Data Visualization (CSCI 627/490)

Trees

Dr. David Koop
Networks

• Network: nodes and edges connecting the nodes
• Formally, $G = (V,E)$ is a set of nodes $V$ and a set of edges $E$ where each edge connects two nodes.
• Nodes == items, edges connect items
• Both nodes and edges may have attributes

Arrangement of Networks and Trees

- **Node-Link Diagrams**
  - Connection Marks
  - ✓ NETWORKS ✓ TREES

- **Adjacency Matrix**
  - Derived Table
  - ✓ NETWORKS ✓ TREES

- **Enclosure**
  - Containment Marks
  - ✓ NETWORKS ✓ TREES

[Munzner (ill. Maguire), 2014]
Network Data Represented in Tables

Nodes

<table>
<thead>
<tr>
<th>ID</th>
<th>Atom</th>
<th>Electrons</th>
<th>Protons</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

Edges

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
<th>Bonds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Networks Need Layouts!

- Need to use spatial position when designing network visualizations
- Otherwise, nodes can **occlude** each other, links hard to distinguish
- How?
  - With bar charts, we could order using an attribute…
  - With networks, we want to be able to see connectivity and topology (not in the data usually)
- Possible metrics:
  - Edge crossings
  - Node overlaps
  - Total area
Layout Algorithms

[Force-Directed and CoLa, M. Bostock]
Bundling Strength

This is illustrated in figure 14, which shows visualizations using a balloon layout (node labels disabled) and a radial layout, respectively. Bundling reduces visual clutter, making it easier to perceive the high-level connectivity information while still being able to inspect the low-level relations that were responsible for the bundles by interactively manipulating the bundling strength.

Another aspect that was commented on was how the bundles gave an impression of the hierarchical organization of the data as well, thereby strengthening the visualization of the hierarchy. More specifically, most of the participants particularly valued the automatic lifting of the low-level relations that were responsible for the bundles to implicit relations between items at higher levels of the hierarchy.

In determining the usability of our technique, based on our own interaction with the visualizations and feedback gathered from participants, we contend that bundling should be used in conjunction with additional data sets and feedback regarding the resulting visualizations.

Although our main focus while developing hierarchical edge bundling was on the visualization itself, interaction is an important aspect of interacting with the visualizations. Based on our own insight and feedback, we believe that bundling could be used in conjunction with additional data sets and feedback regarding the resulting visualizations.
Adjacency Matrix

- Change network to tabular data and use a matrix representation
- Derived data: nodes are keys, edges are boolean values
- Task: lookup connections, find well-connected clusters
- Scalability: millions of edges
- Can encode edge weight, too

[Henry et al., 2007]
Structures from Adjacency Matrices

cliques

bicliques

clusters
Node-Link or Adjacency Matrix?

- Empirical study: For most tasks, node-link is better for small graphs and adjacency better for large graphs
- Multi-link paths are hard with adjacency matrices
- Immediate connectivity or neighbors are ok, estimating size (nodes & edges also ok)
- People tend to be more familiar with node-link diagrams
- Link density is a problem with node-link but not with adjacency matrices
Project

- Working through grading these to provide feedback
- Initial Feedback
  - Some tasks are not tasks
  - Some tasks are technically tasks but are phrased in terms of a visualization
  - Think about the question "Why would someone care?"
- Example: Is there a correlation between the season and types of storms in regions?
  - Who cares?
  - Why do they care?
  - Are there specific instances where we can see how people might use info?
Project

• Next steps:
  - Start thinking about the designs that help answer the questions
  - Tasks should drive your design
  - Different designs are great
    • Multiple views
    • Single view with details on demand
    • Interaction design (linked highlighting, navigation)
    • In general, don't force the user to make choices without first seeing an overview
Assignment 4
Trees

- Trees are directed acyclic networks
  - each edge has a direction: the origin is the parent, the destination is the child
  - cannot get back to a node after leaving it
- ...plus each node has at most one parent node
- A tree has a root (every other node hangs off it)
- Can consider enclosure in trees using parent-child relationships
Quantifying the Space-Efficiency of Tree Representations

A comparison of the space-efficiency of various 2D graphical representations of tree structures is presented. As part of the evaluation, a novel metric is introduced that quantifies the distribution of area across nodes in a tree representation, and that can be applied to all the forms in Figure 1. Some representations, such as node-link diagrams (Figure 1A) and layered representations (Figure 1B), from “classical” node-link diagrams to concentric and radial representations (Figures 1E and 1F), are more space-efficient than others. However, it is not clear initially if treemaps, or any other representation, will still be optimal with respect to such alternative metrics.

Clearly, it would be useful to have some way to quantitatively distinguish the four possibilities in Figure 2, e.g. in terms of their mean area exponent as well as the area they allocate to nodes and labels. Our analysis inspires a set of design guidelines as we turn our attention to putting these ideas into practice.

Acknowledgments

D. Koop, CSCI 627/490, Fall 2022

Michael J. McGuffin and Jean-Marc Robert

Canada, E-mail: michael.mcguffin@etsmtl.ca.

References

Node-Link Diagram

- Trees are graphs
- …but we have more structure
- Horizontal or vertical
- Idea 1: partition space for each node via recursion
- Idea 2: “Tidy” Drawing
  - Wetherell & Shannon: Don’t waste space (overlapping parent nodes is ok)
  - Reingold and Tilford: Keep symmetry, subtrees look similar

[WS Alg., Reingold and Tilford, 1981]
Reingold-Tilford Algorithm

- Recurse on left and right subtrees
- Shift subtree over as long as it doesn’t overlap
- Place parent centered above the subtrees
- Originally, only binary trees, extended by Walker

[Reingold and Tilford, 1981]
Icicle Plot

- Line marks
- Vertical position shows depth
- Horizontal position shows links and sibling order
- Scalability: 1 pixel leaves, but harder to label

[Bostock, 2011]
Radial Node-Link

• Use polar coordinates instead of rectilinear
• Same layout algorithms work (e.g. Reingold-Tilford)
• Benefit: space usage, labels
Sunburst

- Icicle plot in a radial layout
- Reading labels?
- Intuitive navigation

[Heer et al., 2012]
Indented Outline

• Like a filesystem tree
• Use horizontal position to show depth, vertical positions show sibling/order
Car/Truck Treemap

Truck Sales Slip, Tripping Up Chrysler

Over the past few years, Chrysler executives said they were following the lead of Toyota and Honda, focusing on vehicles that met the needs of their customers. But as American consumers turned away from large trucks and S.U.V.’s in 2006, Chrysler continued to churn out big vehicles, which are now sitting unsold at dealerships across the country.

Change in sales from 2005 to 2006

<table>
<thead>
<tr>
<th>Change</th>
<th>No. 2005 sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10%</td>
<td>Below</td>
</tr>
<tr>
<td>-2.5%</td>
<td>Below industry average</td>
</tr>
<tr>
<td>0</td>
<td>Sales</td>
</tr>
<tr>
<td>+10%</td>
<td>Above industry average</td>
</tr>
<tr>
<td>+100%</td>
<td>Above</td>
</tr>
</tbody>
</table>

Dodge Ram

Car/Truck Treemap

Trucks/Vans/S.U.V.’s 1.6 million
Cars 0.5 million
Pickups, minivans and S.U.V.’s made up 76 percent of Chrysler’s sales, which left it vulnerable when consumers shifted to cars.

General Motors

-8.7%
Trucks/vans/S.U.V.’s 2.5 million
Cars 1.6 million
G.M. introduced new versions of its large S.U.V.’s in late 2005, hoping they would bolster sales. Instead, sales of big vehicles were hurt when gas prices climbed. One of the few standouts was the Chevrolet HHR, new in 2005.

Toyota

+12.5%
Trucks/vans/S.U.V.’s 1.1 million
Cars 1.5 million
Toyota rolled out a new version of the Camry, and once again it was the country’s best-selling car.

Corolla sales also jumped, along with gas prices. Toyota could not escape the decline in sales of supersized S.U.V.’s like its Sequoia.

Honda

+3.2%
Trucks/vans/S.U.V.’s 0.7 million
Cars 0.8 million
Like the Corolla, the small Honda Civic did well, but the Accord stalled. Buyers, it seems, are waiting for the new version to be released this year.

[A. Cox and H. Fairfield, NYTimes, 2012]
Car/Truck Treemap

Ford -3.3%
Trucks/vans/S.U.V.s 1.8 million
Cars 1.1 million

Even the country’s best-selling vehicles, the F-Series, slumped in 2006, with sales dropping 13 percent. One of Ford’s bright spots was the new Fusion sedan, which made its debut in late 2005 and sold well in its first full year.

Reading the Chart
Boxes are scaled proportionally according to number of cars sold in 2006.

Change in sales from 2005 to 2006
Below | Above industry average | No 2005 sales
-10% | -2.6 | 0 | +10 | +100

Many of these vehicles were introduced in 2005.

[A. Cox and H. Fairfield, NYTimes, 2012]
Treemap

• Containment marks instead of connection marks
• Encodes some attribute of the items as the **size** of the rectangles
• Not as easy to see the intermediate rectangles
• Scalability: millions of leaf nodes and links possible

• Need a layout algorithm!
Layout Algorithms

• How do we generate the area marks?
• What considerations should we try to keep in mind?
Layout Algorithms

• How do we generate the area marks?
• What considerations should we try to keep in mind?
  - area true to quantitative value
  - show hierarchy
  - aspect ratio
• Also…
  - ordering
  - stability
Treemap Layouts: Slice

- Just divide horizontally
- Dice is similar, just vertical
- Problem: Bad aspect ratio!
  - Very skinny rectangles
  - Makes it harder to compare sizes, see labels, select rectangles
  - Want rectangles that are closer to squares
  - Aspect ratio = width/height
Treemap Layouts: Slice & Dice

- Split at each level into strips
- At each step, orientation of division (horizontal/vertical) changes
- Better, but some rectangles still have bad aspect ratio
Treemap Layouts: Strip

- Consider aspect ratio when adding rectangles
- Do one row at a time by processing rectangles in sorted order by size
  - Check if adding the next rectangle to the row improves aspect ratio
  - When it doesn't, go to next row
- Problem: Last rectangles have bad aspect ratios
- Solution: Look ahead to decide if would be better to add to previous row
Treemap Layouts: Squarify

- Slice & Dice and Strip can lead to bad **aspect ratios**
- Solution: Strip only uses rows, allow columns to be used, too
- Choose divisions (x/y) based on the width/height of region in order to maintain good aspect ratios
  - Use left and right side
  - Process large rectangles first
- Ordering not preserved which may cause issues if the data is updated
Squarification Algorithm

These steps are repeated until all rectangles have been processed. Again, an optimal result cannot be guaranteed, and counterexamples can be set up. The order in which the rectangles are processed is important. We found that a decreasing order usually gives the best results. The initially large rectangle is then filled in first with the larger subrectangles.

3.2 Algorithm

Following the example, we present our algorithm for the layout of the children in one rectangle as a recursive procedure \texttt{squarify}. This procedure lays out other rectangles in horizontal and vertical rows. When a rectangle is processed, a decision is made between two alternatives. Either the rectangle is added to the current row, or the current row is fixed and a new row is started in the remaining subrectangle. This decision depends only on whether adding a rectangle to the row will improve the layout of the current row or not.

We assume a datatype \texttt{Rectangle} that contains the layout during the computation and is global to the procedure \texttt{squarify}. Its support function \texttt{width()} that gives the length of the shortest side of the remaining subrectangle in which the current row is placed and a function \texttt{layoutrow()} that adds a new row of children to the rectangle. To keep the description simple, we use some list notation: ++ is concatenation of lists, is the list containing element , and is the empty list. The input of \texttt{squarify()} is basically a list of real numbers, representing the areas of the children to be laid out. The list row contains...
Squarification Algorithm

These steps are repeated until all rectangles have been processed. Again, an optimal result can not be guaranteed, and counterexamples can be set up. The order in which the rectangles are processed is important. We found that a decreasing order usually gives the best results. The initially large rectangle is then filled in first with the larger subrectangles.

3.2 Algorithm

Following the example, we present our algorithm for the layout of the children in one rectangle as a recursive procedure \textit{squarify}. This procedure lays out the rectangle in horizontal and vertical rows. When a rectangle is processed, a decision is made between two alternatives. Either the rectangle is added to the current row, or the current row is fixed and a new row is started in the remaining subrectangle. This decision depends only on whether adding a rectangle to the row will improve the layout of the current row or not.

We assume a datatype \texttt{Rectangle} that contains the layout during the computation and is global to the procedure \textit{squarify}. Its support function \texttt{width()} that gives the length of the shortest side of the remaining subrectangle in which the current row is placed and a function \texttt{layoutrow()} that adds a new row of children to the rectangle. To keep the description simple, we use some list notation: ++ is concatenation of lists, \texttt{[e]} the list containing element \texttt{e}, and \texttt{} the empty list. The input of \textit{squarify()} is basically a list of real numbers, representing the areas of the children to be laid out. The list \texttt{row} contains the layout during the computation.

\[ \text{[Brus et al., 1999]} \]
Squarified Treemaps

(a) File system

(b) Organization

[Brus et al., 1999]
Squarified Layout

- Sort values
- Switch orientation whenever necessary to obtain best aspect ratios
Improving Treemaps (Cushion)

- Leaves are ok, but it can be difficult to find the hierarchy
- Encode this as shading information
- More effective to understand hierarchy

[van Wijk and van de Wetering, 1999]
Disk Inventory

[Disk Inventory X]
Squarified + Cushioned Treemaps

(a) File system

(b) Organization

[Brus et al., 1999]
**Variations: Marimekko Chart**

**How each state generated electricity in 2019**

- **Vermont**: Nearly all electricity generated came from zero-carbon sources.
- **Illinois**: Leads the nation in the amount of electricity generated from nuclear power.
- **Iowa**: 42% of electricity generated came from wind.
- **Texas**: The biggest electricity producing state, also produced the most clean electricity.
- **West Virginia**: 91% of electricity generation came from coal, more than any other state.

**Total electricity generated in the state in 2019**
- Renewable
- Nuclear
- Fossil fuels

**100% clean electricity**

**50%**

**75%**

**95%**

**U.S. average clean electricity**

**100% fossil fuels**

---

[J. Muyskens, Washington Post]
Nested Circles

• Looks more like cluster diagram, but shows hierarchy
• Containment shown by the layering of semi-transparent circles
• Labeling becomes more difficult

[Bostock, 2012]
Compound Networks

- Add a hierarchy to the network (e.g. from clustering)
- GrouseFlocks: uses nested circles with colors

![Graph Hierarchy 1](image1)
![Graph Hierarchy 2](image2)
![Graph Hierarchy 3](image3)

Fig. 3. Multiple graph hierarchies superimposed on the same graph. In (a), we set the input graph without any hierarchy superimposed to fit. In (b), we have a table of three of the many possible hierarchies which can be superimposed on (a). The first two rows of the table show these graph hierarchies superimposed on the same base graph. As a graph hierarchy defines the types of abstractions which can be visualized by cuts, a single graph hierarchy is not suitable for all interesting views of the graph data.

![Hierarchy Graph](image4)
![Edge Exists](image5)
![Edge Does Not Exist](image6)

Fig. 4. Edge conservation. In (a), an edge exists between two meta nodes at some level of the hierarchy. A valid input graph is shown in (b) where there exist edges which connect leaf nodes which are descendants of both meta nodes. An invalid input graph is shown in (c) where edges do not connect descendants of the two meta nodes.

![Hierarchy Graph](image7)
![Metanode Connected](image8)
![Metanode Not Connected](image9)

Fig. 5. Connectivity conservation. In (a), there is a cycle between two meta nodes at some level of the hierarchy. A valid input graph for this hierarchy is shown in (b) as there exists a cycle in the underlying graph. An invalid input graph is shown in (c) where there is not a cycle in the underlying graph. Thus, subgraphs must be connected for our hierarchies to be topologically preserving.

[Archambault et al., 2008]