Data visualization

Networks

David Hoksza
http://siret.cz/hoksza
Application areas

• Physical networks
  • Power grid, train routes

• Community modelling
  • Users interaction

• Information (knowledge) networks
  • Citations, web pages, peer-to-peer networks, preference networks

• Biology
  • Metabolic pathways, genetic regulatory networks, protein-protein interaction networks
Graph drawing

• Mapping graph attributes (derived from the underlying data) to the aesthetics geometric objects, which represent the graph
  • We will focus on general graphs (there exist separate solutions for trees)

• **Graph drawing** problem

  • **Input**: set of nodes and edges

  • **Output**: positions of the nodes and curve shapes, to represent the edges → graph layout
Graph drawing criteria (1)

Syntactic Validity
- Node Overlap
- Node/Link Intersection

Perceptual Organization
- Alignment
- Symmetry
- Zoning
- T-Shape

Aesthetic Optimality
- Diagram Area
- Number of Link Crossings
- Evenness of Link Lengths
- Sum of Link Lengths

Graph drawing criteria (2)

• However, some of aesthetics criteria can be mutually exclusive

Symmetry optimization

# edge crossings (planarity) optimization

source: Buchheim et al.: Crossings and planarization. Handbook of Graph Drawing and Visualization (2013)
Basic graph layout strategies

• **Tree** layouts (not covered here)

• **Energy-based** layouts
  • Often called force-directed placement algorithms or spring embedders

• **Circular** layouts
Force-directed layouts
Force-directed layout

• **Force-directed placement** (FDP)
  • Eades (84)
  • Kamada and Kawai (89)
  • Fruchterman and Reingold (90)
  • Davidson and Harel (96)
• **Optimization**
  • Multi-scale approaches
  • ...
• **Clustering**
  • Linlog (Noack (07))
  • ...

Classical approaches
FDP – Eades (1)

• **Spring-based** mechanical model that embeds a graph in 2D
  • **Vertices** = charged steel **rings**, **edges** = **springs**
  • Let the **spring forces** on the rings move the **system to a minimal energy state**
    • Spring exerted force \( f_a = c_1 \times \log(d/c_2) \)
    • Nonadjacent vertices **repel** each other with force \( f_r = c_3 / \sqrt{d} \)

• **Algorithm**
  1. Place vertices in random locations. Set \( i = 1 \)
  2. Stop if \( i = M, i = i + 1 \)
  3. Calculate forces acting on each vertex
  4. Move each vertex \( c_4 \times \text{force on vertex} \)

Linear springs (Hook’s Law) turn out to be too strong when vertices are far apart.
FDP - Fruchterman and Reingold

• Includes (even) **vertex distribution** in the layout into the set of criteria

\[ f_a = \frac{d^2}{k} \quad f_r = -\frac{k^2}{d} \quad k = c \frac{\sqrt{\text{area}}}{\#\text{nodes}} \]

- In the **optimal distance**

• Includes **temperature cooling**
  - Cooling bounds vertex movement similarly to simulated annealing

  \[ f_a = \frac{d^2}{k} \quad f_r = -\frac{k^2}{d} \]

Attraction forces computed for neighboring vertices

Repulsion forces computed for all pairs of vertices
area ← $W \times L$ ;
initialize $G = (V, E)$ ;
k ← $\sqrt{\text{area}/|V|}$ ;
function $f_r(x) = k^2/x$ ;
for $i = 1$ to iterations do
    foreach $v \in V$ do
        v.disp := 0; ;
        foreach $u \in V$ do
            if ($u \neq v$) then
                $\Delta \leftarrow v.pos - u.pos$ ;
                v.disp ← v.disp + $(\Delta/|\Delta|) \times f_r(|\Delta|)$ ;
            endif
        endforeach
    endforeach
    function $f_a(x) = x^2/k$ ;
    foreach $e \in E$ do
        $\Delta \leftarrow e.v.pos - e.u.pos$ ;
        e.v.disp ← e.v.disp - $(\Delta/|\Delta|) \times f_a(|\Delta|)$ ;
        e.u.disp ← e.u.disp + $(\Delta/|\Delta|) \times f_a(|\Delta|)$ ;
    endforeach
    foreach $v \in V$ do
        /* limit max displacement to frame; use temp. $t$ to scale */
        v.pos ← v.pos + (v.disp/|v.disp|) \times \min(v.disp, t);
        v.pos.x ← \min(W/2, \max(-W/2, v.pos.x));
        v.pos.y ← \min(L/2, \max(-L/2, v.pos.y));
    endforeach
    $t \leftarrow \text{cool}(t)$ ;
endfor
/* frame: width $W$ and length $L$ */
/* place vertices at random */
/* compute optimal pairwise distance */
/* compute repulsive force */
/* compute attractive force */
/* $e$ is ordered vertex pair .v and .u */
• Adds **edge crossings minimization** criterion and uses **simulated annealing** for optimization

• Simulated annealing
  1. Set initial configuration $\sigma$
  2. Set initial temperature $T$
  3. Choose a new configuration $\sigma'$ from a close neighborhood of $\sigma$
  4. Evaluate energy functions $E$ and $E'$ of configurations $\sigma$ and $\sigma'$
  5. If $(E' < E \text{ or } r < e^{-(E-E')/T})$ then $\sigma \leftarrow \sigma'$
  6. Reduce temperature and go to 3
FDP – Davidson and Harel (2)

• **Energy function**
  1. Repulsive component
  2. Placement component
  3. Edge length component
  4. Crossings component

• **Cooling**
  • Displacement of a vertex limited by circle of decreasing radius centered in current position

\[ \sum_{i,j} \frac{\lambda_1}{d_{ij}^2} \]
\[ \lambda_2 \left( \frac{1}{r_i^2} + \frac{1}{l_i^2} + \frac{1}{t_i^2} + \frac{1}{b_i^2} \right) \]
\[ \lambda_3 d_k^2 \]
\[ \lambda_4 (\#\text{crossings}) \]

Distance to the right, left, top and bottom edges. High \( \lambda_2 \) value favors centered layouts.

\( \lambda_3 \) is a normalization constant causing shortening edges to the necessary minimum.

Increasing \( \lambda_4 \) more heavily penalizes crossings.
FDP - Kamada and Kawai (1)

• **Replacement of the ideal spring lengths by shortest path distance**
  → graph-theoretic approach
  • In a connected graph, all pairs of vertices are connected → no need for repulsive forces

• **Correlates graph distances with the Euclidean distances**
  minimizing the energy of the system corresponds to minimizing the difference between the geometric and graph distances

\[ E = \sum_{i<j} \frac{1}{2} k_{ij} (|\text{pos}_i - \text{pos}_j| - l_{ij})^2 \]

\[ l_{ij} = L \times d_{ij} \quad L = \frac{\text{display\_width}}{\max_{i<j} d_{ij}} \quad k_{ij} = \frac{K}{d_{ij}^2} \]
FDP - Kamada and Kawai (2)

\[ E = \sum_{i<j} \frac{1}{2} k_{ij}(|\text{pos}_i - \text{pos}_j| - l_{ij})^2 = \sum_{i<j} \frac{1}{2} k_{ij} \left( (x_i - x_j)^2 + (y_i - y_j)^2 + l_{ij}^2 - 2l_{ij}\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right) \]

- At a local minimum all the partial derivatives of \( E(x_1, y_1, \ldots, x_n, y_n) \) equal zero (\( \forall j \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} = 0 \))

- In Kamada and Kawai, one node at a time is optimized using the Newton-Raphson method. At each step, the node \( m \) with the highest distance from the minimum (maximum gradient value) is chosen (the others are frozen):

\[ \Delta_m = \sqrt{\left( \frac{\partial E}{\partial x_m} \right)^2 + \left( \frac{\partial E}{\partial y_m} \right)^2} \]
compute pairwise distances $d_{i,j}$ for $1 \leq i \neq j \leq n$;
compute pairwise ideal lengths $l_{i,j}$ for $1 \leq i \neq j \leq n$;
compute pairwise spring strength $k_{i,j}$ for $1 \leq i \neq j \leq n$;
initialize particle positions $p_1, p_2, \ldots, p_n$;

while $(\max_i \Delta_i > \epsilon)$ do
    let $p_m$ be the particle satisfying $\Delta_m = \max_i \Delta_i$;
    while $(\Delta_m > \epsilon)$ do
        compute $\delta x$ and $\delta y$ by solving the following system of equations:
        \[
        \frac{\partial^2 E}{\partial x_m^2}(x_m^{(t)}, y_m^{(t)})\delta x + \frac{\partial^2 E}{\partial x_m \partial y_m}(x_m^{(t)}, y_m^{(t)})\delta y = -\frac{\partial E}{\partial x_m}(x_m^{(t)}, y_m^{(t)});
        \]
        \[
        \frac{\partial^2 E}{\partial y_m^2}(x_m^{(t)}, y_m^{(t)})\delta x + \frac{\partial^2 E}{\partial y_m \partial x_m}(x_m^{(t)}, y_m^{(t)})\delta y = -\frac{\partial E}{\partial y_m}(x_m^{(t)}, y_m^{(t)});
        \]
        $x_m \leftarrow x_m + \delta x$;
        $y_m \leftarrow y_m + \delta y$;

Newton-Raphson method:

• Approximation of the zero of $f(x)$ by iterating through $x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)}$.
• $f'(x_n)dx = -f(x_n)$, $dx = x_{n+1} - x_n$
Relation of KK algorithm to MDS

- The layout energy function \( E = \sum_{i<j} \frac{1}{2} k_{ij} (|\text{pos}_i - \text{pos}_j| - l_{ij})^2 \) is basically the same as the stress known in MDS → instead of the Newton method stress majorization can be used, which is guaranteed to converge

\[ |V|=882, |E|=1533 \]

Aesthetic criteria of FDP layouts

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric</td>
<td>YES</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evenly distributed nodes</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Uniform edge length</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Minimized edge crossings</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Pros and cons of FDP

- Easy interpretation
- Easy implementation
- Easy to add new heuristics
- Run time
- Can end up in local minima
- Does not reflect the inherent graph cluster structure (if required) – see the following slides
Cluster separation of FDP layouts
Dummy attractors

• One can achieve better separation of clusters based on underlying information by adding dummy attractors
  • Requires prior knowledge about the structure of the underlying data
Graph-clustering focused energy modelling

• Layout algorithms can be viewed as having two components — energy model and energy minimization algorithm

• Up to now, the focus of the energy models was on producing generally readable layouts enforced by small and uniform edge lengths → tendency to group large-degree nodes in the center of the layout

• There exist energy models focusing rather on good separation of clusters → LinLog model
LinLog energy models (1)

- Two energy models
  - Node-repulsion
  - Edge-repulsion

- **Not biased** towards grouping together **nodes with high degree** → appropriate for many real-world graphs with right-skewed degree distributions

LinLog energy models (2)

• **Node-repulsion** model of a layout \( p \)

\[
U_{Node}(p) = \sum_{\{u,v\} \in E} ||p(u) - p(v)|| - \sum_{\{u,v\} \in V^2} \ln ||p(u) - p(v)||
\]

Attraction forces

• **Edge-repulsion** model of a layout \( p \)

\[
U_{Edge}(p) = \sum_{\{u,v\} \in E} ||p(u) - p(v)|| - \sum_{\{u,v\} \in V^2} \deg(u) \deg(v) \ln ||p(u) - p(v)||
\]

Nodes represented in terms of their neighboring nodes → edges

Each node’s influence on the layout is proportional to its degree.
Clustering criteria

- **Good clustering** consists of subgraphs with **many internal** and **few external edges** - clustering criteria formalize this notion

\[ V_1, V_2 \subseteq V: \text{cut}(V_1, V_2) = |\{\{v_1, v_2\} \in E|v_1 \in V_1, v_2 \in V_2 \}| \]

- Expected cut is

\[
\frac{|E|}{(|V|^2-|V|)/2} (|V_1||V_2|) = \frac{2|E|(|V_1||V_2|)}{|V|^2-|V|}
\]

\#possible pairs between two node sets

- which is **biased** towards **uneven cluster sizes** → layout strategies driven with this criterion will favor uneven clusters
LinLog clustering criteria

- **Node-normalized cut**
  \[ \text{ncut}(V_1, V_2) = \frac{\text{cut}(V_1, V_2)}{|V_1||V_2|} \]

  - Still **biased** towards uneven partitions if the **number of edges used** as a **measure** of subgraph size

- **Edge-normalized cut**
  \[ \text{ecut}(V_1, V_2) = \frac{\text{cut}(V_1, V_2)}{\text{deg}(V_1) \text{deg}(V_2)} \]
LinLog energy and clustering

- Minimal-energy **node-repulsion** layouts **minimize the ratio** of the **mean distance** between connected nodes to the mean distance between all nodes.

- The distance of two dense and sparsely connected clusters $V_1, V_2$ approximates $1/\text{ncut}(V_1, V_2)$ in minimal-energy node-repulsion LinLog layouts.

- Minimal-energy **edge-repulsion** layouts **minimize the ratio** of the **mean distance** between connected end nodes to the mean distance between all end nodes.

- The distance of two dense and sparsely connected clusters $V_1, V_2$ approximates $1/\text{ecut}(V_1, V_2)$ in minimal-energy edge-repulsion LinLog layouts.

Each node $v$ can be imagined as an end point of the $\text{deg}(v)$ connected edges.
Pseudorandom graph with 400 nodes consisting of 8 equal-sized clusters:

- Cluster 1-4: intra-cluster edge probability = 1
- Cluster 5-8: intra-cluster edge probability = 0.5
- Cluster 1-4: inter-cluster edge probability = 0.2
- Cluster 5-8: inter-cluster edge probability = 0.05
- Cluster 1-4 vs 5-8: inter-cluster edge probability = 0.1

Speed optimization of the FDP layouts
Barnes-Hut approximation (1)

• Division of space using quad-tree → inner nodes form clusters for approximation


Barnes-Hut approximation (2)

• An object is compared either to other objects or cluster centers based on **Barnes-Hut criterion**

\[
\frac{s}{d} \leq \theta
\]

• If for given level of tree the **condition holds**, **repulsive force** is computed only **with respect to the cluster**

• \( \theta \) affects how often the criterion is applied \( \rightarrow O(n^2) \) vs \( O(n) \)
Barnes-Hut approximation (3)

- **Repulsive forces** need to be modified
  \[ f_r(x_i, x_j) = -\frac{k^2}{d(x_i, x_j)} \rightarrow f_r(x_i, x_Q) = -\frac{|Q|k^2}{d(x_i, x_Q)} \quad \cdots \quad x_Q = \frac{\sum_{j \in Q} x_j}{|Q|} \]

- The algorithm is modified so that **quad tree is constructed** at the **beginning of each iteration**
  - Complexity of construction is \( O(dn) \), where \( d \) is depth of the tree
- **Repulsive forces** are computed using the modified \( f_r \) to **clusters** centers of gravitation only
Small world networks

• Many of the real-world networks have so-called small world networks (SWN) property
  • Average path length in a SWN compares to a path length in a random graph
  • (Local) clustering index of SWN nodes are on average orders of magnitudes larger

$$c(v) = \frac{|\text{edges}(\text{neighbors}(v))|}{(k(k - 1))/2}$$

Number of vertices in the neighborhood of $v$

<table>
<thead>
<tr>
<th>Graph</th>
<th>Clustering index</th>
<th>Ave Path Length</th>
<th>Random graph (same number of nodes and edges)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB</td>
<td>0.9666</td>
<td>3.2043</td>
<td>0.0243</td>
</tr>
<tr>
<td>“Resyn assistant”</td>
<td>0.9518</td>
<td>3.2847</td>
<td>0.1942</td>
</tr>
<tr>
<td>Mac OS 9</td>
<td>0.3875</td>
<td>2.8608</td>
<td>0.0179</td>
</tr>
<tr>
<td>Web²</td>
<td>0.1078</td>
<td>3.1</td>
<td>2.3e-4</td>
</tr>
<tr>
<td>.edu sites²</td>
<td>0.156</td>
<td>4.062</td>
<td>0.0012</td>
</tr>
</tbody>
</table>

Multi-scale algorithms

• **Most networks** are not only SWN, but their **components form SWN** as well → hierarchy of SWNs → **multi-scale networks**
  - Leaf level of the hierarchy consists of cliques
• Series of graph representations with **different levels of details** → **optimization of the layout** with respect to these coarse abstractions of the original graph
  - One has to define the notion of coarse-scale representations of a graph, in which the combinatorial structure is significantly simplified but features important for visualization are well preserved.

• The multi-scale algorithms differ in
  - Bottom-up vs top-down
  - Coarsening
  - Underlying layout methods

We will show here just two examples
Multi-scale algorithms – Harel & Koren

• **Coarsening** process based on a *k*-center approximation –
  • Find *k* nodes, so that the maximum distance of any node to arbitrary of the *k* nodes is minimized
  • NP-hard, but can be 2-approximated

• The algorithm in each iteration
  • Finds centers of clusters
  • Finds layout for the cluster centers
  • Moves nodes close to their respective cluster centers
Input: graph $G = (V, E)$
Output: Find $L$, a nice layout of $G$

$\text{Iterations} = 4$ ; /* number of iterations in local beautification */
$\text{Ratio} = 3$ ; /* ratio of vertex sets in consecutive levels */
$\text{Rad} = 7$ ; /* radius of local neighborhood */
$\text{MinSize} = 10$ ; /* size of coarsest graph */

Compute all-pairs shortest path lengths $d_{i,j}$;
Initialize layout $L$ by placing vertices at random;
$k \leftarrow \text{MinSize}$;

while $k \leq |V|$ do

$C \leftarrow \text{K-Centers}(G(V, E), k)$;
$\text{radius} = \max_{v \in C} \min_{u \in C} \{d_{vu}\} \ast \text{Rad}$;
$\text{LocalLayout}(d_{C \times C}, L(C), \text{radius}, \text{Iterations})$;
foreach $v \in V$ do

$L(v) \in L(\text{center}(v)) + \text{rand}$;
$k \leftarrow k \ast \text{Ratio}$

return $L$;

K-Centers($G(V, E), k$);
Input: Graph $G = (V, E)$ and constant $k$
Output: Compute set $S \subseteq V$ of size $k$, such that $\max_{v \in V} \min_{s \in S} \{d_{sv}\}$ is minimized
$S \leftarrow \{v\}$ for some arbitrary $v \in V$;
for $i = 2$ to $k$ do

find vertex $u$ farthest away from $S$;
(i.e., such that $\min_{s \in S} \{d_{us}\} \geq \min_{s \in S} \{d_{ws}\}, \forall w \in V$);
$S \leftarrow S \cup \{u\}$;

return $S$;

LocalLayout($d_{V \times V}, L, k$, Iterations);
Input: APSP matrix $d_{V \times V}$, layout $L$, constants $k$ and Iterations
Output: Compute locally nice layout $L$ by beautifying $k$-neighborhoods
for $i = 1$ to $\text{Iterations} \ast |V|$ do

Choose the vertex $v$ with the maximal $\Delta_k^v$;
Compute $\delta_k^v$ as in Kamada-Kawai;
$L(v) \leftarrow L(v) + (\delta_k^v(x), \delta_k^v(y))$;
Closest center to $v$
Assign
Multi-scale algorithms - Auber

- Iteratively **filter out edges** with low edge clustering index $\rightarrow$ remaining **connected components** form nodes of the **quotient graph**

$$s(A, B) = \text{cut}(A, B)/|A||B|$$

**strength**$(u, v)$

$$= s(M(u), W(u, v)) + s(W(u, v), M(v)) + s(M(u), M(v)) + s(W(u, v)) + |W(u, v)|/|M(u) + W(u, v) + M(v)|$$

Any edge connecting two of $M(u), M(v), W(u, v)$ is a part of a 4-cycle going through $(u, v)$

Ratio of 3-cycles using $\{u, v\}$
Very complex or specific networks
In case of complex data, node-link diagrams are notoriously difficult to visualize and interpret → hairballs

- *Junk food of network visualization* — very low nutritional value, leaving the user hungry

- A complex network does not necessarily communicate complex information
Interpretability of graph layouts

• Interface problem
  • Graph sizes tend to grow while the size of the medium we use to visualize them tends to stay the same

• Visual encodings

• Smart layouts
  • Flow diagrams, Circular layouts, matrix methods, hive plots, hierarchies, …
  • Often aim at specific use cases

• Interactivity (navigation)
  • Zooming, panning, fish-eye, …
Visual encodings

Nodes

Edges
Flow diagrams

• **Sankey diagrams** – flow magnitude proportional to connection width

[Image of Sankey diagram]

https://bost.ocks.org/mike/sankey/
Circular layout

- Vertices placed on the circumference of an embedding circle
  - Often the vertices are first grouped into clusters

Basic layout

- Edges drawn as straight lines

Edge bundling

- Bundle the paths of edges in cluster-like structures in a diagram so that they follow similar routes

Combined circular layout


source: [http://bl.ocks.org/mbostock/7607999](http://bl.ocks.org/mbostock/7607999)

Circos

- **Software** for laying out **relationships** (graphs)

- Basically **converts tabular** data to **circular** layout (any graph can be expressed as a table)
Applying Circos to Visualizing Tables using a Circular Layout

Relationships between elements in your data set are indicated by links. Links can indicate a simple relationship (A-B), a relationship that has positional information (A-C), or a unidirectional relationship (A-D). In each case, the link is formatted differently.

Links with variable thickness can represent the extent or magnitude of the relationship between elements.

When links are colored based on the elements that they relate, spotting patterns is easier. In particular, when relationships have a direction, links can be colored by source or target element.

Ribbons touch the row segments, but terminate a short distance before reaching the column segment.

In the ratio layout, a ribbon represents two cells, except for cells on the diagonal which are represented by a single ribbon. By following the size of the ribbon from one segment to another, you can visually estimate the ratio of (A,B):(B,A).

Transposible cells (e.g., (A,B) and (B,A)) are shown by a single ribbon (right). The ribbon now ends directly at both row and column segments and its ends are of variable thickness, which is the cell value for which the end’s segment is a row. For example, if (A,B)=2 and (B,A)=10 then the ribbon’s end touching A is thickness 2 and the end touching B is thickness 10.

Table visualization is implemented in the tableviewer utility, bundled with Circos. The parse-table script parses a tabular data file into an intermediate format, which is then used by make-conf to create Circos configuration and data files. Circos is then used to generate the final image.
Matrix methods

• Methods based on analysis of the adjacency matrix

• Requires some support from the visualization tool

  • Rows and columns of the matrix can be rearranged → patterns

Hive plots

- **Nodes** mapped to and positioned on radially distributed linear axes → linear layout of nodes
  - Can be divided into segments
- **Edges** drawn as curved links

- Graph structure can be mapped to
  - Axis, Position, Color

source: http://www.hiveplot.net/conference/vizbi2011/poster/krzywinski-hiveplot-poster.png
Hierarchical layouts

- Sunburst
- Circle packing
- Tree map
- Icicle
Interactivity
Zooming & panning

• Not always is the interface problem solvable with the layout algorithm $\rightarrow$ zoom & pan, i.e. navigation in visualization

• **Zooming** is *simple for graphs*, which contain only simple geometrical objects (nodes and edges)
  
  • **Geometric** (standard) – only changes the scale of magnification
  
  • **Semantic** – modifies what is being shown; either more details, different representation of the data or even different data
Focus and context

• Zooming suffers from **loosing context** – when zoomed in, all context is lost → difficult to pan → **decreased usability**

• We need a technique **combining** both **overview** (context) and **detail** information (focus) in one view → **focus+context** techniques
Fisheye (1)

• Popular **distortion** viewing **technique** that magnifies nearby objects while shrinking distant objects

• With **graphs**, we want to see **details of the specific subgraph** while still seeing the **whole structure**

Fisheye (2)

• Distortion technique – user selects a **focus point** and the **layout is distorted**

  • **Polar** transformation – transformation on a polar coordinate system \((r, \theta)\) - distortion applied equally in all the directions from the focus points (\(\theta\) kept unchanged) up to some point
    • suitable for objects such as maps

  • **Cartesian** transformation – scales x and y positions individually, does not require curving the lines
    • better for, e.g., scatter plots

[https://bost.ocks.org/mike/fisheye/](https://bost.ocks.org/mike/fisheye/)
Tools and data
Graph languages - GraphML

- **XML-based format** for exchanging graph structure data
- Stores structural information as well as the graphical information

```xml
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <key id="d0" for="node" attr.name="color" attr.type="string">
    <default>yellow</default>
  </key>
  <key id="d1" for="edge" attr.name="weight" attr.type="double"/>
  <graph id="G" edgedefault="undirected">
    <node id="n0">
      <data key="d0">green</data>
    </node>
    <node id="n1"/>
    <node id="n2">
      <data key="d0">blue</data>
    </node>
    <node id="n3">
      <data key="d0">red</data>
    </node>
    <node id="n4"/>
    <node id="n5">
      <data key="d0">turquoise</data>
    </node>
    <edge id="e0" source="n0" target="n2">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e1" source="n0" target="n1">
      <data key="d1">1.0</data>
    </edge>
    <edge id="e2" source="n1" target="n3">
      <data key="d1">2.0</data>
    </edge>
    <edge id="e3" source="n3" target="n2"/>
    <edge id="e4" source="n2" target="n4"/>
    <edge id="e5" source="n3" target="n5"/>
    <edge id="e6" source="n5" target="n4">
      <data key="d1">1.1</data>
    </edge>
  </graph>
</graphml>
```
Graph languages - DOT

- Plain text graph description language
- Supported by GraphViz, Gephi, ...

```dot
graph graphname {
    // This attribute applies to the graph itself
    size="1,1";
    // The label attribute can be used to change the
    label of a node
    a [label="Foo"];  // Here, the node shape is changed.
    b [shape=box];
    // These edges both have different line properties
    a -- b -- c [color=blue];
    b -- d [style=dotted];
}
```

![Graph diagram]

62
Software

- **GraphViz**
- **Gephi**
- **Cytoscape**
- **Circos**
Network data sets collections

• Stanford Large Network Dataset Collection

• Pajek data sets

• R package igraphdata
Sources

• Roberto Tamassia (2013) Handbook of Graph Drawing and Visualization. Chapman and Hall/CRC

• Chaomei Chen (2006) Information Visualization: Beyond the Horizon. Springer