CDS 6324
DATA VISUALIZATION

Lecture 10: Graph & Tree Visualization

Outline

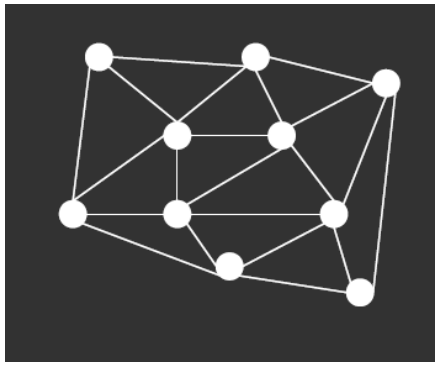
- **Visualizing Trees**
- **Visualizing Graphs**

Goals

- Overview of layout approaches
- Assess strengths and weaknesses
- Insight into implementation techniques

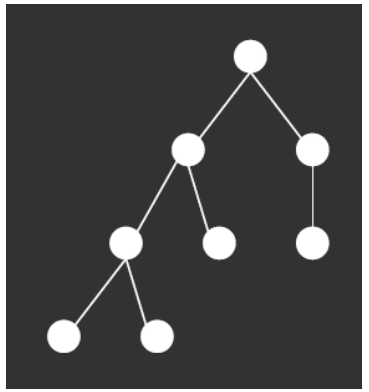
Graphs and Trees

- **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

- A primary concern of graph drawing is the spatial arrangement of nodes and edges.
- Often (but not always) the goal is to effectively depict the graph structure:
  - Connectivity, path-following
  - Network distance
  - Clustering
  - Ordering (e.g., hierarchy level)

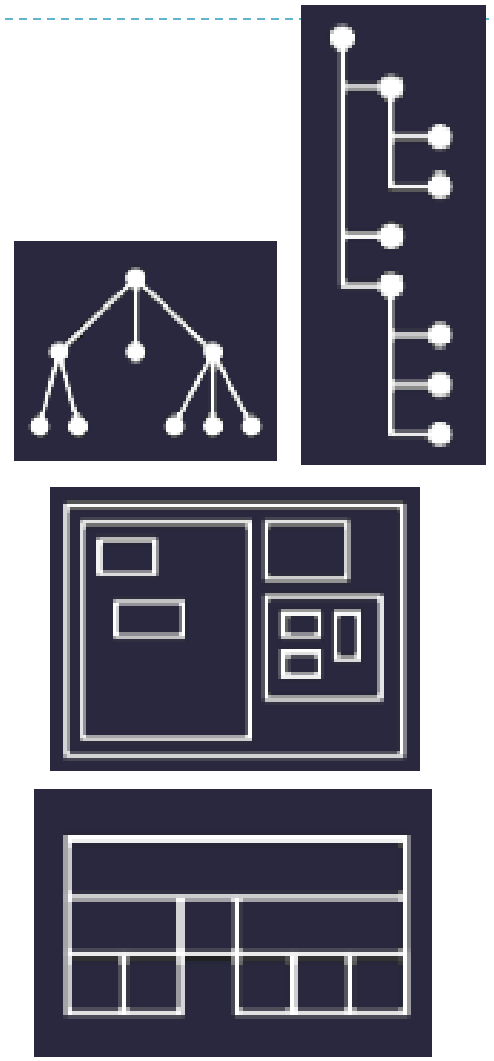
Applications

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

Tree Layout

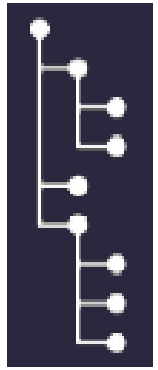
Tree Visualization

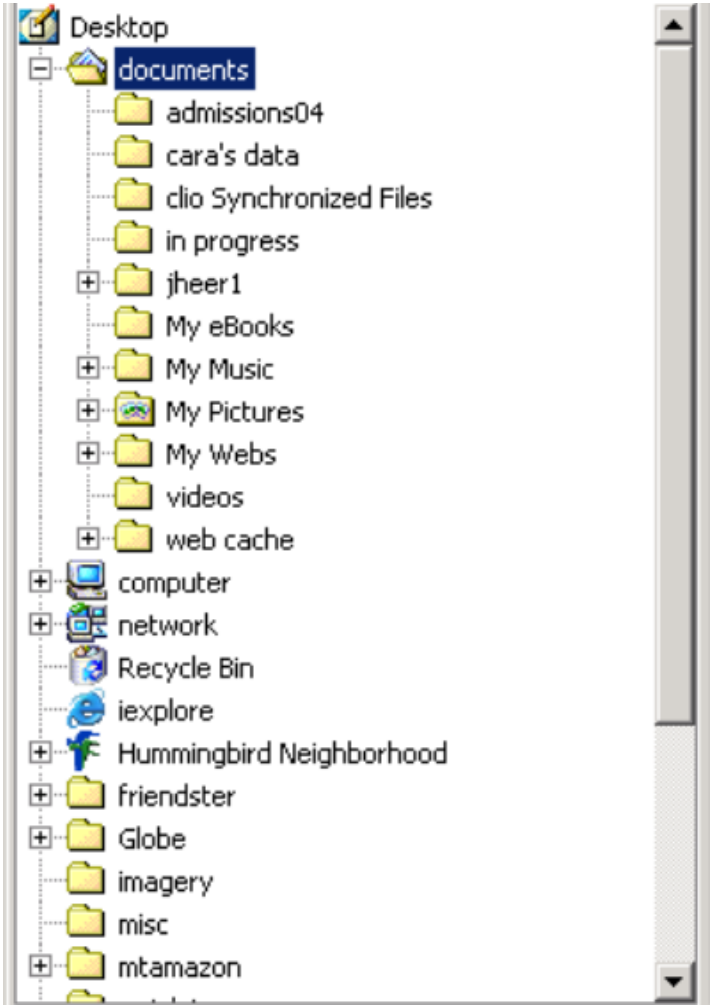
- **Indentation**
  - Linear list, indentation encodes depth
- **Node-Link diagrams**
  - Nodes connected by lines/curves
- **Enclosure diagrams**
  - Represent hierarchy by enclosure
- **Layering**
  - Relative position and alignment



Indentation

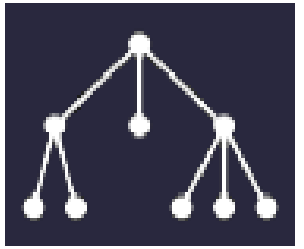
- Places all items along vertically spaced rows
- Indentation used to show parent/child relationships
- Commonly used as a component in an interface
- Breadth and depth contend for space
- Often requires a great deal of scrolling





Node-Link Diagram

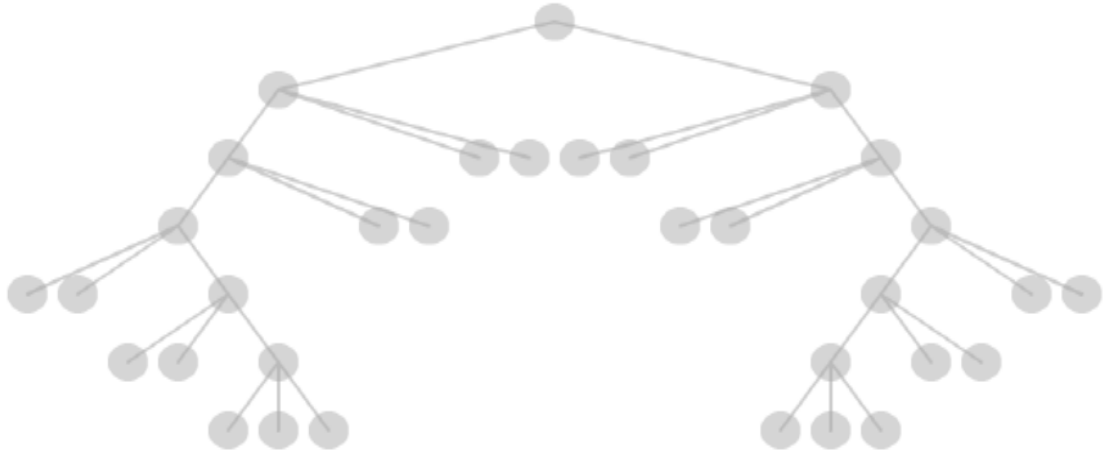
- Nodes are distributed in space, connected by straight or curved lines
- Typical approach is to use 2D space to break apart breadth and depth
- Often space is used to communicate hierarchical orientation (e.g., towards authority or generality)



Node-Link Diagram

Basic Recursive Approach

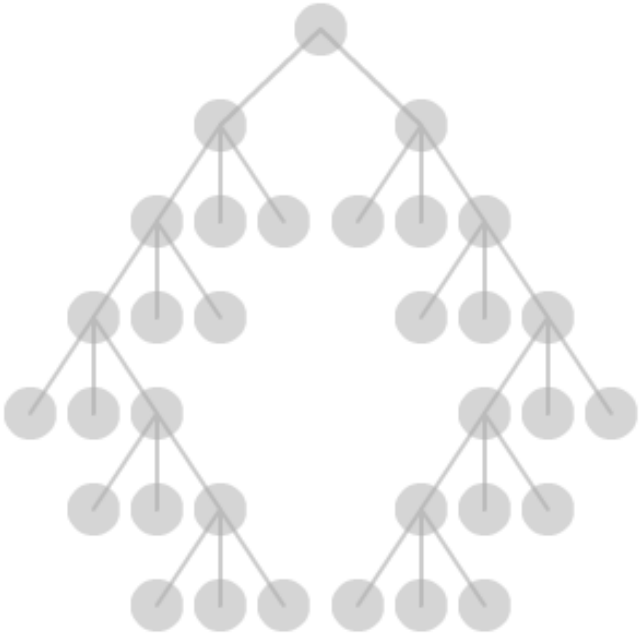
- Repeatedly divide space for subtrees by leaf count
  - Breadth of tree along one dimension
  - Depth along the other dimension
- **Problem**: exponential growth of breadth



Node-Link Diagram

Reingold & Tilford's "Tidy" Layout

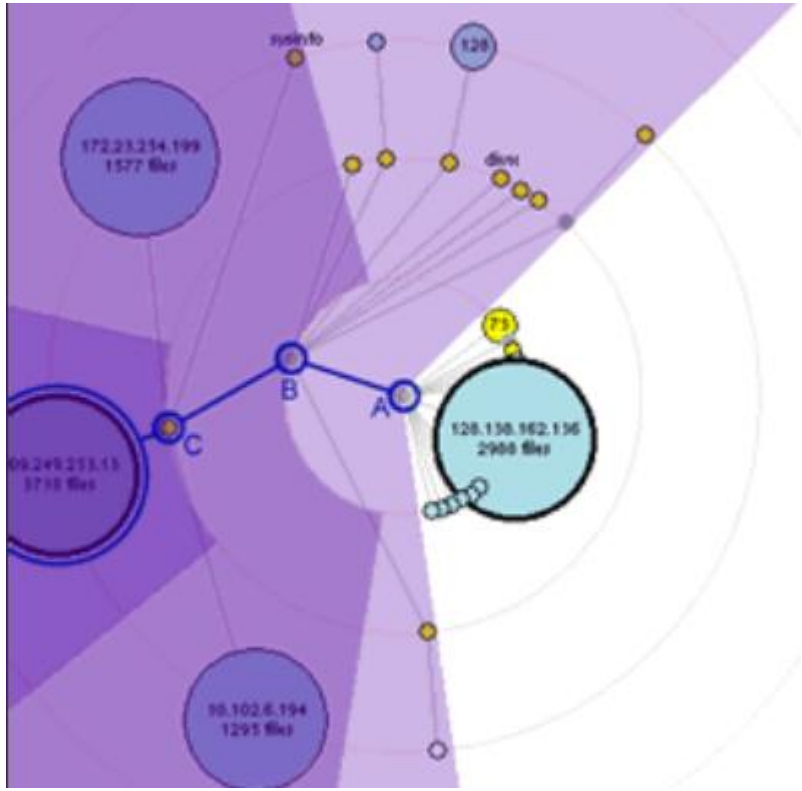
- **Goal**: make smarter use of space, maximize density and symmetry.
- Originally binary trees, extended by Walker to cover general case.
- Corrected by Buchheim et al. to achieve a linear time algorithm.



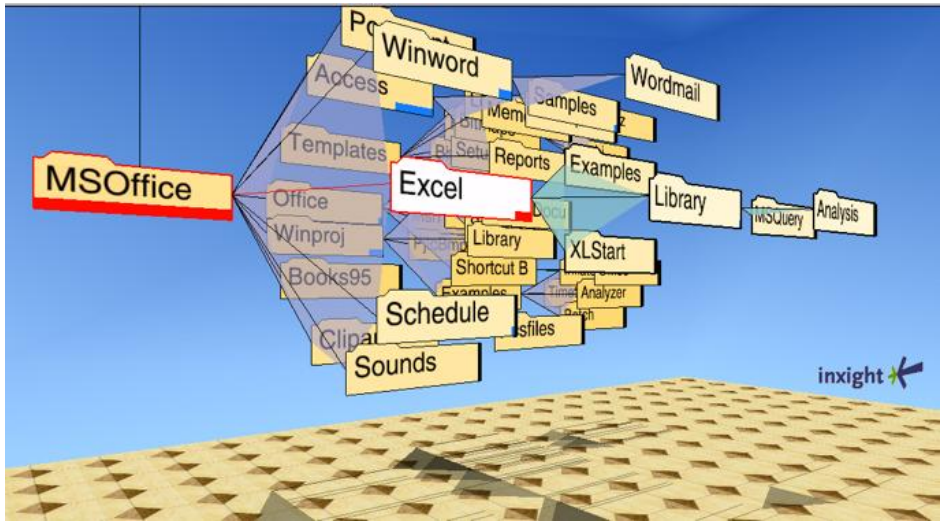
Node-Link Diagram

Radial Layout

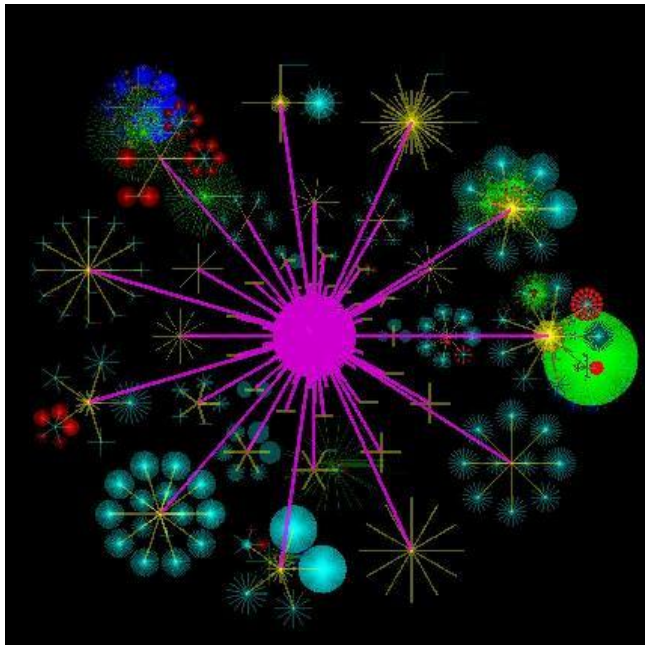
- Node-link diagram in polar co-ordinates.
- Radius encodes depth, with root in the center.
- Angular sectors assigned to subtrees (typically uses recursive approach).
- Reingold-Tilford method could be applied here.



Node-Link Diagram

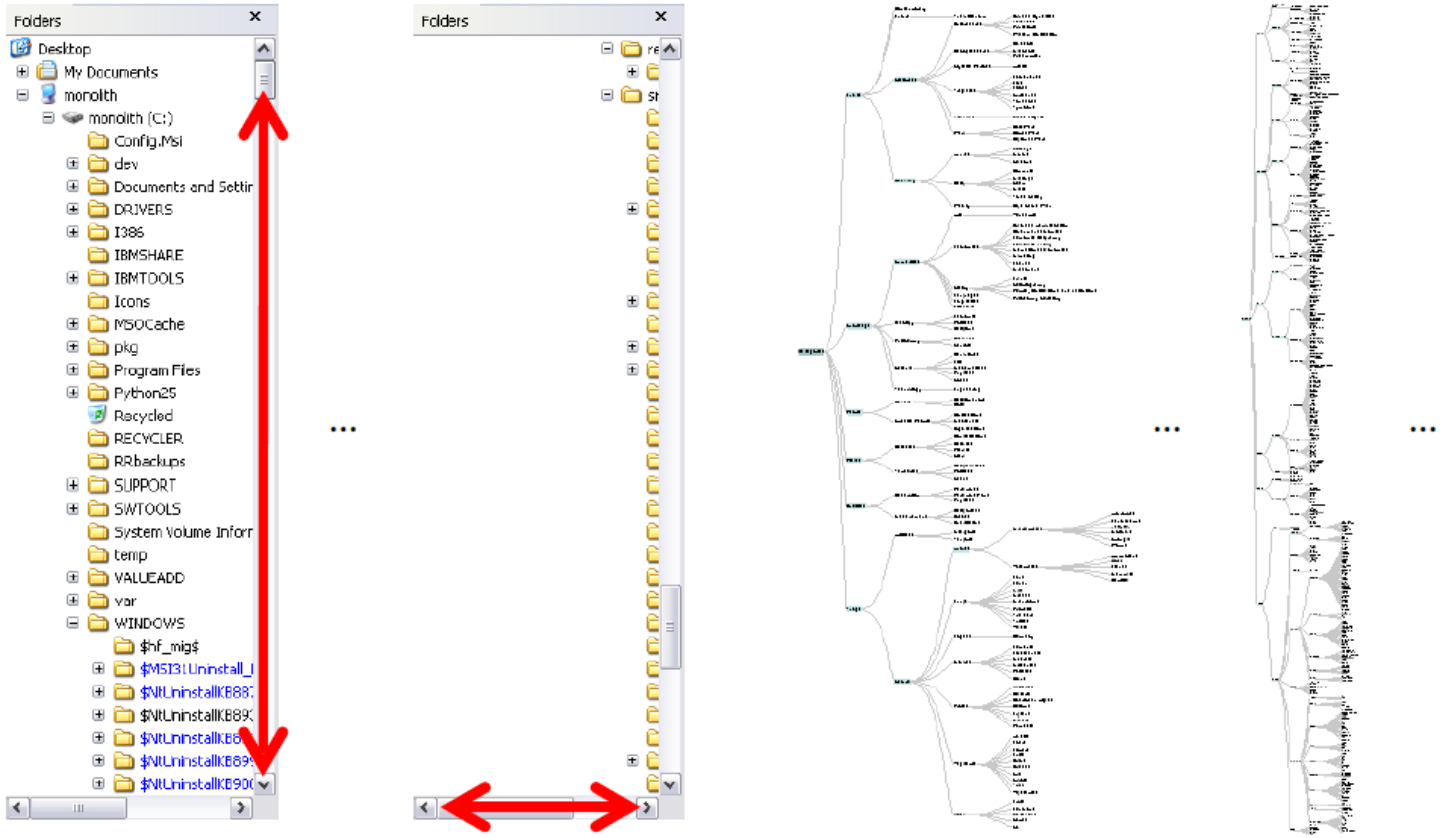


Layout in 3D to form Cone Trees.



Node-Link Diagram

Visualizing Large Hierarchies



Indented Layout

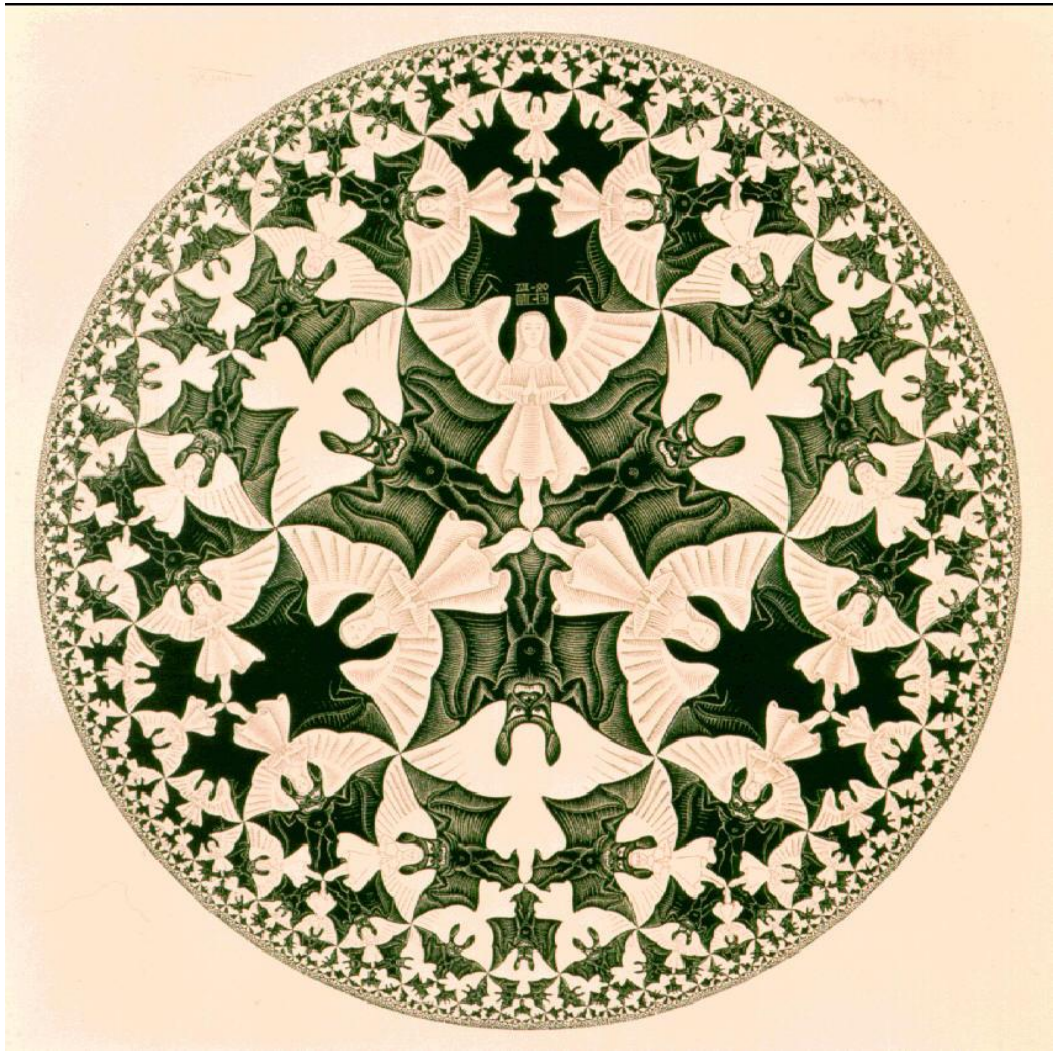
Reingold-Tilford Layout

Node-Link Diagram

More Nodes, More Problems…

- **Tree breadth often grows exponentially**
  - Even with tidy layout, quickly run out of space
- **Possible Solutions**
  - Filtering
  - Focus+Context
  - Scrolling or Panning
  - Zooming
  - Aggregation

Node-Link Diagram

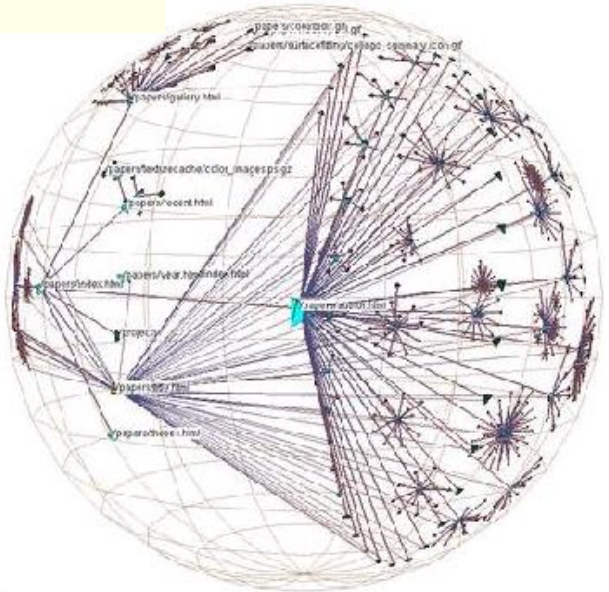


MC Escher, *Circle Limit IV*

Node-Link Diagram

Hyperbolic Layout

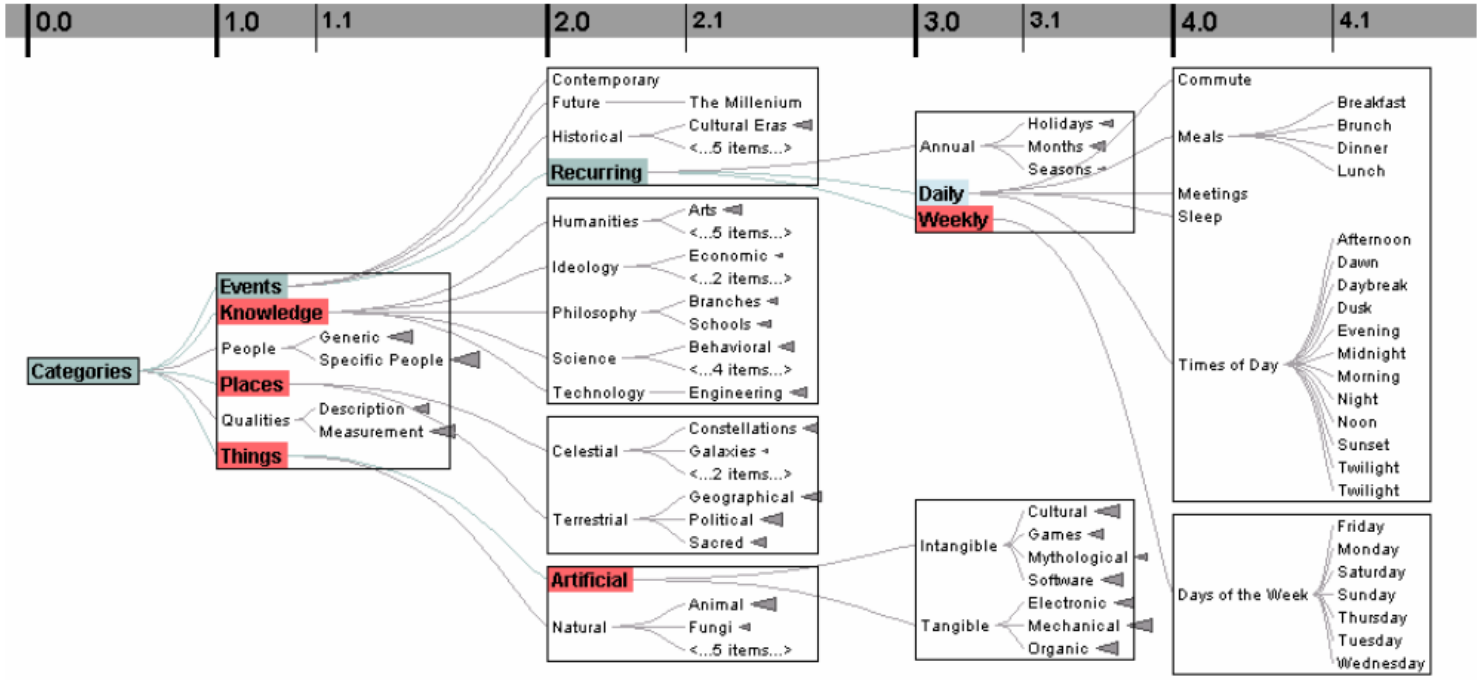
- Perform tree layout in hyperbolic geometry, project the result on to the Euclidean plane.
- Why? Like tree breadth, the hyperbolic plane expands exponentially!
- Also computable in 3D, projected into a sphere.



https://en.wikipedia.org/wiki/Hyperbolic_tree

Node-Link Diagram

Degree-of-Interest Trees



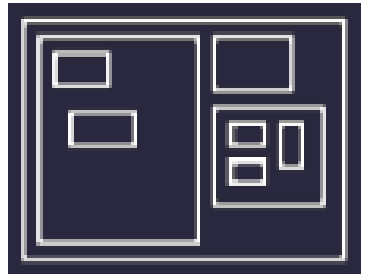
Enclosure Diagram

- Encode structure using **spatial enclosure.**
  Popularly known as **treemaps**

- **Benefits**

  Provides a single view of an entire tree

  Easier to spot large/small nodes



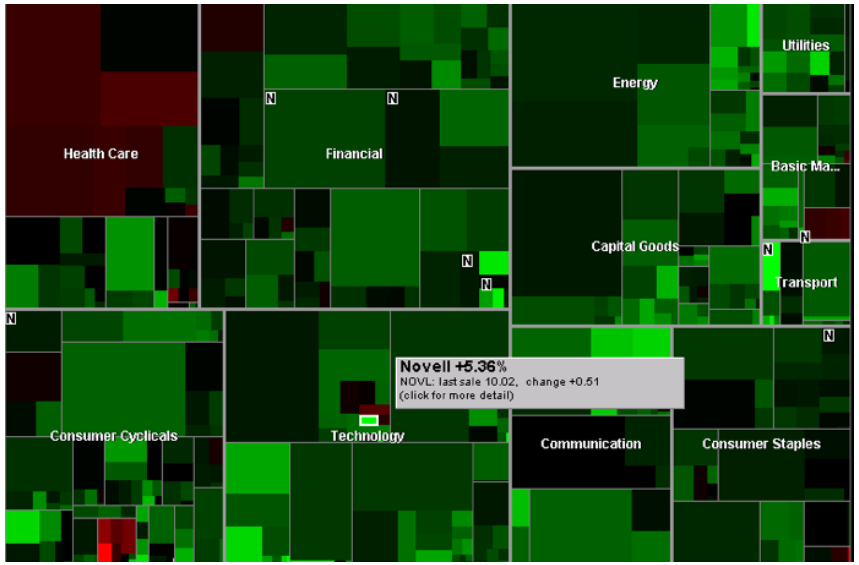
- **Problems**

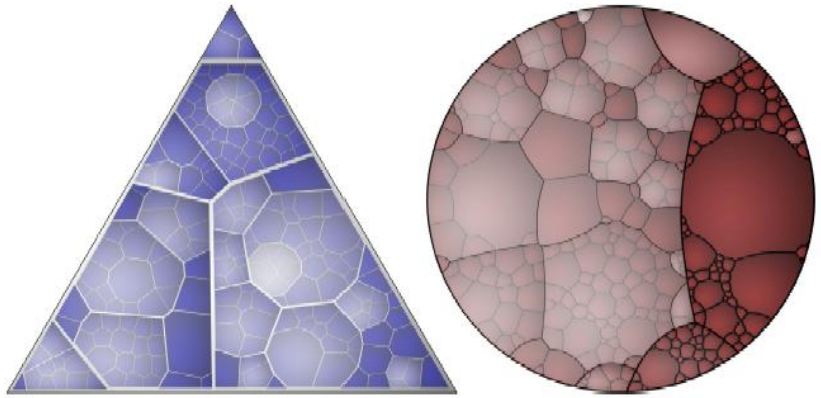
  Difficult to accurately read structure / depth

Enclosure Diagram

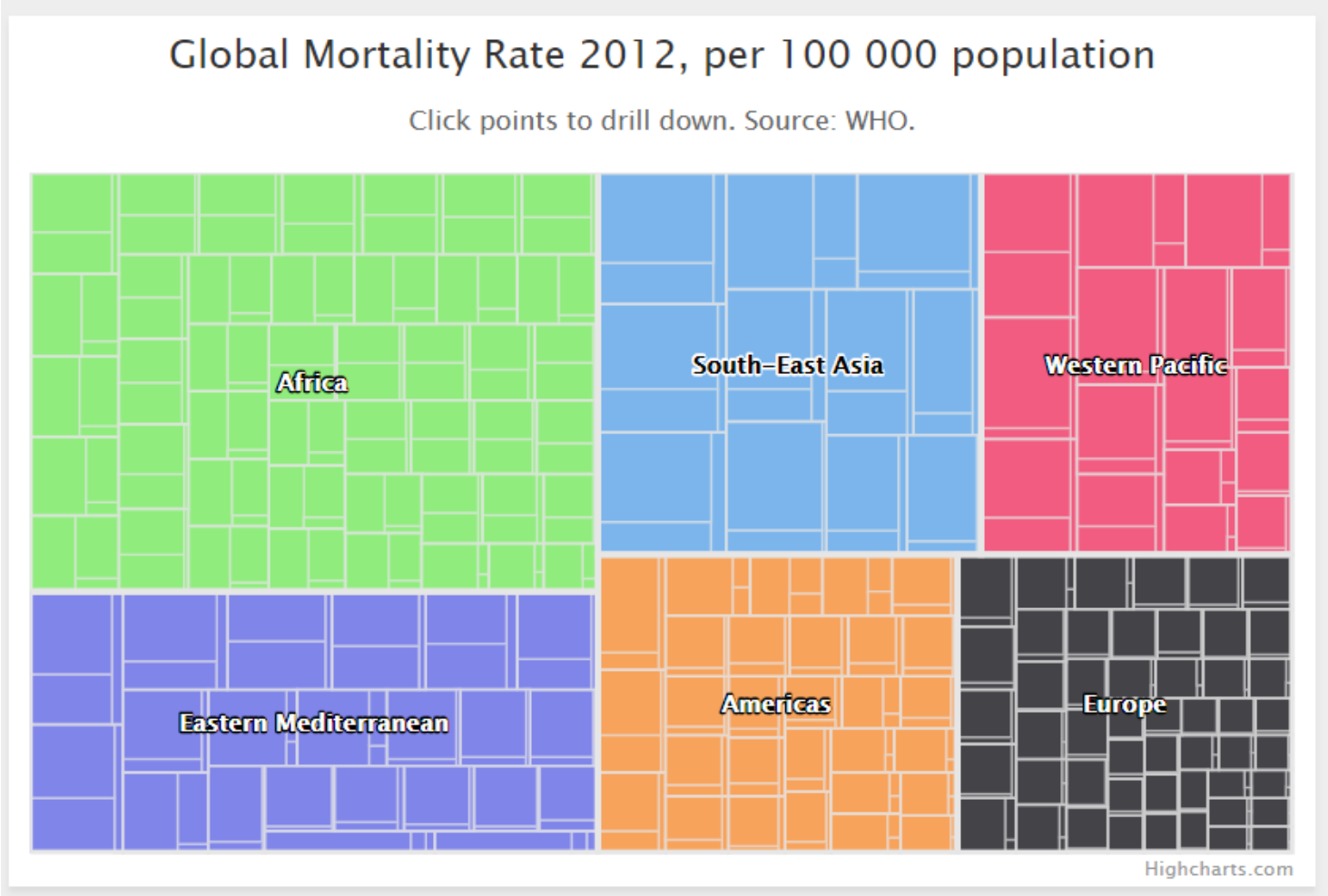
Treemaps

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



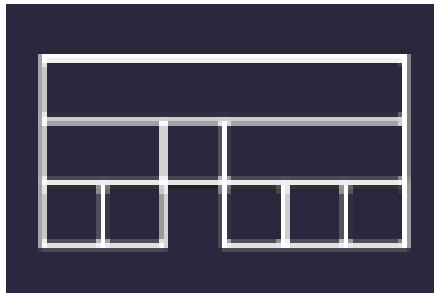


Hierachical Treemaps for Large Dataset



Layered Diagrams

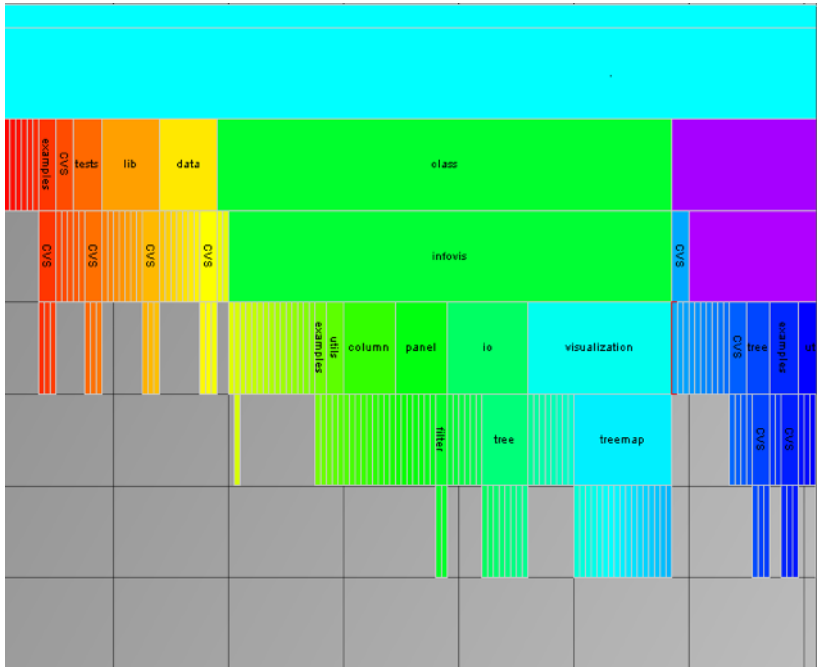
- Signify tree structure using
  - Layering
  - Adjacency

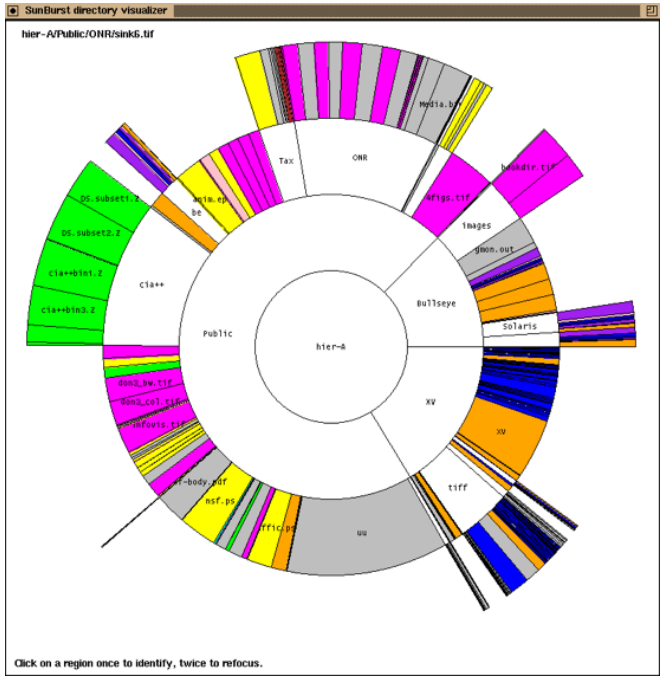


- Involves recursive sub-division of space.

Layered Diagrams

Icicle & Sunburst Trees

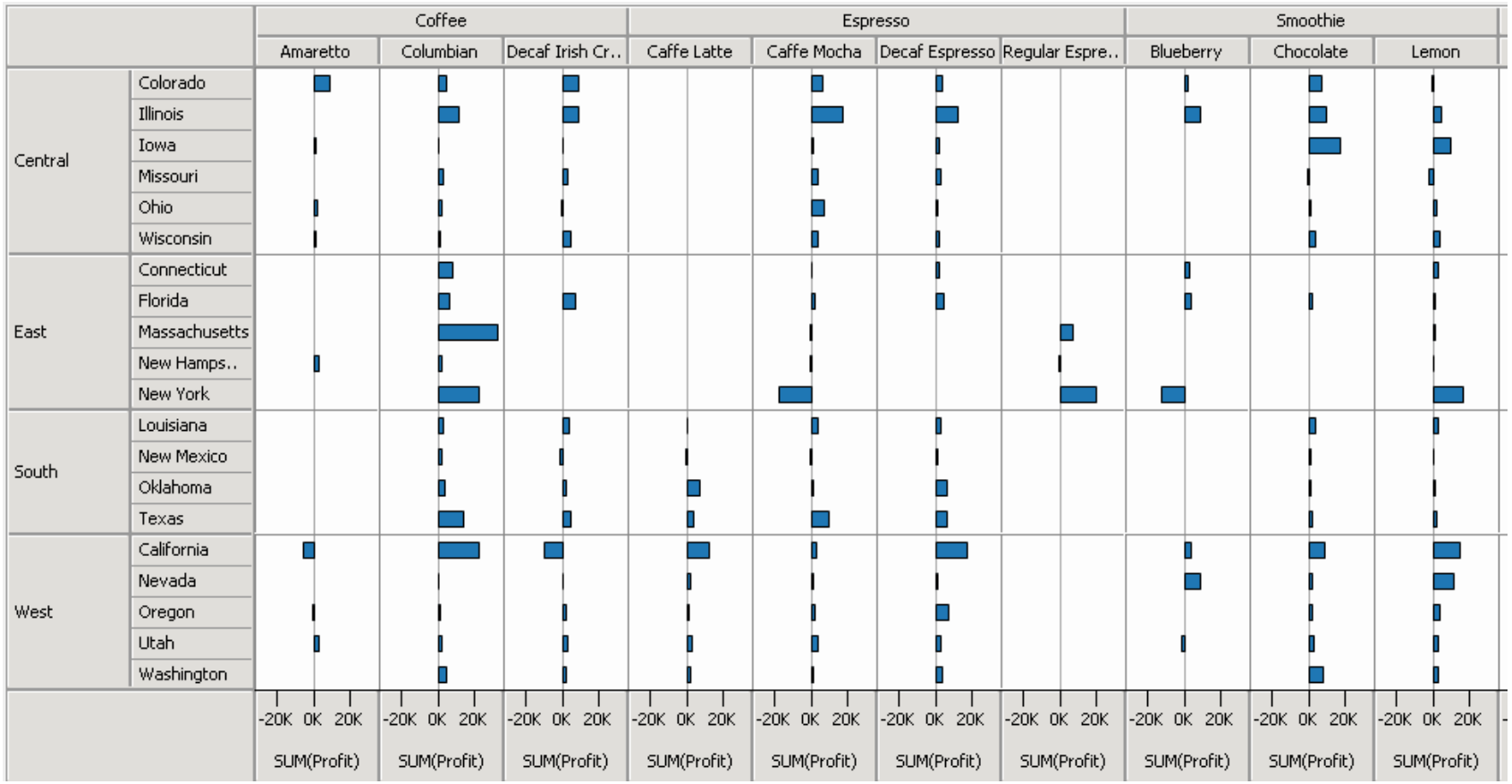




- Higher-level nodes get a larger layer area, whether that is horizontal or angular extent.
- Child levels are layered, constrained to parent's extent

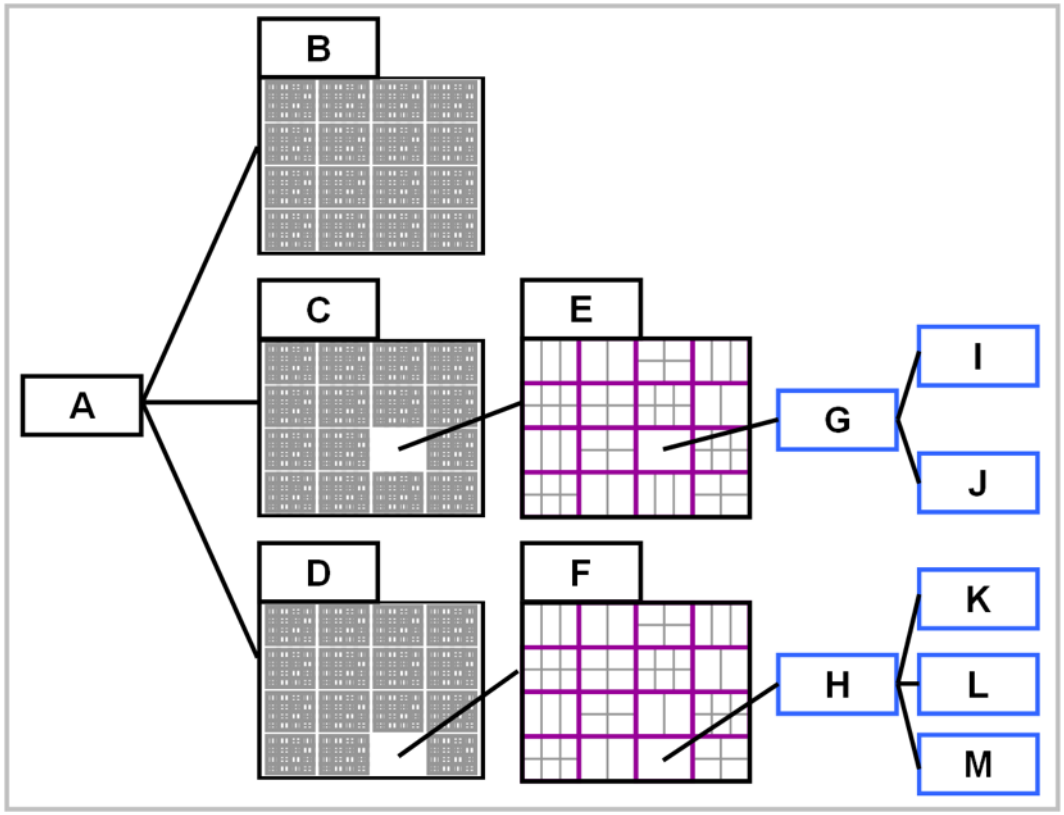
Layered Diagrams

Layered Tree Drawing



Hybrid Trees

"Elastic Hierarchies"



- Node-link diagram with treemap nodes.

Graph Layout

Approaches to Graph Drawing

- **Calculation using Graph Structure**
  - Spanning tree layout
  - Sugiyama-style (hierarchical) layout
  - Adjacency matrix layout
- **Optimization Methods**
  - Constraint satisfaction
  - Force-directed layout
- **Attribute-Driven Layout**
  - Layout using data attributes, not linkage

Calculation using Graph Structure

Calculation using Graph Structure

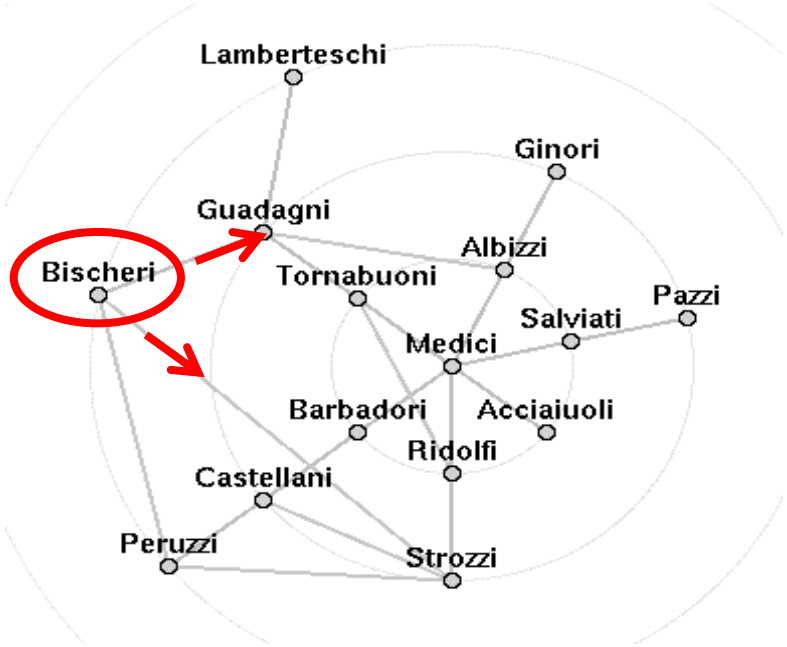
Tree-based graph layout

- Select a **tree-structure** out of the graph
  - Minimum spanning tree
  - Breadth-first-search tree (BFS)
  - Depth-first-search tree (DFS)
  - Other domain-specific structures
- Use a tree layout algorithm
- **Benefits**: Fast, supports interaction & refinement
- **Drawbacks**: Limited range of layouts

Calculation using Graph Structure

Spanning Tree Layout

- **Many graphs have useful spanning trees**
  - Websites,
  - Social Networks



Calculation using Graph Structure

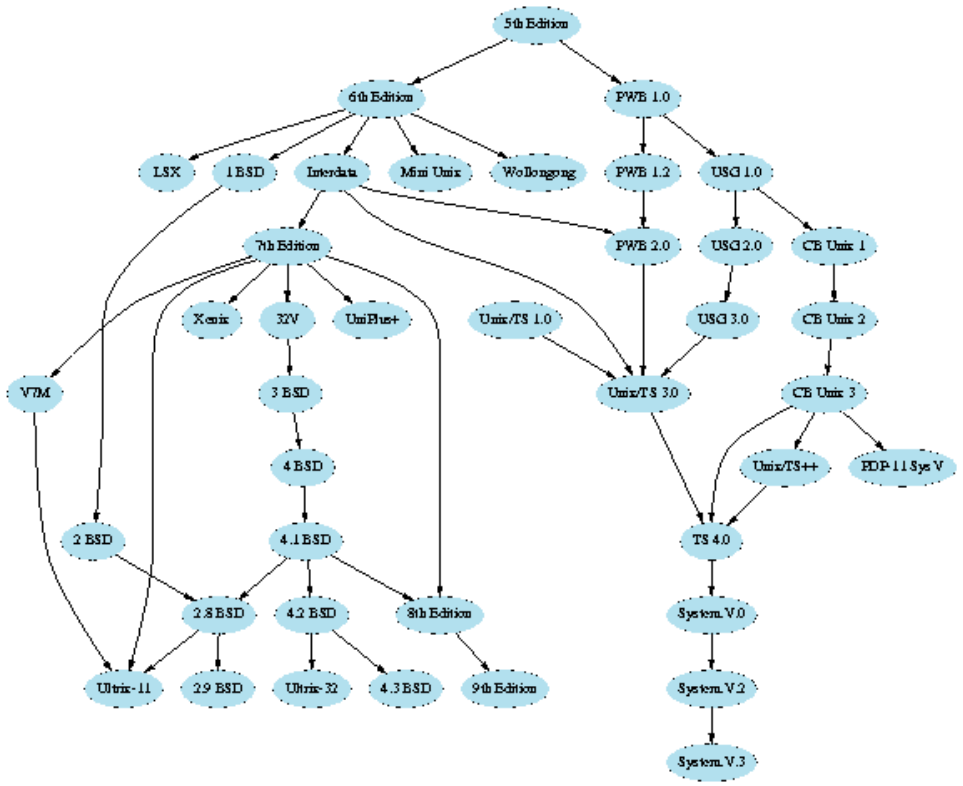
Hierarchical graph layout

- Use **directed-structure** of graph to form layout
- Order graph into distinct levels
  - this determines one dimension
- Now **optimize within levels**
  - determines the second dimension
  - minimize edge crossings, etc
- Great for directed acyclic graphs, but often misleading in case of cycles

Calculation using Graph Structure

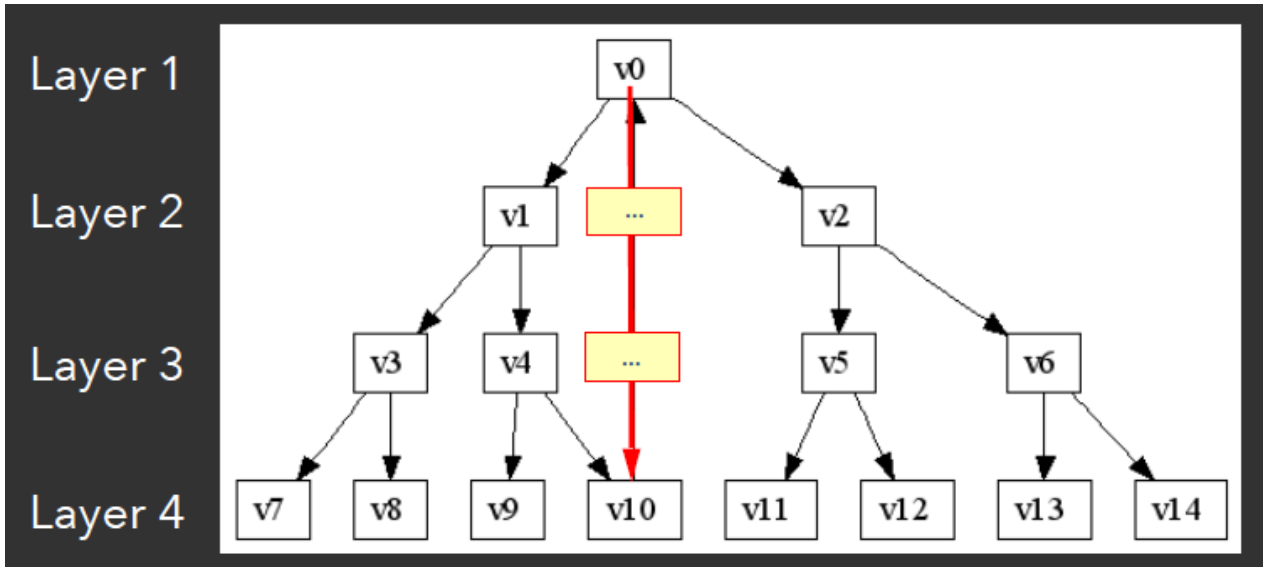
Sugiyama-Style Layout

- Evolution of the UNIX operating system
- Hierarchical layering based on descent



Calculation using Graph Structure

Sugiyama-Style Layout

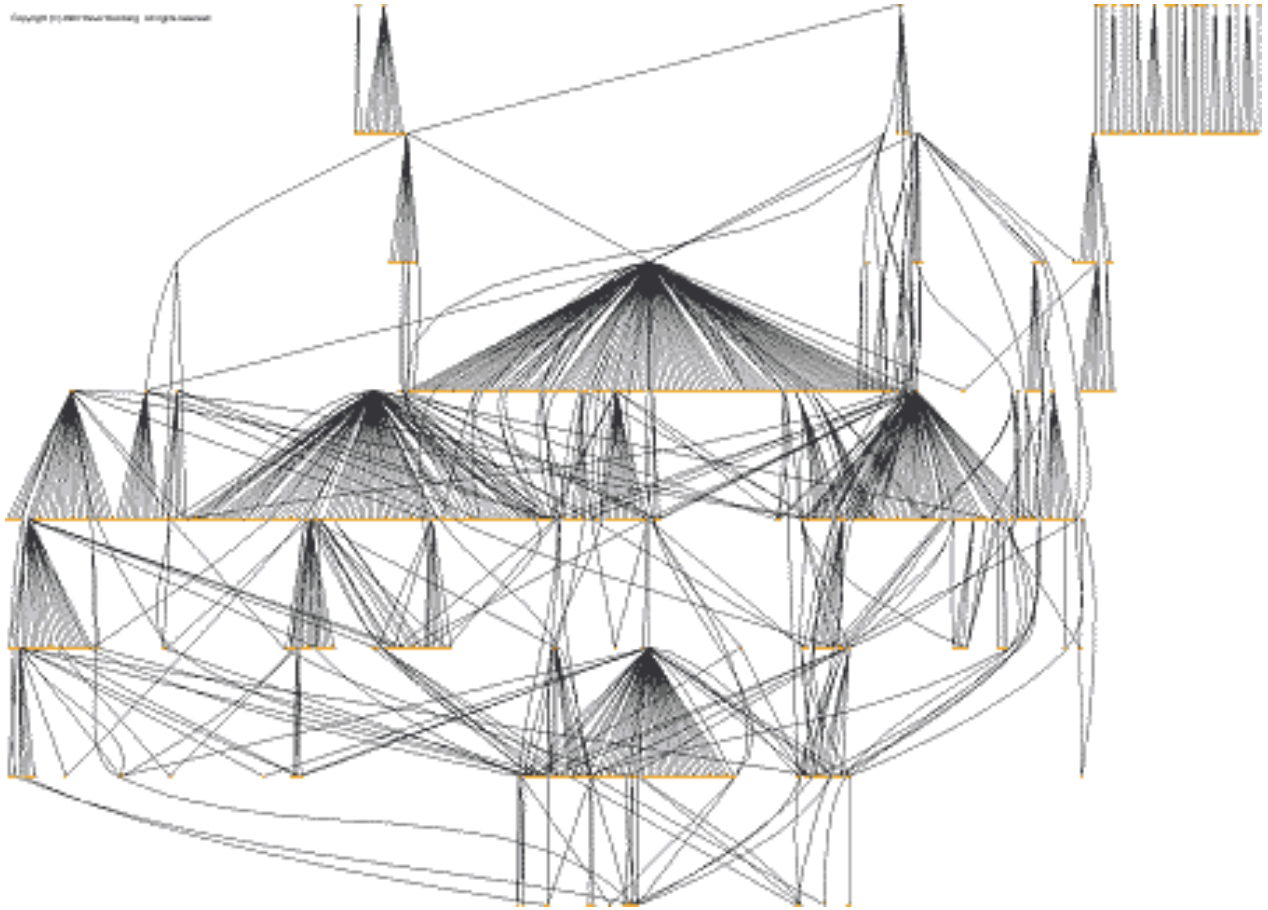


- Reverse edges to **remove cycles**
- Assign nodes to **hierarchy layers**
- Create dummy nodes to **"fill in" missing layers**
- Arrange nodes within layer, **minimize edge crossings**
- **Route edges** – layout splines if needed

Calculation using Graph Structure

Sugiyama-Style Layout

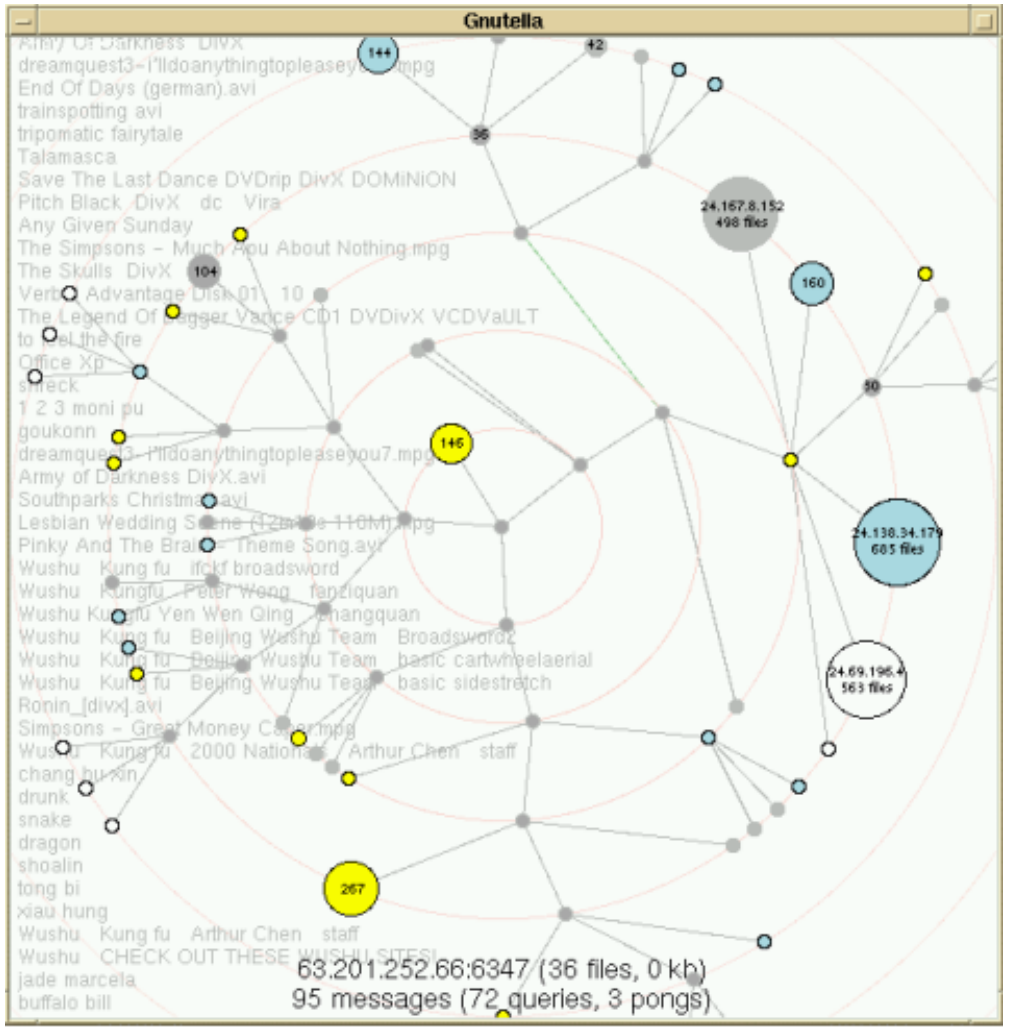
- Hierarchical Layout



Calculation using Graph Structure

Radial layout

- Animated exploration of graphs with radial layout



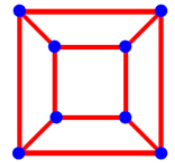
Optimization Techniques

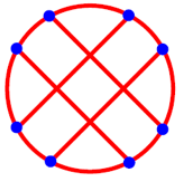
Optimization Techniques

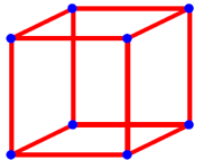
- **Treat layout as an *optimization problem***
  - Define layout using an *energy model along with constraints: equations the layout should obey.*
  - Use optimization algorithms to solve
- **Commonly posed as a physical system**
  - Charged particles, springs, drag force, …
- **We can introduce directional constraints**
  - *DiG-CoLa (Di-Graph Constr Optimization Layout) [Dwyer 05]*
  - Iterative constraint relaxation

Optimization Techniques

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

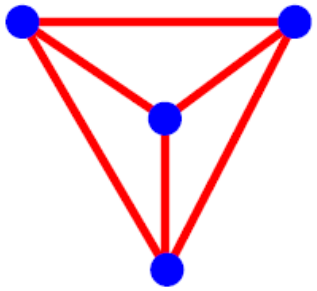




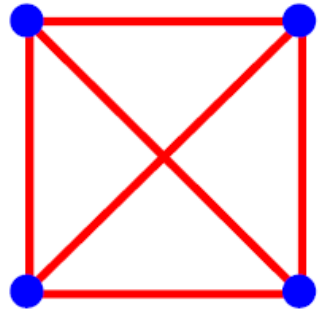


but, can't do it all!

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



min # crossings



max symmetries

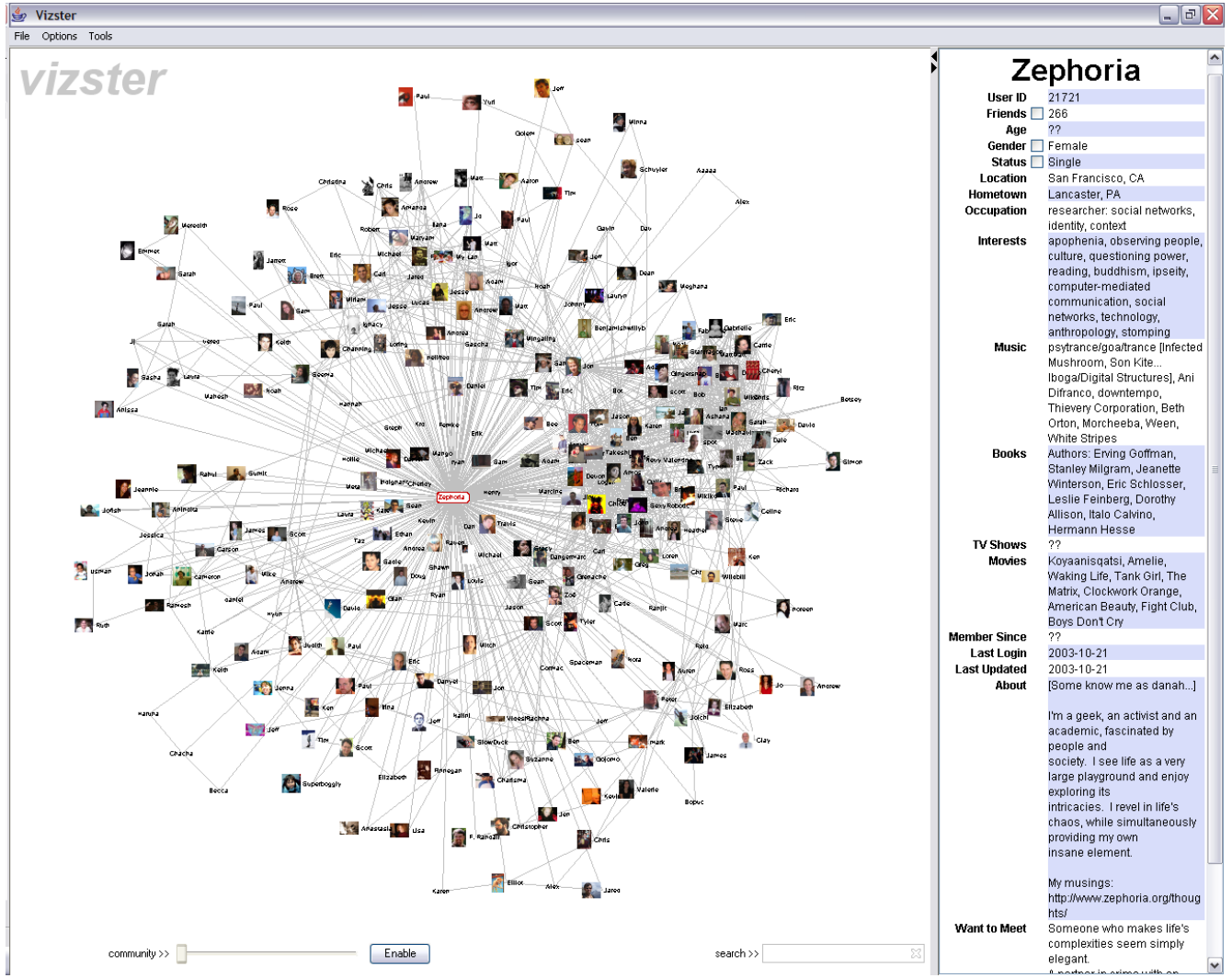
Optimization Techniques

Force-directed Layout

- Nodes = charged particles
  with air resistance
- Edges = springs

- **Iteratively calculate forces, update node positions**
- Naïve n-body calculation is $O(N^2)$
- O(N log N) using quadtree or k-d tree
- Numerical integration of forces at each time step

Optimization Techniques



Optimization Techniques

Constrained Optimization

- **Minimize stress function**

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

  where

  X: node positions, d: optimal edge length,

  w: normalization constants

- *Note: Try to place nodes d_{ij} apart*

Optimization Techniques

Constrained Optimization

- **Add hierarchy ordering constraints**

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

where

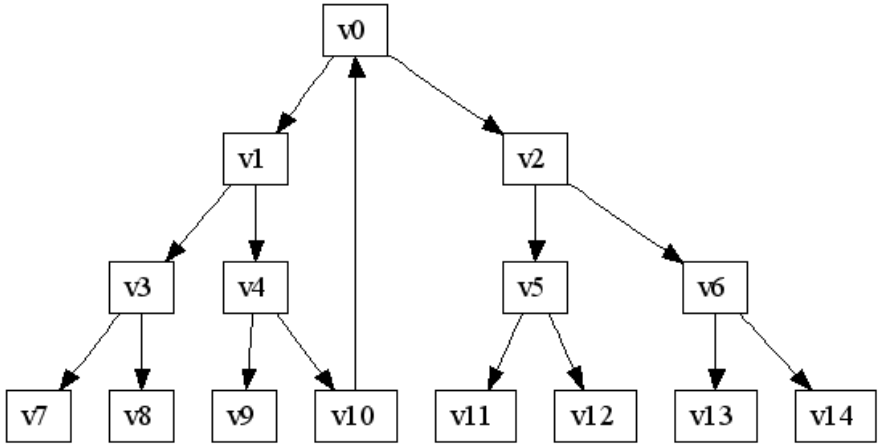
y: node y-coordinates

δ: edge direction (e.g., 1 for i->j, 0 for undirected)

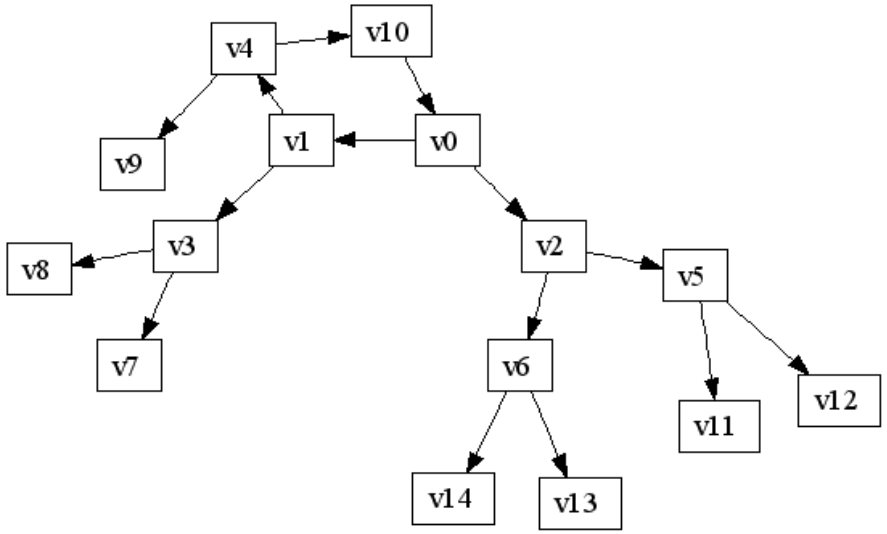
- *Note: If i points to j, it should have a lower y-value*

Optimization Techniques

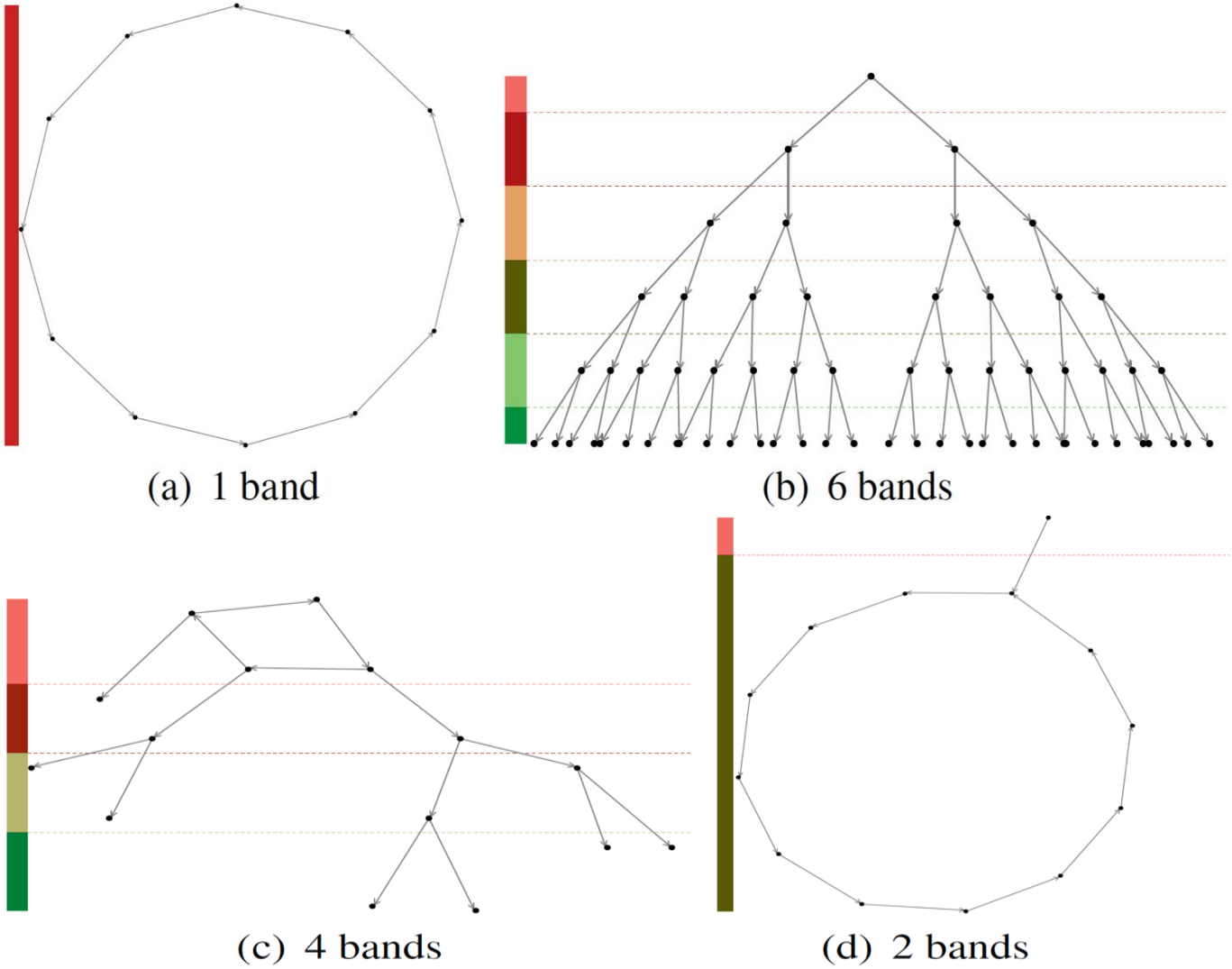
Sugiyama layout (dot)
Preserve **tree structure**



DiG-CoLa method
Preserve **edge lengths**



Optimization Techniques



Optimization Techniques

Iterative Constraint Relaxation

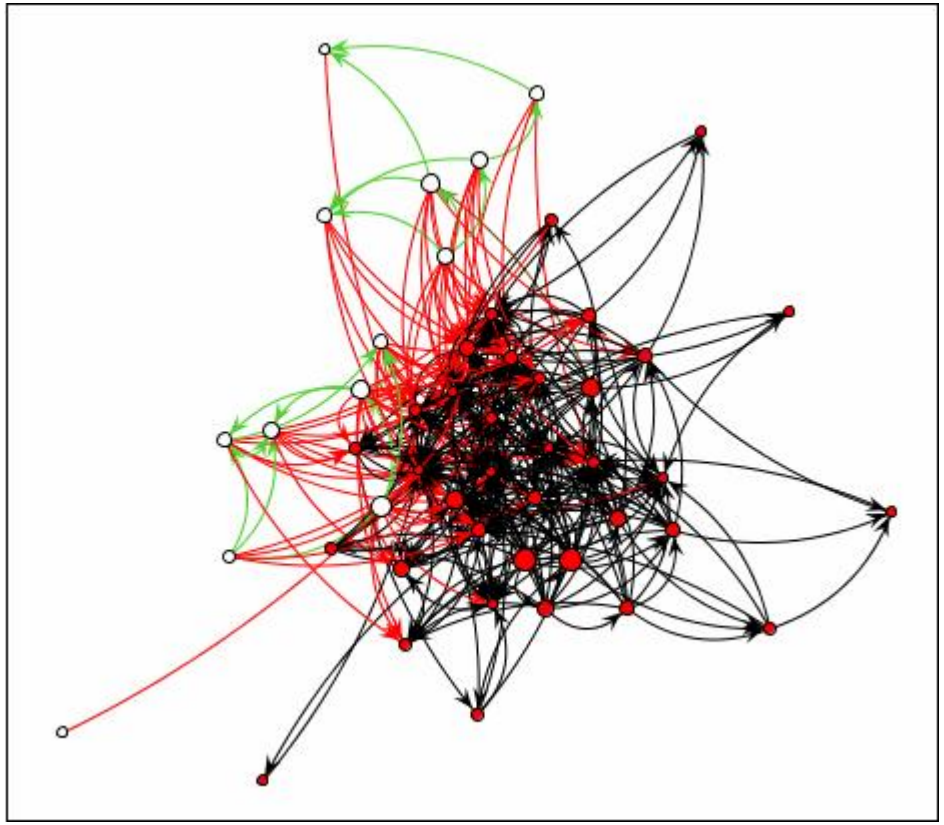
- Quadratic programming is complex to code and computationally costly. Is there a simpler way?
- Iteratively **relax each constraint** [Dwyer 09]
- Given a constraint (e.g., $| x_i - x_i | = 5$)
  - Simply push the nodes to satisfy!
  - Each relaxation may **clobber prior results**
- But this typically **converges quickly**
- Enables **expressive constraints!**

Use the Force!

http://mbostock.github.io/d3/talk/20110921/

Optimization Techniques

- Limitations of Node-Link Layout



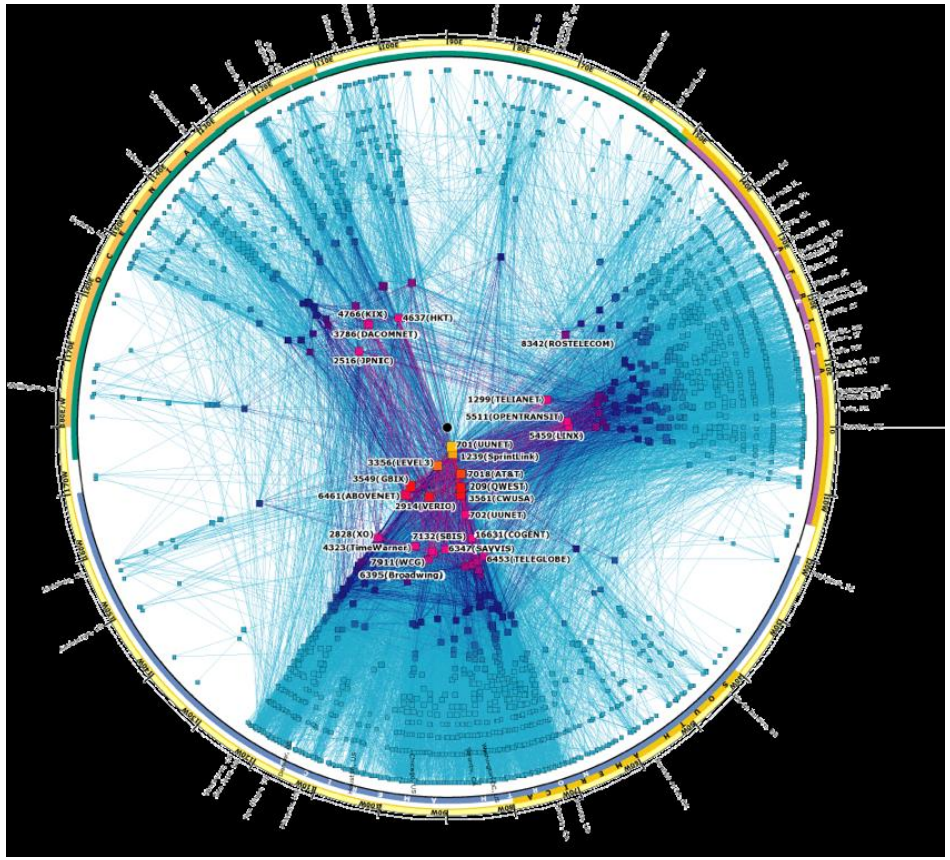
Attribute–driven Layout

Attribute-Driven Layout

- Large node-link diagrams **get messy**!
  - Is there additional structure we can exploit?
- **Idea**: Use **data attributes** to perform layout
  - For example, scatter plot based on node values
- *Dynamic queries / brushing to explore…*

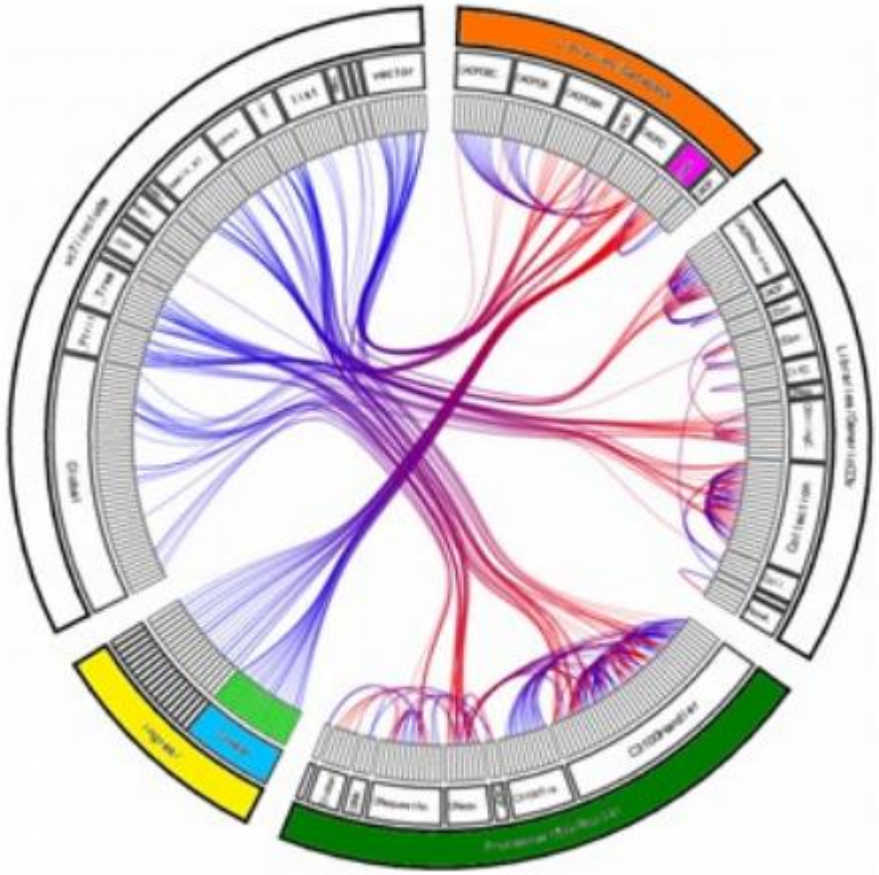
Attribute-Driven Layout

- The "**Skitter**" Layout
  - Internet Connectivity
  - Radial Scatterplot
- Angle = Longitude
  - Geography
- Radius = Degree
  - # of connections
    (a statistic of the nodes)

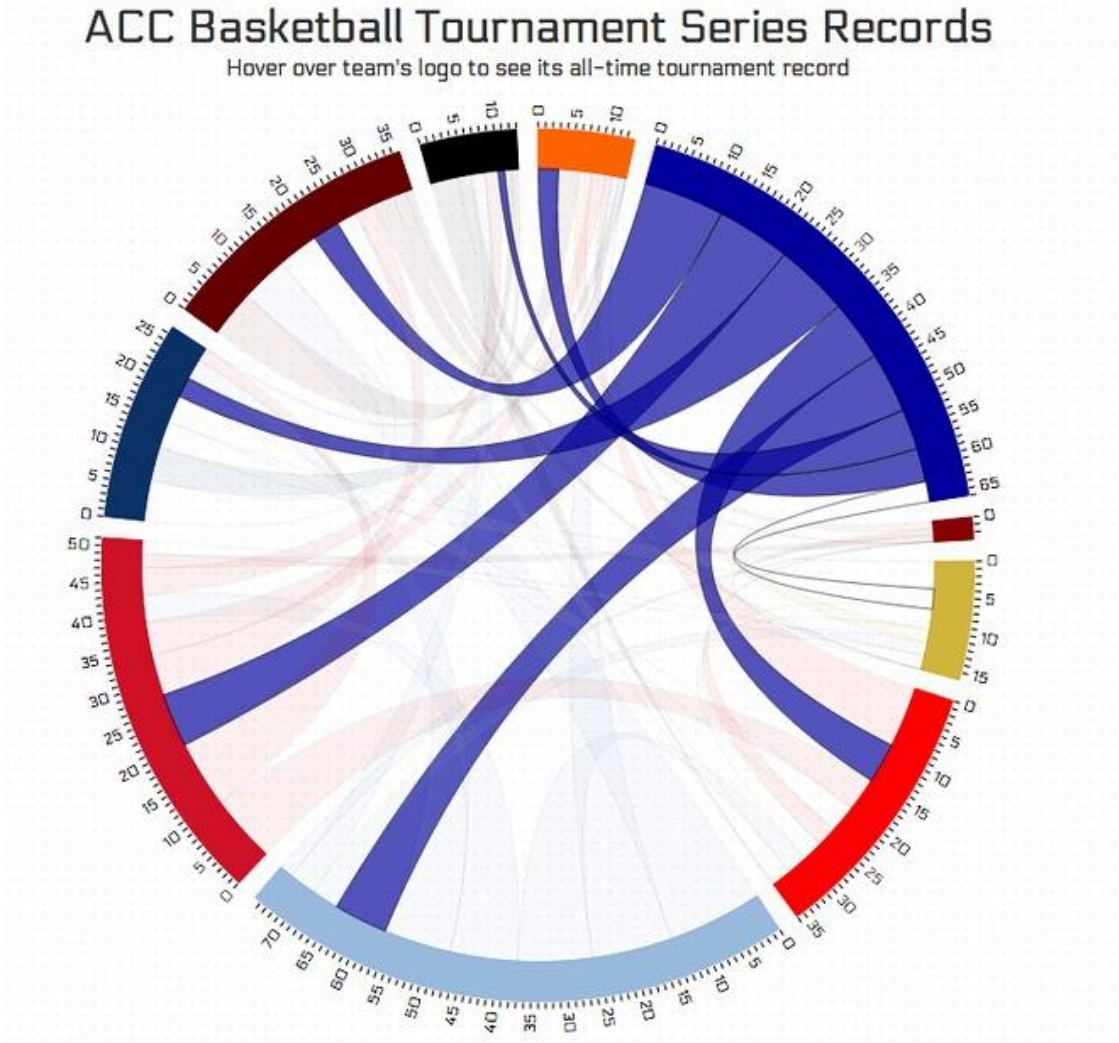


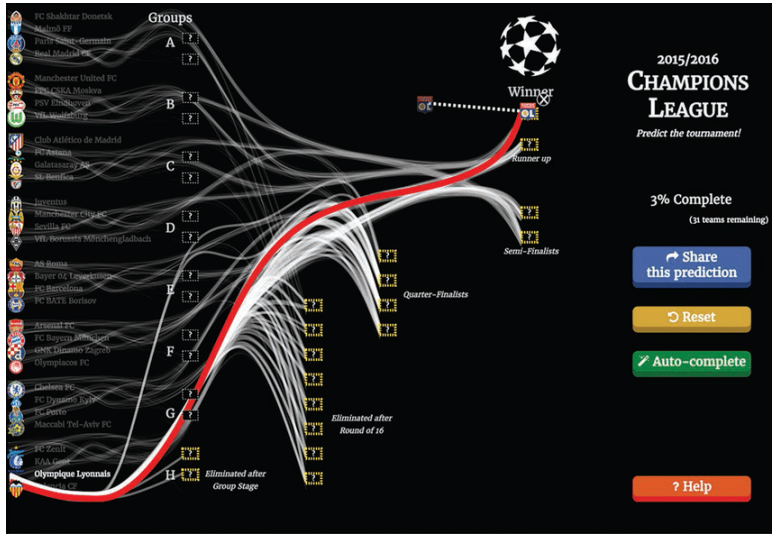
Attribute-Driven Layout

- The **Call Graph**
- Three concentric rings show containment
  - Files
  - Classes
  - Methods
- The curved lines indicate function calls



Example: Graph Visualizations in Sports





Summary

Visualizing Trees

- Indentation
- Node-Link diagrams
- Enclosure diagrams
- Layering

Visualizing Graphs

- Calculation using Graph Structure
  - Tree-based, hierarchical layouts
- Optimization Methods
  - Force-directed layout
- Attribute-Driven Layout
  - Skitter layout