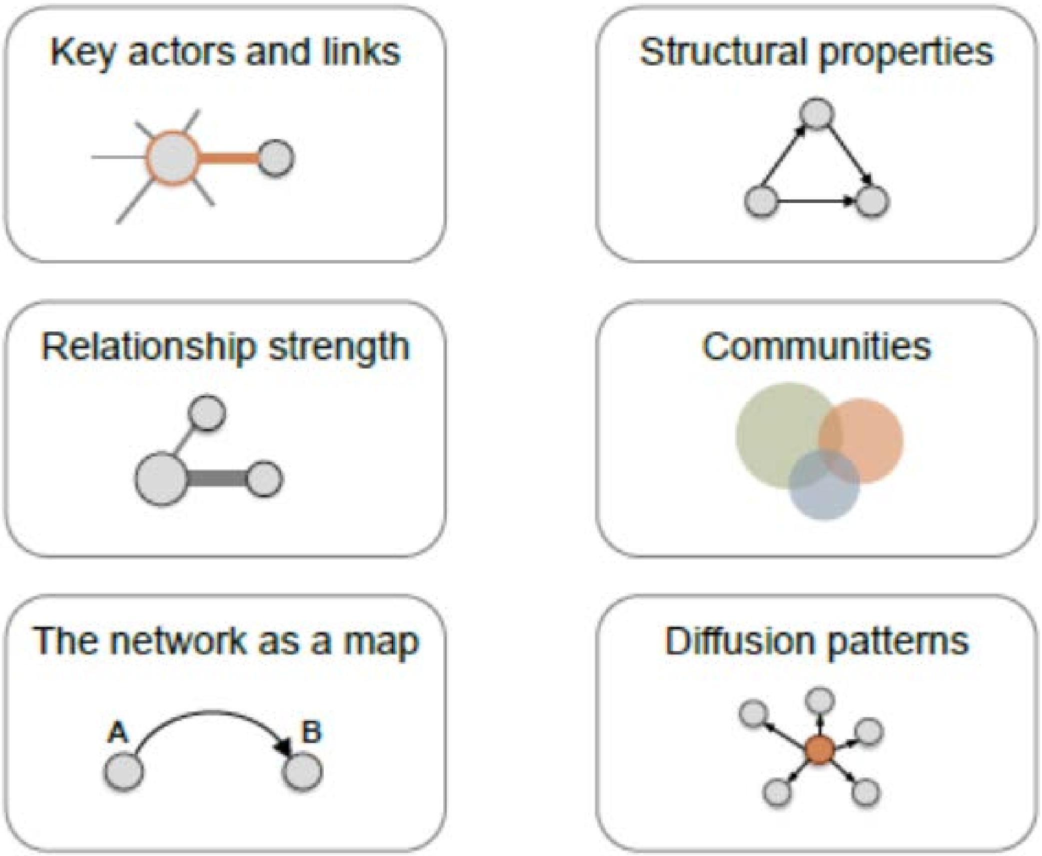
Graph Visualization

Ng Yen Kaow

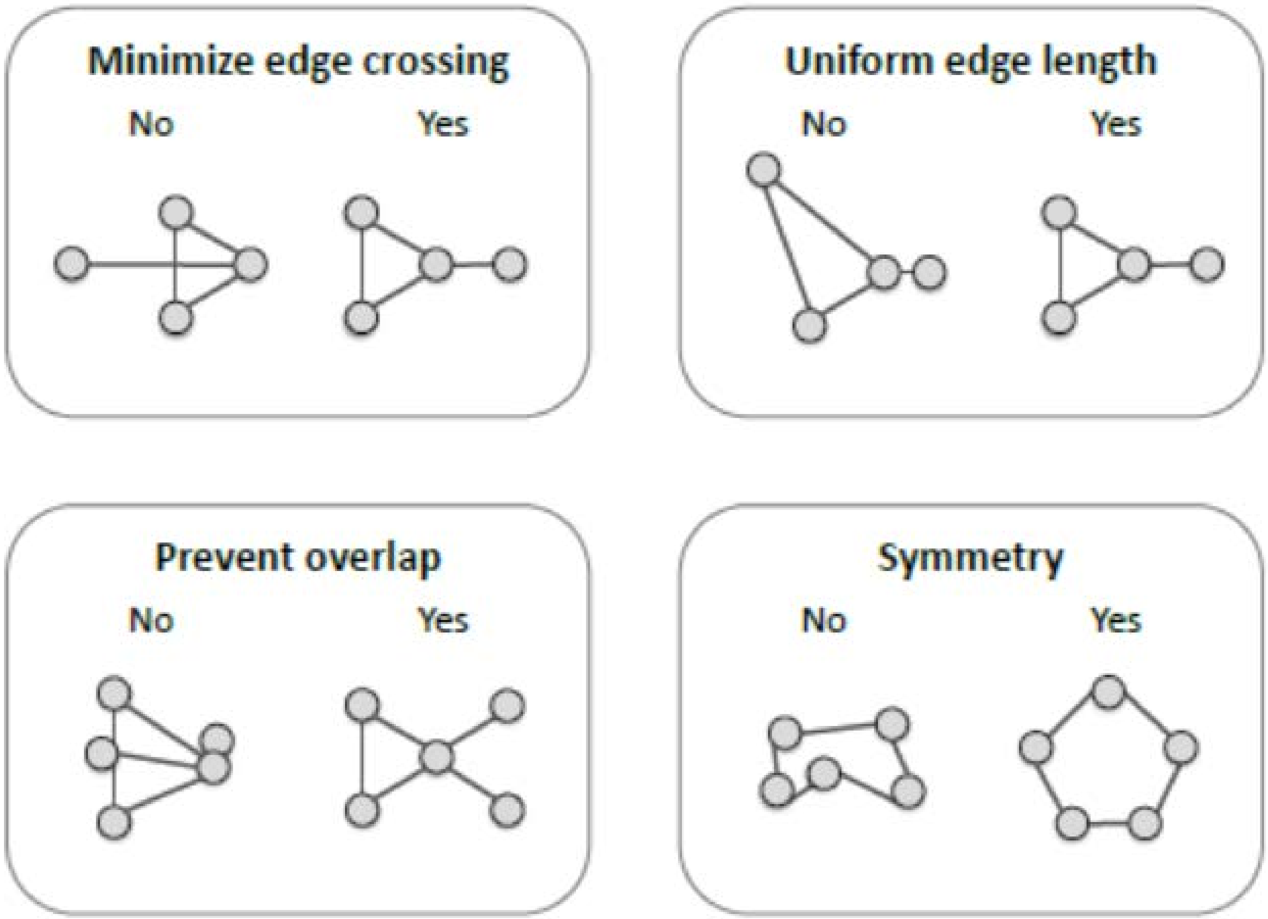
Network visualization

- From "Network Visualization with R" (kateto.net/polnet2015)
  - The goals of visualization are



Network visualization

- From “Network Visualization with R” (kateto.net/polnet2015)
  - The aesthetic issues of visualization

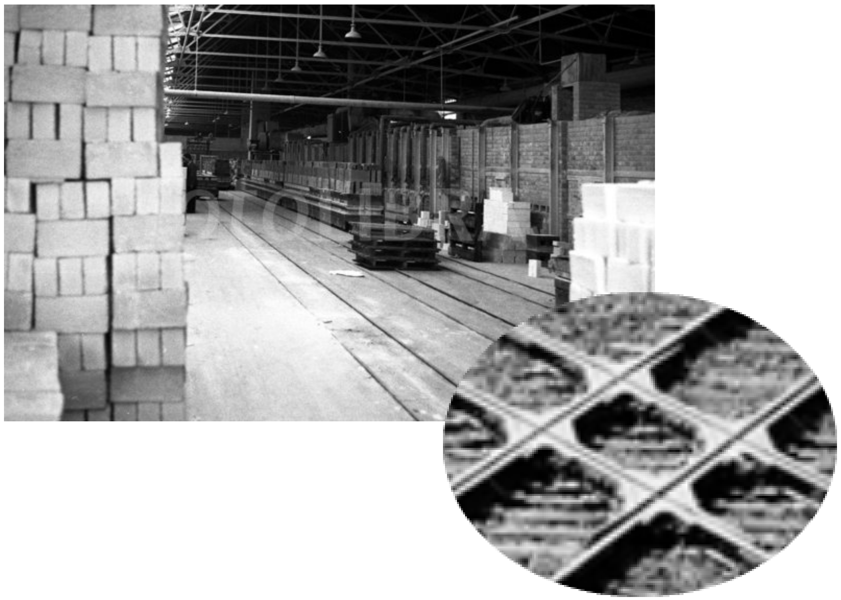


Aesthetics

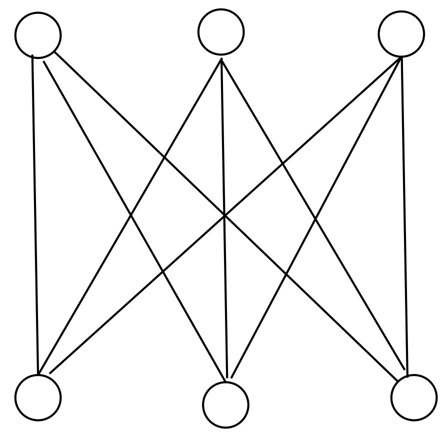
- Graph drawing is not simple
  - Turán's brick factory problem
    - Crossing number
  - Clustered planarity
  - There are conferences dedicated to these problems
    - International Symposium on Graph Drawing & Network Visualization

Turán's brick factory problem

- Origin: Hungarian mathematician Pál Turán



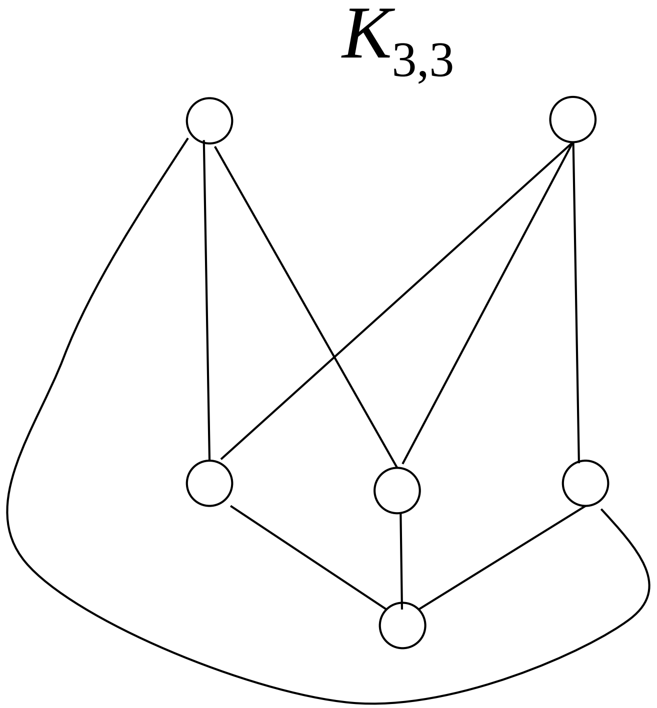
$K_{3,3}$



- How many crossings can there be in a complete bipartite graph?

Turán's brick factory problem

- For $K_{3,3}$, 1

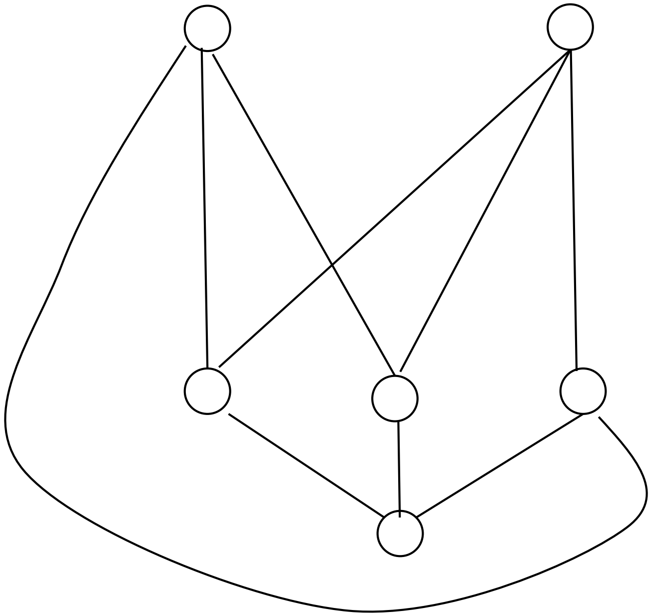


Turán's brick factory problem

- Zarankiewicz’s conjecture
  - For a complete bipartite graph $K_{m,n}$, at most $\left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{m}{2} \right\rfloor \left\lfloor \frac{m-1}{2} \right\rfloor$

$$K_{3,3} : \left\lfloor \frac{3}{2} \right\rfloor \left\lfloor \frac{2}{2} \right\rfloor \left\lfloor \frac{3}{2} \right\rfloor \left\lfloor \frac{2}{2} \right\rfloor = 1$$

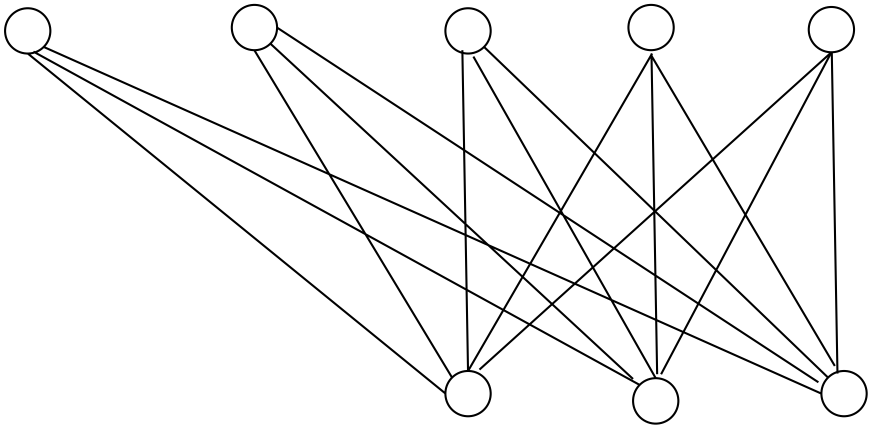
$K_{3,3}$



Turán's brick factory problem

- What about $K_{5,3}$

$$K_{5,3}:\left\lfloor\frac{5}{2}\right\rfloor\left\lfloor\frac{4}{2}\right\rfloor\left\lfloor\frac{3}{2}\right\rfloor\left\lfloor\frac{2}{2}\right\rfloor=4$$



- See if you can get 4 crossing edges

Turán's brick factory problem

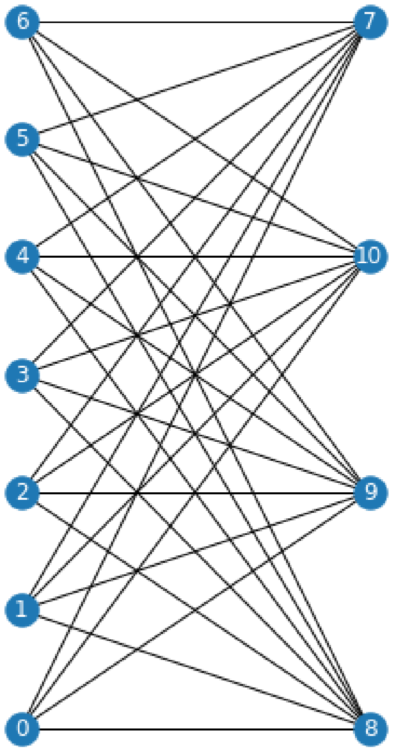
- Try $K_{7,4}$

$$K_{7,4}:\left\lfloor\frac{7}{2}\right\rfloor\left\lfloor\frac{6}{2}\right\rfloor\left\lfloor\frac{4}{2}\right\rfloor\left\lfloor\frac{3}{2}\right\rfloor=18$$

Graph layout

- R allows many graph layouts
- One of these is the bipartite layout
  - Show $K_{7,4}$ using the bipartite layout

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.complete_bipartite_graph(7, 4)

partition = nx.get_node_attributes(G, 'bipartite')

nodes = [i for i in partition.keys() if partition[i]==0]

nx.draw(G, pos=nx.bipartite_layout(G, nodes))

plt.show()
```



Other NetworkX layouts

`bipartite_layout`	Position nodes in two straight lines
`circular_layout`	Position nodes on a circle
`kamada_kawai_layout`	Position nodes using Kamada-Kawai path-length cost-function
`planar_layout`	Position nodes without edge intersections
`random_layout`	Position nodes uniformly at random in the unit square
`shell_layout`	Position nodes in concentric circles
`spring_layout`	Position nodes using Fruchterman-Reingold force-directed algorithm
`spectral_layout`	Position nodes using the eigenvectors of the graph Laplacian
`spiral_layout`	Position nodes in a spiral layout
`multipartite_layout`	Position nodes in layers of straight lines

Layered/hierarchical graph drawing

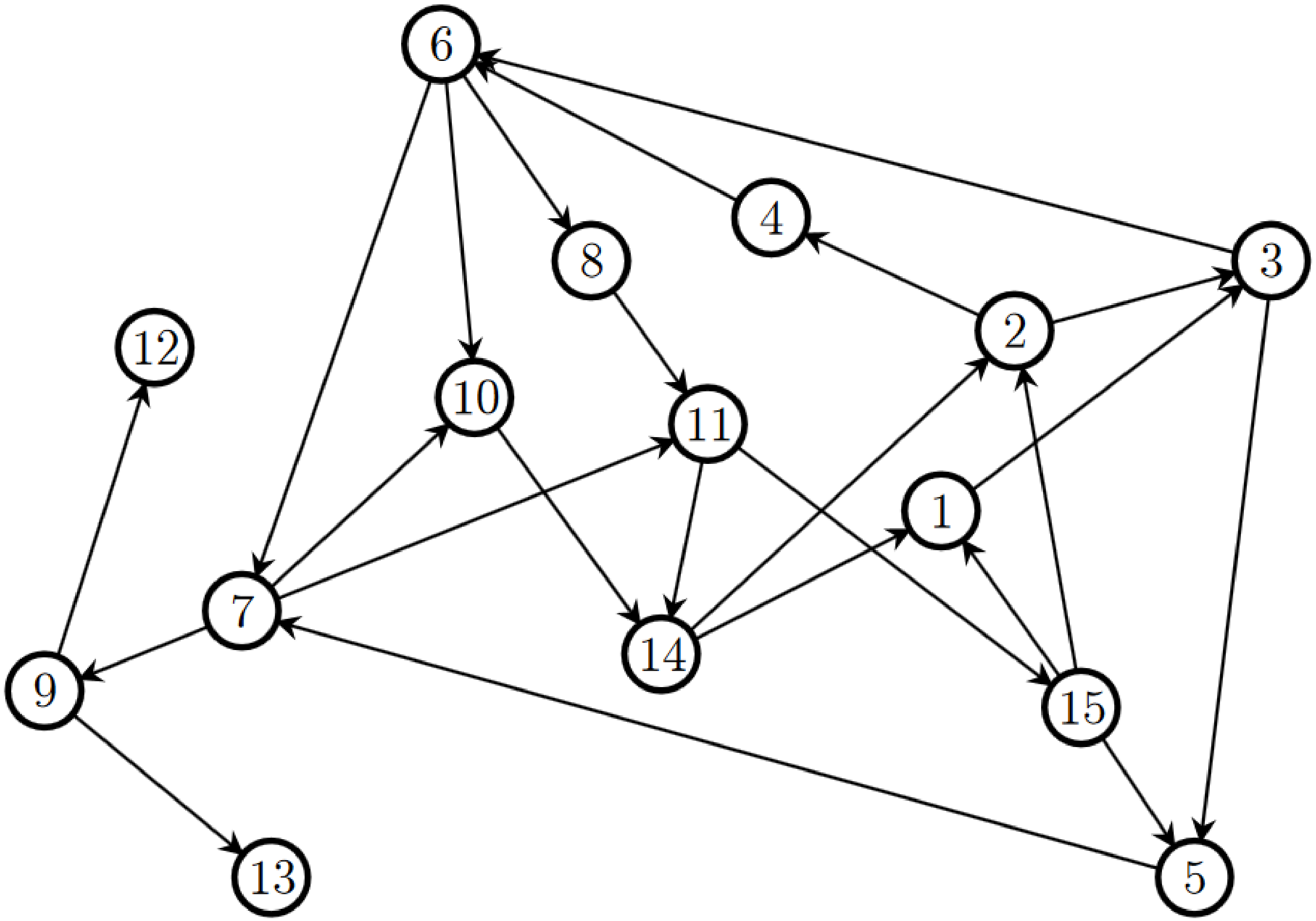
- A directed graph can be represented as a "hierarchy"
  - A cycle-free digraph (roughly)
  - Process flow diagram
  - Function call diagram
- An example is the Sugiyama layout

Sugiyama layout

- Follows the following principles as much as possible
  - Edges should point in a uniform direction
  - Short, straight edges
  - Uniformly distributed nodes
  - Minimize edge crossing

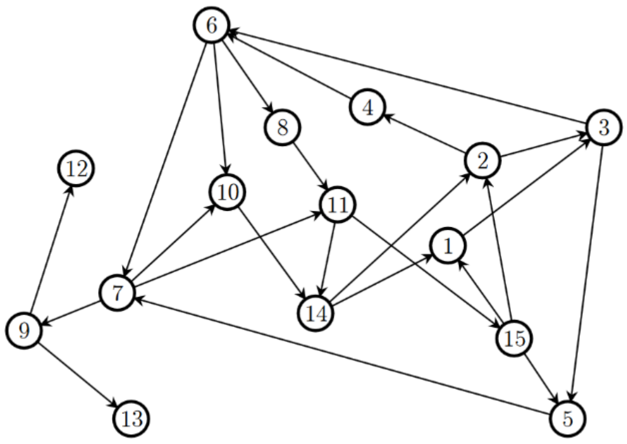
Sugiyama layout

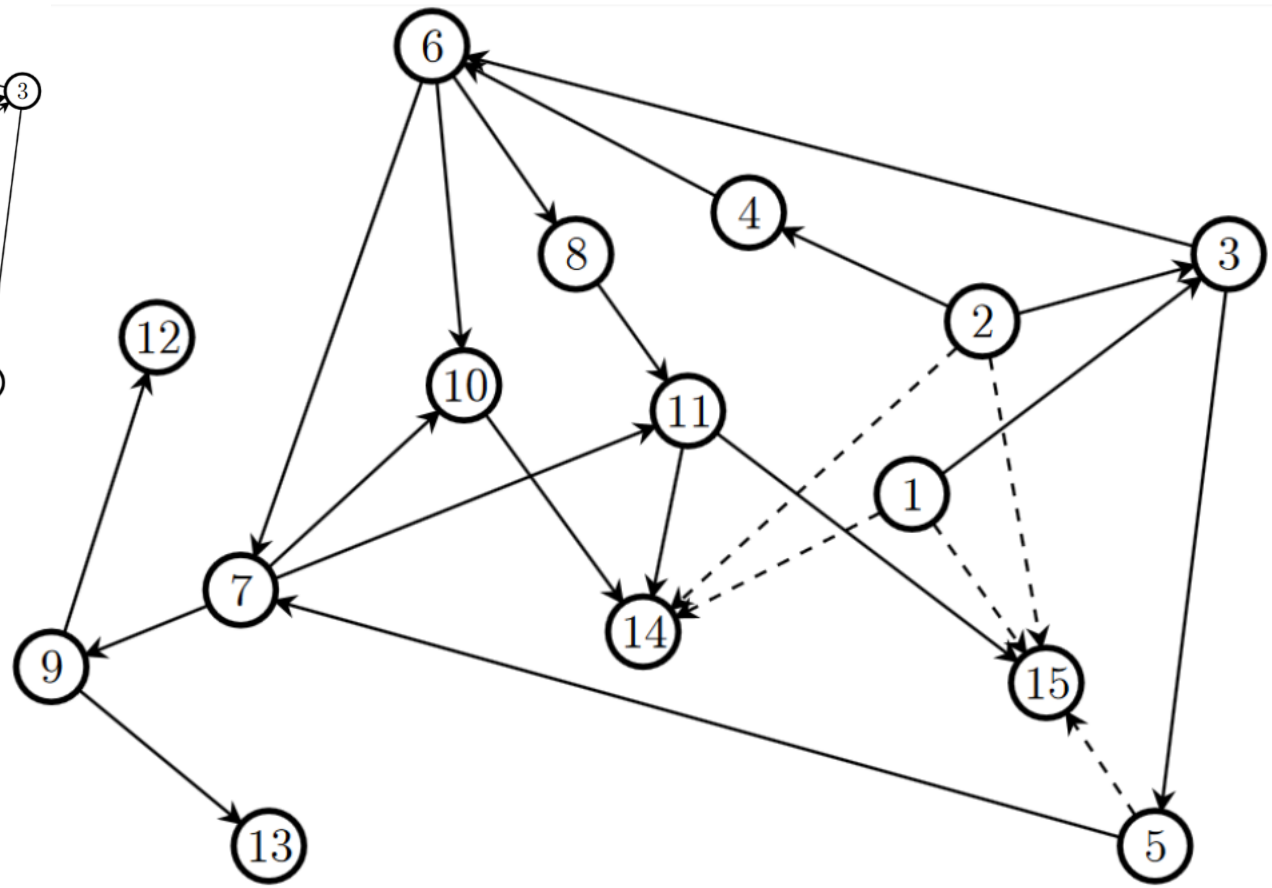
- Example: Convert the following graph to Sugiyama layout



Sugiyama layout

1. Remove cycles

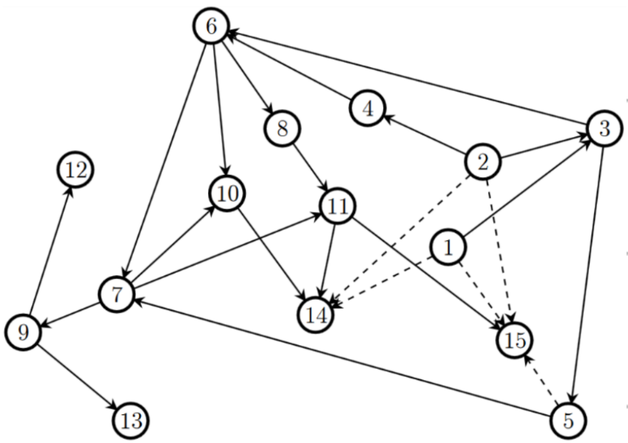


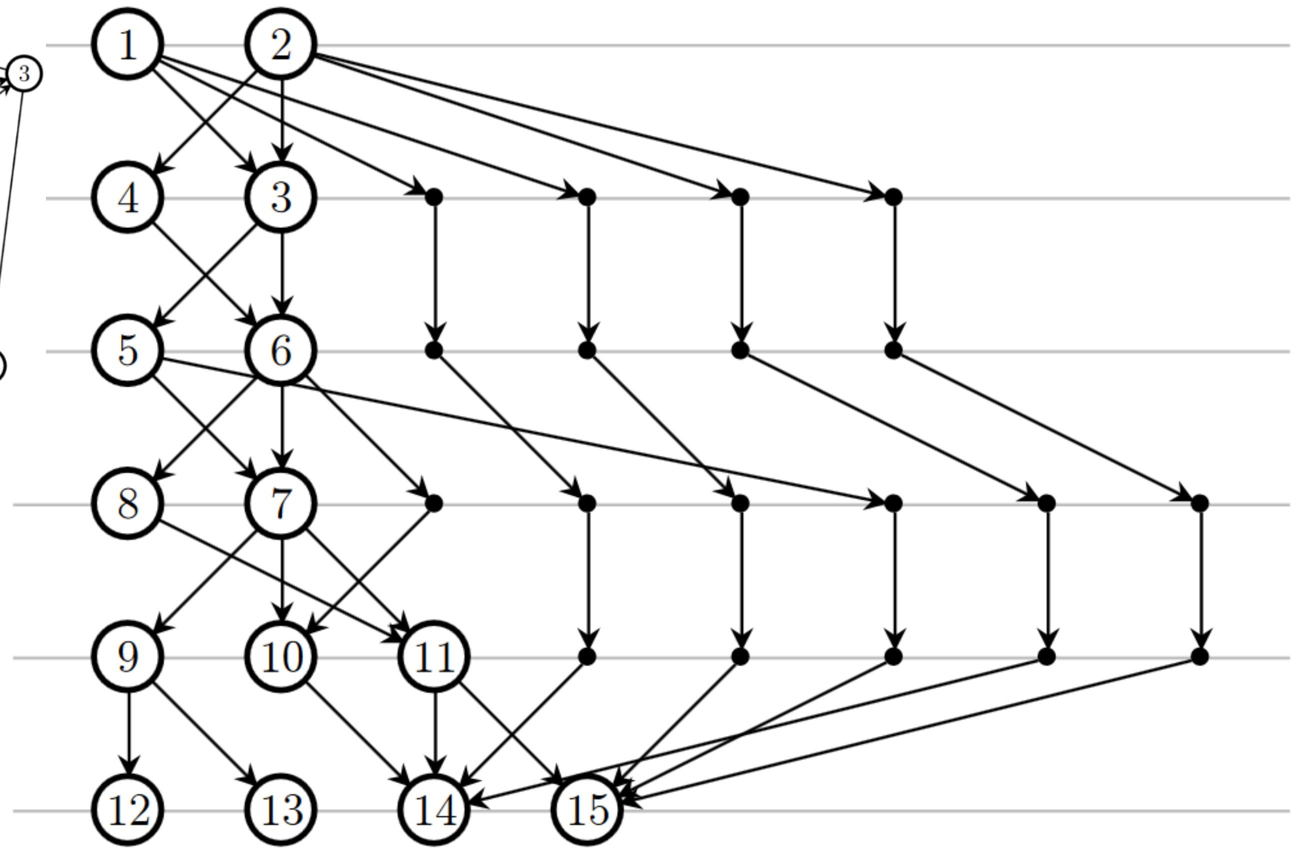


From “Handbook of Graph Drawing and Visualization”, Roberto Tamassia

Sugiyama layout

2. Arrange nodes in levels

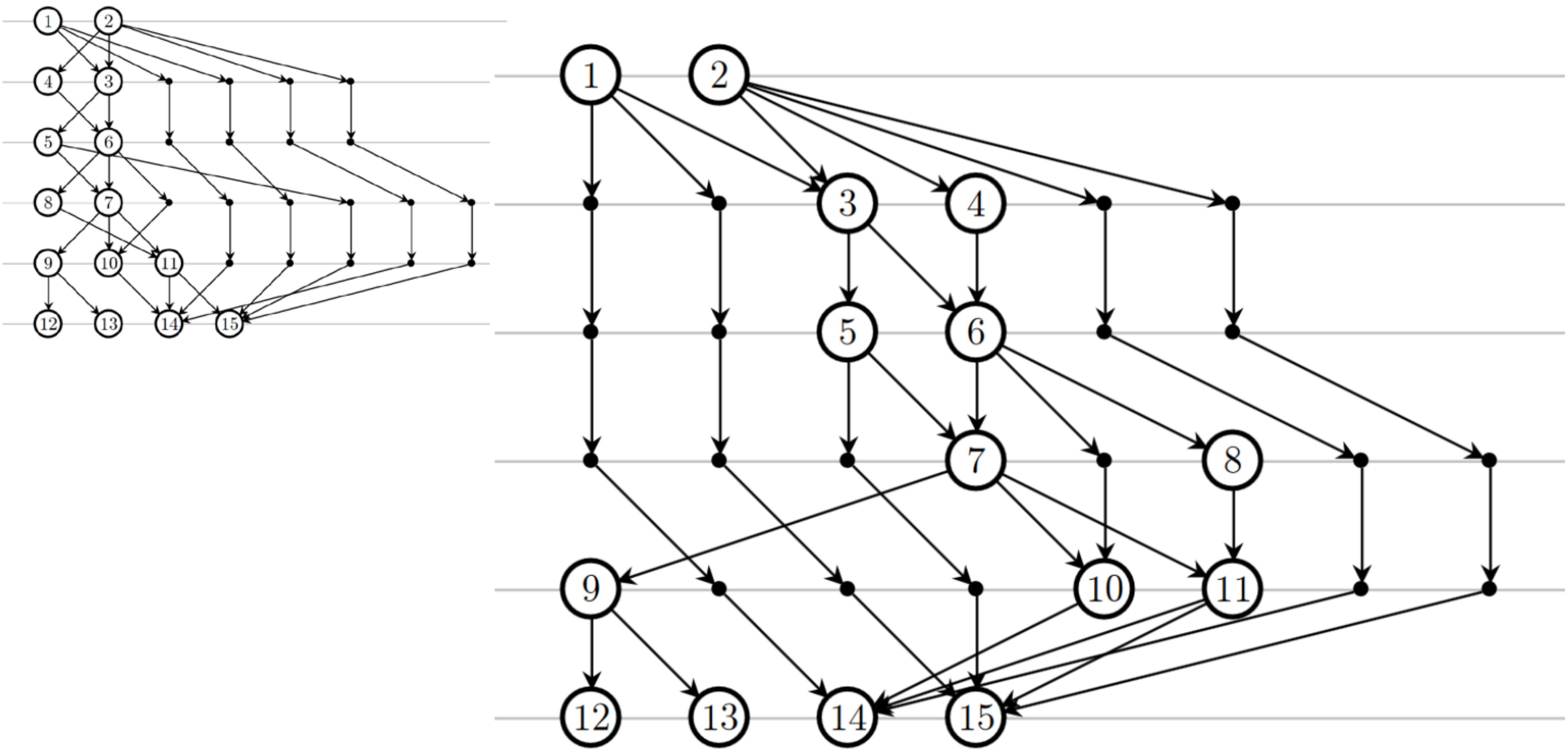




From “Handbook of Graph Drawing and Visualization”, Roberto Tamassia

Sugiyama layout

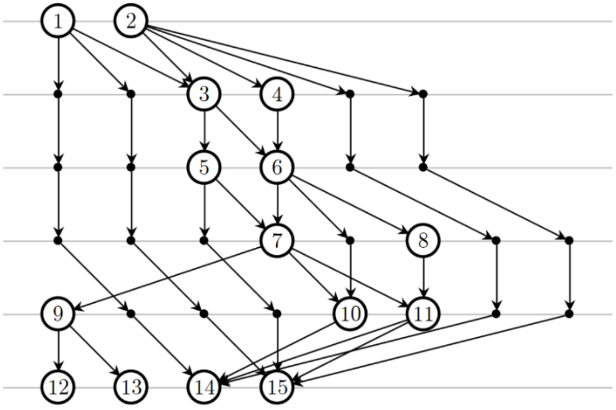
3. Reduce crossing edges

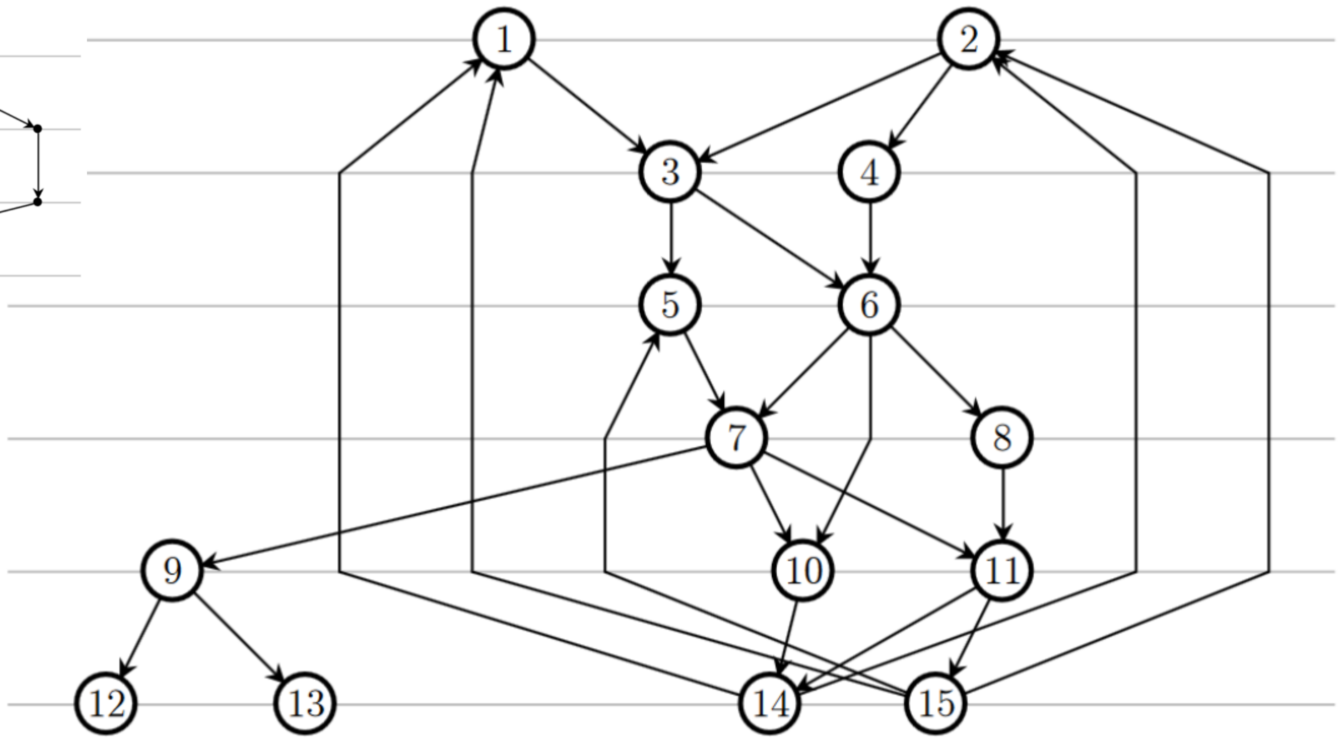


From “Handbook of Graph Drawing and Visualization”, Roberto Tamassia

Sugiyama layout

4. Straighten edges





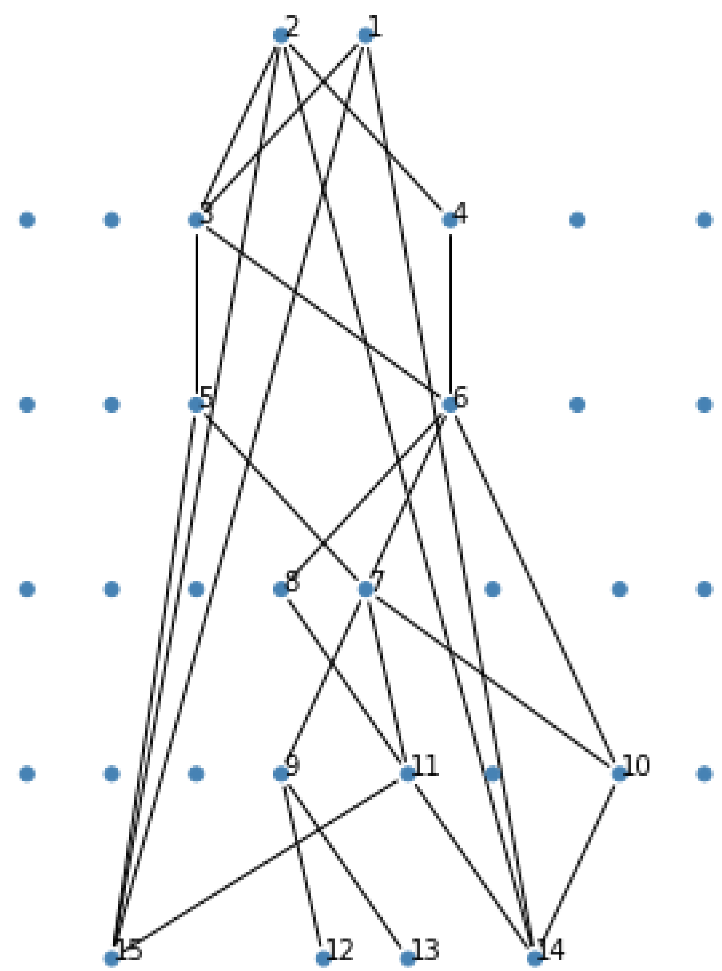
From “Handbook of Graph Drawing and Visualization”, Roberto Tamassia

Sugiyama layout

- How to do it in Python (igraph)

Regrettably this layout is not available in NetworkX

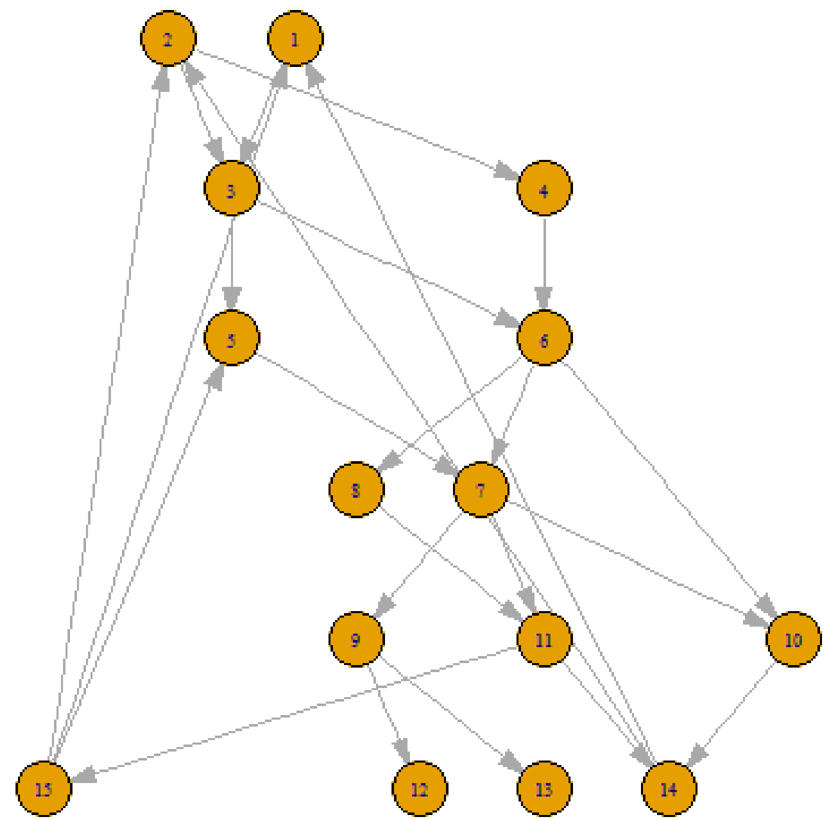
```
import igraph as ig

edges = [[x, y] for (x, y) in
np.genfromtxt('sugiyama.csv',
delimiter=',', dtype=np.integer)]

G = ig.Graph(edges=edges)

layers = [0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 4,
5, 5, 5, 5]

layout = G.layout_sugiyama(layers=layers)

_, ax = plt.subplots()

ig.plot(G, layout=layout,
vertex_label=list(range(1,16)), target=ax)

ax.invert_yaxis()
```



Sugiyama layout

- How to do it in R (igraph)

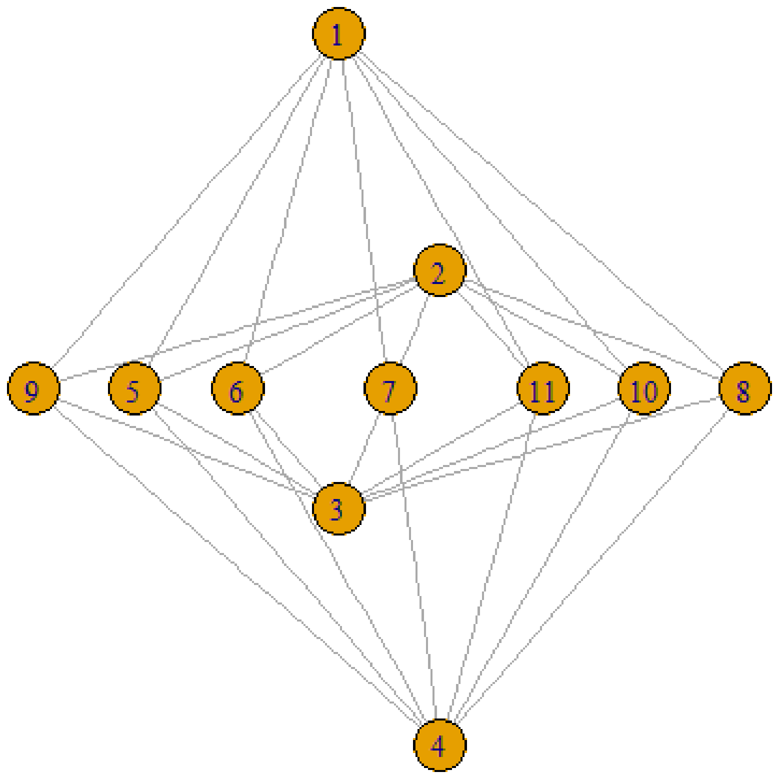
```
# Construct graph
G <-
graph_from_data_frame(sugiyama)

# Define layers
layers <- list(c(1, 2), c(4, 3),
c(5,6), c(8,7), c(9,10,11),
c(12,13,14,15))

# Perform layout
sugilayout <-
layout_with_sugiyama(G,
layers=apply(sapply(layers,
function(x) V(G)$name %in% x), 1,
which))

# Show Sugiyama plot
plot(G, layout=sugilayout$layout,
vertex.label.cex=0.9,
edge.arrow.size=0.1)
```



Same as the Python output

$K_{7,4}$ revisited

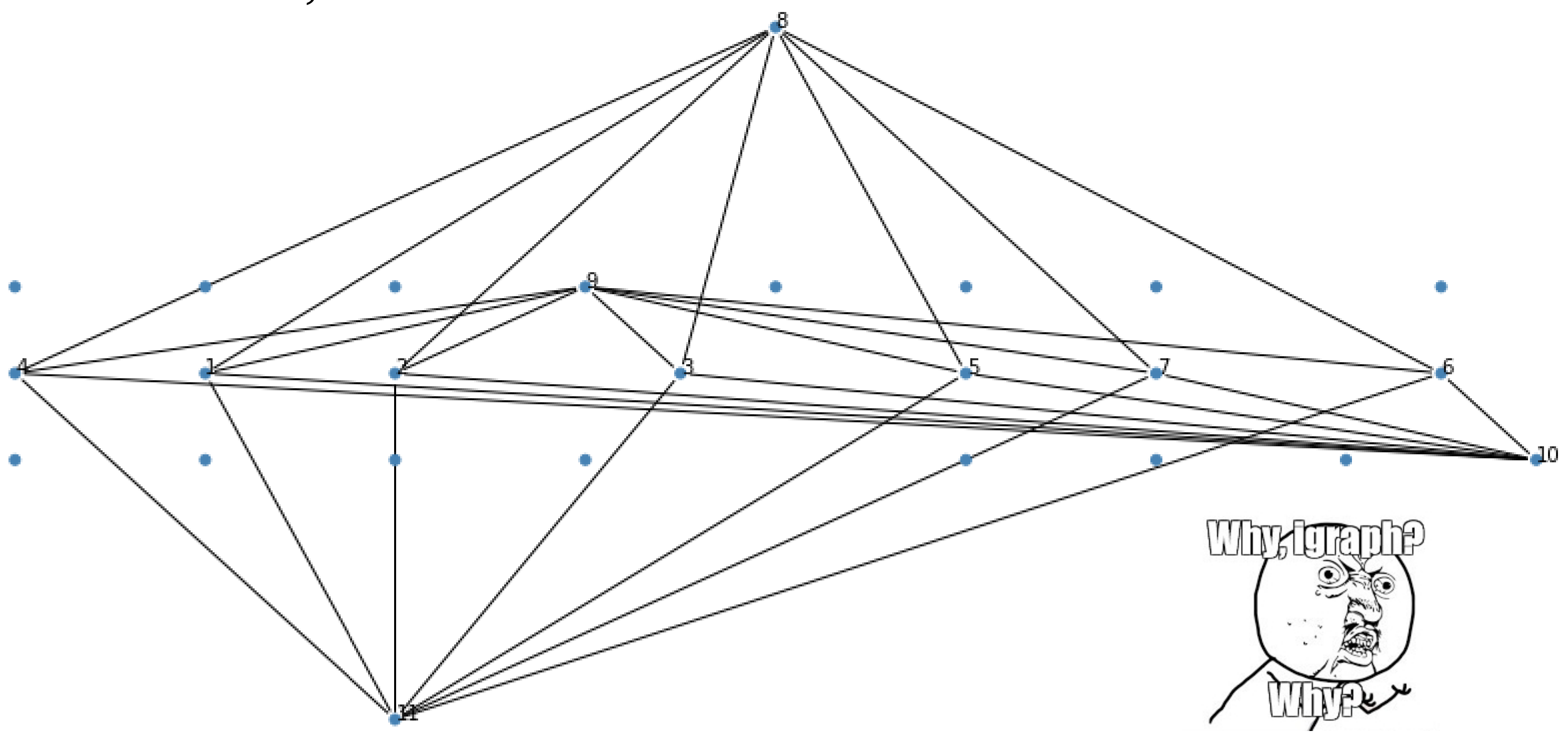
- Show $K_{7,4}$ using Sugiyama layout (in R)




```
G <- graph_from_data_frame(K74, directed=FALSE)
layers <- list(c(1), c(), c(2), c(5,6,7,8,9,10,11), c(3), c(), c(4))
sugilayout <-  layout_with_sugiyama(G, layers=apply(sapply(layers, function(x) V(G)$name
%in% x), 1, which))
plot(G, layout=sugilayout$layout, vertex.label.cex=0.9)
```

$K_{7,4}$ revisited

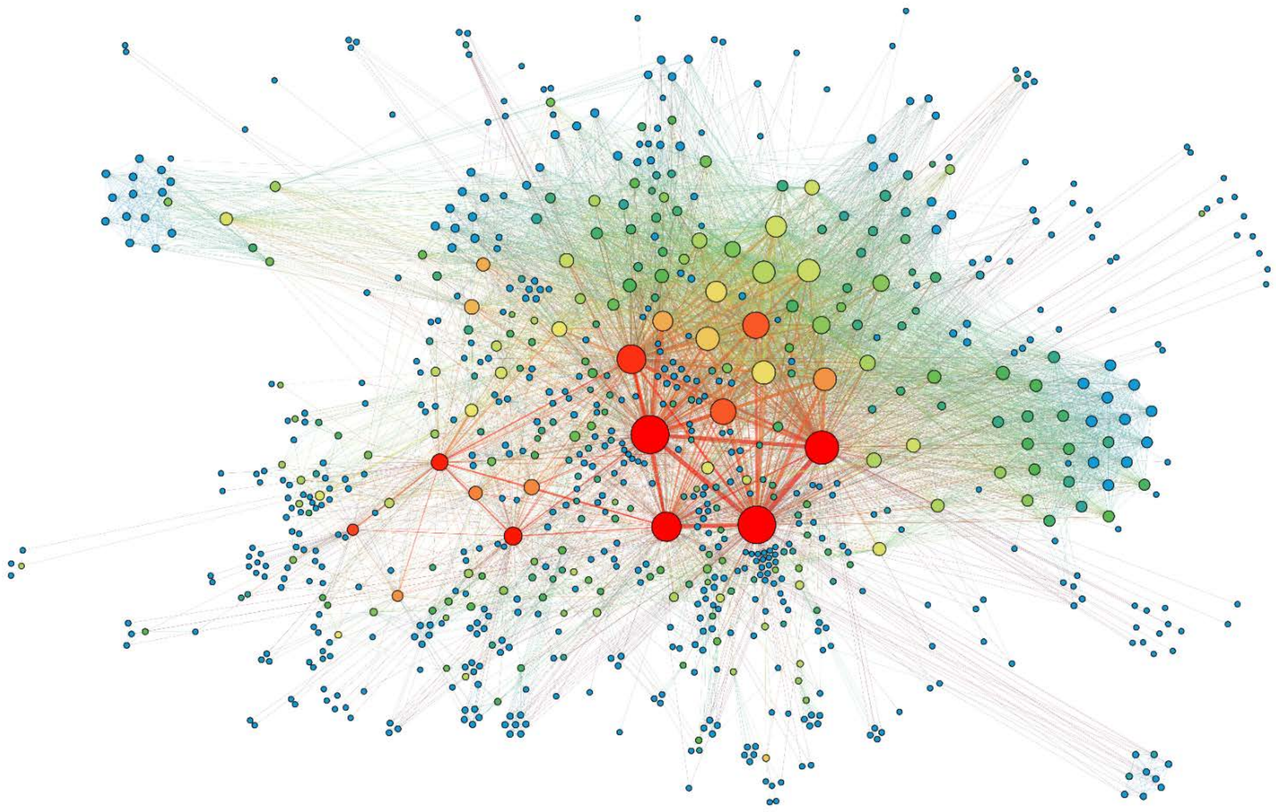
- Show $K_{7,4}$ using Sugiyama layout (Python)




```
G = nx.complete_bipartite_graph(7, 4)
edges = list(G.edges)
G = ig.Graph(edges=edges)
layers = [4, 4, 4, 4, 4, 4, 4, 0, 3, 5, 8]
layout = G.layout_sugiyama(layers=layers)
_, ax = plt.subplots()
ig.plot(G, layout=layout, vertex_label=list(range(1,12)), target=ax)
```

Force-directed graph layout

- Commonly available, e.g `nx.spring_layout`



- PalsGraph adds community information in applying the forces

Force-directed graph layout

- PalsGraph implementation
  - Each pair of vertexes i and j repel each other with a force F_r/d_{ij}
  - Each pair of vertexes i and j that are connected by an edge attracts each other with a force F_a/d_{ij}
  - (The centers of) two communities i and j repel each other with a force F_{rc}/d_{ij}
  - (The centers of) each community c attracts every vertex v in community with force $F_{ac}d_{cv}$
  - The center of the graph attracts vertices

Software for Visualization

Ng Yen Kaow

graphviz

- A set of command-line tools

`dot`	Plots hierarchical layouts of directed graphs
`neato`	Plots spring model layouts
`twopi`	Plots radial layouts
… and a few more	

- Grandfather of graph-drawing tools
- File format (“dot”) widely supported
- Used by other software
  - Source codes visualization – doxygen
    - https://www.doxygen.nl/
  - Python modules, e.g. graph-tool
- As a Python module
  - https://github.com/pygraphviz/pygraphviz

Python modules

- Most popular modules are
  - NetworkX (https://networkx.org/)
  - igraph (http://igraph.org/)
    - For R or Python or C
    - Actually, mostly made for R until recently
  - graph-tool (https://graph-tool.skewed.de/)
    - For Python only
  - Networkit (https://networkit.github.io/)
    - For Python only
- In general, graph-tool is the fastest and NetworkX is the slowest (pure Python)
  - However, NetworkX is likely the most complete, well-documented, and easy to use

Gephi

- GUI tool (but very powerful)
- https://gephi.org/
- File format (gexf) widely supported
- Export graphviz file from Python

```
def export_edge_list(dismat, labels=None, filename="edges.csv", delim=",", header=True):
    f = open(savedir + filename, 'w')
    if header:
        f.write("Source,Target\n")
    loc = np.where(dismat > 0)
    for (i, j) in [(i, j) for (i, j) in zip(loc[0], loc[1]) if i < j]:
        if labels == None:
            f.write(str(i) + delim + str(j) + "\n")
        else:
            f.write("\"" + labels[i] + "\"" + delim + "\"" + labels[j] + "\"\n")
    f.close()
```

- Allows vertices to be colored by their communities
  - See “`gephi_communities.pdf`”