Phylograph: A multifunction Java editor for handling Phylogenetic trees

Lloréns, C.[1,2] Futami, R. [1] Panadero, J.,[3] Celda, B.[4] and Moya, A.[2]

1- Biotech Vana, Valencia, Spain

2- Instituto Cavanilles de Biodiversidad y Biología Evolutiva
Universitat de Valencia, Spain

3-Centro Nacional de Biotecnología, Madrid, Spain

4- Servicio Central de Soporte a la Investigación Experimental
Universitad de Valencia, Spain

Correspondence should be addressed to:

carlos.llorens@uv.es
carlos.llorens@biotechvana.org

Key Words
Large trees; Multifunctional Editing; Java; Layout algorithms

Running Head: Phylograph

Abstract

In this paper we introduce Phylograph; a multifunctional and powerful tree editor particularly indicated for large Phylogenetic trees. Phylograph reads tree Newick and Nexus descriptions from an input file and displays a graph drawing of the tree. Several options facilitate the representation of the tree in different editable formats, including a HTML format suited to databases, in which links to accessions of online databases may be incorporated in output files. Phylograph easily roots the tree using one or more outgroups, simply via the computer mouse. It incorporates a wide set of functions to quickly expand, compress, invert, and/or rotate the tree. Phylograph also allows the cutting of branches and the incorporation of features and decorations such as brackets, boxes, and arrows. The program is a Java application. This means that it runs on most personal computers and workstations as a standard program. We also present here an overview of the algorithms used by Phylograph to represent Tree drawings of both, rooted and unrooted trees.

Availability: Biotech Vana Bioinformatics (http://biotechvana.org)

Introduction

Based on similarity scoring, Phylogenetic analyses reconstruct the evolutionary history of biological species, genes, and proteins taken as Operative Taxonomical Units (OTUs). Each branch or cluster depicted in the Phylogenetic reconstruction usually represents a clade or monophyletic group comprised of all the sampled descendants of a single ancestral lineage. Three inference methods based on three optimization criteria are commonly used to reconstruct evolutionary history from molecular data: neighbour joining (Saitou and Nei 1987), Parsimony (Eck and Dayhoff 1966; Kluge and Farris 1969), and maximum likelihood (Aldrich 1997 and references therein). The overall

efficiency of these methods in reconstructing the true tree is known to fluctuate in substitution rate, transition-transversion ratio, and sequence divergence (Miyamoto and Cracraft 1991; Nei and Kumar 2000). Results are usually saved in two of the most commonly accepted formats: Nexus and Newick (Maddison et al. 1997; http://evolution.gs.washington.edu/phylip/newicktree.html). To be phylogenetically interpreted, the information contained in both Newick and Nexus formats should be graphically represented as a branching tree, using tree editors, where each node with descendants represents the most recent common ancestor of the descendants, and edge lengths correspond to time estimates. With the current explosion in the amount of genomic data available, and exponential increases in computing power, biologists are able to consider larger scale problems in Phylogeny: that is, the reconstruction of evolutionary trees on hundreds or thousands of OTUs. One difficulty arises, however, in the interpretation and analysis of large trees; leaves overlap and font sizes are usually too small to be easily read. Therefore, large trees must be magnified or expanded to be clearly interpreted. Moreover, to be publishable, Phylogenetic trees usually require certain modifications and decorations that require the use of additional image editors. We have designed Phylograph, as a multifunctional application, to be a versatile tree editor capable of managing both large and small trees. It can handle, edit, and decorate all kinds of Phylogenetic trees and export them in different kinds of outputs.

Phylograph overview

System

Phylograph is a Java application that runs on most personal computers and workstations as a standard program. The Model View Controller “MVC” (a programming pattern to maintain the independence data and visualization), has been used to divide the application into three different layers (Model, View, and Controller);

The model layer contains the program's logic and executable functions; The view layer defines the graphical user's interface, presenting all visual elements in a main window (buttons, lists, text-fields, etc); and the controller layer provides the connection between the other two layers.

Functions

Phylograph is based on a main graphical interface (window) that incorporates a control panel (Figure1), which provides a set of options, as shown in Table 1. Phylograph also allows users to, via the computer mouse, rotate the tree, root the tree with one or more outgroups, hide branches; generate subtrees, change the colors of branches and decorate the tree with labels and brackets, which may be dragged or, in the case of brackets, resized as desired.

Drawing tree algorithms

Graphical representations of Phylogenetic trees constitute a particular case of graphs (for an extensive overview of graphs see Sugiyama 2002). According to the graph theory, a tree $T\ (V,E)$ is an abstract structure used to describe a limited set of nodes or vertices (V) connected by edges (E) or segments not allowed to overcross. A graph drawing is the spatial or graphical representation of the graph (in this case the graph drawing is a tree drawing). So the tree is transformed by "divide and conquer" principles, from the abstract representation $T\ (V,E)$ into some arrangement of geometric objects (subtrees) enclosed in a multi-dimensional space called the drawing space, where $1 \leq m$ and $m \in N$. The drawing space for representing Phylogenetic trees is usually two-dimensional and more rarely three-dimensional. Phylogenetic trees $T\ (V,E,\delta)$ usually incorporate the length (δ) or extent of an edge as an additional variable usually defined by the genetic or protein distance between two nodes, or, in the case of majority-rule consensus trees generated by programs such as Consense of Phylip

(Margush and McMorris 1981; Felsenstein 2002), by numbers that correspond to consensus values defined by all groups occurring more than a certain percentage level in a consensus tree reconstructed from a file containing a certain number of trees. There are essentially two concepts for achieving drawings of Phylogenetic trees, rooted and unrooted (for an overview of Phylogenetic trees see Carrizo 2004). Rooted trees are "n-ary" trees, where there is a specially designated node (the root), which is the common ancestor for the remaining nodes in a hierarchy of parents and children (note that a tree is "n-ary" if every internal node has no more than "n" children). OTUs are called leaves and nodes that are not leaves are called internal nodes. Children of the same parent are called siblings. Nodes are partitioned into subtrees where the level of a node is defined by letting the root at level zero, and a node at level "l" has children at level ("l"+1). The number of subtrees of each node is called its degree, and the maximum degree of all nodes is called the degree of the tree. The degree of a node is usually defined by letting the OTUs at degree zero. Unrooted trees represent the branching order, but do not indicate the root, or location, of the last common ancestor. Based on algorithm 1, a rooted tree is easy to layout by recursion (for an extensive overview of recursion see Shoenfield 2000; Causey 2001; Cori, Lascar, and Pelletier 2001). Basically, as shown in Figure 3, Phylograph visits in preorder traversal a given thrichotomic tree *T (N,E,δ),* takes the first open bracket as the root (Node 0) and recursively split it into subtrees to store the information. In other words, Phylograph takes the root and step forward along the upper pathway of open brackets (nodes) and detects a name or character defined by a comma (OTU J). As the tree is thrichotomic, the algorithm steps backward looking for new commas (Nodes 1 and 2 respectively), and steps forward again in order to find another character defined by a comma (OTU A). As descent is not allowed for OTUs, the next step is to read the sibling of OTU A that is Node 3, which, is at the same time

parent of OTUs B and C. Next, the algorithm step backwards to achieve again node 1 and visits Node 4→ OTU F→ Node 5 → OTU D → OTU E, and repeats the process with the subtree defined by Node 6. The tree and nodes are thus reordered by degree of subtrees. Finally, all the information concerning the tree is stored in a virtual list from which Phylograph allows the user to depict the tree drawing as both, slanted and rectangular cladograms, and as a Phylogram. Radial trees are layout via algorithm 2, which is a linear-time algorithm adapted with several modifications from Bachmaier et al. (2005). As shown in Figure 4, Phylograph remove the root and reconsiders subtree levels to establish a new node at level zero. To achieve a more aesthetically balanced structural fold Phylograph virtually identifies the longest chain of nodes. In accordance with this virtual shape, it reorders the subtree allocations. Then, the tree is visited in preorder traversal and all vertices are assigned a wedge "ω" of angular width proportional to its number of leaves, as shown in Figure 5. Then, the wedge of a given inner vertex "*u"* is divided among its children *"v",* and children allocated in the drawing space. To improve and homogenize the amount of space needed by both small and big subtrees, we have designed an equal distant wedges method, as summarized in algorithm 3. Starting from the root, the algorithm visits each node and creates a wedge for each visible subtree from that node and swings these until the arcs of separation between wedges are equal. Thus, by symmetry and the better utilization of misused spaces, a more harmonic visualization of the radial tree is achieved, as shown in Figure 6.

Concluding remarks

Phylograph´s applicability has been inspired by the versatilities of other applications such as TreeView (Page 1996), Drawgram/Drawtree (Felsenstein 2002), and Baobab (Dultier and Galtier 2002). The most remarkable requirements in

Phylograph´s development were provided by the Gypsy database project (Lloréns et. in preparation). We conduct this online database, analyzing and classifying LTR retroelements, according to their evolutionary perspectives. In this project, Phylogenies are large and in continuous progress. Therefore, the required tree editor would be expected to have the capacity to dynamically manage, edit, root, and decorate graphical representation of large Phylograms and cladograms, as well as making it very easy to use in the handling of any kind of tree. We have satisfactory checked Phylograph in this project; particularly the HTML output generated by Phylograph is quite useful for easily linking Phylogenetic data to other information such us genbank accessions etc.

Literature

Aldrich, J., 1997. R.A. Fischer and the making of Maximum Likelihood 1912-1922. Statistical Science. 12: 162-176

Bachmaier, C., Brandes, U., and Schlieper, B. 2005. Drawing Phylogenetic Trees (Extended Abstract) In Deng and D. Du (Eds.): ISAAC 2005, LNCS 3827, pp. 1110–1121.

Carrizo, S. F. 2004. Phylogenetic trees: An information visualization perspective. In Y.-P. Phoebe Chen, editor, *Asia-Pacific Bioinformatics Conference (APBC 2004)*, volume 29 of *CRPIT*, pages 315–320.

Causey, R.L. 2001. Logic, Sets, and Recursion. Jones & Bartlett.

Cori, R., Lascar, D., and Pelletier, D. H. 2001. Recursion Theory, Godel's Theorems, Set Theory, Model Theory. Oxford University Press.

Dultier, J. and Galtier, N. 2002. Baobab: A Java tree editor for large Phylogenetic trees. Bioinformatics. 18(6): 892-893.

Eck, R. V., and Dayhoff, M. O. 1966. Atlas of Protein Sequence and Structure. National Biomedical Research Foundation, Silver Spring, Maryland.

Felsenstein, J. 2002. PHYLIP (Phylogeny Inference Package) version 3.6a3. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle.

Kluge, A. G., and J. S. Farris. 1969. Quantitative phyletics and the evolution of anurans. Systematic Zoology 18: 1-32.

Maddison, D.R., Swofford, D.L., and Maddison. W.P. 1997. NEXUS: an extensible file format for systematic information. Syst Biol. 46(4): 590-621.

Margus, T. and McMorris, F.R. 1981. Consensus n-trees. Bull. Math. Biol. 43: 239-244.

Miyamoto, M. M., and Cracraft, J. L.1991. Phylogenetic inference, DNA sequence analysis, and the future of molecular systematics. Pp. 3–17 *in* M. M. Miyamoto and J. L. Cracraft, eds. Recent advances in Phylogenetic studies of DNA sequences. Oxford University Press, Oxford, England.

Nei, M., and Kumar, S. 2000. Molecular evolution and Phylogenetics. Oxford University Press, Oxford, England.

Page, R. D. M. 1996. TREEVIEW: An application to display Phylogenetic trees on personal computers. Computer Applications in the Biosciences **12**: 357-358.

Saitou, N. and M. Nei. 1987. The Neighbor-joining method: a new method for reconstructing Phylogenetic trees. Mol.Biol.Evol. **12**: 850-862.

Shoenfield, J. R. 2000. Recursion Theory. A K Peters Ltd.

Sugiyama, K. 2002. Graph drawing and applications for software and knowledge engineers.World, Scientific Series on Software Engineering and Knowledge Engineering Vol 11.

Figure legends

Figure 1

Phylograph´s control panel.

Figure 2

Screen shot of Phylograph functions executable with the mouse

Figure 3

Recursive layout of a Phylogenetic tree

Figure 4

Reordering the tree to draw a radial tree

Figure 5

Radial tree drawing

Figure 6

unrooted tree optimization

Table 1 Phylo

Phylo functions
1. Open a tree file; save graphical representation as a project; re-edit the projects; exit
2. Choose size, type of fonts and colors for OTUs, labels, bootstrap values, links and attachments
3. Export the tree as an image (in png format); as a postcript file; and as format suitable for databases. The database format combines an HTML file with a png file; each OTU depicted in tree may incorporate a link to other HTML files and to Websites and online databases.
4. Import, export, and edit a list of attachments specifics of each OUT; the list may be written and modified by users in a simple text file.
5. Import, export, and edit a list of online addresses (for instance genbank accessions) that specifically link each OTU to other files and databases. This function is only available to generate HTML outputs; the list may be also written and modified in a text file.
6. Compare two trees via two parallel windows, each one occupying half of the screen.
7. Help
8. Open a treefile
9. Open a project file
10. Save a treefile project
11. Step-by step horizontally expand a tree (specially indicated for large trees)
12. Step-by step horizontally compress a tree (specially indicated for large trees)
13. Step-by step vertically expand a tree (specially indicated for large trees)
14. Step-by step vertically to compress a tree (specially indicated for large trees)
15. Expands the entire tree (specially indicated for large trees)
16. Compress the tree (specially indicated for large trees)
17. Fit the tree in the window
18. Move the tree right
19. Move the tree left
20. Move the tree up
21. Move the tree down.
22. Shows or hide bootstrap values upper to a given number (by default zero).
23. Invert the tree
24. Depict the tree as a radial (unrooted) tree
25. Depict the tree as a rectangular cladogram
26. Depict the tree as a slanted cladogram
27. Depict the tree as a phylogram

Algorithm 1: tree layout

Input: Newick or Nexus tree

Data: δ : Edge lengths
B : bootstrap values
"(" : Open parenthesis → vertices or nodes (step forward)
L : Leaves or OTUs (any number, word, or character defined by a comma)
")": Close parenthesis → vertices or nodes (step backwards)
"," : Vertices or leaves separator
":" : Bootstrap and edge length values separator
";" : Tree end and optional information separator
M : Optional information concerning the tree identity (n-ary tree)

Output: spanning ordered tree $T(V, E, \delta)$

1. If the input-tree is provided in nexus format the program directly turns the tree from Nexus format to Newick format
2. Read the Newick tree from the left to the right
3. Optional → remove the text behind the Newick tree end to delete possible commentaries, and store the tree identity (n-ary tree)
4. Let first parenthesis as vertex 0 (root). Then create a vertex in drawing space and remove the first open parenthesis and the last closed parenthesis in Newick tree

    a. Search for colons; If colons are not found, the Newick tree has not edge lengths or bootstrap values. Then, fix edge length values =1 for all vertices

5. Search recursively for next vertex in the resultant Newick subtree

    i. Search for commas in the resultant Newick subtree.

    ii. If a comma is found; then such subtree is a vertex.
    iii. Else, such subtree is a leaf

    iv. Add a new vertex to the spanning tree

    v. If the Newick tree has distances; search for the last colon; else, assign edge length = 1 to this vertex.

        1. If the next Newick subtree defines a vertex, then load the text behind the last closed parenthesis. Get texts in front and behind the colon (bootstrap and edge length values); then, remove them

        2. If the next Newick subtree defines an leaf, store the text behind the colon (edge length); then remove it. If the leaf name is surrounded with quotes, remove the quotes.

        3. Assign bootstrap and edge length values to this vertex

    vi. If this vertex corresponds to a leaf, algorithm finishes for this branch (descent is not allowed for leaves).

    vii. Else, if the vertex corresponds to an internal vertex; then remove the first open and the last closed
    viii. parentheses of the Newick subtree; then separate inner subtrees of the same level from subtree. Repeat recursively this process with each internal vertex until reach a leaf again

6. Preorder traversal and set level, degree, and number of leaves of the spanning tree $T(V, E, \delta)$
7. Allocate x, y coordinates for all $v \in T(V, E, \delta)$
    a. Let in preorder traversal, X coordinates for all *v* following
        i. $X_v = X_u + \delta(u, v)$
            1. where (Xroot = 0) and $u \leftarrow parent(v)$.
    b. Let Y coordinates for all *v* following
        i. For leaves → in preorder traversal following $Y_v = n_v \cdot k$
            1. where"n" is the number of leaf assigned to "*v*" and "*k*" an arbitrary constant

        ii. For internal vertices → in postorder traversal following $Y_u = \frac{\bar{Y}_{v_n}}{2}$

Algorithm 2: radial layout

Input: rooted tree $T(V, E, \delta)$

Data: δ : edge lengths
l : vertex arrays (number of leaves or OTUs in subtrees)
degree: number of subtrees of a vertex.
level: vertex levels letting level zero for root.

Output: x, y coordinates for all $v \in V$

1. Remove the root from the rooted $T(V, E, \delta)$ and reconsider subtree levels, establishing a new node at level zero.

    a. If the degree of root equals 2 then $T(V, E, \delta)$ is a dichotomic; remove node 0 and let the last child of root at level zero (drawing root); then, add the first child of the old root to drawing root. At this point, the drawing root has degree 3.

    b. Else, if the degree of the root is higher than 2 then $T(V, E, \delta)$ is a multichotomic; the last child of root is set at level zero (drawing root) and its father (old root) is set at level one as another children of the drawing root and all vertex levels are relabeled.

2. If the edge of a given vertex has negative value, let this length according to the minimum length among all tree edge

3. Else, if edges have zero length, the tree is $T(V, E)$; then , let edge lengths equal 1.

4. In postorder traversal, identify the linearly-largest chain of vertices (number of vertices), then, reorder the spanning $T(V, E, \delta)$ to aesthetically equilibrate the further fold of the radial layout. Else, if are possible chains are equal then choose the first option loaded.

5. Allocate the coordinates of the drawing root (vertex 0) at the point (0, 0) of the drawing space with a virtual wedge of angular 2π width.

6. Identify subtrees *T(v)*; and let the number of leaves for each subtree, considering only leaves.

    a. Leaves have a value of 1.
    b. Vertices have a value equal to its number of descendants.

7. Do recursively in preorder traversal:

    a. Assign to each subtree *T(v)* a wedge "ω" of angular width proportional to its number of leaves in $T(V, E, \delta)$ according to:

$$\omega = 2\pi \cdot \frac{[leaves(T(v))]}{[leaves(T(V, E, \delta))]}$$

    b. Divide, proportionally, the wedge of a given inner vertex "*u"* among its children *"v"*

    c. Allocate "x, y" coordinates for all vertices in the drawing space, according to the algorithm below and rules 1-3:

$$(x, y)_v = (x, y)_u + \delta(u, v) \cdot (\cos(\tau + \psi_v), \sin(\tau + \psi_v))$$

    1. If "*v"* is a internal node then $\psi_v = \frac{\omega_v}{2}$

    2. If "*v"* is a leaf and the first child of *"u"* then $\psi_v = \frac{\omega_v}{100}$ → counterclockwise

    3. If "*v"* is a leaf and the last child of *"u"* then $\psi_v = \frac{99}{100} \cdot \omega_v$ → clockwise

Algorithm 3: unrooted tree optimization algorithm

Input: unrooted tree $T(V, E, \delta)$

Data: x, y coordinates from vertices
φ: angle of a vertex in polar coordinates
r: radius of a vertex in polar coordinates
τ: starting angle of a subtree
ώ: ampleness of a wedge

Output: optimized unrooted tree

8. Starting at root, in preorder traversal, do in each vertex:
   a. If the vertex is a node, set wedges for all visible subtrees from that node following these steps:
      i. In preorder traverse all the subtree, get the angle formed between each vertex of that subtree and the current working node transforming rectangular coordinates to polar by its x and y coordinates.
      The rightmost and leftmost angles of a subtree are the right and left borders of its wedge. This wedge defines a triangle of minimum area enclosing the subtree.

$$a = x_2 - x_1$$
$$b = y_2 - y_1$$
$$r = \sqrt{a^2 + b^2}$$
$$\cos(\phi) = a/r$$
$$\sin(\phi) = b/r$$

   b. Get the angle of separation values between the left border of each wedge and the right border of the next. If current node is the root w3 will be the third children, else, it will be the father subtree.

$$\alpha_1 = \omega 2_R - \omega 1_L$$
$$\alpha_2 = \omega 2_R - \omega 2_L$$
$$\alpha_3 = \omega 1_R - \omega 3_L$$

   c. Get the mean value of that separation values. At the end, all the separation between wedges will be set to this value.

$$\overline{\alpha} = (\sum \alpha)/3$$

   d. Get the difference between the separation value and the mean value to correct the position of each subtree.

$$\delta_1 = \overline{\alpha} - \alpha_1$$
$$\delta_2 = \overline{\alpha} - \alpha_2$$
$$\delta_3 = \overline{\alpha} - \alpha_3$$

   e. If the vertex is the root, it has three.
   The first children subtree will be kept fixed.
   Rotate second and third children subtree $vertex_\tau + \delta_1$
   Rotate third children subtree to $vertex_\tau + \delta_2$

   f. Else, it has two children subtrees and a father subtree. The father subtree will be kept fixed since it has been previously optimized.
   Rotate first and second children subtrees to $vertex_\tau + \delta_3$
   Rotate second children subtree to $vertex_\tau + \delta_1$

   g. Finally, translate polar coordinates into rectangular for graphical visualization.