Basic Graph Layout Techniques

• Force-direct layout
• Arc-diagram
• Adjacent matrix
• Circular layout
Adjacent Matrix Representations

• An adjacent 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.
Adjacency matrix visualizations of a 43-node, 80-edge network. Left: with a random ordering of rows and columns.
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.
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.
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-direct layout
• Arc-diagram
• Adjacent 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).
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 \(\text{angle}(p) = p \times 2 \times \pi / N\) 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

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


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 leaf.
Tree Visualization

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

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

Flow map

Colorado migration data from 1995 to 2000

Hierarchical cluster tree of US murder rates

dendrogram
Dendrogram good at showing hierarchical clustering results
Treevis. Net by Hans-Jorg Schulz
Tree Visualization

- **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 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.
Tree Visualization

Treemap of the FFmpeg software
Tree Visualization
Treemapping via Balanced Partitioning

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 coherence).
Additional Readings


Acknowledgment

• Thanks materials from
  – Dr. Jie Zhang