TreeSnatcher Plus

Tutorial 2 Idea, challenge, and strategy

All images are copyrighted by their respective owners. If you are the owner of one of the images and wish it to be removed from these documents, please inform us.

Aim

TreeSnatcher Plus converts images of phylogenetic trees back into a machine-readable representation that can be used within another Bioinformatics application. The program uses the Newick representation which is widely used in Bioinformatics and Biology.

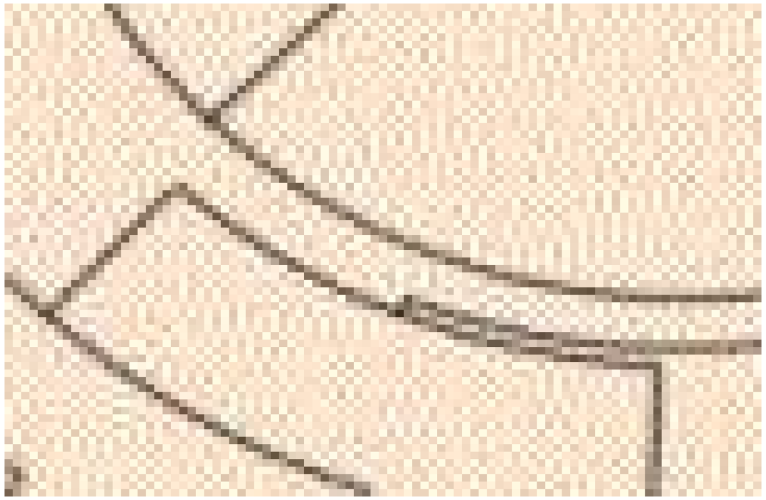
This functionality can be of help whenever the data source for a publication is out of reach, or if you want to play through What if-scenarios with the tree depicted. You can modify other people's trees as you need or draw your own tree.

Challenge

An image of a phylogenetic tree is made of arbitrarily shaped, not necessarily thin, straight or bent, intersecting lines. The species are in most cases represented by their names. The image may contain additional elements, e.g. icons. Different objects are likely to overlap the tree itself.

The whole image consists of a mosaic of colored pixels that have no meaning when looked at in isolation. Without some heavy lifting, a computer can neither recognize lines and intersections nor read text.

Using TreeSnatcher Plus, you first need to preprocess the image before the program can identify the branching positions and the branches themselves. As soon as you are satisfied with the result, TreeSnatcher Plus calculates the Newick expression for your tree. It relies on you to supervise the whole workflow.

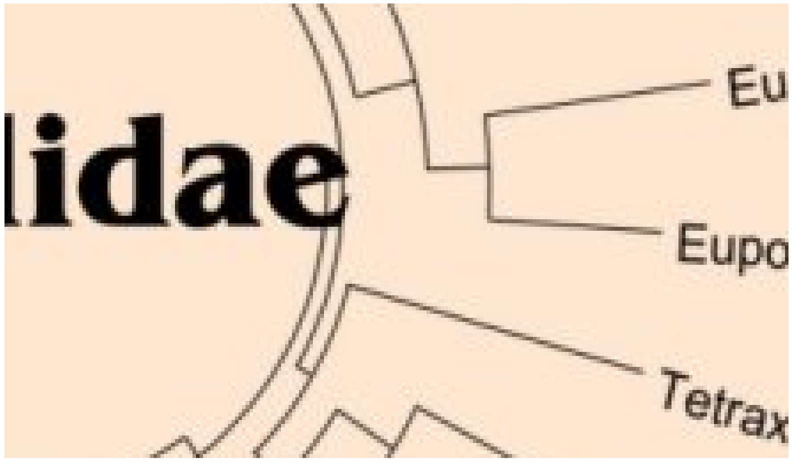


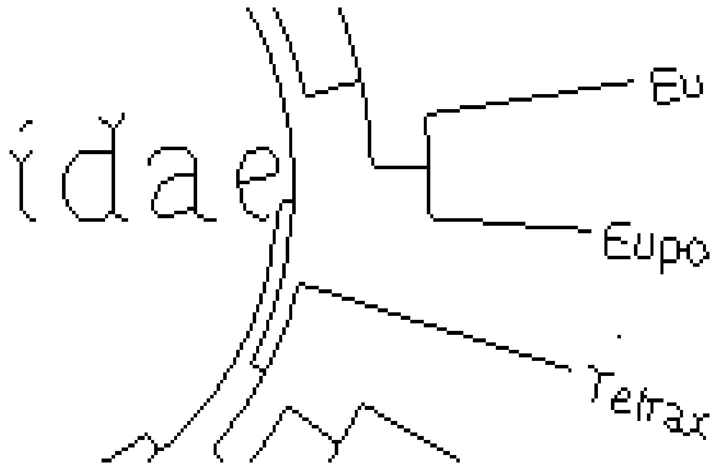
This is a magnified portion of an image showing a phylogenetic tree. The computer cannot grasp the whole image like a human but must construct its view stepwise from the pixel colors and their spatial relations.

On this basis, how would you define what a line is, whether it is straight or bent, where lines begin, intersect,...?

Strategy

Below you are shown an original portion of the same image before and after effective preprocessing. Tree and text are separated, lines and intersections are as thin as possible and clearly distinguishable from the background. The foreground is black and the background is white.

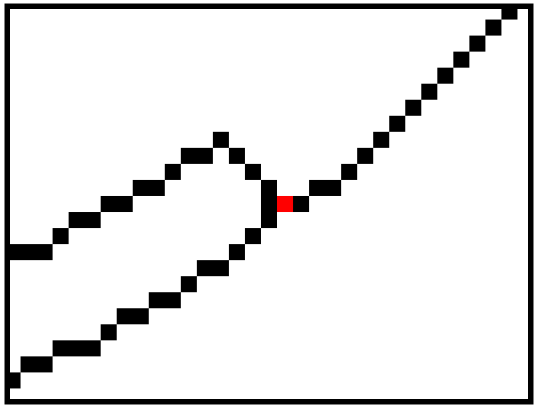


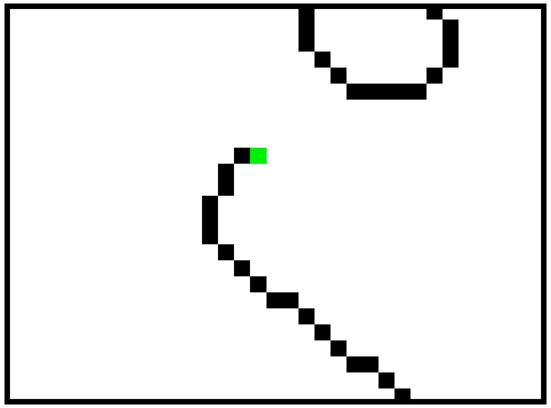


For preprocessing, TreeSnatcher Plus offers all tools you need. With experience, it is fairly straightforward. Your superior goal is always to achieve a result like in the right image on the last slide. Please have a look at the tutorial about preprocessing.

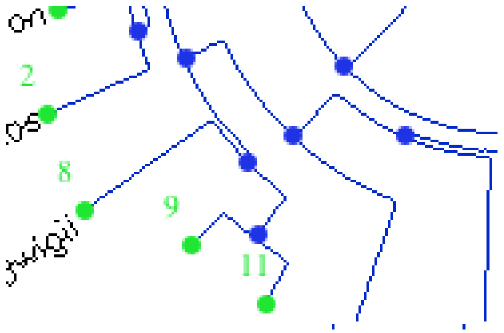
In the preprocessed image, the program can look for branching positions. For those, it counts how many among the eight neighboring pixels of each black pixel are black.

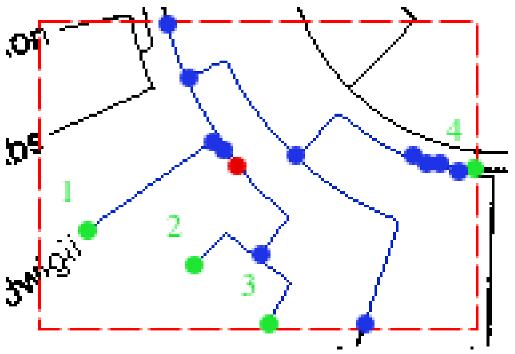
An inner node – a branching position – is averaged over the positions of all black pixels that have at least two neighboring black pixels on their own.
A tip – the position of a species, gene or taxon - is formed for a pixel that has exactly one black pixel as neighbor. Black pixels with two black neighboring pixels lie on a path.





After adept preprocessing, you get a result like in the left image:
Blue nodes represent branching positions, green nodes tip positions.
No nodes need to be added or removed.

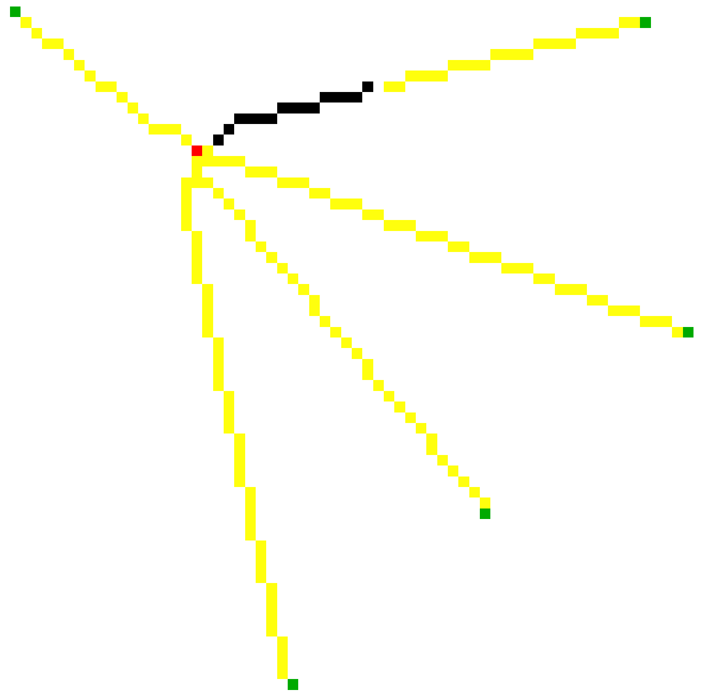




The right image shows the result after inapt preprocessing: As some parallel lines collide, the program 'finds' false and too many nodes there.
If you do not remove them and/or modify the image, the tree topology will be wrong.

You may **Add**, **Remove** and **Move Node**s or use the drawing tools and repeat the automatic node placement.

When the node placement is finished, TreeSnatcher Plus can determine the positions of branches in the tree. A branch exists between a pair of nodes (i.e. inner nodes and tips) if they are linked by a gapless path of black pixels.



Beginning at a tip, the program visits all black pixels in the neighborship and colors them yellow, then visits their neighbors etc. If another node is within reach, it tries to lengthen the path to this node.

Using this strategy, the program is able to determine paths between tips and inner nodes. As the discovered path is not necessarily the shortest possible, it then walks the path backwards in a parsimonious fashion in order to determine the lowest distance between the nodes.
It is more complicated to find paths between inner nodes as there are always multiple directions to investigate.

In the illustration, due to a missing black pixel, the program was unable to 'walk down' the path from the tip in the top right corner to the inner node. Therefore, it will not form a branch for this pair of nodes.

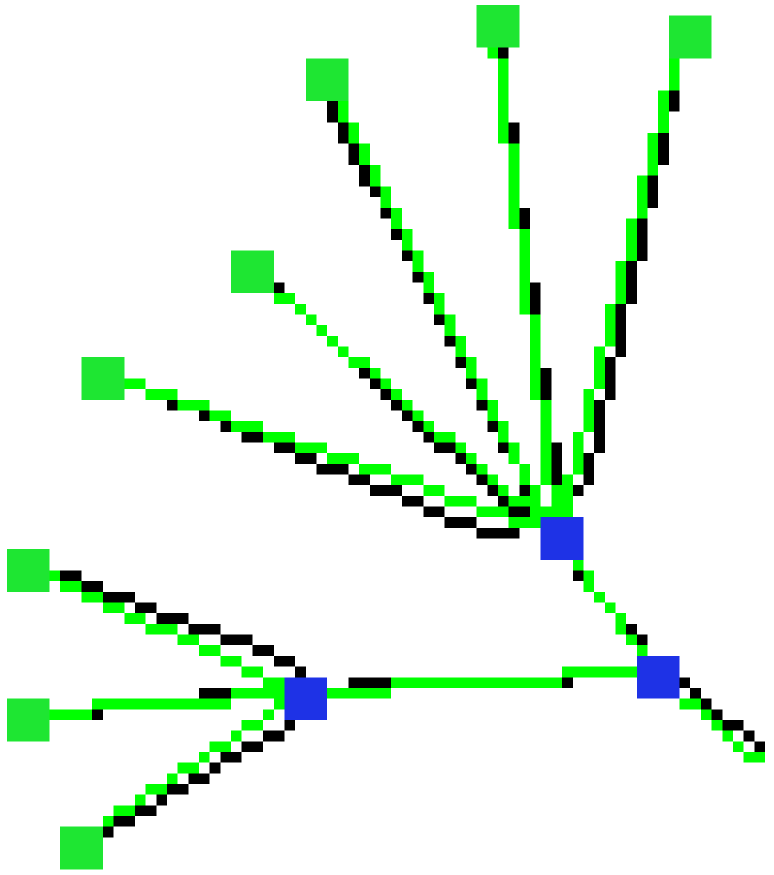
A possible result might look like this:

A green line between two nodes means that a foreground path (formed by black pixels) between them exists. This is a branch. For the time being, the length of the branch is the path length, measured in pixels.

Branches and nodes are logical objects that overlap the image but are not part of it. If you click at them, you can modify their properties.

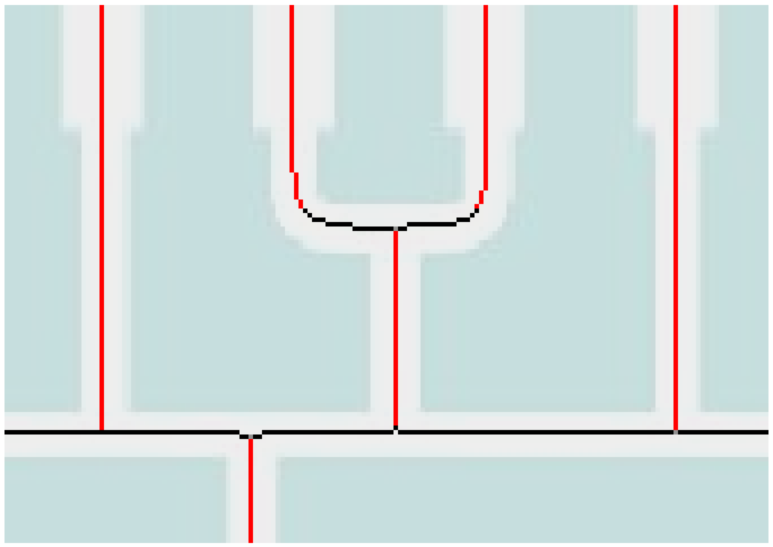
Amongst others, you can assign a name to an outer node, move or remove a node or assign a custom length to a branch.

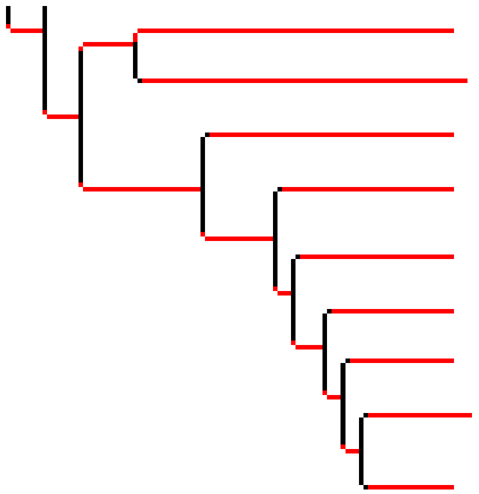
You never need to drag branches on your own. If you do it anyway, the program cannot calculate their length because there is no path.



If there are branches absent in the tree, you can modify the image with the drawing tools to delete black pixels and fill gaps. After that, you repeat the **automatic** calculation of branches and branch lengths.

TreeSnatcher Plus can try to recognize rectangular trees. For this, you select the option **Rectangular.** Make sure that the image is well prepared.
Have a look at how the program has identified length relevant (red) and irrelevant (black) path segments:



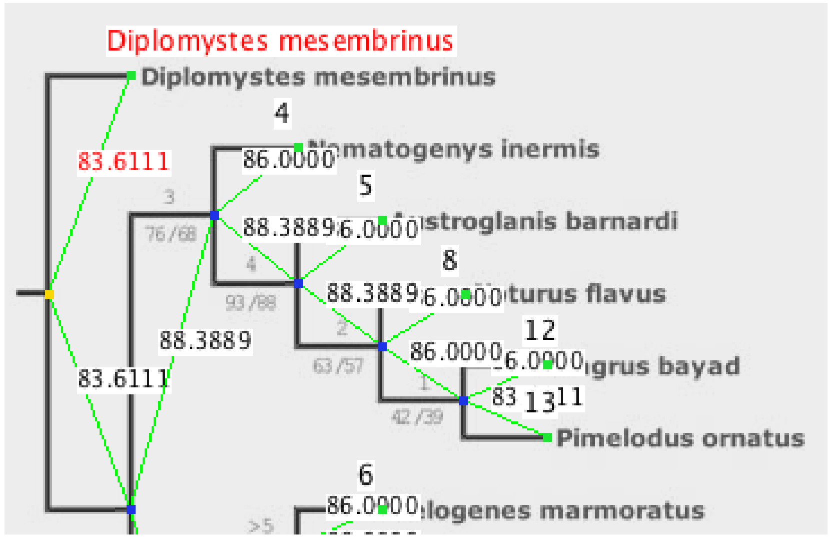


The approach works more accurate for sufficiently long and straight paths and sharp bends. You should experiment with different images.

You might want to scale the tree. For this, you can use a line of known length in the original image, e.g. a scale bar, or you assign a custom length to a branch and let TreeSnatcher Plus scale the tree based on this metric.
You can use the branch lengths that the program has calculated and custom branch lengths simultaneously.

If you wish, you can choose a node from which the Newick expression shall be constructed recursively. This node is highlighted in yellow.

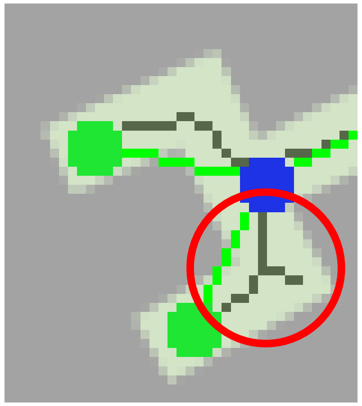
Currently, TreeSnatcher Plus cannot read taxon names from the source image. Either you accept the default numbers or you type in the species names.

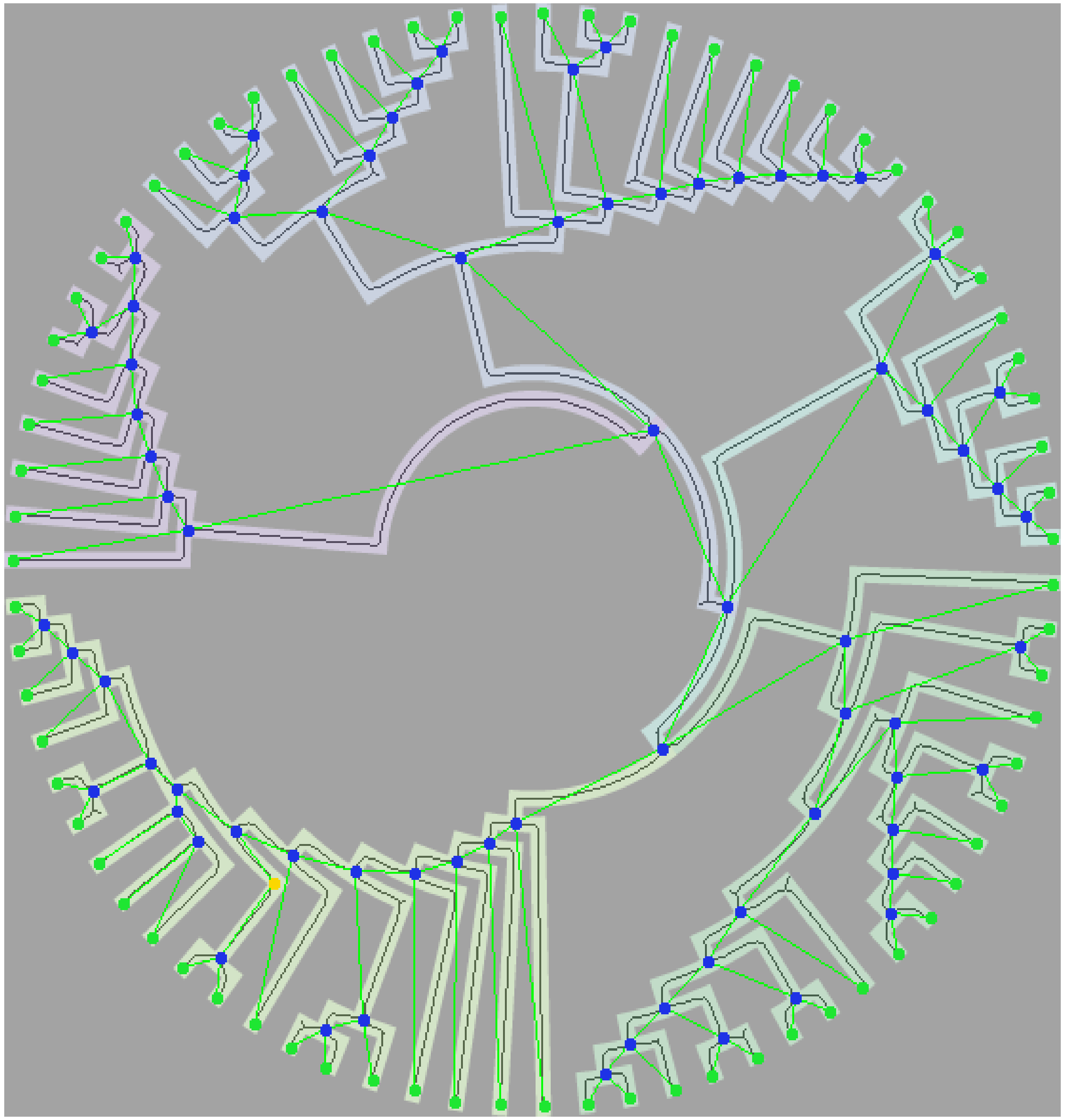


Another tree - the processed image is superimposed on the source image.

Notice how thin the lines are in the processed image compared to those in the original image. **Thinning** is one of the preprocessing tools.

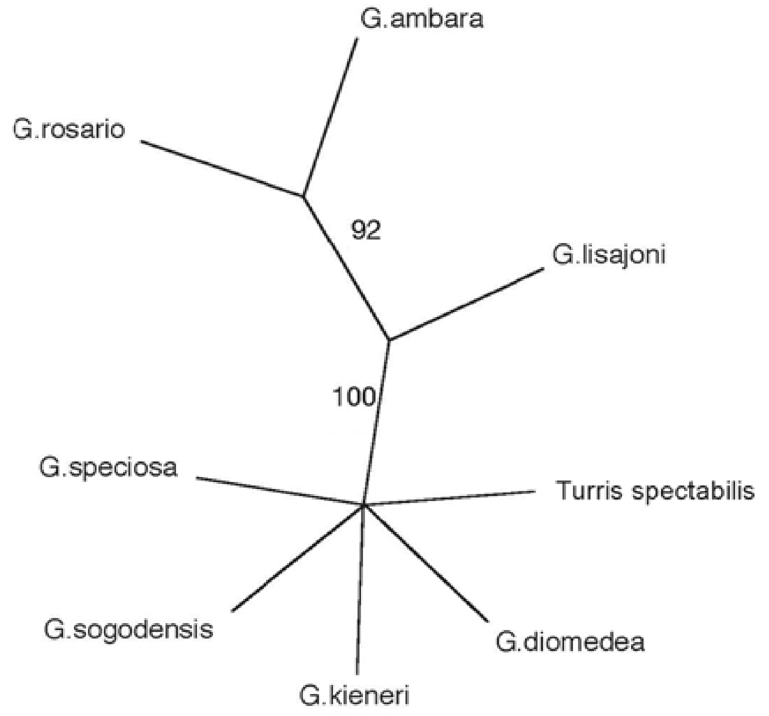
In spite of preprocessing, some mistakable pixels are still there. However, as long as there are no misplaced nodes, the branches will be set correctly.





In the end, TreeSnatcher Plus converts the tree topology into a Newick tree representation. Branch lengths are optional.

From here...



...to there

(G. rosario:95.8763, G. ambara:94.8454, (G. Lisajoni:90.7216, (G. speciosa:98.9691, G. sogodensis:77.3196, G. Kieneri:101.0309, 6:74.2268, Turris spectabilis:101.0309):100.0000):92.0000);

You might want to save the Newick representation to the clipboard or store it in a text file.