Journal of Southeast University (English Edition) Vol. 22, No. 2, pp. 176 – 179 June 2006 ISSN 1003—7985

Fast algorithm for constructing neighbor-joining phylogenetic trees

Chen Ningtao[1] Wang Nengchao[2] Shi Baochang[2]

([1] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

([2] Parallel Computing Institute, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: To improve the performance of Saitou and Nei's algorithm (SN) and Studier and Keppler's improved algorithm (SK) for constructing neighbor-joining phylogenetic trees and reduce the time complexity of the computation, a fast algorithm is proposed. The proposed algorithm includes three techniques. First, a linear array $A[N]$ is introduced to store the sum of every row of the distance matrix (the same as SK), which can eliminate many repeated computations. Secondly, the value of $A[i]$ is computed only once at the beginning of the algorithm, and is updated by three elements in the iteration. Thirdly, a very compact formula for the sum of all the branch lengths of operational taxonomic units (OTUs) i and j is designed, and the correctness of the formula is proved. The experimental results show that the proposed algorithm is from tens to hundreds times faster than SN and roughly two times faster than SK when N increases, constructing a tree with 2 000 OTUs in 3 min on a current desktop computer. To earn the time with the cost of the space and reduce the computations in the inner-most loop are the basic solutions for algorithms with many loops.

Key words: phylogenetic tree; neighbor-joining method; fast algorithm; progressive multiple alignment

Phylogenetic trees are used to express the evolutionary history of a group of organisms[1]. Construction of an evolutionary history for a set of contemporary taxa based on their pairwise distance is NP-complete for various optimality criteria[2, 3]. Many greedy heuristics have been proposed, among which, the neighbor-joining (NJ) method[4 – 6] is widely used by molecular biologists due to its efficiency and simplicity. Especially, the NJ method is very popular in multiple sequence alignment (MSA) because it is applicable to any type of evolutionary distance data. The output of the NJ method is used to direct the grouping of sequences during the multiple alignment process. Most recent famous MSA software such as CLUSTAL W[7], T-Coffee[8], MAFFT[9], MUSCLE[10] etc. use the NJ method as the main option. But the time complexity of the original NJ algorithm is $O(N^5)$, where N is the number of operational taxonomic units (OTUs). For a part of this reason, MUSCLE and MAFFT also use the UPGMA[11] to construct trees since the time complexity of it is only $O(N^3)$. However, the UPGMA does not build the true evolutionary tree to guide a progressive alignment in line with biological expectations though it may sometimes get a higher SP score than the NJ method[10]. Studier and Keppler[5] succeeded in reducing the time complexity of NJ from $O(N^5)$ to $O(N^3)$, but there is still space to reduce the executing time for the NJ algorithm. The time complexity of the proposed algorithm in this paper is also $O(N^3)$; however, it is roughly two times faster than SK on sufficient data sets.

Received 2005-09-19.

Foundation item: The National Natural Science Foundation of China (No. 60473015).

Biographies: Chen Ningtao (1976—), male, graduate; Shi Baochang (corresponding author), male, doctor, professor, sbchust0382 @ sina. com.

1 NJ Algorithm

Neighbor-joining seeks to build a tree which minimizes the sum of all branch lengths; i. e., it adopts the minimum-evolution (ME) criterion. Many studies have corroborated NJ's performance in reconstructing correct evolutionary trees. For small numbers of taxa, NJ solutions are likely to be identical to the optimal ME tree[6]. Neighbor-joining begins with a star tree, then iteratively finds the nearest neighboring pair (i. e., the pair that induces a tree of the minimum sum of branch lengths) among all possible pairs of nodes (both internal and external). The nearest pair is then clustered into a new internal node, and the distances of this node to the rest of the nodes in the tree are computed and used in later iterations. The algorithm terminates when $N-2$ internal nodes have been inserted into the tree (i. e., when the star tree is fully resolved into a binary tree). The framework of NJ is defined by three components: the criterion used to select pairs of nodes; the formula used to reduce the distance matrix; and the branch length estimation formula. Fig. 1 shows an example of the NJ tree, which indicates the evolutionary relationship of eight OTUs and which is an unrooted tree. The rectangles denote the terminal nodes or OTUs and the circles denote the internal nodes, respectively. The numbers beside the rectangles denote OTUs; the numbers beside the circles denote the orders that OTUs are inserted into the tree, and the data above the lines are the branch lengths of two neighbors. It is obvious that OTUs 1 and 2 are clustered first, generating a new combined OTU. The branch lengths between 1 and the new OTU and 2 and the new OTU are 5 and 2, respectively. After OTUs 7 and 8 are clustered, there are only two OTUs, and they are clustered. The tree is built now. The NJ algorithm is described briefly as follows.

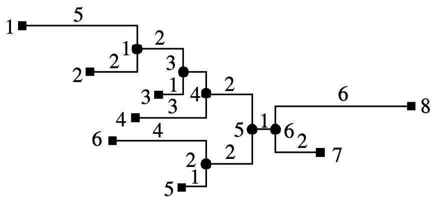

Fig. 1 Neighbor-joining tree

Define the following symbols: D_{ij} and L_{ab} as the distance between OTUs i and j and the branch length between nodes a and b, respectively; S_{ij} as the sum of all the branch lengths of OTUs i and j. Note that $\boldsymbol{D}$ is a symmetric square matrix but not a triangle matrix as $\boldsymbol{S}$.

$$S_{ij} = \frac{1}{2(N-2)}\sum_{k=0}^{N-1}(D_{ik}+D_{jk}) + \frac{1}{2}D_{ij} + \frac{1}{N-2}\sum_{m<n}D_{mn} \tag{1}$$

where $i=0, 1, \ldots, N-2$; $j=i+1, i+2, \ldots, N-1$; $k, m, n \neq i, j$.

S_{ij} is the sum of the least-squares estimates of branch lengths[4]. At every iterative step, the minimum S_{ij} indicates that nodes i and j are the nearest evolutionary neighbors.

Suppose that OTUs i and j are the neighbors to be joined in the tree. They are clustered as a new one, denoted by X. The distance between this combined OTU X and another OTU k is given by

$$D_{Xk} = \frac{D_{ik}+D_{jk}}{2} \qquad k=0, 1, \ldots, N-1; k\neq i, j \tag{2}$$

L_{iX} and L_{jX} are estimated by

$$L_{iX} = \frac{D_{ij}+D_{iZ}-D_{jZ}}{2},\ L_{jX} = \frac{D_{ij}+D_{jZ}-D_{iZ}}{2} \tag{3}$$

where $D_{iZ} = \sum_{k=0}^{N-1} D_{ik}/(N-2)$, $D_{jZ} = \sum_{k=0}^{N-1} D_{jk}/(N-2)$, $k \neq i, j$; Z represents a group of OTUs including all but i and j, and D_{iZ} and D_{jZ} are the distances between i and Z and j and Z, respectively. Finally, OTUs i and j are deleted from the current OTUs lists, X is added, and the number of current OTUs reduces one, and a new iteration starts.

The pseudo code of the NJ algorithm is described as follows:

```
Input: $N$ is the number of OTUs, $D_{ij}$ is the distance between any pair OTUs, and $D_{ii}$ is set to infinity.
Output: NJ tree.
1 While $N>2$ do
  1. 1 For $i=0$ to $N-1$ do
  1. 2 For $j=i+1$ to $N$ do
    1. 2. 1 Compute $S_{ij}$ according to Eq. (1).
  1. 3 Search the minimum $S_{ij}$ to get neighbors $i$ and $j$.
  1. 4 OTUs $i$ and $j$ are clustered as a new OTU $X$.
  1. 5 Compute $D_{Xk}$ according to Eq. (2).
  1. 6 Compute branch lengths $L_{iX}$ and $L_{jX}$ according to Eq. (3).
  1. 7 Delete OTUs $i$ and $j$, and add $X$ to current OTU lists.
  1. 8 $N=N-1$.
  End of while
2 Cluster the last two OTUs.
```

2 Time Complexity Analysis of NJ Algorithm

According to the NJ's pseudo code, the time complexity of the outer most loop is $O(N)$; lines 1. 1 and 1. 2 are $O(N^2)$; line 1. 3 is $O(N^2)$; line 1. 4 is $O(1)$; line 1. 5 is $O(N)$; line 1. 6 is $O(N)$; line 1. 7 is $O(1)$ and line 2 is $O(N)$. As for line 1. 2. 1, because Eq. (1) comprises three parts, we will analyze the time complexity of each separately. The first part contributes the time $O(N)$, the second is $O(1)$ and the third is $O(N^2)$ because we should consider all the m and n but i and j. So the most expensive step is Eq. (1), which induces that the total time of NJ is $O(N^5)$ if we compute it directly. So the key point to optimize the NJ algorithm is to optimize the computation of Eq. (1).

For the three parts of Eq. (1), there are in fact many repeated computations of the same data in parts 1 and 3. For example, in part 1, when computing S_{12}, D_{13} is added; when computing S_{14}, D_{13} is also added. In part 3, when computing S_{12}, D_{34} is added; when computing S_{15}, D_{34} is also added. In other words, there are close relationships between the computations of S_{ij}. Though the NJ algorithm should compute the triangle matrix $\boldsymbol{S}$ iteratively to search the nearest neighbors, it is fortunate that the computations of S_{ij} only need the distance matrix $\boldsymbol{D}$, which stimulates us to design the fast algorithm to avoid the repeated computations in $\boldsymbol{D}$.

3 Fast Algorithm for Constructing NJ Tree

Our motivation is very simple: we try to reduce the computations in the inner-most loop. There are the most expensive computations for the time complexity of the whole algorithm. Pre-processing outside the double loops of lines 1. 1 to 1. 2 of the pseudo code has been done, the results of which are used for the computations of S_{ij}(line 1. 2. 1).

Define a linear array $A[N]$ and a buffer t, the former is used to store the sum of every row of $\boldsymbol{D}$ and the latter to store all the sum of $\boldsymbol{D}$, then

$$A_i = \sum_{j=0}^{N-1} D_{ij} \qquad j \neq i \tag{4}$$

$$t = \sum_{i=0}^{N-1} A_i \tag{5}$$

Define the first and third parts of S_{ij} as v and w,

$$v \overset{\text{def}}{=} \frac{A_i + A_j - 2D_{ij}}{2(N-2)} \tag{6}$$

$$w \overset{\text{def}}{=} \frac{t - 2(A_i + A_j) + 2D_{ij}}{2(N-2)} \tag{7}$$

Then

$$S_{ij} = v + \frac{1}{2}D_{ij} + w = \frac{t - A_i - A_j}{2(N-2)} + \frac{1}{2}D_{ij} \tag{8}$$

Compared with Eq. (1), this modified Eq. (8) is simpler and more efficient. To compute A_i and t, the time complexities are $O(N^2)$ and $O(N)$, respectively. But their computation is parallel to the double loops of lines 1.1 to 1.2 of the pseudo code of the NJ algorithm, which does not increase the total time complexity. And the time complexity of Eq. (8) is only $O(1)$.

Proof ① For v,

$$A_i - D_{ij} = \sum_{k=0}^{N-1} D_{ik}(k \neq i) - D_{ij} = \sum_{k=0}^{N-1} D_{ik}(k \neq i,j)$$

$$A_j - D_{ij} = \sum_{k=0}^{N-1} D_{jk}(k \neq j) - D_{ij} = \sum_{k=0}^{N-1} D_{jk}(k \neq i,j)$$

$$(A_i - D_{ij}) + (A_j - D_{ij}) = A_i + A_j - 2D_{ij} = \sum_{k=0}^{N-1} (D_{ik} + D_{jk})(k \neq i,j)$$

② For w, $A_i + A_j$ denotes the sum of rows and columns of i and j, respectively; $t - 2(A_i + A_j) + 2D_{ij}$ denotes the sum of the remaining elements of $\boldsymbol{D}$ except those elements related to i and j; $2D_{ij}$ is added because D_{ij} and D_{ji} are deleted twice from rows and columns of i and j. Then

$$t - 2(A_i + A_j) + 2D_{ij} = \sum_{m<n} D_{mn} \qquad m,n \neq i,j$$

Considering Eq. (8) again, t and N are constants for every S_{ij} at each iteration, so they need not be computed to search the minimum S_{ij}, and the constant factor 1/2 is also ignored, then we can get a more simple form of S_{ij}(Eq. (9)) compared with Eq. (8).

$$S_{ij} = D_{ij} - \frac{A_i + A_j}{N-2} \tag{9}$$

Eq. (9) does not include t, which implies that we need not compute the sum of A_i any more as in Eq. (5) iteratively. This saves $O(N)$ time computation.

Considering the distance matrix $\boldsymbol{D}$, suppose that OTUs i^* and j^* are the neighbors to be joined in the tree, then the two rows and columns of i^* and j^* are deleted from $\boldsymbol{D}$, and a new row and column (denoted by X) are added. D_{Xk} is computed according to Eq. (2). In fact, we need not compute A_i again as in Eq. (4), because most values of the remaining A_i are unchanged between every two successive iteration steps. A_X can be computed in $O(N)$ time. For the remaining A_i, only D_{ii^*} and D_{ij^*} are deleted, and D_{iX} is added. So

$$A_i = A_i - D_{ii^*} - D_{ij^*} + D_{iX} \tag{10}$$

which saves $O(N^2)$ time for each iteration. Then Eq. (3) can be rewritten as

$$D_{iZ} = \frac{A_i - D_{ij}}{N-2}, \qquad D_{jZ} = \frac{A_j - D_{ij}}{N-2} \tag{11}$$

which saves $O(N)$ time compared with Eq. (3). Now the new algorithm is described as follows:

```
Compute $A_i$ according to Eq. (4).
While $N>2$ do
  For $i=0$ to $N-1$ do
  For $j=i+1$ to $N$ do
    Compute $S_{ij}$ according to Eq. (9).
  Search the minimum $S_{ij}$ to get neighbors $i^*$ and $j^*$.
  OTUs $i^*$ and $j^*$ are clustered as a new OTU $X$.
  Compute $D_{Xk}$ according to Eq. (2).
  Compute branch lengths $L_{iX}$ and $L_{jX}$ according to Eqs. (3) and (11).
  Delete OTUs $i^*$ and $j^*$, and add $X$ to current OTU lists.
  Update the remaining $A_i$ according to Eq. (10) and compute $A_X$.
  $N=N-1$.
End of while
Cluster the last two OTUs.
```

The time complexity of the new algorithm is $O(N^3)$.

4 Results and Discussion

Studies are performed on a main frequency 2.4 GHz Intel Celeron personal computer (256 MB RAM) with a Windows 2000 operating system. The distance matrix $\boldsymbol{D}$ is generated by three methods: ① Global pairwise alignments by dynamic programming, which is the slowest but the most accurate; ② K-mer distance[10,12], which is linear time complexity for pairwise distance estimation; ③ Random generation by computing pseudo numbers, which is just to provide a fast method to generate data.

To prove the correctness and high efficiency of our algorithm, a comparison has been made with Saitou and Nei's algorithm (SN)[4] and Studier and Keppler's algorithm (SK)[5] on sufficient datasets. The three algorithms can get the same results for the same input, but our algorithm is from tens to hundreds times faster than SN and roughly two times faster than SK when N increases. Fig. 2 shows the comparison of the three algorithms. Our algorithm is able to construct the tree with 2 000 OTUs in 3 min on a current desktop computer. As for Studier and Keppler's algorithm[5], we should point out that they introduced a similar technique compared to ours, the formula of S_{ij} is as

$$S_{ij} = D_{ij}(N-2) - (A_i + A_j) \qquad (12)$$

However they calculated t and all the A_i iteratively, for which there are many repeated computations as we have shown in section 3.

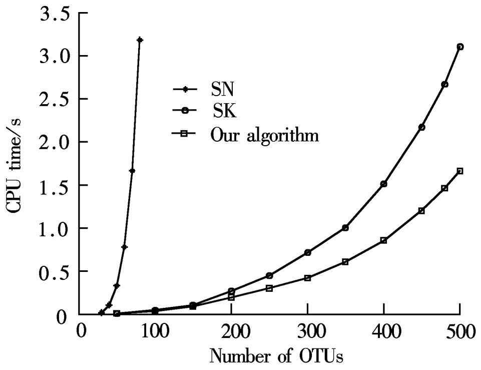

Fig. 2 Comparison of the three algorithms

5 Conclusion

In this paper, a fast algorithm for constructing neighbor-joining phylogenetic trees has been developed, which is $O(N^3)$ time complexity compared with $O(N^5)$ of SN, the same as SK, while faster than the latter. The key point of our algorithm is to reduce the repeated computations. Our work serves as an example of how to reduce the time complexity when there are many loops in programs for some algorithms. We should try to reduce the computations in the inner-most loop. This may be done by studying the relationships between the elements to be computed.

References

[1] Yang C, Khuri S. PTC: an interactive tool for phylogenetic tree construction [A]. In: *IEEE Proceedings of the Computational Systems Bioinformatics*[C]. California: Stanford University, 2003, **8**: 476 – 477.

[2] Foulds L R, Graham R L. The sterner problem in phylogeny is NP-complete [J]. *Advances Appl Math*, 1982, **3**: 43 – 49.

[3] Day W. Computational complexity of inferring phylogenies from dissimilarity matrices [J]. *Bull Math Biol*, 1987, **49** (4): 461 – 467.

[4] Saitou N, Nei M. The neighbor-joining method: a new method for reconstructing phylogenetic trees [J]. *Mol Biol Evol*, 1987, **4**(4): 406 – 425.

[5] Studier J A, Keppler K J. A note on the neighbor-joining algorithm of Saitou and Nei [J]. *Mol Biol Evol*, 1988, **5** (6): 729 – 731.

[6] Saitou N, Imanishi T. Relative efficiencies of the Fitch-Margoliash, maximum-parsimony, maximum-likelihood, minimum-evolution, and neighbor-joining methods of phylogenetic tree construction in obtaining the correct tree [J]. *Mol Biol Evol*, 1989, **6**(5): 514 – 525.

[7] Thompson J D, Higgins D G, Gibson T J. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice [J]. *Nucleic Acids Res*, 1994, **22**(22): 4673 – 4680.

[8] Notredame C, Higgins D G, Heringa J. T-Coffee: a novel method for fast and accurate multiple sequence alignment [J]. *J Mol Biol*, 2000, **302**(1): 205 – 217.

[9] Katoh K, Misawa K, Kuma K, et al. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform [J]. *Nucleic Acids Res*, 2002, **30** (14): 3059 – 3066.

[10] Edgar R C. MUSCLE: multiple sequence alignment with high accuracy and high throughput [J]. *Nucleic Acids Res*, 2004, **32**(5): 1792 – 1797.

[11] Sneath P H A, Sokal R R. *Numerical taxonomy* [M]. San Francisco: Freeman, 1973.

[12] Edgar R C. Local homology recognition and distance measures in linear time using compressed amino acid alphabets [J]. *Nucleic Acids Res*, 2004, **32**(1): 380 – 385.

创建 neighbor-joining 进化树的快速算法

陈宁涛[1]　王能超[2]　施保昌[2]

([1] 华中科技大学计算机科学与技术学院,武汉 430074)
([2] 华中科技大学并行计算研究所,武汉 430074)

摘要:为了改善 Saitou 和 Nei 提出的 neighbor-joining 进化树算法(SN)及 Studier 和 Keppler 的改进算法(SK),降低计算的时间复杂度,设计了一种快速算法. 该算法涉及 3 种技术:第一,引入一个线性数组 $A[N]$,用于存储距离矩阵每一行的值,以减少许多重复计算;第二,$A[i]$的值在算法开始时全部计算,在迭代步中间只进行更新 3 个变化的值;第三,设计了一个紧凑的公式用于计算 OTUs 之间的边长,并对该公式进行了证明. 实验结果表明:随着节点数的增多,该算法比 SN 算法快几十倍到上百倍,比 SK 算法快 2 倍以上;在一台桌面计算机上,该算法能在 3 min 左右创建具有 2 000 个节点的进化树. 以空间换时间,减少最内层循环的计算量是设计多重循环算法的基本思路.

关键词:进化树;邻接法;快速算法;进化多序列比对

中图分类号:TP301. 6