Biotechvana Bioinformatics, Collection 2008. Software: Phylograph 1.0. ISSN 1988-7957

Phylograph: A multifunction Java editor for handling Phylogenetic trees

Lloréns, C.[1,2] Futami, R. [1] Vicente-Ripolles, M. [1,3] and Moya, A.[2,4]

1 - Biotechvana, Valencia, Spain
2 - Instituto Cavanilles de Biodiversitat i Biología Evolutiva Universitat de València, Spain
3 - Departament de Sistemas Informátics i Computació Universitat Politécnica de València
4 - CIBER de Epidemiología y Salud Pública (CIBERESP), Spain

Corresponding author: carlos.llorens@biotechvana.com

Availability: Available online February 1, 2008. Phylograph is distributed under the terms of the Biotechvana Private Source License (URL 1).

In this paper we introduce Phylograph; a multifunctional tree editor particularly indicated for large trees. The application reads trees up to 1000-1200 leaves and constructs and edits graph drawings in different layouts. Phylograph roots the tree using as outgroup a single leave or a whole branch, simply via the computer mouse. The tool incorporates a wide set of functions to expand, compress, invert, and/or rotate a tree. Phylograph allows the cutting of branches and the incorporation of decorations such as tags, brackets, boxes, and arrows. The tool also allows the user to save the tree drawing as a re-editable project and offers the choice of various exportable image formats, including a HTML format suited to databases. Phylograph is a Java application. This means that the tool runs on personal computers as a standalone program. We also present here an overview of the algorithms used by Phylograph to represent the tree drawings.

Keywords: Large trees | Multifunctional Editing | Java | Layout algorithms

INTRODUCTION

Phylogenetic analyses reconstruct, based on similarity scorinPhylogenetic analyses reconstruct the evolutionary history of biological species, genes, and proteins based on similarity scoring. A phylogenetic tree usually represents the evolutionary distances among the Operative taxonomical Units (OTUs), which are represented by leaves. Phylogenetic analyses usually save the outputs in two of the most commonly accepted formats - nexus (1) or newick (URL 2) – To phylogenetically interpret a tree users graphically represent it as a branching graph, where each node with descendants represents the most recent common ancestor of the descendants, and edge lengths correspond to time or distance estimates. This is, Phylogenetic trees constitute a particular case of graph theory (2) where OTUs are called leaves and branches not representing leaves are called nodes. Children of the same parent are called siblings. According to 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. The graph drawing is the spatial or graphical representation of the graph. The tree is transformed by "divide and conquer" principles, from an abstract representation *T (V,E)* into an arrangement of geometric objects (subtrees) enclosed in a multi-dimensional space called the drawing space. A tree *T (V,E,δ)* usually incorporates information regarding the length (δ) or extent of an edge as an additional variable defined by the genetic or protein distance between two nodes. In the case of majority-rule consensus trees (MRC) trees (3) this variable is defined by the numbers that correspond to consensus values defined by all groups occurring more than a certain percentage level. There are essentially two types of phylogenetic trees, rooted and unrooted (4). Rooted trees are "n-ary" trees where there is a specially designated node (the root) that is the common ancestor for the remaining nodes in a hierarchy of parents and children (a tree is "n-ary" if every internal node has no more than "n" children). Nodes are partitioned into subtrees where the level of a node is defined by letting the root at level zero. Note therefore that 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 (a subtree) 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. With the recent explosion in the amount of genomic data available, and exponential increases in computing power, biologists are currently able to consider larger scale problems in phylogeny. That supposes the construction of evolutionary trees on hundreds or thousands of taxa. When working with trees containing more than 50 OTUs some graphical problems arise in the interpretation of large trees; leaves overlap and font sizes are usually too small to be easily read. Consequently, trees must be magnified or expanded to be clearly interpreted. Graphical representations of phylogenetic trees usually need certain modifications and decorations that require the use of additional image editors. With the aim to obtain an editor capable of handle, edit, and decorate all kinds of phylogenetic trees we have designed Phylograph.

OVERVIEW

System

Phylograph is a Java application. This means that the tool runs on personal computers (PCs) and workstations as a standalone program. The Model View Controller "MVC"; a programming pattern to maintain the independence and visualization of data was used to divide the application into three 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 and presents all visual elements in a main window (buttons, lists, text-fields, etc). The controller layer provides the connection between the other two layers.

Functions

As shown in Figure 1, Phylograph allows user to manage tree description up to 1000-1200 leaves via a control panel that incorporates a wide set of options summarized in Table 1. Through the computer mouse, users can rotate and root the tree using as outgroup a single leave or a whole branch. The tool allows users to hide branches, generate subtrees, change the colors of branches and OTUs, etc. Users can also use Phylograph to decorate the tree with dynamic labels and brackets that may be dragged or resized (see the Section below, "Empirical example"). All implementations can be saved as a project to improve or modify the decoration and/or information background of the tree.

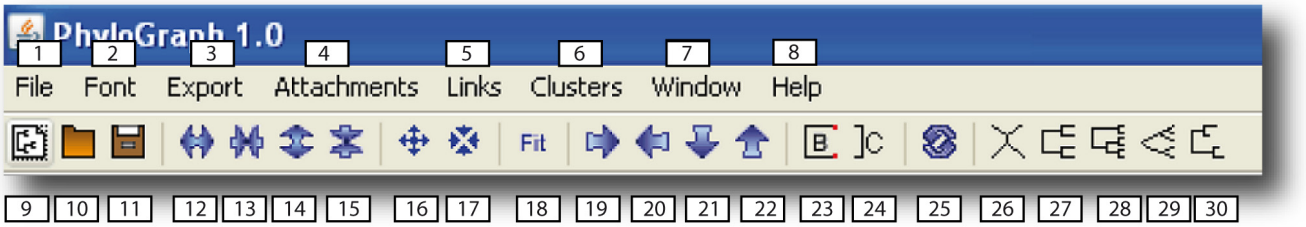


Figure 1. Phylograph´s control panel.

Table 1: Phylograph functions

1.	Open a tree file; save the tree as a project; re-edit the project; 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 a HTML format. This format combines an HTML file with a png file where each OTU depicted in the tree may incorporate a link to other HTML files
4.	Import, export and edit a list of information attachments specifics of each OTU (for instance scientific name), the list may be written and modified by users in a plain 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 when generating HTML outputs, the list may be written and modified in a plain file
6.	Create and save cluster annotations and decorations
7.	Compare two trees each one occupying half of the screen
8.	Help, this manual
9.	Open a treefile
10.	Open a project file
11.	Save a treefile project
12.	Horizontally expand a tree
13.	Horizontally compress a tree
14.	Vertically expand a tree
15.	Vertically to compress a tree
16.	Expands the entire tree
17.	Compress the entire tree
18.	Fit the tree in the window
19.	Move the tree right
20.	Move the tree left
21.	Move the tree up
22.	Move the tree down
23.	Shows or hide bootstrap values upper to a given number (by default zero).
24.	Create and save cluster annotations and decorations
25.	Invert the tree
26.	Depict the tree as a radial tree
27.	Depict the tree as a rectangular phenogram
28.	Depict the tree as a rectangular cladogram
29.	Depict the tree as a slanted cladogram
30.	Depict the tree as a phylogram

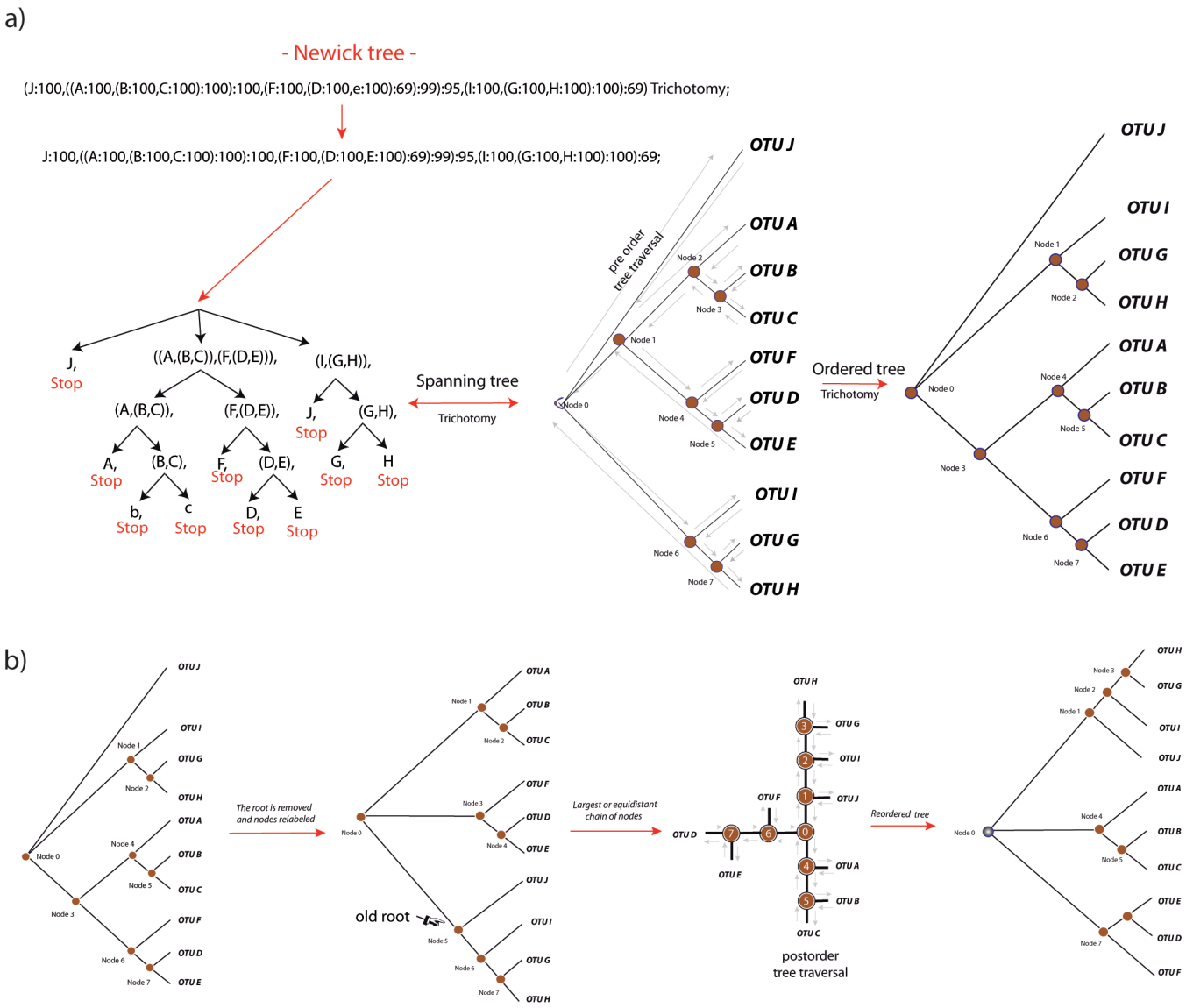


Figure 2. a) Recursive layout of a phylogenetic tree b) Reordering the tree to draw a radial tree

Methodology

There are essentially two concepts for achieving drawings of phylogenetic trees, rooted and unrooted or radial (4). Phylograph uses three algorithms to layout the graph: A rooted tree is easy to layout by recursion (5-7) using Algorithm 1. Radial trees are layout from the combination of Algorithms 2 and 3. Algorithm 2 is a linear-time algorithm adapted with several modifications from Bachmaier et al. (8). This algorithm resolves a radial layout that Phylograph re-visits to optimize the amount of space needed by both small and big subtrees, using the "equal distant wedges" algorithm summarized in Algorithm 3. Two examples follow:

Rooted trees: As shown in Figure 2, Phylograph visits in preorder traversal a given tree T (N,E,δ) and takes the first open bracket as the root (Node 0). The algorithm recursively splits the tree into subtrees and takes the root to step forward along the upper pathway of open brackets (nodes) in order to detect a name or character defined by a comma (OTU J, in the example). As the exemplified tree is thrichotomic, the algorithm steps backward looking for new commas (Nodes 1 and 2 respectively). From that point, the algorithm steps forward again looking for 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 parent of "OTUs B" and "C". In the next movement, algorithm 1 steps backward to reach Node 1. From that point it visits Node 4, OTU F, Node 5, OTU D, Node 5, OTU E and repeats the process considering the subtree defined by Node 6. Finally, tree and nodes are reordered by the degree of subtrees, and the information concerning the topology is stored in a virtual list with which Phylograph allows users to depict the drawing in several formats.

Radial trees: As shown in Figure 3a, radial trees are layout via algorithm 2, which removes the root reconsidering all subtree levels to establish a new node at level zero. Then, the program reorders subtree allocations and all vertices are assigned a wedge "ω" of angular width proportional to its number of leaves. Subsequently, Algorithm 3 re-visits each node and swings the nodes and the leaves until the arcs of separation between wedges are equal in symmetry and harmonic visualization (Figure 3b).

Algorithm 1. Tree layout

Input: Newick or other tree format

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 tree $T(V, E, \mathsf{d})$

1. If the input-tree is given in other format the program directly turns the tree from this 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 zero (root). Then create a vertex in the 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 a subtree is a vertex.
    iii. Else, such a 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 a 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
    parentheses of the Newick subtree; then separate inner subtrees of the same level from subtree.
    viii. 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 $T(V, E, \mathsf{d})$
7. Allocate x, y coordinates for all $v \in T(V, E, \mathsf{d})$
    a. Let in preorder traversal, X coordinates for all *v* following
        i. $X_v = X_u + \mathsf{d}(u, v)$
            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 $T(V, E, \mathsf{d})$

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 tree $T(V, E, \mathsf{d})$ and reconsider subtree levels, establishing a new node at level zero.

    a. If the degree of root equals 2 then $T(V, E, \mathsf{d})$ 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 $T(V, E, \mathsf{d})$ 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 $T(V, E, \mathsf{d})$ to aesthetically equilibrate the further fold of the radial layout. Else, if there are possible equal chains 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 $T(V, E, \mathsf{d})$ according to:

$$\mathsf{w} = 2\mathsf{p} \cdot \frac{[leaves(T(v))]}{[leaves(T(V, E, \mathsf{d}))]}$$

    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:

$$(x, y)_v = (x, y)_u + \mathsf{d}(u, v) \cdot (\cos(\mathsf{t} + \mathsf{y}_v)\ \sin(\mathsf{t} + \mathsf{y}_v)$$

    1. If *"v"* is a internal node then $\mathsf{y}_v = \frac{\mathsf{w}_v}{2}$

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

    3. If *"v"* is a leaf and the last child of *"u"* then $\mathsf{y}_v = \frac{9}{100} \cdot \mathsf{w}_v$ → clockwise

Algorithm 3. Unrooted tree optimization algorithm

Input: unrooted tree $T(V, E, \mathrm{d})$
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

1. 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 traversal 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.

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

   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.

$$\mathrm{a}_1 = \mathrm{w2}_R - \mathrm{w1}_L$$
$$\mathrm{a}_2 = \mathrm{w2}_R - \mathrm{w2}_L$$
$$\mathrm{a}_3 = \mathrm{w1}_R - \mathrm{w3}_L$$

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

$$\bar{\mathrm{a}} = (\sum \mathrm{a})/3$$

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

$$\mathrm{d}_1 = \bar{\mathrm{a}} - \mathrm{a}_1$$
$$\mathrm{d}_2 = \bar{\mathrm{a}} - \mathrm{a}_2$$
$$\mathrm{d}_3 = \bar{\mathrm{a}} - \mathrm{a}_3$$

   e. If the vertex is the root, it has three.
   The first children subtree will be kept fixed.
   Rotate second and third children subtree $vertex_t + \mathrm{d}_1$
   Rotate third children subtree to $vertex_t + \mathrm{d}_2$

   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 $vertex_t + \mathrm{d}_3$
   Rotate second children subtree to $vertex_t + \mathrm{d}_1$

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

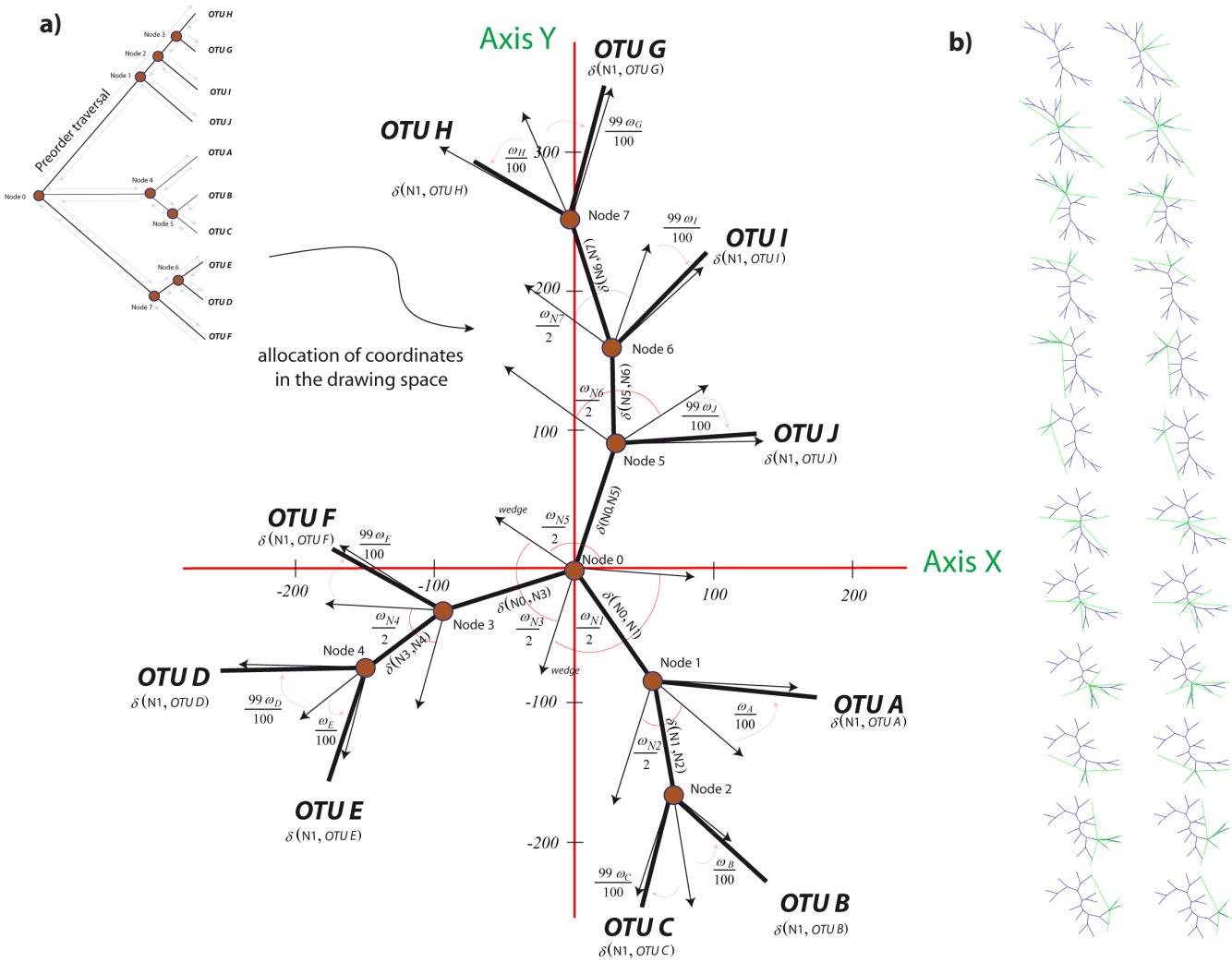


Figure 3. a) Radial tree drawing. b) Unrooted tree optimization

EMPIRICAL EXAMPLE

In Figure 4 we summarize a number of examples of the background of functions implemented in Phylograph:

a) Phylograph allows the user to open and fit multiple trees into the workspace and display multiple layouts at the same time clicking on "Window → Tile windows". Five types of layouts - unrooted, phylogram, slanted cladogram, rectangular cladogram and phenogram -, are allowed.

b) Trees can be rooted at any node and/or leave simply right-clicking the mouse and selecting the option "Set root" on the target node or OTU. The tree is rearranged based on the new root. The tree can be returned to its original topology right-clicking on any node or OTU and selecting "Unroot".

c) Phylograph also allows users to edit subtrees right-clicking in any node in the tree and selecting "Edit subtree". The new subtree is opened automatically in an individual window and can be edited separately without affecting its original parent tree.

d) User can hide/show one or more nodes simply right-clicking on each node and selecting "Hide/Show node".

e) Font and color preferences for all components of Phylograph can be edited by clicking on "Font → Set default fonts". A graphical dialog allows users to personalize all components (OTUs, brackets, etc) globally but also, each single component can be colored right-clicking on it and selecting "Fonts & colors".

f) Phylograph shows bootstrap values and these values can be filtered to show only values greater or equal to a minimum value specified by the user.

g) Branches stroke color can be modified right-clicking on a node and selecting "Set color" to highlight nodes of interest. All branches of that node are coloured recursively by default.

h) Two default files called attachment_files" and "url_files" are available in the Phylograph subfolder "user_files" and can be used to save and edit specific attachments and URLs per OTU that Phylograph opens and fits in the tree in a single step. Attachments or URLs are appended to the tree clicking on "Attachment → Show/Hide attachment" or "Links → Add/Remove URLs to OTUs" respectively.

i) Users can create a cluster label and save it right-clicking on the node representative of that cluster. Selecting "Save cluster" a bracket and a label are subsequently drawn on screen. Labels can be resized holding down the "Shift" key and dragging the mouse. Each new cluster-label created can be saved automatically in a default "cluster_file" available in the Phylograph subfolder "user_files". When working with other trees based on the same clusters saved in this file, if the

user selects the option "Clusters → Show/hide clusters" in the menu bar, Phylograph labels all clusters of the tree that match with the saved cluster.

j) Right-clicking on a node and selecting "Rotate node" the tool rotate the whole branch.

INSTALLATION

Phylograph is distributed in two versions: a self-installable executable package for Microsoft Windows platforms and a zip package compatible with all platforms. Java applications do not require to be installed on the computer to run as its source code is interpreted on runtime by the Java Runtime Environment previously installed on the computer but we also provide the Windows installer which automatically creates shortcuts to the application. For executing the Windows installer version, simply double-click on the installer and follow instructions during installation. This process automatically generates desktop and start menu shortcuts. To execute the java version of the software, open a command-line interface; locate the application folder named 'phylograph'; and finally, type 'java phylo.Main'. To open a command line interface in Windows systems press the taskbar's 'Start' button; select 'Run…'; type 'cmd.exe' and accept.

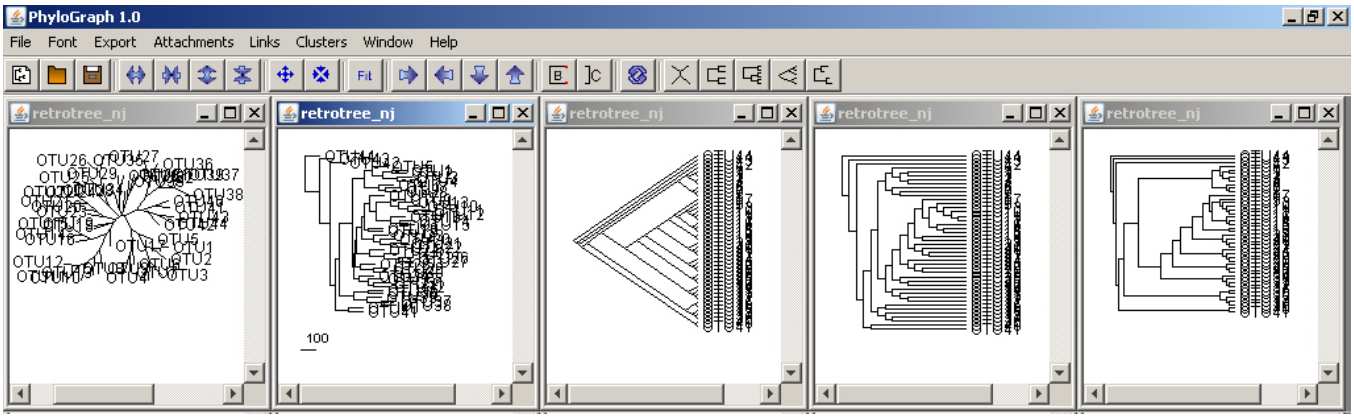
REQUIREMENTS

The software version is a Java application. This means that the tool runs on most PCs as a standalone program. Make sure before installing the tool that a Java Runtime Environment (JRE) is previously installed on your computer. A JRE can be downloaded and installed from Sun Microsystems' web site at URL 2. This application requires a version 6 update 2 of the JRE to run. To know if a JRE is currently installed on your system, click "Start", then select "Run", type "cmd" to open a command-line window and, finally, type "java -version" to know the current version installed on your computer. The process is show in Figure 6. If an error message is prompted, it means that a JRE is not properly installed on your computer.

CONCLUDING REMARKS

Phylograph was developed by us when dealing with the first version of the Gypsy database project (9). Applicability of this tool was inspired in other software such as TreeView (10), Drawgram/Drawtree (URL 4), and Baobab (11), etc. However, the required tree editor we were after was expected to have the capability to handle, edit, root, decorate and save graphical representation of large trees, "easy and fast". This first version of Phylograph is remarkable in these two functions, and indeed provides a wide background of other functions that make of Phylograph, a powerful tool in the handling of any kind of tree.

a)



b)

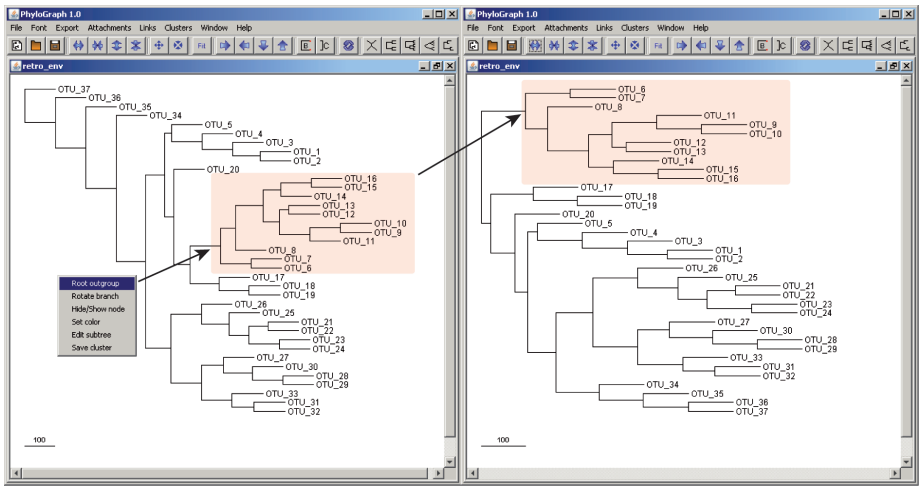
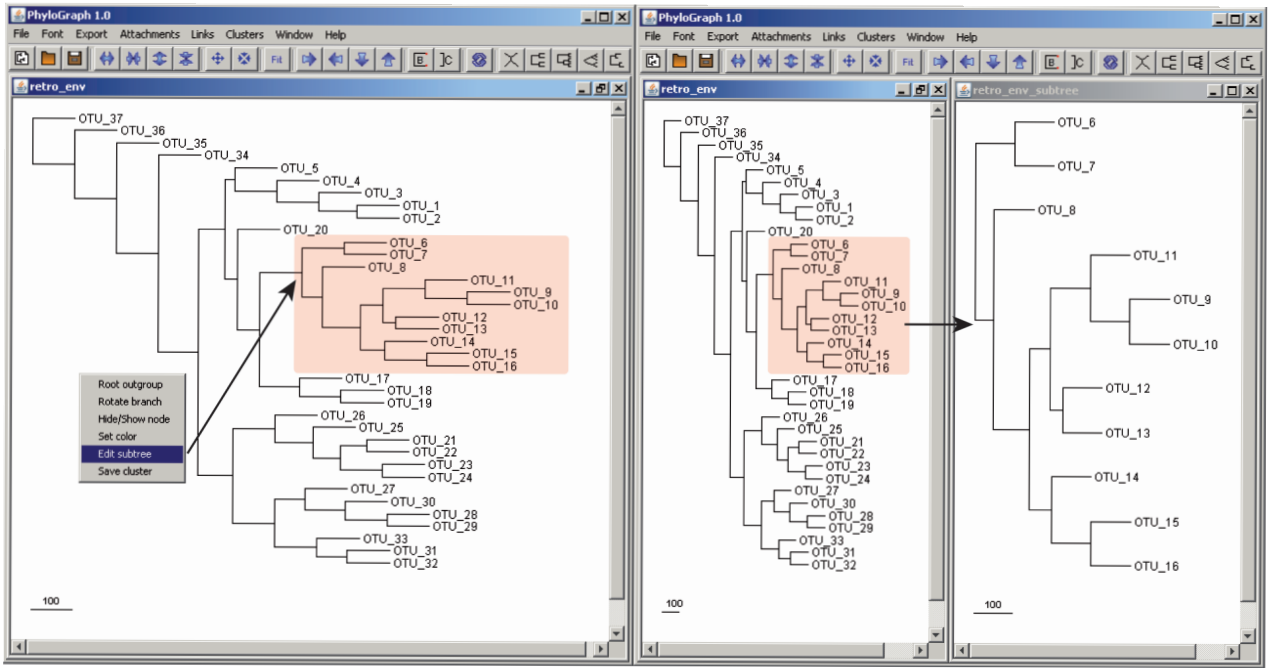
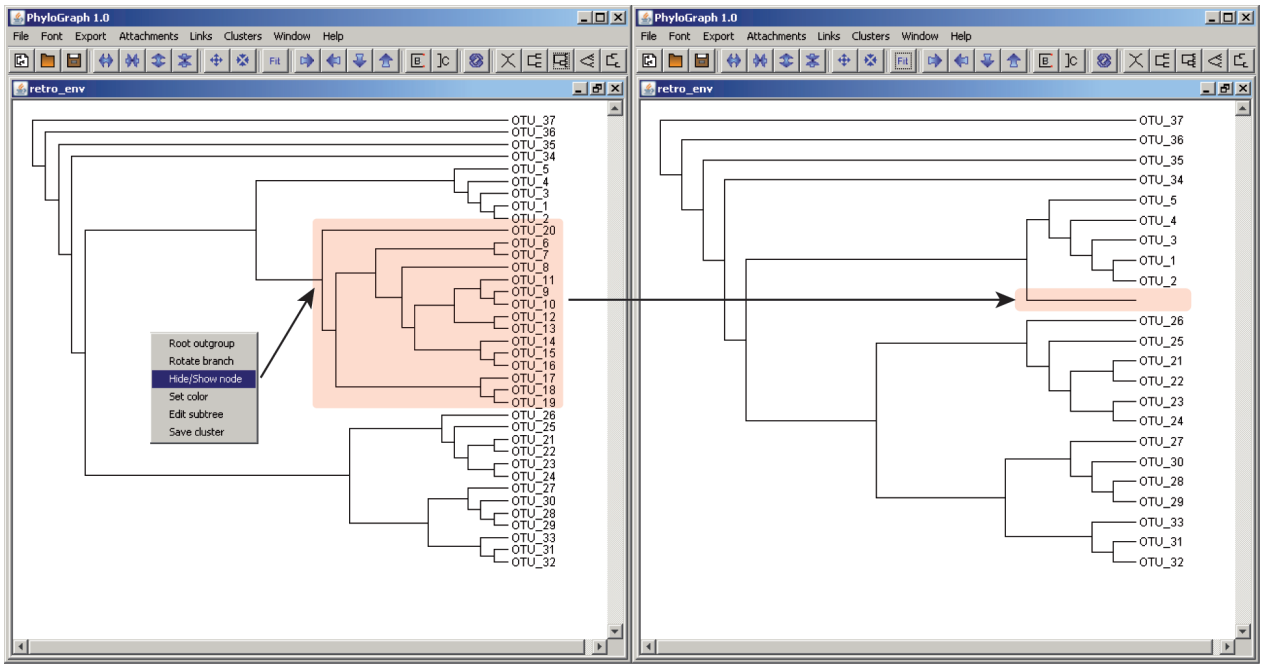


Figure 4. a) Available layouts. b) Rooting trees right-clicking with the mouse

c)



d)



e)

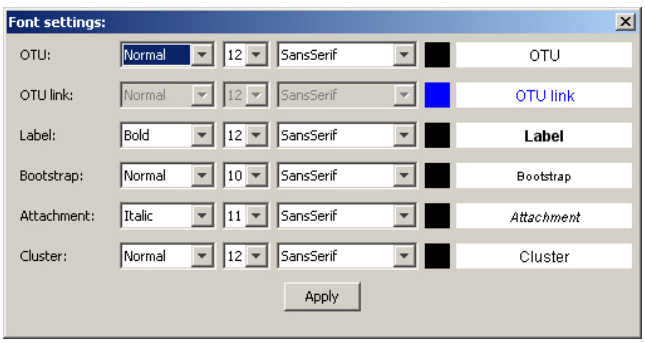


Figure 4 (continuation). c) Subtrees´ edition. d) Hide/Show one or more nodes. e) Font and color editor

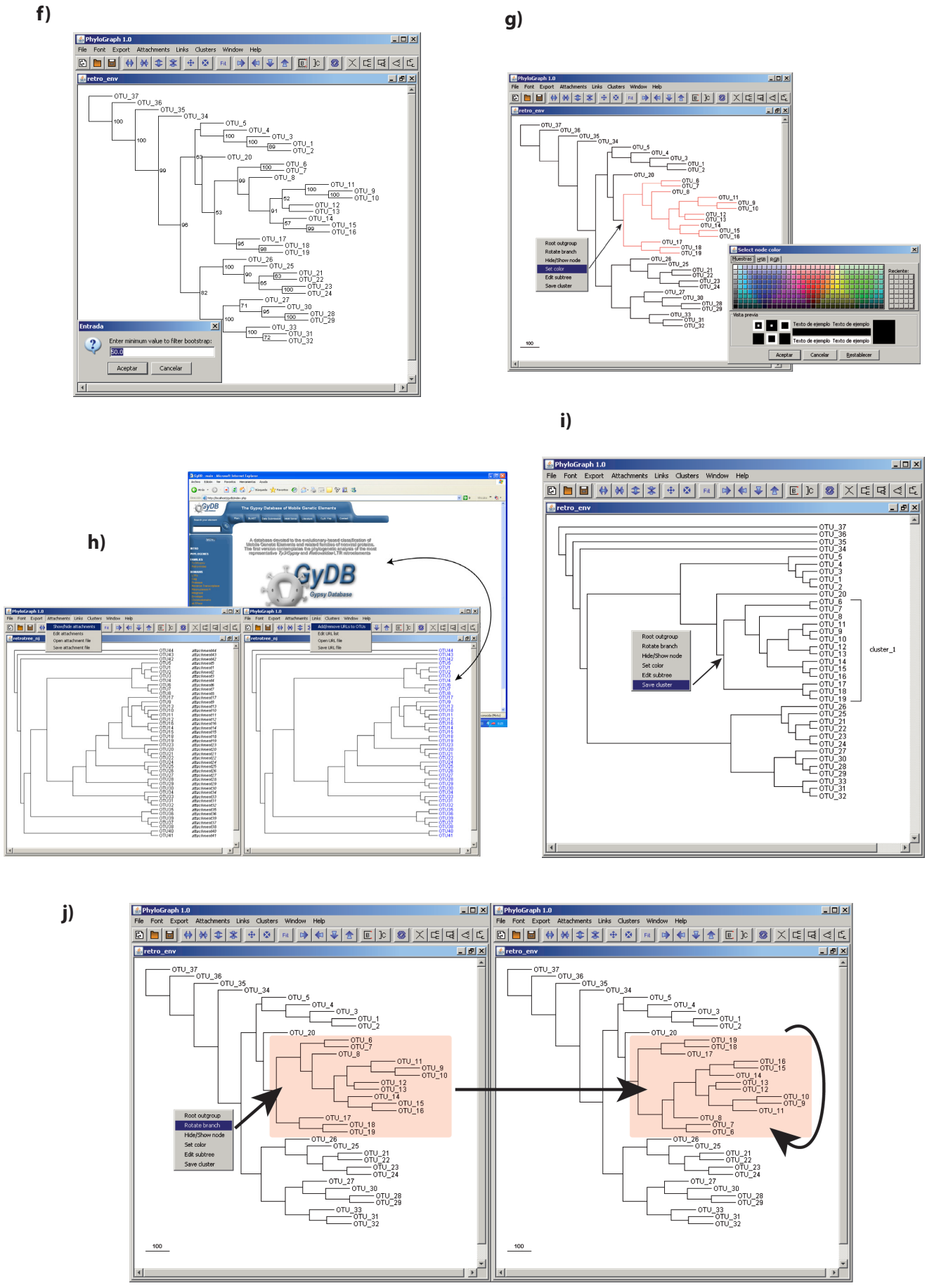


Figure 4 (continuation). f) Bootstrap values. g) Setting node stroke color. h) Attachments and URL files i) Creating and drawing clusters. j) Rotating a node

ACKNOWLEDGMENTS

We thank Rachel Epstein for language revision and the Servei Central de Suport a la Investigació Experimental (SCSIE) at UVEG for technical support. Biotechvana Bioinformatics has been awarded the NOVA 2006 by IMPIVA and Conselleria d`Empresa, Universitat I Ciència of Valencia. The research has been partly supported by grants IMCBTA/2005/45, IMIDTD/2006/158 and IMIDTD/2007/33 from IMPIVA, and by grant BFU2005-00503 from MEC to AM.

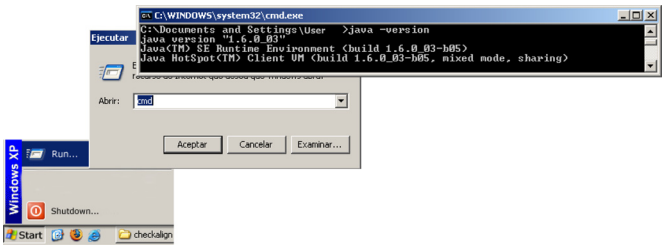


Figure 5. Checking the Java Runtime Environment installation.

LITERATURE

1. Maddison,D.R., Swofford,D.L. and Maddison,W.P. (1997) *Syst. Biol.*, **46**, 590-621.

2. Sugiyama,K. (2002) World.Sci.Pub.Co.Pte.Ltd, Pittsburg.

3. Margus,T. and McMorris,F.R. (1981) *Bull. Math. Biol.*, **43**, 239-244.

4. Carrizo,S.F. (2004) Asia-Pacific Bioinformatics Conference (APBC 2004).

5. Shoenfield,J.R. (2000) A K Peters (Ed).

6. Causey,R.L. (2001) Jones & Bartlett (Eds.).

7. Cori,R., Lascar,D. and Pelletier,D. (2001) Oxford University Press.

8. Bachmaier,C., Brandes,U. and Schlieper,B. (2005) In Deng and D.Du : (ed.), pp. 1110-1121.

9. Llorens,C., Futami,R., Bezemer,D. and Moya,A. (2007) *(2008) Nucleic Acids Research* (NAR) 36 (Database-Issue):38-46

10. Page,R.D. (1996) *Comput. Appl. Biosci.*, **12**, 357-358.

11. Dultier,J. and Galtier,N. (2002) *Bioinformatics*, **18**, 292-293.

URLS

1. Private source license: http://biotechvana.uv.es/bioinformatics/main.php?document=terms_pcl
2. Newick: http://evolution.gs.washington.edu/phylip/newicktree.html
3. Sun Microsystems: http://www.java.com
4. Phylip: http://evolution.gs.washington.edu/phylip.html

SPONSORS