Optimal Leaf Ordering for Two and a Half Dimensional Phylogenetic Tree Visualisation

Tim Dwyer[†] **Falk Schreiber**[‡]

[†] School of Information Technologies
Madsen Building F09
University of Sydney
NSW 2006
Australia
Email: dwyer@it.usyd.edu.au

[‡] Bioinformatics Center
Institute of Plant Genetics and Crop Plant Research
06466 Gatersleben
Germany.
Email: schreibe@ipk-gatersleben.de

Abstract

Two and a half dimensional graph visualisation is the stacking of a set of related graphs, each drawn in a plane, into the third dimension to support visual comparison. A new aesthetic criterion is introduced for the $2\frac{1}{2}$D visualisation of sets of related phylogenetic trees. This aesthetic requires that the number of crossings between edges linking similar nodes in adjacent trees is minimised allowing a user to easily track the position of a particular species through its lifetime in the stack. We introduce an algorithm for efficiently minimising such crossings and discuss its complexity. Finally, we suggest a new "barrel" layout style which benefits from such crossing minimisation.

Keywords: Phylogenetic Trees, Crossing Minimisation, Three Dimensional Visualisation, Two and a Half Dimensional Visualisation, Bioinformatics

1 Introduction

Phylogenetic trees (also called *evolutionary trees*) show the ancestral relationships among a set of species. The leaves of these trees represent species, the internal nodes refer to (hypothetical) ancestors. An example of a conventional drawing of a phylogenetic tree is given in Figure 1*. Phylogenetic trees play an important role in life sciences and are used for many applications such as understanding evolutionary processes, designing new drugs and measuring genetic variations between species.

Biologists often have to study a set of related phylogenetic trees instead of a specific tree. Such a situation arises when:

- Given one data set (e.g. the protein sequences of a specific enzyme in different species) common methods and computer programs for reconstructing phylogenetic trees, such as maximum parsimony and maximum likelihood methods, often produce a set of related trees, not a single optimal one. Furthermore, the tree with the best score (the "optimal" tree) does not necessarily

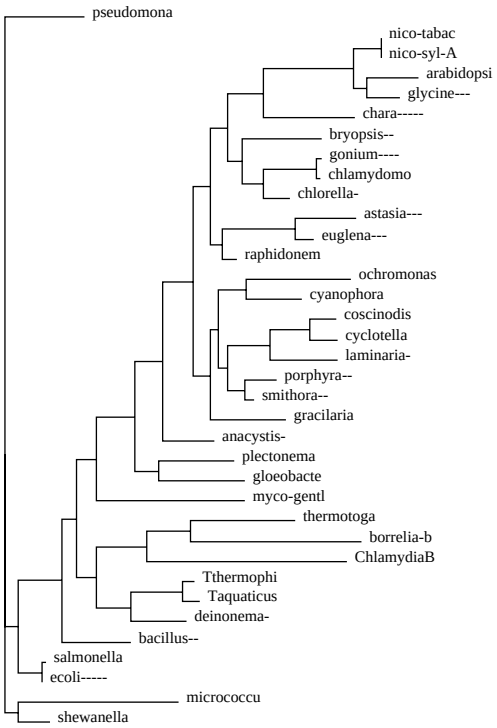


Figure 1: An example of a conventional drawing of a phylogenetic tree (a phenogram). The leaves indicate species; the internal nodes represent hypothetical ancestors for the species; and branch lengths give an indication of evolutionary time. The leaf ordering is entirely arbitrary.

*Data-set from (Delwiche, Kuhsel & Palmer 1995). Used by permission.

show the correct ancestral relationships among a set of species and therefore "nearly optimal" trees have to be considered.

- To improve the quality of the predictions often a set of different trees based on different morphological and molecular data sets is built. One example would be the set of trees built from sequences of different enzymes, each tree computed using the protein sequences of a specific enzyme in different species.

Therefore, to analyse the evolutionary relationships between species biologists often wish to compare a set of related phylogenetic trees in order to find the most likely one. That is, the tree which probably shows the ancestral relationship best considering all given trees. To help users perform this analysis we consider a method of visualisation for such a set of trees making similarities and differences between the trees easily recognisable.

Two and a half dimensional graph visualisation is the stacking of a set of related graphs into the third dimension to support visual comparison(Brandes, Dwyer & Schreiber 2003). In this paper we explore the use of $2\frac{1}{2}$D graph visualisation, also called stratified graph visualisation, for the comparison of sets of phylogenetic trees. The paper is structured as follows: Section 2 deals with common visualisation approaches for phylogenetic trees and introduces the optimal leaf ordering problem. In Section 3 we present an algorithm for the stratified tree ordering problem. Its complexity is analysed in Section 4. Finally, further refinements for the visualisation of sets of phylogenetic trees are discussed in Section 5.

2 Background

2.1 Visualising Sets of Phylogenetic Trees

As mentioned above, comparing sets of phylogenetic trees is an important part of many biologists work. As such, a number of tools for such tree comparisons have been suggested. Few, however, allow for the detailed inspection of a number (> 2) of trees at a time. Recent tools allow for browsing of trees as vertices in a similarity graph (Klingner & Amenta 2002). Detailed analysis of a smaller subset of the trees is restricted to a view of the consensus tree. Another recent tool (Munzner, Guimbretiére, Tasiran, Zhang & Zhou 2003) allows for comparison of two graphs side by side. The visualisation method described in this paper is differentiated by the ability to simultaneously display the entire structure of a number of similar trees.

2.2 Stratified Tree Visualisations

Phylogenetic trees are hierarchies representing the evolutionary relationships between species. The structure of such trees defines only parent–child relationships. No ordering of the leaves of the trees is specified. (Stewart, Hart, Berry, Olson, Wernert & Fischer 2001) introduced a $2\frac{1}{2}$D method for visualising the output of their phylogenetic inference program. As their program approaches an optimal solution it regularly outputs approximate trees which gradually converge to a best estimate. They found it useful to visualise these trees stacked in the order that they were produced by the algorithm. Their visualisation also allows the user to track the position of groups of leaves across multiple trees by joining the similar leaves with edges. However, doing so with the default leaf arrangements often leads to a large number of crossings between these edges when viewed from above (ie, perpendicular to the leaf edges), see Figure 2. Their visualisation tool allows for the user to manually swap nodes in the tree in order to better arrange these leaf nodes but doing so is arduous. In coming sections we explore an algorithm which performs this task automatically.

2.3 Optimal Leaf Ordering

We can improve the readability of these diagrams by arranging the leaves such that the number of crossings between edges connecting similar leaves in adjacent trees is minimised. Figures 3 and 4 show $2\frac{1}{2}$D visualisations of a set of phylogenetic trees with the default ordering and then with crossings removed.

We'll call the problem of minimising crossings in a $2\frac{1}{2}$D tree visualisation the *stratified tree ordering problem - STOP*. We define a *one-layer STOP* to be a STOP with only two layers (or strata) in which we keep the permutation of one tree fixed. This problem is somewhat similar to another leaf ordering that is often considered in relation to phylogenetic trees called *dendrogram seriation*. Recall that phylogenetic trees are usually generated from a hierarchical clustering of an underlying multidimensional dataset. Dendrogram seriation is the process of arranging the tree such that the order of the leaves minimises the distance between neighbouring leaves in the underlying data space. Solving this problem is akin to finding a solution to the travelling salesman problem subject to the tree constraints. (Morris, Asnake & Yen 2003) proposed a simulated annealing solution. (Bar-Joseph, Gifford & Jaakkola 2001) gave a dynamic programming method for finding the optimal solution which runs in $O(n^4)$ time using $O(n^2)$ space. In the next section we give an algorithm for solving a one-layer, binary tree STOP in $O(n^2 \log n)$ time and $O(n)$ space.

3 STOP Solver Algorithm

Crossings can be reduced in a one layer STOP by reordering the leaves of one tree while leaving the other fixed in a similar fashion to the one-layer crossing minimisation problem well known in graph drawing and computational geometry(Battista, Eades, Tamassia & Tollis 1999). A major difference, however, is that while the one-layer crossing minimisation problem has been shown to be NP-complete (Eades & Wormald 1994), we can find an optimal solution for a one-layer STOP in polynomial time using a divide and conquer strategy. One caveat, explained below, is that the unfixed tree must be strictly binary.

As in the crossing minimisation problem for general layered graphs, crossings can be reduced globally in an n-layer STOP with a layer by layer sweep approach. That is, we visit each pair of layers in the stack, solving each local one-layer STOP in turn. In early experiments only a few sweeps were required for the STOP to converge to a globally optimal, or near optimal, solution.

Before we explain the algorithm we show that the problem can be divided into a set of independent subproblems. In the sequel we define a one-layer STOP as a directed acyclic graph $G = (V, E)$ encompassing the unfixed tree and the leaves of the fixed tree. $L \in E$ is the set of edges connecting the leaves of the unfixed tree and the leaves of the fixed tree. It is the crossings in L that need to be minimised. We define a set of virtual edges between an internal node $v \in V$ and the neighbours of its descendent leaves as $F_v = \{(v, u_2) : (u_1, u_2) \in L \wedge \exists \text{ directed path}(v, u_2)\}$.

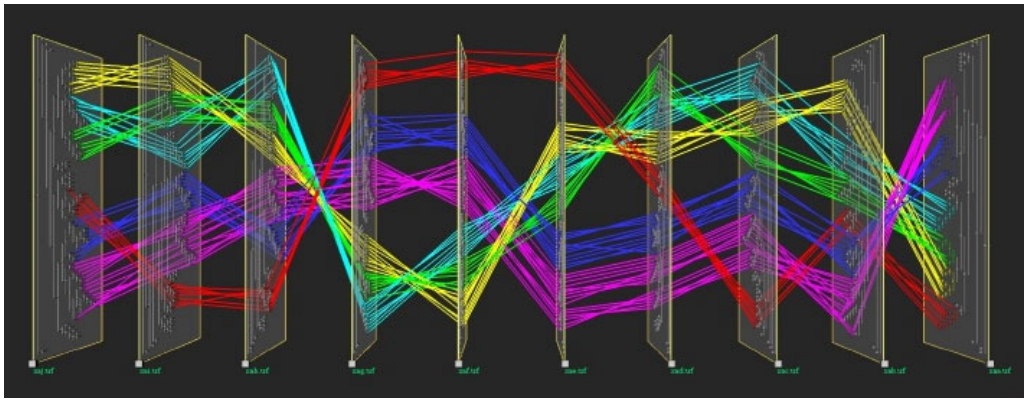
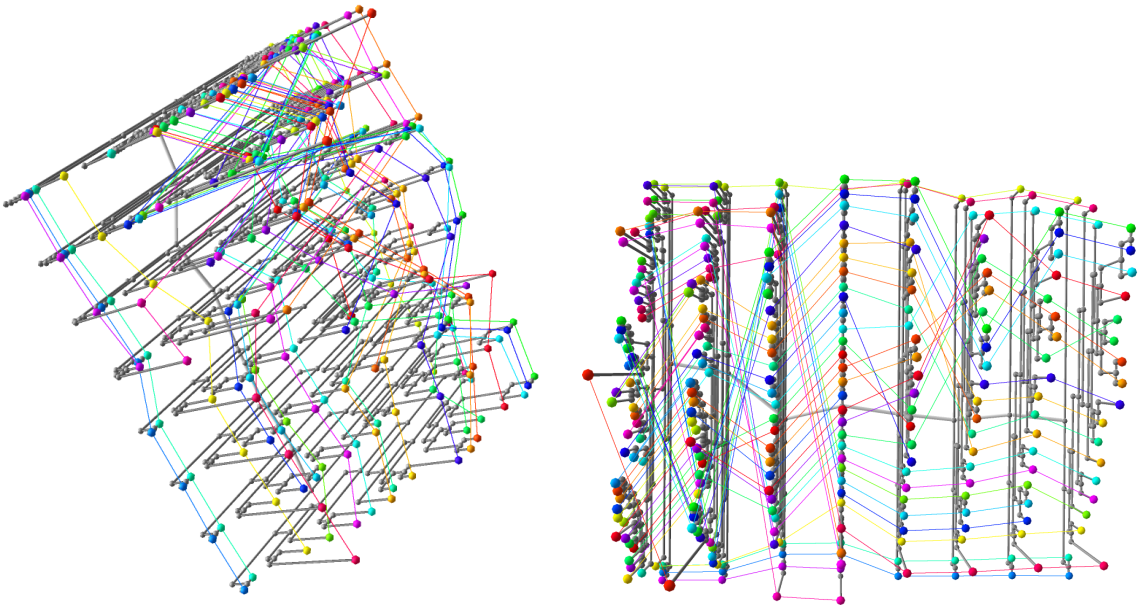


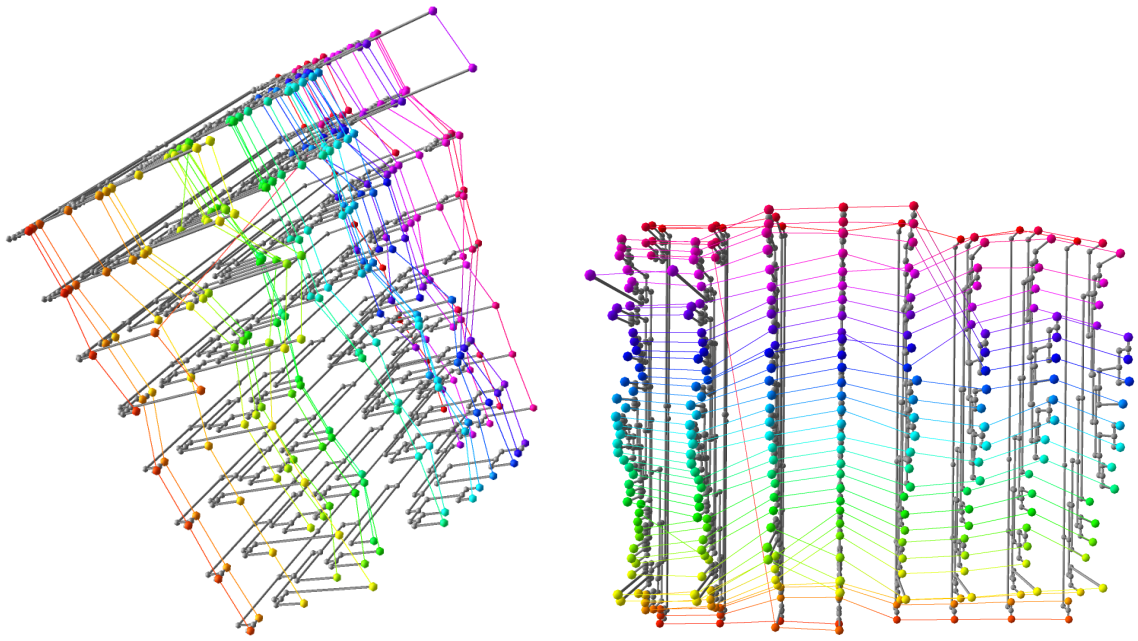
Figure 2: The $2\frac{1}{2}$D visualisation of a set of phylogenetic trees proposed by Stewart et. al. Note the large number of crossings between the coloured edges joining similar species in adjacent trees. Visualisation produced by TreeViewer: `http://www.avl.iu.edu/projects/Tree3D/`. Used by permission.



(a) Oblique view

(b) Top view

Figure 3: Two views of a $2\frac{1}{2}$D visualisation of a set of eight phylogenetic trees with unmodified leaf orderings. Note the large number of crossings between edges linking leaves on adjacent planes.



(a) Oblique view

(b) Top view

Figure 4: Another pair of views of the $2\frac{1}{2}$D visualisation from Figure 3 but with crossings minimised using Algorithm 1. Note that crossings have been greatly reduced.

Lemma 3.1 *In a particular permutation of the tree above v, if there are c crossings between edges in F_v and $E \setminus F_v$ then for all permutations of the subtree T of v: $cross(T) \geq c$.*

Proof When we collapse all the edges below v into the new edge set F_v we remove all the crossings between edges of leaves that are descendants of v. The only crossings remaining are those between F_v and the remaining edges in E. In a planar drawing of T the leaves below v must remain together, ie. a leaf below one of v's siblings cannot be placed between the leaves below v. Therefore, any permutations of the leaves below v cannot reduce crossings further and the statement above holds.

A visual demonstration is given in Figure 5(a). It shows two trees connected at the leaves by edges. Note that there are four crossings in the permutation shown. The permutation of the bottom tree is fixed and hence the full tree is not shown. In Figure 5(b) the same tree is shown again with subtrees below the second level collapsed into single nodes. It is clear that rearranging the nodes in the subtrees will not reduce crossings any further. In Figure 5(c) crossings have been minimised by rearranging the nodes on level 2.

Theorem 3.2 *A one-layer STOP can be recursively decomposed into a number of independent subproblems.*

It follows from Lemma 3.1 that an optimal ordering of a particular level in the tree remains optimal regardless of how we arrange the lower levels. Thus, an effective divide and conquer strategy is to process the tree bottom up eliding subtrees as described in Lemma 3.1. Such a recursive procedure is shown in Algorithm 1.

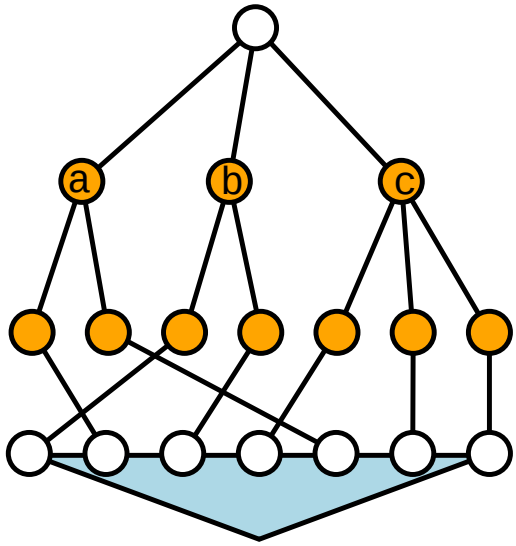
Note that for an unfixed tree of depth 1 the algorithm functions identically to the well known *adjacent-exchange* or *greedy-switch*(Battista et al.

Algorithm 1 Reduce crossings below internal node T of a tree

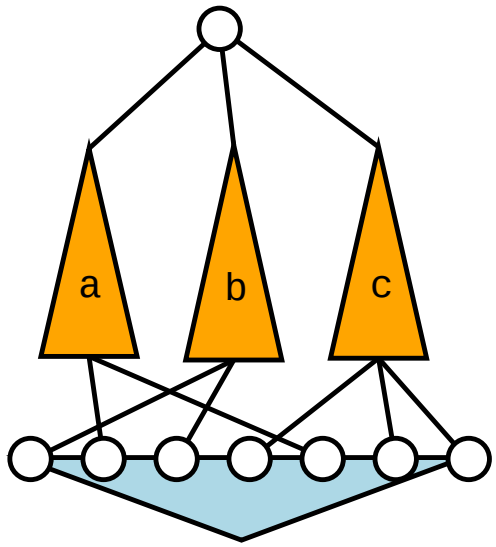
procedure $decross(T)$

Require: $T.children$ is an array indexed from 0 in which each element is also a tree node. If T is a leaf $|T.children| = 0$ and $|T.edges| = 1$ else $|T.children| > 0$ and $|T.edges| = 0$.

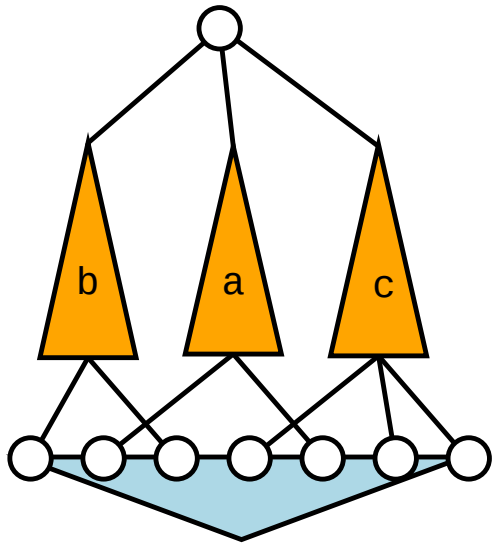
1: **for** $u \in T.children$ **do**
2: $T.edges \leftarrow T.edges \cup decross(u)$
3: **end for**
4: **for** $i \in \{x \in \mathbb{Z} : 0 < x < |T.children|\}$ **do**
5: $u \leftarrow T.children[i-1]$
6: $v \leftarrow T.children[i]$
$\{(u, v)$ are adjacent children$\}$
7: $c_{uv} \leftarrow countCrossings(u, v)$
8: $c_{vu} \leftarrow countCrossings(v, u)$
9: **if** $c_{vu} > c_{uv}$ **then**
10: $T.children[i-1] \leftarrow v$
11: $T.children[i] \leftarrow u$
12: return $T.edges$
13: **end if**
14: **end for**



(a) A tree with leaves linked to another tree



(b) After collapsing the subtrees below the nodes on level 2



(c) Rearranged to minimise crossings

Figure 5: Demonstration that one-layer crossing minimisation problem subject to tree constraints is decomposable

1999). For trees with internal nodes of high degree it is possible that an algorithm based on the *median heuristic* might converge more quickly. However since most of the input trees considered are binary or of degree no greater than 3 this is not an issue in our application. In fact, a result of Theorem 3.2 is that for binary trees the algorithm reaches an optimal solution for the one-layer problem in a single pass.

An obvious potential bottleneck in the algorithm is the crossing count. A naïve algorithm would check every edge attached to u against every edge of v leading to time $O(|E_u||E_v|)$. The following algorithm is a simplification of a two layer crossing counting algorithm by (Barth, Jünger & Mutzel 2002). Their algorithm has running time $O(|E| \log |V_{small}|)$ where E is the set of edges and V_{small} is the smaller of the two sets of vertices. In our case the $|V_{small}| = 2$ so the existence an $O(|E_u| + |E_v|)$ time algorithm is obvious.

Algorithm 2 Count edge crossings between two nodes

procedure *countCrossings*(u, v)
Require: u and v each have lists of edges E_u and E_v

```
1:  E ← E_u ∪ E_v
2:  sort E using radix sort {sort order is (π_2(tail), u < v)}
3:  crossCount ← vCount ← 0
4:  for e ∈ E do
5:      if e.head = v then
6:          vCount ← vCount + 1
7:      else
8:          crossCount ← crossCount + vCount
9:      end if
10: end for
11: return crossCount
```

4 Complexity

We consider first the complexity for a one-layer stop only. A single descent of a binary tree with n leaves takes exactly $n-1$ steps. The most expensive part of Algorithm 1 is the call to the *countCrossings* subroutine. Counting crossings with the algorithm given in Algorithm 2 takes $O(|E|)$ time for each internal node of the tree where E is the set of edges connected to leaves descendent from that node and $|E| = n$. The best case scenario is for a perfectly balanced binary tree. In this case the recursive cost function:

$$T(n) \leq 2T(\frac{n}{2}) + cn$$

is easily reducible to:

$$T(n) \leq dn \log n$$

by the well known general upper bound for such recursion(Weiss 2002).

So we have a lower bound of $\Omega(n \log n)$. The worst case is a tree of height n where the cost is:

$$\begin{aligned} T(n) &= \sum_{i=1}^{n} ci \\ &= c\frac{n(n+1)}{2} \end{aligned}$$

Giving an upper bound of $O(n^2)$. However, since the output of the clustering algorithms used in these applications is usually a fairly balanced tree, typical running times will be closer to the lower bound.

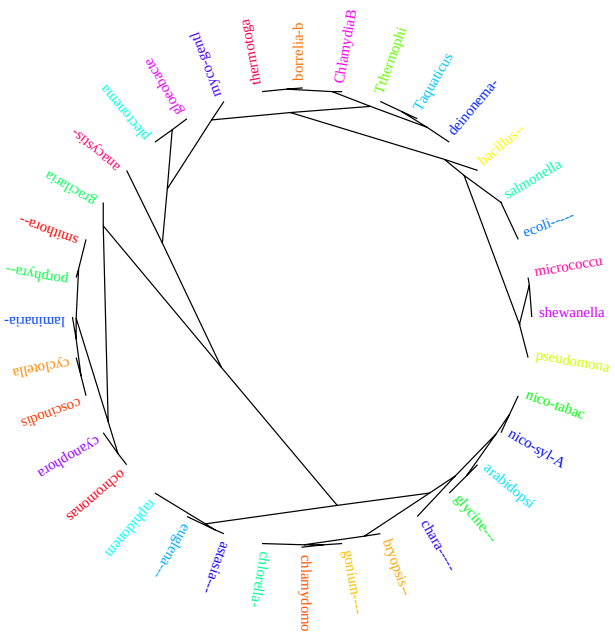


Figure 6: An example of a circular drawing of the phylogenetic tree from Figure 1.

As mentioned above, to minimise crossings for a set of n-trees we apply the one-layer STOP algorithm to each pair of adjacent layers in a layer-by-layer sweep strategy. We have found that this approach converges to a reasonable solution in a small number (< 10) passes. Further work might involve testing under what conditions this approach fails to reach a globally optimal solution. However, in practice we have found the approach at least adequate for test cases of the size shown in the figures. Larger cases might not be practical anyway due to the obvious bottleneck in wet-ware, ie, the ability of a human to understand an overly complex visualisation.

5 Further Refinements

5.1 Barrel Trees

On seeing the $2\frac{1}{2}$D visualisations shown in Figure 4 it was thought that arranging the trees in a configuration with the leaves spaced evenly around the perimeter of the circle might be a better utilisation of the available space. This was achieved using a weighted barycentre algorithm, arranging the leaves first and then recursively placing the parent of each pair of children (leaves or internal nodes) at the barycentre of those children weighted by branch lengths c_{bl}. That is, the position in the plane for each internal node T is given by:

$$pos(T) = \frac{1}{\|T.children\|} \sum_{c \in T.children} c_{bl}(c_x, c_y)$$

An example of this style of layout for a single tree is shown in Figure 6. Figure 7 gives the $2\frac{1}{2}$D "barrel" view of the set of trees.

5.2 Consensus Tree

A further refinement might be to pre-compute the consensus tree for the set of trees. Consensus trees play an important role in the tree comparison applications described by (Klingner & Amenta 2002) and (Munzner et al. 2003). There are different definitions for the consensus tree but generally it is a tree which somehow tries to cover all the trees in the set. It can be computed in linear time in a fairly straightforward way. Nodes in the consensus tree with less than three children are common to all trees in the set. Knowing this is useful in two ways:

- Drawing such nodes only on one plane, perhaps the topmost plane, might significantly simplify the visualisation. Other nodes can still be drawn on parallel planes, thus focusing the user's attention to the areas of change.
- Algorithm 1 need only be applied to subtrees which are dissimilar across the set. This should significantly speed up crossing minimisation for a set of trees which are largely similar.

5.3 Ordering of Strata

In the examples discussed in this paper we stack the trees according to an ordinal variable, such as time or the probability that the tree is correct. Another possibility for improved comparison or where no ordinal variable is available across the set of trees might be to find a stacking order for the trees that minimises differences between adjacent trees in the stack. Such an approach was used for comparing metabolic pathways in (Brandes et al. 2003). Note that the method used for calculating similarity measures between these pathways (general graphs with labelled vertices) does not extend to the trees with unlabelled internal nodes considered here. However, there are many well-known methods for measuring distances between arbitrary trees which would be appropriate, most famously: (Robinson & Foulds 1981).

6 Conclusion

We have described an algorithm for minimising the number of crossings in the edges linking similar leaves in adjacent planes of a $2\frac{1}{2}$D phylogenetic tree visualisation. We have also suggested a new layout for such $2\frac{1}{2}$D tree visualisations: the "Barrel" view.

An initial evaluation of the methods used involved conducting cognitive walkthroughs with a number of biologists. Their feedback has for the most part been very positive. They found the paradigm very intuitive and felt $2\frac{1}{2}$D stacking was a significant improvement over viewing a set of diagrams side by side. We hope to implement the additional improvements described above in the near future and then obtain further feedback from experts.

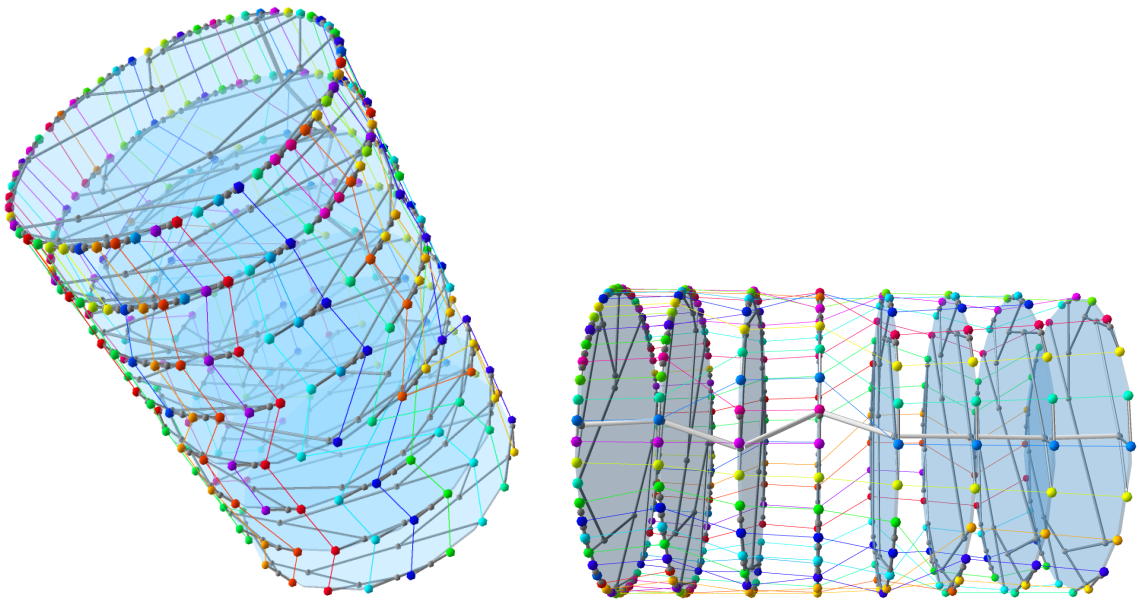
References

Bar-Joseph, Z., Gifford, D. & Jaakkola, T. (2001), 'Fast optimal leaf ordering for hierarchical clustering', *Bioinformatics* **17**, S22–S29.

Barth, W., Jünger, M. & Mutzel, P. (2002), Simple and efficient bilayer cross counting, *in* 'Proceedings of Graph Drawing 2002', Vol. 2528 of *Lecture Notes in Computer Science*, Springer, Berlin, pp. 130–141.

Battista, G. D., Eades, P., Tamassia, R. & Tollis, I. (1999), *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall, Englewood Cliffs, NJ.

Brandes, U., Dwyer, T. & Schreiber, F. (2003), Visualizing related metabolic pathways in two and a



(a) Oblique view

(b) Top view

Figure 7: A visualisation of the same data-set used in earlier figures but arranged in the circular style shown in cross-section in Figure 6.

half dimensions, *in* 'Proceedings of Graph Drawing 2003', Lecture Notes in Computer Science, To Appear - Springer, Berlin.

Delwiche, C., Kuhsel, M. & Palmer, J. (1995), 'Phylogenetic analysis of tufa sequences indicates a cyanobacterial origin of all plastids', *Molecular Phylogenetics and Evolution* **4**(2), 110–128.

Eades, P. & Wormald, N. (1994), 'Edge crossings in drawings of bipartite graphs', *Algorithmica* **10**, 379–403.

Klingner, J. & Amenta, N. (2002), Case study: Visualing sets of evolutionary trees, *in* 'Proceedings of IEEE Information Visualization, 2002', pp. 71–74.

Morris, S., Asnake, B. & Yen, G. (2003), 'Optimal dendrogram seriation using simulated annealing', *Information Visualization* **2**(2), 95–104.

Munzner, T., Guimbretiére, F., Tasiran, S., Zhang, L. & Zhou, Y. (2003), Treejuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility, *in* 'Proceedings of SIGGRAPH'.

Robinson, D. & Foulds, L. (1981), 'Comparison of phylogenetic trees', *Mathematical Biosciences* **53**, 131–147.

Stewart, C., Hart, D., Berry, D., Olson, G., Wernert, E. & Fischer, W. (2001), Parallel implementation and performance of fastDNAml — a program for maximum likelihood phylogenetic inference, *in* 'Proceedings of SC2001', ACM.

Weiss, M. A. (2002), *Data Structures and Problem Solving Using Java*, Addison Wesley.