Visualizing Phylogenetic Trees by Spring-Embedder Models

Yaw-Ling Lin* and Po-Shun Yu

Department of Computer Science and Information Engineering

Providence University

Taichung 433, Taiwan, ROC

yllin@pu.edu.tw peteryu@cs.pu.edu.tw

Received 26 April 2006; Revised 26 October 2006 and 23 January 2007; Accepted 12 March 2006

Abstract. A phylogenetic tree is a graphical representation of the evolutionary relationship between taxonomic groups. The term phylogeny refers to the evolution or historical development of a plant or animal species, or even a human tribe or similar group. In order to make ourself to understand the structure of phylogenetic tree. We present a spring-embedder model for drawing rooted and unrooted phylogenetic trees with straight edges. Our heuristic strives for uniform edge lengths, and we develop it in analogy to forces in natural systems, for a simple, elegant, conceptually intuitive, and efficient algorithm. These algorithms are implemented on a web-based phylogeny visualization system that interoperates with existing tools developed for phylogenetic processing including CLUSTAL W, PHYLIP, PAUP. The molecular biologists can also manually construct their phylogenetic tree via existing system in, e.g., the Phylip format produced by CLUSTAL W as input format of the web system to which this data is to be fed.

Keywords: algorithm, phylogenetic trees, trees drawing, computational biology, force-directed placement, computational geometry

1 Introduction

Trees are widely used to represent evolutionary, historical, or hierarchical relationships in various fields of classification. Biologists use the information contained in the DNA sequences of a collection of organisms, or taxa, to infer the evolutionary relationships among those taxa. *Phylogenetic trees* typically represent the evolutionary history of a collection of extant species or the line of descent of some genes, and may also be used to classify individuals (or populations) of the same species. Numerous phylogenetic inference methods, e.g. maximum parsimony, maximum likelihood, distance matrix fitting, subtrees consistency, and quartet based methods have been proposed over the years [1,2,3,4,5,6,7].

Furthermore, it is rather common to compare the same set of species w.r.t. different biological sequences or different genes, hence obtaining various trees. The resulting trees may agree in some parts and differ in others. In general, one is interested in finding the largest set of items on which the trees agree. This fact motivates the compelling need to compare different trees in order to achieve consensus or extract partial agreements. For measuring the similarity / difference between trees, several distance measures have been proposed [8], e.g. the symmetric difference metric [9], the nearest-neighbor interchange metric [10], the subtree transfer distance [11], the Robinson and Foulds (RF) metric [12], and the quartet metric [13,14].

The descendent subtree of a phylogenetic tree T is the subtree composed by all edges and nodes of T descending from a vertex. In our previous works [15], we presented linear time algorithms for finding all leaf-agree descendent subtrees as well as all isomorphic descendent subtrees. We also showed that computing all pairs normalized cluster distances between descendent subtrees of two phylogenetic trees can be done in linear time. Furthermore, we showed that finding nearest subtrees for a collection of pairwise disjointed subsets of leaves can be done in $O(n)$ time.

1.1 Graph Drawing

Information visualization has become a large field and sub fields are beginning to emerge. The abstract combinatorial relation among objects is usually represented by a graph; thus 1Information visualization has become a large field and sub fields are beginning to emerge. The abstract combinatorial relation among objects is usually

* Correspondence author

represented by a graph; thus the handling of graphs is considered with respect to information visualization. The visualization of graphs is one of the most important subjects in the field of the information visualization. Excellent bibliographic surveys [17,18,19,20,21] exist for graph drawing.

The basic graph drawing problem can be put simply: given a set of nodes with a set of edges (relations), calculate the position of the nodes and the curve to be drawn for each edge. Of course, this problem has always existed, for the simple reason that a graph is often defined by its drawing. The annotated bibliography by Battista et al. [22] gathers hundreds of papers studying what a good drawing of a graph is. That is where the problem becomes more intricate: it requires the definition of properties and a classification of layouts according to the type of graphs to which they can be applied.

Many constraints in use are also expressed in terms of aesthetic rules imposed on the final layout. Nodes and edges must be evenly distributed, edges should all have the same length, edges must be straight lines, isomorphic sub-structures should be displayed in the same manner, edge-crossings should be kept to a minimum, etc. Trees have received the most attention in the literature. Consequently, additional aesthetics rules have also been formulated for them. For example, nodes with equal depth should be placed on a same horizontal line; distance between sibling's nodes is usually fixed, etc. The Reingold and Tilford algorithm for trees [23] is a good example of a layout algorithm achieving these aesthetics goals. Isomorphic subtrees are laid out in exactly the same way, and distance between nodes is a parameter of the algorithm. On the other hand, the more straightforward and naive algorithm for displaying a tree, consisting of distributing the available horizontal space to subtrees according to their number of leaves, actually fails to achieve some of the aesthetic rules listed above. Although the adjective "aesthetic" is used, some rules were originally motivated by more practical issues. For instance, minimization of the full graph area might be an important criterion in applications. Some of the rules clearly apply to a certain category of graphs or layouts only; others have a more "absolute" character.

2 Force-directed Placement

Spring embedding is a local optimization technique, which starts with an initial (usually random) layout and then iteratively improves this layout by viewing edges between nodes as springs, thus leading to attractive and repulsive forces based on the desired distance between the nodes. Spring embedding [24,25,26,27] uses the physical metaphor of springs.

According to Hooke's law, there are attractive and repulsive forces based on the desired distance between the nodes. The spring embedding algorithm starts with a random layout and then computes for a number of iterations the forces for each object and moves it into the direction of the overall force until a state of minimal energy is reached. Nodes are usually not moved by the full amount of the force, but are limited by a maximum amount known as temperature. In a technique known as temperature scheduling the temperature is reduced per iteration, so nodes are more and more limited in their movement.

A number of algorithms have been developed based upon the spring embedding idea; an influential algorithm extended later in various ways is Eades' algorithm [28]. Besides springs of logarithmic strength governed by Hook's law it views nodes and non-adjacent vertices as electrically charged particles repelling each other.
Fruchterman and Reingold [24] presented a modification of the force directed, spring-embedder model of Eades [28] for drawing undirected graphs with straight edges. In analogy to forces in natural systems, the heuristic method computes attractive and repulsive forces and simultaneously moves all nodes according to these forces, where the moved distance is bounded by a temperature t. This process is iterated for some rounds. The algorithm terminates if either the maximal force acting at a node falls below a user defined threshold or if the maximal number of iterations is exceeded.

Kamada and Kawai's algorithm [26] is based on graph-theoretic distances between pairs of vertices. There the graph nodes are considered as particles that are all connected by springs whose ideal lengths are equal to the graph-theoretic distances between their two endpoint particles multiplied by the desirable length of one edge. The goal of the algorithm is to find a balanced spring system. The major drawback of this method is its high computational cost as partial differential equations need to be solved.

Davidson and Harel's algorithm [29] uses simulated annealing. In each step, the layout is improved by comparing the current position of a node to one randomly selected with the neighborhood of the node. Step by step, the temperature is reduced and the neighborhood gets smaller. The comparison of current to randomly selected position involves weighted factors for a number of criteria such as overall stress, edge crossings, etc.

2.1 Hook's Law

The behavior of the spring is usually governed by the Hook's law. Assume A and B are two mass points connected with a spring. Let $\vec{L}$ be the vector pointing from B to A. Let R be the spring rest length. Then, the elastic force exerted on is:

$$\vec{F} = -k_{\text{Hook}}(|\vec{L}| - R)\vec{L}/|\vec{L}|) \quad (1)$$

2.2 Coulomb's law

The precise magnitude of the electric force that a charged particle exerts on another is given by Coulomb's law. That is, the magnitude of the electric force that a particle exerts on another particle is directly proportional to the product of their charges and inversely proportional to the square of the distance between them. The direction of the force is along the line joining the particles. Let q_1 and q_2 be the charges of particle 1 and particle 2, respectively; $\vec{r}$ is the vector joining particle 1 to particle 2. Let ε_0 be the permittivity constant $(\varepsilon_0 = 8.85 \times 10^{-12} C^2/(N \cdot m^2))$. Then we have:

$$\vec{F} = \frac{q_1 q_2 \vec{r}}{4\pi\epsilon_0 |\vec{r}|^3} \quad (2)$$

This formula applies to elementary particles and small charged objects as long as their sizes are much less than the distance between them.

2.3 Force Equilibrium

Consider the following analogy: the leaf nodes behave as atomic particles exerting repulsive forces on one another; the forces induce movement. Our algorithm will resemble molecular simulations, sometimes called n-body problems. Following Eades and Fruchterman et al, however, we can apply unrealistic forces in an unrealistic manner. For example, we can make only vertices that are neighbors attract each other, but all vertices repel each other. Though inspired by natural systems such as springs or macro-cosmic gravity, it must be pointed out that the "forces" may not be correctly named. Forces are used to calculate velocity for every time quantum (and thus displacement, since the time of a quantum is unity), whereas true forces induce acceleration. The distinction is important, because the real definition leads to dynamic equilibria (pendulums, orbits), and we seek static equilibria.

Pseudo-code for the algorithm is given in Figure 4. We have not said anything about the initial configuration, the input, or the output. The initial configuration could be all or partly specified, but normally vertices are placed randomly in the frame. Different functions might have been chosen for $\overrightarrow{F_L}$ and $\overrightarrow{F_R}$.

The ideas behind the algorithm is as following. First the initial displacement routine layouts the leaf nodes on a circle as their natural ordering in the topological tree (depth first) searching order. In the following, we layout the vertices with level 1, V_1, in a smaller circle. The subsequent vertices of V_2, V_3,…, $V_{\ell-1}$ are then set up in similar manner; finally the centers V_ℓ are placed around the origin. Now the problem comes to simulate the force-directed placement as shown in Figure 4. Note that the **REPLACE** (i) routines are scheduled in the order from level 0 to level $\ell - 1$ in their adjacent ordering, while the centers are fixed at the origin as the anchor points.

In the aesthetic viewpoint, symmetry and homogenized are very important elements and we try to find some way to make our graph more smooth and good looking for these two reasons. Hook's law and coulomb's law are very useful when we draw a tree graph. When tree's edge becomes too long or the length's variation too high, the hook's law will reduce it variation. When tree's leaf becomes high density or sparseness, the coulomb's law will make it regularly distribution. According to these two constraints the graph will better than before.

2.4 Changing the Looks

In a phylogenetic tree, every leaf node represents a species, each edge denotes a relationship between two neighboring species, and the length of an edge indicates the evolutionary distance between species. The distance of a path in a phylogenetic tree must be as close as the evolutionary distance between two species. It is desirable for the phylogenetic viewing system to be capable of supporting various types of drawings: unrooted tree, radial tree, rooted tree, slanted cladogram, rectangle cladogram, and phylogram. Because there are many kinds of phylogenetic tree formats, supportability for various types of input data and interactive editing are major concerns in evaluating drawing software.

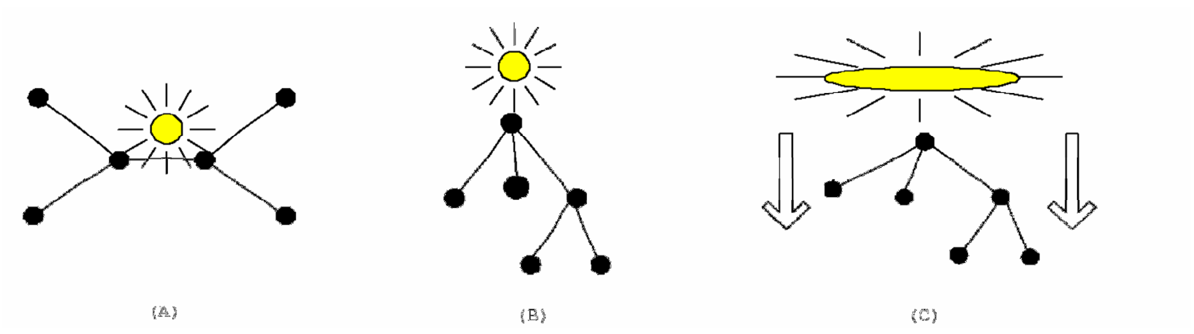


Fig. 1. Different drawings of a same tree. Diagram (A) draws the usual radial style tree; diagram (B) shows a rooted tree by placing a single strong pole near the root; diagram (C) shows another rooted tree by placing a fixed unidirectional force field from the top.

2.5 Input and Format

The phylogeny visualization system can interoperate with existing tools developed for phylogenetic processing, e.g., CLUSTAL W [30], BLAST [31], PHYLIP [32], PAUP [33]. These existing tools are treated as software components. The molecular biologists can also manually construct their phylogenetic tree via existing system in, e.g., the Phylip format produced by CLUSTAL W or Nexus format for PAUP, as input format of the web system to which this data is to be fed. It will read tree data in Phylip format, and then display graphical views of the phylogenetic tree. It also provides several editing functions to give a user-friendly interface. By using several control parameters, users can easily and interactively manipulate the shape of phylogenetic trees. There are various options allow you to modify.

3 The Tree Drawing Algorithm

The degree of a vertex v in G, written $\deg_G(v)$, is the size of its neighborhood. A vertex v is a *leaf* (or *end-point*) of a graph G if $\deg_G(v) = 1$. The longest distance among all vertices of a graph is called its *diameter*. A tree with diameter less than or equal to 2 is called a *star*. Given a tree T, we can delete all the leaves of T resulting in a smaller tree T'. This trimming operation defines a function $trim(T) = T'$). The trimming operation can be repeated until the remaining subgraph T' is empty. Since each vertex v in T will eventually be trimmed, we can associate an integer with v specifying the number of trimming operations taken before v becomes a leaf. This *level* function from $V(T)$ to $[0 \ldots \lfloor (n-1)/2 \rfloor]$ is recursively defined as follow:

$$level(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf of } T \\ k & \text{if } v \text{ is a leaf of } trim^{(k)}(T) \end{cases}$$

Thus, whenever we traverse nodes of a phylogenetic tree T in their *increasing* levels ordering, we ensure that the "outer" nodes have already been visited before "inter" nodes. The reader can easily check that the algorithm CALC-LEV(T) shown in Figure 2 correctly computes the level of each vertex within an unrooted tree in linear time.

Given a (rooted or unrooted) $T = (V, E)$, the general approach of our algorithm first partitions the vertices in terms of their levels. Let $V_i = \{v \in V \mid level(v) = i\}$ denote the set of vertices with level i. Let $\ell = \max\{level(v) \mid v \in V\}$ denote the the largest level of the given tree. Note that a vertex v with level ℓ is the *center* of the tree; it is easily seen that a tree has at least one and at most two centers.

Theorem 1 *Given an unrooted tree T with n vertices, the algorithm CALC-LEV(T) correctly computes levels of vertices of T in $O(n)$ time.*

Proof. Let u be the starting leave, and let c be the center of T whose distance to u is larger. Recall that the number of centers in T is either one or two. Furthermore, let P denote the path from u to c in T. It is readily verified that the procedure UB-LEV finds the correct level value for each vertex v in T *unless* v is a vertex lies upon the path P. The situation is illustrated at Figure 3.

```
CALC-LEV($T$)        ▷ Main routine for computing the node levels of an unrooted tree $T = (V, E)$.
1 Let $u$ be a leaf node with $deg(u) = 1, uv \in E(T)$; let $level[u] \leftarrow 0$.
2 UB-LEV($u, v$)
3 FN-LEV($u, v$)

UB-LEV($u, v$)        ▷ Upper bound of level of $v$.
1 $\ell_{\max} \leftarrow -1$
2 for each vertex $x \in Adj[v] \setminus \{u\}$ do
3     $\ell \leftarrow$ UB-LEV($v, x$)
4     if $\ell_{\max} < \ell$ then $\ell_{\max} \leftarrow \ell$
5 $level[v] \leftarrow 1 + \ell_{\max}$; return $level[v]$

FN-LEV($u, v$)        ▷ Finer modification for $level[v]$.
1 if $level[v] > 0$ then $level[v] \leftarrow 1 + 2\text{nd}\max\{level[x] \mid x \in Adj[v]\}$
2 for each vertex $x \in Adj[v] \setminus \{u\}$ do
3     if $level[x] > level[v] + 1$ then FN-LEV($v, x$)
```

Fig. 2. A linear time algorithm for calculating the node levels of an unrooted tree. It is used to make our tree graph drawing easier.

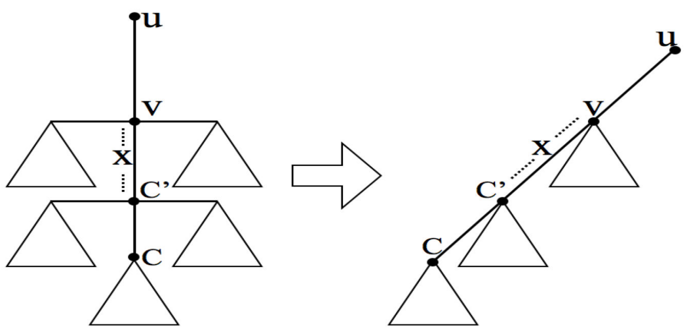


Fig. 3. The procedure UB-LEV (u, v) over estimates levels of vertices in T.

In the following, we show that procedure FN-LEV(u,v) correctly identifies vertices of P and reset each vertex to the correct level. It is not hard to verify that the correct level of v, $level[v]$, that was over estimated in UB-LEV(u,v), shall just be one plus the second highest level of neighbors of v, as illustrated in Figure 3. Thus Step 1 of FN-LEV(u,v) refines $level[v]$ accordingly.

To further refine the level of each vertex x on P, FN-LEV(u,v) recursively calls upon itself on Step 3 when the correct followed vertex x is identified by the condition that x was over estimated with condition $level[v]$ being greater than $level[v]+1$ as being tested at Step 3.

For the time-complexity analysis, it is easily observed that both procedures UB-LEV (u,v) and FN-LEV(u,v) are basically the typical post-ordered tree traversal algorithm; thus the time complexity is exactly $O(n)$, where n denote the number of vertices on T. □

The algorithm produces it's drawing in three distinct stages: partition of vertices, initial placement, and refinement. First, after the level partition of vertices into $\{V_0,V_1,\ldots,V_\ell\}$, we pin point the centers (V_ℓ) of T at the origin. Then, we add the vertices of $V_{\ell-1}$ by placing them initially at the positions determined by their graph distances to a subset of the elements of V_ℓ. The positions of the vertices in $V_{\ell-1}$ are modified using a force-directed layout method. This process of adding new vertices and refining their positions is repeated for $V_{\ell-2},\ldots,V_1,V_0$. The refined positions of the elements of V_0 constitute the final layout of the vertices of T. Note that we only draw vertices of T up to this point. When all the vertices have been placed we draw the edges of T as straight-line segments connecting their endpoints.

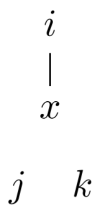
```
DRAW-PHYLO($T$)      ▷ draw a phylogenetic tree $T = (V, E)$.
**parameter** $MaxIterate$: iterations upper bound; $\delta$: negligible displacement distance.
1 Partition vertices $V = V_0 + V_1 + \cdots + V_\ell$ into different levels; $V_\ell$ is the center nodes.
2 INITIAL-DISPLACEMENT($T$)
3 $k \leftarrow 0; \Delta d \leftarrow \infty$
4 **while** $k \le MaxIterate$ and $\Delta d > \delta$ **do**      ▷ check boundary condition.
5       $\Delta d \leftarrow 0$
6       **for** $i \leftarrow 0$ **to** $\ell - 1$ **do**      ▷ Displacement from the outside within.
7             REPLACE(i)
8       $k \leftarrow k + 1$
9 **return** $pos[v], v \in V(T)$ represents the placement of vertices on the plane.

REPLACE($i$)      ▷ Adjusting positions of vertices at level $i$.
**parameter** $c_L$: Leaves force coefficient; $c_R$: Root force coefficient.
1 **if** $i \ge 1$ **then for each** vertex $v \in V_i$ **do**      ▷ internal nodes
2       $pos[v] \leftarrow (\sum_{u\in Adj[v]} mass[u] \cdot pos[u]) / \sum_{u\in Adj[v]} mass[u]$
3 **else for each** vertex $v \in V_0$ **do**      ▷ leaves
4       $\vec{F}_L \leftarrow \sum_{u\in V_0\setminus\{v\}} q[u]\cdot q[v]\cdot \overrightarrow{(pos[u]-pos[v])}/\|u,v\|^3$      ▷ Coulomb: electric force
5       $\vec{F}_R \leftarrow \sum_{u\in Adj[v]} \vec{F}(v,u)$      ▷ Hook: springs' force.
6       $pos[v] \leftarrow pos[v] + c_L \cdot \vec{F}_L + c_R \cdot \vec{F}_R$      ▷ Adjusting the leaf $v$.
```

Fig. 4. Drawing phylogenetic trees using force-directed displacement.

3.1 Placements and Refinement of Vertices

The second and third phase of the algorithm is the placement and refinement stages, respectively. In the *i*-th placement stage, the vertices of set V_i are intelligently placed in are intelligently placed in R^2. In the *i*-th refinement stage a local force-directed method is used to obtain better positions for the vertices of V_i. After the placement and refinement phases for V_i have been completed, the process is repeated for V_{i-1}, V_{i-2}, all the way to V_0. Consider the general placement case. Suppose the refinement and placement phases for V_i have been completed and we want to start the placement phase for V_{i-1}.

Note that vertices of V_{i-1} are adjacent to at least one vertex of V_i and possibly some vertices of level > i. The idea behind the intelligent placement is that every vertex v is placed "close" to its optimal position as determined. The intuition is that if we can place the vertices close to their optimal positions from the very beginning, then the refinement phases need only a few iterations of a local force-directed method to reach the minimal energy state.



For example, the following "three closest to x vertices" strategy starts by setting $pos[x]$ to the center of the mass $(mass[i] \cdot pos[i] + mass[j] \cdot pos[j] + mass[k] \cdot pos[k]) / (mass[i] + mass[j] + mass[k])$ of i, j, and k, the three vertices closest to x. This is followed by a force-directed modification of the position vector of x with the energy function E calculated only at the three points i, j, and k. This makes the procedure very fast, and in our tests it produced good results. More details about the placement algorithm can be found in [24].

While the refinement is calculated using a force-directed method, it is important to note that the forces are calculated locally. For each level of the filtration V_i, we perform REPLACE(i) to update the vertex positions, where REPLACE(i) is a scheduling function which can be specified at the beginning of the execution. The scheduling routine REPLACE(i) and the tree drawing algorithm DRAW-PHYLO(T) are illustrated at Figure 4.

3.2 Speeding up the algorithm

An important technique in n-body simulations is to approximate the effect of distant bodies as a single pole. Doing this reduces the n-body simulation from $\theta(n^2)$complexity to $\theta(n\log n)$. On the other hand, we need not faithfully imitate a celestial, chemical, or atomic system – we desire only that the results be pleasing. This allowed us to make one timesaving adjustment already described: vertices are only repulsive to their neighbors, resulting an *O(n)* time complexity. Another possibility is to approximate the set of leaf nodes as a single large pole. Doing so, the repulsive force of each leaf can be computed in constant time without scarifying the aesthetics goals.

3 Experimental Results

The resulting phylogeny drawing / visualization routine is rather efficient. The implementation of the algorithm Experimental Results the resulting phylogeny drawing / visualization routine is rather efficient. The implementation of the algorithm is written in Perl CGI scripts run under the FreeBSD 5.4 system with Apache web server V1.3, while the experimental machine is equipped with dual Pentium-III CPU for 1000Mhz, 1000MB DRAM and 480 GB disk using RAID 5. Our tested consists of a server and some random tree generators, which are supported by bioperl connected via a 100 Mb/sec fast Ethernet switch.

Just as an example of showing the efficiency of the force-directed displacement algorithm, we note that these integrated Perl programs produce the embedding / layout of a given 500 nodes phylogenetic tree in the average of about 0.0713 seconds; the experiment is done by randomly generating 1,000 trees and taking the average out of these total drawing times. On the other hand, consumption of system resource only 6492k of memory and no I/O operations.

The system is still under developed, and some of our preliminary results can be access via our phylogeny visualization web site at http://bioinfo.cs.pu.eud.tw/drawtree/ [34]. Most of our drawing algorithms for phylogenetic trees have been incorporated into the visualization system with easy Internet accessibility. It is mostly built by using the conventional Perl CGI scripts, which has a long reputation of being easily integrated with HTML and CGI programs.

Our method is focus on the shape and executive speed. We do not use the real data because our graph will not show the node information; maybe we complete it in the future. Another reason is that even if using the meaningful data set the shape will very similar to the result by our program. We support another viewpoint of phylogenetic tree, and make it generate fast. Some of method may build the graph very slow, and the graph is hard to identify. That is why we had to do.

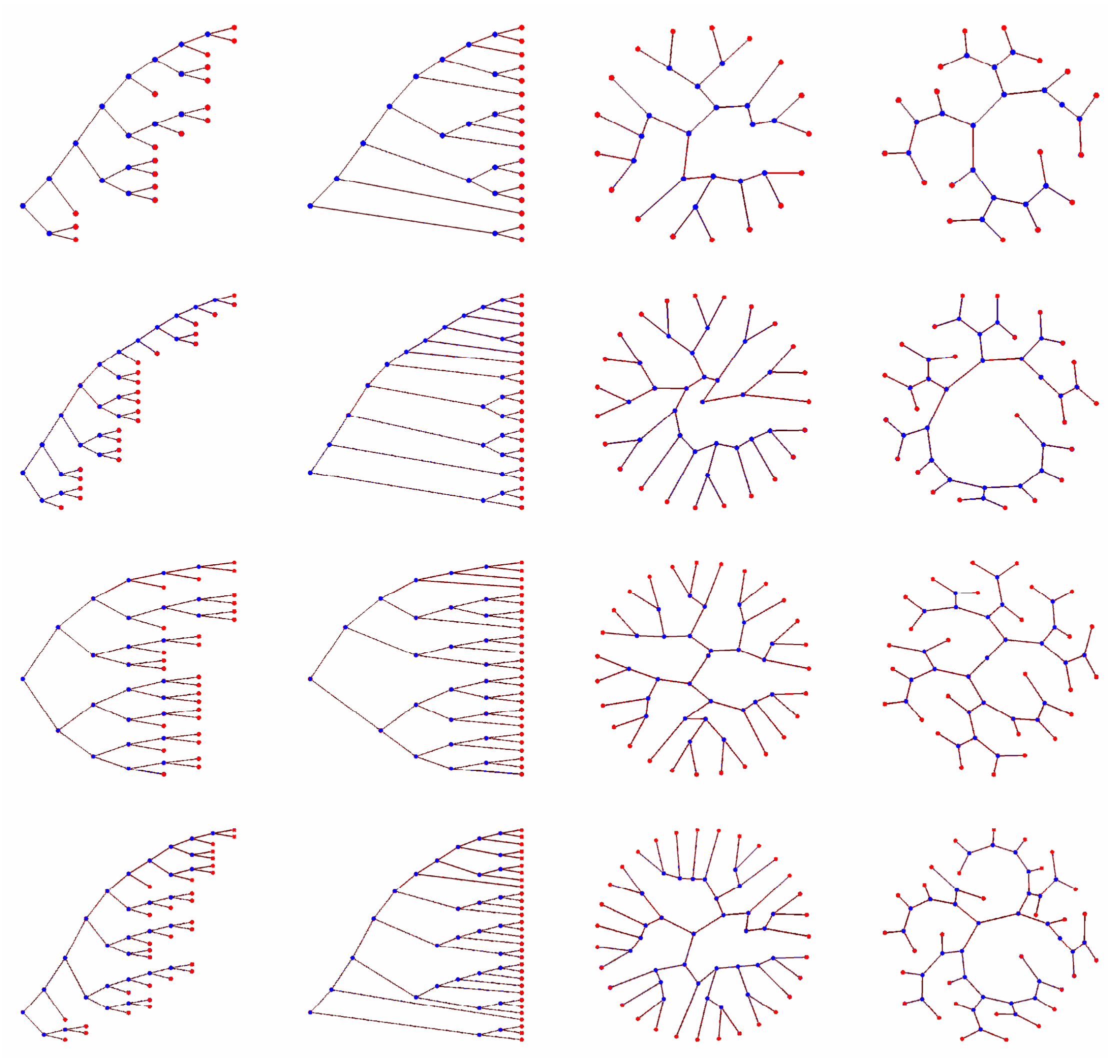


Fig. 5. Visualizations of four random trees, each with four different views including two rooted views and the initialization setting of the unrooted version of the tree and the force-displacement balanced final results produced by our phylogeny visualization system.

5 Concluding Remarks

For now, our method provides a linear time algorithm for tree drawing. It's very fast and easy to implement using any programming language. We change our tree drawing style to make it easy to identify all nodes; you can see it very clearly.

Also we tried to distribute the whole species in a phylogenetic tree as uniformly as possible on the whole output screen. Users can select the tree type (rectangular cladogram, slanted cladogram, phylogram, unrooted tree, or radial tree), resize the tree, and change the branching patterns of the phylogenetic tree (rooted, unrooted, or half-rooted). In the future, we may try to make our program become a module of bioperl to make more people to use.

6 Acknowledgement

The authors would like to thank Hsun-Chang Chang and Ying-Hung Lin for their assistance of building up the Web system, and we thank many instructive discussions and information concerning the biologic phylogeny visualization web system within the Providence University Bioinformatics Forum [35].

References

[1] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, "Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions", *SIAM Journal on Computing*, Vol.0 No.3, pp.405–421, 1981.

[2] V. Berry and O. Gascuel "Inferring evolutionary trees with strong combinatorial evidence", *Theoretical Computer Science*, Vol.240, No.2, pp.271–298, 2000.

[3] J. Felsenstein, "Numerical methods for inferring evolutionary trees", *Quarterly Review on Biology*, Vol.57, No.4, pp.379–404, 1982.

[4] W. M. Fitch, "Toward defining the course of evolution: Minimal change for a specific tree topology", *Systematic Zoology*, Vol.20, pp.406–441, 1971.

[5] D. Gusfield, "Efficient algorithms for inferring evolutionary trees", *Networks*, Vol.21, pp.19–28, 1991.

[6] N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogenetic trees", *Molecular* Biology Evolution, Vol.4, pp.406–425, 1987.

[7] K. Strimmer and A. von Haeseler, "Quartet puzzling: a quartet maximum-likelihood method for reconstructing tree topologies," *Molecular Biology and Evolution*, Vol.13, No.7, pp.964–969, 1996.

[8] DasGupta, He, Jiang, Li, Tromp, and Zhang, "On distances between phylogenetic trees", *In Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp.427–436*,* 1997.

[9] D. F. Robinson and L. R. Foulds, "Comparison of weighted labelled trees", *In Combinatorial mathematics, VI (Proc. Sixth Austral. Conf., Univ. New England, Armidale)*, *Lecture Notes in Mathematics 748,* Springer-Verlag, Berlin, pp.119–126, 1979.

[10] M. S. Waterman and T. F. Smith, "On the similarity of dendrograms," *Journal of Theoretical Biology*, Vol.73, pp.789–800, 1978.

[11] B. L. Allen and M. Steel, "Subtree transfer operations and their induced metrics on evolutionary trees", *Annals of Combinatorics*, Vol.5, pp.1–13, 2001.

[12] D. F. Robinson and L. R. Foulds, "Comparison of phylogenetic trees", *Math. Biosci*, Vol.53, No.1-2, 1981, pp.131–147.

[13] G. S. Brodal, R. Fagerberg, and C. N. Pedersen, "Computing the quartet distance between evolutionary trees in time O(n log2 n)", *ISAAC, Lecture Notes in Computer Science* 2223, Springer-Verlrg, Berlin, pp.731–742, 2001.

[14] G. Estabrook, F. McMorris, and C. Meacham "Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units", *Systematic Zoology*, Vol. 34, No.2, pp.193–200, 1985.

[15] Y. L. Lin and T. S. Hsu, "Efficient algorithms for descendent subtrees comparison of phylogenetic trees with applications to co-evolutionary classifications in bacterial genome", *In The 14th Annual International Symposium on Algo-*

rithms and Computation (ISAAC'03), *Lecture Notes in Computer Science 2906*, Springer-Verlag, Berlin, pp. 339–351, 2003.

[16] G. di Battista, P. Eades, R. Tamassia, and I.G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1999.

[17] J. Diaz, J. Petit, and M. Serna, "A survey on graph layout problems" *ACM Computing Surveys*, Vol.4, 2002, pp.313–356.

[18] F. Cruz and R. Tamassia, "Online tutorial on graph drawing", *http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf.*

[19] I. Herman, G. Melan, and M. S. Marshall, "Graph visualization and navigation in information visualization: A survey", *IEEE Transactions on Visualization and Computer Graphics*, Vol.6, No.1, pp.24–43, 2000.

[20] R. Tamassia, "Graph drawing," In *CRC Handbook of Discrete and Computational Geometry, Jacob E. Goodman and Joseph O'Rourke, editors*, CRC Press, 1997.

[21] R. Tamassia, "Advances in the theory and practice of graph drawing," *Theoretical Computer Science*, Vol.217, No.2, pp.235–254, 1999.

[22] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, "Algorithms for drawing graphs: an annotated bibliography", *Computational Geometry Theory and Applications*, Vol.4, pp.235–282, 1994.

[23] E.M. Reingold and J.S. Tilford, "Tidier drawing of trees", *IEEE Transactions on Software Engineering*, Vol.7, No.2, pp,223–228, 1981.

[24] J. Fruchterman and M. Reingold, "Graph drawing by force-directed placement", *Software - Practice and Experience*, Vol.21, No.11, pp.1129–1164, 1991.

[25] G. d. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1999.

[26] T. Kamada and S. Kawai "An algorithm for drawing general undirected graphs" *Information Processing Letter*, Vol.31, No.1, pp.7–15, 1989.

[27] J. Kruskal, "Multidimensional scaling by optimizing goodness to fit to non-metric hypotheses", *Psychometrika*, Vol.29, pp.1–27, 1964.

[28] P. Eades, "A heuristic for graph drawing", *Congressus Numerantium*, Vol.42, pp.146–160, 1984.

[29] R. Davidson and D. Harel, "Drawing graphs nicely using simulated annealing", *ACM Transactions on Graphics*, Vol.15, No.4, PP.301–331, 1996.

[30] J.D. Thompson et al., "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment," *Nucleic Acids Research*, Vol.22, pp.4673–4680, 1994.

[31] S.f. altschul and m.s. boguski and w. gish and j.c. wootton, "Issues in searching molecular sequence databases," *Nature Genet.*, Vol.6, pp.119–129, 1994.

[32] J. Felsenstein, *Phylip: Phylogeny inference package*, Version 3.5c, 1993.

[33] D.L. Swofford et al., "Phylogenetic inference," In *Molecular Systematics*, Sinauer Associates, Inc, 2nd edition, 1996.

[34] Providence University. Phylogeny visualization system. http://bioinfo.cs.pu.edu.tw/PHD/.

[35] Providence University. Bioinfo forum. http://bioinfo.cs.pu.edu.tw/.