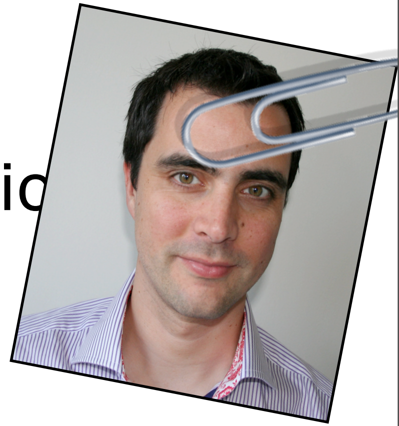
Network Visualization

Everything You Always Wanted to Know (But Were Afraid to Ask)

Talk at Bocoup, Boston MA. Nov 1st 2011

Short bio

- Computer Science PhD in Eindhoven, NL (2005)
- Main research expertise
  - Information Visualization
  - Network/Graph Analysis and Visualizatio
  - Analytics interfaces
  - Collaborative work



- Formerly in IBM's Visual Communication Lab in Cambridge, MA (2006-2009)
- Embedded in IBM/ILOG R&D team in Paris since (2009-current)

Why **Networks**?

- Network data (relationships between items) occurs in many contexts
  - Social networks
  - Gene activation networks
  - Financial (fraud) networks
  - Source code relationships
  - Planning, scheduling
  - Food networks

  - <Your favorite usecase here>

Why Network **Visualization**?

- Not many other options for humans to reason about network structure…

A → B

C → A

D → B

C → D

C → E

E → D

Why Network **Visualization**?

- Not many other options for humans to reason about network structure…

A → B

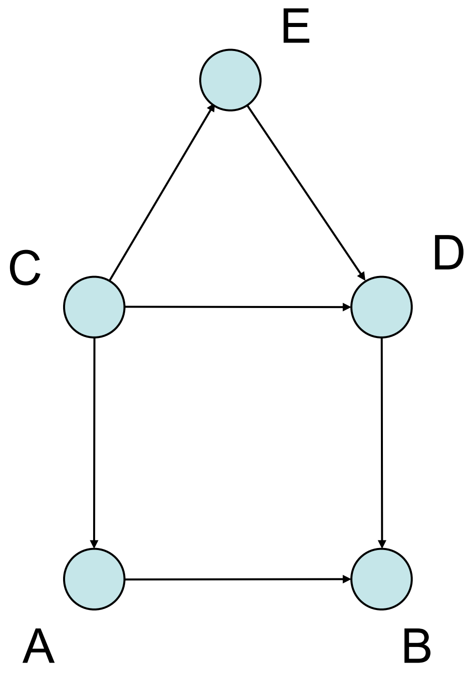
C → A

D → B

C → D

C → E

E → D



Why Network **Visualization**?

- Not many other options for humans to reason about network structure…

A → B

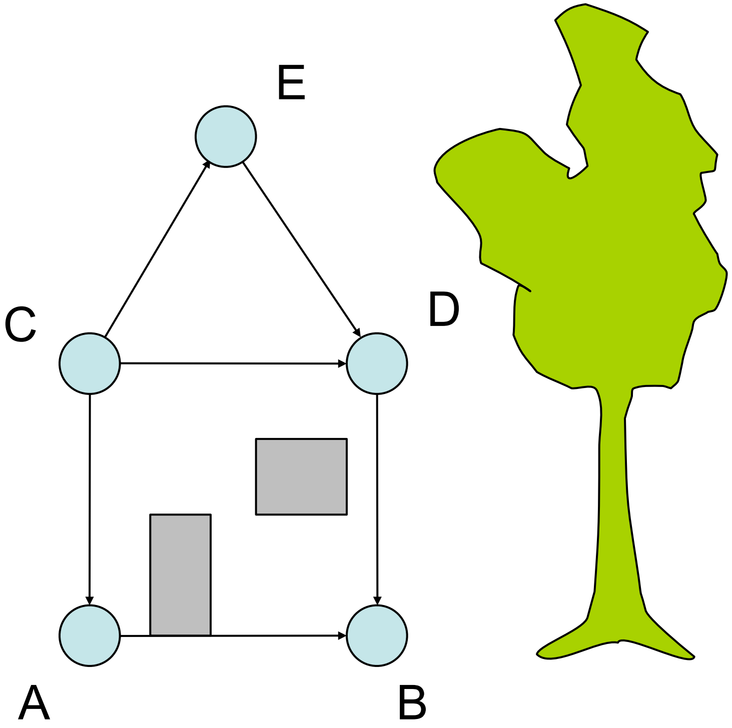
C → A

D → B

C → D

C → E

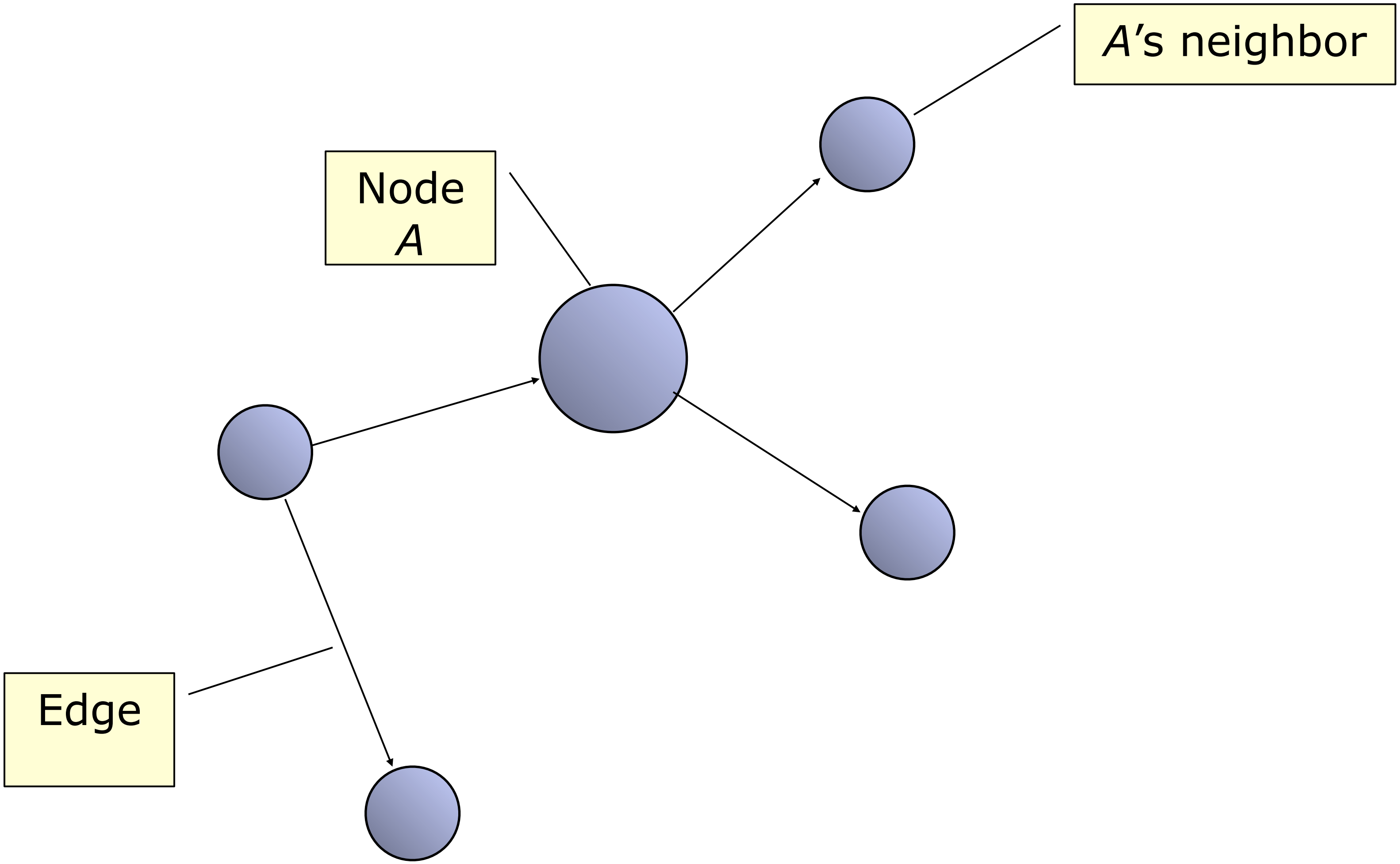
E → D



Some terminology

- The mathematical term for network is *graph*.
- A graph consists of a set of *nodes* and a set of *edges*.
- The set of nodes represent the related items.
- The set of edges represent the relationships between 2 nodes.

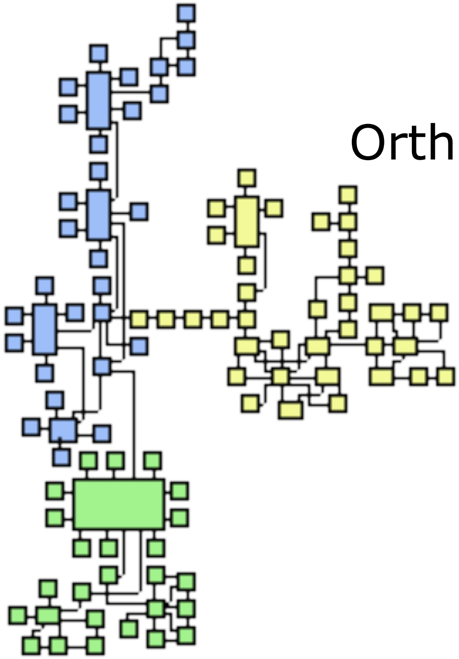
Or, graphically:



How to draw a graph?

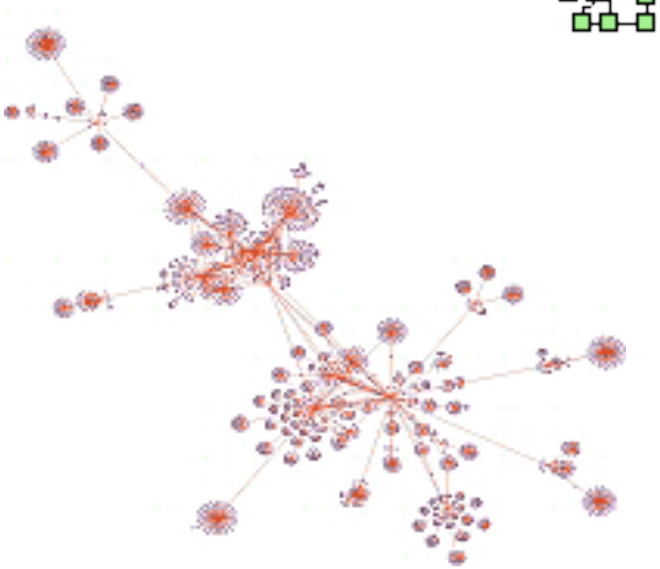
- For each node find a point in 2D (or 3D) space such that the structure is “optimally” shown.
- Domain of *Graph Drawing*
- How to create the “best” possible node link layout of a given network
- “best?”

What is ‘best’?

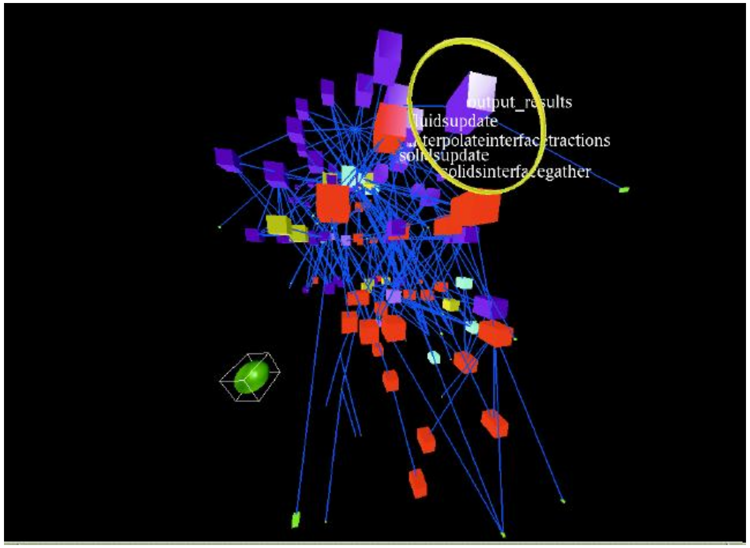


Orthogonal?

“Organic”?

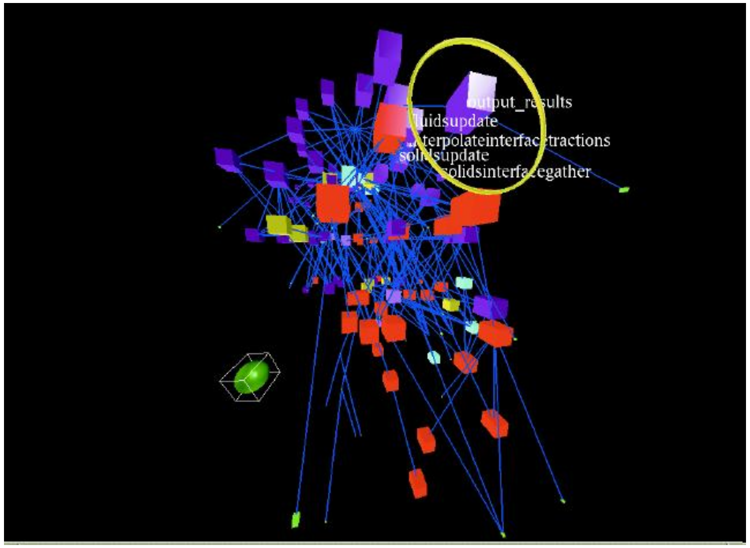


3D?



Ok, Since I mentioned 3D

- Flashy and sexy
- But
  - Need to project 3D layout to 2D screen
  - Introduces extra edge and node overlaps
  - Large navigation overhead
- Don’t, unless you have a problem domain that lends itself to 3D (i.e. molecular models)



In this talk

- High level overview of area
- Mostly practical tips from the trenches
  - Tips for picking the right layout.
  - Tips for implementing some layouts.
  - Tips for scaling to larger graphs.

Picking the right layout

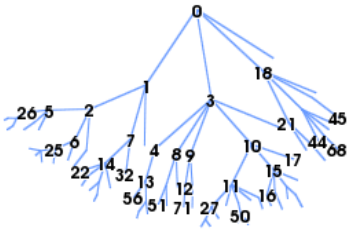
Better than random?

Please

Is my data hierarchical?

yes

Tree layout!

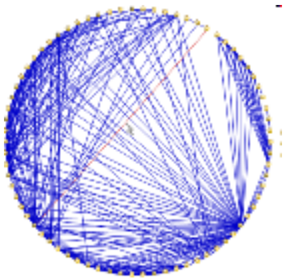


Yes, almost

Adapt data or layout

Marginally

Circular layout…

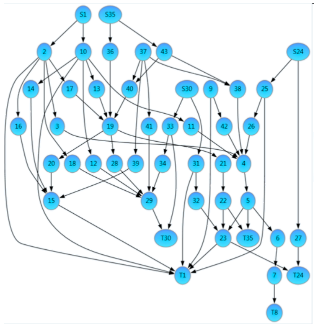


Not really

Can I 1D order my nodes?

yes

Layered layout



Not really

Need domain specific layout?

yes

Symmetrical layout

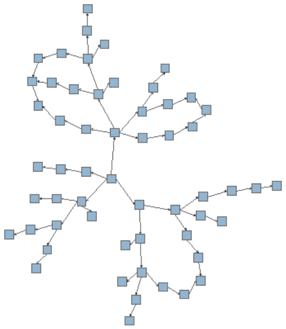
Orthogonal layout

Planar layout

Custom built layout with paper submitted to Graph Drawing

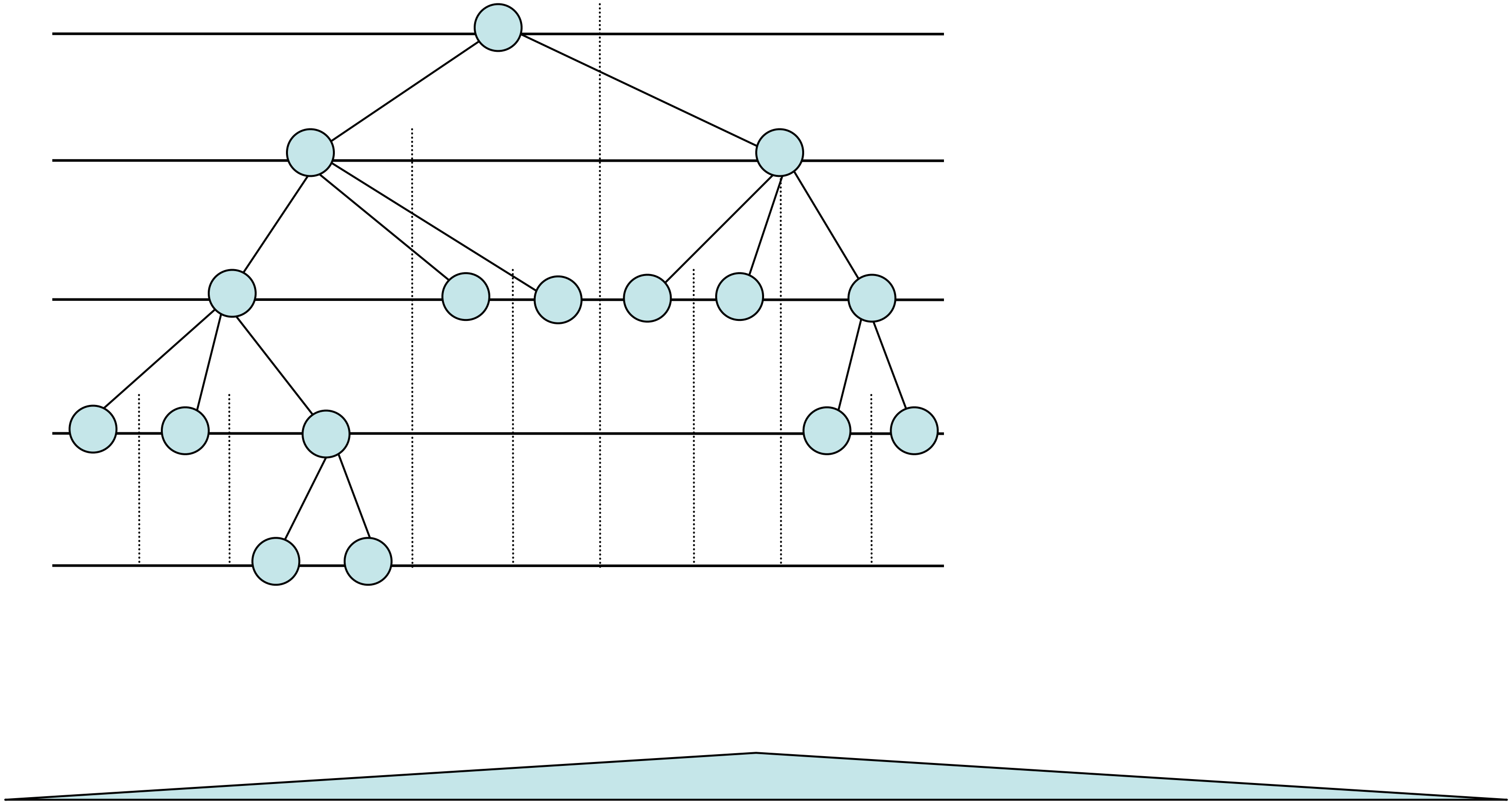
Not really

Force directed layout



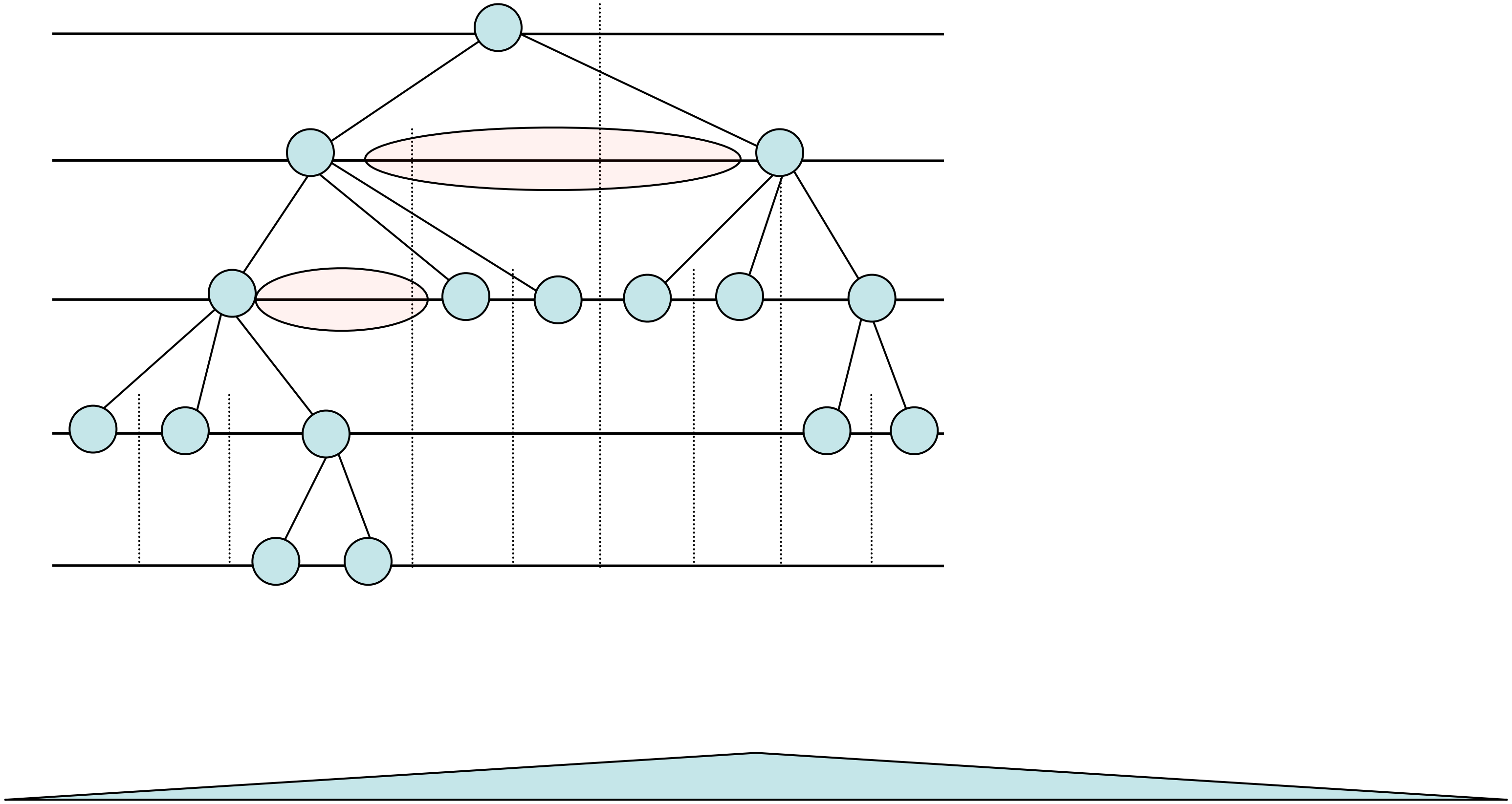
Easy case : your data is hierarchical

- Simple but naïve recursive approach



Easy case : your data is hierarchical

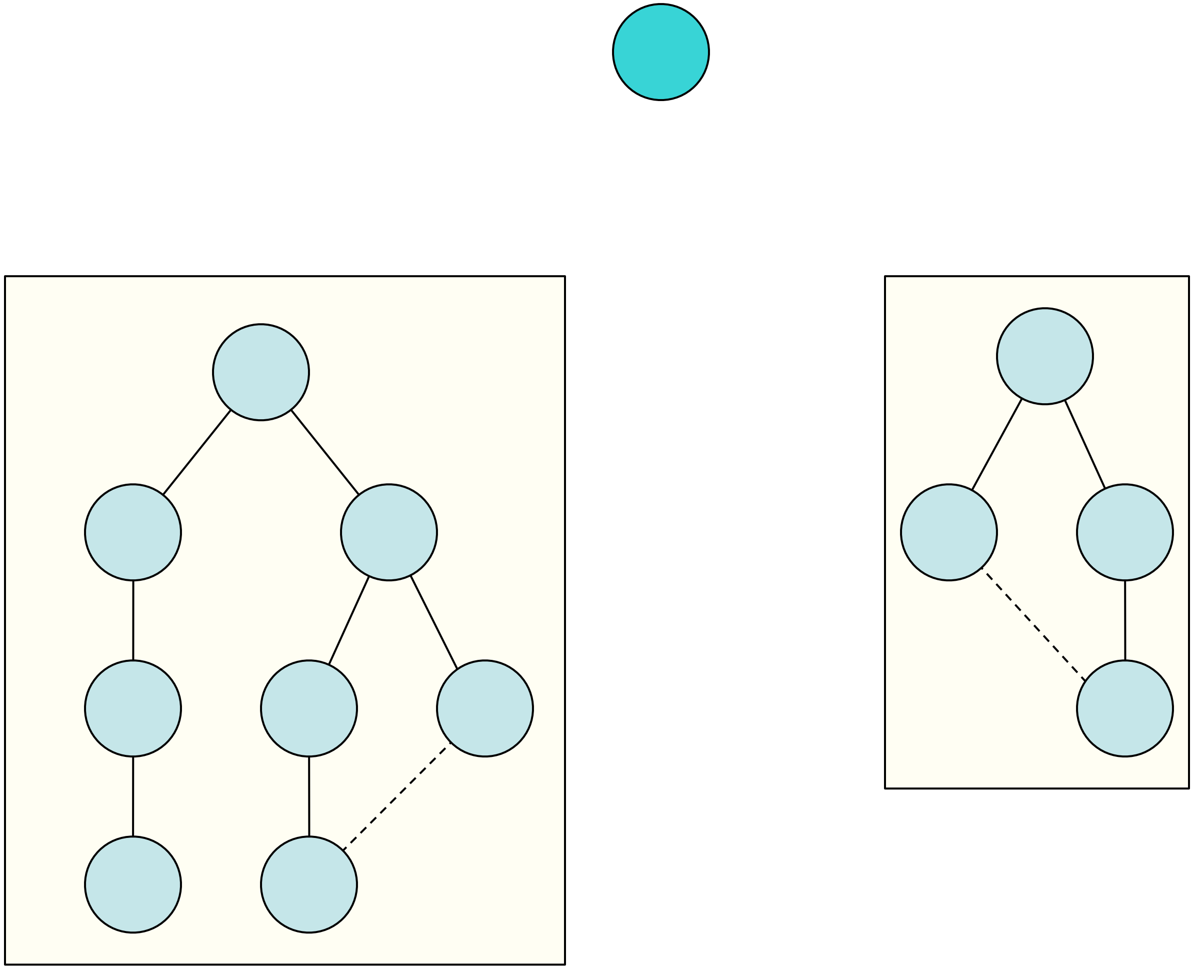
- Simple but naïve recursive approach



Better : Reingold-Tilford algorithm

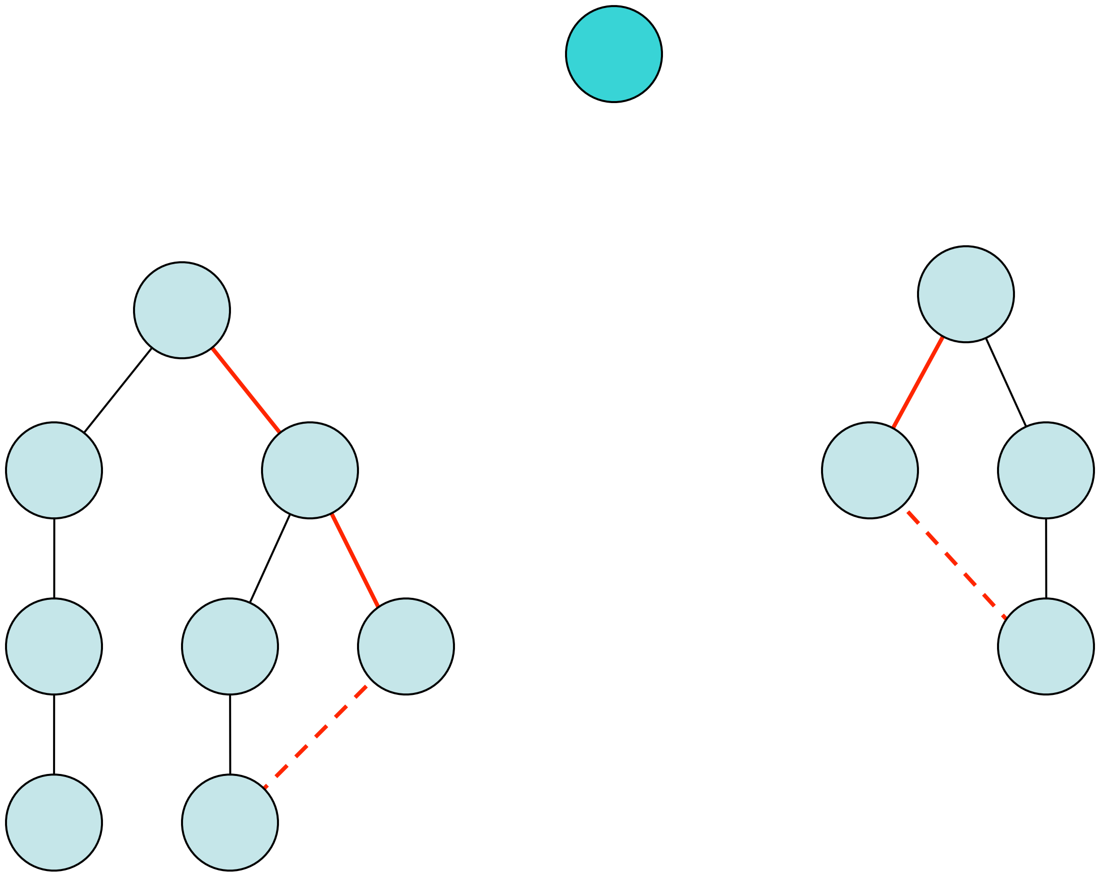
- Criteria:
  - Nodes layered by depth in tree
  - No edge crossings
  - Similar subtrees drawn in similar ways
  - Compact representation
- Approach:
  - Bottom up recursive approach
  - For each parent make sure every subtree is drawn
  - Pack subtrees as closely as possible
  - Center parent over subtrees

Sample

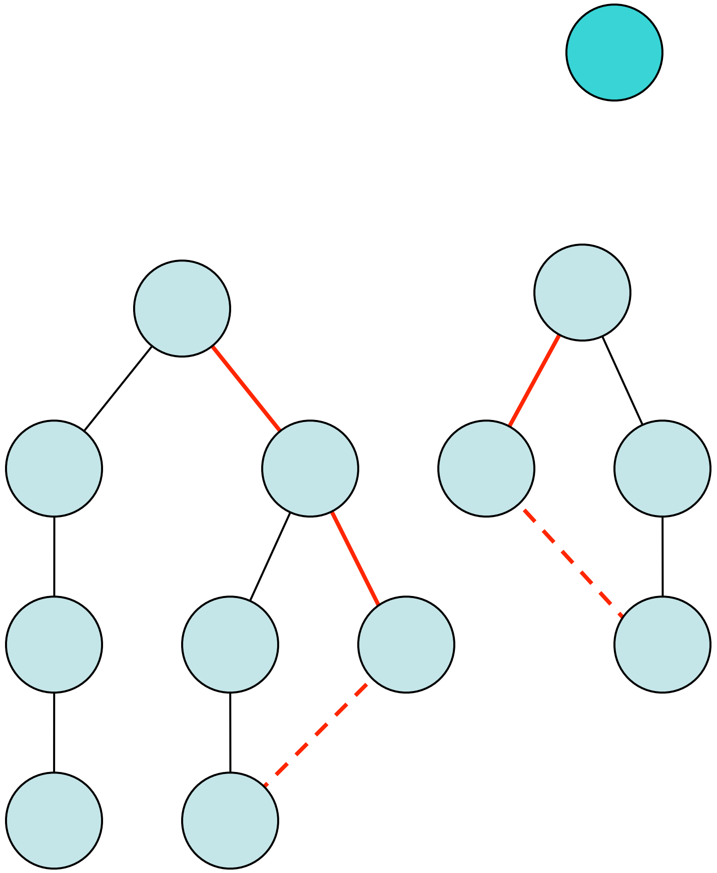


Subtrees already drawn with RT algorithm

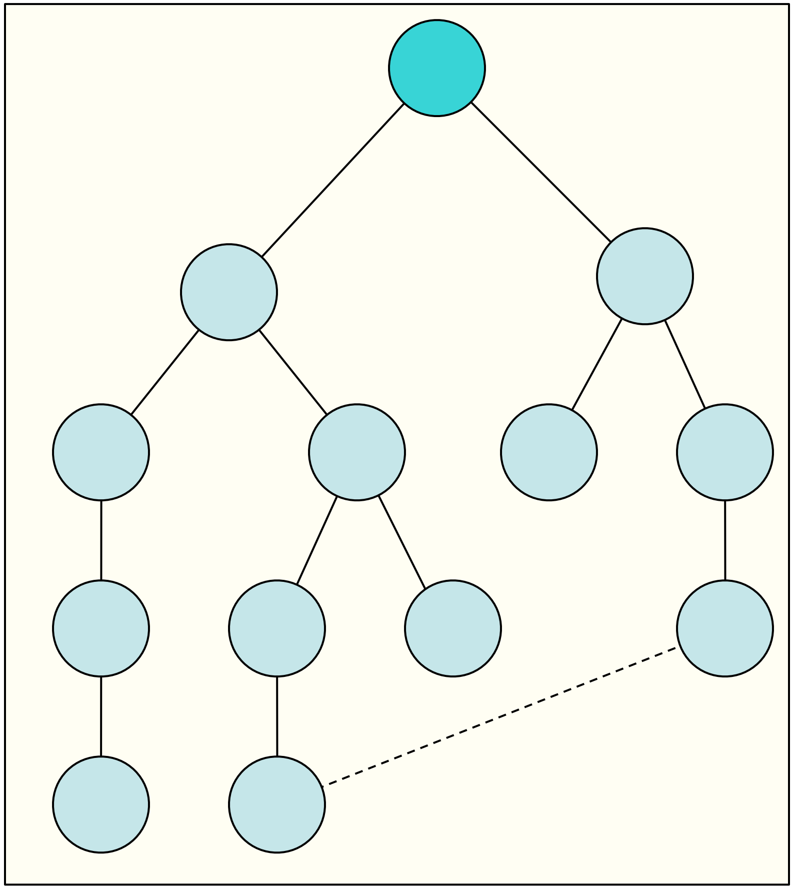
Compress subtrees



Compress subtrees

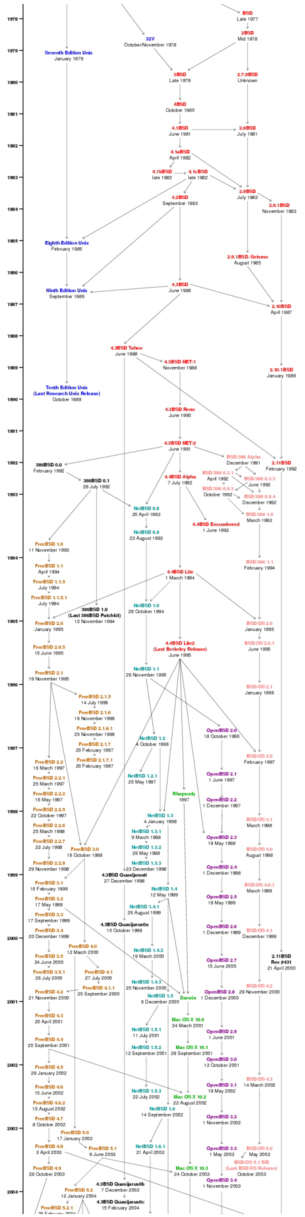


Center parent and repeat



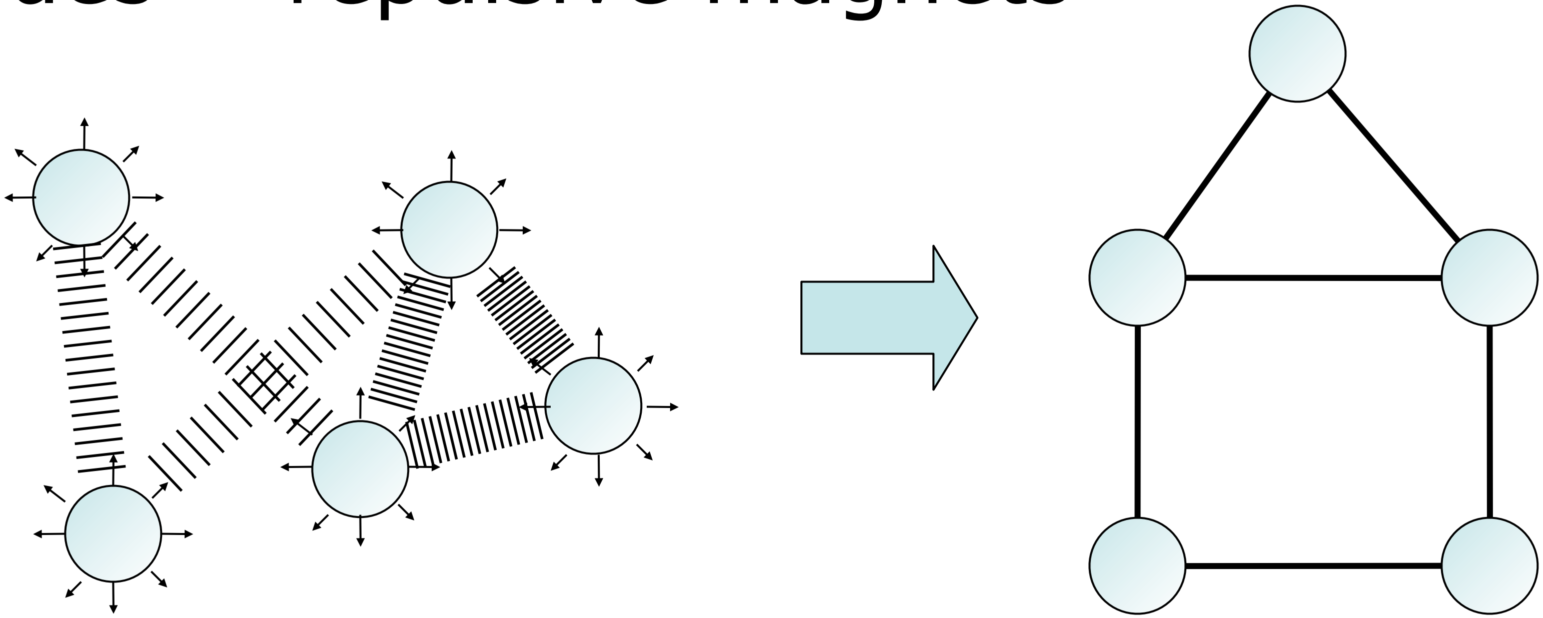
Nodes have intrinsic order

- Treelike with multiple parents
- One axis typically represents time or distance
- Use layered layout (aka Sugiyama layout)
- WARNING : do not attempt to implement yourself
- Use free or commercial alternatives
  - AT&T Graphviz (c++, unix, remote, Free)
  - Graph# (.NET, WPF, Free)
  - YFiles, Tom Sawyer (Java,$$$)

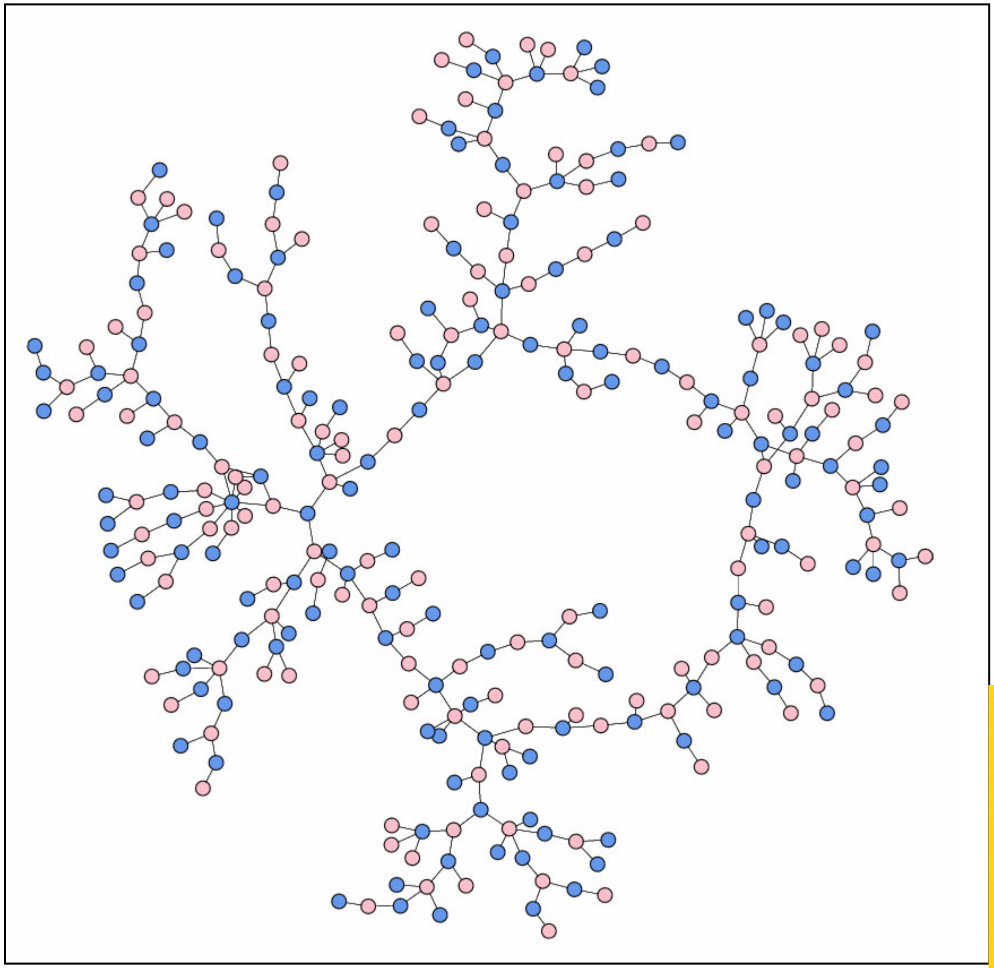


Force directed layout

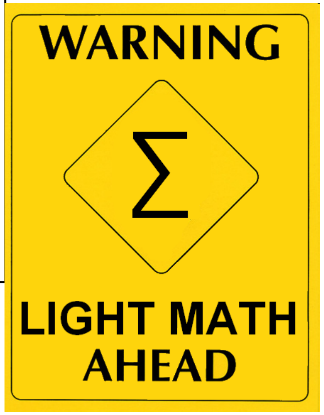
- Most-used catch-all layout.
- Physics model, edges = springs, nodes = repulsive magnets



Generates nice looking layouts

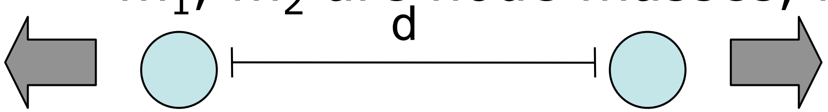


Highschool dating network

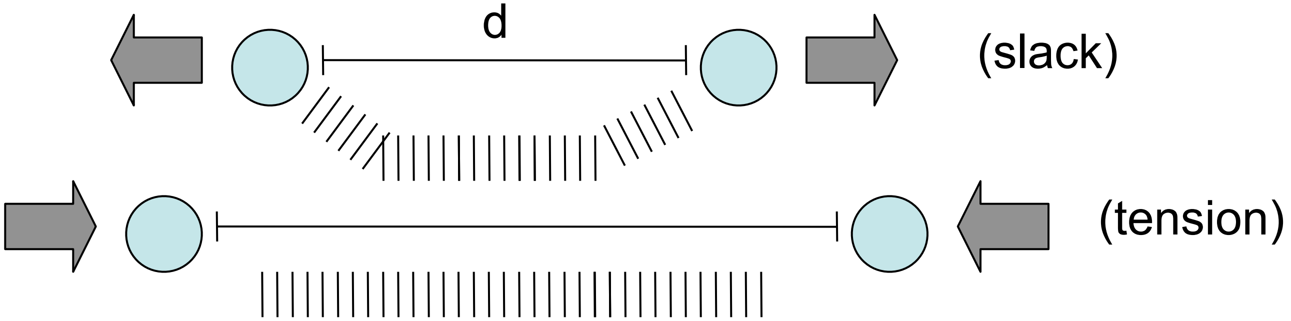


Force model

- Many variations, but usually physical analogy:
- Repulsion : $f_R(d) = C_R * m_1 * m_2 / d^2$ (inverse gravity)
  - m_1, m_2 are node masses, both usually 1



- Attraction : $f_A(d) = C_A * (d - L)$ (spring law)
  - L is the rest length of the spring or optimal edge length

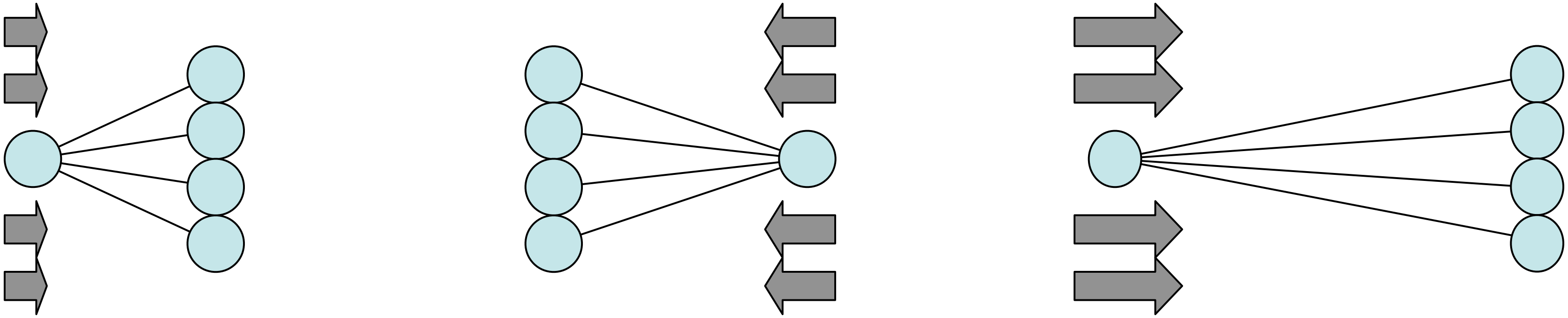


Computing actual node positions

- Start from random layout
- Loop:
  - For every node pair compute repulsive force between pair
  - For every edge compute attractive force
  - Accumulate forces per node
  - Update node position in direction of accumulated force
- Stop when layout is ‘good enough’

What nobody talks about

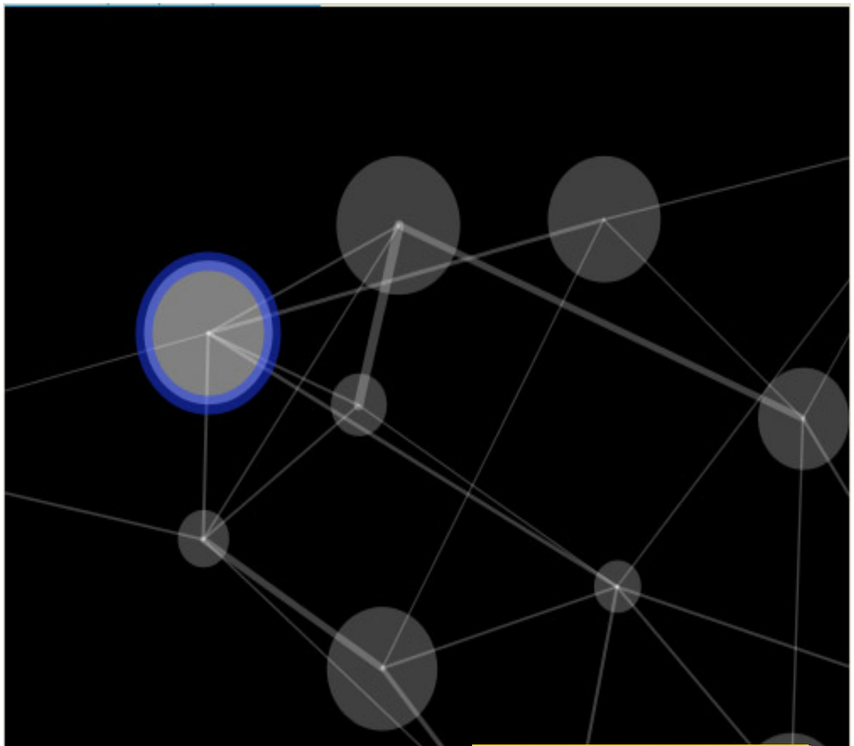
- How to pick constants C_R and C_A
- C_R found experimentally (typically around 0.001)
- C_A is trickier
  - Too small will take ages to converge
  - Too large will be unstable

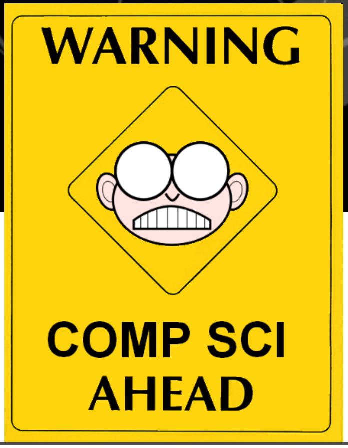


- I typically use a variable C_A per node and divide by the degree of the node

Force directed layouts

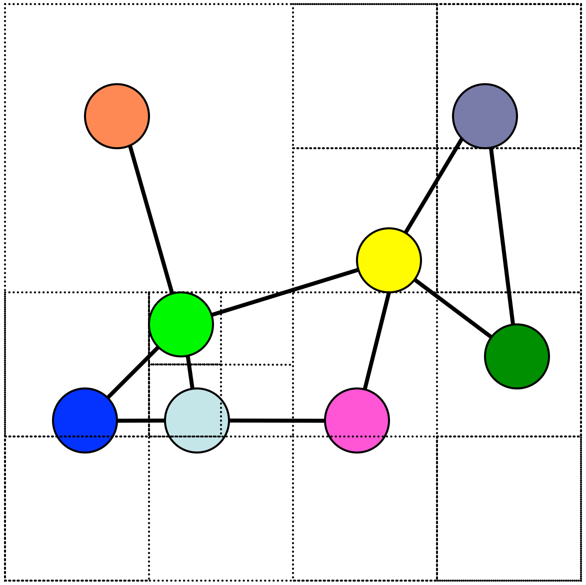
+ Very flexible, pleasing layouts on many types of graphs
+ Can add custom forces
+ Relatively easy to implement
- Repulsion loop is O(n^2) **per iteration**

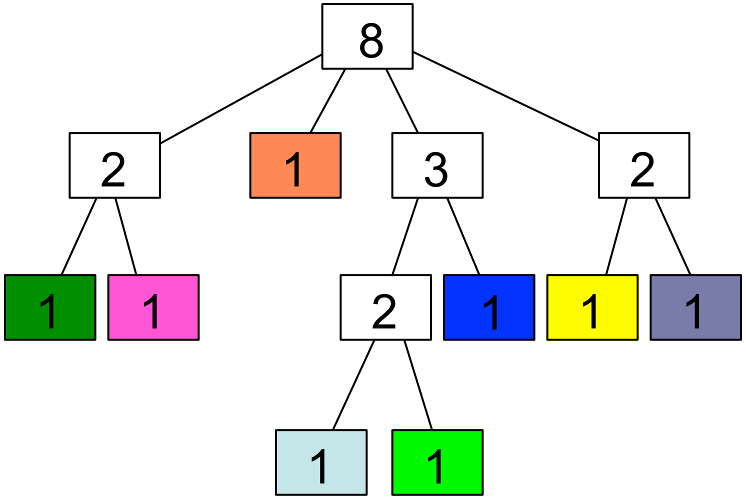




Force directed layout speedup

- Barnes-Hut multibody algorithm
- Compute quadtree for current layout
- For each non empty cell in quadtree, store total nodes in cell and center of mass (COM) of all leafnodes

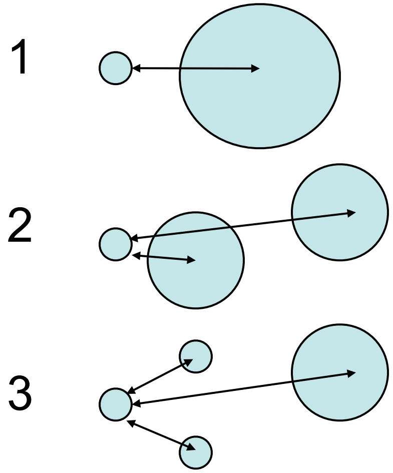




Estimating the repulsive force

- To compute the total force on a node n, look at the distance between n and the COM of the top cell
  - If bigger than a threshold we can estimate using the cell’s COM as a repulsor
  - If smaller we ‘open up’ the cell and sum the estimates for its subcells
- Complexity per iteration: O(NlogN) instead of O(N^2)

```
function double estimateForce(n:Node,cell:Cell) {
    float distance = d(n,cell.CenterOfMass)
    if (distance/cell.dimensions < threshold)
        return $\sum$ estimateForce(n,cell.children)
    else
        return $C_R$ * 1 * cell.leafcount / $\text{distance}^2$
}
```



My Top 5 FD Pet Peeves and Pitfalls

My Top 5 FD Pet Peeves and Pitfalls

5. Fixed number of iterations.

My Top 5 FD Pet Peeves and Pitfalls

5. Fixed number of iterations.
4. Keep a FD algorithm running in the background to deal with data updates or interaction.

My Top 5 FD Pet Peeves and Pitfalls

5. Fixed number of iterations.
4. Keep a FD algorithm running in the background to deal with data updates or interaction.
3. Failing to test for disconnected components before running FD.

My Top 5 FD Pet Peeves and Pitfalls

5. Fixed number of iterations.
4. Keep a FD algorithm running in the background to deal with data updates or interaction.
3. Failing to test for disconnected components before running FD.
2. Failing to test for trees before running FD.

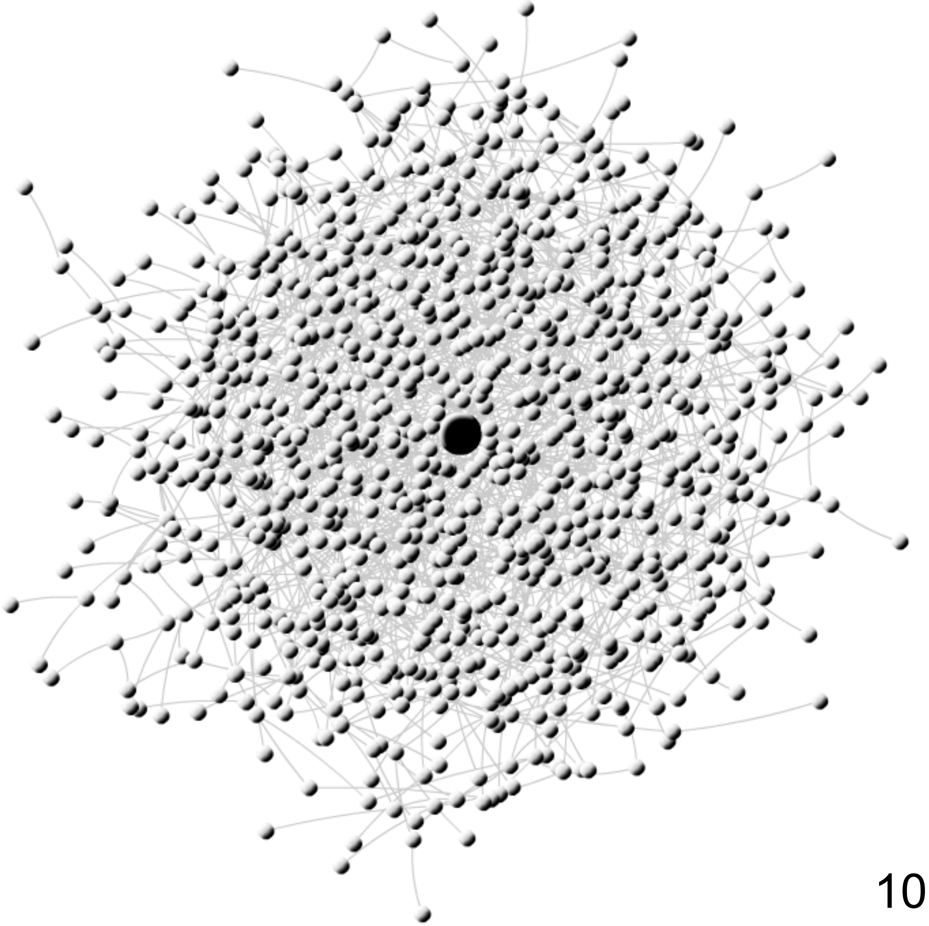
My Top 5 FD Pet Peeves and Pitfalls

5. Fixed number of iterations.
4. Keep a FD algorithm running in the background to deal with data updates or interaction.
3. Failing to test for disconnected components before running FD.
2. Failing to test for trees before running FD.
1. Showing the user your full FD optimization EVERY SINGLE TIME they run a layout.

THE problem with node link layouts:

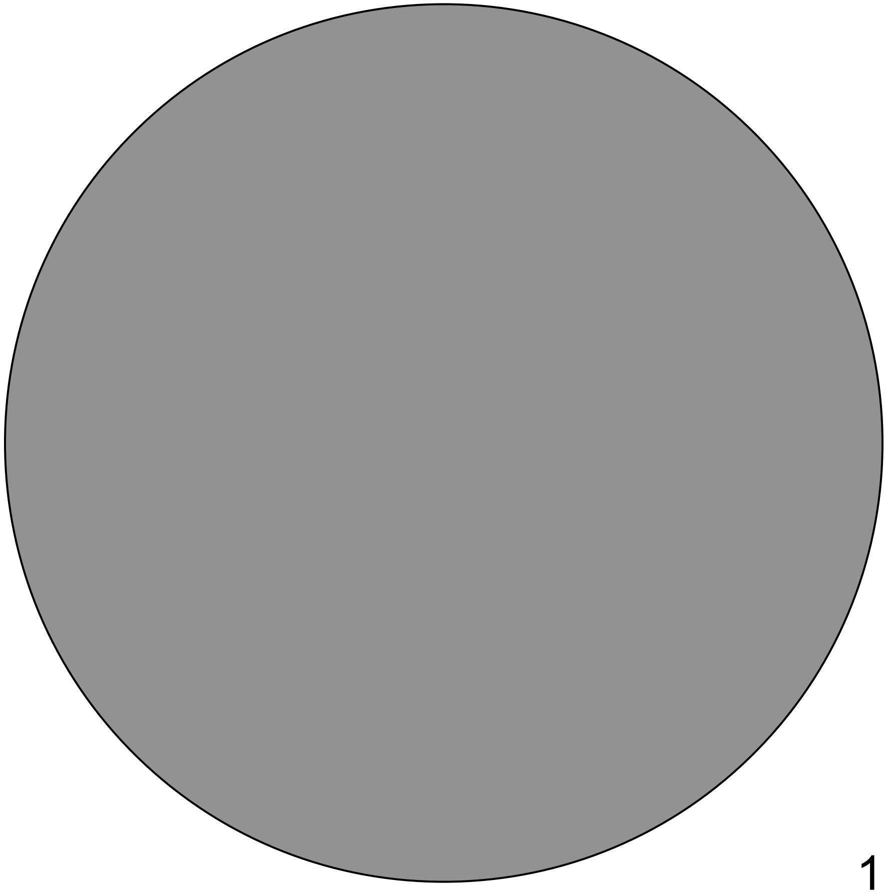
- They map distance in the graph to distance on screen (which is good)
- If the maximum distance in the graph is small, the screen distance between nodes will be small.
- Remember six degrees of separation?
- A lot of networks have small average node distance

The “Hairball” Problem



1000 nodes

The “Hairball” Problem



100.000 nodes
(extrapolation)

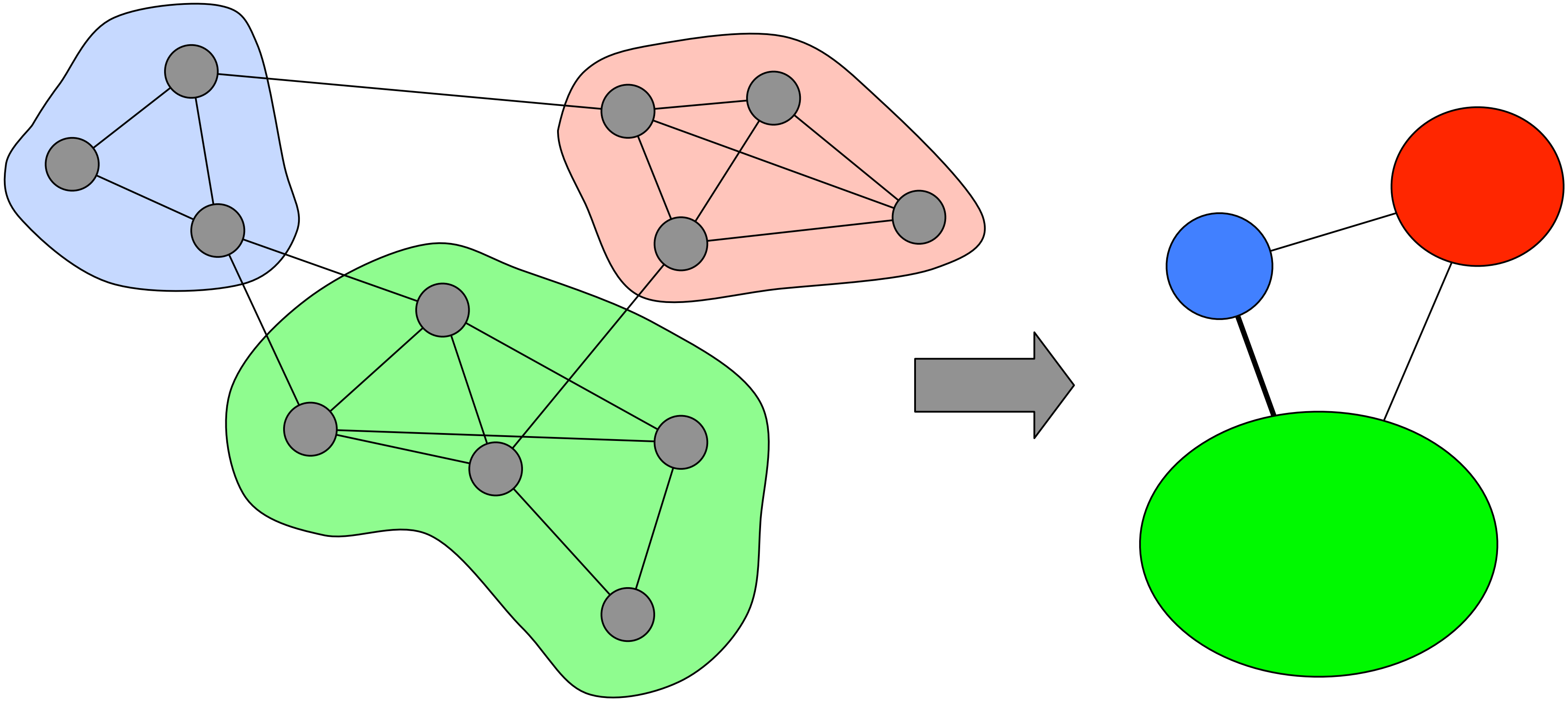
Now what?

Solutions

- Three high level classes of solutions
  - Abstraction : Clustering
  - Alternative representations : Matrices
  - Filtering : Partial views

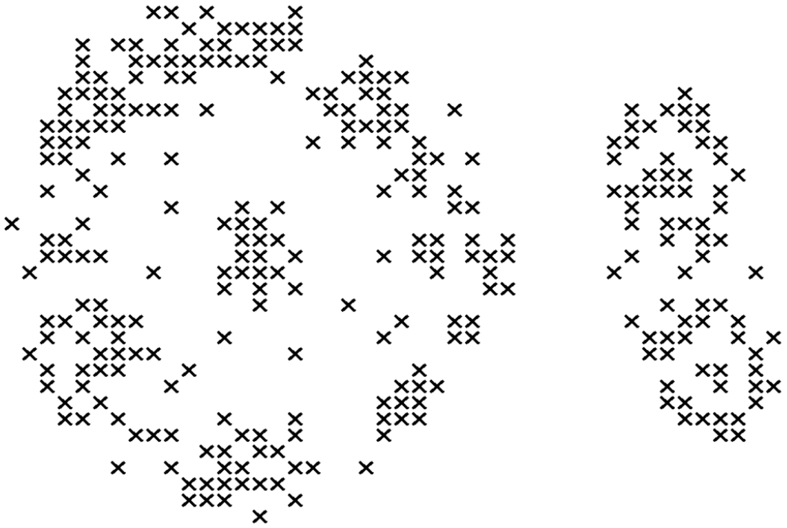
Clustering

- Idea : “Divide and conquer by grouping nodes together in clusters.”



Structural clustering

Underspecified problem:



Pick a graph clustering algorithm

Set the clustering granularity

Compute abstracted graph

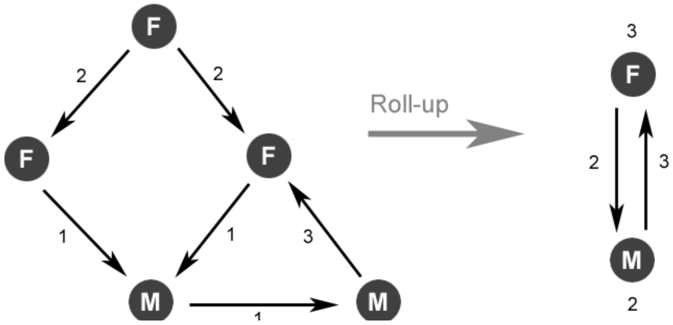
Apply recursively if clusters still too big

Main drawbacks

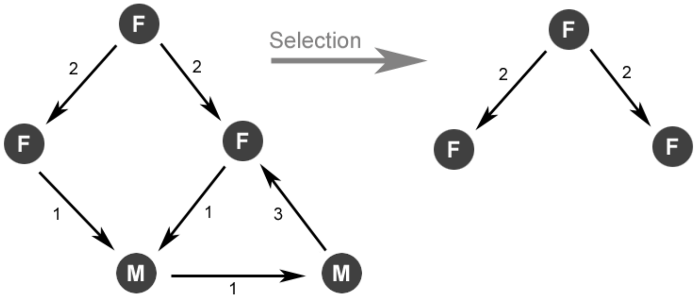
- Good clustering algorithms are expensive > O(N^2)
- On dense graphs that do not have a very explicit structure you'll get
  - Yet another hairball (albeit smaller)
  - Different results depending on clustering algorithm
  - Interpretability problems (what does a cluster represent)

Attribute based clustering

- If we have attribute based information we can cluster on (ordinal) attributes.
- Fast : O(N)
- Easy to interpret
- Similar to OLAP in databases



Rollup: Pick one or more attributes of interest, and aggregate nodes based on those attributes. Here: a roll-up of a social network based on gender.

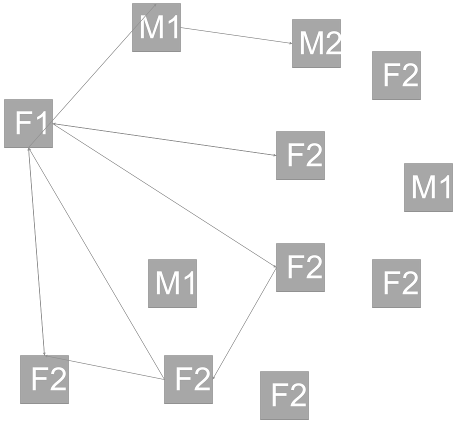


Selection: show only nodes that have specified values on given attributes. a.k.a. “induced subgraph”
Here: selection on gender=female

Pivotgraph

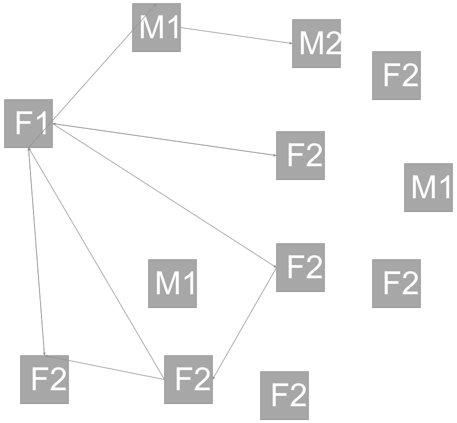
Pivotgraph

Multivariate graph

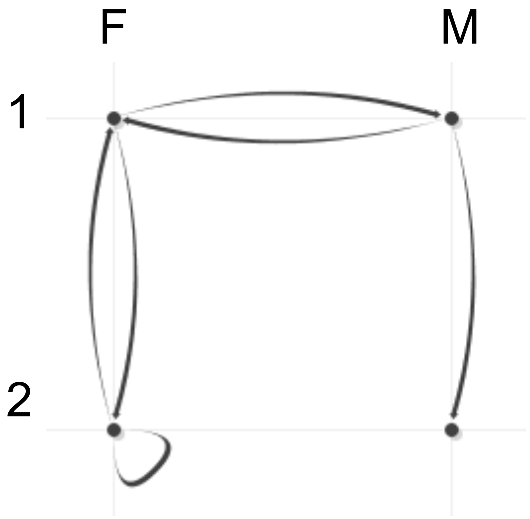


Pivotgraph

Multivariate graph

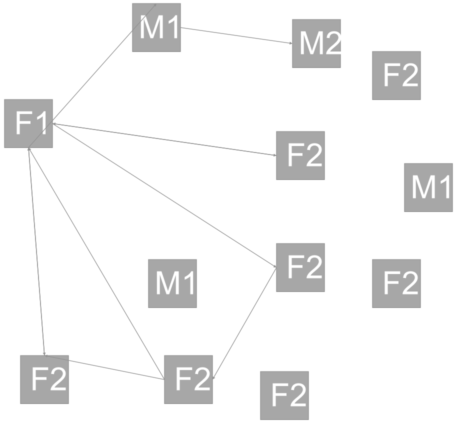


Roll up, project

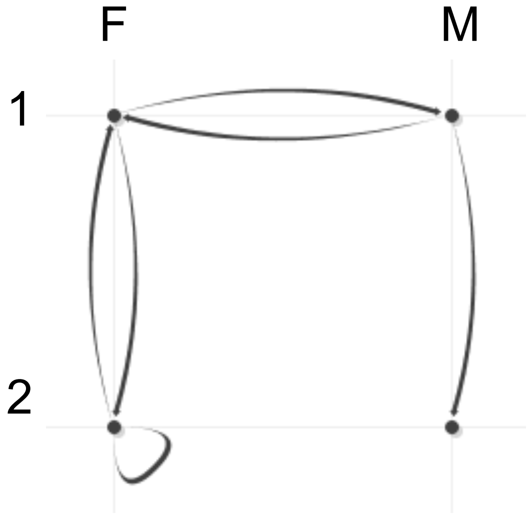


Pivotgraph

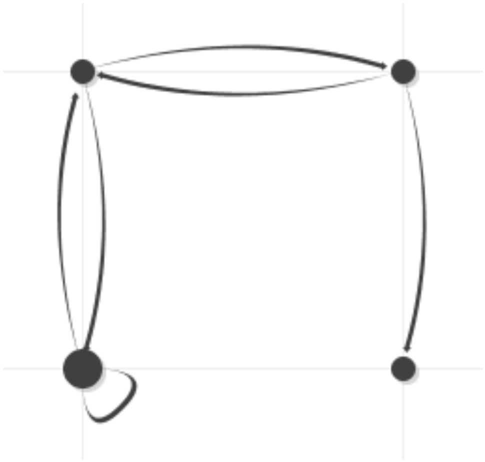
Multivariate graph



Roll up, project

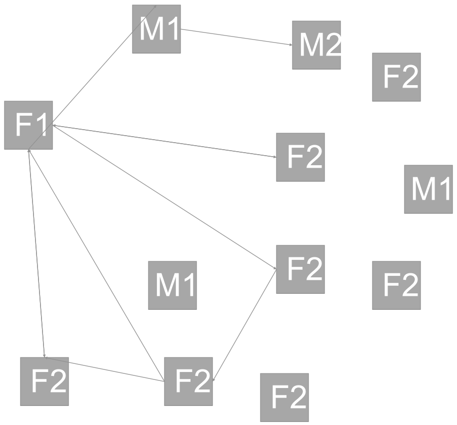


Node size

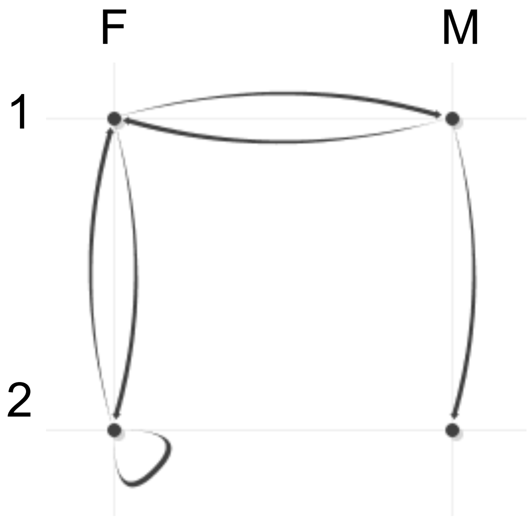


Pivotgraph

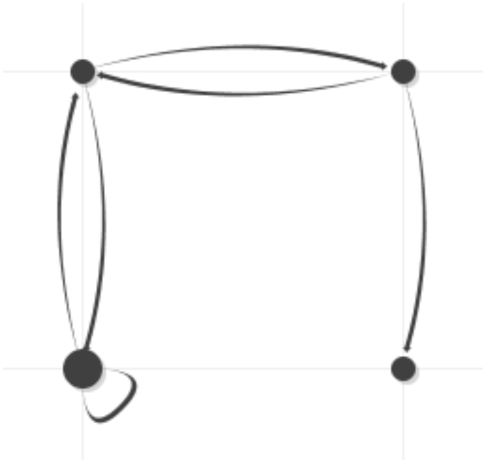
Multivariate graph



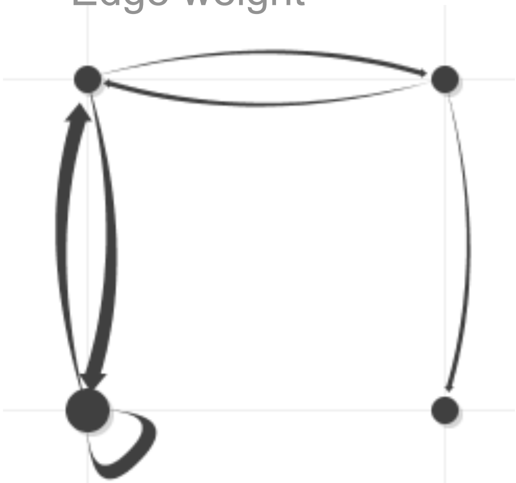
Roll up, project



Node size

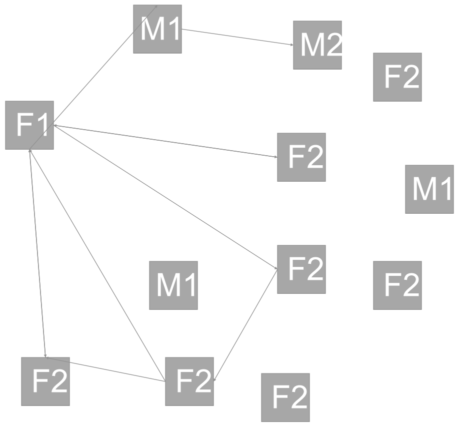


Edge weight

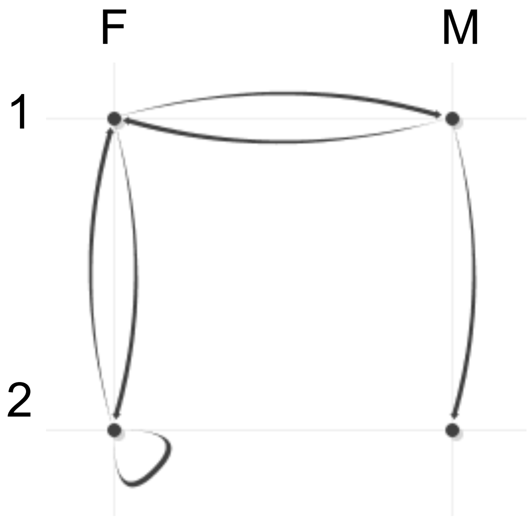


Pivotgraph

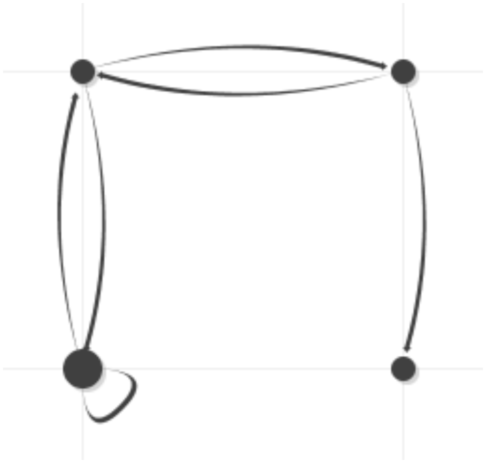
Multivariate graph



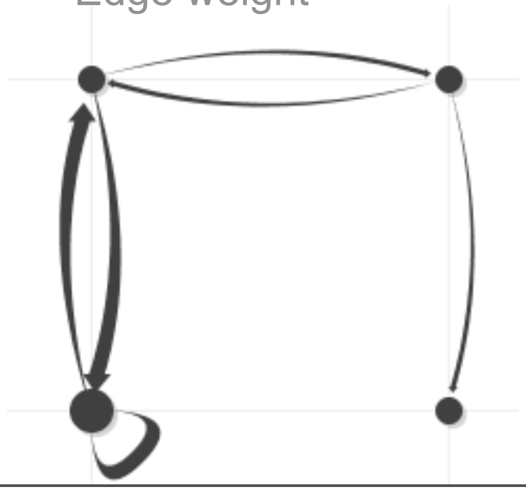
Roll up, project



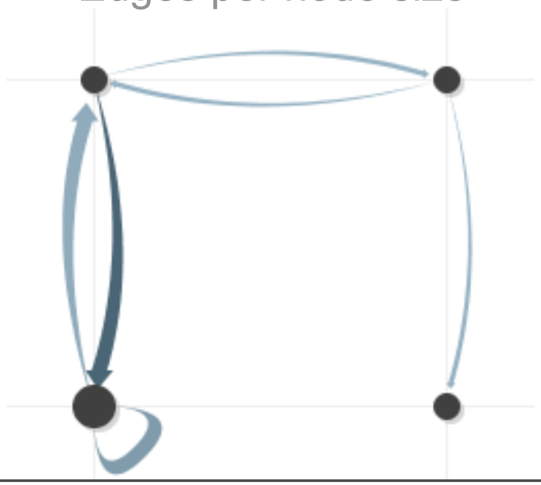
Node size



Edge weight

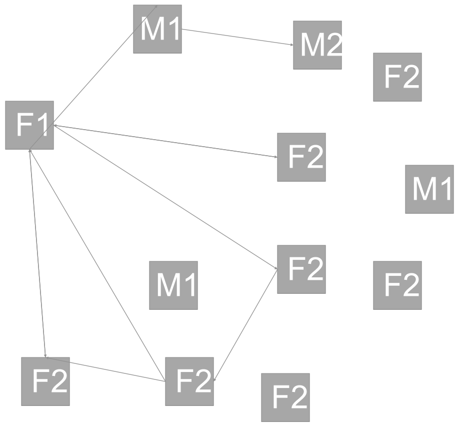


Edges per node size

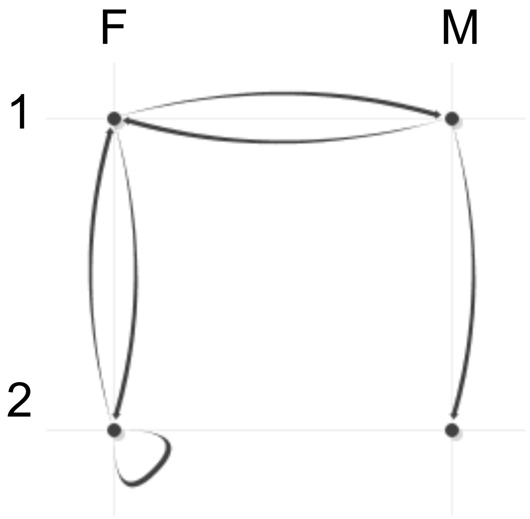


Pivotgraph

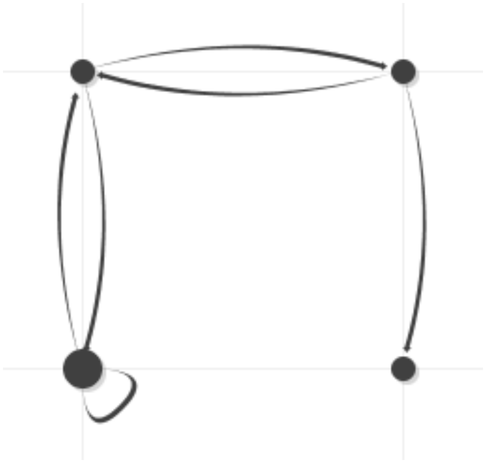
Multivariate graph



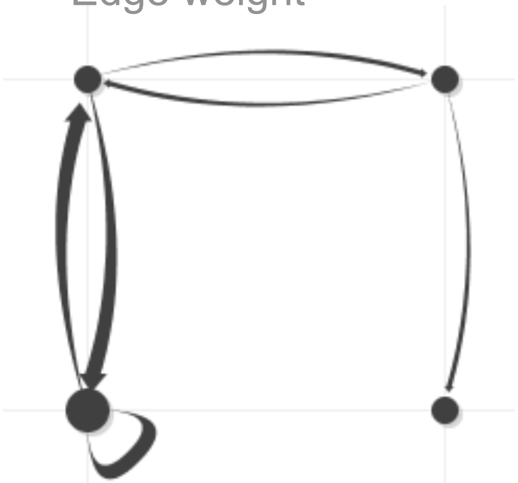
Roll up, project



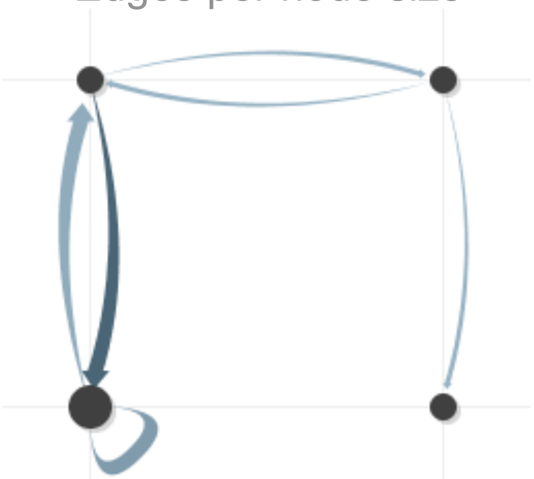
Node size



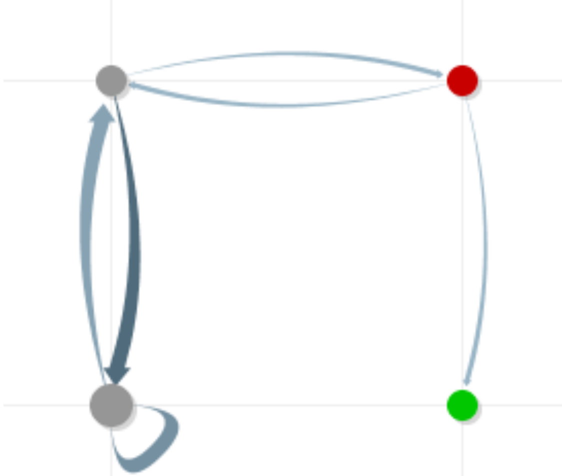
Edge weight



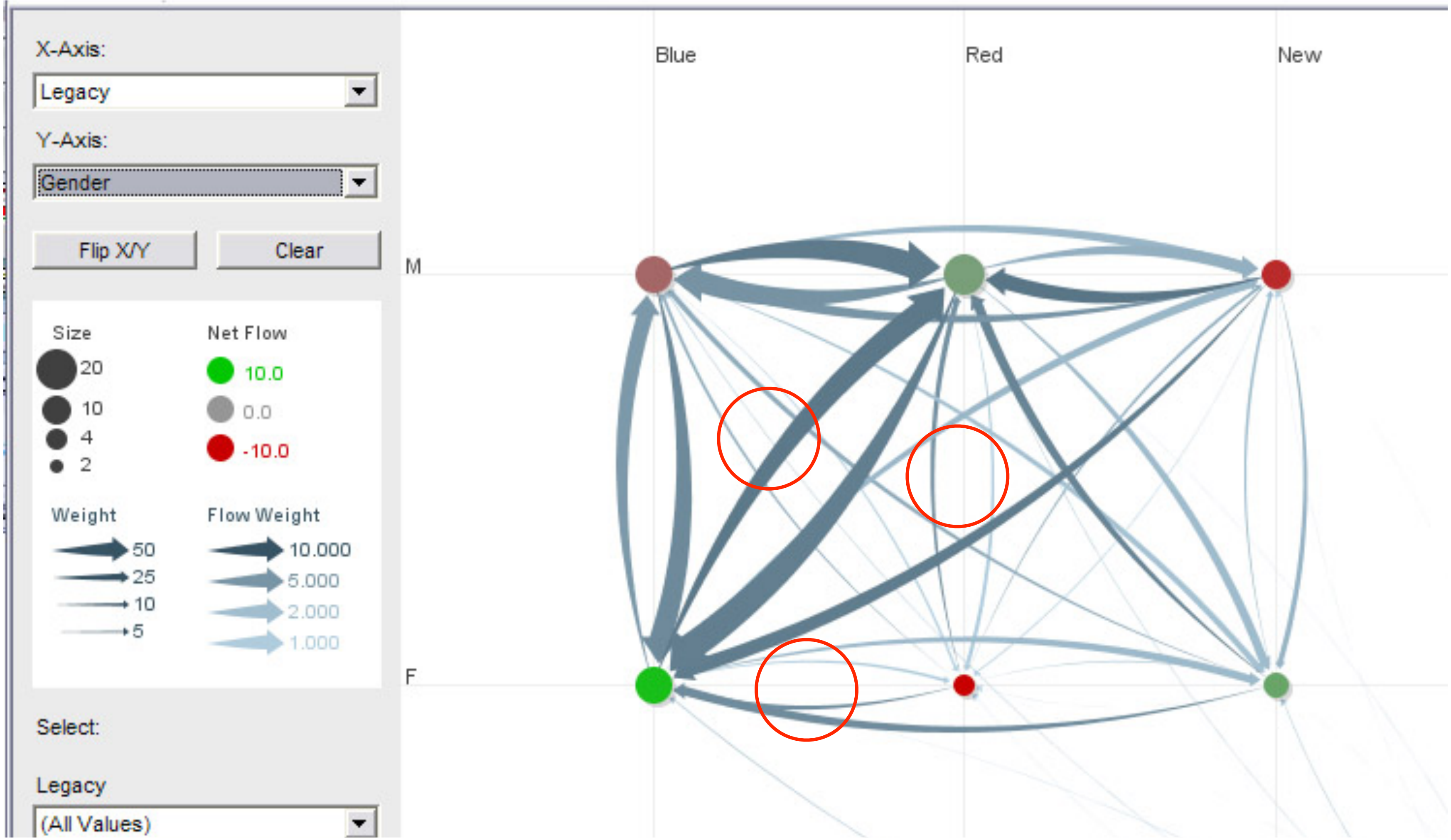
Edges per node size



Net flow in/out



Sample – 2 merged companies



X-Axis: Legacy

Y-Axis: Gender

Select:

Legacy (All Values)

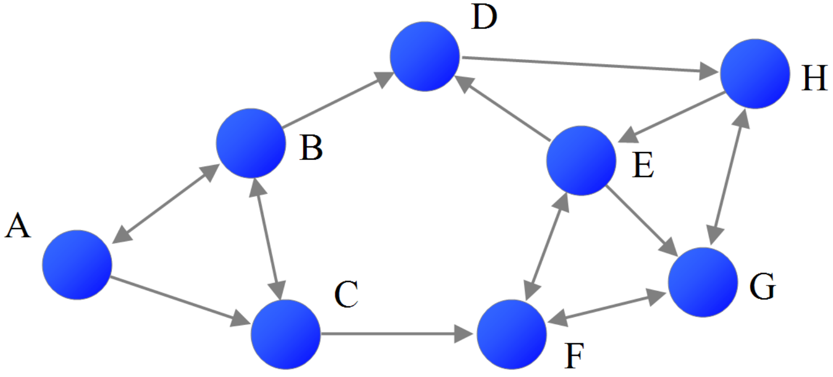
Showing e-mail patterns (X-axis : Previous company; Y-axis : Gender)

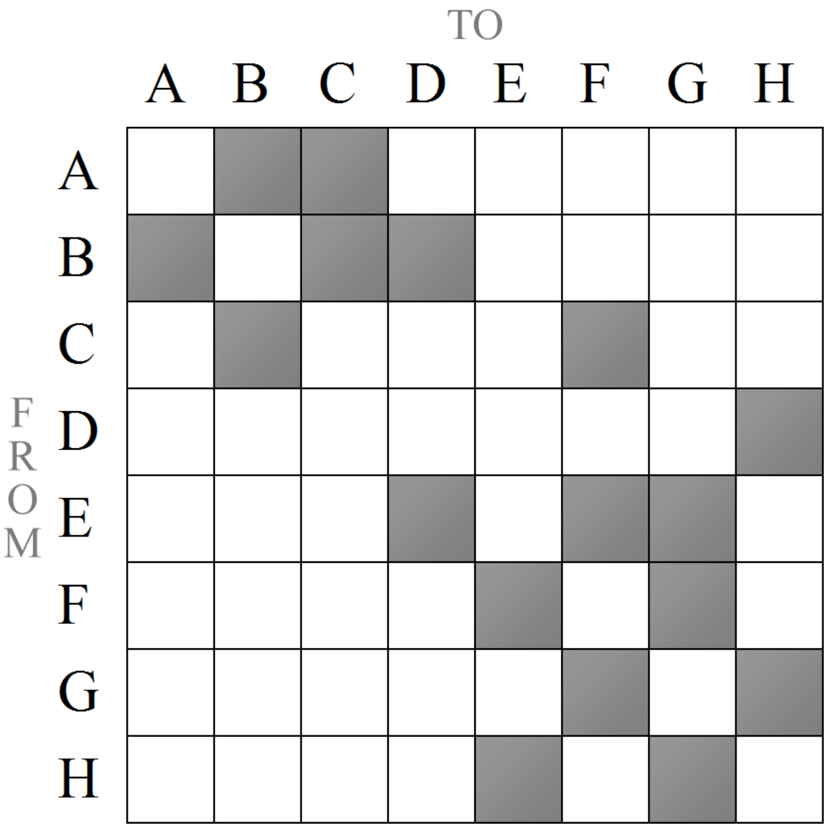
Clustering is

- Useful to reduce data complexity
- Computationally expensive
- Conceptually easy to understand
- Great if your graph has an obvious clustering structure
- In other cases use attribute clustering for interpretability

Alternative representations

- Idea : "Let's do away with node link diagrams all together!"





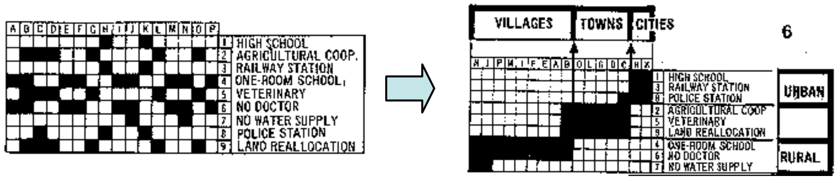
TO

FROM	A	B	C	D	E	F	G	H
A		■	■					
B	■		■	■				
C		■				■		
D								■
E				■		■	■	
F					■		■	
G						■		■
H					■		■	

Node link diagram (right) and associated adjacency matrix (left)

A giant adjacency matrix

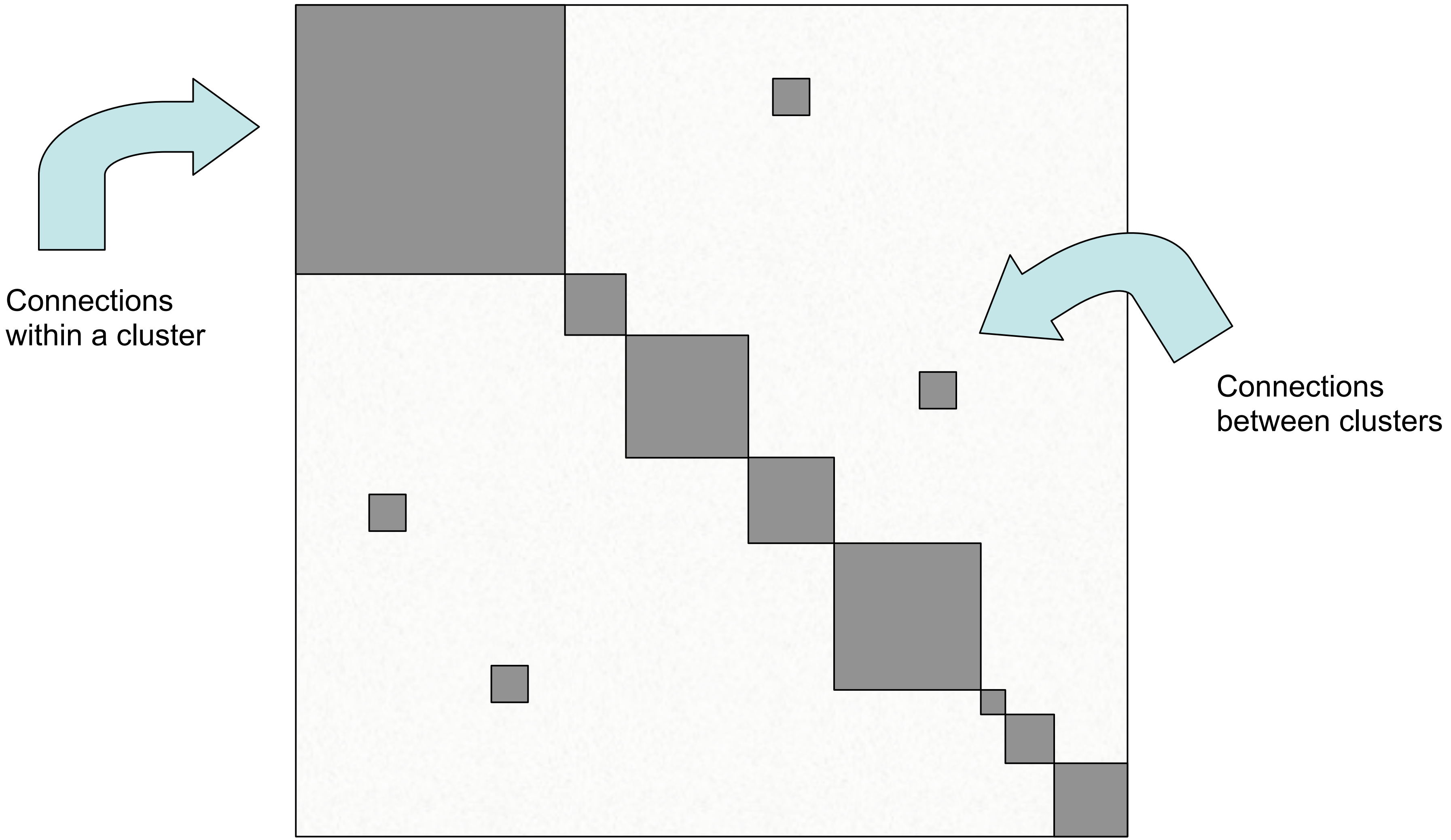
Reordering



Bertin 1981

- Matrices have one degree of freedom
- Reorder nodes such that similar rows/columns are kept together
- Define a dissimilarity measure
- Create a linear ordering such that the total dissimilarity is minimal
- Find a minimal tour in a graph where edges represent dissimilarity
- NP complete (Traveling Salesmap problem)

A reordered adjacency matrix



- Use TSP heuristics directly
- Sparse matrix bandwidth minimization

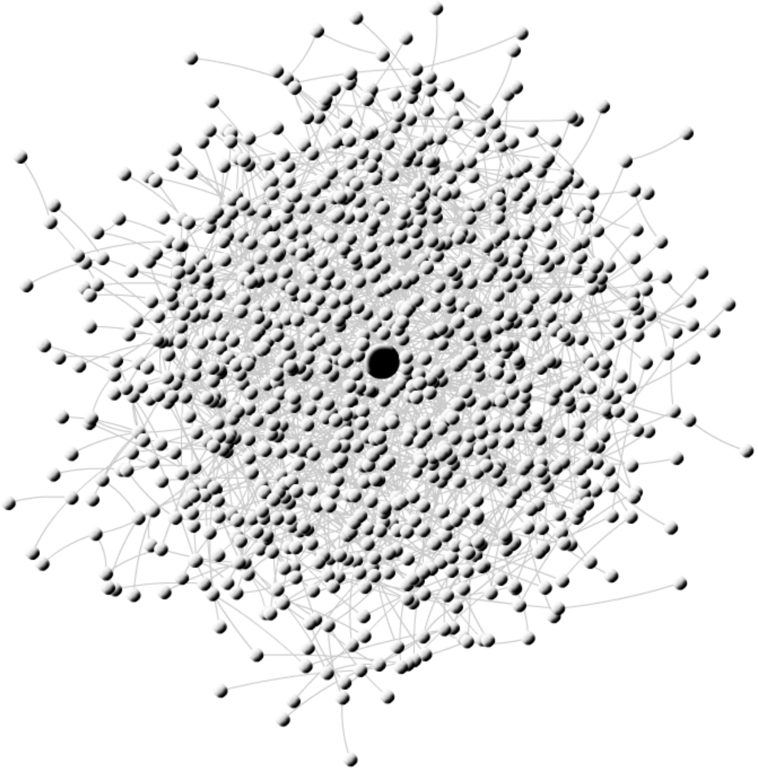
Matrices

- Can handle arbitrarily dense graphs (no hairballs!)
- Scale well with clustering approaches
- Are more difficult to interpret
- Cannot display paths through networks of length > 1

Partial Views

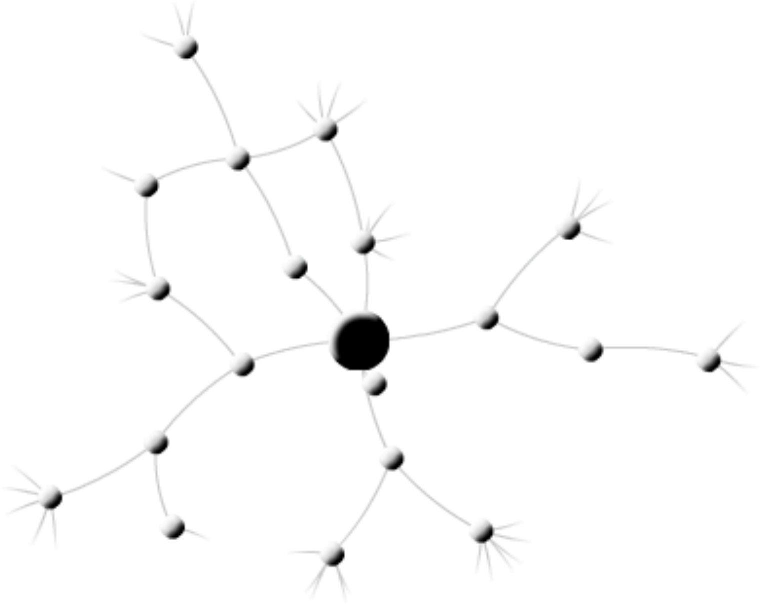
- Idea : “Forget overviews! Who needs overviews!”
- Actually, not a lot of people truly need overviews of entire networks.
- Most people are interested in the local area around a point of interest
  - Social networks : me
  - Fraud analysis : a flagged account
  - Program dependencies : the piece of code I’m working on
- Global data not always available (online)

Partial views : idea



Global view

Full context

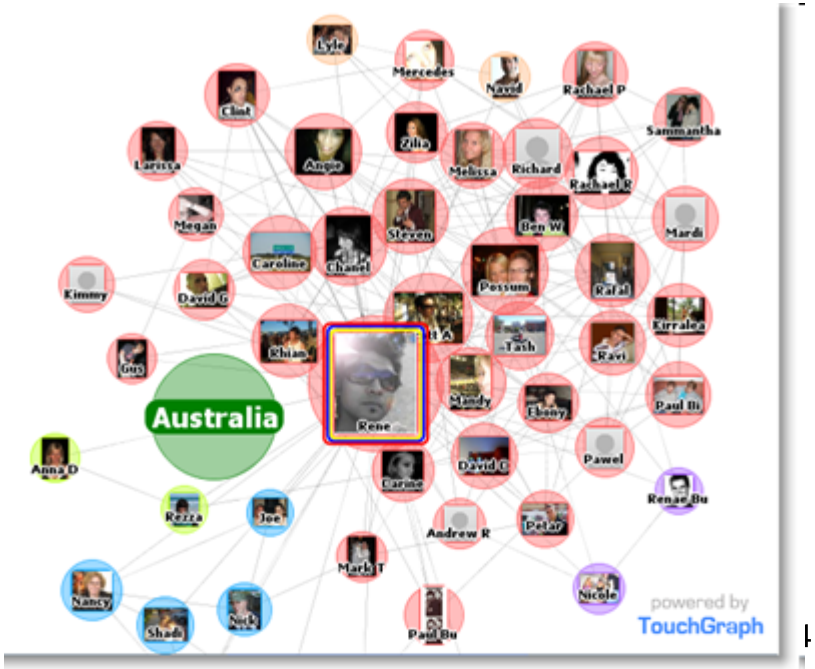


Local view

Less context

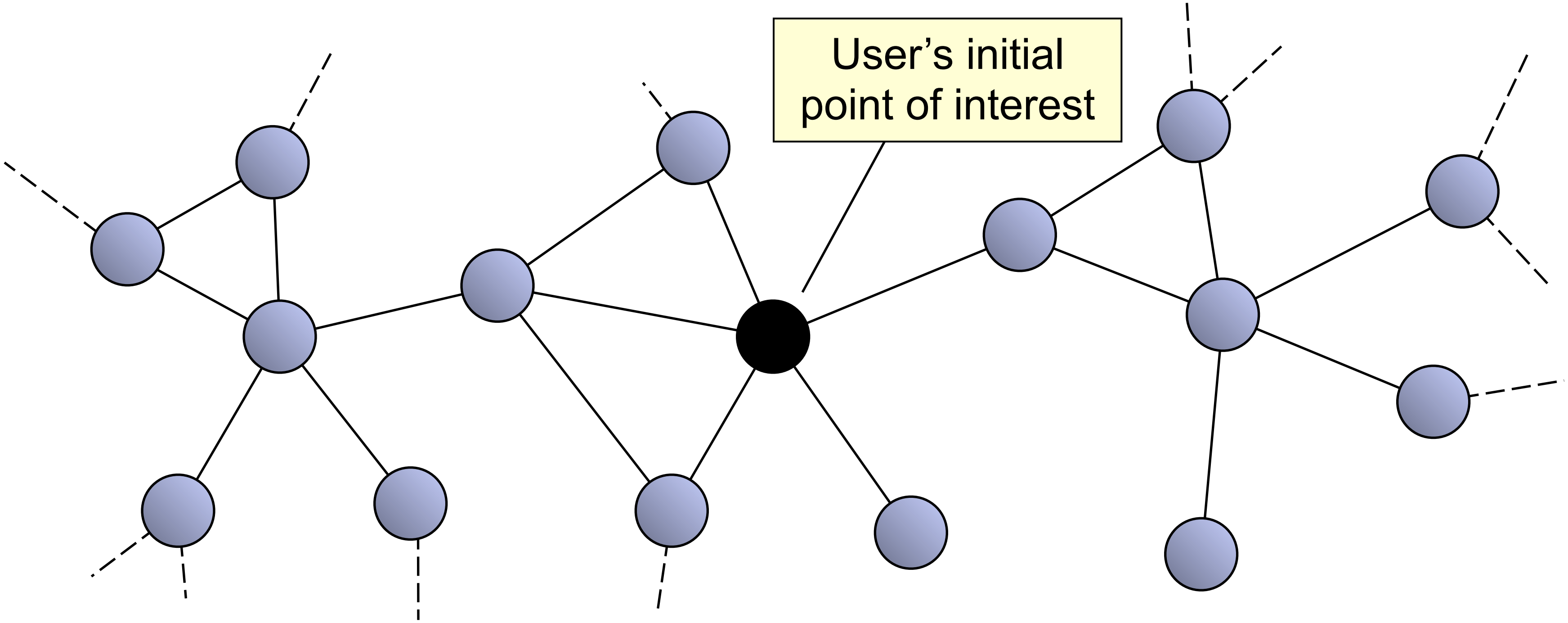
Make context expandable

- User clicks a node *x*
- Add *x*’s neighbors to graph
- Compute new layout
- Display expanded context
- Demo : Touchgraph

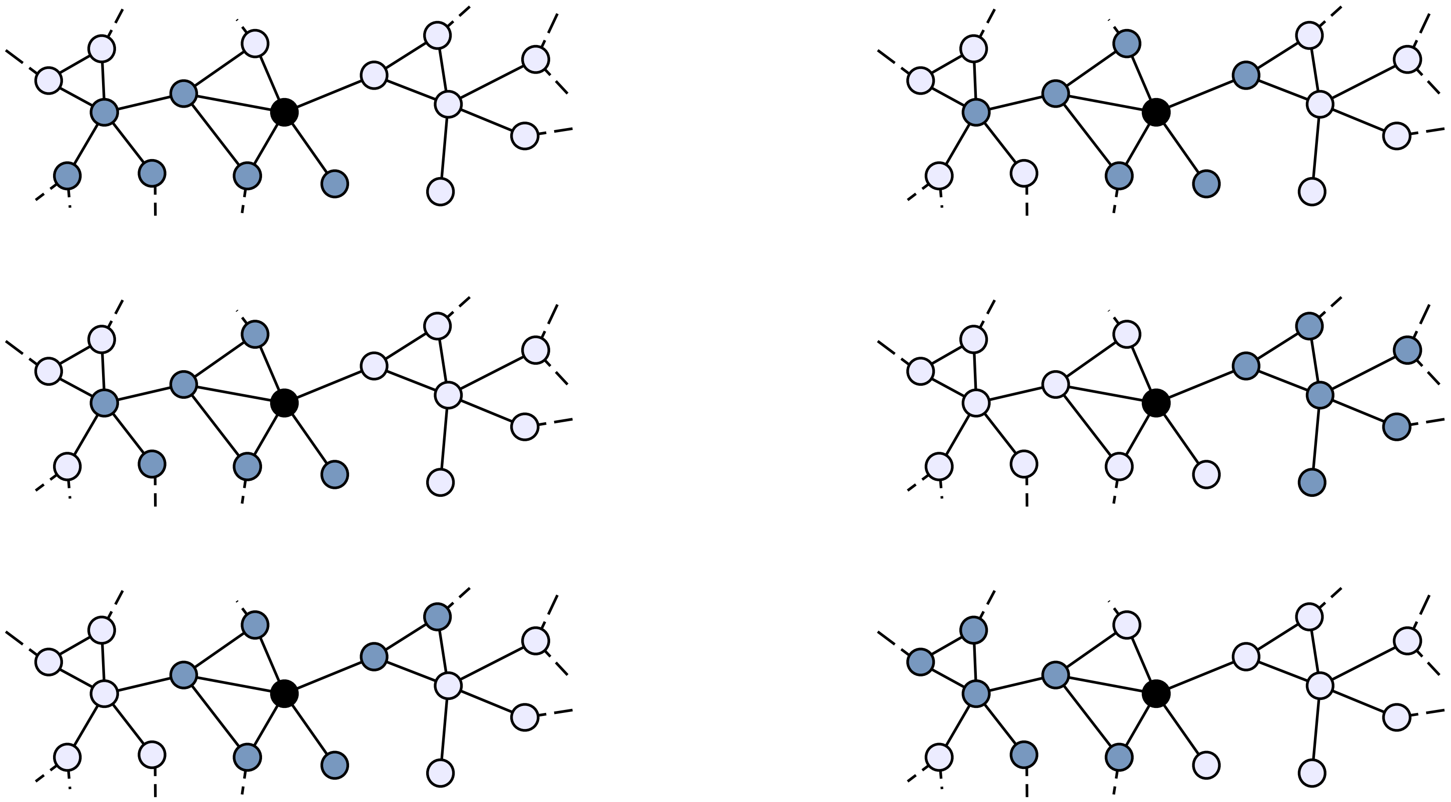


Partial navigation

- Instead of showing the full network, show a smaller contextual network around a user specified point of interest.

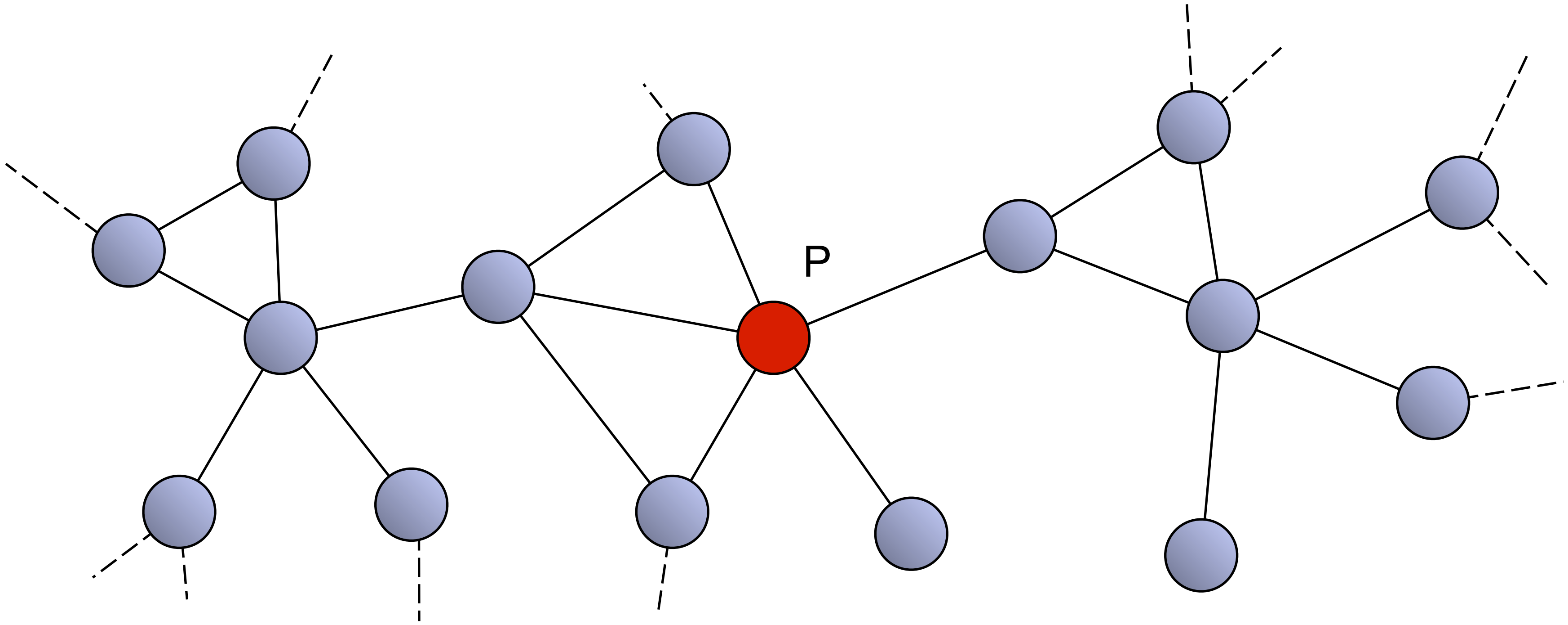


Many contexts possible!



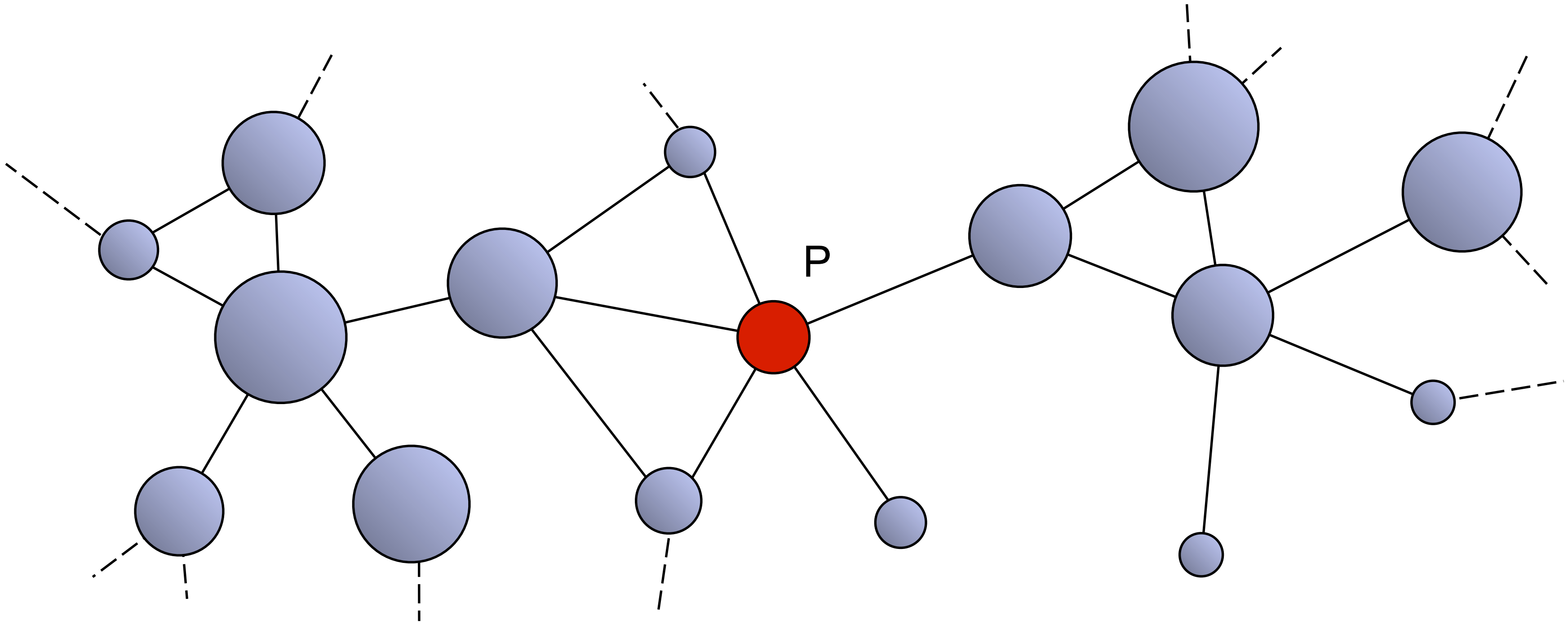
- Which one is the best one?

Compute ‘interest’ for each item



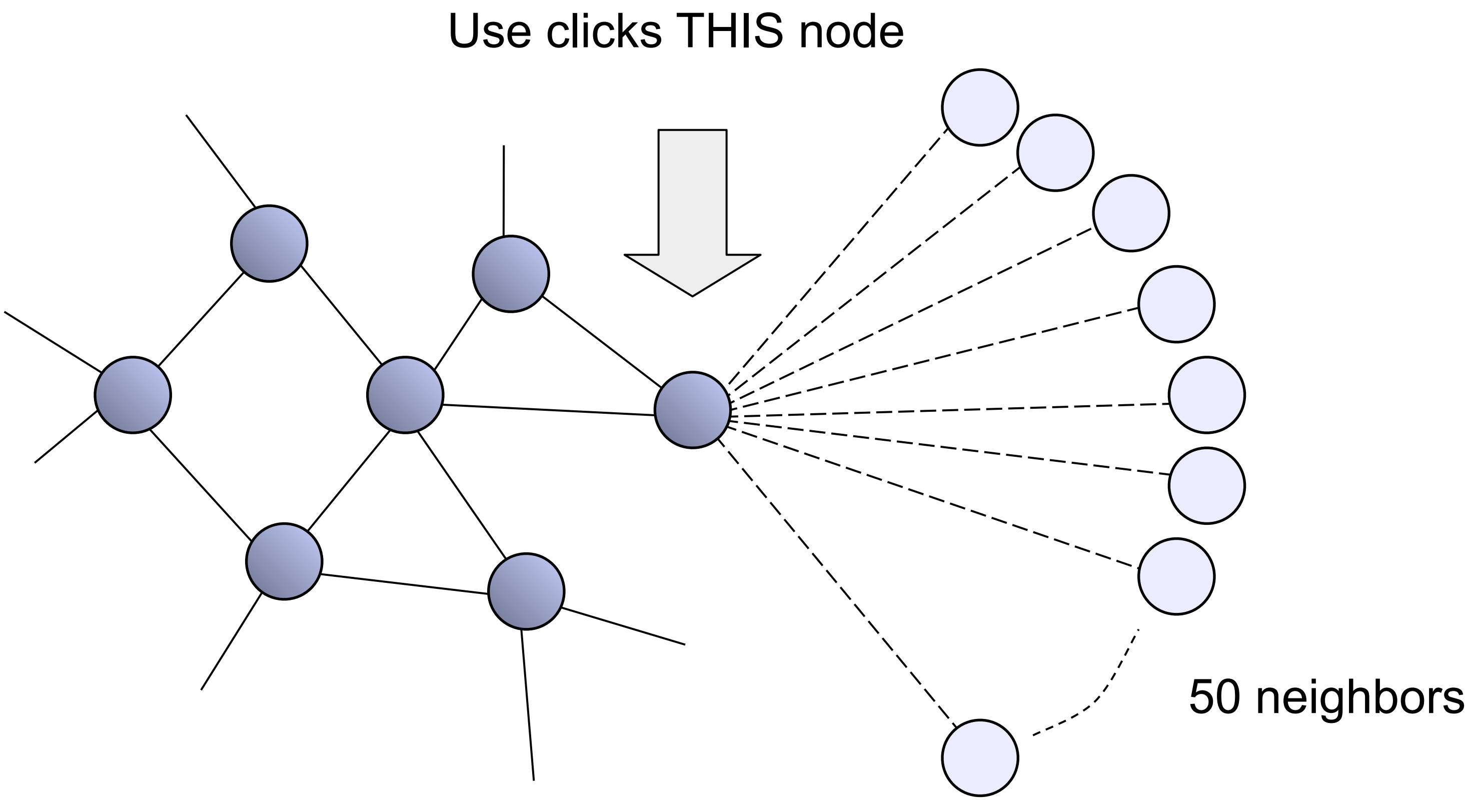
- Interest function is domain dependent but can be based on:
    - Network topology (critical points of failure)
    - Item attributes (high dollar value items)
    - Historical data (recently added items)
    - User annotation (items flagged by users)
    - Any combination of the above
- Best contextual network is the one with highest total interest.

Compute ‘interest’ for each item



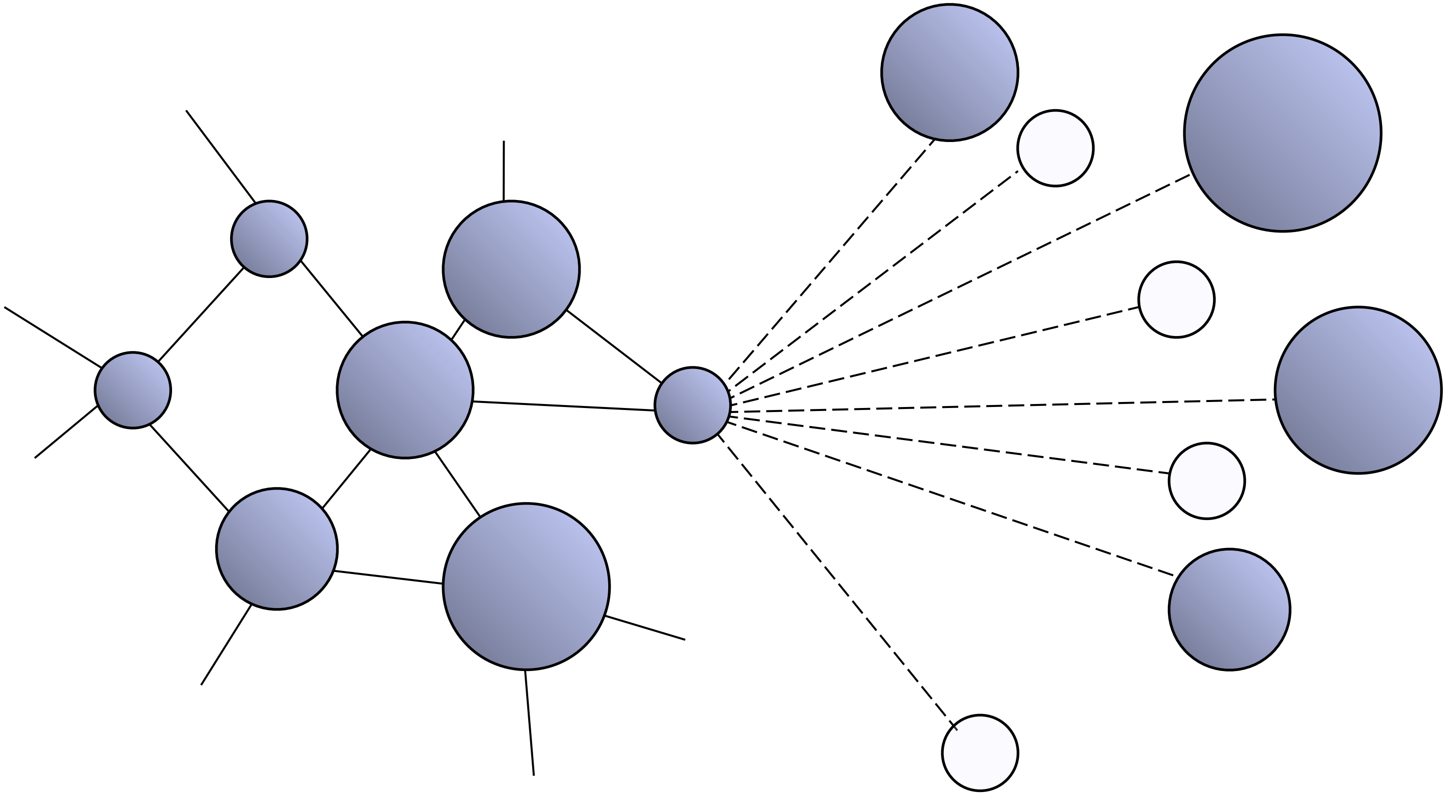
- Interest function is domain dependent but can be based on:
  - Network topology (critical points of failure)
  - Item attributes (high dollar value items)
  - Historical data (recently added items)
  - User annotation (items flagged by users)
  - Any combination of the above
- Best contextual network is the one with highest total interest.

Bonus solution : High degree nodes



- Need to avoid over-expanding the context

Use the same interest function



- Only bring in the N most interesting neighbors
- Repeat click to bring in more

Partial view layouts

- Feed the user information bit by bit
- Lend themselves well to online environments
- Easily interpretable (like NLD’s)
- Can be ‘tamed’ with appropriate Degree of Interest functions.

Summary

- Drawing small graphs with ‘nice’ structure is solved for practical cases
- Before choosing a layout, look at the structure of your graph.
  - Tree layouts you can easily implement yourself.
  - Layered layouts you should not attempt to implement yourself.
  - Force directed layouts are fun to play with.
- Scaling your network visualizations : I’ve shown 3 general categories
  - Clustering
  - Alternative Representations
  - Partial views
- Open and active research area!

Thanks!

- Questions, remarks, suggestions, monetary rewards, insults?
- <email removed>