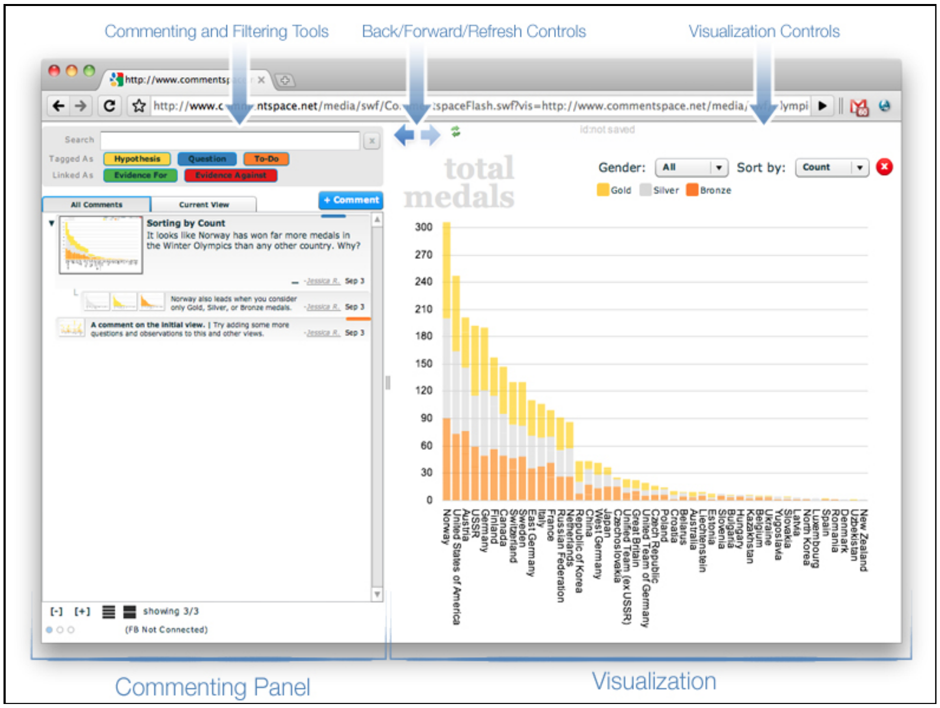
Graph Layout

Maneesh Agrawala

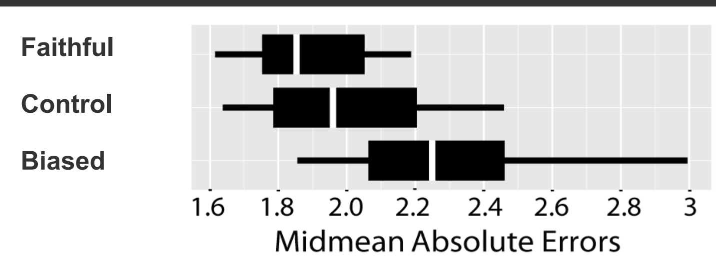
CS 448B: Visualization
Spring 2016

Last Time: Collaborative Visual Analysis



Results

Can having access to others' graphical judgments (accurate or not) impact subsequent users' judgments?



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: 5/11
- Project progress presentation: 5/23 in class (3-4 min) slide presentation
- Final poster presentation: 6/3 12:15-3:15pm Location: TBD
- Final paper: 6/5 11:59pm

Grading

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

In-Class Project Presentations

Dates: 5/23 In class presentation: 3 min per group

- Description of problem you are addressing
- Relevant prior work, how your work is different
- Description/storyboard and of approach
- A list of milestones for finishing the project by the deadline

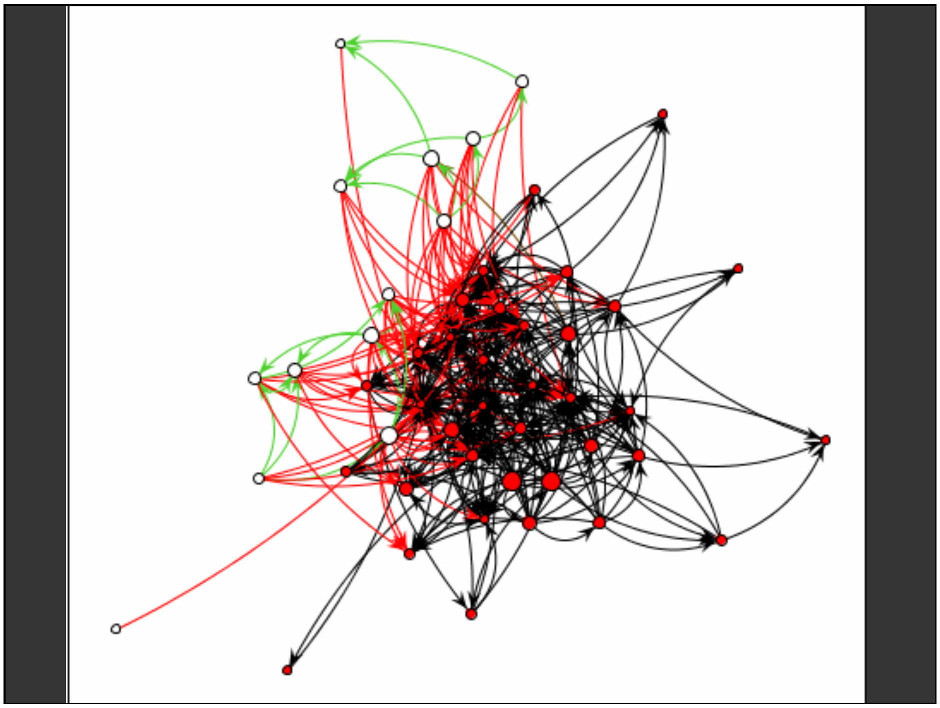
Scheduling

- Slides due 10am on 5/23 (make sure you can finish in 3 min)
- Split projects into two groups of 20 projects (you must attend)
- One group will present here, one in Gates 219
- All others should post feedback on piazza

Afterwards should post on the wiki

- Review of relevant prior work with full references
- List of milestones (scope your project appropriately)

Graph Layout

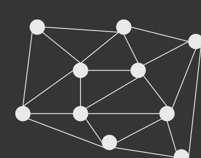


Graphs and Trees

Graphs

Model relations among data

Nodes and *edges*

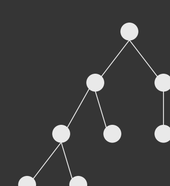


Trees

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

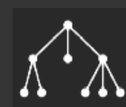
Tournaments
Organization Charts
Genealogy
Diagramming (e.g., Visio)
Biological Interactions (Genes, Proteins)
Computer Networks
Social Networks
Simulation and Modeling
Integrated Circuit Design

Tree Visualization

Indentation

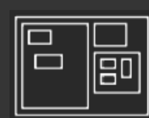
- Linear list, indentation encodes depth

Node-Link diagrams



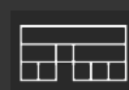
- Nodes connected by lines/curves

Enclosure diagrams



- Represent hierarchy by enclosure

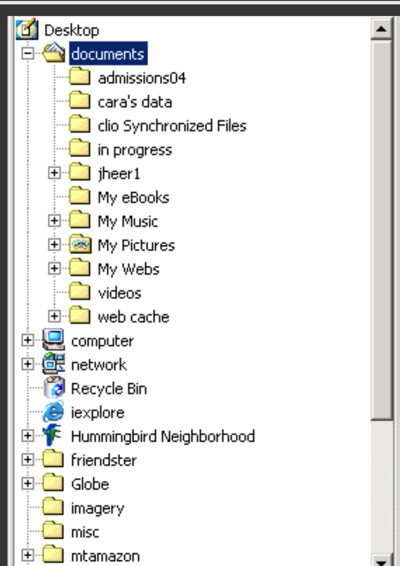
Layering



- 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

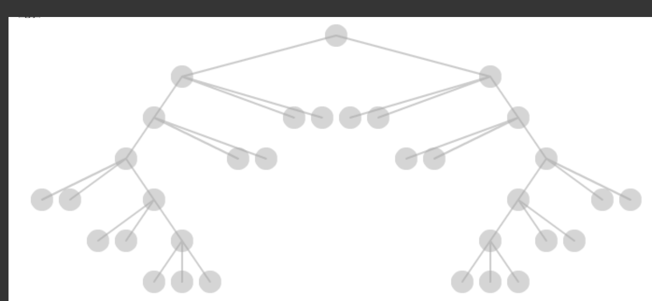
- Typically towards authority or generality

Basic Recursive Approach

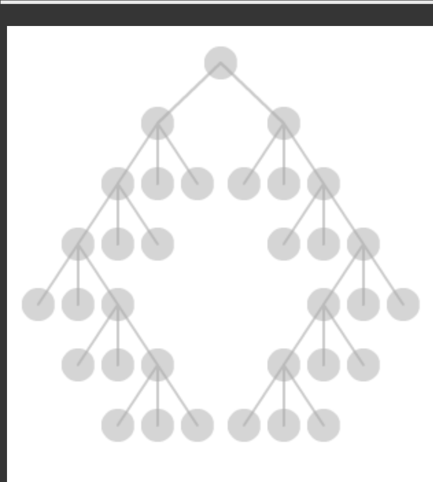
Repeatedly divide space for subtrees by leaf count

- 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

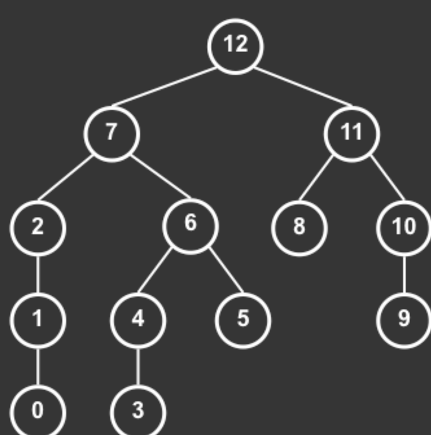
Merge left and right subtrees

- 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

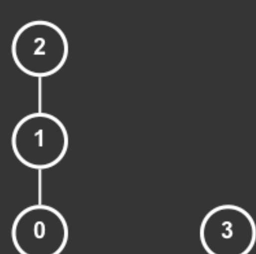


Reingold-Tilford Algorithm

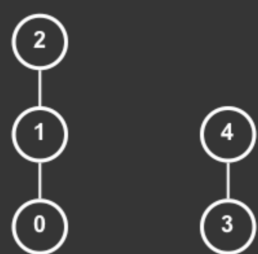
Reingold-Tilford Algorithm

Reingold-Tilford Algorithm

Reingold-Tilford Algorithm



Reingold-Tilford Algorithm



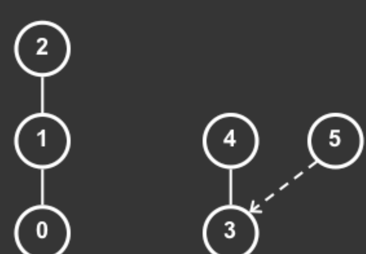
Reingold-Tilford Algorithm

Reingold-Tilford Algorithm

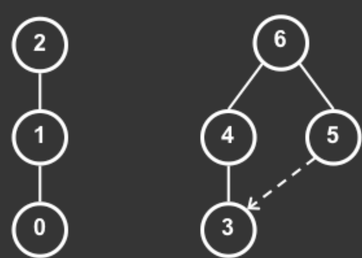
Reingold-Tilford Algorithm

Reingold-Tilford Algorithm

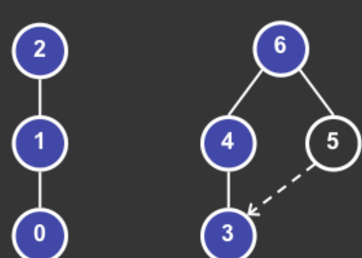
Reingold-Tilford Algorithm



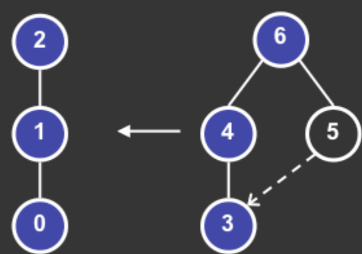
Reingold-Tilford Algorithm



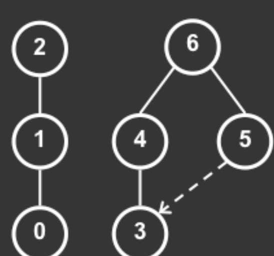
Reingold-Tilford Algorithm



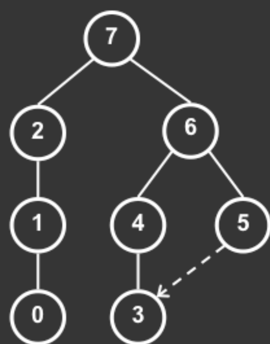
Reingold-Tilford Algorithm



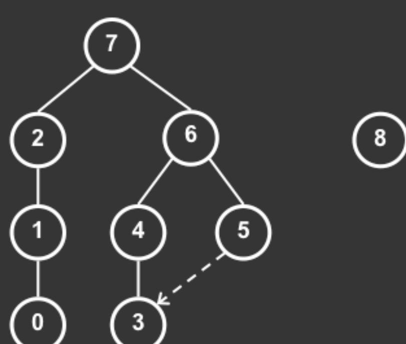
Reingold-Tilford Algorithm



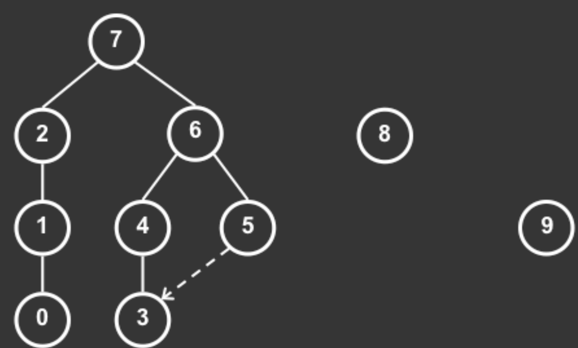
Reingold-Tilford Algorithm



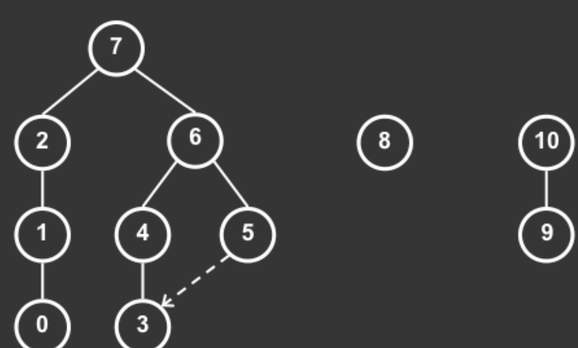
Reingold-Tilford Algorithm



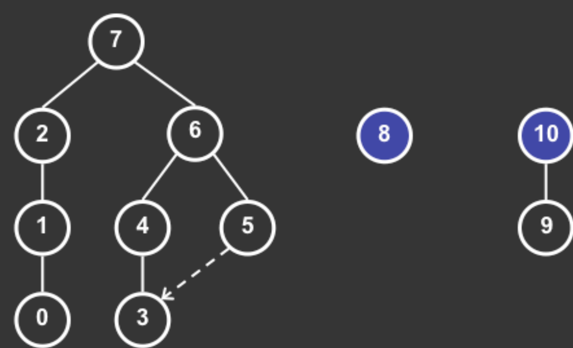
Reingold-Tilford Algorithm



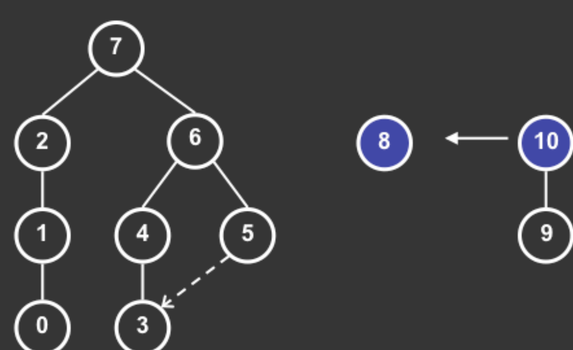
Reingold-Tilford Algorithm



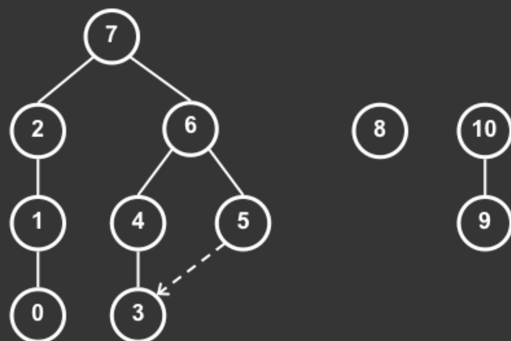
Reingold-Tilford Algorithm



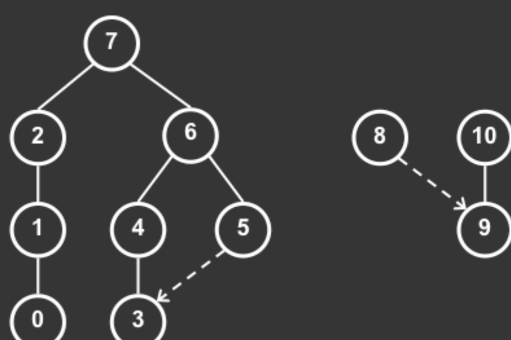
Reingold-Tilford Algorithm



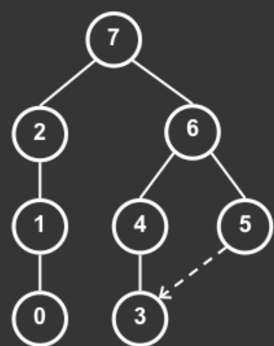
Reingold-Tilford Algorithm



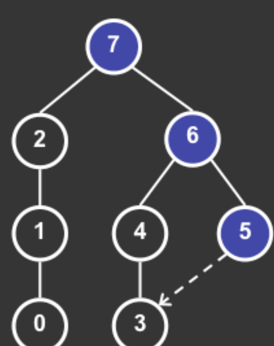
Reingold-Tilford Algorithm



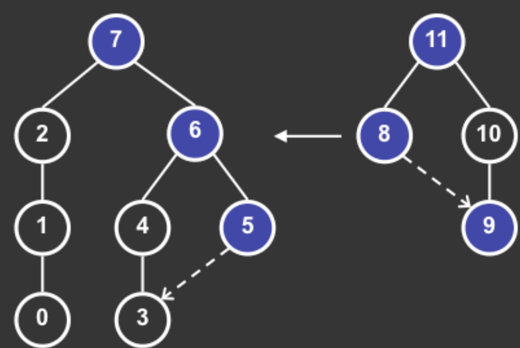
Reingold-Tilford Algorithm



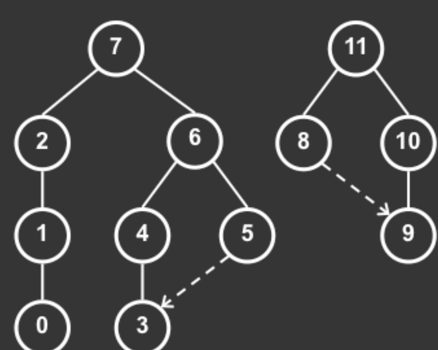
Reingold-Tilford Algorithm



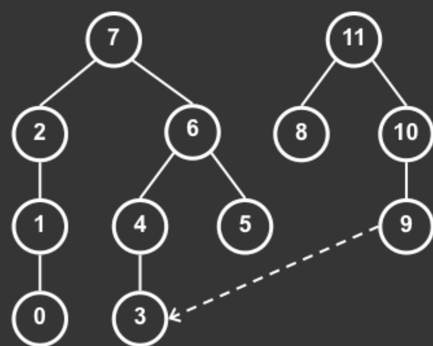
Reingold-Tilford Algorithm



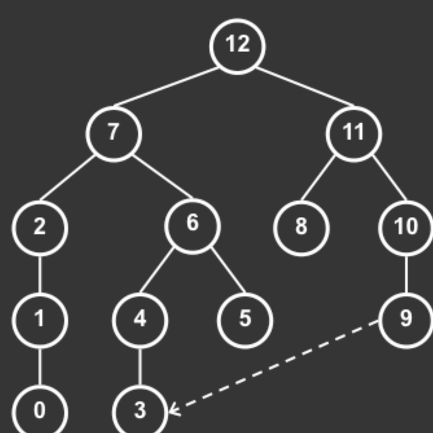
Reingold-Tilford Algorithm



Reingold-Tilford Algorithm



Reingold-Tilford Algorithm



Reingold-Tilford Algorithm

Linear algorithm – starts with bottom-up (postorder) pass

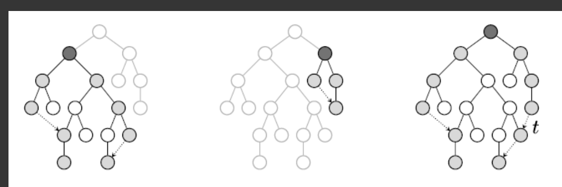
Set Y-coord by depth, arbitrary starting X-coord

Merge left and right subtrees

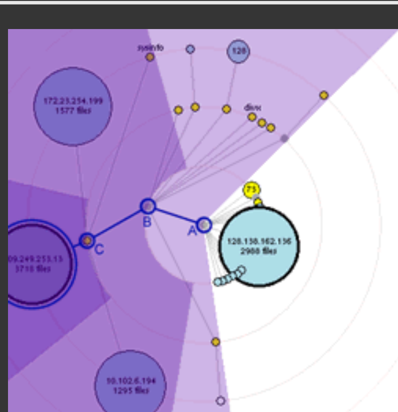
- Shift right as close as possible to left
  - Computed efficiently by maintaining subtree contours
- “Shifts” in position saved for each node as visited
- Parent nodes are centered above their children

Top-down pass (preorder) for assignment of final positions

- Sum of initial layout and aggregated shifts



Radial Layout



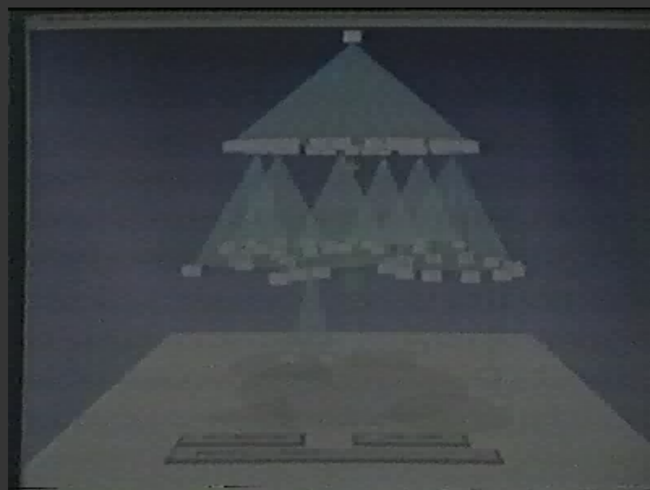
Node-link diagram in polar coords

Radius encodes depth root at center

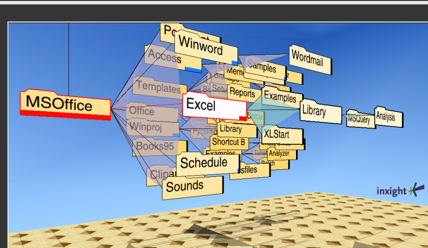
Angular sectors assigned to subtrees (recursive approach)

Reingold-Tilford approach can also be applied here

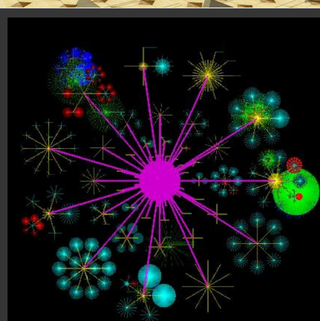
Circular Drawing of Trees in 3D



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

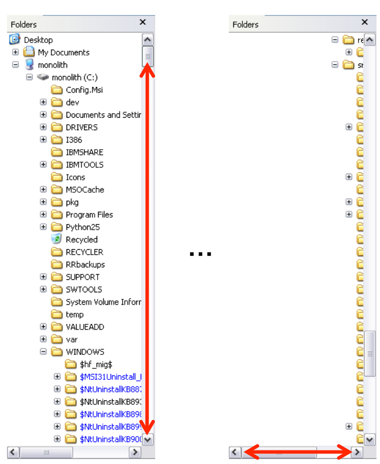
Scale

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

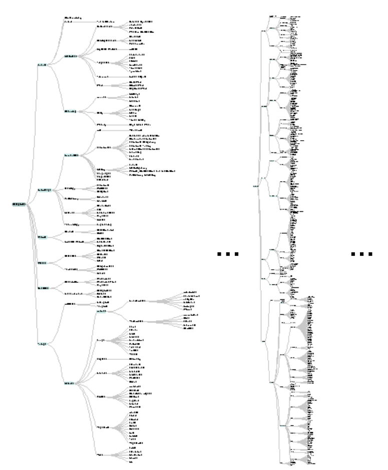
Possible solutions

- Filtering
- Focus+Context
- Scrolling or Panning
- Zooming
- Aggregation

Visualizing Large Hierarchies



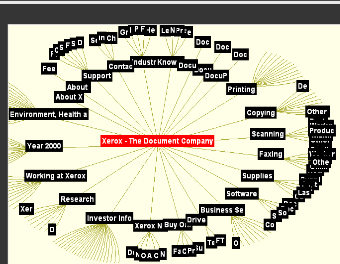
Indented Layout

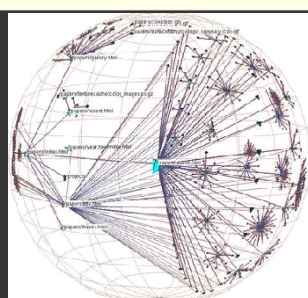


Reingold-Tilford Layout

MC Escher, *Circle Limit IV*

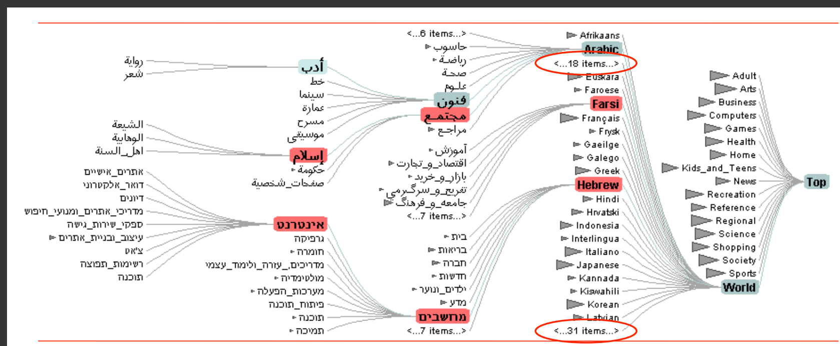
Hyperbolic Layout





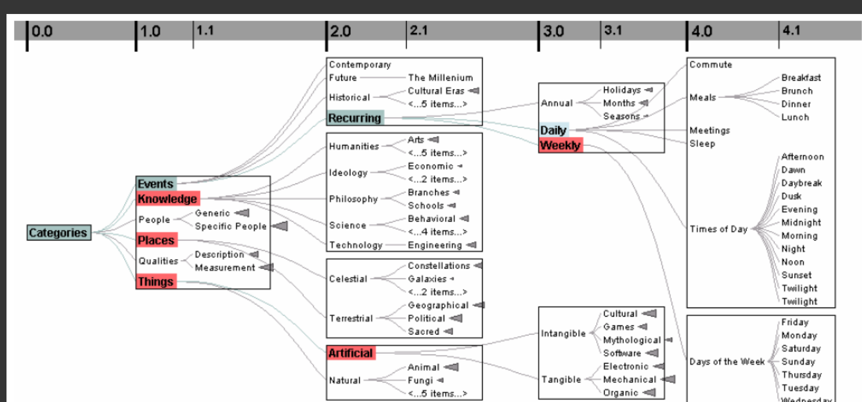
- Layout in hyperbolic space, then project on to Euclidean plane
- Why? Like tree breadth, the hyperbolic plane expands exponentially
- Also computable in 3D, projected into a sphere

Degree-of-Interest Trees [AVI 04]



Space-constrained, multi-focal tree layout

Degree-of-Interest Trees



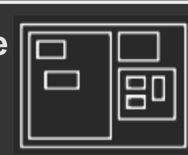
Cull “un-interesting” nodes on a per block basis until all blocks on a level fit within bounds

Attempt to center child blocks beneath parents

Enclosure Diagrams

Encode structure using spatial enclosure

Popularly known as TreeMaps



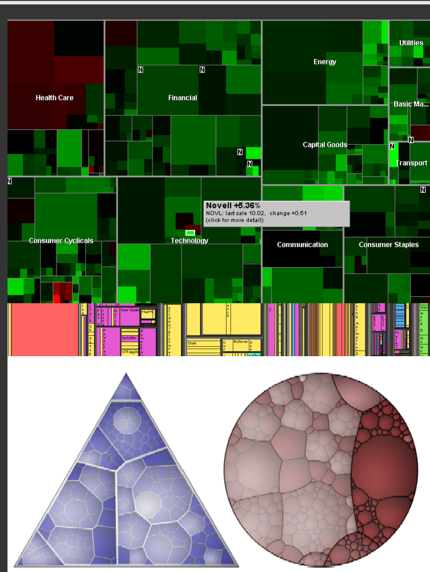
Benefits

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

Problems

- Difficult to accurately read depth

TreeMaps



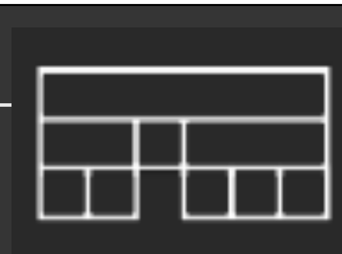
Recursively fill space based on node size

Enclosure signifies hierarchy

Additional measures to control aspect ratio of cells

Often uses rectangles, but other shapes are possible, e.g., iterative Voronoi tesselation.

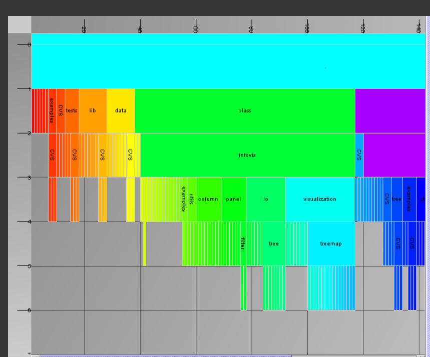
Layered Diagrams

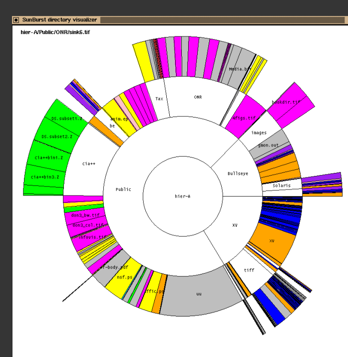


Signify tree structure using
Layering
Adjacency
Alignment

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

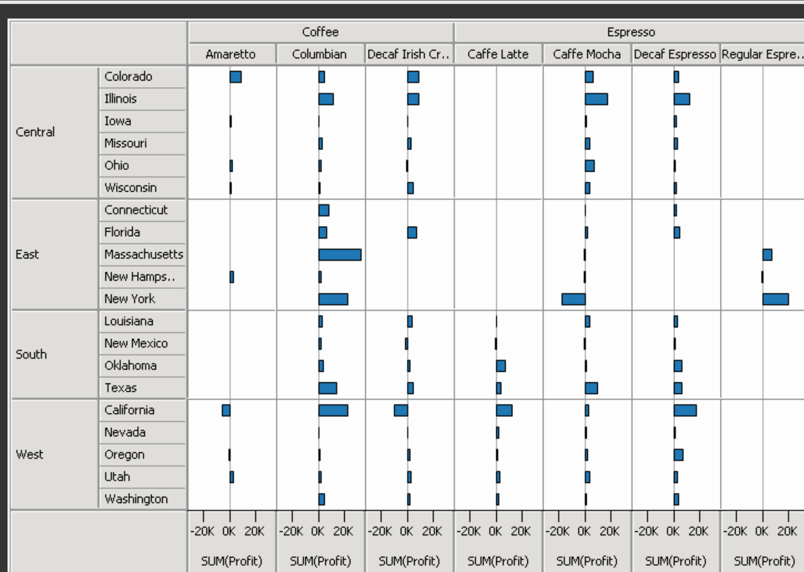
Icicle and Sunburst Trees



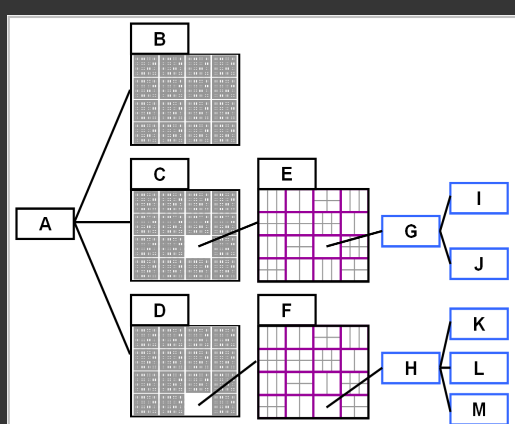


Higher-level nodes get a larger layer area, whether that is horizontal or angular extent
Child levels are layered, constrained to parent's extent

Layered Tree Drawing



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

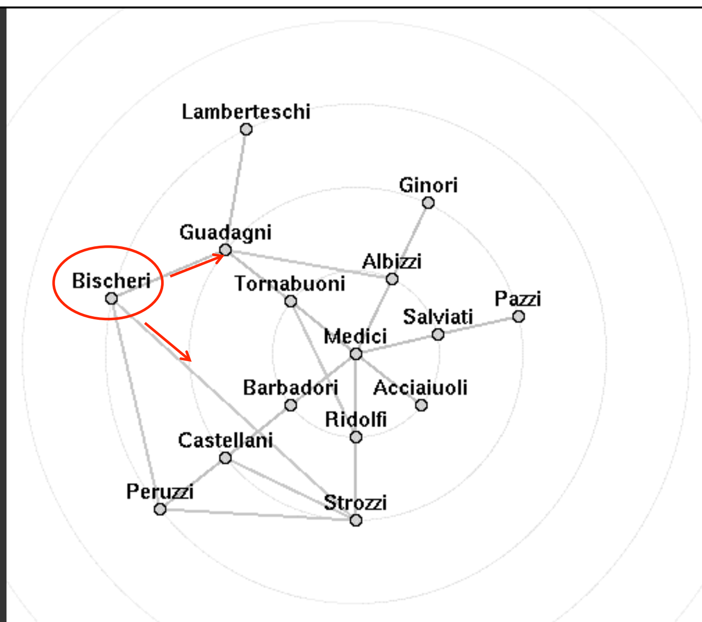
- Websites, Social Networks

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

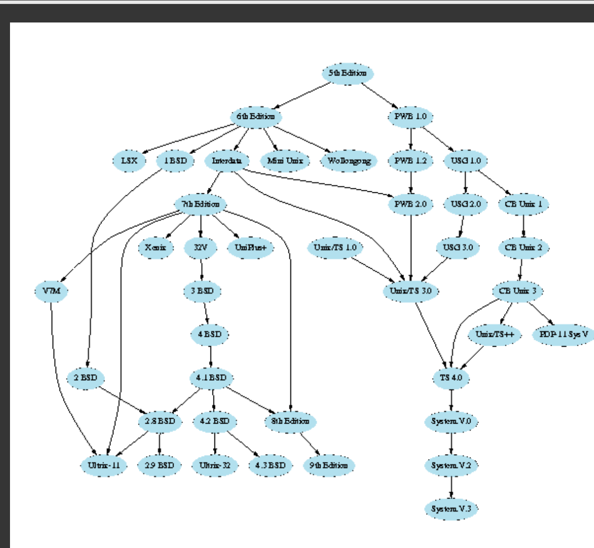


Spanning tree layout may result in arbitrary parent node

Sugiyama-style graph layout

Evolution of the UNIX operating system

Hierarchical layering based on descent



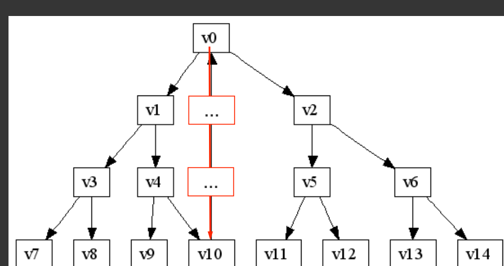
Sugiyama-style graph layout

Layer 1

Layer 2

Layer 3

Layer 4



Reverse some edges to remove cycles

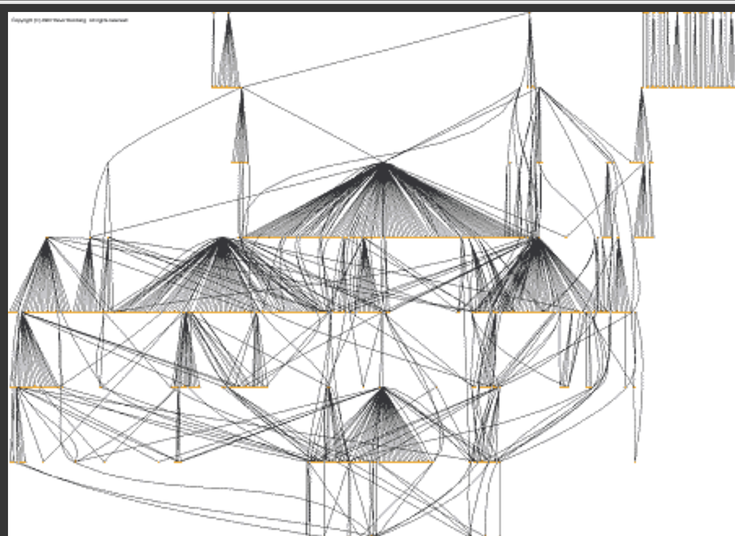
Assign nodes to hierarchy layers → Longest path layering

Create dummy nodes to “fill in” missing layers

Arrange nodes within layer, minimize edge crossings

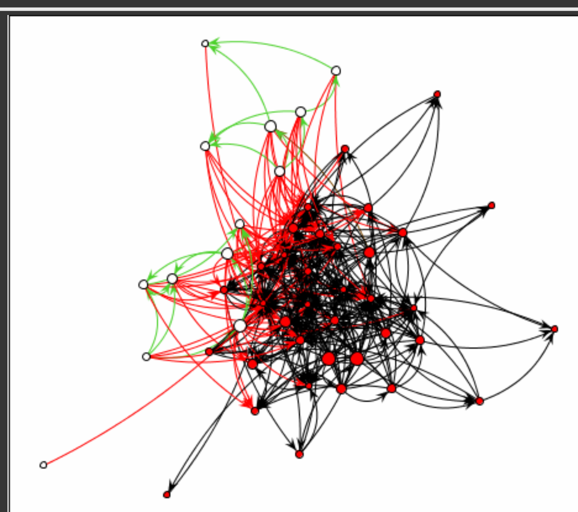
Route edges – layout splines if needed

Hierarchical graph layout

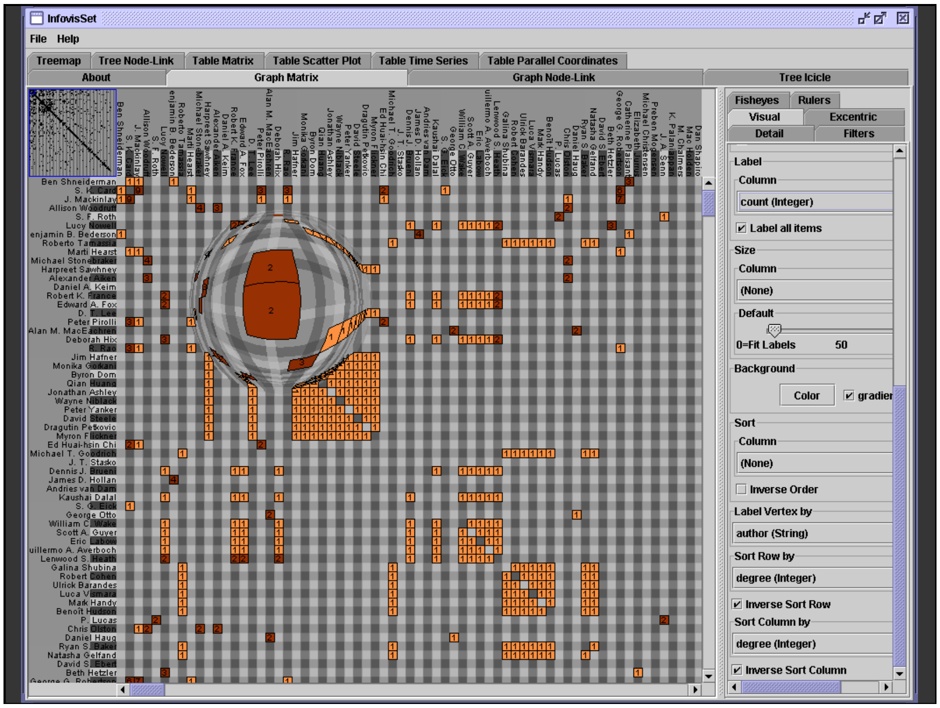


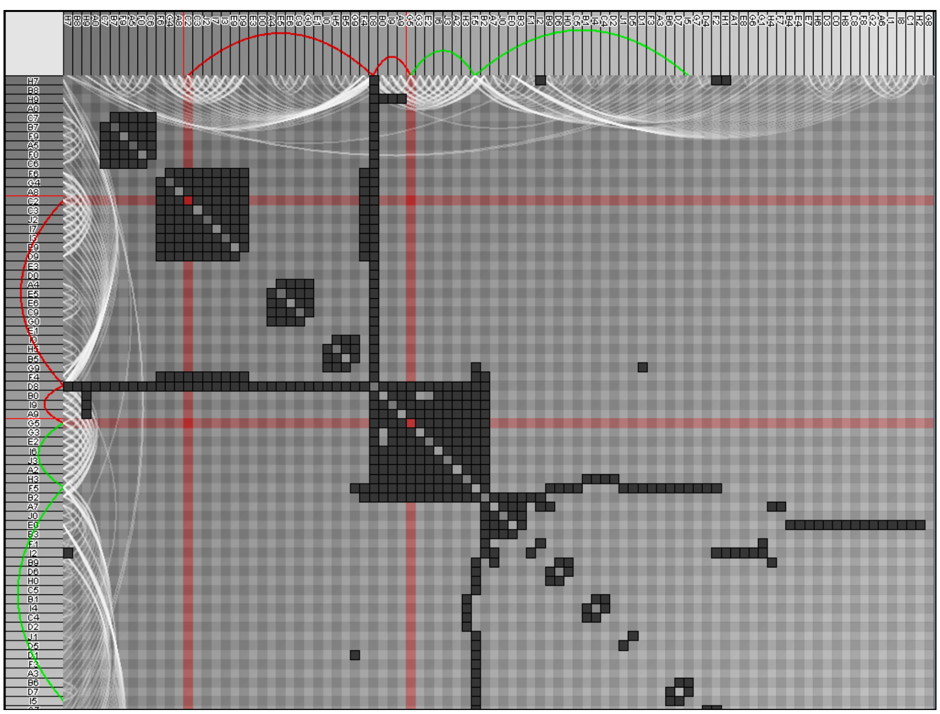
Gnutella network

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
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

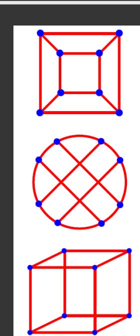
Common approach for undirected graphs

- *Force-Directed Layout* most common

Can also introduce directional constraints

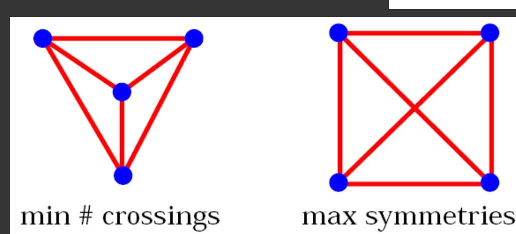
- *DiG-CoLa* (Di-Graph Constrained Optimization Layout) [Dwyer 05]

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.

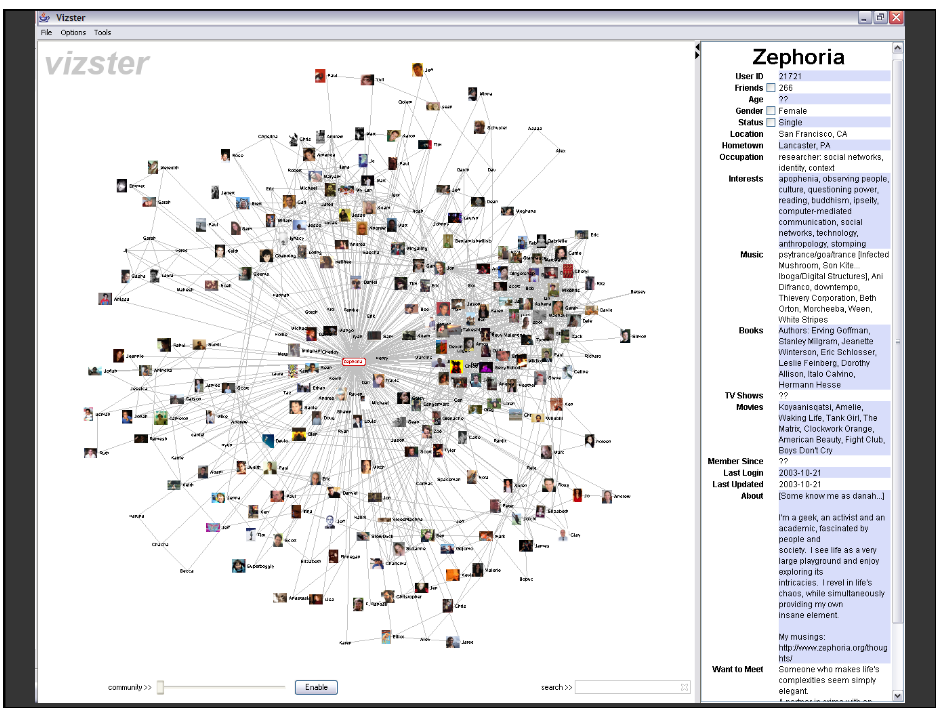
Force-Directed Layout

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

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

- 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

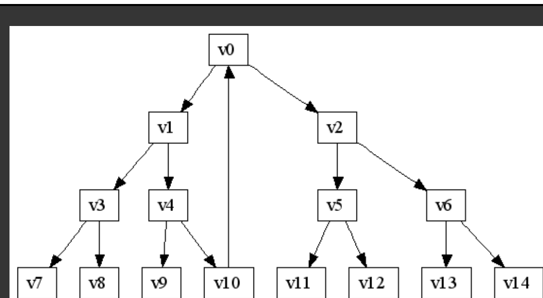
$$E_H(y) = \Sigma_{(i,j) \in E} (y_i - y_j - \delta_{ij})^2$$

- 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**

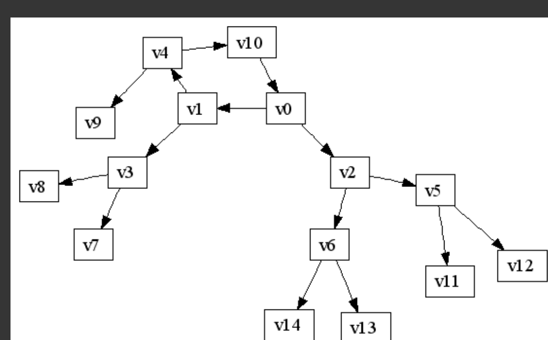
Sugiyama layout (dot)

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

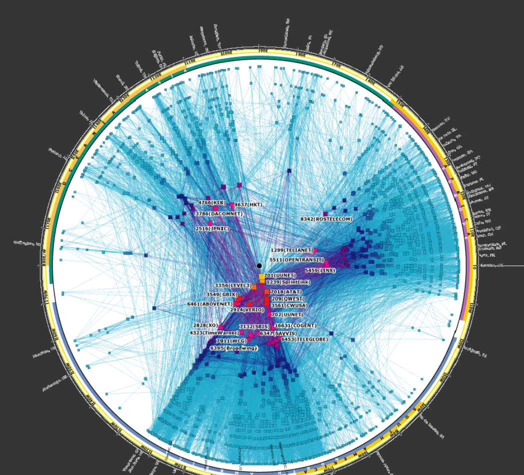
- Internet Connectivity
- Radial Scatterplot

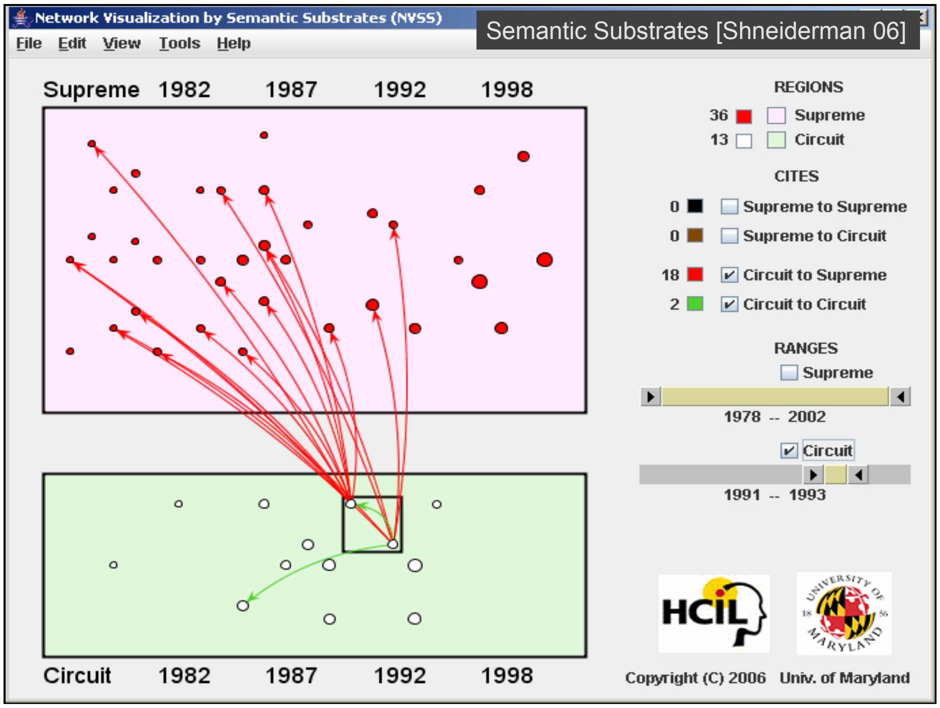
Angle = Longitude

- Geography

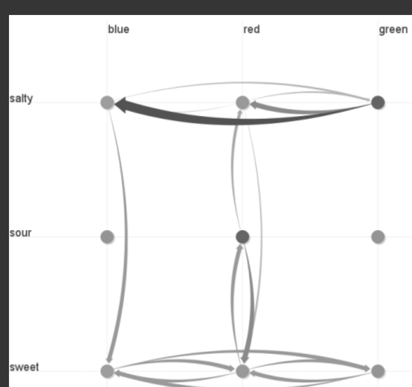
Radius = Degree

- # of connections
- (a statistic of the nodes)



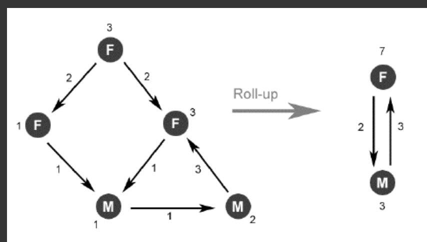


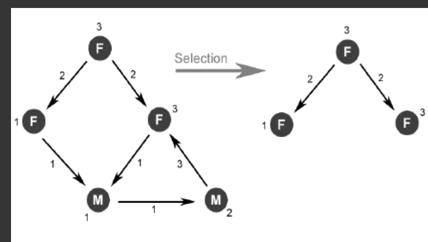
PivotGraph [Wattenberg 2006]



Layout aggregated graphs according to node attributes

Operators





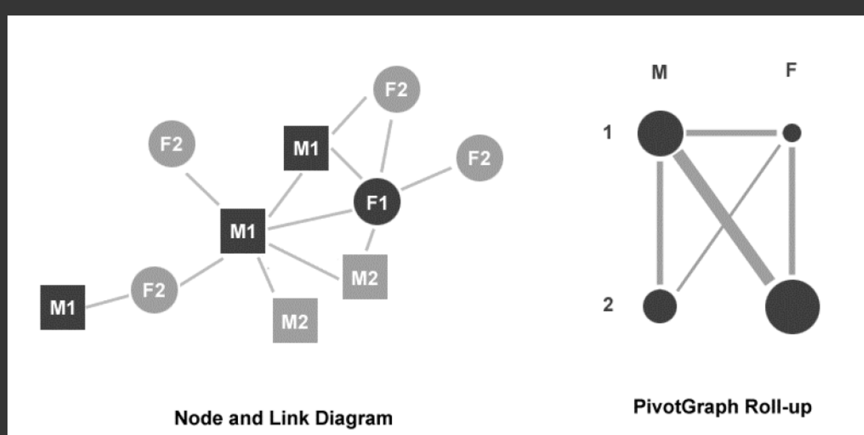
Roll-Up

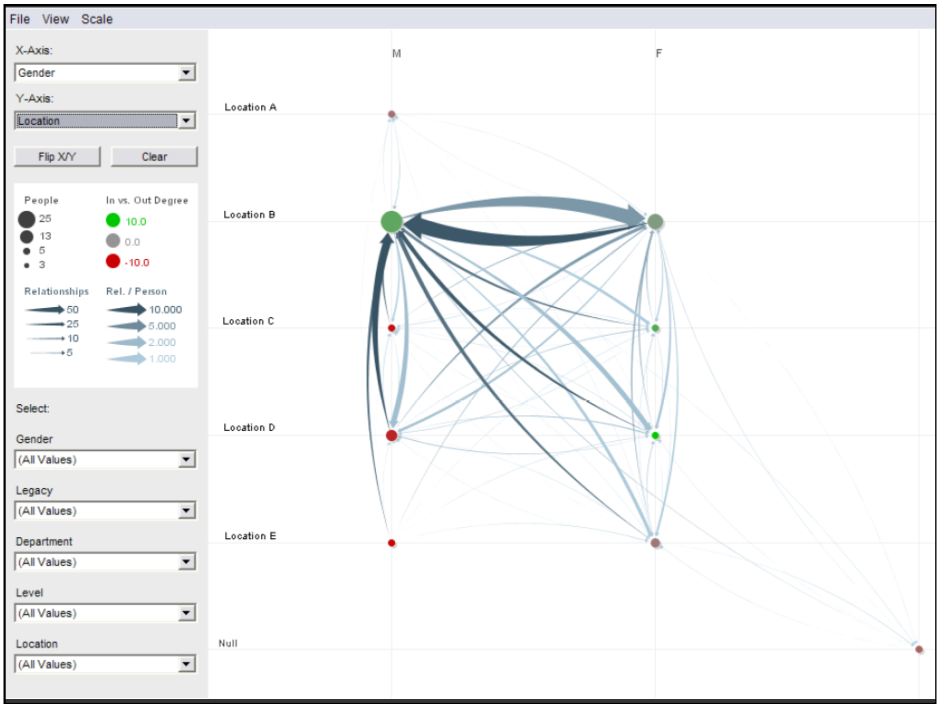
Aggregate items with matching data values

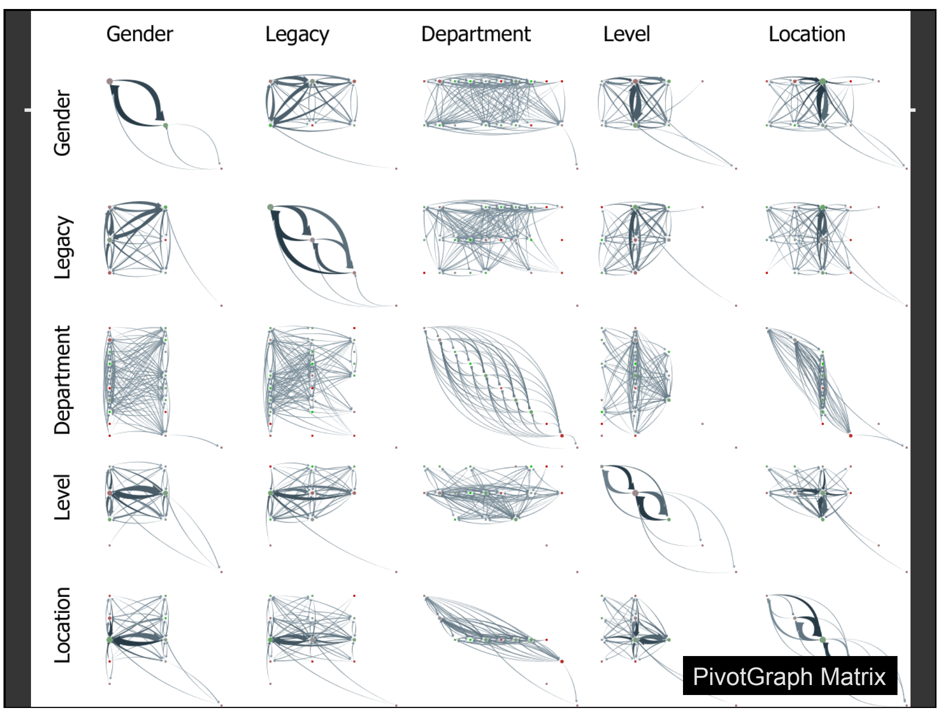
Selection

Filter on data values

PivotGraph




File View Scale
X-Axis:
Gender
Y-Axis:
Location
Flip X/Y
Clear
People
In vs. Out Degree
25
13
5
3
10.0
0.0
-10.0
Relationships
Rel. / Person
50
25
10
5
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


Gender
Legacy
Department
Level
Location
Gender
Legacy
Department
Level
Location
PivotGraph Matrix

Limitations of PivotGraph

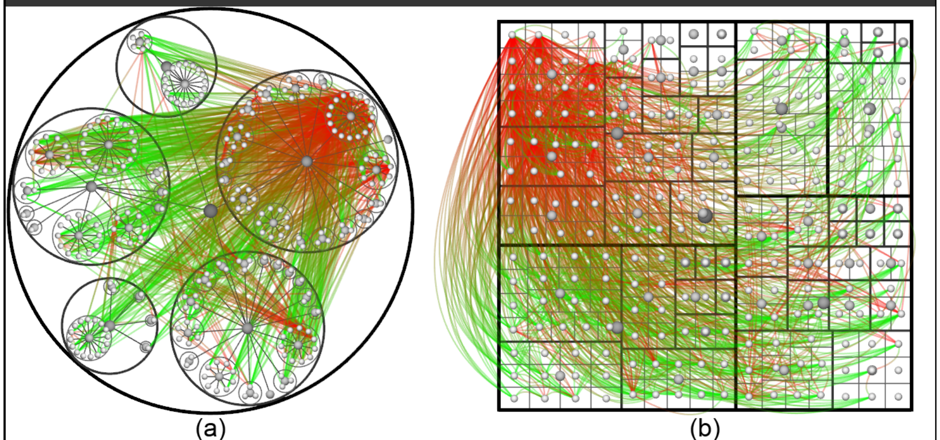
Only 2 variables (no nesting as in Tableau)

Doesn't support continuous variables

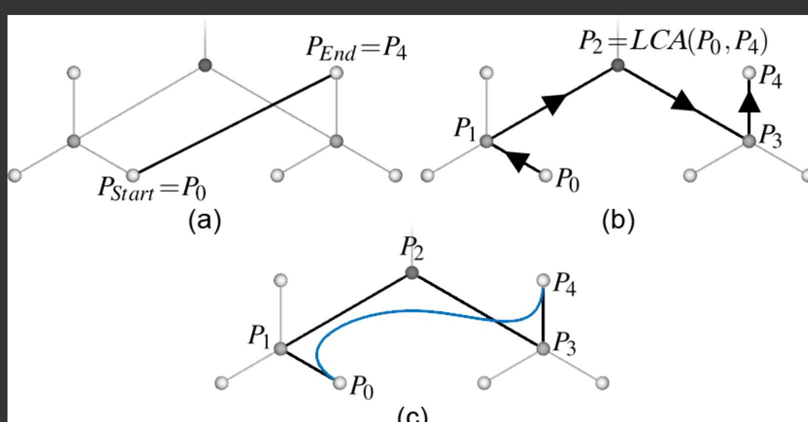
Multivariate edges?

Hierarchical Edge Bundles

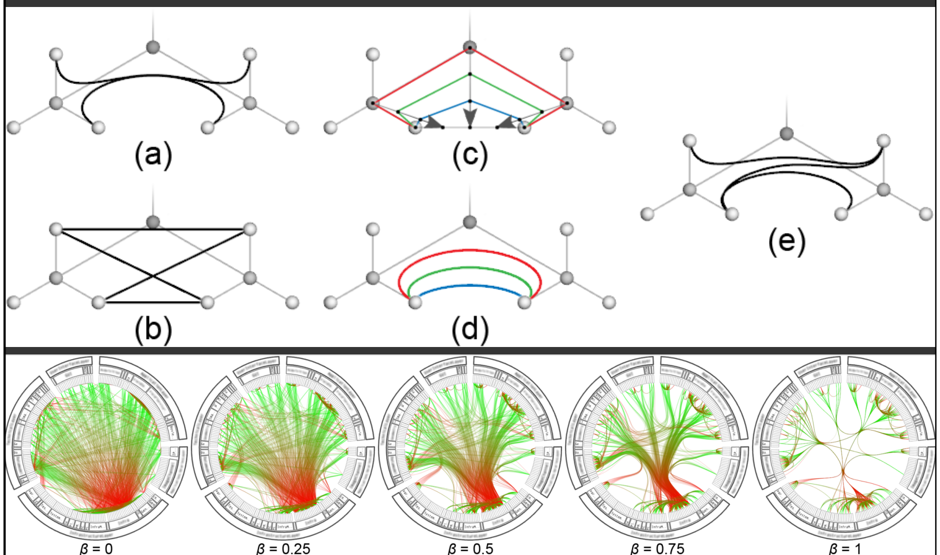
Trees with Adjacency Relations

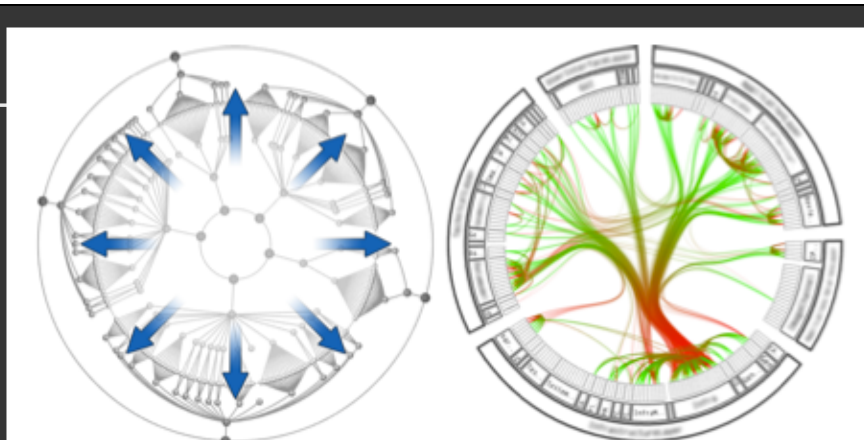


Bundle Edges along Hierarchy



Configuring Edge Tension

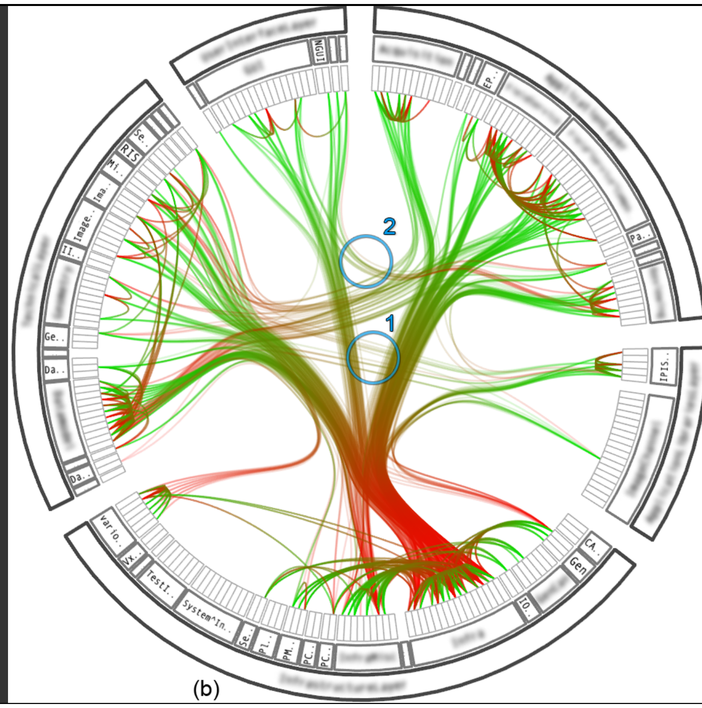




Use radial tree layout for inner circle

Mirror to outside

Replace inner tree with hierarchical edge bundles



(b)

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