CSE 5544: Introduction to Data Visualization

Raghu Machiraju
machiraju.1@osu.edu
Many Thanks to

– Alex Lex, University of Utah
– Torsten Moeller, University of Vienna
– Tamara Munzner, Univ of British Columbia
– Hanspeter Pfister, Harvard University
Graphs

Without graphs, there would be none of these:
To the Editor:

Genome-wide association studies (GWAS) of complex human traits have become an important approach in human genetics. Taken together, GWAS are arguably the largest biological investigations of humans ever conducted. The total number of people genotyped to date with a GWAS array is difficult to know but probably exceeds 1,000,000. Major findings from these studies are that many common diseases have a polygenic architecture, the genetic effect sizes of common SNP variants are small, the identification of the involvement of genes and biological processes not previously suspected, and the association of some loci with different diseases.\(^1\)\(^2\) Critically, the sample sizes necessary to identify robust and replicable findings are beyond those achievable by single groups, and collaborations have rapidly evolved to augment statistical power.
Simpler Examples

Which wine am I?  

Mark Lombardi

Orgcharts

Finite State Machines
Main point of Graph Viz
Eine Grosse Problem
E. Coli metabolism

KEGG: Kyoto Encyclopedia of Genes and Genomes
(http://www.genome.ad.jp/kegg/kegg.html)
Another Example
Another Problem
Supporting Multiple Tasks

Two central tasks:

Explore **topology** of pathway

Explore the **attributes** of the nodes (experimental data)

Need to support both!
Yet Another Problem
Graph Tools & Applications
Gephi
http://gephi.org

The Open Graph Viz Platform

Gephi is a visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs.

Runs on Windows, Linux and Mac OS X. Gephi is open-source and free.

Learn More on Gephi Platform »

Download FREE Gephi 0.7 alpha
Release Notes | System Requirements

Features Quick start Screenshots Videos

Gephi has been accepted again for Google Summer of Code! The program is the best way for students around the world to start contributing to an open-source project. Students, apply now for Gephi proposals. Come to the GSOC forum section and say Hi! to this topic.

Learn More »
Cytoscape

Open source platform for complex network analysis

http://www.cytoscape.org/
Cytoscape Web

http://cytoscapeweb.cytoscape.org/
NetworkX

https://networkx.github.io/

High-productivity software for complex networks

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

Features

- Python language data structures for graphs, digraphs, and multigraphs.
- Nodes can be “anything” (e.g., text, images, XML records)
- Edges can hold arbitrary data (e.g., weights, time-series)
- Generators for classic graphs, random graphs, and synthetic networks
- Standard graph algorithms
- Network structure and analysis measures
- Open source BSD license
- Well tested: more than 1800 unit tests, >90% code coverage
- Additional benefits from Python: fast prototyping, easy to teach, multi-platform

Documentation

all documentation

Examples

using the library

Reference

all functions and methods

Versions

Latest Release

1.8.1 - 4 August 2013

downloads | docs | pdf

Development

1.9.dev

github | docs | pdf

Build successful

Coverage: 95%

Contact

Mailing list
Issue list
Developer guide
Graph Theory Fundamentals

- Tree
- Network
- Hypergraph
- Bipartite Graph
Königsberg Bridge Problem (1736)
Graph

A graph $G(V,E)$ consists of a set of vertices $V$ (also called nodes) and a set of edges $E$ connecting these vertices.
Graphs

An edge $e_{xy} = (x,y)$ connects two vertices $x$ and $y$.
For example $V=\{1,2,3,4\}$, $E=\{(1,2),(1,3),(2,3),(3,4),(4,1)\}$
Graph

A simple graph $G(V,E)$ is a graph which contains **no multi-edges and no loops**
Graph

A directed graph (digraph) is a graph that discerns between the edges $\text{A} \rightarrow \text{B}$ and $\text{A} \leftarrow \text{B}$.

A hypergraph is a graph with edges connecting any number of vertices.
Graph

Independent Set
G contains no edges

Clique
G contains all possible edges
Graph

Path
G contains only edges that can be consecutively traversed

Tree
G contains no cycles

Network
G contains cycles
Graph

*Unconnected graph*
An edge traversal starting from a given vertex cannot reach any other vertex.

*Articulation point*
Vertices, which if deleted from the graph, would break up the graph in multiple sub-graphs.
**Graph**

*Biconnected graph*
A graph without articulation points.

*Bipartite graph*
The vertices can be partitioned in two independent sets.
Tree

A graph with no cycles - or:

A collection of nodes
contains a root node and 0-n subtrees
subtrees are connected to root by an edge
Ordered Tree
Binary Tree

Contains no nodes, or

Is comprised of three disjoint sets of nodes:
  a root node,
  a binary tree called its left subtree, and
  a binary tree called its right subtree
Different Kinds of Graphs

Over 1000 different graph classes

Tree

Bipartite Graph

Network

Hypergraph

A. Brandstädt et al. 1999
A directed graph

An undirected graph

Weighted

Unconnected

Node degrees

A cycle

An acyclic graph

A connected acyclic graph, a.k.a. a tree

A rooted tree or hierarchy

Node depths
Graph Measures

Node degree $\text{deg}(x)$
The number of edges being incident to this node. For directed graphs $\text{indeg}/\text{outdeg}$ are considered separately.

Diameter of graph $G$
The longest shortest path within $G$.

Pagerank
count number & quality of links
Graph Measures

• Topology
  • Number of hops - discrete distance
  • Finding all possible paths
  • Shortest paths between 2 nodes
  • Finding all nodes 1 hop away from target node
  • Finding nodes that are bridges between components
Graph Algorithms (I)

Traversal: Breadth First Search, Depth First Search

- generates neighborhoods
- hierarchy gets rather wide than deep
- solves single-source shortest paths (SSSP)

- classical way-finding/back-tracking strategy
- tree serialization
- topological ordering
Hard Graph Algorithms
(NP-Complete)

Longest path
Largest clique
Maximum independent set (set of vertices in a graph, no two of which are adjacent)
Maximum cut (separation of vertices in two sets that cuts most edges)
Hamiltonian path/cycle (path that visits all vertexes once)
Coloring / chromatic number (colors for vertices where no adjacent v. have same color)
Minimum degree spanning tree
Graph and Tree Visualization
Setting the Stage

How to decide which *representation* to use for which *type of graph* in order to achieve which kind of *goal*?
Different Kinds of Tasks/Goals

Two principal types of tasks: attribute-based (ABT) and topology-based (TBT)

Localize – find a single or multiple nodes/edges that fulfill a given property
  • ABT: Find the edge(s) with the maximum edge weight.
  • TBT: Find all adjacent nodes of a given node.

Quantify – count or estimate a numerical property of the graph
  • ABT: Give the number of all nodes.
  • TBT: Give the indegree (the number of incoming edges) of a node.

Sort/Order – enumerate the nodes/edges according to a given criterion
  • ABT: Sort all edges according to their weight.
  • TBT: Traverse the graph starting from a given node.

list adapted from Schulz 2010
Three Types of Graph Representations

Explicit (Node-Link)

Matrix

Implicit
Example - Connection Marks
Small Trees

Vertical spatial position - depth of tree

Radial length - depth of tree

Figure 7.16: Node-link layouts of small trees. a) Triangular vertical for tiny tree. From [Buchheim et al. 02], Fig. 2d. b) Spline radial layout for small tree. Courtesy of Michael Bostock, made with D3 [Bostock et al. 11], from http://mbostock.github.com/d3/ex/tree.html. (Permission needed.)
Not-So-Small Trees

Color - Strahler centrality metric

Figure 7.17: Three layouts of a larger tree of 3238 nodes. a) Rectangular horizontal node-link layout for large tree. b) BubbleTree node-link layout for same tree. Treemaps layout, showing hierarchical structure with containment rather than connection. Courtesy of David Auber, made with Tulip [Auber et al. 12], from http://tulip.labri.fr/Documentation/3_7/userHandbook/html/ch06.html. (Permission needed.) (can request higher res figs from David if needed. or just make my own with Tulip, with different colors (white bg, diff colormap).)
Explicit Graph Representations

Node-link diagrams: vertex = point, edge = line/arc
Node-link Layouts

Size and color coding for nodes and edges is also common.

Use size coding of edge attributes with different line widths.
Criteria for Good Node-Link Layout

- Minimized edge crossings
- Minimized distance of neighboring nodes
- Minimized drawing area
- Uniform edge length
- Minimized edge bends
- Maximized angular distance between different edges
- Aspect ratio about 1 (not too long and not too wide)
- Symmetry: similar graph structures should look similar

List adapted from Battista et al. 1999
Conflicting Criteria

Minimum number of edge crossings vs. Uniform edge length

Space utilization vs. Symmetry
Explicit Representations

**Pros:**
- is able to depict all graph classes
- can be customized by weighing the layout constraints
- very well suited for TBTs, if also a suitable layout is chosen
[McGrath et al. 1997], [Purchase et al. 2002], and [Huang et al. 2005]

**Cons:**
- computation of an optimal graph layout is in NP
  (even just achieving minimal edge crossings is already in NP)
- even heuristics are still slow/complex (e.g., naïve spring embedder is in $O(n^2)$)
- has a tendency to clutter (edge clutter, “hairball”)
Layouts
Layouts

Figure 7.24: Eight visual encodings of the same tree dataset, using different combinations of visual channels.  

- a) Rectilinear vertical node-link, using connection, with vertical spatial position showing tree depth.  
- b) Rectilinear horizontal layered node-link, using connection, with horizontal spatial position showing tree depth.  
- c) Icicle, with vertical spatial position and size showing tree depth, and horizontal spatial position showing link relationships.  
- d) Radial node-link, using connection, with radial spatial position showing tree depth.  
- e) Concentric circles, with spatial position and size showing tree depth and radial spatial position showing link relationships.  
- f) Nested circles, using radial containment, with nesting level and size showing tree depth.  
- g) Treemap, using rectilinear containment, with nesting level and size showing tree depth.  
- h) Indented outline, using spatial position channels, with horizontal spatial position showing tree depth and (hmm).

From [McGuffin and Robert 10], Figure 1. (Permission


• Maximize space for color coding

• High Information density

• Cannot use white space

---

Layouts

Space Filling

- Containment marks
- Area Marks and spatial position channels
Explicit Representations

Problem #1: computing an optimal layout lies in NP
Solution approach: formulate the layout problem as an optimization problem

1. Conversion of the layout criteria into a weighted cost function:

   \[ F(\text{layout}) = a \cdot |\text{edge crossings}| + \ldots + f \cdot |\text{used drawing space}| \]

2. Usage of a standard optimization technique to find a layout that minimizes the cost function
Explicit Representations

Problem #1: computing an optimal layout lies in NP
Solution approach: formulate the layout problem as an optimization problem

Common variation: conversion using a physical analogy, e.g. springs
Force Directed Layouts
Force Directed Layouts

Physics model:
edges = springs,
vertices = repulsive magnets
in practice: damping

Computationally expensive: $O(n^2)$
Limit (interactive): $\sim 1000$ nodes
**Force Directed Layouts**

<table>
<thead>
<tr>
<th>Technique</th>
<th>force-directed placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Types</td>
<td>network</td>
</tr>
<tr>
<td>View Comp.</td>
<td>connection</td>
</tr>
<tr>
<td>Abstract Tasks</td>
<td>understanding topological structure following paths</td>
</tr>
<tr>
<td>Scalability</td>
<td>nodes: dozens/hundreds</td>
</tr>
<tr>
<td></td>
<td>edges: hundreds</td>
</tr>
<tr>
<td></td>
<td>node/edge density: $E &lt; 4N$</td>
</tr>
</tbody>
</table>

- Proximity does not mean actual grouping
- Avoid edge crossings and node overlaps
- Spatial position does not directly encode attributes
- Non-deterministic
Force Directed Layouts

4. Drawing on Physical Analogies

Ulrik Brandes

Force Directed Layouts

- No intrinsic layering, now what?
- Physics model, edges = springs, nodes = repulsive magnets
Force Directed Layouts

Fig. 4.1. The spring analogy.
Force Directed Layouts

Given a connected undirected graph $G = (V, E)$, let $p = (p_v)_{v \in V}$ be a vector of vertex positions $p_v = (x_v, y_v)$ in the plane. We denote by $\|p_v - p_u\|$ the length of the difference vector $p_v - p_u$, which is the Euclidean distance between positions $p_u$ and $p_v$. Furthermore, we denote by $\overrightarrow{p_u p_v}$ the unit length vector $\frac{p_v - p_u}{\|p_v - p_u\|}$ pointing from $p_u$ to $p_v$. The model of Eades (1984), now known as the

$$f_{\text{rep}}(p_u, p_v) = \frac{c_e}{\|p_v - p_u\|^2} \cdot \overrightarrow{p_u p_v}$$

$$f_{\text{spring}}(p_u, p_v) = c_\sigma \cdot \log \frac{\|p_u - p_v\|}{l}$$

**Input:** connected undirected graph $G = (V, E)$
initial placement $p = (p_v)_{v \in V}$

**Output:** placement $p$ with low internal stress

for $t \leftarrow 1$ to ITERATIONS do
  for $v \in V$ do
    $F_v(t) \leftarrow \sum_{u : (v, u) \notin E} f_{\text{rep}}(p_v, p_u) + \sum_{u : (u, v) \in E} f_{\text{spring}}(p_u, p_v)$
  for $v \in V$ do $p_v \leftarrow p_v + \delta \cdot F_v(t)$

![Fig. 4.2. Magnitude of spring embedder forces.](image-url)
Force Directed Layouts

• Start from random layout
• Loop:
  – For every node pair compute repulsive force
  – For every edge compute attractive force
  – Accumulate forces per node
  – Update node position in direction of accumulated force
• Stop when layout is ‘good enough’
Force Directed Layouts

Figure 20.3 A force-directed 2D layout of protein-protein interactions in yeast (redrawn from [FS03]).
Force Directed Layouts
Nice looking layouts

Highschool dating network
Force directed layouts

+ Very flexible, pleasing layouts on many types of graphs
+ Can add custom forces
+ Relatively easy to implement

- Repulsion loop is $O(n^2)$ per iteration
- Prone to local minima
Address Computational Scalability: Multilevel Approaches

[Schulz 2004]
Abstraction/Aggregation

30k nodes

18 nodes

750 nodes

90 nodes

cytoscape.org
Collapsible Force Layout

Supernodes: aggregate of nodes
manual or algorithmic clustering
Issue

Problem #1: computing an optimal layout lies in NP
Solution approach: formulate the layout problem as an optimization problem

BUT: naïve runtime complexity is still $O(n^2)$!

→ in each optimization step, all vertices have to be checked against all other vertices to determine whether to pull them together (spring) or push them apart (expander)
Scalable Force Directed Layout

- Barnes-Hut multibody algorithm

- Compute quadtree for current layout
- For each non empty cell in quadtree, store total nodes in cell and center of mass (COM) of all leafnodes
Estimating Forces

- To compute the total force on a node n, look at the distance between n and the COM of the top cell
  - If bigger than a threshold we can estimate using the cell’s COM as a repulsor
  - If smaller we ‘open up’ the cell and sum the estimates for its subcells
- Complexity per iteration: $O(N \log N)$ instead of $O(N^2)$

```c
function double estimateForce(n: Node, cell: Cell) {
    float distance = d(n, cell.CenterOfMass)
    if (distance/cell.dimensions < threshold)
        return $\sum$ estimateForce(n, cell.children)
    else
        return $C_R \cdot 1 \cdot cell.leafcount / distance^2$
}
```
Local minima

• Use better initialization than random
• Use better optimization schemes
  – Simulated Annealing (chance of random changes in node position)
  – Stress majorization
  – Other heuristics (barrier jumping, impulse)
Local Minimas -

Figure 7.20: Multi-level graph drawing with sfdp [Hu 05]. a) Cluster structure is visible for a large network of 7220 nodes and 13,800 edges. b) A huge graph of 26,028 nodes and 100,290 edges is a “hairball” without much visible structure. From Yifan Hu’s Gallery of Large Graphs, JDG_Homologycis-n4c6, b14 and b4.
Zoom und Pan

Pan and Zoom with After Effects
Geometric vs Semantic

1. Start with the concept map, and consider this the base zoom level—"zoom level 0."
2. Back away from the map, literally and figuratively, to consider the broader context in which it exists. What are the large social and political constructs that contain the material? Name these constructs, and write these names close to the existing words.
3. Repeat Step #2, continuing to back away—zoom out—and consider broad context. At each zoom, create a new visual representation of the system.
4. Gradually zoom back in, while focusing on a specific aspect of the original concept map. As you zoom in, map boundaries of the map change, forcing you to add details, entities, and actions.
5. After performing several zooms in both directions, consider the results. Which areas were easy to describe, and which were difficult? Did you find yourself making assumptions or guesses about content? The answers indicate the limitations of your knowledge and areas for additional research.

https://www.wickedproblems.com/5_semantic_zoom.php
Fisheye
FishEye
The Fisheye View

- Why fisheye view?
- Problems of visualizing large structures
- Too much to show
  - Easy to get lost
  - Human behavior of perceiving large structures
  - Heavily influenced by the spatial distance to locations
  - Leaving out detail systematically
PlImage imgOrg;

// Lens properties
int lsize = 90, lsize2 = lsize * lsize;
float mag = 2.0f;
float k = -0.00016f;

int offX, offY;
int border, borderViaLens;

public void setup() {
    size(580, 404);
cursor(CROSS);
    imgOrg = loadImage("london.jpg");
    // Calculate image offsets reqd to centre image
    offX = (width - imgOrg.width)/2;
    offY = (height - imgOrg.height)/2;
    // border colours
    border = color(200);
    borderViaLens = color(180);
}

public void draw() {
    background(200);
    image(imgOrg, offX, offY);
    showLens(mouseX, mouseY);
}

public void showLens(int x, int y) {
    int u, v, r2;
    float f;
    for (int vd = -lsize; vd < lsize; vd++) {
        for (int ud = -lsize; ud < lsize; ud++) {
            r2 = ud*ud + vd*vd;
            if (r2 <= lsize2) {
                f = mag + k * r2;
                u = floor(ud/f) + x - offX;
                v = floor(vd/f) + y - offY;
                if (u >= 0 && u < imgOrg.width && v >=0 && v < imgOrg.height) {
                    set(u + x, v + y, imgOrg.get(u, v));
                } else {
                    set(u + x, v + y, borderViaLens);
                }
            }
        }
    }
}
Transformation & Magnification
The Fisheye View

• What is fisheye view?
  • A very wide angle view
  
  • Showing things nearby in great detail while still showing all data
  
  • Providing balance of local detail and global context

• Goal:
  • Smooth integration of local detail and global context

• Technique:
  • Reposition and resize objects
    • A variation on Focus + Context

  • Make objects of interest large, detailed

  • Less important objects successively smaller
Example

\[ \text{DOI}(a, b) = \text{API}(a) - D(a,b) \]
The Fisheye View

Fisheye construction

Three pieces

- **Focal point**
- level of detail
- distance from focus

\[ DOI(x|.) = (LOD(x) - D(.,x)) \]

Priori component  posteriori component
The Fisheye View

• DOI comprised of two parts
  • A priori component: (A priori importance API)
    • Measure of a priori interest independent of current focus
    • Reflecting global importance of structure points
  • A posteriori component:
    • Determining parts of most interest that depends on current focus
    • Measuring relative distance in the current focus
The Fisheye View

- Classified by the magnification functions:
  - Piecewise continuous
    - Constant function (bifocal display)
    - Varying function (perspective wall)
  - Continuous

- Non-continuous Magnification Functions
- Continuous Magnification Functions
- Polyfocal Display
- Bifocal Display
- Perspective Wall
Magnification Function:
A piecewise Fisheye View

Magnification Factor
Fisheye to Graphs
Paris Metro, importance corresponds to number of connections
Fisheye Views of Graphs

Convert **normal** to **fisheye** coordinates

**MAGNIFY** vertices of greater interest

demagnify vertices of lesser interest

Also, recompute positions of all vertices

Analogy: blowing up a balloon

As some points grow farther apart, some get pushed closer together

This happens because there is a fixed area or volume for all the points
Normal to Fisheye Coordinates

Determine the current focus.

Then, for each vertex in the graph:

- Compute new position
- Compute new size
- Determine amount of detail to be shown

Compute visual worth:

- Importance (API) +
- Distance from the focus point

=> A very important node is small if very far from focus point
New Vertex Position

\[ P_{\text{fisheye}}(v, f) = F_1(P_{\text{normal}}(v), P_{\text{normal}}(f)) \]

\[ = \left( G\left( \frac{D_{\text{normal}}}{D_{\text{max}}}, D_{\text{max}} + P_{\text{focus}} \right), G\left( \frac{D_{\text{normal}}}{D_{\text{max}}}, D_{\text{max}} + P_{\text{focus}} \right) \right) \]

\[ G(x) = \frac{(d + 1)x}{dx + 1} \]

\[ D_{\text{max}} = \text{horizontal distance between the screen boundary and the focus} \]

\[ D_{\text{normal}} = \text{horizontal distance between the vertex being transformed and the focus} \]
Various Distortion Factors

\[ x \quad d=0.5 \quad d=2 \quad d=5 \]

\[
\begin{array}{ccc}
.1 & .14 & .25 & .40 \\
.2 & .27 & .42 & .60 \\
.3 & .39 & .56 & .72 \\
.4 & .50 & .66 & .80 \\
.5 & .60 & .75 & .85 \\
.6 & .69 & .82 & .90 \\
.7 & .77 & .88 & .93 \\
.8 & .85 & .92 & .96 \\
.9 & .93 & .96 & .98 \\
\end{array}
\]
New Vertex Size

\[ S_{\text{fisheye}}(v, f) = F_2(S_{\text{normal}}(v), P_{\text{normal}}(v), P_{\text{normal}}(f), API(v)) \]

\( S_{\text{normal}} \) = length of bounding box for normal vertex
\( s = \) vertex-size scale factor

\[ Q_{\text{fisheye}} = F_1 \left( s \times \frac{S_{\text{normal}}}{2}, P_{\text{normal}}(f) \right) \]

\[ S_{\text{geom}} = 2 \min(|Q_{\text{fisheye}} - P_{\text{fisheye}}|, |Q_{\text{fisheye}} - P_{\text{fisheye}}|) \]

\[ S_{\text{fisheye}} = S_{\text{geom}}(c \times API)^e \quad c, e \text{ are constants} \]
**New Vertex Size**

Vertex size is a function of the normal size, the distance from the focus, and the API of the vertex.

- Look at length of side of bounding box.
- Find a point that is
  - $s^* \text{ length}/2$ away from center of vertex
  - $s$ is a vertex-size scale factor
  - Move in direction opposite of the focus
- Convert to fisheye coordinates
- Scale by API
Figure 1: A graph with 134 vertices and 338 edges. The vertices represent major cities in the United States, and the edges represent paths between neighboring cities. (Typically, the edges would be annotated with the distance and driving time between the cities.) The a priori importance value assigned to each vertex is proportional to the population of the corresponding city. Fisheye views of this graph appear in Figures 2–6.
Figure 2: A fisheye view of the graph in Figure 1. The focus is on St. Louis. (The values of the fisheye parameters are $J = 5$, $\epsilon = 0$, $\kappa = 0$, $VWcutoff = 0$; the meanings of these parameters are explained in Sections 4 and 6.)
Figure 2: A fisheye view of the graph in Figure 1. The focus is on St. Louis. (The values of the fisheye parameters are $J = 5$, $r = 0$, $\varepsilon = 0$, $VWcutoff = 0$; the meanings of these parameters are explained in Sections 4 and 6.)

Figure 3: A fisheye view of the graph in Figure 1, with less distortion than in Figure 2. The values of the fisheye parameters are $J = 2$, $r = 0.5$, $\varepsilon = 0.5$, $VWcutoff = 0$. 
Figure 3: A fisheye view of the graph in Figure 1, with less distortion than in Figure 2. The values of the fisheye parameters are $d = 2$, $e = 0.5$, $\kappa = 0.5$, $VWcutoff = 0$.

Figure 5: A fisheye view of the graph in Figure 1. Compare this to Figure 3, with the same distortion and the same focus. Here, the important vertices are larger than in Figure 3, but the unimportant ones are smaller. The values of the fisheye parameters are $d = 2$, $e = 0.75$, $\kappa = 0.75$, $VWcutoff = 0$. 
Computing Detail

- The amount of detail is proportional to the size of the vertex in the fisheye coordinates
- But must not exceed a maximum amount of detail
  - determine font size
  - determine size of nodes to avoid overlap
  - determine resolution of an image
Figure 3: A fisheye view of the graph in Figure 1, with less distortion than in Figure 2. The values of the fisheye parameters are \( J = 2, \epsilon = 0.5, \sigma = 0.5, VW_{something} = 0 \).

Figure 6: A fisheye view of the graph in Figure 1, with unimportant vertices eliminated. Compare this to Figure 3, with the same values of the fisheye parameters, except for the value at which unimportant vertices are eliminated. The values of the fisheye parameters are \( d = 2, \epsilon = 0.5, \sigma = 0.5, VW_{something} = 0.2 \).
Figure 14: A graph with 100 vertices and 124 edges. All edges point downwards. The API of each vertex is related to its display level (e.g., the root has the highest API of 8, node 33 has an API of 4, and node 86 has an API of 2).

Figure 15: A graphical fisheye view of Figure 14. The focus is the vertex labeled 48.
bost.ocks.org/mike/fisheye/
| Dec 18 | Dec 19 | Dec 20 | Dec 21 | Dec 22 | Dec 23 | Dec 24 | Dec 25 | Dec 26 | Dec 27 | Dec 28 | Dec 29 | Dec 30 | Jan 1 | Jan 2 | Jan 3 | Jan 4 | Jan 5 | Jan 6 | Jan 7 | Jan 8 | Jan 9 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| **CLEAN** | **JACK SMITH** | **LEAVE MOG** | **IN** | **FINISH** | **BANKING** | **LEAVE** | **VACATION** | **VACATION** | **VACATION** | **BROAD** | **MOVERS** | **NEW YEARS** | **BACK TO** | **PTAG** | **PTAG** | **PTAG** | **PTAG** | **PTAG** | **PTAG** | **PTAG** |
| 14     | 15     | 16     | 17     | 18     | 19     | 20     | 21     | 22     | 23     | 24     | 25     | 26     | 27     | 28     | 29     | 30     | 31     | 1       | 2       | 3       |
| **DINN** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** | **DINNER** |
| 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. | 10:30a.m. |
| **FISHEYE VIEW CALENDAR** | | | | | | | | | | | | | | | | | | | | | |
Refined with fractal algorithms. A multiscalable font mode is used. Each line is displayed in a font size corresponding to the fractal value of the line.
The inspiration of document lens from the magnifying lens.
Document lens, 3-D effect, no waste of corner space
Organized/Stylistic Layouts
HOLA: Human-like Orthogonal Layout

Study how humans lay-out a graph
Try to emulate layout

Left: human, middle: conventional algo, right new algo

[Kieffer et al, InfoVis 2015]
Manipulating Levels

First interactive tree manipulation

“The mother of all demos”
https://www.youtube.com/watch?v=yJDv-zdhzMY
Styled / Restricted Layouts

Circular Layout
Node ordering
Edge Clutter

c. 3% of all possible edges
c. 6.3% of all possible edges
Example: MizBee [Meyer et al. 2009]
Reduce Clutter: Edge Bundling

Holten et al. 2006
Bundling Strength

mbostock.github.com/d3/talk/20111116/bundle.html
Fixed Layouts

Can’t vary position of nodes
Edge routing important
Techniques
Drawing rooted trees

• Recursion makes it elegant and fast to draw trees
• Possible approaches
  – Indentation
  – Node link
  – Enclosure
  – Layering
Indentation

- Fast and simple to implement
- Could be text only (or HTML)
- Lots of scrolling for big trees
- Easy to lose context

```java
function draw(node: Node, depth: int) {
    println(<depth spaces> + nodelabel);
    for each child c do
        draw(c, depth+1)
    }

draw(root,0);```

Node link diagrams

Naïve recursive approach
Reingold-Tilford type algorithms

• Criteria:
  – Nodes layered by depth in tree
  – No edge crossings
  – Similar subtrees drawn in similar ways
  – Compact representation

• Approach:
  – Bottom up recursive approach
  – For each parent make sure every subtree is drawn
  – Pack subtrees as closely as possible
  – Center parent over subtrees
Recap - Good Criteria

Minimized edge crossings
Minimized distance of neighboring nodes
Minimized drawing area
Uniform edge length
Minimized edge bends
Maximized angular distance between different edges
Aspect ratio about 1 (not too long and not too wide)
Symmetry: similar graph structures should look similar

list adapted from Battista et al. 1999
Tidier Drawings of Trees

EDWARD M. REINGOLD AND JOHN S. TILFORD

Abstract—Various algorithms have been proposed for producing 'tidy' drawings of trees—drawings that are aesthetically pleasing and use minimum drawing space. We show that these algorithms contain some difficulties that lead to aesthetically unpleasing, wider than necessary drawings. We then present a new algorithm with comparable time and storage requirements that produces tidier drawings. Generalizations to forests and many trees are discussed, as are some problems in discretization when alphanumeric output devices are used.

Index Terms—Data structures, trees, tree structures.

INTRODUCTION

In a recent article [6], Weatherall and Shannon presented algorithms for producing “tidy” drawings of trees—drawings that use as little space as possible while satisfying certain aesthetics. The basic task is the assignment of x and y coordinates to each node of a tree after which a straightforward

Fig. 1. Final positioning of example tree as drawn by Algorithm WS.
Criteria:
- Nodes layered by depth in tree
- No edge crossings
- Similar subtrees drawn in similar ways
- Compact representation

Approach:
- Top down recursive approach
- For each parent make sure every subtree is drawn
- Pack subtrees as closely as possible
- Center parent over subtrees
Sample

Subtrees already drawn with RT algorithm
Compare right and left trees and compress
Center parent over subtrees and update
Explicit Tree Visualization

Reingold–Tilford layout

http://billmill.org/pymag-trees/
Sujiyama et al.

Methods for Visual Understanding of Hierarchical System Structures

KOZO SUGIYAMA, MEMBER, IEEE, SHOJIRO TAGAWA, AND MITSUHIKO TODA, MEMBER, IEEE

Abstract — Two kinds of new methods are developed to obtain effective representations of hierarchies automatically: theoretical and heuristic methods. The methods determine the positions of vertices in two steps: First the order of the vertices in each level is determined to reduce the number of crossings of edges. Then horizontal positions of the vertices are determined to improve further the readability of drawings. The theoretical methods are useful in recognizing the nature of the problem, and the heuristic methods make it possible to enlarge the size of hierarchies with which we can deal. Performance tests of the heuristic methods and several applications are presented.

DIGRAPHS are widely utilized in modeling structures of complex systems in various fields where vertices correspond to elements of the systems and edges correspond to relations among the elements. It is empirically recognized that drawings of the digraphs are useful as a visual aid to understand overall images of the structures of the complex systems. For example, block diagrams and flowcharts are commonly used by engineers in performing tasks such as structural modeling, project scheduling, computer programming, etc.

Multilevel digraphs, called hierarchies, constitute an important subclass of digraphs. Interpretive Structural Modeling (ISM) [1] and Program Evaluation and Review Technique (PERT) [2] are well-known techniques in which hierarchies are utilized for modeling structures of systems.

0018-9472/81/0200-0109$00.75 ©1981 IEEE
Sugiyama type layouts

- Great for graphs that have an intrinsic ordering
- ‘Depth’ in graph mapped to one axis
Sugiyama process step 1

- Create layering of graph
  - From domain specific knowledge
  - No cycles
  - Longest path from root
  - Algorithmically determine best layering (NP-Hard)
- Dummy nodes for long edges
Sugiyama process step 2

- Layer by layer crossing minimization (NP)
- Number of heuristics available
Sugiyama process step 3

- Final assignment of x-coordinates
- Routing of edges
Sugiyama

+ Nice, readable top down flow
+ Relatively fast (depending on heuristic used for crossing minimization)
- Not really suitable for graphs that don’t have an intrinsic top down structure
- Hard to implement (use free graphviz lib instead, http://www.graphviz.org)
Sugiyama

- Great for graphs that have an intrinsic ordering
- ‘Depth’ in graph mapped to one axis
Fig. 2. An example of the improvements of a drawing. Step II: (a) → (b). Step III: (b) → (c).
Figure 20.7  Visualizations of the metabolic pathway shown in Figure 20.5 using (a) a force-directed algorithm [KR89] and (b) a hierarchical approach [STT81].
Tree Interaction, Tree Comparison
Design Critique
Connected China

https://goo.gl/YXkWYX
http://china.fathom.info/
Matrix Representations
Matrix Representations

Instead of node link diagram, use adjacency matrix
Matrix Representations

Examples:
Matrix Representations

Well suited for neighborhood-related TBTs

Not suited for path-related TBTs

van Ham et al. 2009
Shen et al. 2007
Order Critical!
Matrix Representations

Pros:
- can represent all graph classes except for hypergraphs
- puts focus on the edge set, not so much on the node set
- simple grid -> no elaborate layout or rendering needed
- well suited for ABT on edges via coloring of the matrix cells
- well suited for neighborhood-related TBTs via traversing rows/columns

Cons:
- quadratic screen space requirement (any possible edge takes up space)
- not suited for path-related TBTs
Special Case: Genealogy
Hybrid Explicit/Matrix

[Image of a diagram with nodes and connections labeled as Bederson et al., Plaisant et al., Shneiderman et al., and PARC.]

NodeTrix
[Henry et al. 2007]
Implicit Layouts

Explicit (Node-Link)

Matrix

Implicit
Explicit vs. Implicit Tree Vis

Fig. 2. (a) Explicit, node-link layout, (b) Implicit layout by inclusion, (c) Implicit Layout by overlap, (d) Implicit layout by adjacency.
Tree Maps

Johnson and Shneiderman 1991
Zoomable Treemap
Example: Interactive TreeMap of a Million Items
Sunburst: Radial Layout

[Sunburst by John Stasko, Implementation in Caleydo by Christian Partl]
Others

Icicle Plot

http://www.example.com/icicle.png
Implicit Representations

Pros:
- space-efficient because of the lack of explicitly drawn edges: scale well up to very large graphs
- in most cases well suited for ABTs on the node set
- depending on the spatial encoding also useful for TBTs

Cons:
- can only represent trees
- since the node positions are used to represent edges, they can no longer be freely arranged (e.g., to reflect geographical positions)
- useless to pursue any task on the edges
- spatial relations such as overlap or inclusion lead to occlusion
Tree Visualization Reference
Summary

Node–Link Diagrams
Connection Marks

Adjacency Matrix
Derived Table

Enclosure
Containment Marks

Munzner 2014