Data Visualization (CSCI 627/490)

Trees & Sets

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

Arrange 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 \textbf{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

β = 0
β = 0.25
β = 0.5
β = 0.75
β = 1

[Holten, 2006]
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
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

Figure 7.5: Comparing matrix and node-link views of a five-node network.

(a) Matrix view. (b) Node-link view. From [Henry et al. 07], Figure 3b and 3a. (Permission needed.)

Matrix views of networks can achieve very high information density, up to a limit of one thousand nodes and one million edges, just like cluster heatmaps and all other matrix views that uses small area marks.

Technique

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Derived Data</th>
<th>View Comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>network</td>
<td>table: network nodes as keys, link status between two nodes as values</td>
<td></td>
</tr>
<tr>
<td></td>
<td>space: area marks in 2D matrix alignment</td>
<td></td>
</tr>
</tbody>
</table>

7.1.3.3 Multiple Keys: Partition and Subdivide

When a dataset has only one key, then it is straightforward to use that key to separate into one region
Clique in Adjacency Matrices

a

b

[Drawings of graphs and matrices representing cliques in adjacency matrices]
Structures from Adjacency Matrices

cliques

bicliques

clusters

[McGuffin]
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
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
Tree Visualizations

- A: Multiple branches with nodes at the tips.
- B: A branched tree structure.
- C: A treemap with rectangular tiles.
- D: A circular tree-like structure.
- E: Concentric circles with segments.
- F: Another concentric circle arrangement.
- G: A grid layout with rectangles.
- H: A horizontal bar chart.

[McGuffin and Robert, 2010]
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

[M. Bostock, 2017]
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
Treemap

D. Koop, CSCI 627/490, Fall 2022

[M. Bostock, 2017]
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.

**SALES CHANGE**

- **Chrysler Group**: -7.6%
  - 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.

**READING THE CHART**

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

**Change in sales from 2005 to 2006**

- **Below**
  - 25,000
  - 100,000

- **Above industry average**
  - 25,000
  - 100,000

**No. 2005 sales**

- **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

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.

[Ray Herring]

**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

<table>
<thead>
<tr>
<th>Change Percentage</th>
<th>2005 Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10%</td>
<td>25,000</td>
</tr>
<tr>
<td>-2.6</td>
<td>100,000</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>+10</td>
<td></td>
</tr>
<tr>
<td>+100</td>
<td></td>
</tr>
</tbody>
</table>

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
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 subrectangle.

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 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 \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, \texttt{[e]} is the list containing element \texttt{e}, and \texttt{[]} 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 \texttt{row} is used to store the current row.

\begin{itemize}
  \item \textbf{Step 1:} Initial rectangle with area 8/3.
  \item \textbf{Step 2:} Divide the rectangle into two subrectangles of areas 6 and 3/2.
  \item \textbf{Step 3:} Further divide the rectangle into subrectangles of areas 4 and 6.
  \item \textbf{Step 4:} Divide the rectangle into subrectangles of areas 6 and 4.
  \item \textbf{Step 5:} Divide the rectangle into subrectangles of areas 6, 4, and 3.
  \item \textbf{Step 6:} Divide the rectangle into subrectangles of areas 6, 4, 3, and 2.
\end{itemize}

\[ \text{[Brus et al., 1999]} \]
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 **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 **Rectangle** that contains the layout during the computation and is global to the procedure **squarify**. Its support functions **width()** that gives the length of the shortest side of the remaining subrectangle in which the current row is placed and a function **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 **squarify()** is basically a list of real numbers, representing the areas of the children to be laid out. The list **row** concatenates these areas.

[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

Figure 4. Binary subdivision of interval

\[ f(x) = \begin{cases} 0, & \text{for } x < -1 \\ \frac{x}{2}, & \text{for } -1 \leq x \leq 1 \\ 1, & \text{for } x > 1 \end{cases} \]

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

[Image of a disk inventory showing the contents of a file named iPhoto (31.3 MB). The inventory includes various files and folders with their sizes listed.]
Squarified + Cushioned Treemaps

(a) File system

(b) Organization

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

How each state generated electricity in 2019

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

[J. Muyskens, Washington Post]

D. Koop, CSCI 627/490, Fall 2022
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

![Input Graph](image1)

<table>
<thead>
<tr>
<th>Graph Hierarchy 1</th>
<th>Graph Hierarchy 2</th>
<th>Graph Hierarchy 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image2" alt="Hierarchy Graph 1" /></td>
<td><img src="image3" alt="Hierarchy Graph 2" /></td>
<td><img src="image4" alt="Hierarchy Graph 3" /></td>
</tr>
</tbody>
</table>

![Graph Hierarchies](image5)

[Archambault et al., 2008]