Data Visualization (CIS 490/680)

Trees & Design

Dr. David Koop
What are the ingredients for a geospatial visualization?
Map Projection + Position Data

- Central Meridian (selected by mapmaker)
- Great distortion at high latitudes
- Examples of two rhumb lines (direction true between any two points)
- Equator touches cylinder if cylinder is tangent
- Reasonably true shapes and distances within 15 degrees of Equator

[USGS Map Projections]
Don't Just Create Population Maps!

*Pet Peeve #208: Geographic profile maps which are basically just population maps.*
Cartograms

- Data: geographic geometry data & two quantitative attributes (one part-of-whole)
- Derived data: new geometry derived from the part-of-whole attribute
- Tasks: trends, comparisons, part-of-whole
- How: area marks from derived geometry, color hue/saturation/luminance
- Scalability: thousands of regions
- Design choices:
  - Colormap
  - Geometric deformation

[New York Times]
Networks

• Why not graphs?
  - Bar graph
  - Graphing functions in mathematics

• 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
Node-Link Diagrams

• Data: nodes and edges
• Task: understand connectivity, paths, structure (topology)
• Encoding: nodes as point marks, connections as line marks
• Scalability: hundreds

...but high density of links can be problematic!

• We need position info—a layout!
Force-Directed Layout

• Nodes push away from each other but edges are springs that pull them together
• Weakness: nondeterminism, algorithm may produce difference results each time it runs

[M. Bostock, 2017]
“Hairball”

[Hu, 2014]
Midterm

• Thursday
• Covers material through this week
• Format:
  - Multiple Choice
  - Free Response (often multi-part)
  - CS 680 students will have extra questions related to the research papers discussed
Project Design

• Start working on turning your visualization ideas into designs
• Sketch
• Options:
  - Try vastly different options
  - Refine an initial idea
Next Week

• Tuesday: Guest Lecture from Prof. Sun
• Thursday: No class, work on projects
Hierarchical Edge Bundling

Fig. 13. A software system and its associated call graph (caller = green, callee = red). (a) and (b) show the system with bundling strength $\beta = 0.85$ using a balloon layout (node labels disabled) and a radial layout, respectively. Bundling reduces visual clutter, making it easier to perceive the actual connections than when compared to the non-bundled versions (figures 2a and 11a). Bundled visualizations also show relations between sparsely connected systems more clearly (encircled regions); these are almost completely obscured in the non-bundled versions. The encircled regions highlight identical parts of the system for (a), (b), and figure 15.

Fig. 14. Using the bundling strength $\beta$ to provide a trade-off between low-level and high-level views of the adjacency relations. The value of $\beta$ increases from left-to-right; low values mainly provide low-level, node-to-node connectivity information, whereas high values provide high-level information as well by implicit visualization of adjacency edges between parent nodes that are the result of explicit adjacency edges between their respective child nodes.

More specifically, most of the participants particularly valued the fact that relations between items at low levels of the hierarchy were automatically lifted to implicit relations between items at higher levels by means of bundles. This quickly gave them an impression of 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. This is illustrated in figure 14, which shows visualizations using different values for the bundling strength $\beta$.

Low values result in visualizations that mainly provide low-level, node-to-node connectivity information. High values result in visualizations that provide high-level information as well by implicit visualization of adjacency edges between parent nodes that are the result of explicit adjacency edges between their respective child nodes.

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, a thick bundle shows the presence of two elements at a fairly high level of the hierarchy, whereas the fanning out of a bundle shows the subdivision of an element into subelements.

Most participants preferred the radial layout over the balloon layout and the squarified treemap layout. Another finding was the fact that the rooted layout and the slice-and-dice treemap layout were considered less pleasing according to several participants. This is probably due to the large number of collinear nodes within these layouts, which causes bundles to overlap along the collinearity axes. This is illustrated in figure 17.

Although our main focus while developing hierarchical edge bundling was on the visualization itself, interaction is an important aspect in determining the usability of our technique. Based on our own insight and feedback gathered from participants, we contend that bundle-based interaction as described below could provide a convenient way of interacting with the visualizations.

Figure 16 shows how the bundling strength $\beta$ could be used in conjunction with...
Hierarchical Edge Bundling

[Holten, 2006]
Hierarchical Edge Bundling

- Flexible and generic method
- Reduces visual clutter when dealing with large numbers of adjacency edges
- Provides an intuitive and continuous way to control the strength of bundling.
  - Low bundling strength mainly provides low-level, node-to-node connectivity information
  - High bundling strength provides high-level information as well by implicit visualization of adjacency edges between parent nodes that are the result of explicit adjacency edges between their respective child nodes

[Holten, 2006]
Bundling Strength

\( \beta = 0 \) \hspace{1cm} \beta = 0.25 \hspace{1cm} \beta = 0.5 \hspace{1cm} \beta = 0.75 \hspace{1cm} \beta = 1 \)

[Holten, 2006]
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]
 Cliques in Adjacency Matrices

a

b

[Gehlenborg and Wong]
Structures from Adjacency Matrices
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 graphs
  - 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: File browser diagrams like Microsoft Explorer.
- B: Hierarchical tree representations.
- C: Treemaps, similar to [14, 26, 6, 30].
- D: Concentric circles, similar to [2, 27, 31].
- E: 2D Graphical Representations of Trees.
- F: Icicle diagrams, similar to [5, 28].
- G: Concentric circles, similar to [2, 27, 31].
- H: Concentric circles, similar to [5, 28].

[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

[Bostock, 2012]
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, CIS 680, Fall 2019

[18, 29]

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**
-7.0%

**Trucks/vans/S.U.V’s**: 1.6 million
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.

**Dodge Ram**

---

### SALES CHANGE

**General Motors**

-8.7%

**Trucks/vans/S.U.V’s**: 2.5 million
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.

---

### TRUCKS, VANS, S.U.V’s

- **Trucks**:
  - Dodge Ram 3500
  - Ford F-150
  - GMC Sierra

- **Vans**:
  - Chrysler Town & County
  - Ford Econoline
  - Honda Odyssey

- **S.U.V’s**:
  - Jeep Liberty
  - Jeep Wrangler
  - Toyota RAV4

### CARS

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

- **Honda**
  - +3.2%
  - **Trucks/vans/S.U.V’s**: 0.7 million
  - 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

Trucks/vans/S.U.V.'s 1.8 million
Cars 0.1 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.

A. Cox and H. Fairfield, NYTimes, 2012

[Car/Truck Treemap]

Sources: Wards Auto/Edmunds

BMW

+2.1%

Trucks/vans/S.U.V.'s 0.1 million
Cars 0.3 million

Mercedes-Benz

+10.3%

Trucks/vans/S.U.V.'s 0.1 million
Cars 0.2 million

Mercedes-Benz, owned by DaimlerChrysler, had a comeback in 2006, thanks to a new version of its flagship S-Class. BMW sales were helped by a new version of its 3 Series sport sedan.

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
Layout Algorithms

• **Aspect ratio** concerns: square is better
• **Slice and dice:**
  - Split at each level into strips
  - At each step, orientation of division (horiz/vert) changes
• **Strip**
  - Order rectangles and move to a new row when aspect ratio gets worse
Improving Treemaps (Squarified)

• Switching from horizontal to vertical cuts may be ok for nicely-behaved trees, but can lead to bad aspect ratios

• Problem: harder to compare sizes, more difficult to select/mouse over the rectangles

• Solution: 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 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 children 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` contains this list.

[Brus et al., 1999]
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 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, {x} is the list containing element \( x \), 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 containing the rectangles at each step is represented as a list.

Fig. 4. Squarification Algorithm

[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

![Diagram of cushion treemap]

[van Wijk and van de Wetering, 1999]
Disk Inventory
Squarified + Cushioned Treemaps

(a) File system  (b) Organization

[Brus et al., 1999]
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

Fig. 3. Multiple graph hierarchies superimposed on the same graph. In (a), we set a input graph without any hierarchy superimposed to fit. In (b), we have a table of three of the many possible hierarchies which are not 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.

Fig. 4. Edge conservation. In (a), there is an edge 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.

Fig. 5. Connectivity conservation. In (a), there is a cycle between the 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.

Hierarchy space which would allow users to see abstractions of their graph data based on attributes. In our software engineering example, it may prove useful to restructure the hierarchy to view methods which are or are not involved with some cross-cutting concern. A hierarchy based on this information would be better than the one of packages and classes to investigate the concern as significant parts of the graph can be abstracted away. Only a few systems allow hierarchy editing and these systems are limited to manual selection of nodes in the graph [7], [14] or provide limited tools for exploring the created hierarchy [25].