构建微生物分子分类系统进化树的快速运算法与数据结构

陆正福 徐丽华 姜成林

(云南大学数学系 云南省微生物研究所 昆明 650091)

许 宗 雄

(台湾清华大学生命科学院 新竹 30043)

摘要 本文介绍了构建系统进化树的 NJ 方法(Neighbor Joining Method)所涉及的算法与数据结构。文中给出了基于数据复用性的算法改进，获得了快速算法 —— FNJ 算法，从而将算法的时间复杂度由 $\Theta(N^5)$ 降低为 $\Theta(N^3)$；并给出了自动绘制进化分枝图的算法。

关键词 FNJ 算法，算法分析和设计，数据结构，系统进化树

微生物分子分类所获得的数据(蛋白质与核酸序列分析等)常常要构建成系统进化树。用系统进化树可以更直观、更科学地说明各分类单员之间的亲缘关系。

从进化距离数据构造系统进化树，国外已进行了许多研究，先后出现了标准算法、UPGMA、Farris 方法、ST 方法、Li 方法、修正的 Farris 方法，以及 NJ 方法。这些方法基于最小进化原理。其中，NJ 方法是一种新的方法，自 1987 年发表以来，为涉及系统进化树的文献普遍引用，但国内在这方面的工作几乎不曾见到。本文 1、2、3 节主要是从程序实现的角度给出实现此方法所涉及的算法与数据结构，改进的目标是提高计算机的时间效率，降低时间复杂度(在样本规模较大时)，为实现计算机自动绘制系统进化树，第 4 节给出了自动绘制算法。

符号约定：$f(n)=\Theta(g(n))$：表示存在正常数 C_1，C_2 和 K，对于所有的 $n>K$，成立 $C_1|g(n)|\leqslant|f(n)|\leqslant C_2|g(n)|$，即算法的计算时间量在最好和最坏情况下就一个常因子范围内而言是相同的；

s1__s2：由构成邻接对的 OTUs1 和 s2 所导出的复合(combined) OTU 的符号表示；

本处未说明的，均在相应符号第一次出现时说明。

1 NJ 方法的算法描述 —— 关于 NJ 算法

文献[1]以实例给出了 NJ 方法，根据此方法可由进化距离数据建立无根树连结拓扑，同时求出每一分支的长度。我们将此方法一般化、形式化、算法化，直接套搬文献[1]的公式，给出它的算法(简称为 NJ 算法)描述如下：

STEP1. 确定所有的 OTU 形成的集合为 $T=\{1, 2, \cdots\cdots, N-1, N\}$，并假定它们组成星状树；初始化有关数据[含 $N(N-1)/2$ 个进化距离值]；置 n 的初值为 N；STEP1 的单步计算时间量为 $F_1(N)=\Theta(N^2)$。

STEP2. 计算每一对 OTU 所对应的最小平方解(least square solution) S_{ij}：

$$S_{ij}=(D_{ij})/2+[\sum_{k\neq i,\ j}(D_{ik}+D_{jk})]/[2\times(n-2)]+(\sum_{\substack{k\neq i,\ j\\ l\neq i,\ j\\ k<l}}D_{kl})/(n-2)$$

其中 D_{ij} 表示 OTUi 和 j 之间的距离值。求出邻接对(neighbour) [a, b]，a、b 满足：$S_{ab}\leqslant S_{ij}$

国家自然科学基金、云南省国际合作基金、云南大学 211 工程基金资助项目

1995-08-[illegible] 收稿

对所有的 i，j∈T，i≠j，即 S_{ab} 是所有 n(n−1)/2 个最小平方解中数值最小的之一，a—b 不唯一时，可任取一对。STEP2 的单步计算时间量为 $F_2(n)=\Theta(n^4)$。

STEP3. 求出分支长度(branch length)，其公式为：

$$L_{ix}=(D_{ij}+D_{iz}-D_{jz})/2 \quad L_{jx}=(D_{ij}+D_{jz}-D_{iz})/2$$

其中 $D_{iz}=\left(\sum_{k\neq i,j} D_{ik}\right)\Big/(n-2)$

$$D_{jz}=\left(\sum_{k\neq i,j} D_{jk}\right)\Big/(n-2)$$

设结点 i、j 通过枝结点 X 相连。STEP3 的单步计算时间量为 $F_3(n)=\Theta(n)$。

STEP4. 修改距离数据，其公式为：

$$D_{(i\_j)k}=(D_{ik}+D_{jk})/2 \quad (k\neq i,j)$$

原来的 D_{ik}、D_{jk}、D_{ki}、D_{kj} 数据不再被使用，相当于形成了新的 OTU 集合：

$$T:=(T\bigcup\{i\_j\})\setminus\{i,j\};$$

n:=n−1，若 n≥3，转 STEP2（以新的 OTU 集合及相应的距离数据，再次运行 STEP2 及其以后各步），否则结束。STEP4 的单步计算时间量为 $F_4(n)=\Theta(n)$

NJ 算法的总计算时间量为

$$F_1(N)+\sum_{n=3}^{N}[F_2(n)+F_3(n)+F_4(n)]=\Theta(N^5)。$$

2 公式变换与算法改进 —— 关于 FNJ 算法

在 NJ 方法的算法描述中，所涉及的公式由于包含了大量的下标判断和重复的数据运算，不便于程序实现和中间结果的复用(reuse)，影响程序执行的效率。因此，我们对有关公式进行了如下的等价变换：

2.1 定义

$D^{(i)}=\sum_k D_{ik}$ 结点 i 与所有其它结点之间的距离的总和，简称为结点 i 的和距；

$D_{sum}=\sum_i D^{(i)}$ 所有结点之间的距离的总和；

$W_{ij}=c\times(D_{ij})-D^{(i)}-D^{(j)}$ （其中 $c=N-2$）

2.2 最小平方解

$$S_{ij}=(D_{sum}-D^{(i)}-D^{(j)})/[2\times(N-2)]+(D_{ij})/2$$
$$=D_{sum}+(W_{ij})/[2\times(N-2)]$$

所以最小的 S_{ij} 所对应的 i—j 邻接对，等于最小的 W_{ij} 所对应的 i—j 邻接对；

2.3 分支长度

$L_{ix}=(D_{ij}+D^{(i)}-D^{(j)})/2$ 叶结点 i 与枝结点 X 之间的长度

$L_{jx}=D_{ij}-L_{ix}$ 叶结点 j 与枝结点 X 之间的长度

$L_{XY}=L_{(i\_j)Y}-(D_{ij})/2$ 枝结点 X 与枝结点 Y 之间的长度

2.4 数据修改公式为

$D_{(i\_j)k}:=(D_{ik}+D_{jk})/2$ 其中 $k\in T\setminus\{i,j\}$

$D^{(k)}:=D^{(k)}-D_{(i\_j)k}$ 其中 $k\in T\setminus\{i,j\}$

$D^{(i\_j)}:=(D^{(i)}+D^{(j)})/2$

$D_{sum}:=D_{sum}-D^{(i)}-D^{(j)}$

$T:=(T\bigcup\{i\_j\})\setminus\{i,j\}$

2.5 改进的算法 —— FNJ 算法

从上述变换可以看出，变换后的公式简洁明了，中间结果的可复用性大大增强，这对于样本数量较大的问题，在提高计算机的时间效率、减少时间复杂度方面具有重要的意义。基于 2.1 ～ 2.4 的公式，改进的算法(简称为快速 NJ 算法或 FNJ 算法)如下：

STEP1. 确定 OTU 的初始集合为 T:={1, 2, ……, N−1, N}，并假定它们组成星状树；输入初始进化距离数据(D_{ij}; i, j∈T}；置 m、n 的初值为 N，Wmin、a、b 的初值为 +∞。

基于初始进化距离数据，计算 $D^{(i)}$，i∈T；其计算时间量为 $f_1(N)=\Theta(N^2)$

STEP2. c:=n−2;

根据循环变量 i、j 执行如下的运算和比较：

计算 T 中每对 OTU 所对应的 W_{ij}，其公式为：

$$W_{ij}=c\times(D_{ij})-D^{(i)}-D^{(j)} \quad i,j\in T,\ i<j;$$

比较 Wmin 和 W_{ij}，若 Wmin>W_{ij}，则执行

Wmin: $=W_{ij}$, a: =i、b: =j，否则继续循环 STEP2 的单步计算时间量为 $f_2(n)=\Theta(n^2)$。

STEP3. m: =m+1

计算分支长度:

$D_{ave}=(D_{ab})/2$

$D_{dif}=(D^{(a)}-D^{(b)})/2$

$L_{am}=D_{ave}+D_{dif}$

$L_{bm}=D_{ave}+D_{dif}$

追加二叉树的结点 m，保存有关数据: 连结关系，D_{ab}, L_{am}, L_{bm}

保存 D_{ab} 的目的在于计算枝结点之间的长度 STEP3 的单步计算时间量为 $f_3(n)=\Theta(1)$

STEP4. 修改数据

$D_{mk}=(D_{ak}+D_{bk})/2$ 其中 $k\in T\setminus\{i,j\}$

$D^{(k)}$: $=D^{(k)}-D_{mk}$ 其中 $k\in T\setminus\{i,j\}$

$D^{(m)}$: $=(D^{(a)}+D^{(b)})/2$

T: $=(T\cup\{m\})\setminus\{a,b\}$

n: =n−1，若 n≥3，转 STEP2（以新的 OTU 集合及相应的数据，再次运行 STEP2 及其以后各步），否则结束。STEP4 的单步计算时间量为 $f_4(n)=\Theta(n)$。

FNJ 算法的总计算时间量为

$$f_1(N)+\sum_{n=3}^{N}[f_2(n)+f_3(n)+f_4(n)]=\Theta(N^3)$$

2.6 NJ 算法和 FNJ 算法的比较（基于相同的数据结构及其操作算法）

	NJ 算法	FNJ 算法
总计算时间量	$\Theta(N^5)$	$\Theta(N^3)$
STEP1 总计算时间量	$\Theta(N^2)$	$\Theta(N^2)$
STEP2 总计算时间量	$\Theta(N^5)$	$\Theta(N^3)$
STEP3 总计算时间量	$\Theta(N^2)$	$\Theta(N)$
STEP4 总计算时间量	$\Theta(N^2)$	$\Theta(N^2)$
比较运算时间量	$\Theta(N^5)$	$\Theta(N^3)$
比较运算所占比例	大（约 75%）	小（约 40%）
变量存取操作时间量	$\Theta(N^5)$	$\Theta(N^3)$
加 / 减法计算时间量	$\Theta(N^5)$	$\Theta(N^3)$
乘 / 除法计算时间量	$\Theta(N^3)$	$\Theta(N^3)$

正如大多数信号与信息处理的快速算法，我们的做法也是尽可能利用数据的可复用性，减少不必要的重复运算（如其中的加法运算），去除大量不必要的比较判断，改变求解目标（如由求解 S_{ij} 改成求解 W_{ij}）和运算对象（如大量对 $\{D_{ij}\}$ 的存取操作改成少量对 $\{D^{(i)}\}$ 的存取操作）。

由这些比较可以看出，对于经常使用的数百个样本的规模，计算时间的改善不下万倍。

3 有关 FNJ 算法的数据结构

下文的数据结构是用 C（C++）语言的结构类型说明的，在实际的程序实现时若采用面向对象的程序设计，则可将数据和操作两部分予以适当的封装。

3.1 进化距离矩阵与和距的表示

在系统进化树的构建过程中，进化距离矩阵是不断变化的，所以可采用动态数据结构——十字链表；为节省内存空间，可只存贮对称矩阵的下（或上）三角部分。

结点的数据构成:

```
struct cross_nodel
  { unsigned int i_row;          // 结点所在行的正整数编号
    unsigned int j_column;       // 结点所在列的正整数编号
    float dist;                  // 结点 i 和 j 的距离
    struct cross_node *down;     // 指向下结点的指针
    struct cross_node *right; }; // 指向右结点的指针
```

在行头结点里另含有 sum_dist 域，用于存放相应 OTU 的和距。

对于 FNJ 算法而言，对该链表的操作算法有:

(1)初始化（建表）;

(2)获取数据 D_{ij}，即 dist 域的数值;

(3)遍历第 i 行及第 j 列，求其和距 $D^{(i)}$（此项操作只对初始数据），从而可求最小平方解、分支长度等;

(4)追加第 M+1 行，其数据域为 $D_{M+1,k}=(D_{ik}+D_{jk})/2$，M 的初值为 N，此处用 M+1

代表 i、j，从而新的 OTU 集合变成 $T_{new}=(T_{old}\cup\{M+1\})\setminus\{i, j\}$；

(5) 置第 i 行及第 i 列的 dist 为零，或删除第 i 行及第 i 列(可只保留其行、列的头结点)，并释放该行、列占用的内存空间；

(6) 删除该链表，释放占用的内存空间。

应该指出：上述操作中，涉及大量的对于结点数据的存取，在样本规模较大时，这对于非随机存取的链式存储结构，是一项费时的操作。然而，若用顺序存储结构如数组实现，虽可随机存取，解决链式存储结构的不足，则又难以适应进化距离矩阵的动态变化。

鉴于上述，我们在用 C(C++) 语言具体实现时，利用了该语言的连续存储区指针和数组在存取上的可替代性，设计了一种能够随机存取和动态变化的数据结构，大致思路是，将所有的行头结点组成动态连续存储区，各行分别组成动态连续存储区，重新设计结点的数据构成：

```
struct cross_node2
   { float dist;                    // 结点 i 和 j 的
                                       距离
     struct cross_node *down; };    // 指向下结点
                                       的指针
```

操作(5)改成：置第 i 列的 dist 为零，或删除第 i 行(但保留其行头结点)，并释放该行占用的内存空间。

3.2 用二叉树结构记录的无根树连结关系及距离数据

系统进化树是无根的，所以二叉树的根只是为了方便操作而设置。

结点的数据构成：

```
struct bin_tree
  {int num;                          // 结点编号
   struct bin_tree *left_child;      // 指向左子结点的
                                        指针
   struct bin_tree *right_child;     // 指向右子结点的
                                        指针
   int left_length;                  // 当前结点与左子
                                        结点之间的长度
   int right_length;                 // 当前结点与右子结点之
                                        间的长度
   int left_right_dist;              // 当前结点的左、右两个
                                        子结点之间的距离
  // 以下为自动绘图算法部分所设置
   int x, y; };                      // 当前结点的坐标
```

对于 FNJ 算法而言，对该链表的操作算法有：

(1) 追加结点并建树

(2) 遍历该树，输出可用于手工绘图的数据；

(3) 删除该树，释放占用的内存空间。

注：若只需要手工绘图的数据，则此树在输出数据后即可删除，否则需要保留到自动绘图结束后才可删除。

4 自动绘图算法

4.1 关于系统进化树的一些约束条件

a. 它是一种拓扑结构，作图时必须表示出正确的连接关系；

b. 它是带有距离的，作图时必须按比例绘制线段；

c. 它是与线段曲直和线段间角度无关的；

d. 为了便于人的观察，系统进化树应在二维平面上绘制，线段间除了在树的枝结点处相交外，不应在别处相交。

4.2 算法

从 4.1 可知，系统进化树的绘制可归结为约束不充分问题，施加不同的附加约束条件，将得到表现形式不同而拓扑等价的系统进化树。常见的表现形式有进化分枝图(cladogram)，接近聚类分析中的聚类分枝树形式，较适于计算机系统的自动绘制；另一种常见的表现形式是网络图论中的加权图(weightedgraph)形式，较适于人工绘制，难以适于计算机系统的自动绘制。

我们在此给出以进化分枝图为表现形式的自动绘图算法；该算法是基于 3.3 的二叉树结构，采用了递归技术，因此形式上很简练。其算法描述如下：

STEP1. 初始化有关数据，包括纵向间隔值 yinter，横向放大值 xmult，计数变量 count=0，根结点的横向坐标值 xroot;

STEP2. 先序递归遍历二叉树，确定各结点的横向坐标值:

左子结点的域 x:=xroot+left_length*xmult

右子结点的域 x:=xroot+right_length*xmult

STEP3. 后序递归遍历二叉树，确定各结点的纵向坐标值:

每到达一次叶结点，执行:

{该叶结点的域 y:=count*yinter; count:=count+1}

枝结点的域 y:=(左子结点的域 y+右子结点的域 y)/2

STEP4. 用 Brensenham 算法(可参阅任何有关图形学的书籍)连结各相应结点对;

STEP5. 显示和打印。

4.3 注记

鉴于加权图的直观性，自然就有如何用计算机系统自动绘制加权图的问题，这将涉及对系统进化树的网络图论性质的研究以及人工智能和数据库技术的应用[2~4]，还涉及计算几何与图形学中的一些问题。

随着工作的深入，样本规模越来越大(已逾千数)，如何实现任意规模系统进化树(不论何种表现形式)的自动绘制，从而突破规模、速度、形式等的限制，达到更理想的境地?

上述两个问题都需要从方法、算法、数据结构上进行更多的研究，我们拟以另文报道。

基于本文介绍的系统进化树构造算法与数据结构，我们已用 C++ 语言在微机上自行设计了 FRAP (Fast Reconstructing Aalgorithms and Programs) 程序，构成了我们设计的模式分类与信息处理软件 BIOPRIPS 的一个组成部分。

借助计算机进行分子分类，需要好的方法，还需要好的算法及与之相关的数据结构，而后才有高效的程序实现。本文所做的工作旨在抛砖引玉，期望有更新更好的方法、算法及更合适的与之相关的数据结构的面世。

参 考 文 献

[1] Naruya Saitu, Masatoshi Nei. Mol Biol Evol, 1987, 4 (4): 406 ~ 425.

[2] Bondy J A, Murty U. S R. Graph Theory With Applictions, Macmillan Press Ltd. 1987.

[3] 邹海明，余祥宣. 计算机算法基础. 武汉：华中工学院出版社，1985.

[4] 傅京孙，蔡自兴，徐光佑. 人工智能及其应用. 北京：清华大学出版社，1987.

FAST RECONSTRUCTING ALGORITHMS AND DATA STRUCTURES OF THE PHYLOGENETIC TREES FOR MICROBIAL MOLECULAR CLASSIFICATION

Lu Zhengfu Xu Lihua Jiang Chenglin

(*Department of Mathematics, Yunnan Institute of Microbiology, Yunnan University, Kunming* 650091)

Xu Zongxiong

(*College of Life Science, TaiWan Tsing Hua University, Hsinchu* 30043)

Abstract The algorithms and data structures involved in the neighbour_joining method (NJM) for the reconstructing of phylogenetic trees are introduced in this paper. Based on the data reuse, we designed the fast neighbour_joining algorithm which can reduce the time complexity from $\Theta(N^5)$ to $\Theta(N^3)$, and the automatic plotting algorithm for cladogram.

Key words Fast neighbour joining algorithm, Analysis and design of algorithms, Data structure, Phylogenetic tree / evolutionary tree