Tree Flow: Displaying Structural Changes to Hierarchies

Drew Harry and Martin Wattenberg
Olin College, IBM Research

ABSTRACT
We describe Tree Flow, a new technique for visualizing changes to the structure of a hierarchy, and discuss a prototype tool that implements this method. Roughly speaking, Tree Flow works by creating a timeline in which the x-axis corresponds to date information, and the y-axis corresponds to the depth-first ordering of data in a time-varying tree structure. We present several examples of real-world data presented in this format and describe how the visualizations reveal important features of the data.

CR Categories and Subject Descriptors: Graph and Hierarchy Visualization, Applications of Visualization, Time-Varying Data, Text and Document Visualization
Additional Keywords: Organizational Chart

1 INTRODUCTION
For hundreds of years people have been concerned with displaying changes in data over time (Tufts, 1986). While numerical time series may be the most studied examples, designers have long recognized the need to show the history of non-numerical structures as well. For example, as biologists first explored the theory of evolution they drew by hand sophisticated and beautiful diagrams to portray the appearance and disappearance of species (Hanrahan, 2001).

Today an increasing amount of data is available in highly structured formats, such as XML, and it is common to keep histories of such structured artifacts. As a result, there is an opportunity and need to find ways to visualize these histories. In this paper we discuss the particular case of visualizing the history of an object with a tree structure, such as a company organization chart or a package of software libraries. Since hierarchically-structured data is so common—e.g., all XML documents—we believe this is a broadly useful area of inquiry.

One common technique in visualizing changes to graph structures is to use animation; see for example work on the social network visualizations (Mutton, 2004). Although compelling, animations are often not amenable to careful analysis and can make it difficult to consider two different time periods simultaneously—in other words, they violate Tufte’s principle that comparisons should be made across space, not time. (Tufts, 1986)

An approach that avoids the problems of animation, used often outside the field of graph visualization, is a timeline — in essence a diagram in which the x-coordinate of each item corresponds to its timestamp. While generally easy to read, timeline displays present a significant challenge to the visualization designer since one degree of freedom has been spent on conveying the passage of time. The key questions for a timeline display are: how are items ordered on the y-axis, and how are different items visually connected?

Within these constraints there turns out to be a surprisingly rich design space. One example of a complex timeline is the ThemeRiver display (Havre et al., 2002), which connects items using a carefully constructed set of curves. Another example is History Flow (Viegas et al, 2004), in which the items are passages of text in a changing document.

In this paper we introduce Tree Flows, timelines that use a depth-first ordering of nodes in a tree to define item y-coordinates, and connect corresponding items at different time periods in lines. Tree Flows can be considered a computational extension of some of the manual techniques used by biologists a century ago to depict evolution; unlike those diagrams, however, Tree Flows can show complex patterns of rearrangements of nodes in which parent-child relations can change over time.

In Section 2 we describe the simple construction that creates Tree Flows, and in Section 3 we provide examples of Tree Flow diagrams applied to two data sets: the organizational chart of a large company, and the package structure of the standard Java libraries. Section 4 discusses technical implementation considerations as well as issues in scaling to very large data sets.

2 CONSTRUCTING TREE FLOW DIAGRAMS

2.1 The basic construction
Consider a set of items, arranged in a tree structure that varies over time. For example, in a business the items might be employees and the trees might be a set of snapshots of the organizational chart of a company (that is, data on who reports to whom) taken at different times. The items may appear and disappear from the trees, corresponding to employees joining and leaving the company; and they may also change their parent/child relationships, as employees change managers.

A Tree Flow is a timeline depicting these changes. To construct a Tree Flow, we work as follows. For each version of the tree structure, we arrange the nodes in the tree vertically according to a depth-first ordering. The x-coordinate of each set of nodes is determined by the timestamp of the tree version. Corresponding

Figure 1 - Example tree states.
Figure 2 – Schematic results of the Tree Flow diagram process.

Nodes in successive versions are then connected by lines, much as in history flow diagram corresponding passages of text care connected. The technique is also reminiscent of the TreeJuxtaposer visualization in [Munzner et al, 2003]. One might say that Tree Flow bears a relation to TreeJuxtaposer that is similar the relation History Flow bears to standard two-file visual diffs.

To illustrate the construction of a Tree Flow diagram, we will consider the four example trees shown in Figure 1. From those states we will first draw a schematic to clearly illustrate the conversion process (Figure 2) and finally show the output of the Tree Flow algorithm.

Depth first ordering preserves some of the structural elements of the tree, while losing others. The ordering ensures that all the descendents of a top level node are adjacent. To bring attention to these groups, children of the root element who are themselves parents of other elements are grouped with all their descendents, and drawn with a gap between them and the previous line along the y-axis. This is analogous to the grouping of text into paragraphs, and the application of space before. Thus at the top of every group is an element that is a direct descendent of the root element. The grouping in the depth first listing and the addition of spacing can be seen in the vertical node listings in Figure 2.

After the elements are arranged at each time step, lines are drawn connecting each element. This makes visible the fundamental processes in the evolution of a tree – node creation, node movement, and node deletion.

Node addition can be seen in state 2, where we see the creation of nodes I and J. In viewing this schematic, it is important to note that not all movement in the y-axis indicates change. For example, the line for node D moves vertically to make room for node I, even though it hasn’t moved in the tree. Meaningful movement is coded using different colors, as discussed in section 2.2. In Figure 2, node creation is shown as a green box around the node. In state 3, we can observe nod movement; node D’s parent has changed from node B to node C. Because of this change, both nodes D and J must move. Both lines are colored to show that this movement represents actual movement of those nodes within the tree.

There are two ways in which an element’s position in the tree can change; either the parent of the element itself changes, or the parent of an ancestor of the element changes. Visually distinguishing between these movement types proved visually distracted – in Tree Flow, both types of movement are orange. The specific type of movement can usually be inferred from context.

The schematic Tree Flow in figure 2 doesn’t scale well. To create the actual diagrams, the labels at each tree state are not drawn, and the lines representing each node are drawn tightly spaced. It is these units that we use to make a Tree Flow schematic diagram in figure 2 is shown in figure 3.

2.2 Overlying data

The construction above lays out the basic geometry of a Tree Flow diagram. Here we discuss displaying additional dimensions of data using color, line thickness, and other visual attributes.

Because the process of converting the trees to a one-dimensional structure loses the hierarchical nature of the data, a major design goal was to represent the original hierarchy using a combination of white space, line thickness and color.

To separate sections of the tree at depths lower than two, the lines are colored according to their depth in the tree. The root element in the tree is dark blue, and successively deeper elements in the tree are lighter blues. This creates a striation effect, in which a deep but thin structure can be easily recognized as different from a shallow but wide structure. This distinction is discussed further in section 3.2.

In many trees, elements are free to enter and leave the tree, not just be rearranged within it. To show entry into the tree, lines are drawn with a green gradient, changing into their depth-appropriate color. When an element leaves the tree, it is drawn as a gradient changing from its depth-appropriate color turning into red. To accentuate the striation, nodes of depth 1 are drawn with thicker lines.

3 Examples

The examples in Section 2 used mocked-up data. In this section we describe two applications to real data. The first example shows how Tree Flow may be used to understand the history of a large software project. The second example, in which we apply Tree Flow to an evolving organizational chart of a large software company, is oriented toward understanding change in business structure.

3.1 Java Package Structure

One natural source of changing tree structures is software packages. In a large software project, class libraries typically evolve over time, and understanding this evolution—part of what Booch refers to as “software archaeology”—is considered an important problem (see Booch, 2002; Hunt and Thomas, 2002).

Tree Flow diagrams are a natural way to track such changes.

To show how this tool can be used to understand an evolving codebase, we chose the standard libraries that ship with the Java runtime. These libraries may be familiar to many readers, and in scope are representative of a large software project. We consider the history of these libraries from Java version 1.0 to the most recent version, 1.5 (sometimes referred to as 5.0), for a total of six snapshots.

The standard Java classes are arranged into units called packages. It is these units that we use to make a Tree Flow
diagram. Package hierarchies in Java are delimited by periods, so the package java.util.regex shows that the package java contains the package util, which in turn contains the package regex. Packages containment typically means a sort of specialization: for instance, the java.awt.image package (meant for image processing) is contained within the java.awt package (meant for general graphics and user interface code). The highest-level division is into two subsets, java and javax. The java package contains the core language classes, while javax contains a more recent GUI toolkit (Swing) and a variety of other utility classes.

Sun maintains archives of the documentation packages for all past versions of Java. From these archives, we built trees representing the package structure at each step in Java's history. The Java versions included in the data are version 1.0, 1.1.8, 1.2.2, 1.3.1, 1.4.2, and 1.5.0. These were selected because they were stable versions of major releases.

Using this data, we generated TreeFlow diagrams representing the evolution of the package hierarchy over time. Because this tree grows monotonically (that is, nodes can be added, but are never deleted) we do not use the gradient coloring to distinguish between arriving and leaving nodes. The diagram for all packages beneath the java package within the tree is shown in figure 4. Within this tree there is no movement, so the visualization shows when and where growth occurred in the package hierarchy. In general, the history of growth in java is one of filling in holes. Many of what would become the major packages in later versions of Java existed in the 1.0 release, though there was very little specialization – the first release consisted of only 7 packages. Java's most expansive period of growth was in its second release. Many more top-level packages were created (eg java.sql, java.text, java.rmi, java.math) and additional specialization was provided to older packages (java.awt.event, java.util.zip). Further expansion occurs in 1.2.2, after which point the hierarchy becomes largely stable. This is clear from the solid and smooth flows of the lines of the visualization.

The javax package, first created in version 1.2.2, tells a different story. Created after the explosive expansion that occurred in the java package in 1.1.8, the javax structure is generally more stable. The swing hierarchy, the original motivation for the creation of the javax tree, has remained almost completely unchanged. Meanwhile, significant additions to javax came in the form of additions in top-level packages like javax.naming, javax.net, javax.print, javax.imageio, etc. It is clear that this tree has become something of a dumping ground (especially in version 1.5.0) for major new groups of packages that Sun feels don't belong in the java hierarchy. Considering both figures 4 and 5 together, it is clear that the majority of development has shifted over time from packages in the java tree to those within the javax tree.

3.2 Organizational Change

Our second example is a visualization of organizational change. We were given access to data on the reporting structure of thousands of employees at a large corporation, over the course of a five-month period. At a given moment, this structure (often referred to as an "org chart") forms a tree: each node of the tree is an employee, the "parent" of a node is the employee's supervisor, and the "children" are the direct reports of the employee. Organizational charts are extremely useful structures, but their size and complexity can make them difficult to read; these difficulties have inspired visualizations such as the SpaceTree. (Grosjean et al, 2002)

Visualizations like SpaceTree are static, showing a snapshot of an organization at a particular point in time. But the org chart is hardly a static structure, and its evolution displays considerable complexity. Employees enter and leave the company, change managers, and can be promoted or demoted. Understanding these changes is important in a number of ways. At a most basic level, high-level executives or business transformation consultants need to know what is going on in the company, and it may be difficult for them to get an overview of organizational change occurring several levels below them in the corporate hierarchy. More specifically, there may be patterns of organizational change which are useful indicators of problems or successes. A unit that has experienced significant turnover and shifts in management may be in trouble, for instance.

There are a few changes in settings between these examples and the previous case. Lines are significantly thinner in these diagrams. The organization charts being considered have an order of magnitude more nodes, and so nodes at depth two or lower are drawn very thinly, while higher level nodes are still drawn at a reasonable thickness. The x-axis is also much more compressed to fit many more time steps worth of data about this organization. Also, it is important to remember that arrival and departure in these diagrams is context sensitive – arrivals are considered to be employees who were transferred into this department, and departures represent leaving the department, not the entire company.

Figure 6 shows one branch of the Tree Flow diagram for the organizational chart data. The arrival, transfer, and leaving of employees is visible. We can see that this is a fairly stable department. The two biggest and most noticeable features are the additions of entire sub-departments, once in March and again in May. After the first transfer (in March) we can see that many more employees were subsequently transferred out of the sub-department than is normal for this department. We can also see how a direct report to the department manager left, and was very quickly replaced by another employee from within the department. Because this graph takes place during the beginning of a summer, we can infer that this department was not undergoing significant growth, but was instead taking on many summer interns.

The activity in figure 7 is significantly more complicated, and represents a wider range of types of activities that can be made visible using Tree Flow. Around March 25th, two sub-departments undergo a process in which hierarchy is created where there was none before. By watching how the color pattern changes from a solid blue to a lighter blue punctuated with darker lines, we can see that managers have been created in this group, adding another layer of hierarchy. It might also be significant, in this case, that the promotions happened from within the sub-department – no outside managers were brought in.

Starting June 8th we see two important processes – the transfer of an entire sub-department, and a change in management of a sub-department. The process of changing managers in this particular company is fascinating and complex. The changes don't take place all at once, but are phased in over the course of a few weeks. First, a new sub-department manager is transferred in, and one of the old managers assigned to him. About a week later, the other main manager of the group is transferred, and finally, another week later, the original sub-department manager is transferred out. The intricacies of this process are made clear, and could reveal inefficiency or confusion about the organizational status of that sub-department. Indeed, when we showed these charts to people involved in this organization, they quickly began speculating about the implications of the changes they saw, and relating them to events in the company's history. This behavior led one researcher to term the prototype "a machine for generating gossip."
4 TECHNICAL CONSIDERATIONS

The implementation of Tree Flow which produced the images in Section 3 is written in Java 1.4, using the standard Java2D drawing package. Elements of the Piccolo architecture (Bederson et al, 2004) were used to provide easy zooming and panning capabilities, as well as to build a foundation for potentially more interactive visualizations. The tree data structures are maintained using the Java Universal Network Graph (JUNG) package. (O’Madadhain et al, 2005)

5 CONCLUSIONS AND FUTURE DIRECTIONS

We have presented Tree Flow diagrams, a technique for displaying time-varying hierarchical structures. Tree Flow diagrams have the potential to convey a complex sequence of changes to a tree structure, including addition, deletion, and movement of nodes and subtrees. We have shown above how this technique can make sense of changes to an organizational chart and how it can reveal trends in the growth of a software library.

There are several directions for future research in this area. The most pressing direction is developing methods of interaction with complex Tree Flow timelines. Navigating through a dense diagram can be difficult, and it might be beneficial to use some sort of focus and context technique. Finding ways to filter the data and perform dynamic queries—perhaps looking at particular subtrees, or searching for particular patterns of change—is another important topic.

We have also found that specific aspects of the data lend themselves to different configurations of the visualization. Frequency of activity, the amount of available time steps, and the depth of the tree all seem to have an effect on the best way to communicate the underlying patterns in the data. There is room for exploring these differences in tree activity profiles, perhaps in the direction of characterizing different sorts of behavior. It would also be valuable to apply this tool to a specific case study, for example in the field of organizational behavior, to reveal previously inaccessible patterns, as well as address the specific needs of researchers who frequently work with this kind of data.

More generally, the question of displaying changes to tree structures is part of a broad class of visualization problems. As disk space becomes cheaper, we will be presented with more opportunities to mine data based on archives of structured objects. Are there similar timeline-based visualization techniques for displaying changes to an arbitrary graph structure, for example?

Acknowledgments

Thanks to Steve Farrell for providing data sets. Jonathan Feinberg contributed significantly to the architecture of the prototype Tree Flow implementation. Conversations with Kushal Dave and Casey Dugan were also extremely helpful.

REFERENCES


Figure 4 – The evolution of the java package hierarchy, as visualized using Tree Flow. Java versions 1.0.0 through 1.5.0 shown.

Figure 5 – The evolution of the javax package hierarchy, as visualized using Tree Flow. Javax was added in version 1.2.2, so only versions 12.2 to 1.5.0 are shown here.
Figure 6 – Example Tree Flow visualization for a branch of a large company. The addition, removal, and transfer of employees is visible.

Figure 7 – Example Tree Flow visualization for a more complicated department. In this figure, we see complex reorganizations of the department hierarchy.