Network science / Graph mining

Metrics for analyzing a connected world

Erwan Le Merrer[1]

[1]Inria, Rennes

ESIR 2022

Outline

1. Graphs and representations
2. Classical metrics
3. Modeling and generating graphs
4. Exploring graphs
5. Importance metrics
6. Community metrics
7. Comparing graphs
8. TVGs: time varying graphs

Outline

1. Graphs and representations
2. Classical metrics
3. Modeling and generating graphs
4. Exploring graphs
5. Importance metrics
6. Community metrics
7. Comparing graphs
8. TVGs: time varying graphs

Graphs ?

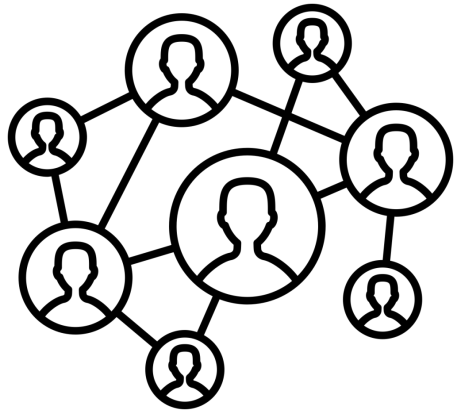


Figure: A graph: entities (nodes/vertices) and connections (edges)

An abstraction/representation for reasoning about characteristics of

- physical networks (computers, roads, circuits).
- relational data.

Focus on the structure rather than on the details of modeled objects

The omnipresence of graphs in applications

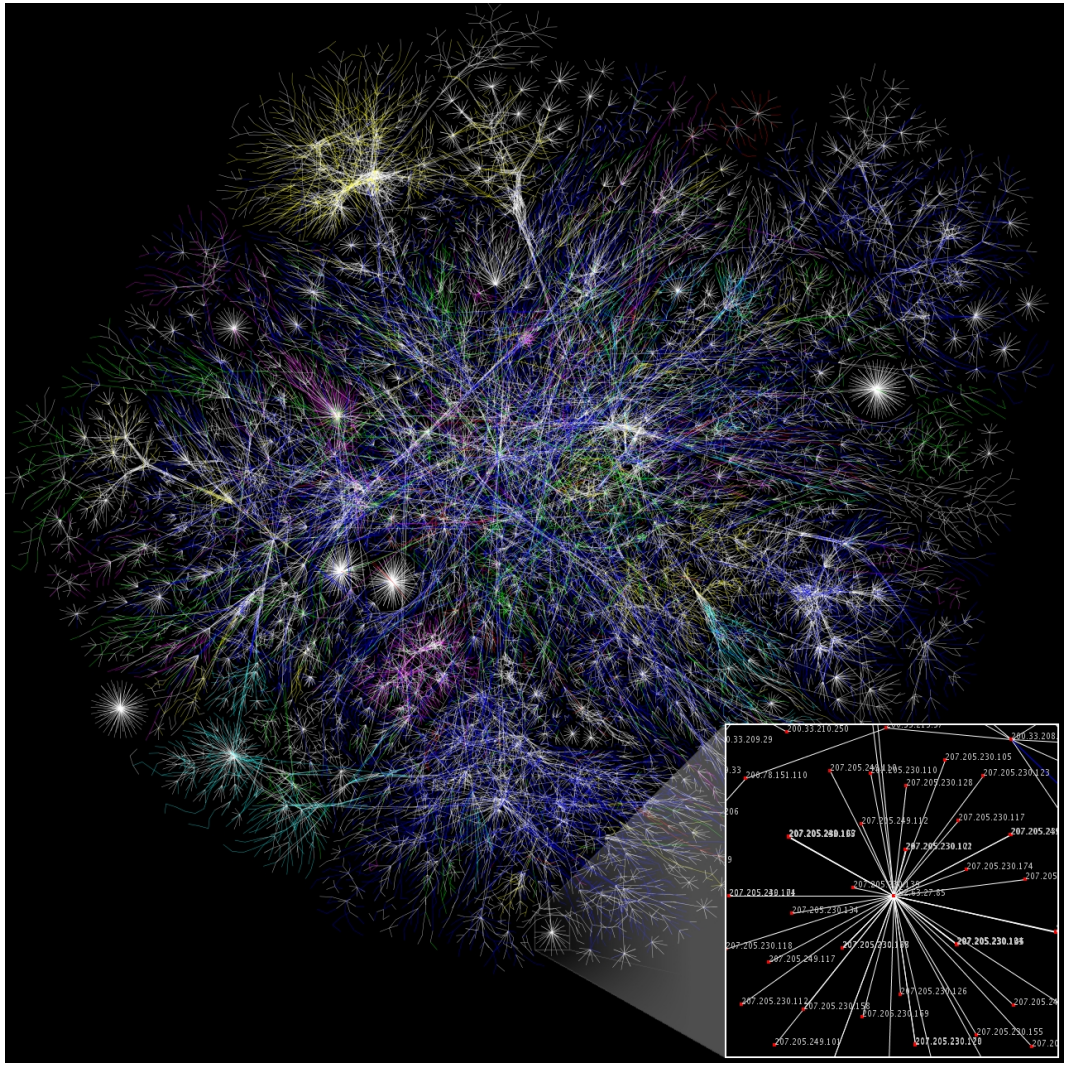


Figure: The Internet AS graph

The omnipresence of graphs in applications

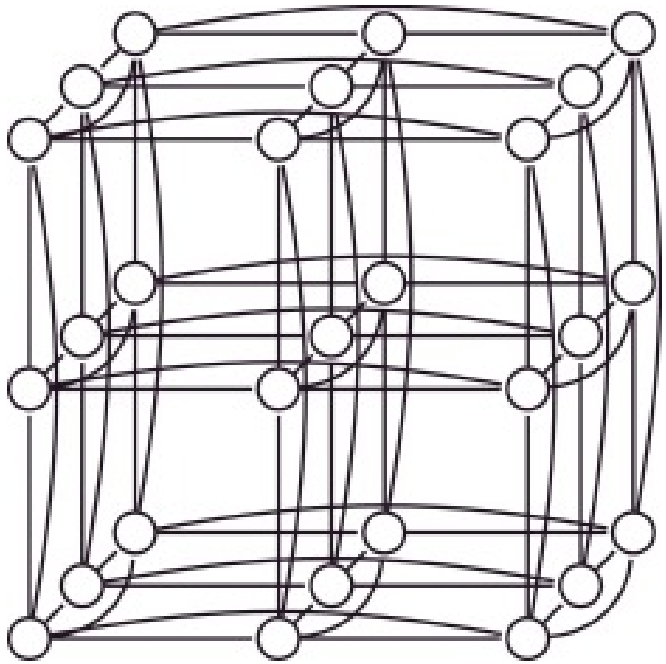


Figure: Interconnecting system-on-chips in a datacenter rack

The omnipresence of graphs in applications

- exemple use in social nets, epidemics...

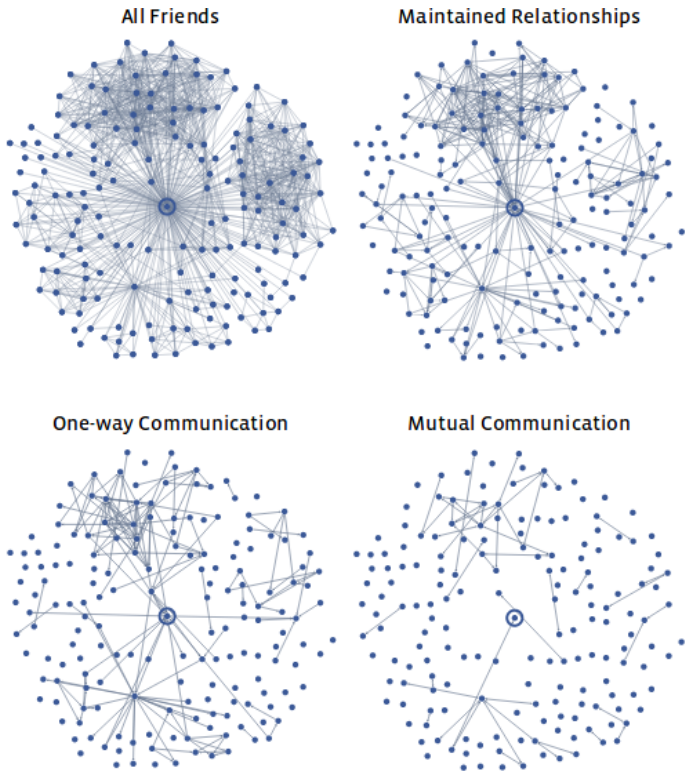


Figure 3.8: Four different views of a Facebook user's network neighborhood, showing the structure of links coresponding respectively to all declared friendships, maintained relationships, one-way communication, and reciprocal (i.e. mutual) communication. (Image from [286].)

Figure: From Networks, Crowds, and Markets: Reasoning about a Highly Connected World . By David Easley and Jon Kleinberg. Cambridge University Press, 2010.

e.g.: recommendations on YouTube

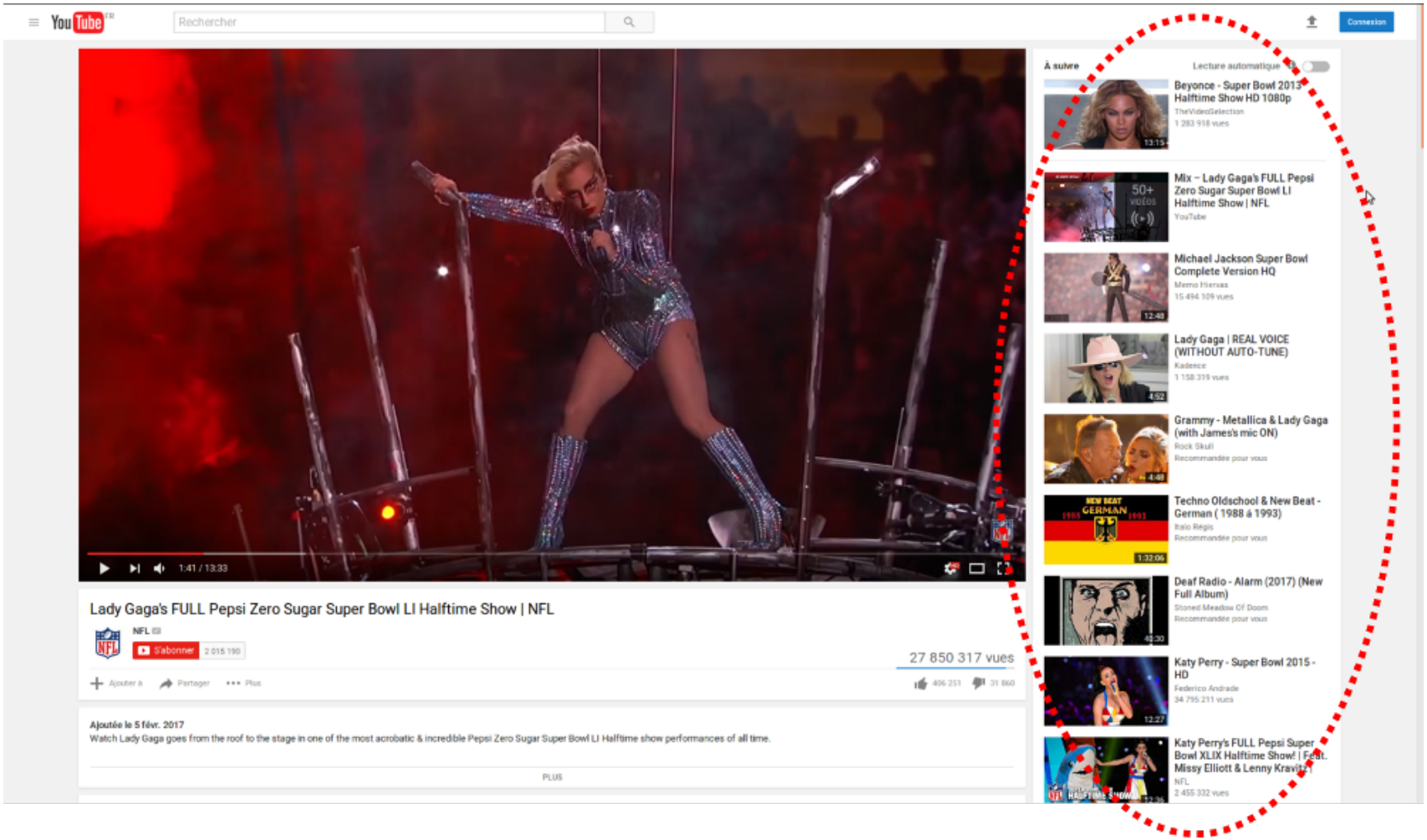
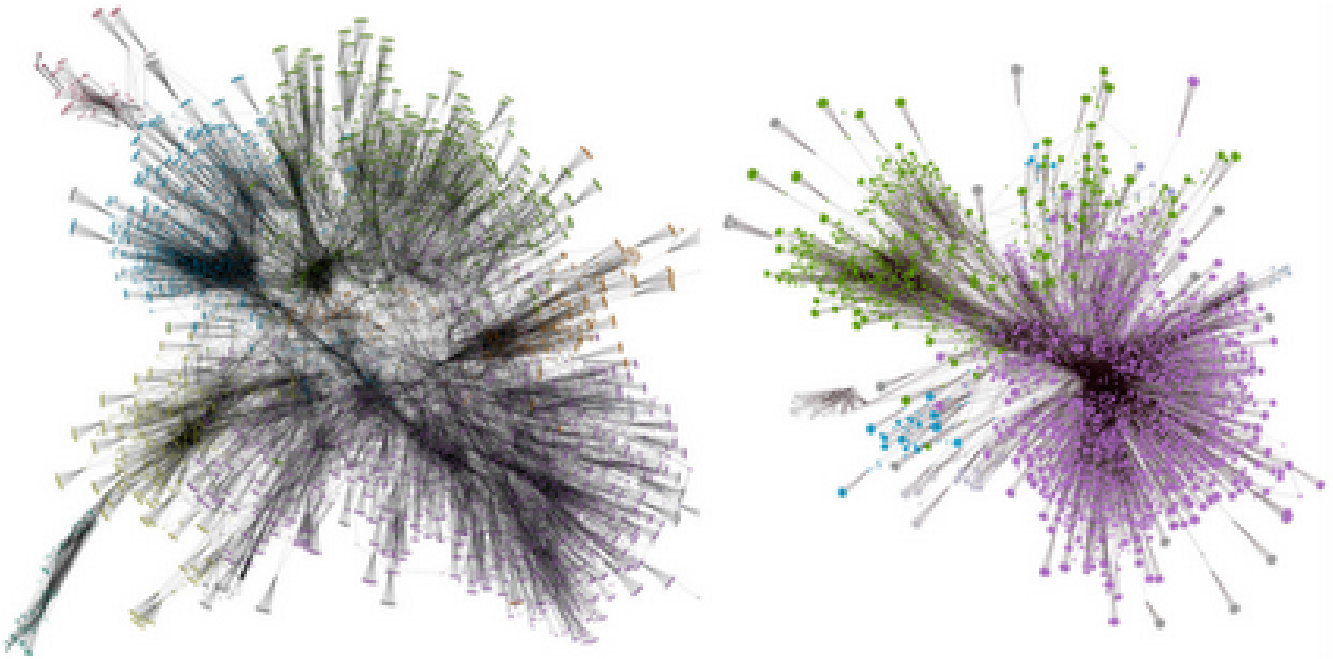


Figure: Recommandations: contextual, personalized?

e.g.: recommendations on YouTube

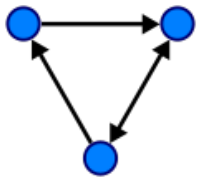


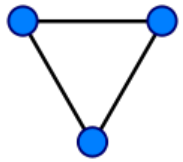
4-hops graphs from a YouTube video, new user (left) and returning user (ri

Figure: Blank profile vs. my recommandations

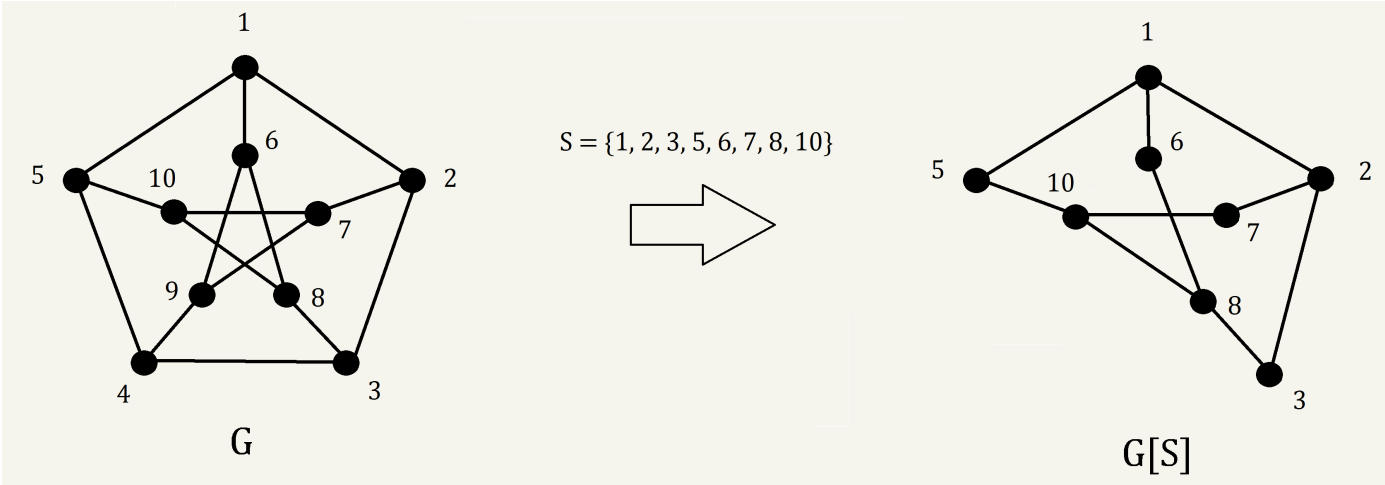
Core notions (1)

- *Directed* and *undirected* graphs:
  - in directed graphs, edges have orientation (arrow end)



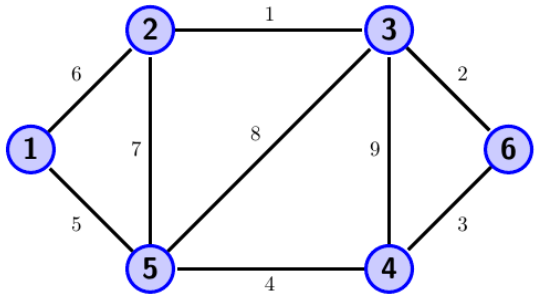


- A *subgraph* of G: formed by a subset of nodes/vertices and edges from G.

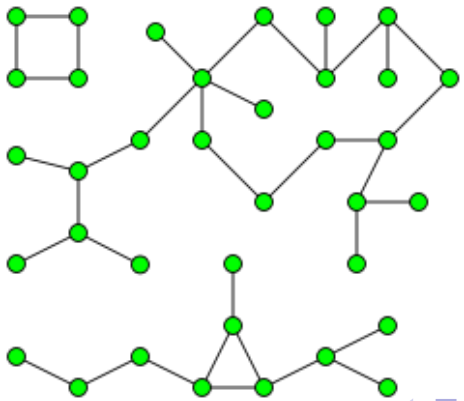


Core notions (2)

- Edge *weight*: value assigned as a label to an edge.
  - e.g., distance in km of a road from city 1 to 2.

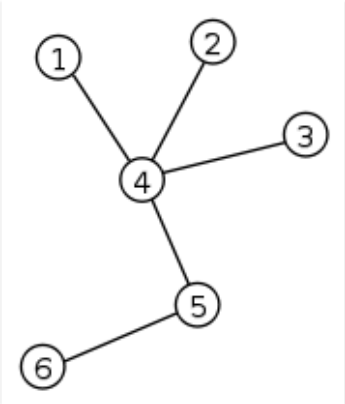


- Graph *connectivity:*
  - A graph is *connected* if there is a *path* btw any pair of vertices.
  - Otherwise, *connected components* are the subgraphs in which paths exist.

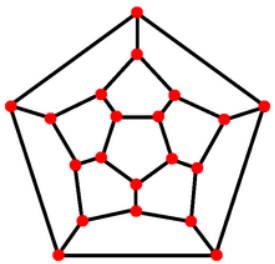


Core notions (3)

- A *cycle*: a path in which a vertex is reachable from itself.
  - Example of an *acyclic* connected graph: a *tree*



- A *planar* graph: can be draw without any edges crossing each other.



Special *topologies*

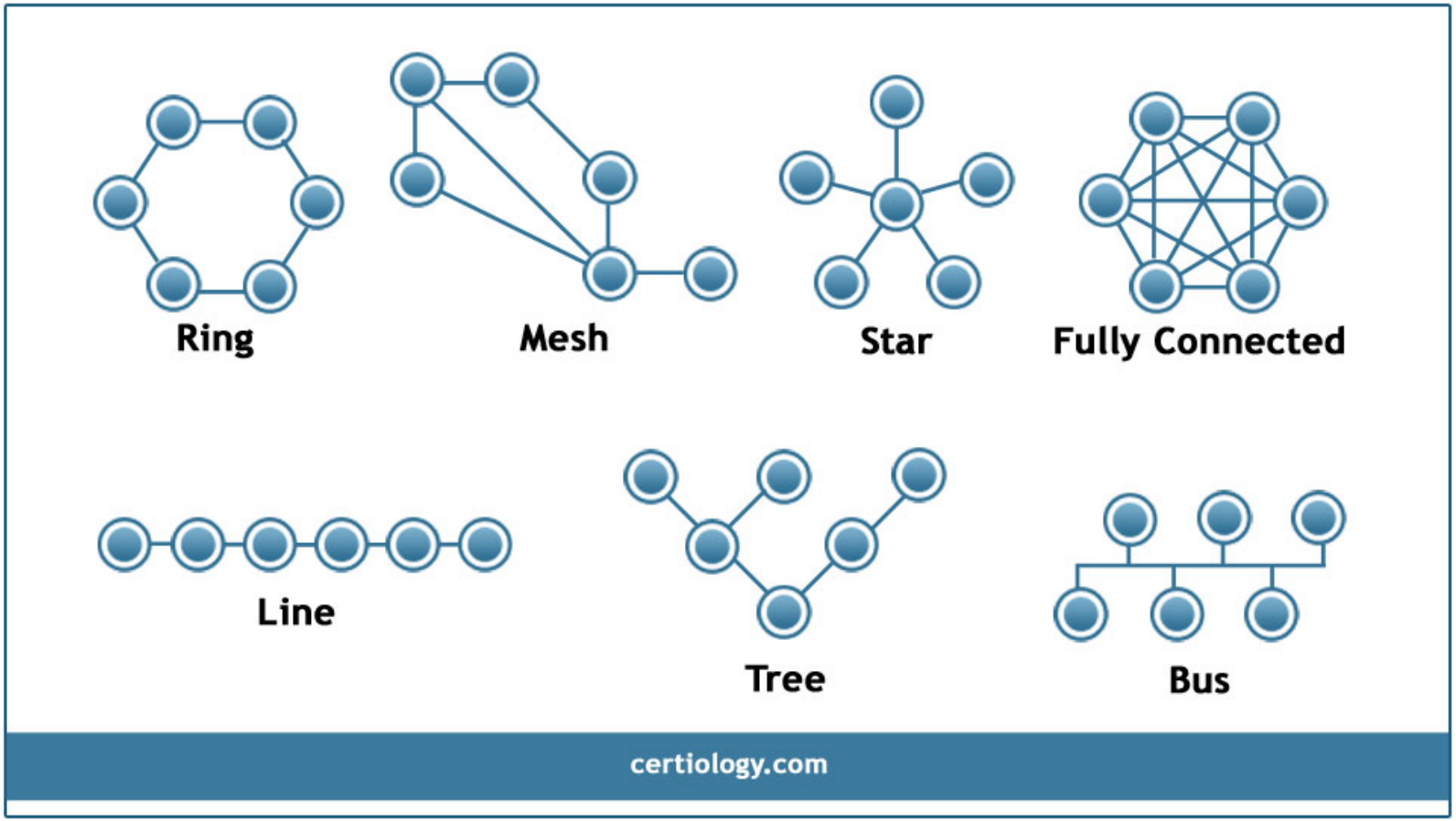
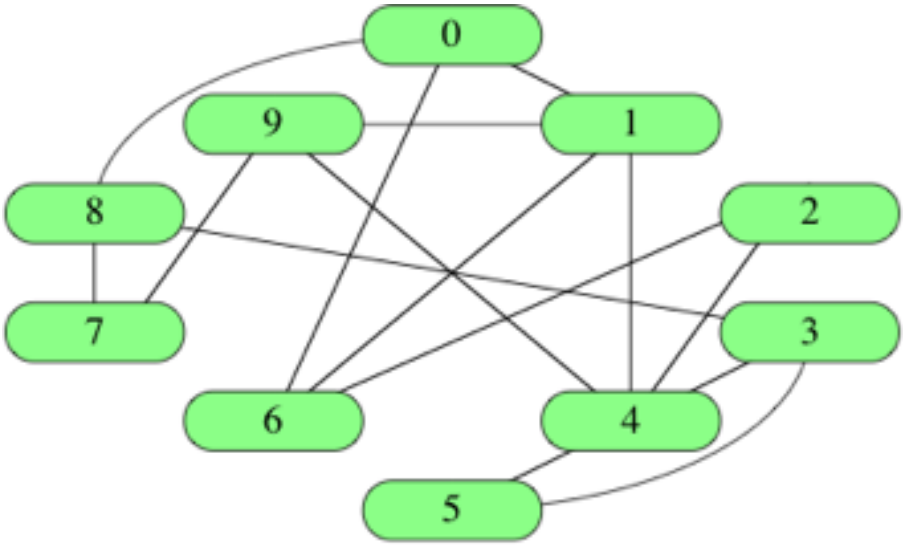


Figure: Graphs to remember, often used as illustrations

Adjacency list or edge list representations



Graph $G(V, E)$, with V: nodes and E: edges.
Edge list:

[[0,1], [0,6], [0,8], [1,4], [1,6], [1,9], [2,4], [2,6], [3,4], [3,5], [3,8],[4,5], [4,9], [7,8], [7,9]]

$O(|V|)$ access time to find an edge, but $O(|E|)$ space in memory.
Adjacency list:

[[1, 6, 8], [0, 4, 6, 9], [4, 6], [4, 5, 8], [1, 2, 3, 5, 9], [3, 4], [0, 1, 2], [8, 9], [0, 3, 7], [1, 4, 7]]

$O(1)$ access time to vertex , but $O(|V|)$ to access a given edge.[1]

[1] https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs

Matrix representation

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	0	1	0	1	0
1	1	0	0	0	1	0	1	0	0	1
2	0	0	0	0	1	0	1	0	0	0
3	0	0	0	0	1	1	0	0	1	0
4	0	1	1	1	0	1	0	0	0	1
5	0	0	0	1	1	0	0	0	0	0
6	1	1	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	1
8	1	0	0	1	0	0	0	1	0	0
9	0	1	0	0	1	0	0	1	0	0

Figure: Matrix representation of previous graph

Find edge presence in $O(1)$ time, but $\Theta(V^2)$ space in memory.
1's to be replaced by edge weigths for weighted graphs.

Example tool families for manipulating graphs

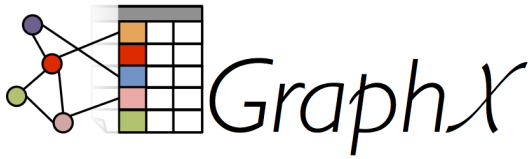


Figure: For massive graphs (cannot fit into on server's memory)

X – Stream

Figure: Big graph processing on a single machine

Figure: For a database-like handling of graphs

NetworkX

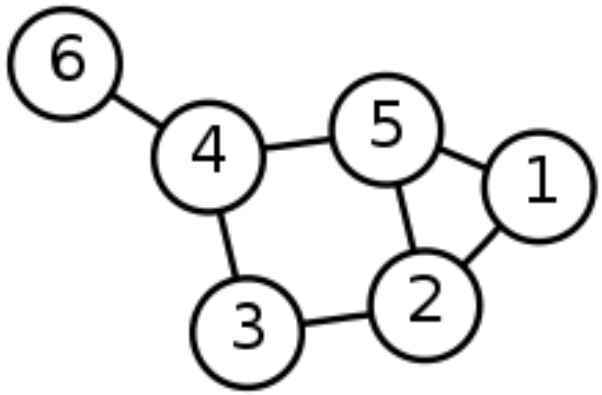
Figure: Prototyping in Python, lots of contributions

Stanford Large Network Dataset Collection

Outline

1. Graphs and representations
2. Classical metrics
3. Modeling and generating graphs
4. Exploring graphs
5. Importance metrics
6. Community metrics
7. Comparing graphs
8. TVGs: time varying graphs

Basic notations



- $G(V,E)$: graph G with node set V, connected by edge set E.
  - $V = \{1,2,3,4,5,6\}$;
    $E = [[1,5],[1,2],[2,3],[2,5],[3,4],[4,5],[4,6]]$
- Number of nodes is $n = |V|$, edges is $m = |E|$.
- Neighbors of node i are set $\Gamma(i)$.
  - $\Gamma(1) = \{2,5\}$

Degree of a node

- The degree d_v of node v is equal to $|\Gamma(v)|$ (its number of neighbors).
- Degree span: $0 \leq d_v \leq n-1$ (if no self loops).
- *Degree distribution* P(d) is the probability distribution of each degree in the current graph:

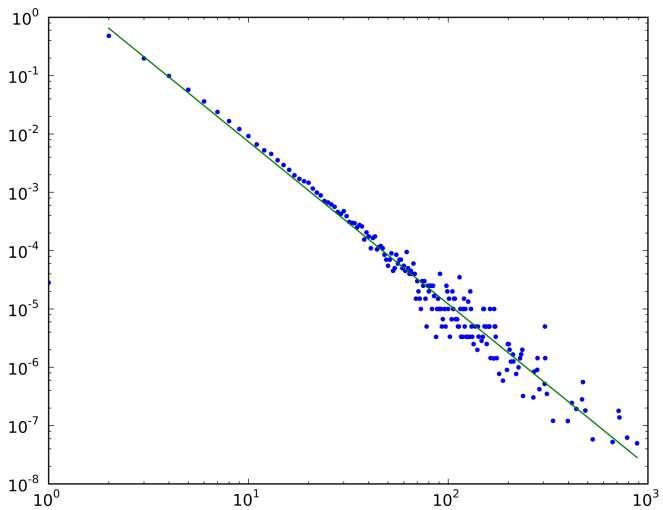


Figure: Degree distribution: x-axis is degree, y-axis is probability

- In(out)-degree of v: counts incoming(outgoing) edges only.

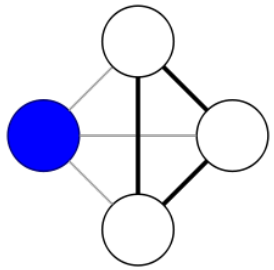
Clustering coefficient

- Every two nodes in a *clique* are neighbors.
- *Local clustering coefficient of a node i measures "how close are $\Gamma(i)$ from being a clique":*

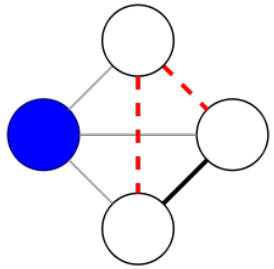
$$C_i = \frac{2|e_{jk} : v_j, v_k \in \Gamma(v_i), e_{jk} \in E|}{d_i(d_i - 1)}$$

- Average clustering coefficient:

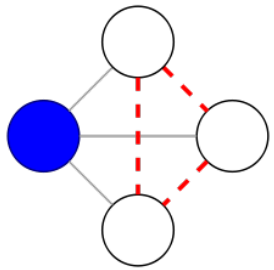
$$\bar{C} = \frac{1}{n} \sum_{i=1}^{n} C_i$$



c = 1



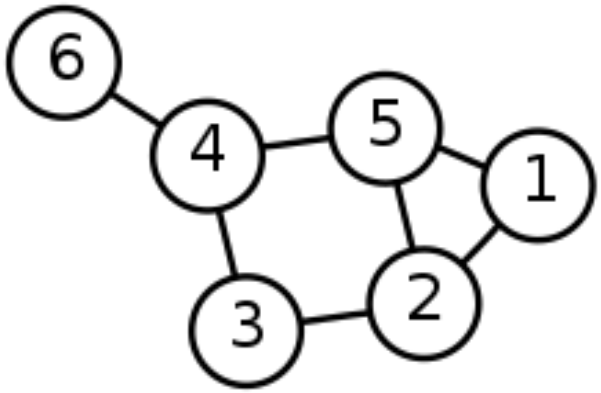
c = 1/3



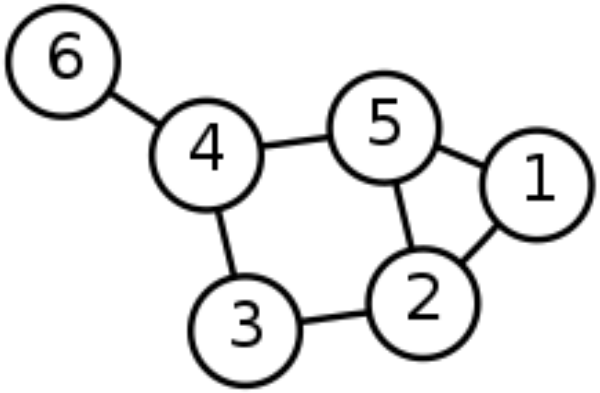
c = 0

Path lengths

- *Path*: sequence of adjacent nodes connecting two nodes (if exists).
  - e.g., two paths btw 6 and 1: (4,5,1) and (4,3,2,5,1).
  - One *hop*: one transition from a node to another.
- *Shortest path*: path of minimal cardinality.
  - Distance $dist(6,1) = |(4,5,1)| = 3$
- *Single-source shortest path* (SSSP): shortest paths from node i to all other nodes ($V \setminus i$).
- *All-pairs shortest paths (APSP)*: SSSP from $\forall i \in V$.



Diameter



- *Average path length*: average of all-pair shortest distances in the graph.
- *Diameter*: longest path of the APSP, i.e., greatest distance between any pair of vertices.
  - $diam(G) = |(4,5,1)| = 3$, starting at node 6.

The Laplacian matrix $L_G = D - A$:

- *D is the degree matrix a diagonal-matrix with* $D(i,i)$ *is the degree of the* i*th node in* G
- *A is the adjacency matrix, with* $A(i,j) = 1$ *if and only if* $(i,j) \in E$

$$L_G(i,j) = \begin{cases} deg(i) & \text{if } i = j \\ -1 & \text{if } (i,j) \in E \equiv 1 \\ 0 & \text{otherwise} \end{cases}$$

Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

Spectral analysis (2)

For an (undirected) graph G and its ***Laplacian matrix*** $L = D - A$ *with eigenvalues* $\lambda_0 \leq \lambda_1 \leq ... \leq \lambda_{n-1}$*:*

- $\lambda_0 = 0$*, as* $v_0 = (1, 1, ..., 1)$ *satisfies* $Lv_0 = 0$ *(row sum and column sum of* L *are* 0*)*
- *# of connected components in* G *is the algebraic multiplicity of the 0 eigenvalue (*$\Longrightarrow$ $\lambda_2 = 0$ *iff* G *is disconnected)*
- *the smallest non-zero eigenvalue of* L *is called the* ***spectral gap***
- *the second smallest eigenvalue of* L *(could be zero) is the* ***algebraic connectivity*** *of* G
- *...*

Spectral analysis (3)

An example of a result: the diameter of a non complete graph G satisfies:

$$diam(G) \leq \lceil \frac{\log(vol(G)/\delta)}{\log \frac{\lambda_{n-1}+\lambda_1}{\lambda_{n-1}-\lambda_1}} \rceil,$$

with δ the minimum degree of G and $vol(G)$ is the sum of the degrees of the vertices in G.

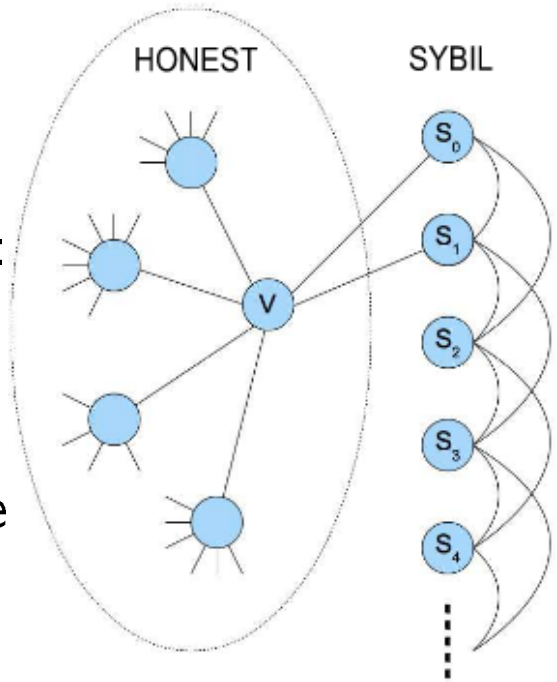
. . . and multiple results from graph theory, in general or for specific graphs

Conductance

- The conductance $\Phi(C)$ of a set C of vertices in a given graph G is the ratio between the number of edges going out from C and the number of edges inside C:

$$\Phi(C) = \frac{|cut(C)|}{vol(C)},$$

where $vol(C)$ is the sum of the degrees of the vertices in C.

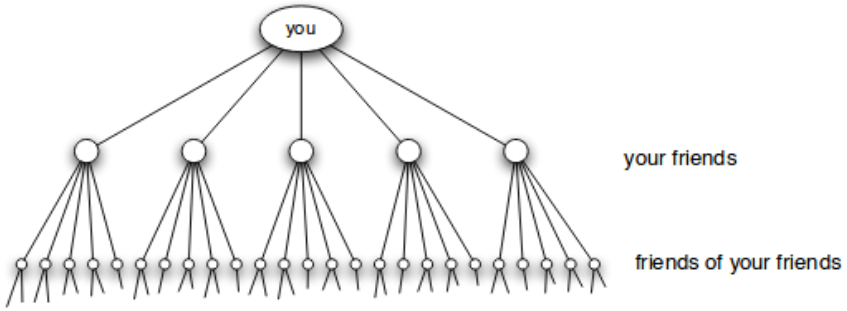


Expansion

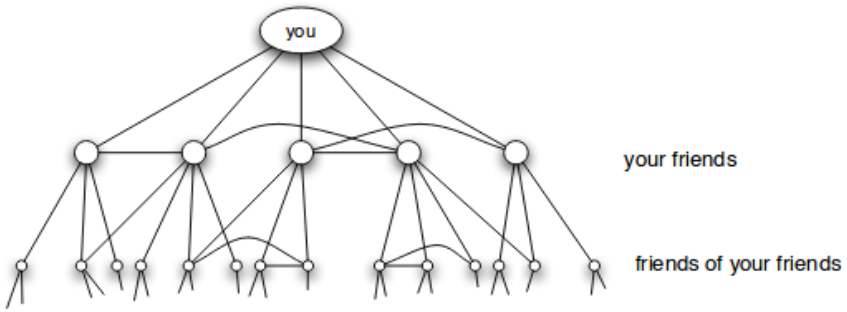
- *Expansion* of G: mean number of nodes that are reached in h hops from all nodes:

$$e_G(h) = \frac{1}{n^2} \sum_{v \in V} |C_v(h)|,$$

with $C_v(h)$ the set of reachable nodes from v in h hops.



(a) *Pure exponential growth produces a small world*



(b) *Triadic closure reduces the growth rate*

Resilience

- Measures the robustness of a graph:

$$r_G(h) = \frac{1}{|E|} \sum_{v \in V} l(v, |C_v(h)|),$$

with $l(v, |C_v(h)|)$ the number of edges that need to be removed to split $C_v(h)$ into 2 sets (of roughly the same size). h: distance (hops).

Outline

1. Graphs and representations
2. Classical metrics
3. Modeling and generating graphs
4. Exploring graphs
5. Importance metrics
6. Community metrics
7. Comparing graphs
8. TVGs: time varying graphs

The Erdős–Rényi random graph

- Model $G(n,p)$ for generating a canonical random graph.
  - Create n nodes.
  - Every pair of nodes connected with independant probability p.

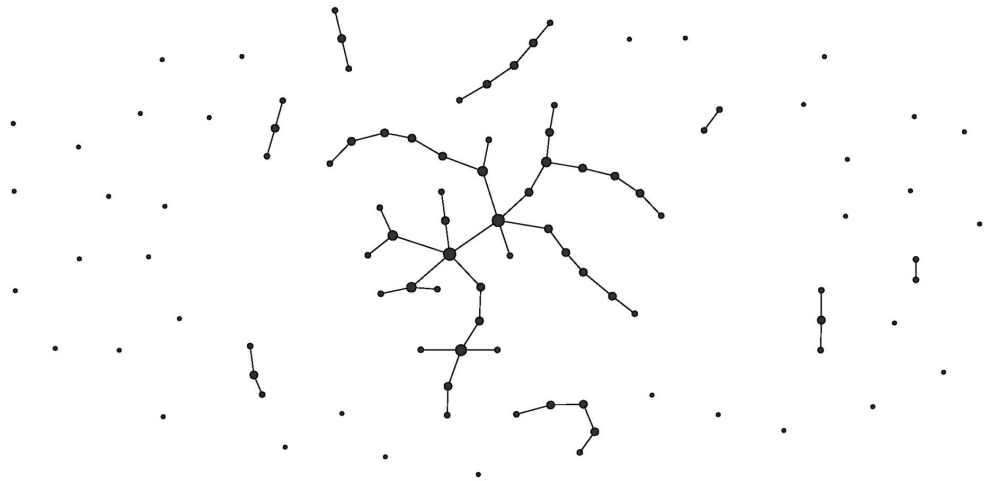
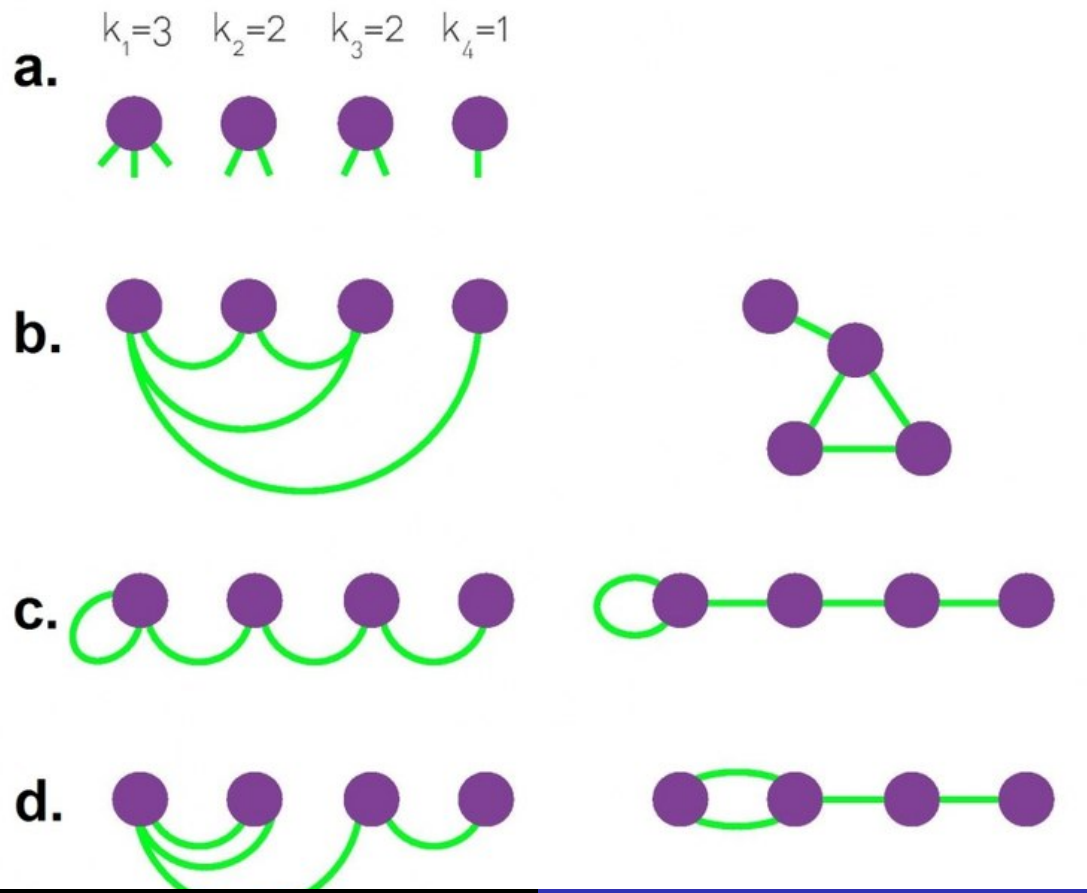


Figure: A random graph with $p = 0.01$.

- If $np = 1$, G almost surely has a largest component of $n = O(n^{2/3})$.
- $p = \frac{\ln n}{n}$ is a threshold for G's connectivity.
- For a large n, resulting degree distribution is Poisson

The configuration model

- Arbitrary degree distribution: to choose
  - Create n nodes with each a given target degree, fitting the distribution
  - Loop: take one node with remaining "free" neighbor, and selected another node randomly to add an edge



The Barabási–Albert scale-free graph

- Model to generate a graph with *power-law* degree-distribution.
  - Create m_0 nodes, as a connected graph.
  - Iteratively add one node, and connect it to $m < m_0$ nodes, with probability depending on the degree of existing nodes: $p_i = \frac{d_i}{\sum_j d_j}$ (method called preferential attachment).

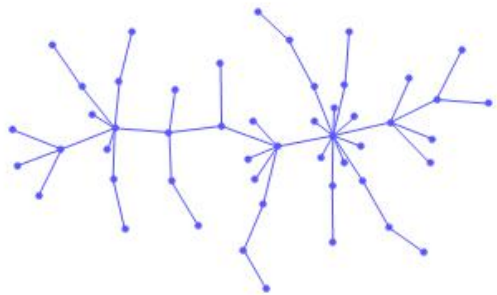


Figure: A Barabási-Albert graph with $n = 50$ and $m_0 = 1$.

- Well connected nodes "accumulate" incoming links: rich gets richer
- Resulting degree distribution is $P(d) \sim d^{-3}$.
- Average path length is $\frac{\ln n}{\ln \ln n}$.

A real structure example

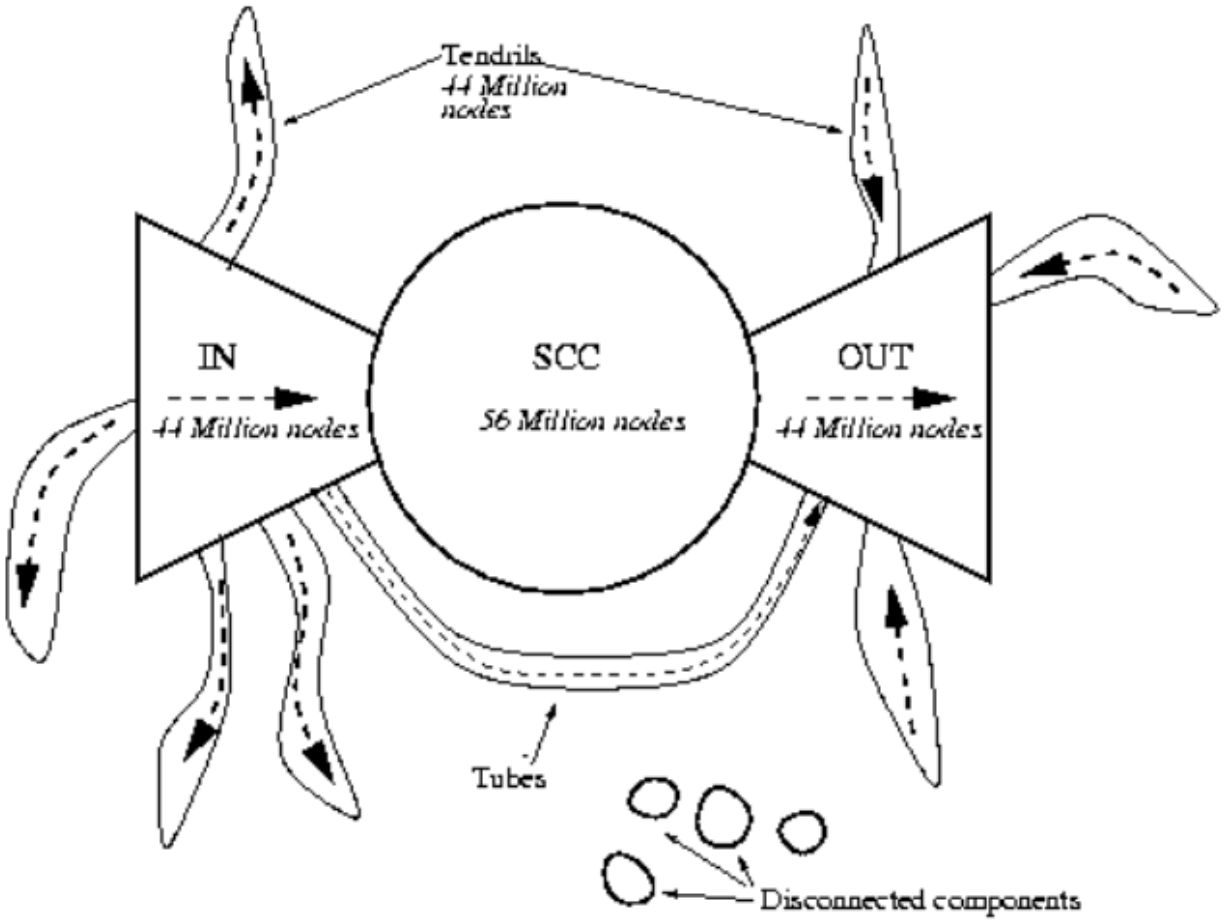
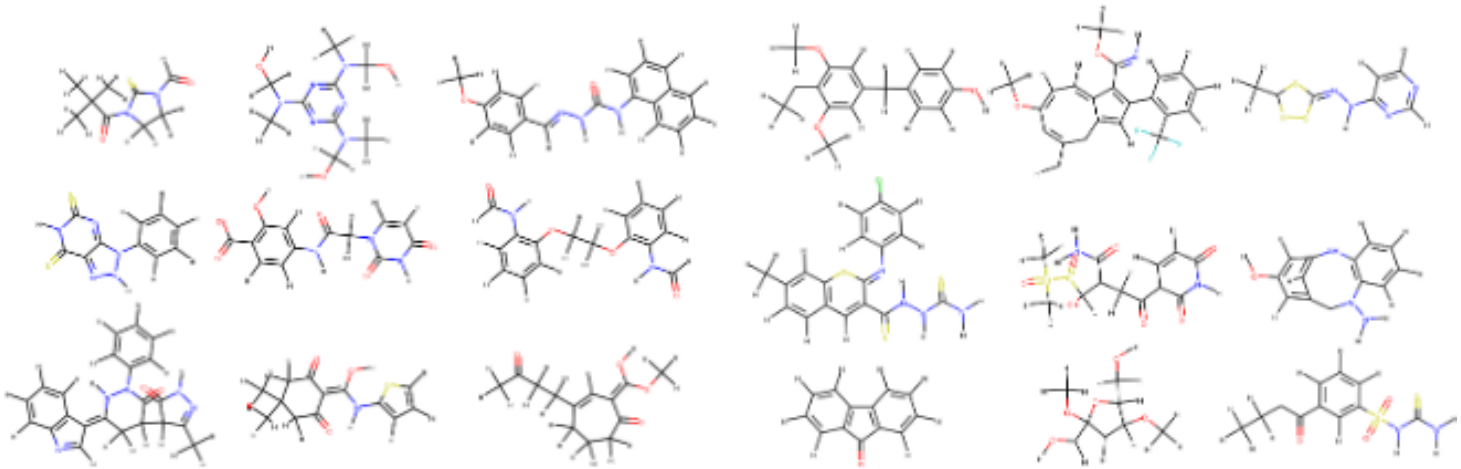


Figure: Bow-tie structure of the web

Generating graphs with neural nets: GraphGen (1)

- How to generate graphs from an unknown generative process?

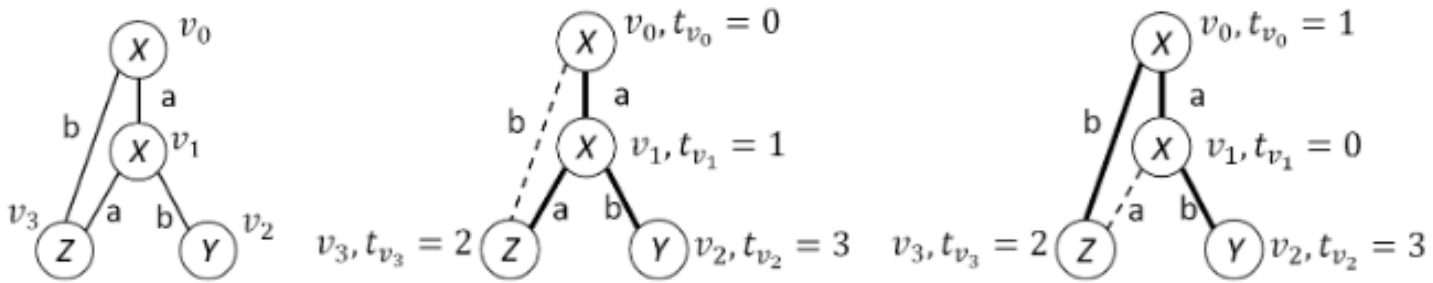


(a) Real graphs

(b) GraphGen

Generating graphs with neural nets: GraphGen (2)

- Instances of graphs in a categorie $\rightarrow$ learn $\rightarrow$ generate others [8]
- Converting a graph to a sequence



(a) Graph (b) DFS traversal on the graph (c) DFS traversal on the graph

Figure: Extracting DFS "codes" from a graph to learn from.

- 5-tuples $(t_u, t_v, L_u, L_{(u,v)}, L_v)$, with L the label. Fig (b): $(0,1,X,a,X),(1,2,X,a,Z),(2,0,Z,b,X),(1,3,X,b,Y)$.

Generating graphs with neural nets: GraphGen (3)

- A recurrent neural network (RNN) learns a DFS sequence S $p(S) = \prod_{i=1}^{m+1} p(s_i|s_1, ..., s_{i-1})$ (i.e., conditional distribution over the elements.)
- Generation from $m = 1$ to $m = |E|$, one single edge at a time

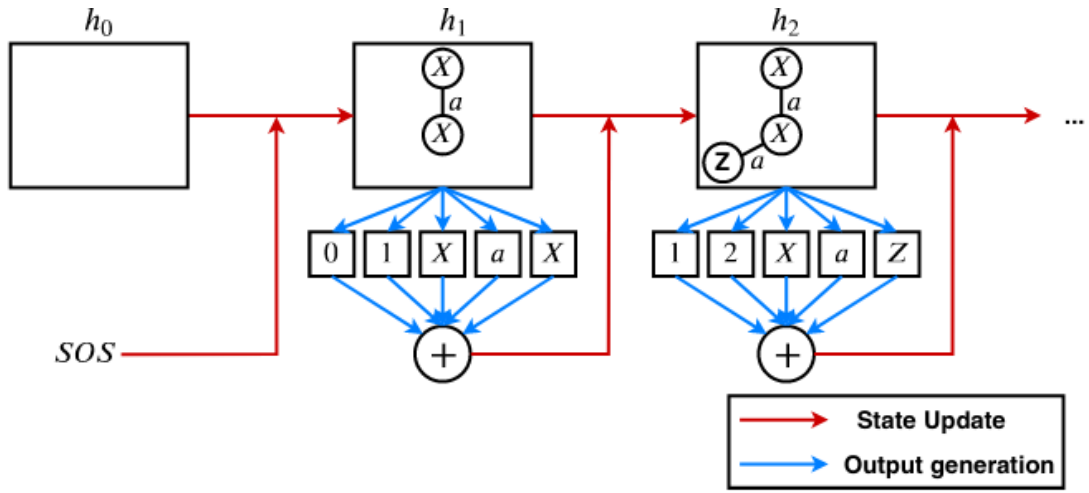
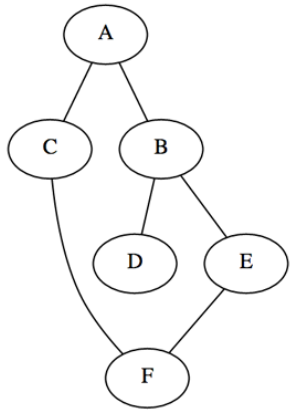


Figure 3: Architecture of GraphGen. Red arrows indicate data flow in the RNN whose hidden state h_i captures the state of the graph generated so far. Blue arrows show the information flow to generate the new edge tuple s_i.

Outline

1. Graphs and representations
2. Classical metrics
3. Modeling and generating graphs
4. Exploring graphs
5. Importance metrics
6. Community metrics
7. Comparing graphs
8. TVGs: time varying graphs

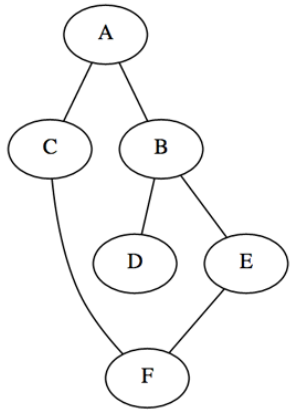
Depth first search



- Graph exploration, from a given start node, *depth* first:

```
def dfs(graph, start):
    visited, stack = set(), [start]
    while stack:
            vertex = stack.pop()
            if vertex not in visited:
                visited.add(vertex)
                stack.extend(graph[vertex] - visited)
    return visited
```

Breadth first search

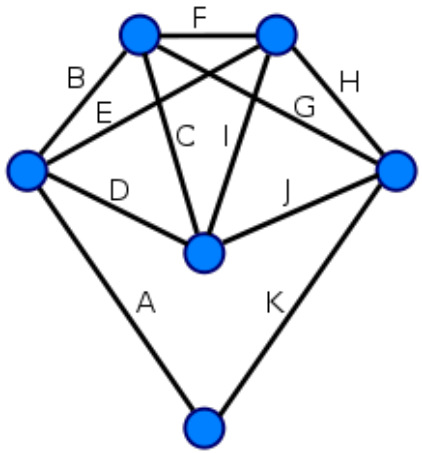


- *breadth* first:

```
def bfs(graph, start):
    visited, queue = set(), [start]
    while queue:
        vertex = queue.pop(0)
        if vertex not in visited:
            visited.add(vertex)
            queue.extend(graph[vertex] - visited)
return visited
```

- Queue $\rightarrow$ search in vertices breadth (FIFO)

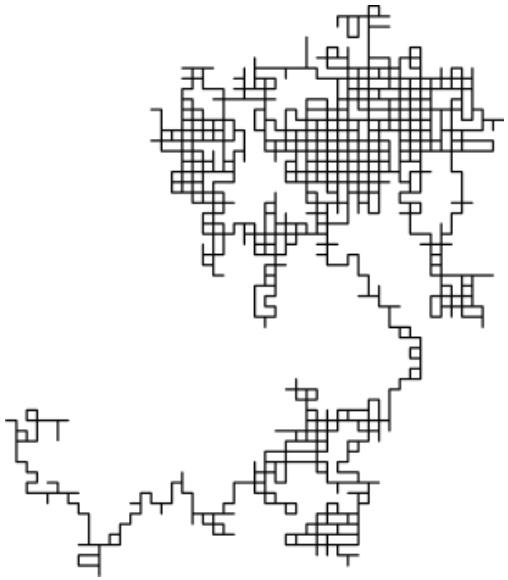
Eulerian path

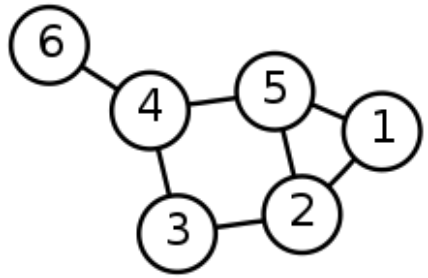


- An Eulerian path visits every edge exactly once (allowing for revisiting vertices).
- Euler's Theorem: A connected graph has an Euler cycle if and only if every vertex has even degree.

Random walk

- Randomized exploration.
- Given a graph and a *start* node, a simple random walk [1] proceeds by random steps:
  - selects uniformly at random a neighbor from walk position
  - jump on it
  - loop process





RDW(6,7hops)=(6,4,3,4,5,4,3,2)

Figure: Random walk on a grid (i.e., 4 neighbors per node)

Random walks - app1: sampling (biased)

- Select a random node in the graph (but biased).
- Given a graph, a *start* node, and a "large" h use a simple random walk:
  - selects uniformly a neighbor; jump on it; $h \leftarrow h-1$
  - loop until $h \leq 0$
- Results in probability of node j to be selected: $P_j = \frac{d_j}{\Sigma_{i=0}^{n} d_i}$

Random walks - app1: sampling (uniform)

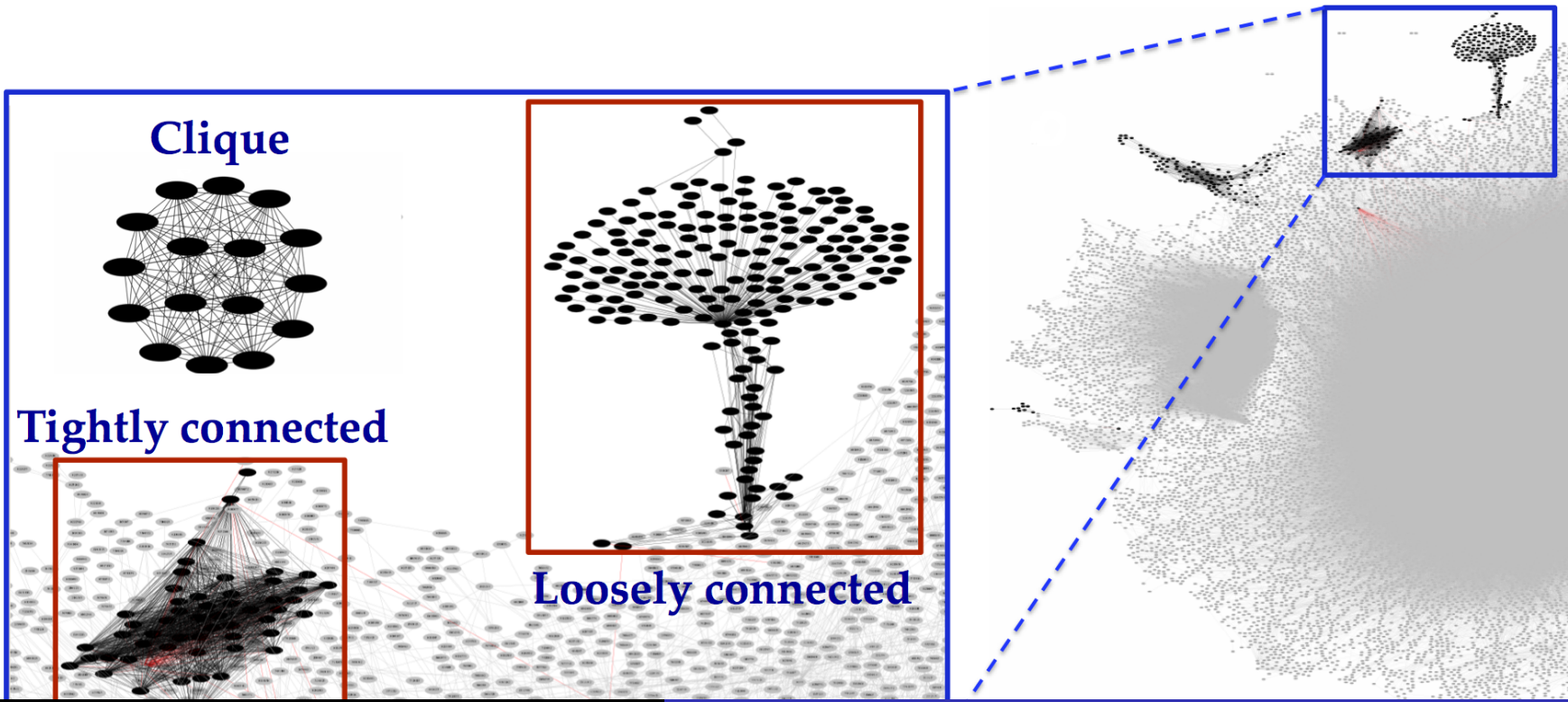
- Select a random node in the graph uniformly (Metropolis-Hastings method).
- Same as for biased except that, from current node i:
  - generate $p \sim U(0,1)$
  - selects uniformly a neighbor j; jump on it if $p \leq \min\{1, \frac{d_i}{d_j}\}$, else stay on i
- Results in probability of node j to be selected: $P_j = \frac{1}{\Sigma_{i=0}^{n} d_i}$

Random walks - app2: counting

- Distributed computation of n; based on the *birthday paradox* [6].
  - Sample uniformly nodes: $X_{t+1} \leftarrow X_t \cup j$
  - Stop when "collision" after l samples, i.e., when a node j appears twice in X_t
  - $\hat{n} = \sqrt{l^2/2}$

Random walks - app3: sybil detection

- "Early-terminated random walk starting from a non-Sybil node in a social network has a higher degree-normalized (divided by the degree) landing probability to land at a non-Sybil node than a Sybil node". [7]
  - observation holds because the limited number of attack edges forms a narrow passage from the non-Sybil region to the Sybil region in a social network.

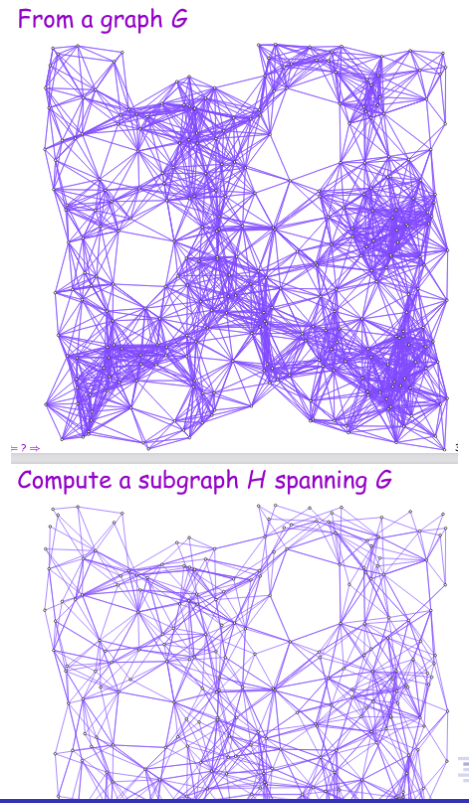


Spanner

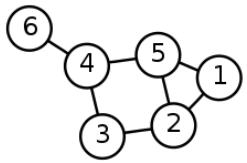
- A spanner H of a graph G: subgraph of G with *few* edges and *short* distances.[2] Tradeoff between number of edges and distance stretch.
- (α, β)-spanner of $G \iff \forall (u, v)$: $dist_H((u,v)) \leq \alpha \times dist_G((u,v)) + \beta$, with α: multiplicative stretch, β: additive stretch.

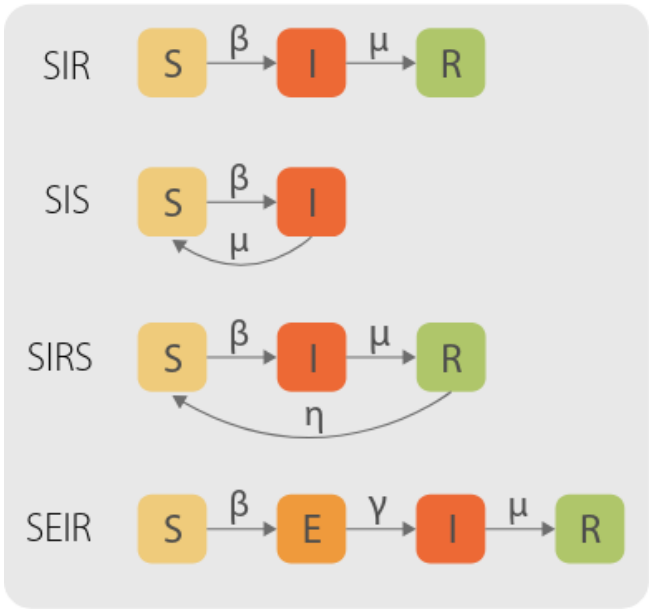
```
H := {}
For each edge (u,v) in E do
  If dist_H((u,v)) > 2k-1 do
    add (u,v) to H
```

- S is a $(2k-1, 0)$-spanner of G.
- $|V_H| < n_G^{1+1/k}$.

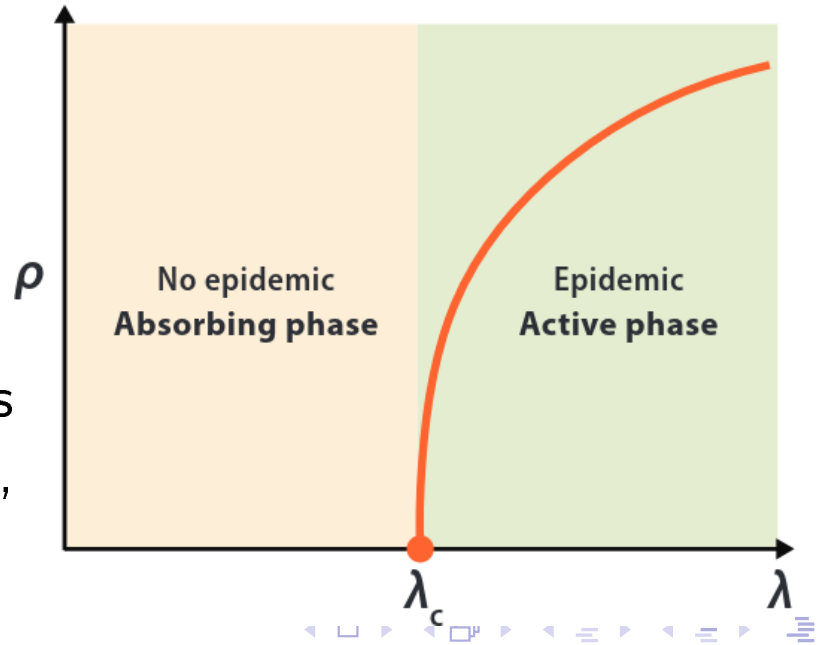


Epidemics on graphs (1)





- SIS-like: $\lambda = \beta/\mu$
- Order parameter ρ: transition point λ_c, s.t. for $\lambda > \lambda_c \rightarrow \rho > 0$, while for $\lambda \leq \rightarrow \rho = 0$



Epidemic on graph nodes: models with states Susceptible, Infected, Recovered, Exposed.[9]

Epidemics on graphs (2)

- Application to marketing: which initial node for best spread?

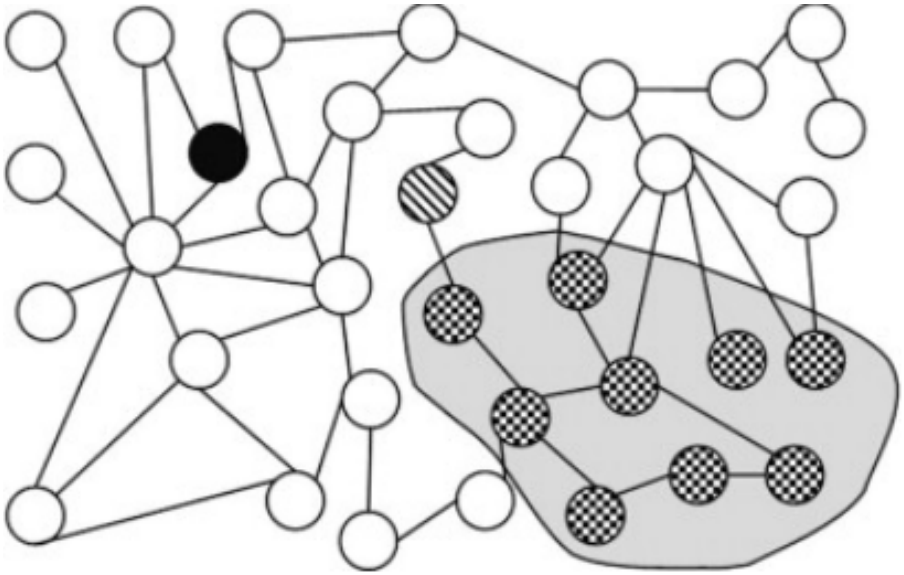


Fig. 1. Example of a social network. Black node denotes the globally central node; chequered nodes denote the potential market; hatched node denotes the node central w.r.t. the potential market.

Figure: cf "A targeted approach to viral marketing"