CactusTree: A Tree Drawing Approach for Hierarchical Edge Bundling

Tommy Dang*
Department of Computer Science
Texas Tech University

Angus Forbes†
Department of Computer Science
University of Illinois at Chicago

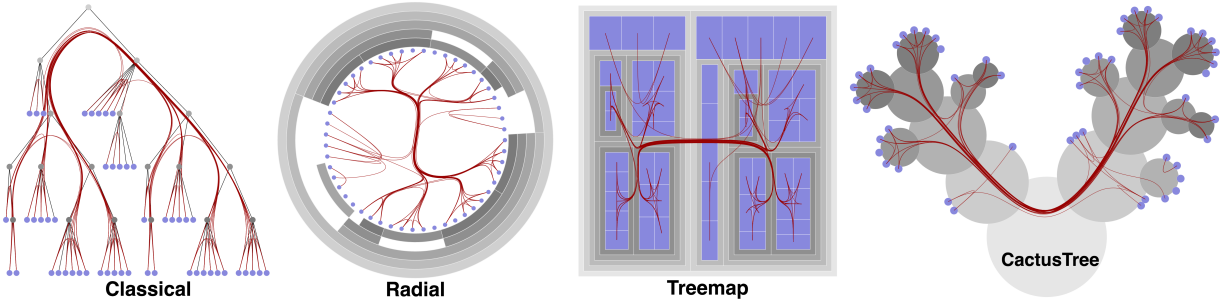

Figure 1: Examples of hierarchical edge bundling on a hierarchical dataset using different tree layouts. Leaf nodes are in blue. Bundled links are in red.

Abstract

This paper introduces *CactusTree*, a novel visualization technique for representing hierarchical datasets. We introduce details about the construction of *CactusTrees* and describe how they can be used to represent nested data and relationships between elements in the data. We explain how our design decisions were informed by tasks common to a range of scientific domains. A key contribution of this article is the introduction of descriptive features that can be used to characterize trees in terms of their structural and connective qualities.

Keywords: Hierarchical edge bundling, tree layouts, taxonomies.

1 Introduction

Trees are one of the most fundamental data types. However, existing tree layout techniques are not always adequate for many real-world datasets. Layouts that aim to visually simplify complex trees can also make it difficult to perform common tasks, such as finding nodes or subtrees with particular characteristics. In addition to facilitating reasoning about hierarchical relationships, many application domains require the presentation of non-hierarchical relations between data items [14]. For example, in the domains of molelcular biology, researchers analyze intracellular signaling pathways that can be composed of nested sets of biomolecules. It is also important to show how particular biomolecules influence or are influenced by others [8]. Similarly, in the domain of biodiverstiy informatics, the hierarchical structure of a phylogenic tree can usefully be overlaid with additional information that connects nodes in the tree in order to show, for instance, properties of a food web [26]. Many other examples can be found in real-world datasets.

Rather than investigating the bundling techniques themselves, as most existing approaches [1, 16, 21] have attempted, we explore the effective use of tree layouts to support hierarchical structure recognition and to minimize ambiguity introduced by bundling cross-edges (also called non-hierarchical connections/links in this paper). Based on our analysis of important features of tree layouts, we propose a new tree layout— *CactusTree*— for visualizing the structure and connectivity of nested trees.

*e-mail: tommy.dang@ttu.edu
†e-mail: aforbes@uic.edu

2 Related Work

This paper introduces a new tree visualization technique with the specific goal of untangling overlaid bundles of intersecting edges. That is, we aim to address the problem of collinearity, discussed in the original paper on hierarchical edge bundling [14]. Here, we discuss relevant work related to tree visualization and edge bundling.

2.1 Tree Visualization Techniques

A wide range of tree layouts have been introduced as general techniques to encode hierarchical data and in support of specific tasks. Schulz [27] maintains *Treevis.net*, a comprehensive website that describes of a large number of tree layouts (292 in total as of February 2017) gathered from conference proceedings and journal articles. Each of these layouts have advantages and disadvantages when used for particular tasks.

A TreeMap [29] is a space-filling technique that maps a hierarchical dataset onto a rectangular region. The effective use of space enables comparison of attributes of leaf nodes such as size and color coding, and therefore helps to highlight patterns and outliers in large hierarchies. Clever variations of TreeMaps, including MartketMap [32] and Squarified TreeMaps [5], ensure low aspect ratio rectangles (where most rectangles are nearly square) replacing the "slice-and-dice" method used in the original TreeMap layout. Cushion TreeMaps [31] add intuitive shading to help improve the perception of hierarchical structure. Arbitrary polygons [2] and circular glyphs [11] can be used instead of rectangles to create more visually attractive and useful layouts. Despite their popularity, TreeMaps can be difficult to decipher in some situations, such as when used to represent deeply nested hierarchies. ArcTrees [24] is one of the early efforts to overlay non-hierarchical links onto TreeMaps.

Kruskal and Landwehr introduce Icicle Plots [18], which encode hierarchical data by stacking child rectangles directly on top of parent nodes. This makes it easier to see the hierarchical structure, but also assigns valuable screen space in assigning large areas to intermediate nodes. When using Icicle Plots to represent dense datasets that contain a large number of leaf nodes, the leaf nodes can be pushed close together, making them hard to see.

Beck et al. [3] introduce a generalization of Bosman's Pythagoras Trees to visualize arbitrarily branching hierarchical structures. Each

node in the hierarchy is represented as a rectangle which is sized based on the collective size of its children. Colors are used to encode the depth of nodes inside the hierarchy. This tree structure can also take a variety of forms that are created by adjusting different parameters such as length, width, order, and color. While this technique has an appealing aesthetic quality, the gaps between areas representing hierarchical structure may interfere with how users perceive non-hierarchical connections between nodes.

Kobsa [17] describes a study to compare several well-known information visualization systems for tree hierarchies in a between-subjects experiment. The study showed a significant difference in completion times and correctness between structure-related versus attribute-related tasks on various tree layouts. McGuffin and Robert [22] presents an in-depth survey of tree layouts that introduces a range of metrics to define the information density of different tree layouts. These metrics provide design guidelines for the use of layouts for certain tasks, such as maximizing space-efficiency and supporting labeling.

2.2 Hierarchical Edge Bundling

Holten introduces hierarchical edge bundling (hereafter, HEB) [14], a technique to group links between adjacent edges by routing them through parent nodes in order to re-enforce the hierarchical structure of the data. HEB is a widely used technique that is used for a range of applications, and has also been extended for particular contexts. For instance, to make it possible to "bundle" the edges without requiring a control mesh [6] or hierarchy, Holten and van Wijk [15] use a self-organizing algorithm. In this approach, edges are modeled as flexible springs that can attract each other while node positions remain fixed. Various factors, such as translucency [21], color [16], and depth effect, can be considered to aid in the perception of bundles. However, HEB is most often used as originally described [14], and is readily available for some layouts via visualization toolkits, such as D3.js [4].

To the best of our knowledge, somewhat surprisingly, HEB has not been evaluated systematically. In a survey paper on edge bundling techniques, Zhou et al. [34] summarize some studies of HEB, which tend to indicate user preference for visualizations that use HEB in comparison to those that do not. Xu et al. [33], while not explicitly focused on HEB, examine visualizations that use varying degrees of curvature, finding that links with high-curvature can adversely affect how well users interpret data. Bach et al. [1] investigate Confluent Drawings [9], a technique for bundling edges in node-link diagrams based on network connectivity. The authors also present a user study that compares edge-compression techniques, including Confluent Drawings, power graphs [10], ordered bundles [25], and edge bundling.

McGee and Dingliana [21] perform user experiments to evaluate the impact of bundling on user performance on different tasks using a set of randomly generated undirected compound graphs with varying sizes and edge densities. In their study, graphs are presented with a range of different levels of edge bundling using a simple balloon tree layout. Within the context of their experimental setup, their results indicate that bundling can actually hinder users in path tracing tasks, both in terms of accuracy and completion time.

3 Overview of Visualization Tasks

There are many tasks related to visualizing compound graphs in a range of scientific domains, including those that involve biological pathways [20], ontology alignment [23], and taxonomic classifications [7]. Through in-depth discussions with systems biologists, taxonomists, and ontology researchers, we identified two primary tasks important for visual exploration of hierarchical datasets: characterizing hierarchical structures and identifying connections between nodes in the hierarchy [19].

T1: Effectively characterize hierarchical structure—Biological pathways are usually composed of a number of sub-pathways, themselves containing other sub-pathways and biochemical reactions between elements within them. It is not uncommon for this nested structure to have a depth of more than ten levels [12, 30]. More importantly, these classifications (hierarchies) may change from year to year as new discoveries or interpretations are made [13].
T2: Minimize ambiguity introduced by edge bundling—HEB trades details for overview. In other words, following an edge from its source to its target can lead to the perception of incorrect connectivity if edges are not clearly separated within the bundles [1]. An ideal tree visualization technique to best support HEB should minimize this loss.

3.1 Classifying Layouts to Support Visualization Tasks

Schulz [27] classifies tree layouts in terms of three main features, based on their structural layout: *dimensionality* (2D or 3D), *representation* (explicit or implicit), and *alignment* (axis-parallel, radial, and free). Schulz et al. [28] further propose a generic tree layout pipeline to produce both implicit and explicit tree layouts. In this paper, we focus on layouts that support our primary tasks when applied to complex, real-world datasets. We look only at 2D graphical representations of tree structures (3D tree layouts are less popular and evaluating 3D tree layouts would require a more extensive investigation of interactions, such as rotating, panning, and zooming).

Node-link vs. containment vs. stacking

This describes the encoding of a parent-child relationship by either: (a) drawing a link (*node-link*, such as in Classic trees or Radial trees); (b) nesting children within the parent (*containment*, such as TreeMaps or Balloon layouts); or (c) having a spatial area of the child abut its parent (*stacking*, such as Icicle plots).

Root-centric vs. parent-centric

In *root-centric* layouts, all layout operations are made w.r.t. the tree's root. In *parent-centric* layouts, all layout operations are made w.r.t. a node's parent [28]. Fig. 2 shows an example of *root-centric* and *parent-centric* layouts. Icicle and *CactusTree* drawing strategies are very similar: child nodes are stacked directly on their parent nodes. However, the width of leaf nodes in Icicle plot (left) is equally divided based on root width, while child nodes in *CactusTree* are stacked along the half-arcs of their parent center with an assigned orientation (the *alpha* input parameter in Algorithm 1), and thus produces a *parent-centric* layout.

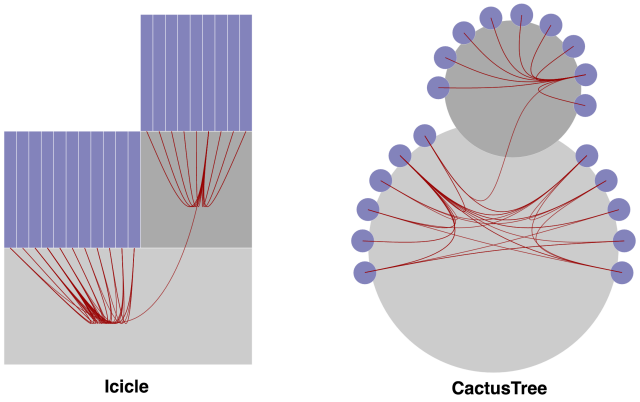

Figure 2: We hypothesize that bundled cross-edges in a *parent-centric* layout (right) are more discernible than a *root-centric* layout (left). The hierarchy in this example is the *animate* subpackage structure within the *flare* software project. Each blue leaf node represents a source code file. Red links depict how these files refer each other.

4 THE *CactusTree* LAYOUT

We designed the *CactusTree* layout based on the following two observations:

• *CactusTree* is a fractal-based technique which recursively stacks child nodes on top of their parent. Fractal appearance trees are aesthetically appealing [3] and similar to natural structures such as trees, leaves, ferns, clouds, coastlines, or mountains. Therefore, they characterize hierarchical structures effectively and are easy to remember, supporting **T1**.

• Longer paths with hairpin curves require more effort to trace. For example, in Fig. 3(a) the straight black link is much easier to trace than the red link [21], which is again easier to trace than the blue link. Fig. 3(b) compares an example of HEB on a circular (e.g., *CactusTree*) layout with a linear (e.g, Icicle Plot) layout. In this simple example, the black parent node contains 6 child nodes evenly distributed circularly (red nodes) or linearly (blue nodes). The blue bundled link connecting two center neighboring nodes in the linear layout has a sharper turn compared to the red bundled link in the circular layout. In Fig. 3(c), we show the two neighboring nodes on the right; the blue link is not only sharper but also longer than the red link. Moreover, when we compare Fig. 3(b) and (c), the turning angles and the lengths of two red links are the same while they are both different for two blue links. Consequently, we prefer a circular layout rather than a linear layout for **T2**.

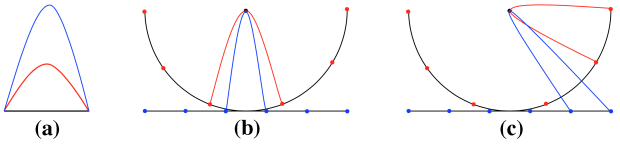

Figure 3: We hypothesize that shorter links, with less sharp turns require less effort to understand. Examples of HEB using a circular vs. linear arrangement: in (a) the blue link requires more effort to trace than red link since the eye has to travel further to verify the connection; in (b) the blue link has a sharper turn than red link; in (c) the blue link is not only sharper but also longer than the red link.

Based on our analysis of and hypotheses about the above tree layout qualities, we introduce *CactusTrees* to support the two primary analysis tasks discussed in Section 3. Since drawing explicit links between parent and child nodes may generate visual confusions with cross-edges, we use *stacking* (touching) to represent parent-child relationships in our proposed tree layout. Specifically, each intermediate node in *CactusTree* is represented as a circle. Child nodes are stacked along the circular half-arcs of their parent node. In other words, child node distribution is calculated w.r.t its parent location and orientation (*parent-centric* layout). Therefore, any duplicated subtrees have *stable* shapes regardless of its location within the tree. Moreover, the circular distributions of child nodes around their parents potentially allow *wider turns* when bundling cross-edges in the tree layout. We use shadings to indicate depth of a node within the nested structure. The darker the circle, the deeper it is in the hierarchy; leaf nodes are given a distinct color (blue).

The construction of *CactusTree* layouts is summarized in Algorithm 1. The algorithm first orders the child nodes of the current node by their weights. The total weight for a node is the sum of all immediate leaf nodes, each given a weight of 1, plus the weights of each of its subtrees (recursively calculated). We then call Algorithm 2 to produce a second ordering in which the maximum weight subtrees are put in the center of the list, while leaf nodes are distributed equally on both sides. Algorithm 2 simply adds an ordered node list into the middle of a new array list (initialized to be empty). Since maximum weight subtrees are in the center of the array list, we make sure that the tree grows upward. The radius of each node is computed based on its calculated weight (number of leaf nodes) in the current subtree using Algorithm 3.

Algorithm 1 *CactusTree* Layouts

```
procedure DRAWCactusTree(currentNode, x, y, alpha)
    Let childList be the list of children of the current node
    // Measure weight of the current tree, each leaf node weighs 1
    totalWeight = 0
    for each child in the childList do
        totalWeight += weight(child)
    // Draw a circle at (x,y) with radius returned from getRadius
    DrawCircle(x, y, getRadius(totalWeight))
    // Sort childList: leaf nodes first then larger subtrees
    orderedList = sortChildNodesByWeight(childList)
    // Order sibling nodes: larger subtrees are put in the middle
    centeredList = orderMaxInCenter(orderedList)
    for each child in the centeredList do
        alpha += (weight(child)/totalWeight)/2
        x2 = x+getRadius(weight(child))*cos(alpha)
        y2 = y+getRadius(weight(child))*sin(alpha)
        // Draw subtrees by calling Algorithm 1 recursively
        DrawCactusTree(child, x2, y2, alpha)
        alpha += (weight(child)/totalWeight)/2
```

Algorithm 2 Order sibling nodes: maximum weight in the center

```
procedure ORDERMAXINCENTER(orderedList)
    Let centeredList be an empty array list
    // Keep adding the ordered nodes to the middle of centeredList
    for each node in the orderedList do:
        centeredList.add(centeredList.size/2, node)
    return centeredList
```

Algorithm 3 Get radius of a node based on its weight

```
procedure GETRADIUS(weight)
    // Define scale factor between parent and child nodes
    var scaleFactor = 0.75
    return Math.pow(weight, scaleFactor)
```

5 CASE STUDY

CactusTree supports a range of interaction capabilities to help a user focus on a substructure of interest. Fig. 4 shows an example of zooming into a subtree (on the right) of the *flare* package hierarchy (on the left). Users can expand/collapse a branch of a tree by a simple click.

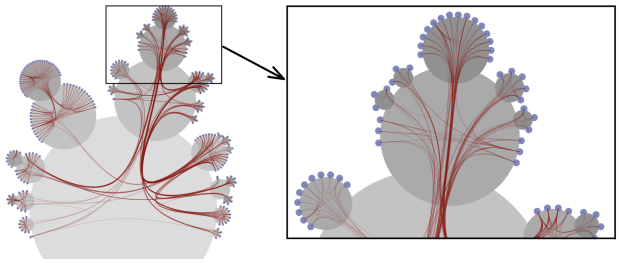

Figure 4: Zooming into a *CactusTree* to see details about substructures and internal connectivity.

CactusTree also supports brushing/selecting nodes and non-hierarchical links. Fig. 5(a) shows a subtree of the mammal hierarchy, called *carnivora*, which contains 63 meat-eating species. The data was downloaded from OneZoom [26] and overlaid with the

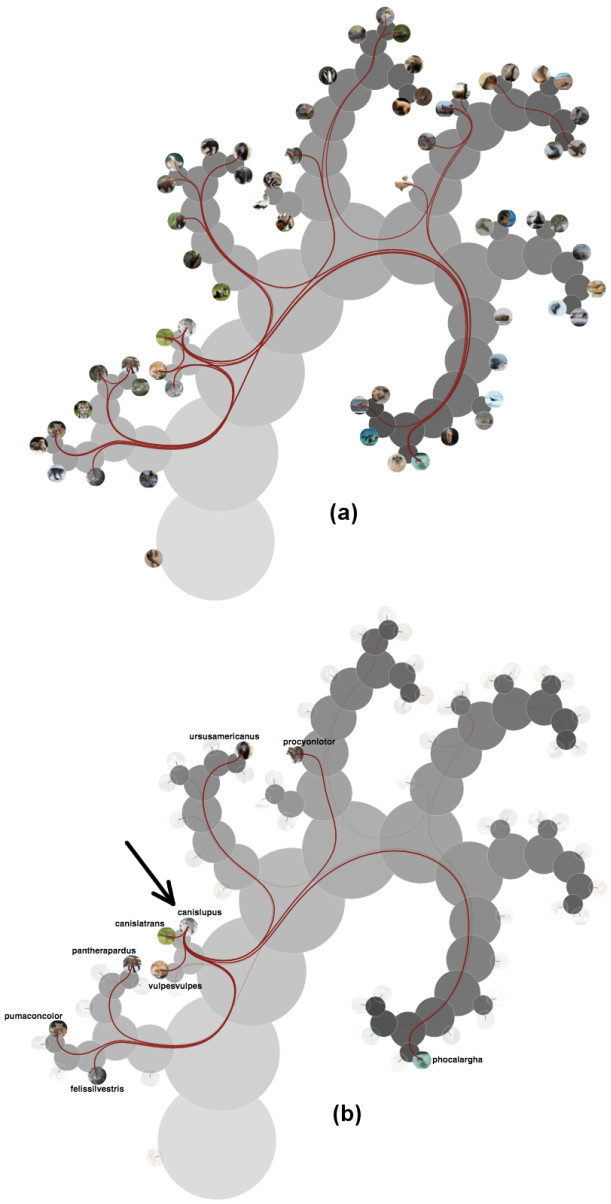

Figure 5: Visualizing *carnivora* hierarchy within the mammal evolutionary tree: (a) leaf nodes are displayed as images of species within the *carnivora* hierarchy and red links depict prey-predator relationships; (b) Selecting a leaf node *canis lupus* (gray wolf) highlights the direct species in its food chain.

prey-predator data provided by taxonomy experts. In this figure, we display the images of species within the *carnivora* food chain. The red links connect predators to their preys. Fig. 5(b) shows brushing a leaf node under the arrow, *canis lupus* (gray wolf). As depicted, gray wolf is an important species in the *carnivora* food chain. Notably, gray wolf eats its evolutionary sibling, *canis latrans* (coyote).

Since child nodes in *CactusTree* are stacked along the half-arcs of their parent, every parent node has a separated entry for bundled links to its child nodes (the entry is the center of the other half-arc). Consequently, we can visualize interconnectivities between multiple *CactusTrees* as depicted in the following Fig. 6 (which is much more intuitive than interconnecting other layouts, such as TreeMaps or Radial trees) . This example contains three biological pathways: Influenza Infection, HIV life cycle, and Signaling by ERBB2 (from left to right). Selecting the links between two pathways highlights how they are interconnected, for example we can display which biochemical reactions related to the causal relation between HIV and Influenza Infection.

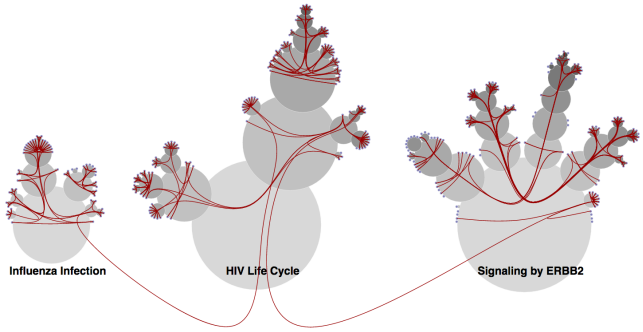

Figure 6: An example of multiple, interconnected trees using *CactusTrees* to represent biological pathways.

CactusTree is implemented in Javascript using the D3.js library. The demo, source code, and project documentation are available on our Github repository at `http://cactustrees.github.io`.

6 Conclusions and Future Work

The paper introduces a new tree visualization technique that is geared specifically towards more effectively representing overlaid bundles of intersecting links between nodes in the tree. We believe that this is a noteworthy approach toward solving the problem of collinearity that was observed in the original paper on hierarchical edge bundling (see Fig. 17 in Holten [14]). Rather than investigating the bundling techniques themselves, as most existing approaches have attempted, here we have explored the potential of modifying to the underlying tree layout.

Overlapping nodes/branches can occur in *CactusTree* for very complex hierarchies. In this case, we can increase the scale factor (in Algorithm 3) between a parent node and its children to avoid collisions. An example of the mammal hierarchy (41 levels of depth) with different scale factors is depicted in Figure 7. At $ScaleFactor = 0.75$ (the right most tree), no collisions is detected.

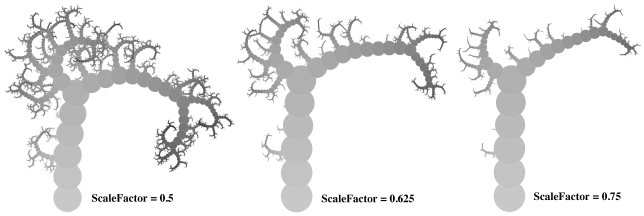

Figure 7: *CactusTree* for the mammal hierarchy with different scales factor in computing node size.

For future work we plan to conduct more extensive studies of HEB on different tree layouts and to look at more involved tasks on more complicated data. For instance, we want to explicitly examine a user's understanding of high level inter-cluster connectivity trends by asking the user to identify which cluster/parent node is most strongly connected to a selected cluster/parent node. We also plan to examine user understanding of low level intra-cluster connectivity trends by testing how well a user can identify the connectivity within a cluster/parent node. Understanding how interactions, such as rotating, panning, and zooming, support these tasks is also an interesting future investigation.

REFERENCES

[1] B. Bach, N. H. Riche, C. Hurter, K. Marriott, and T. Dwyer. Towards unambiguous edge bundling: Investigating confluent drawings for network visualization. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):541–550, 2017.

[2] M. Balzer, O. Deussen, and C. Lewerentz. Voronoi treemaps for the visualization of software metrics. In *Proceedings of the ACM Symposium on Software Visualization*, pp. 165–172, 2005.

[3] F. Beck, M. Burch, T. Munz, L. Di Silvestro, and D. Weiskopf. Generalized pythagoras trees for visualizing hierarchies. In *Proceedings of the International Conference on Information Visualization Theory and Application*, pp. 17–28, 2014.

[4] M. Bostock, V. Ogievetsky, and J. Heer. D^3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[5] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *In Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pp. 33–42, 1999.

[6] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 14(6):1277–1284, 2008.

[7] T. Dang, N. Franz, B. Ludäscher, and A. Forbes. ProvenanceMatrix: A visualization tool for multi-taxonomy alignments. In *Proceedings of the ISWC Workshop on Visualizations and User Interfaces for Ontologies and Linked Data*, vol. 1456, p. 13, 2015.

[8] T. Dang, P. Murray, J. Aurisano, and A. G. Forbes. Reactionflow: An interactive visualization tool for causality analysis in biological pathways. *BioVis 2015*, In press, 2015.

[9] M. Dickerson, D. Eppstein, M. T. Goodrich, and J. Y. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. In G. Liotta, ed., *11th International Symposium on Graph Drawing, Revised Papers*, pp. 1–12. Springer, 2004.

[10] T. Dwyer, N. H. Riche, K. Marriott, and C. Mears. Edge compression techniques for visualization of dense directed graphs. *IEEE Transactions on Visualization Computer Graphics*, 19(12):2596–2605, 2013. doi: doi.ieeecomputersociety.org/10.1109/TVCG.2013.151

[11] F. Fischer, J. Fuchs, and F. Mansmann. ClockMap: Enhancing circular treemaps with temporal glyphs for time-series data. In *Proceedings of EuroVis, Short Papers*, pp. 97–101, 2012.

[12] N. M. Franz, N. M. Pier, D. M. Reeder, M. Chen, S. Yu, P. Kianmajd, S. Bowers, and B. Ludäscher. Taxonomic provenance: Two influential primate classifications logically aligned. *ArXiv e-prints*, Dec. 2014.

[13] C. E. Hinchliff, S. A. Smith, J. F. Allman, J. G. Burleigh, R. Chaudhary, and et al. Synthesis of phylogeny and taxonomy into a comprehensive tree of life. *Proceedings of the National Academy of Sciences*, 112(41):12764–12769, 2015.

[14] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, Sept. 2006.

[15] D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. In *Proceedings of the 11th Eurographics / IEEE - VGTC Conference on Visualization*, pp. 983–998, 2009.

[16] D. Holten and J. J. van Wijk. A user study on visualizing directed edges in graphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2299–2308, 2009.

[17] A. Kobsa. User experiments with tree visualization systems. In *Proceedings of the IEEE Symposium on Information Visualization*, INFOVIS '04, pp. 9–16. IEEE Computer Society, Washington, DC, USA, 2004. doi: 10.1109/INFOVIS.2004.70

[18] J. B. Kruskal and J. M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):pp. 162–168, 1983.

[19] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry. Task taxonomy for graph visualization. In *Proceedings of the 2006 AVI Workshop on BEyond Time and Errors: Novel Evaluation Methods for Information Visualization*, BELIV '06, pp. 1–5. ACM, New York, NY, USA, 2006. doi: 10.1145/1168149.1168168

[20] A. Lex, C. Partl, D. Kalkofen, M. Streit, S. Gratzl, A. M. Wassermann, D. Schmalstieg, and H. Pfister. Entourage: Visualizing relationships between biological pathways using contextual subsets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2536–2545, 2013.

[21] F. McGee and J. Dingliana. An empirical study on the impact of edge bundling on user comprehension of graphs. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pp. 620–627, 2012.

[22] M. J. McGuffin and J.-M. Robert. Quantifying the space-efficiency of 2d graphical representations of trees. *Information Visualization*, 9(2):115–140, June 2010.

[23] M. Nasir, O. Hoeber, and J. Evermann. Supporting ontology alignment tasks with edge bundling. In *Proceedings of the 13th International Conference on Knowledge Management and Knowledge Technologies*, pp. 11:1–8, 2013.

[24] P. Neumann, S. Schlechtweg, and S. Carpendale. ArcTrees: Visualizing relations in hierarchical data. In *Proceedings of the Seventh Joint Eurographics / IEEE VGTC Conference on Visualization*, EUROVIS'05, pp. 53–60. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2005. doi: 10.2312/VisSym/EuroVis05/053-060

[25] S. Pupyrev, L. Nachmanson, S. Bereg, and A. E. Holroyd. *Edge Routing with Ordered Bundles*, pp. 136–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi: 10.1007/978-3-642-25878-7_14

[26] J. Rosindell and L. J. Harmon. OneZoom: A fractal explorer for the Tree of Life. *PLoS Biology*, 10(10):e1001406, 2012.

[27] H.-J. Schulz. Treevis.net: A tree visualization reference. *Computer Graphics and Applications, IEEE*, 31(6):11–15, Nov 2011.

[28] H.-J. Schulz, Z. Akbar, and F. Maurer. A generative layout approach for rooted tree drawings. In *Visualization Symposium (PacificVis), 2013 IEEE Pacific*, pp. 225–232. IEEE, 2013.

[29] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on graphics (TOG)*, 11(1):92–99, 1992.

[30] L. Strömbäck and P. Lambrix. Representations of molecular pathways: An evaluation of SBML, PSI MI and BioPAX. *Bioinformatics*, 21(24):4401–4407, 2005.

[31] J. J. Van Wijk and H. van de Wetering. Cushion Treemaps: Visualization of hierarchical information. In *Proceedings of the 1999 IEEE Symposium on Information Visualization*, pp. 73–78, 1999.

[32] M. Wattenberg. Visualizing the stock market. In *CHI '99 Extended Abstracts on Human Factors in Computing Systems*, pp. 188–189, 1999.

[33] K. Xu, C. Rooney, P. Passmore, D.-H. Ham, and P. H. Nguyen. A user study on curved edges in graph visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2449–2456, 2012.

[34] H. Zhou, P. Xu, X. Yuan, and H. Qu. Edge bundling in information visualization. *Tsinghua Science and Technology*, 18(2):145–156, 2013.