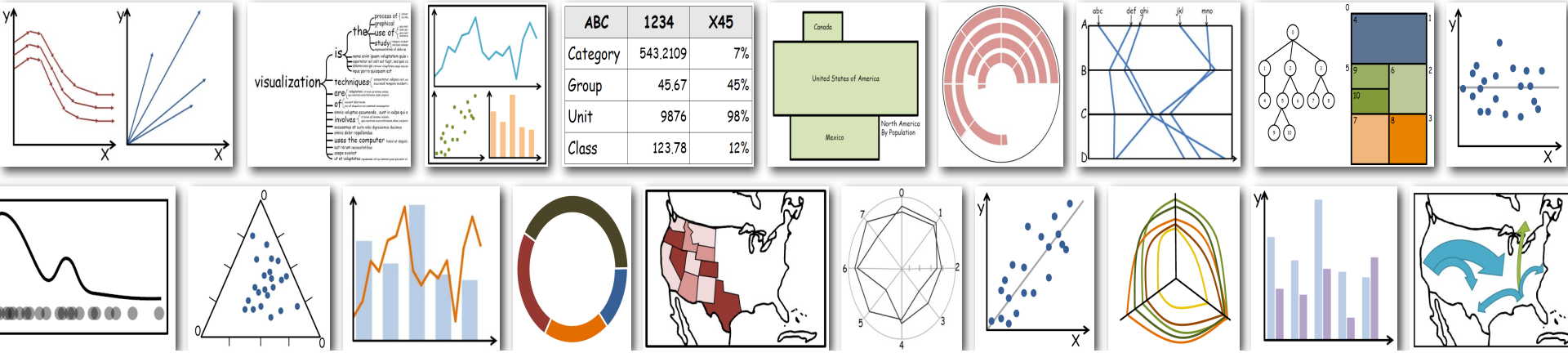


Lecture : Visualizing Graphs and Networks

DATA ANALYSIS & VISUALIZATION
FALL 2021

Dr. Muhammad Faisal Cheema
FAST-NU

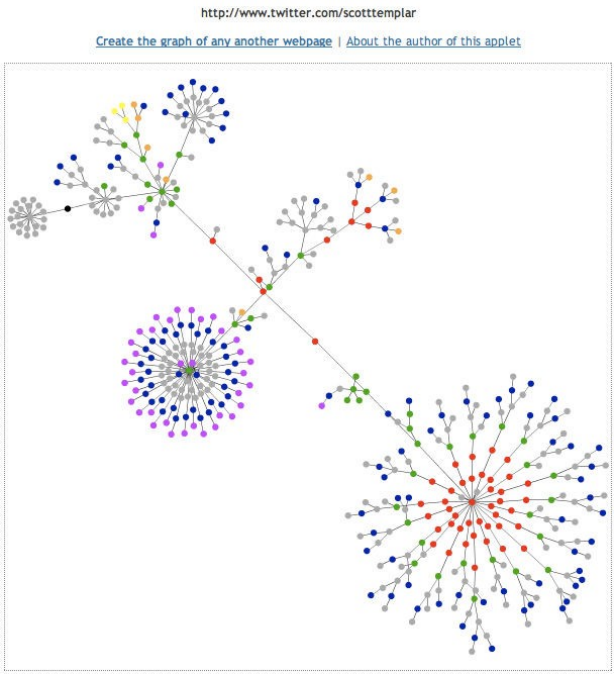
Scatterplots; dot plots; line charts, etc.

Until now, our data points were “independent of one another”

In “relational data”, it’s the **relationship between points** that matters

We live in a connected world

- Online Social networks: Facebook, Twitter ~ people connected online
- Information networks: WWW ~ web pages connected through hyperlinks
- Computer networks: The internet ~ computers and routers connected through wired/wireless connections
- What is a network? “any collection of objects in which some pairs of these objects are connected by links” [Easley and Kleinberg, 2011]



Visualizing Relations

Why relations? Isn't all data inherently relational?

- Visualising data: seeing the patterns between the data values and attributes that emerge and associate or disassociate in some way
- Visualising relations: when how one datum relates to another is an element in itself

- We want to see the overall structure of the data set
- Patterns emerge from structure as well as from values/ attributes

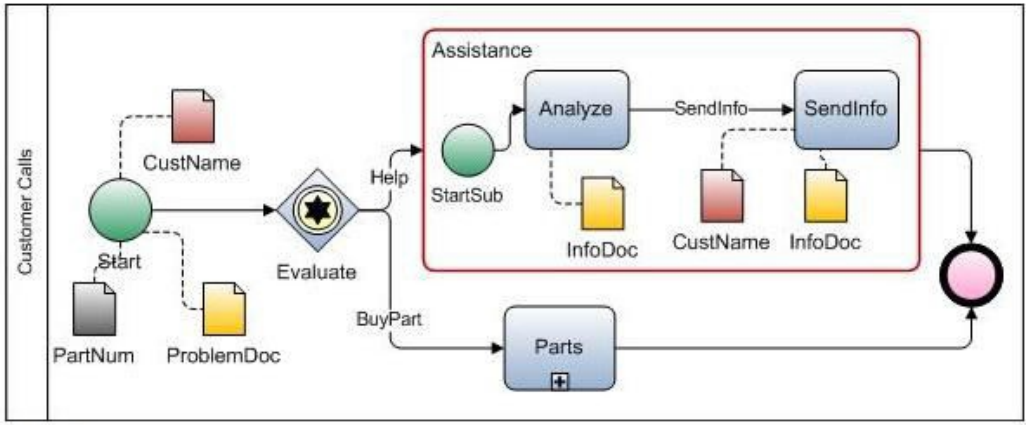
Graph and Network Uses

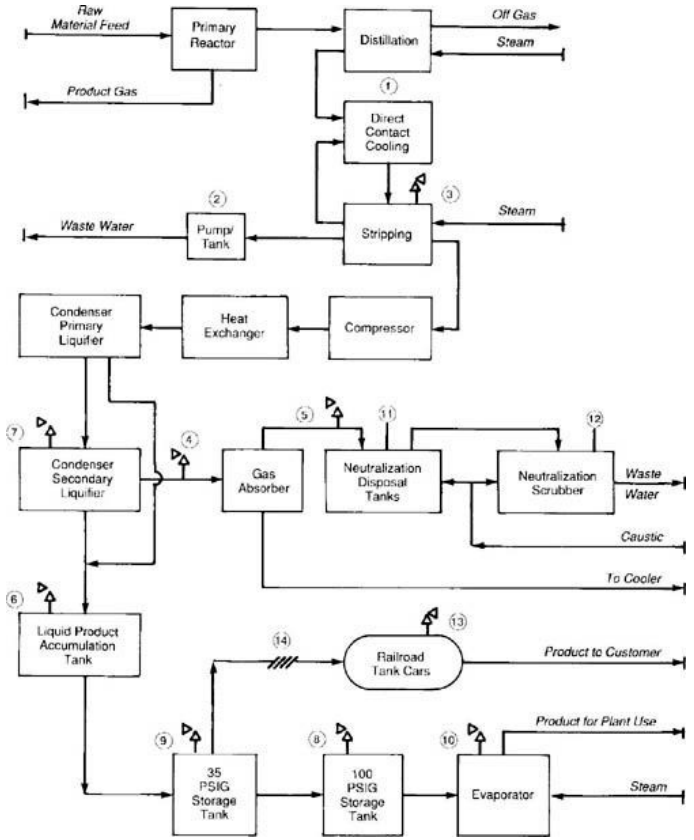
- In Information Visualization, any number of data sets can be modeled as a graph
  - Telephone system
  - World Wide Web
  - Distribution network for on-line retailer
  - Call graph of a large software system
  - Semantic map in an AI algorithm
  - Set of connected friends
  - Social Networks

More Examples…

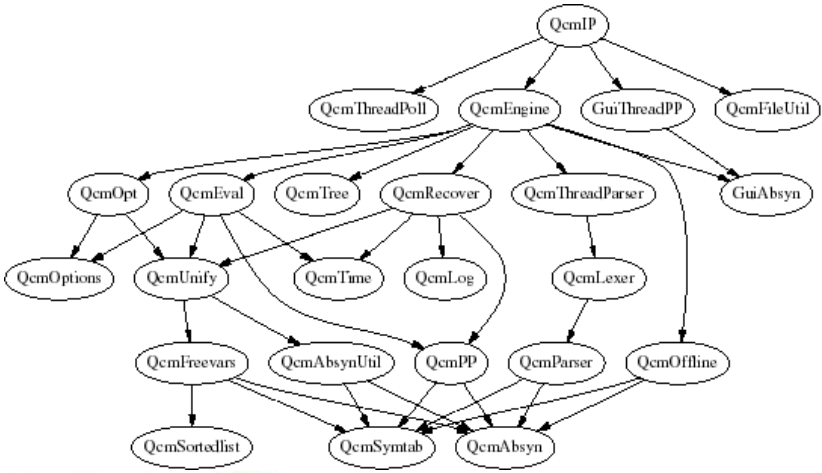
- Process Visualization (e.g., Visio)
- Dependency Graphs
- Biological Interactions (Genes, Proteins)
- Computer Networks
- Concept maps
- Ontologies
- Simulation and Modeling
- Probability maps

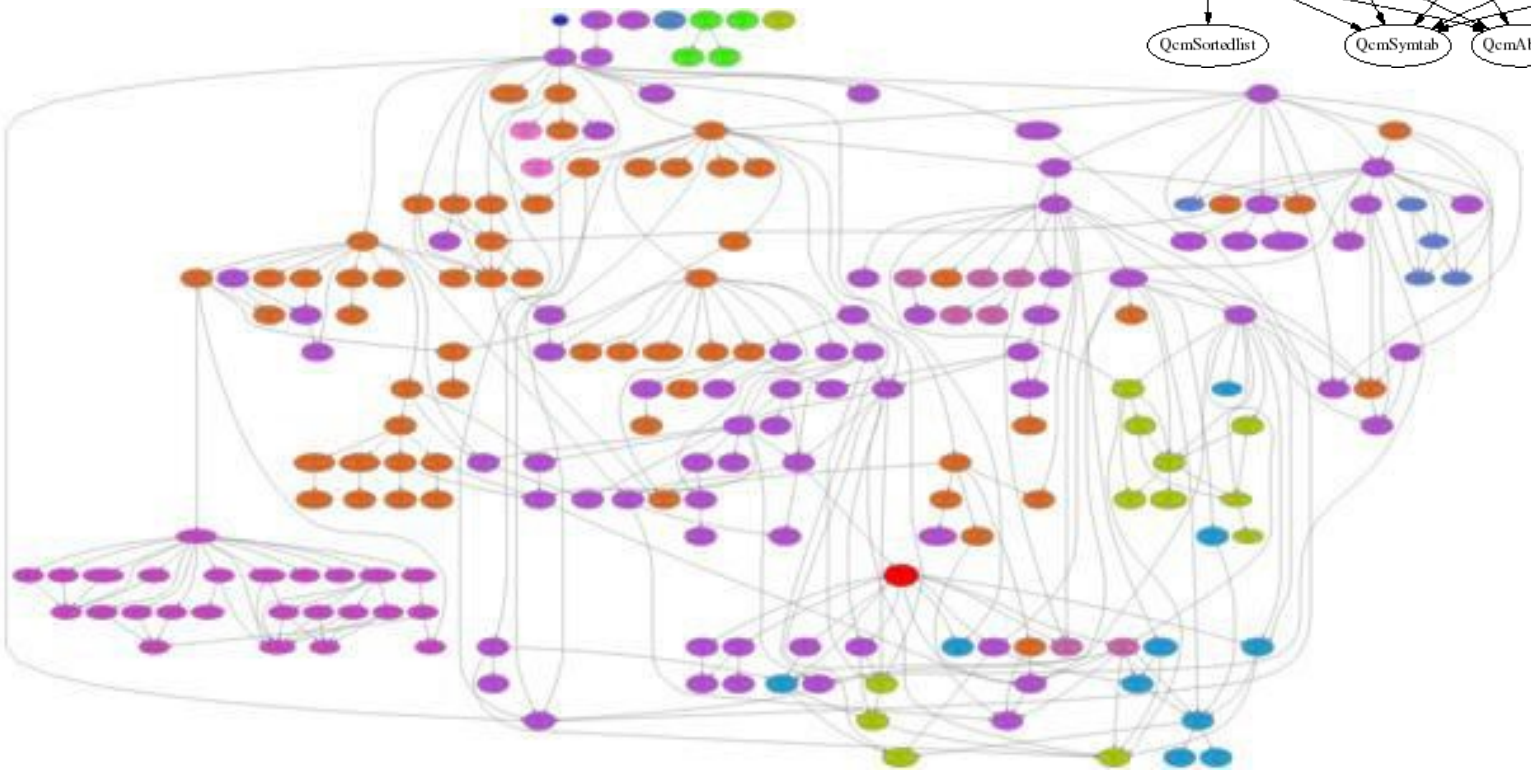
Process flow diagrams



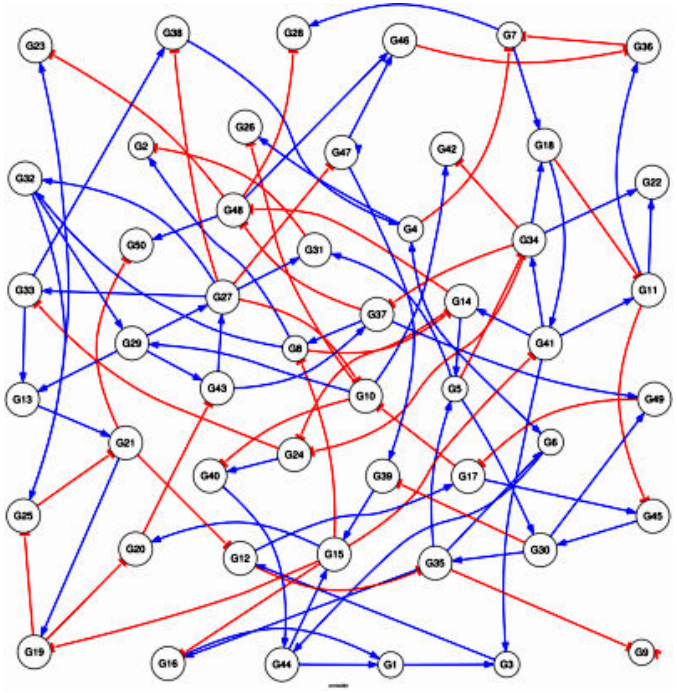


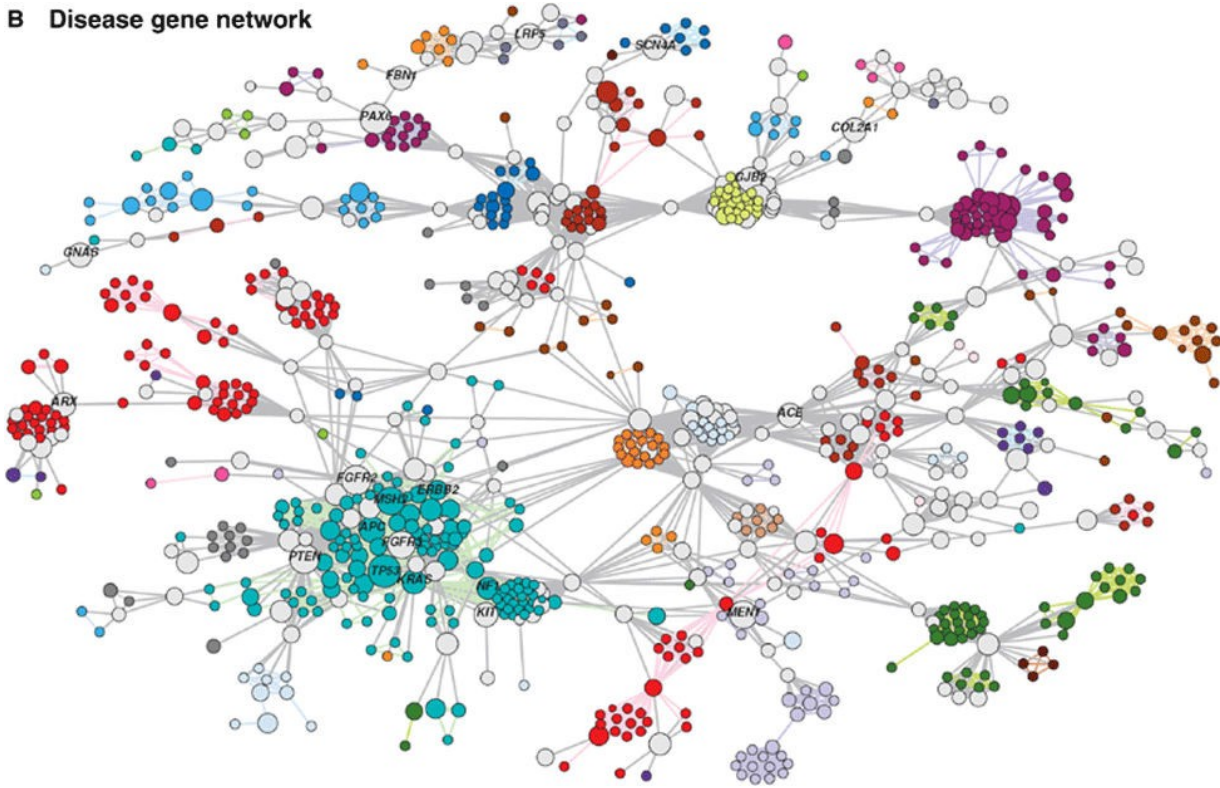
Dependency graphs



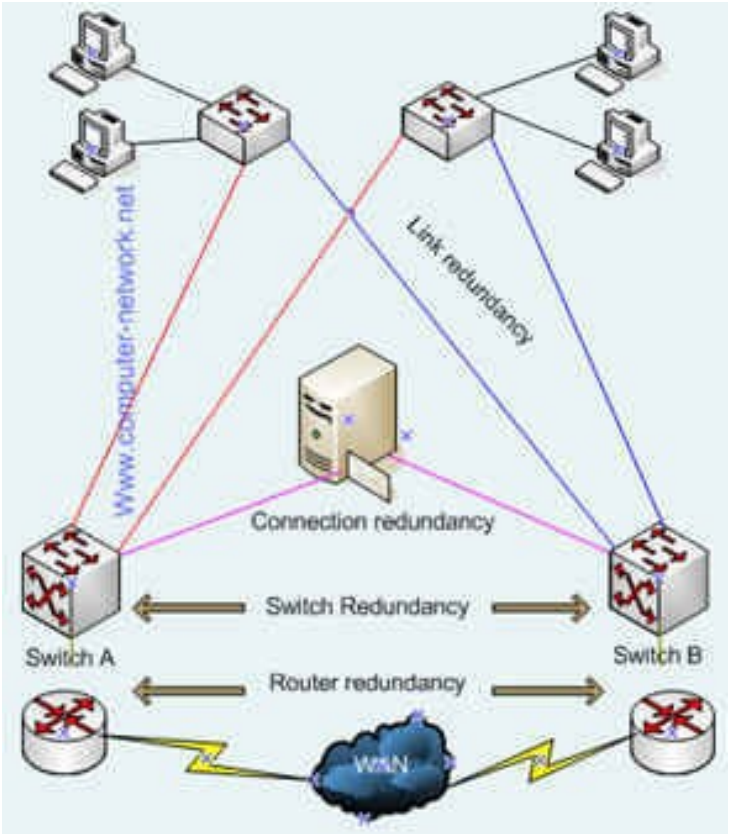


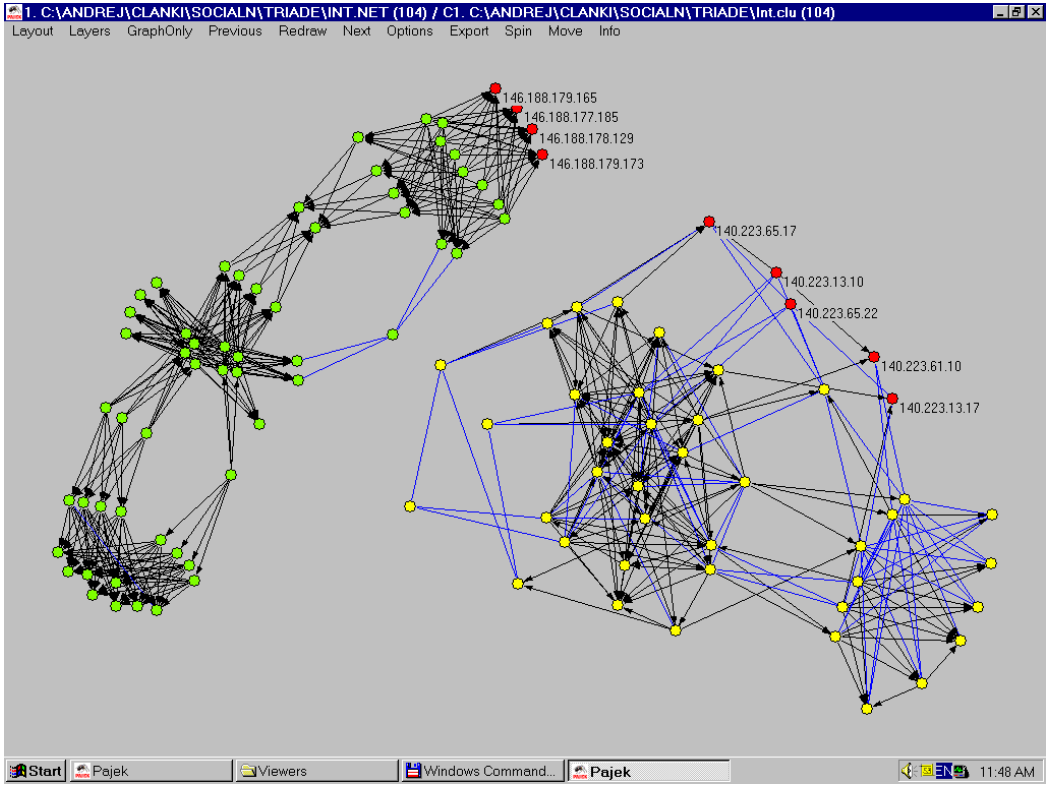
Gene networks





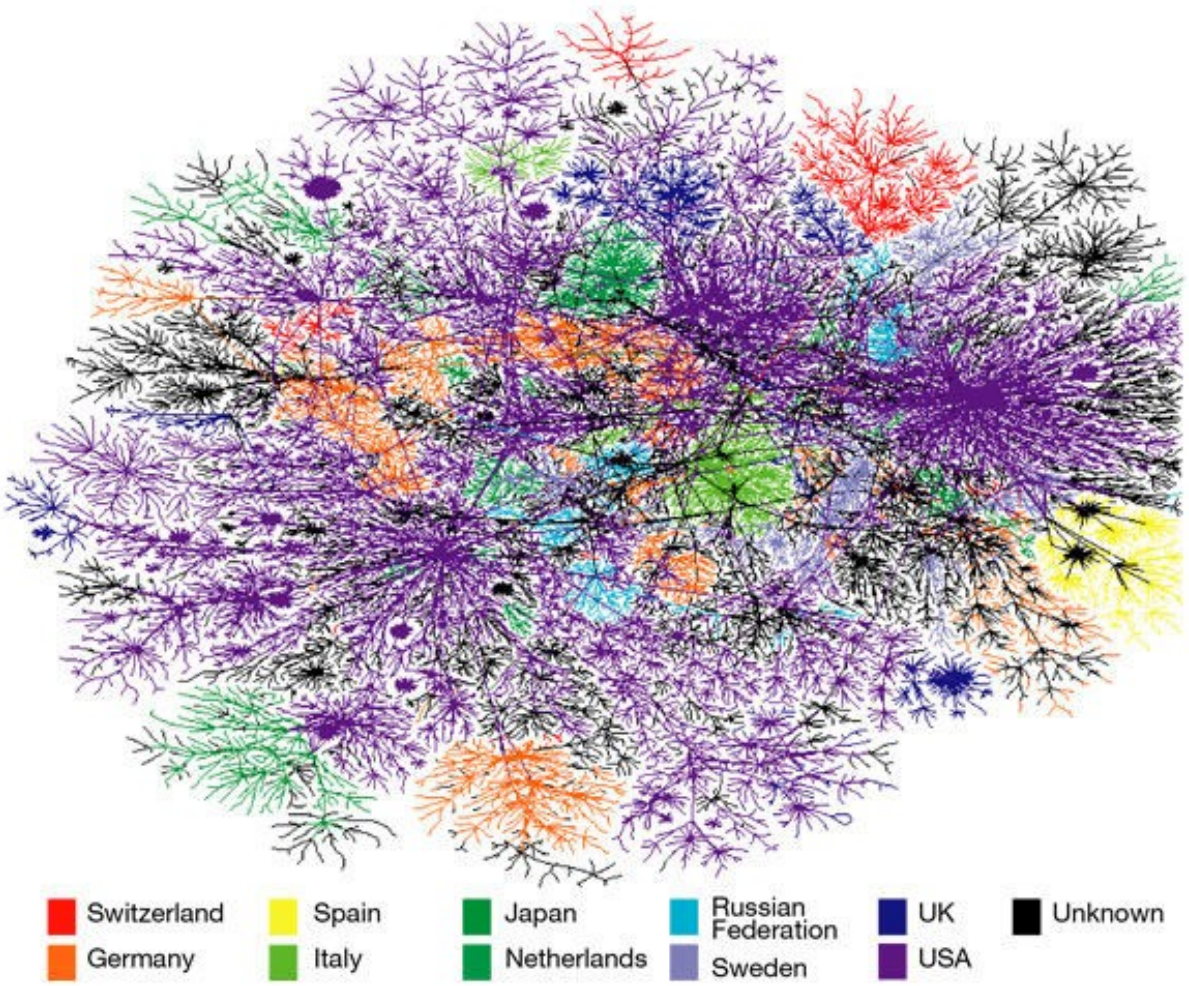
Computer networks





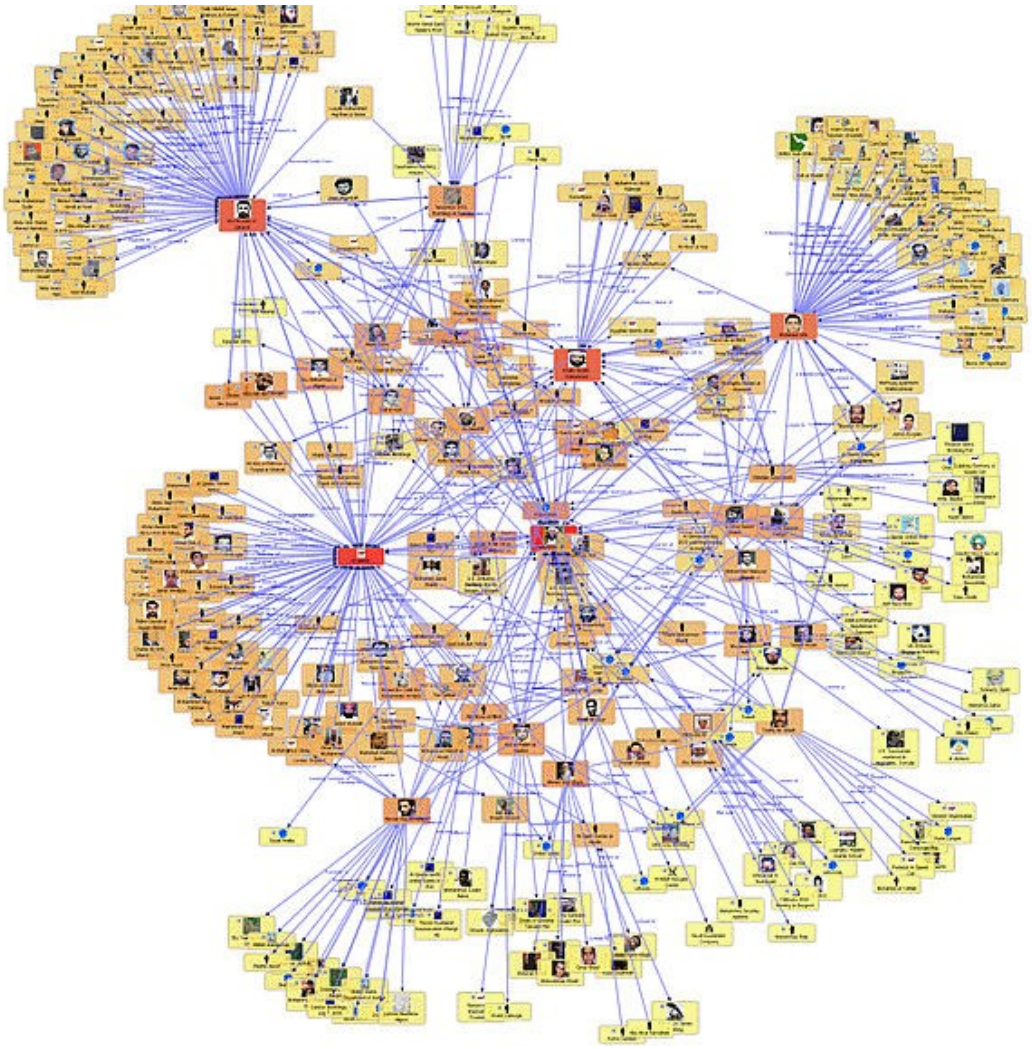
Internet ?

- What does the Internet look like
  - Email paths

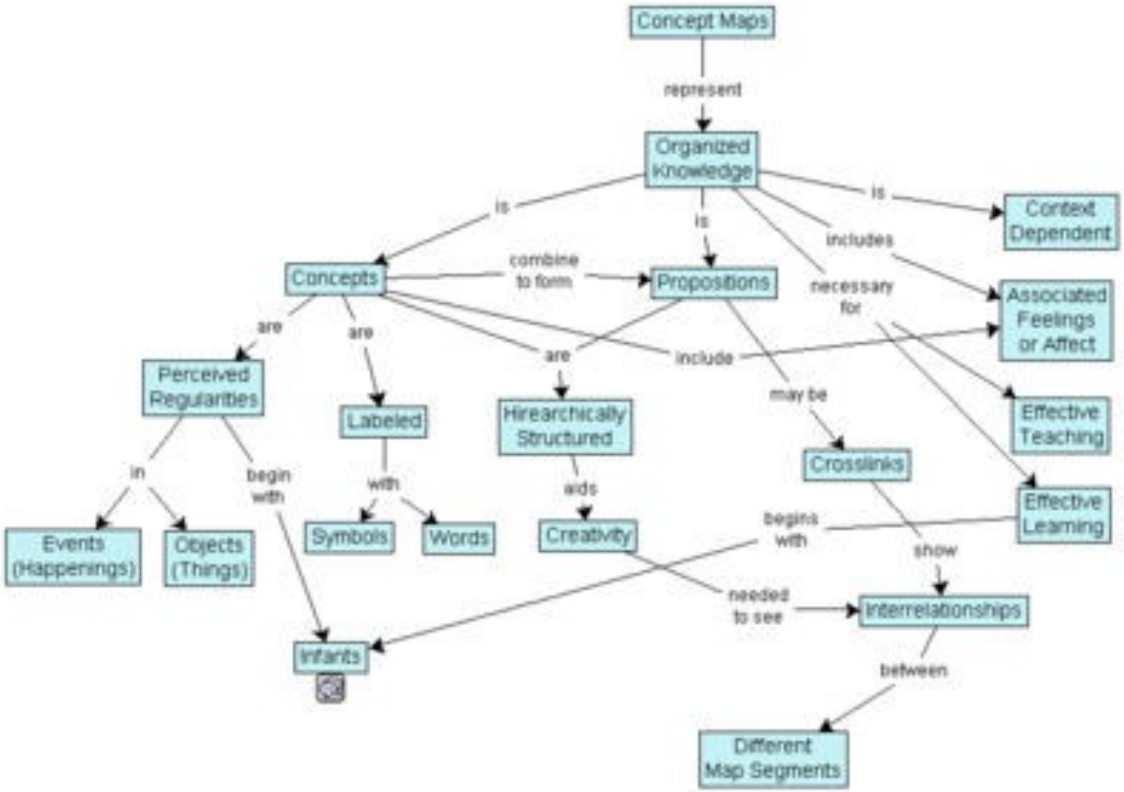


Nature, 406, 353-354(27 July 2000)

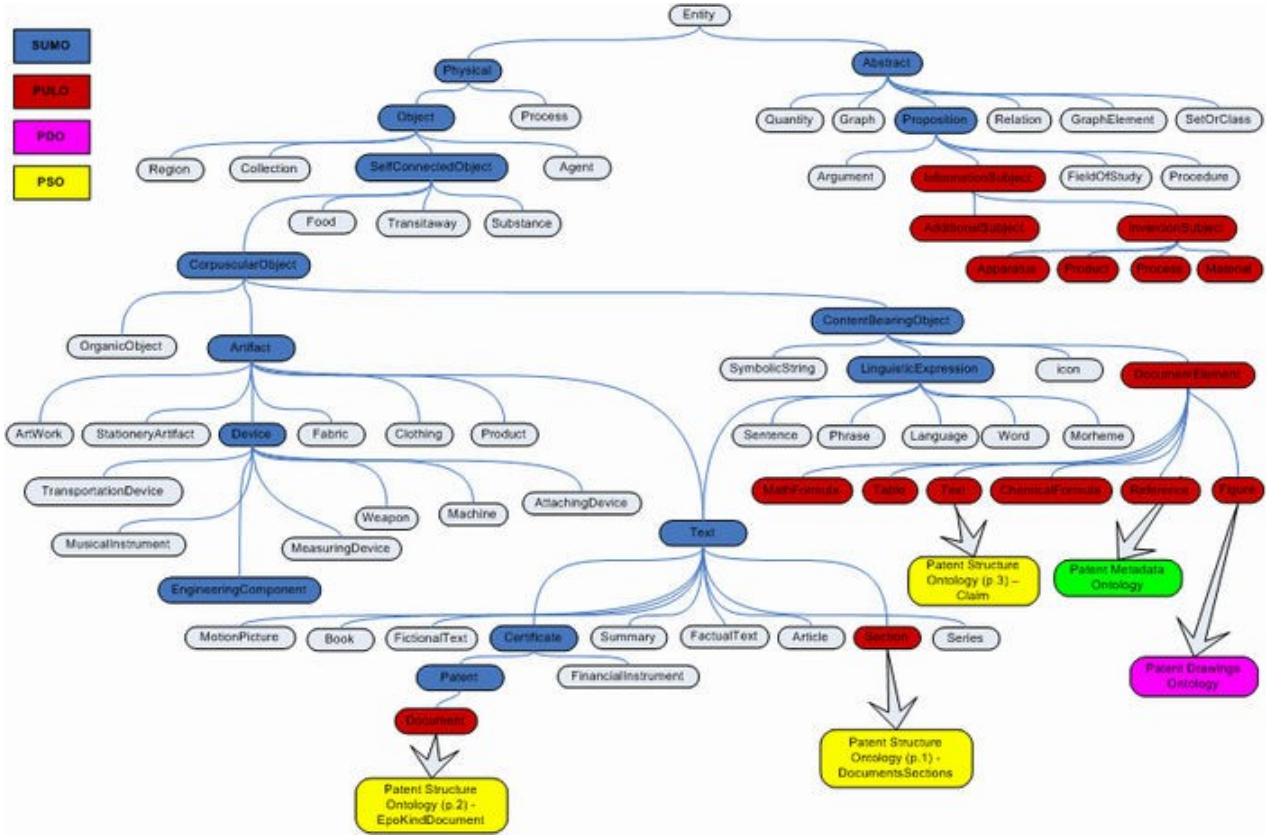
Social networks



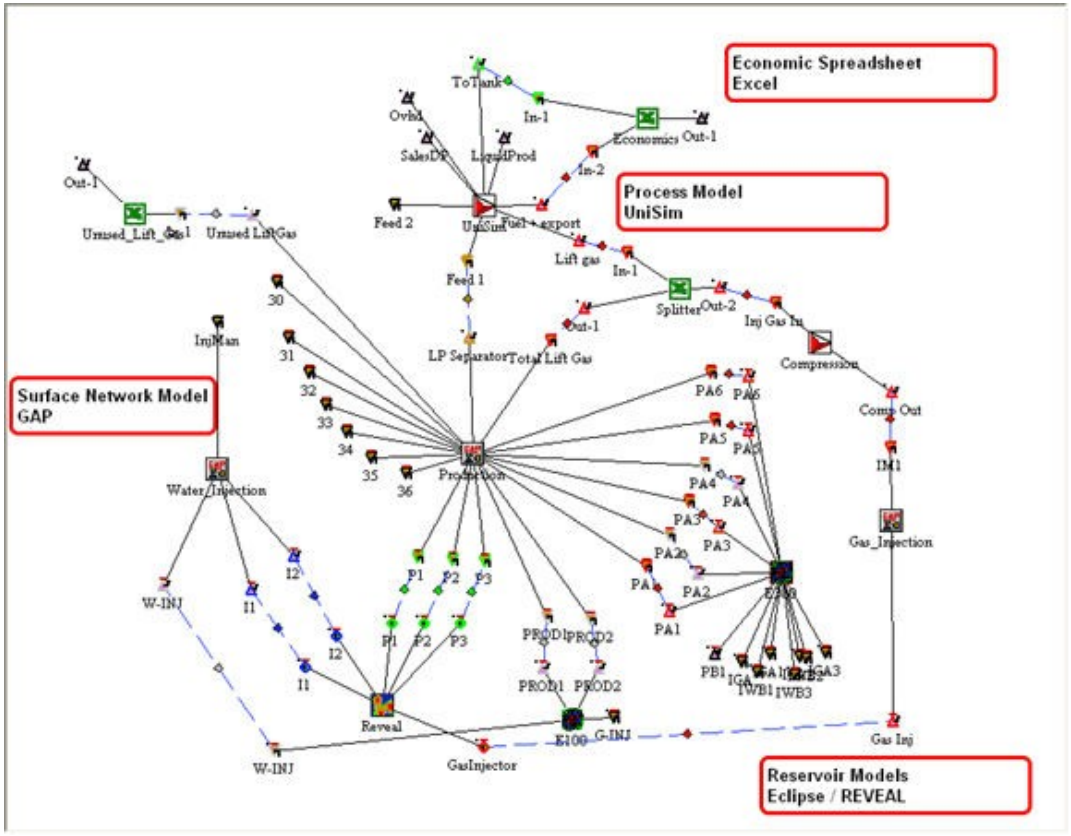
Concept maps



Ontologies

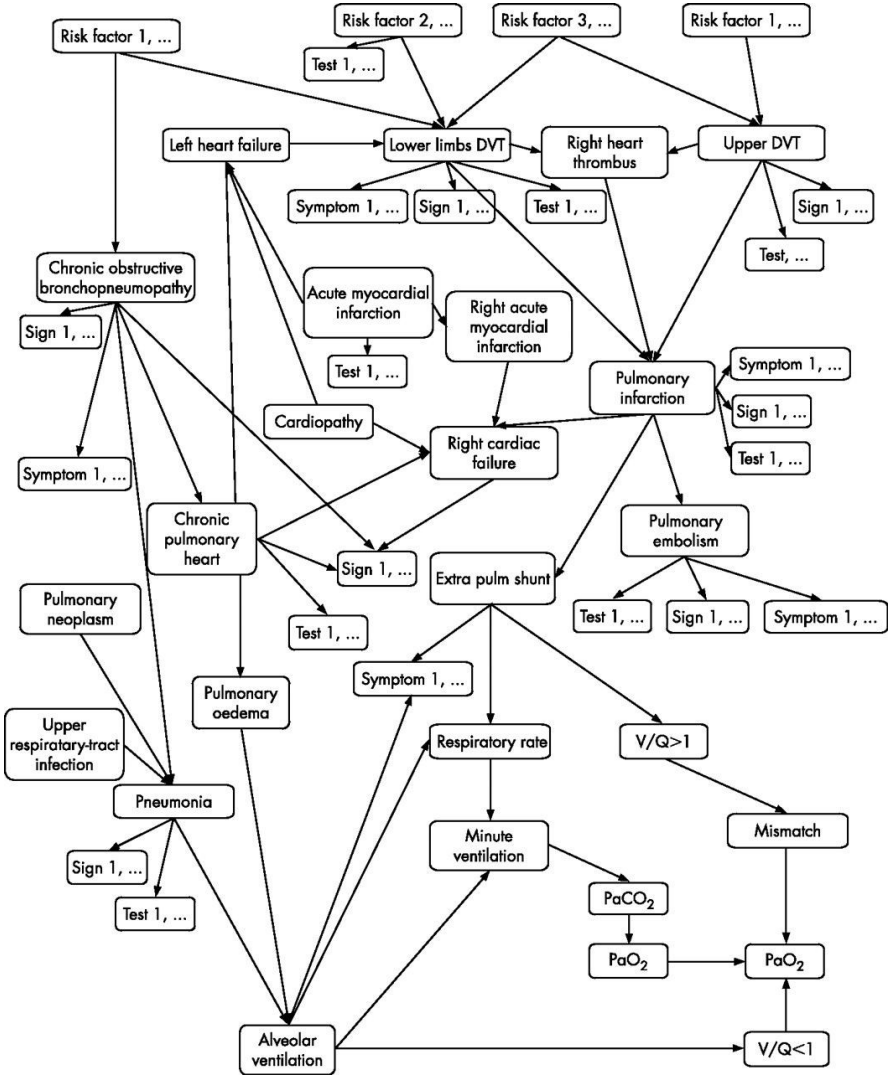

SUMO
PULO
PDO
PSO
Entity
Physical
Abstract
Object
Process
Quantity
Graph
Proposition
Relation
GraphElement
SetOrClass
Region
Collection
SelfConnectedObject
Agent
Argument
InformationSubject
FieldOfStudy
Procedure
Food
Transitaway
Substance
AdditionalSubject
InventionSubject
Apparatus
Product
Process
Material
CorpuscularObject
ContentBearingObject
OrganicObject
Artifact
SymbolicString
LinguisticExpression
icon
DocumentElement
ArtWork
StationeryArtifact
Device
Fabric
Clothing
Product
Sentence
Phrase
Language
Word
Morheme
TransportationDevice
Weapon
Machine
AttachingDevice
MathFormula
Table
Text
ChemicalFormula
Reference
Figure
MusicalInstrument
MeasuringDevice
Text
Patent Structure Ontology (p.3) – Claim
Patent Metadata Ontology
EngineeringComponent
MotionPicture
Book
FictionalText
Certificate
Summary
FactualText
Article
Section
Series
Patent
FinancialInstrument
Patent Drawings Ontology
Document
Patent Structure Ontology (p.1) - DocumentsSections
Patent Structure Ontology (p.2) - EpoKindDocument

Simulation graphs

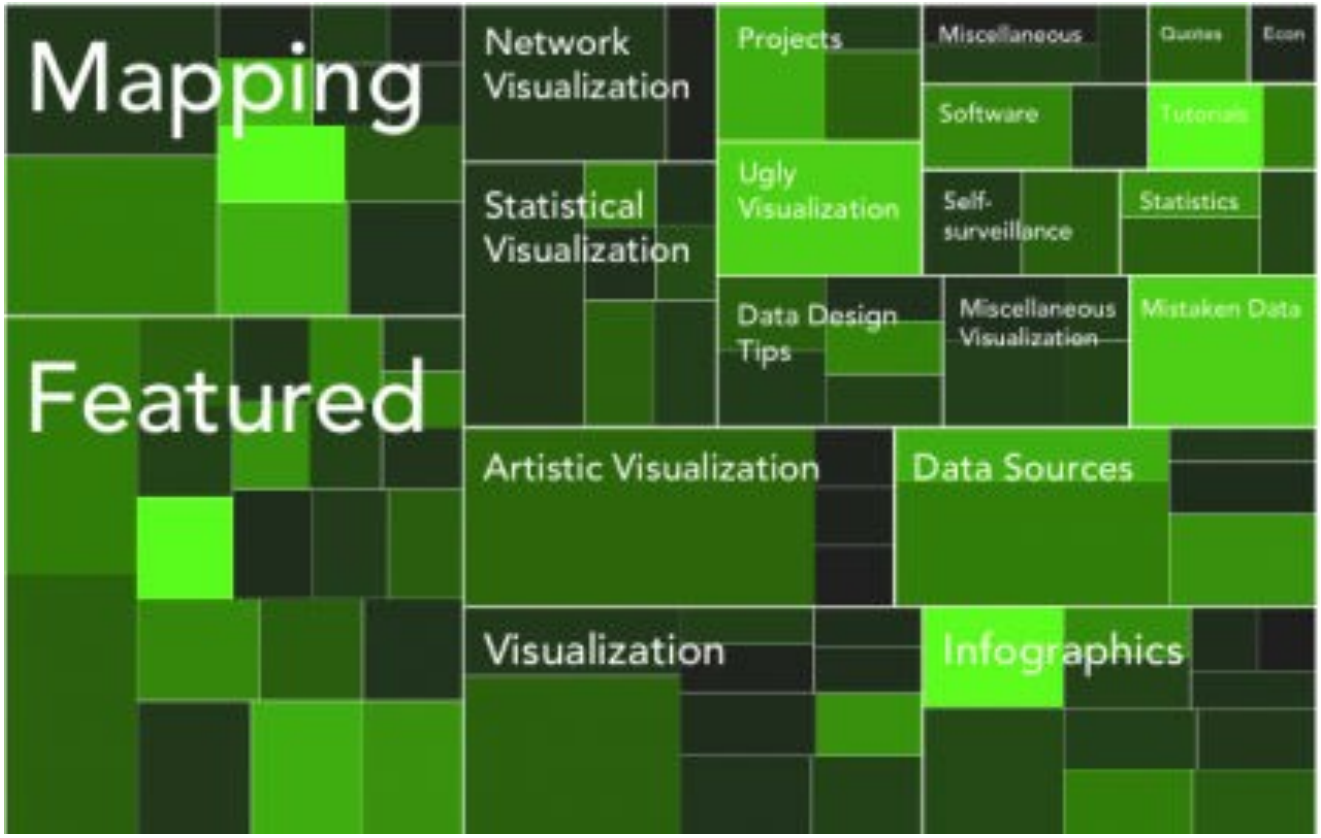


Probability maps

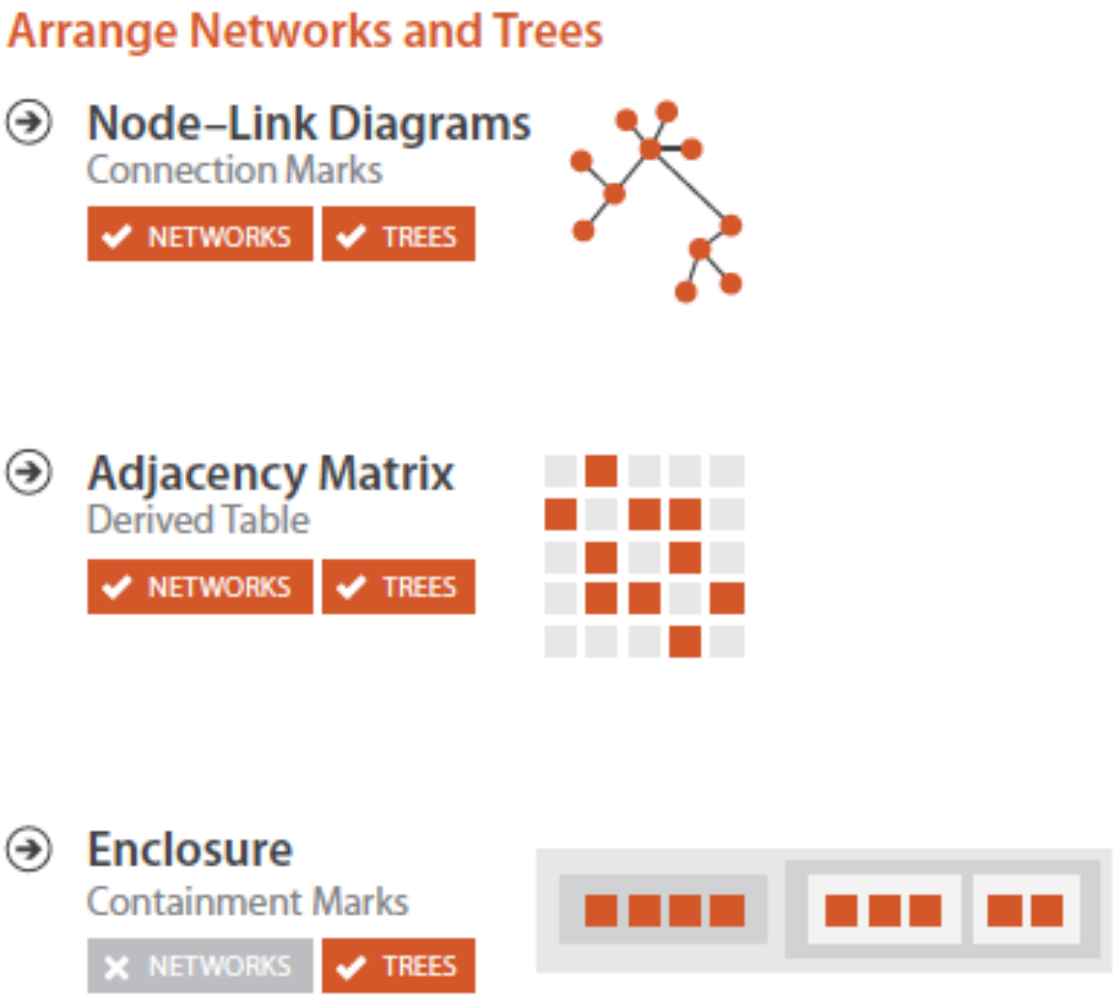
- Bayesian network



Is this a network?



- Several ways to represent relations
- Choice depends on relation, task and scale

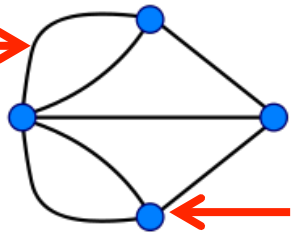


T. Munzner, Visualization Analysis and Design.

Graphs as Network Models

- Graphs are only ONE way to represent networks
  - Most powerful
  - Most popular
- Graphs are well suited for topology-related problems
  - Distance through the network (hops), propagation, clusters defined by connectivity

{ arc , link , edge }



{ node , vertex }

points	lines	
vertices	edges, arcs	math
nodes	links	computer science
sites	bonds	physics
actors	ties, relations	sociology

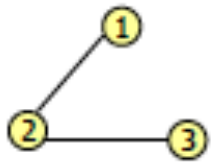
Source: L. Adamic SNA class @coursera

A formal definition

- A graph is a way of specifying **relationships** among a **collection** of **items**.
- A graph consists of a **set of objects**, called **nodes** or **vertices**, with certain pairs of these **objects** connected by **links** called **edges**.
  - Easley and Kleinberg, 2011

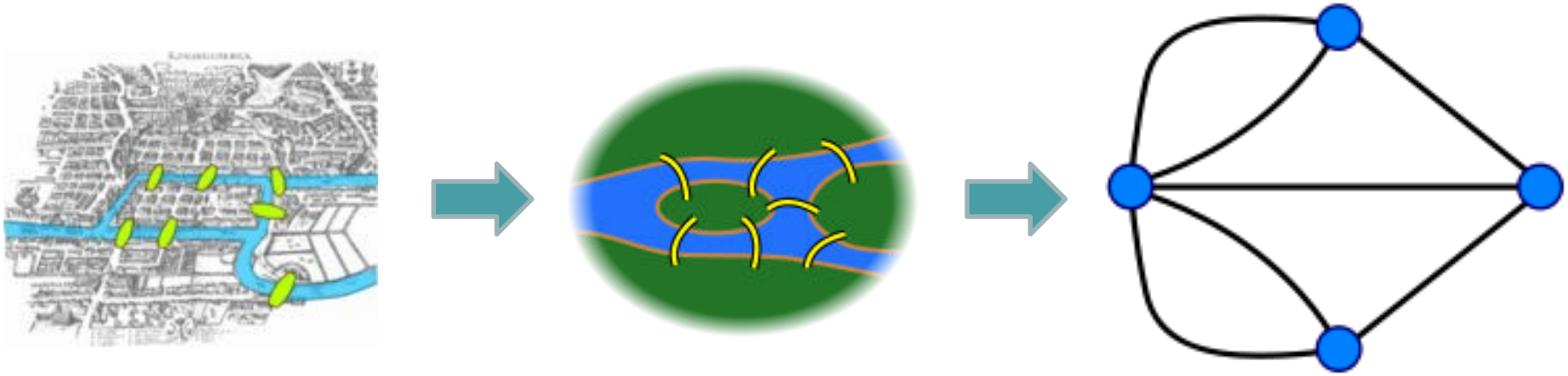
	1	2	3
1	0	1	0
2	1	0	1
3	0	1	0

Adjacency matrix



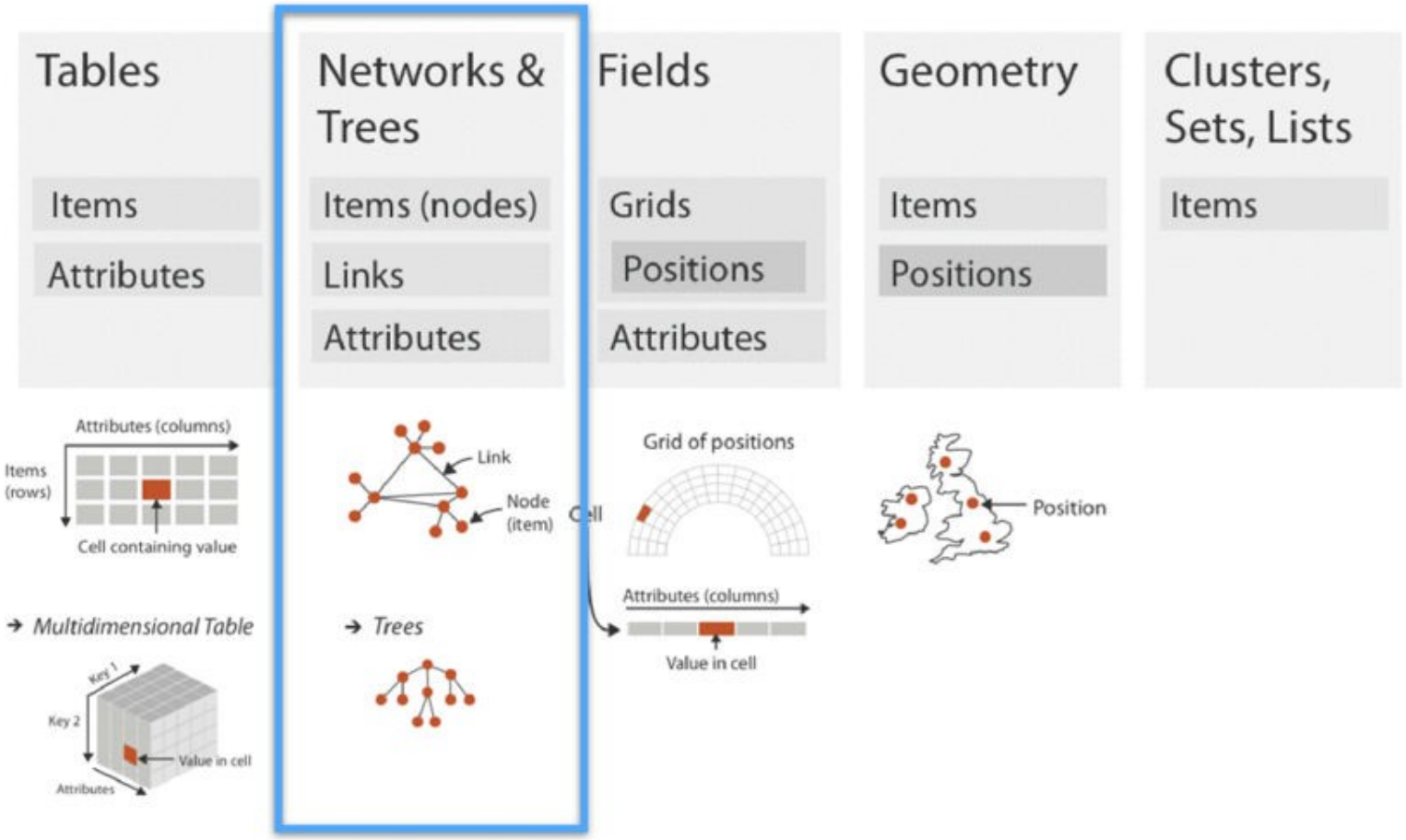
A bit of history: Graph models

- Around 1735, the mathematician Venn Euler set the foundation for graph theory by creating a model to represent the problem of the “7 bridges of Königsberg”



Source: http://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg and “Linked” by A-L. Barabasi

Why graphs and hierarchies?

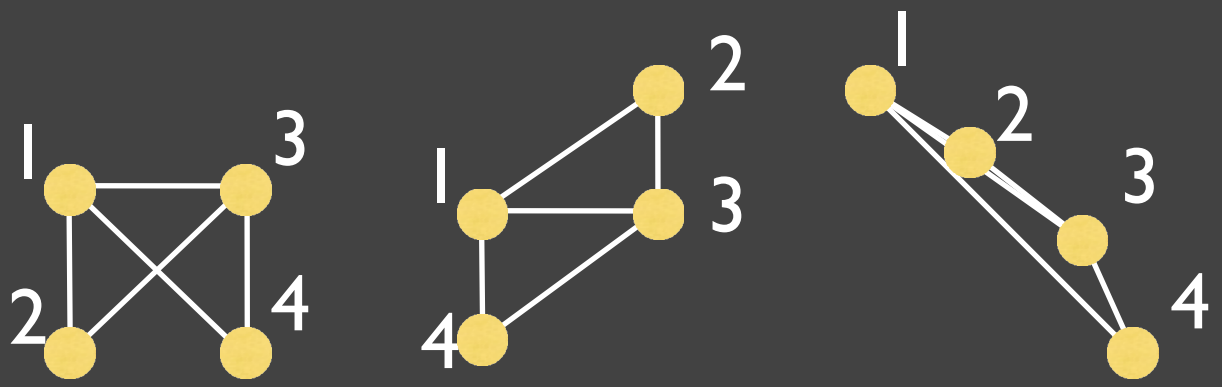


http://www.sci.utah.edu/~miriah/cs6630/lectures/L13-trees-graphs.pdf

DEFINITIONS

GRAPH

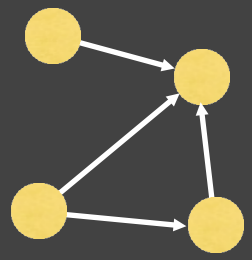
- **A graph *G* consists of a collection of vertices (or nodes) *V* and a set of edges *E*, consisting of vertex pairs.**
- **An edge e_{xy} = *(x,y)* connects two vertices *x* and *y*.**
  - for example: V={1,2,3,4}, E={(1,2),(1,3),(2,3),(3,4),(4,1)}



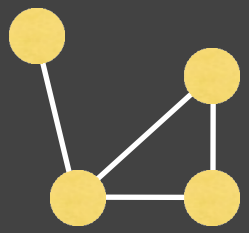
Graph Terminology

- **Graphs can have cycles**
- Edges can be directed or undirected
- Degree of a vertex = # connected nodes
  - In-degree and out-degree for directed graphcs
- Graph edges can have values (weights)
  - Nominal (N), ordinal (O), quantitative (Q)

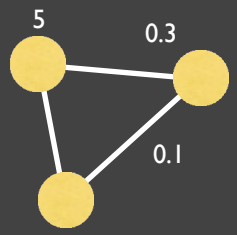
a bunch of definitions



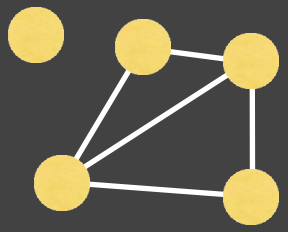
A directed graph



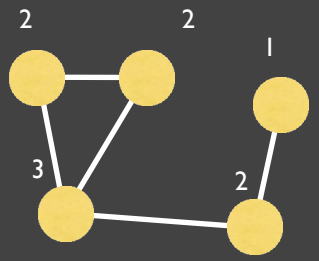
An undirected graph



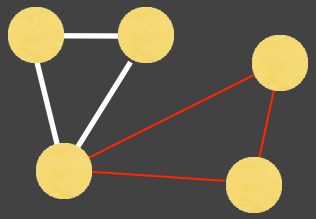
Weighted



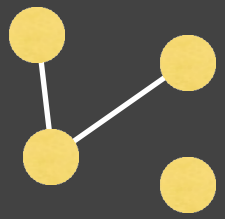
Unconnected



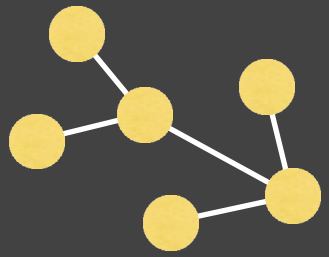
Node degrees



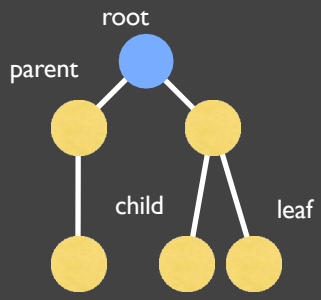
A cycle



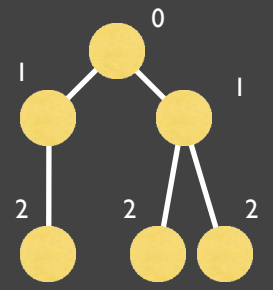
An acyclic graph



A connected acyclic graph,
a.k.a. a **tree**



A rooted tree
or hierarchy



Node depths

Graphs are more complicated than trees

Graph Drawing considerations

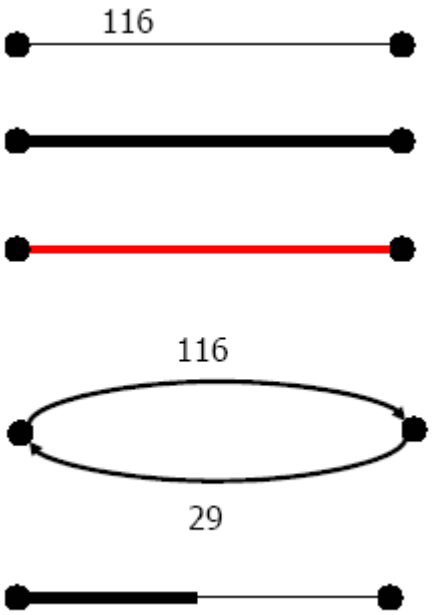
Vertex Issues

- Shape
- Color
- Size
- Location
- Label

Edge Issues

- Color
- Size
- Label
- Form
- Polyline, straight line, orthogonal, grid, curved, planar, upward/downward, ...

Edge Drawing Strategies



Label

Thickness

Color

Directed

Complexity Considerations

- **Crossings**-- minimize towards planar
- **Total Edge Length**-- minimize towards proper scale
- **Area**-- minimize towards efficient use of space
- **Maximum Edge Length**-- minimize longest edge
- **Uniform Edge Lengths**-- minimize variances
- **Total Bends**– minimize orthogonal towards straight-line

Graph Visualization Problems

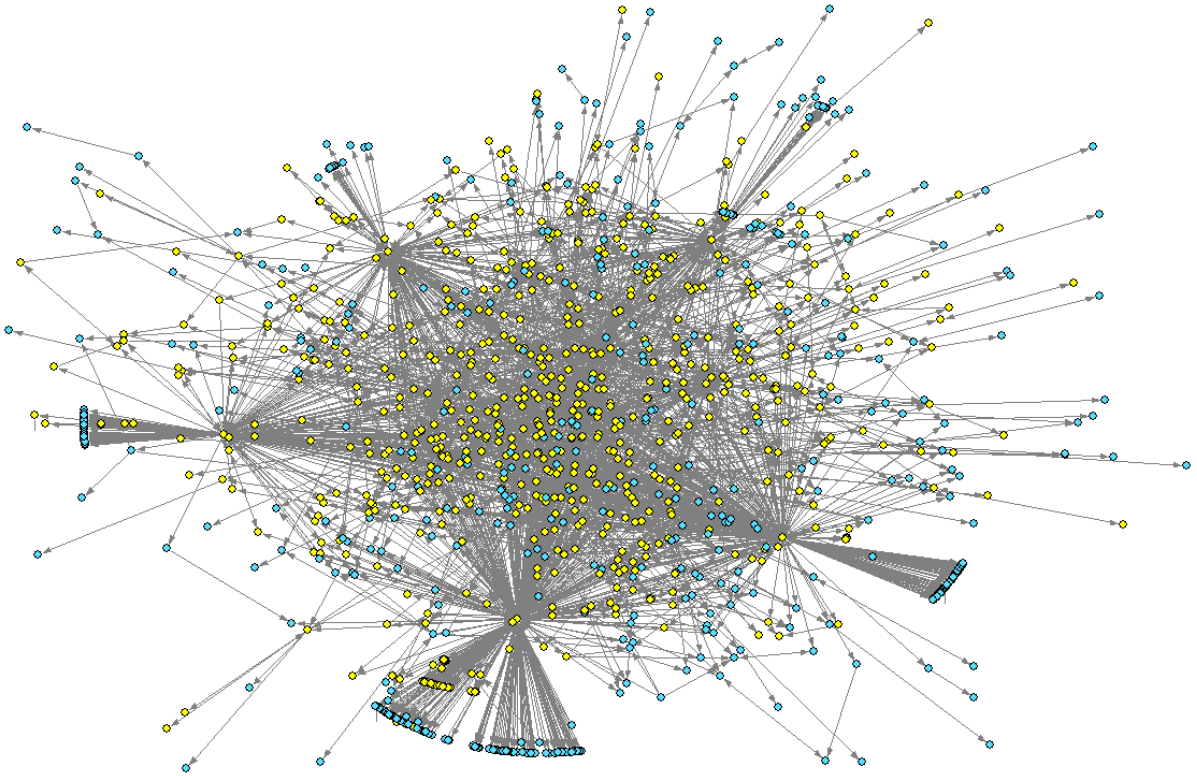
- Graph layout and positioning
  - Make a concrete rendering of abstract graph
- Scale
  - Not too much of a problem for small graphs, but large ones are much tougher
- Navigation/Interaction
  - How to support user changing focus and moving around the graph

Graph Layout

- How to position the nodes and edges?
  - Avoid clutter
  - Maintain appropriate relations

The Hairball Problem

- How to position the nodes and edges?
  - Avoid clutter
  - Maintain appropriate relations



Layout Subproblems

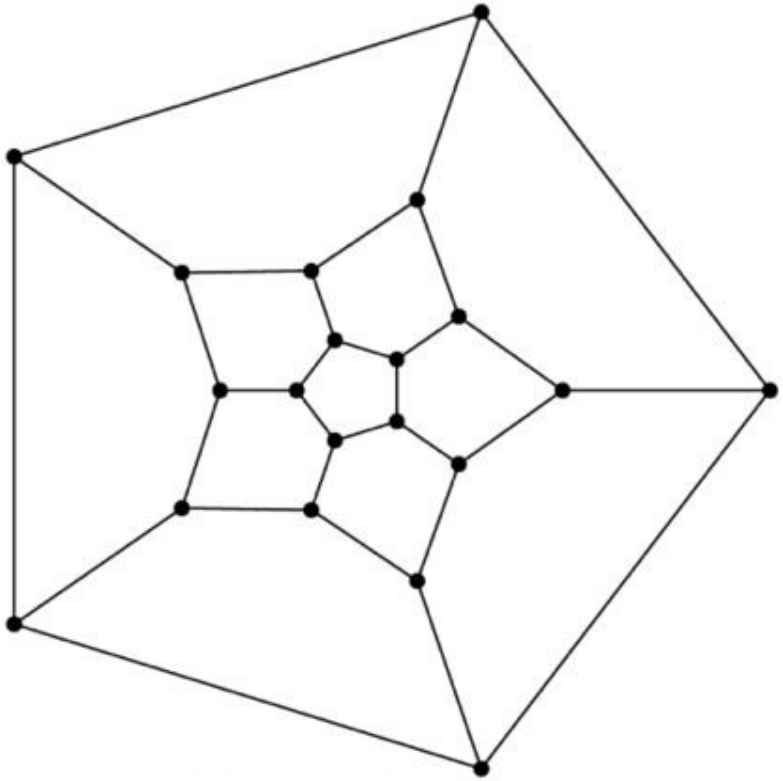
- Rank Assignment
  - Compute which nodes have large degree, put those at center of clusters
- Crossing Minimization
  - Swap nodes to rearrange edges
- Subgraph Extraction
  - Pull out cluster of nodes
- Planarization
  - Pull out a set of nodes that can lay out on plane

Graph Visualization
vs
Graph Drawing

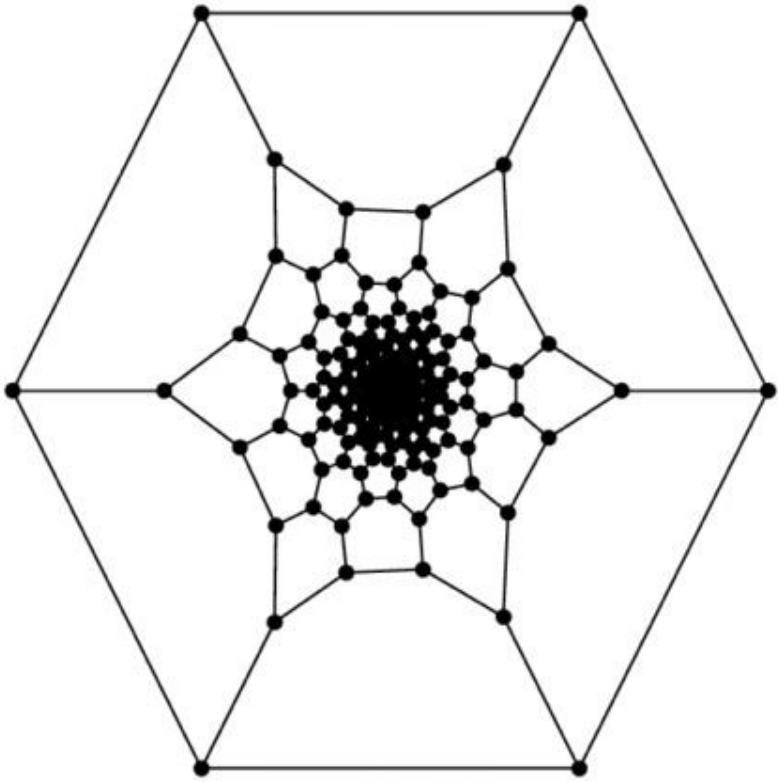
- Need to be careful with graph drawing algorithms
  - Some layouts may be mathematically correct but produce visual clutter
- In visualization, visual clutter makes the visualization useless.

Tutte Embedding

for planer layouts



Dodecahedron



Le(*C*60)

Follows Tutte Embedding correctly, but visually cluttered

http://www.cs.arizona.edu/~kpavlou/Tutte_Embedding.pdf

Graph Visualization

A primary concern of tree/graph visualization is the spatial arrangement of nodes and edges.

Often (but not always) the goal is to effectively depict the graph structure:

- Connectivity, path-following
- Topological distance
- Clustering / grouping
- Ordering (e.g., hierarchy level)

Major Graph visualization Techniques

- Node-Link Graph Layout
  - Sugiyama-Style Layout
  - Force-Directed Layout
- Alternatives to Node-Link Diagrams
  - Matrix Diagrams
- Attribute-Driven Layout & Hive Plots

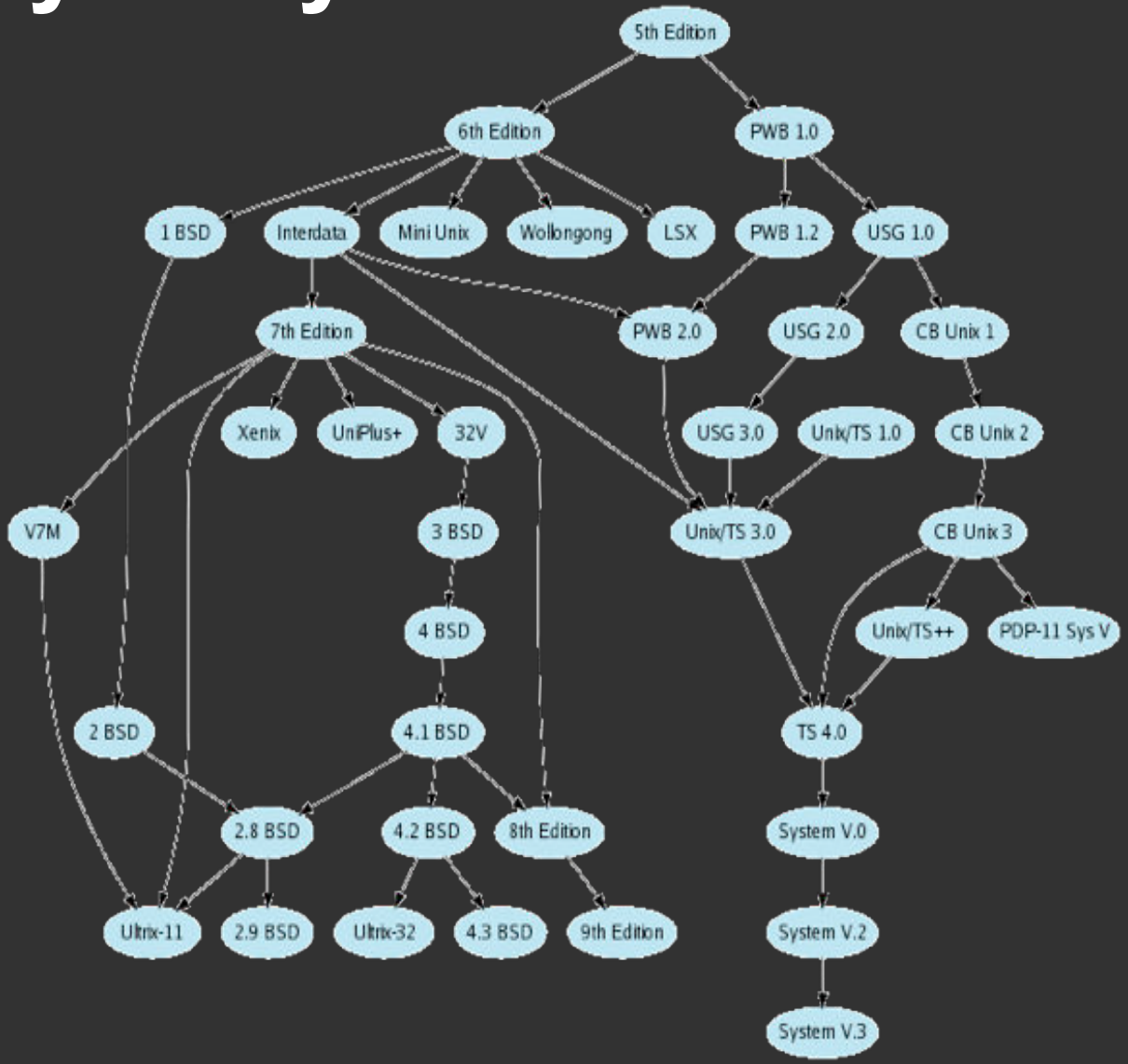
Node-Link Graph Layout

Sugiyama-Style Layout

Sugiyama-Style Layout

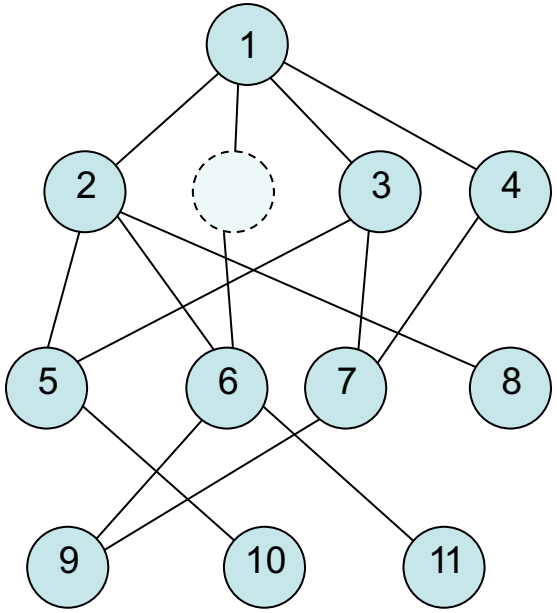
Evolution of the UNIX operating system

Hierarchical layering based on descent



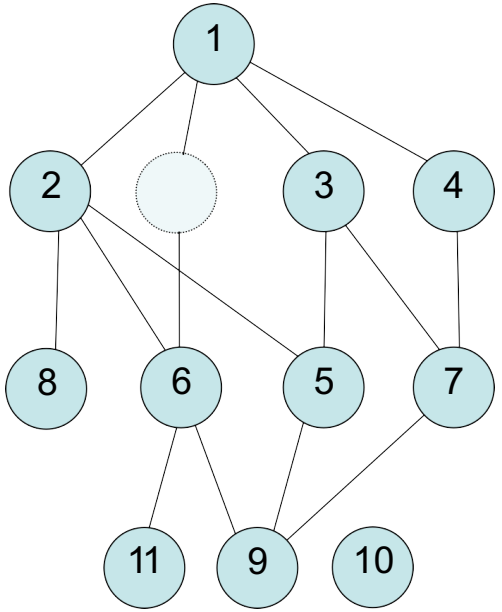
SUGIYAMA STEP 1

- **create layering of graph**
  - from domain specific knowledge
  - longest path from root
  - algorithmically determine best layering (NP-Hard)
- **dummy nodes for long edges**



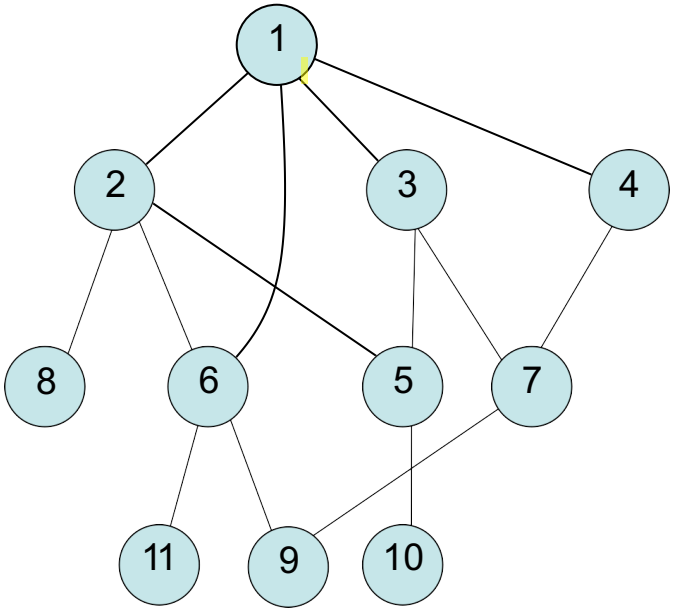
SUGIYAMA STEP 2

- **minimize crossings layer by layer (NP-hard)**
- **numerous heuristics available**

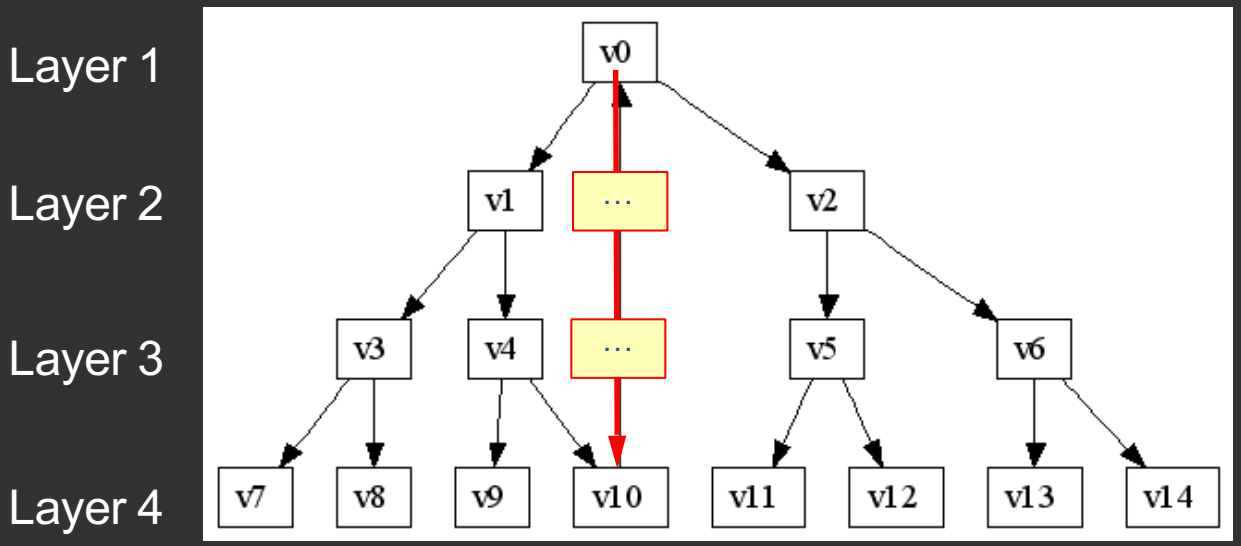


SUGIYAMA STEP 3

- **final assignment of x-coordinates**
- **routing of edges**



Sugiyama-Style Layout



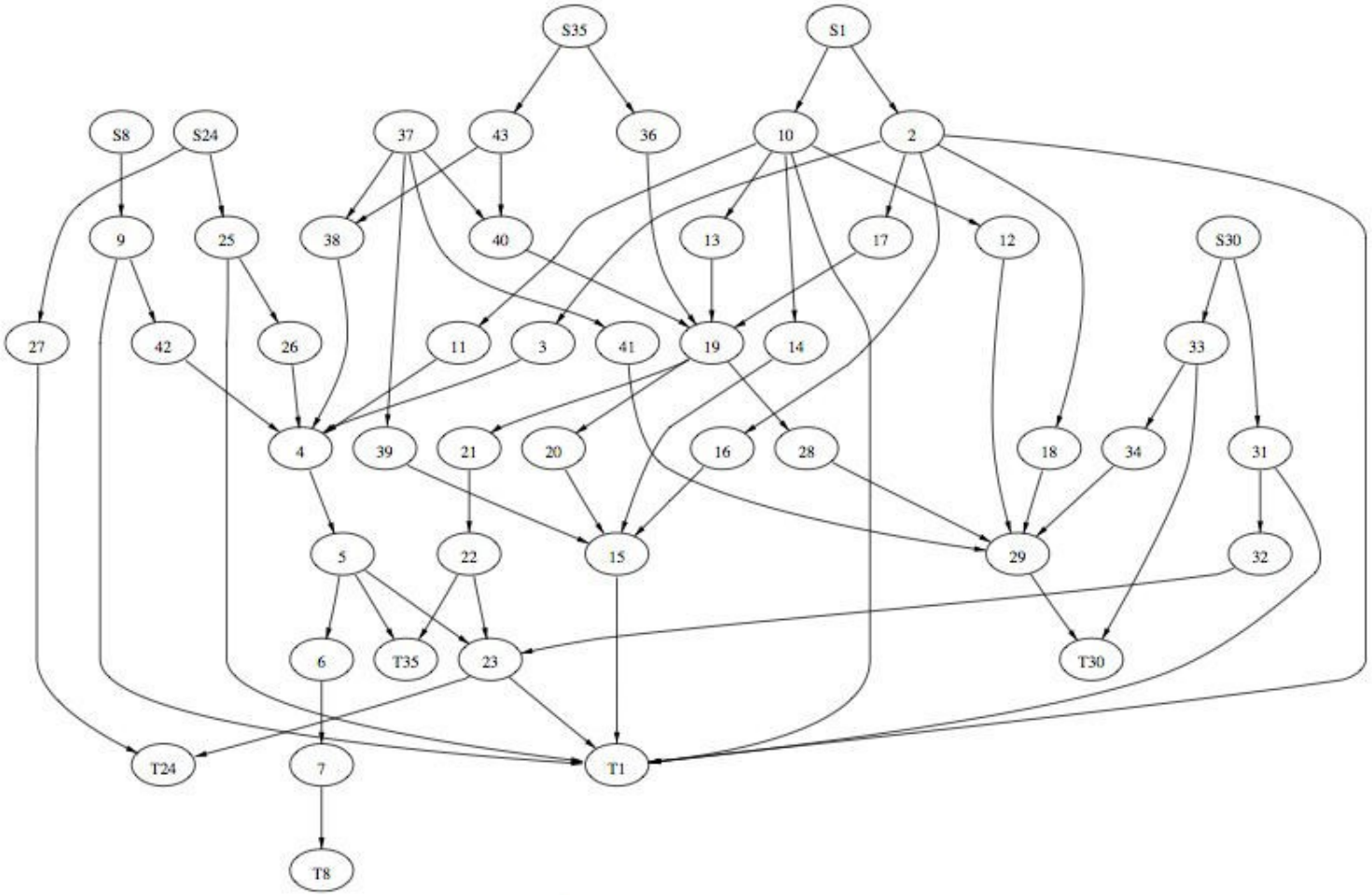
Reverse edges to remove cycles

Assign nodes to hierarchy layers

Create dummy nodes to “fill in” missing layers

Arrange nodes within layer, minimize edge crossings

Route edges –layout splines if needed



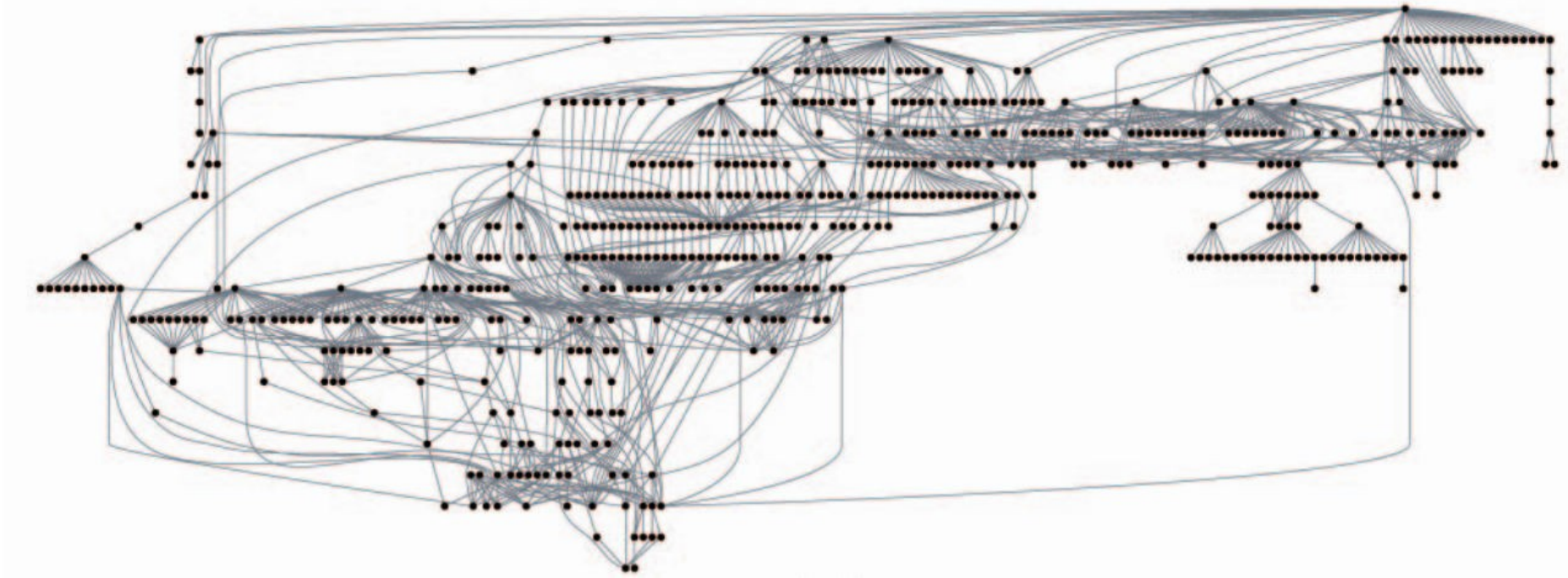
Gansner 1993

SUGIYAMA

+ nice, readable top down flow
+ relatively fast (depending on heuristic used for crossing minimization)

- not really suitable for graphs that don't have an intrinsic top down structure
- hard to implement

Produces Hierarchical Layouts



Sugiyama-style layout emphasizes hierarchy.

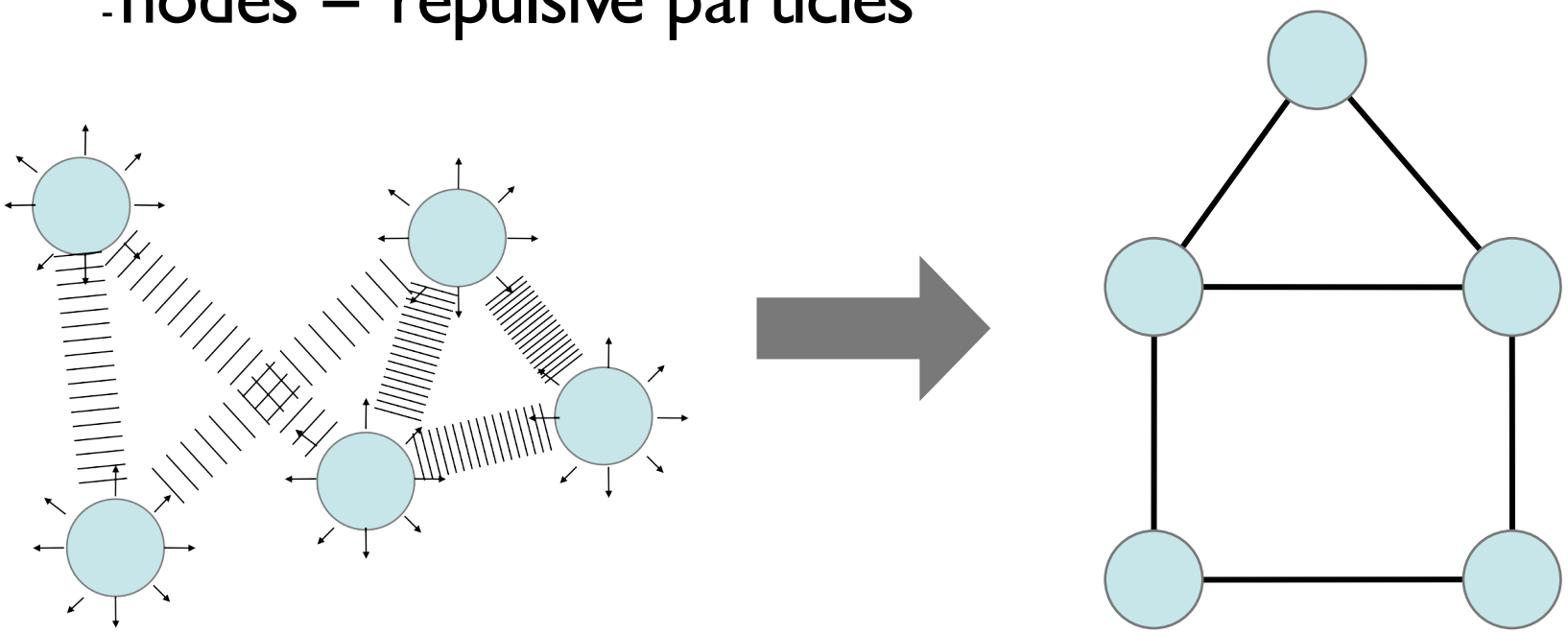
However, cycles in the graph may mislead.

Long edges can impede perception of proximity.

Force directed layouts

FORCE DIRECTED LAYOUT

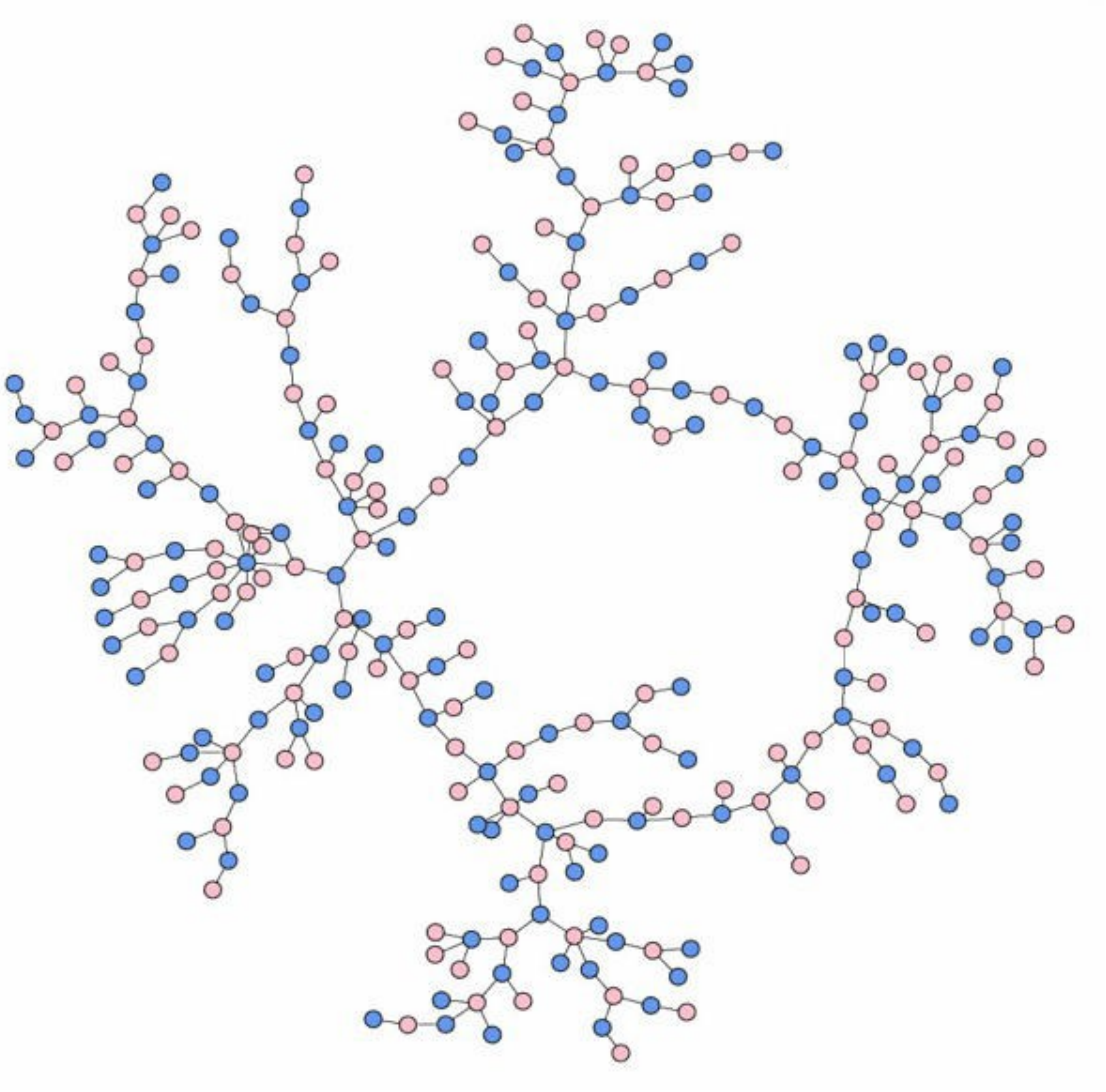
- **no intrinsic layering, now what?**
- **physics model**
  - edges = springs
  - nodes = repulsive particles



Force-directed Layouts

- We want edges to be neither too small or too large
  - Physical analogy: **Springs compress or expand to achieve ideal length**
- We don’t want vertices to bunch up together
  - Physical analogy: **Electric charges with the same sign don’t bunch up**

AESTHETIC RESULTS



FORCE MODEL

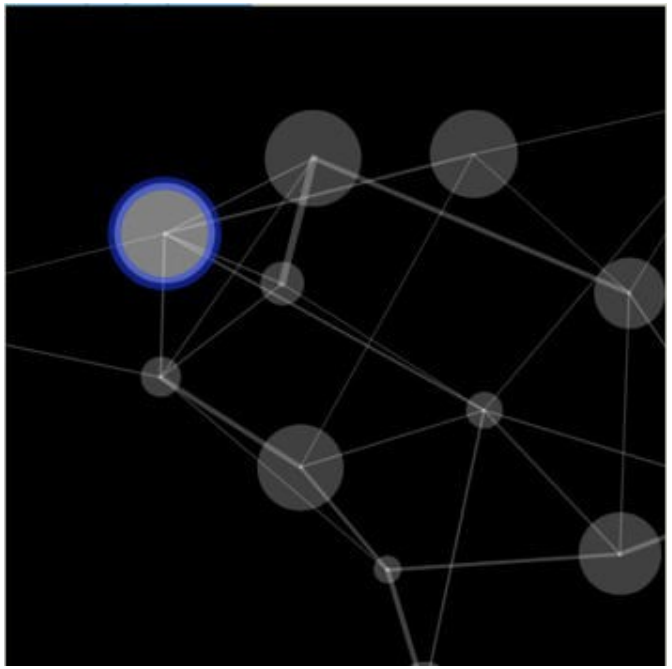
- **many variations, but usually physical analogy:**
  - **repulsion** : $f_R(d) = C_R * m_1 * m_2 / d^2$
    - *m_1, m_2 are node masses*
    - *d is distance between nodes*
  - **attraction** : $f_A(d) = C_A * (d - L)$
    - *L is the rest length of the spring*
    - *i.e. Hooke's Law*
- **total force on a node x with position x'**
  - $\sum_{\text{neighbors}(x)}$: $f_A(||x'-y'||) * (x'-y') + -f_R(||x'-y'||) * (x'-y')$

ALGORITHM

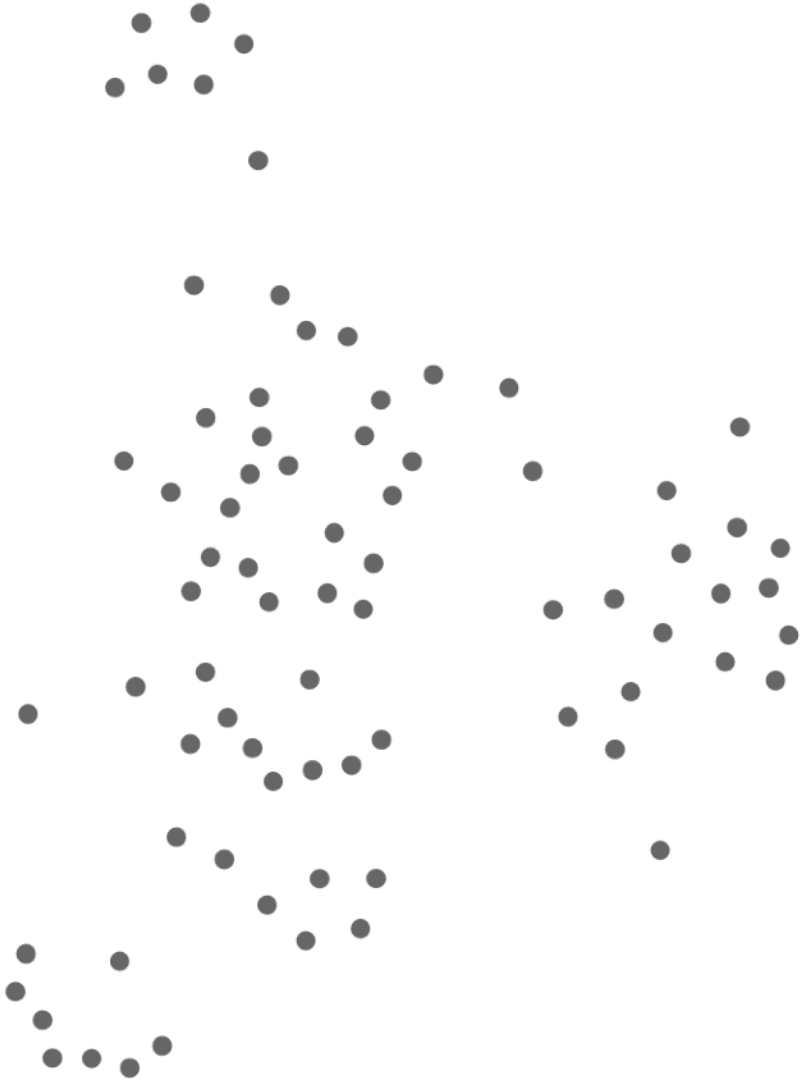
- **start from random layout**
- **(global) loop:**
  - for every node pair compute repulsive force
  - for every edge compute attractive force
  - accumulate forces per node
  - update each node position in direction of accumulated force
- **stop when layout is 'good enough'**

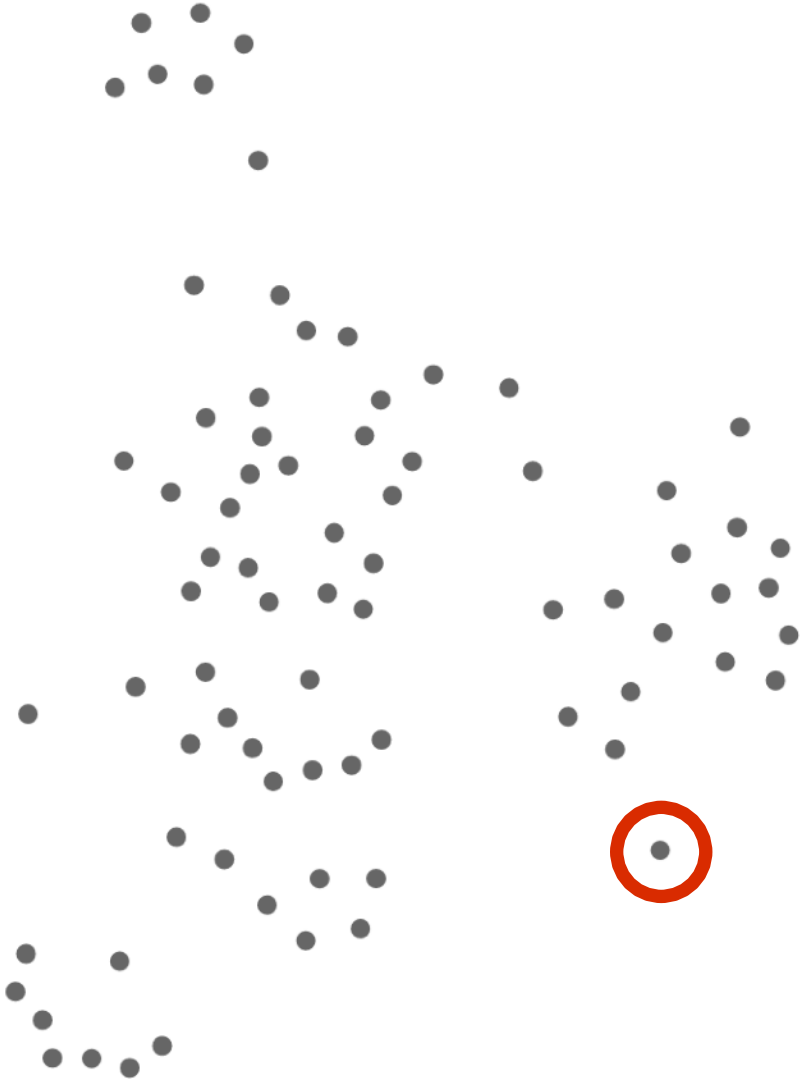
FORCE DIRECTED LAYOUTS

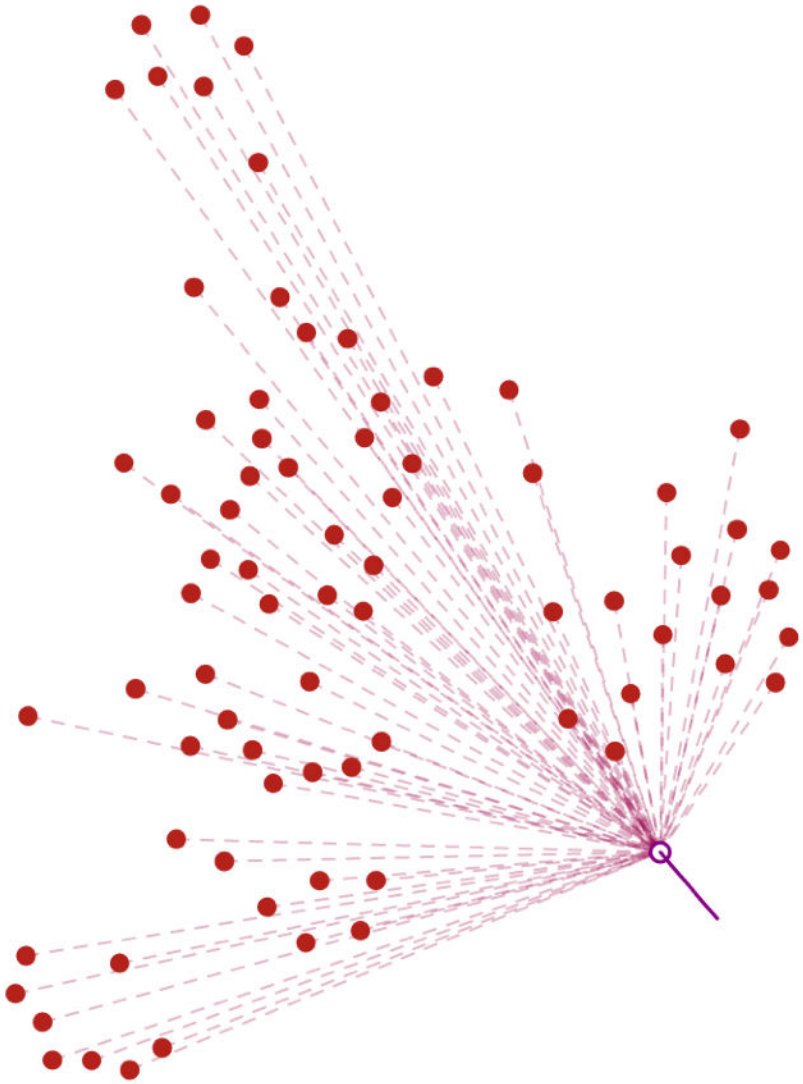
+ **very flexible, aesthetic layouts on many types of graphs**
+ **can add custom forces**
+ **relatively easy to implement**



- **repulsion loop is $O(n^2)$ per iteration**
  - can speed up to O(N log N) using quadtree or k-d tree
  - Implemented in d3
- **prone to local minima**
  - can use simulated annealing







Naive calculation of forces at a point uses sum of forces from all other n-1 points.

Force-Directed Layout in d3

Nodes = charged particles with air resistance

Edges = springs

$F = q_i * q_j / d_{ij}^2$

$F = -b * v_i$

$F = k * (L - d_{ij})$

At each timestep, calculate forces acting on nodes.
Integrate for updated velocities and positions.

D3's force layout uses **velocity Verlet** integration.
Assume uniform mass ***m*** and timestep ***Δt***:

$F = ma \rightarrow F = a \rightarrow F = \Delta v / \Delta t \rightarrow F = \Delta v$

Forces simplify to velocity offsets!

N-Body Force

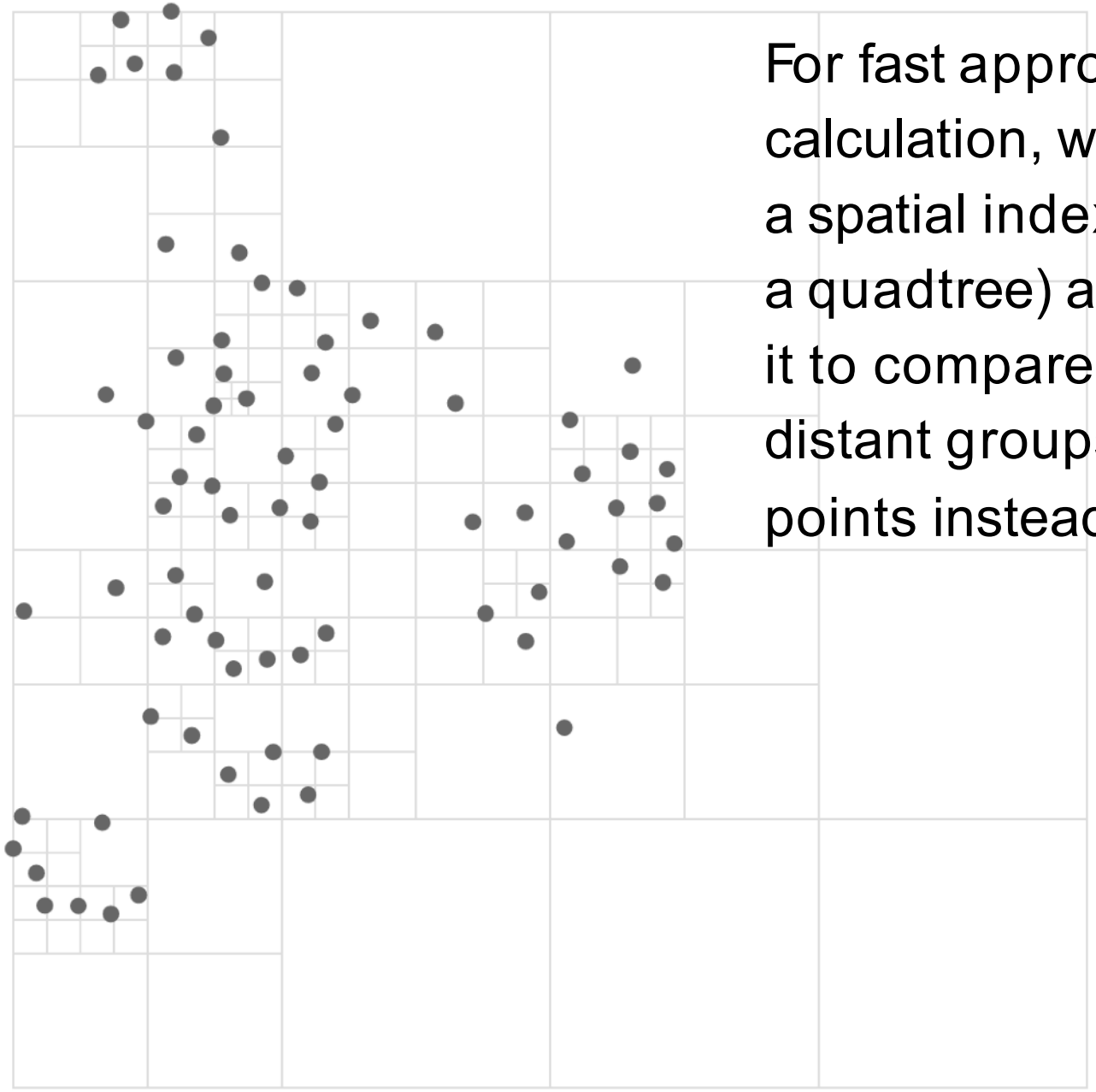
Naïve calculation of repulsive force doesn't scale!

Comparing all pairs of nodes is $O(V^2)$

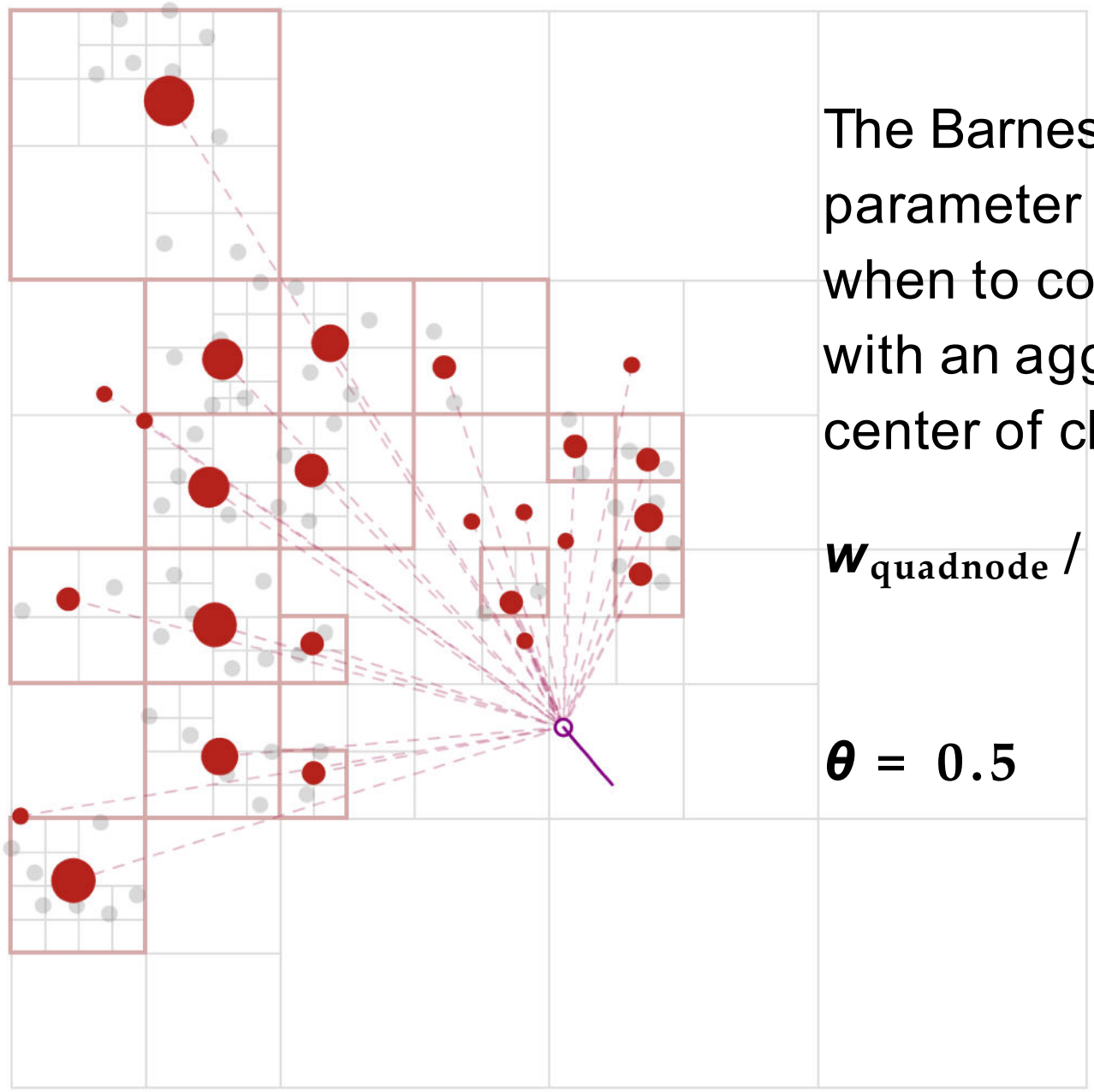
We can approximate force calculations using a spatial index (e.g., quadtree) to achieve O(V log V)

One such approach is the **Barnes-Hut algorithm**, originally created for astronomical simulations.

The key idea is to approximate forces from distant nodes by comparing to aggregate centers of charge rather than individual nodes.



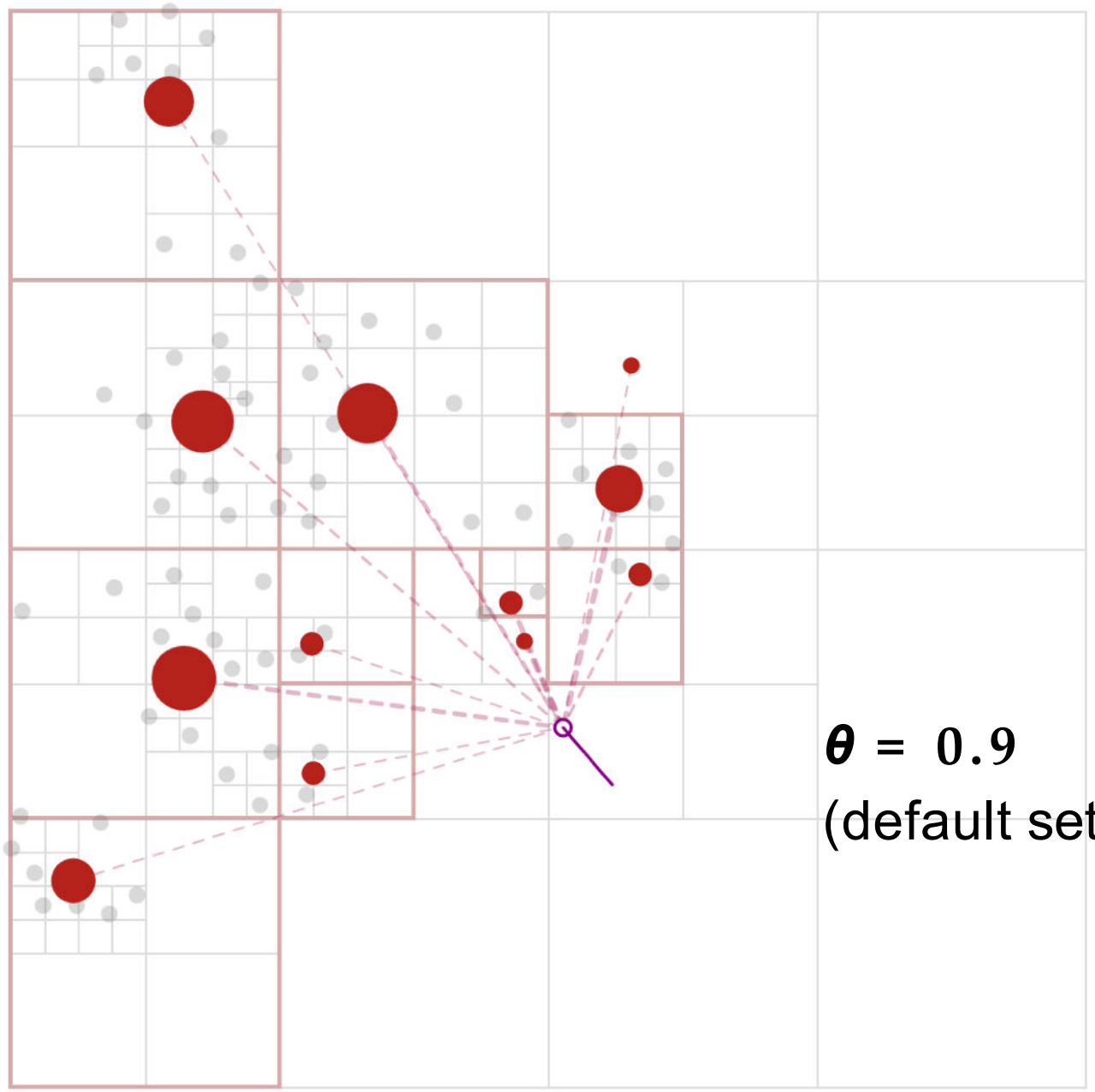
For fast approximate calculation, we build a spatial index (here, a quadtree) and use it to compare with distant groups of points instead.

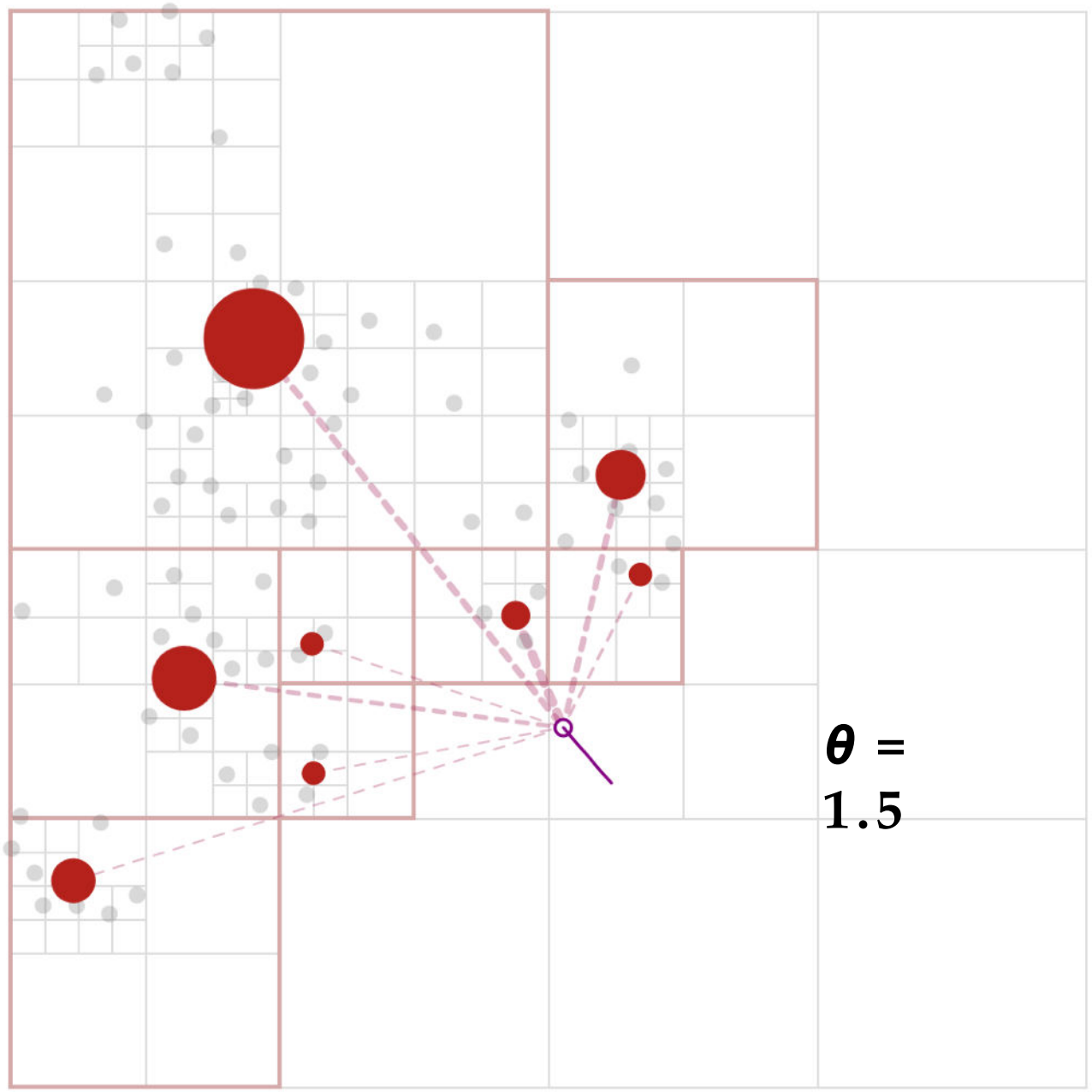


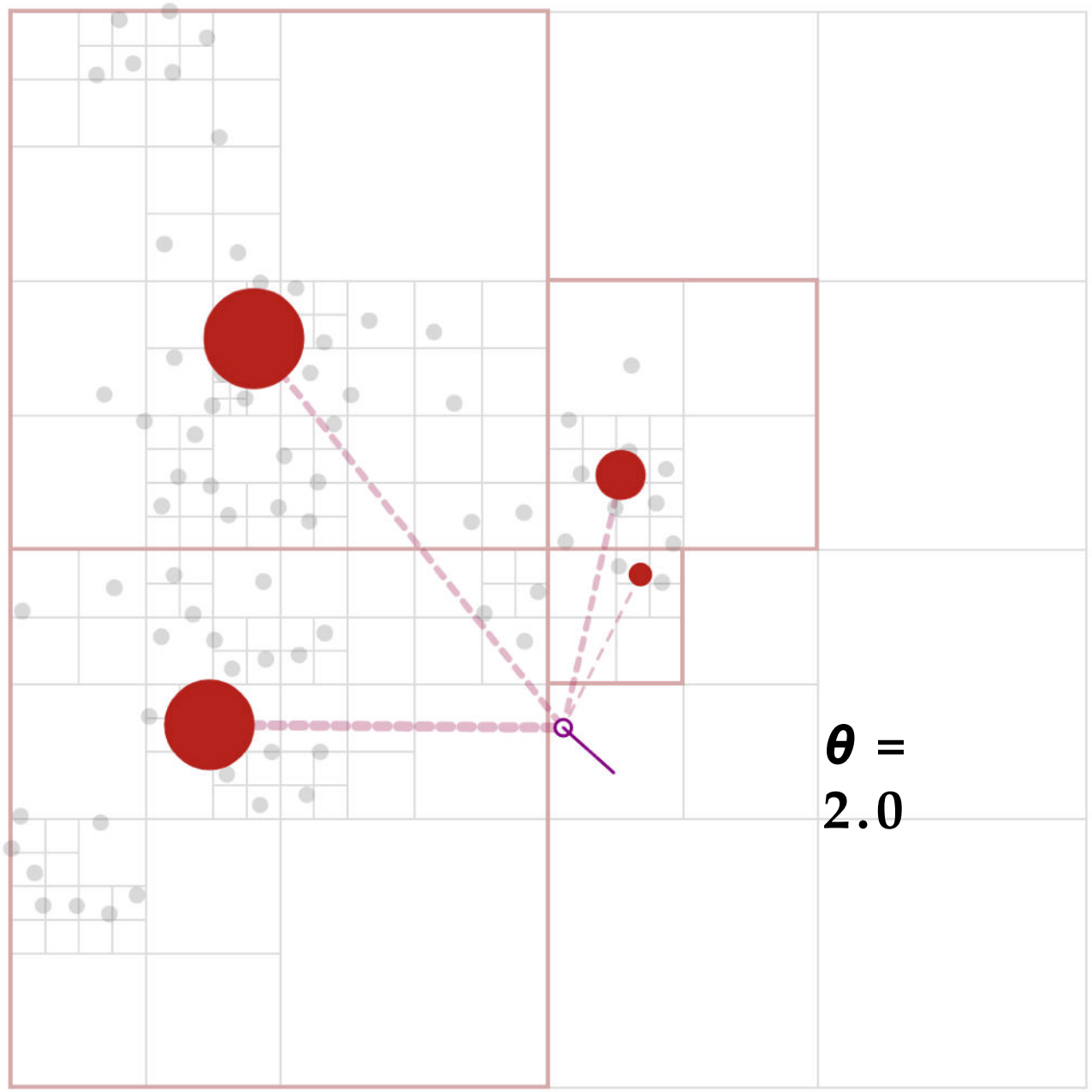
The Barnes-Hut θ parameter controls when to compare with an aggregate center of charge.

$$w_{\text{quadnode}} / d_{ij} < \theta \ ?$$

$$\theta = 0.5$$


$\theta = 0.9$
(default setting)


θ =
1.5


θ =
2.0

How forces work in d3

And how we can customize Force Directed Layouts

http://mbostock.github.io/d3/talk/20110921/

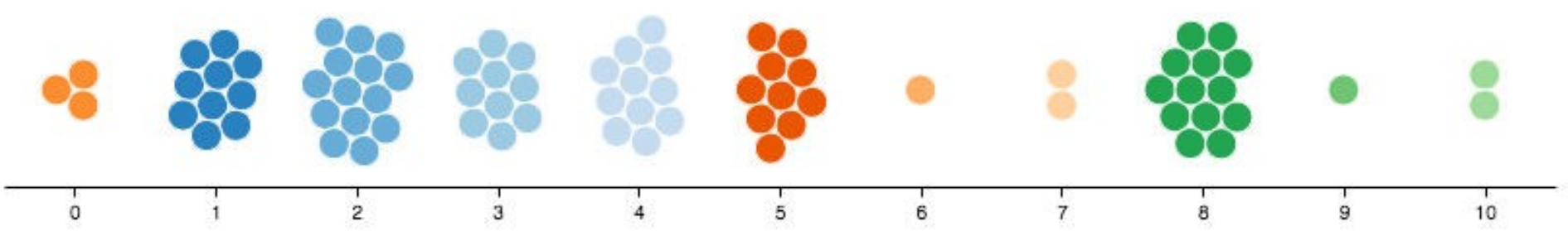
Customized Force Layouts

Different forces can be composed to create an expressive space of custom layouts.

A **beeswarm plot** can be made by combining:

Attractive **X** and **Y** forces to draw nodes of a certain category to a desired point

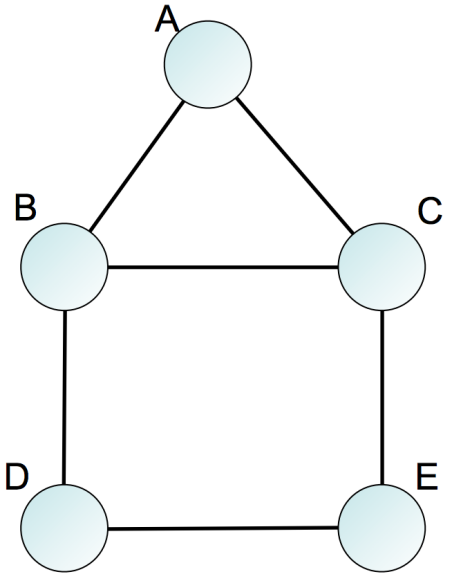
Collide force to detect collision & remove overlap



Matrix Diagrams

Adjacency Matrix

Alternate to node-link diagram: adjacency matrix

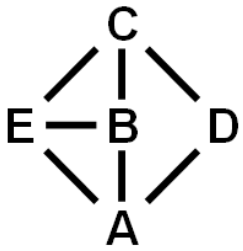


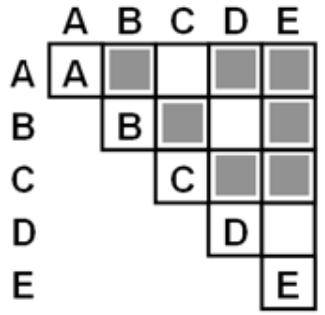
	A	B	C	D	E
A		■	■		
B	■		■	■	
C	■	■			■
D		■			■
E			■	■	

Slide by Frank van Ham

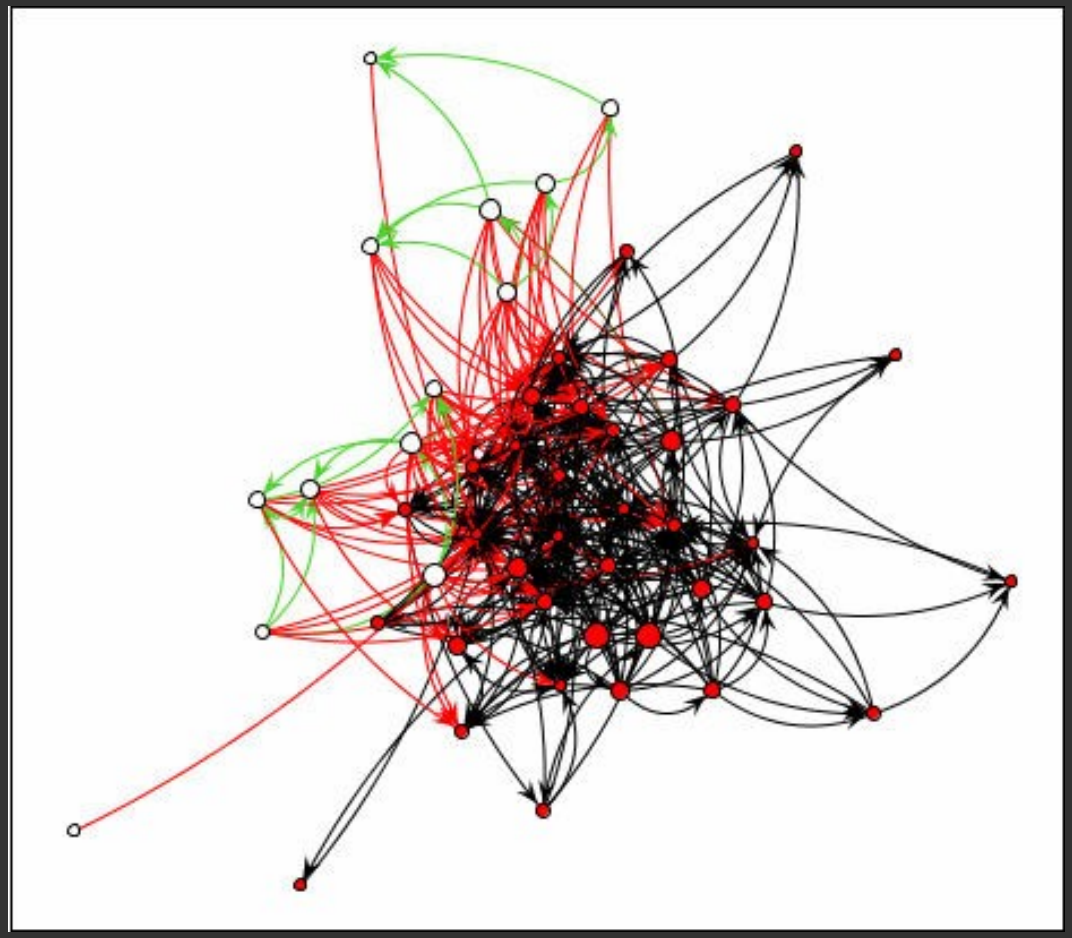
Adjacency Matrix

- Change network to tabular data and use a matrix representation
- Derived data: nodes are keys, edges are boolean values
- Task: lookup connections, find well-connected clusters
- Scalability: millions of edges
- Can encode edge weight, too

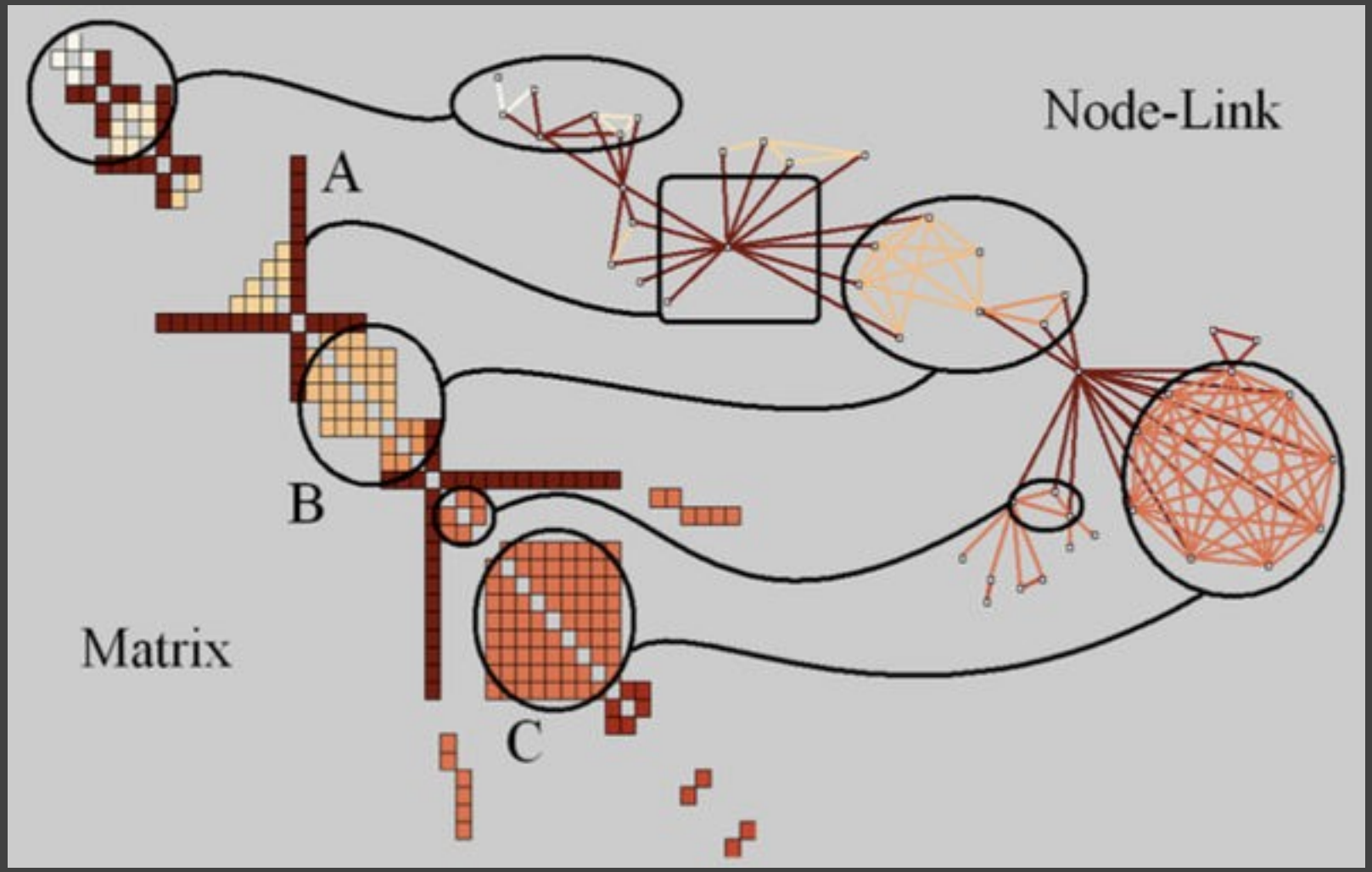




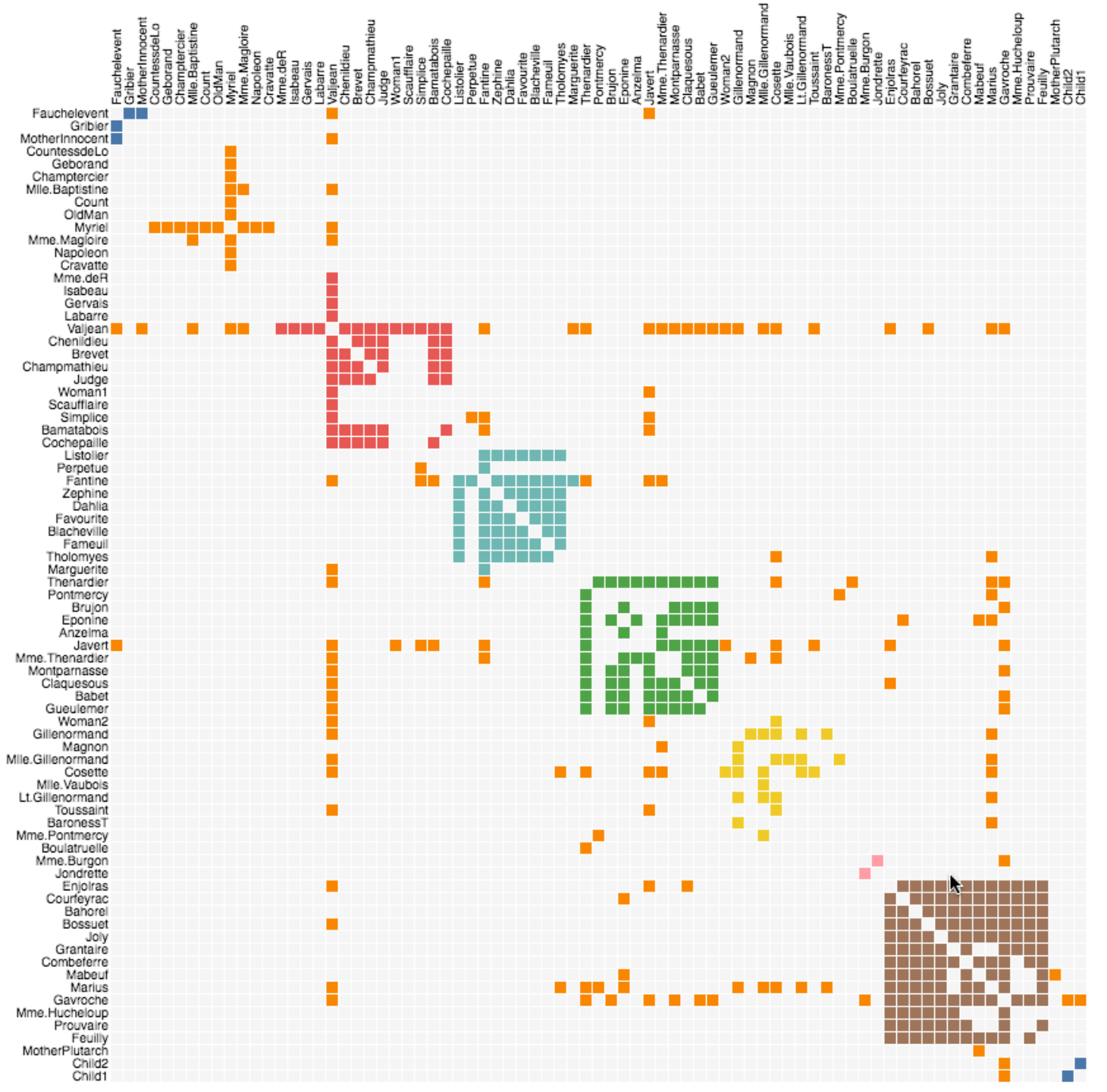
Limitations of Node-Link Layouts



Edge-crossings and occlusion! Poor scalability….



Adjacency Matrices


Fauchelevent
Gribier
MotherInnocent
CountessdeLo
Geborand
Champtercier
Mlle.Baptistine
Count
OldMan
Myriel
Mme.Magloire
Napoleon
Cravatte
Mme.deR
Isabeau
Gervais
Labarre
Valjean
Chenildieu
Brevet
Champmathieu
Judge
Woman1
Scaufflaire
Simplice
Bamatabois
Cochepaille
Listolier
Perpetue
Fantine
Zephine
Dahlia
Favourite
Blacheville
Fameuil
Tholomyes
Marguerite
Thenardier
Pontmercy
Brujon
Eponine
Anzelma
Javert
Mme.Thenardier
Montparnasse
Claquesous
Babet
Gueulemer
Woman2
Gillenormand
Magnon
Mlle.Gillenormand
Cosette
Mlle.Vaubois
Lt.Gillenormand
Toussaint
BaronessT
Mme.Pontmercy
Boulatruelle
Mme.Burgon
Jondrette
Enjolras
Courfeyrac
Bahorel
Bossuet
Joly
Grantaire
Combeferre
Mabeuf
Marius
Gavroche
Mme.Hucheloup
Prouvaire
Feuilly
MotherPlutarch
Child2
Child1

Graph Viewer

Graph Viewer

Roll-up by:

All

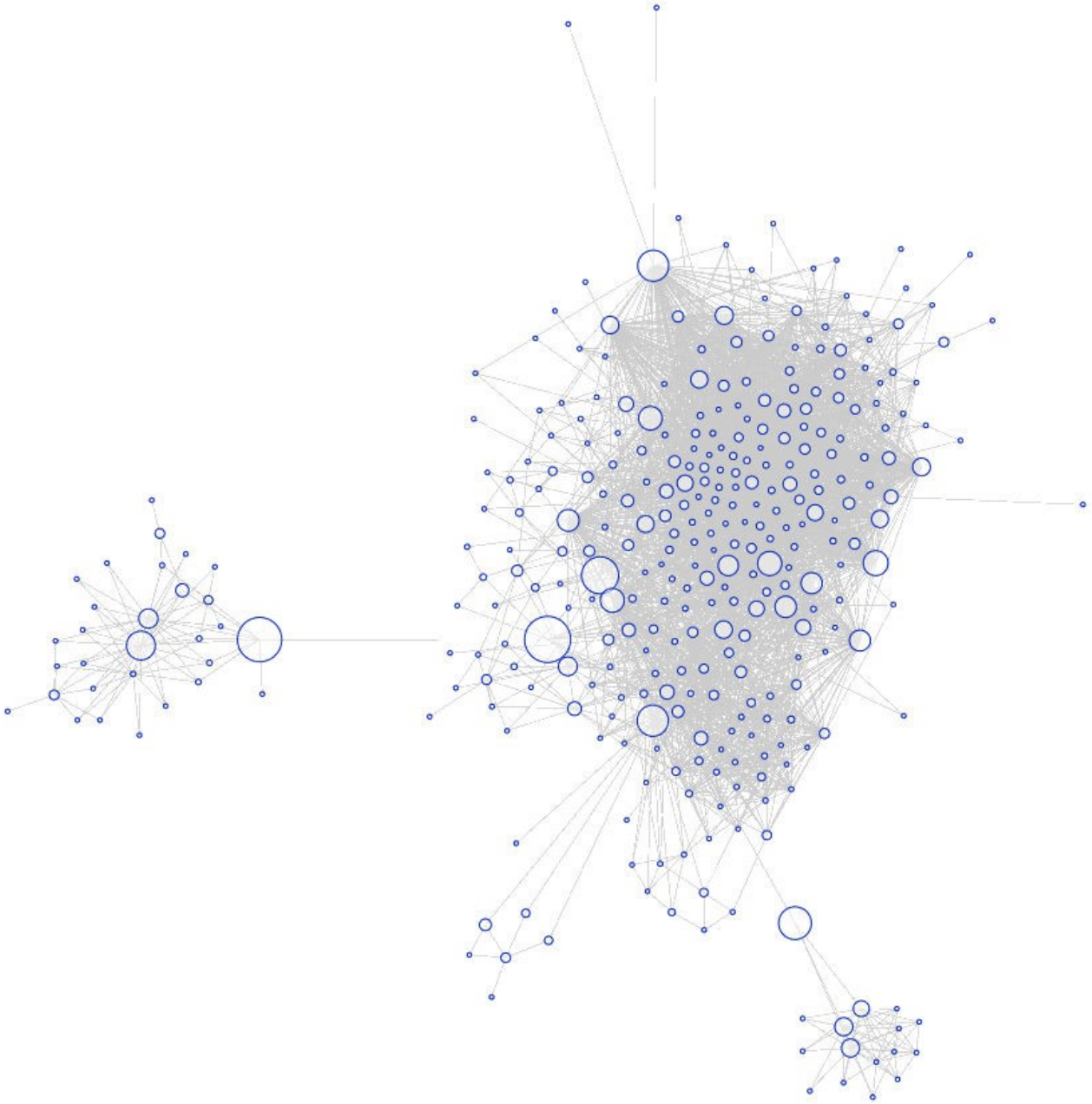
Visualization:

Node-Link

Sort by:

None

Edge centrality filters:



Images

Animate

Graph Viewer

Graph Viewer

Roll-up by:

All

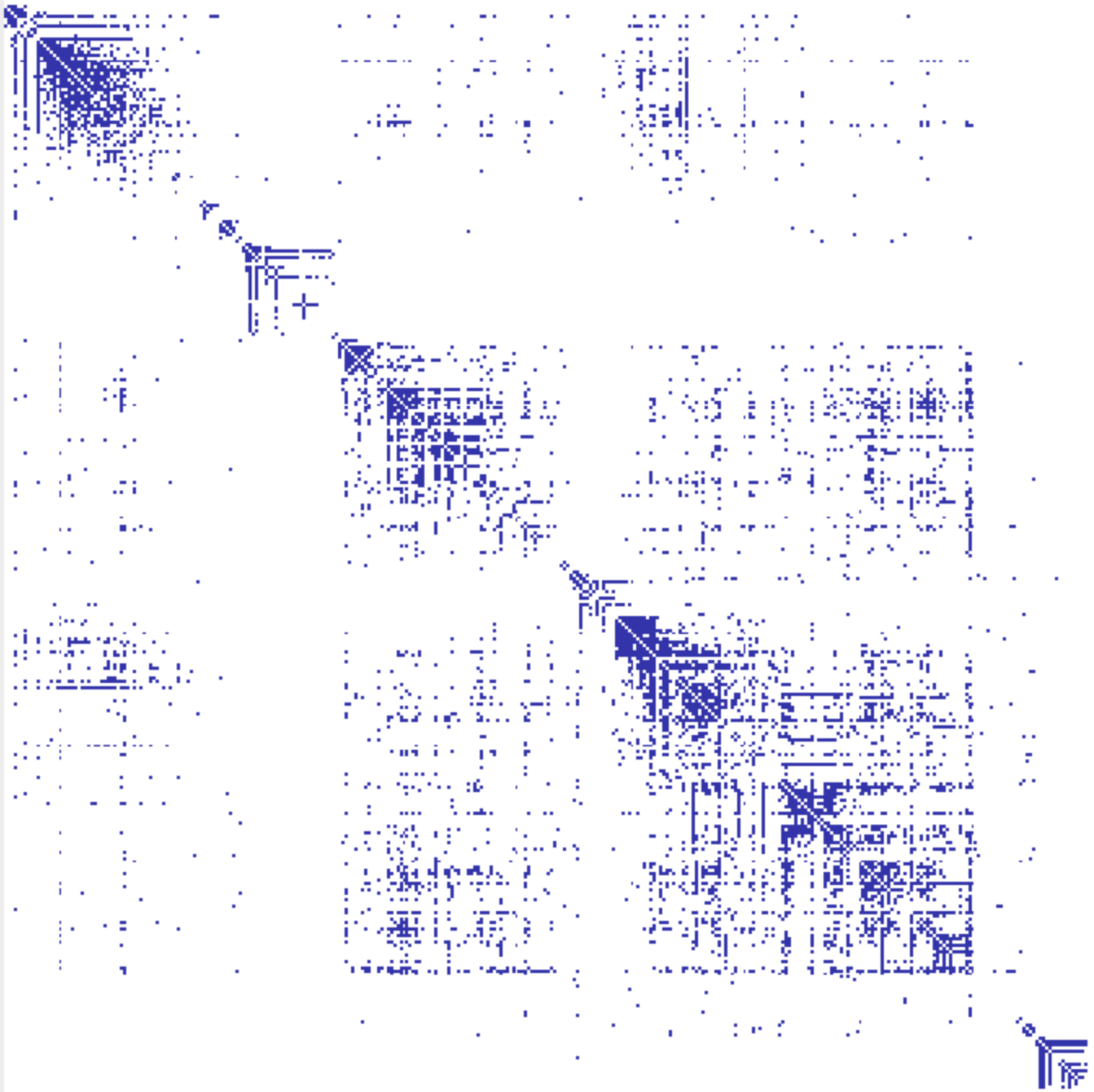
Visualization:

Matrix

Sort by:

Linkage

Edge centrality filters:



Graph Viewer

Graph Viewer

Roll-up by:

All

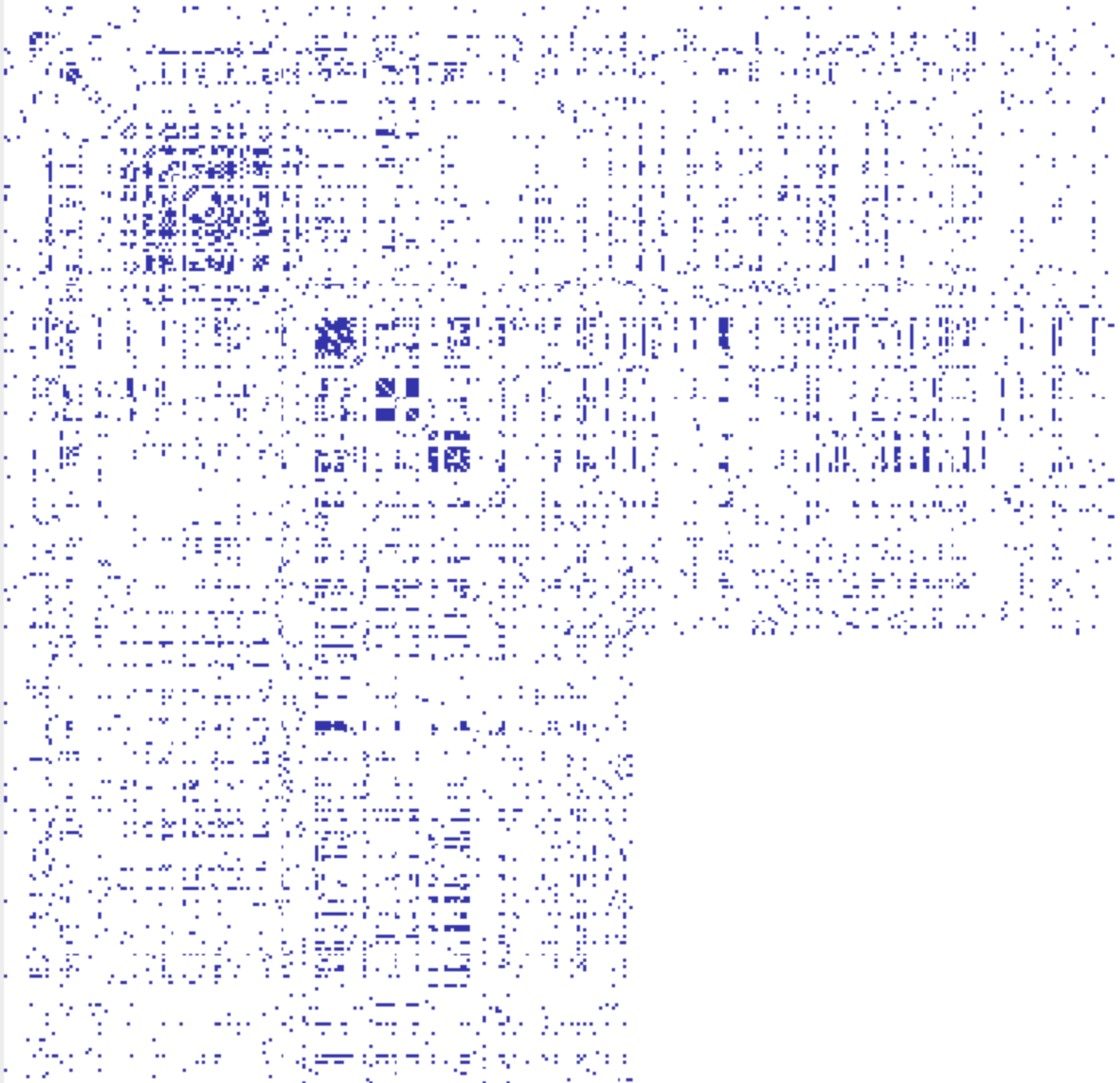
Visualization:

Matrix

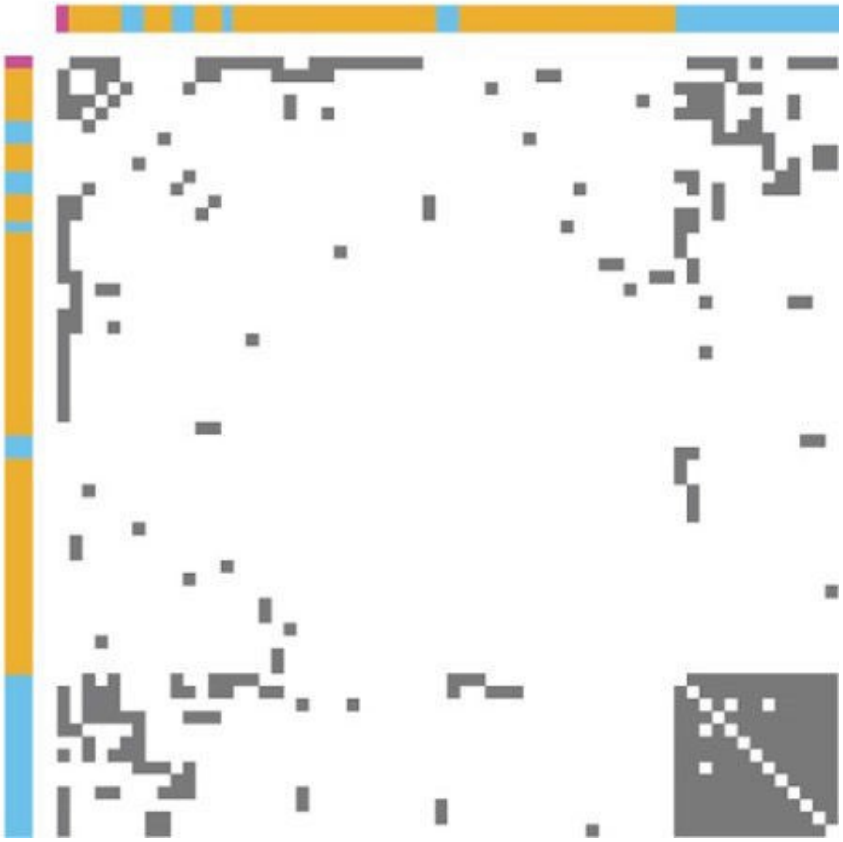
Sort by:

None

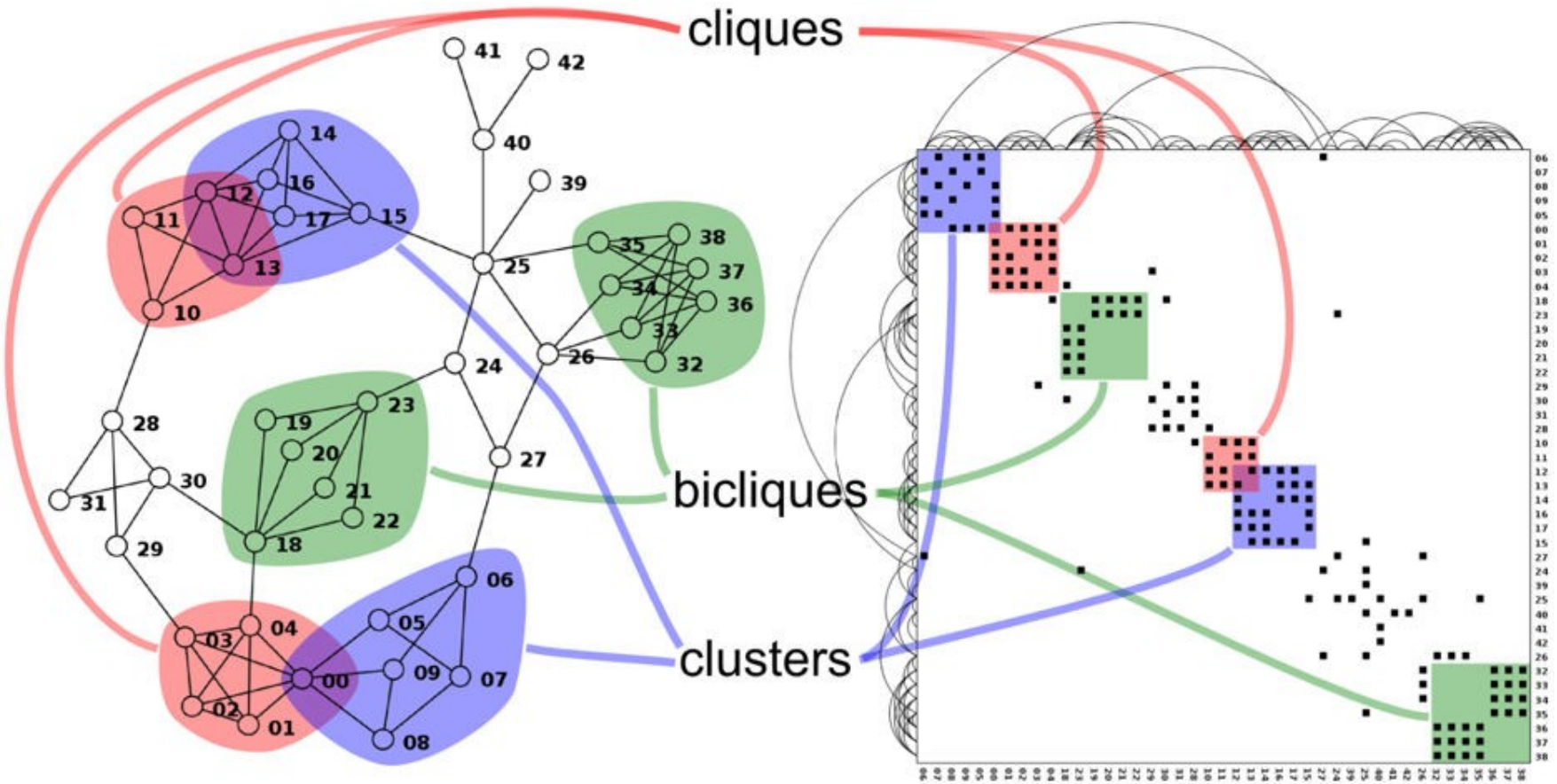
Edge centrality filters:



Cliques in Adjacency Matrices



Node link and Adjacency Matrix (can co-exist together)



[McGuffin]

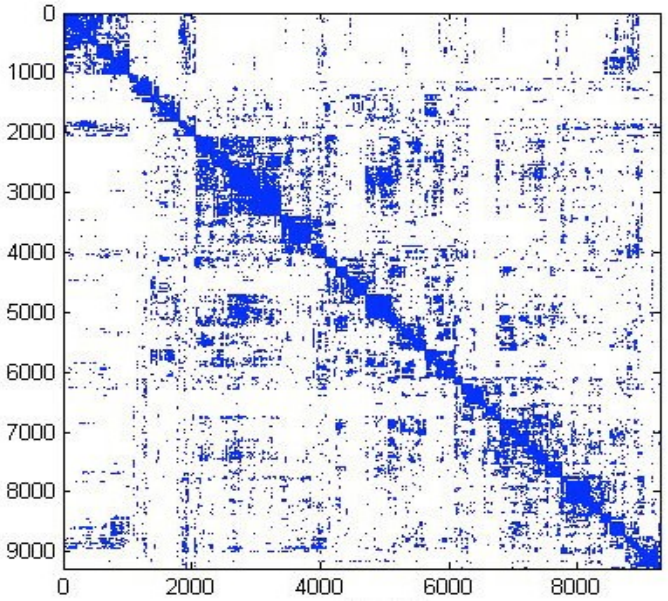
Adjacency Matrix

Pros:

- great for dense graphs
- visually scalable
- can spot clusters

Cons:

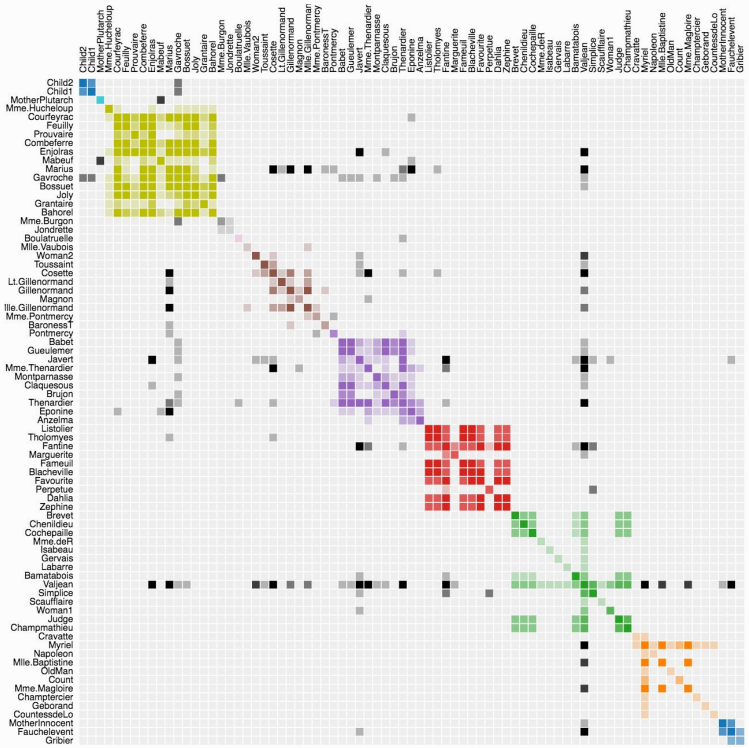
- row order affects what you can see
- abstract visualization
- hard to follow (multilink) paths



Node-Link or Adjacency Matrix?

- Empirical study: For most tasks, node-link is better for small graphs and adjacency better for large graphs
- Immediate connectivity or neighbors are ok, estimating size (nodes & edges also ok)
- People tend to be more familiar with node-link diagrams

Les Misérables Co-occurrence



Source: The Stanford GraphBase.

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

Attribute-Driven Layout

Attribute-Driven Layout

Large node-link diagrams **get messy**!
Is there additional structure we can exploit?

Idea: Use **data attributes** to perform layout
For example, scatter plot based on node values

Attributes may be associated with nodes or edges
or may be statistical properties of the graph.

Use dynamic queries / brushing to explore…

Attribute-Driven Layout

The “Skitter” Layout

Internet Connectivity

Radial Scatterplot

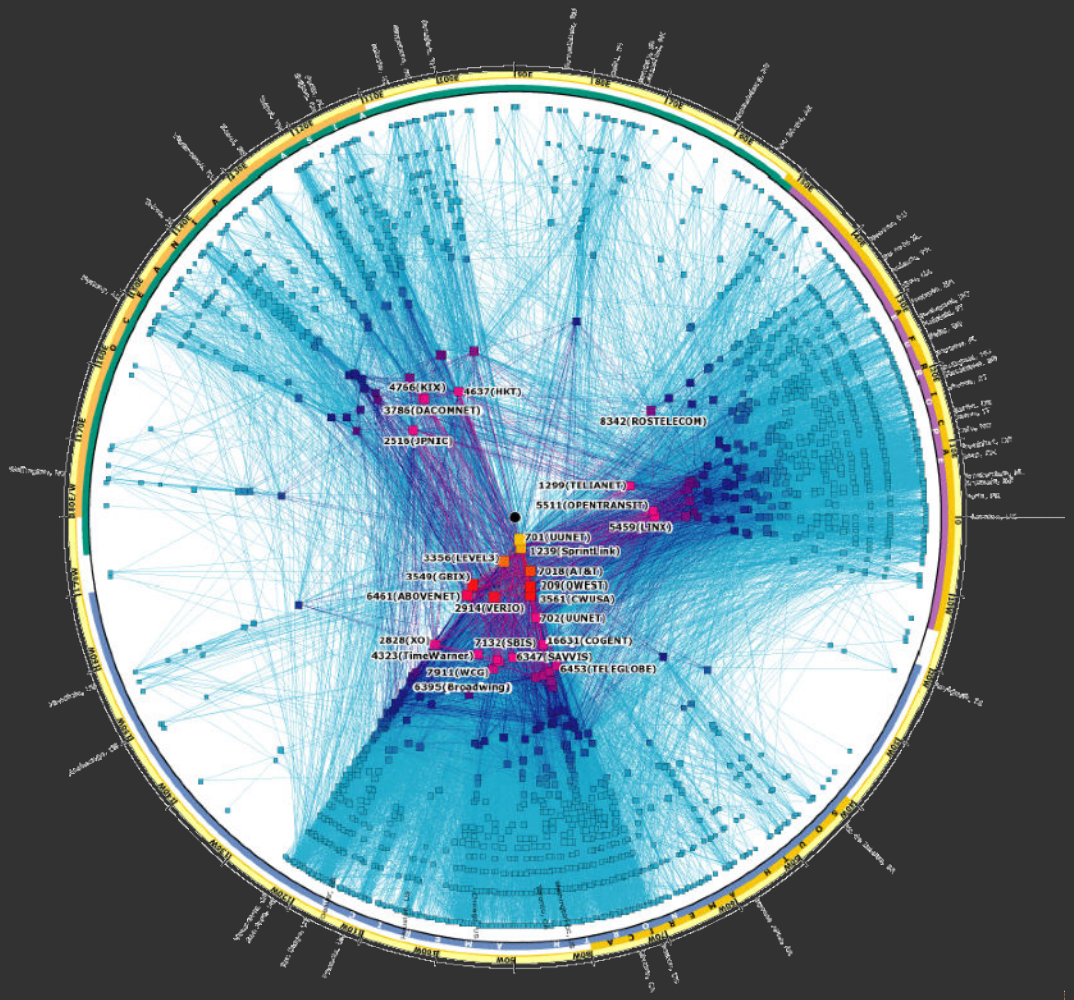
Angle = Longitude

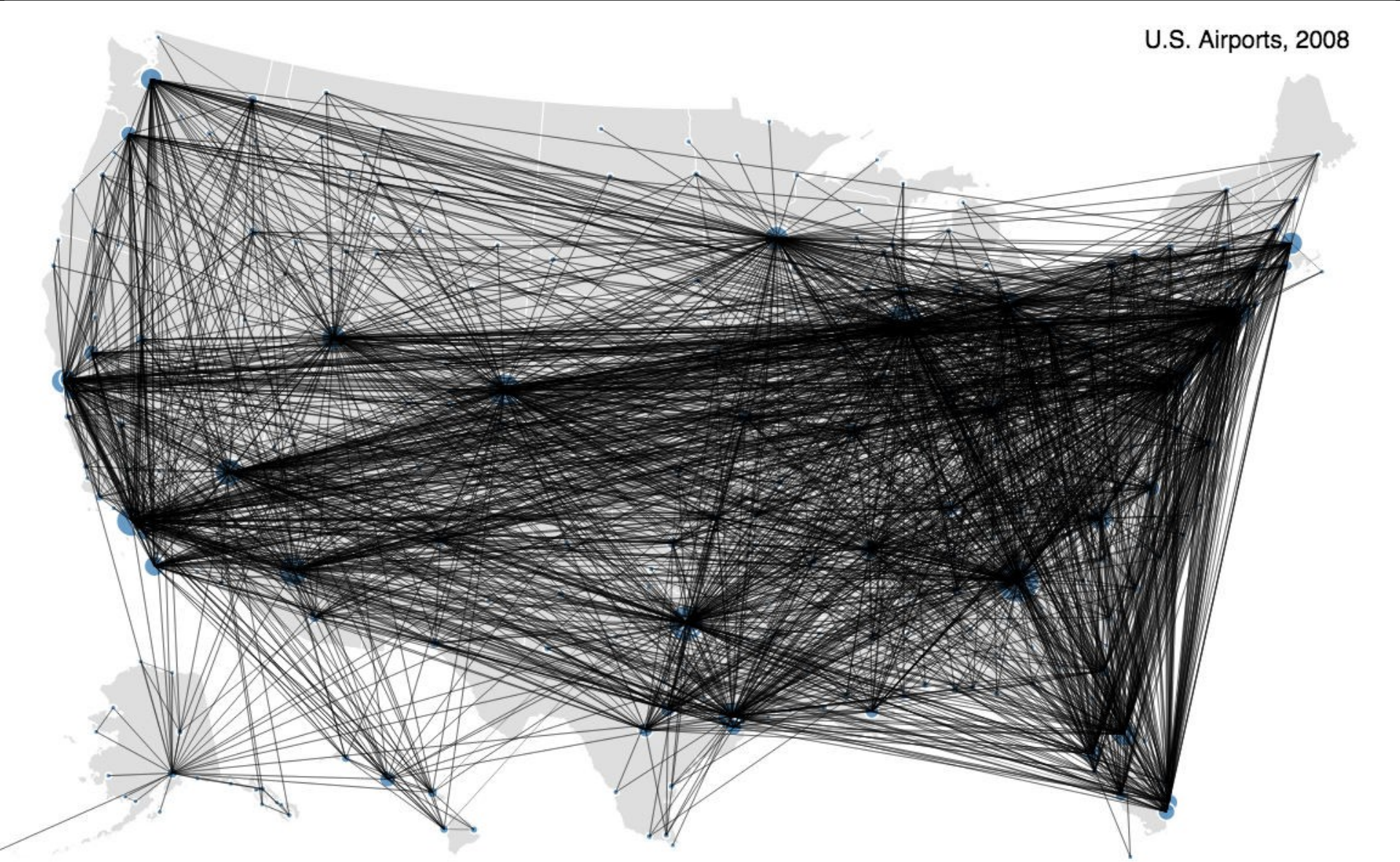
Geography

Radius = Degree

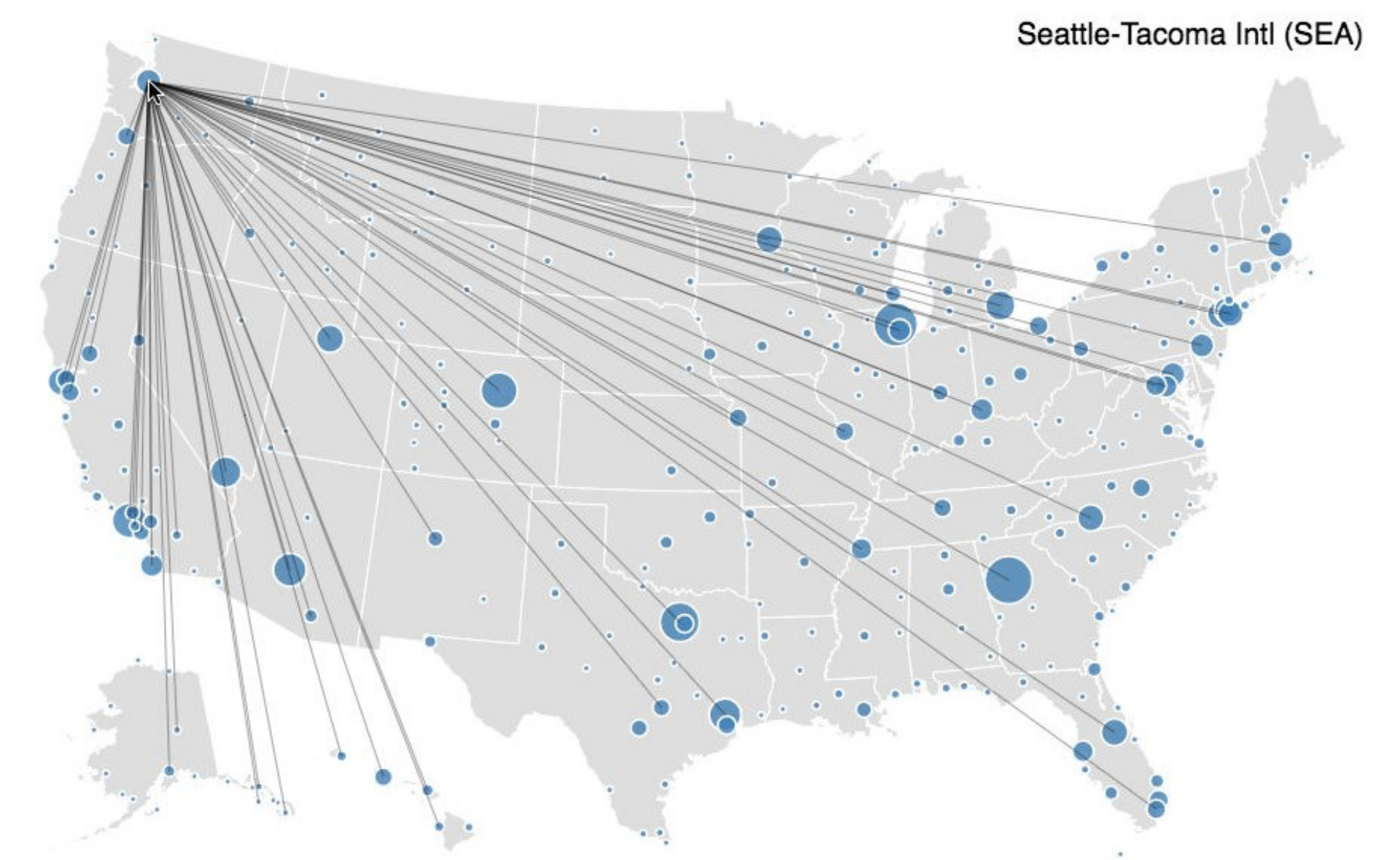
# of connections

(a statistic of the nodes)



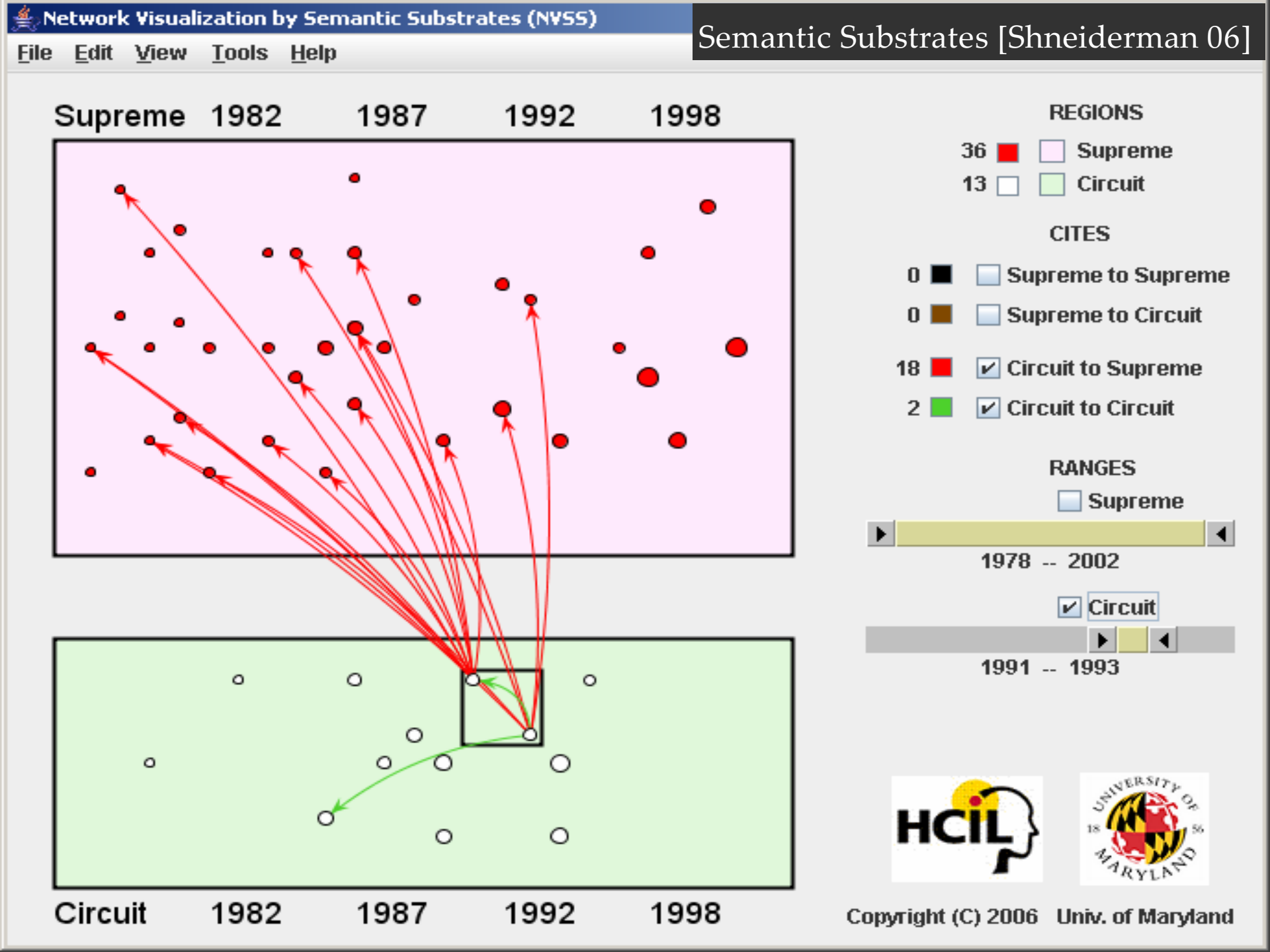


Drawing all edges is not particularly useful here…

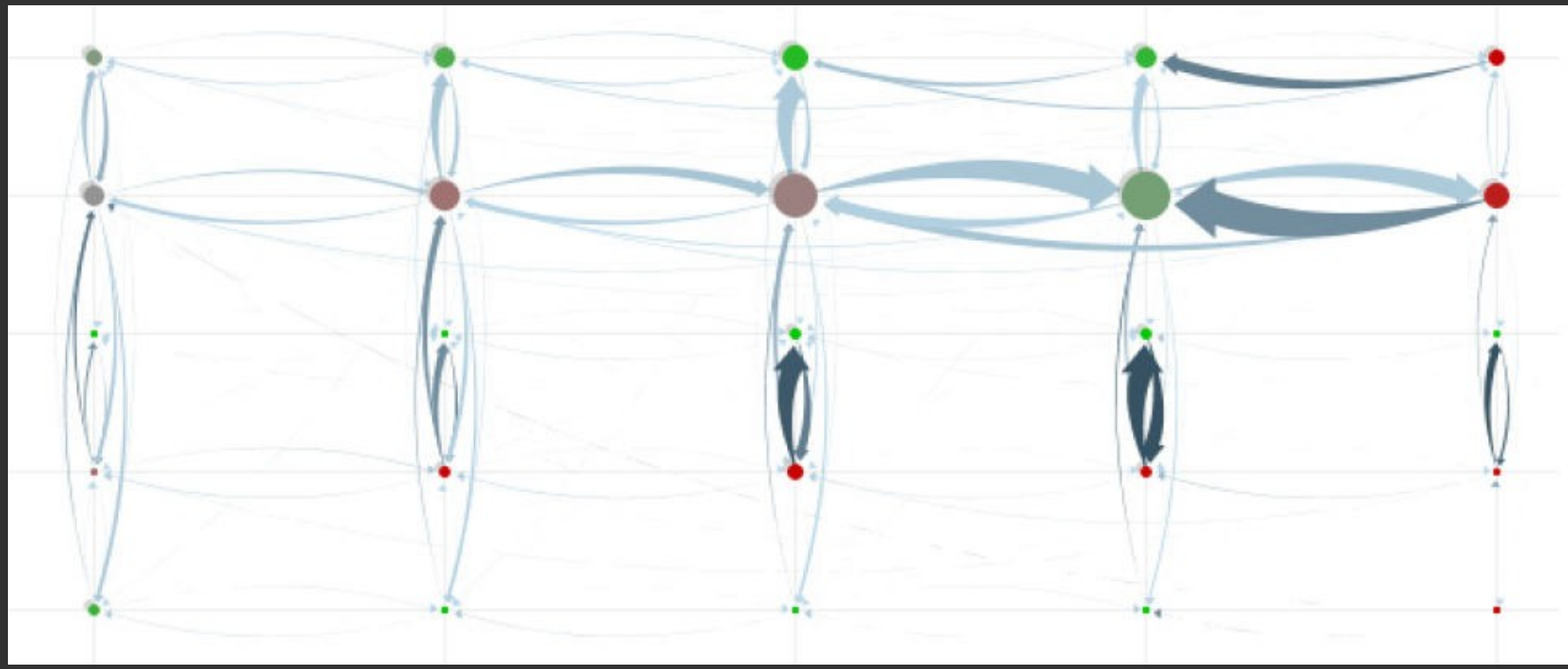


Node layout determined by geographic location.
Adjacent edges shown on node selection.

Semantic Substrates [Shneiderman 06]

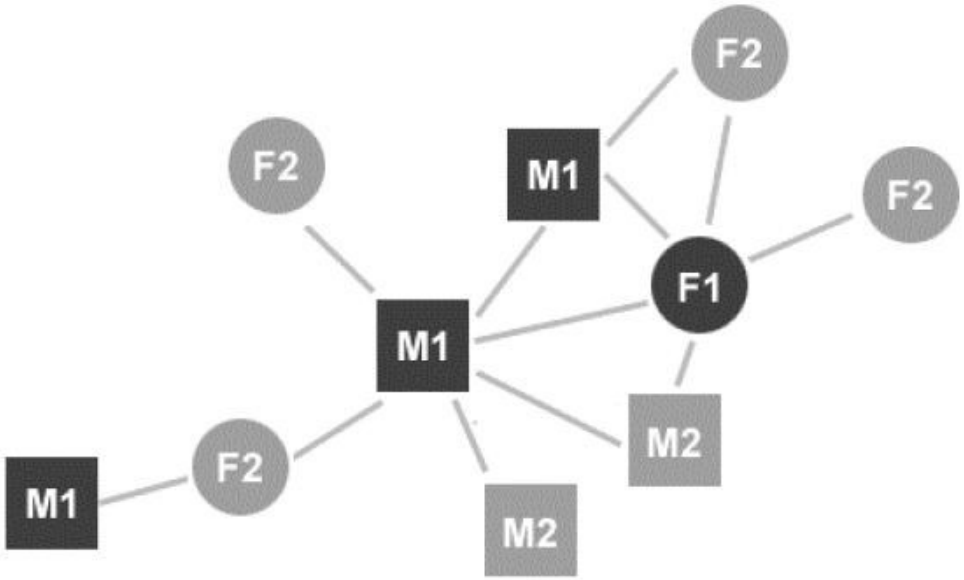


PivotGraph [Wattenberg ’06]

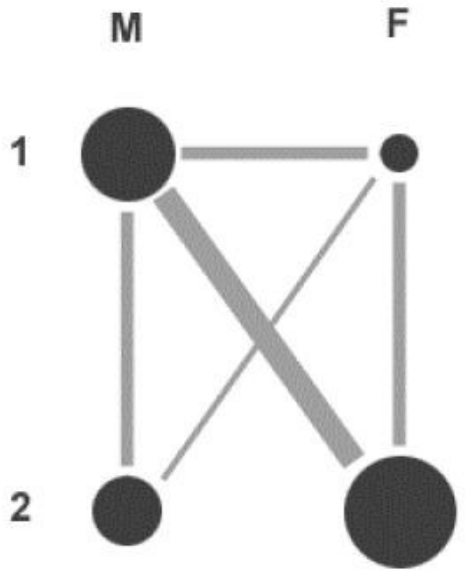


Layout aggregate graphs using node attributes.
Analogous to pivot tables and trellis display.

PivotGraph



Node and Link Diagram



PivotGraph Roll-up

File View Scale

X-Axis:

Gender

Y-Axis:

Location

Flip X/Y Clear

People

25
13
5
3

In vs. Out Degree

10.0
0.0
-10.0

Relationships

50
25
10
5

Rel. / Person

10.000
5.000
2.000
1.000

Select:

Gender

(All Values)

Legacy

(All Values)

Department

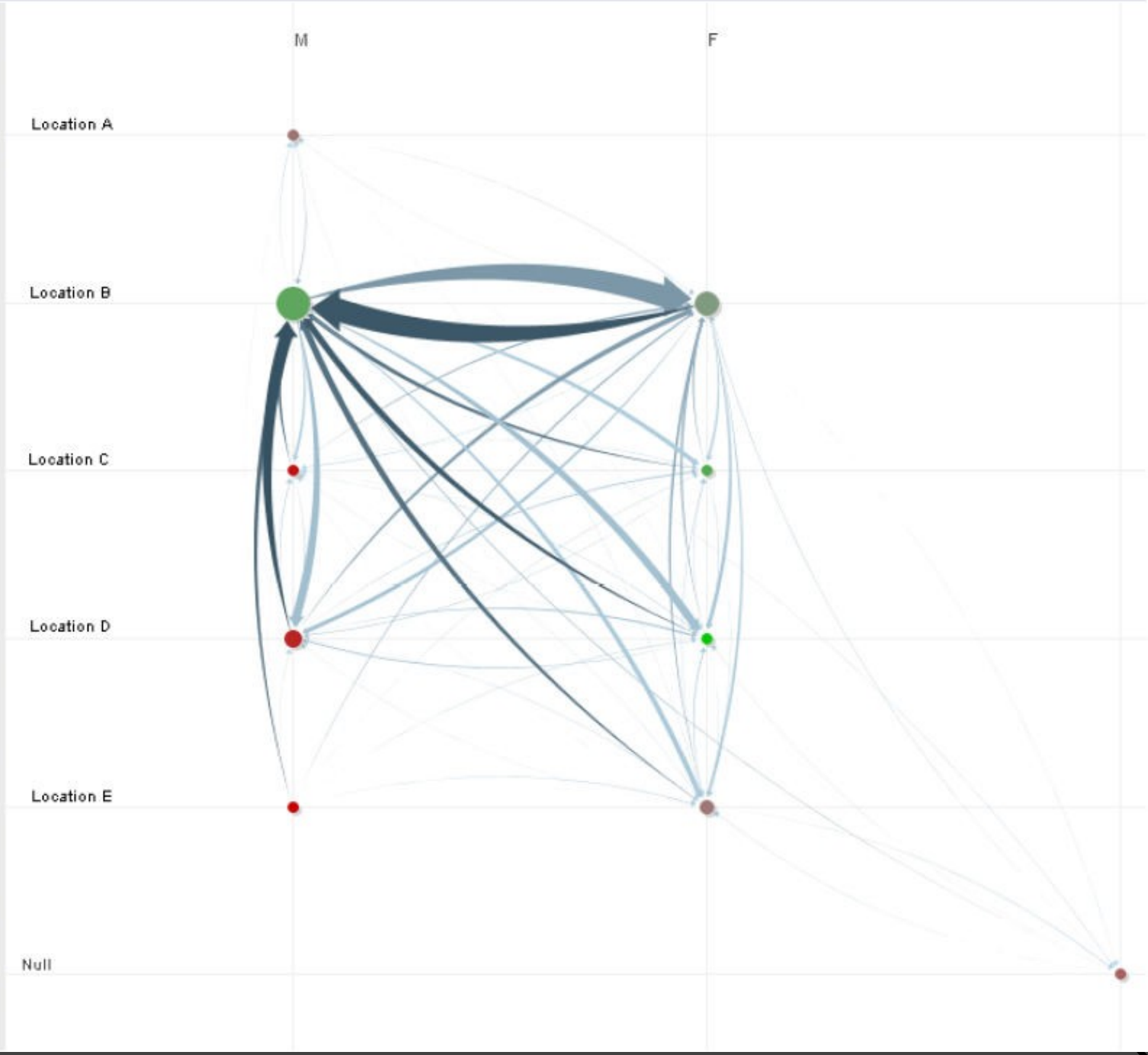
(All Values)

Level

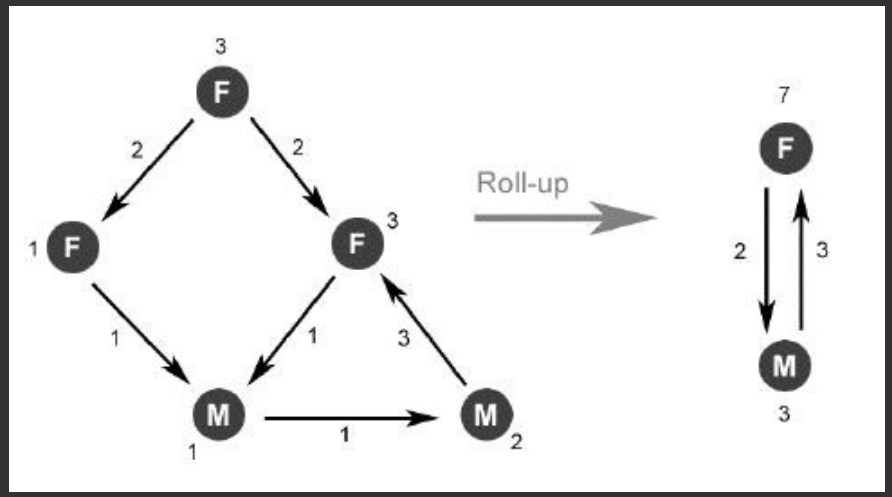
(All Values)

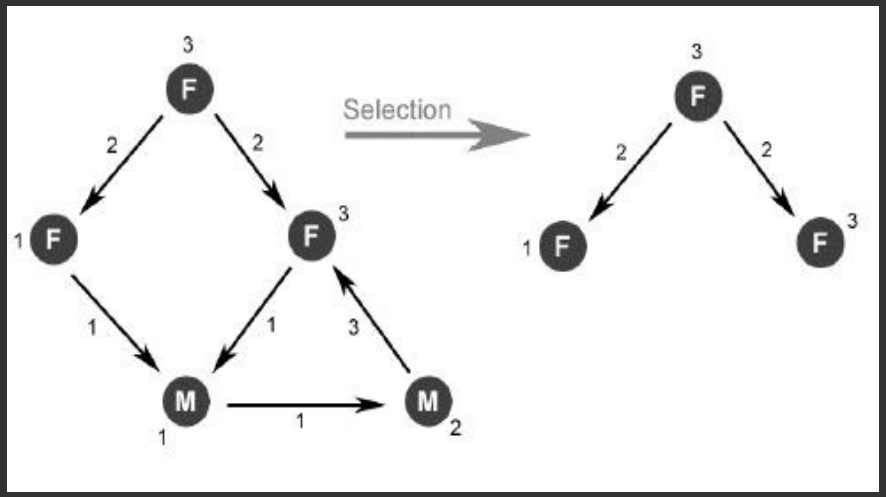
Location

(All Values)



Operators



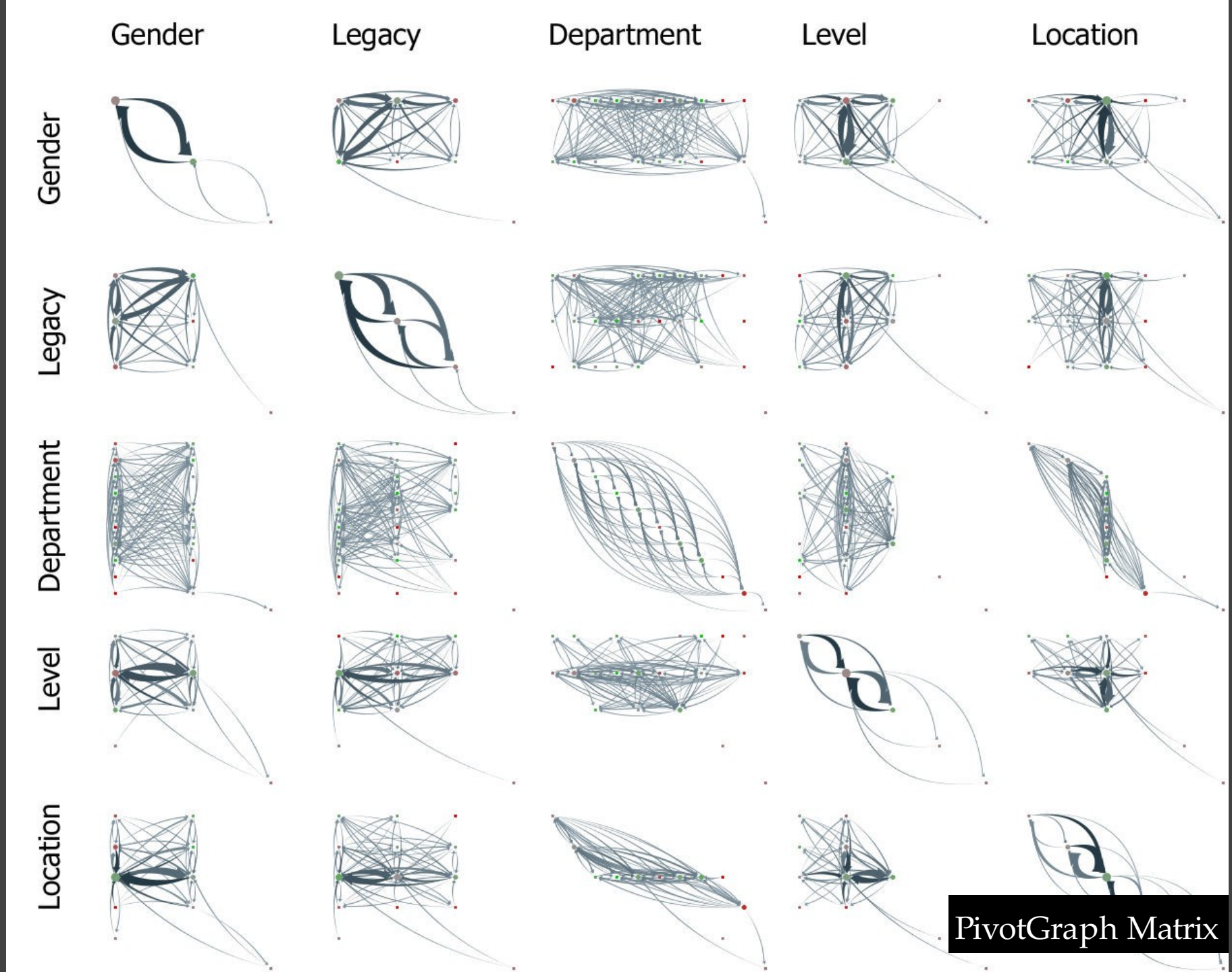


Roll-Up

Aggregate items with matching data values

Selection

Filter on data values


Gender
Legacy
Department
Level
Location
Gender
Legacy
Department
Level
Location
PivotGraph Matrix

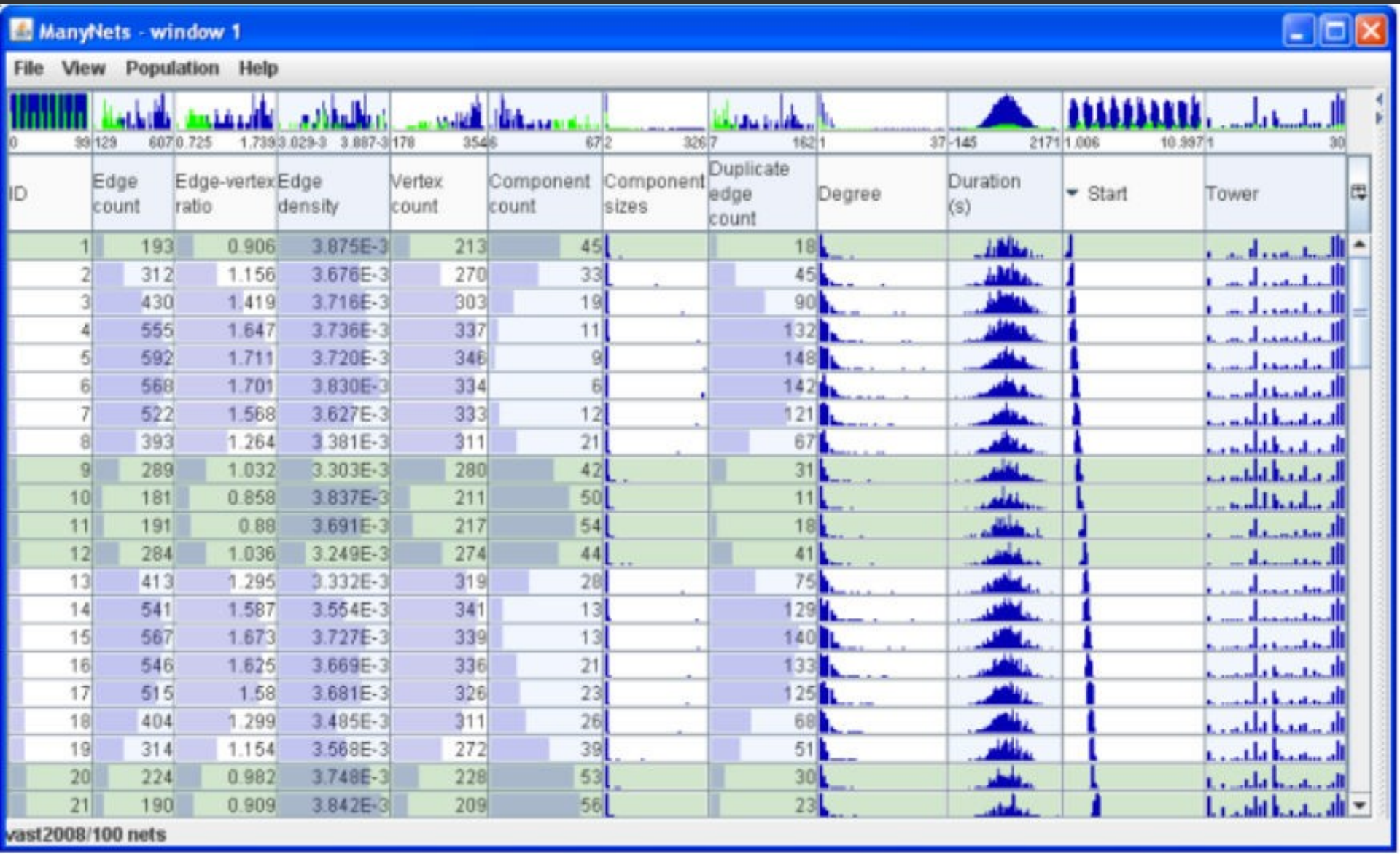
Limitations of PivotGraph

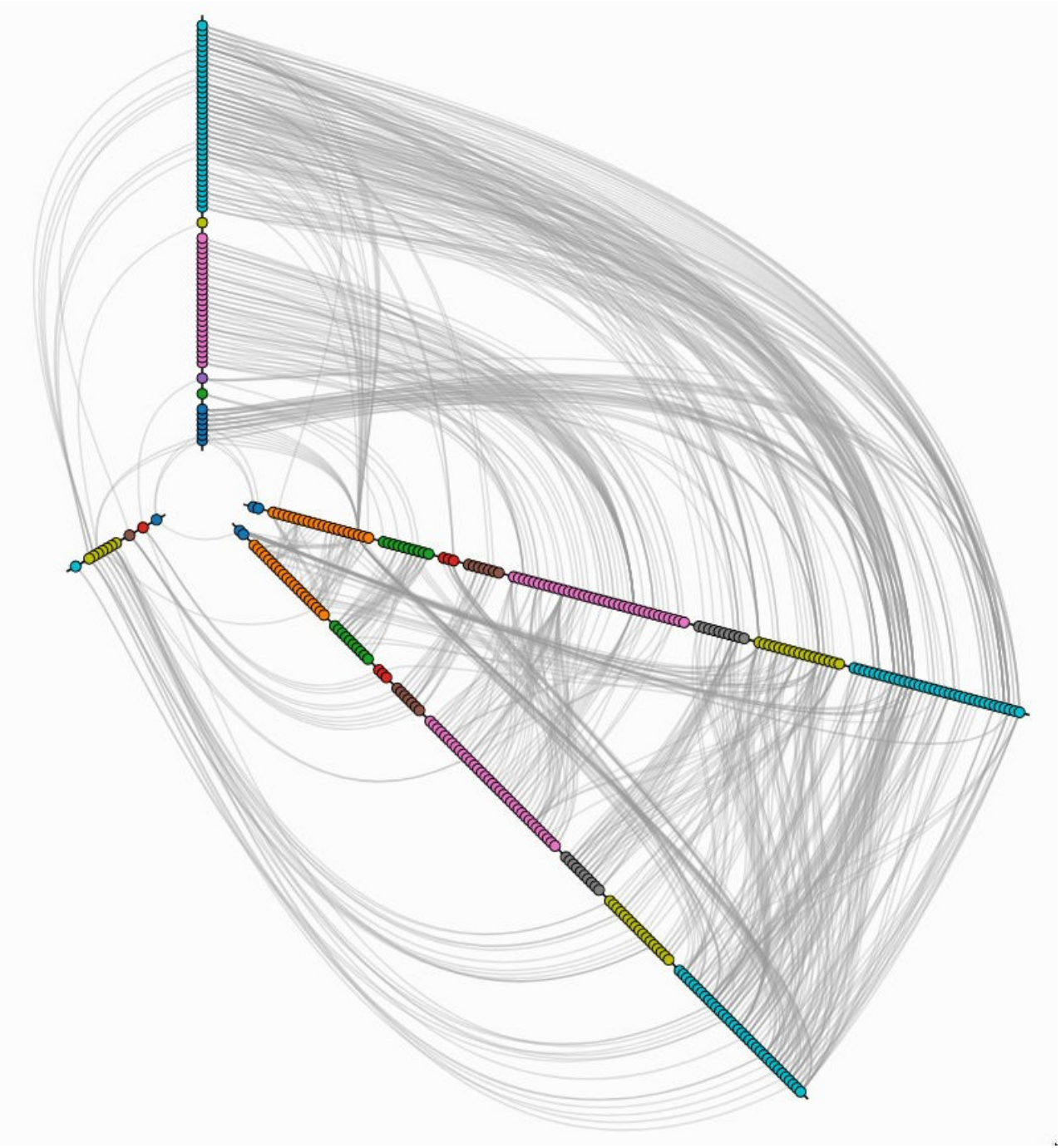
Only 2 variables (no nesting as in Tableau)

Doesn't support continuous variables

Multivariate edges?

ManyNets [Freire et al. ’10]





HivePlots

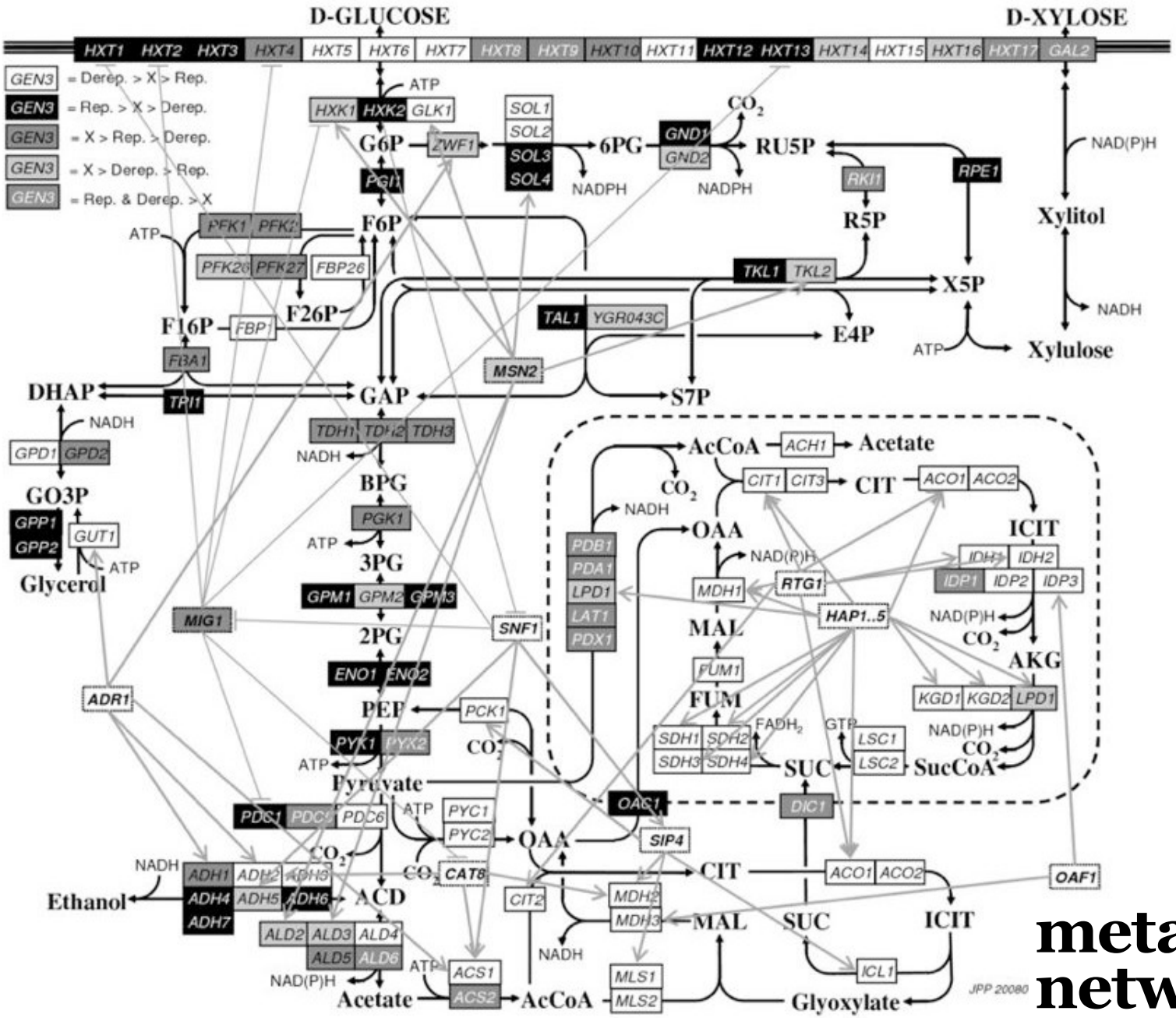
[Krzywinski '11]

Nodes (dots) may be replicated.

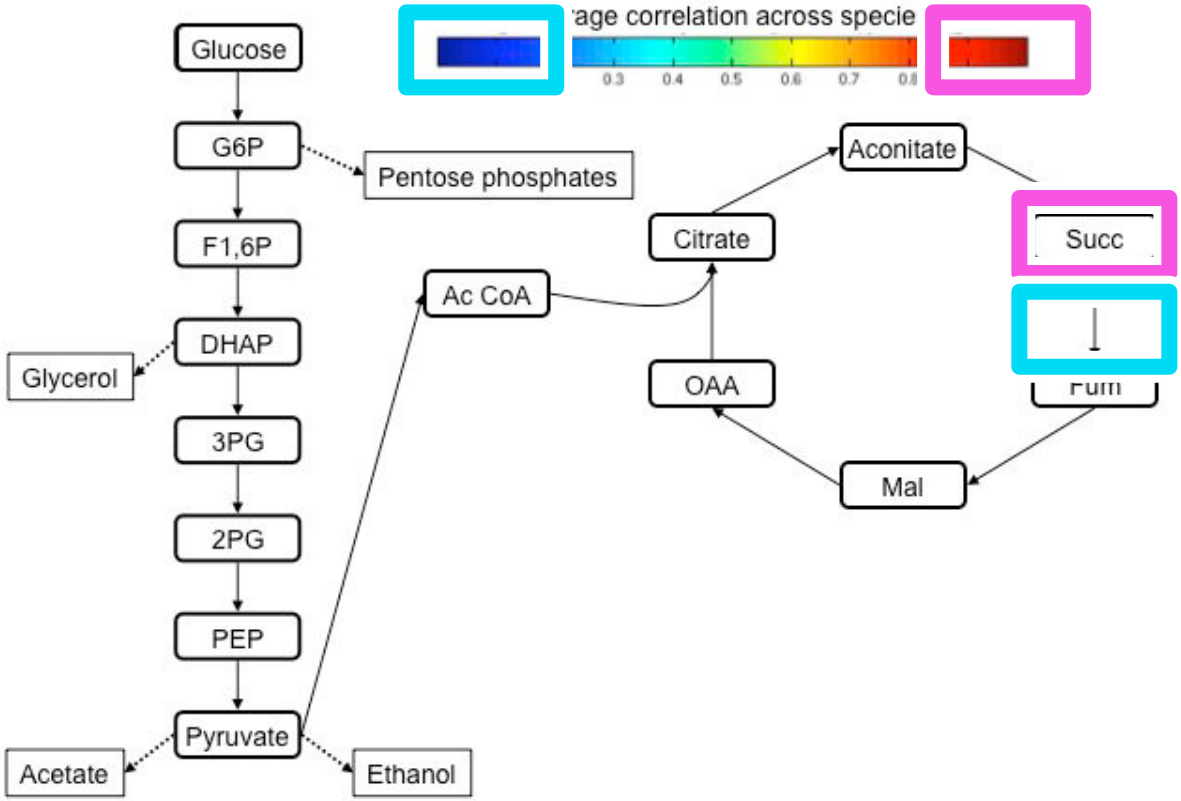
Nodes sorted on radial axes by network statistics (e.g., by degree).

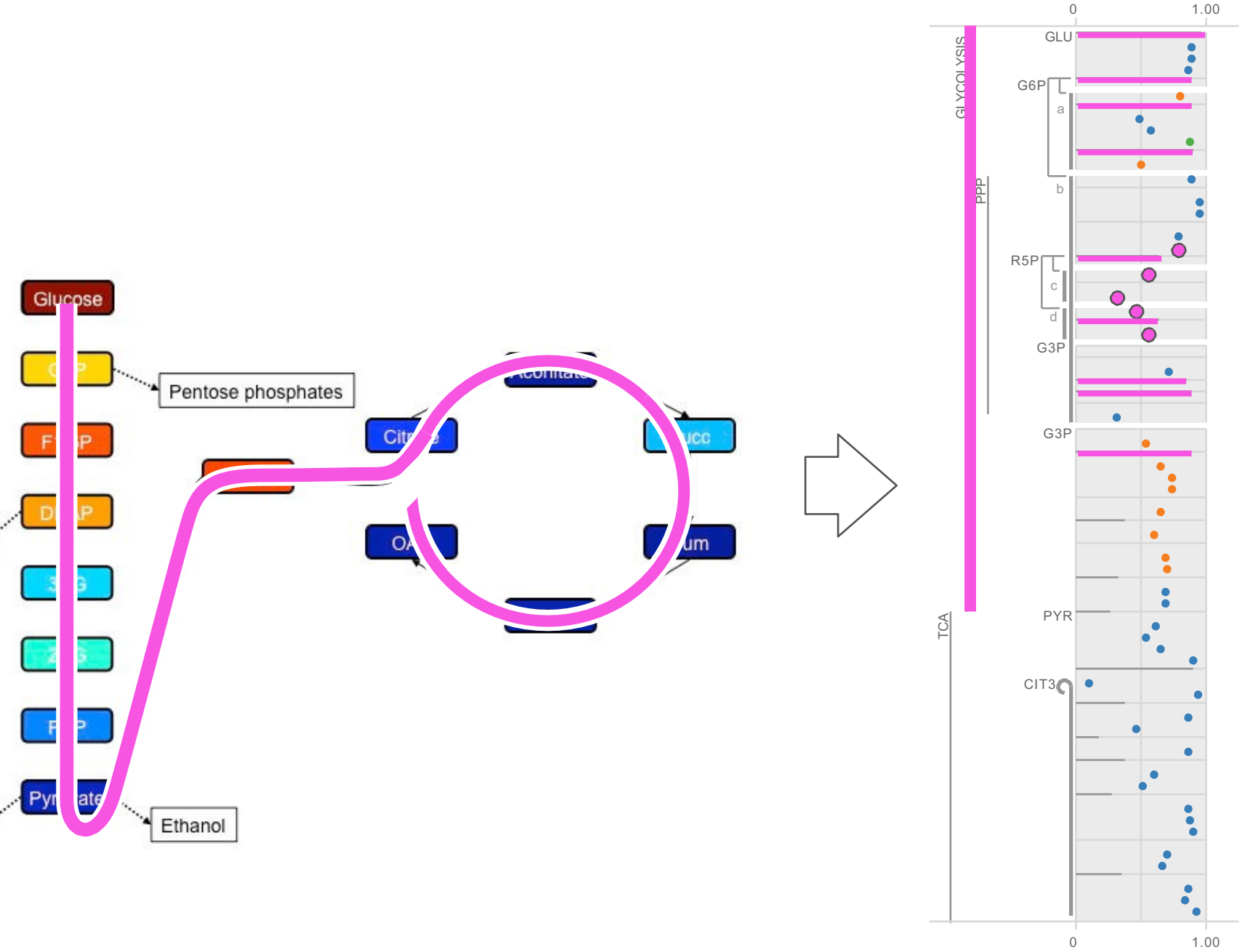
Different axes may contain different subsets of nodes.

egweb.bcgsc.ca



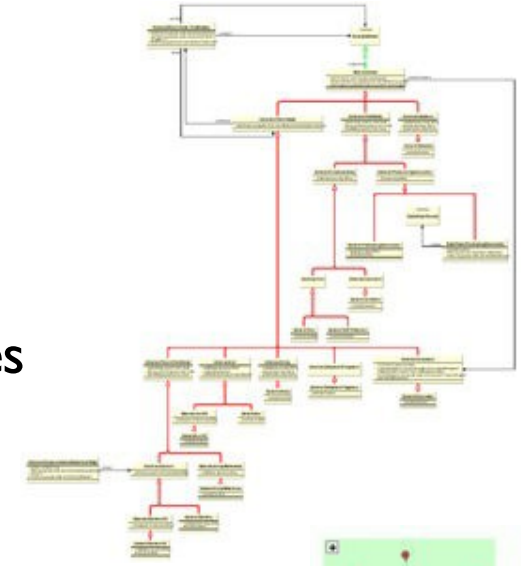
metabolic network

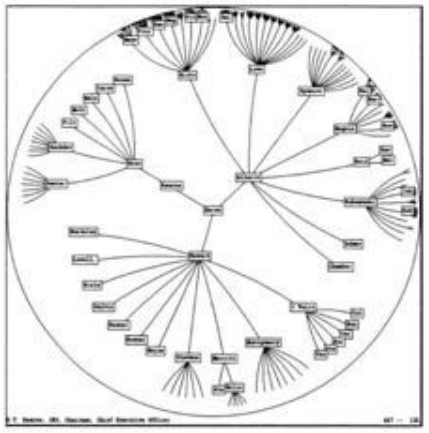

age correlation across specie
0.3
0.4
0.5
0.6
0.7
0.8
Glucose
G6P
Pentose phosphates
F1,6P
DHAP
Glycerol
3PG
2PG
PEP
Pyruvate
Acetate
Ethanol
Ac CoA
Citrate
Aconitate
Succ
OAA
Mal

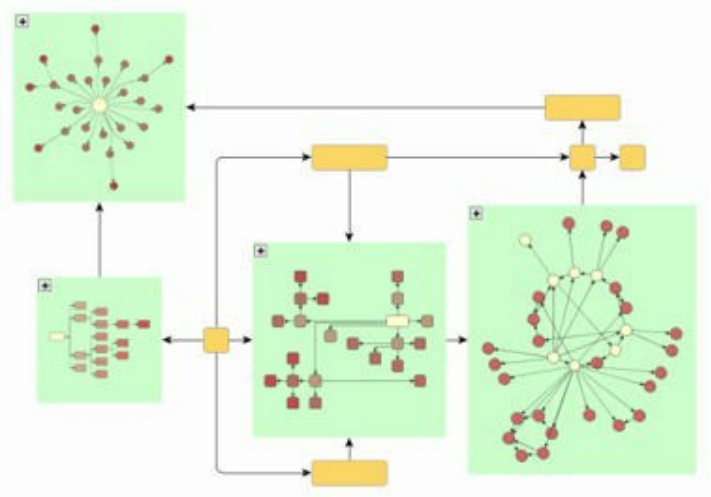

Glucose
Pentose phosphates
Ethanol
0
1.00
GLYCOLYSIS
PPP
TCA
GLU
G6P
a
b
R5P
c
d
G3P
G3P
PYR
CIT3
0
1.00

OTHER NODE LINK LAYOUTS

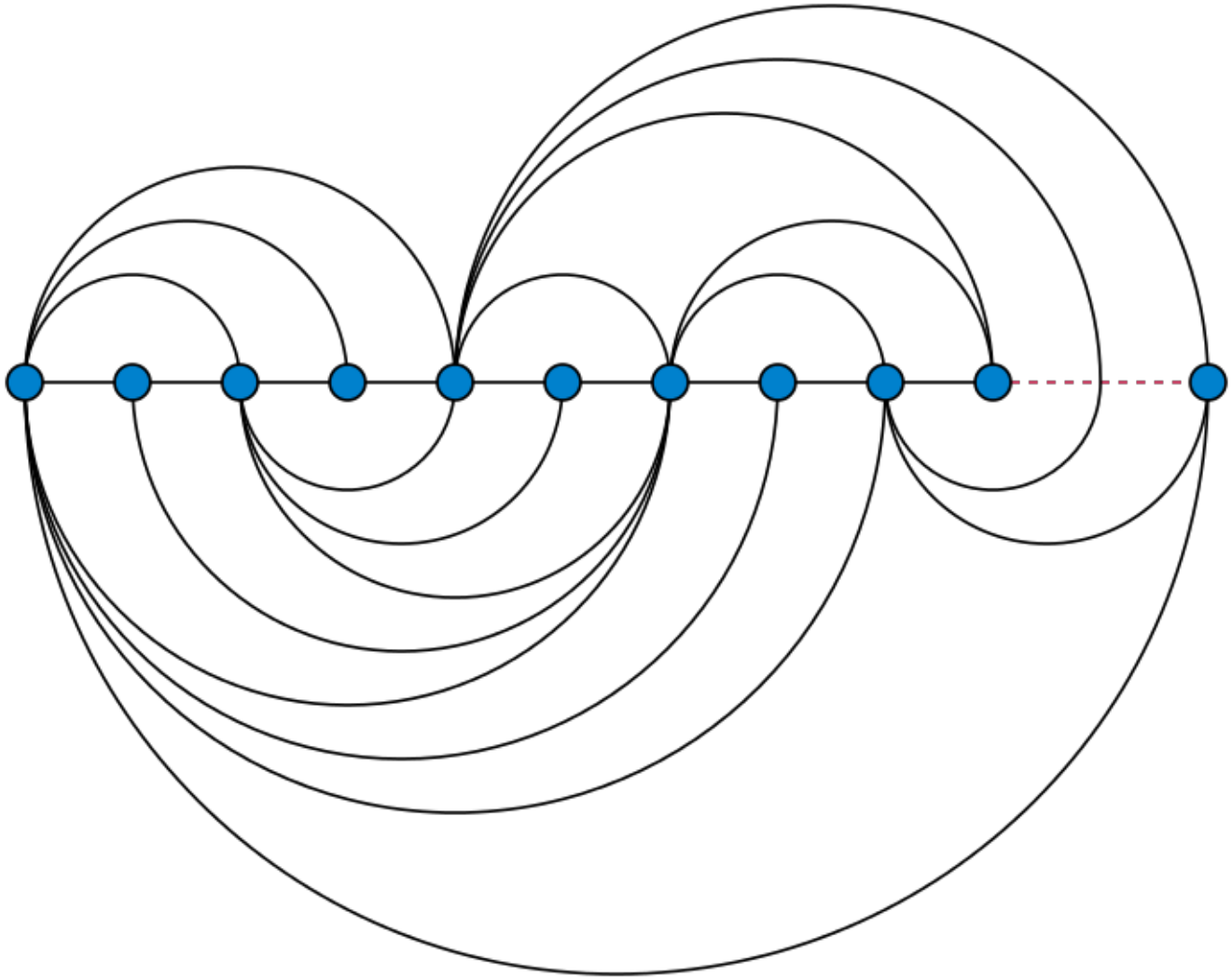
- **orthogonal**
  - great for UML diagrams
  - algorithmically complex
- **circular layouts**
  - emphasizes ring topologies
  - used in social network diagrams
- **nested layouts**
  - recursively apply layout algorithms
  - great for graphs with hierarchical structure







More node link layouts - Arc Diagram



Summary: Hierachies & Networks

Graph Layout

Spanning Tree Layout, “Sugiyama” Layout

Arc Diagrams

Force-Directed Layout, Optimization Methods

Matrix Diagrams

Attribute-Driven Layout