

PhyloBrowser: A visual tool to explore phylogenetic trees

Tamara Tershakovec

December 18, 2012

Submitted in partial fulfillment of the  
requirements for the degree of  
Master of Science  
Department of Computer Science  
New York University

---

Prof. Dennis Shasha

---

Prof. Gloria M. Coruzzi

©2012  
Tamara Tershakovec  
All Rights Reserved

## Acknowledgements

Primary acknowledgements go to my research advisor, Dennis Shasha, for his patient and unwavering support.

I would also like to thank my second reader, Gloria Coruzzi, for encouraging and showcasing my work.

Kranthi Varala helped immensely in explaining the biological and statistical concepts involved in this project.

The Virtual Plant team got me started in biological data visualization and was a joy to work with.

And finally, thanks to Chris Poultney, who put me in touch with Professor Shasha and started me on my way back to NYU.

## Table of Contents

<b>INTRODUCTION .....</b>	<b>6</b>
SUMMARY .....	6
BIGPLANT PROJECT.....	6
BIOLOGY BACKGROUND.....	6
<i>Cladogram definition</i> .....	6
<i>GO terms</i> .....	7
<i>p-value</i> .....	7
<i>Partitioned branch support</i> .....	8
<b>RELATIONSHIPS BETWEEN NODES, GENES AND GO TERMS.....</b>	<b>9</b>
<i>Concepts</i> .....	9
<i>Data files</i> .....	9
<i>Data structures</i> .....	9
GO TERMS TO GENES TO NODES .....	9
USE CASES .....	10
<i>Finding genes responsible for traits</i> .....	10
<i>Guessing the roles of genes</i> .....	10
<b>OVERVIEW .....</b>	<b>11</b>
GRAPHICAL USER INTERFACE.....	11
<i>GO term filter can itself be filtered</i> .....	13
<i>Heatmap</i> .....	14
<i>Tooltips</i> .....	14
<i>Uploading</i> .....	15
DATA FILES OVERVIEW .....	16
<i>Newick file</i> .....	16
<i>Partitioned Branch Support file</i> .....	16
<i>Gene to GO file</i> .....	16
<i>GO annotation File</i> .....	17
DATA STRUCTURES OVERVIEW .....	18
<i>root</i> .....	18
<i>Partitioned Branch Support</i> .....	18
<i>goTermIdToNodesMap</i> .....	18
<i>goTermNameToGoldMap and goTermIdToGoNameMap</i> .....	18
<i>geneMap</i> .....	18
<b>JAVASCRIPT .....</b>	<b>19</b>
JAVASCRIPT IS FAST .....	19
LOADER.JS .....	19
BIGPLANTTREE.JS .....	19
<i>Explanation of D3</i> .....	19
<i>Using D3</i> .....	20
<i>using the phylogram library</i> .....	20
TREECOMPONENTS.JS.....	21
STATICDATA.JS.....	21
DYNAMICDATA.JS .....	21

COLLAPSING.JS.....	21
<b>ADDITIONAL ISSUES.....</b>	<b>23</b>
STYLING .....	23
MINI-APPS .....	23
<i>ProcessGeneToGoFile.html</i> .....	23
<i>ProcessGodescriptionFile.html</i> .....	23
LOADING THE CLADOGRAM - EXPLANATION OF URL PARAMETERS .....	24
TECHNOLOGIES.....	24
<i>jQuery</i> .....	24
<i>D3</i> .....	25
<i>Phylogram layout</i> .....	25
<i>JavaScript Newick format parser</i> .....	25
<i>Mustache templates for html</i> .....	25
TOOLS.....	25
<i>WebStorm</i> .....	25
<i>Firebug and Chrome Developer Tools</i> .....	25
<i>Minimizers</i> .....	25
FUTURE WORK.....	26
<i>Family names &amp; rectangles</i> .....	26

## Introduction

### *Summary*

The PhyloBrowser is an interactive web-based data visualization developed to explore genome-scale phylogenetic trees and their associated metadata in a web browser. It depicts the relationships between genes, nodes and Gene Ontology terms (GO terms) through a graph that can be filtered by several data categories to narrow and highlight data of interest to biologists. The tree is visually depicted in the form of a Cladogram.

The PhyloBrowser visualization allows users to

- manually filter cladogram nodes by Partitioned Branch Support (PBS) value (Sorenson & Franzosa 2007) , associated genes or associated GO (Gene Ontology) terms
- upload lists of genes and GO terms as an alternative to manual filtering
- view a heatmap of related nodes and GO terms
- view detailed information on each node

The PhyloBrowser cladogram is easy to access, as no software needs to be installed. Only a modern web browser and internet connection are required.

In addition, two web-based mini apps are provided to pre-process data files for faster page loading. These apps will be useful when setting up PhyloBrowser for use with other data sets from species spanning any kingdom of life.

#### **WEB ACCESS to PhyloBrowser:**

[http://branchesandleaves.com/phylobrowser/Cladogram.html?newickFile=genus31MP\\_labeled.tre&pbsFile=bpPBS.json&css=css/bigPlantWhite.css&geneToGOFile=geneToGoT10filtered.txt&GOannotationFile=go-t10-annotFiltered.txt&colorsFile=colors.json&speciesFile=BigPlantv1Species.json](http://branchesandleaves.com/phylobrowser/Cladogram.html?newickFile=genus31MP_labeled.tre&pbsFile=bpPBS.json&css=css/bigPlantWhite.css&geneToGOFile=geneToGoT10filtered.txt&GOannotationFile=go-t10-annotFiltered.txt&colorsFile=colors.json&speciesFile=BigPlantv1Species.json)

### *BigPlant project*

BigPlant is an NSF Plant Genome project of the New York Plant Genomics Consortium. The goal of this project is to understand the genetic changes underlying the origin of seed plants. One of the results of this research is a phylogenetic tree composed of genome sequences from 150 seed plant species (Lee et al 2011); this is the data that the PhyloBrowser Cladogram visualization depicts.

### *Biology background*

#### **Cladogram definition**

*A branching treelike diagram used to illustrate evolutionary (phylogenetic) relationships among organisms. Each node, or point of divergence, has two branching lines of descendance, indicating evolutionary divergence from a common ancestor. The endpoints of the tree represent individual species, and any node together with its descendant branches and subbranches constitutes a clade.*

*- The American Heritage® Science Dictionary*

In the data used by the PhyloBrowser cladogram, the endpoints are 150 seed plants researched by the Big Plant project (Lee et al 2011). The nodes leading to the endpoints represent an assumed ancestral node i.e., a common ancestor from which the current species evolved. (Please note that the branch lengths in this cladogram graph do not represent any data. )

### GO terms

Gene Ontology (GO) terms are descriptions of functionality, structure and other characteristics of living organisms ([www.geneontology.org](http://www.geneontology.org)). The GO project provides a controlled vocabulary of **terms** for describing gene product characteristics and gene product annotation data from **GO** Consortium. The GO terms are organized in a hierarchical tree from most specific to least specific. For example, the specific term “serotonin binding” is a child of the more general term “binding”. Genes are found to be associated with GO terms through experimental studies.

Based on the research results of the Big Plant project, each node may have a list of genes and a list of GO terms associated with it (Lee et al 2011). The data produced by Big Plant -- gene/node/GO term relationships and numerical values for those relationships -- are the basis for the interactivity of the PhyloBrowser cladogram visualization.

### p-value

In statistics, the p-value is the probability of the observed result occurring by chance. When the p-value is less than a certain significance level, i.e., it is very unlikely, then the "null hypothesis" (typically a default position, such as "there is no effect") is rejected. Thus, lower the p-value, higher the significance of the result.

The classic example is a coin toss. In one example, the null hypothesis is that "the coin is unbiased" and the result of 20 tosses is 14 heads and 6 tails. The p-value of this result has been calculated to be 0.058. This is the likelihood that a fair coin would give this result. By convention, the null hypothesis can be rejected when the p-value is less than 0.05. Because the p-value of 0.058 is greater than 0.05, we say that "the null hypothesis cannot be rejected based on this p-value", i.e., the coin is likely to be unbiased.

### **Partitioned branch support**

Partitioned branch support (PBS) is a measure of the level of support provided by a gene to the existence of a node (“point of divergence”) (Sorenson & Franzosa 2007). If you were to remove all genes that provide support for a node then that node would collapse -- the branching point dividing its child subgroups would disappear and all the subgroups would become direct children of the immediate parent node.



## Relationships between Nodes, Genes and GO terms

There are three parts to understanding the data in the PhyloBrowser Cladogram visualization.

### Concepts

The first, is understanding the relationships between the biological terms. There are several levels to the relationships, as nodes are related to genes and GO terms, but genes are also related to GO terms.

### Data files

The second, is understanding the data files. The biological relationships between genes, GO terms, and nodes are extracted from the data, but they may be broken up across several files that need to be assembled into a coherent structure. Understanding how the data corresponds to the concepts above is useful to a developer maintaining or extending the application.

### Data structures

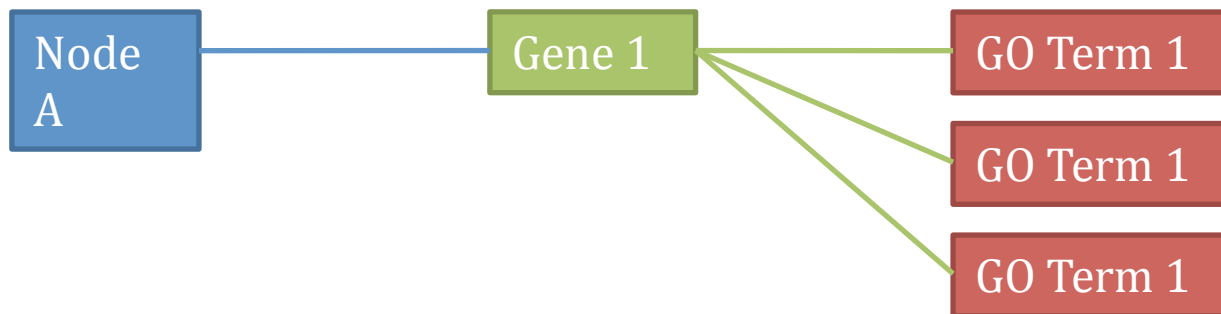
The third, is understanding the data structures. The information in the data files is processed into a slightly different form for better code organization and efficiency.

The data files and structures will be explained in more detail in subsequent sections.

## *GO terms to genes to nodes*

Nodes on the cladogram can be related both to many genes and to many GO terms. Genes in turn can be related to many GO terms themselves.

The PBS calculations give you a list of genes that provide positive PBS for a node and hence each node has a list of genes associated with it. Each node has two lists – the list of genes that provide partitioned branch support (PBS) for that node and the GO term list. The GO term list is derived from the gene list for each node using the gene –GO term mappings. Therefore the GO term list associated with the cladogram is the union of all GO terms associated with any of the genes that provide PBS to any of the nodes in the cladogram. Within this large set of GO terms are a smaller set that show a significant relationship to a node.



The larger set of GO terms for a node is created by finding all the GO terms that relate to the genes for that node. The smaller set of GO terms is the set that has a significant enough relationship to the node to be in the GO term list of the PBS file.

These GO term lists were manually assembled through the Virtual Plant software platform ([www.virtualplant.org](http://www.virtualplant.org)) (Katari et al 2010). Each of the gene lists is tested for over-representation of GO terms by using the VirtualPlant BioMaps function (Fisher's Exact Test). This test groups the input list of genes by GO terms, then checks if any of the terms are represented by a higher number of genes than they should be, given the number of genes associated with that GO term in the whole Arabidopsis genome. It also generates the p-value for the significance of this over-representation, measuring the strength of the relationship between the GO term and the node.

## Use Cases

### Finding genes responsible for traits

One use of the PhyloBrowser Cladogram visualization with Big Plant data (Lee et al 2011), is in identifying genes responsible for plant traits. For example, a biologist may be interested in a plant trait such as drought resistance. Some distantly related species in the Big Plant tree may have that trait. If the biologist compares the genes that support the divergence of the resistant species from the closest relatives that are *not* drought resistant at multiple points in the tree, the biologist may be able to find genes that are responsible for that trait.

### Guessing the roles of genes

A second use of the PhyloBrowser visualization is to prioritize research into unknown genes. A biologist may have a list of genes from an experiment. Some of these genes may not have been studied before so there is little to no information on them. If a gene from this list is found to have Partitioned Branch Support (PBS) for nodes spread out across the tree, this gives the researcher a clue as to the role of the gene. This can be done either by finding traits specific to the species at those branches or by looking at common Gene Ontology terms that are over-represented at those nodes. A researcher

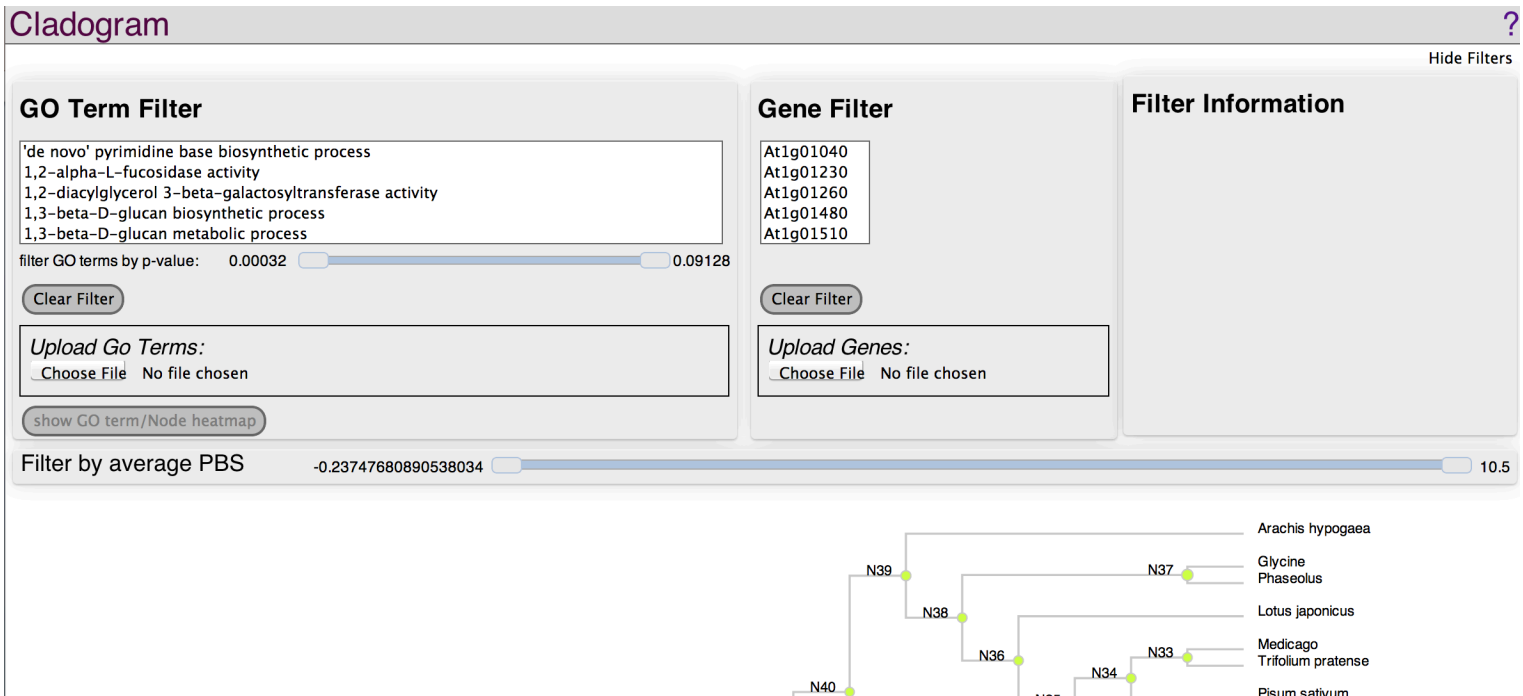
could find these nodes by clicking on the gene in the gene filter or uploading a set of genes.

## Overview

### Graphical User Interface

The PhyloBrowser Cladogram visualization shows a cladogram tree graph and components for filtering this graph. Image 1 shows the basic layout – the filtering components are shown above the graph. These include the GO term filter, the gene filter, the PBS slider and the GO term p-value slider. In addition, there is a feedback panel that lists the selected genes and GO terms.

In the first image, no filters are selected. You can see the GO term filter, the Gene filter, the info panel labeled “Filter Information” and part of the tree at the bottom. The p-value slider is below the GO Term Filter and the PBS value slider is at the bottom of the components section (just above the graph).



**Image 1**

In the second image, the GO term “amine transmembrane transporter activity” is selected. This is reflected in the info panel and in the visualization collapsing to show only those nodes relating to this GO term. In this example, only one term is selected, but the filter supports multiple selections.

## Cladogram

Hide Filters

### GO Term Filter

amide biosynthetic process  
amide transmembrane transporter activity  
amine binding  
amine biosynthetic process  
amine catabolic process

filter GO terms by p-value: 0.00032 0.09128

Clear Filter

Upload Go Terms:

Choose File No file chosen

show GO term/Node heatmap

### Gene Filter

At1g01040  
At1g01230  
At1g01260  
At1g01480  
At1g01510

Clear Filter

Upload Genes:

Choose File No file chosen

### Filter Information

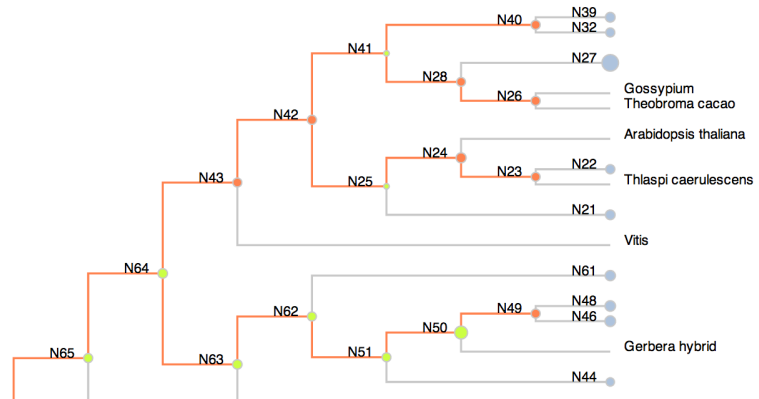
**Selected GO terms:**

amide transmembrane transporter activity

Filter by average PBS

-0.23747680890538034

10.5



**Image 2**

In image 3 you can see three filters at work: The previously shown GO term filter, the gene filter and the PBS slider are all in use.

### **GO term filter**

One item from the GO term filter, “amine transmembrane transporter activity”, is selected, and can be seen in the info panel.

### **Gene filter**

One gene is selected from the gene filter, and can also be seen in the info panel.

### **PBS filter**

The left PBS slider thumb is moved slightly to the right to limit the nodes to those possessing average PBS values within the range shown by the slider.

### Cladogram graph

The nodes are sized by their average PBS values. The GO term filter, gene filter and PBS slider cumulatively filter the nodes, meaning that the only nodes shown are those that relate to the GO terms and genes selected, *and* are within the PBS slider's values. These nodes are shown in orange. The branches of the tree are collapsed if none of their nodes are within the filter parameters. These collapsed nodes are shown in light blue. Nodes that are not within filter parameters but cannot be collapsed because they are part of the path to a node that is shown remain in the default color of green.

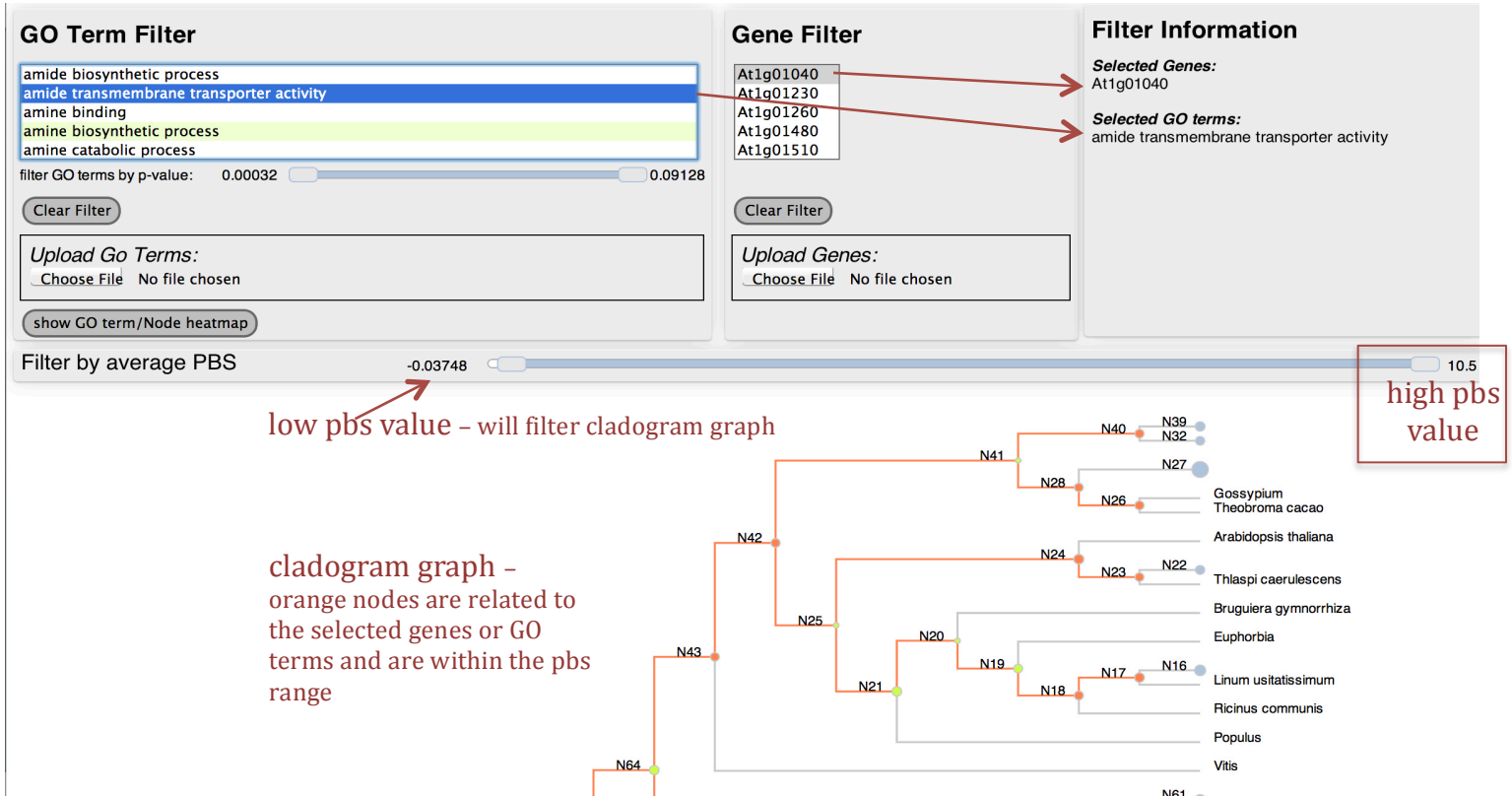


Image 3

### GO term filter can itself be filtered

The p-value slider can filter the GO terms by their p-value, to show GO terms within a specific range. The items in the GO term filter which have p-values are highlighted in green. When the p-value slider is engaged, only these GO terms will be shown. Sliding the p-value slider refines the filter further. The slider has two "thumbs", which means it can be limited from the low or high value.

**GO Term Filter** GO term with p-value is shown in green

- amide biosynthetic process
- amide transmembrane transporter activity
- amine binding
- amine biosynthetic process
- amine catabolic process

filter GO terms by p-value: 0.00032 0.09128

Clear Filter

p-value slider filters GO terms

### Heatmap

If any GO terms or genes are selected, the “show GO term/Node heatmap” button will be enabled. When this button is pressed, a heatmap pops up, showing the relationships between nodes and GO terms. If a relationship has a p-value associated with it, that square will be colored red, more intensely for a lower (more significant) p-value. Hovering over a node/GO term intersection will show a tooltip with the GO term, the node name, and the p-value.

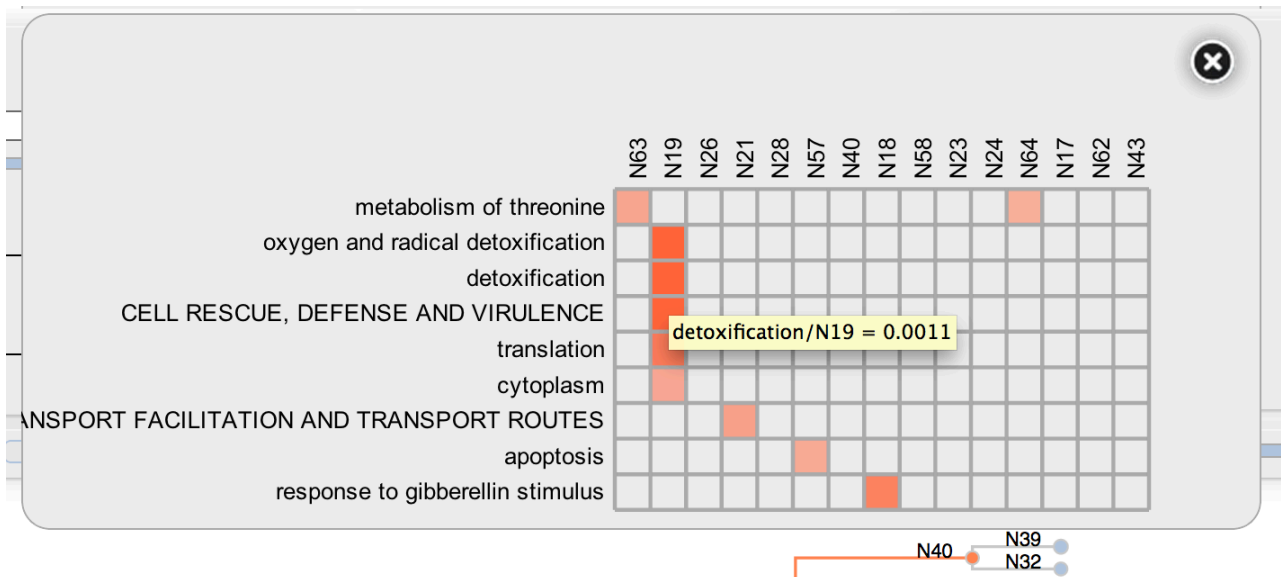


Image 4

### Tooltips

Details about a node will pop up in a tooltip when a user hovers over a node. You can see the partial details of node N29 in Image 5 below.

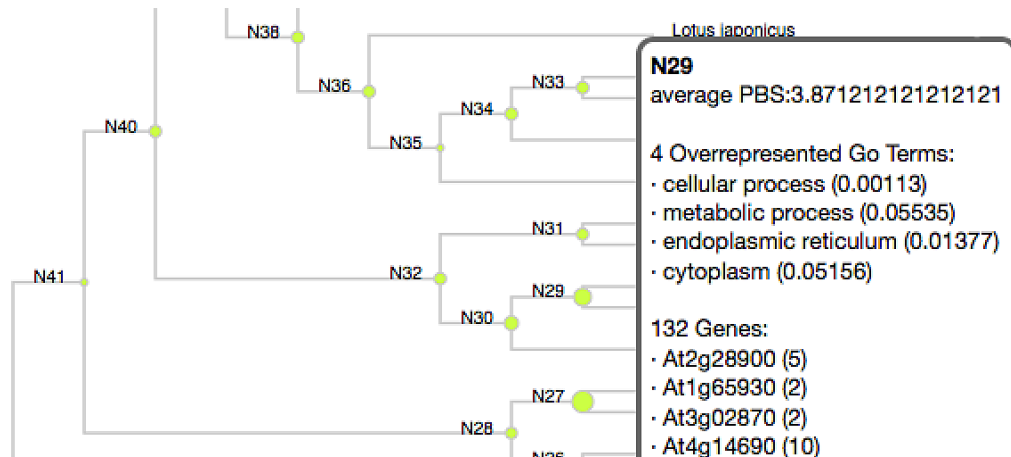


Image 5

### Uploading

Users can upload a plain text file of gene ids or GO term ids separated by line breaks. If a user uploads a list of genes in the GO term upload, no terms will match and nothing will be selected. The user is alerted to any mismatches. For example, in the following image shows the results of uploading a text file with the following information:

```
GO:0046509
I'm not a go term!!
GO:0006207
```

Two matches are shown and an alert highlights the mismatch.

Uploading is available in the following browser versions:

- Google Chrome 6 and later
- Firefox 3.6 and later
- Safari 6 (Safari 6 will support the standard File API)
- Internet Explorer 10 (Preview 2+)

## Data Files Overview

There are four data files necessary to render the visualization. These are

### Newick file

The Newick file contains the data that describes the tree structure – nodes and endpoints. It is based on the results of gene research by the Big Plant project, and hopefully will continue to grow as more sequencing and analysis is completed. You can read more details about the Newick file format here: [http://en.wikipedia.org/wiki/Newick\\_format](http://en.wikipedia.org/wiki/Newick_format).

### Partitioned Branch Support file

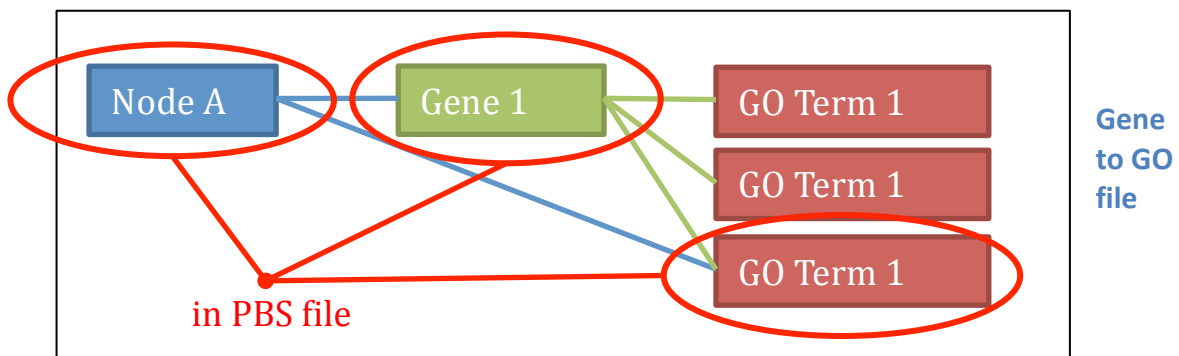
Also a result of the research at Big Plant, the PBS file contains supplementary data for each node of the tree described by the Newick file. The endpoints, corresponding to actual plants, do not have gene or GO term data and are not included in the PBS file. Each node may have two lists of supplemental data:

**List 1:** overrepresented GO terms: each GO term item includes the GO term id, description, p-value and type. The type is usually GO but can sometimes be MIPS (Munich Information Center for Protein Sequences). If it is MIPS, the data is ignored.

**List 2:** genes associated with that node by partitioned branch support. This includes the gene id and the partitioned branch support value.

A sample of the format, for node with id "N63", is:

```
"N63" : {
  "ovrTerm" : [
    {
      "pValue" : "0.07041",
      "desc" : "metabolism of threonine",
      "id" : "01.01.06.04",
      "type" : "MIPS"
    }
  ],
  "genePartition" : [
    {
      "pbs" : "-1",
      "atOrtholog" : "At3g14990"
    }
  ]
}
```

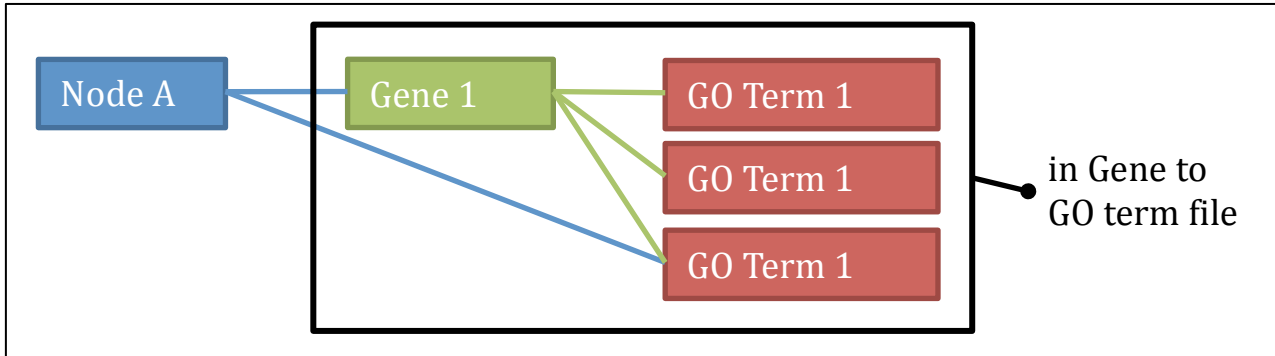




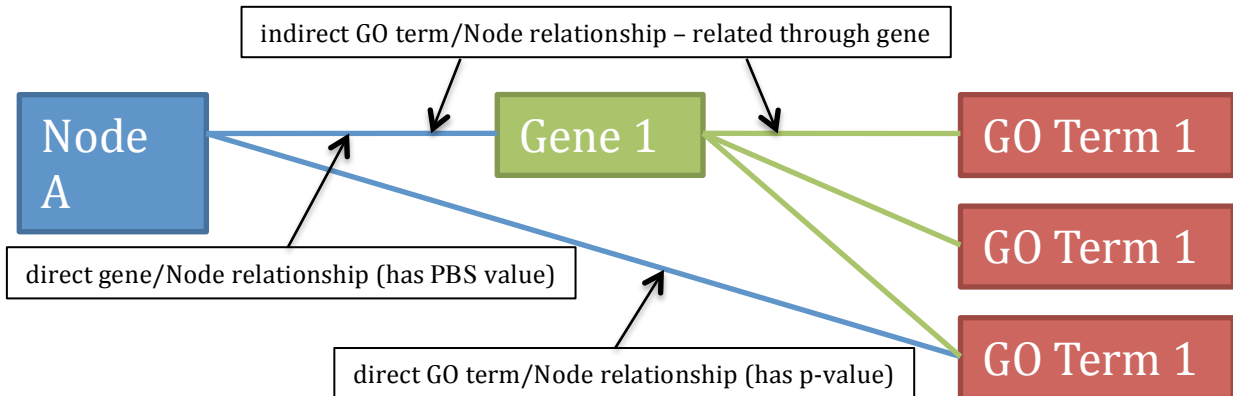
This file maps genes to GO terms. This will include GO terms that are associated with a certain node, but did not have a significant enough p-value to be included in the PBS file. Each line consists of a gene name, a vertical bar ("|") character, and a series of GO terms separated by spaces. This file should be pre-processed by the mini-app ProcessGeneToGoFile.html to limit it to only the relevant genes.

A sample of the format is

```
At1g01770 | GO:0005575 GO:0008150
```



Please note that GO terms are related to nodes indirectly in the gene-to-GO file, and GO terms are related directly to nodes through the PBS file. The GO terms directly related to nodes are assembled through VirtualPlant ([www.virtualplant.org](http://www.virtualplant.org)), as previously mentioned. This relationship will have a p-value, the first will not.



### GO annotation File

This maps a GO term id to the term's description. Each line consists of the GO term id, a vertical bar ("|") character, and the GO term's description. This file can be filtered by the mini-app ProcessGodescriptionFile.html. This will discard any GO terms not used by the PhyloBrowser Cladogram for faster loading.

A sample of the format is

## Data Structures Overview

The data loaded from the four files is stored across several data structures. Some of the mapping structures are complex because of the interconnectedness of the data. For example, sometimes the visualization requires all the nodes for a certain GO term, other times it requires all the GO terms for a certain node. Genes need to be mapped to their related GO terms and also to their related nodes. Hash maps do not support alphabetization, so for the GO term select UI to be alphabetized, a separate array is required for GO term names. The same is true for alphabetizing genes.

The data loaded at initialization is contained in the staticData.js file:

### root

The Newick file data is parsed into a tree data structure stored in the variable root. In addition to the standard tree, the nodes are processed so that each node has a backlink to its parent node. This is needed for the collapsing behavior to work.

### Partitioned Branch Support

As the PBS file is in JSON format, it becomes a JSON object in JavaScript immediately upon loading. No parsing is necessary. During initialization, each node is supplemented with its calculated average PBS value.

### goTermIdToNodesMap

This maps GO term IDs to an array of node ids and, when available, average p-value. It is created during initialisation by processing the PBS and gene-to-GO files.

### goTermNameToGoldMap and goTermIdToGoNameMap

Though it may seem redundant to have two data structures, each which maps in the opposite direction of the other, it is necessary. The goTermNameToGoldMap is needed to get from the GO term select UI to the goTermIdToNodesMap. The goTermIdToGoNameMap is needed to show the list of selected GO term names in the info panel. These are straightforward mappings with no other information besides id-to-name and name-to-id.

### geneMap

The geneMap contains two lists: one of nodes (nodeList) and one of go terms (goTermList). The nodeList is used to filter the tree when a gene is clicked.

## Javascript

Besides the html and css files, there is a library of javascript files which process and present the data. I have tried to keep the roles of each file/module separate for code clarity and organization.

### *JavaScript is fast*

One surprise in working on the PhyloBrowser visualization was that JavaScript is really fast. The "keep extended list" is re-created from scratch every single time a gene or GO term is clicked, or the PBS slider is moved. The GO term selector UI is re-created every time the p-value slider is moved. The visualization is re-created every time its attributes change. The decision to optimize was abandoned because there was no need, and optimization has the potential to introduce extra complexity. I followed the standard priority rules of coding -- "make it work, make it clean, make it fast". Once the code was clean, there was no need for the last part, as JavaScript was already blazingly fast. The code is more robust, clearer, and more maintainable than a version that has complicated (and unnecessary) optimization tweaks.

### *loader.js*

contains functions to

- load data files
  - generic function that takes two parameters:
    - 1) url parameter name
    - 2) name of event to fire after data is loaded
  - wrappers for the generic function to load all the individual files
- load dynamic css file
- upload files with genes and GO terms

### *bigPlantTree.js*

uses the D3 javascript visualization framework to render and update the phylogram.

### **Explanation of D3**

D3 has a different approach to working with data than most other visualization frameworks. Most visualization frameworks combine data and visual information into one object, which makes it easy to understand, inspect and alter the view. For example, if an item has a value below some threshold, make that item invisible – the item encapsulates both data and view. In D3, the visual objects are entirely separate from the data objects. This separation can be initially difficult to understand, as it is such a different paradigm, but once understood it is a clever and elegant way to work with data.

D3's approach is based on three actions: enter, transition and exit. Each action results in a selection of nodes that can then be created, removed, colored, sized, moved or

otherwise manipulated. The enter action will return a selection of new nodes, the transition action will return a selection of nodes which should be updated, and the exit action will return a selection of nodes which are not needed in the visualization.

*“When data is bound to a selection, each element in the data array is paired with the corresponding node in the selection. If there are fewer nodes than data, the extra data elements form the enter selection, which you can instantiate by appending to the enter selection.” -- d3js.org*

Using the previous example of an item whose value falls below some threshold and thus should not be shown in the visualization, D3’s approach will not make the item invisible, but completely remove it from the visualization. When conditions are such that the item should be shown again, D3’s algorithm will include the item as part of the enter action, and the object will be created again.

### Using D3

The nodes are styled by assigning functions to attributes. For example, for the attribute “size” a function is used to define the size of each circle based on the p-value. This makes it easy to introduce complex behavior rather than a fixed value. D3 also provides scaling utilities which can be given a size upper or lower bound.

D3 draws on a page by programatically adding svg elements to the page. This does not require the HTML Canvas element. The elements of a node (circle, text) are subsets of an svg group, so they will be manipulated together. For example, D3 generates code for node N42:

```
<g class="node" transform="translate(400,319.8720397949219)">
  <circle id="N42" r="3.3132346425161523" style="fill: rgb(206, 255, 89);
  stroke: rgb(204, 204, 204);">
  <text x="-10" text-anchor="end" style="fill-opacity: 1;">N42</text>
</g>
```

The tree size needs to be recalculated every time the tree is collapsed or expanded. D3 does not automatically shrink the space for the tree when the nodes are collapsed, so that instead of seeing more compact data when nodes were collapsed, the space between the remaining nodes would stretch to fill up the vertical space. This led to absurd situations where there could be a total of three nodes visible, each five inches apart. To calculate the vertical space, the number of leaves (not nodes!) is counted, and then multiplied by a vertical space constant.

### using the phylogram library

The default cluster layout uses curved lines between nodes which looks graceful but nothing like a phylogram. A JQuery add-on is used to draw the connecting lines with right angles.

## *treeComponents.js*

contains functions to create components to interact with the cladogram, including

- GO term multi-select to filter cladogram nodes
- Gene multi-select to filter cladogram nodes
- p-value slider to filter GO terms in multi-select box
- PBS slider to filter cladogram nodes

## *staticData.js*

stores the data loaded from the four files. The data structures are

- root: the Newick file data
- pbs: the partitioned branch support file's data
- goTermNameToGoldMap
- goTermIdToGoNameMap
- goTermIdToNodesMap: GO term id mapped to a list of nodes

## *dynamicData.js*

stores the data for filtering and otherwise changing the visualization.

- selectedGenes – list of genes from filter component
- selectedGOterms – list of GO term ids from filter component
- keepExpandedList – list of nodes that should be visible and expanded
- lowHighArray – array of two values from PBS Slider

## *collapsing.js*

This class handles collapsing of nodes, both manually (via user click) and programatically (via filter manipulation). These two classes of collapsing need to be maintained separately, since a node that was collapsed manually should remain collapsed even when programatically collapsed nodes are expanded.

“toggleManualCollapse” is the only function for manual collapsing and is relatively straightforward. It sets the variable “manuallyCollapsed” to true.

The function “collapseTree” checks the list of nodes to keep expanded, and marks each of these nodes as “expanded”. It also marks each of these nodes’ parents as expanded (and, recursively, those parents’ parents until the root node). The parent nodes need to be marked, otherwise they will be collapsed and the node hidden.

Then -- and this is critical to the way D3 works -- the children of the unmarked nodes are *removed* from the data structure and stored in a temporary variable. D3’s exit() selection will automatically find the removed nodes, and the D3 code will hide the nodes when the visualization is updated.

One problem with this strategy is that nodes may miss being marked with information. For example, a user may manually collapse node N40. Then, the user may filter by a gene that is associated with a child of N40, but because the child does not exist in the data structure, it is not marked with this information. This does not matter because the child is hidden. However, if the user now manually expands N40, the child should be show that it is associated with the gene (in the current color scheme, it should be orange and have an orange line leading to it). Instead, the orange line will abruptly end at N40, because N40's children did not exist at the time of filtering. Because of this issue, the highlighting is recalculated on any tree update.

## Additional Issues

### Styling

One base css file is used to configure the general visual properties, it is specified in the Cladogram.html file. A second css file is specified in the url for styling specific to a skin. This means it is as easy to change the skin as changing one variable in the url.

There are two dynamically loaded files for skinning the visualization:

- 1) css file: A supplemental css file describes the colors of the html elements. Currently there are two versions – a light background css and a dark.
- 2) colors file in json format: the colors file describes the colors for svg elements in the graph. (Note: it would be cleaner to use a css-formatted file for this as well, and read the css values with javascript. However, certain add-ons – Chrome's Ad-Block, for example – make this buggy and unreliable.)

### Mini-apps

Two mini-apps are provided for winnowing data. The file that provides gene/GO term associations and the file that provides GO term id/GO term name associations are both created for ALL GO terms and genes across ALL organisms. This means that there is a great amount of superfluous data in these files, leading unnecessary load time for the app. These two mini-apps process these files and remove all data bloat. They must be run in the order described, as the second used the first file as an input.

#### ProcessGeneToGoFile.html

This mini-app takes two inputs: the PBS file and the gene-to-GO-terms file. A list of all unique genes is compiled from the data in the PBS file, and used to winnow the genes in the gene-to-GO-term file. In the currently used files, the number of genes was reduced from 18,268 to 3,104.

To use the mini-app, the url must include two parameters: pbsFile and geneToGOFile, for example:

```
../ProcessGeneToGoFile.html?pbsFile=bpPBS.json&geneToGOFile=geneToGoT10.txt
```

The web page will then display the result, which can be cut and pasted into a text file. A good policy is to use the file old name plus the text "Filtered", for example "geneToGoT10Filtered.txt".

#### ProcessGodescriptionFile.html

This mini-app takes two inputs: the winnowed file from the previous mini-app, and the GO-term-to-description file. A list of unique GO terms is gathered from the gene-to-GO-term mappings, and used to winnow the data in the second file. In the currently used files, the number of terms was reduced from 31,917 to 3,592 -- a significant savings of both load and processing time.

To use the mini-app, the url must include two parameters: geneToGOFile and GOannotationFile, for example:

```
../ProcessGODEscriptionFile.html?geneToGOFile=geneToGoT10.txt&GOannotationFile=go-t10-annot.txt
```

The web page will then display the result, which can be cut and pasted into a text file. A good policy is to use the file old name plus the text "Filtered", for example "go-t10-annotFiltered.txt".

## *Loading the Cladogram - explanation of url parameters*

To load the Cladogram web page, six parameters need to be included in the url. Four of these are the aforementioned data files, and two are for custom rendering. At this stage there are two skins for the Cladogram, one with a light background and the other with a dark background. As these are loaded dynamically, the files specifying the visual properties must be included in the url.

The parameter names are

- newickFile

- pbsFile

the winnowed files:

- geneToGOFile

- GOannotationFile

the display files:

- colorsFile

- css

This results in a long url, but allows users to easily change which data files and skinning files to load:

```
../Cladogram.html?newickFile=genus31MP_labeled.tre&pbsFile=bpPBS.json&css=css/bigPlantWhite.css&geneToGOFile=geneToGoT10filtered.txt&GOannotationFile=go-t10-annotfiltered.txt&colorsFile=colors.json
```

## *Technologies*

The Cladogram is written in mainly javascript, with html for web page creation and css for styling.

Several javascript libraries were used, including

### **jQuery**

For element manipulation, user interaction and several elements such as sliders.



## D3

A visualization framework, for layout and styling of the graph. D3 can be used in most modern browsers -- everything except IE8 and below. Parts of D3 may work in older browsers. See

[d3js.org](http://d3js.org)

## Phylogram layout

A plug-in for D3 for Phylogram layout formats the base cluster layout with right angle edges. It can be found at:

<https://gist.github.com/1036776>

## JavaScript Newick format parser

can be found at:

<https://github.com/jasondavies/newick.js>

## Mustache templates for html

Mustache templates allow for separation of javascript and HTML when generating new page elements. Without templates, string concatenation must be used. The Mustache templates can be externalized in a separate html file and called on when needed.

<http://mustache.github.com/>

## Tools

Several IDEs, debuggers and other tools assisted in development:

### WebStorm

For javascript, html and css coding. This IDE proved to be extremely useful in helping write Javascript code. Its highlighting, code complete and formatting capabilities made it much easier to organize code and find bugs.

### Firebug and Chrome Developer Tools

For testing and debugging, in particular the console, DOM exploration, and capability to set breakpoints.

### Minimizers

Several on-line javascript minimizers are available to shrink the source code. These remove whitespace, optimize and shorten variable names to shrink the size of the code. I used Google Closure Compiler because it checks for errors while compiling, and because it allows for several levels of compiling.

“Whitespace Only” does not shrink variable or function names, and does not optimize, but is helpful for debugging since the names are still meaningful rather than, for example, “function a(b)”.

“Simple Optimizations” can be used after the code has been thoroughly tested since the code will be unrecognizable from its original form.

“Advanced Optimization” is the strongest but has the capacity to break the code.

I keep the javascript code separated into many files for easier organization, so they need to be concatenated in order to minimize them. The easiest way is to grab the list of javascript files from the html file (excluding libraries such as D3 or JQuery) and used a command shell to concatenate them with the command

```
cat file1.js. file2.js > concatenatedJS.js
```

The contents of concatenatedJS.js can then be compiled at

<http://closure-compiler.appspot.com/home>

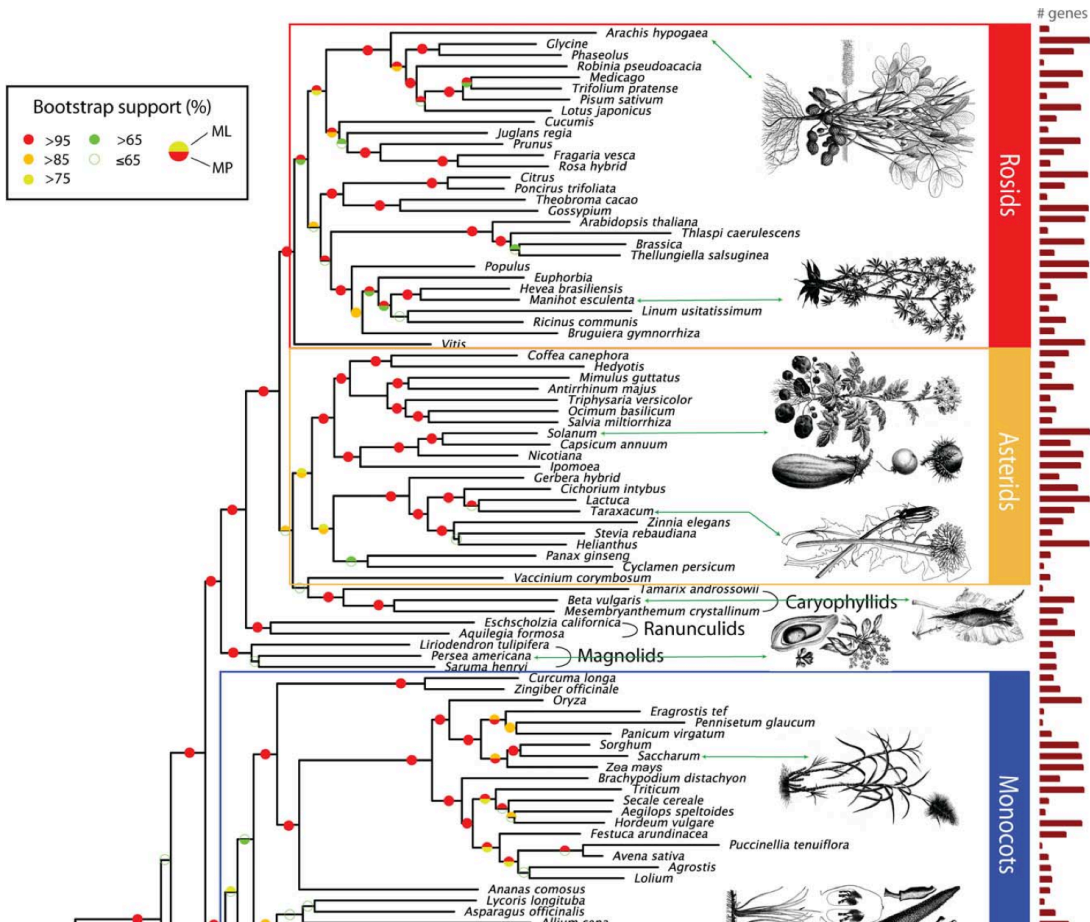
The naming convention for minimized files is to add min, so that concatenatedJS.js would be saved as concatenatedJS.min.js after minimizing.

## ***Future work***

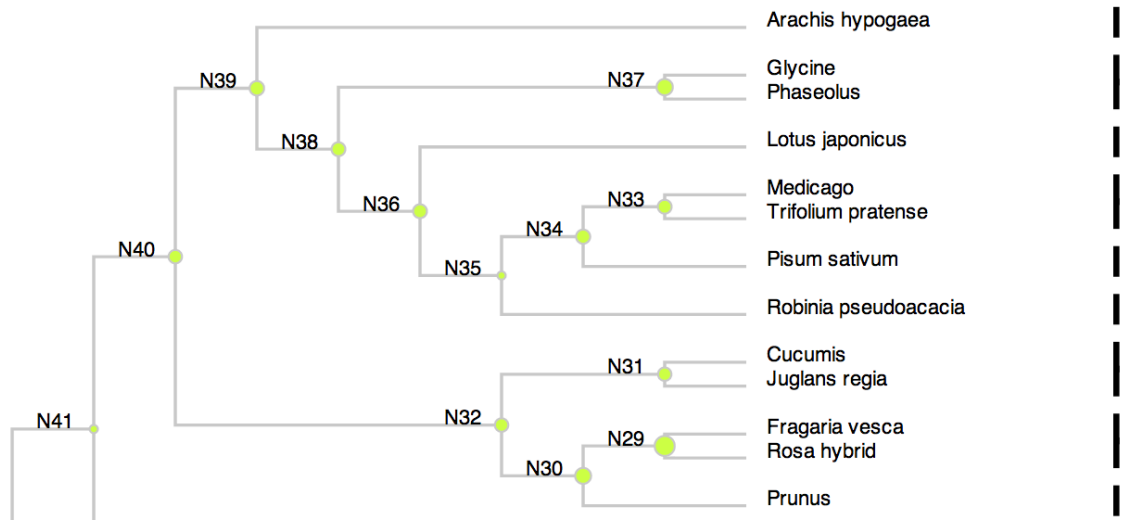
### **Family names & rectangles**

One visual addition to the PhyloBrowser visualization was not able to be completed due to unavailable data. This was the color bar legend for clades to the right of the tree leaves. This was based on the paper “A Functional Phylogenomic View of the Seed Plants (Lee et al 2011). An excerpt figure from this paper is shown below to demonstrate this legend. The color bar legend idea can be seen in the vertical red, yellow and blue bars on the right showing the clades rosids, asterids and monocots.

This figure of the BigPlant Cladogram is from Lee et al 2011.



There is a placeholder in the code for showing something similar. Each species has a rectangle to the right of it that can be filled with red, yellow, blue, or any legend color. Because there is currently no data, the rectangles show as black. Until the data exists, the code to show the rectangles is commented out, but is shown below:



## REFERENCES:

Katari MS, Nowicki SD, Aceituno FF, Nero D, Kelfer J, Thompson LP, Cabello JM, Davidson RS, Goldberg AP, Shasha DE, Coruzzi GM, Gutierrez RA (2010) "VirtualPlant: A software platform to support Systems Biology research." *Plant Physiol.* 152(2): 500-15. (BigPlant Reference).

Lee E, Katari M, Kolokotronis S, Cibrian A, Stamatakis A, Ott M, Little D, Stevenson D, McCombie WR, Chiu J, Martienssen R, Brenner E, Coruzzi G, DeSalle R (2011) "High resolution phylogeny of the seed plants: A functional phylogenomic view." *PLoS Genetics* Dec;7(12):e1002411. Epub 2011 Dec 15. (Virtual plant reference)

Sorenson MD, Franzosa EA (2007) *TreeRot*. 3 ed: Boston University, Boston, MA. (PBS reference)