Graph Layout

Maneesh Agrawala
Jessica Hullman

CS 294-10: Visualization
Fall 2014

Announcements
Final project

Design new visualization method
- Pose problem, Implement creative solution

Deliverables
- Implementation of solution
- 8-12 page paper in format of conference paper submission
- 1 or 2 design discussion presentations

Schedule
- Project proposal: 10/27
- Project presentation: 11/10, 11/12
- Final paper and presentation: TBD, likely 12/1-12/5

Grading
- Groups of up to 3 people, graded individually
- Clearly report responsibilities of each member

In-Class Project Presentations

Dates: 11/10 and 11/12
- Description of problem you are addressing
- Survey of related work
- Description/storyboard and demo of approach
- A list of milestones for finishing the project by the deadline

Scheduling
- Send me dates you cannot attend class by tomorrow
- Next class We will ask for volunteers to present on each day
Graph Layout
Graphs and Trees

Graphs
Model relations among data
*Nodes and edges*

Trees
Graphs with hierarchical structure
- Connected graph with N-1 edges
*Nodes as parents and children*

Spatial Layout

Primary concern – layout of nodes and edges

Often (but not always) goal is to depict structure
- Connectivity, path-following
- Network distance
- Clustering
- Ordering (e.g., hierarchy level)
Applications

- Tournaments
- Organization Charts
- Genealogy
- Diagramming (e.g., Visio)
- Biological Interactions (Genes, Proteins)
- Computer Networks
- Social Networks
- Simulation and Modeling
- Integrated Circuit Design

Tree Visualization

- Indentation
  - Linear list, indentation encodes depth

- Node-Link diagrams
  - Nodes connected by lines/curves

- Enclosure diagrams
  - Represent hierarchy by enclosure

- Layering
  - Layering and alignment

Tree layout is fast: $O(n)$ or $O(n \log n)$, enabling real-time layout for interaction.
Indentation

- Items along vertically spaced rows
- Indentation shows parent/child relationships
- Often used in interfaces
- Breadth/depth contend for space
- Often requires scrolling

Node-Link Diagrams

- Nodes distributed in space, connected by straight/curved lines
- Use 2D space to break apart breadth and depth
- Space used to communicate hierarchical orientation (typically towards authority or generality)
Basic Recursive Approach

Repeatedly divide space for subtrees by leaf count
- Breadth of tree along one dimension
- Depth along the other dimension

Problem: exponential growth of breadth
Reingold & Tilford’s Tidier Layout

Goal: maximize density and symmetry.

Originally for binary trees, extended by Walker to cover general case.

This extension was corrected by Buchheim et al to achieve a linear time algorithm.

Reingold-Tilford Layout

Design concerns
Clearly encode depth level
No edge crossings
Isomorphic subtrees drawn identically
Ordering and symmetry preserved
Compact layout (don ’t waste space)
Reingold-Tilford Algorithm

Linear algorithm – starts with bottom-up (postorder) pass of the tree
Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- “Shifts” in position saved for each node as visited
- Parent nodes are centered above their children

Top-down (preorder) pass for assignment of final positions
- Sum of initial layout and aggregated shifts
Reingold-Tilford Algorithm

0

Reingold-Tilford Algorithm

1
0
Reingold-Tilford Algorithm

1 2

0 1

Reingold-Tilford Algorithm

2 1

0 3
Reingold-Tilford Algorithm
Reingold-Tilford Algorithm
Reingold-Tilford Algorithm

2
1
0

6
4
5
3

Reingold-Tilford Algorithm

2
1
0

6
4
5
3
Reingold-Tilford Algorithm
Reingold-Tilford Algorithm

![Diagram of a tree structure with nodes labeled 0 to 7.]

Reingold-Tilford Algorithm

![Diagram of a tree structure with nodes labeled 0 to 8. A dashed line connects nodes 3 and 5.]

17
Reingold-Tilford Algorithm
Reingold-Tilford Algorithm

Reingold-Tilford Algorithm
Reingold-Tilford Algorithm

Diagram 1:

Diagram 2:

Reingold-Tilford Algorithm
Reingold-Tilford Algorithm
Reingold-Tilford Algorithm

Reingold-Tilford Algorithm
Reingold-Tilford Algorithm

Linear algorithm – starts with bottom-up (postorder) pass of the tree
Y-coord by depth, arbitrary starting X-coord
Merge left and right subtrees
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- “Shifts” in position saved for each node as visited
- Parent nodes are centered above their children
Top-down pass (preorder) for assignment of final positions
- Sum of initial layout and aggregated shifts

Radial Layout

Node-link diagram in polar coords
Radius encodes depth root at center
Angular sectors assigned to subtrees (recursive approach)
Reingold-Tilford approach can also be applied here
Circular Drawing of Trees in 3D

Cone trees – 3D layout

Balloon Trees = 2D Cone Trees
Not just flattening – circles must not overlap
Problems with Node-Link Diagrams

Scale
Tree breadth often grows exponentially
Even with tidier layout, quickly run out of space

Possible solutions
Filtering
Focus+Context
Scrolling or Panning
Zooming
Aggregation

Visualizing Large Hierarchies

Indented Layout
Reingold-Tilford Layout
Hyperbolic Layout

Layout in hyperbolic space, then project onto Euclidean plane

Why? Like tree breadth, the hyperbolic plane expands exponentially

Also computable in 3D, projected into a sphere
Degree-of-Interest Trees [AVI 04]

Space-constrained, multi-focal tree layout

Cull “un-interesting” nodes on a per block basis until all blocks on a level fit within bounds
Attempt to center child blocks beneath parents
Enclosure Diagrams

Encode structure using spatial enclosure
Popularly known as TreeMaps

Benefits
- Provides a single view of an entire tree
- Easier to spot large/small nodes

Problems
- Difficult to accurately read depth

TreeMaps

Recursively fill space based on node size
- Enclosure signifies hierarchy
- Additional measures to control aspect ratio of cells
- Often uses rectangles, but other shapes are possible, e.g., iterative Voronoi tesselation.
Layered Diagrams

- Signify tree structure using
- Layering
- Adjacency
- Alignment

Involves recursive sub-division of space
Can apply the same set of approaches as in node-link layout

Icicle and Sunburst Trees

Higher-level nodes get a larger layer area, whether that is horizontal or angular extent
Child levels are layered, constrained to parent’s extent
### Layered Tree Drawing

<table>
<thead>
<tr>
<th></th>
<th>Coffee Latte</th>
<th>Decaf Irish C . .</th>
<th>Caffe Mocha</th>
<th>Decaf Espresso</th>
<th>Regular Espresso</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Colorado</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Illinois</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iowa</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Missouri</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ohio</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wisconsin</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>East</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connecticut</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Florida</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Massachusetts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>New Hampshire</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>New York</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>South</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Louisiana</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>New Mexico</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oklahoma</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Texas</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>West</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>California</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Nevada</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oregon</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Utah</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Washington</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Hybrids are also possible...

“Elastic Hierarchies”
Node-link diagram with treemap nodes
Graph Visualization

Approaches to Graph Drawing

Direct calculation using graph structure
- Tree layout on spanning tree
- Hierarchical layout
- Adjacency matrix layout

Optimization-based layout
- Constraint satisfaction
- Force-directed layout

Attribute-driven layout
- Layout using data attributes, not linkage
Spanning Tree Layout

Many graphs are tree-like or have useful spanning trees
- Websites, Social Networks

Use tree layout on spanning tree of graph
- Trees created by BFS / DFS
- Min/max spanning trees

Fast tree layouts allow graph layouts to be recalculated at interactive rates

Heuristics may further improve layout

Spanning tree layout may result in arbitrary parent node
Sugiyama-style graph layout

Evolution of the UNIX operating system
Hierarchical layering based on descent

Reverse some edges to remove cycles
Assign nodes to hierarchy layers \(\rightarrow\) Longest path layering
Create dummy nodes to “fill in” missing layers
Arrange nodes within layer, minimize edge crossings
Route edges – layout splines if needed
Hierarchical graph layout

Gnutella network

Limitations of Node-Link Layout

Edge-crossings and occlusion
Optimization Techniques

Treat layout as an optimization problem
Define layout using a set of constraints: equations the layout should try to obey
Use optimization algorithms to solve

Common approach for undirected graphs
Force-Directed Layout most common

We can introduce directional constraints
DiG-CoLa (Di-Graph Constrained Optimization Layout) [Dwyer 05]

Optimizing “Aesthetic” Constraints

Minimize edge crossings
Minimize area
Minimize line bends
Minimize line slopes
Maximize smallest angle between edges
Maximize symmetry

but, can’t do it all.

Optimizing these criteria is often NP-Hard, requiring approximations.
Force-Directed Layout

Edges = springs \[ F = -k \cdot (x - L) \]
Nodes = charged particles \[ F = G \cdot m_1 \cdot m_2 / x^2 \]

Repeatedly calculate forces, update node positions
Naïve approach \( O(N^2) \)
Speed up to \( O(N \log N) \) using quadtree or k-d tree
Numerical integration of forces at each time step
Constrained Optimization Layout

Minimize stress function

\[
\text{stress}(X) = \sum_{i<j} w_{ij} \left( \|X_i - X_j\| - d_{ij} \right)^2
\]

- \(X\): node positions, \(d\): optimal edge length,
- \(w\): normalization constants
- Use global \((\text{majorization})\) or localized \((\text{gradient descent})\) optimization

\(\rightarrow\) Says: Try to place nodes \(d_{ij}\) apart

Add hierarchy ordering constraints

\[
E_H(y) = \sum_{(i,j) \in E} (y_i - y_j - \delta_{ij})^2
\]

- \(y\): node \(y\)-coordinates
- \(\delta\): edge direction (e.g., 1 for \(i \rightarrow j\), 0 for undirected)

\(\rightarrow\) Says: If \(i\) points to \(j\), it should have a lower \(y\)-value

Sugiyama layout (dot)
Preserve tree structure

DiG-CoLa method
Preserve edge lengths
Attribute-Driven Layout

Large node-link diagrams get messy!
Is there additional structure we can exploit?

Idea: Use data attributes to perform layout
  - e.g., scatter plot based on node values
Dynamic queries and/or brushing can be used to explore connectivity
Attribute-Driven Layout

The “Skitter” Layout
- Internet Connectivity
- Radial Scatterplot

Angle = Longitude
- Geography

Radius = Degree
- # of connections
- (a statistic of the nodes)
**PivotGraph** [Wattenberg 2006]

Layout aggregated graphs according to node attributes

---

**Operators**

- **Roll-Up**
  - Aggregate items with matching data values

- **Selection**
  - Filter on data values
PivotGraph

Node and Link Diagram

PivotGraph Roll-up
Limitations of PivotGraph

Only 2 variables (no nesting as in Tableau)
Doesn’t support continuous variables
Multivariate edges?
Hierarchical Edge Bundles

Trees with Adjacency Relations
Bundle Edges along Hierarchy

(a) $P_{\text{Start}} = P_0$

(b) $P_{\text{End}} = P_4$

(c) $P_2 \Rightarrow \text{LCA}(P_0, P_4)$

Configuring Edge Tension

(a) $\beta = 0$

(b) $\beta = 0.25$

(c) $\beta = 0.5$

(d) $\beta = 0.75$

(e) $\beta = 1$
Use radial tree layout for inner circle
Mirror to outside
Replace inner tree with hierarchical edge bundles
Summary

Tree Layout
Indented / Node-Link / Enclosure / Layers
How to address issues of scale?
  - Filtering and Focus + Context techniques

Graph Layout
Tree layout over spanning tree
Hierarchical “Sugiyama” Layout
Optimization (Force-Directed Layout)
Attribute-Driven Layout