Graph Layout

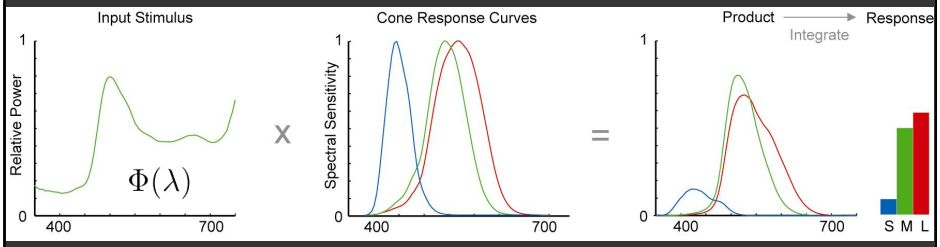
Maneesh Agrawala

CS 448B: Visualization
Fall 2017

Last Time: Color

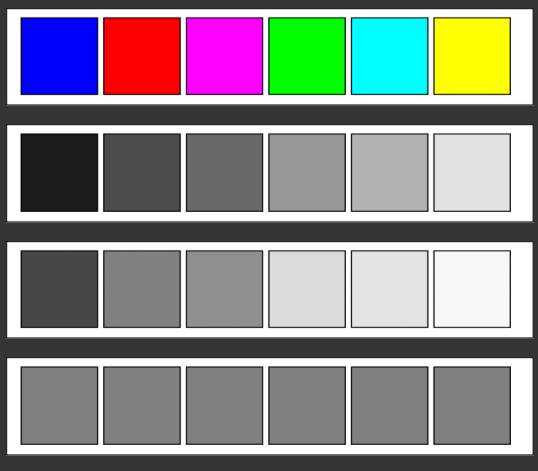
Computing Cone Response

Integrate cone response with input spectra



$$L = \int \Phi(\lambda) L(\lambda) \mathrm{d}\lambda$$
$$M = \int \Phi(\lambda) M(\lambda) \mathrm{d}\lambda$$
$$S = \int \Phi(\lambda) S(\lambda) \mathrm{d}\lambda$$

L vs. Luminance, L*



Corners of the RGB color cube

Luminance values

L* values

L from HLS
All the same

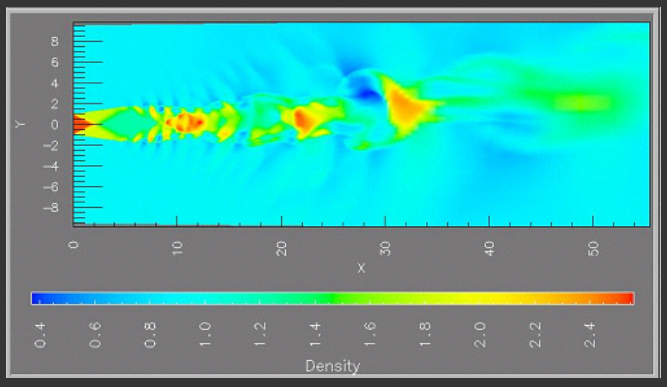
Palette Design + Color Names

Minimize overlap and ambiguity of color names

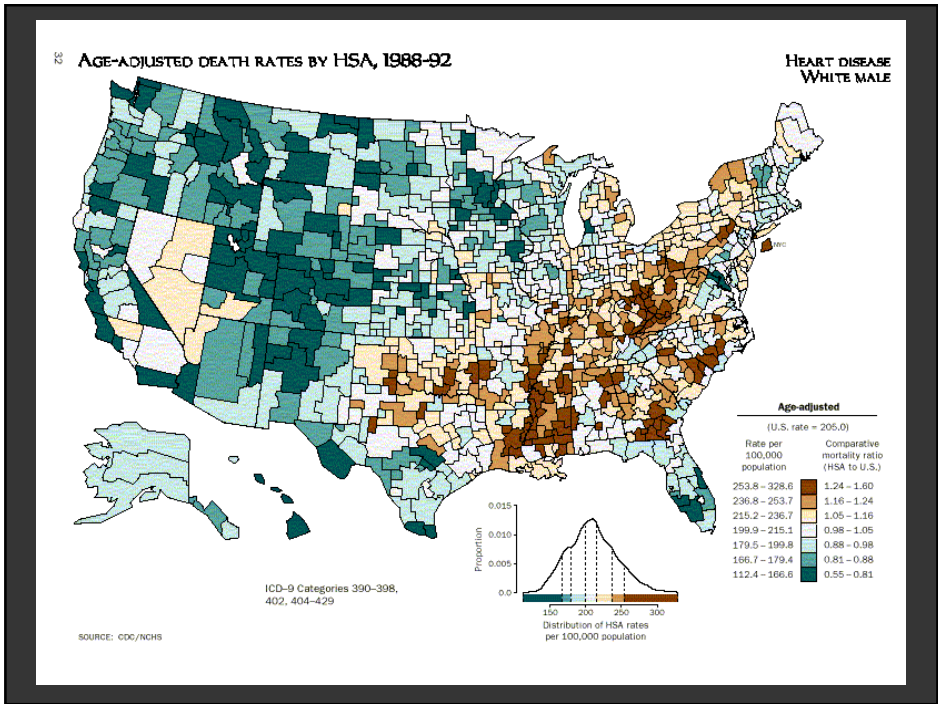
Color Name Distance										Salience	Name
0.00	1.00	1.00	1.00	0.98	1.00	1.00	1.00	1.00	0.20	.47	**blue** 62.9%
1.00	0.00	1.00	0.97	1.00	1.00	1.00	1.00	0.96	1.00	.90	**orange** 93.9%
1.00	1.00	0.00	1.00	1.00	1.00	1.00	1.00	0.90	0.99	.67	**green** 79.8%
1.00	0.97	1.00	0.00	1.00	0.95	0.99	1.00	1.00	1.00	.66	**red** 80.4%
0.98	1.00	1.00	1.00	0.00	0.96	0.91	0.97	1.00	0.99	.47	**purple** 51.4%
1.00	1.00	1.00	0.95	0.96	0.00	0.97	0.93	0.98	1.00	.37	**brown** 54.0%
1.00	1.00	1.00	0.99	0.91	0.97	0.00	1.00	1.00	1.00	.58	**pink** 71.7%
1.00	1.00	1.00	1.00	0.97	0.93	1.00	0.00	1.00	1.00	.67	**grey** 79.4%
1.00	0.96	0.90	1.00	1.00	0.98	1.00	1.00	0.00	1.00	.18	**yellow** 31.2%
0.20	1.00	0.99	1.00	0.99	1.00	1.00	1.00	1.00	0.00	.25	**blue** 25.4%
Tableau-10								*Average*	*0.97*	*.52*	

http://vis.stanford.edu/color-names

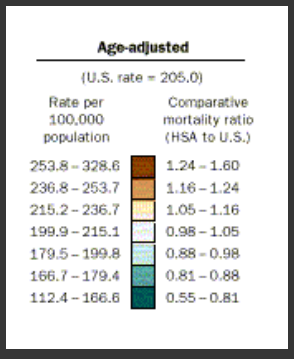
Avoid rainbow color maps!

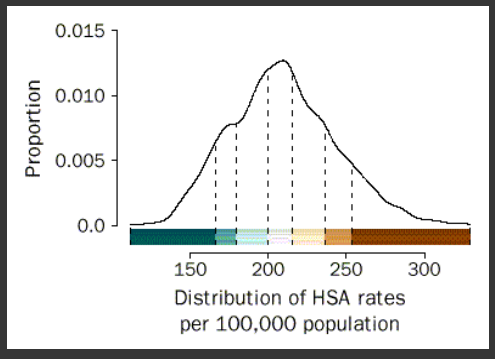


1. People segment colors into classes
2. Hues are not naturally ordered
3. Different lightness emphasizes certain scalar values
4. Low luminance colors (blue) hide high frequencies



Classing quantitative data





Age-adjusted mortality rates for the United States

Announcements

Final project

Design new visualization method (e.g. software)

- Pose problem, Implement creative solution
- Design studies/evaluations less common but also possible (talk to us)

Deliverables

- Implementation of solution
- 6-8 page paper in format of conference paper submission
- Project progress presentations

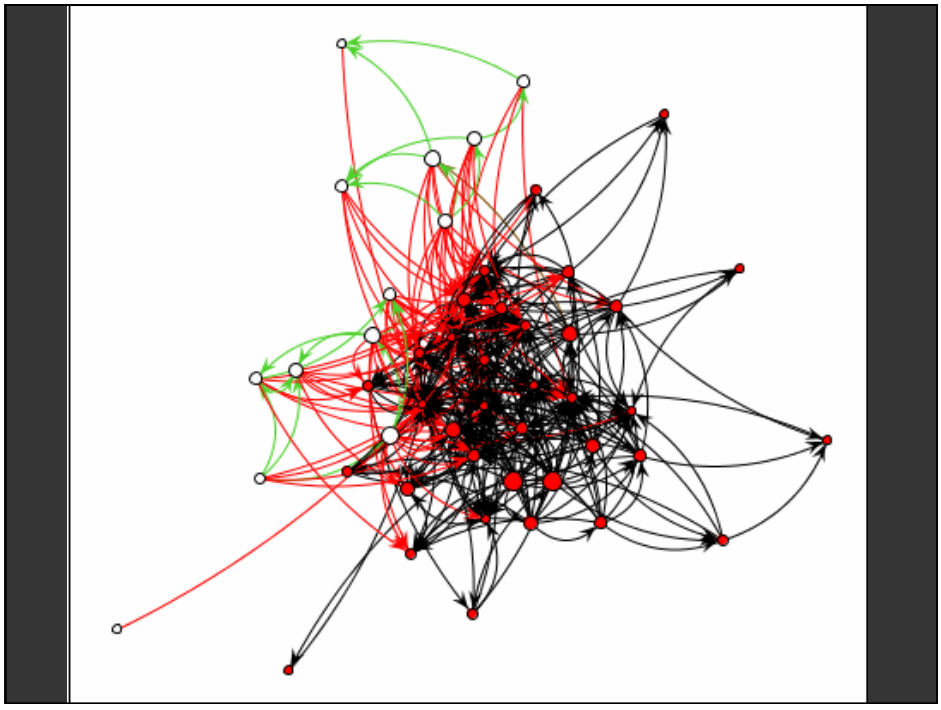
Schedule

- Project proposal: Mon 11/6
- Project progress presentation: 11/13 and 11/15 in class (3-4 min)
- Final poster presentation: 12/6 Location: Lathrop 282
- Final paper: 12/10 11:59pm

Grading

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

Graph Layout

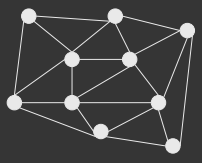


Graphs and Trees

Graphs

Model relations among data

Nodes and *edges*

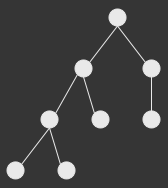


Trees

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

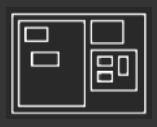
Tournaments
Organization Charts
Genealogy
Diagramming (e.g., Visio)
Biological Interactions (Genes, Proteins)
Computer Networks
Social Networks
Simulation and Modeling
Integrated Circuit Design

Tree Visualization

Indentation
- Linear list, indentation encodes depth

Node-Link diagrams
- Nodes connected by lines/curves

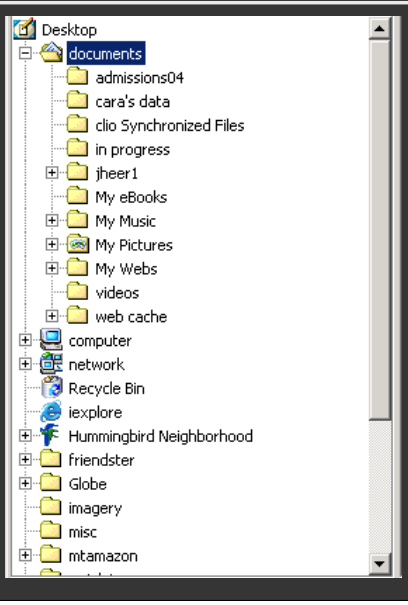
Enclosure diagrams
- Represent hierarchy by enclosure



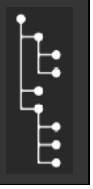
Layering
- Layering and alignment

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

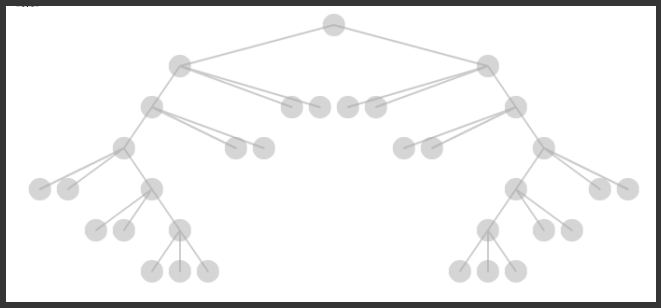
- Nodes distributed in space, connected by straight/curved lines
- Use 2D space to break apart breadth and depth
- Space used to communicate hierarchical orientation
  - Typically towards authority or generality

Basic Recursive Approach

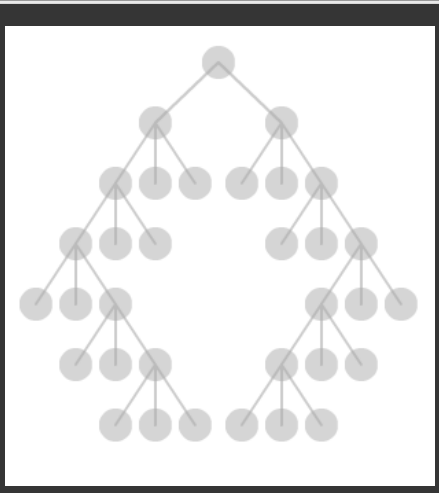
Repeatedly divide space for subtrees by leaf count

- 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

Set Y-coord by depth, arbitrary starting X-coord

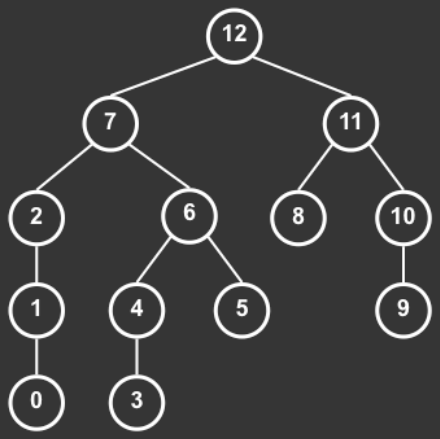
Merge left and right subtrees

- 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

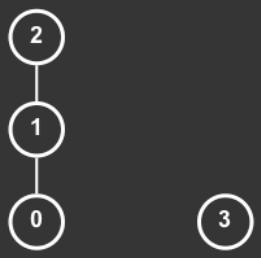


Reingold-Tilford Algorithm

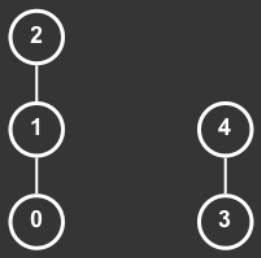
Reingold-Tilford Algorithm

Reingold-Tilford Algorithm

Reingold-Tilford Algorithm



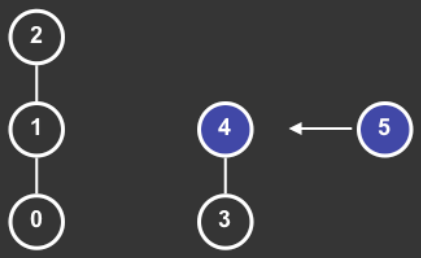
Reingold-Tilford Algorithm



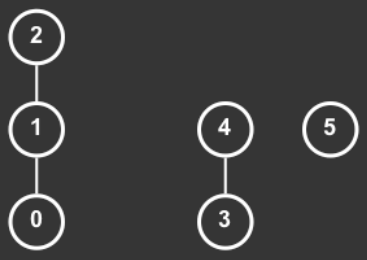
Reingold-Tilford Algorithm

Reingold-Tilford Algorithm

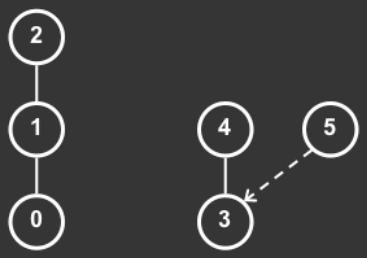
Reingold-Tilford Algorithm



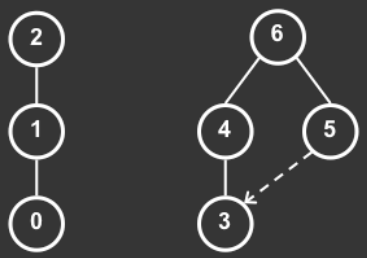
Reingold-Tilford Algorithm



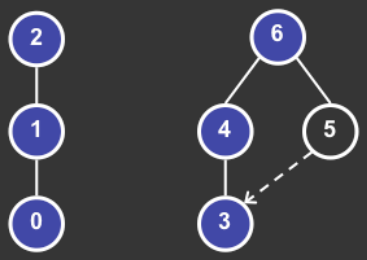
Reingold-Tilford Algorithm



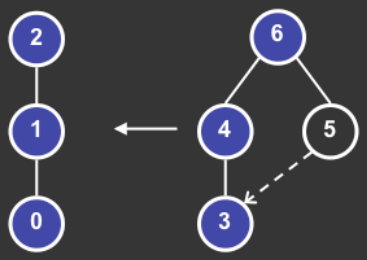
Reingold-Tilford Algorithm



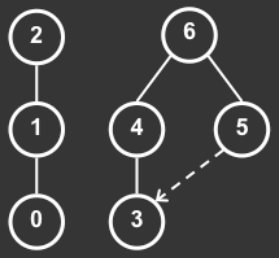
Reingold-Tilford Algorithm



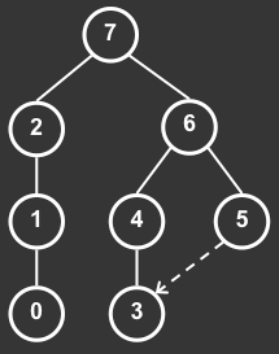
Reingold-Tilford Algorithm



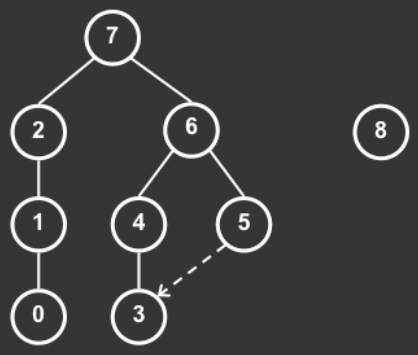
Reingold-Tilford Algorithm



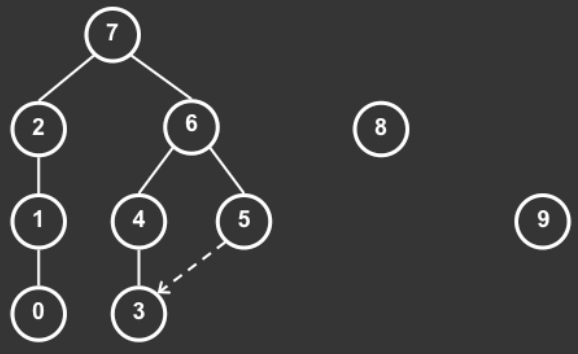
Reingold-Tilford Algorithm



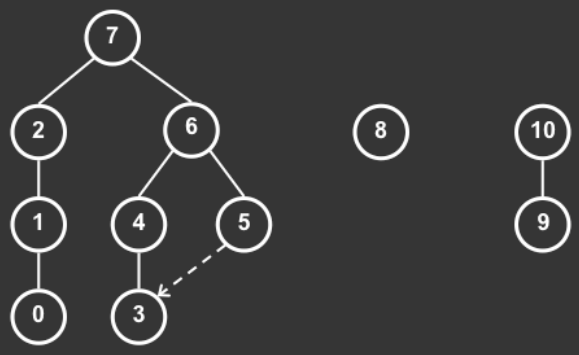
Reingold-Tilford Algorithm



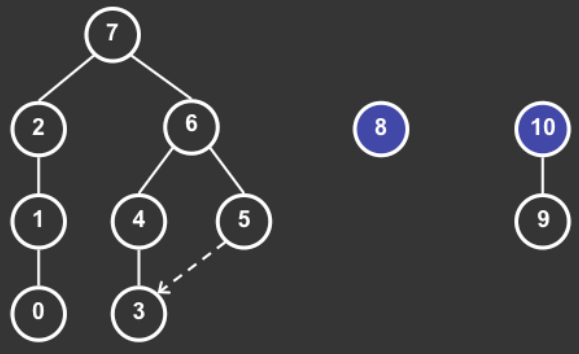
Reingold-Tilford Algorithm



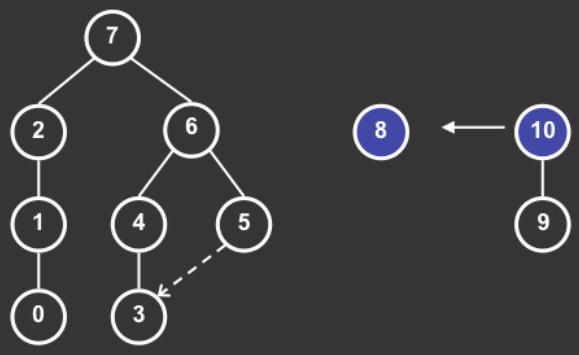
Reingold-Tilford Algorithm



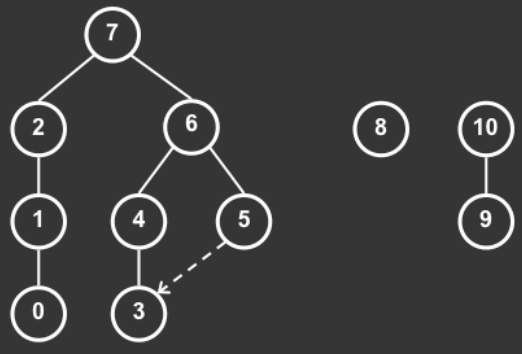
Reingold-Tilford Algorithm



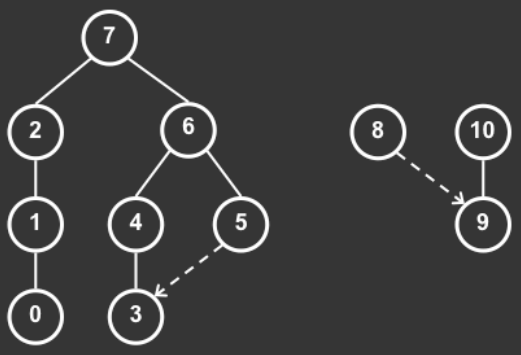
Reingold-Tilford Algorithm



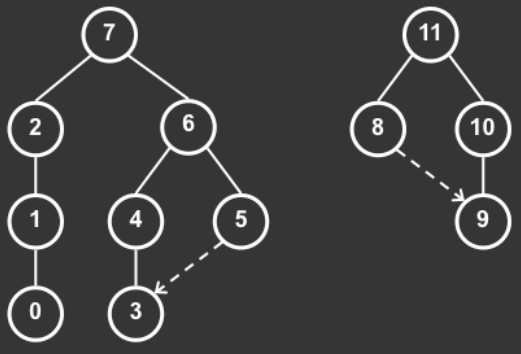
Reingold-Tilford Algorithm



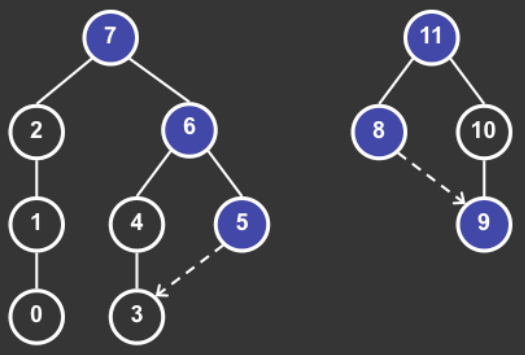
Reingold-Tilford Algorithm



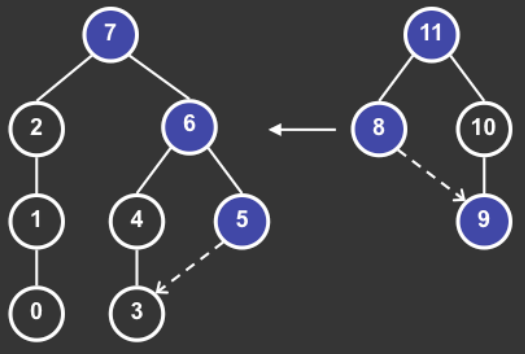
Reingold-Tilford Algorithm



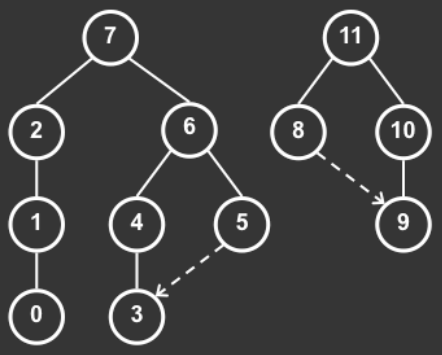
Reingold-Tilford Algorithm



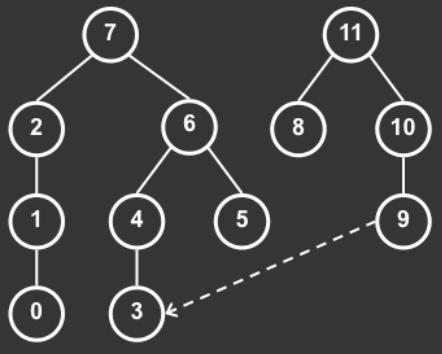
Reingold-Tilford Algorithm



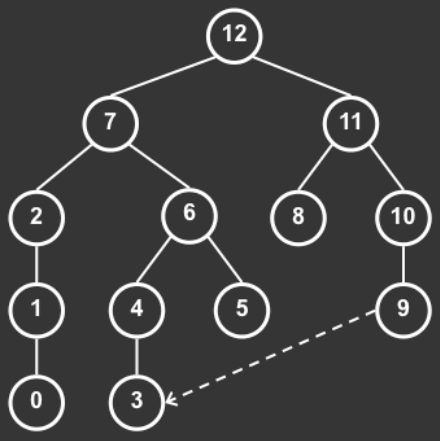
Reingold-Tilford Algorithm



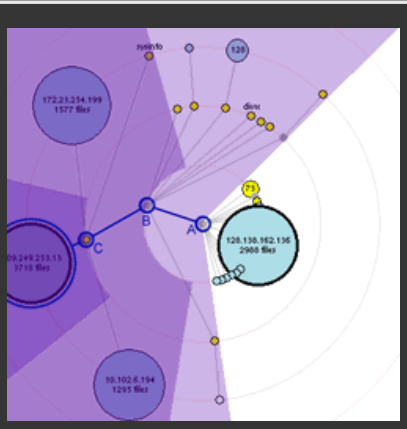
Reingold-Tilford Algorithm



Reingold-Tilford Algorithm



Radial Layout



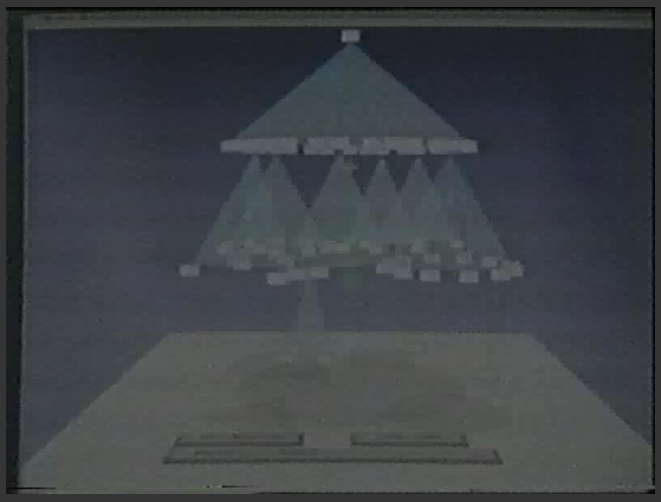
Node-link diagram in polar coords

Radius encodes depth root at center

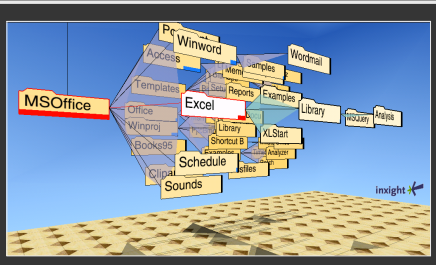
Angular sectors assigned to subtrees (recursive approach)

Reingold-Tilford approach can also be applied here

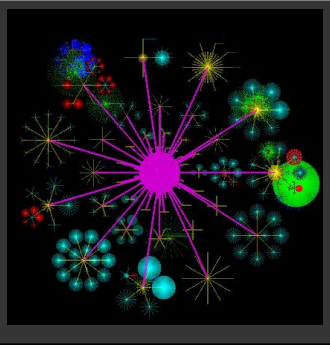
Circular Drawing of Trees in 3D



Circular Drawing of Trees



Cone trees – 3D layout



Balloon Trees = 2D Cone Trees

- Not just flattening – circles must not overlap

Problems with Node-Link Diagrams

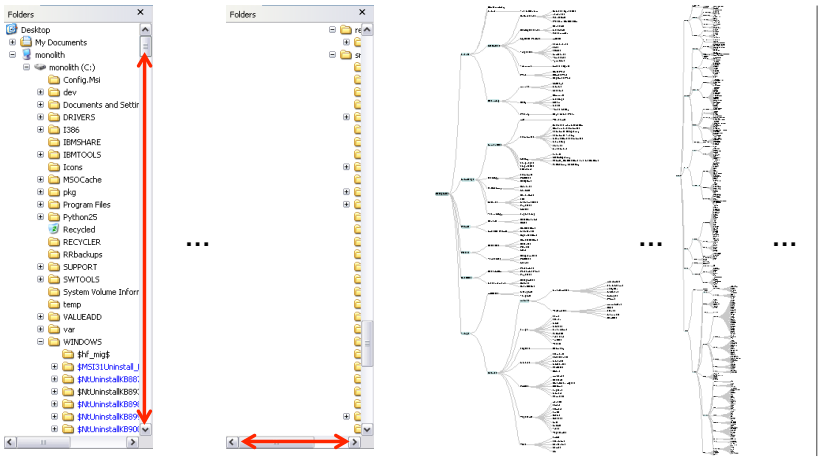
Scale

- 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

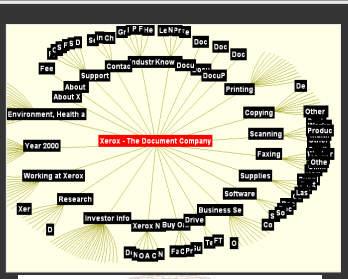


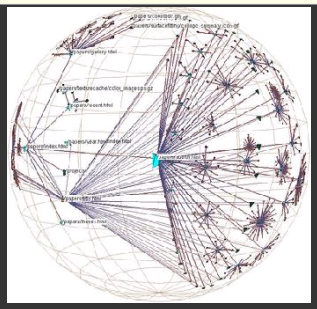
Indented Layout

Reingold-Tilford Layout

MC Escher, *Circle Limit IV*

Hyperbolic Layout



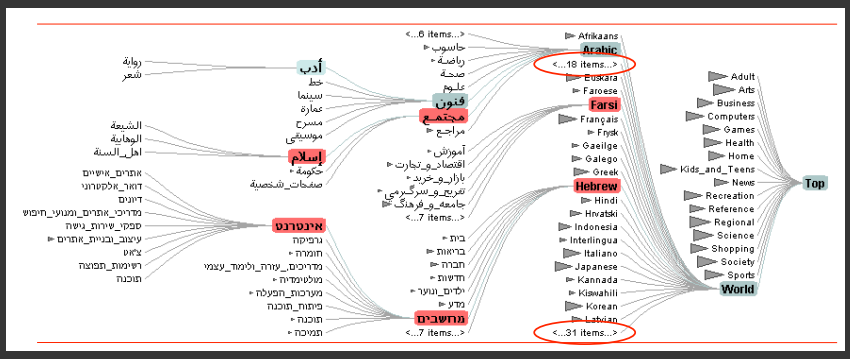


Layout in hyperbolic space, then project on to Euclidean plane

Why? Like tree breadth, the hyperbolic plane expands exponentially

Also computable in 3D, projected into a sphere

Degree-of-Interest Trees [AVI 04]

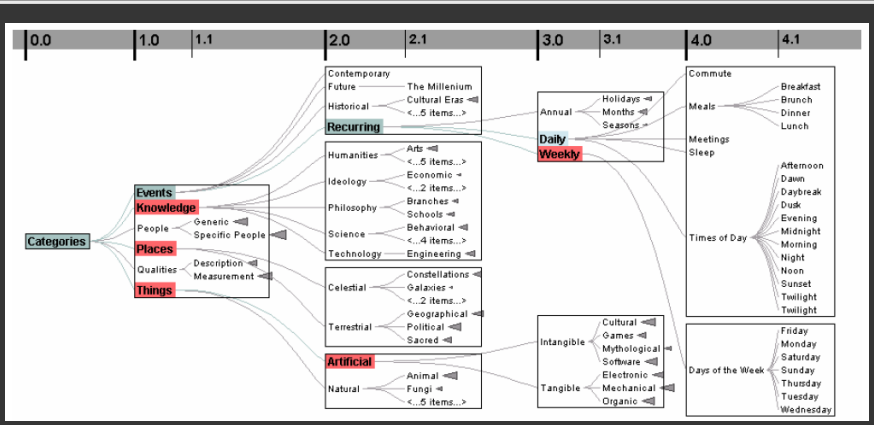


Space-constrained, multi-focal tree layout

https://www.youtube.com/watch?v=RTQ0N4QY0yc

https://bl.ocks.org/mbostock/4339083

Degree-of-Interest Trees



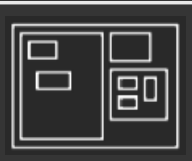
Cull “un-interesting” nodes on a per block basis until all blocks on a level fit within bounds

Center child blocks under parents

https://www.youtube.com/watch?v=RTQ0N4QY0yc

https://bl.ocks.org/mbostock/4339083

Enclosure Diagrams



Encode structure using spatial enclosure
Popularly known as TreeMaps

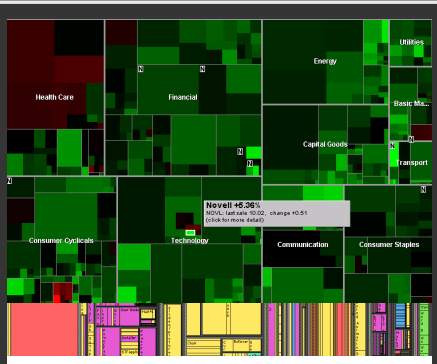
Benefits

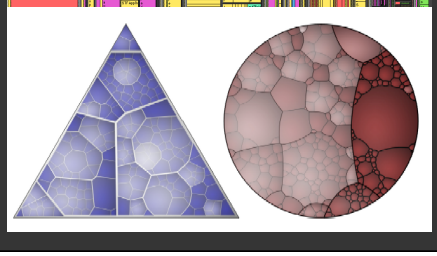
- Provides a single view of an entire tree
- Easier to spot large/small nodes

Problems

- Difficult to accurately read depth

TreeMaps

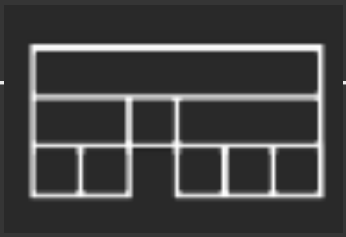




- Recursively fill space based on node size
- Enclosure signifies hierarchy
- Additional measures to control aspect ratio of cells
- Often uses rectangles, but other shapes are possible, e.g., iterative Voronoi tesselation.

https://finviz.com/map.ashx

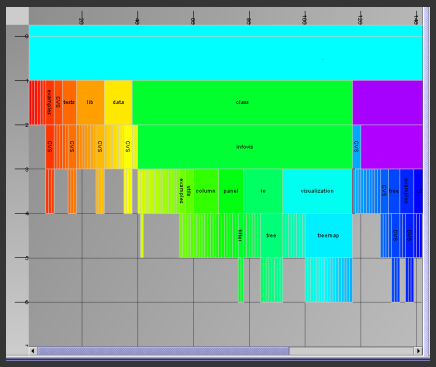
Layered Diagrams

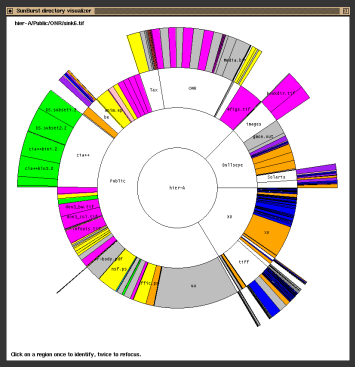


Signify tree structure using
Layering
Adjacency
Alignment

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

Icicle and Sunburst Trees

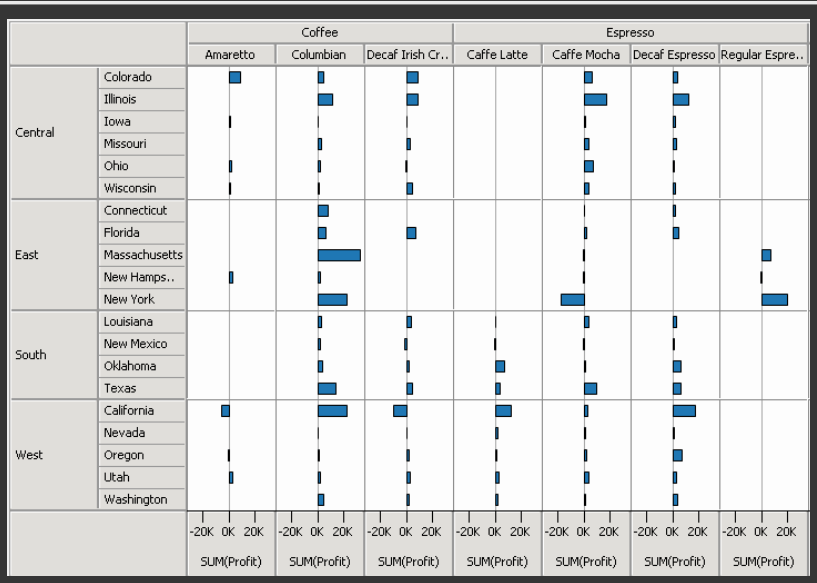




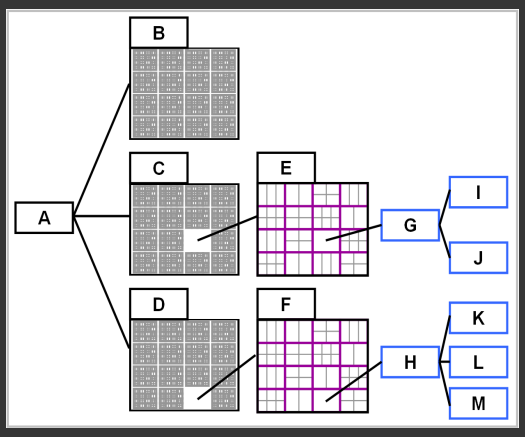
Higher-level nodes get a larger layer area, whether that is horizontal or angular extent

Child levels are layered, constrained to parent's extent

Layered Tree Drawing



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

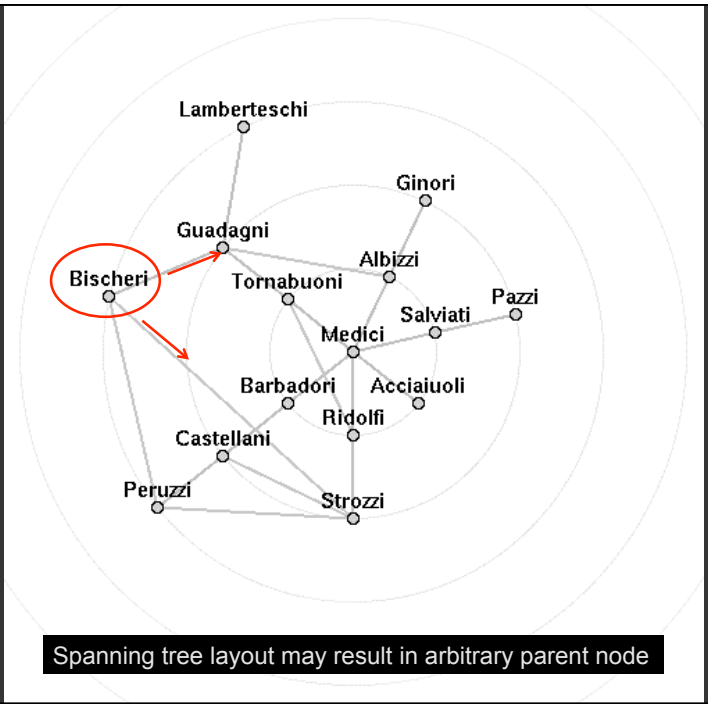
- Websites, Social Networks

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

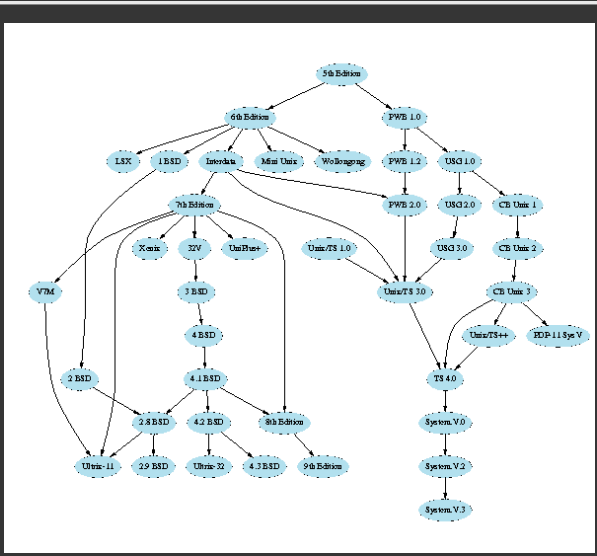
Heuristics may further improve layout



Sugiyama-style graph layout

Evolution of the UNIX operating system

Hierarchical layering based on descent



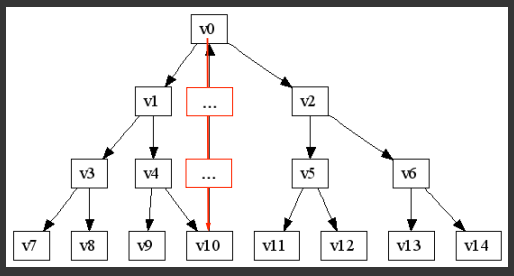
Sugiyama-style graph layout

Layer 1

Layer 2

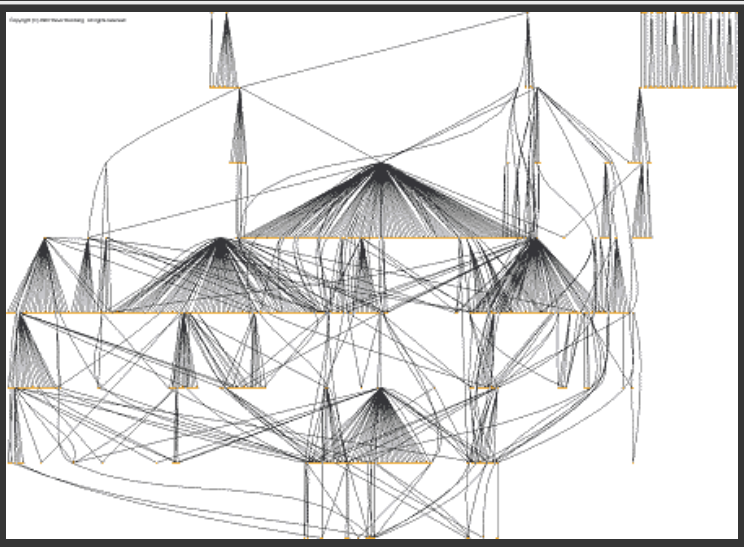
Layer 3

Layer 4



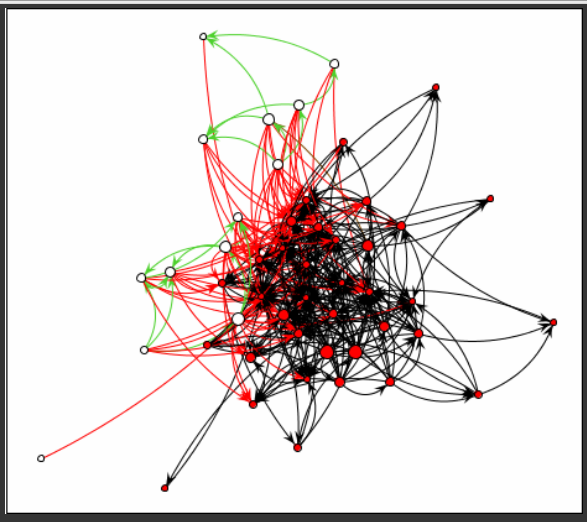
Reverse some edges to remove cycles
Assign nodes to hierarchy layers → Longest path layering
    Create dummy nodes to “fill in” missing layers
Arrange nodes within layer, minimize edge crossings
    Route edges – layout splines if needed

Hierarchical graph layout

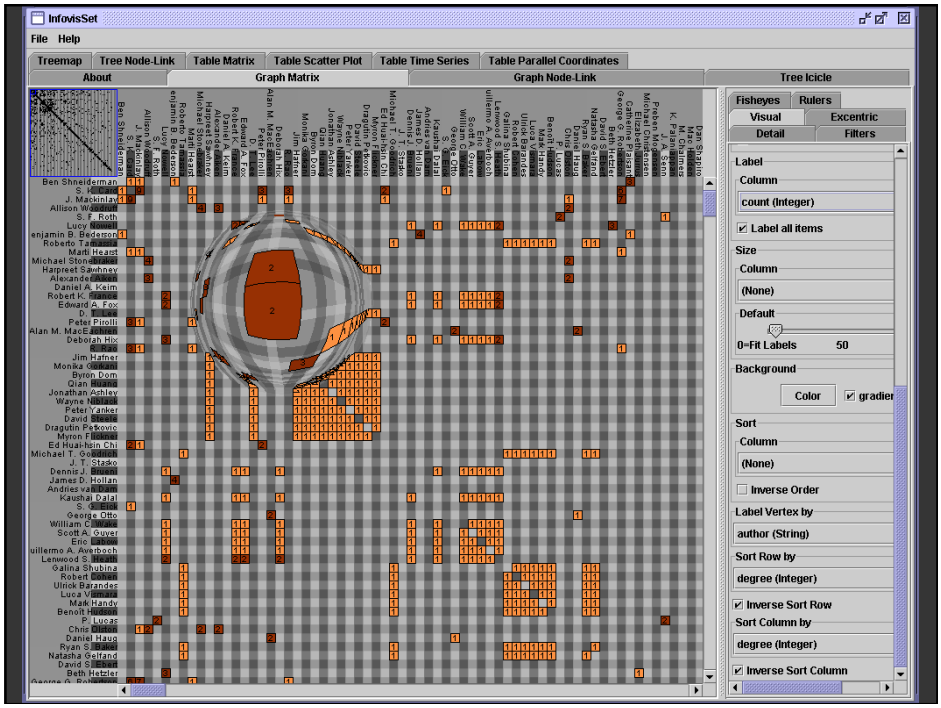


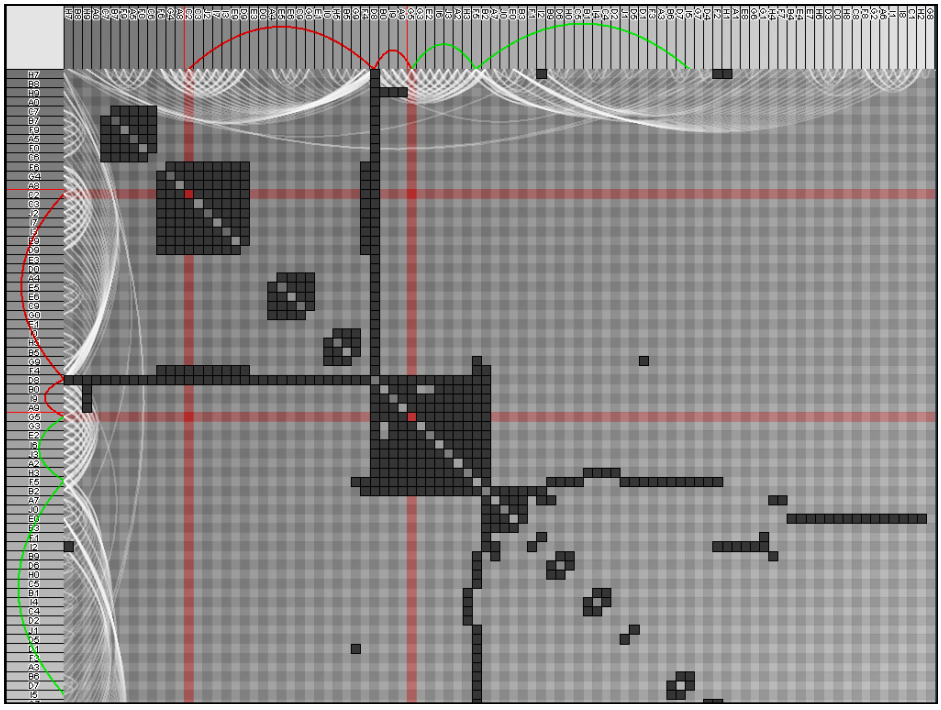
Gnutella network

Limitations of Node-Link Layout



Edge-crossings and occlusion


InfovisSet
File Help
Treemap
Tree Node-Link
Table Matrix
Table Scatter Plot
Table Time Series
Table Parallel Coordinates
About
Graph Matrix
Graph Node-Link
Tree Icicle
Fisheyes
Rulers
Visual
Excentric
Detail
Filters
Label
Column
count (Integer)
Label all items
Size
Column
(None)
Default
0=Fit Labels
50
Background
Color
gradier
Sort
Column
(None)
Inverse Order
Label Vertex by
author (String)
Sort Row by
degree (Integer)
Inverse Sort Row
Sort Column by
degree (Integer)
Inverse Sort Column

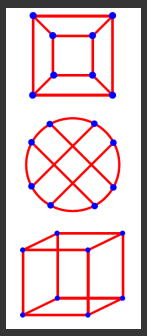


Optimization Techniques

- 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
- Can also introduce directional constraints
- *DiG-CoLa* (Di-Graph Constrained Optimization Layout) [Dwyer 05]

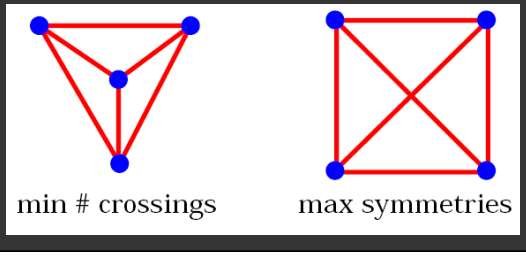
Optimizing "Aesthetic" Constraints

- 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.



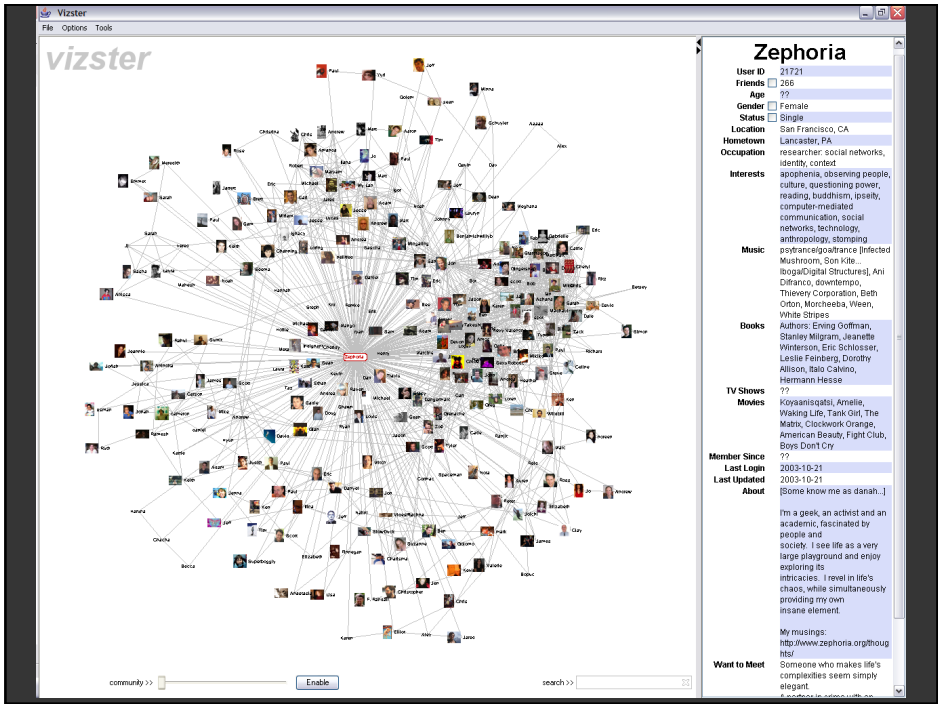
Force-Directed Layout

Edges = springs $F = -k * (x - L)$

Nodes = charged particles $F = G*m_1*m_2 / x^2$

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

$$\text{stress}(X) = \Sigma_{i<j}\, w_{ij}\, (\, ||X_i - X_j|| - d_{ij}\,)^2$$

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

→ **Says: Try to place nodes d_{ij} apart**

Add hierarchy ordering constraints

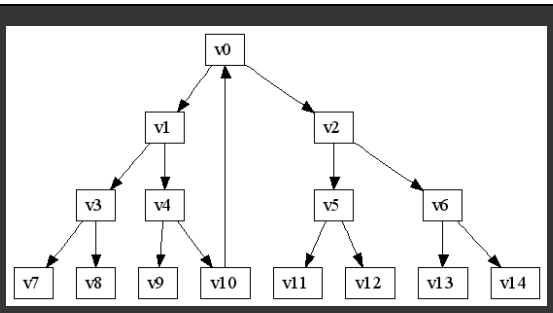
$$E_H(y) = \Sigma_{(i,j)\in E}\, (\, y_i - y_j - \delta_{ij}\,)^2$$

- y: node y-coordinates
- δ : edge direction (e.g., 1 for i→j, 0 for undirected)

→ **Says: If *i* points to *j*, it should have a lower y-value**

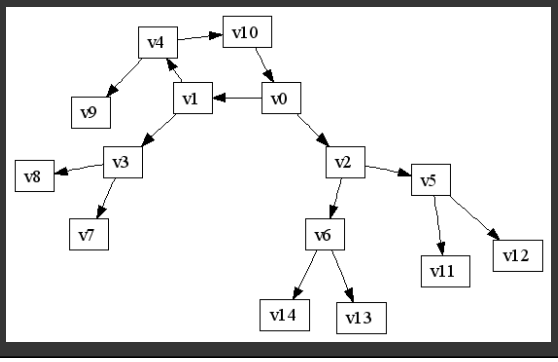
Sugiyama layout (dot)

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

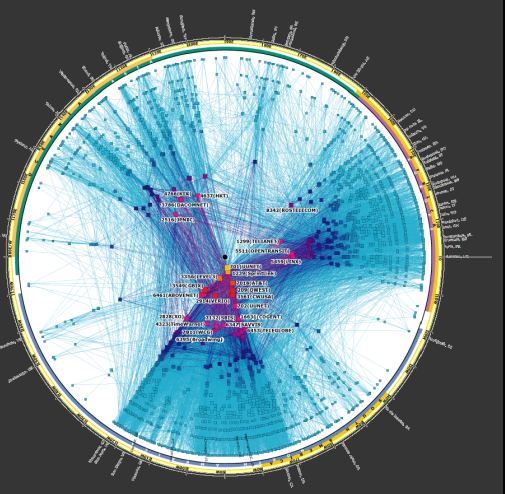
Dynamic queries and/or brushing can be used to explore connectivity

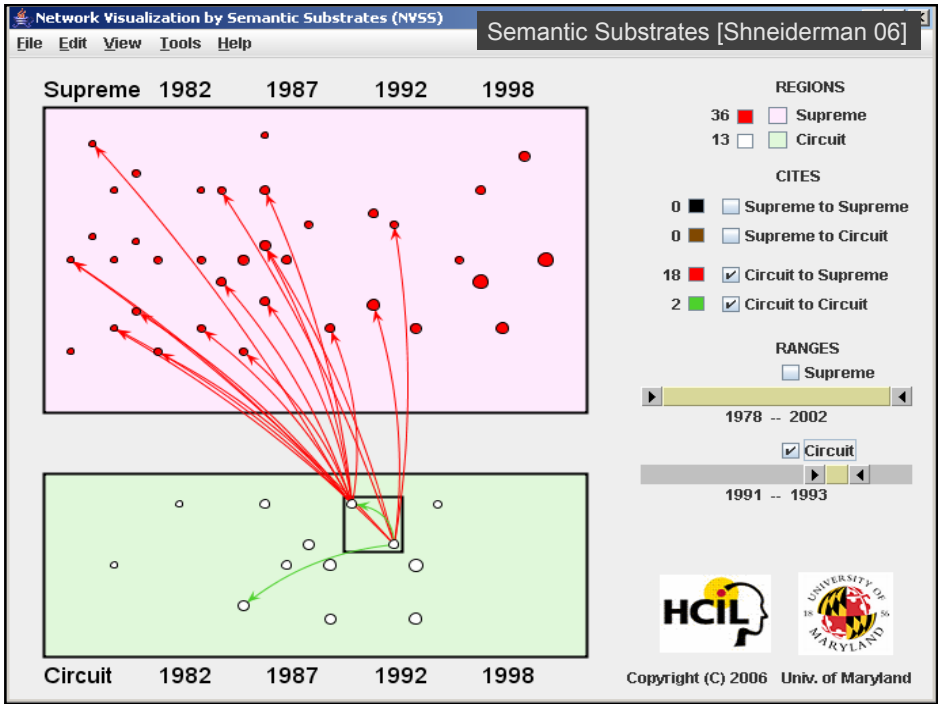
Attribute-Driven Layout

The "Skitter" Layout
- Internet Connectivity
- Radial Scatterplot

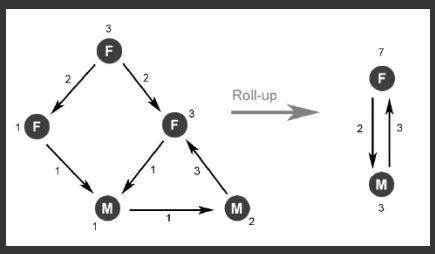
Angle = Longitude
- Geography

Radius = Degree
- # of connections
- (a statistic of the nodes)



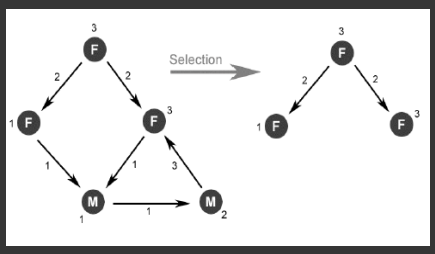


PivotGraph [Wattenberg 2006]



Roll-Up

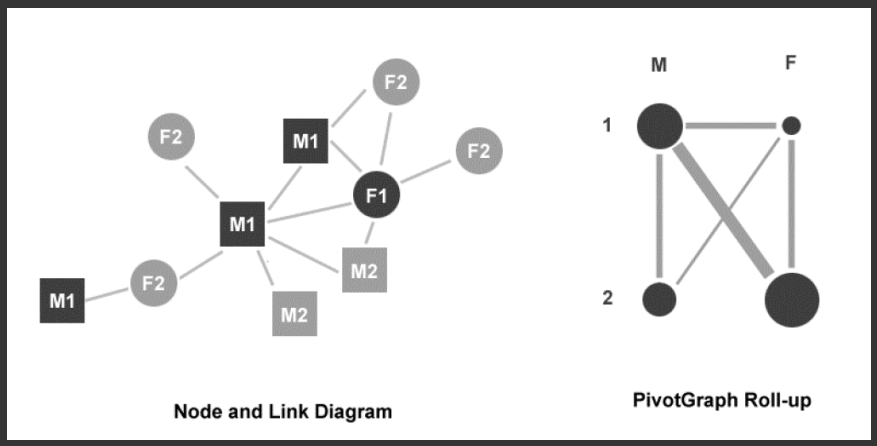
Aggregate items with matching data values

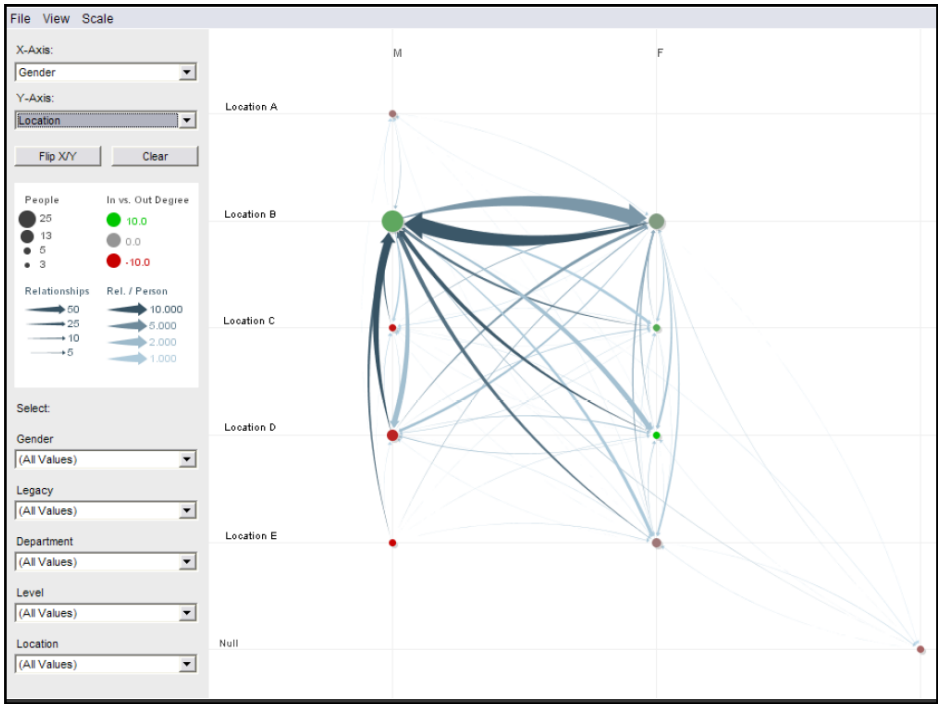


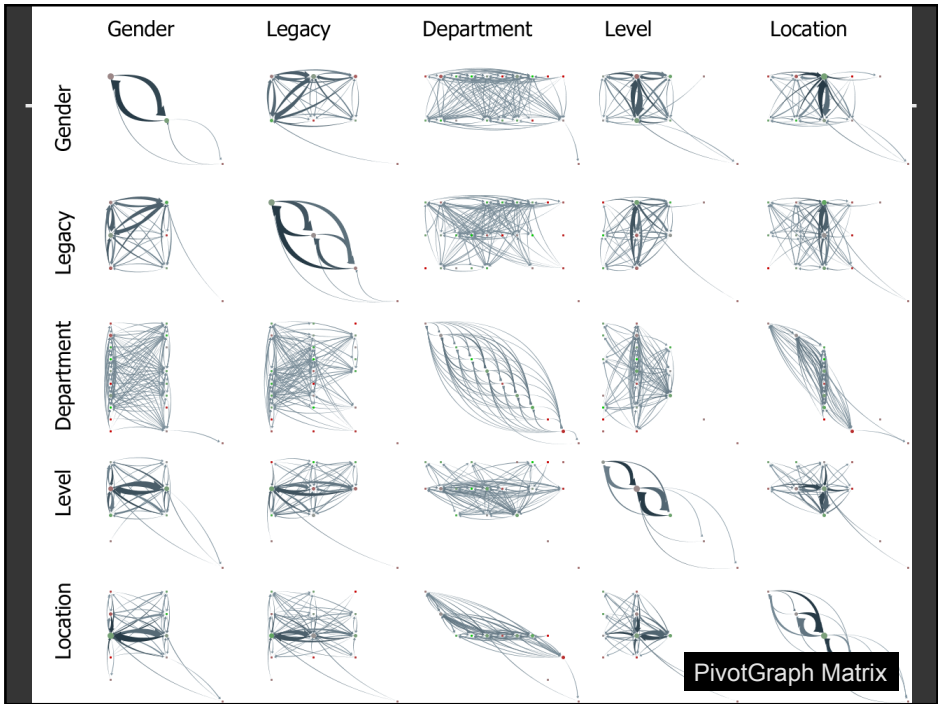
Selection

Filter on data values

PivotGraph


F2
F2
M1
F2
M1
F1
M1
F2
M2
M2
Node and Link Diagram
M
F
1
2
PivotGraph Roll-up


File View Scale
X-Axis:
Gender
Y-Axis:
Location
Flip X/Y
Clear
People
25
13
5
3
In vs. Out Degree
10.0
0.0
-10.0
Relationships
50
25
10
5
Rel. / Person
10.000
5.000
2.000
1.000
Select:
Gender
(All Values)
Legacy
(All Values)
Department
(All Values)
Level
(All Values)
Location
(All Values)
M
F
Location A
Location B
Location C
Location D
Location E
Null

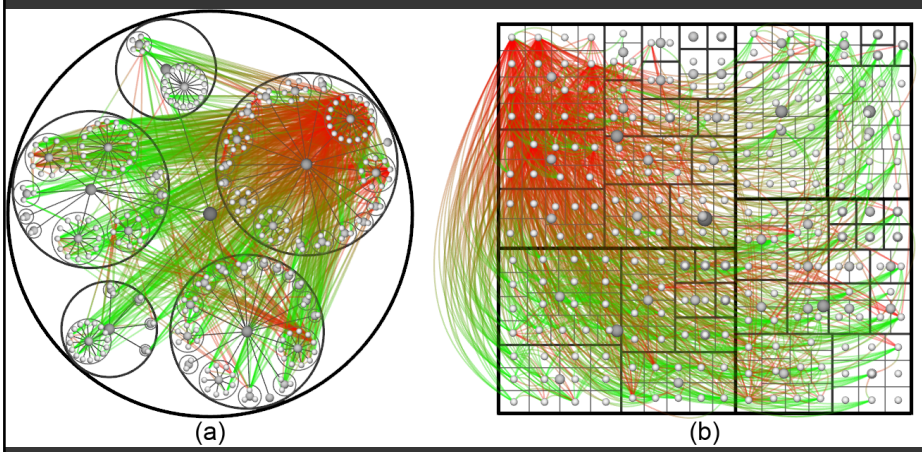


Limitations of PivotGraph

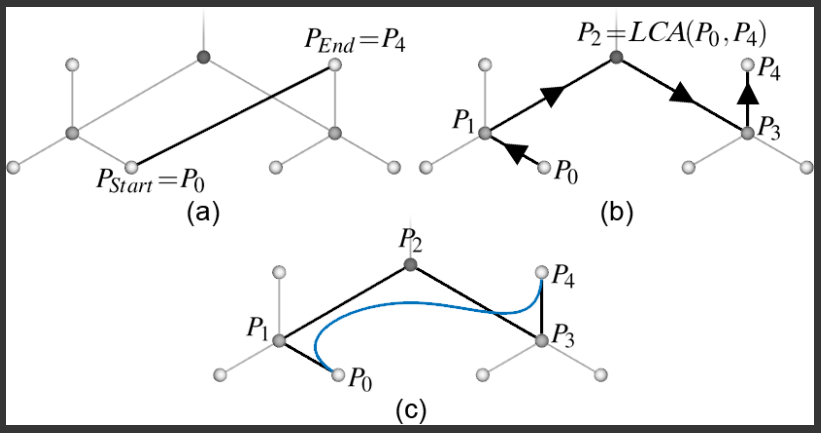
- Only 2 variables (no nesting as in Tableau)
- Doesn't support continuous variables
- Multivariate edges?

Hierarchical Edge Bundles

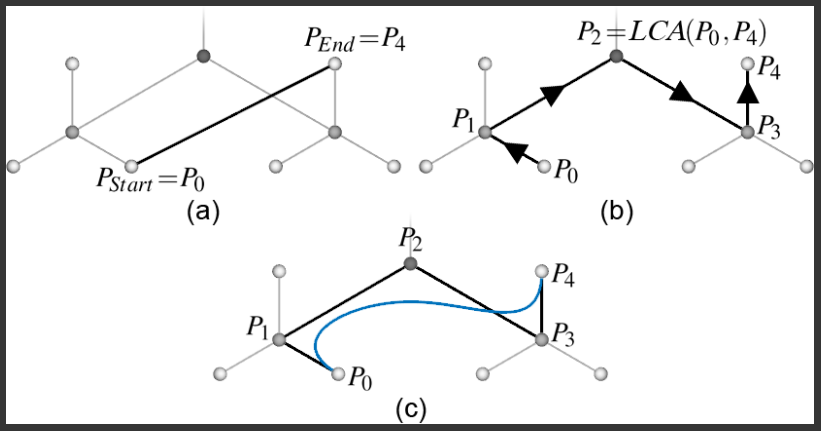
Trees with Adjacency Relations



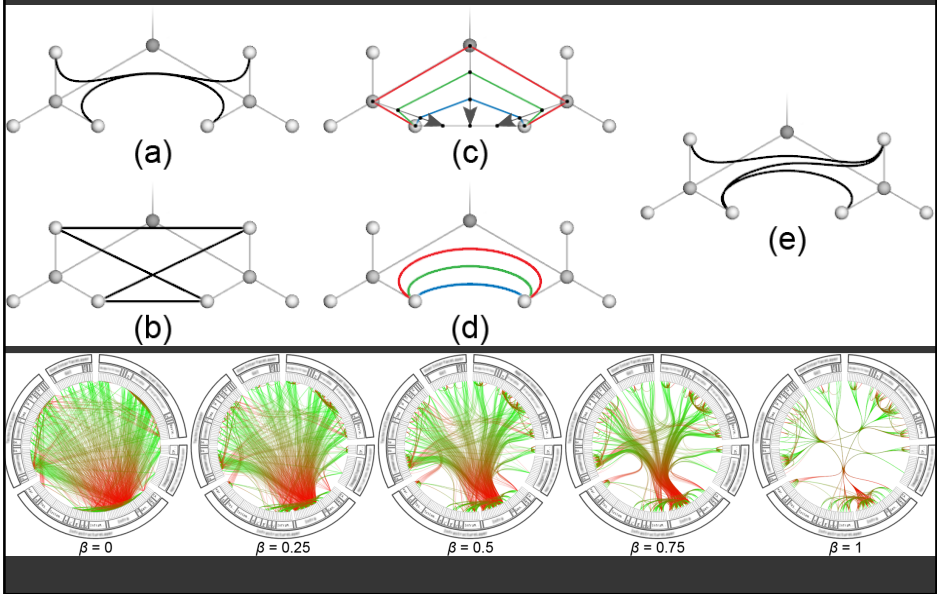
Bundle Edges along Hierarchy

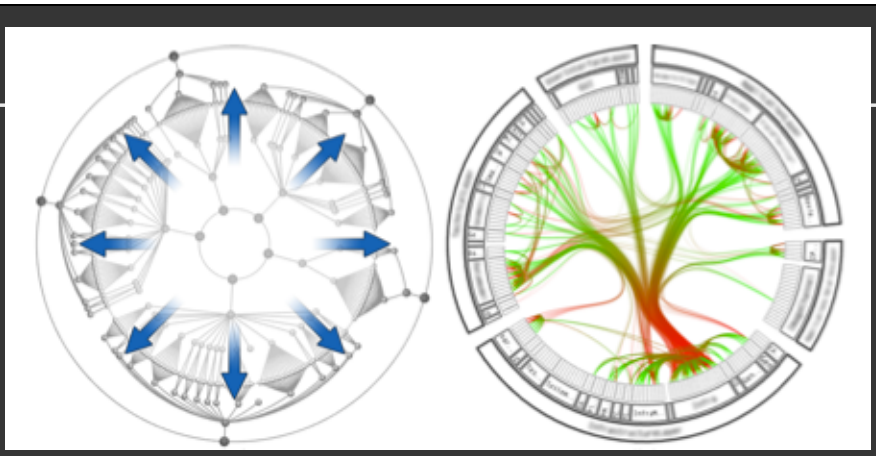


Bundle Edges along Hierarchy



Configuring Edge Tension

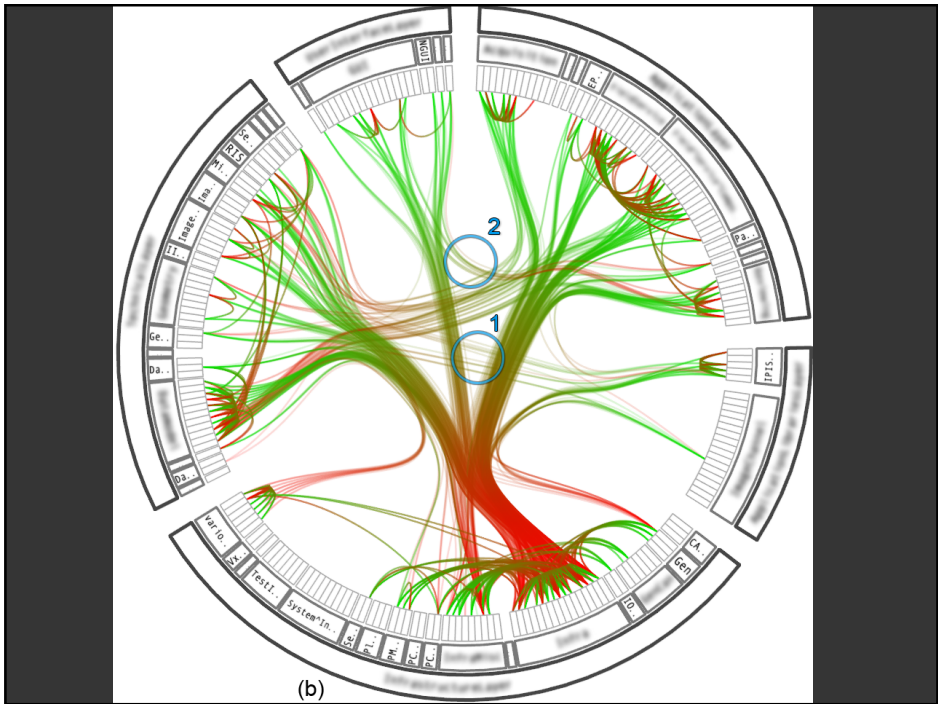




Use radial tree layout for inner circle

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