Arc Diagrams and Barycenter Ordering

The linear arrangement of nodes in an arc diagram has many advantages.

As already mentioned, there is room to the right of each node for a long text label, if desired. The space to the right of nodes can also be used to display small graphics, such as line charts for each node, possibly to show a quantity associated with the node that evolves with time.

Arc diagrams can also be incorporated as an axis within a larger graphic or visualization

Also, as mentioned, the nodes within an arc diagram can be sorted in different ways, which can be useful for seeing relationships between nodes with specific attribute values.

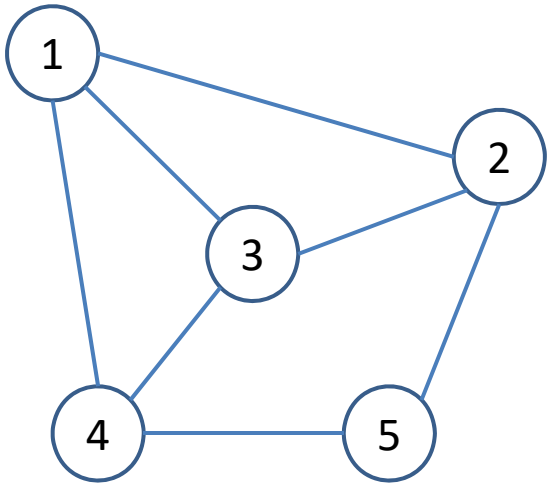
Despite the advantages of arc diagrams, and the room available to draw labels beside nodes, if there are too many edges that cross each other, it becomes difficult to read the edges. We next introduce an alternative visualization technique that eliminates edge crossings.

Basic Graph Layout Techniques

- Force-directed layout
- Arc-diagram
- Adjacency matrix
- Circular layout

Adjacency Matrix Representations

- An adjacency matrix contains one row and one column for each node of a network.
  - Given two nodes i and j, the entry located at (i, j) and (j, i) in the matrix contain information about the edge(s) between the two nodes.
  - Typically, each cell contains a boolean value indicating if an edge exists between the two nodes.
  - If the graph is undirected, the matrix is symmetric.



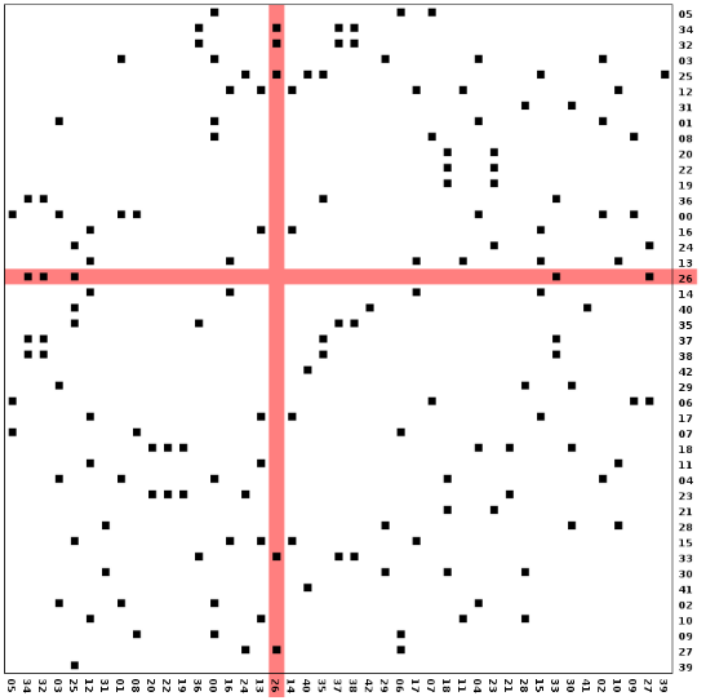
	1	2	3	4	5
1		T	T	T	
2	T		T		T
3	T	T		T	
4	T		T		T
5		T		T	

Adjacency Matrix Representations

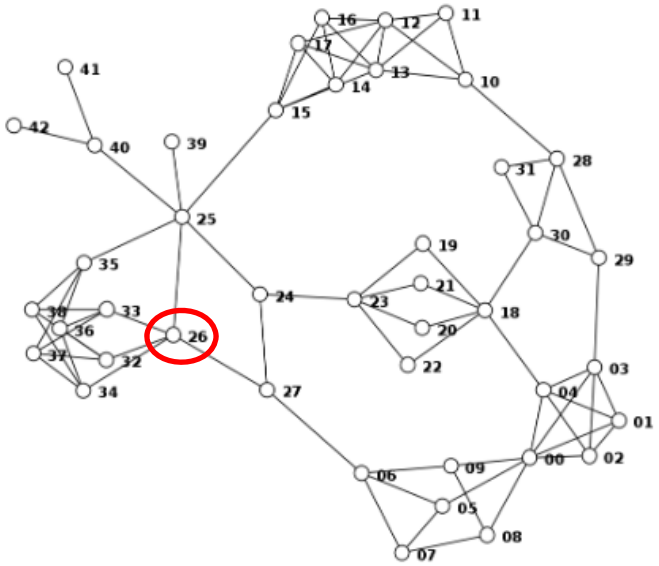
- An adjacency matrix contains one row and one column for each node of a network.
  - Given two nodes i and j, the entry located at (i, j) and (j, i) in the matrix contain information about the edge(s) between the two nodes.
  - Typically, each cell contains a boolean value indicating if an edge exists between the two nodes.
  - If the graph is undirected, the matrix is symmetric.

- Pros:
  - Visualizing a network as a matrix has the advantage of **eliminating all edge crossings**, since the edges correspond to non-overlapping entries.
- Cons:
  - The ordering of rows and columns greatly influences how easy it is to interpret the matrix.
  - *Difficult to follow a path in the graph*.
  - Limited by screen resolution.

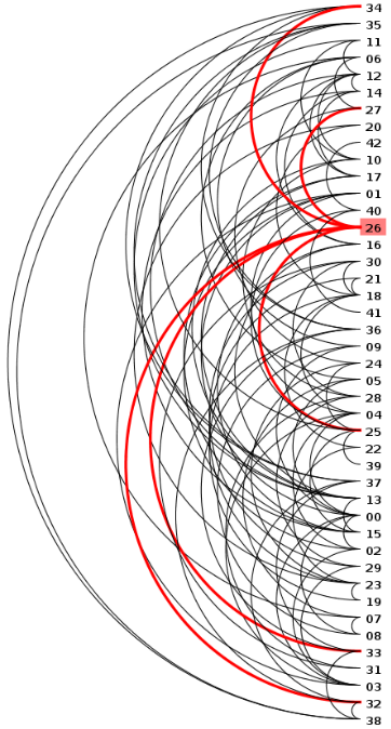
An Example



Adjacency matrix



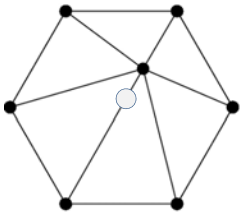
Force-directed



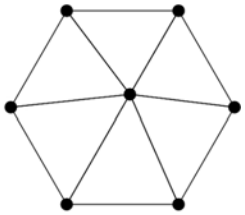
Arc diagram

Adjacency matrix visualizations of a 43-node, 80-edge network. Left: with a random ordering of rows and columns.

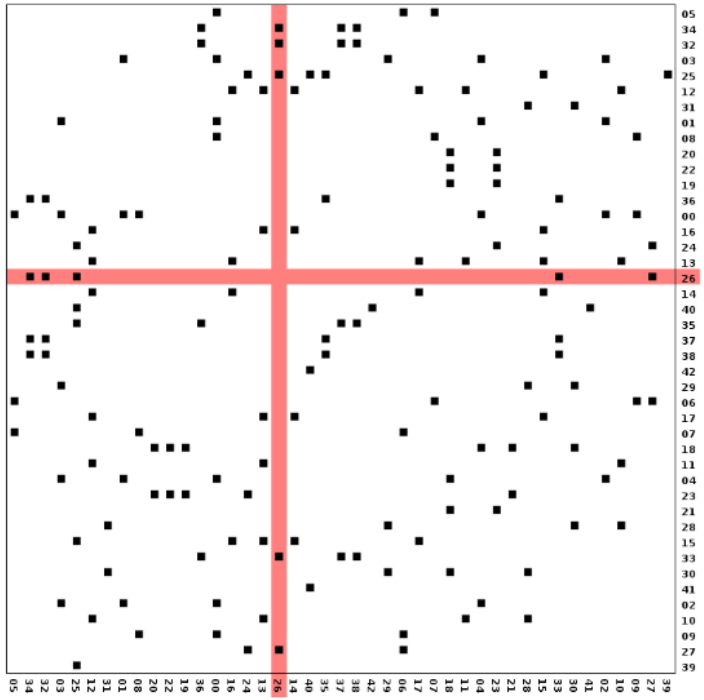
An Example

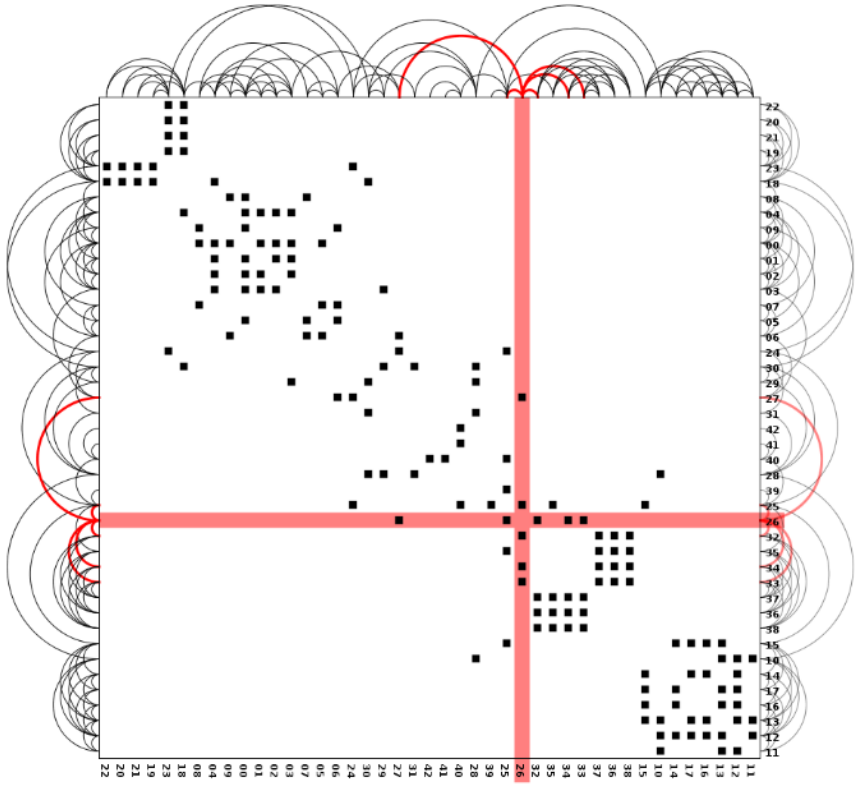


before



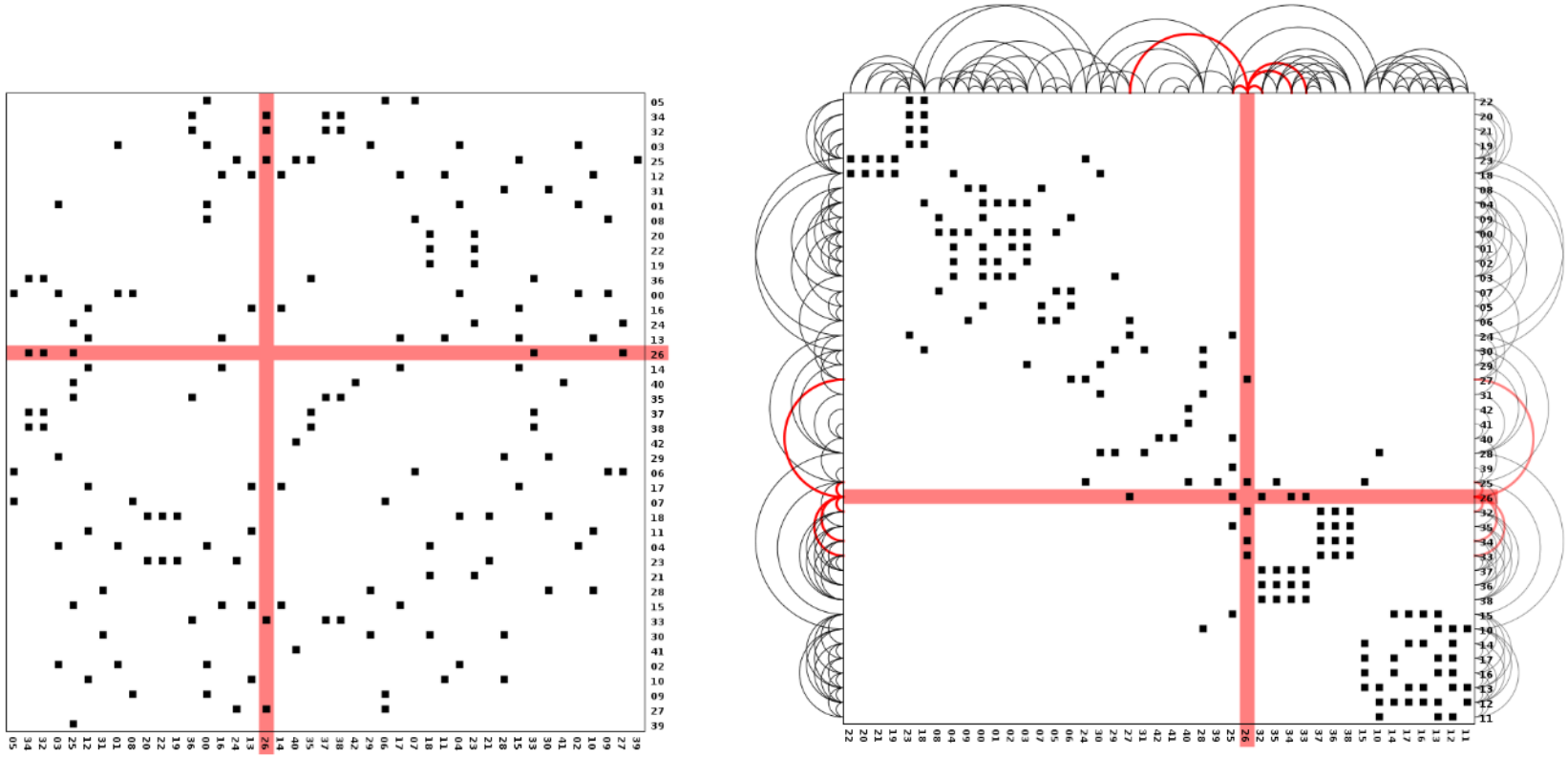
after





Adjacency matrix visualizations of a 43-node, 80-edge network. Left: with a random ordering of rows and columns. Right: after **barycenter ordering** and adding arc diagrams. The multiple arc diagrams are redundant, but reduce the distance of eye movements from the inside of the matrix to the nearest arcs.

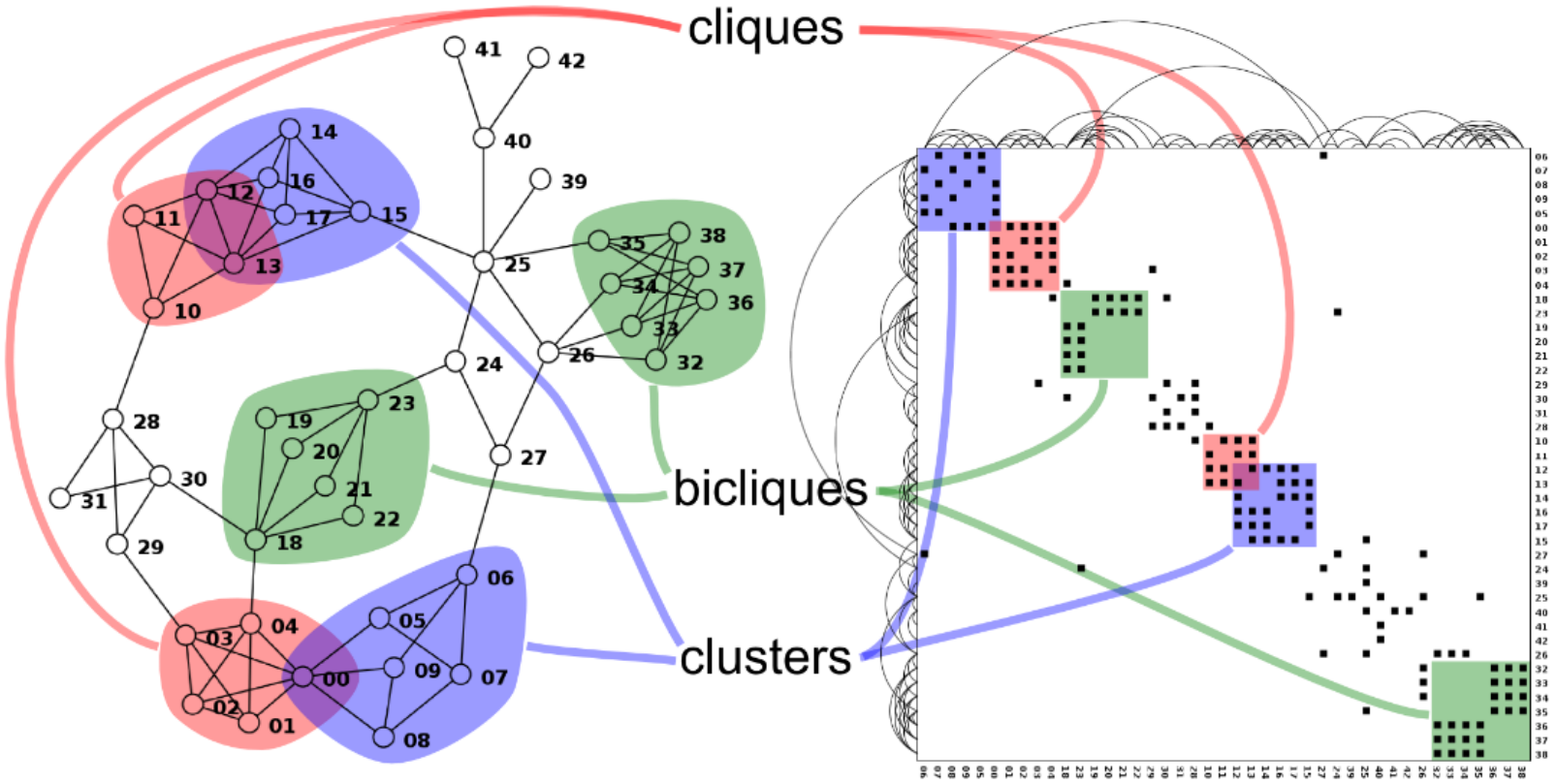
An Example



Interestingly, by bringing nodes “closer” to their neighbors with the **barycenter heuristic**, this pushes the edges (filled-in matrix cells) **closer to the diagonal of the matrix**, making certain **patterns** appear in the positions of the cells.

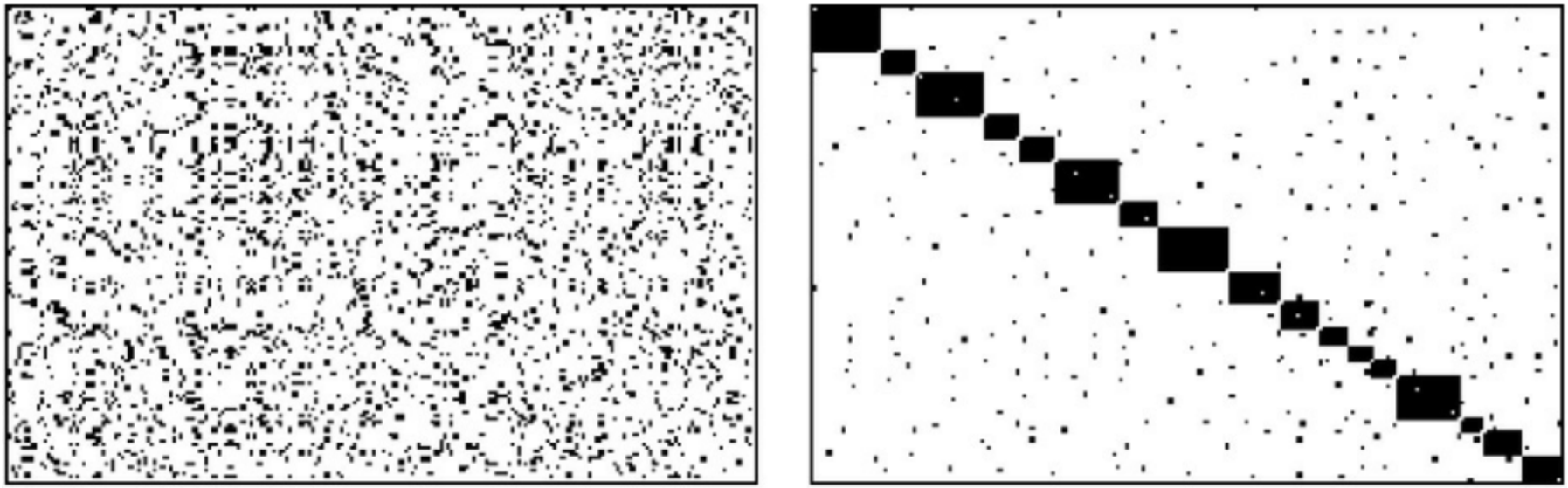
Adjacency Matrix Representations

Certain **subgraphs** (subsets of nodes and edges in the graph) correspond to easy-to-recognize patterns in the adjacency matrix, given an appropriate ordering of rows and columns.



Patterns corresponding to interesting subgraphs appear **along the diagonal** of an appropriately ordered adjacency matrix (say, via barycenter ordering)

Another Example

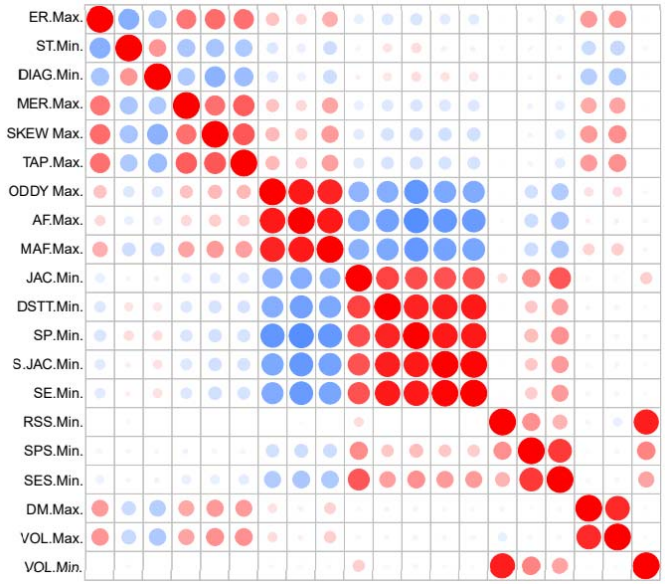


The adjacency matrix of a 210-vertex graph with 1505 edges composed of 17 dense clusters. On the left, the vertices are ordered randomly and the graph structure can hardly be observed. On the right, the vertex ordering is by cluster and the 17-cluster structure is evident. Each black dot corresponds to an element of the adjacency matrix that has the value one, the white areas correspond to elements with the value zero.

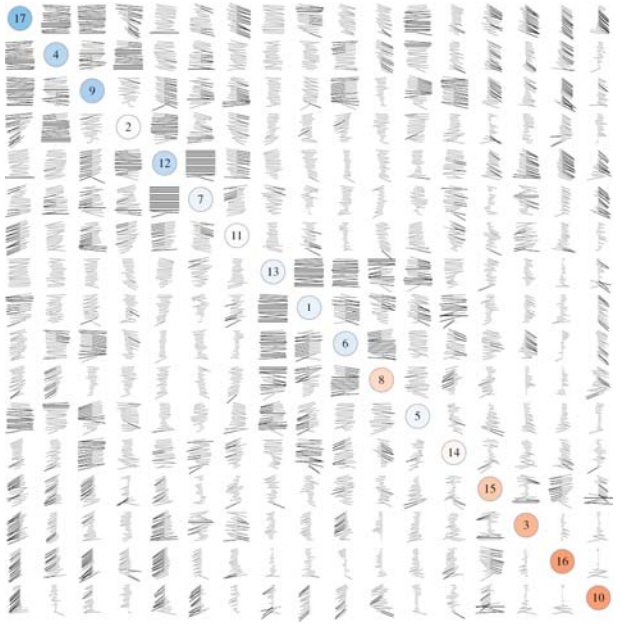
Matrix diagonalization in itself is an important application of clustering algorithms.

Other advantages:

Matrices have the added advantage of also being able to display information related to each edge within the entries of the matrix. For example, if the edges are weighted, this weight can be shown in the color of the entry.



Entries can also contain small graphics or glyphs, as in Brandes and Nick's "gestaltmatrix" where each entry contains a glyph showing the evolution of the edge over time.



Limitations:

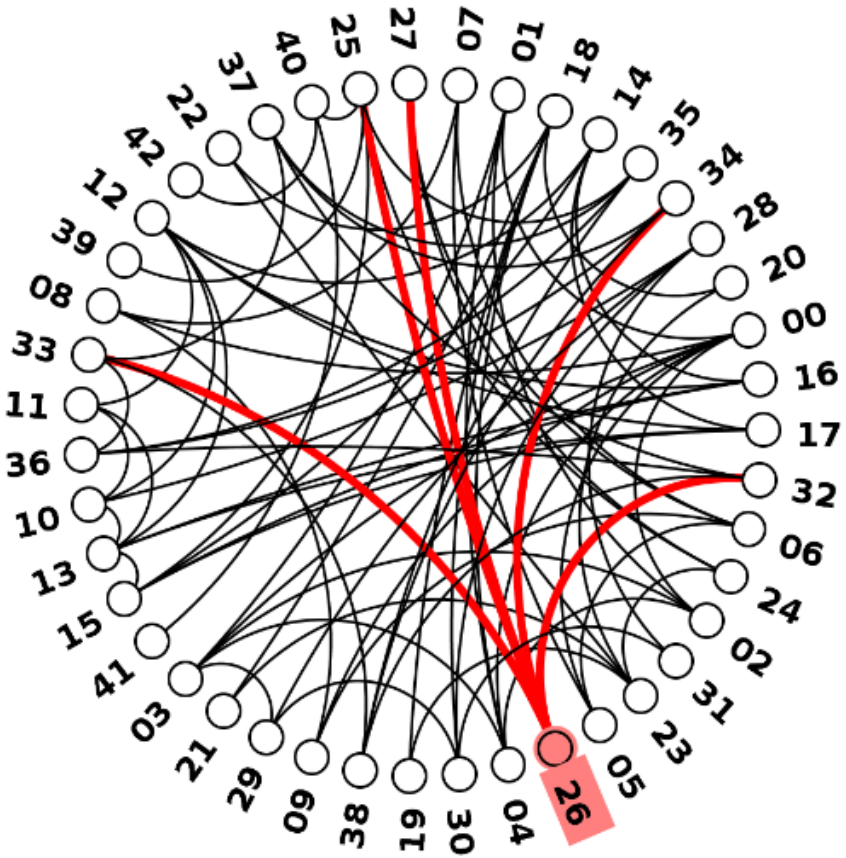
An important disadvantage of using adjacency matrices, however, is that the **space they require is $O(N^2)$** where N is the number of nodes.

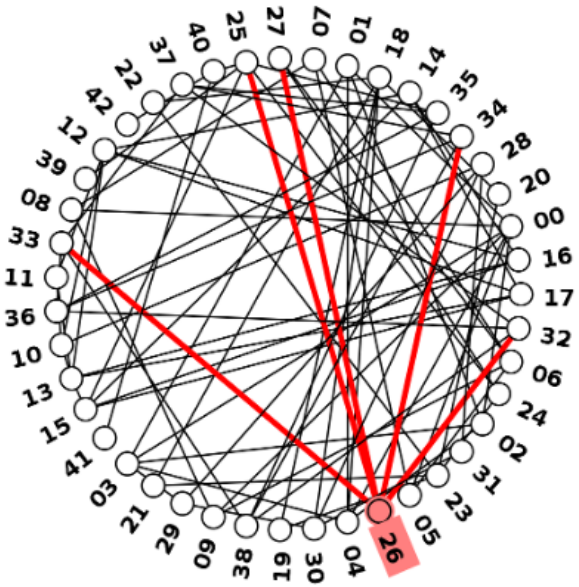
Basic Graph Layout Techniques

- Force-directed layout
- Arc-diagram
- Adjacency matrix
- Circular layout

Circular Layouts

- Position nodes on the circumference of a circle, while the edges are drawn as curves rather than straight lines.

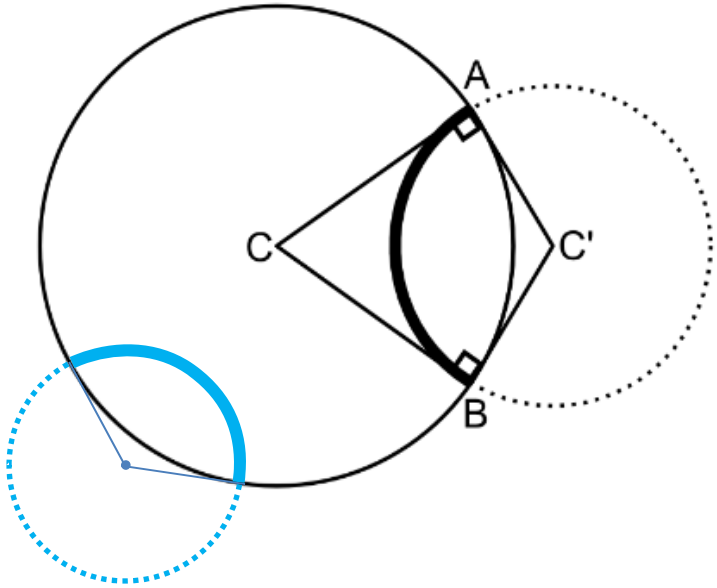




Circular Layouts

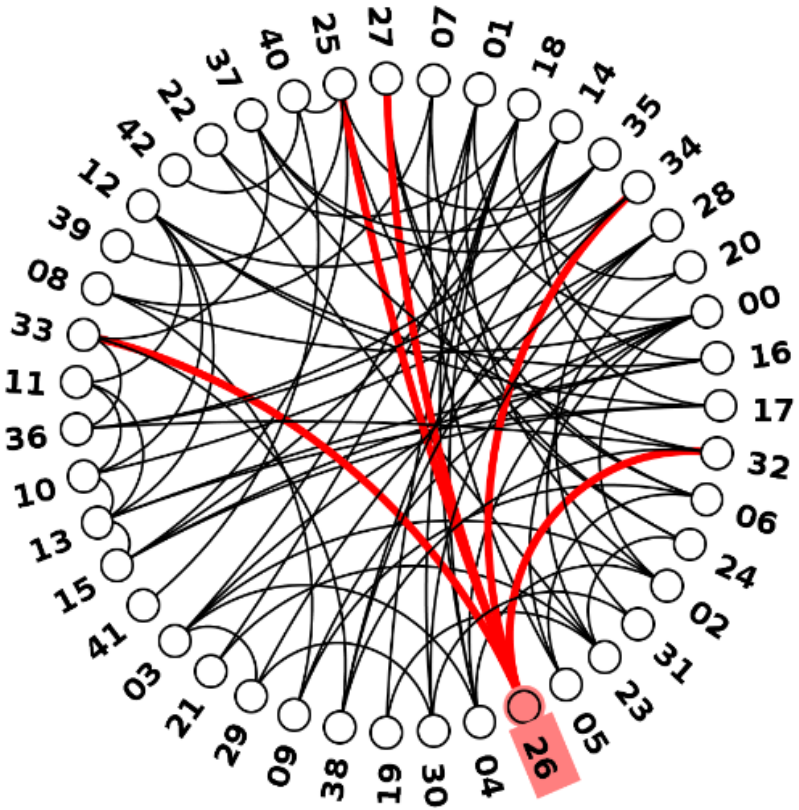
Let C be the center of the circular layout.

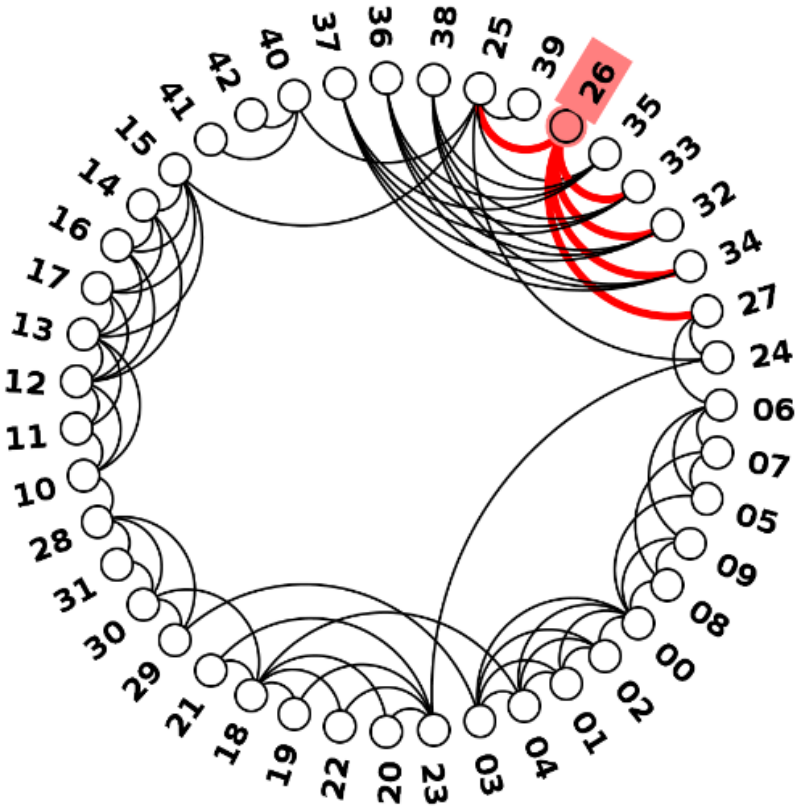
To draw a curved arc between A and B on the circumference, we draw a circular arc that is tangent to the lines AC and BC. The center C' of the arc can be found by finding the intersection between a line through A that is perpendicular to AC, and a line through B that is perpendicular to BC.



Circular Layouts

Again, the order chosen for the nodes greatly influences how clear the visualization is. The barycenter heuristic can be applied again to this layout, **with a slight modification to account for the "wrap around" of the circular layout**.

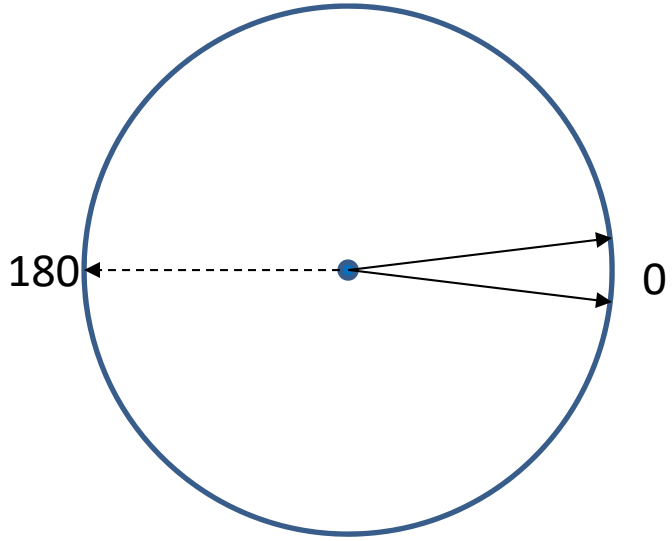




Circular Layouts

To correctly adapt the barycenter heuristic to this layout, consider how to compute the “average position” of the neighbors of a node.

As an example, if one neighbor is positioned at an angle of 10 degrees, and another is at an angle of 350 degrees, simply taking the numerical average yields $(10 + 350)/2 = 180$ degrees, whereas the intuitively correct barycenter is at 0 degree (or 360 degrees).

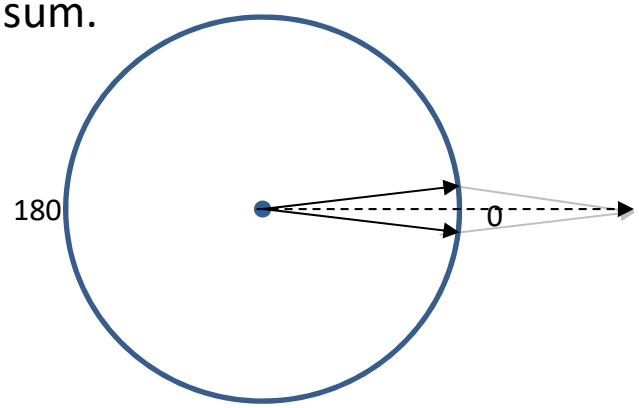


Circular Layouts

To correctly adapt the barycenter heuristic to this layout, consider how to compute the “average position” of the neighbors of a node.

As an example, if one neighbor is positioned at an angle of 10 degrees, and another is at an angle of 350 degrees, simply taking the numerical average yields $(10 + 350)/2 = 180$ degrees, whereas the intuitively correct barycenter is at 0 degree (or 360 degrees).

So, to correctly compute the barycenter, we do not compute averages of angles. Instead, we convert each node to a unit vector in the appropriate direction, add these unit vectors together, and find the angle of the vector sum.



Circular Layouts

To correctly adapt the barycenter heuristic to this layout, consider how to compute the "average position" of the neighbors of a node.

As an example, if one neighbor is positioned at an angle of 10 degrees, and another is at an angle of 350 degrees, simply taking the numerical average yields $(10 + 350)/2 = 180$ degrees, whereas the intuitively correct barycenter is at 0 degree (or 360 degrees).

So, to correctly compute the barycenter, we do not compute averages of angles. Instead, we convert each node to a unit vector in the appropriate direction, add these unit vectors together, and find the angle of the vector sum.

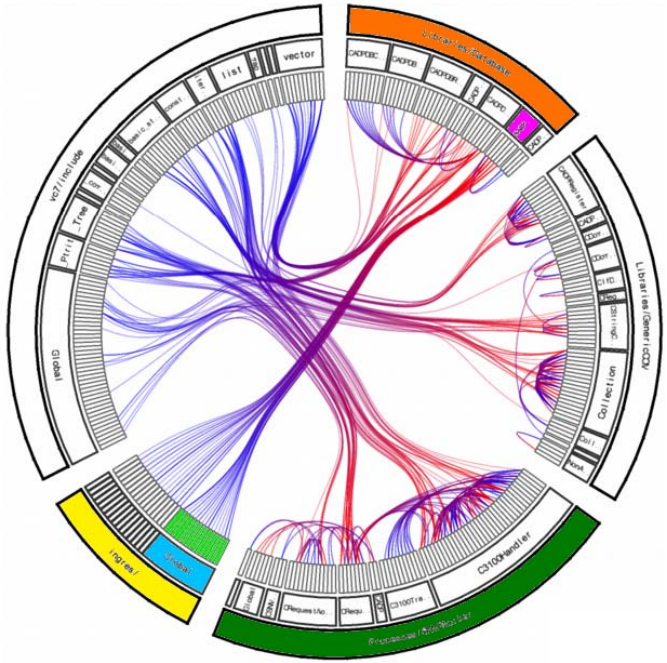
Define the function `angle(p)=p*2*pi/N` giving the angle of a node at position p. Then, the pseudo-code for the barycenter heuristic becomes

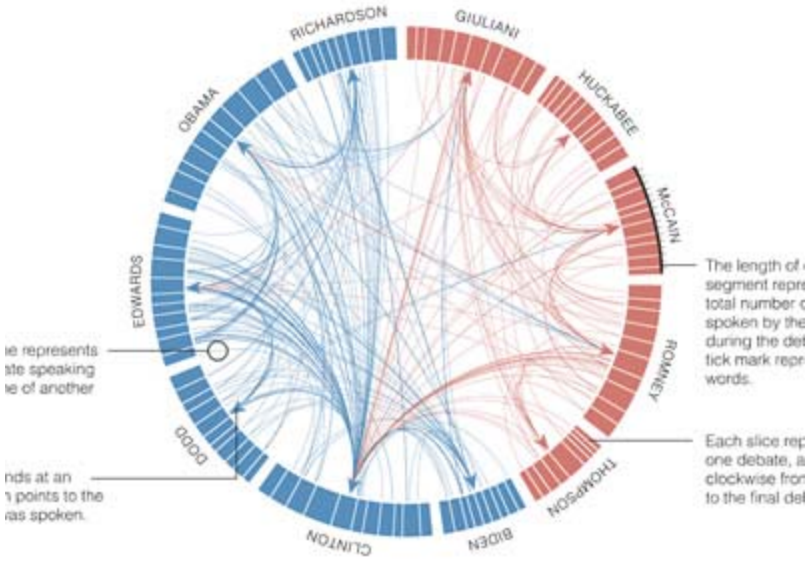
```
1 // compute average position of neighbors
2 for i1 = 0 to N-1
3       node1 = nodes[i1]
4       p1 = positionOfNode(i1)
5*      sum_x = cos(angle(p1))
6*      sum_y = sin(angle(p1))
7       for j = 0 to node1.neighbors.length-1
8            i2 = node1.neighbors[j]
9            node2 = nodes[i2]
10           p2 = positionOfNode(i2)
11*          sum_x = sum_x + cos(angle(p2))
12*          sum_y = sum_y + sin(angle(p2))
13*        orderedNodes[p1].average
14*             = angleOfVector(sum_x,sum_y)
15
16 // sort the array according to the values of average
17 sort( orderedNodes, comparator)
```

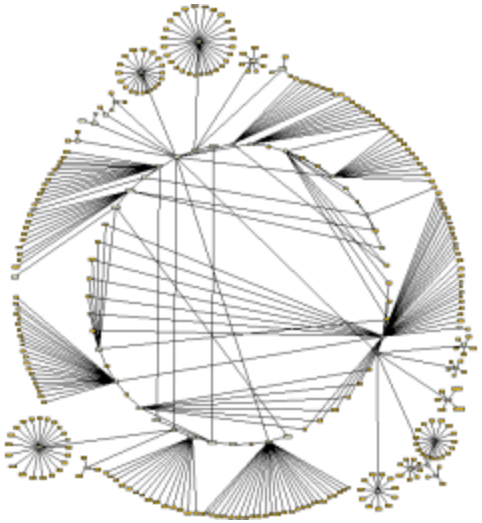
```
function angleOfVector( x, y )
    hypotenuse = sqrt( x*x + y*y )
    theta = arcsin( y / hypotenuse )
    if x < 0
       theta = pi - theta
    // Now theta is in [-pi/2,3*pi/2]
    if theta < 0
       theta = theta + 2*pi
    // Now theta is in [0,2*pi]
    return theta
```

Or, you can use function **atan2** to get the angle and recalibrate it to the range [0, 2*pi]

More Examples



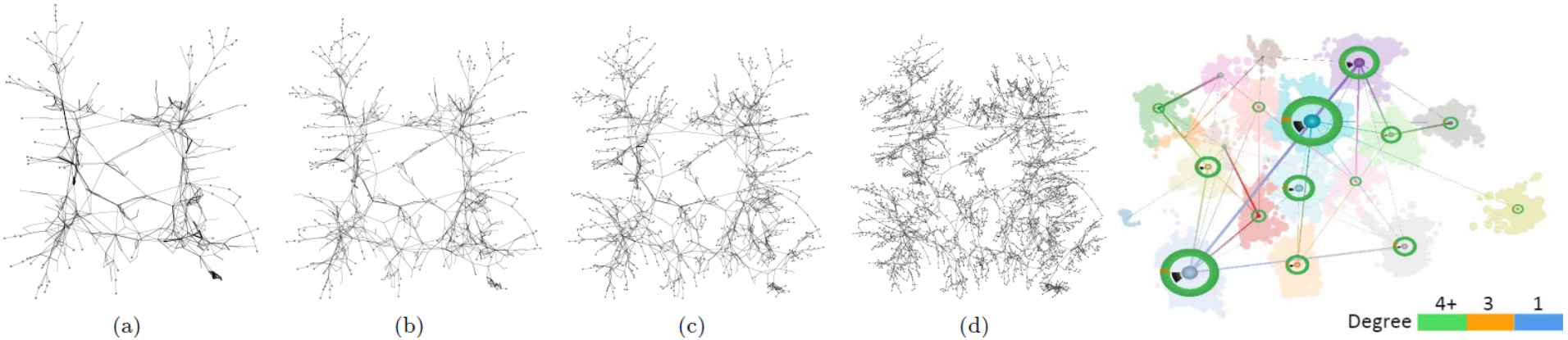




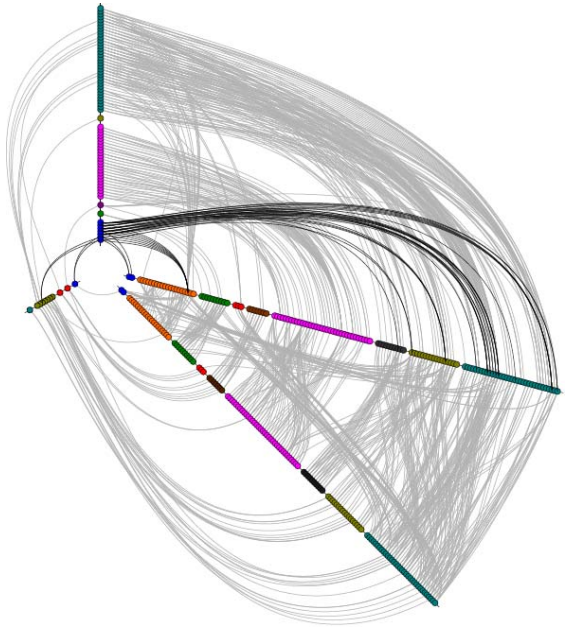
Comparison of Layout Techniques

	node-link diagram	circular layout	arc diagram	adjacency matrix
Height of each node's label	$O(1/\sqrt{N})$ (best)	$O(\pi/N)$	$O(1/N)$	$O(k_1/N)$
Easy to perceive paths	yes	somewhat	somewhat	no
Avoids edge crossings	no	no	no	yes
Avoids ambiguity from edges passing close to nodes	no	yes	yes	yes
Can depict an ordering of nodes	no	yes	yes	yes
Can depict information about each edge	somewhat	somewhat	somewhat	yes
Node labels all have the same orientation, for easier reading	yes	no	yes	yes

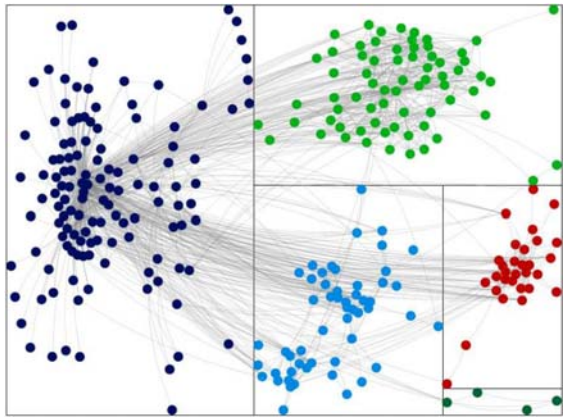
Other layout techniques



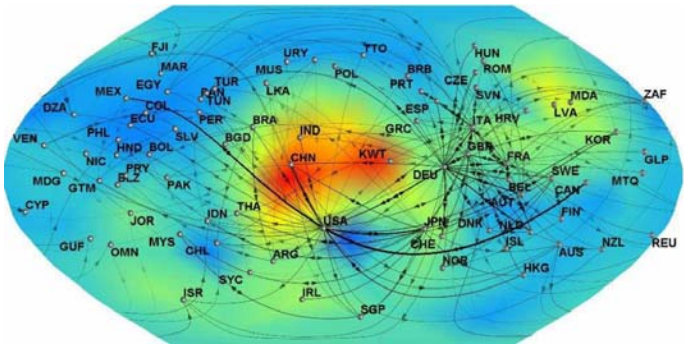
Multi-level layout



hive plot showing dependencies between classes of the Flare visualization tool-kit
http://bost.ocks.org/mike/hive/



The group-in-a-box layout from NodeXL



The GeoSOM layout for an international metal trading network

Additional Reading

Helen Gibson, Joe Faith and Paul Vickers. “A survey of two-dimensional graph layout techniques for information visualization.” Information Visualization, 12 (3-4): 324-357, 2013.

Vehlow, Corinna, Fabian Beck, and Daniel Weiskopf. “The state of the art in visualizing group structures in graphs.” *Eurographics Conference on Visualization (EuroVis)-STARs*. Vol. 2. 2015.

Hadlak, Steffen, Heidrun Schumann, and Hans-Jörg Schulz. "A survey of multi-faceted graph visualization." *Eurographics Conference on Visualization (EuroVis)*. 2015.

Peter Oliver, Eugene Zhang, and Yue Zhang, "Scalable Hypergraph Visualization", IEEE Transactions on Visualization and Computer Graphics, Vol 30(1), 2024, (IEEE Vis 2023), to appear.

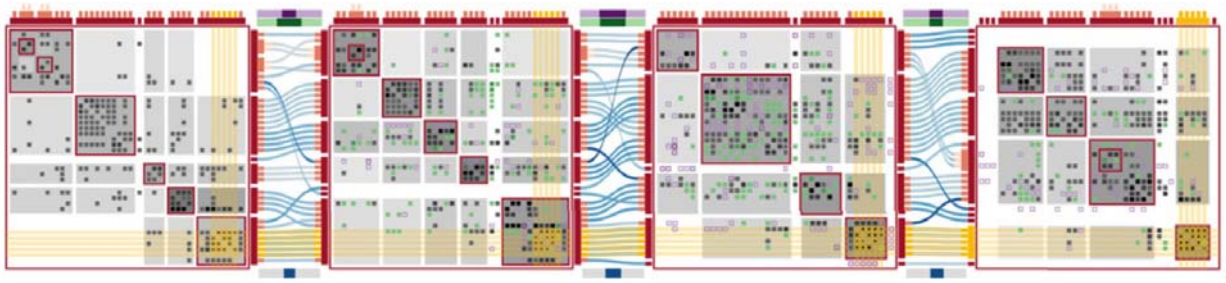
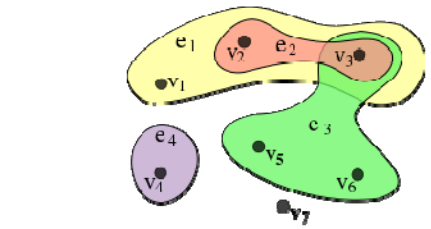
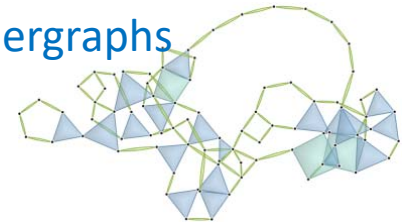


Fig. 2. Friendship network among students over time [8] from left to right, where the intensity of the relationship is encoded in the edge weights mapped to the brightness of shades of gray, green (added edges), or purple (removed edges). For each point in time, the hierarchical group structure is encoded within the matrix using indented nested contours and visualized by icicle plots attached to the matrix. To compare hierarchies, we draw time transition edges that connect corresponding leaves of the icicle plots. Groups of friends change over time, while some groups remain relatively stable, e.g., the group selected at the last time point and highlighted in yellow over all points in time.



Hypergraphs



TREE

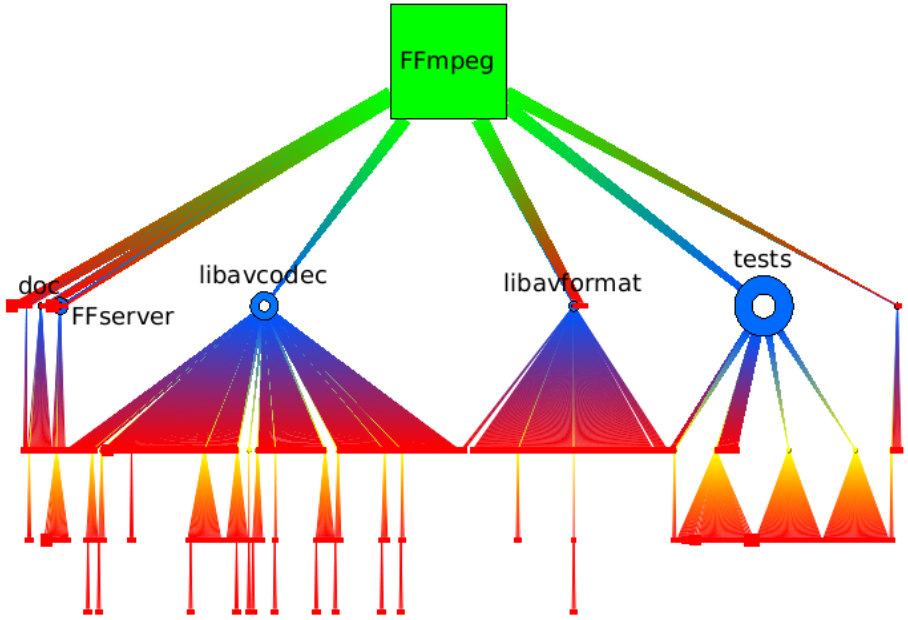
Tree Visualization

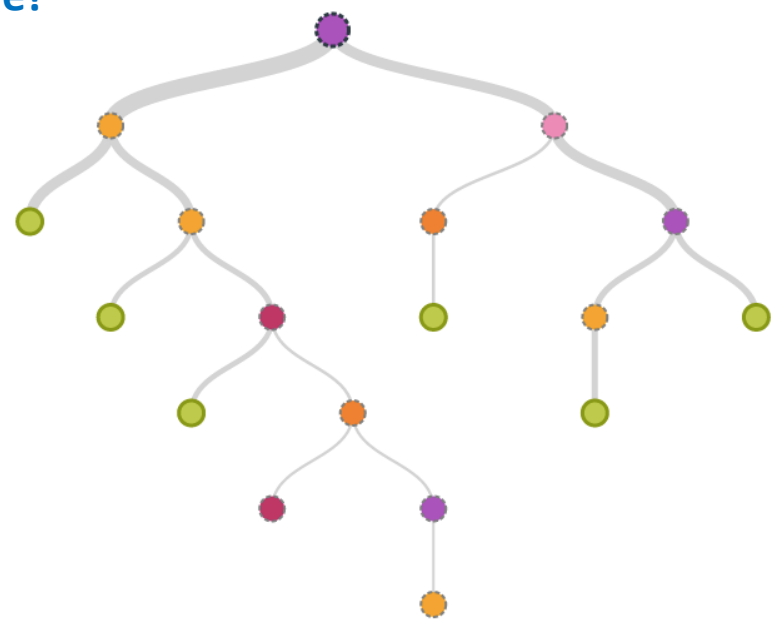
- A tree is defined as a set of nodes and edges.
- Every edge has a pair of nodes: the parent node and the child node.
- **A child node has only one parent node**.
- **Between any two notes in the tree, there is a unique path**.
- A tree is a network of connected nodes where there are **no loops**.
- Root node: the single node that has no parents.
- Leaf node: the nodes that have no children.
- Depth of the tree: number of nodes from the root to the lowest leaf.

Tree Visualization

•Ball-and-stick visualization: use the position and appearance of the glyphs

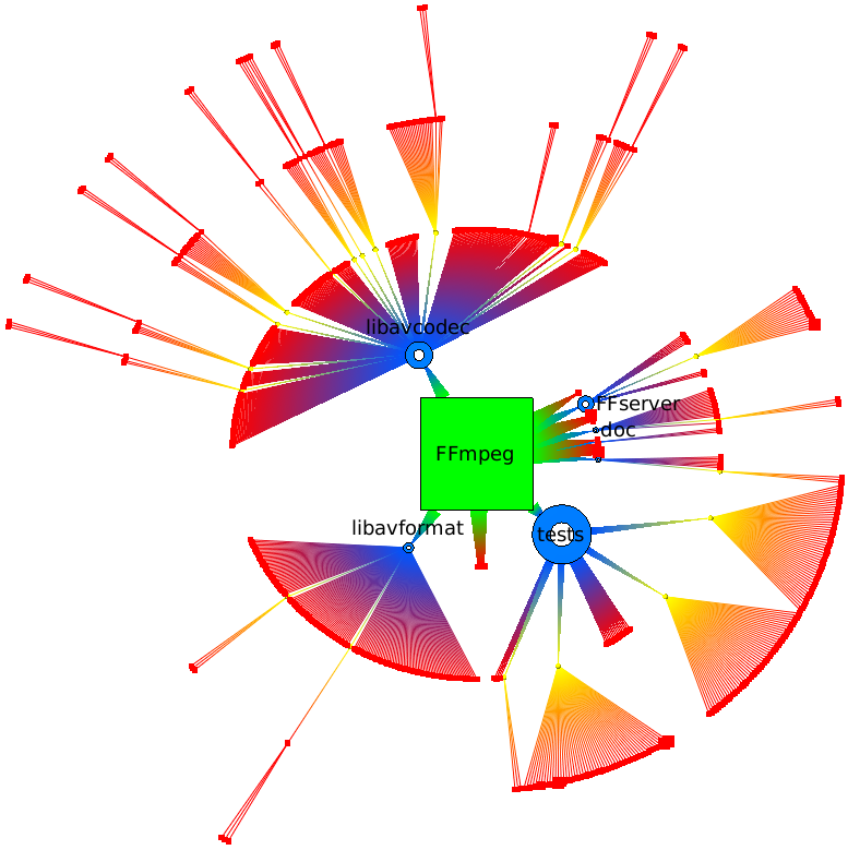
An upside-down tree!

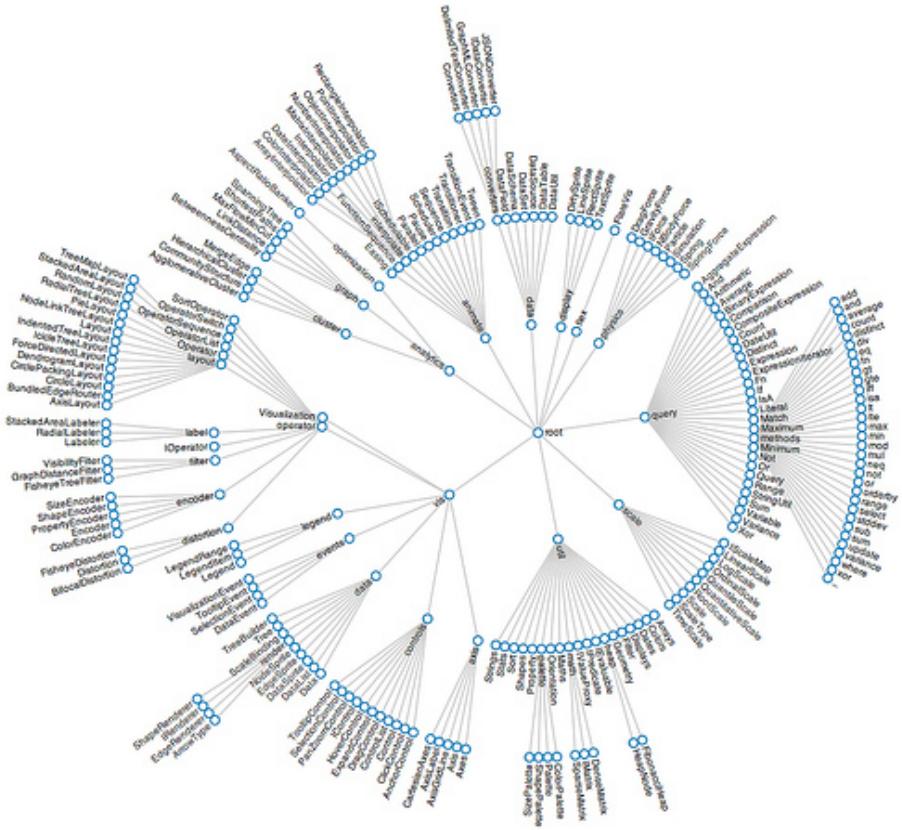




Rooted-Tree Layout of the FFmpeg software

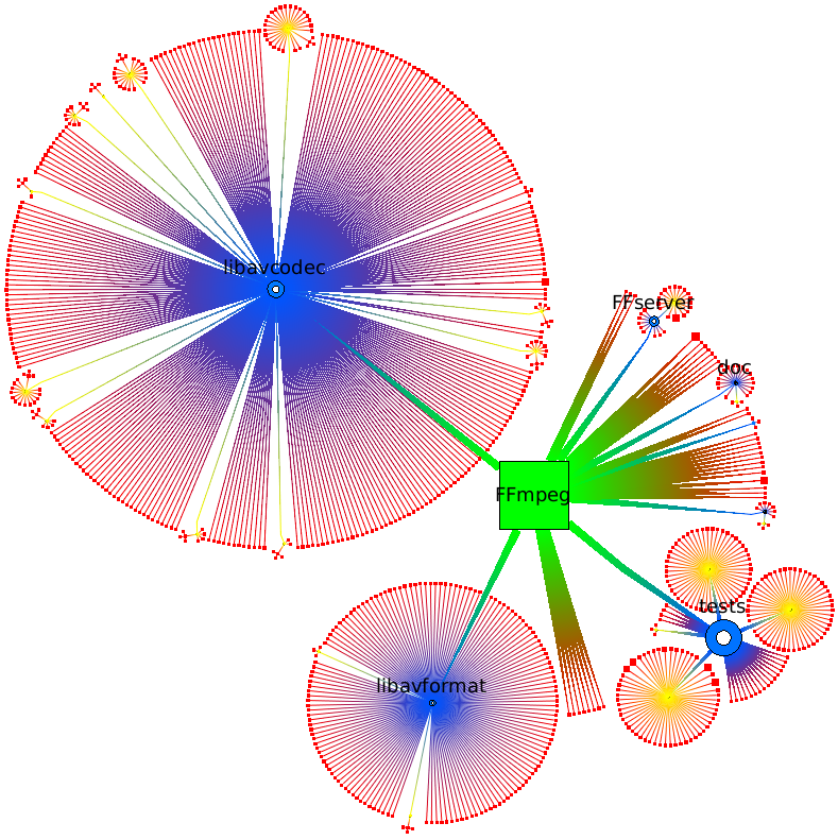
Tree Visualization

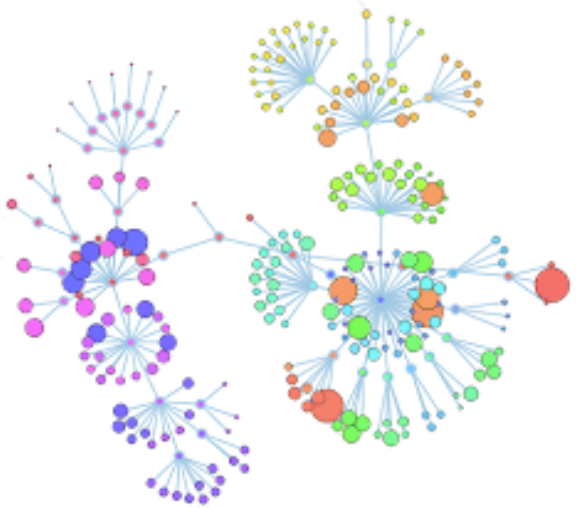


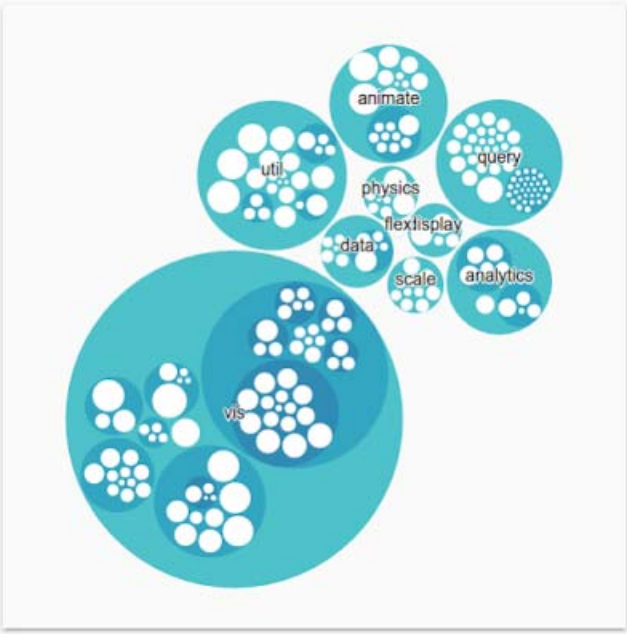


Radial-Tree Layout

Tree Visualization

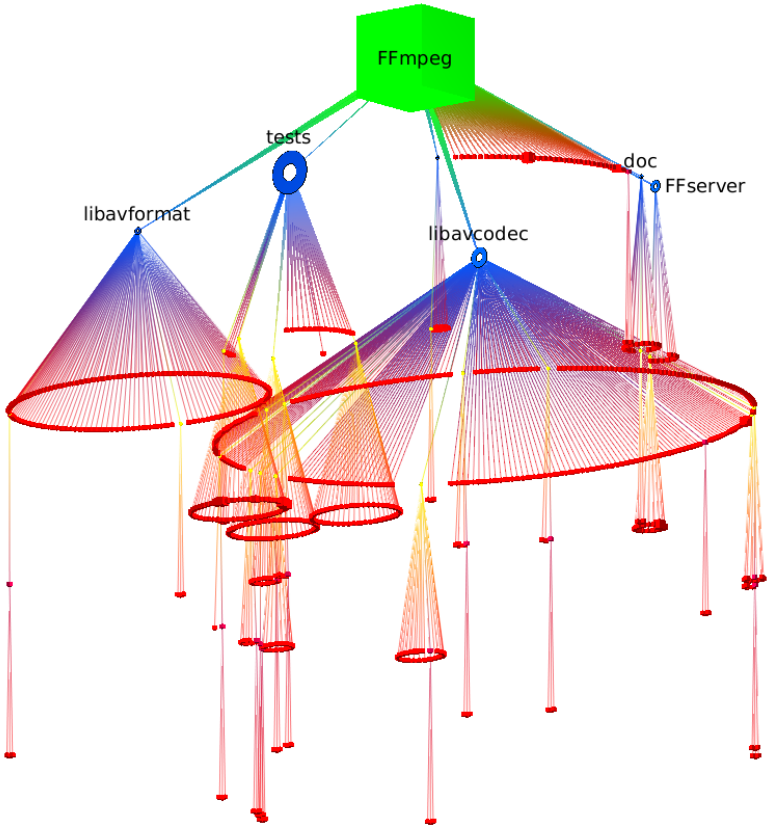


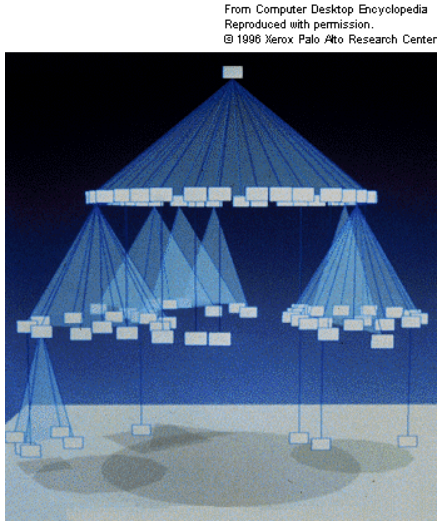


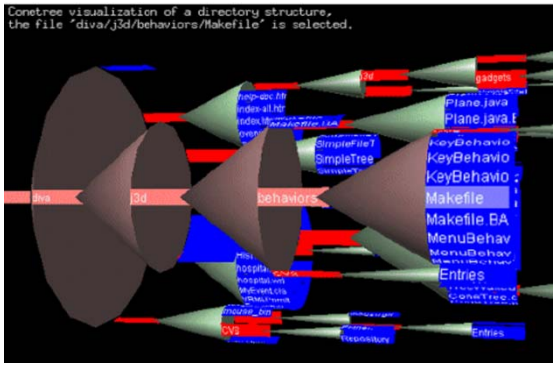


Bubble-Tree Layout

Tree Visualization

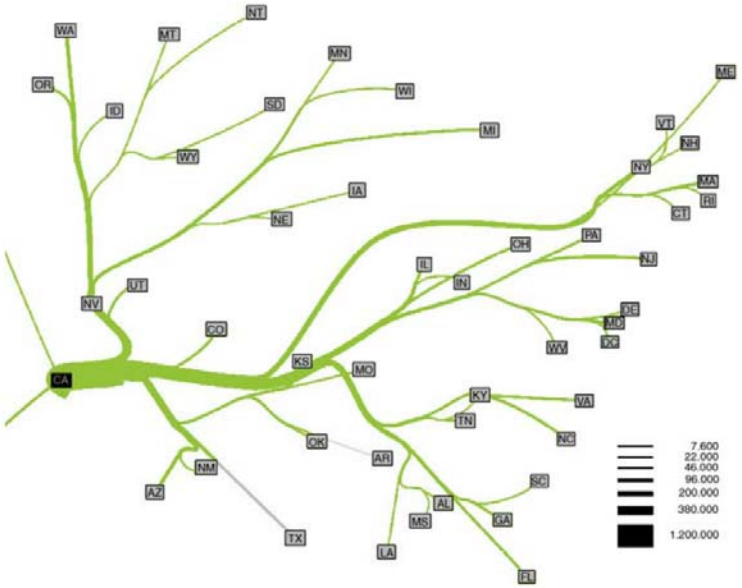






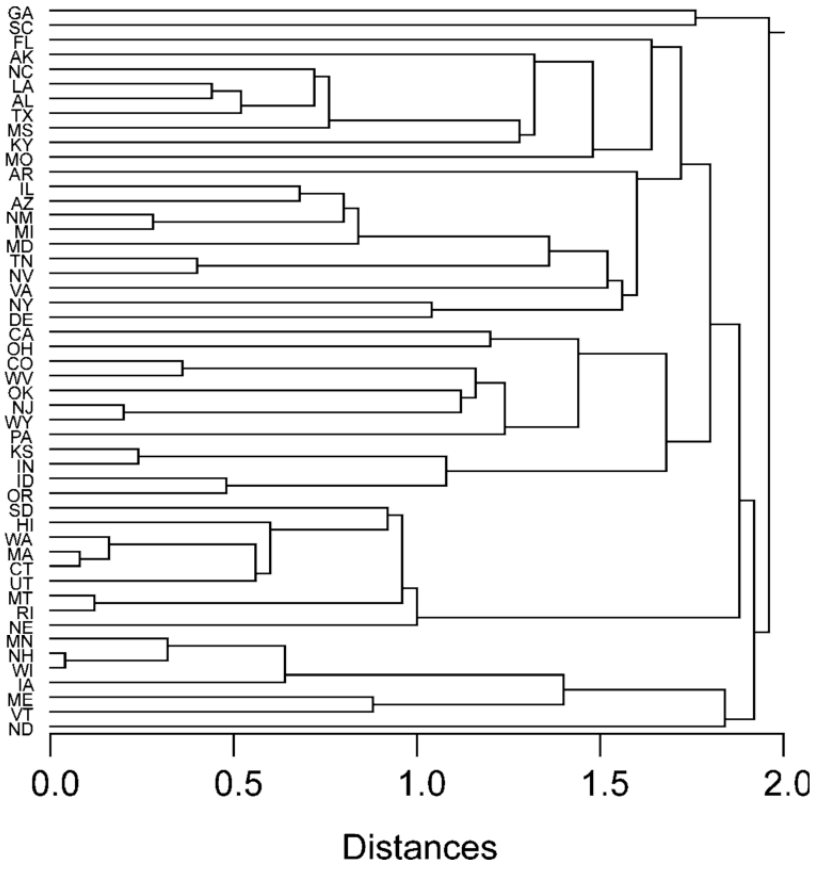
3D Cone-Tree Layout

Tree Visualization



Colorado migration data from 1995 to 2000

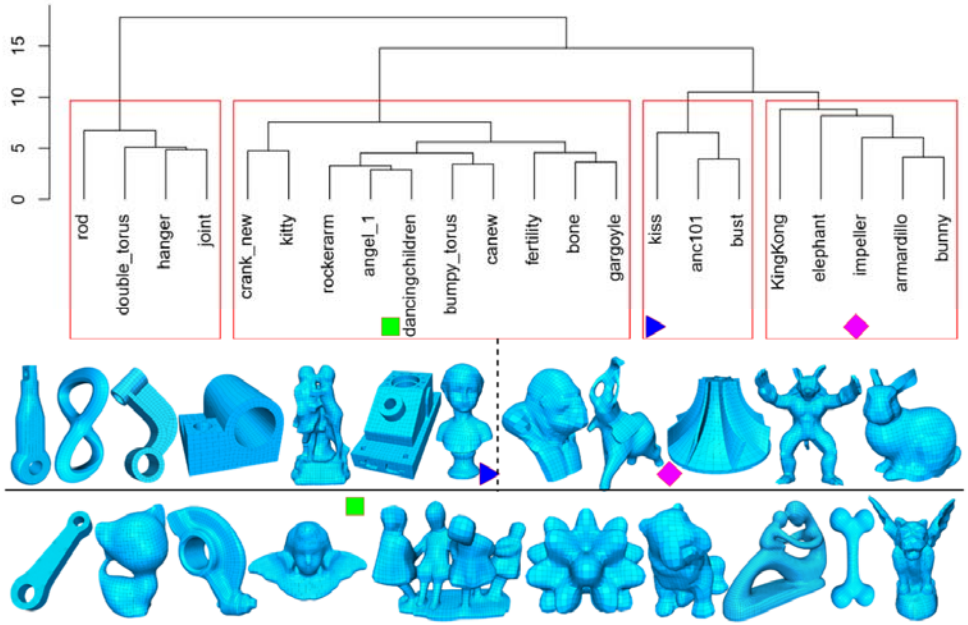
Flow map

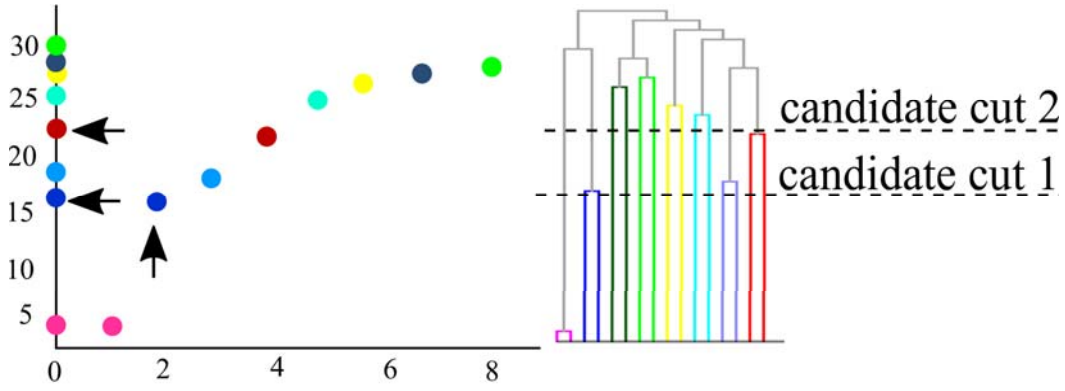


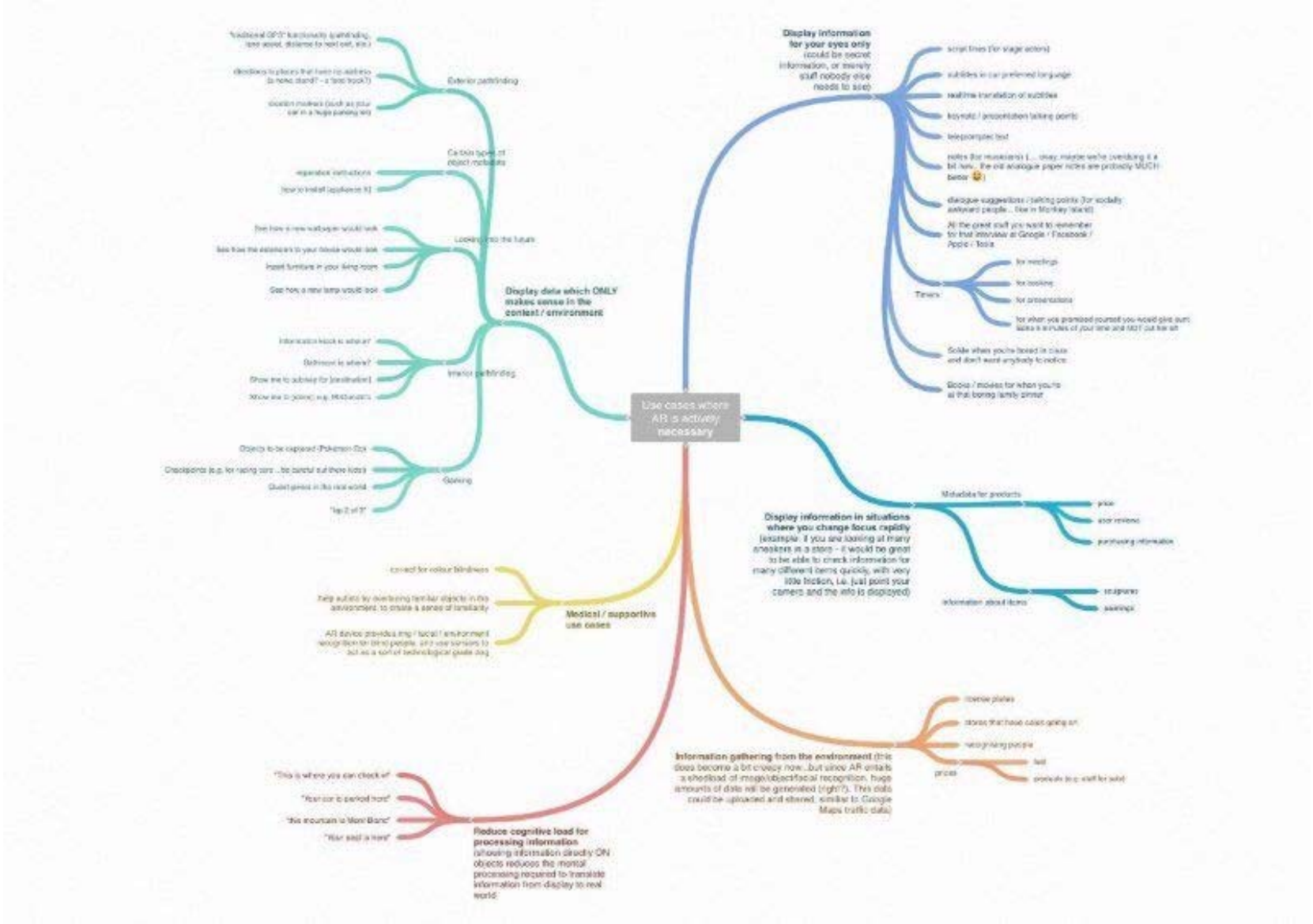
Hierarchical **cluster tree** of US murder rates

dendrogram

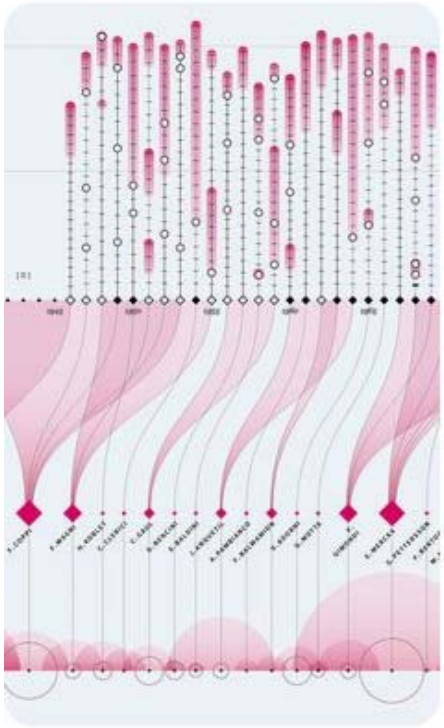
Dendrogram good at showing hierarchical clustering results



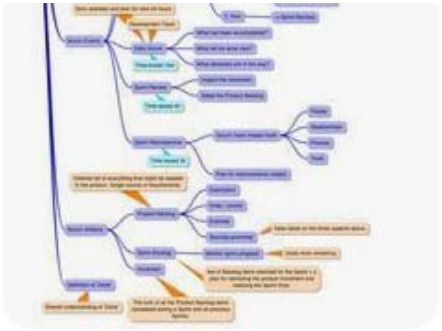



Use cases where AR is actively necessary
Display information for your eyes only (could be secret information, or merely stuff nobody else needs to see)
script lines (for stage actors)
subtitles in our preferred language
realtime translation of subtitles
keynote / presentation talking points
teleprompter text
notes (for musicians (... okay, maybe we're overdoing it a bit now... the old analogue paper notes are probably MUCH better 😉)
dialogue suggestions / talking points (for socially awkward people... like in Monkey World)
All the great stuff you want to remember for that interview at Google / Facebook / Apple / Tesla
Timers
for meetings
for cooking
for presentations
for when you promised yourself you would give ours extra 5 minutes of your time and NOT cut her off
Solve when you're bored in class and don't want anybody to notice
Books / movies for when you're at that boring family dinner
Display data which ONLY makes sense in the context / environment
Exterior pathfinding
"traditional GPS" functionality (pathfinding, time saved, distance to next exit, etc.)
directions to places that have no address (a notice board? - a bus kiosk?)
location markers (such as your car in a huge parking lot)
Certain types of object metadata
separation instructions
how to install (appliance X)
Looking into the future
See how a new wallpaper would look
See how the extension to your house would look
Insert furniture in your living room
See how a new lamp would look
Interior pathfinding
Information kiosk is where?
Bathroom is where?
Show me to subway for [destination]
Show me to [store], e.g. McDonald's
Gaming
Objects to be captured (Pokemon Go)
Checkpoints (e.g. for racing cars ...be careful out there kids!)
Quest areas in the real world
"lap 2 of 3"
Display information in situations where you change focus rapidly (example: if you are looking at many sneakers in a store - it would be great to be able to check information for many different items quickly, with very little friction, i.e. just point your camera and the info is displayed)
Metadata for products
price
user reviews
purchasing information
information about items
sculptures
paintings
Medical / supportive use cases
correct for colour blindness
help autists by overlaying familiar objects in the environment, to create a sense of familiarity
AR device providing (local) environment recognition for blind people, and use sensors to act as a sort of technological guide dog
Information gathering from the environment (this does become a bit creepy now...but since AR entails a shedload of image/object/facial recognition, huge amounts of data will be generated (right?). This data could be uploaded and shared, similar to Google Maps traffic data)
license plates
stores that have sales going on
racing/racing purpose
fast
price
products (e.g. stuff for sale)
Reduce cognitive load for processing information (showing information directly ON objects reduces the mental processing required to translate information from display to real world)
"This is where you can check in"
"Your car is parked here"
"the mountain is Mont Blanc"
"Your seat is here"

My Reebok ZR8 treadmill will be the end of me!



99 Giri d'Italia | La Lettura #281 dataviz



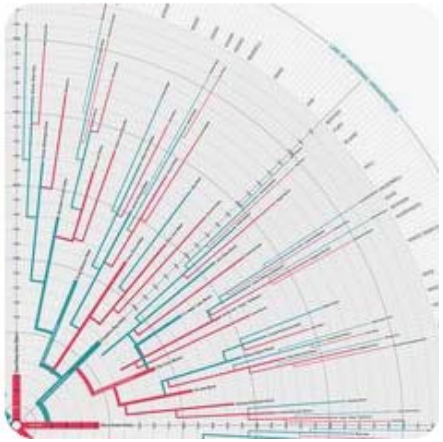
scrum guide mind map

La mia vita in infografica

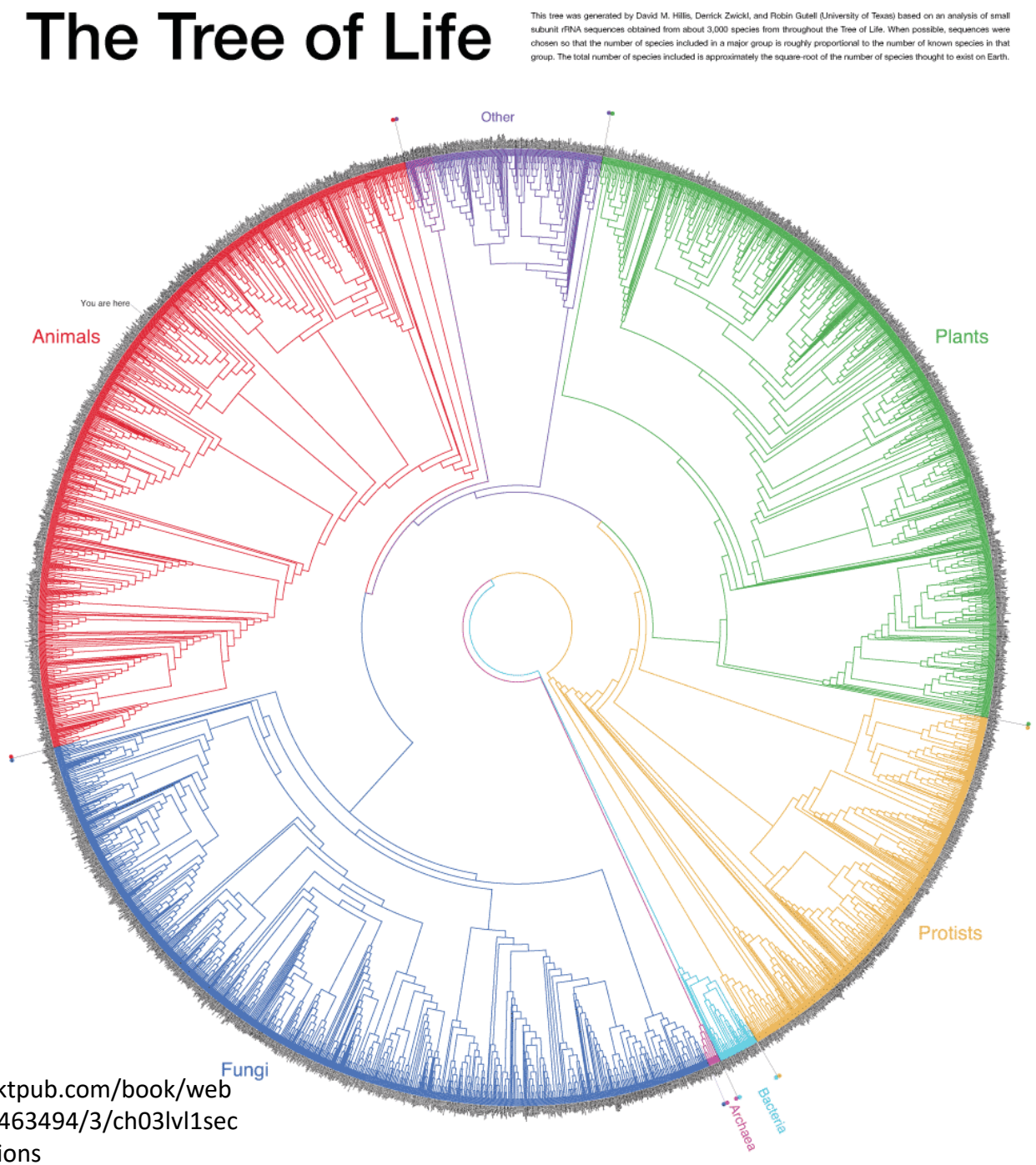
Faizan Ahmed on Twitter

Fivethirtyeight.com reveals the whole election in one gutsy...

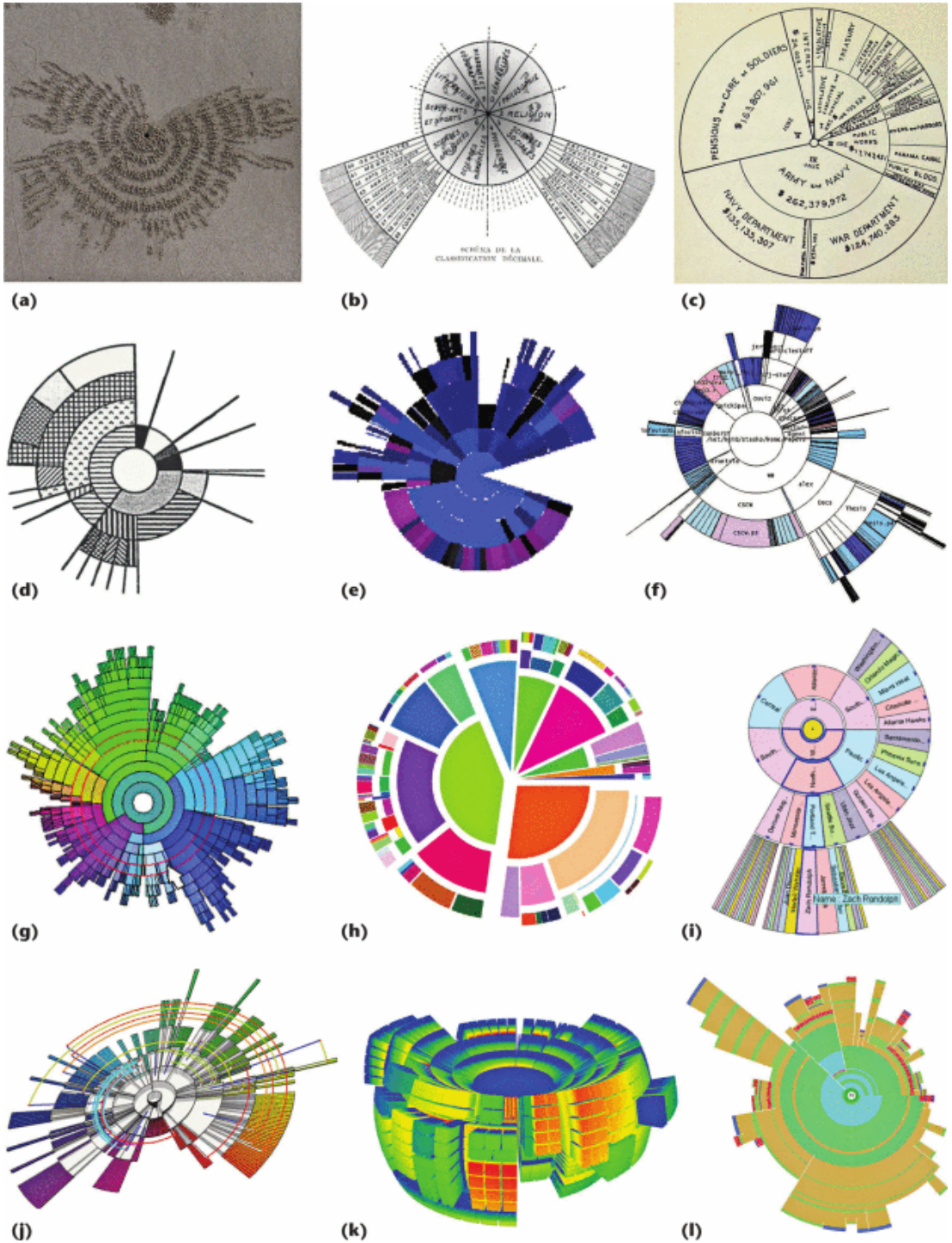
Bullet Journal Printable MS Word DIY Daily Routine Zone...



https://www.pinterest.com/pin/255086766377817048/



https://subscription.packtpub.com/book/web_development/9781786463494/3/ch03lvl1sec17/tree-based-visualizations

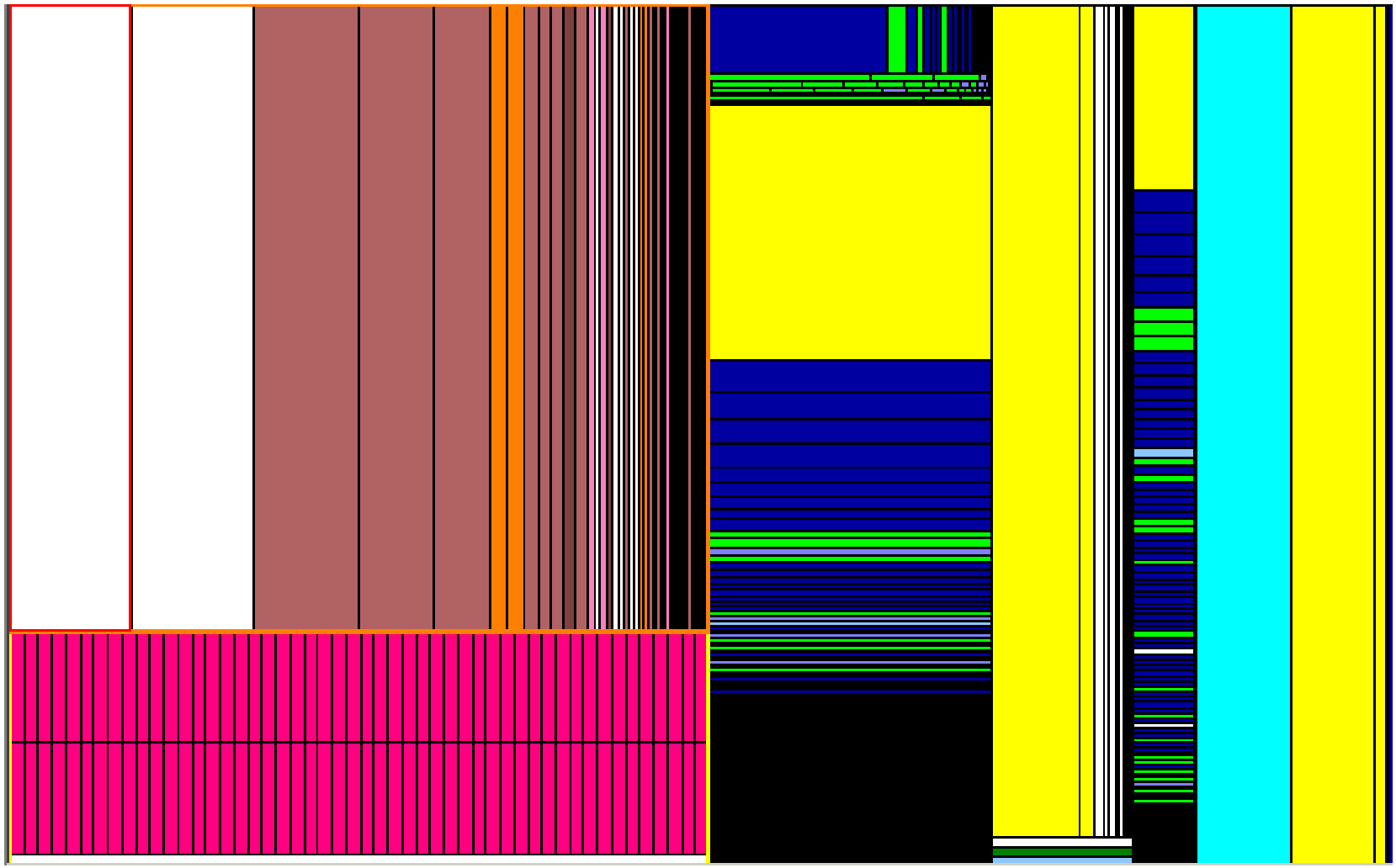


Treevis. Net by
Hans-Jorg Schulz

Treemap

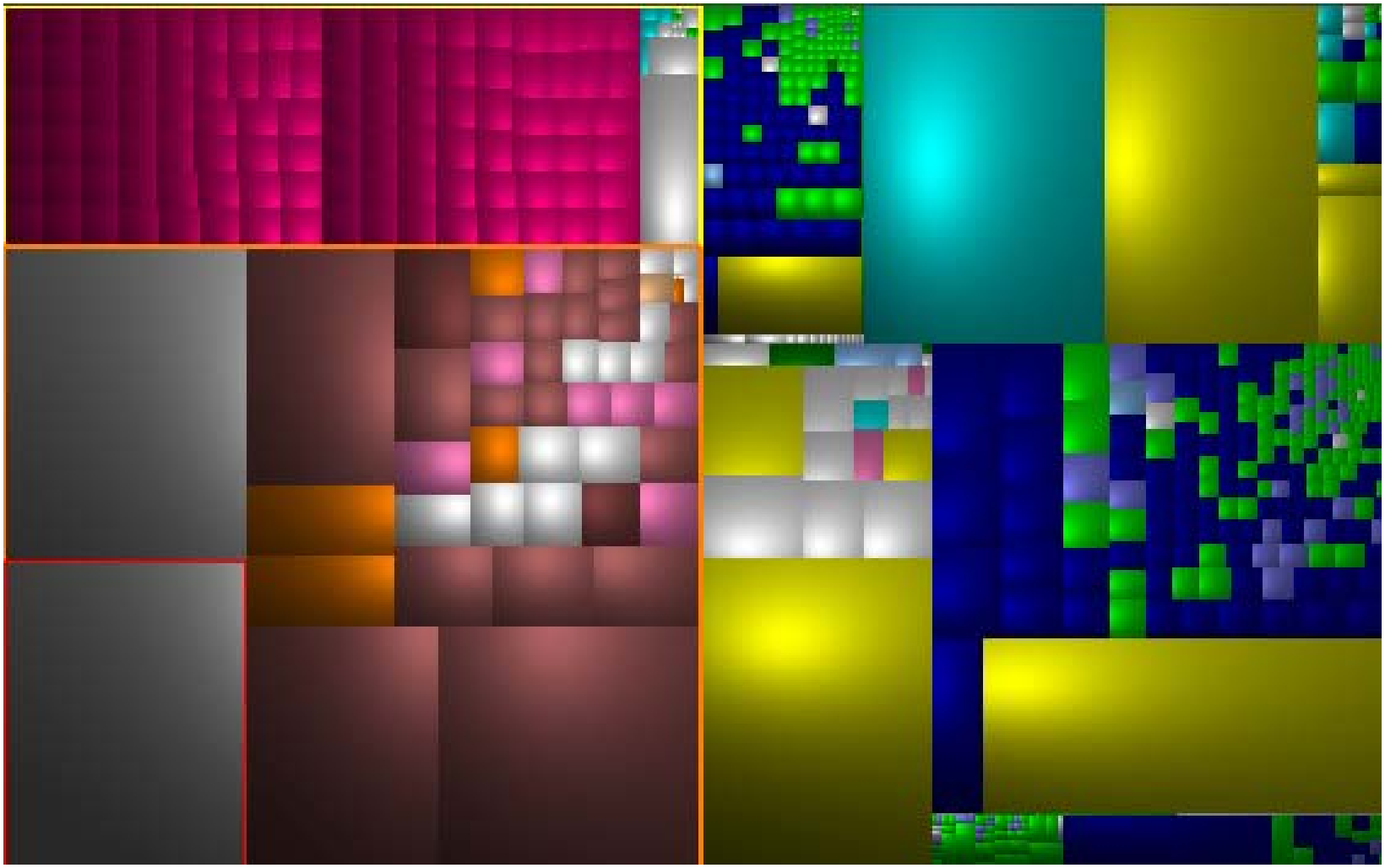
- **Treemap** method: visualize the tree structure that uses virtually every pixel of the display space to convey information.
- Every subtree is represented by a rectangle, that is further partitioned into smaller rectangles which correspond to its children.
- The position of the slicing lines determines the relative sizes of the child rectangles.
- For every child, repeat the slicing recursively, swapping the slicing direction from vertical to horizontal or conversely

Treemap



Treemap of the FFmpeg software

Tree Visualization



Treemapping via Balanced Partitioning

Computational Visual Media (Proceedings of CVM 2019)

Cong Feng[1] Minglun Gong[2] Oliver Deussen[3] Hui Huang[1*]
[1]Shenzhen University [2]Memorial University of Newfoundland [3]University of Konstanz

Figure 1. Our treemapping that relies on balanced partitioning is extremely easy to implement. One instance of our size-balanced partitioning (a) shows a better aspect ratio compared to other methods, e.g., (b) spiral placement, (c) strips, (d) slice and dice.

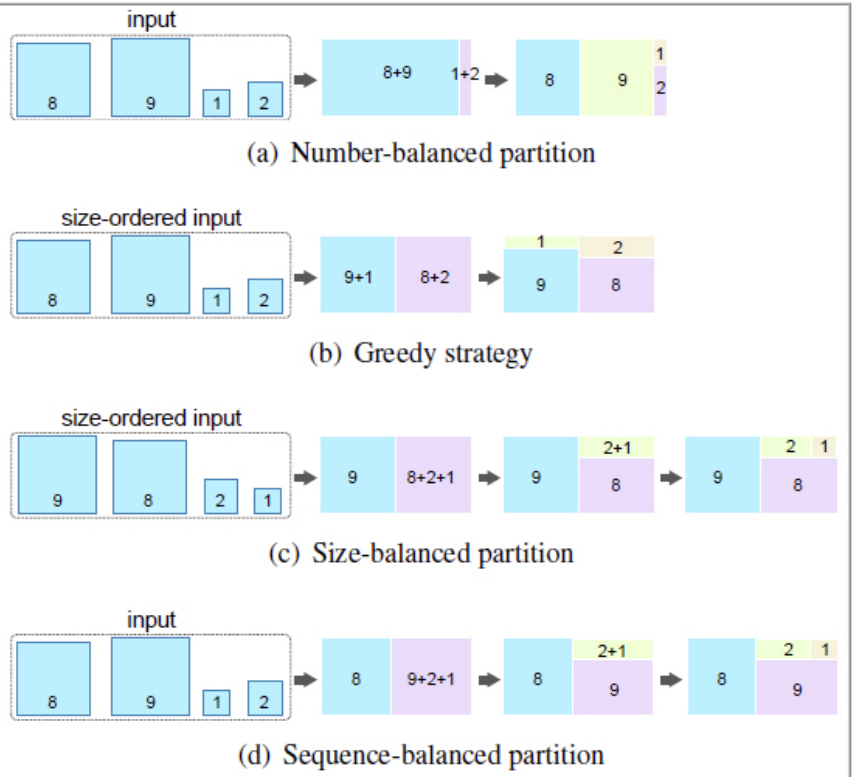
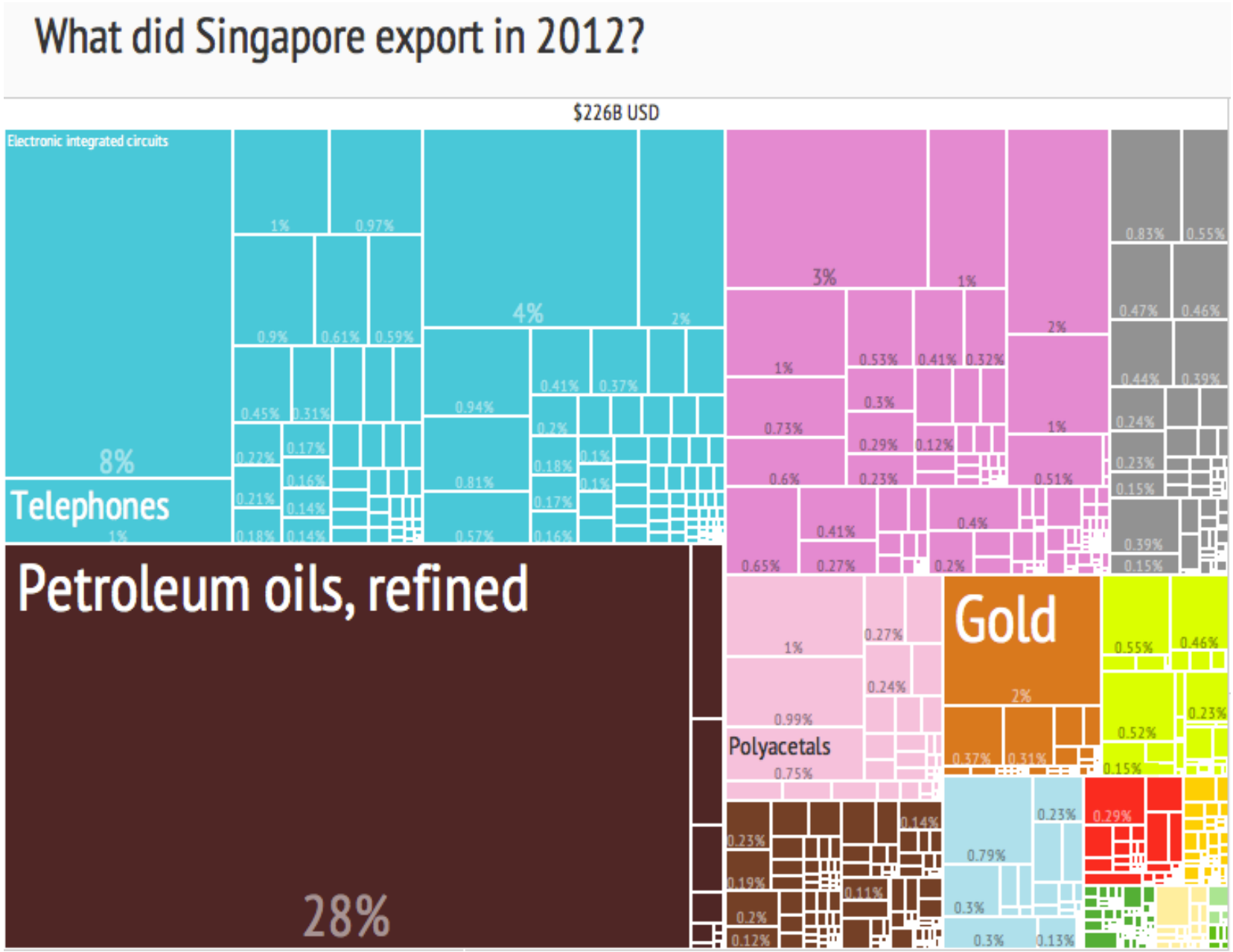


Figure 2. Comparison among different partitioning approaches.

Abstract

We present a treemapping method based on balanced partitioning that enables in one variant very good aspect ratios, in another good temporal coherence for dynamic data, and in the third a good compromise between these two aspects. To layout a treemap, we divide all children of a node into two groups. These groups are further divided until we reach groups of single elements. Then these groups are combined to form the rectangle representing the parent node. This process is performed for each layer of a given hierarchical dataset. In one variant of our partitioning we sort child elements first and built two as equal as possible sized groups from big and small elements (size-balanced partition), which achieves good aspect ratios for the rectangles, but less good temporal coherence. The second variant takes the sequence of children and creates the as equal as possible groups without sorting (sequence-based, good compromise between aspect ratio and temporal coherency). The third variant splits the children sets always into two groups of equal cardinality regardless of their size (number-balanced, worse aspect ratios but good temporal



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

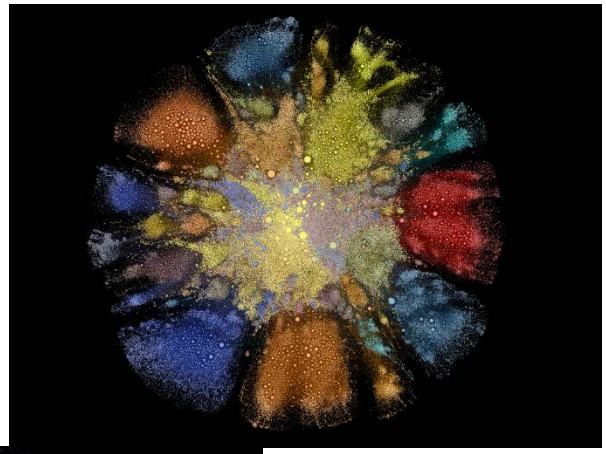
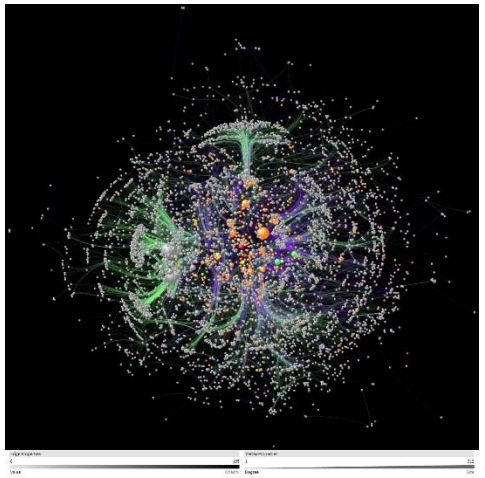
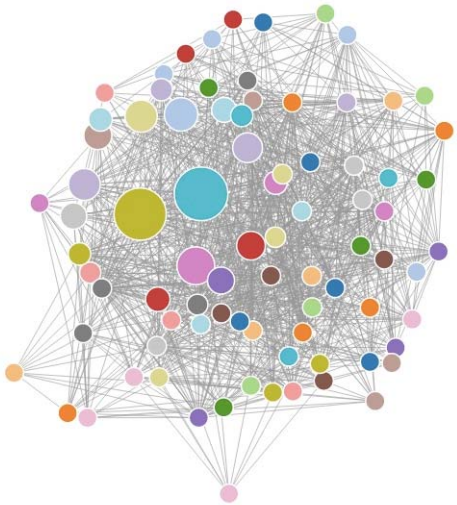
Additional Readings

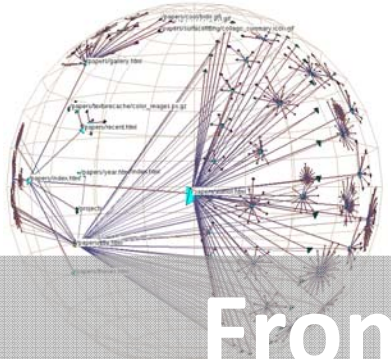
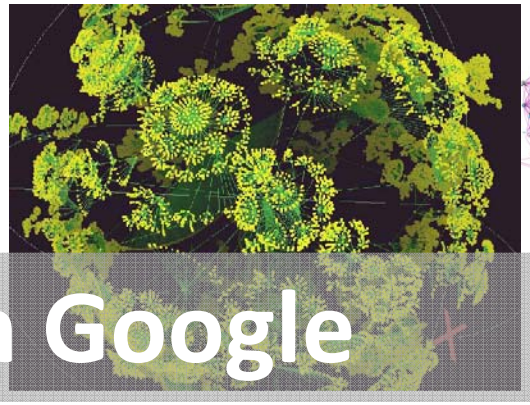
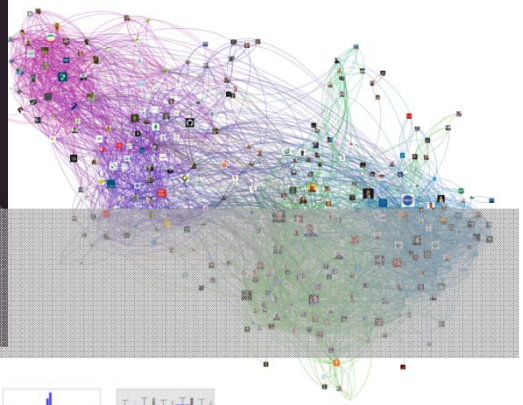
- Martin Graham and Jessie Kennedy. "A survey of multiple tree visualisation." *Information Visualization* 9.4 (2010): 235-252.

- Michael J. McGuffin. "Simple Algorithms for Network Visualization: A Tutorial". TsingHua Science and Technology, Vol. 17(4), August, 2012.

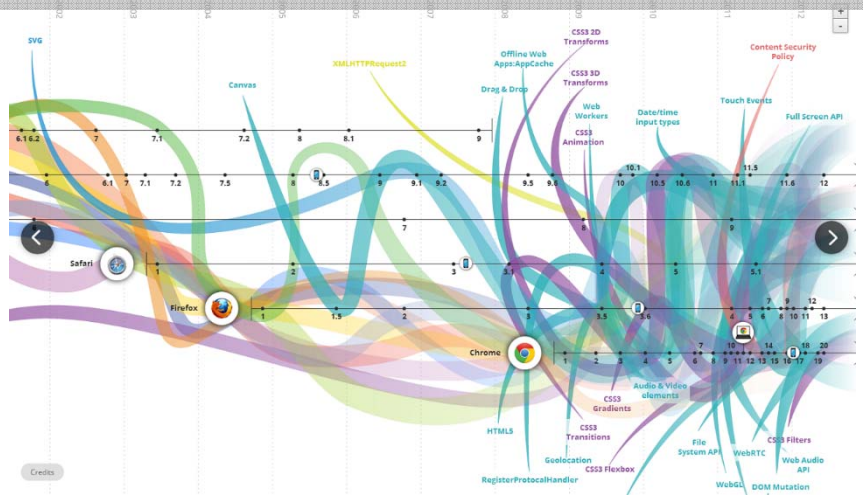
- T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J.J. van Wijk, J.-D. Fekete, and D.W. Fellner. "Visual analysis of large graphs: state-of-the-art and future research callenges", Computer Graphics Forum, Vol.30 (6): 1719-1749, 2011.

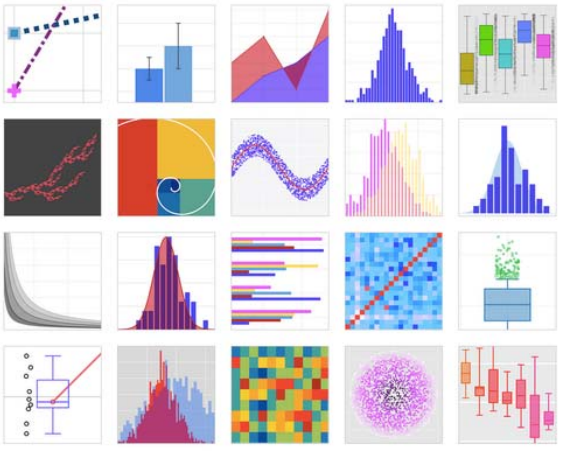
- Ivan Herman, Guy Melancon, M. Scott Marshall. "Graph Visualisation in Information Visualisation: a Survey". IEEE Transactions on Visualization and Computer Graphics, Vol.6(1), pp. 24-44, 2000.

From Google


SVG
Canvas
XMLHTTPRequest2
Offline Web Apps:AppCache
Drag & Drop
CSS3 2D Transforms
CSS3 3D Transforms
Web Workers
CSS3 Animation
Date/time input types
Content Security Policy
Touch Events
Full Screen API
Safari
Firefox
Chrome
HTML5
CSS3 Gradients
CSS3 Transitions
Geolocation
CSS3 Flexbox
RegisterProtocolHandler
Audio & Video elements
File System API
WebRTC
WebGL
CSS3 Filters
Web Audio API
DOM Mutation



Acknowledgment

- Thanks materials from
  - Dr. Jie Zhang