TreeJuxtaposer: Scalable Tree Comparison Using Focus+Context with Guaranteed Visibility

Article *in* ACM Transactions on Graphics · July 2003

DOI: 10.1145/1201775.882291 · Source: DBLP

5 authors, including:

Tamara Munzner
University of British Columbia - Vancouver
132 PUBLICATIONS **7,579** CITATIONS

SEE PROFILE

Francois Guimbretiere
Cornell University
91 PUBLICATIONS **4,834** CITATIONS

SEE PROFILE

Serdar Tasiran
Koc University
101 PUBLICATIONS **2,715** CITATIONS

SEE PROFILE

Yunhong Zhou
49 PUBLICATIONS **3,202** CITATIONS

SEE PROFILE

TreeJuxtaposer: Scalable Tree Comparison using Focus+Context with Guaranteed Visibility

Tamara Munzner *†‡ François Guimbretière§ Serdar Tasiran‡¶ Li Zhang‡ Yunhong Zhou‡

Abstract

Structural comparison of large trees is a difficult task that is only partially supported by current visualization techniques, which are mainly designed for browsing. We present TreeJuxtaposer, a system designed to support the comparison task for large trees of several hundred thousand nodes. We introduce the idea of "guaranteed visibility", where highlighted areas are treated as landmarks that must remain visually apparent at all times. We propose a new methodology for detailed structural comparison between two trees and provide a new nearly-linear algorithm for computing the best corresponding node from one tree to another. In addition, we present a new rectilinear Focus+Context technique for navigation that is well suited to the dynamic linking of side-by-side views while guaranteeing landmark visibility and constant frame rates. These three contributions result in a system delivering a fluid exploration experience that scales both in the size of the dataset and the number of pixels in the display. We have based the design decisions for our system on the needs of a target audience of biologists who must understand the structural details of many phylogenetic, or evolutionary, trees. Our tool is also useful in many other application domains where tree comparison is needed, ranging from network management to call graph optimization to genealogy.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types

Keywords: information visualization, Focus+Context, realtime rendering, tree drawing, phylogenetic tree

1 Introduction

Biologists have been working towards discovering the evolutionary tree of the history of species since the time of Darwin. Until a decade ago, the main information available to guide them was key morphological features found by painstaking observation of living organisms and the fossil record. The scope of these morphological analyses was usually limited to a few dozen species, but the recent explosion of molecular data from DNA and protein sequencing has allowed biologists to tackle increasingly large sets of species. Today some groups of systematic biologists, or systematists, have reconstructed trees of thousands of nodes, and many in the field hope to infer a complete Tree of Life, estimated to contain over ten million nodes, within the next ten years [1].

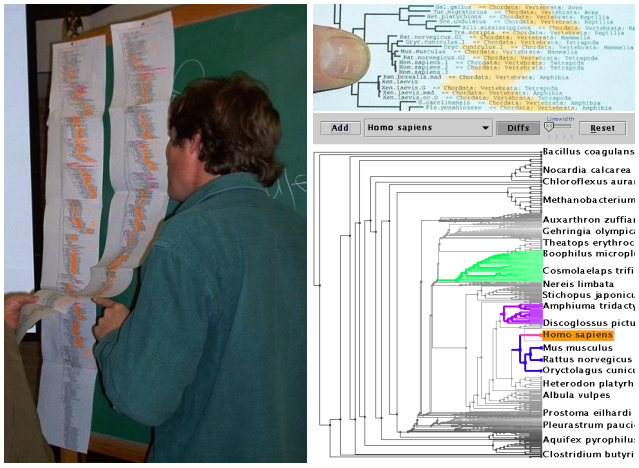

Figure 1: Left, Right Top: Biologists faced with inadequate tools for comparing large trees have fallen back on paper, tape, and highlighter pens. Right Bottom: TreeJuxtaposer is a scalable tool for interactive exploration and comparison of trees.

Although one might think that determining the Tree of Life describing all species is mostly done except for a few minor exceptions, the opposite is true. There is still controversy on how to categorize well-studied groups like vertebrates and even primates, and the situation with bacteria and viruses is even less clear. These trees are of interest not only to evolutionary biologists, but also in such domains as pharmaceutical drug design: determining the possible therapeutic function of an unknown plant sample may require a large number of expensive lab tests. By deciding where the plant fits into the tree of already-recognized evolutionary relationships, one can perform a much more targeted set of tests.

Biologists have many tools for creating trees through automatic reconstruction [Swofford 1998]. A single run of a reconstruction package may generate dozens or hundreds of trees, many such runs with different parameters could occur daily, and systematists may grapple with a particular dataset for years. One common method for comparing trees is to treat each tree as a point in a "treespace" and define a metric in the treespace to determine how different two trees are. For example, TreeSet [Amenta and Klingner 2002] is such a tool that helps biologists winnow and compare large sets of trees at a high level. However, a single number that summarizes the difference between two trees is too coarse for biologists who need to understand the structural difference between two trees. No

*Email: tmm@cs.ubc.ca, francois@cs.umd.edu, stasiran@ku.edu.tr, l.zhang@hp.com, yunhong.zhou@hp.com
†University of British Columbia
‡Hewlett Packard Systems Research Center
§University of Maryland
¶Now at Koç University in Turkey.

[1] http://research.amnh.org/biodiversity/center/features/tol.html

tools for comparing trees in full structural detail exist now, but they would be extremely helpful for biologists who must determine the true tree amidst these many possibilities.

In this paper we will use several biological terms, which we now define. A **phylogenetic tree**, also known as a **phylogeny**, is an encoding of hypothesized evolutionary relationships where the leaves of the tree represent currently extant species or genes. An interior tree node represents an inferred ancestral species from which all the nodes in the subtree underneath it are descended, and these subtrees are called **clades**. Ideally, phylogenetic trees are binary, because an interior node represents the bifurcation point where one species split into two distinct new species. However, higher degree nodes are often seen in a phylogenetic tree when the order of bifurcation cannot be determined because of missing data or uncertainty. Typically leaves are labelled with species names but the inferred interior nodes often have no labels.

A **consensus tree** is made by combining multiple trees into a single tree that represents the clades that are common to all the input trees. In the case when the input trees are binary, the consensus tree may have nonbinary interior nodes, indicating a structural difference. Although consensus trees can help systematists detect areas of difference, characterization of the differences is not easy to accomplish through pure visual inspection of a single consensus tree because structural information has been lost: the same consensus tree describes many possible situations, and distinguishing between these situations is important for the biologists.

A related fundamental operation underlying many biological investigations between trees is determining whether a clade is **monophyletic** across trees. This problem can be stated as discovering whether the nodes in a subtree (ancestor plus all descendants) in one tree form a clade in another tree, or have become a forest.

1.1 Requirements

We have identified the following task requirements from discussions with our target audience of systematists.

Automatic identification of structural differences. Humans do not perform well at detection tasks, so we need computational support for locating the areas of difference between two trees in order to visually mark them. We also need efficient structural difference computation algorithms in order to extend brushing [Becker and Cleveland 1987] to tree comparison.

Differences characterization. The biologists do not simply need to discover *whether* two trees are different: they must understand *how* topological structures differ in detail. For this task, biologists often need to inspect small tree areas at the same time, if possible in the context of the full tree.

Scalability in tree and display size. The stated goal of the Tree of Life project is to create trees of millions of nodes within the next several years, and we would like to support navigation and comparison at that scale. Many biologists still study trees of only a few dozen nodes, and comparing even these small trees is not easy through unsupported visual inspection alone. Several groups are already working with thousands of nodes, and their existing tools for biological tree drawing do not adequately support trees of this size. As a result, the systematists are often forced to fall back on scotchtaping dozens of pieces of paper together and manually annotating this single large tree, as in Figure 1. We would also like to take advantages of new display technologies such as the 9 megapixel IBM T221 display. While most systematists do not yet have such displays, having more pixels with which to explore huge data sets is an important aid to scalability, so our system should work well on them.

We thus must design algorithms that depend on the minimum of the total number of visible nodes (tied to the number of pixels in the display), and the total number of nodes in the tree.

Finally we observed that systematic biologists use a wide variety of operating systems, including Macs, Unix/Linux, and Windows. We chose to build our system using Java and OpenGL using the GL4Java [2] bindings. By using Java, we gain multi-platform support in return for a minor sacrifice of efficiency.

1.2 Contributions

Our TreeJuxtaposer system is a new tool for comparing and browsing large trees. TreeJuxtaposer relies on three basic techniques:

Structural comparison. The structural comparison is done by associating each node in one tree to its most "similar" node, the *best corresponding node*, in the other tree. We propose a similarity measure between nodes and design efficient algorithms for computing the similarity measure between any two nodes and for computing the best corresponding nodes. These algorithms allow us to highlight areas of difference automatically and support structural brushing: interactive highlighting of the areas in the other trees that correspond to what is under the mouse in the active tree window. Our algorithms typically run in almost linear pre-processing time and provide lookup in almost constant time during interactive exploration.

Guaranteed visibility. We identify the concept of guaranteed visibility, the property that marked areas are always visible no matter what navigation has been performed by the user. It would defeat our purpose if our algorithms automatically detected all such areas but the user missed an area because it was out of the frustum or it subtended less than a single pixel of screen area.

AccordionTree navigation. We present a technique for tree navigation that is based on global rectangular Focus+Context distortion, and is well adapted to phylogenetic trees. Our algorithm incorporates a novel extension to the quadtree data structure and supports efficient distortion-based navigation and guaranteed visibility of rectilinear areas. Our technique provides good information density both for non-binary and binary trees, as needed by biologists, unlike several previously proposed scalable tree layout methods such as [Lamping et al. 1995; Munzner 1998]. Our progressive rendering algorithm is similar to these systems, providing a guaranteed frame rate for trees of up to 500,000 nodes on a range of display sizes, from a laptop to a high resolution 9 megapixel display.

2 Previous Work

Most phylogeny tree viewers only handle small trees; an exception is the TreeWiz [Rost and Bornberg-Bauer 2002] system that scales to 75,000 nodes. It does not provide any explicit features to ease the comparison of large trees, and navigaton is awkward because each viewpoint change spawns a new window. In our current implementation we can compare four trees of 75,000 nodes each at interactive frame rates. MacClade [Maddison and Maddison 1992] is perhaps the most sophisticated of the many tools for interactive manipulation of phylogenetic trees, but it is not designed for scalability.

Focus+Context is a popular information visualization approach of showing an area of distorted aggregate context around an easily changeable focus point to allow a large overview integrated with details in limited screen real estate [Robertson et al. 1991; Munzner 1998]. Early examples of global Focus+Context systems, where changing the focus point affects the entire visible area, include Document Lens [Robertson and Mackinlay 1993] and Continuous

[2] http://www.jausoft.com/gl4java.html

Zoom [Bartram et al. 1995]. One of our contributions is efficient algorithms to allow this technique to be used for visualizing large phylogenetic trees.

Tree visualization is a highly active area of research well described in a recent survey [Herman et al. 2000], and we limit our discussion here to systems that focus on Focus+Context exploration of the tree topology. The recent DOI Tree [Card and Nation 2002] and SpaceTree [Plaisant et al. 2002] systems make heavy use of aggregation by automatically determining when to collapse a subtree and display it as a glyph. Our opposite approach is to present as much information as possible given the pixel count of the display. Section 5.3 discusses the justification and contributions of our choice of maximum visible detail compared to visual aggregation.

Few systems have been explicitly designed for tree comparison, despite its importance in many domains. The TimeTube and Visualization Spreadsheet of Chi et al [Chi et al. 1998; Chi and Card 1999] share some goals with TreeJuxtaposer, but focus on showing addition and deletion of nodes via color coding on the same combined layout of all trees of interest. Our work shows the structural differences more effectively, as their approach is similar to the consensus tree construction discussed in Section 1. Graham presents a system for global focus+context manipulation of multiple linked tree views [Graham and Kennedy 2001]. However, it scales to less than 10,000 nodes and only links the perfect matches at the leaves, while we solve the more general problem of finding best corresponding nodes in the interior.

Brushing and linked highlighting [Becker and Cleveland 1987; Ward and Martin 1995] are often used with smaller data sets where the all the data points are assumed to be visible on the screen. We are exploring how the same advantages can be delivered for large data structure for which screen limitations might not guarantee that all elements are visible on the screen. Although we are the first to identify guaranteed visibility as such, the discussions of *information residue* [Furnas 1997] and *desert fog* [Jul and Furnas 1998] have influenced our thinking.

3 Algorithms

We next describe and discuss new algorithms for computing structural differences, drawing a global focus+context layout, and guaranteeing visibility.

3.1 Structural Comparison

Structural comparison algorithms underlie three central aspects of our system: automatic structural difference marking, structural brushing, and guaranteed visibility. We present a new method for computing the best corresponding node between two trees in near-linear average time during preprocessing. We also explain the construction that allows us to quickly compute how "similar" two nodes are by using a known rectangular range searching data structure. Such computation is necessary for the guaranteed visibility computations described in Section 3.3.

Both algorithms depend on a definition of similarity. Our strategy is to first associate each node in one tree with the most "similar" node in the other tree according to a carefully chosen similarity measure. Then, we can visualize the structural difference between two trees by highlighting those nodes that do not have a very good match. Our definitions of similarity and BCNs are explicitly designed to support visual highlighting that pinpoints the nonmonophyletic clades of interest to biologists. The association between nodes also allows us to implement linked highlighting conveniently.

Associating leaf nodes is straightforward as each leaf node is labeled by a name, and we can associate two leaves if they have the same name. However, the association between internal nodes

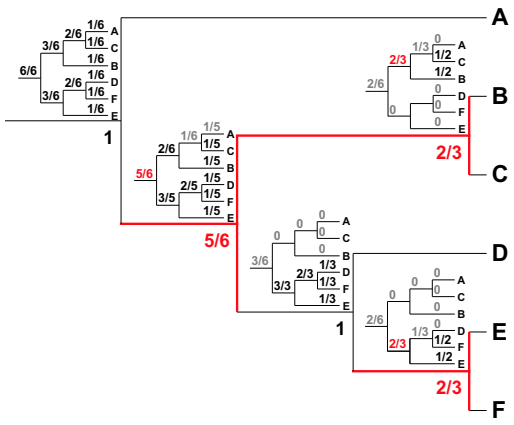

Figure 2: We show the calculation of best corresponding node scores and the highlighting of structural difference based on them. At each interior node of the large tree, we show the similarity score between that node and every node in the other tree. We only compute the scores in the simplified spanning tree, emphasized in black, and use the highest one as the BCN. Structural differences are shown by marking the nodes with BCN score < 1 in red.

is less clear. Our similarity definition can be regarded as an extension of previous work on consensus trees [Adams 1972; Sokal and Rohlf 1981; Margush and McMorris 1981] and the RF metric for trees [Robinson and Foulds 1981], both used extensively in the biology community. If for each node $v \in T$, let $L(v)$ denote the set of the labels of the leaves in the subtree rooted at v, then using the RF metric a node $v_1 \in T_1$ is mapped to a node $v_2 \in T_2$ if $L(v_1) = L(v_2)$. Because the previous work only considers perfect matching, it may cause two intuitively similar trees to have very low similarity score. We extend this definition by using a similarity that is used to measure the difference between two sets — the similarity $S(A, B)$ between two sets A, B is defined to be $\frac{|A \cap B|}{|A \cup B|}$. One nice property of this measure is that the function defined by $d(A, B) = 1 - S(A, B)$ is a metric, meaning that both $A = B$ if and only if $d(A, B) = 0$ ($S(A, B) = 1$), and $d(A, C) \leq d(A, B) + d(B, C)$. This measure has also been used for detecting similar documents in the Stanford SCAM project [Shivakumar and García-Molina 1995] and in the AltaVista search engine [Broder et al. 1997; Broder 1998].

Using this measure we are now able to compare two internal nodes according to the sets they represent or the sets of leaves underneath them. For two nodes $v_1 \in T_1$ and $v_2 \in T_2$, we define the similarity $S(v_1, v_2)$ between them as $S(L(v_1), L(v_2))$. For $v_1 \in T_1$, the **best corresponding node** $\mathrm{BCN}(v) \in T_2$ is defined to be the node that maximizes the similarity score, i.e. $\mathrm{BCN}(v) = \arg\max_{u \in T_2} S(u, v)$, with the tie broken arbitrarily, as shown in Figure 2. This extension is appealing because it reflects the fact that in phylogenetic trees an internal node represents the evolutionary event leading to the creation of the species underneath the node. BCN is not a one-to-one mapping, and therefore not symmetric.

In what follows, we explain how to efficiently compute the similarity score between any two nodes and the BCN for each node, after almost linear time preprocessing.

Similarity score query. For any pair $v_1 \in T_1, v_2 \in T_2$, we would like to be able to quickly compute $S(v_1, v_2)$. In a naive implementation, one may pre-compute all the pairwise similarity scores and store them, requiring quadratic space and compu-

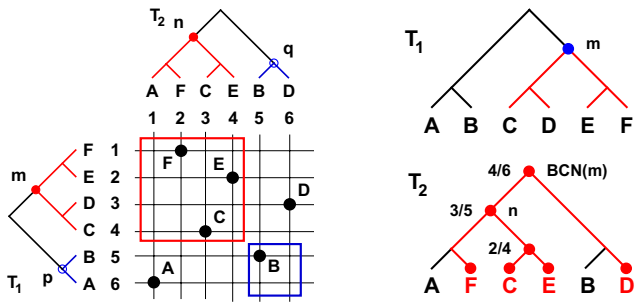

Figure 3: **Left:** When comparing two trees, computing $S(v_1, v_2)$ is equivalent to computing the number of points inside a rectangle, where each label is mapped to a point in the plane. The intersection of the leaf sets for subtrees rooted at m and n is the upper left square (C,E,F), and is the lower right square (B) for p and q. **Right:** To compute the best corresponding node of m we compute the similarity for each node in T_2 and pick the node with the greatest value. The BCN computation can be accelerated by computing only those values at the nodes of the simplified spanning tree, shown with red circles. For example, at node n the set of leaves below n and m is (C,E,F) while the set below n or m is (A,C,D,E,F), so $S(m, n) = 3/5$. The root of T_2 has the highest score and is the BCN of m.

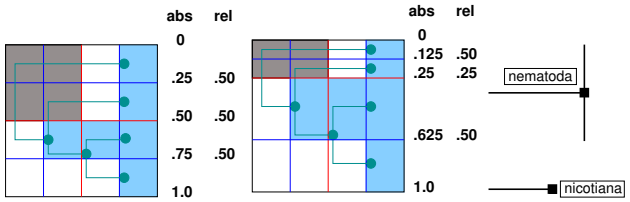

Figure 4: **Left:** Sparse quadtree construction. A bottom-level quadtree cell, shown here in light blue, is created for each node in the tree. Higher level cells are only created if at least one child exists, so the grey cell in the upper left corner is not created in the quadtree. The spatial extent of a quadtree cell is determined by the relative *split* value of the line dividing its parent in two: the red line divides the top-level cell, and the blue lines divide lower levels. In the undistorted state, these split values are all .5. **Middle:** In this simple zoom, the absolute position of every cell boundary and tree edge has changed, but only the split value of the red line differs. **Right:** Node geometry. **Top:** Interior node. **Bottom:** Leaf node.

tation time. Here, we present a connection between this problem and the classical rectangular range searching problem. In preprocessing, we first traverse the tree and compute the size of $L(v)$ for each node v in the tree. This reduces the problem of computing $S(v_1, v_2)$ to computing $|L(v_1) \cap L(v_2)|$ as $|L(v_1) \cup L(v_2)| = |L(v_1)| + |L(v_2)| - |L(v_1) \cap L(v_2)|$. We then map each leaf to a point in the plane by the following procedure: we assign a number $x_i(v)$ to each leaf node v according to its order in the post-order traversal of the tree T_i for $i = 1, 2$. For a label ℓ, suppose that $v_1 \in T_1$ and $v_2 \in T_2$ are the nodes with the label ℓ. We map ℓ to the point with coordinates $(x_1(v_1), x_2(v_2))$. In a post-order traversal the nodes in a subtree are in consecutive order, so an intersection query reduces to computing the number of points inside a query rectangle, as shown in Figure 3 Left. This problem is well-known in computational geometry, and efficient solutions are available [Preparata and Shamos 1990]. We implement one algorithm that requires $O(n \log n)$ pre-processing time and $O(n)$ space and can answer any query in $O(\log^2 n)$ time.

Best corresponding nodes. In previous work [Day 1985], all the perfectly matching pairs, namely pairs of nodes with similarity score 1, can be computed in the optimal linear time. Computing the best corresponding nodes is more difficult. Even if we use the data structure we build for computing pairwise similarity scores, it would require quadratic time. We can do better by realizing that the BCN can only appear at certain nodes in the tree.

For a tree T and any subset of leaves L, we define the **simplified spanning tree** $T(L)$ of L to be the subtree that is formed by first computing the spanning tree of the leaves in L and then replacing each path of degree-two nodes by a single edge, to compress long chains of nodes with only a single child. We observe that the BCN of $v_1 \in T_1$ must be a node in the simplified spanning tree $T_2(L(v_1))$, as shown in Figure 3 Right. By this observation, it is shown in [Zhang 2003] that we can compute the best corresponding nodes in an incremental fashion in a total of about $O(n^{1.5})$ time. That algorithm is however too complicated for implementation. In practice, we use the fact that the BCN of a node $v_1 \in T_1$ can be computed in time $O(|L(v_1)| \log n)$ after linear time pre-processing: we pre-compute a data structure to answer least common ancestor queries efficiently [Harel and Tarjan 1984], construct the simplified spanning tree $T_2(L(v_1))$ explicitly, and then traverse $T_2(L(v_1))$ to compute the BCN of v_1. We then simply run this method for each node in T_1. Although in the worst case this method would require quadratic running time, for balanced trees the total running time is almost linear ($O(n \log^2 n)$). Fortunately, trees in phylogeny and many other domains tend to be fairly balanced. Therefore, the simple method works very efficiently in practice.

3.2 Spatialization and Drawing

Phylogenetic trees are most commonly laid out using one of three methods: rectangular, slanted, or circular. After discussions with our target population we chose the rectangular layout as a good compromise between compactness and readability (many biologists find circular layouts difficult to read). The layout is done in a standard way: we first place all the leaves in a vertical column, equally spaced as shown in Figure 4, and then recursively calculate the coordinates of the interior nodes from the bottom up.

We chose a Focus+Context navigation method of expanding or contracting rectangular areas, as if the tree had been laid out on a stretchable rubber sheet [Sarkar et al. 1993]. The border is always anchored to the frame, so the effects of distortion are global rather than local: growing some areas necessarily implies shrinking others, and vice versa. The global distortion approach allows us to strictly guarantee visibility of marked areas, as we will discuss in Section 3.3. We constrain the deformation so that reshaping a rectangle propogates along both the row and column that contains it. Figures 4 and 5 show that modifying a node also affects the subtree beneath it and all of its siblings at the same level in the tree. Basing everything on rectangular distortions simplifies the conceptual model of interaction and exploits the naturally hierarchical structure of subtrees. The AccordionDrawer technique is named after the visual effect that compressing some strips and expanding others is reminiscent of the bellows of an accordion, but one can of course deform in both dimensions, not just one.

Focus+Context quadtree. Drawing a geometric representation of the tree imposes a mapping between the abstract elements of the tree and the spatial extents in which we lay out and draw their geometric representations. Every node is uniquely associated with the edge between it and its parent. For simplicity we consider this node-edge pair as a single primitive, whose geometric representation is different for interior and leaf nodes, as shown in Figure 4 Right. We need a spatially recursive data structure to hierarchically

store and access these geometric objects. A quadtree is the appropriate choice, but we must extend it to support our requirements of distortion-based navigation and guaranteed visibility.

Quadtree construction is described in Figure 4. The height of the bottom quadtree grid is the number of leaves in the tree, and its width is the tree depth. We build a sparse grid, with a cell at the lowest level created only when it contains a tree node. At higher levels, a cell is only created if at least one child cell already exists. We build increasingly coarse grid levels in the quadtree, with the top-level grid containing only a single cell. The depth of the quadtree is $O(\log n)$, where n is the number of leaves in the tree.

We attach a node-edge pair to the smallest quadtree cell that completely encloses it. These attachments are permanent: geometric elements do not move from cell to cell. Navigation changes the absolute positions of quadtree cell boundaries, but the edges they contain keep the same relative offsets to the cell borders. Using the analogy above, both the geometry and the quadtree cell boundaries are on the rubber sheet, and the boundary lines are the handles with which to stretch or shrink the sheet. Therefore, our quadtree structure has a fixed cell-element attachment relationship but the geometry of cells in the quadtree are flexible as they may change due to the user interaction. This is different than typical quadtree use in which the geometry of quadtree is fixed but the cell-element attachment changes.

We store the cell boundaries hierarchically, as a *split position* with a value between 0 and 1 that determines the allocation of space between a cell's children relative to its own borders. The grid is uniform when every split is set to 0.5, and Figure 4 shows how a simple global deformation can be the result of changing a single split value. In more extreme cases a child boundary might need to move outside of the current absolute position of its parent cell boundaries, and we must also change the parent's own split in order to maintain the hierarchical relationship between parent and child cell borders. The worst case of a ripple effect of deformations all the way up to the root quadtree cell is bounded by the depth of the grid, and is thus $O(\log n)$ work to update cell boundaries. Lookup of absolute position has the same complexity, with a a similar hierarchical quadtree traversal from the top down. We store relative split positions instead of absolute coordinates because it makes the cost of each update small while it is still efficient to compute the coordinates of each cell on the fly. For efficiency, we also cache the coordinates so that they do not have to be recalculated within a single frame, when no navigational changes can occur.

Progressive rendering. Just as we design for the situation where the visible elements in the scene are a small fraction of the total nodes in the tree, we also support realtime interaction when the number of elements that can be drawn in a single frame is only a small fraction of the visible elements. Our drawing algorithm is similar in spirit to that of H3Viewer [Munzner 1998; Munzner 2000], where the scene is divided into small pieces that require roughly constant amounts of time to render, and the work units are ordered according to their visual importance. We have a fixed time quota per frame to ensure rapid interactive system response. We use the first frame's time to draw the most important items in the back buffer, then progressively add detail to the scene in subsequent front buffer frames if the user is not actively moving the structure. In this case, the quantum of work is a quadtree cell, or more specifically the tree node-edge pairs attached to that cell. The key problem is choosing the correct drawing order, because a poor choice will lead to distracting flickering for small trees and a complete breakdown of realtime interaction for big trees. The most obvious solution of simply starting at the tree root fails dramatically with distortion-based navigation.

One possible criterion for visual importance is to order the queue by the current screen area of the cells. However, this ordering ignores the topology of the tree, and users found it distracting to see disconnected subtrees appear while the rendering progressed. Also, when interacting with direct manipulation, the visual importance is more dependent on proximity to the current locus of attention than screen size alone. We thus begin with the site of the most recent user interaction, presumably the focus of the user's attention, and work outwards from there in order to prioritize drawing visibly connected components. We enqueue cells based on the tree topology: after we draw an edge attached to the cell, we enqueue the quadtree cells that contain the tree edge's parent and children. The accompanying video shows the efficacy of our approach.

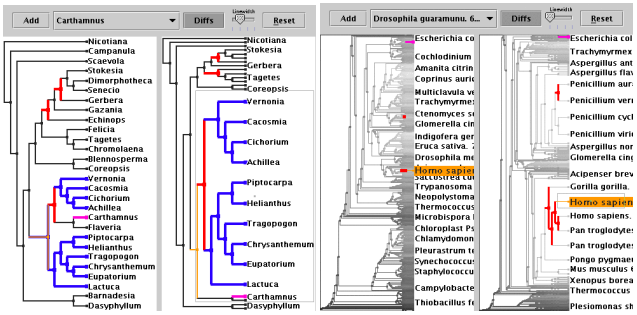

Figure 5: Tree comparison between two variants of a single phylogenetic reconstruction run, with the exact location of structural differences (in biological terms, nonmonophyletic clades) marked in red. The left tree is in the undistorted overview position, while parts of the right side have been expanded. **Left:** A small 55-node tree. **Right:** A larger 1600 node tree.

3.3 Guaranteed Visibility

In Section 3.1 we describe how to compute the differences between two trees. Here we discuss the problem of ensuring that these differences are always made visible to users. We use the term **guaranteed visibility** (GV) for the property that some mark deemed important is always visible onscreen. (See Section 4 for a discussion of the mark types considered important in our system.) In a graphics system there are three main reasons that a highlighted object would not be seen on the screen: culling for being outside the viewing frustum (frustum culling), culling because its projection on the screen is smaller than a pixel (LOD culling), or occlusion by another object. While these situations are not an extreme concern when browsing, they are major breakdowns in a system designed for identifying differences because they force users to carry out an exhaustive search of the entire navigable area. Humans perform poorly in such a detection task, and moreover a search would force the users to abandon a carefully chosen current point of view.

Culling. In the vast majority of graphics and visualization systems, the viewport is smaller than the area of interest, introducing the possibility of frustum culling. In our approach we obviate that problem by relying on a global Focus+Context technique, where the full dataset is presented on the screen at all times.

Even if we guarantee that each dataset element has a projection inside the viewport, an element could be too small to be seen, introducing LOD culling. That is, it could subtend less than a single screen pixel after applying the transformation from world to screen space. To guarantee visibility of marked areas, before culling a node because its projected size on the screen is too small, we must efficiently query whether the spatial extent of the subpixel cell encloses any highlighted objects.

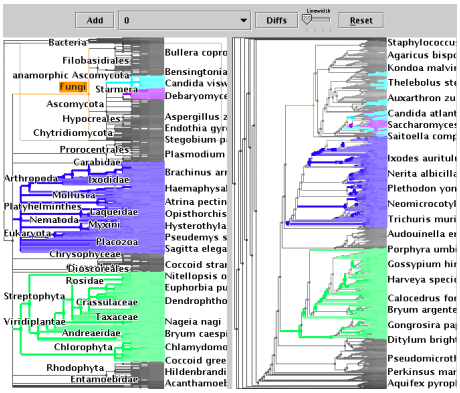

Figure 6: We compare 7K-node nonbinary taxonomy (left) with a 6K-node binary phylogeny (right) by marking clades/subtrees. We mark four clades, and the top two are not monophyletic: they become forests on the other side.

In Section 3.1 we described the algorithm for computing the similarity score between two nodes efficiently. That algorithm is also useful for providing GV efficiently. In GV drawing, the decision of highlighting a cell depends on whether the cell contains nodes that belong to a range selected in the other tree. When we build the hierarchy of quadtree cells at startup, we perform a post-order traversal of the quadtree to store the minimum and maximum indices of objects attached to a cell's descendants. We then associate the interval between the minimum and maximum indices to each cell. This association is conservative as a cell may contain nodes from different parts of the tree and that are not necessarily contiguous.

In order to use the range-checking data structure, we must continue to use object indices based on a post-order traversal of the tree topology rather than spatial extent. The range converges as we descend the quadtree hierarchy, and is necessarily exact for the lowest level cells that contain only a single node.

We do the range check only for subpixel cells when we consider whether to halt spatial recursion. If there are no marked areas, we simply stop. If we have already highlighted this cell because of an attached marked object, we can likewise halt. Otherwise, we continue the descent until we have resolved the mark location, thus avoiding potential false positive marks from inexact ranges. The approximation means that some unnecessary work will be performed, but we can cache the results until the marks are next changed (as described in Section 4). The important property of our algorithm is that no necessary work will ever be missed, and Figure 7 shows that the overhead is acceptable.

Occlusion and labels. The only source of occlusion in our system is text labels for interior edges, because we are using a 2D representation. Although many phylogenetic trees have labels only at the leaf edges, as in Figure 5, we support labels at internal edges for full generality as shown in Figures 6 and 8. We calculate the potential position of a label relative to the current position of its edge, as shown in Figure 4 Right, and draw it only if it does not occlude any previously drawn labels in the scene. (We use standard axis-aligned bounding boxes for efficient collision detection [Möller and Haines 1999].) The onscreen label density is user-controllable, by changing the size of the buffer zone around each label that determines the size of the bounding box. The drawing order described in Section 3.2 thus has a strong influence on which labels are visible. To improve legibility over complex backgrounds while limiting the occlusion in the tree interior, we use a contrasting one-pixel border rather than a background rectangle.

4 TreeJuxtaposer

We have implemented the techniques described above in TreeJuxtaposer, a global Focus+Context system for comparing large phylogenetic trees. The system handles simultaneous comparison of several trees using structural difference computations for every possible pair, and can also be used as a browser for a single tree.

Figure 5 shows a typical layout while comparing two small phylogenies. Each tree is drawn in its own panel using a rectilinear layout with the root on the left. Edges are rendered as simple lines (linewidth is controllable with slider on the tool panel) and nodes are indicated by a small square.

Our color scheme is designed to scale well even to very large trees. Our design results in large areas of densely packed edges, particularly at the leaf level. We wish to avoid total visual uniformity in those areas which would lead to a featureless expanse when not highlighted and excessive visual impact when highlighted. We thus modulate brightness and saturation of the tree edges.

Unmarked edges are rendered in lighter shade of grey as they are further away from the root. The resulting brightness gradations provide a redundant coding of topological information. Densely packed areas provide contextual landmark features aiding user orientation, and in expanded focus areas brightness is an additional clue about the current topological depth of the area under consideration. In highlighted areas we modulate the saturation of each node depending on the current visual extent of its subtree, so that densely packed areas are desaturated to conterbalance the visual impact of their aggregation.

4.1 Visual query mechanisms

To help the analysis of the differences between trees, nodes can be colored in four different ways. The most basic mechanism is linked mouseover highlighting: the node underneath the mouse cursor and the best corresponding nodes in all other trees are temporarily redrawn in in gold. Their labels are also unconditionally drawn with a gold background, so mouseover causes temporary popup for the vast majority of labels that have been suppressed to achieve the target visual density. Because manipulation of subtrees is both biologically important and central to our navigation scheme, we also indicate the extent of the subtree underneath that highlighted node in the active window with an unobtrusive frame. These changes are made with a combination of xor drawing and pixel readback in the front buffer, for immediate response without incurring the costs of a full scene redraw. The second highlighting mechanism operates through a standard search interface to let users rapidly find a node with a known name by selecting it from an alphabetized list. It is then marked in magenta and can be expanded on demand.

The third way to color trees is to highlight structural differences by marking nodes for which $S(v, BNC(v)) \neq 1$ in red. Our similarity definition was chosen so that the nodes marked in red pinpoint the nonmonophyletic clades. An unmarked node, a node with a BCN score of 1, does not imply the subtree beneath it has an isomorphic counterpart in the other tree. If we marked every node with non-identical structures underneath, the marks would be so numerous as to be useless. Rather, if all the nodes in a subtree are unmarked, then there does exist a structurally identical counterpart. Our design target is trees that are mostly similar, with scattered areas of difference. However, difference highlighting can be toggled off in cases where the trees are so dissimilar that the tree would be overwhelmingly red.

Finally, the entire subtree beneath a node can be highlighted in a user-chosen color, and the BCN of each node in the subtree is also highlighted in the other tree. That is, if the subtree T_{sel} is highlighted, then all the nodes in $\cup_{v \in T_{sel}} BCN(v)$ are highlighted in the other tree. Highlighting the subtree beneath a red edge will

result in highlighted nodes in the other tree that form a forest rather than a contiguous subtree. Figure 5 Left shows a nonmonophyletic clade: the subtree underneath the bottom red edge of the right-hand tree is blue, and on the left side the blue areas form a forest. Figure 6 shows four marked clades on the left-hand tree, and the top two monophyletic clades are scattered throughout the right-hand tree.

We do not maintain an explicit list of all highlighted nodes, because traversing that list would be linear in the total node count when large parts of the tree are marked. We delay the decision of what color to use for a node until render time, so that we only pay the cost of checking for the visible nodes. As described in Section 3.1, each subtree is associated with a range bounded by two integers, and it takes $O(\log^2 n)$ time to check whether two subtrees have leaves in common, or equivalently, to find whether a subtree should be highlighted. We cache these colors until the user's next change in marking subtrees or toggling the structural difference mark display. We thus maintain the interactivity of the system even when very large subtrees are selected.

4.2 Navigation

We navigate in TreeJuxtaposer with rubber-sheet style expansions and contractions of rectangular areas. The main navigation mode uses the tree topology as a starting point: the rectangular boundary of any subtree (by default, the one under the mouse) can be adjusted either by directly dragging the boundary rectangle corner to a new place or by animated transitions in fixed increments. During comparison users often want analagous areas in each tree to be the focus and find it cumbersome to navigate separately in each window. We provide the option of linked navigation, where the subtrees beneath the best corresponding nodes in other windows are resized in lockstep with that of the active window. Entire forests can be grown or shrunk with linked fixed-interval transitions. Unlinked navigation in a single tree is possible by first dragging out an arbitrary screen-space rectangle, then deforming it to the desired size.

5 Results and Discussion

5.1 Performance

TreeJuxtaposer is a highly scalable system, even compared to previous work that address the simpler problem of browsing: we can interact in real time with a single tree of 550,000 nodes. When comparing two trees, the system can handle a sum of up to 277,000 nodes, as shown in Figure 8. Both benchmarks were run on a 2.4 GHz Pentium III machine with 2GB of RAM, using java 1.4 with an heap of 1100 MB and an nVidia Quadro4 700XGL graphics card. It runs well on a large variety of displays from a simple laptop to the latest high resolution 3800x2400 pixel IBM T221 flatpanel (driven by an nVidia Quadro4 900XGL). We do not show that range in the figures because printing a full-resolution screenshot would require more area than a standard page, see instead the accompanying video.

In Figure 7 we show the overhead of our major design choices by graphing the time required to render an entire scene as the number of tree nodes increases. We see that the computation cost is linear in the number of nodes up to a threshold, after which the cost stops increasing. We have thus achieved our goal of bounding our computation costs by the display size rather than the total number of nodes when the tree is large. Our techniques of progressive rendering and guaranteed visibility do incur overhead compared to our baseline of LOD culling (which is itself always better than the naive approach). We succeed at mitigating the cost of GV with caching. Although progressive rendering is expensive, prioritizing the elements by visual importance is a major contribution to the effectiveness of the interactive experience when using TreeJuxtaposer.

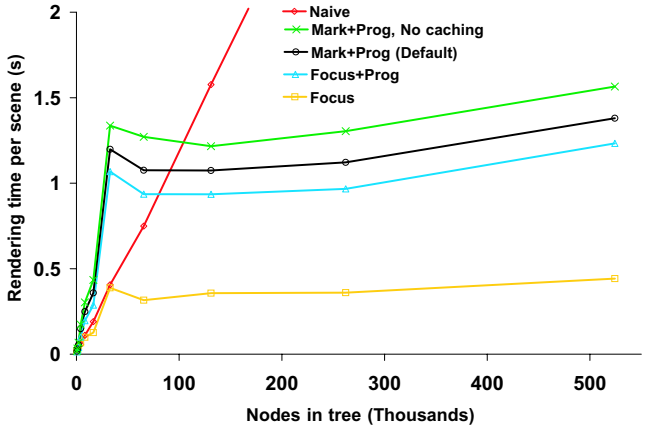

Figure 7: Statistics showing the overhead of our design choices, with the benchmark of complete binary trees ranging from 2^8 leaves to 2^{18} leaves. The red line is the naive case, where we render all edges starting from the root. The yellow baseline shows LOD culling. The cyan line has LOD culling and progressive rendering. The green line shows LOD culling, progressive rendering, and guaranteed visibility support. The black line, which represents the normal state of TreeJuxtaposer, has the previous features but caches the GV computations for added efficiency. These performance figures are for a 2.2GHz Pentium Xeon with 2GB of RAM, using java 1.4 with a heap of 1500 MB and an nVidia Ti4600 graphics card.

Our progressive rendering algorithm maintains a constant frame rate on displays of up to nine megapixels using only a single graphics card. Nevertheless, we see the inevitable limit of this approach even if we had an optimal ordering for drawing: as display size grows, the amount of time to completely render a scene also grows and the proportion of the scene that we can draw in a single frame decreases. This problem is not specific to our application or configuration. For example, a similar problem was reported [Guimbretière 2001] while using brainstorming tools on the Stanford Interactive Mural, a 9 megapixel digital whiteboard driven by a 32-node rendering cluster.

We will continue to work towards the motivating example of handling the entire Tree of Life. Thus far we have focused somewhat more on time efficiency than space efficiency, so our current bottleneck is memory footprint rather than computation. We believe that making several obvious refinements will lead to major memory efficiency improvements.

5.2 Structural Comparison

Our algorithms free the user from the painstaking task of manually identifying patterns, or worse yet manually confirming that a given pattern does not exist in the dataset. The applicability of our method of computing best corresponding nodes has been confirmed by every biologist to whom we have shown the system, which is heartening given the plethora of different metrics for whole-tree comparison that are currently in use [Robinson and Foulds 1981; Sokal and Rohlf 1981]. We have found that the comparison features of this tool are appreciated even by the many biologists who still work with relatively small datasets.

As for efficiency, the query cost is negligible, and our preprocessing algorithms are very efficient in practice. For example, BCN computation takes only 7 seconds at startup time to compare a tree of 137,000 nodes with one of 140,000 nodes, and the range tree construction takes 29 seconds. Both benchmarks are on the Xeon PC mentioned above using the datasets shown in Figure 8.

5.3 Focus+Context

Previous Focus+Context literature focuses mostly on browsing. While SpaceTree [Plaisant et al. 2002] and H3 [Munzner 1998; Munzner 2000] are able to display large trees, these systems are ill-suited for the task of comparison because they cannot guarantee that marked areas will be visible on the screen. SpaceTree uses unconstrained 2D navigation which means that many nodes can be outside the frustum, while H3 culls away subtrees whose projected areas will be less than one pixel on the screen. (Moreover, the H3 layout results in very poor information density with the common phylogenetic case of binary trees.)

Our prevention of frustum and subpixel culling comes at a cost. Global Focus+Context navigation techniques could be confusing to novice users because highly compressed areas are sometime difficult to identify. Guaranteeing visibility of subpixel selected areas incurs the computational expense of a range check before halting the quadtree traversal. Nevertheless, we see from Figure 7 that caching the information leads to acceptable performance.

Visual aggregation versus glyphs. In TreeJuxtaposer we draw as much detail as possible, down to the level of one pixel. This maximum visible detail approach is diametrically opposed to displaying aggregate information in a glyph, as exemplified by DOITree [Card and Nation 2002] and SpaceTree [Plaisant et al. 2002], where visual encoding techniques deliver abstract semantic information about a structure and hide the details. Because we use true geometry rather than a monolithic glyph, we can use the very lightweight query mechanism of mouseover highlighting to quickly get structural and label information at the pixel level. Similarly, our guaranteed visibility framework goes beyond simply indicating the existence of a marked area - the exact position of the mark imparts information about the location of that marked area within the subtree. Although we have designed the navigation system for maximum fluidity, we also recognize that having high information density of any given static view is useful in minimizing the total amount of navigation that must be undertaken by the user.

In most glyph-based systems, expansions are explicitly triggered by user selection of a point of interest, and sometimes contractions must also be explicitly requested. In our approach, user navigation implicitly controls expansions and contractions of subtrees, a feature that supports faster assimilation of unfamiliar dataset structure.

We have taken a particularly extreme approach in this maximum-detail direction to explore its potential, because the preponderance of previous work has investigated the benefits of aggregation. Future systems may well benefit from hybrid approaches that merge the benefits of both visual simplicity and the power of detail.

5.4 Guaranteed Visibility

The concept of guaranteed visibility has proven to be very powerful because it relieves our users from the job of exhaustive exploration, by providing direction on areas of interest as navigation targets. It is of course not limited to this specific application and can be applied to other information visualization system as well. For example, visible landmarks are critical for wayfinding in the physical world [Lynch 1960], and we conjecture that GV will help users maintain their orientation when navigating through large information spaces.

Occlusions. While occlusion is a familiar problem in 3D systems (for example, the influential Cone Tree [Robertson et al. 1991]), it can also be present in 2D approaches because objects can mask each others such as in the DOI Tree [Card and Nation 2002] or simple items like labels can hide highlighted areas. In our system only labels can create occlusions. While we alleviate this problem by using a contrasting border rather than the usual opaque background rectangle for the label, marks can still sometimes be hidden. Users must occasionally turn off label drawing briefly in order to locate areas of interest. Although this solution is not ideal, it does not occur often in practice, especially because most phylogenetic trees do not have interior node labels. Furthermore, mouseover highlighting where labels appear briefly as popups does allow users to stay oriented even in the interior if they have chosen a sparse label density. While we considered translucent labels, they would be very difficult to read, and the progressive rendering algorithm described in Section 3.2 would have to be much more complex to also support back-to-front drawing semantics.

Guaranteed frame rate. Our work uncovers an interesting interaction between guaranteed visibility and guaranteed frame rate algorithms. With a generic guaranteed frame rate algorithm, some objects may be culled because the system is running out of time to draw them. Even in a GV system with progressive rendering, navigating toward a selected mark can be difficult because the marks are not guaranteed to be in the first frame, only in the finished scene, so the mark could disappear from view during interaction. In the worst case, the entire scene could be marked so there is no way to guarantee that all marked areas are drawn within the first frame. We could address the situation where the number of marked areas is small and they could have been visible had they been ordered early in the drawing queue. Although our design decision to evaluate on the fly whether an element is marked makes it expensive to keep track of marked areas explicitly, it could be interesting to integrate marking status with our current drawing order criteria.

Our current approach to guaranteed frame rate is purely geometric. It might be fruitful to exploit inter-frame coherency through texture, as done in the Talisman system [Torborg and Kajiya 1996]. We could start by saving the result of the previous scene as a texture, map it to reflect the changes in expansions and contractions, and then fill in only the areas of major change geometrically.

Visibility versus detectability. In this paper we have focused on the notion of visibility: the fact that an object could be seen on the screen. However, something visible can still be difficult to detect: an object made from 1x4 red pixels is technically visible, but could be difficult to detect on a 200 dpi display. The relationship between the two notions is in general non-trivial: factors such as saturation, hue, brightness, and visual extent enter into play. For example, simply increasing the visual extent of highlighted areas would lead to undesirable occlusion, and using other visual encoding techniques such as a moving outline could prove distracting.

Indirect guaranteed visibility. Our definition of guaranteed visibility is strict, mandating that all marked objects are always directly visible on the screen. If instead some navigation is acceptable, the underlying goal of ensuring that highlighted objects are not missed can be approximated through other mechanisms. For instance, graphical `diff` tools have a marked scrollbar that can be considered an always-visible index, even though the main view is only partial. Another approach would be automated navigation, where a series of viewpoint changes shows all interesting places briefly, in the style of Asimov's Grand Tour [Asimov 1985]. Strict GV has enough constraints that it cannot be integrated into many previous systems, but indirect GV adds enough flexibility that it could be added to glyph-based systems such as DOI Trees [Card and Nation 2002]. The multiscale navigation analysis of Jul and Furnas discusses the need for visible "residue", which is a form of indirect GV.

5.5 Other application domains

Our current system is a standalone application targeted to the area of phylogeny. We would like to integrate our system with existing phylogeny manipulation tools like Mesquite[3] at the API level,

[3] `http://www.mesquiteproject.org`

so that the results of more sophisticated biological queries can be graphically explored in large trees.

While phylogenetic tree comparison was our primary intended task, we believe that our system will be useful in numerous other domains. Although the nomenclature of *monophyletic clades* is unfamiliar outside of phylogenetics, our definition of similarity leads to a visual indication of the exact areas of structural divergence between trees that is broadly applicable. In biology alone there are many other problems requiring tree comparison, including comparing the dendrograms resulting from alternate hierarchical clusterings of microarray data [Seo and Shneiderman 2002]. Formal methods for verifying computer hardware and software generate huge proof trees where comparison could guide the developers in refining their solver algorithms [Neufeld et al. 1997]. Web designers are often interested in comparing the hyperlink structure of their sites before and after major site reorganizations [Chi and Card 1999]. Figure 8 shows a networking example of two spanning trees of the Internet backbone router topology. This is the same dataset used in a widely distributed series of four posters [4], one per year, but it was essentially impossible to use them to compare the network structure from year to year through visual inspection. It had never been explored in an interactive system due to its sheer size before we loaded it into TreeJuxtaposer.

6 Future work

Our current definition of similarity does not take edge weights into account. Many biologists use trees with weighted edges, where the weights represent either elapsed time or levels of uncertainty. We would like to develop a structural comparison algorithm that deals properly with edge weights. Defining an appropriate similarity measure is a challenging problem, and even some obvious extensions are difficult to compute efficiently.

We would also like to explore further the concept of guaranteed visibility. Our work already identifies some important aspects of this concept but much more needs to be done. For example, while we guarantee that marked areas will be shown to the user, our marks are relatively coarse-grained: our structural difference marks do not distinguish between a contiguous subtree and a separated forest. We intend to extend the notion of guaranteed visibility so that marks can reflect more information about the hidden information they represent.

7 Conclusion

We have presented a system that allows interaction with and detailed structural comparisons between trees of over 100,000 nodes each, and browsing single trees of half a million nodes. Our approach to visual structural comparison and algorithms for efficient structural difference computation fill a needed gap. Our new global Focus+Context navigation algorithm allows scalable exploration and comparison. We have introduced the concept of guaranteed visibility, and found it to be a useful property in expanding the reach of our systems.

Acknowledgements

We thank systematic biologists David Hillis, Bob Jansen, Derrick Zwickl, and Will Fischer of UT Austin for explaining biological tasks, feedback on prototypes, and allowing us to use their datasets in Figures 1 through 6. The data in Figure 8 was graciously provided by Bill Cheswick. Thanks to Nina Amenta and Katherine St. John for inspiring this project, discussing ideas, and providing partial funding through NSF/DEB-0121682. We appreciate discussions with Valerie King on algorithms and with Catherine Plaisant on the name "guaranteed visibility".

[4] `http://www.peacockmaps.com`

References

ADAMS, E. N. 1972. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology 21*, 390–397.

AMENTA, N., AND KLINGNER, J. 2002. Case study: Visualizing sets of evolutionary trees. In *Proc. InfoVis 2002*.

ASIMOV, D. 1985. The grand tour: a tool for viewing multidimensional data. *SIAM J. Sci. Statist. Computing 6*, 1, 128–143.

BARTRAM, L., HO, A., DILL, J., AND HENIGMAN, F. 1995. The continuous zoom: A constrained fisheye technique for viewing and navigating large information spaces. In *Proc. UIST '95*, 207–215.

BECKER, R. A., AND CLEVELAND, W. S. 1987. Brushing scatterplots. *Technometrics 29*, 127–142. Reprinted in Dynamic Graphics for Data Analysis, W. S. Cleveland and M. E. McGill eds., Chapman and Hall, New York, (1988).

BRODER, A. Z., GLASSMAN, S. C., MANASSE, M. S., AND ZWEIG, G. 1997. Syntactic clustering of the Web. In *Proc. Sixth International World Wide Web Conference*, 391–404.

BRODER, A. Z. 1998. On the resemblance and containment of documents. In *SEQS: Sequences*, 21–29.

CARD, S. K., AND NATION, D. 2002. Degree-of-interest trees: A component of an attention-reactive user interface. In *Proc. Advanced Visual Interfaces (AVI 2002)*.

CHI, E. H., AND CARD, S. K. 1999. Sensemaking of evolving web sites using visualization spreadsheets. In *Proc. InfoVis 1999*, 18–25.

CHI, E. H., PITKOW, J., MACKINLAY, J., PIROLLI, P., GOSSWEILER, R., AND CARD, S. K. 1998. Visualizing the evolution of web ecologies. In *Proc. of ACM CHI 98 Conference on Human Factors in Computing Systems*, 400–407.

DAY, W. H. E. 1985. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification 2*, 7–28.

FURNAS, G. W. 1997. Effective view navigation. In *Proc. SIGCHI 97*.

GRAHAM, M., AND KENNEDY, J. 2001. Combining linking & focusing techniques for a multiple hierarchy visualisation. In *5th Int'l Conf. on Information Visualisation - IV2001, University of London*, IEEE Computer Society Press, 425–432.

GUIMBRETIÈRE, F. 2001. *Fluid interaction for high-resolution wall-size displays*. PhD thesis, Stanford University.

HAREL, D., AND TARJAN, R. E. 1984. Fast algorithms for finding nearest common ancestors. *SIAM Journal of Computing 13*, 338–355.

HERMAN, MELANÇON, G., AND MARSHALL, M. S. 2000. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics 6*, 1, 24–43.

JUL, S., AND FURNAS, G. W. 1998. Critical Zones in Desert Fog: Aids to Multiscale Navigation. In *Proc. UIST '98*, 97–106.

LAMPING, J., RAO, R., AND PIROLLI, P. 1995. A Focus+Content Technique Based on Hyperbolic Geometry for Viewing Large Hierarchies. In *Proc. CHI '95*, 401–408.

LYNCH, K. 1960. *Image of the City*. MIT Press.

MADDISON, D., AND MADDISON, W., 1992. MacClade: Analysis of phylogeny and character evolution.

MARGUSH, T., AND MCMORRIS, F. R. 1981. Consensus n-trees. *Bulletin of Mathematical Biology 3*, 239–244.

MÖLLER, T., AND HAINES, E. 1999. *Real-Time Rendering*. AK Peters.

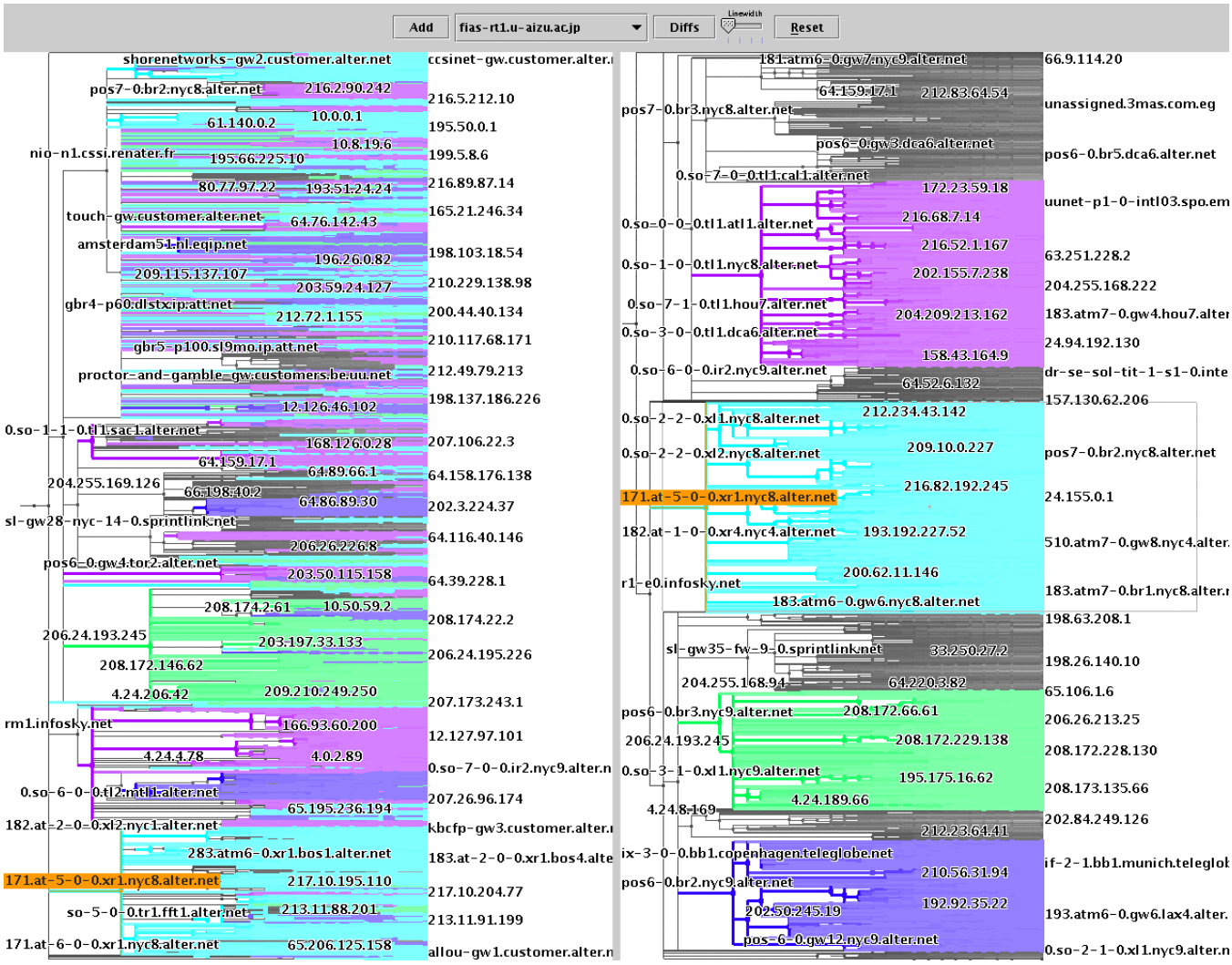

Figure 8: Internet backbone router spanning tree, with topologies inferred from two sources: UDP/traceroute on the left (137,000 nodes) and ICMP/ping (140,000 nodes) on the right. Marking subtrees shows that the two different packet types can take very different routes.

MUNZNER, T. 1998. Drawing Large Graphs with H3Viewer and Site Manager. In *Proc. Graph Drawing '98, Lecture Notes in Comp. Sci. 1547*, Springer-Verlag, 384–393.

MUNZNER, T. 2000. *Interactive Visualization of Large Graphs and Networks*. PhD thesis, Stanford University.

NEUFELD, E., KUSALIK, A. J., AND DOBROHOCZKI, M. 1997. Visual metaphors for understanding logic program execution. In *Graphics Interface '97*.

PLAISANT, C., GROSJEAN, J., AND BEDERSON, B. 2002. SpaceTree: Design evolution of a node link tree browser. In *Proc. InfoVis 2002*.

PREPARATA, F. P., AND SHAMOS, M. I. 1990. *Computational Geometry: An Introduction*, 3rd ed. Springer-Verlag.

ROBERTSON, G. G., AND MACKINLAY, J. D. 1993. The document lens. In *Proc. UIST '93*.

ROBERTSON, G., MACKINLAY, J., AND CARD, S. 1991. Cone Trees: Animated 3D Visualizations of Hierarchical Information. In *Proc. SIGCHI '91*, 189–194.

ROBINSON, D. F., AND FOULDS, L. R. 1981. Comparison of phylogenetic trees. *Mathematical Biosience 53*, 131–147.

ROST, U., AND BORNBERG-BAUER, E. 2002. Treewiz: interactive exploration of huge trees. *Bioinformatics 18*, 1, 109–114.

SARKAR, M., SNIBBE, S. S., TVERSKY, O. J., AND REISS, S. P. 1993. Stretching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens. In *Proc. UIST '93*, 81–91.

SEO, J., AND SHNEIDERMAN, B. 2002. Interactively exploring hierarchical clustering results. *IEEE Computer 35*, 7 (July), 80–86.

SHIVAKUMAR, N., AND GARCÍA-MOLINA, H. 1995. SCAM: A copy detection mechanism for digital documents. In *Proc. 2nd Annual Conf. on Theory and Practice of Digital Libraries*.

SOKAL, R. R., AND ROHLF, F. J. 1981. Taxonomic congruence in the Leptopodomorpha re-examined. *Systematic Zoology 30*, 309–325.

SWOFFORD, D. L., 1998. PAUP* 4.0 - Phylogenetic Analysis Using Parsimony (*and Other Methods).

TORBORG, J., AND KAJIYA, J. T. 1996. Talisman: Commodity realtime 3D graphics for the PC. In *SIGGRAPH 96*, 57–68.

WARD, M. O., AND MARTIN, A. R. 1995. High dimensional brushing for interactive exploration of multivariate data. In *Proc. IEEE Visualization '95*, 271–278.

ZHANG, L. 2003. On matching nodes between trees. Tech. Rep. 2003-67, HP Labs.