 个核苷酸。A、G、C、T出现的频率分别是 22.45%、28.10%、24.97%、24.48%。双核苷酸AA、AG、AC、AT分别出现 177、102、113、161 次。所以，在A出现的下一个位置上，A、G、C、T的分布等于对应核苷酸对的出现次数除以 A 出现的次数，即 32.01%、18.45%、20.43%、29.11%。这组核苷酸对的分布显然和一致独立分布的模型不符。事实上，双核苷酸AA就满足统计检验不等式 (2.6)。所以说，描述 DNA 序列需要更强大的随机模型。

描述 DNA 序列的马尔科夫链是一列定义在 $\Sigma = \{A, G, C, T\}$ 上的随机变量 $X_1, X_2, \cdots, X_n, \cdots$，这里 X_j 的状态表示 DNA 序列第 j 个位置上的核苷酸。对于任何的 $n > 1$，在给定 $X_1, X_2, \cdots, X_{n-1}$ 的状态的前提下，X_n 的状态仅和 X_{n-1} 的状态有关，与位置 n 无关。 这一性质叫马尔科夫性质。

为了准确地描述马尔科夫链模型，我们先引进条件概率。 在事件 E 发生的条件下，事件 F 发生的条件概率，$\Pr[F \mid E]$，等于：

$$\Pr[F \mid E] = \frac{\Pr[EF]}{\Pr[E]},$$

这里 $\Pr[E] > 0$。如果 $\Pr[E] = 0$，我们约定 $\Pr[F \mid E] = 0$。如果 E 和 F 是两独立事件，$\Pr[F \mid E] = \Pr[F]$。

假设事件 $E_1, E_2, \ldots, E_k$ 满足下列两个条件：

(a) $E_i \cap E_j = \phi$ 对于任何 $i \neq j$;

(b) $E_1 \cup E_2 \cup \cdots \cup E_k = \mathcal{X}$。

则对任何事件 F,

$$\Pr[F] = \sum_{i=1}^{k} \Pr[FE_i] = \sum_{i=1}^{k} \Pr[F \mid E_i] \Pr[E_i]。 \tag{2.7}$$

这个公式称为全概率公式。

重新回到描述 DNA 序列的马尔科夫链。精确地讲，马尔科夫性质定义为：

$$\Pr[X_{j+1} = x_{j+1} \mid X_j = x_j, \cdots, X_1 = x_1] = \Pr[X_{j+1} = x_{j+1} \mid X_j = x_j], \tag{2.8}$$

$$\Pr[X_{j+1} = a \mid X_j = b] = \Pr[X_2 = a \mid X_1 = b], \tag{2.9}$$

其中 $x_1, x_2, \ldots, x_{j+1}, a, b$ 都属于核苷酸集合 $\{A, G, C, T\}$。公式 (2.8) 是指知道当前状态马尔科夫链的将来状态和过去历史无关。而公式 (2.9) 指给定当前的状态，下个时间点状态的条件概率和当前的时间点无关。也就是说，马尔科夫链具有 “一致性” 和 “ 无记忆” 两个重要性质。

令 $p_{ba} = \Pr[X_2 = a | X_1 = b]$, $a, b \in \{A, G, C, T\}$。由 p_{ba} 组成的矩阵，

$$P = \begin{pmatrix} p_{AA} & p_{AG} & p_{AC} & p_{AT} \\ p_{GA} & p_{GG} & p_{GC} & p_{GT} \\ p_{CA} & p_{CG} & p_{CC} & p_{CT} \\ p_{TA} & p_{TG} & p_{TC} & p_{TT} \end{pmatrix}, \tag{2.10}$$

叫做马尔科夫链的转移矩阵。

由于马尔科夫链的一致性，转移矩阵告诉我们一个位置上的核苷酸对于前一个位置上核苷酸的依赖关系。如果转移矩阵中的各行相同，那么核苷酸在一个位置上的概率分布和前一个位置的状态无关，并且各个位置上的概率分布相同。这也就是说，前面介绍的 DNA 序列的一致独立分布模型是马尔科夫链的一个特例。

知道了一个位置上的核苷酸，转移矩阵里和这个核苷酸对应的行给出了下一个位置上核苷酸的概率分布。但是，第一个位置上核苷酸的概率分布是

一个未知数。为了建立一个 DNA 序列的马尔科夫链模型，我们不仅需要给出转移矩阵，而且还要给出它的初始概率分布：

$$\pi = \{\pi_A, \pi_G, \pi_C, \pi_T\}.$$

根据全概率公式 (2.7)，第一位置上的概率分布可以从初始概率分布计算得到：

$$\begin{aligned} \Pr[X_1 = x] &= \sum_{y\in\Sigma} \Pr[X_0 = y, X_1 = x] \\ &= \sum_{y\in\Sigma} \Pr[X_0 = y] \Pr[X_1 = x | X_0 = y] \\ &= \sum_{y\in\Sigma} \pi_y p_{yx}, \end{aligned}$$

等价地，

$$(\Pr[X_1 = A], \Pr[X_1 = G], \Pr[X_1 = C], \Pr[X_1 = T]) = \pi \times P。$$

使用归纳法，我们还可以得出第 $n > 1$ 个位置上的概率分布：

$$(\Pr[X_n = A], \Pr[X_n = G], \Pr[X_n = C], \Pr[X_n = T]) = \pi \times P^n。 \tag{2.11}$$

一个完整的马尔科夫链模型有初始状态分布和转移矩阵两部分。描述本小节开始给出的 DNA 序列的马尔科夫链 $\mathcal{M}$ 的初始分布应为核苷酸的出现频率。$\mathcal{M}$ 转移矩阵是：

$$P = \begin{pmatrix} 0.3201 & 0.1845 & 0.2043 & 0.2911 \\ 0.2312 & 0.2457 & 0.3179 & 0.2052 \\ 0.2114 & 0.3236 & 0.2325 & 0.2335 \\ 0.1426 & 0.3665 & 0.2305 & 0.2604 \end{pmatrix}, \tag{2.12}$$

其中核苷酸 x 对应的行等于在 x 之后 A、G、C、T 出现的频率。

作为马尔科夫链模型的一部分，初始状态分布的选取必然对模型的精确性有影响。幸运的是其影响有限，尤其是对 DNA 的马尔科夫模型。当马尔科夫链满足一些简单性质时，在一个位置上的核苷酸状态分布快速地收敛到下列线性系统的解向量 $\psi = (\psi_A, \psi_G, \psi_C, \psi_D)$：

$$\psi = \psi P。$$

ψ 叫做该马尔科夫链的 “平稳分布”。基于初始状态分布影响的有限性以及平稳分布（如果存在）非常接近各个状态在整个所描述序列中出现的频率，我们往往把一个 DNA 序列中核苷酸的出现频率作为该 DNA 序列的马尔科夫模型的初始核苷酸分布。

在基因序列中，密码子中的第三个位置上的核苷酸分布和前两个位置上的核苷酸高度关联。所以，描述基因编码序列需要使用二阶马尔科夫链。$n \geq 1$

阶马尔科夫性质定义为：

$$\begin{aligned} & \Pr[X_{j+1} = x_{j+1} | X_j = x_j, \cdots, X_2 = x_2, X_1 = x_1] \\ = \quad & \Pr[X_{j+1} = x_{j+1} | X_j = x_j, X_{j-1} = x_{j-1}, \cdots, X_{j-n+1} = x_{j-n+1}]。 \end{aligned} \quad (2.13)$$

其中 $x_1, x_2, \ldots, x_{j+1}$ 都属于 $\{\mathsf{A}, \mathsf{G}, \mathsf{C}, \mathsf{T}\}$。高阶马尔科夫链可以看作是在马尔科夫链模型中加入了记忆；从而提高了模型的功能。但它的弱点是需要更多的数据来估计这类模型的转移矩阵。例如，DNA 的二阶马尔科夫链模型的转移矩阵有多达 64 个元素。

§2.3 字匹配的散列表方法

散列表 (hash table) 是在计算机各个领域都经常用到的一个数据结构。假设有 n 个对象 $o_1, \ldots, o_n$，每个对象有一个检索关键字 k_i。散列表提供了从检索关键字 k_i 到对象 o_i 的平均时间为 $O(1)$ 的快速检索。

散列表有几种实现的方式，无论哪种实现方式，都需要一个事先选定的散列函数 h。给定任何一个检索关键字 k，$h(k)$ 返回一个从 0 到 $N-1$ 的整数值。有了散列函数 h，就可以将所有 $h(k_i)$ 相同的那些对象 O_i 放在同一个队列里，并在一个长度为 N 的数组 M 中将这个队列的头记录在 $M[h(k_i)]$ 位置。在本书中，我们将 M 称为主表。

这样，在使用一个检索关键字 k 检索时，就可以在计算散列函数 $h(k)$ 之后从 $M[h(k)]$ 中取出相应队列的头。如果是查找，就可以遍历此队列中的所有的对象 o 来找到散列表中具有相同关键字 k 的那一个或多个对象。如果是插入，则可以简单的把新对象加到队列的头上。在绝大多数应用中，散列函数的选择都会使得 $N = O(n)$。图 2.1 给出了这样一个散列表的示意图。

下面我们简单分析一下这样一个散列表的时间和空间复杂度。首先选择常数 $c > 1$ 并令 $N = c \cdot n$。这样的话，容易看出，散列表的空间复杂度由主表和所有的队列构成。由于每一个 o_i 在所有的队列中只出现一次，所有队列的总长度刚好等于 n，而主表的长度为 $N = O(n)$。所以，这样一个散列表的空间复杂度为 $O(n)$。

而每次检索的时间不超过一个队列的长度。因为 $c > 1$，只要我们选择的散列函数足够好，且所有的对象的关键字互不相同的条件下，容易证明队列的平均长度为 $O(1)$。所以每次检索只需要 $O(1)$ 的时间就能够完成了。

在另外一种常见的散列表实现方式中，具有相同散列值的对象（称为*碰撞*）不放在一个队列中，而是直接储存在主表里。对于一个对象 o 和它的关键字 k，散列表的构建算法会首先尝试在 $M[h(k)]$ 处记录这个对象 o。但是如果此处已经被另一个对象占用，则构建算法会使用一系列的散列函数 $h_i(k)$ 来寻找下一处可以储存这个对象的空位置。通常情况下，人们会选择素数 p 并使用 $h_i(k) = (h(k) + i \cdot p) \bmod N$ 这样一个系列。而在检索时，也采用同样的

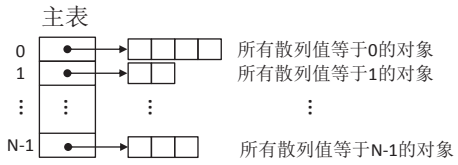

图 2.1: 一个基于队列的散列表结构。

顺序来遍历这些有碰撞的对象。同样，在 $c > 1$ 的情况下，也可以证明这样一个散列表的检索只需要 $O(1)$ 的时间。

在不同的应用中，散列表起到的地位和作用也不同，也会有大同小异的实现方式。在本书里，我们介绍的重点放在散列表在字符串匹配中的使用。

设想我们需要索引一个很长的文本 T，使得对每一个新给出的长度为 ℓ 的字符串 P，能够迅速的查找 P 在 T 中的精确出现位置。一个办法就是把 T 中每个位置的长度为 ℓ 的子字符串放入一个散列表。散列表中的每个对象是位置 i，每个对象的关键字是位置 i 处长度为 ℓ 的子字符串。而且我们要求这个散列表是用前述的队列方式解决碰撞问题的。

这样，在查找 P 时，我们只需要计算 P 的散列值，并在散列表中返回那个相应散列值下的位置队列就可以了。

事实上，即便待查询的字符串 P 的长度超过了 ℓ，我们仍然可以用上面的散列表加速查询算法。具体方法是，使用 P 的一个长度为 ℓ 的子字符串 P' 进行查询，找到所有 P' 在 T 中精确出现的位置后，遍历这些位置，看哪些位置给出了 P 的精确匹配。这样一来，P' 的所有匹配位置（也就是散列表中对应于 P' 的散列值的那个队列）平均只需要 $O(1)$ 时间就能得到，这样就省去了对文本中那些不匹配的位置的检查，大大提高了查询的速度。

仅仅就对于要找 P 的精确匹配而言，存在更高效的数据结构（譬如后缀数组）来索引文本。但是，上述使用散列表的办法有一个其它数据结构不可比拟的好处，那就是，即便我们要找的是 P 在 T 中的近似匹配，上述办法仍然可以起到一定的作用。具体的办法是，我们对 P 的每一个长度为 ℓ 的子字符串都进行上述办法的查询。这样一来，P 和它在 T 中的近似匹配共享了一个长度为 ℓ 的子字符串，那么上述办法就可以把这个近似匹配找出来。事实上，这正好就是在 DNA 序列的近似匹配中常用的策略。

除了散列表，还有若干种可以提供严格字符串匹配的数据结构。常见的包括字符树 (trie tree)、后缀树 (suffix tree) 和后缀数组 (suffix array)，特别是后缀树和后缀数组，在生物信息学的一些软件具体实现中得到了广泛的应用。这些标准的数据结构在常见的算法书中都有介绍，在这里我们就不单独介绍了。

§2.4 点阵法

点阵是一种简单的考察两序列相似性的可视化方法。在前一章介绍的动态规划序列比对算法还没有发现之前，Gibbs 和 McIntyre（1970）提出使用点阵法比较 DNA 或蛋白序列。这种方法目前仍然被广泛使用于显示 (i) 两序列的比对结果以及 (ii) 长基因组序列上所发生的诸如反转和复制等重组突变事件。

为了显示两序列，S 和 T，的相似性，将 S 和 T 分别沿 X- 轴和 Y- 轴方向放置；从原点向外依次排列各序列的字符。S 和 T 中的第 i 个字符，$T[i]$ 和 $S[i]$，分别对应二维格的第 i 行和第 i 列。S 和 T 的点阵按下列方法得到：对于所有的 i 和 j，如果 $S[i]$ 和 $T[j]$ 相等，则在坐标是 (i,j) 的格点里加上一个点。显然，在对角线上的点列

$$(k,j),(k+1,j+1),\cdots,(k+l,j+l)$$

表示序列段 $S[k,k+l]$ 和 $T[j,j+l]$ 相等，称为对角匹配线条。由于只有 4 种核苷酸和 20 种氨基酸，无论两个序列是否同源，它们的点阵里都有许多孤立的点和长度很短的对角匹配线条。

当 S 和 T 比较短时，点阵可以很清楚的反映它们的相似性。当 S 和 T 是长 DNA 序列时，点阵中将会有太多的点以至于它们的局部相似性不能很好的表现出来。如果每个核苷酸在两序列里出现的频率是 1/4，那么，将有几乎 1/4 的格点里画有点。为了增加视觉效果，我们往往抹去孤立点以及长度小于阈值的对角匹配线条。对于蛋白序列阈值通常设为 3；对于 DNA 序列，阈值较大。

在使用点阵显示两序列比对时，我们在匹配列对应的格点上画上一个点 (图 2.2B)。序列比对里的空位在其点阵上造成对角匹配线条断开，并且其中的一段沿 X- 或 Y- 轴方向平移若干位置，出现在另一条对角线上。例如，图 2.2A 中的比对有三个空位，形成四个对角匹配线条。同样的，在比对中的非配列导致同一对角线上的多个匹配线条。注意，图 2.2A 中的比对不含非配列。

在进化的过程中，基因组序列经历多次重组突变，从而导致各个基因组的结构有明显的差异。例如，不同生物体基因组的染色体的数目不等；同一基因簇在不同基因组里有不同数目的成员。

序列片段复制是一种频频发生的重组事件。在一个复制事件中，一个序列段被复制；得到的备份植入基因组的不同区域。复制事件造成两个或者多个高度相似的序列区域。这些区域在长基因组序列自身比较的点阵里对应平行的对角匹配线条 (图 2.2C)。

序列反转是另一种常见的重组突变事件。在进化过程中，基因组中的一序列段可以反向的植入同一位置，如下图所示：

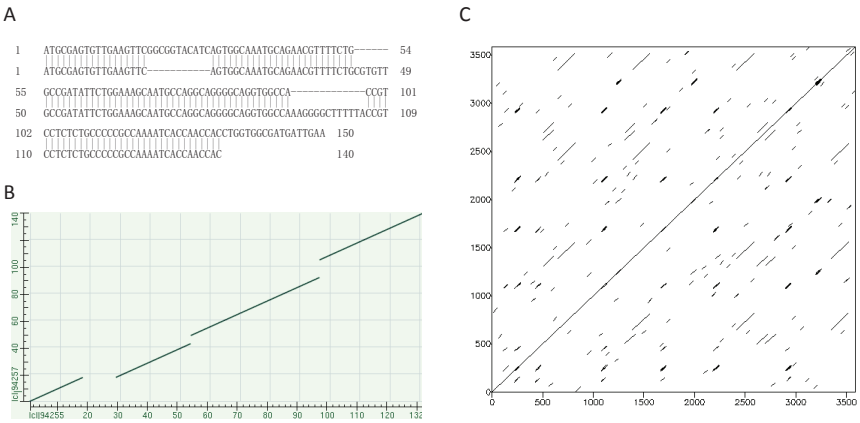

图 2.2: (A) BLAST 程序输出的一个双序列比对。(B) A 中比对的点阵表示。(C) 一个长序列和其自身的点阵。位于顶端的三条平行的对角匹配线条对应一个复制产生的三个序列段。出自 Dotmatcher 程序(http://emboss.bioinformatics.nl/cgi-bin/emboss/dotmatcher)。

$$
\begin{aligned}
&5'\text{-GCCATTCT}\mathbf{AGGCTTA}\text{AACGTT-}3'\\
&3'\text{-CGGTAAGA}\mathbf{TCCGAAT}\text{TTGCAA-}5'\\
&\qquad\Downarrow\\
&5'\text{-GCCATTCT}\mathbf{TAAGCCT}\text{AACGTT-}3'\\
&3'\text{-CGGTAAGA}\mathbf{ATTCGGA}\text{TTGCAA-}5'
\end{aligned}
$$

这一重组突变事件叫做反转。在反转事件前 DNA 的上链和事件后 DNA 的下链的点阵里 (假设前者沿 X- 轴反向放置), 在反转发生的区域 (从第九个碱基到第 15 个碱基) 上出现一条反向对角匹配线条：

$$(9,15),(10,14),(11,13),(12,12),(13,11),(14,10),(15,9)。$$

所以, 点阵也可以显示两基因组序列之间的反转事件。

英语语言里有几百个结构特别的字。无论是从左到右还是从右到左读这些字里的字母, 我们得到一样的字母序列。这些字叫回文字。常见的回文英文字包括 “dad”、“mom”、“civic”、和 “radar”。中文里也有许多回文句子。例如, “ 人人为我、我为人人”。

基因组中也存在许多具有一定生物功能的回文字, 这里回文的定义稍微不同。一个 DNA 双链序列具有回文结构是指在 5′-3′ 链上从左到右读和在 3′-5′ 链上从右到左读得到一样的序列。例如, 限制切酶 EcoR1 的识别序列是下列回文序列：

$$
\begin{aligned}
&5'\text{-GAATTC-}3'\\
&3'\text{-CTTAAG-}5'
\end{aligned}
$$

细菌基因组里有大量的回文序列段。最新研究发现人基因组 Y- 染色体上也有大量的回文序列。这种回文结构有助于 Y- 染色体的自身修复。在 Y- 染色体 5′-3′ 链和它的互补链的点阵里，一个回文序列段对应有两条长度相同，正交的对角匹配线条所组成的十字。

§2.5 *FASTA 程序

FASTA(Pearson & Lipman, 1988) 是最早提出的快速比对程序之一。它是通过缩小比对搜索空间来减少比对所需时间的一种方法。这里，我们先用一个简单例子来介绍 FASTA 如何利用短字匹配压缩比对搜索空间的思想。考虑 AGCTGACACCG 和 TATGCCAACCG。图 2.3A 是这两个 DNA 序列的点阵，其中有 33 个代表核苷酸匹配的点。在这些匹配点中，只有 14 个出现在长度是 2 或 4 的对角匹配线条里；其余的 19 个是孤立点。点阵中那条从左上角到右下角的路径代表 S 和 T 之间的一个最优比对。当所有的孤立匹配点被抹去以后（图 2.3B），我们可以清楚的看到那条最优比对路径穿过两条长度分别是 2 和 4 的对角匹配线条，突显在包含多个对角匹配线条的带状区域里。这个例子说明利用短字匹配可以有效地估计最优全序列 (局部) 比对所在的区域；和整个比对图相比，这个带状区域的面积要小得多。从而，如果只在这个带状区域里应用动态规划比对算法，我们就可以快速地计算出最优或者接近最优的全序列 (局部) 比对。

考虑两序列, S 和 T，以及它们的点阵 M。如果 S 和 T 的长度分别是 m 和 n, M 有 $m+n-1$ 条对角线并且 M 中的每个格点位于唯一的一条对角线上。格点 (x,y) 所在的对角线记为 D_{x-y}。我们还称 $x-y$ 是 D_{x-y} 的偏差。显然，主对角线是 D_0，其偏差是 0。偏差是负的对角线在 D_0 的右边，而偏差为正的对角线位于 D_0 的左边。

假设 P 是点阵上的一条路径, w 是一个正整数。我们用 $P \pm w$ 表示以 P 为中心的带状区域，即

$$P \pm w = \{(x_0, y_0 + j) \mid \exists (x_0, y_0) \in P, \exists j \in [-w, w]\}。$$

FASTA 算法有以下几个步骤：

1. 使用散列表找出输入序列间的 k- 字匹配以及其所在的对角线，这里 k 是一个程序参数，并选出含 k- 字匹配最多的 10 条对角线；
2. 在每条选出的对角线上使用动态规划方法和适当的打分矩阵确定出一个最优对角匹配区域;
3. 将在第二步中找到的区域使用水平和垂直的线段衔接起来，得到一个高分局部比对路径 P；
4. 在 $P \pm w$ 里使用 Smith-Waterman 算法计算出一个最优局部比对，这里 w 是 FASTA 的另一个程序参数。

在第一步里，我们扫描一个输入序列，S，的每一个位置，建立一个 k- 字的散列表。然后，我们再扫描另一输入序列，T，的每一个位置，查看该位置上的 k- 字是否在散列表里出现。若出现，利用它在 S 和 T 的位置，i 和 j，确定它所在的对角线 D_{i-j}。使用列表的方法，记忆每个对角线上的出现的 k- 长匹配线条的总个数，以及它们的位置。选出含 k- 字匹配对最多的 10 个对角线。

在 FASTA 的第二步，我们使用打分矩阵，计算在第一步里选出的每条对角线上的最优局部区域。这样做的理由有两个。其一是在比对蛋白序列时，每个 k- 字匹配的分数不一样。其二是含 k- 字匹配多的对角匹配线条并不一定得分就高。比如说，有下列两个比对：

X: AGAGCATCGAAT　　X: AGACCATGCAAT
Y: AGACCATCCAAT　　Y: AGACGAACCAAT

在左边的比对中，虽然只有一个 4- 字匹配，但是却有 10 个匹配列。虽然右边的比对含两个 4- 字匹配，但它只有 9 个匹配列。在对角线上使用局部比对可以找出使用 k- 字不能凸显的相似区域。

考虑任何一条选出的对角线 D_t。假设 D_t 上有 k 个格点以及它的第一个格点的 x- 和 y- 坐标分别是 u 和 v, $u - v = t$。D_t 上的 k 格点依次为：

$$(u, v), (u+1, v+1), \cdots, (u+k-1, v+k-1)。$$

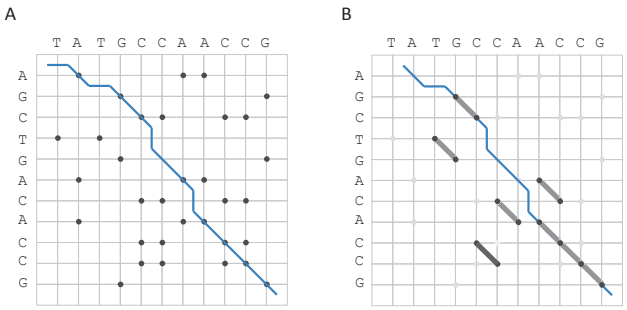

图 2.3: (A) AGCTGACACCG 和 TATGCCAACCG 的点阵，从左上角到右下角的路径对应在匹配列、非匹配列和增减列的分数分别是 3、-3 和 -2 的计分标准下的一个最优比对。(B) 抹去 19 个孤立匹配点后的点阵。灰色粗线表示剩下的 6 个对角匹配线条。

再假设匹配列和非匹配列的分数分别是 m 和 $-g$。定义

$$A_0 = 0,\ \ S_0 = 0,$$
$$A_i = \max\left\{\begin{array}{ll} A_{i-1} + m & \text{如果} s_{u+i} = t_{u+i-t} \\ A_{i-1} - g & \text{如果} s_{u+i} \neq t_{u+i-t} \\ 0 & \end{array}\right\},$$
$$S_i = \max(S_{i-1}, A_i),\ \ i = 1, 2, \cdots, k。$$

那么，D_t 上的最优局部比对分数是 S_k。读者可以思考如何引入位置变量记忆最优局部比对起点和终点的位置。

我们已经在每个选出的对角线上计算出一个最优局部比对区域。这样，一共得到 10 个对角局部比对区域，对应 10 个无空位局部比对。在 FASTA 的第三步，我们利用这 10 个对角局部比对区域的起点和终点，将他们的子段用最优的方式连接起来组成一个局部比对路径，P，(图 2.4)。如何连接这 10 个局部区域可以归约成 20×20 比对图上的一个比对问题 (习题 6)。这样，第三步仅需要常数次运算。

假设 P 上的格点是 $(x_1, y_1), (x_2, y_2), \cdots, (x_k, y_k)$。修饰后的 Smith-Waterman 局部比对算法使用下列公式计算最优比对分数。令 $v = (i, j) \in P \pm w$。$W(i, j)$ 在 v 处的最优比对分数。如果 v 在 $P \pm w$ 的第一行，$W(i, j) = 0$。如果 v 在其它行，我们有下列三种情况:

1. $(i-1, j), (i-1, j-1), (i, j-1)$ 都在 $P \pm w$ 里。我们有：

$$W(i, j) = \max\left\{\begin{array}{l} 0 \\ W(i-1, j) + \delta(s_i, -) \\ W(i, j-1) + \delta(-, t_j) \\ W(i-1, j-1) + \delta(s_i, t_j) \end{array}\right\},$$

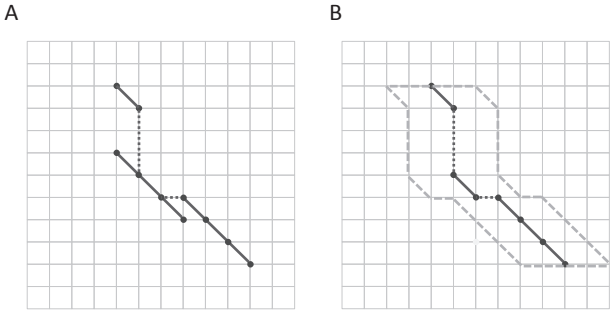

图 2.4: (A) FASTA 第三步的示意图。在这个例子中，三条对角线条需要用垂直或者水平的线条连接起来形成一个局部比对路径。(B) 从 A 中的局部比对路径得到的带状局部比对区域，这里带宽是 $4(2w)$。

其中，s_i 和 t_j 分别是 S 和 T 的第 i 个和 j 个字符，$\delta(,)$ 是计分函数。

2. 只有 $(i-1,j),(i-1,j-1)$ 在 $P\pm w$。我们有：

$$W(i,j)=\max\left\{\begin{array}{l}0\\ W(i-1,j)+\delta(s_i,-)\\ W(i-1,j-1)+\delta(s_i,t_j)\end{array}\right\}。$$

3. 只有 $(i,j-1),(i-1,j-1)$ 在 $P\pm w$。我们有：

$$W(i,j)=\max\left\{\begin{array}{l}0\\ W(i,j-1)+\delta(-,t_j)\\ W(i-1,j-1)+\delta(s_i,t_j)\end{array}\right\}。$$

因为 $P\pm w$ 里最多有 $2w|P|$ 个格点，我们可以使用 $O(2w|P|)$ 运算得出这个带状区域里的最优局部比对。

§2.6 BLAST 程序

最常用的同源序列搜索工具是BLAST 程序系列: BLASTn、BLASTp、PSI-BLAST 等等。BLAST 程序使用和 FASTA 不同的技巧来提高同源序列搜索的速度和性能。BLAST 程序可以分解为两部分：(1) 计算查询序列和数据库中序列之间的局部比对； (2) 对每个局部比对计算一个 E- 值，表示该比对是否具有 (统计意义上的) 显著性。BLAST 程序将得到的局部比对按其显著性一一列出。由于 BLAST 程序设计体现了算法方法和统计理论的完美结合，BLAST 已经成为生物信息学的经典程序之一。

§2.6.1 基本算法：连续核的概念

我们首先以查询 DNA 序列数据库为例。假设查询序列是 Q，搜索的数据库是 D。对于 D 中的每一个序列 S，BLAST 算法的第一步是找到 Q 的每个 k- 字在 S 里的严格匹配 (即同一个 k- 字)。这里使用的 k 是 BLAST 的一个用户可设置的参数。这些严格匹配可以使用 §2.3 中介绍的字匹配方法或其它途径在 $O(|S|)$ 时间内确定。BLAST 将在这些严格匹配的附近寻找高分局部比对，所以它们被称为连续核。

BLAST 采用这一步的原因如下。当 k 的值介于 10 到 20 之间，两个同源序列的比对包含一个 k- 字匹配的概率相当大；而在 Q 和 S 的两个具体的位置上，产生这样一个严格匹配的概率却很小。所以，仅对于产生严格匹配的那些位置进行更为复杂的同源性检查要比盲目地在每一对位置上进行同源性检查有效的多。

BLAST 算法的第二步是把每一个连续核向两端进行无空位的扩张，试图找到一个高分的局部无空位比对。一般来讲，随着扩张的进行，比对的得分逐步增加，然后达到一个最大值之后再逐步减少。因此，当比对的得分上升、

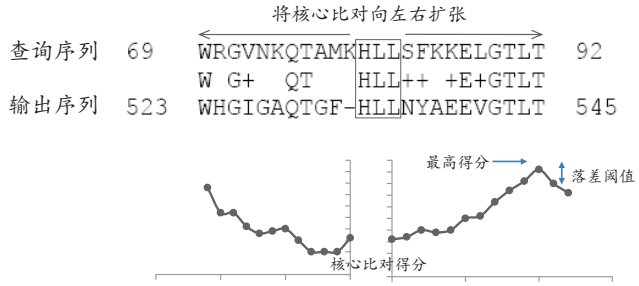

图 2.5: BLAST 算法第二步的示意图。核心比对是HLL和其自身之间的匹配。输出比对的第一行是查询序列中从第 69 个氨基酸到第 92 个氨基酸的序列段，而第二行是数据库中一个序列的子段

然后再下降到和当前最高得分有一定落差时，扩张将不再进行（图 2.5）。将所就得到的无空位比对记为 $\mathcal{A}$。不难看出，得到 $\mathcal{A}$ 所用的时间和扩张的长度成正线性比。如果 $\mathcal{A}$ 的分数高于预先设定的阈值，则基本上可以断定它就是一个同源序列比对的一部分。此时，BLAST 才调用更为精确的具有平方级时间复杂度的 Smith-Waterman 方法来做局部比对。

通过连续核和无空位扩张两步的过滤，BLAST 大大减少了需要做平方级时间比对的次数，所以大大提高了同源查找的速度。但正如在 §2.1 中所讨论的那样，BLAST 为此实际上也损失了一定的灵敏度。这是因为两个真正的同源序列之间的比对 不一定包含一个长度为 k 的连续核匹配；这样的同源序列比对就不能被 BLAST 发现。我们还不难看出，通过增大 k 的值，可以进一步提高速度，但是会损失更多的灵敏度。反之亦然。最初 BLAST 缺省的 k 对 DNA 序列设为 11。

BLAST 后来的版本还提出了双核匹配的方法。具体的说，只有在发现两个不重叠的连续核同时出现在一个很短的区域时，它才去做无空位扩张。实践证明这样做更能提高速度和灵敏度取舍的效率。

因为氨基酸的相似性不能简单的看是否匹配，同源查询程序需要 BLOSUM 或者其它的打分矩阵来确定蛋白值同源序列。所以，如果查询的是蛋白质序列，BLAST 不再要求 k- 字严格匹配，而是要求查询序列里的 k- 字和数据库序列里的 k- 字的无空位比对的得分超过一定的阈值。而且，对于蛋白质序列，k 的取值要小得多，一般是 3 或者 4。这是因为，即使 $k = 3$, BLAST 也要考虑多达 $8000 (= 20^3)$ 个子字。

§2.6.2 E- 值的计算公式

BLAST 程序的第二个特点是将得到的局部比对按其统计显著性一一列出。假设使用查询序列 Q 搜索一个数据库 D 得到一个分数为 s 的局部比对，它的 p- 值定义为 $\Pr[S(D,Y) \geq s]$, 其中 $S(D,Y)$ 代表一个和 Q 等长的随机

字符序列 Y 和 D 中序列之间的最优局部比对的得分。p- 值 0.1 表示当选取一随机序列进行查询时，有十分之一的机会得到一个得分不低于 s 的局部比对；p- 值 0.000001 表示使用随机序列对数据库 D 进行一百万次同源查询大约仅有一次我们可以得到更好的局部比对。所以，一个局部比对的 p- 值越小，它就越显著。

考虑长度分别为 m 和 n 的两条随机序列，X 和 Y。根据局部比对的 Altschul 和 Karlin (1993) 统计理论，X 和 Y 之间的无空位局部比对的最高得分，$S(m,n)$, 满足极值分布:

$$\Pr[S(m,n) < x] = \exp(-Kmne^{-\lambda x}),$$

其中 λ 和 K 是依据比对计分法则确定的常数。因而，

$$\Pr[S(m,n) \geq x] = 1 - \exp(-Kmne^{-\lambda x})。$$

另外，X 和 Y 之间得分高于 s 的无空位局部比对的数目的 E- 值（期望值）等于

$$E(m,n) = Kmne^{-\lambda S},$$

不难看出，p- 值和 E- 值有下列简单关系：

$$\Pr[S(m,n) \geq x] = 1 - \exp(-E(m,n))。$$

所以，BLAST 采用 E- 值表达一个局部比对的显著性。两序列的长度之积 mn 称为有效搜索空间的规模。因为参与比对的序列的两端不可能形成高比分局部比对，BLAST 将有效搜索空间的规模调整为

$$(m - \bar{L})(n - \bar{L}),$$

其中 $\bar{L}$ 是最优无空位比对的平均长度。假设数据库中有 l 条序列，总字符数是 N。数据库搜索的总期望值, $E(X)$, 等于在各个序列上期望值的和，即

$$E(X) = \sum_Y E(|X|,|Y|) = \sum_Y K(m - \bar{L})(|Y| - \bar{L})e^{-\lambda S} = K(m - \bar{L})(N - l\bar{L})e^{-\lambda S}, \quad (2.14)$$

其中 $|\cdot|$ 是对应序列的长度。

Altschul-Karlin 统计也适用于含空位比对。对于无空位局部比对，参数 K 和 λ 的值可以根据公式算出。但对于含空位局部比对，它们的值只能通过模拟实验得到。例如，在使用 BLAST2.2.18 搜索 SwissProt 数据库时，计算含空位比对的 E- 值所使用的 K 和 λ 分别是 0.267 和 0.041。图 2.6A 中给出的局部比对的得分是 83。依据 (2.14), 它的 E- 值按下列计算得到：

$$0.041 \times (234 - 111) \times (124438793 - 332988 \times 111) \times e^{-0.267 \times 83} \approx 0.105。$$

A

```
 Score = 36.6 bits (83), Expect = 0.11, Method: Compositional matrix adjust.
 Identities = 29/86 (33%), Positives = 44/86 (51%), Gaps = 9/86 (10%)

Query 1   MSLPIIKVH-WLDHSRAFRLLWLLDHLNLEYEIVPYKR-DANFRAPPELKKIHPLGRSPL 58
          M+ P +KV+ W       R L  L+   ++YE+VP  R D + R P  L + +P G+ P+
Sbjct 1   MATPAVKVYGWAISPFVSRALLALEEAGVDYELVPMSRQDGDHRRPEHLAR-NPFGKVPV 59

Query 59  LEVQDRETGKKKILAESGFIPQYVLQ 84
          LE  D         L ES  I ++VL+
Sbjct 60  LEDGDL------TLFESRAIARHVLR 79
```

B

数据库	序列数目	字符总数	打分矩阵	$\bar{L}$	K	λ
SwissProt	332,988	124,438,792	BLOSUM62	111	0.267	0.041

图 2.6: 使用 BLASTp(版本 2.2.18) 搜索 SwissProt 的一个例子。(A) 输出的一个含空位比对。查询蛋白质序列是酵母菌 GST-1(P40582.1); 搜索返回的序列是玉米 GST-4(P46420.2)。(B) SwissProt 的规模以及和打分矩阵 BLOSUM62 有关的计算 E- 值的常数。经许可摘自 Chao & Zhang (2009)。

BLAST 不仅给出输出局部比对的得分 S 和对应的 E- 值, 还输出局部比对的比特分 (bit score)。局部比对的比特分, S', 是从比对的原得分 S 经过规范化得到:

$$S' = (\lambda S - \ln K)/\ln 2。$$

例如, 图 2.6A 中给出的局部比对的比特分是:

$$[0.267 \times 83 - \ln(0.041)]/\ln 2 \approx 36.58。$$

一个局部比对在不同的打分矩阵下分数不同。比对的比特分去除了打分矩阵以及数据库大小的影响, 仅仅表示一个比对所含的信息量。因此, 利用比特分, 我们可以得到来自不同数据库搜索的局部比对的相对显著性。

§2.6.3 BLAST 程序系列

BLAST 程序 (Altschul et al., 1990) 将用户输入的查询序列与数据库里的序列逐一比较, 输出它和数据库序列之间的高分的局部比对。每个输出的局部比对表示查询序列和比对里出现的另一序列同源。使用 BLAST 查询 (图 2.7) 有四个环节:

1. 粘贴或者上传查询序列到 BLAST 网页上的输入框。用户可以按 FASTA 格式输入一个 DNA/蛋白质的序列, 也可以仅仅给出查询序列的数据库编号。
2. 选取一个 BLAST 程序。
3. 选取适当的序列数据库。供选择的 DNA 数据库有 nr, chromosome, 和 est 等。蛋白质数据库有 nr, Swissprot, Pdb 等。

4. 设置程序的一些可变参数。这些参数包括打分矩阵，是否过滤数据库中序列的低复杂度区域，和是否将查询限制到某个模式生物基因组，等等。

   BLAST 程序系列包括五个基本 BLAST 程序：

- BLASTn 程序，用于查找一个 DNA 序列数据库中和查询 DNA 序列同源的序列。
- BLASTp 程序，用于查找一个蛋白质序列数据库中和查询蛋白质序列同源的序列。
- BLASTx 程序，用于查找一个蛋白质序列数据库中和查询 DNA 序列同源的序列。为了比较一条 DNA 序列和一条蛋白质序列，我们需要把前者转换成蛋白质序列或把后者转换成 DAN 序列。因为基于蛋白质序列推断同源性更可靠，BLAST 先把查询 DNA 序列转换成蛋白质序列再搜索蛋白质序列数据库。DNA 的 5′-3′ 链可以按三种不同的阅读框转录和翻译成一个蛋白质，而它的互补链也有同样多的可能对应一个蛋白质。因

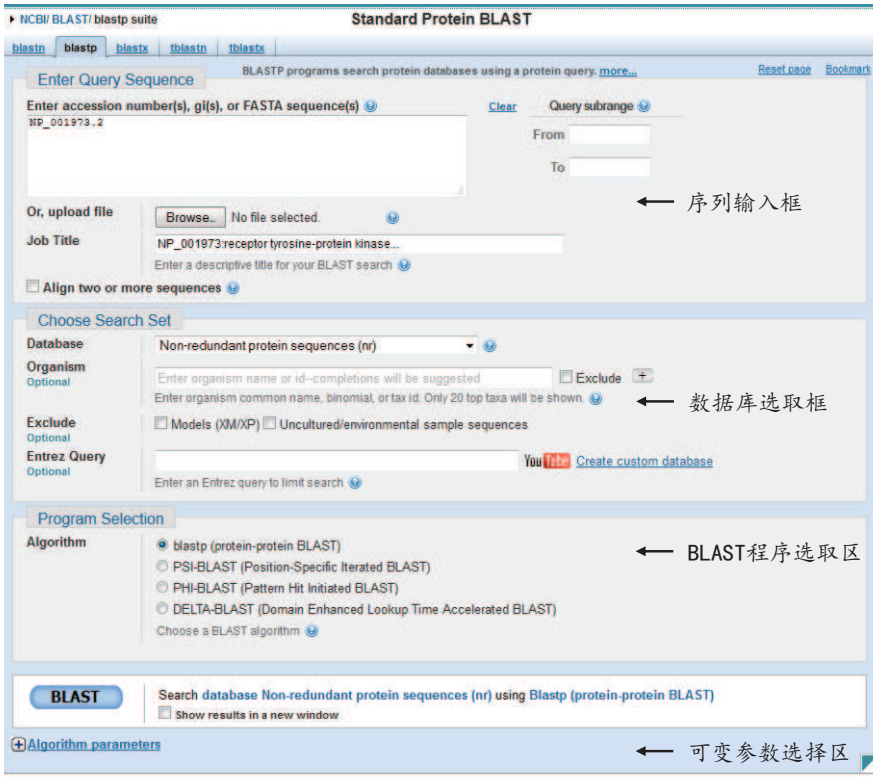

图 2.7: NCBI 服务器 (http://blast.ncbi.nlm.nih.gov/Blast.cgi) 上使用 BLASTp 进行数据库搜索的网页。

为蛋白编码序列可以出现在 DNA 的任何一个链上，BLASTx 程序在搜索蛋白质数据库时同时考虑了这六种可能性。

- tBLASTn 程序，用于查找一个 DNA 序列数据库中和查询蛋白质序列同源的序列。和 BLASTx 一样，数据中 DNA 序列先按六种不同的阅读框转换成蛋白质序列，再和输入蛋白质序列比对。
- tBLASTx 程序，用于查找一个 DNA 序列数据库中和查询 DNA 序列同源的序列。但和 BLASTn 程序不一样，它按照六种不同的阅读框将输入 DNA 序列和数据库里每一个 DNA 序列转换成蛋白序列再进行比对。

除上面提到的五个基本 BLAST 程序以外，还有两个 BLAST 程序值得一提。它们分别是 PSI-BLAST(Altschul, et al., 2001) 和 MegaBLAST(Zhang et al., 1990)。PSI-BLAST 是利用迭代的方式来提高蛋白序列搜索的性能。这个程序首先调用 BLASTp 程序得到一组同源序列，然后从所得到的同源序列建立一个多序列比对，并使用得到的这个多序列比对再度搜索数据库。这个过程可以依据用户的需求重复多次。

MegaBLAST 用于搜索基因组序列。由于基因组序列很长，MegaBLAST 通过将字长度 k 的默认值从 11 增加到 28 来提高同源搜索的速度。(k 的值最大可取到 64。) 它采用的另外一个措施是使用更简单的空位罚分基本模型。具体细节，我们在此不加以讨论。

§2.7 散核方法

在前一章中我们给出了局部序列比对的 Smith-Waterman 算法。由于其复杂性是平方级的，直接使用这个方法对于搜索 DNA 和蛋白质序列数据库来讲就太慢了。在本章里，我们也介绍了 BLAST 程序对于同源序列查找的提速方式。但是，这种提速方式会造成同源序列查询灵敏度在较大程度上的下降。在本节中，我们介绍如何使用散核方法来提高同源搜索的灵敏度，以达到灵敏度和速度之间更高效的互换。

§2.7.1 散核模型

BLAST 的基本算法利用了一个连续核的方法来达到提速的目的。也就是先从两个待比对的序列 S 和 T 中快速的找出所有 的 (i, j) 对，使得在 S 里从 i 开始的 k 个连续字符严格匹配在 T 里从 j 开始的 k 个连续字符。这样的一对 (i, j) 称为一个连续核匹配。在一个核匹配的基础上向序列的两端扩展产生一个无空位比对。如果无空位比对的得分超过预先设定的阈值，再调用 Smith-Waterman 算法进行局部的序列比对。

由于通过连续核匹配对需要做序列比对的位置进行过滤，两个序列的大多数位置对会因为无法产生核匹配而不会消耗算法的运行时间，所以可以大大提高同源序列查找的速度。然而，这同时也降低了查找的灵敏度。也就是