# Data visualization

Networks

David Hoksza
siret.ms.mff.cuni.cz/hoksza
bioinformatika.mff.cuni.cz/cusbg

# Application areas

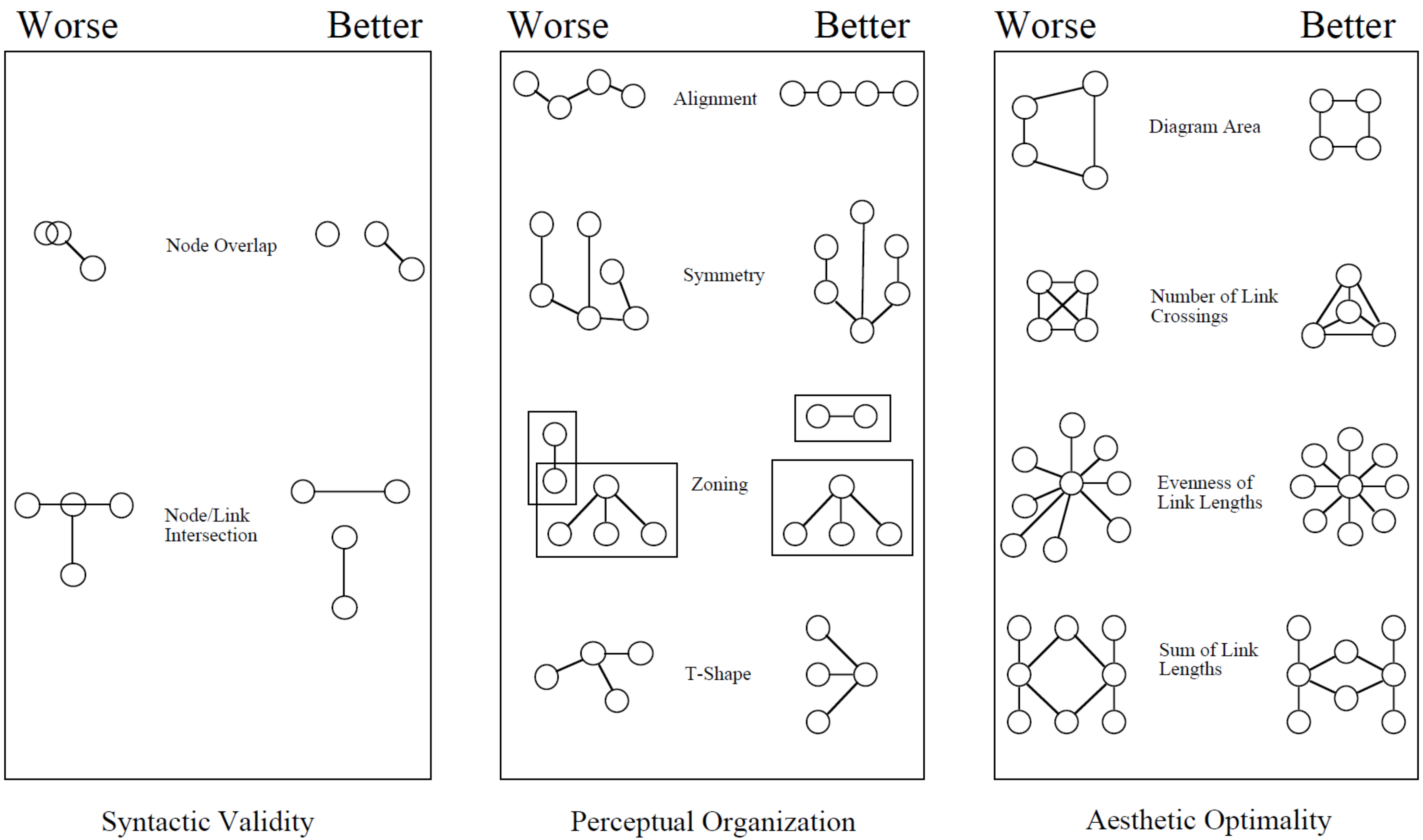
- Physical networks
  - Power grid, train routes
- Community modelling
  - Users interaction
- Information (knowledge) networks
  - Citations, web pages, peer-to-peer networks, preference networks
- Biology
  - Metabolic pathways, genetic regulatory networks, protein-protein interaction networks

# Graph drawing

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

- **Graph drawing** problem
  - **Input**: set of nodes and edges
  - **Output**: positions of nodes and curved shapes representing edges → graph layout

# Graph drawing criteria (1)

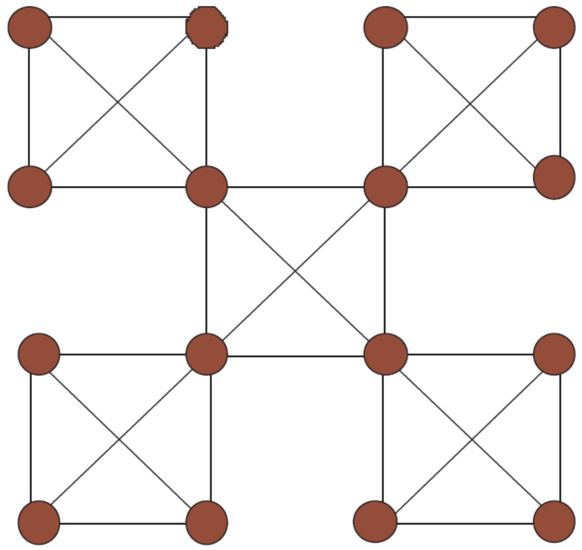




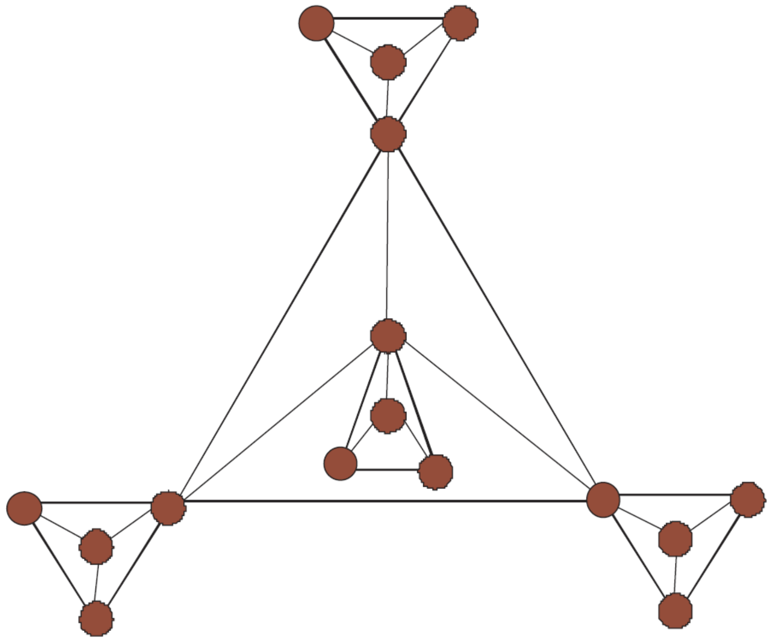

source: Kosak et al. Automating the layout of network diagrams with specified visual organization. Systems, Man and Cybernetics 24(3). 1994

# 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

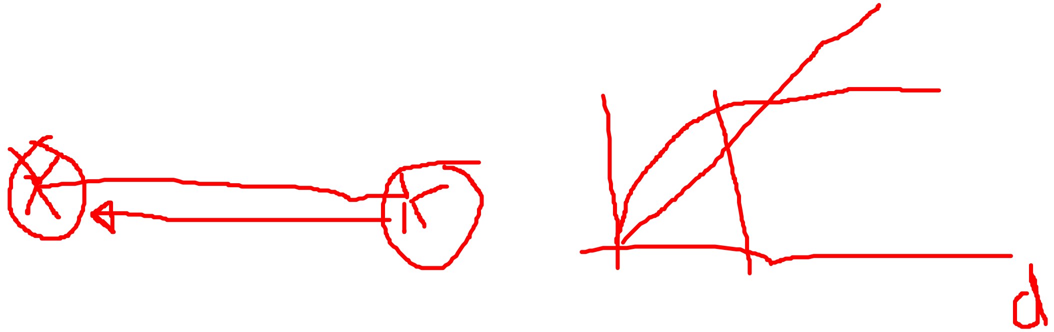
# Force-directed layouts

# 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)

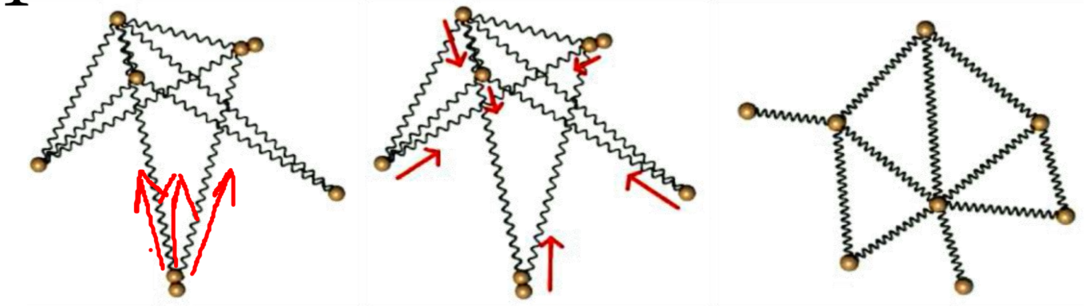



- **Spring-based** mechanical model that embeds a graph in 2D
  - **Vertices** = charged steel **rings**, **edges** = **springs**
  - Let the **spring forces** on the rings move the **system to a minimal energy state**
    - **Spring** exerted force $\boldsymbol{f_a = c_1 \times \log(d/c_2)}$
    - **Nonadjacent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

No force for $d = c_2$

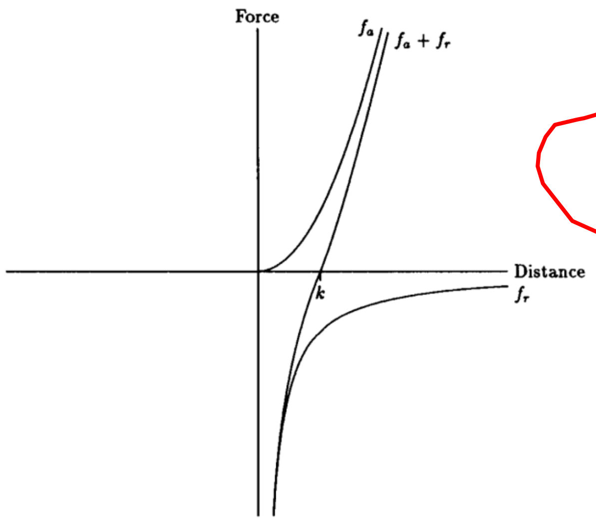
Linear springs (Hook's Law) turn out to be too strong when vertices are far apart.

- **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 $c_4 \times$ force_on_vertex



# FDP - Fruchterman and Reingold

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




$$f_a = \frac{d^2}{k}$$

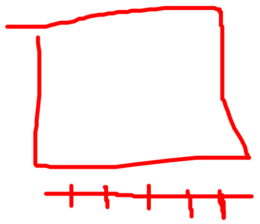
Attraction forces computed for neighboring vertices

$$f_r = -\frac{k^2}{d}$$

Repulsion forces computed for all pairs of vertices

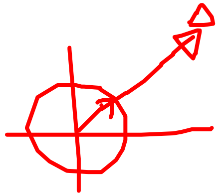
$$k = C\frac{\sqrt{area}}{\#nodes}$$

Optimal distance



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

```
area ← W ∗ L ;                                  /* frame: width W and length L */
initialize G = (V, E) ;                         /* place vertices at random */
k ← √(area/|V|) ;                               /* compute optimal pairwise distance */
function f_r(x) = k²/x ;                        /* compute repulsive force */
for i = 1 to iterations do
    foreach v ∈ V do
        v.disp := 0; ;                          /* initialize displacement vector */
        for u ∈ V do
            if (u ≠ v) then
                Δ ← v.pos − u.pos ;             /* distance between u and v */
                v.disp ← v.disp + (Δ/|Δ|) ∗ f_r(|Δ|) ;   /* displacement */
    function f_a(x) = x²/k ;                    /* compute attractive force */
    foreach e ∈ E do
        Δ ← e.v.pos − e.u.pos ;                 /* e is ordered vertex pair .v and .u */
        e.v.disp ← e.v.disp − (Δ/|Δ|) ∗ f_a(|Δ|);
        e.u.disp ← e.u.disp + (Δ/|Δ|) ∗ f_a(|Δ|);
    foreach v ∈ V do
                          /* limit max displacement to frame; use temp. t to scale */
        v.pos ← v.pos + (v.disp/|v.disp|) ∗ 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));
    t ← cool(t) ;                               /* reduce temperature for next iteration */
```





Stephen G. Kobourov: Spring Embedders and Force Directed Graph Drawing Algorithms. Handbook of Graph Drawing and Visualization (2013)

# FDP – Davidson and Harel (1)

- 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 $\boldsymbol{\sigma'}$ **from a close neighborhood of** $\boldsymbol{\sigma}$
  4. **Evaluate energy functions** $E$ and $E'$ of configurations $\sigma$ and $\sigma'$
  5. **If** $(\boldsymbol{E' < E}$ **or** $\boldsymbol{r < e^{-(E-E')/T}})$ **then** $\boldsymbol{\sigma \leftarrow \sigma'}$
  6. **Reduce temperature** and go to 3

# FDP – Davidson and Harel (2)

- **Energy function**
  1. **Repulsive** component $\sum_{i,j} \frac{\lambda_1}{d_{ij}^2}$
  2. **Placement** component $\lambda_2\left(\frac{1}{r_i^2} + \frac{1}{l_i^2} + \frac{1}{t_i^2} + \frac{1}{b_i^2}\right)$
  3. **Edge length** component $\lambda_3 d_k^2$
  4. **Crossings** component $\lambda_4(\#\text{crossings})$

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

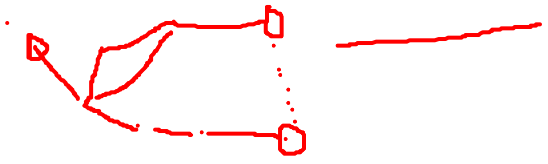
$\lambda_3$ is a normalization constant causing shortening edges to the necessary minimum.

Increasing $\lambda_4$ more heavily penalizes crossings.

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

# FDP - Kamada and Kawai (1)

- **Replacement of the ideal spring lengths by the 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



Energy function to be minimized

$$E = \sum_{i<j} \frac{1}{2} k_{ij} \left( |pos_i - pos_j| - l_{ij} \right)^2$$

"Spring" strength

$$l_{ij} = L \times d_{ij} \qquad L = \frac{\text{display\_width}}{\max_{i<j} d_{ij}} \qquad k_{ij} = \frac{K}{d_{ij}^2}$$

Shortest path between $i$ and $j$

# FDP - Kamada and Kawai (2)

$$E = \sum_{i<j} \frac{1}{2} k_{ij} \left( \left| pos_i - pos_j \right| - l_{ij} \right)^2 = \sum_{i<j} \frac{1}{2} k_{ij} \left( \left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2 + l_{ij}^2 - 2 l_{ij} \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \right)$$

- At a **local minimum all the partial derivatives of $\boldsymbol{E(x_1, y_1, \ldots, x_n, y_n)}$ 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 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:

$$\frac{\partial^2 E}{\partial x_m^2}(x_m^{(t)}, y_m^{(t)})\delta x + \frac{\partial^2 E}{\partial x_m \partial y_m}(x_m^{(t)}, y_m^{(t)})\delta y = -\frac{\partial E}{\partial x_m}(x_m^{(t)}, y_m^{(t)});$$

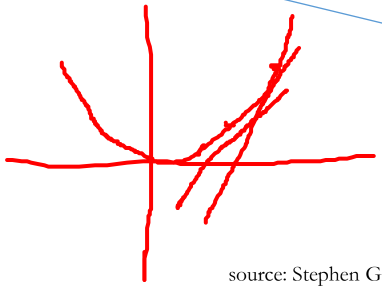
$$\frac{\partial^2 E}{\partial y_m \partial x_m}(x_m^{(t)}, y_m^{(t)})\delta x + \frac{\partial^2 E}{\partial y_m^2}(x_m^{(t)}, y_m^{(t)})\delta y = -\frac{\partial E}{\partial y_m}(x_m^{(t)}, y_m^{(t)})$$

    $x_m \leftarrow x_m + \delta x$;

    $y_m \leftarrow y_m + \delta y$;

$O(n^3)$ using Floyd-Warshall algorithm

Two−dimensional Newton−Raphson method

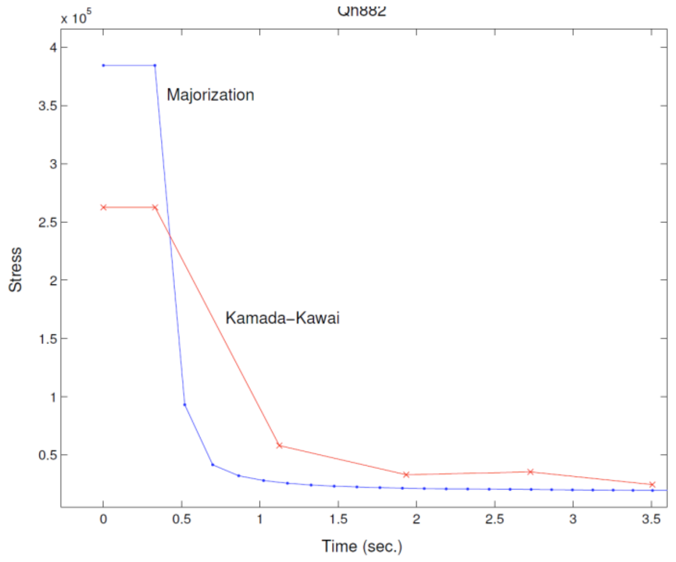




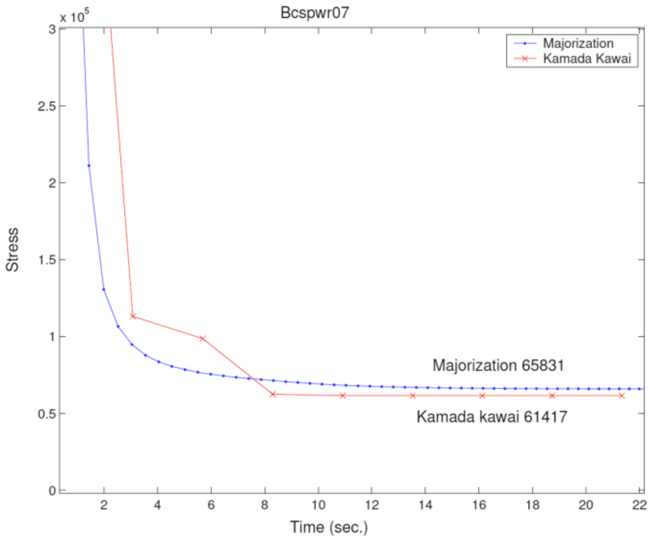
source: Stephen G. Kobourov: Spring Embedders and Force Directed Graph Drawing Algorithms. Handbook of Graph Drawing and Visualization (2013)

# Relation of KK algorithm to MDS

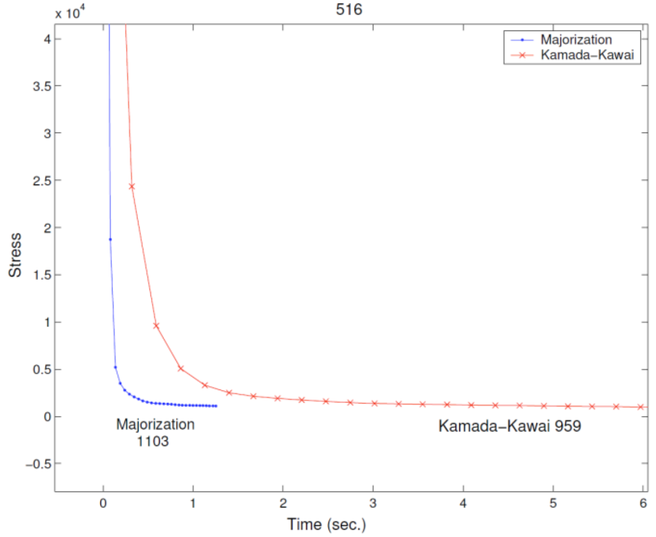
- The layout **energy function** $E = \sum_{i<j} \frac{1}{2} k_{ij} \left( |pos_i - pos_j| - l_{ij} \right)^2$ is basically the same as the **stress** known in MDS → instead of the Newton method **stress majorization** can be used, which is guaranteed to converge




|V|=882, |E|=1533




|V|=882, |E|=1533




|V|=516, |E|=729

source: Gansner et al.: Graph Drawing by Stress Majorization. . In Proceedings 12th Symposium on Graph Drawing (GD) (2004)

# Aesthetic criteria of FDP layouts

| Criteria | Eades (1984) | Kamada and Kawai (1989) | Fruchterman and Reingold (1991) | Davids and Harel (1996) |
|---|---|---|---|---|
| Symmetric | YES | YES | | |
| Evenly distributed nodes | | YES | YES | YES |
| Uniform edge length | YES | YES | YES | YES |
| Minimized edge crossings | | YES | YES | YES |

Source: Chen, Information visualization: Beyond the Horizon. Springer (2006)

# Pros and cons of FDP

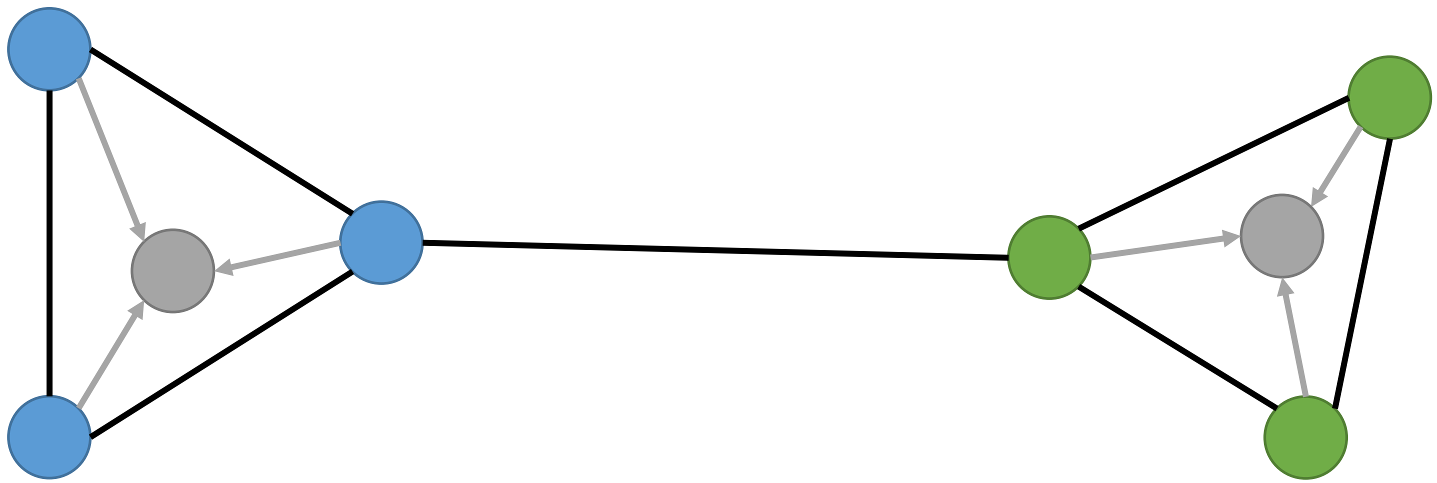
- Easy **interpretation**
- Easy **implementation**
- Easy to add **new heuristics**

- **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



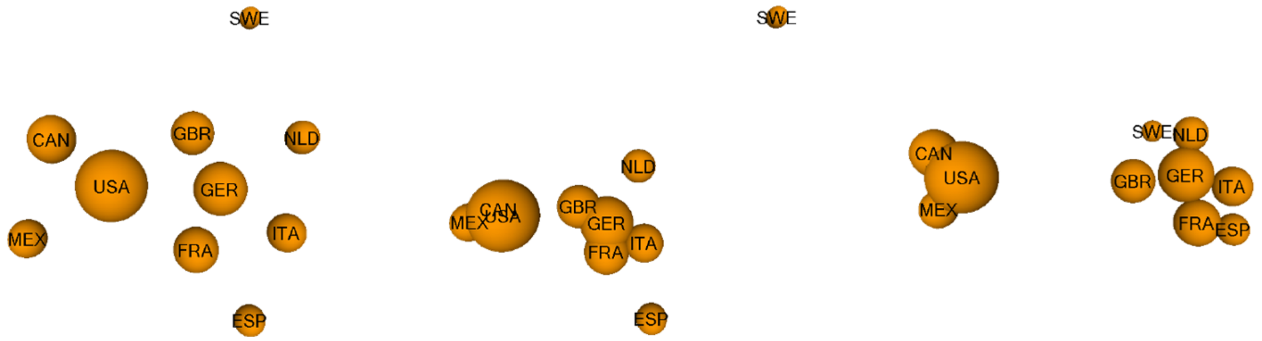
# Graph-clustering focused energy modelling

- Layout algorithms can be viewed as having two components – *energy model* and *energy minimization algorithm*
- **Up to now,** the focus of the energy models has been 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**

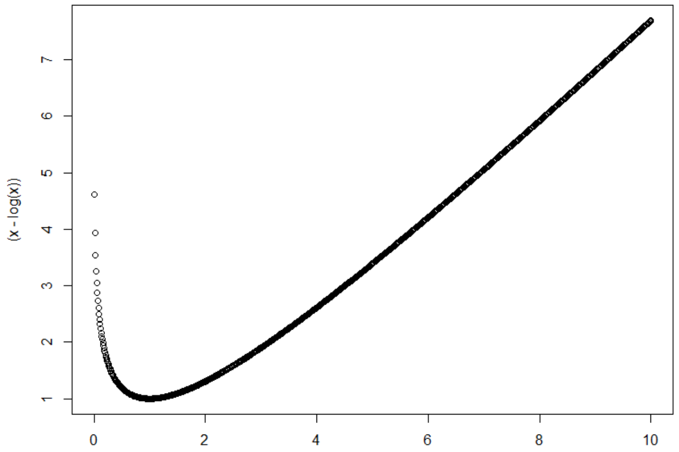
- **Not biased** towards grouping together **nodes with high degree** → appropriate for many real-world graphs with right-skewed degree distributions




(a) Fruchterman-Reingold (b) Node-repulsion LinLog (c) Edge-repulsion LinLog

source: Nock, A.: Energy Models for Graph Clustering . JGAA 11(2) (2007)

# 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

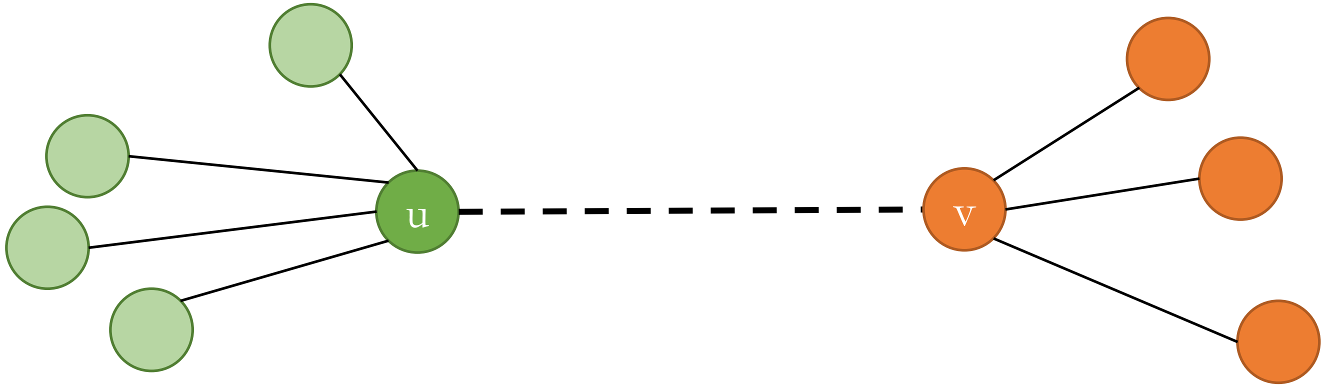
Repulsion forces

Nodes represented in terms of their neighboring nodes → edges

- **Edge-repulsion** model of a layout $p$

$$U_{Edge}(p) = \sum_{\{u,v\} \in E} \|p(u) - p(v)\| - \sum_{\{u,v\} \in V^2} \overset{5}{\deg(u)}\, \overset{4}{\deg(v)} \ln\|p(u) - p(v)\|$$

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

- $V_1, V_2 \subset V$: $\mathbf{cut}(\boldsymbol{V_1}, \boldsymbol{V_2}) = \left|\left\{\{v_1, v_2\} \in E \mid v_1 \in V_1, v_2 \in V_2\right\}\right|$
  - Expected cut is

Probability of an edge between two arbitrary nodes

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

#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**

$$\mathbf{ncut}(\boldsymbol{V_1}, \boldsymbol{V_2}) = \frac{\mathrm{cut}(V_1, V_2)}{|V_1||V_2|}$$

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

- **Edge-normalized cut**

$$\mathbf{ecut}(\boldsymbol{V_1}, \boldsymbol{V_2}) = \frac{\mathrm{cut}(V_1, V_2)}{\deg(V_1)\deg(V_2)}$$

# LinLog energy and clustering

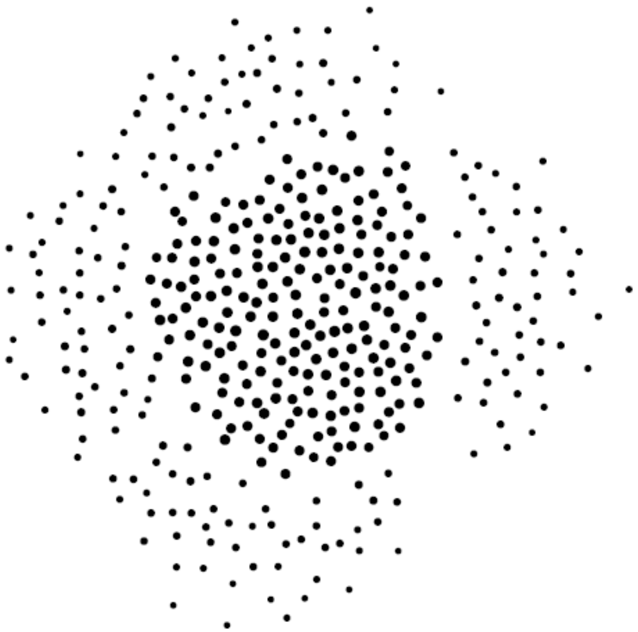
Each node $v$ can be imagined as an end point of the $\deg(v)$ connected edges

- 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 $1/\mathrm{ncut}(V_1, V_2)$ in minimal-energy node-repulsion LinLog layouts

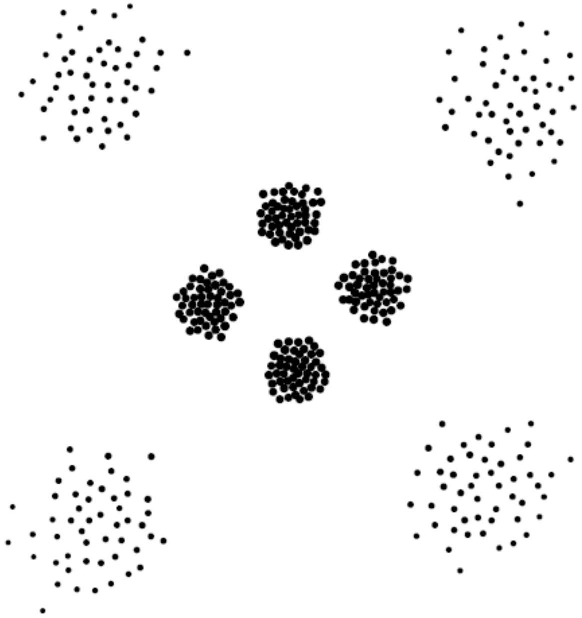
- Minimal-energy **edge-repulsion** layouts **minimize the ratio** of the **mean distance** between **connected *end* nodes** to the mean distance between **all *end* nodes**
- The distance of two dense and sparsely connected clusters $V_1, V_2$ approximates $1/\mathrm{ecut}(V_1, V_2)$ in minimal-energy edge-repulsion LinLog layouts

Pseudorandom graph with 400 nodes consisting of 8 equal-sized clusters:

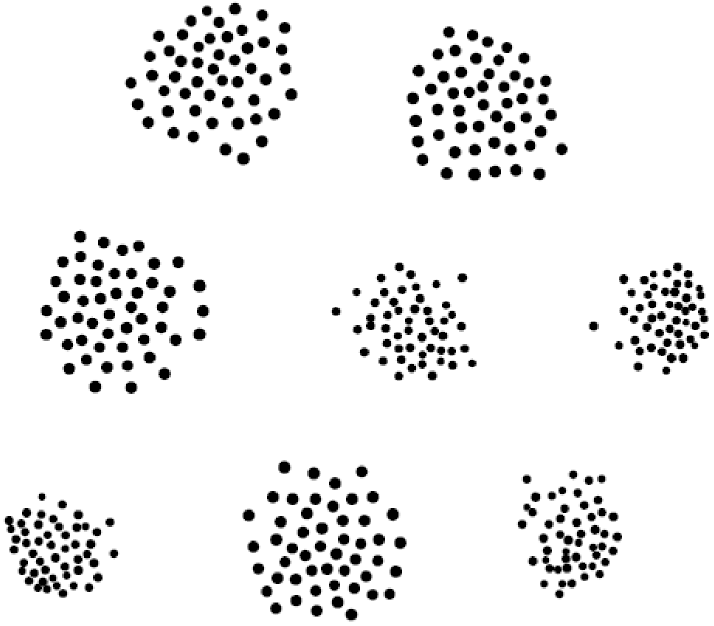
- Cluster 1-4 : intra-cluster edge probability = 1
- Cluster 5-8 : intra-cluster edge probability = 0.5
- Cluster 1-4 : inter-cluster edge probability = 0.2
- Cluster 5-8 : inter-cluster edge probability = 0.05
- Cluster 1-4 vs 5-8 : inter-cluster edge probability = 0.1



Fruchterman-Reingold



Node-repulsion LinLog



Edge-repulsion LinLog



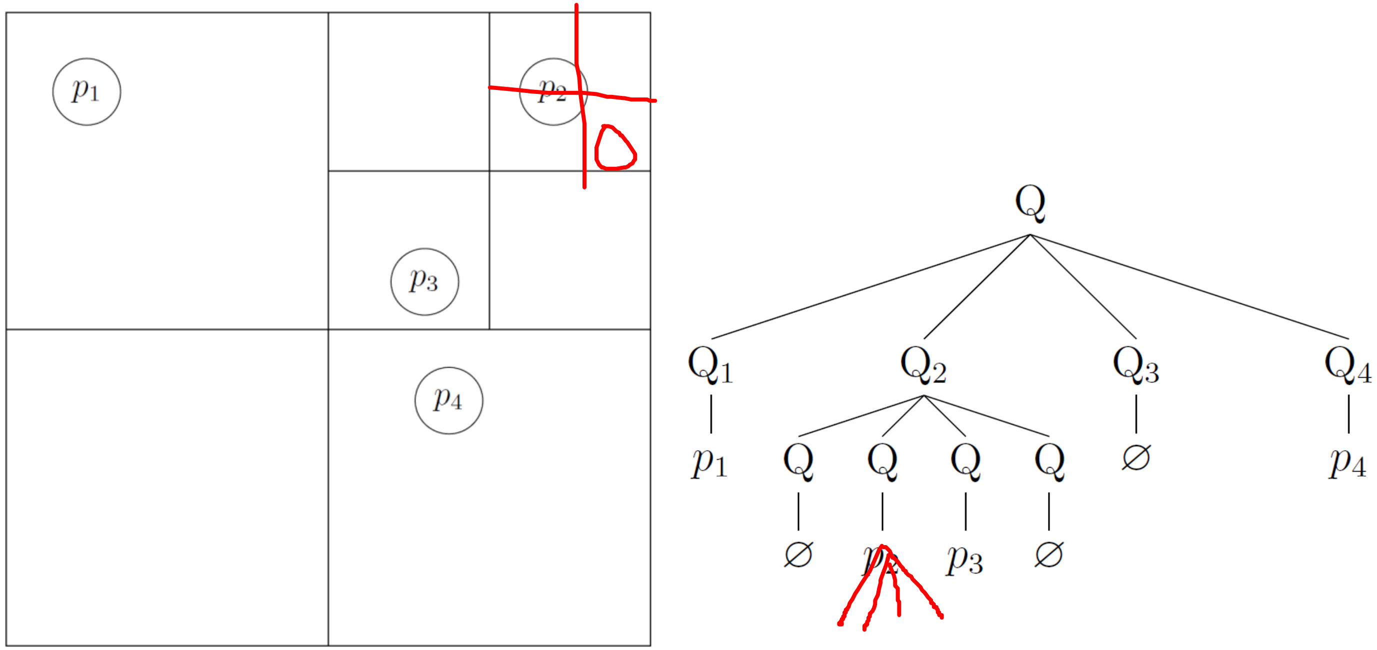
source: Nock, A.: Energy Models for Graph Clustering . JGAA 11(2) (2007)

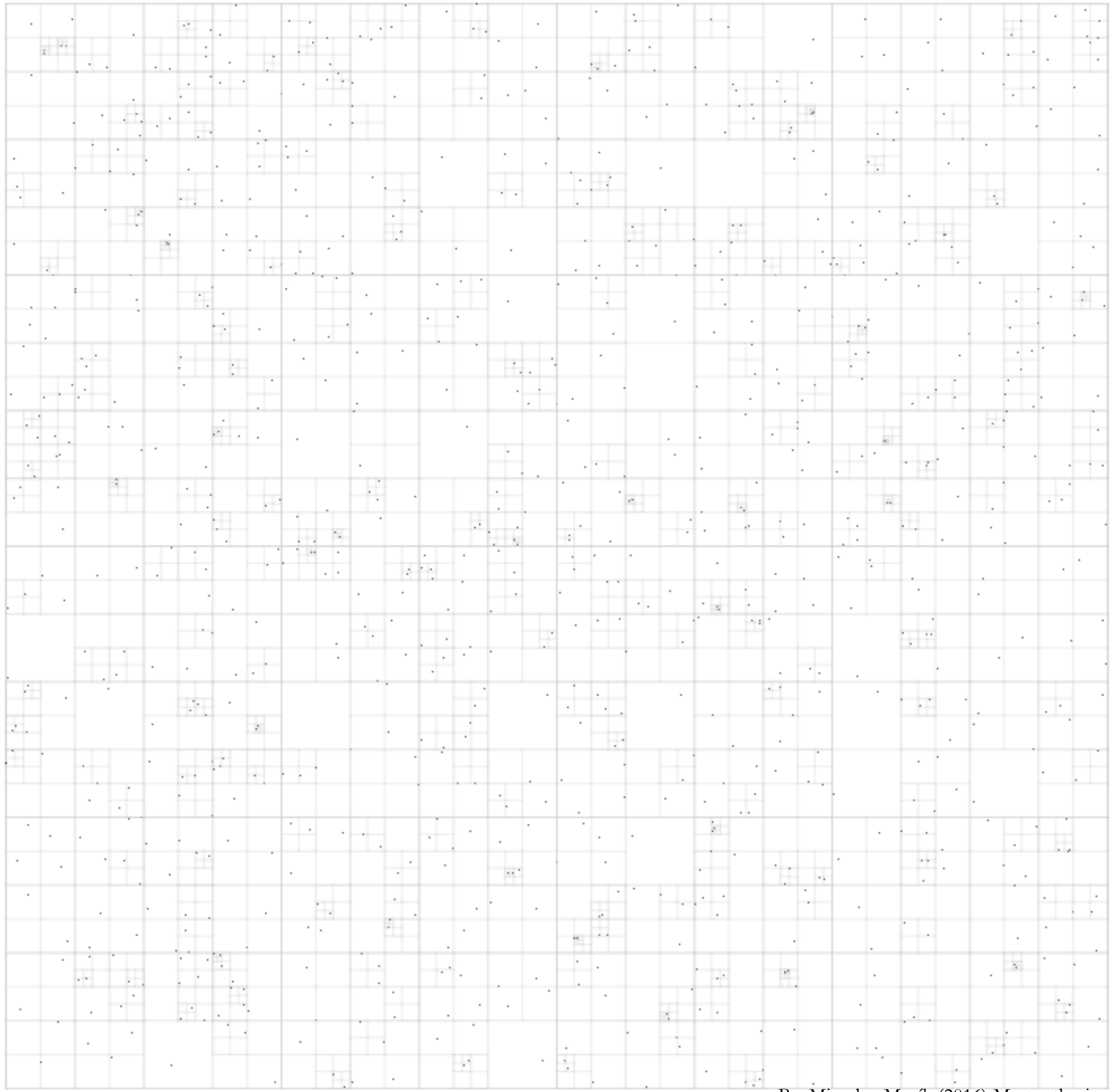
# Speed optimization of the FDP layouts

*Barnes, J., Hut, P. (1986). A hierarchical O(N log N) force-calculation algorithm. Nature, 324(6096), 446–449.*

# Barnes-Hut approximation (1)

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



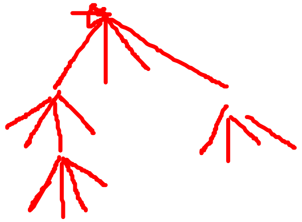



source: Bc. Miroslav Macík (2016) Master thesis

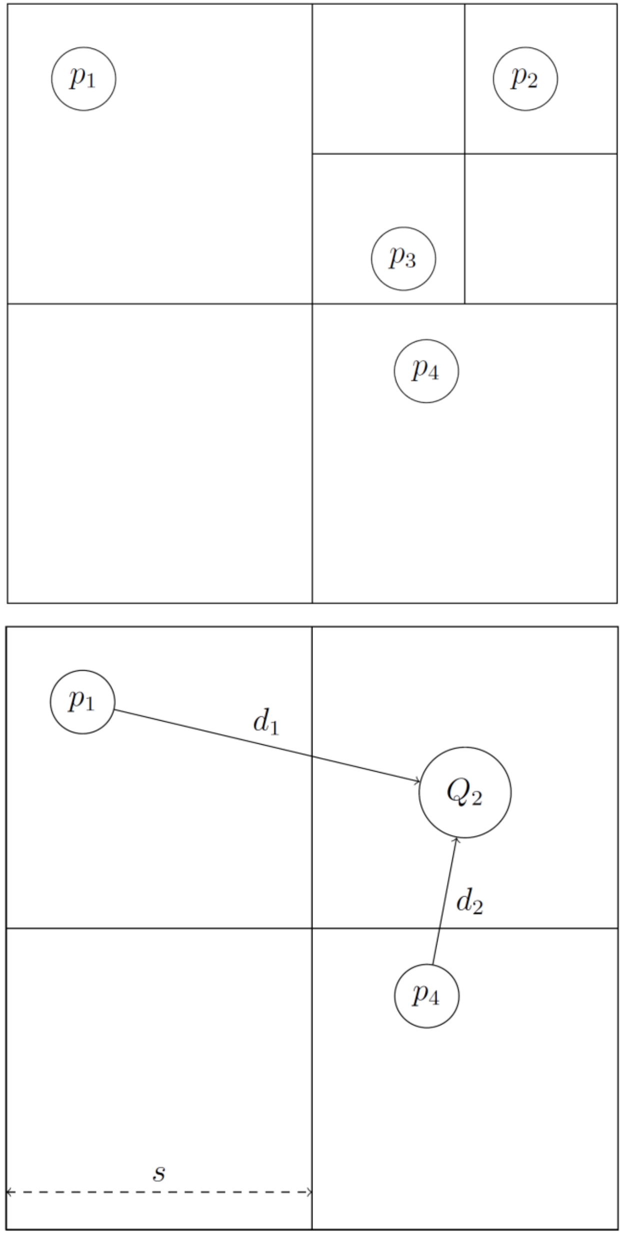
# Barnes-Hut approximation (2)

- 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**, the **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)

Center of gravitation

- **Repulsive forces** need to be **modified**

$$f_r(x_i, x_j) = -\frac{k^2}{d(x_i, x_j)} \to f_r(x_i, x_Q) = -\frac{|Q|k^2}{d(x_i, x_Q)} \ldots \; x_Q = \frac{\sum_{j \in Q} x_j}{|Q|}$$

- 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

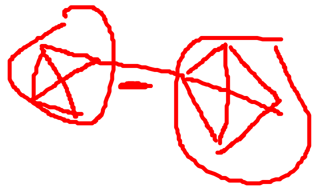
- Many of the real-world networks have so-called **small world networks (SWN)** property
  - Average path length in a SWN compares to a path length in a random graph
  - Clustering indexes of SWN nodes are on average orders of magnitudes larger

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

Number of vertices in the neighborhood of $v$

How close neighbors of $v$ resemble a clique.

| Graph | Clustering index | Ave Path Length | Random graph (same number of nodes and edges) | |
|---|---|---|---|---|
| IMDB | 0.9666 | 3.2043 | 0.0243 | 2.6694 |
| "Resyn assistant" | 0.9518 | 3.2847 | 0.1942 | 1.8195 |
| Mac OS 9 | 0.3875 | 2.8608 | 0.0179 | 3.3196 |
| Web[2] | 0.1078 | 3.1 | 2.3e-4 | - |
| .edu sites[2] | 0.156 | 4.062 | 0.0012 | 4.048 |

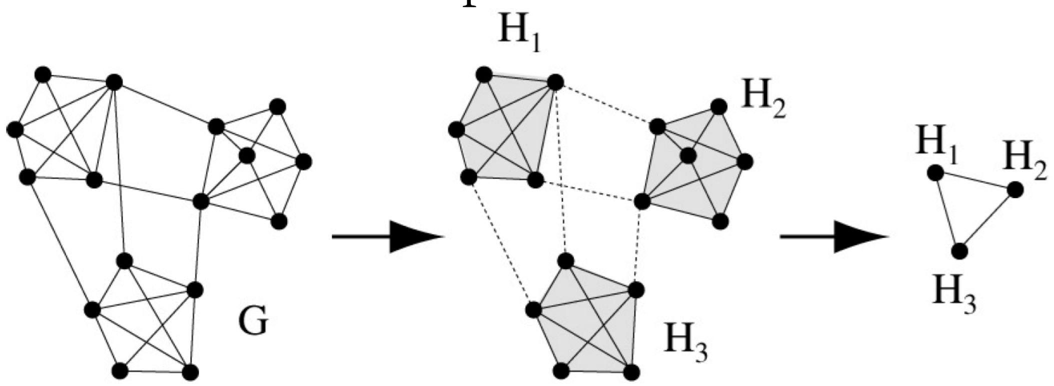




source: Auber et. al.: Multiscale Visualization of Small World Networks. INFOVIS'03 (2003)

# 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**

source: Stephen G. Kobourov: Spring Embedders and Force Directed Graph Drawing Algorithms. Handbook of Graph Drawing and Visualization (2013)

**Input**: graph $G = (V, E)$
**Output**: Find $L$, a nice layout of $G$
$Iterations = 4$ ; `/* number of iterations in local beautification */`
$Ratio = 3$ ; `/* ratio of vertex sets in consecutive levels */`
$Rad = 7$ ; `/* radius of local neighborhood */`
$MinSize = 10$ ; `/* size of coarsest graph */`
Compute all-pairs shortest path lengths $d_{i,j}$;
Initialize layout $L$ by placing vertices at random;
$k \leftarrow MinSize$;
**while** $k \leq |V|$ **do**
- $C \leftarrow$ **K-Centers**$(G(V, E), k)$;
- $radius = \max_{v \in C} \min_{u \in C} \{d_{vu}\} * Rad$;
- **LocalLayout**$(d_{C \times C}, L(C), radius, Iterations)$;
- **foreach** $v \in V$ **do**
  - $L(v) \in L(center(v)) + rand$;
- $k \leftarrow kRatio$

**return** $L$;

Closest center to $v$

Assign

**K-Centers**$(G(V, E), k)$;
**Input**: Graph $G = (V, E)$ and constant $k$
**Output**: Compute set $S \subseteq V$ of size $k$, s.t. $\max_{v \in V} \min_{s \in S} \{d_{sv}\}$ 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** $Iterations * |V|$ **do**
- Choose the vertex $v$ with the maximal $\Delta_v^k$;
- Compute $\delta_v^k$ as in Kamada-Kawai;
- $L(v) \leftarrow L(v) + (\delta_v^k(x), \delta_v^k(y))$;

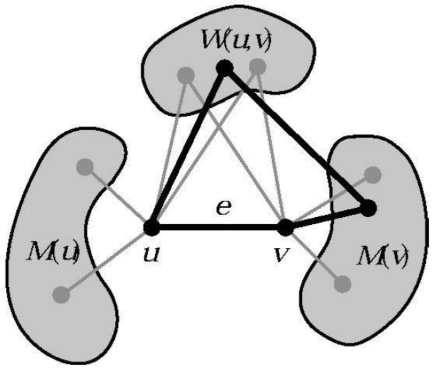
# Multi-scale algorithms - Auber

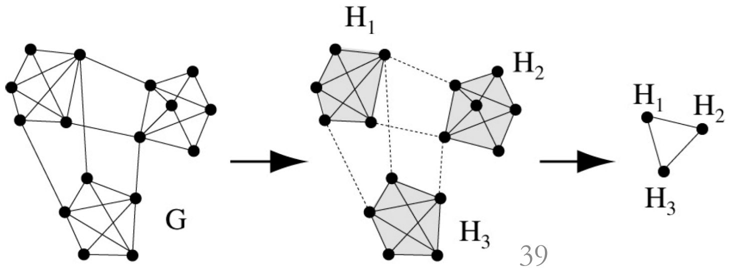
- Iteratively **filter out edges** with low edge clustering index → remaining **connected components** form nodes of the **quotient graph**

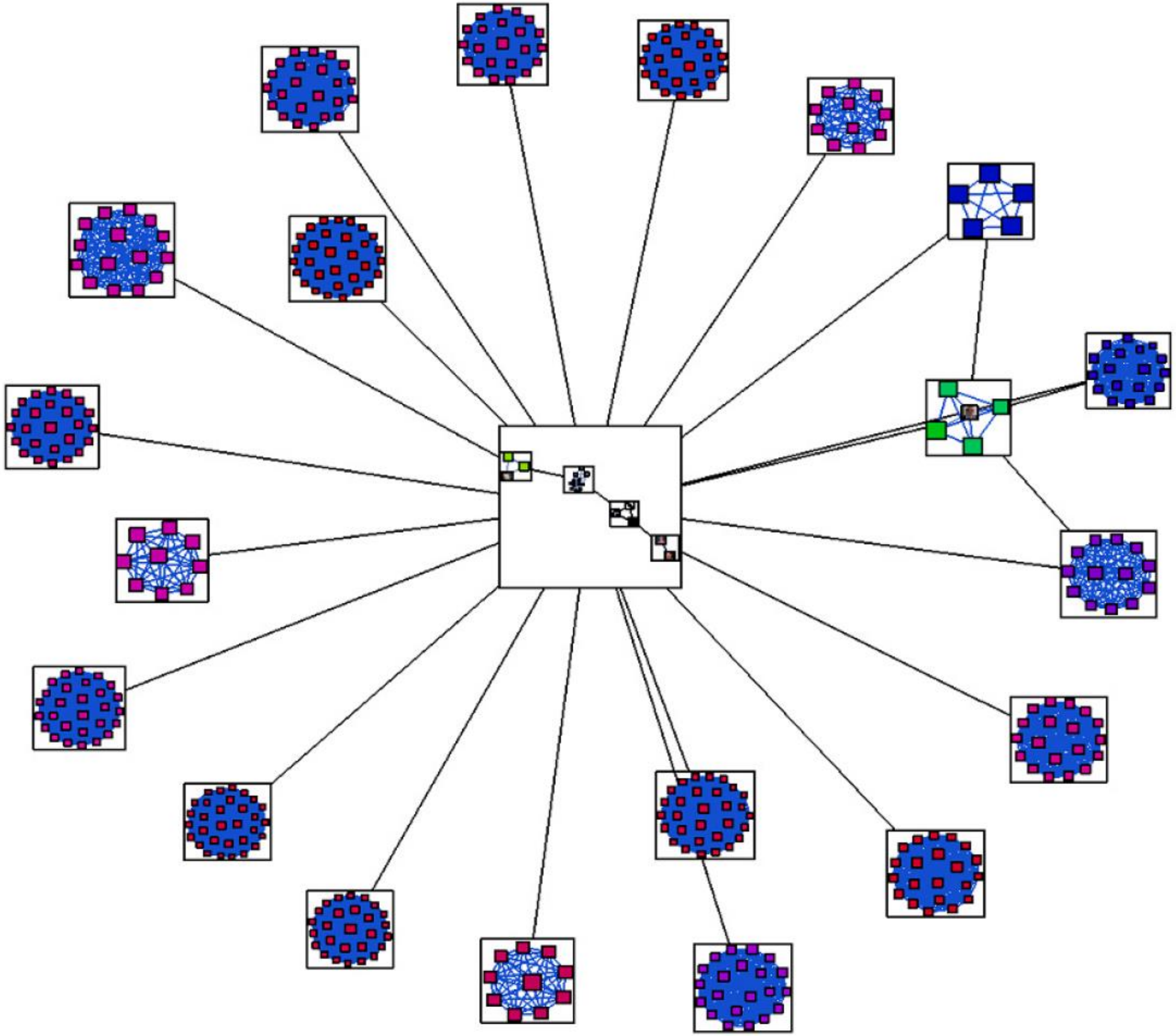
$$s(A,B) = cut(A,B)/|A||B|$$

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

$$\begin{aligned} &strength(u,v) \\ &= s\big(M(u), W(u,v)\big) + s\big(W(u,v), M(v)\big) + s\big(M(u), M(v)\big) \\ &+ s\big(W(u,v)\big) + \underbrace{|W(u,v)|/|M(u) + W(u,v) + M(v)|}_{\text{Ratio of 3-cycles using } \{u,v\}} \end{aligned}$$



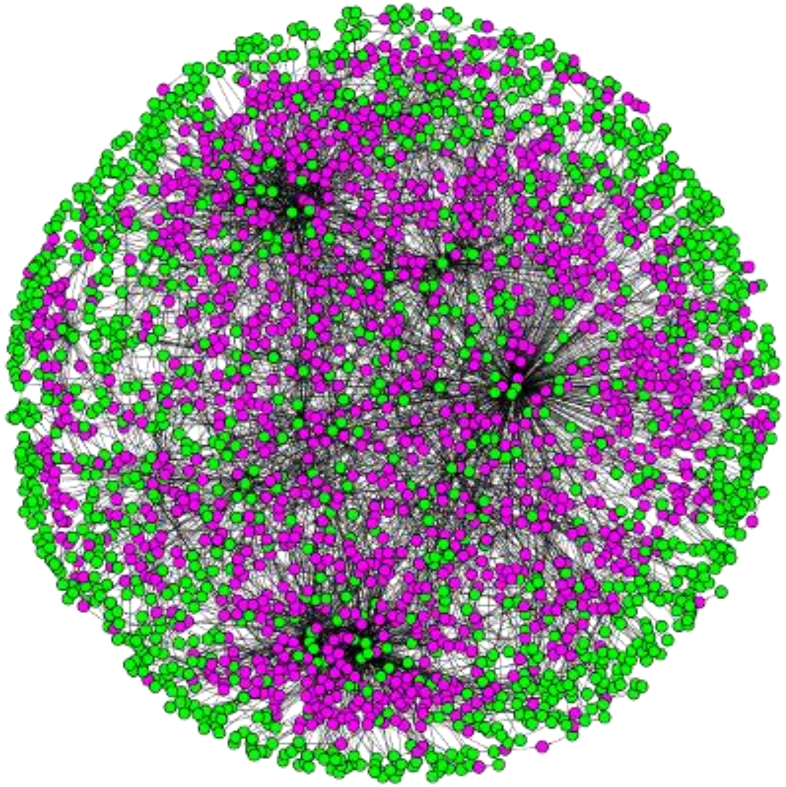


# Very complex or specific networks

# Hairballs

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



E. coli metabolic network visualized using Cytoscape

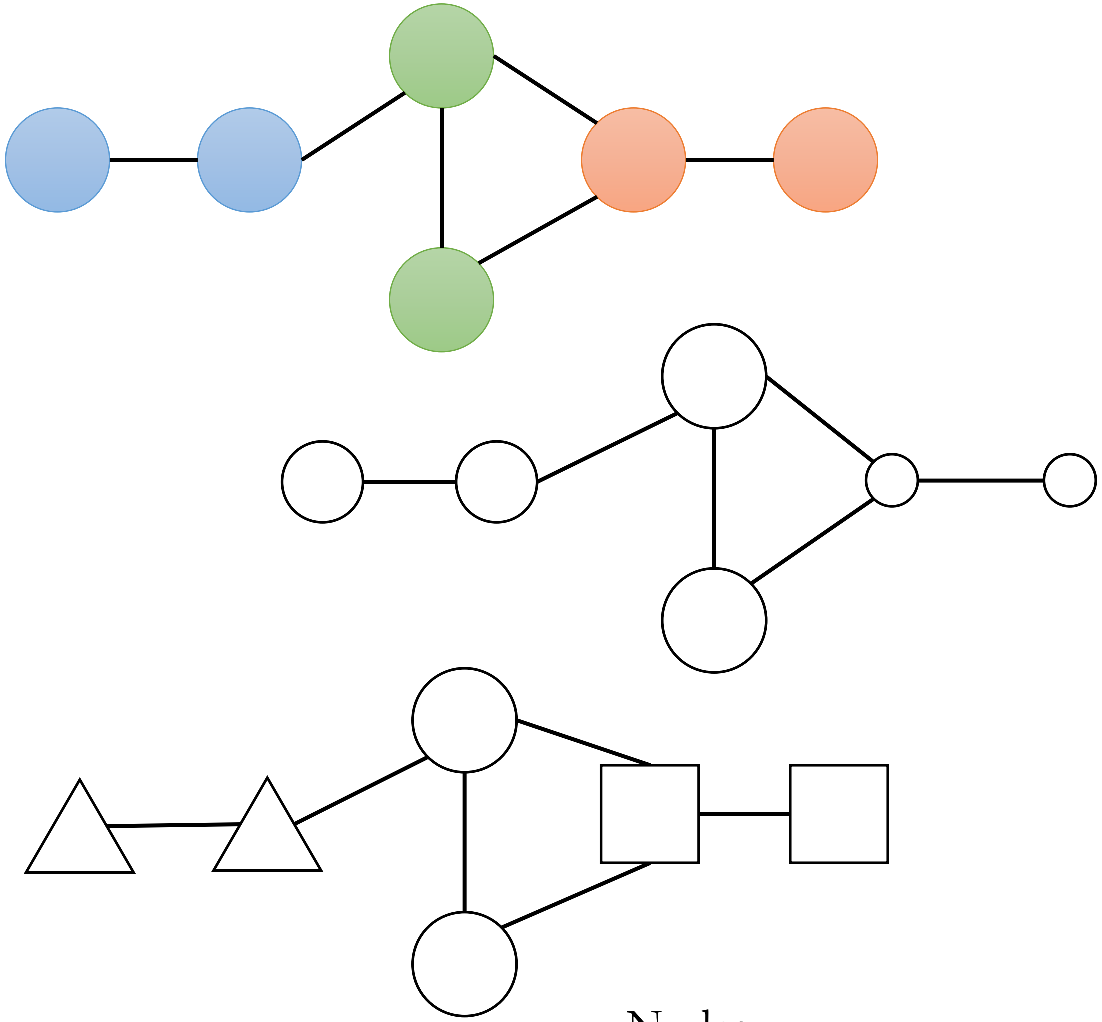
# Interpretability of graph layouts

- **Interface problem**

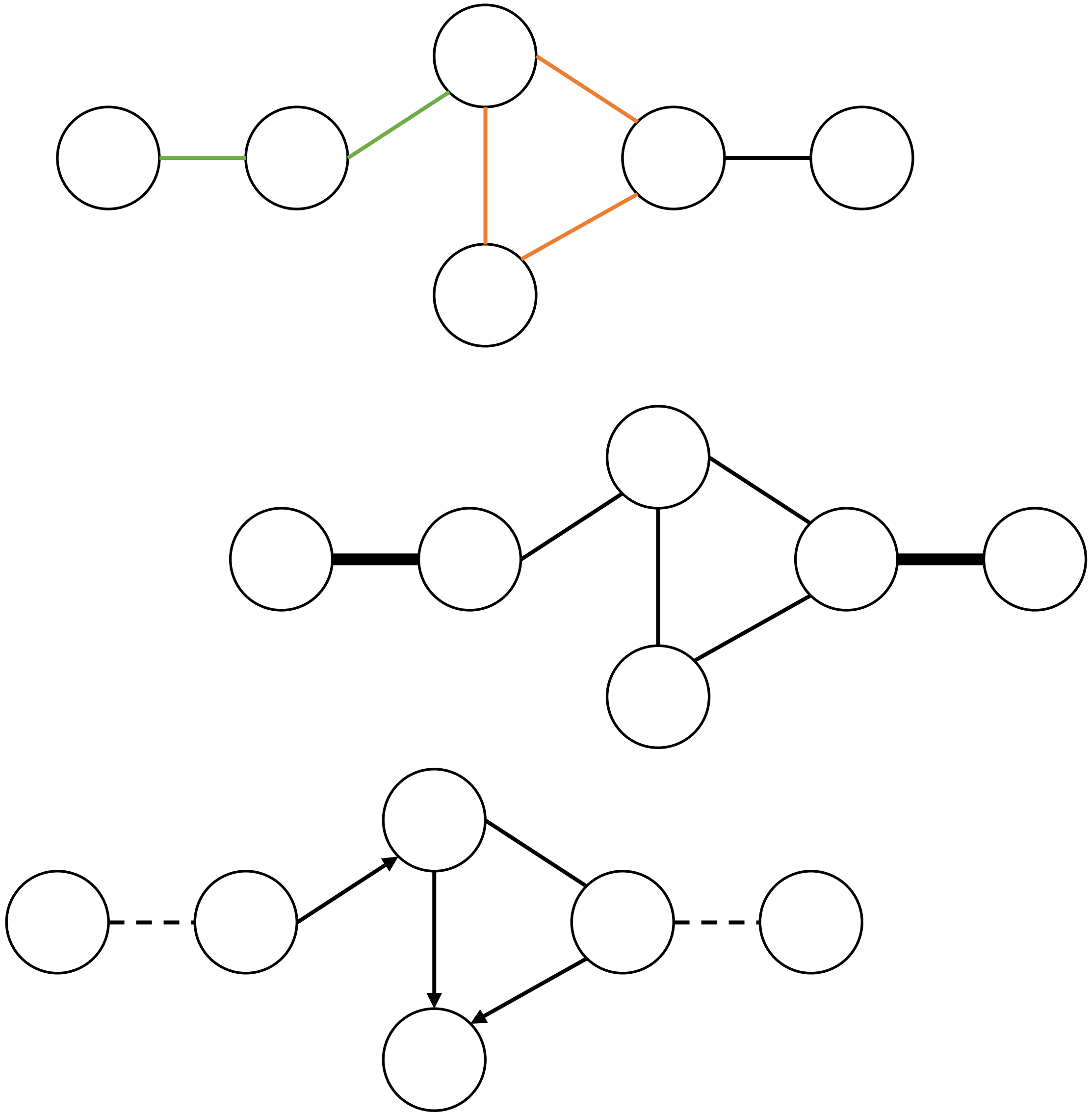
  - Graph **sizes** tend to **grow** while the **size of the medium** we use to visualize them tends to stay **the same**

  - **Solutions**
    - **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



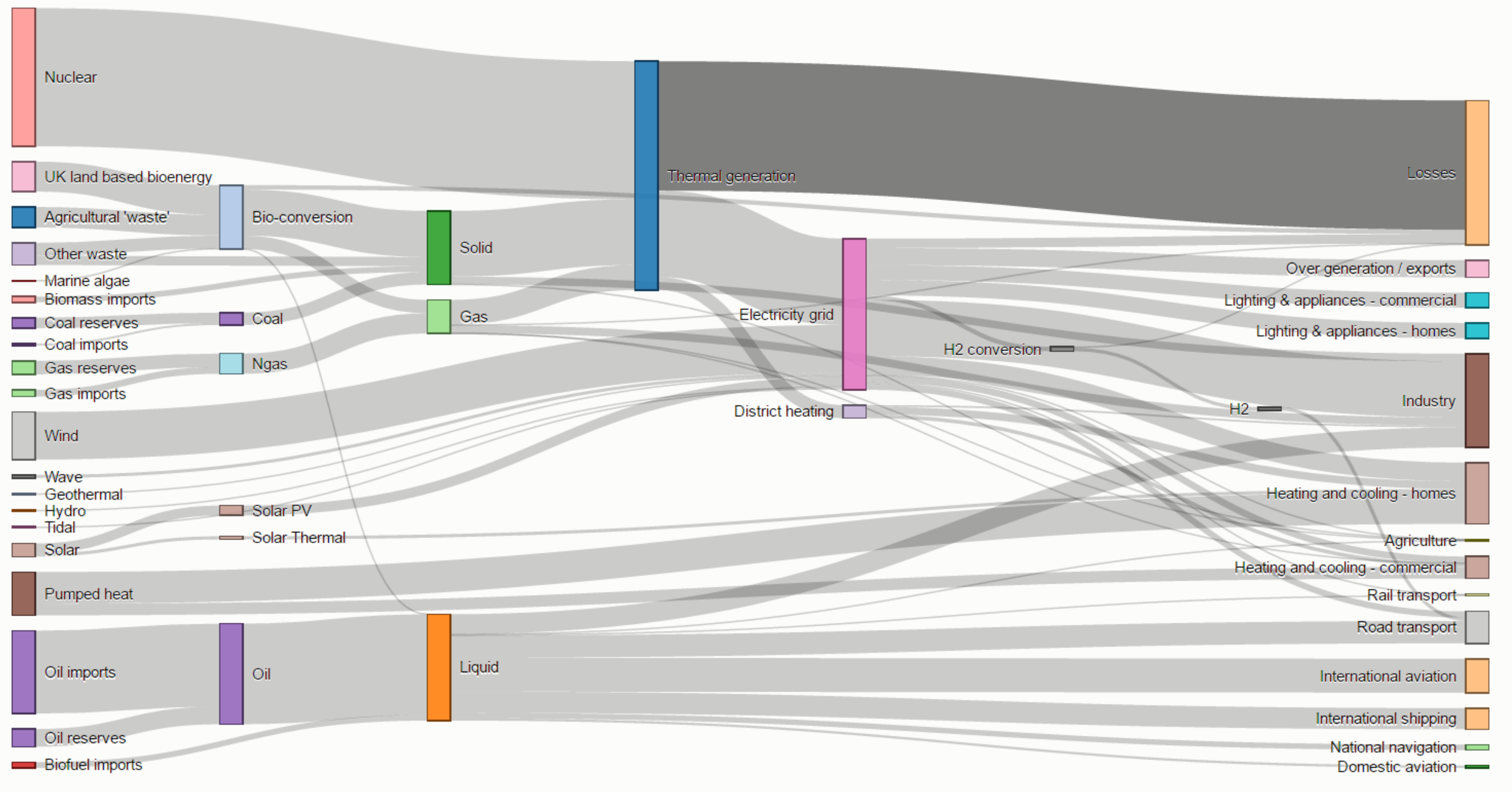
Nodes



Edges

# Flow diagrams

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





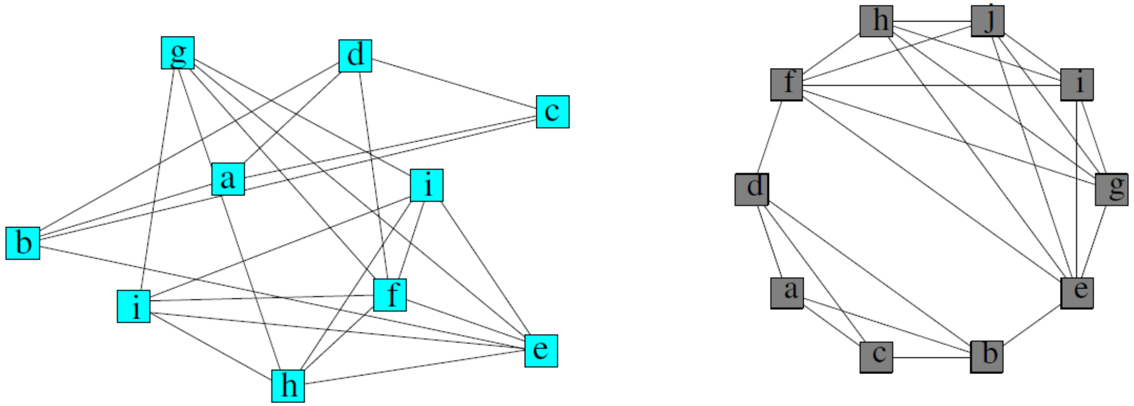

https://bost.ocks.org/mike/sankey/

# Circular layout

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

**Basic layout**

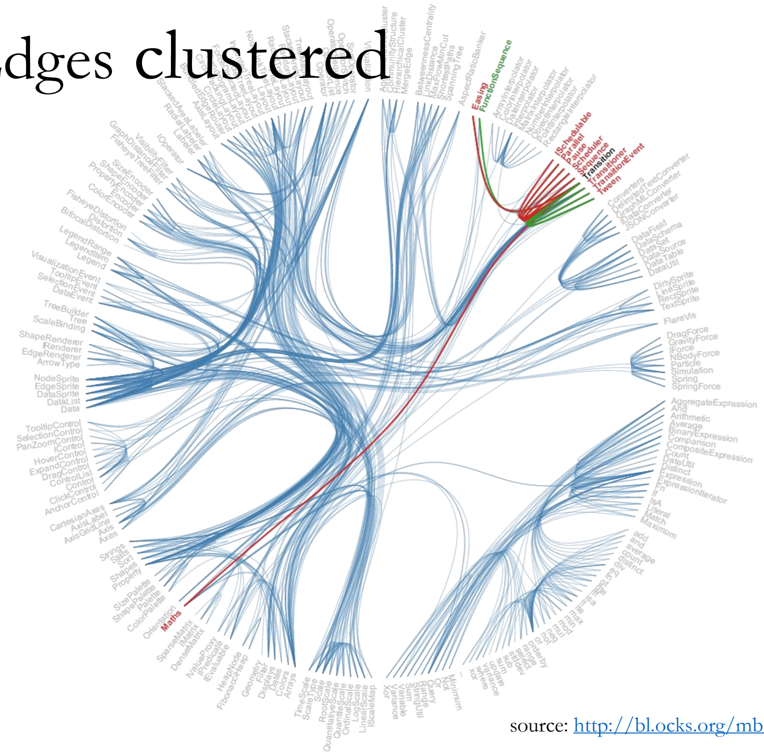
- Edges drawn as straight lines




source: Six and Tollis: Circular Drawing Algorithms. Handbook of Graph Drawing and Visualization (2013)

**Edge binding**

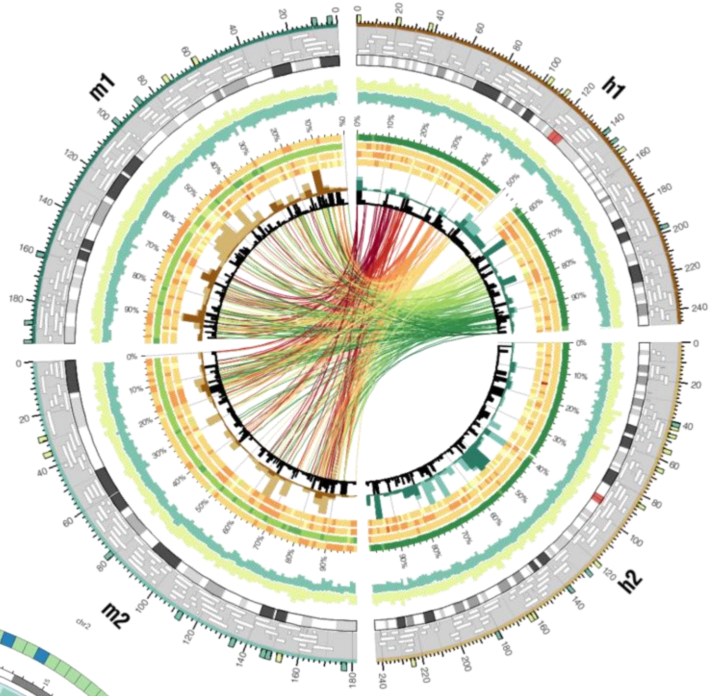
- Edges clustered

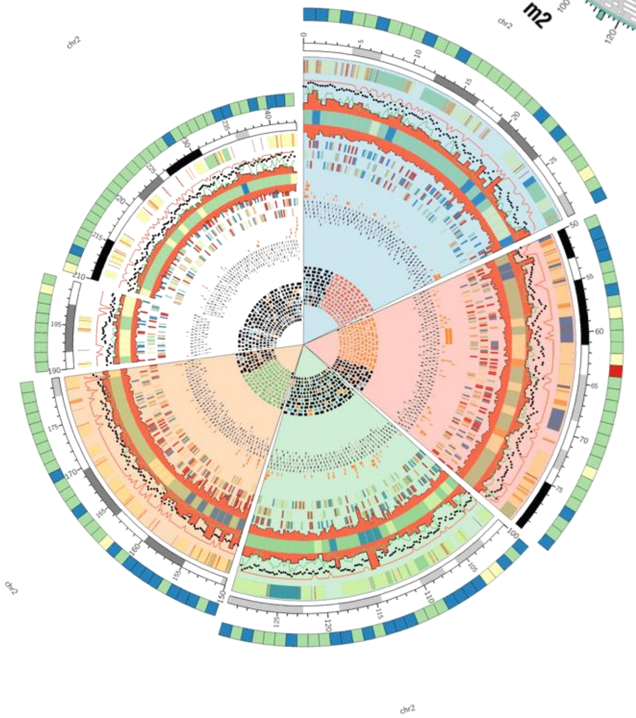


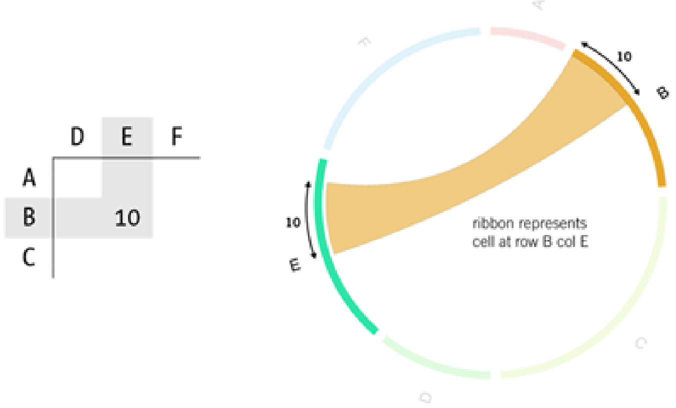
source: http://bl.ocks.org/mbostock/7607999

# Circos

- Software for laying out **relationships** (graphs)
- Basically **converts tabular data to a circular** layout (any graph can be expressed as a table)

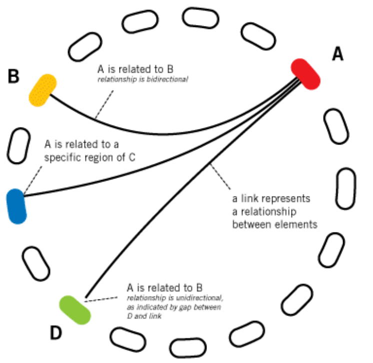




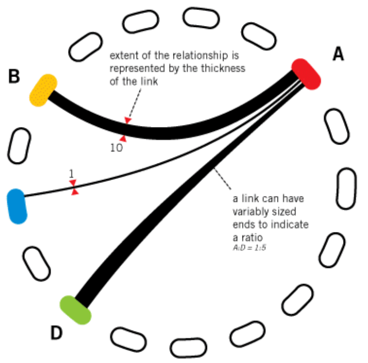
# Applying Circos to
# Visualizing Tables using a Circular Layout

## /1



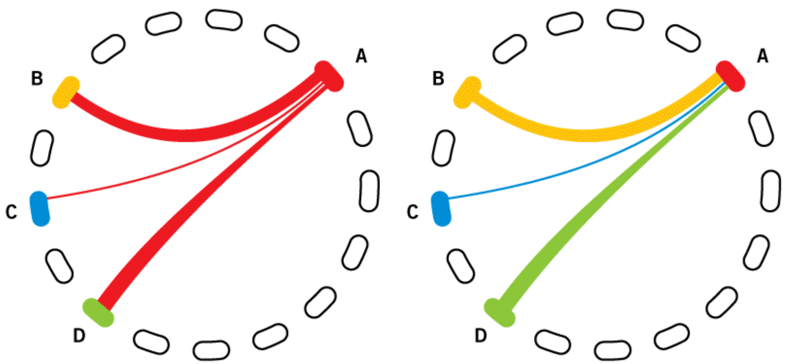

Relationships between elements in your data set are indicated by links. Links can indicate a simple relationship (A-B), a relationship that has positional information (A-C), or a unidirectional relationship (A-D). In each case, the link is formatted differently.

## /2



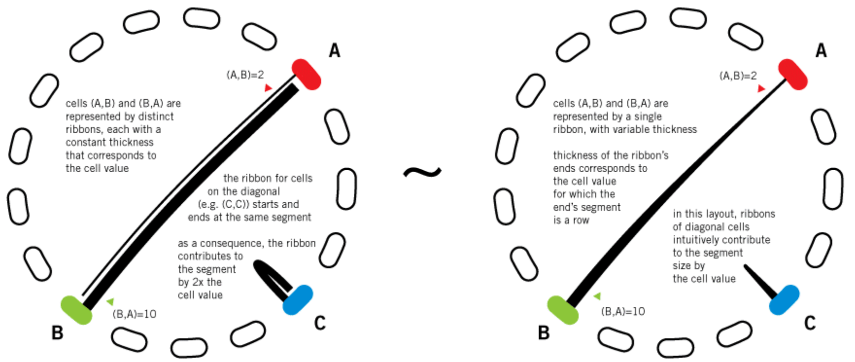

Links with variable thickness can represent the extent or magnitude of the relationship between elements.

## /3




▲ 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.

## /4



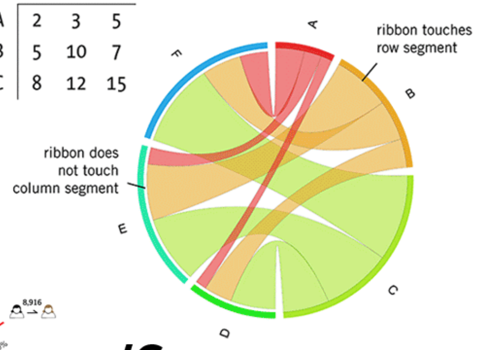

Transpositive cells (e.g. (A,B) and (B,A)) are here shown by a single ribbon (right). The ribbon now ends directly at both row and column segments and its ends are of variable thickness, which is the cell value for which the end's segment is a row. For example, if (A,B)=2 and (B,A)=10 then the ribbon's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

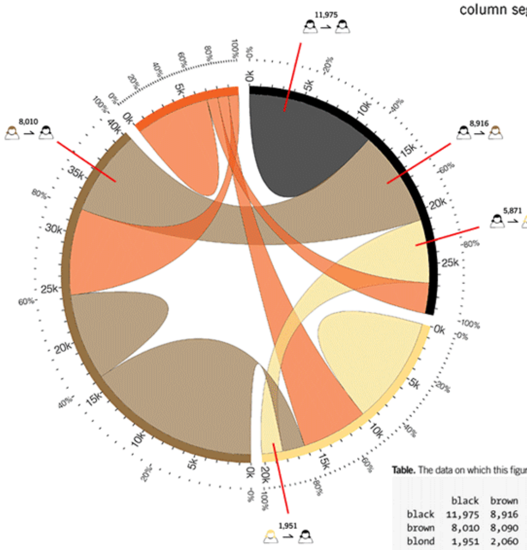
*Table visualization is implemented in the tableviewer utility, bundled with Circos. The parse-table script parses a tabular data file into an intermediate format, which is then used by make-conf to create Circos configuration and data files. CIrcos is then used to generate the final image.*

## /5

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

| | D | E | F |
|---|---|---|---|
| A | 2 | 3 | 5 |
| B | 5 | 10 | 7 |
| C | 8 | 12 | 15 |



## /6

◀ 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).

**Table.** The data on which this figure is based.

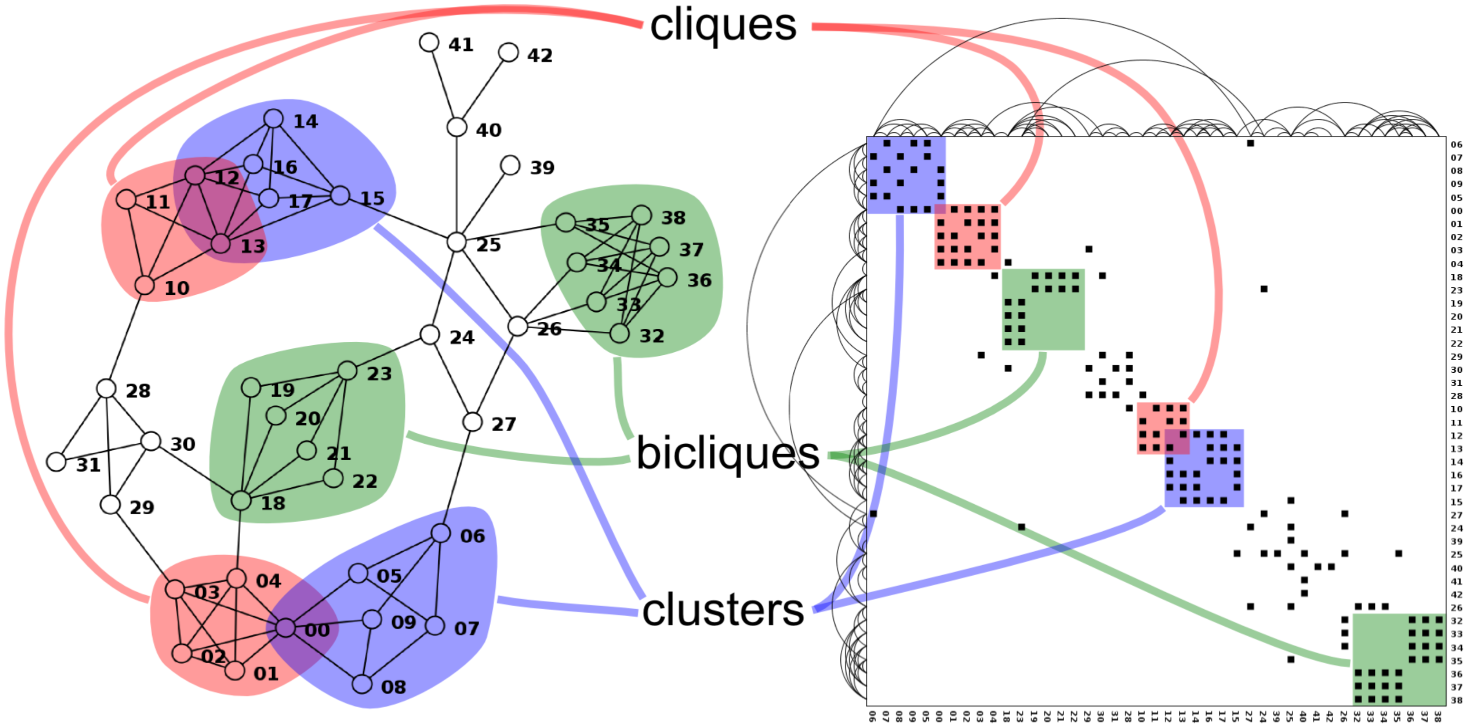
| | black | brown | blond | red |
|---|---|---|---|---|
| black | 11,975 | 8,916 | 5,871 | 2,868 |
| brown | 8,010 | 8,090 | 16,145 | 8,045 |
| blond | 1,951 | 2,060 | 10,048 | 6,171 |
| red | 1,013 | 940 | 990 | 6,907 |



source: http://circos.ca/guide/tables/

# Matrix methods

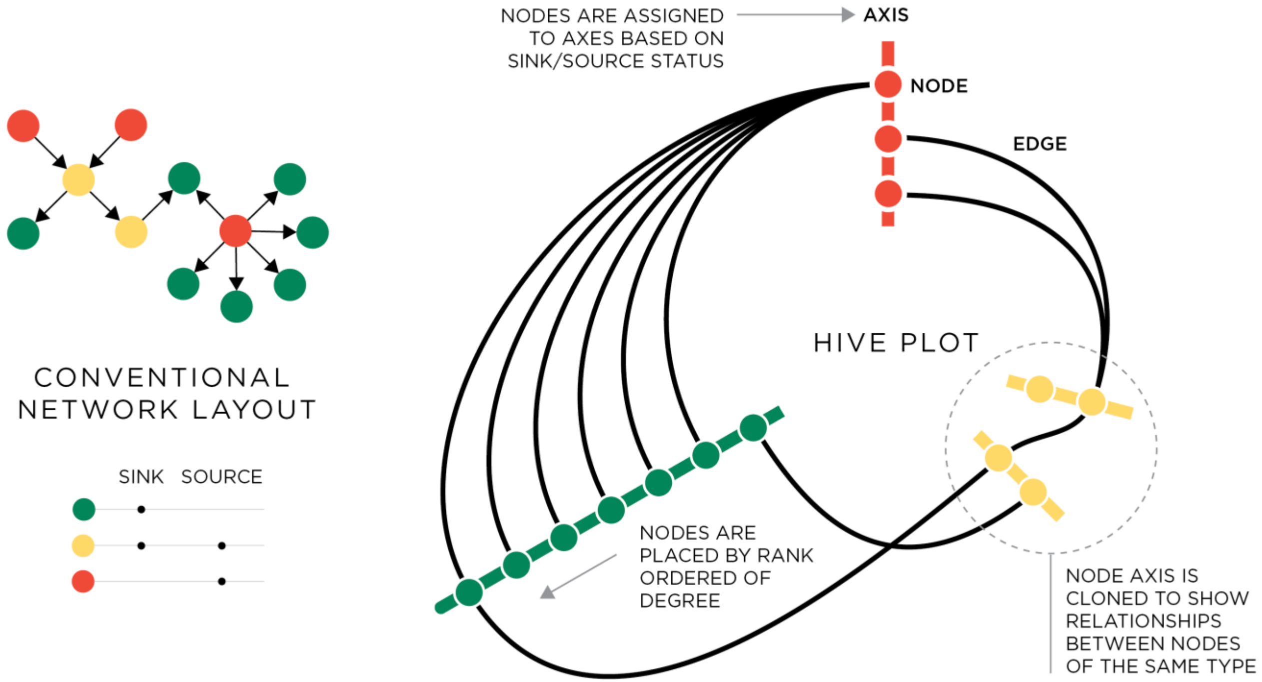
- Methods based on **analysis of the adjacency matrix**
- Requires some **support from** the visualization **tool**
  - Rows and columns of the matrix can be rearranged → patterns




source: McGuffin. Simple Algorithms for Network Visualization: A Tutorial (2012)

# Hive plots

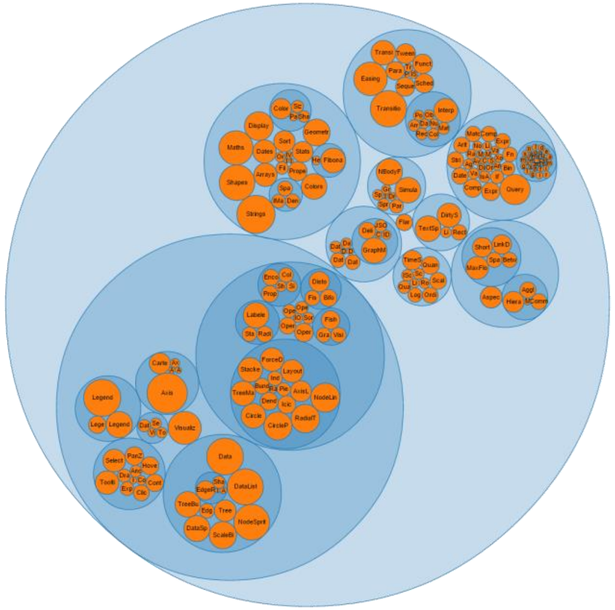
- **Nodes** mapped to and positioned **on radially distributed linear axes** → linear layout of nodes
  - Can be divided into **segments**
- **Edges** drawn as **curved links**
- Graph structure can be mapped to
  - Axis, Position, Color



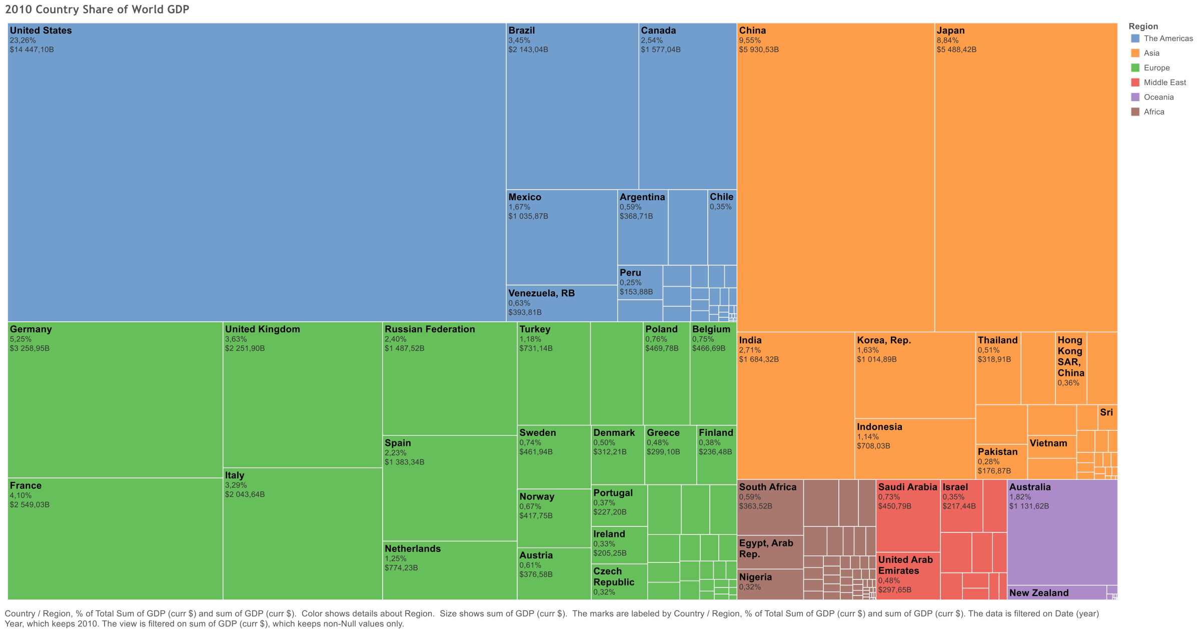

source: http://www.hiveplot.net/conference/vizbi2011/poster/krzywinski-hiveplot-poster.png

# Hierarchical layouts

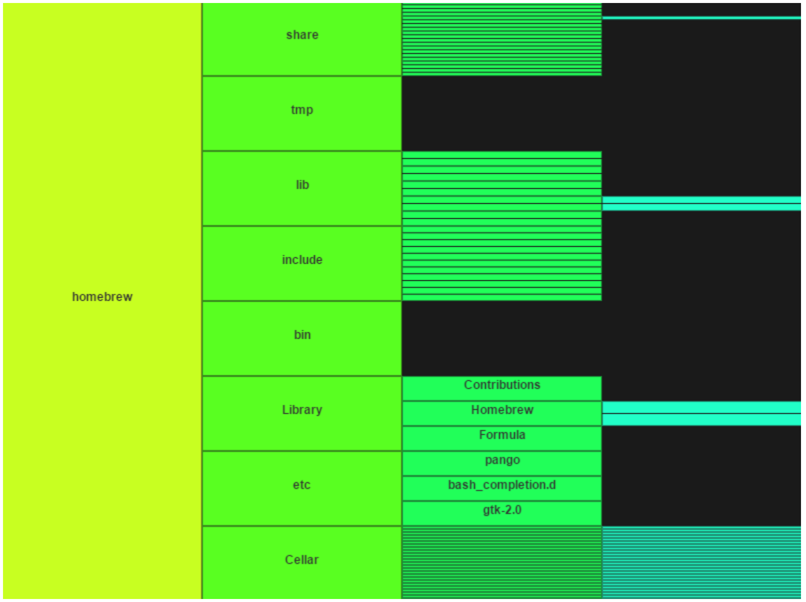
Sunburst



Circle packing




Tree map




Icicle

# Interactivity

# Zooming & panning

- Not always is the interface problem solvable with the layout algorithm → 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

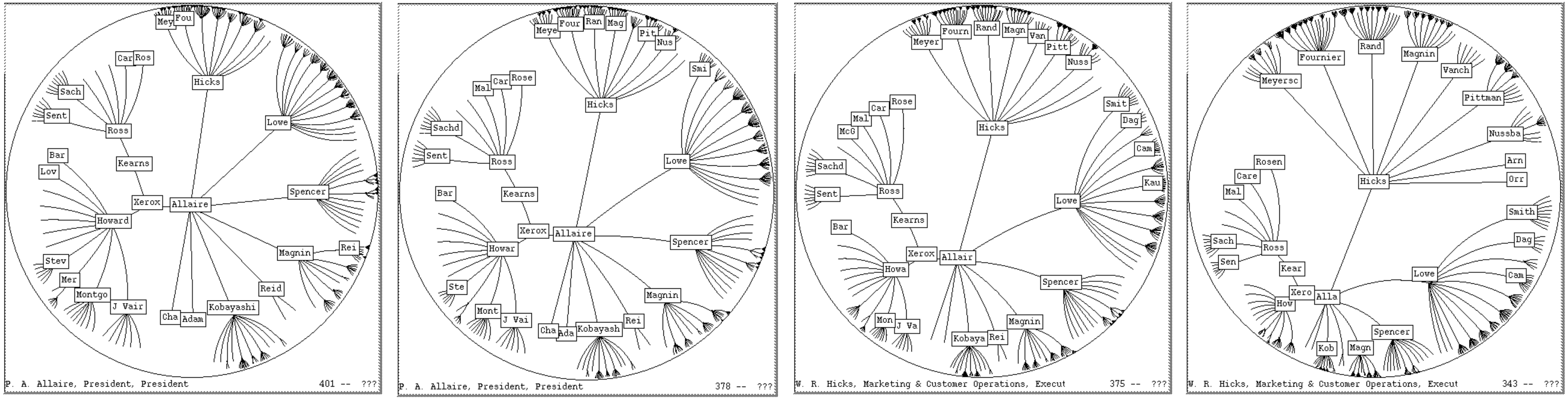
# Focus and context

- Zooming suffers from **loosing context** – when zoomed in, all context is lost → difficult to pan → **decreased usability**
- A technique **combining** both **overview** (context) and **detail** information (focus) in one view would be desirable → **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**




source: Lamping et al. A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. ACM (1995)

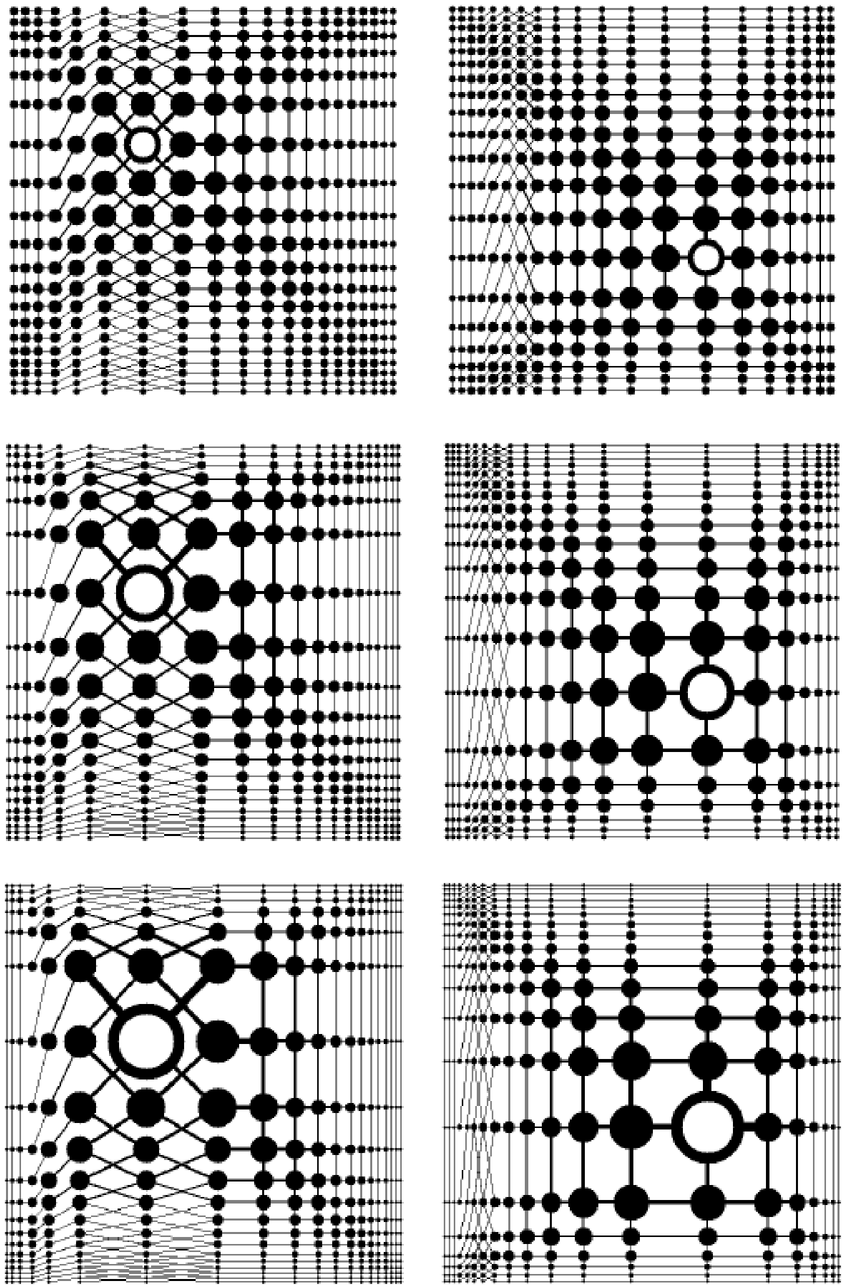
# Fisheye (2)

- 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

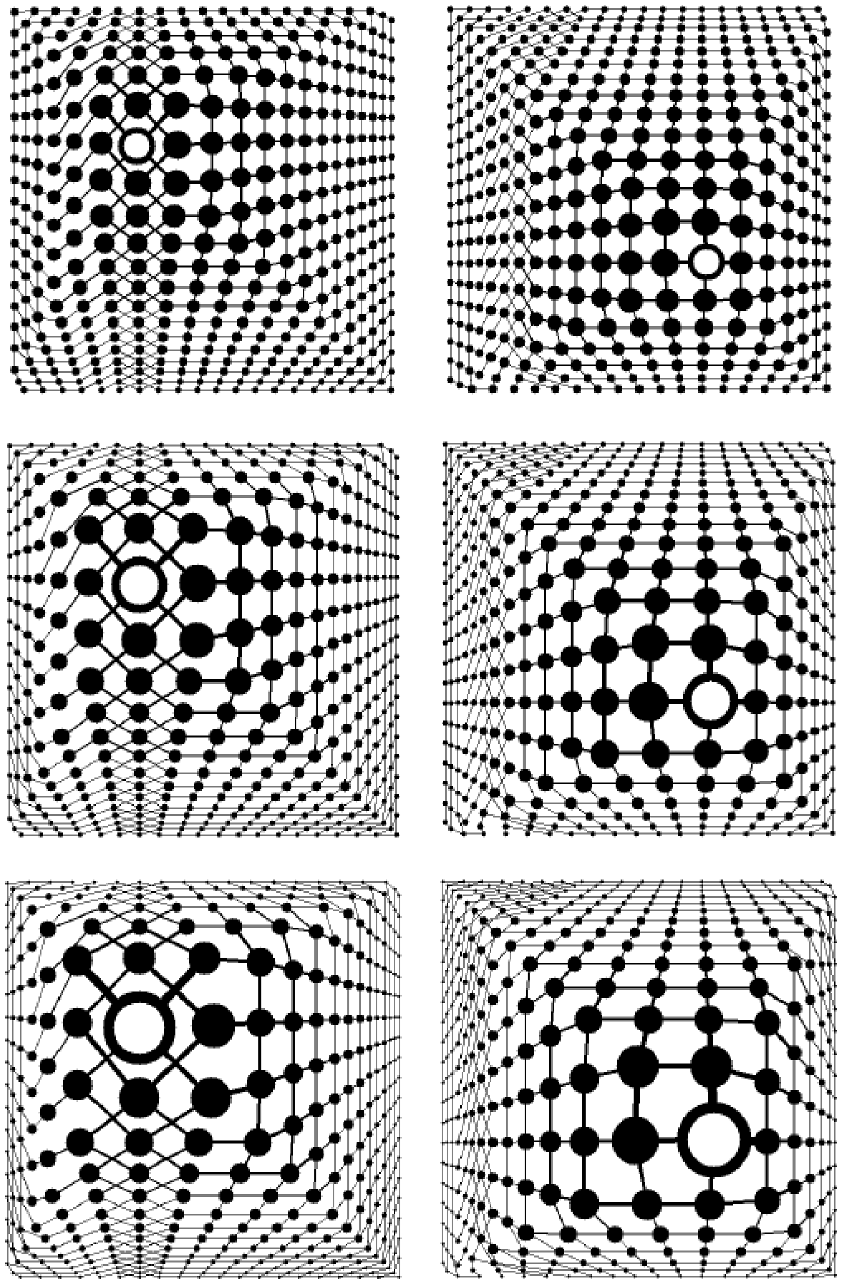
[https://bost.ocks.org/mike/fisheye/](https://bost.ocks.org/mike/fisheye/)

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

## Cartesian distortion



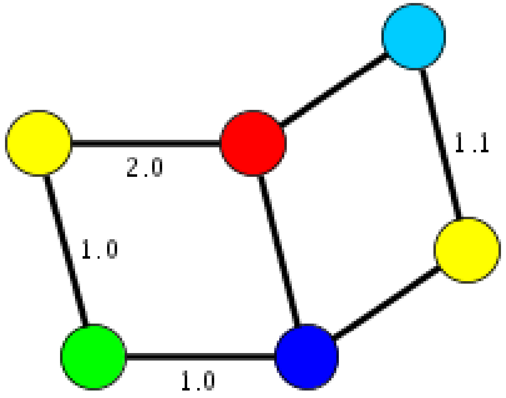
## Polar distortion



# Tools and data

# Graph languages - GraphML

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

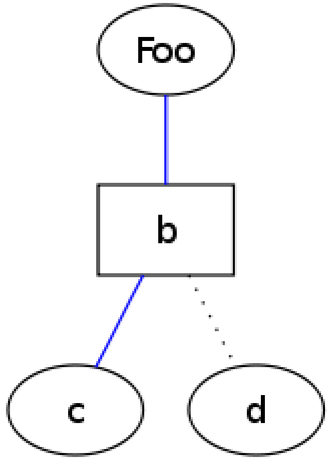



```xml
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
        http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <key id="d0" 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

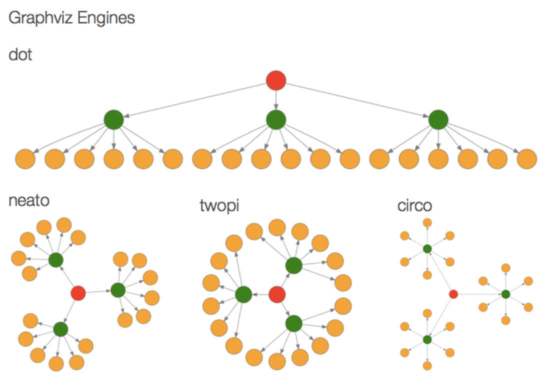
- Plain text graph description language
- Supported by GraphViz, Gephi, ..

```
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];
 }
```

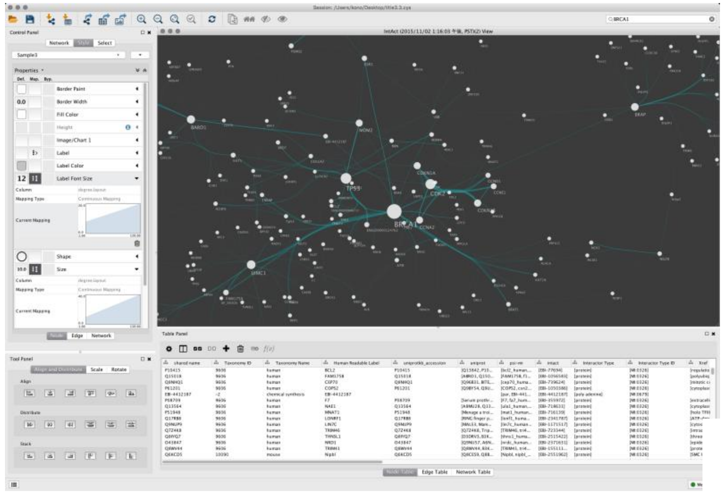


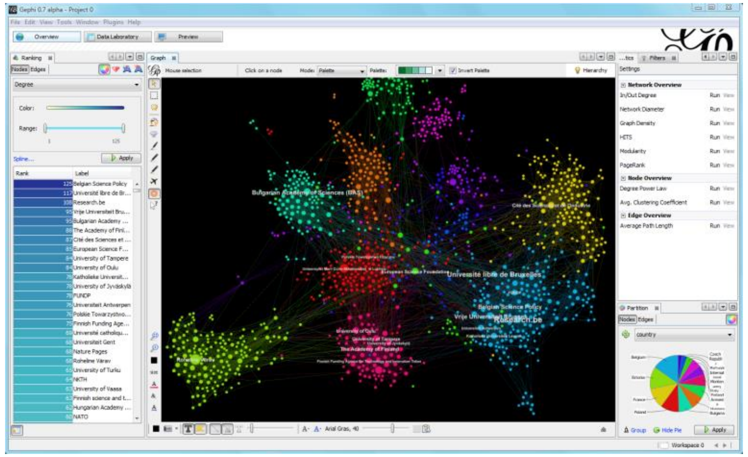
# Software

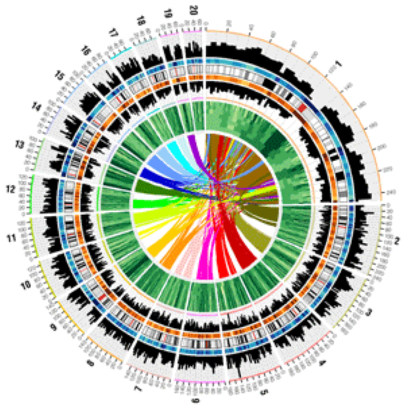
- [GraphViz]()
- [Gephi]()
- [Cytoscape]()
  - Cytoscape.js
- [Circos]()



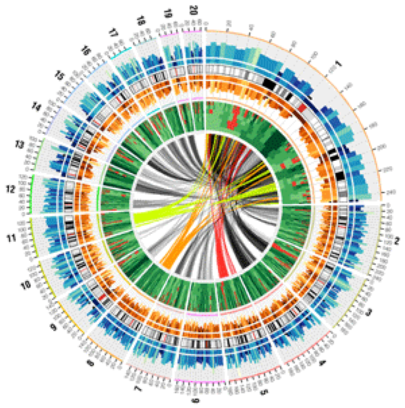

source: http://rich-iannone.github.io/DiagrammeR/graphviz_and_mermaid.html







RULES OFF



RULES ON

# 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