Treefoil: Visualization and Pair Wise Comparison of File-system Trees

Shannon Bauman, James Clawson, Josh Cothran, Jeanie Miskelly, Zach Pousman
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332 USA
{baumas, jamer, infinite, miskelly, zpousman} @ cc.gatech.edu

ABSTRACT
Much work has been done to visualize single trees in various domains, including those of biology (taxonomy) genetics (genomics) and file systems. Much less attention has been paid to the pair-wise comparison of trees to find similarities and differences. We introduce a visualization tool, Treefoil, to perform pair-wise comparisons on large file system data. This tool is useful for systems administrators and web developers in particular but also to computer users in general. Our visualization technique presents an overview and detail interface, based on a dotplot for overview tasks and a dual pane file browser for the detail views. This paper outlines the design of our system, its implementation, and how it helps solves the problems associated with pair-wise comparison of hierarchical data structures.

Keywords
Information Visualization, Dotplot, Information Hierarchy, Tree Comparison, File system browser

INTRODUCTION
Directed acyclic graphs, commonly called trees, are used in most existing file systems. Trees are a data structure that the majority intermediate and advanced computer users understand and rely upon to do many tasks. Local file browsers, remote file browsers (Gopher and FTP), web sites, and even application menus tend to reflect this structure in their GUI's. Today's existing tools and their visual representations may be passable for viewing a single hierarchy but are sub-optimal when it comes to the task of comparing two trees. Treefoil is an information visualization tool to assist in the comparison of two file system hierarchies. Our tool is optimized to compare the change from a tree ***a*** to a modified version of the same tree, tree ***a'***. Comparing trees ***a*** and ***a***' is relevant in many situations; two of the most common are the comparing of different backed-up versions of a hard drive or file system and comparing an uploaded or FTPed website to the same website stored on a local PC. A user wants to see how similar the two trees are, where the differences lie (and their character), among other tasks. We provide an answer to the problem of comparing two instances of a tree structure that have changed over time.

RELATED WORK
Several key advances punctuate the history of visualizing directed graphs and file-system trees. The first visual representations of trees began in WIMP systems with the rise of personal computers. These visualizations were predominantly textual, though they were interactive (folders could be opened and closed, displaying their contents). These systems were built into the GUI interfaces of various operating systems and remained static until the 1990s, when an explosion of systems attempted to bring true interactivity, better representations of global structure, and detailed information to hierarchies. Johnson and Schneiderman [5] introduced tree-map visualizations for file-system data (a system designed in response to the fact that Dr. Shneiderman's personal hard disk was rapidly running out of space and his desire to know the exact contents on the drive so that he could easily optimize and organize). This visualization is planar and very space efficient, showing well the global structure. Some problems were identified, and these were solved or mitigated by further work [1, 11, 12]. Other attempts to visualized file-system data include Xerox Parc's work to display file hierarchies in three space [10], work to display hierarchies in parabolic space [6], and even 3-Dimensional parabolic space [7]. Yet none of these tools allow the comparison of two file hierarchies.

A smaller body of work details the comparison of file-trees, or any kinds of trees or graphs. Wittenburg and Sigman's Treeviewer [14] is a system that shows additions and deletions to a web search query. Along with Huang and Eades' visualization of graphs [4], use animations to show changes in hierarchical structure, at least to some degree. Further, Graham, Kennedy, and Hand [2], discussed the challenges of visualizing multiple overlapping hierarchies (a related problem to ours, in the discipline of biological taxonomy) and suggests that all visualization techniques are a "choice between showing change over

time or the change over space." They suggest techniques including the highlighting nodes of user-interest, and filtering via clustering, neither of which has been explored in the research literature.

Dotplots, a general coordinate matching approach for comparisons of items, was developed extensively in the biological sciences [7, 9]. It has since been applied to a small degree in software visualization. Helfman [3] even showed a file-tree comparison using dotplots, though his real focus was on applying dotplots to software visualization for programmers. That work is cited by others building systems for software visualization. We take Helfman's work in a different direction, and introduce a system for comparing file-trees using a dotplot visualization.

DESIGN DETAILS

We present Treefoil, an information visualization tool that supports the comparison of two file-system trees. The application was designed around the requirements of file-systems in particular. We outline a set of user tasks that motivate the work.

1. Compare the file-system at a global level for quick overview of similarities and differences.
2. Understand the changes that have occurred as the tree was changed over time.
3. Identify additions or deletions from the tree.
4. Compare structural changes to the tree, the reorganizations of the folders and files.
5. Locate files of interest (new files, large files, popular files) and get details on these items.
6. View details of files and folders including attributes like name, create date, creator, owner, hitCount (for internet pages), size, etc. Possibly, a user wants to compare two or more pages' details.

Our design consists of two interactive visualizations. First is a **Dotplot View**. The dotplot view provides a basic overview of the level of similarity of the two trees and visually highlights areas of interest. Second, a **Tree-view Inspector** allows the user to more thoroughly explore the structure of the two trees. These two visualizations work together to provide a system of overview and detail. The dotplot view provides global overview and the tree-view inspector provides details.

The application provides users with powerful interactive tools in the dotplot overview. A docking palette implementing **Dynamic Query sliders** allows users to specify constraints on the view. Users can quickly highlight files and folders of interest based on a variety of criteria (including hit count, size, and age, as well as additions and deletions). In the detailed Tree-view Inspector, we have designed a set of **Contextual Anchors**, which supplements tree comparison in the tree-view inspector. We discuss each of these aspects of the tool in further detail here.

Dotplot View

The overview visualization, a modified dotplot, provides the user with a quick assessment of the differences between two trees. A dotplot is a simple visualization of comparison. We give an example of a very simple dotplot in Figure 1. The two structures to be compared are subdivided into their constituent elements, be they lines of code, sequences of nucleic acid, or files and folders. The list of elements is arranged according to a preorder traversal along the axes of a cartesian plane. A dot (or, when the number of parts is small, a dash) is placed at every point where nodes from both trees are identical.

By glancing at the graph, a user can quickly evaluate the overall similarity of the two trees. If the trees are identical, a diagonal line is drawn (Figure 1), projecting from the origin. Many scattered dots indicate that the trees are structurally very different. This technique shows a translocated subtree as a space-shifted line segment (Figure 2). Duplicated files will be visible because a node from one will match multiple nodes from the other tree.

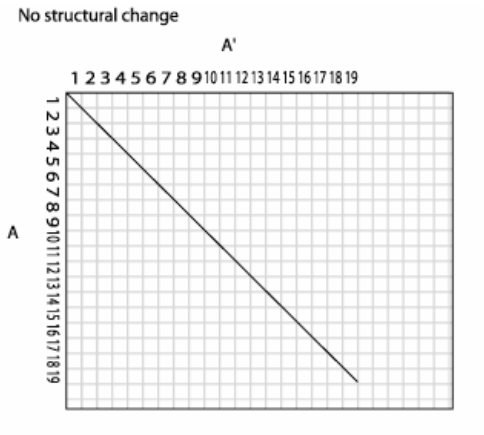

Figure 1: Identical Trees

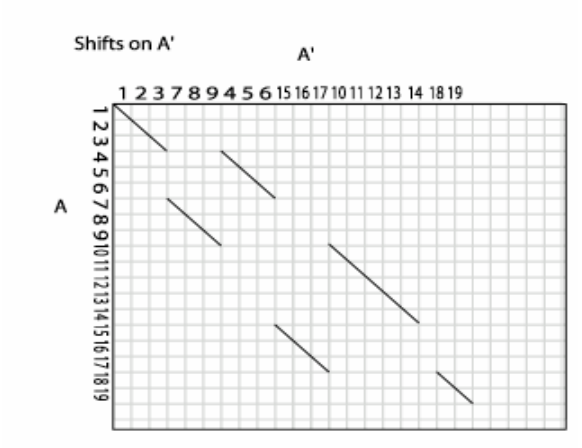

Figure 2: Shifted Subtree

The absence of dots indicates that nodes on one tree do not match any of the nodes on the other tree; hence nodes were either added or deleted. This results in columns or rows without any dots. Color can be added to better show these additions or deletions. In Figure 3, the red horizontal lines show the nodes that were deleted from tree ***a***. The green vertical lines show the new nodes in tree ***a***'.

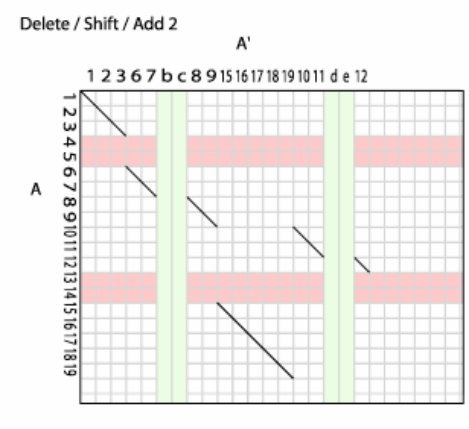

Figure 3: Deleted, Added, and Shifted Nodes

Figure 4 shows what a segment of a typical file system may look like. It shows an example of an addition, a deletion, as well as an example of a duplication of a whole folder.

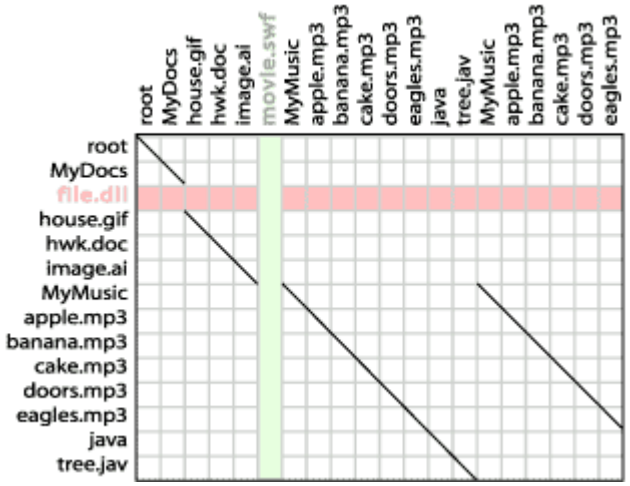

Figure 4: Deleted, Added and Duplicated Nodes

Interaction

Figure 5 shows two different features of the application that has been designed. In this figure, the two axes do not have labels, but instead show green and blue pixels. The green pixels here represent files and the blue pixels represent folders. Labels are not used on these axis because of the large number of nodes represented on the screen. However, as the user moves the mouse around the interface, a "detail" window moves with the mouse. When the detail window hovers over a colored pixel, it will display the name of the files or folders over which it is hovering. When the user clicks the mouse button, it will drop the detail window at that location. The user will then be able to "nudge" the window up or down to see other files that are close to where the window was originally placed. The window can be raised back to its hover state and moved again by clicking

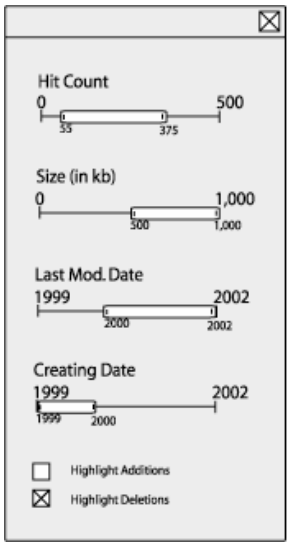

Figure 6: Dynamic Query Palette

outside of the boxed area. By clicking on one of the file or folder names, the user will be brought to the Tree-view Inspector (mentioned later) at that location in the file structure.

In our original designs, we suggested rotating the dotplot 45°. We did so for two reasons. First, a rotated dotplot would place the center "match" line as a vertical, instead of a diagonal. We felt that this would assist the mental modeling and reading of the visualization since a user would more easily track the dots. Second, a rotation afforded the ability to place labels on the axes horizontally. This would, intuitively, be much easier to read. However, this was not implemented in the current application. We have a standard, non-rotated dotplot instead.

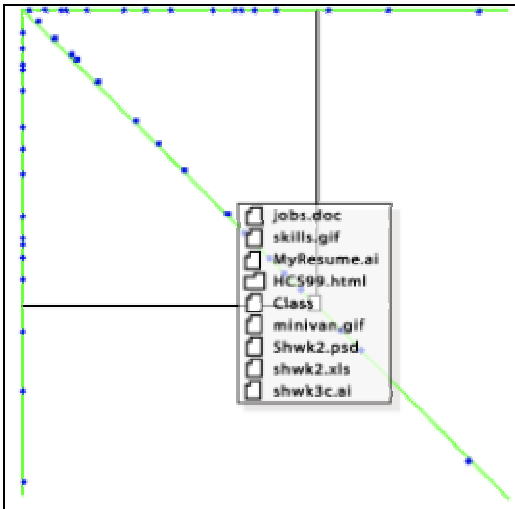

Figure 5: Using the detail window to show file names

Dynamic Query Palette

Dynamic interaction is key to the success of our tool. Systems such as UMD's Home Finder [13] have successfully incorporated dynamic interaction with data by the use of a palette and we have found that they are particularly useful for intensive exploration of data. A palette is essentially a bounded section of the screen that is sometimes moveable, and sometimes locked into a location. A pallet normally contains various types of interaction widgets, including sliders and check boxes.

These types of palettes offer a good way for the user to explore trees based on attribute criteria. A visualization of the concept is illustrated below (Figure 6). This design allows a user to inspect all the data to determine the lower and upper bounds of a particular variable of the dataset. It also affords the viewing of a range of values for example files that are larger than 500k but smaller than 1 MB. The design applies the same methods to each of the variables, letting users build complex queries easily and quickly.

The interface would reflect the changes made by the user by highlighting and fading. As the user makes a dynamic query, nodes that fall into the specifications of that query will be highlighted in a user-selected color. Nodes that are no longer within the query terms will turn gray. This will allow the user still see the context of the whole hierarchy, but also see the items that match the query.

Another aspect of the palette would allow the user to select specific tools to assist in analyzing the trees. One such tool highlights both deleted and added files, thus making them more obvious to the user. Another sets the folder depth of the inspector view. This allows users to configure a global setting for folder depth allowing a traversal of n levels down, or allowing a traversal of the entire depth. These dynamic interactions afford efficient browsing of the similarities and differences between two trees.

The Tree-view Inspector

The Tree-view Inspector is the portion of the application that affords the user the ability to focus on specific files or folders for direct comparison. The visualization used for this detail oriented work is a side-by-side pairing of the canonical "explorer view" provided by many existing operating systems. We chose this display in part because of its ubiquity and familiar operation. In our design, the features of a standard (single tree view) file system explorer are maintained, but extended to deal with our comparison tasks. Some details of the representational aspects and the animations are shown below (Figure 7).

We propose an animated transition between the two visualizations. The Tree-view Inspector is a detail view, while the Dotplot view is an overview. When a user launches the tool, the first visualization available to her is the Dotplot overview. When a user requires further details

into the structure of the tree, or metadata regarding a folder or file, she may click on a point in the Dotplot view to bring up the Tree-view Inspector. The animation we designed shrinks the Dotplot to an iconographic representation while at the same time bending the two arms of the dotplot towards vertical. At the end of the transition, the Tree-View Inspector is full-sized and manipulable.

We attempt to keep certain elements of both visualizations constant to assist users in interacting with the system effectively (and minimizing learning time). The color-coding of folder and file information, as well as the colors of added, deleted, and highlighted files is the same between the two views. Also, the application opens the Tree-view Inspector to the exact file or folder that was selected in the Dotplot view, opening whatever folders are necessary to make the file in question visible. Lastly, we design a symmetric operation, which lets a user switch from the Tree-view Inspector to the Dotplot with a single click on the small version of the Dotplot. Whatever file or folder the user had selected at the time becomes the item of focus in the Dotplot view.

The Tree-view Inspector works analogously to the Microsoft Windows file explorer. A fundamental interaction is the user's ability to open and close folders and sub-folders to "drill-down" to files of interest. Tabular fields of metadata hold the size, hit count, creation date and other attributes of each file and folder. An important addition to the functionality provided is the use of a keystroke to toggle the browsing mode. A user can browse the two trees in an **unconstrained mode** and in a **constrained mode**. In the unconstrained mode, an action on one tree, a click to expand a folder say, changes only the clicked file tree. In the constrained interaction mode (actuated by the user holding down the □ or the ctrl key), a browsing operation is mirrored between the two trees. If a folder is not present, having been added only to one of the trees being explored, then no action is taken. We also introduce the notion of contextual anchors below which extends the functionality of the visualization tool even further.

Contextual Anchors

Contextual Anchors are a way to provide further interactive control in the Tree-view Inspector. We wanted to provide interactive tools so that users can more effectively navigate as well as continue to keep some elements of structure visible in the details view (the detail view being limited to approximately 25 items per tree). For example, a user may seek to track the location of a single file or folder or a group of items from the ***a*** tree to the ***a'*** tree and Contextual Anchors make this easy. We detail the designs in this section for three kinds of Contextual Anchors, however, the constraints of time prevent us from implementing all of them.

Stretchable Context Anchors:
Stretchable Context Anchors visually connect a pair of identical folders or files of interest in the two file hierarchies. This is accomplished via stretchable "rubber bands," with barbell ends. There are 16 possible rubber band colors, and one color is associate with one pair of files. In the application a user would click on an icon or name or invoke a menu. The menu would provide the following functionality:

Create Context Anchor – This would allow a user to create the context anchor and select a color for it (from one of 16 easily differentiated colors)

Snap to Context Anchor – This option would allow a user to bring the matching file in the other file system to the same vertical position on the screen, "snapping" them together using the current file-system as the focus of the operation.

Snap to Matching Context Anchor – Allows a user to snap the current display to the other identical file, forcing the current tree to open and close folders to match it (instead of the other way around as in 2).

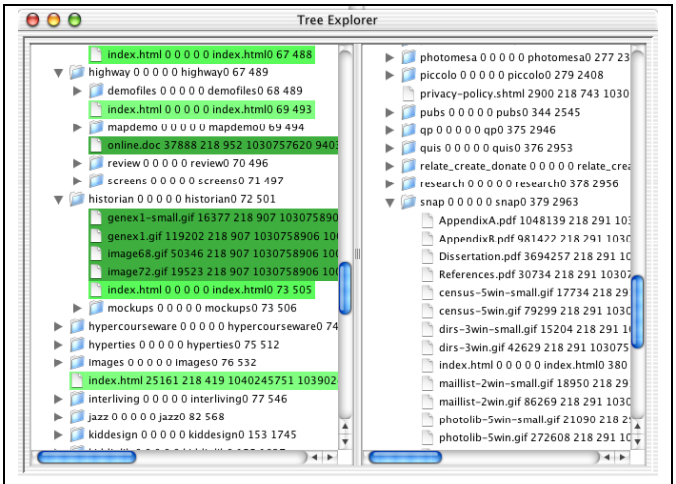

Figure 7: Tree-view Inspector: The two file system hierarchies are shown side by side.

When a user clicks on an existing Context Anchor at either the endpoints or the line itself, a similar contextual menu is displayed. It contains similar elements, but with the ability to turn off the selected Context Anchor or all of the Context Anchors, as in Figure 8.

The other two applications of contextual anchors are more involved. The first, which we call the Overlay approach, superimposes the two trees (with 100 pixels of horizontal offset). Since the tool is specialized to deal with trees that are similar to one another, a single set of labels could be used for at least some of overlapping trees Transparency and other contextual feedback could be used to assist in comparison.

The third application, which we call the Morph approach, actually takes the first tree (*a*) and animates it into the second tree (*a'*) over a specified period of time. In this approach, our use of Contextual Anchors is a direct analog to Macromedia Flash's shape hints. In both techniques, Contextual Anchors provide an opportunity for the user to "sculpt" the visualization. We have found the use of techniques to be invaluable in these more Cartesian-oriented domains, and are interested in testing the utility of user-guided information visualization in this domain.

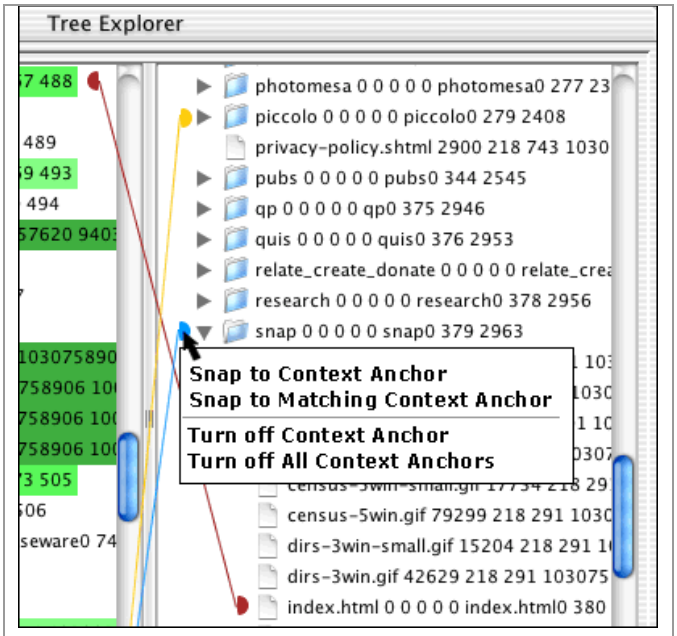

Figure 8: Context Anchors. The Context anchors are the barbells between identical files in the two hierarchies. Shown also is the contextual menu that appears with a user click.

IMPLEMENTATION

This system was created in Java 1.4.1 for deployment onto personal computers. The software reads in two XML files as parameters, and then displays the comparison of the two trees stored within the files. The XML schema stores both trees are stored in the XML files in a depth first manner, a convention that is maintained in our visualization.

A compromise was made related to the number of nodes the application could take in to analyze. While 60,000-100,000 nodes would have been preferred, we settled for visualizing 3,000-5,000 nodes. This was due mostly to the processing power and memory requirements of the machines we were using. Currently, it takes around 10 seconds to read in, store, and then display 3,000 nodes. This is a number that we hope to be an acceptable one-time-only pre-loading time for users.

The implementation of the system focused primarily on creating the Dot Plot view, as well as the Tree-View Inspector and being able to switch intelligently between the two. Other features such as the dynamic query sliders were not implemented as designed. We chose to implement a palette that had the same basic functionality without the performance problems common to Dynamic Queries. This was due in large part to time constraints, as well as to limited programming experience by the team. It is something that will be done in future work.

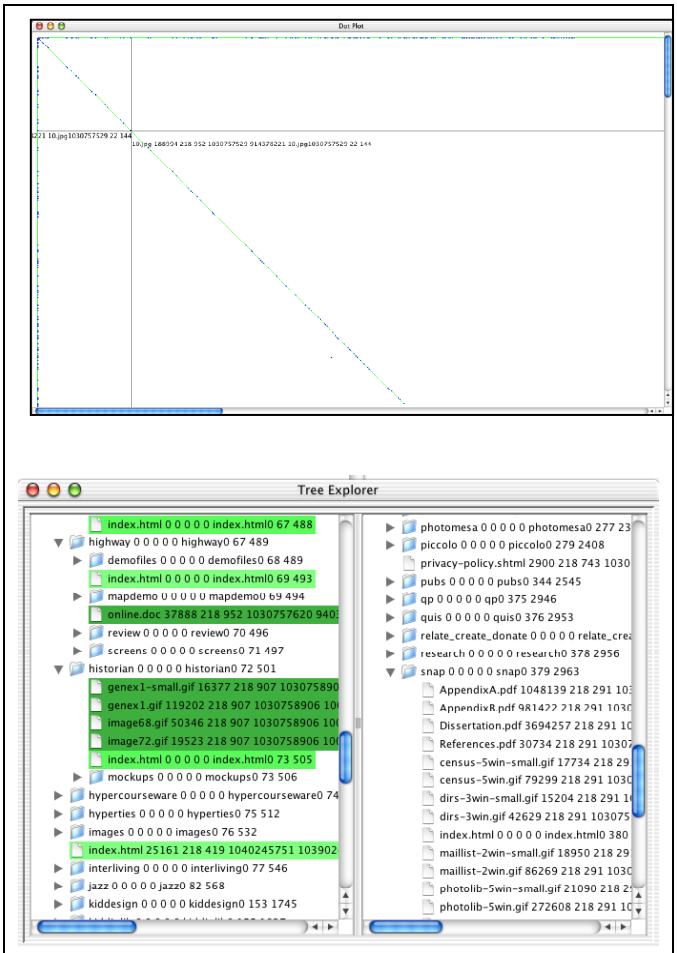

Figure 9: The implemented Treefoil application. The dotplot view shows the crosshairs on a particular file. The Tree-view Inspector shows files highlighted by creation date.

FUTURE WORK

One feature that is to be added to the system at a later time is the contextual anchors. Java code has been found that can be molded into our system to fit these needs.

Dynamic Queries have always been a fundamental part of the design of the system, and therefore will be fully implemented in the future. These will allow for smooth interaction with the system by the user.

The initial design of the application had the dot-plot view rotated at a 45° angle. This would allow for the user to view the results of the plot by scanning straight down the page, as apposed to scanning diagonally across the page.

The zoom windows on the dot plot have not yet been implemented. The dynamically changing zoom window may be processor intensive, and therefore may pose a problem to implement. In addition, the idea of allowing multiple zoom windows has been brought forth, and may also be pursued at a later time.

The other major area of future work is in user testing. Testing needs to be done on multiple different fronts to see the effectiveness of this interface in use. Though there are not many existing systems, we would like to perform quantitative as well as qualitative studies to determine the system's effectiveness , efficiency, and likeability.

CONCLUSION

In this paper we presented Treefoil a dynamic tool that aids a user in the cognitive tasks associated with the pair-wise comparison of two hierarchical data structures. Specifically it address the ways in which our tool affords the user the ability to easily recognize and identify changes between two trees and, in addition, accomplish the more fine grained tasks of identifying node additions, deletions, translocations, and duplications.

The system design introduces the idea of interactive dotplot views for file system data. As well, we introduce a dual-pane tree explorer that uses two standard tree explorers in an innovative way to allow the constrained viewing of folders and files among the two trees of comparison. Stretchable contextual anchors, visual representations of matching between two identical files are also brought to bear on this domain. Our stretchable context anchors act as interaction elements to focus and refocus the detail views. The Treefoil system as implemented does not contain some of these features. We plan to implement them in further iterations of the application (see Future Work) as well as to perform a user evaluation on the system to determine its applicability to professional and casual computer users.

ACKNOWLEDGMENTS

We wish to acknowledge John Stasko, James Eagan, and the whole of the CHI community. Also, our moms.

REFERENCES

1. Bruls, M., Huizing, K. and Wijk, J. J. v. (2000). *Squarified Treemaps*. Proc. of VisSym '00 (May 2931, Amsterdam, The Netherlands), Springer Wien New York, 33-42.
2. Graham, M., Kennedy, J. B. and Hand, C. *The Challenge of Visualising Multiple Overlapping Classification Hierarchies*. Proc. of User Interfaces to Data Intensive Systems (UIDIS '99) (Edinburgh, UK, September 5-6, 1999), IEEE Computer Society Press, 42-5
3. J. Helfman, (1995) *Dotplot Patterns: a Literal Look at Pattern Languages* , TAPOS, 2(1):31-1,1995
4. Huang, M. L., Eades, P. and Wang, J., *On-line Animated Visualization of Huge Graphs using a Modified Spring Algorithm.* Journal of Visual languages and Computing, 1998, 9 (6): 623-645.
5. Johnson, B. and Shneiderman, B., (1991) *Treemaps: A SpaceFilling approach to the visualization of hierarchical information structures*, in Proc. IEEE Visualization '91, pp. 284-291, San Diego, IEEE Computer Society Press
6. John Lamping and Ramana Rao.(1994) *Laying out and visualizing large trees using a hyperbolic space* . In Proceedings of UIST'94, pages 13--14, 1994
7. Maizel J., and Lenk, R., (1981) *Enhanced graphic matrix analysis of nucleic acid and protein sequences.* Proceedings of the National Academy of Science, Genetics, USA, volume 78, pages 7665–7669,
8. Munzner, T., 1997, *H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space* , IEEE Symposium on Information Visualization
9. Pustell, J., and Kafatos, F. (1982) *A high speed, high capacity homology matrix: Zooming through sv40 and polyoma*. Nucleic Acids Research, 10(15): 4765–4782
10. Robertson, G. G., Mackinlay, J. D., and Card, S. K. " *Cone trees: Animated 3D visualizations of hierarchical information*." Proc. of CHI '97, New Orleans, LA, 189-194.
11. Stasko, John, Catrambone, Richard, Guzdial, Mark and McDonald, Kevin, (2000) *An Evaluation of Space-Filling Information Visualizations for Depicting Hierarchical Structures,* International Journal of Human-Computer Studies, Vol. 53, No. 5, November 2000, pp. 663-694.
12. van Wijk, J. J. and van de Wetering, H. (1999). *Cushion Treemaps: Visualization of Hierarchical Information.* Proc. of IEEE InfoVis '99 (October 25-26, San Francisco, California, USA), IEEE Computer Society Press, 73-78.
13. Williamson,C. and Shneiderman B. (1992) *The Dynamic HomeFinder: Evaluating Dynamic Queries in a Real-Estate Information Exploration System.* In Proc of ACM SIGIR 92 (June 1992), 338 -- 346.

14. Wittenburg ,K.and Sigman ,E., (1997) *Visual Focusing and Transition Techniques in a Treeviewer for Web Information Access* , in Proc. Visual Languages '97, pp. 20-27, Capri, Italy, Sept 23-26, 1997. IEEE Computer Society Press.

15. Wittenburg ,K.and Sigman ,E., (1997) *Visual Focusing and Transition Techniques in a System for Web Information Access* , in Proc. Visual Languages '97, pp. 20-27, Capri, Italy, Sept 23-26, 1997. IEEE Computer Society Press.