Mining Complex Networks

2023 CMS Summer Meeting

François Théberge
theberge@ieee.org

Tutte Institute for Mathematics and Computing
uOttawa adjunct professor (Mathematics and Statistics)

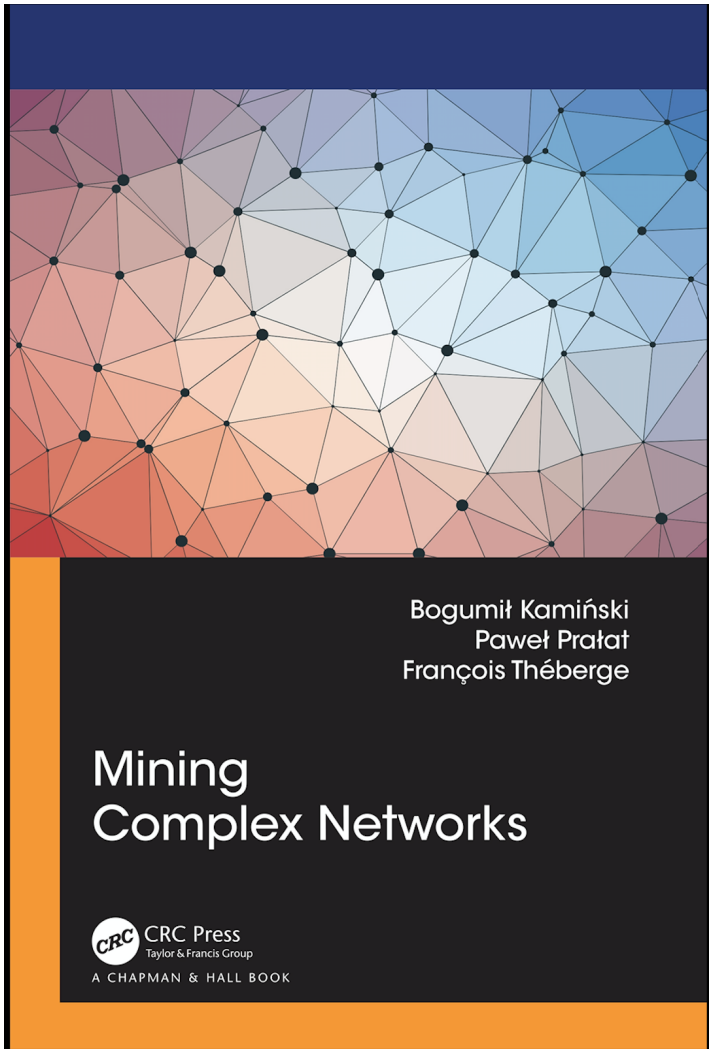
June 2023

Roadmap

1. Part I
   - **Background material**
   - Relational data and graphs
   - Random graph models
2. Part II
   - Measures of centrality
   - Degree correlation
   - Graph (vertex) embedding
   - *First notebook*
3. Part III
   - Community detection
   - *Second notebook*
   - Hypergraphs and other topics

Background material

Textbook (2022)



Background material

Textbook (2022)

- More emphasis on graph **mining** rather than modelling
- Between simple introductory text and comprehensive reference
- Written with one-term university course in mind
- Companion **notebooks** in Python and Julia
- 7 core chapters with theory, experiments and practical advice
- 4 extra chapters with examples of applications

Topics in this course correspond to the 7 core chapters.

References

Slides and introductory notebooks (Python):
github.com/ftheberge/CMS2023_MiniCourse

CRC textbook (2022):
www.torontomu.ca/mining-complex-networks

Companion notebooks (1 per chapter, Python and Julia):
github.com/ftheberge/GraphMiningNotebooks

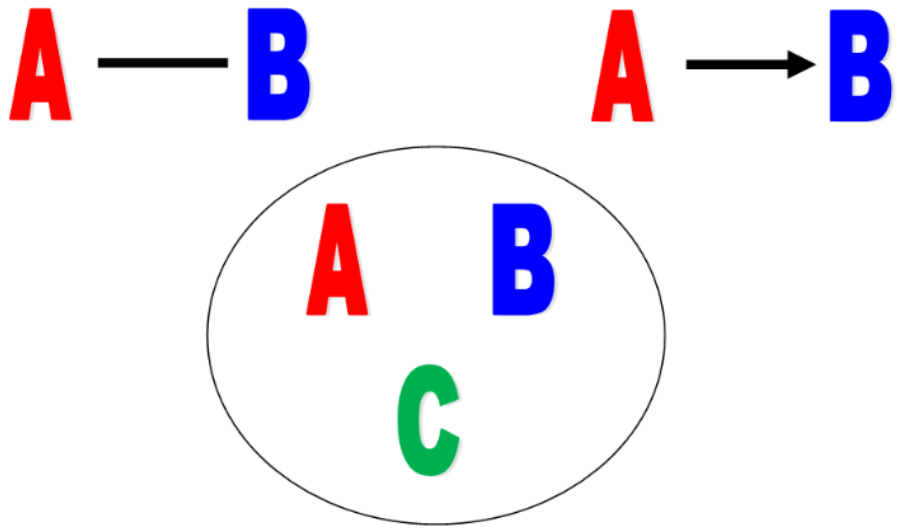
Graph package used:
python-igraph
(also available in R, C and Mathematica)

Roadmap

1. Part I
   - Background material
   - **Relational data and graphs**
   - Random graph models
2. Part II
   - Measures of centrality
   - Degree correlation
   - Graph (vertex) embedding
   - *First notebook*
3. Part III
   - Community detection
   - *Second notebook*
   - Hypergraphs and other topics

Relational Data

- examples of relations between entities:
  - *A* and *B* are friends
  - *A* sends an email to *B*
  - *A*, *B* and *C* are in the same team
- the above are modelled as edges or hyperedges:



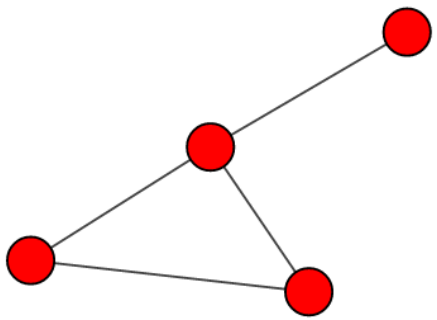
Graphs

For a graph $G = (V, E)$, let $n = |V|$ and $m = |E|$

We map the vertices to integers indices $v_1 \ldots v_n$ for convenience

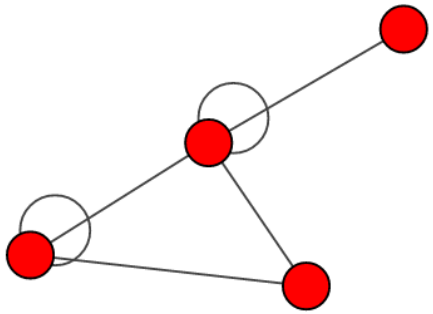
Let $A = (a_{ij})$, the adjacency matrix s.t. $a_{ij} > 0 \iff (v_i, v_j) \in E$

Undirected (unweighted) graph: $a_{ij} = a_{ji} \in \{0, 1\}, a_{ii} = 0$.

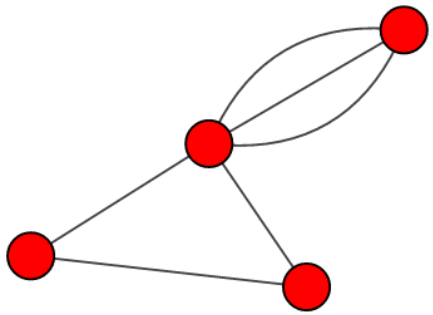


Graphs

Undirected graph with self-loops: some $a_{ii} = 1$.

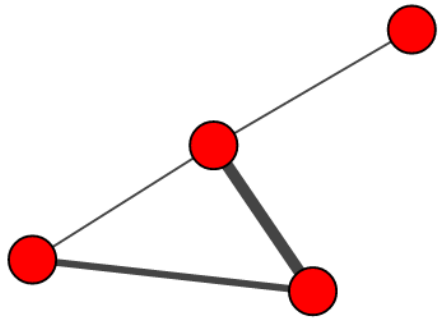


Multigraph: $a_{ij} \in \mathbb{N}$

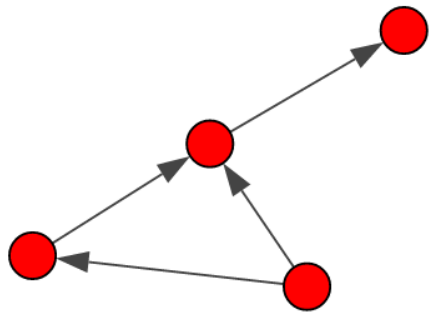


Graphs

Weighted graph: $a_{ij} \geq 0$



Directed graph: can have $a_{ij} \neq a_{ji}$



Graphs

$w = (v_{i_0}, v_{i_1}, ..., v_{i_k})$ is a **walk** between v_{i_0} and v_{i_k} if all $(v_{i_l}, v_{i_{l+1}}) \in E$. A **path** is a walk without repeated nodes.

For unweighted graphs, its **length** is the number of edges. For weighted graphs, the sum of edge weights.

A **connected component** for an undirected graph is a maximal subgraph for which any 2 nodes are connected by a path.

For directed graphs, we distinguish **strong** connected components (connections via directed paths) and **weak** ones.

The **distance** between two nodes (v_i, v_j) is the minimum path length between them.

The **diameter** of a (connected) graph is the maximum distance between two nodes.

Graphs

The **degree** of node v in the number of edges incident to it.

For directed graphs, we usually distinguish **in-degree** and **out-degree**.

The **degree distribution** describes the distribution of node degrees for a given graphs via statistics such as: minimum and maximum degree (δ, Δ), mean degree, median degree, etc.

Graphs

Many networks, in particular social networks, exhibit **homophily**:

Friends share common friends with higher than random probability.

We can measure this by looking at **triangles** in a graph

Consider an undirected graphs $G = (V, E)$:

A *triad* is a subgraph of 3 nodes forming a tree; let $n_\wedge$ the number of triads in a graph G;

A *triangle* is a fully-connected subgraph with 3 nodes; let n_Δ be the number of triangles in graph G;

The **global clustering coefficient** or *graph transitivity* of G is defined as

$$C_{glob} = \frac{3n_{\Delta}}{n_{\wedge}}.$$

For a given node $v_i \in V$ of degree d_i, let $n_i(e)$ be the number of edges between neighbours of v_i;

Node v_i's **local clustering coefficient** is defined as:

$$c_i = \frac{n_i(e)}{\binom{d_i}{2}}.$$

Clustering

The **(average local) clustering coefficient** for graph G is:

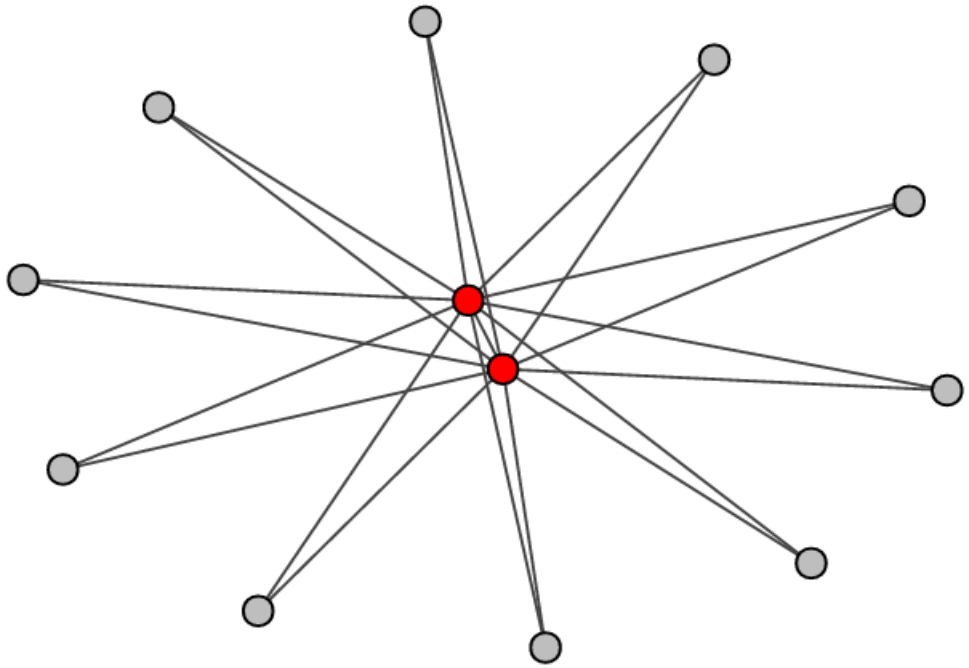
$$C_{loc} = \frac{1}{n} \sum_i c_i$$

the average over all c_i's.

Quantities C_{glob} and C_{loc} are often similar, but this can be misleading;

Clustering

Consider the following graph with $n + 2$ nodes (n grey nodes):



Clustering

In this graph, $n_\Delta = n$ and $n_\wedge = 3n + n(n-1) = n^2 + 2n$, so

$$C_{glob} = \frac{3}{n+2} \to 0$$

with the limit as $n \to \infty$.

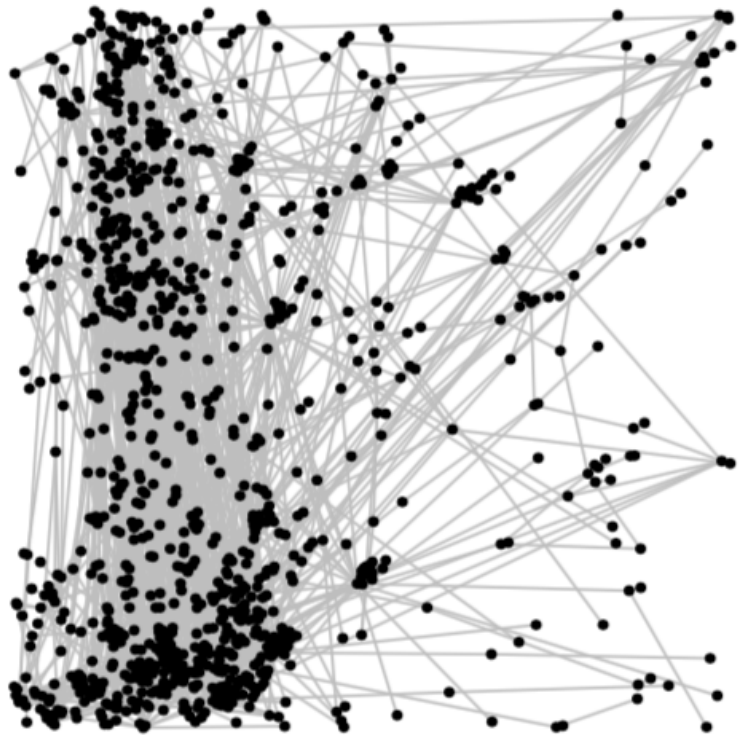
For the 2 red nodes, we get $c_i = 2/(n+1)$ while for the grey nodes, $c_i = 1$. The (average local) clustering coefficient is therefore:

$$C_{loc} = \frac{n^2 + n + 4}{n^2 + 3n + 2} \to 1.$$

Datasets

GitHub developers

Nodes: web and ml developers (37.7k)
Edges: starring common repositories (289k)



(b) GitHub ml subgraph

Datasets

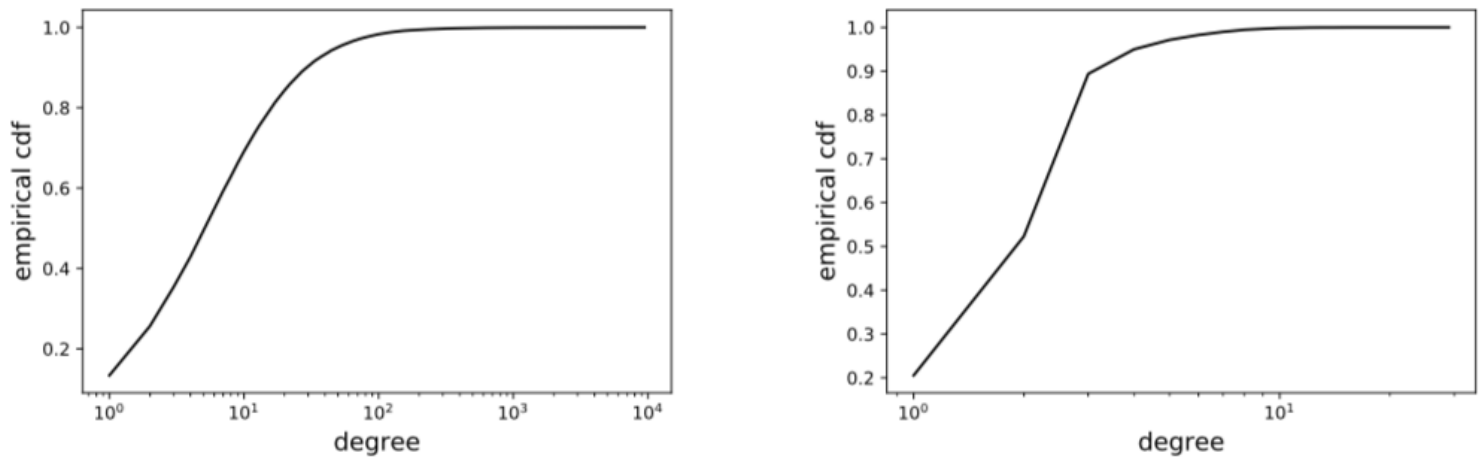
Europe electric grid

Nodes: high voltage stations (13.8k)
Edges: physical lines (17.2k)

(c) European Grid network

Comparing Github and Grid graphs

The first difference is in the **degree distribution**
GitHub graph has many low degree nodes and some very large degree nodes, typical of social graphs



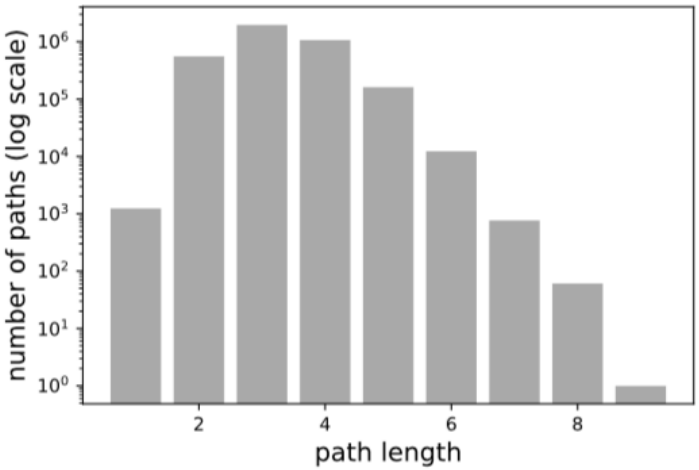
(a) GitHub graph

(b) European Grid network

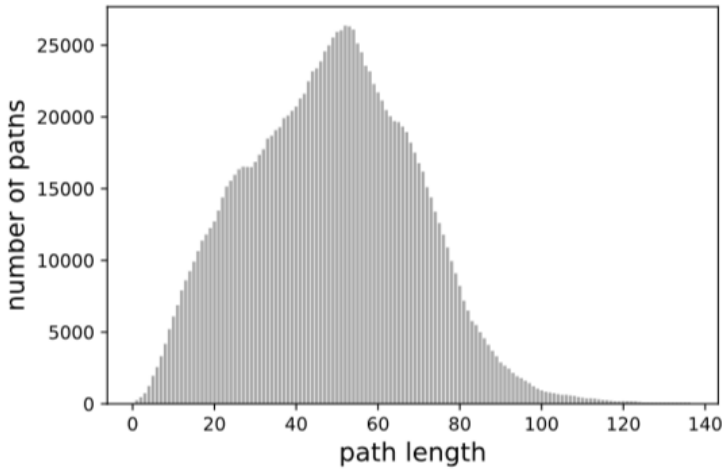
FIGURE 1.3
Empirical cumulative degree distribution (CDF) for the two graphs under study.

Comparing Github and Grid graphs

Shortest path lengths (distances between nodes) are also very different.



(a) GitHub graph



(b) European Grid network

GitHub graph exhibits the **small-world** phenomenon.

Comparing Github and Grid graphs

graph	GitHub	GitHub (ml)	GitHub (web)	Grid
# nodes	37,700	9,739	27,961	13,844
# edges	289,003	19,684	224,623	17,277
δ	1	0	0	1
$\langle k \rangle$	15.332	4.042	16.067	2.496
median degree	6	2	6	2
$d_{quant_{99}}$	138	39	145	8
Δ	9,458	482	8,194	16
diameter	11	13	9	147
# components	1	2,466	297	59
the largest component	37,700	7,083	27,653	13,478
# isolates	0	2,308	285	0
C_{glob}	0.012	0.034	0.014	0.100
C_{loc}	0.193	0.141	0.207	0.113

TABLE 1.1
Basic descriptive statistics for the GitHub and the Grid graphs. The GitHub subgraphs built with the two types of developers (ml and web) are also included. $d_{quant_{99}}$ refers to the 99^{th} quantile for the degree distribution.

Power law

In a **power law** function, one quantity varies as a power of the other.

This is often observed in real, social-type graphs, in particular for the **degree distribution**.

Let p_k the proportion of nodes with degree k, then

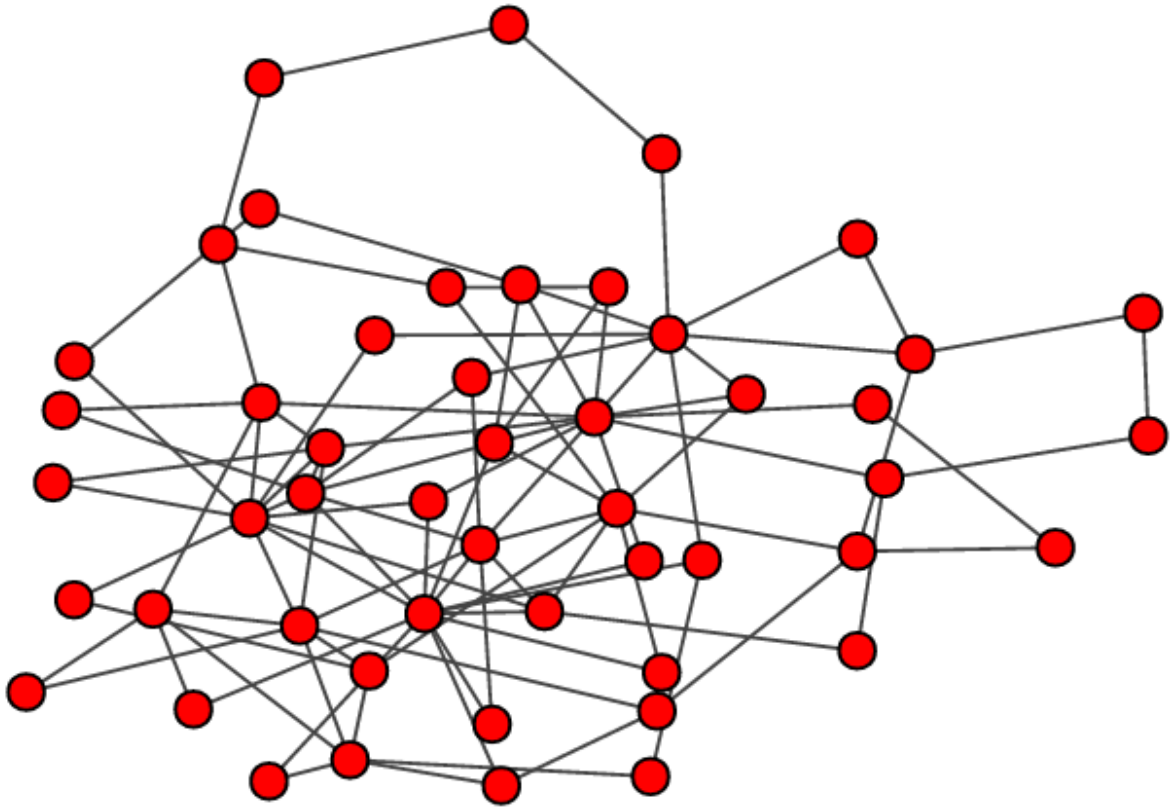
$$p_k = ck^{-\gamma}$$

is an example of a power-law distribution, where c is the normalizing constant.

For degree distribution in social networks, the values typically observed are $2 \leq \gamma \leq 3$.

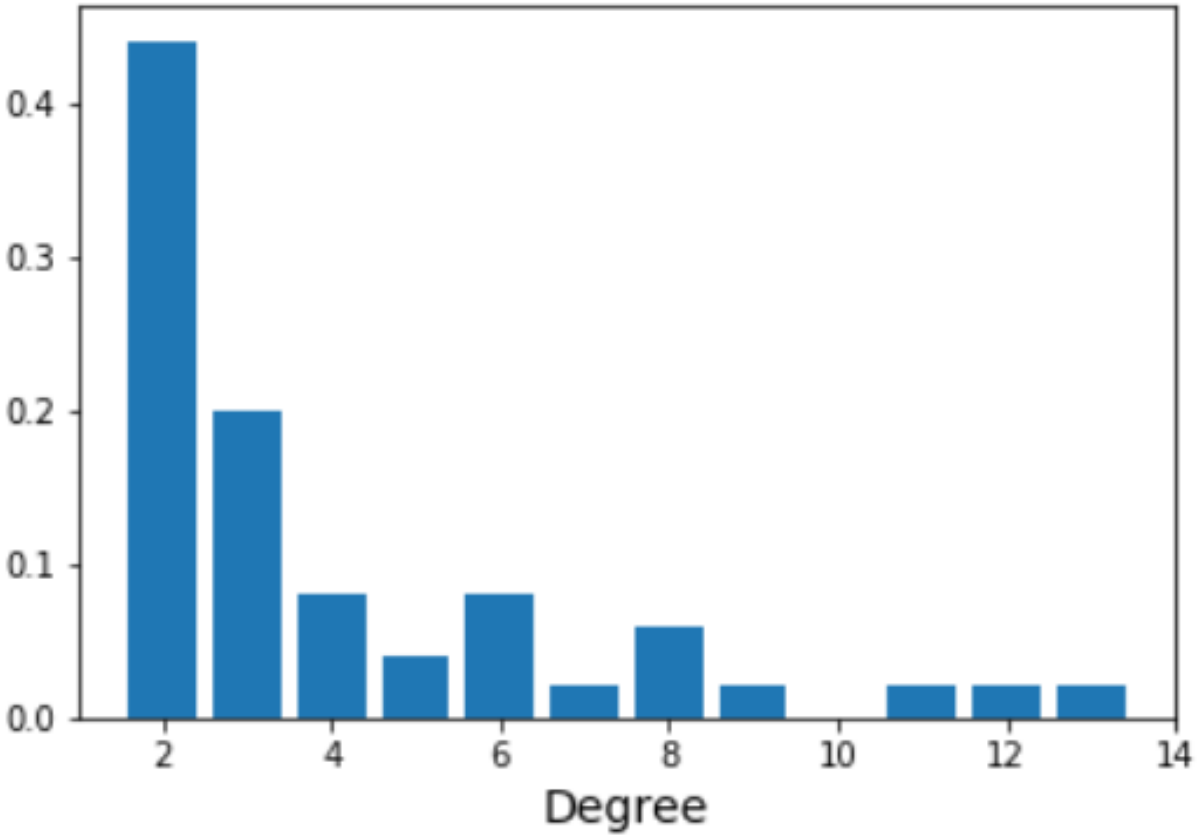
Power law

Here is an example of a graph with power-law degree distribution where $\gamma = 2$:



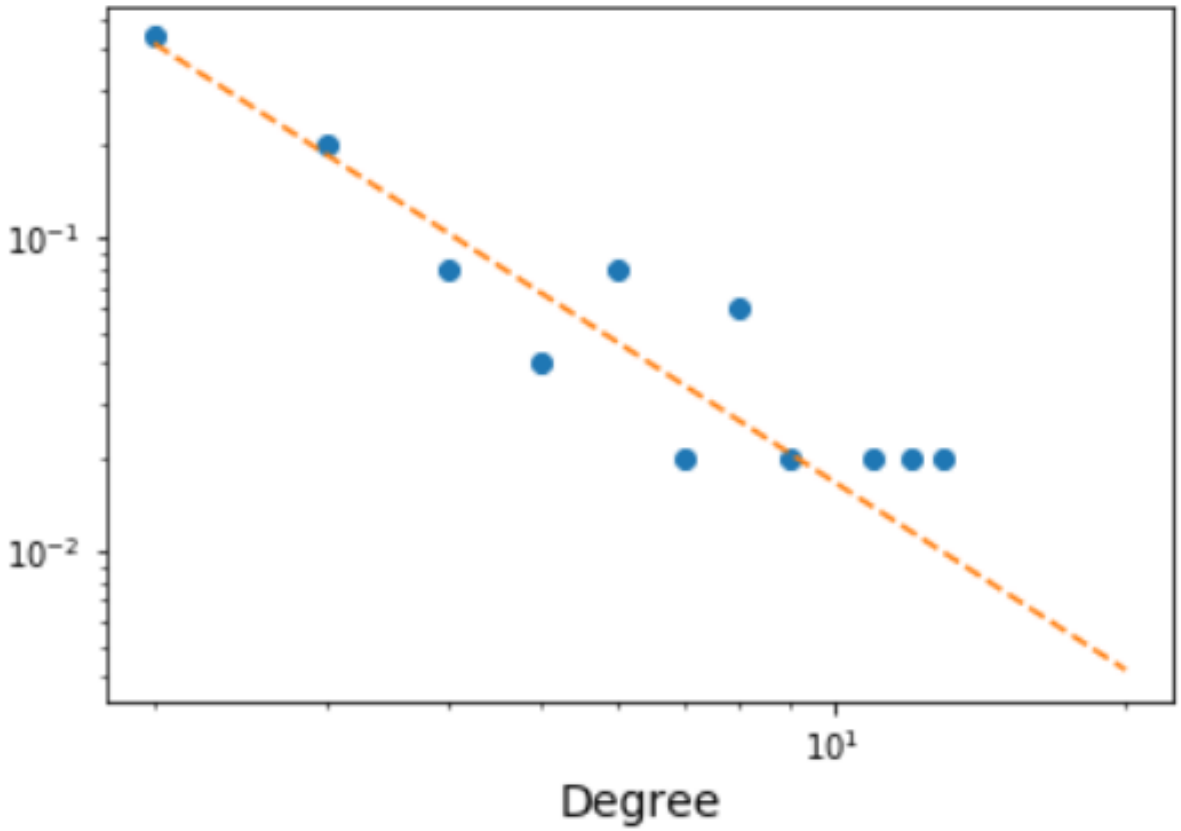
Power law

and it's degree distribution:



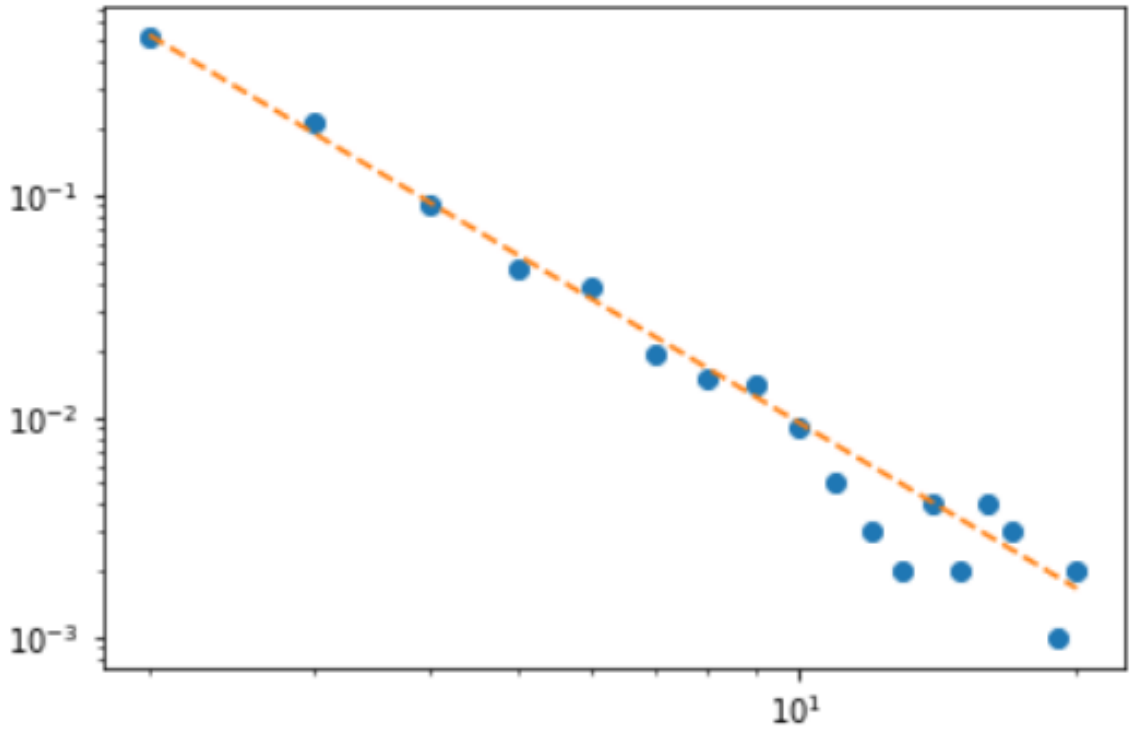
Power law

On a log-log plot, we see a roughly linear relation (with log binning of degrees):



Power law

Here is another log-log plot for a much larger graph with $\gamma = 2.5$:



Power law

Therefore, power-law graphs exhibit a degree distribution with:

- a fairly large number of small degree nodes, and
- a *long tail* which amounts to the presence of high degree nodes.

Such high degree nodes are called hubs (hubs and authorities for directed graphs).

Summary

We saw several simple descriptive statistics to characterize graphs such as:

- degree distribution
- diameter
- clustering coefficients

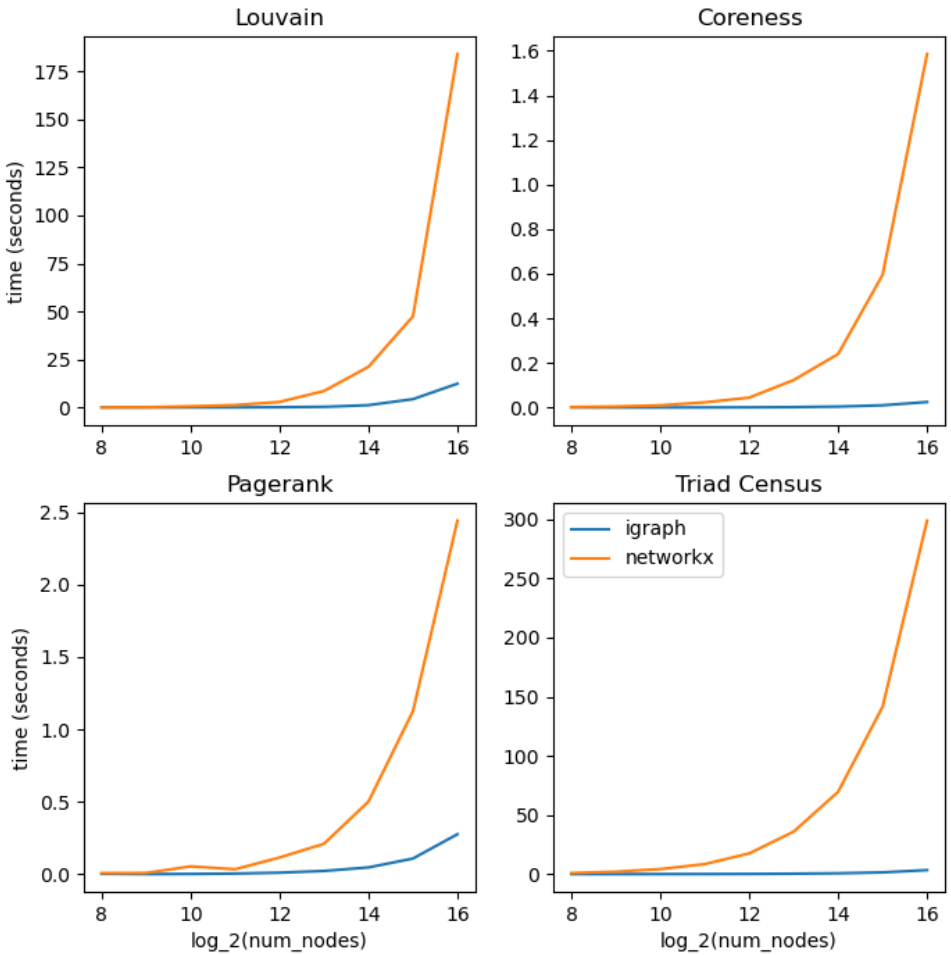
In practice, **EDA** (exploratory data analysis) is an important part of a data mining task.

When possible, **visualization** is also useful.

There are several good tools for handling graphs. For Python users, we recommend **python-igraph**, which is used in our notebooks.

Summary

igraph shows good scalability



Roadmap

1. Part I
   - Background material
   - Relational data and graphs
   - **Random graph models**
2. Part II
   - Measures of centrality
   - Degree correlation
   - Graph (vertex) embedding
   - *First notebook*
3. Part III
   - Community detection
   - *Second notebook*
   - Hypergraphs and other topics

Random Graph Models

The theory of random graphs is at the intersection of graph theory and probability.

It is an active area of research:

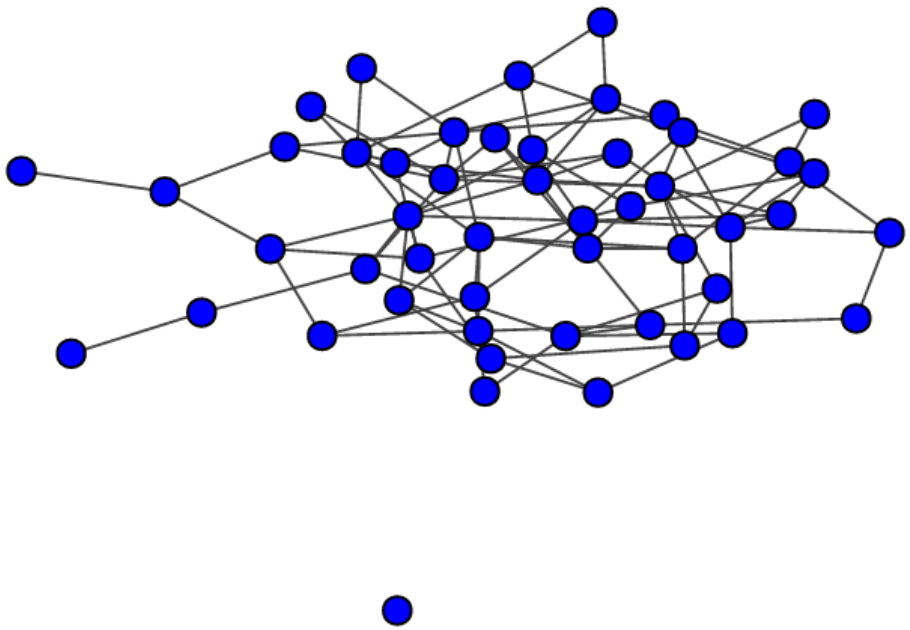
- uncovering properties of typical graphs
- find “surprising” objects
- help understand network formation (eg. preferential attachment models explain power-law degree distribution)
- for data science:
  - create synthetic yet realistic graphs to test various scenarios
  - benchmark algorithms

We review a few commonly used models.

Erdős-Rényi Models

The $G(n, m)$ model:

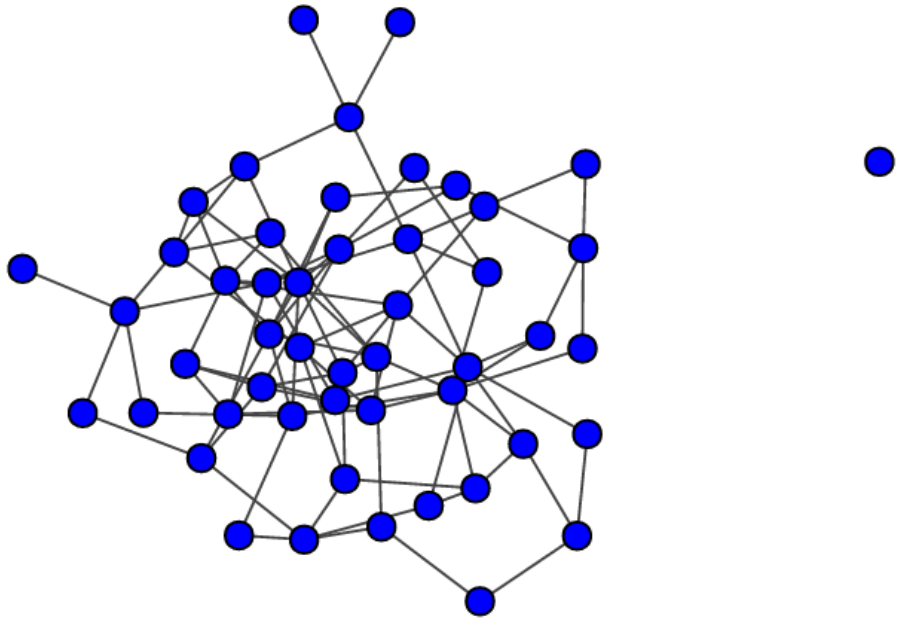
- n nodes
- m edges chosen at random from $N = \binom{n}{2}$ pairs
- average degree $k = 2m/n$



Erdős-Rényi Models

The $G(n, p)$ model:

- n nodes
- each possible $N = \binom{n}{2}$ node pair is connected with probability p
- expected number of edges is Np
- expected average degree $k = p(n-1)$
- allows for easy calculation of graph statistics



Erdős-Rényi Models

Number of edges:

Let P_m the probability of getting m edges with the $G(n, p)$ model, and let $N = \binom{n}{2}$.

$$P_m = \binom{N}{m} p^m (1-p)^{N-m}$$

Given that N is large and p is (typically) small, we can use the Poisson approximation to the binomial with $\lambda = Np$:

$$P_m \approx \frac{e^{-\lambda} \lambda^m}{m!}$$

Erdős-Rényi Models

Degree distribution:

Let p_k, the probability that some node has degree k in $G(n, p)$.

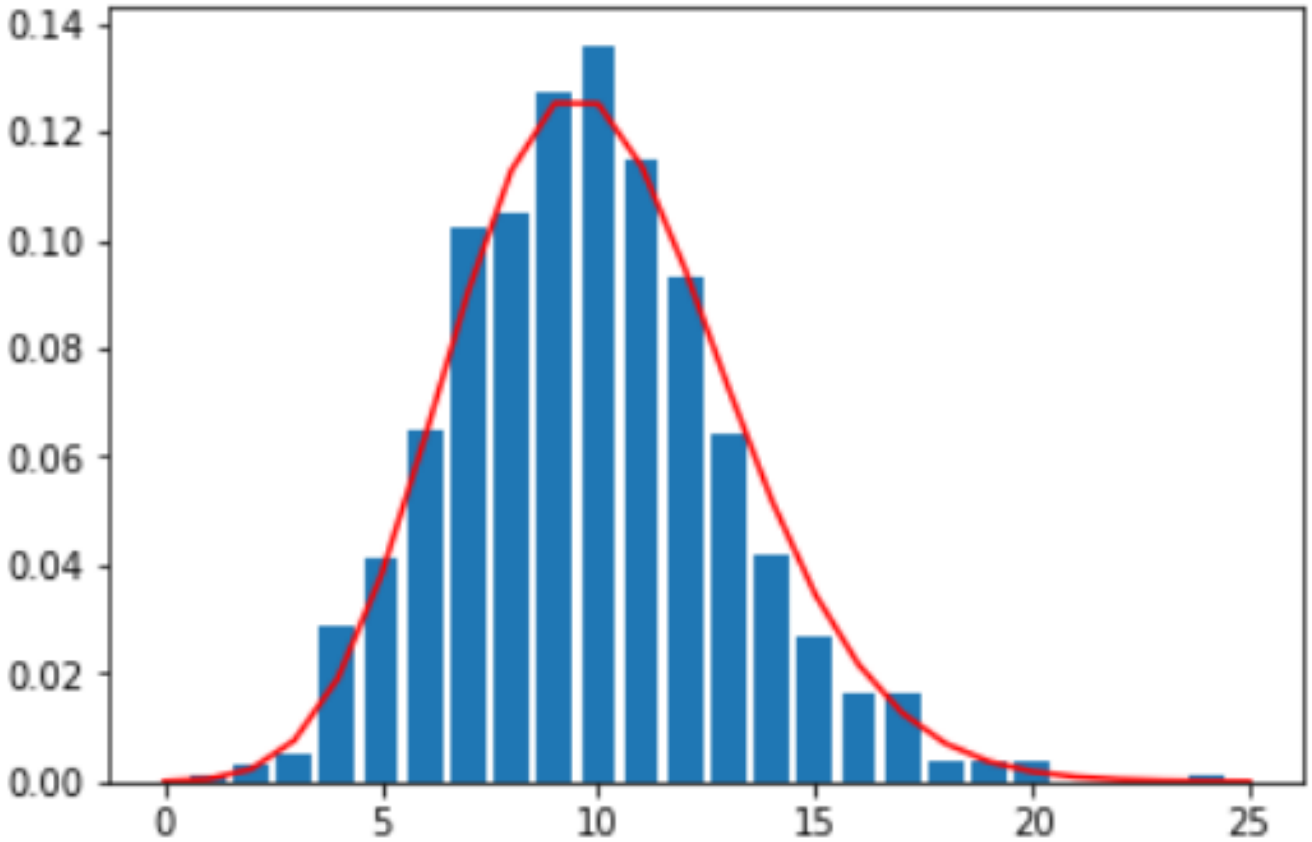
$$p_k = \binom{n-1}{k} p^k (1-p)^{n-1-k} \approx \frac{e^{-\lambda}\lambda^k}{k!}$$

with $\lambda = (n-1)p$, the expected average degree.

In view of the Poisson distribution, such models do not generate *hubs*, the high-degree nodes typically seen in social-type networks.

Erdős-Rényi Models

Degree distribution ($n = 1000,\ p = .01$):



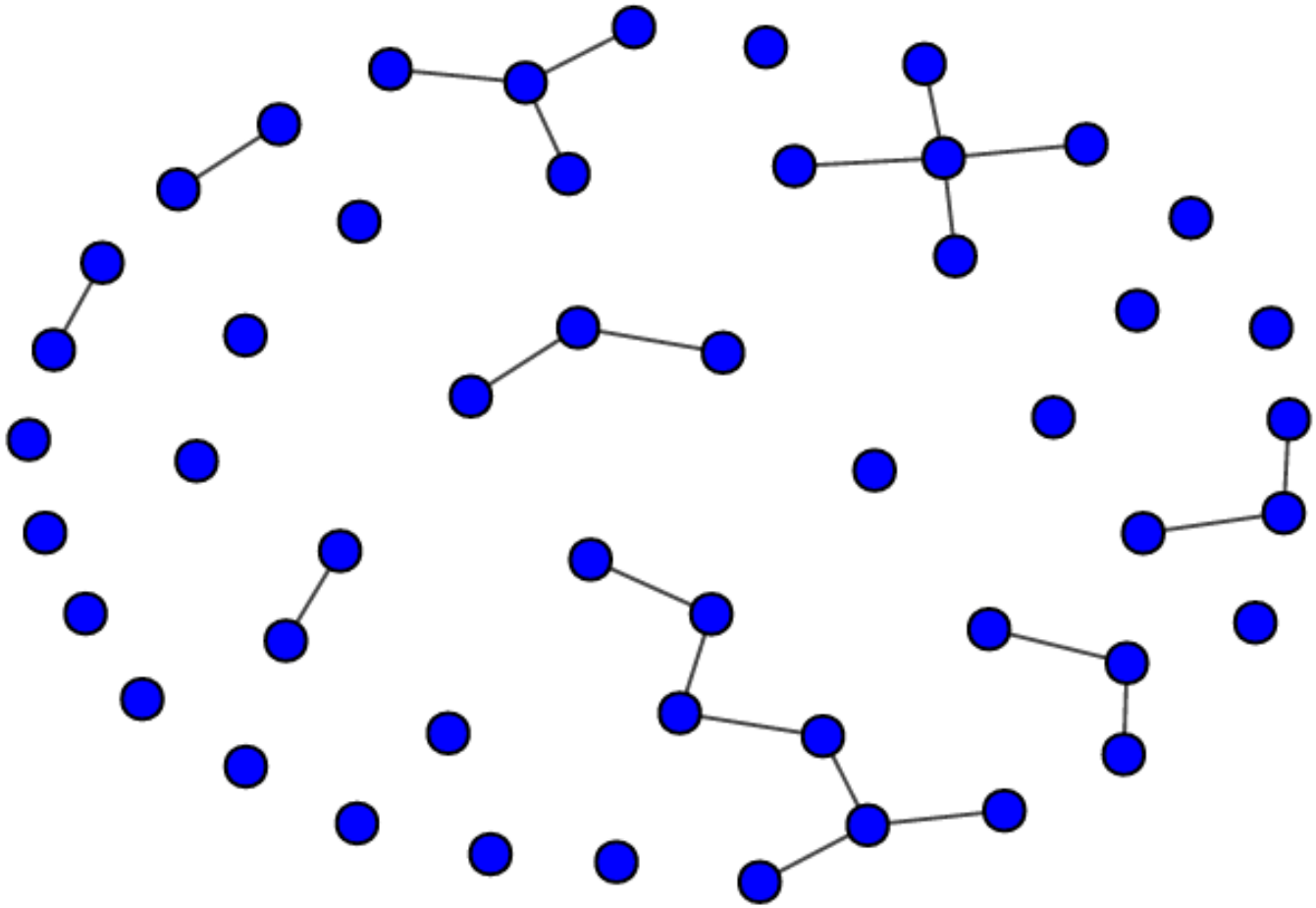
Giant connected component

For $G(n, p)$ with expected average degree $k = p(n-1)$, we distinguish 3 regimes:

1. $k < 1$: subcritical regime; no giant component, connected components are mostly trees.
2. $k > 1$: supercritical regime; single giant component, small components are mostly trees.
3. $k >> \log(n)$: connected regime; no isolated nodes or small components.

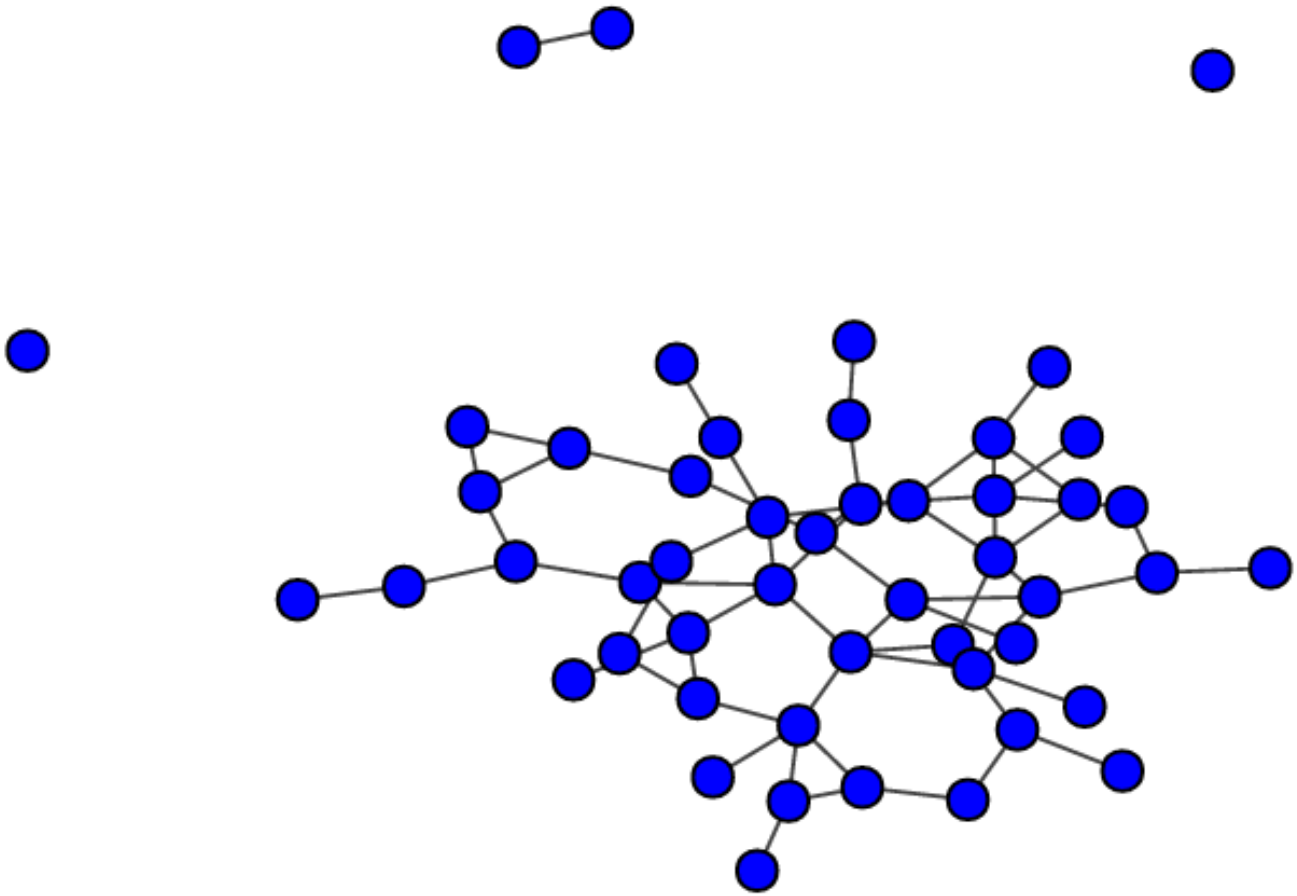
Erdős-Rényi Models

Subcritical regime:



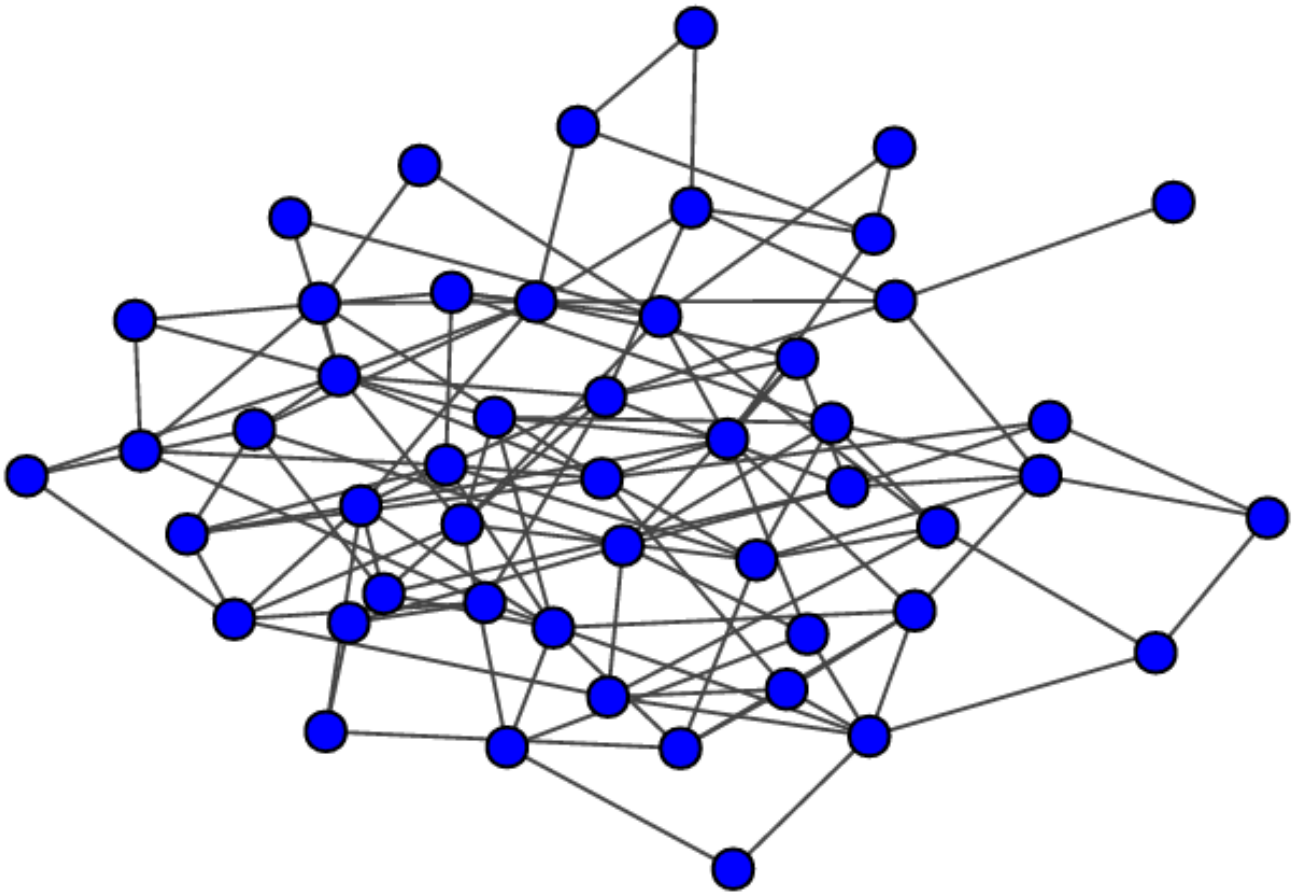
Erdős-Rényi Models

Supercritical regime:



Erdős-Rényi Models

Connected regime:



Chung-Lu Models

Degree distribution in Erdős-Rényi Models is likely non-realistic

Let G be a graph with vertices $V = \{v_1, \ldots, v_n\}$.

With Chung-Lu models, we also consider a degree sequence: $d_i = deg_G(v_i)$, with $vol(V) = \sum_i d_i = 2|E|$.

Degree distribution could be **power-law**.

Model I: probability of an edge between vertices v_i and v_j is:

$$p_{ij} = \frac{d_i d_j}{vol(V)}, \ i \neq j \text{ and } p_{ii} = \frac{d_i^2}{2vol(V)}.$$

This is also known as the **Bernoulli Chung-Lu** model.

Chung-Lu Models

If we define $\mathcal{CL}_1(G)$ to be the distribution of graphs obtained with Model I, then for $G' = (V, E') \sim \mathcal{CL}_1(G)$:

- $\mathbb{E}_{G' \sim \mathcal{CL}_1(G)}(deg_{G'}(v_i)) = d_i, 1 \leq i \leq n$,
- $\mathbb{E}(|E'|) = |E|$
- there are no multi-edges, and
- there can be self-edges.

Generating graphs with Model I is not scalable, as it requires $O(n^2)$ Bernoulli experiments.

Chung-Lu Models

Model II is generated from a scalable algorithm with $O(|E|)$ steps only.

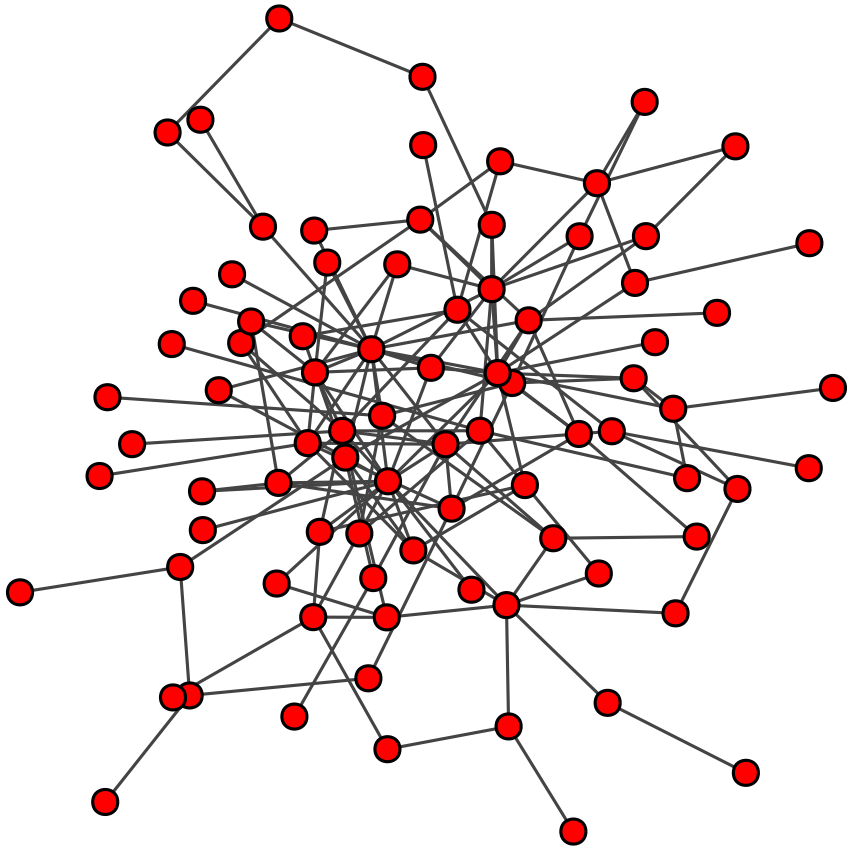
Generate a graph over vertices V by selecting $|E|$ edges $e = (u_1, u_2)$ where each u_i is independently sampled from V according to the multinomial distribution where $p(v_i) = d_i / vol(V)$.

If we define $\mathcal{CL}_2(G)$ to be the distribution of graphs obtained with Model II, then for $G' = (V, E') \sim \mathcal{CL}_2(G)$:

- $\mathbb{E}_{G' \sim \mathcal{CL}_2(G)}(deg_{G'}(v_i)) = d_i, 1 \leq i \leq n$,
- there can be multi-edges, and
- there can be self-edges.

Chung-Lu Models

Chung-Lu graph with power-law degree distribution:



Configuration Model

The Chung-Lu models are probabilistic models for edge generation, with **expected** degree sequence.

With the **configuration model**, we specify an exact degree sequence for the nodes: $d_1, \ldots, d_n$.

The sequence needs to be **graphic**.

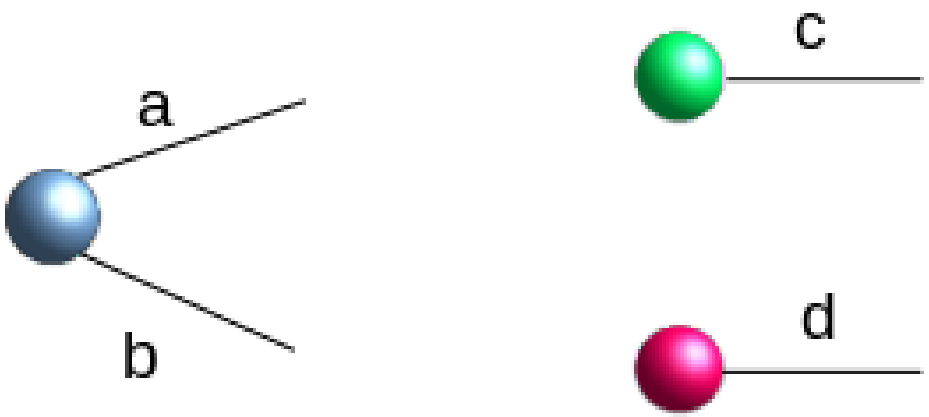
Each graph with n nodes and this exact degree sequence is assigned the same probability.

This is fast to generate ... but graphs may have loops and multi-edges.

Configuration Model

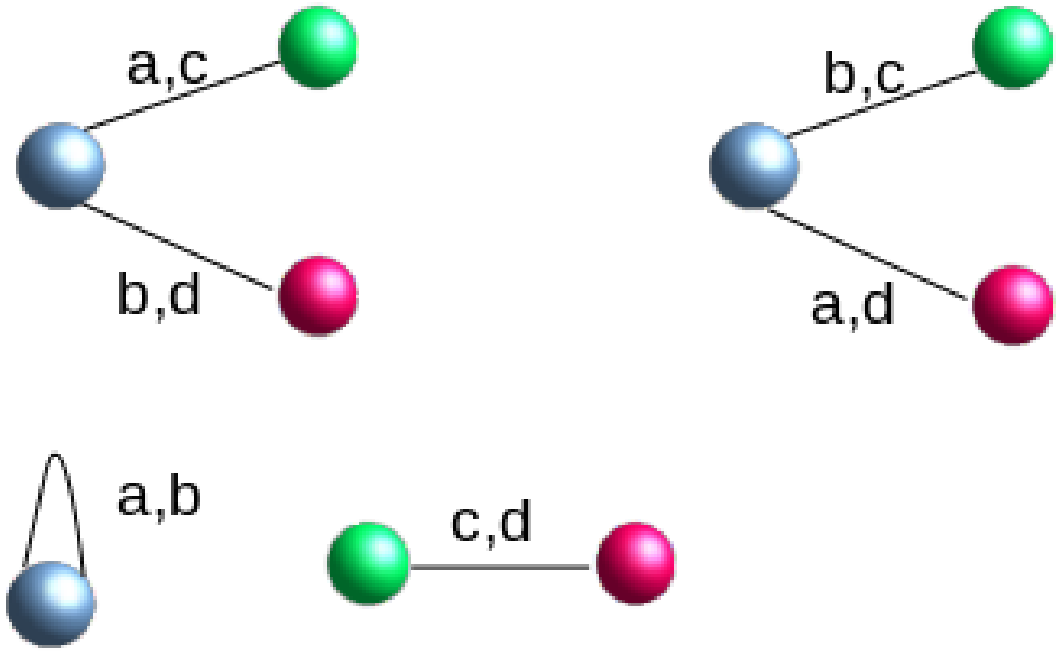
For each node i, we assign d_i stubs (half-edges);

Stubs are connected at random.



Configuration Model

This can generate self-edges and multi-edges



Configuration Model

For this model, the expected number of edges between nodes $i \neq j$ is:

$$e_{ij} = \frac{d_i d_j}{2m - 1}$$

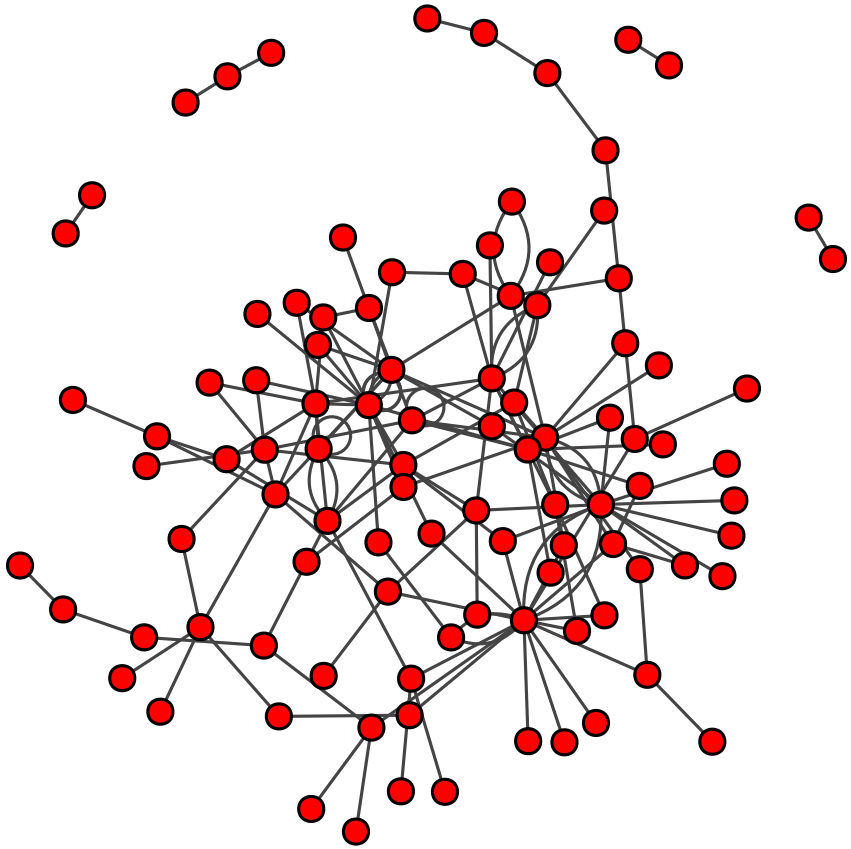
and:

$$e_{ii} = \frac{d_i(d_i - 1)}{2(2m - 1)}$$

Several algorithms to generate such graphs are available in `igraph`.

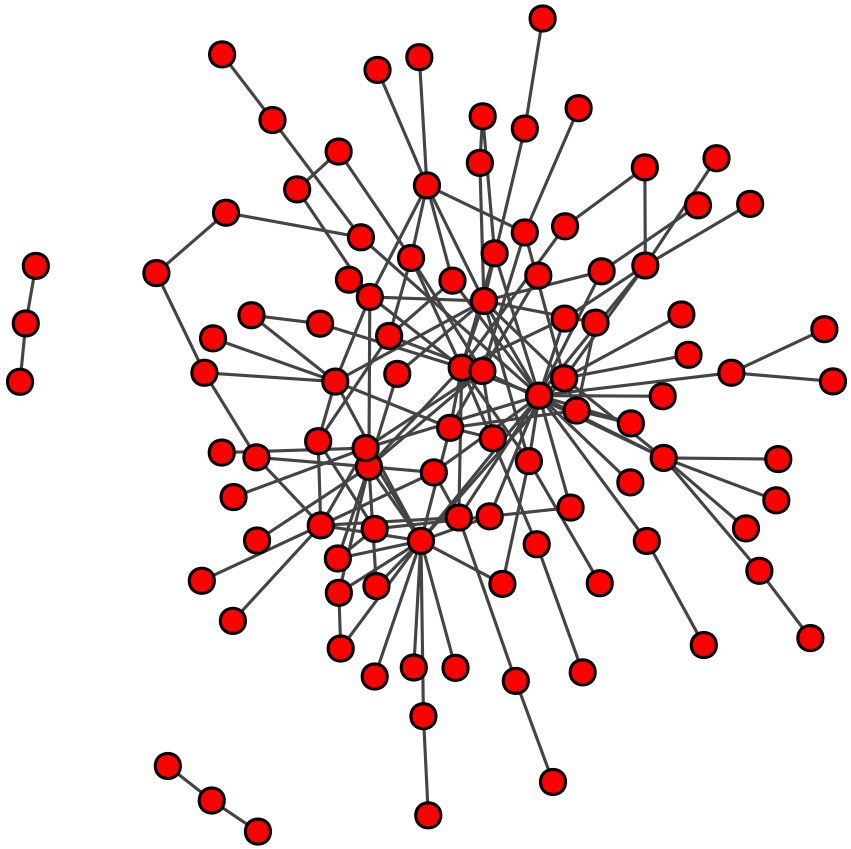
Configuration Model

Simple method (can generate loops, multiedges):



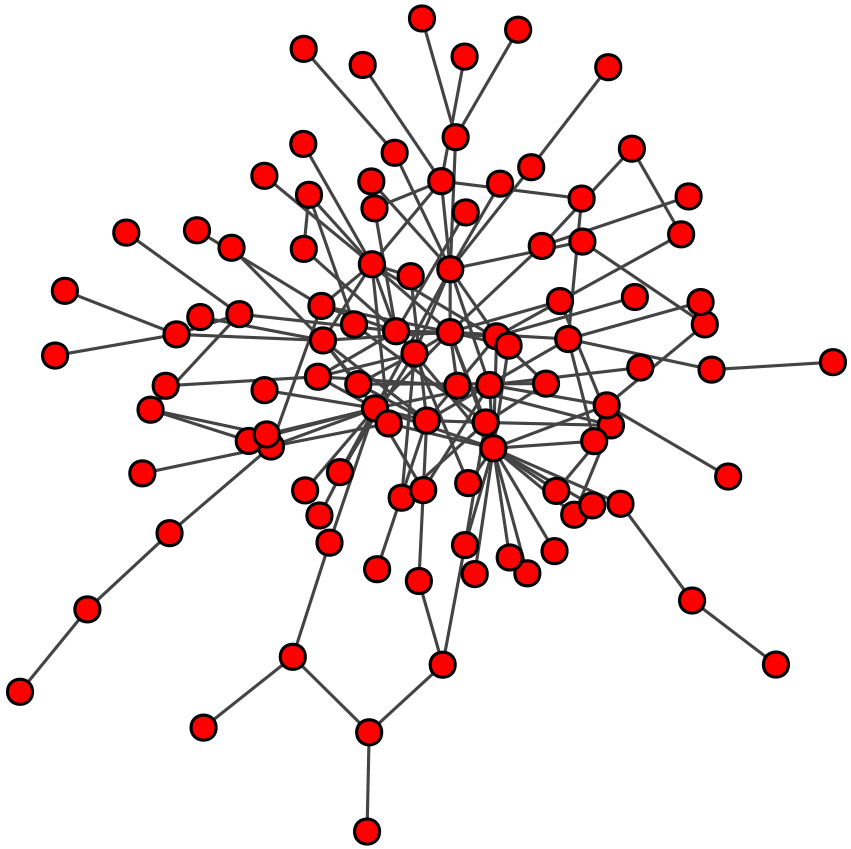
Configuration Model

Avoiding loops and multiedges (restart if stuck):



Configuration Model

Viger's algorithm (returns a connected graph):



Example - GitHub Subgraph

Modelling the GitHub graph:

Graph	Base	Binomial	Chung-Lu	Config.	Config.(V)
nodes	7,083	7,083	7,083	7,083	7,083
edges	19,491	19,491	19,491	19,491	19,491
$\delta = d_{min}$	1	0	0	1	1
d_{mean}	5.504	5.504	5.504	5.504	5.504
d_{median}	2	5	3	2	2
$\Delta = d_{max}$	482	18	406	482	482
diameter	13	10	11	10	11
components	1	25	1,063	70	1
largest	7,083	7,058	5,992	6,940	7,083
isolates	0	23	1,035	0	0
C_{glob}	0.0338	0.0007	0.0198	0.0185	0.0171
C_{loc}	0.1412	0.0006	0.0258	0.0242	0.0319

TABLE 2.8
Comparison of a few descriptive statistics for the base graph (a subgraph of the GitHub graph) with 4 random graph models.

Small world

Real graph do not typically look like random graphs.

Some observed characteristics include:

- relatively short paths between nodes (small diameter)
- local density behaviour (triangles, communities)
- large number of low degree nodes
- presence of high degree nodes (hubs, authorities)
- power law degree distribution

Barabasi-Albert

This is an example of a **preferential attachment** model, also referred to as *the rich gets richer*

A graph is generated node by node:

- start from some initial graph (ex: empty graph, small clique) at step $t = 0$
- at step $t > 0$, add a new node with (up to) m edges to existing nodes
- the probability that the new node has an edge with (existing) node v_i is proportional to $d_i(t)$, the degree of node v_i before adding this new node.
- continue until n nodes are generated

Barabasi-Albert

The model favours attaching to high degree nodes, thus forming hubs;

Asympotically, the proportion of degree k nodes is:

$$p_k \approx 2m(m+1)k^{-3}$$

There are several variations on that theme.

Roadmap

1. Part I
   - Background material
   - Relational data and graphs
   - Random graph models
2. Part II
   - **Measures of centrality**
   - Degree correlation
   - Graph (vertex) embedding
   - *First notebook*
3. Part III
   - Community detection
   - *Second notebook*
   - Hypergraphs and other topics

Centrality Measures

There are two families of centrality measures $c(v), v \in V$:

- Matrix based
- Distance based

We look at a few examples from both families.

For all measures, we can normalize such that $c(v) \in [0, 1]$.

We consider **undirected** graphs, but most measures can be generalized to directed graphs.

Degree Centrality

Let $G = (V, E)$ with adjacency (or weight) matrix A, so $a_{ij} > 0 \iff (v_i, v_j) \in E$

For an unweighted graph, $a_{ij} \in \{0, 1\}$

Definition: The *degree centrality* of node $v_i \in V$ in an undirected graph is:

$$c^D(i) = d_i = \sum_{j \in V} a_{ij}$$

The *normalized degree centrality* for unweighted graphs is obtained by dividing the above quantities by $n - 1 = |V| - 1$ assuming no self-loops.

Eigenvector Centrality

We can define centrality as a combination of high degree, and being connected to highly central nodes:

$$c(i) = \sum_{j \in V} a_{ij} c(j)$$

Definition: The *eigenvector centrality* of node v_i is a solution of:

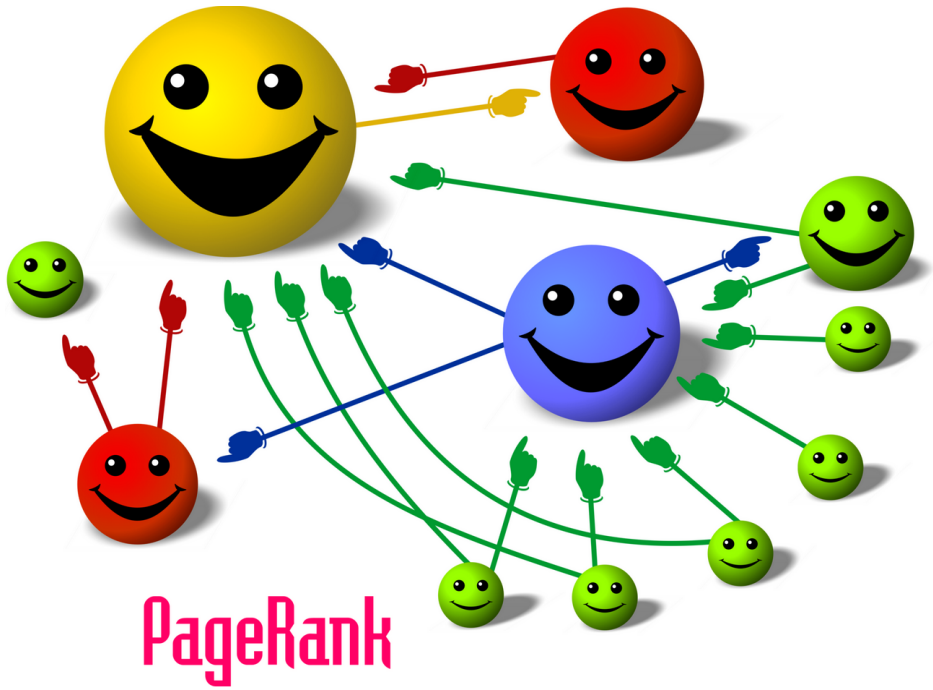
$$\lambda c^E(i) = \sum_{j \in V} a_{ij} c^E(j)$$

For a connected graph, eigenvector centrality c^E is the eigenvector corresponding to the leading eigenvalue of A, which is real and positive (Perron-Frobenius).

PageRank Centrality

Another measure of centrality in the PageRank algorithm used by Google

Idea: important pages are linked to by many important pages.



REF: Wikipedia

PageRank Centrality

Imagine a random walk where, given we are at node v_i:

- with probability $1 - d$: jump to a random node $v_j \in V$.
- with probability d: jump to node v_j with probability a_{ij}/d_i^{out}.

PageRank values are obtained by solving:

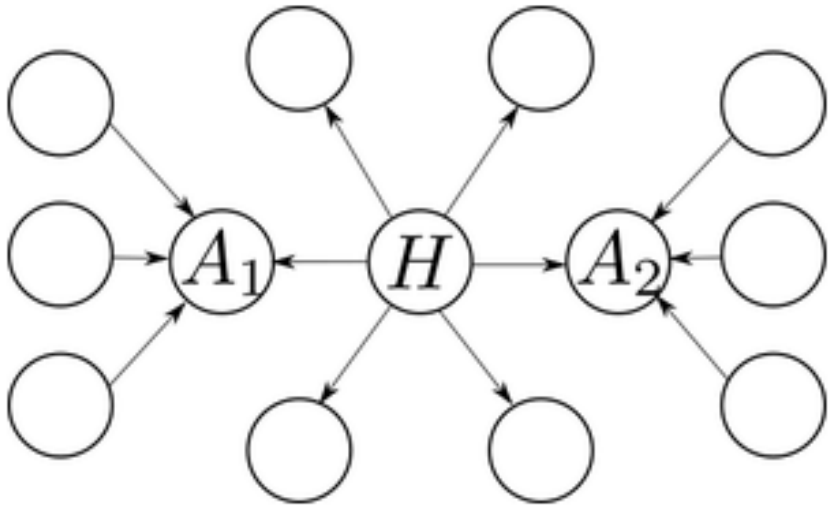
$$R(i) = \frac{1-d}{n} + d \sum_{j \to i} \frac{R(j)}{d_j^{out}}$$

Solution can be obtained with algebraic or iterative (power) methods.

Hubs and Authorities

In a directed graph, we define a *hub* as a node with high out degree, and an *authority* as a node with high in degree.

In an undirected graph, the two concepts are the same.



REF: blog.scrapinghub.com

Hubs and Authorities

We define hub centrality for node v_i as:

$$c^H(i) = \nu \sum_j a_{ij} c^A(j)$$

and authority centrality as:

$$c^A(i) = \mu \sum_j a_{ji} c^H(j)$$

Let $\lambda = \frac{1}{\nu\mu}$, we get:

$$A^t A c^A = \lambda c^A$$

$$A A^t c^H = \lambda c^H$$

Closeness Centrality

Another approach is to consider **shortest paths** between nodes.

A **geodesic** is a shortest path between 2 nodes (smallest number of hops, or smallest sum of edge weights).

Let d_{ij}, the length of a geodesic between nodes v_i and v_j, with $d_{ii} = 0$.

For disconnected graphs, we may have $d_{ij} = \infty$; for unweighted graphs, we can set $d_{ij} = n$ in such cases.

Definition: The **closeness centrality** of node v_i is defined as:

$$c^C(i) = \left(\sum_{j \neq i} d_{ij} \right)^{-1}$$

Betweennes Centrality

Another approach is to consider all geodesics; let:

n_{jk}: number of geodesics between nodes v_j and v_k

$n_{jk}(i)$: number of geodesics between nodes v_j and v_k passing through node v_i.

Definition: The *betweenness centrality* of node v_i is:

$$c^B(i) = \sum_{j \neq i} \sum_{k \neq i,j} \frac{n_{jk}(i)}{n_{jk}}$$

Edge Centrality

Betweenness centrality can be re-defined for the edges $e \in E$; let:

n_{jk}: number of geodesics between nodes v_j and v_k

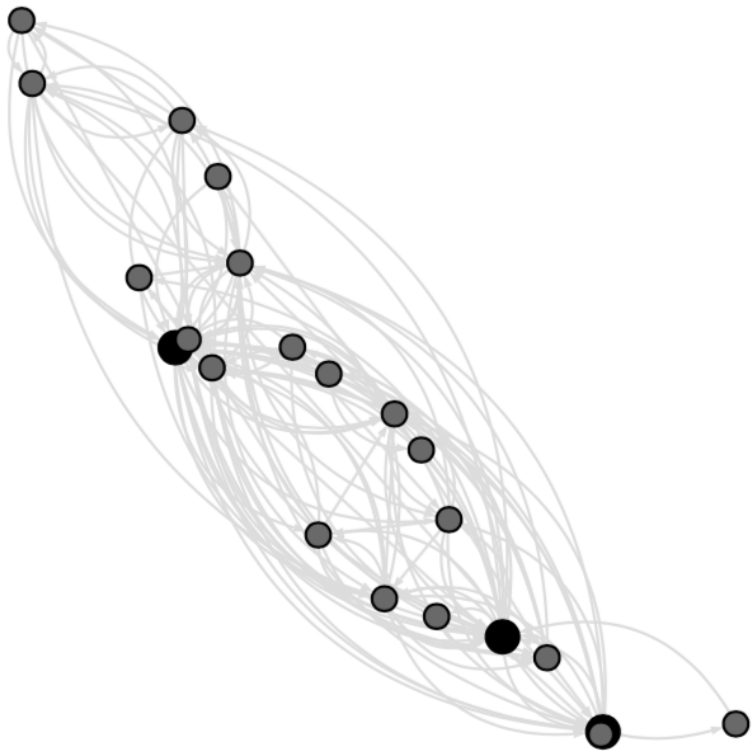
$n_{jk}(e)$: number of geodesics between nodes v_j and v_k passing through edge e.

Definition: The **betweenness centrality** of edge e is:

$$c^B(e) = \sum_j \sum_{k \neq j} \frac{n_{jk}(e)}{n_{jk}}$$

Airport Dataset

Nodes: US airports (464)
Edges: passenger volume (directed, weighted) (12k)
California subgraph:



Airport Dataset

airport	degree	pagerank	authority	hub	between	closeness
LAX	0.117	0.215	1.000	1.000	0.507	0.318
SFO	0.090	0.173	0.970	0.907	0.362	0.318
SAN	0.079	0.124	0.689	0.726	0.014	0.292
OAK	0.047	0.073	0.474	0.441	0.030	0.288
SJC	0.042	0.067	0.416	0.390	0.047	0.300

TABLE 3.5
Top ranked nodes sorted with respect to degree centrality.

Roadmap

1. Part I
   - Background material
   - Relational data and graphs
   - Random graph models
2. Part II
   - Measures of centrality
   - **Degree correlation**
   - Graph (vertex) embedding
   - *First notebook*
3. Part III
   - Community detection
   - *Second notebook*
   - Hypergraphs and other topics

Degree Correlation

Let p_i, the proportion of nodes of degree i in G, the **degree distribution**

The degree correlation measures the relationship between degree of nodes linked by edges

In an **assortative** network, high degree nodes tend to be more connected to other high degree nodes, and the same for low degree nodes.

In a **disassortative** network, high degree nodes tend to be more connected to low degree nodes, and vice-versa.

Degree Correlation

Definition: The *average nearest neighbors degree* function for nodes of degree k is given by:

$$k_{nn}(k) = \sum_{k'} k' P(k'|k)$$

with $P(k'|k)$ the probability that from an edge incident to a degree k node leads to a degree k' node.

If there is no correlation (**neutral** network), we get

$$k_{nn}(k) = \frac{\sum_i i^2 \cdot p_i}{\sum_i i \cdot p_i}$$

where p_i: proportion of nodes of degree i.

Degree Correlation

The **friendship paradox** states that the average degree of a node's neighbours is typically higher than its own degree.

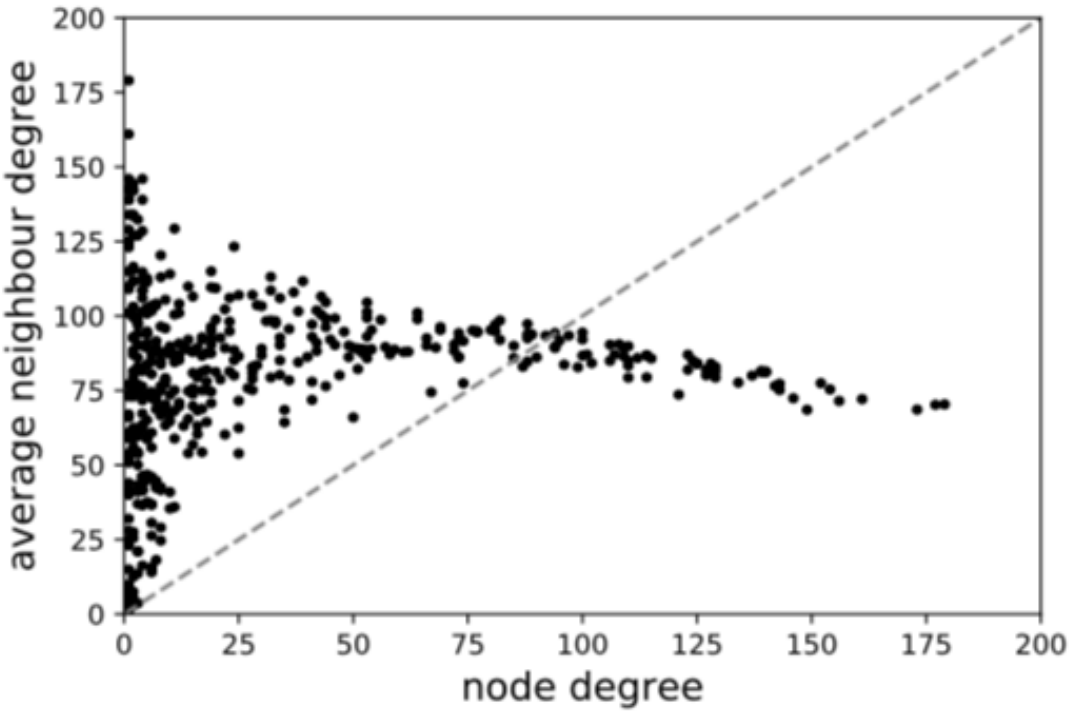
This is due to the fact that in many networks:

$$\frac{\sum_i i^2 \cdot p_i}{\sum_i i \cdot p_i} >> \sum_i i \cdot p_i$$

Simply stated, it is more likely to link with hubs (high degree nodes).

Degree Correlation

For the US airport graph:



(a) Friendship paradox

Degree Correlation

The degree correlation function can be approximated by

$$k_{nn}(k) = ak^{\mu}$$

where μ is the **correlation exponent** and:

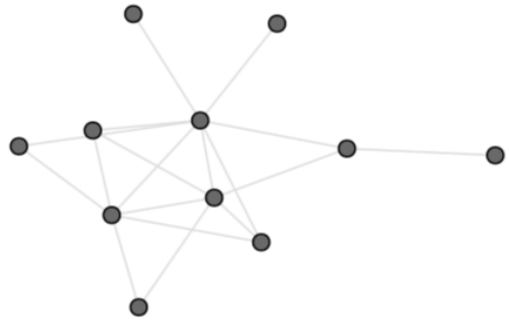
- $\mu > 0$ for assortative networks
- $\mu \approx 0$ for neutral networks, and
- $\mu < 0$ for disassortative networks

Degree Correlation

(a) +1 assortativity (AR)

(b) -1 assortativity (NE).

(c) Positive assortativity (ND and SD)



(d) Negative assortativity (MO).

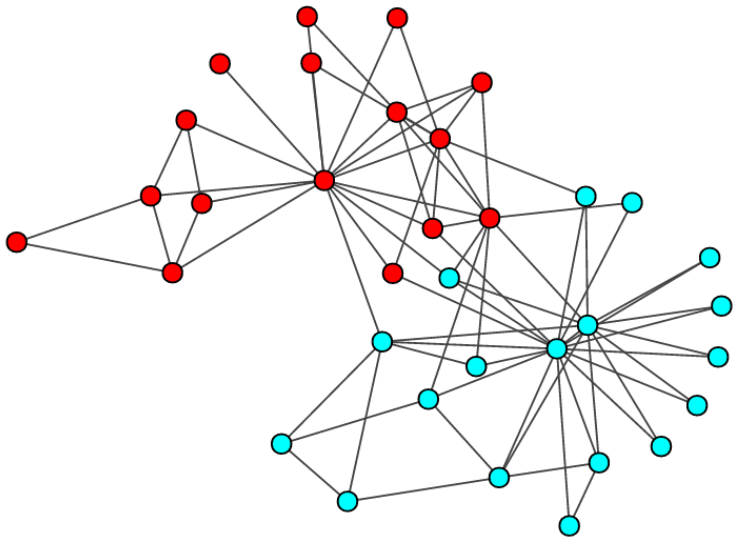
FIGURE 4.7
State subgraphs for the US airport graph with positive ((a) and (c)) and negative ((b) and (d)) assortativity.

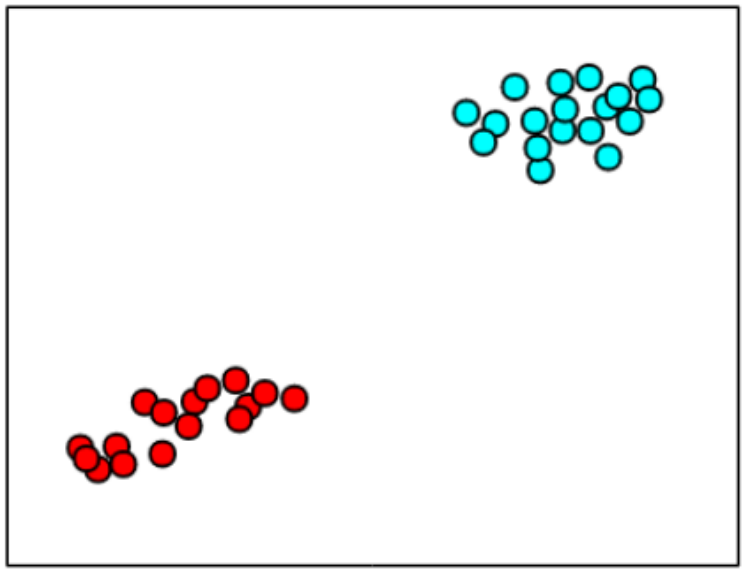
Roadmap

1. Part I
   - Background material
   - Relational data and graphs
   - Random graph models
2. Part II
   - Measures of centrality
   - Degree correlation
   - **Graph (vertex) embedding**
   - *First notebook*
3. Part III
   - Community detection
   - *Second notebook*
   - Hypergraphs and other topics

Graph Embedding

Map graph vertices to vector (feature) space





Graph Embedding

Map *similar* nodes to nearby location in vector space.
Similar may have different meaning:

- close w.r.t. graph topology
- similar role in graph (ex: similar degree)
- similar node attributes

Graph Embedding

Examples of applications:

- feature learning (rather than engineering)
- visualization
- link prediction
- community detection
- anomaly detection
- network evolution (dynamics)

Graph Embedding

Several methods are based on random walks and the SkipGram approach for word embedding.

- A word can be characterized by the company it keeps
- Words in similar context (nearby word(s)) have similar meaning
- Consider a window around each word; build ”word vectors” (ex: word2vec)
- Use those as training data

In graphs: nodes are “words” and we define walks in various ways.

node2vec

- node2vec defines biased random walks
- mix of breadth and depth first search

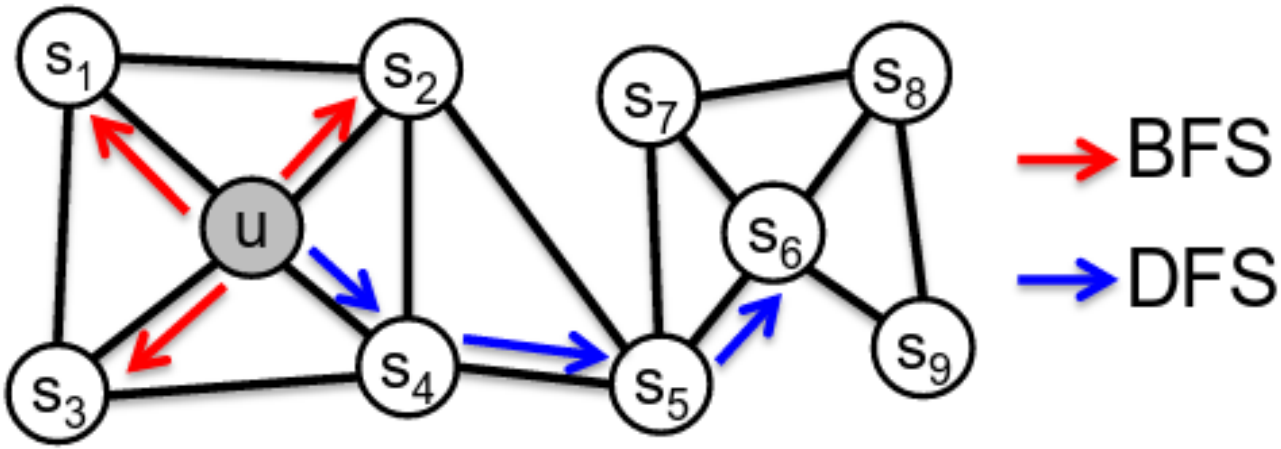


Figure 1: BFS and DFS search strategies from node u ($k = 3$).

REF: node2vec: Scalable Feature Learning for Networks, A. Grover and J. Leskovec

Other Algorithms

Category	Year	Published	Method	Time Complexity	Properties preserved
Factorization	2000	Science[26]	LLE	$O(\|E\|d^2)$	1^{st} order proximity
	2001	NIPS[25]	Laplacian Eigenmaps	$O(\|E\|d^2)$	
	2013	WWW[21]	Graph Factorization	$O(\|E\|d)$	
	2015	CIKM[27]	GraRep	$O(\|V\|^3)$	$1-k^{th}$ order proximities
	2016	KDD[24]	HOPE	$O(\|E\|d^2)$	
Random Walk	2014	KDD[28]	DeepWalk	$O(\|V\|d)$	
	2016	KDD[29]	*node2vec*	$O(\|V\|d)$	$1-k^{th}$ order proximities, structural equivalence
Deep Learning	2016	KDD[23]	SDNE	$O(\|V\|\|E\|)$	1^{st} and 2^{nd} order proximities
	2016	AAAI[30]	DNGR	$O(\|V\|^2)$	$1-k^{th}$ order proximities
	2017	ICLR[31]	GCN	$O(\|E\|d^2)$	$1-k^{th}$ order proximities
Miscellaneous	2015	WWW[22]	LINE	$O(\|E\|d)$	1^{st} and 2^{nd} order proximities

Table 1: List of graph embedding approaches

REF: Graph Embedding Techniques, Applications, and Performance: A Survey, P. Goyal and E. Ferrara

Which embedding?

- which embedding algorithm should I use?
- how to select the parameters?
- how do I know if the representation is good?
- GIGO: bad representation in vector space leads to bad results ...

Results can vary a lot between algorithms, and with the choice of parameters...

Proposed Framework

Given $G = (V, E)$ on n vertices and an embedding of its vertices to k-dimensional space, $\mathcal{E} : V \to \mathbb{R}^k$.

Our goal is to assign a "divergence score" to this embedding.

The lower the score, the better the embedding is. This will allow us to compare several embeddings, possibly in different dimensions.

Code: github.com/KrainskiL/CGE.jl

LNCS vol 12091: doi.org/10.1007/978-3-030-48478-1_4

Overview

We have two main ingredients in our framework:

- **Graph topology view:** a good, stable **graph clustering algorithm**;
- **Spatial view:** we introduce the **Geometric Chung-Lu (GCL)** model based on the degree distribution **d** and the embedding $\mathcal{E}$.

With **GCL**, edge probabilities $p_{i,j} \propto 1/dist_{\mathcal{E}}(v_i, v_j)$, so long edges should occur less frequently than short ones.

Moreover, as with original model, expected degrees **d** are preserved.

Framework

Step 1

Run some stable *graph* clustering algorithm on G to obtain a partition $\mathbf{C}$ of the vertex set V into ℓ communities $C_1, \ldots, C_\ell$.

Let:
c_i: proportion of edges with both endpoints in C_i
$c_{i,j}$: proportion of edges with one endpoint in C_i and the other one in C_j

Define:

$$\bar{\mathbf{c}} = (c_{1,2}, \ldots, c_{1,\ell}, c_{2,3}, \ldots, c_{2,\ell}, \ldots, c_{\ell-1,\ell}), \ \hat{\mathbf{c}} = (c_1, \ldots, c_\ell)$$

These vectors characterize partition $\mathbf{C}$ from the perspective of G.

The embedding $\mathcal{E}$ does *not* affect the vectors $\bar{\mathbf{c}}$ and $\hat{\mathbf{c}}$.

Given the GCL model and embedding $\mathcal{E}$, we compute:
b_i: expected proportion of edges within C_i
$b_{i,j}$: expected proportion of edges with one endpoint in C_i and the other one in C_j

We get:

$$\bar{\mathbf{b}}_{\mathcal{E}} = (b_{1,2}, \ldots, b_{1,\ell}, b_{2,3}, \ldots, b_{2,\ell}, \ldots, b_{\ell-1,\ell})$$

$$\hat{\mathbf{b}}_{\mathcal{E}} = (b_1, \ldots, b_\ell)$$

These vectors characterizes partition $\mathbf{C}$ from the perspective of the embedding $\mathcal{E}$.

Framework

Step 4

We compute the Jensen-Shannon divergence (JSD) between $(\bar{\mathbf{c}}, \hat{\mathbf{c}})$ and $(\bar{\mathbf{b}}_{\mathcal{E}}, \hat{\mathbf{b}}_{\mathcal{E}})$.

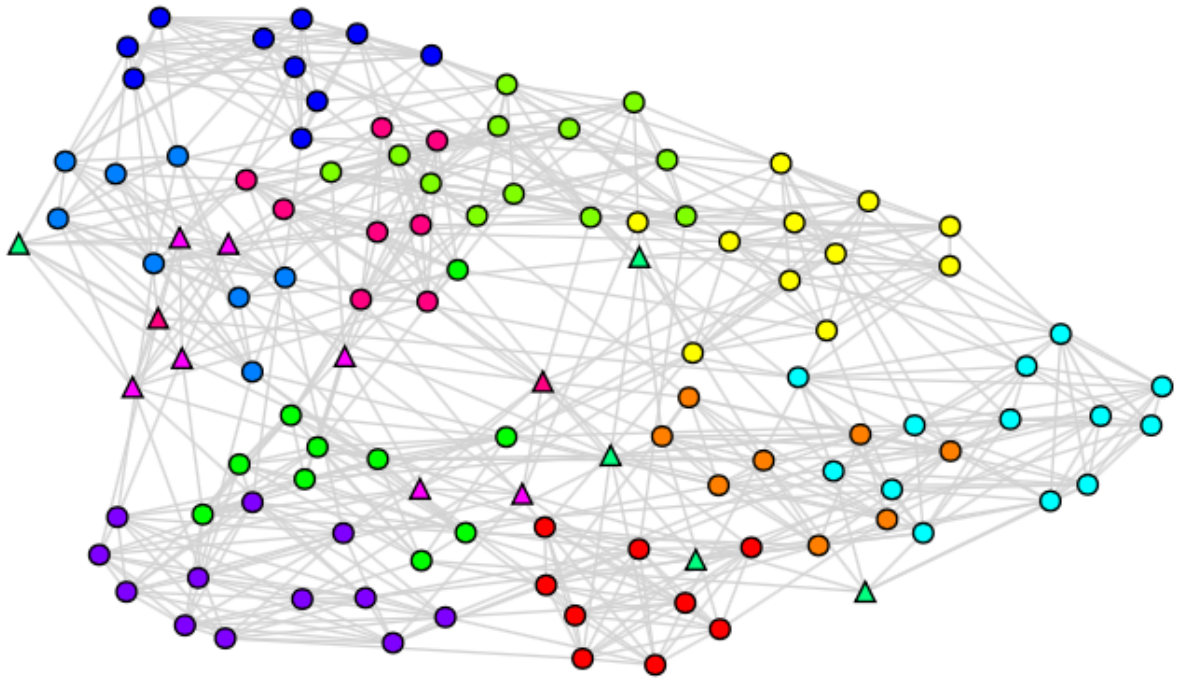
To compare several embeddings for the same graph, we repeat steps 3-4 and compare the divergence scores (lower score is better).

Steps 1-2 are done only once, so we use the same partition into ℓ communities for each embedding.

Dataset

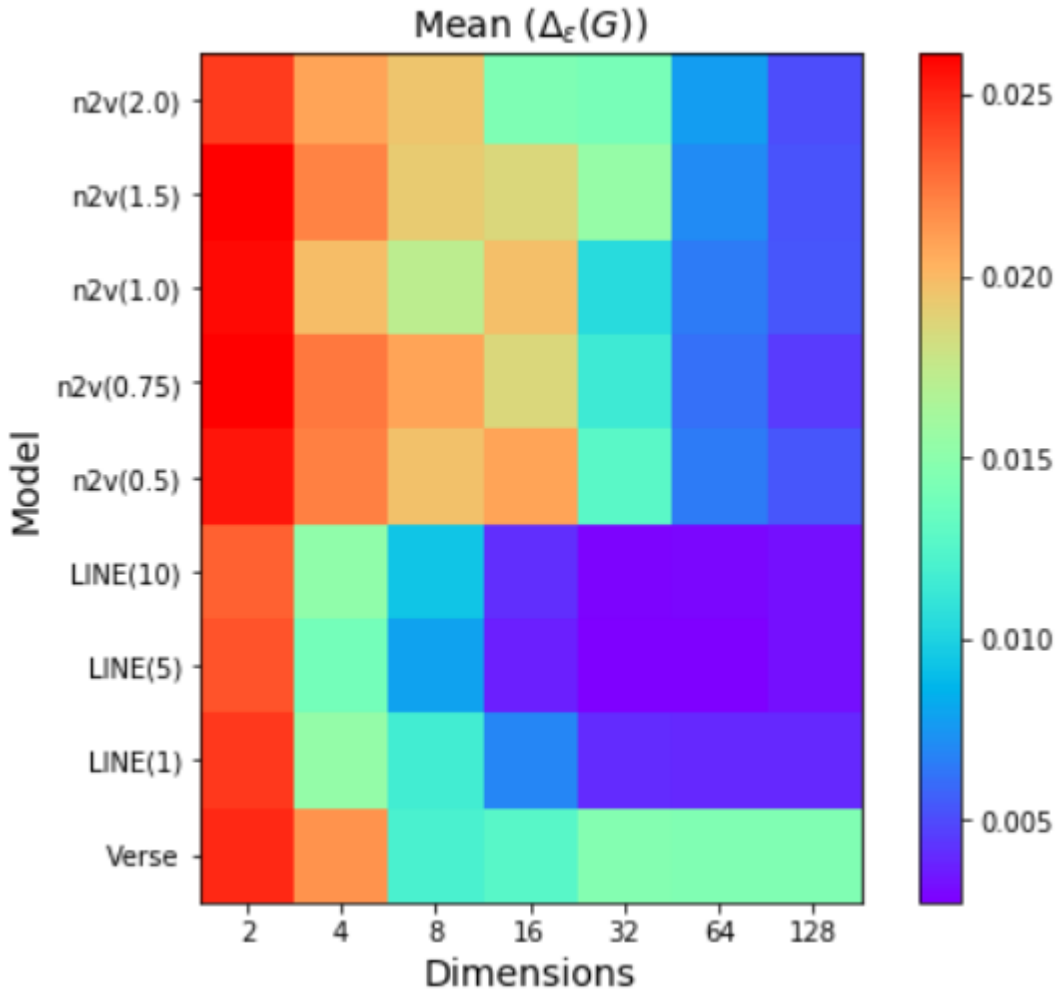
US College Football

Nodes: teams (115)
Edges: games played (613)



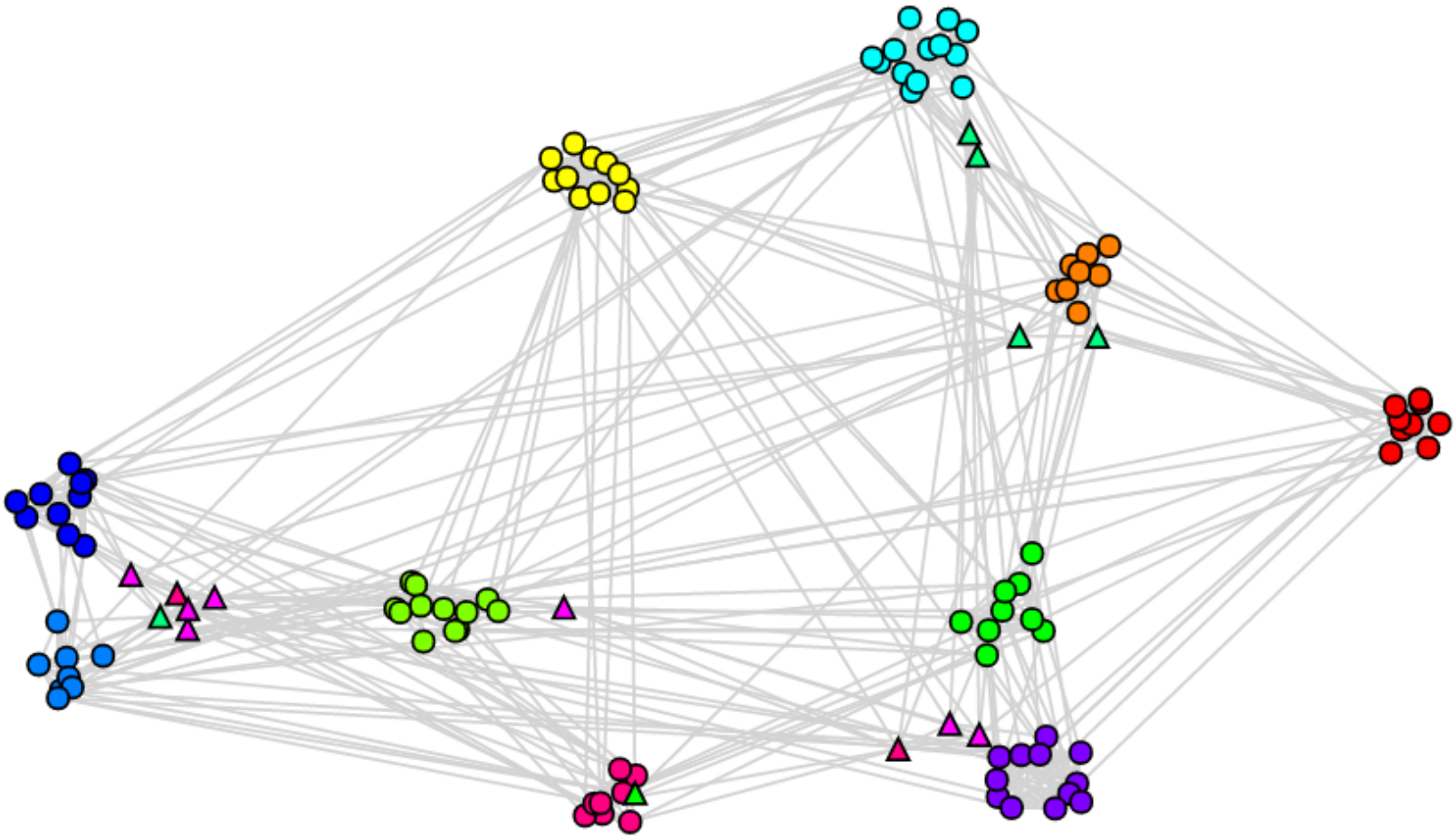
Football Graph

630 embeddings



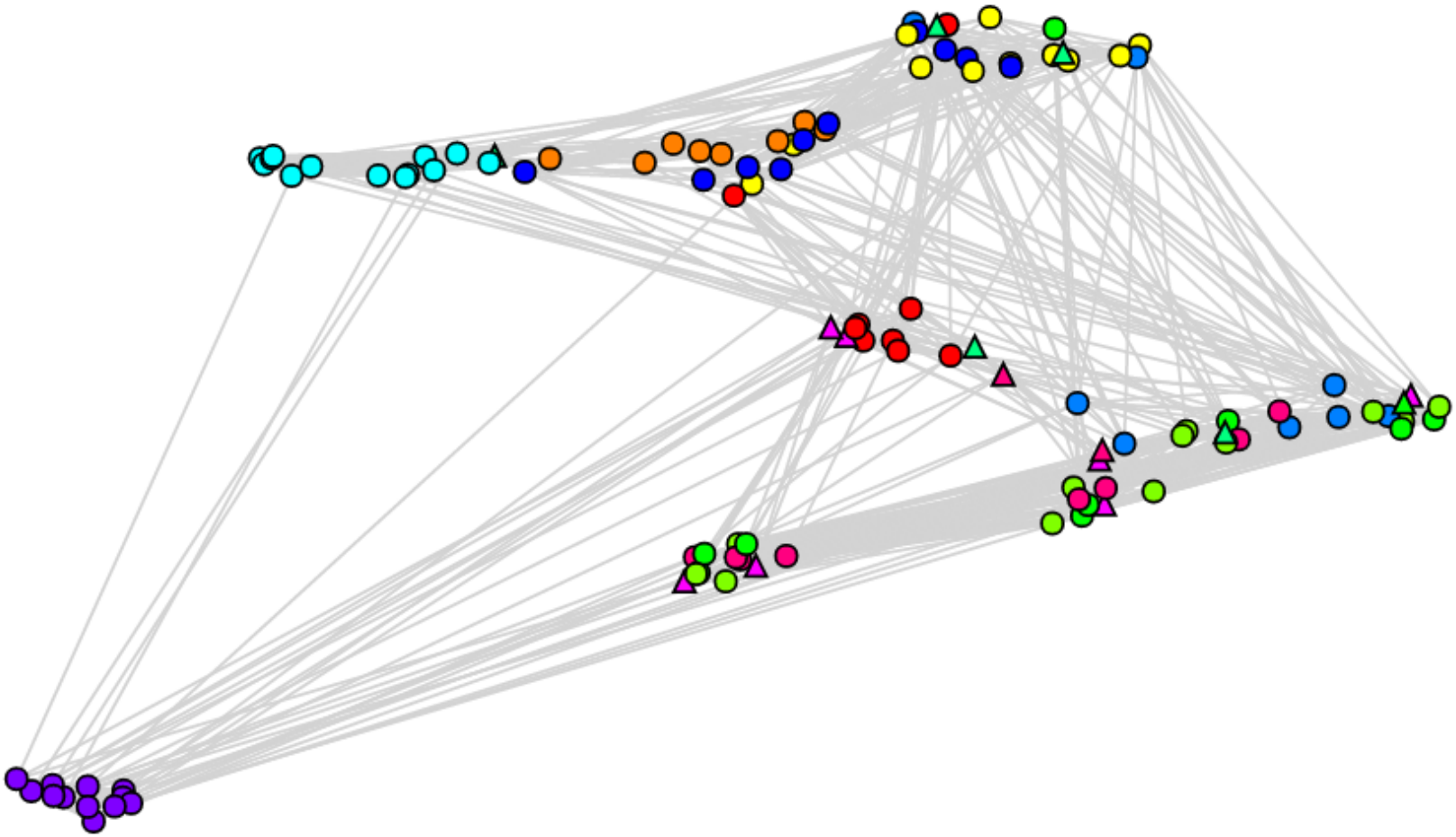
Football Graph

low divergence embedding



Football Graph

high divergence embedding



Roadmap

1. Part I
   - Background material
   - Relational data and graphs
   - Random graph models
2. Part II
   - Measures of centrality
   - Degree correlation
   - Graph (vertex) embedding
   - ***First notebook***
3. Part III
   - Community detection
   - *Second notebook*
   - Hypergraphs and other topics

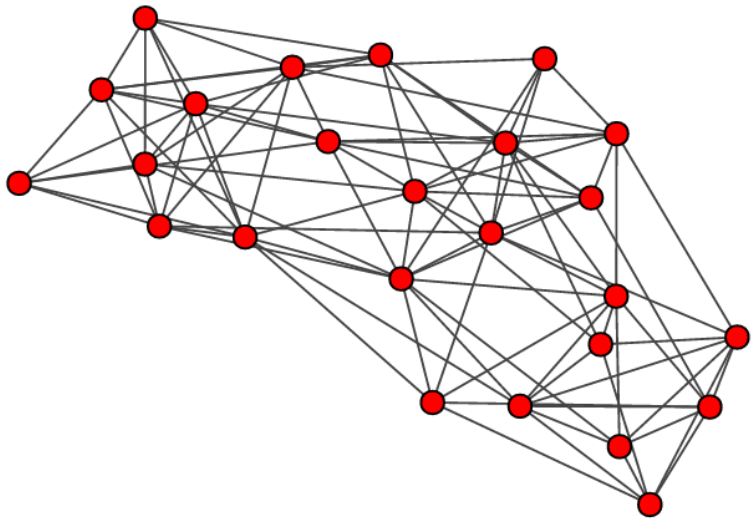
Roadmap

1. Part I
   - Background material
   - Relational data and graphs
   - Random graph models
2. Part II
   - Measures of centrality
   - Degree correlation
   - Graph (vertex) embedding
   - *First notebook*
3. Part III
   - **Community detection**
   - *Second notebook*
   - Hypergraphs and other topics

Definitions

We start from two fundamental hypothesis [Ref: Barabasi]:

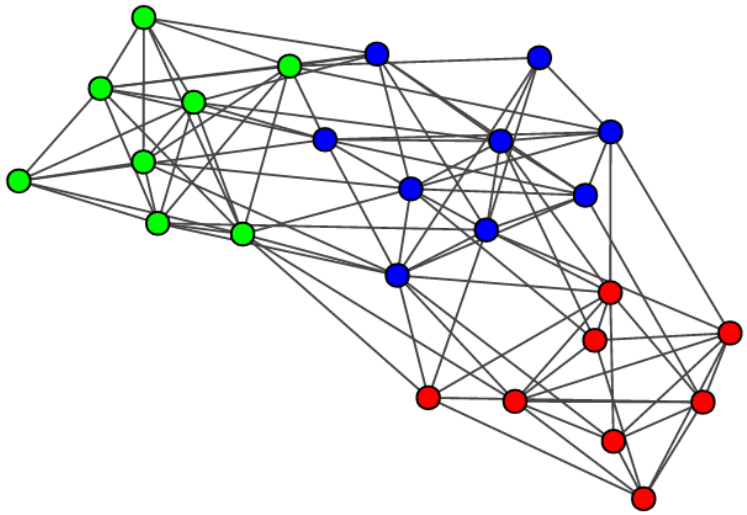
- A network's community structure is uniquely encoded in its wiring diagram.



Definitions

We start from two fundamental hypothesis [REF: Barabasi]:

- A network's community structure is uniquely encoded in its wiring diagram.
- A community is a locally dense connected subgraph in a network.



Definitions

For a graph $G = (V, E)$, consider the connected subgraph C induced by a subset of nodes $V_C \subset V$ with $v_i \in V_C$ for some node v_i.

Define the *internal degree* for node $v_i \in V_C$ as its degree within subgraph C, denoted $d_i^{int}(C)$.

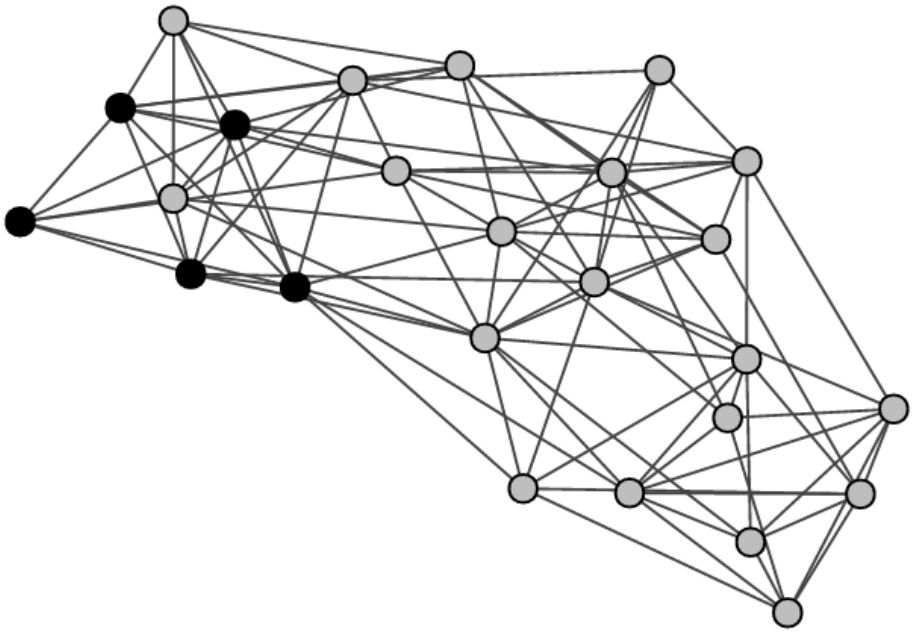
The *external degree* for node v_i is $d_i^{ext}(C) = d_i - d_i^{int}(C)$, where d_i is the total degree for node v_i in G.

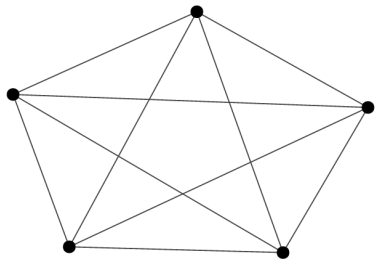
C is a **strong community** if $d_i^{int}(C) > d_i^{ext}(C)$ for each $v_i \in V_C$.

C is a **weak community** if $\sum_{i \in V_C} d_i^{int}(C) > \sum_{i \in V_C} d_i^{ext}(C)$.

Definitions

A **clique** is a fully connected subgraph of G:

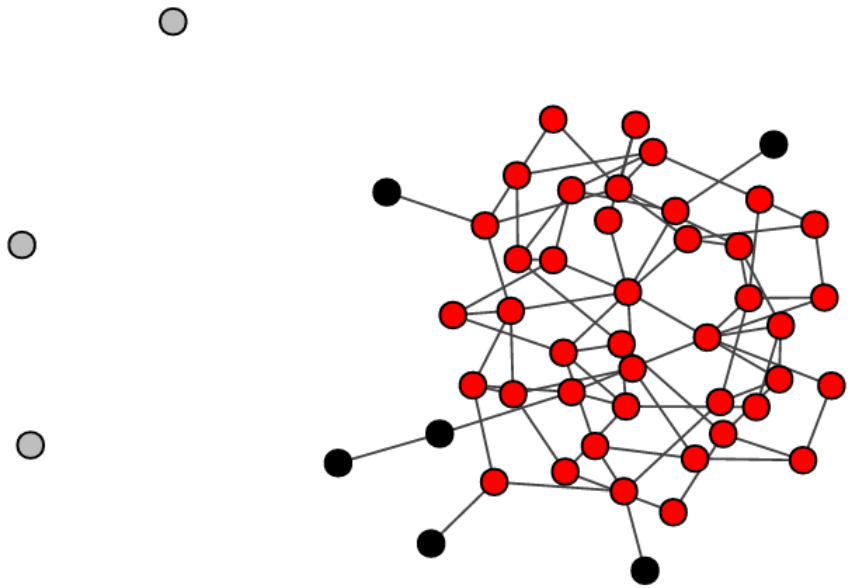




Definitions

A **k-core** is a maximal connected subgraph of G where all nodes have degree at least k.

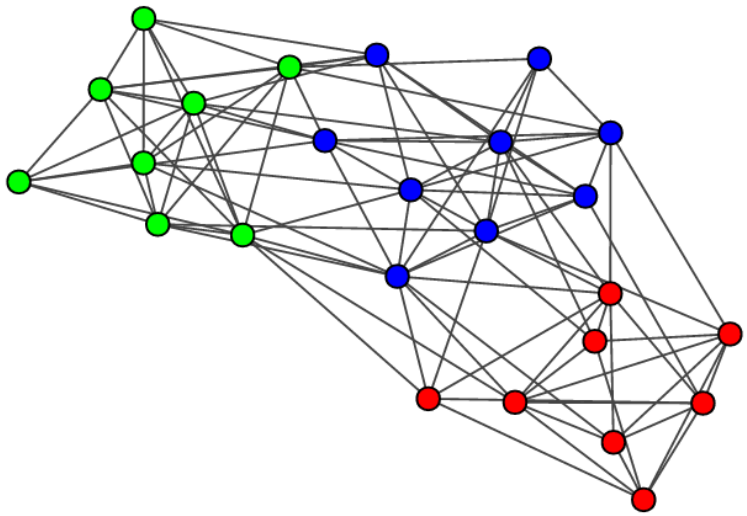
Example of a graph and vertices with coreness 0, 1 and 2:



Definitions

A **clustering** of the graph $G = (V, E)$ of size k is a partition of the nodes $V = V_1 \cup \cdots \cup V_k$ where:

- all $V_i \cap V_j = \emptyset, \ i \neq j$
- for each part (or cluster) V_i, the induced subgraph G_i is connected.



Benchmarks

Benchmark models with **communities**:

- good way to test and compare algorithm
- control the noise level, the community sizes, etc.
- ground-truth is rarely available with real graphs
- when available, ground-truth may not coincide with the fundamental hypothesis

Stochastic Block Models

In this model, we fix the number of nodes n and the number of communities k;

For the communities, we either:

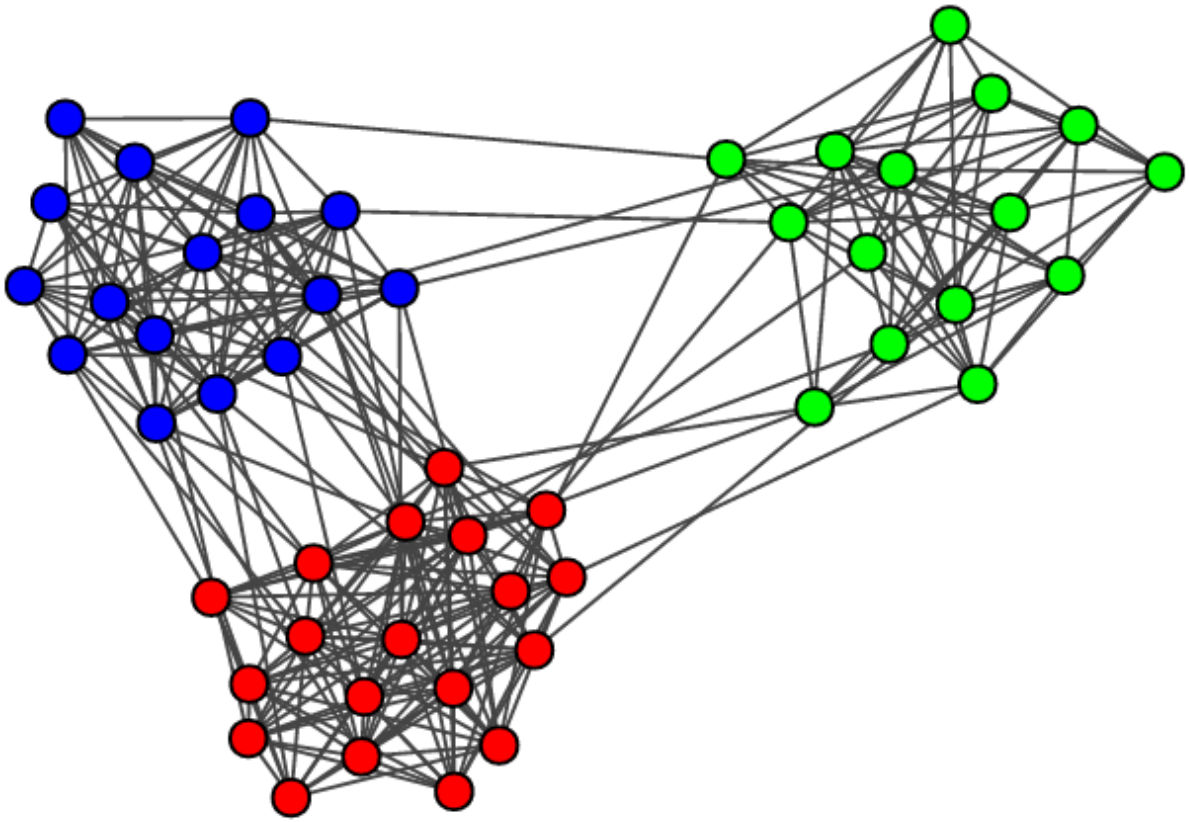
- divide the nodes equally between each community, or
- assign each node independently to community i with probability p_i with $\sum_{i=1}^{k} p_i = 1$.

For each pair of nodes resp. in communities i and j, add an edge with probability $P(i, j)$.

A special case is to assign all $P(i, i) = p_{in}$ and all $P(i, j) = p_{out}, \ i \neq j$.

Stochastic Block Models

Example graph:



LFR

Fix the number of nodes n.
The Lancichinetti-Fortunato-Radicchi (LFR) benchmark has 3 main parameters:

- γ_1: node degrees follow a power law distribution with $p_n \propto n^{-\gamma_1}$; recommended values are $2 \leq \gamma_1 \leq 3$.
- γ_2: community sizes follow a power law distribution with $p_k \propto k^{-\gamma_2}$; recommended values are $1 \leq \gamma_2 \leq 2$.
- $0 \leq \mu \leq 1$: for each node, this is the expected proportion of edges linking to other communities, while $(1 - \mu)$ is the proportion within its own community.

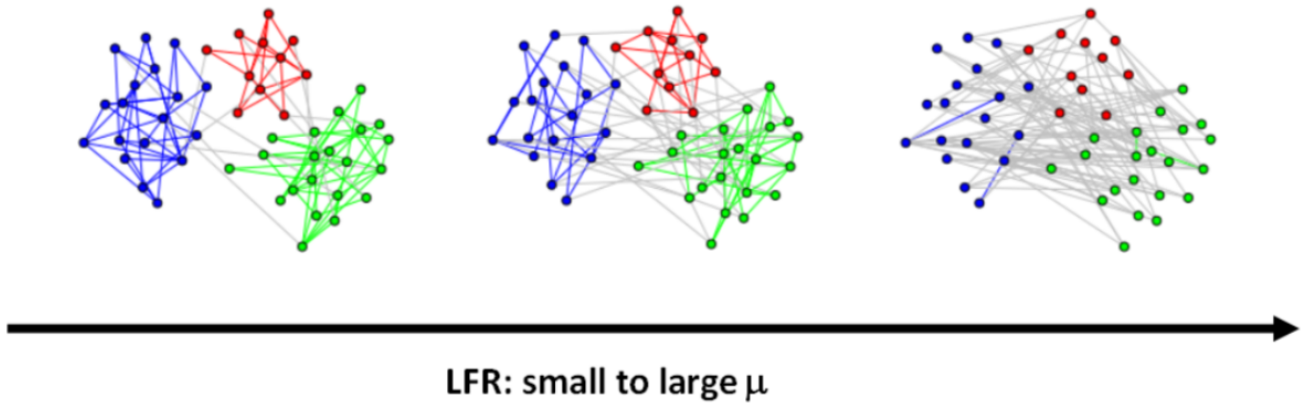
μ is called the noise level, or mixing parameter.
Each node is assigned to a single community

LFR

LFR's scalability is somewhat limited; some fast variants do early stop.

It can be challenging to analyze it theoretically.

The mixing parameter μ can lead to unnaturally-defined networks (anti-communities):



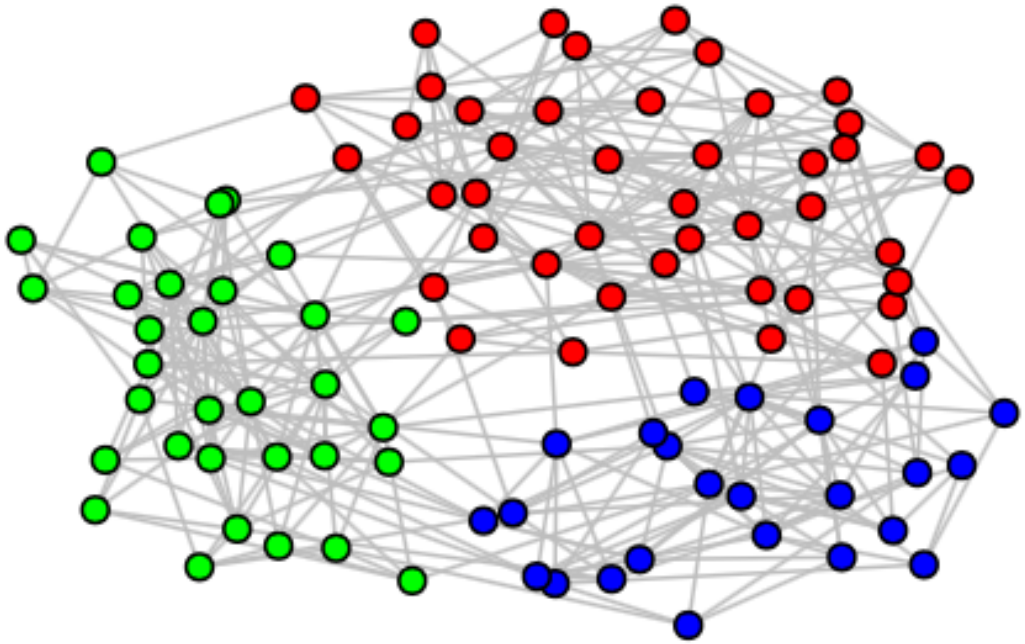
ABCD

Artificial Benchmarks for Community Detection

- power-law degree and community size distribution as with LFR
- mixing parameter ξ: trade-off between pure communities and random graph
- based on the Configuration or Chung-Lu model
- simple and highly scalable
- even faster parallel version available soon
- Code: github.com/bkamins/ABCDGraphGenerator.jl
- Network Science Journal (2021): doi.org/10.1017/nws.2020.45

ABCD

ABCD graph with $\xi = 0.2$ and 3 communities:



Node can be added that do not connect w.r.t. a community

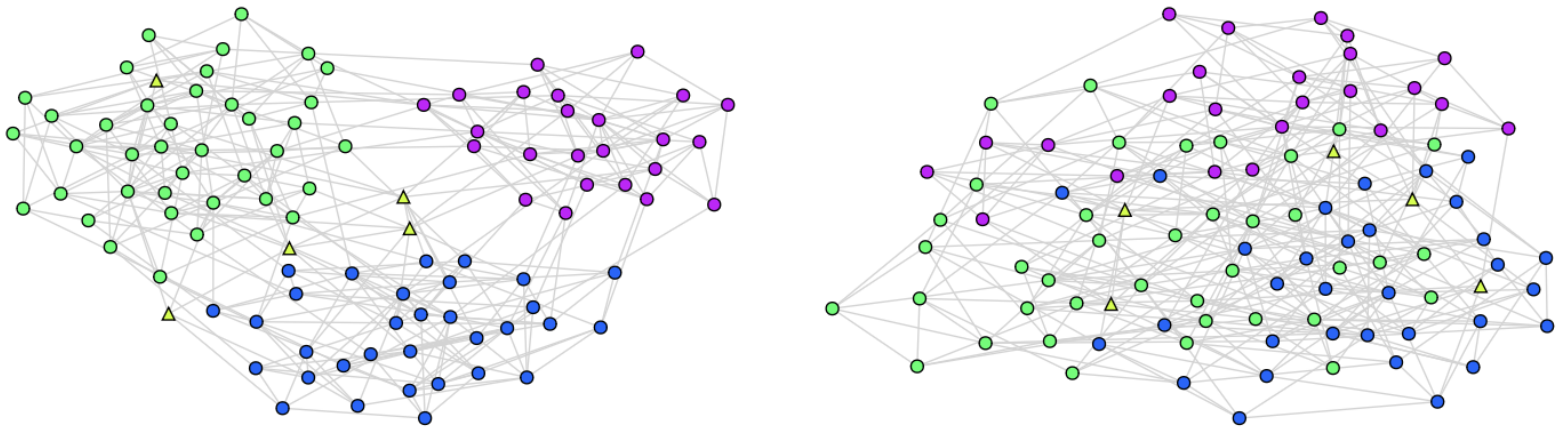


Fig. 2: Two examples of **ABCD+o** graphs with low level of noise ($\xi = 0.2$, left) and high level of noise ($\xi = 0.4$, right). The number of outliers is $s_0 = 5$.

Measures to compare Graph Clusterings

Such measures are important for:

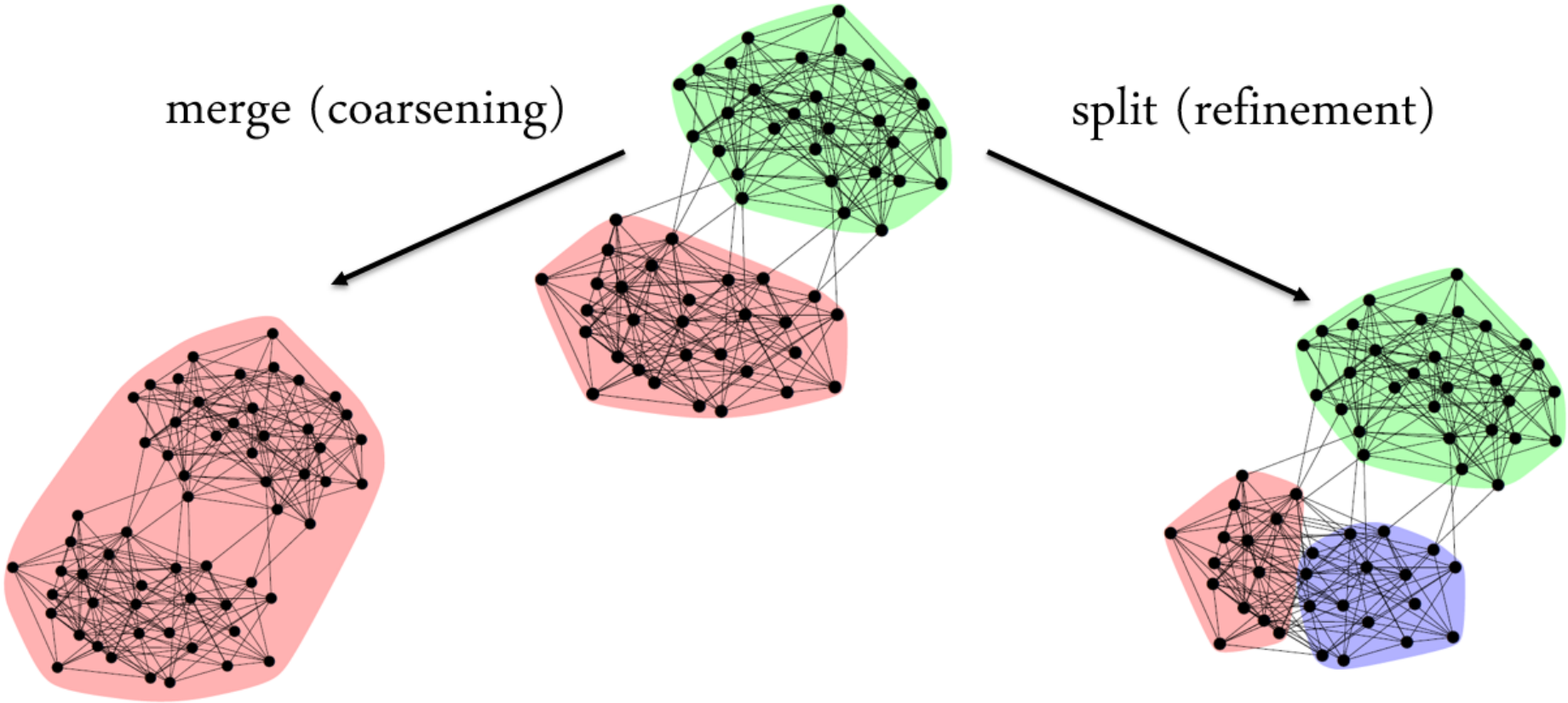
- compare clustering algorithms w.r.t. quality, stability
- compare to ground truth if available

Some recommended measures are:

- Adjusted Mutual Information (AMI)
- Adjusted RAND Index (ARI)
- Graph-aware ARI (AGRI)
- Code: github.com/ftheberge/graph-partition-and-measures

Complementarity

Graph-aware and agnostic measures have opposite behaviours with respect to resolution issues.



Complementarity

Let G with ground-truth community $\mathbf{A}$ and let $\mathbf{B_1}$ and $\mathbf{B_2}$ be a coarsening and a refinement of $\mathbf{A}$ respectively.

Under some conditions, $\mathbf{A}$ is closer to $\mathbf{B_2}$ than to $\mathbf{B_1}$ under the graph-agnostic measures, and the opposite is true under the graph-aware measures.

Larger number of clusters are favoured when using graph-agnostic measures, smaller number for graph-aware measures.

Complementarity

Let $\mathcal{G}(n, p, q, \mathbf{A})$: random graphs with n vertices split into a partition $\mathbf{A}$ with p (resp. q) the proportion of randomly selected pairs of vertices in *same* (resp. *different*) parts of $\mathbf{A}$ sharing an edge.

Theorem 1 *Consider* $G_{\mathbf{A}} \sim \mathcal{G}(n, p, q, \mathbf{A})$ *with* $\mathbf{B}_1 > \mathbf{A}$ *a coarsening of* $\mathbf{A}$ *and* $\mathbf{B}_2 < \mathbf{A}$, *a refinement of* $\mathbf{A}$ *such that* $|P_{\mathbf{A}}|^2 < |P_{\mathbf{B}_1}| \cdot |P_{\mathbf{B}_2}|$. *Then*

(i) $PC_{mn}(\mathbf{A}, \mathbf{B}_1) < PC_{mn}(\mathbf{A}, \mathbf{B}_2)$.

(ii) $\mathbb{E}_{G_{\mathbf{A}}}[PC_{mn}(\mathbf{A}, \mathbf{B}_1; G_{\mathbf{A}})] > \mathbb{E}_{G_{\mathbf{A}}}[PC_{mn}(\mathbf{A}, \mathbf{B}_2; G_{\mathbf{A}})]$, *if* $p > q \frac{|P_{\mathbf{B}_1} \setminus P_{\mathbf{A}}|}{|P_{\mathbf{A}} \setminus P_{\mathbf{B}_2}|}$.

Modularity

Barabasi's third fundamental hypothesis:

- Randomly wired networks lack an inherent community structure.

Modularity uses random wiring as a *null model* to quantify the community structure for some graph partition.

Given:

$$G = (V, E),\ E \subset V \times V,\ \ |V| = n,\ |E| = m$$

- Sample m edges $e = (u_1, u_2)$
- Each u_i is independently sampled from V such that $p(u_i) \propto deg_G(u_i)$
- Let $\mathcal{G}$, the distribution of such graphs

Modularity

The **modularity** of a partition $\mathbf{A} = \{A_1, \ldots, A_k\}$ of V can be written as:

$$q_G(\mathbf{A}) = \frac{1}{|E|} \sum_{A_i \in \mathbf{A}} \left(e_G(A_i) - \mathop{\mathbb{E}}_{G' \sim \mathcal{G}} (e_{G'}(A_i)) \right)$$

- $e_G(A_i) = |\{e \in E : e \subseteq A_i\}|$ is the *edge contribution*
- $\mathbb{E}_{G' \sim \mathcal{G}}(e_{G'}(A_i))$ is the *degree tax*

Maximum modularity $q^*(G)$ of a graph is the maximum value taken by the above over all possible partitions.

Modularity

Barabasi's fourth fundamental hypothesis:

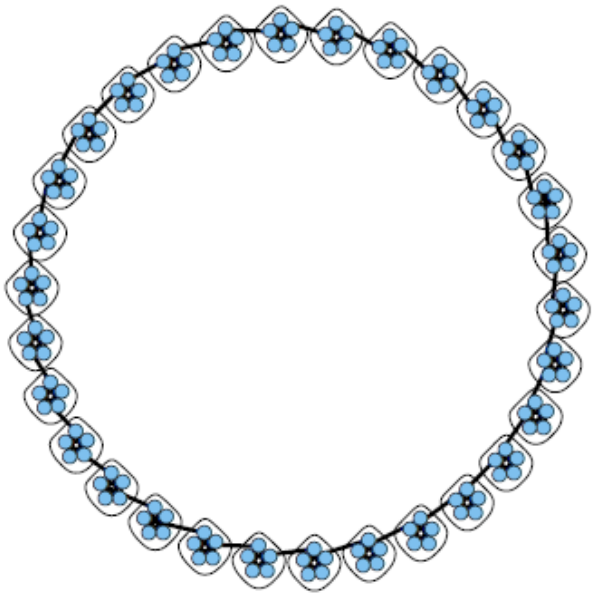
- For a given network, the partition with maximum modularity corresponds to the optimal community structure.

There are however some known issues with modularity ...

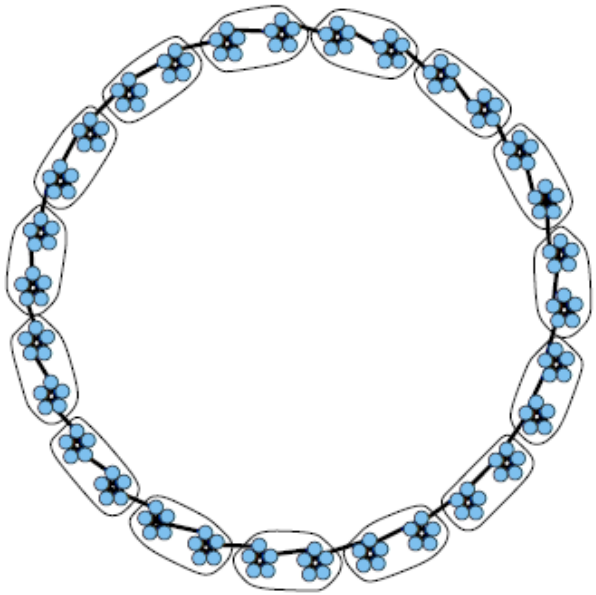
"Optimal" may not always translate to "intuitive".

Resolution Issue

Modularity-based algorithm suffer from the resolution limit issue
Ring of cliques example:



Q = 0.876



Q = 0.888

Clustering algorithms

Several clustering algorithms are based on modularity:

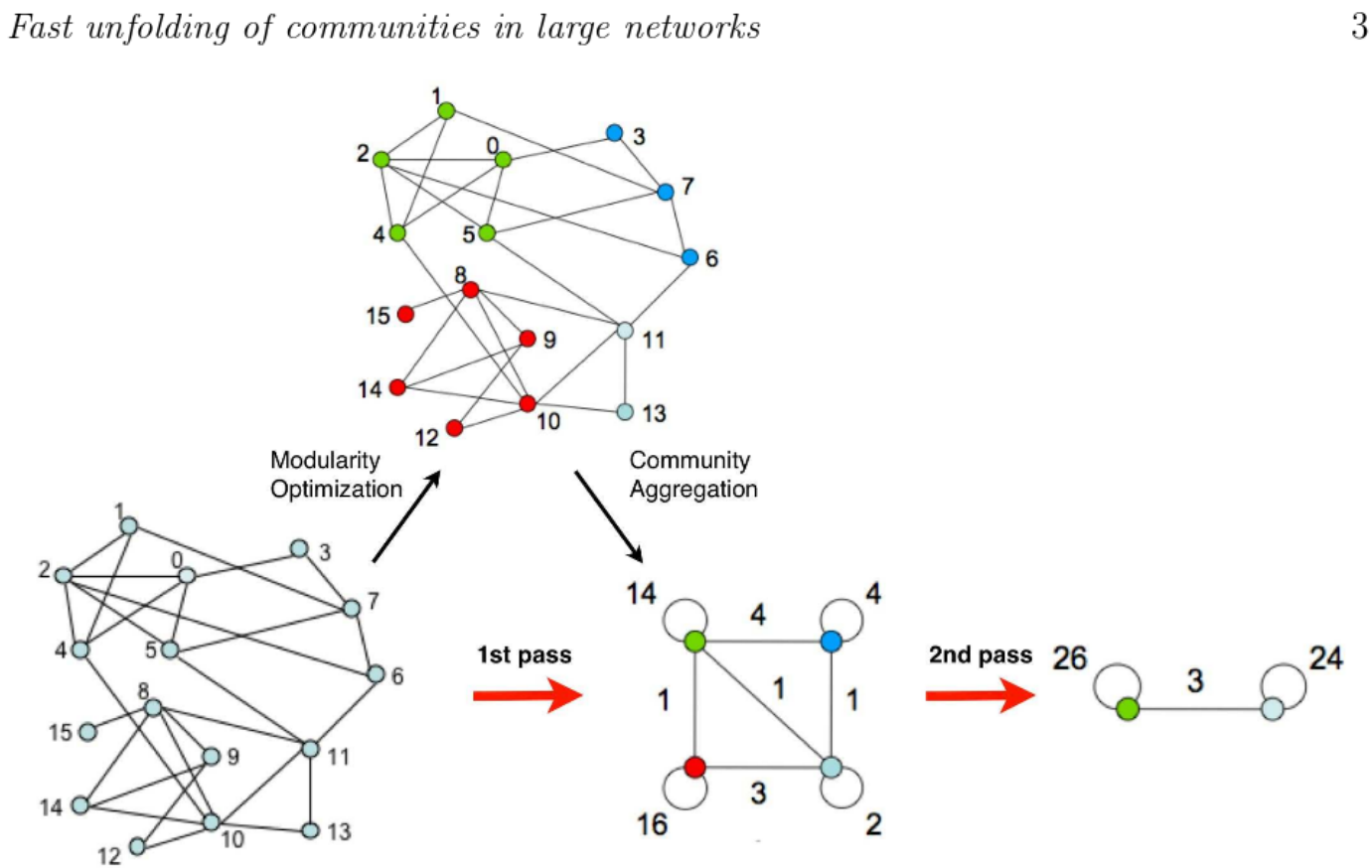
- CNM
- Multilevel Louvain (ML)
- ECG (ensemble clustering)
- Leiden

There are several other good algorithms, including:

- Infomap (IM)
- WalkTrap (WT)
- Label propagation
- etc.

Multilevel (Louvain) algorithm

Ref: Blondel *et al.*, arXiv:0803.0476v2



ECG Algorithm

The ECG algorithm is a consensus clustering algorithm for graphs. The steps are:

- generation step: k randomized level-1 partitions from Louvain (ML) algorithm: $\mathcal{P} = \{P_1, \ldots, P_k\}$.
- integration step: run ML on a re-weighted version of the initial graph $G = (V, E)$. Derived edge weights are obtained through co-association in $\mathcal{P}$.

Code: github.com/ftheberge/graph-partition-and-measures

```
pip install partition-igraph
pip install partition-networkx
```

ECG Algorithm

Main advantages of ECG:

- greatly reduces the resolution limit issues
- generation steps yields useful derived edge weights (ex: community membership strength)
- improved stability over Louvain
- improved results over benchmark graphs

Network Science (2019): doi.org/10.1007/s41109-019-0162-z

Resolution Issue

Results over generalized ring-of-cliques:

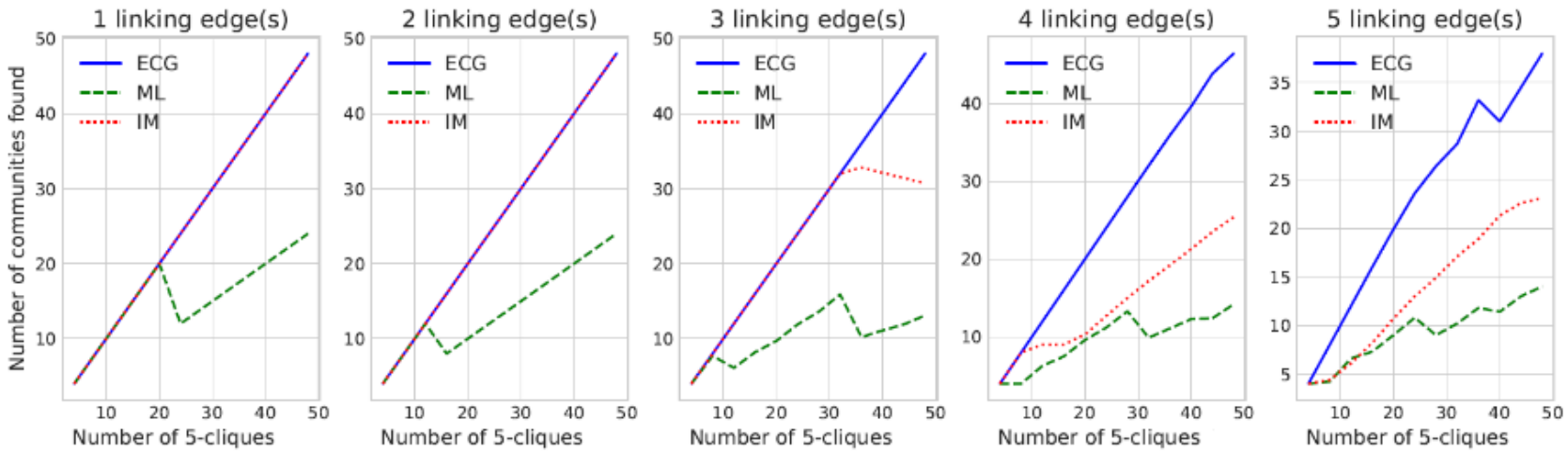
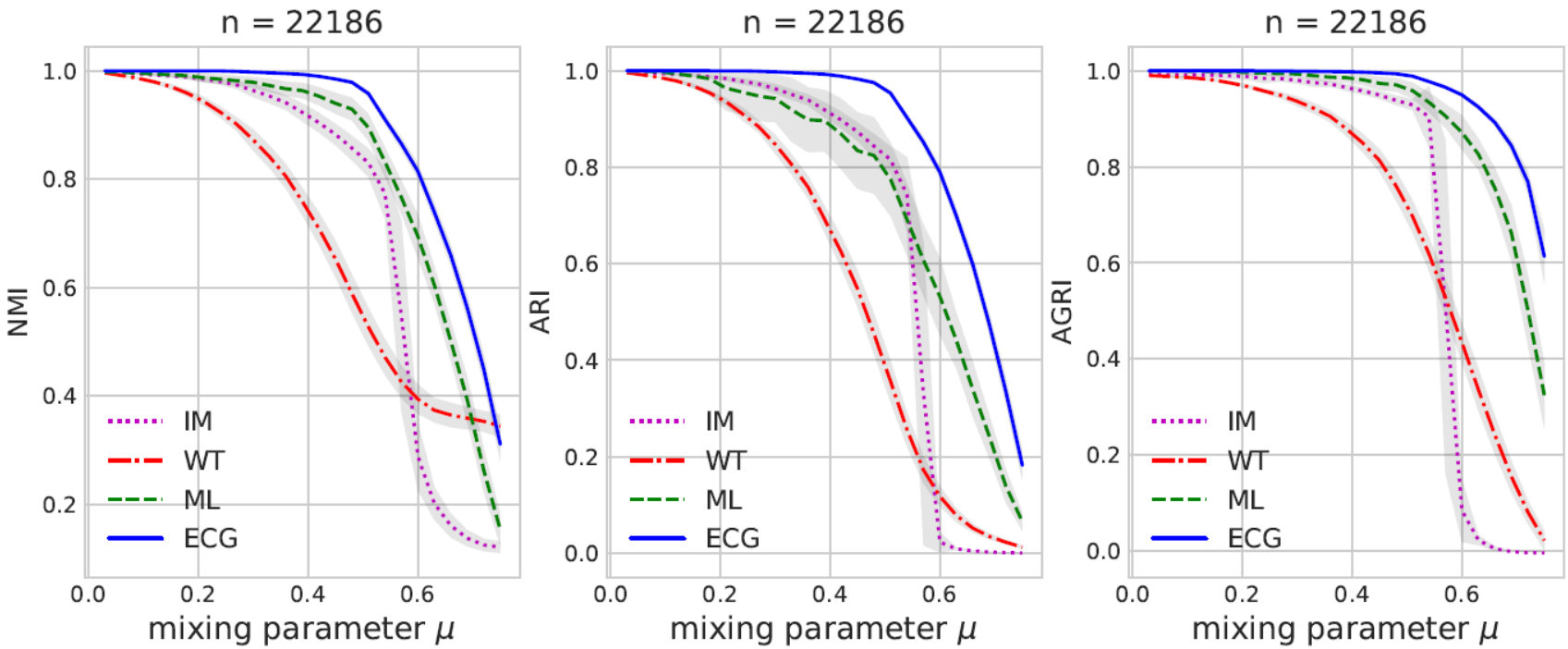


Figure 2 In each plot, we consider l cliques of size $m = 5$ where contiguous cliques are linked by 1 to 5 edges, respectively. We compare the number of communities found by the InfoMap (IM), Louvain (ML) and ECG algorithms. The resolution limit phenomenon is clearly seen with the ML algorithm. The IM algorithm fails to find the right number of communities when we increase the number of edges between the cliques, while ECG remains more stable.

Comparison Study over LFR Graphs

LFR graphs with 22,186 nodes:



Roadmap

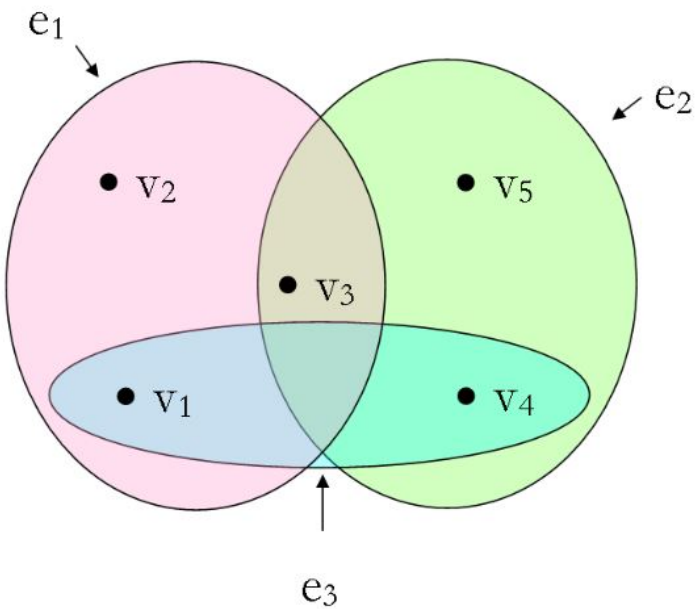
1. Part I
   - Background material
   - Relational data and graphs
   - Random graph models
2. Part II
   - Measures of centrality
   - Degree correlation
   - Graph (vertex) embedding
   - *First notebook*
3. Part III
   - Community detection
   - ***Second notebook***
   - Hypergraphs and other topics

Roadmap

1. Part I
   - Background material
   - Relational data and graphs
   - Random graph models
2. Part II
   - Measures of centrality
   - Degree correlation
   - Graph (vertex) embedding
   - *First notebook*
3. Part III
   - Community detection
   - *Second notebook*
   - **Hypergraphs and other topics**

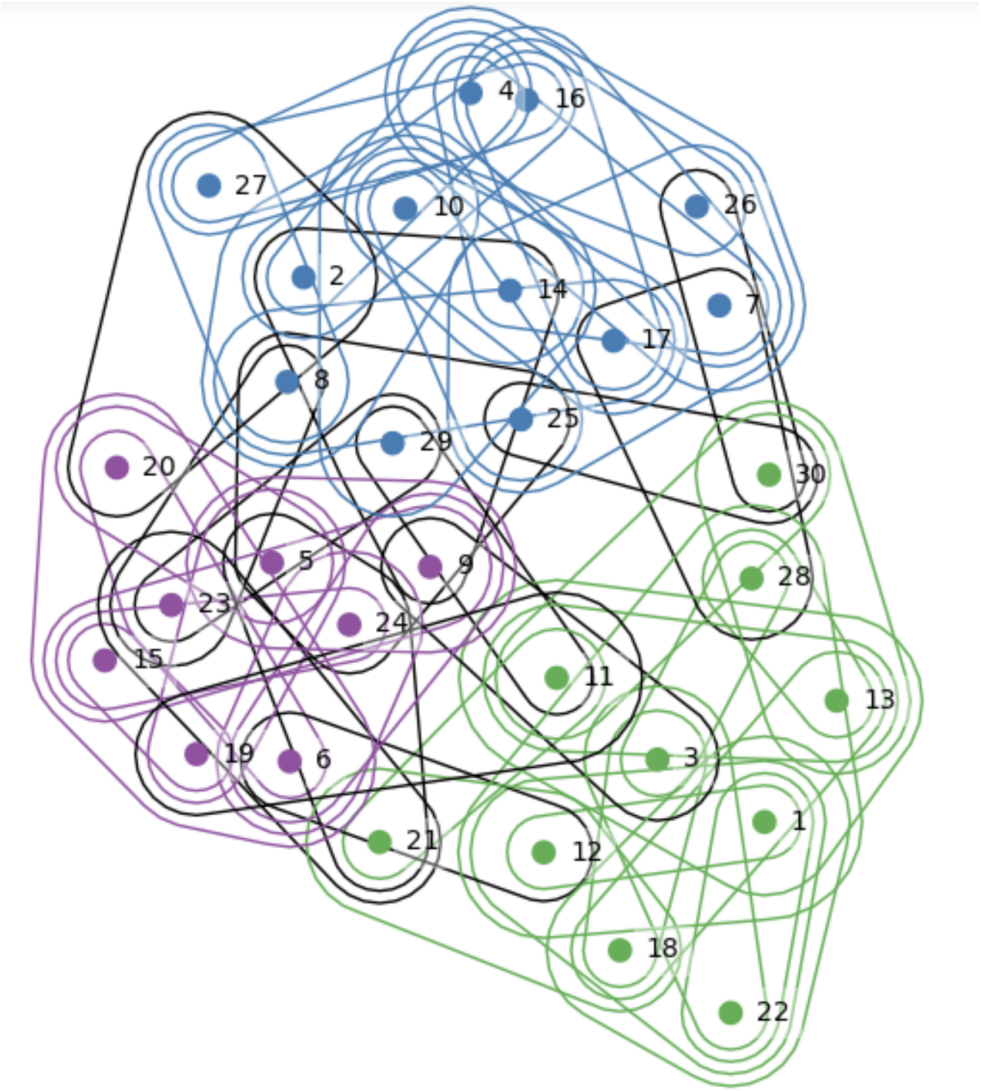
Hypergraphs

- $H = (V, E)$ where $|V| = n$, $|E| = m$
- $e \in E$: (hyper)-edges where $e \subseteq V$, $|e| \geq 2$
- Edges can have weights
- We consider undirected hypergraphs



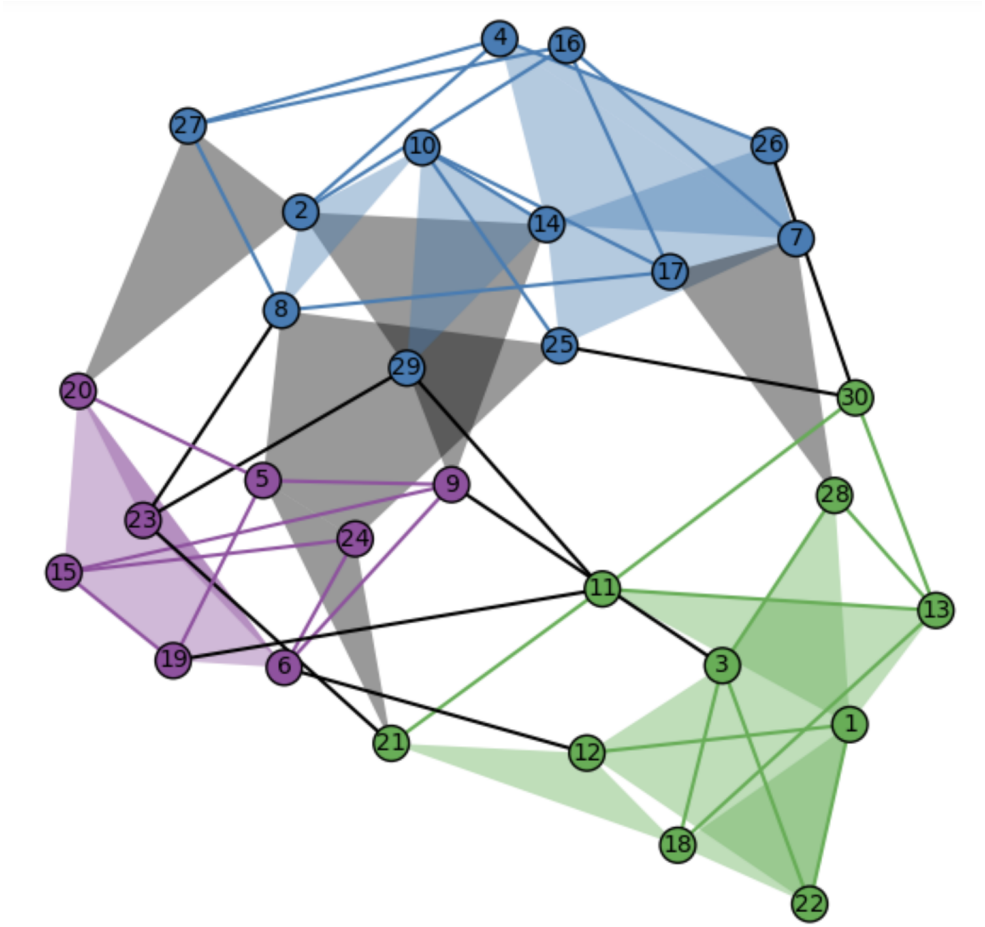
Hypergraphs

Euler diagram visualization (3 communities)



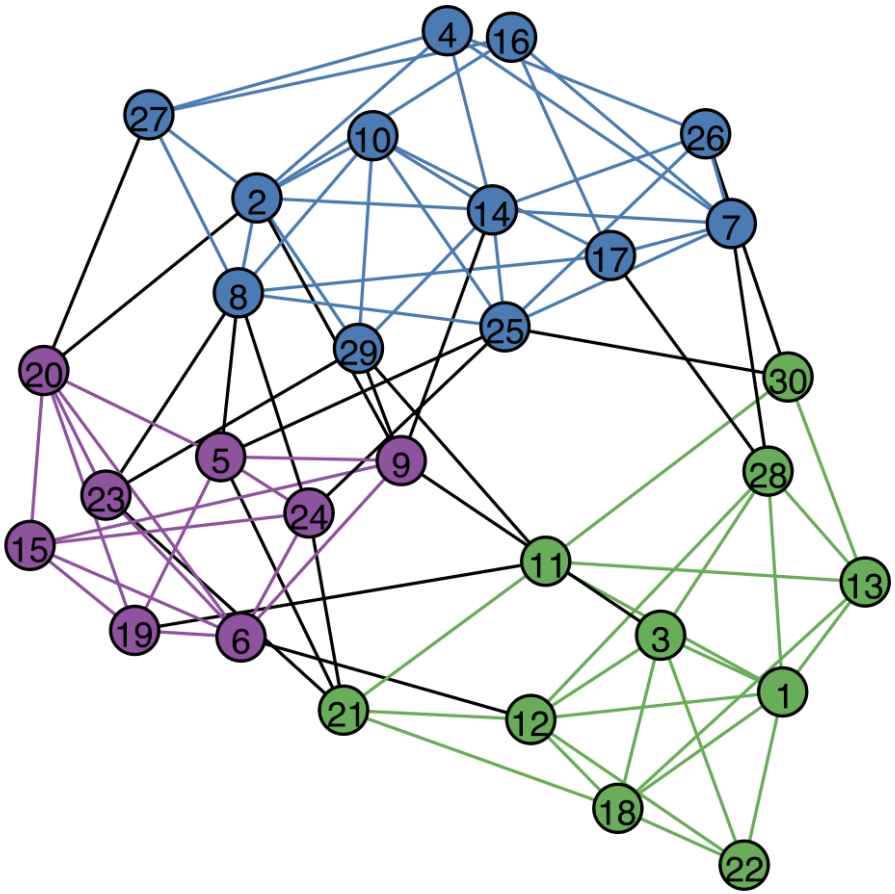
Hypergraphs

Simplicial complex like visualization



Hypergraphs

$G = H_{[2]}$ (2-section graph)



Why Hypergraphs?

Some data are better modeled with hypergraphs, such as:

- email exchanges
- posting in same thread
- tracking co-locations
- categorical data modeling

Hypergraphs

- Few hypergraph-based algorithms exist in data science
- They are typically slower
- Some have an equivalent graph representation

Graph modularity

(q) Can we define a modularity function on hypergraphs?

Recall the modularity of a partition $\mathbf{A}$ of graph G as:

$$q_G(\mathbf{A}) = \frac{1}{|E|} \sum_{A_i \in \mathbf{A}} \left(e_G(A_i) - \mathop{\mathbb{E}}_{G' \in \mathcal{G}} (e_{G'}(A_i)) \right)$$

$e_G(A_i) = |\{e \in E : e \subseteq A_i\}|$ is the *edge contribution* and, $\mathbb{E}_{G' \in \mathcal{G}}(e_{G'}(A_i))$ is the *degree tax*.

The degree tax is the expected value of the edge contribution term over the graphs $G' \sim \mathcal{CL}(G)$, the Chung-Lu model.

Chung-Lu Model for Hypergraphs

Consider a hypergraph $H = (V, E)$ with $V = \{v_1, \ldots, v_n\}$.

Let $E = \bigcup E_d$ with E_d the edges of size d.

Let $deg(v_i)$ the node degrees and $vol(V) = \sum_{v_i \in V} deg(v_i)$.

Let $p_i = deg(v_i)/vol(V)$.

For the Hypergraph Chung-Lu model ($\mathcal{H}$), we generate an edge of size d via a **multinomial experiments** with d trials where we select vertex v_i with probability p_i.

We generate $|E_d|$ edges of size d, the number of such edges in H.

Nodes may appear more than once in an edge (generalization of loops).

Chung-Lu Model for Hypergraphs

Let $\mathcal{H}$, the hypergraph Chung-Lu model; we can show that:

$$\mathbb{E}_{H' \sim \mathcal{H}}[deg_{H'}(v_i)] = \sum_{d \geq 2} \frac{d \cdot |E_d| \cdot deg(v_i)}{vol(V)} = deg(v_i)$$

We use this generalization of the Chung-Lu model as our null model (*degree tax*) to define hypergraph modularity.

Hypergraph Modularities

Let $H = (V, E)$ and $\mathbf{A} = \{A_1, \ldots, A_k\}$, a partition of V.
Several definitions can be used to quantify the *edge contribution* given **A**, such as:

(a) all vertices of an edge have to belong to one of the parts to contribute; this is a **strict** definition;

(b) the **majority** of vertices of an edge belong to one of the parts.

Hypergraph Modularities

Let $e_H^{d,c}(A_i)$, the number of hyperedges of size d that have exactly c members in A_i.

$$q_H^{d,c}(\mathbf{A}) = \frac{1}{|E|} \sum_{A_i \in \mathbf{A}} \left(e_H^{d,c}(A_i) - \mathbb{E}_{H' \sim \mathcal{H}}[e_{H'}^{d,c}(A_i)] \right)$$

We get a **general definition for the hypergraph modularity** given *hyper-parameters* $w_{d,c} \in [0, 1]$:

$$q_H(\mathbf{A}) = \sum_{d \geq 2} \sum_{c=\lfloor d/2 \rfloor + 1}^{d} w_{d,c}\, q_H^{d,c}(\mathbf{A}).$$

Hypergraph Modularities

This definition is flexibile. For example the strict variant can be obtained as follows:

$$q_H^s(\mathbf{A}) = \sum_{d \geq 2} q_H^{d,d}(\mathbf{A}),$$

that is, only the slices with $c = d$ are considered.

Clustering Hypergraphs

$\mathbf{G = H_{[2]}}$

Consider hypergraph $H = (V, E)$

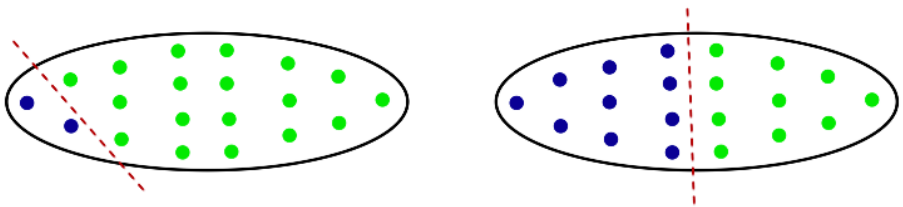
To partition V, one may consider its 2-section graph $G = H_{[2]}$

Algorithms such as **Louvain**, **Leiden** or **ECG** can then be applied

Kumar *et.al.* (2019) proposed a refinement to the Louvain algorithm for hypergraphs

- build weighted $G = H_{[2]}$ and run Louvain
- re-weight edges in H based on their measure of homogeneity between parts
- repeat until convergence

Re-weighting favours *purer* edges:



$$t = \left(\frac{1}{2} + \frac{1}{18}\right) \times 20 = 11.111$$

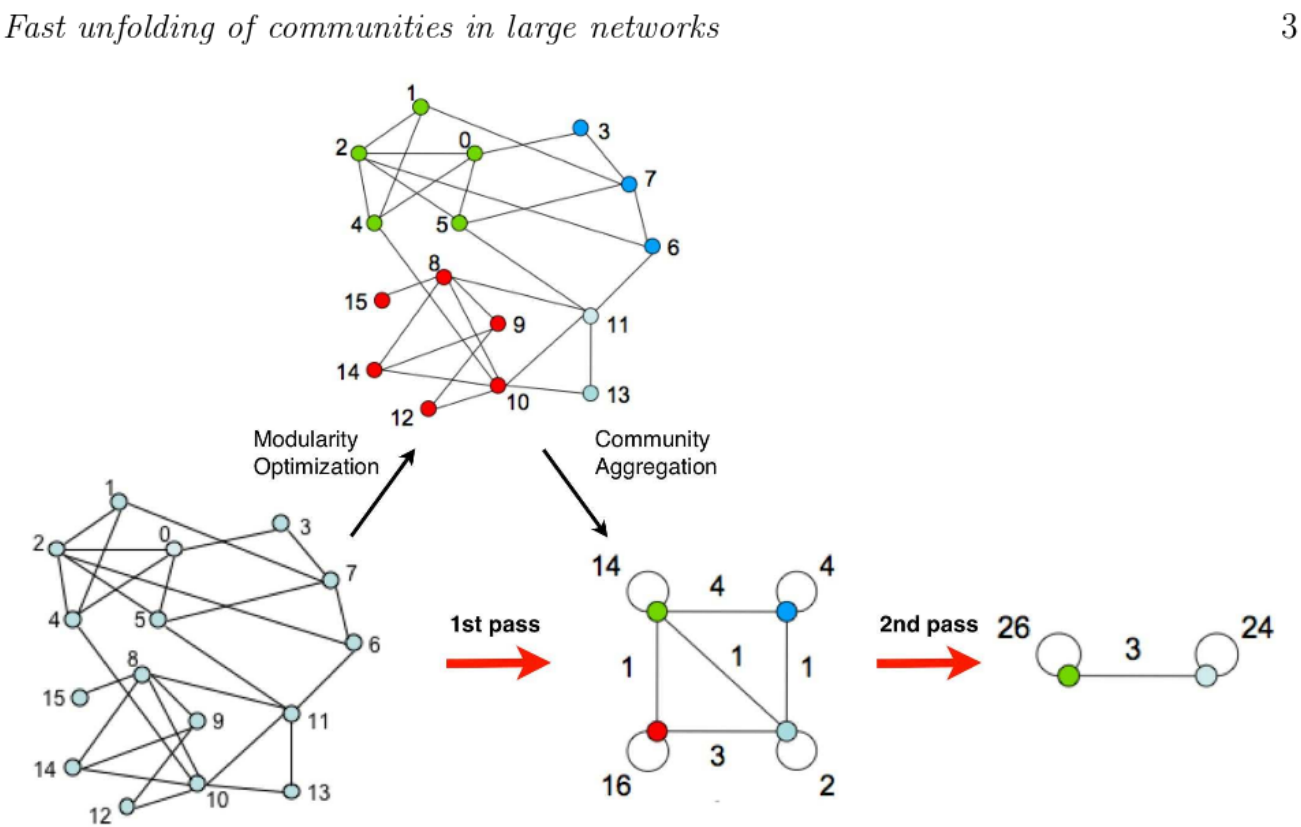
$$t = \left(\frac{1}{10} + \frac{1}{10}\right) \times 20 = 4$$

Previous algorithm are mainly graph-based

HMLL (*Hypergraph Maximum Likelihood Louvain*), Chodrow *et al. (2021)*:

- Defined for several choices of hypergraph modularity including our *strict* one
- Louvain algorithm, but moving sets of nodes rather than collapsing for phases 2+
- Can not *lift from the ground* without small sized edges

This is easy with graphs



Ref: Blondel *et al.*, arXiv:0803.0476v2

This is one of the main issue with agglomerative hypergraph clustering:

- start with every **node** in its own cluster
- need **small sized edges** to start the agglomerative process
- **bias** toward small sized edges at the beginning of the process

A simple solution:

- alleviate the issue by forcing **small sized edges** via $G = H_{[2]}$ and the corresponding modulary objective $q_G()$;
- gradually move toward **hypergraph-based** objective $q_H()$.

The proposed solution:

$$q(\mathbf{A}, \alpha) := \alpha \cdot q_H(\mathbf{A}) + (1 - \alpha) \cdot q_{H_{[2]}}(\mathbf{A})$$

where $\alpha \in [0, 1]$.

- should we always increase α?
- initial value for α?
- when to update?
- by how much?

We use the **h-ABCD** benchmark for our initial experiments:

- power law degree and community size distribution
- arbitrary edge size distribution
- based on the configuration model
- k community hypergraphs and one “background” hypergraph (noise)
- github.com/bkamins/ABCDHypergraphGenerator.jl

Recall the objective function:

$$q(\mathbf{A}, \alpha) := \alpha \cdot q_H(\mathbf{A}) + (1 - \alpha) \cdot q_{H_{[2]}}(\mathbf{A})$$

- Start from $\alpha_0 = d$ (default to $d = 0$)
- Consider only non-decreasing sequences for α
- Switch value ($\alpha = \alpha_i$ for $i \geq 1$) when the number of communities reaches nc^i, (n nodes, $0 < c < 1$)
- Change to $\alpha_i = 1 - (1 - d)(1 - b)^i$, where $0 \leq b \leq 1$

Hypergraph Louvain

Grid of parameters

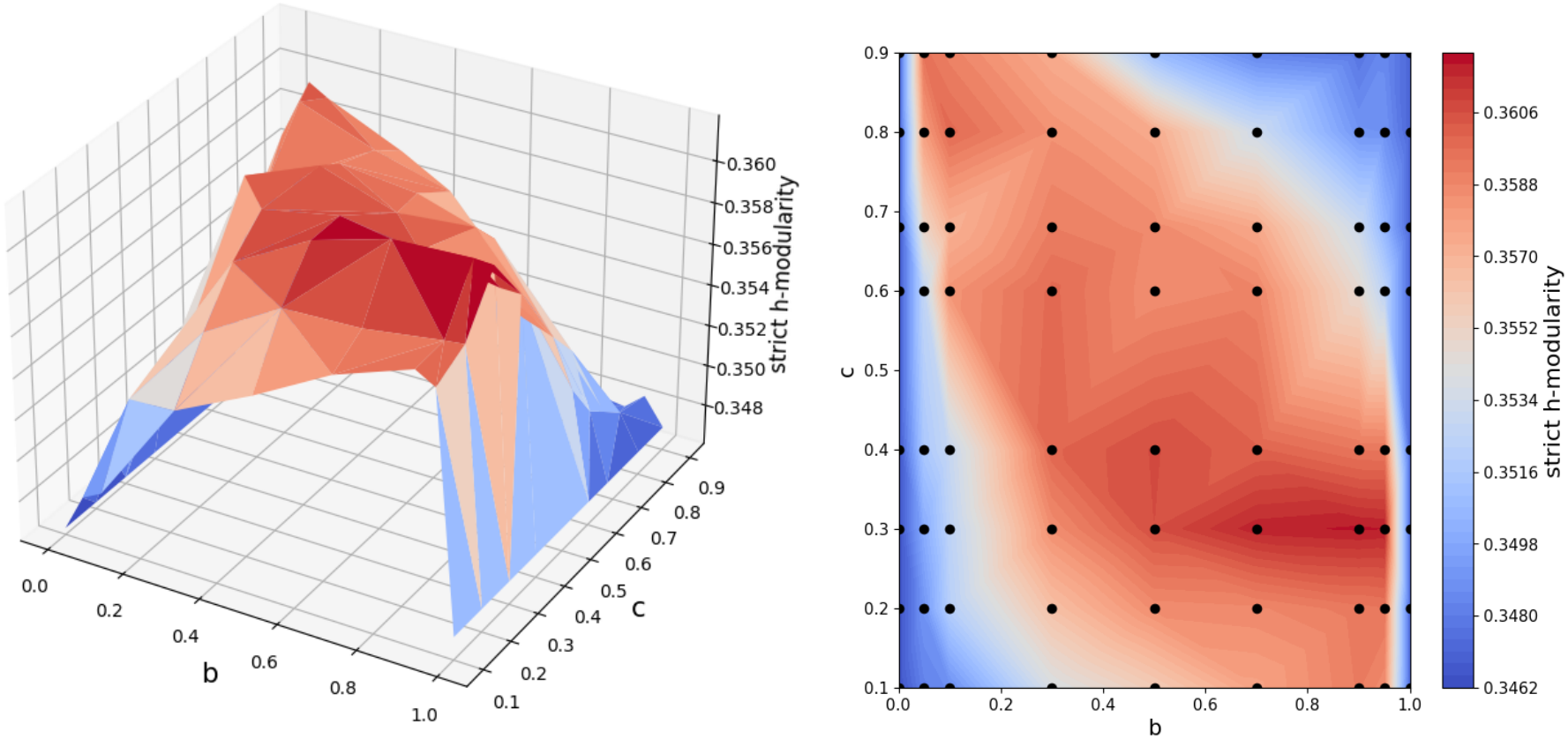
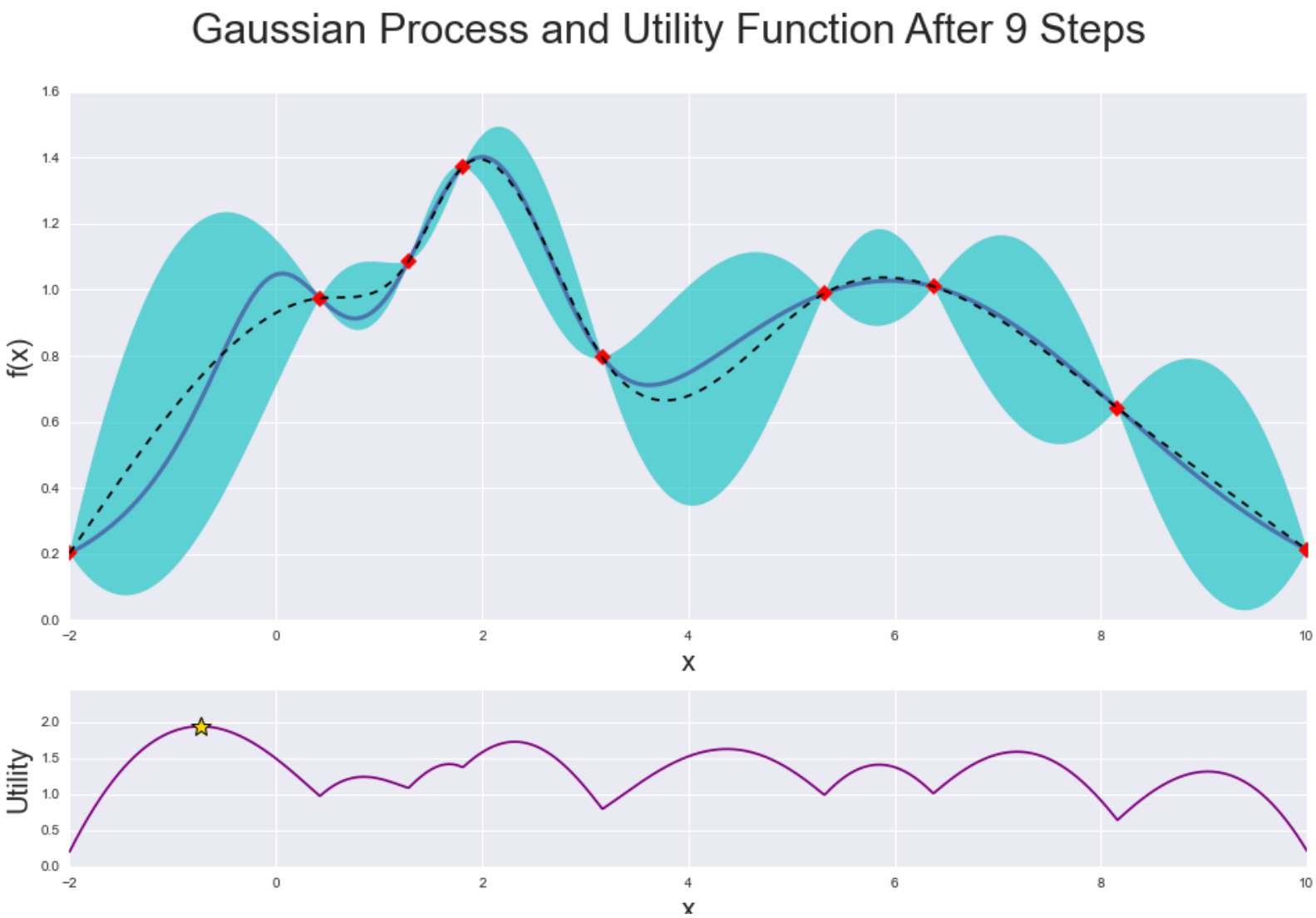


Figure: strict h-ABCD, large noise

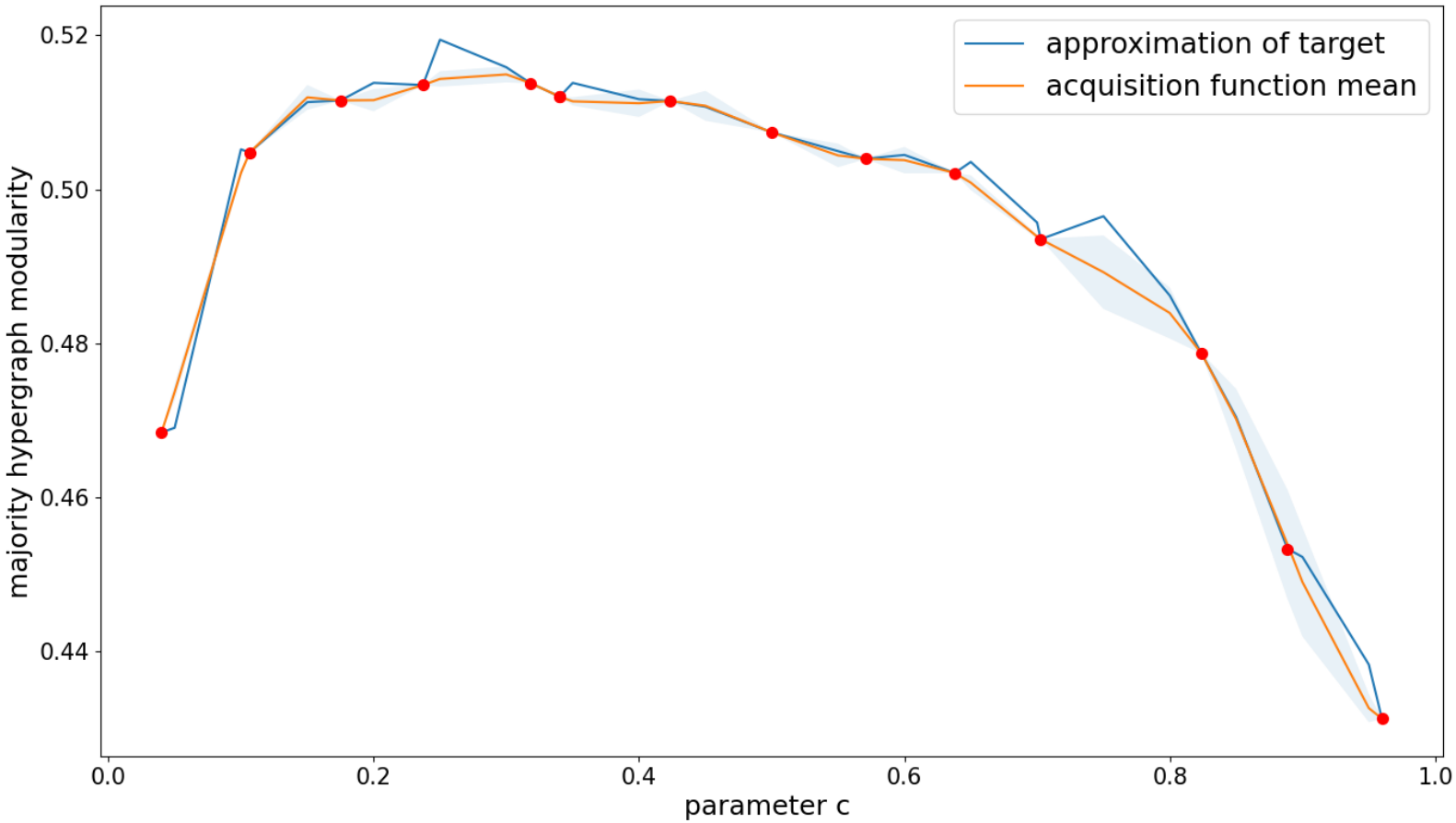
Hypergraph Louvain

Bayesian optimization to set parameters

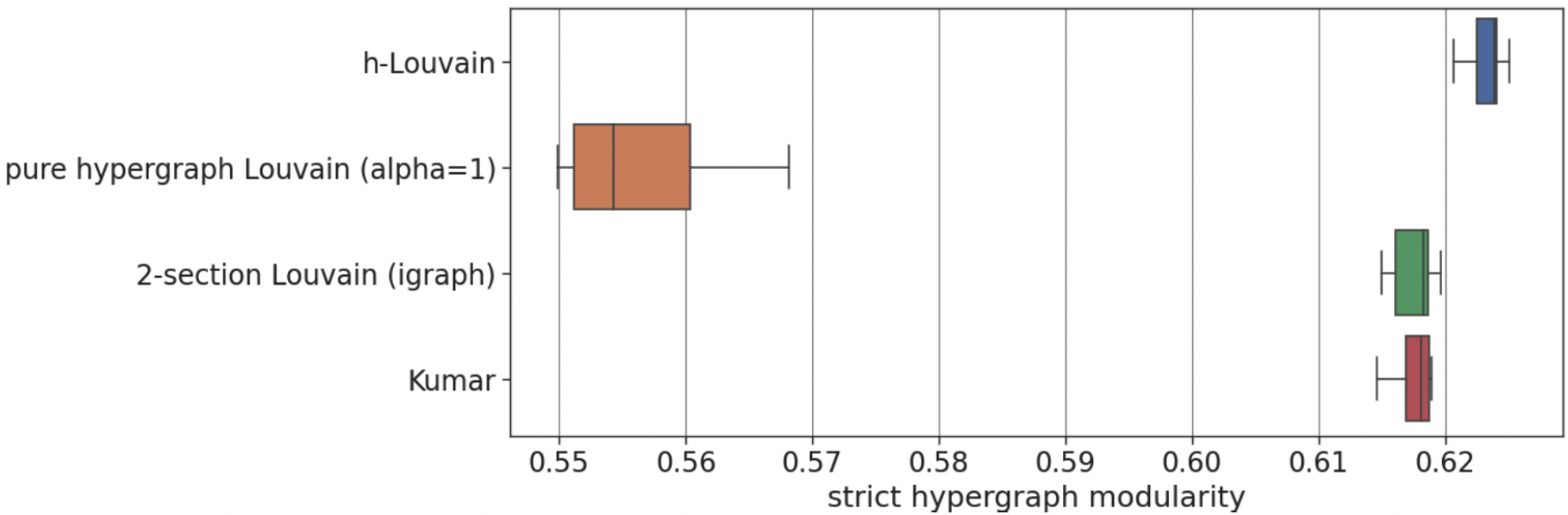


from: github.com/bayesian-optimization/BayesianOptimization

Example: optimizing parameter *c*



ex: h-ABCD hypergraphs with *strict* community edges:



Next steps

- Finalize the “auto-configurable” h-Louvain algorithm
- Test on several flavours of the h-ABCD benchmark
- Test on real hypergraphs and compare with other algorithms

Illustration: GoT hypergraph

Data from github.com/jeffreylancaster/game-of-thrones
Nodes: named characters
Edges: groups of characters in a scene (weight=duration).

Top characters:

	name	degree	strength
11	Jon Snow	306	39221
56	Tyrion Lannister	263	39899
10	Daenerys Targaryen	220	30644
5	Cersei Lannister	184	24981
43	Sansa Stark	183	25009

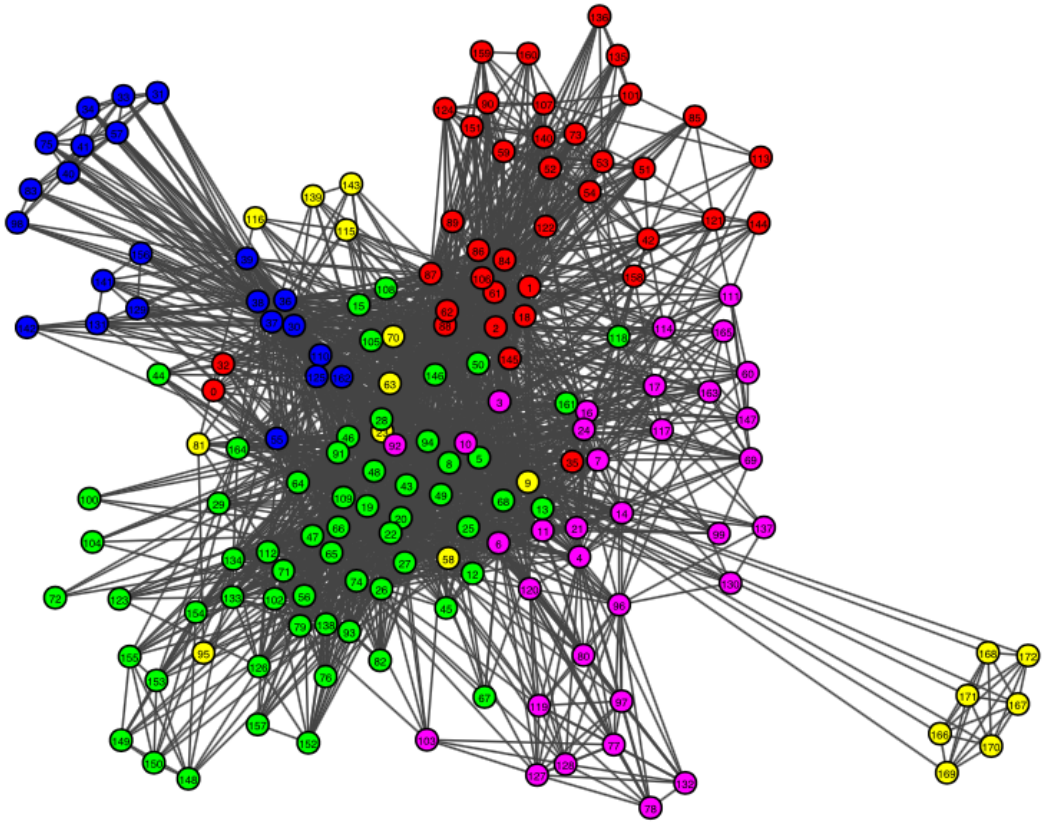
Illustration: GoT hypergraph

Example: top characters in same (hypergraph-based) cluster as Daenerys Targaryen:

	name	degree	strength	betweenness	pagerank	cluster
10	Daenerys Targaryen	220	30644	0.117762	0.031988	2
14	Jorah Mormont	118	19344	0.005032	0.020308	2
57	Missandei	92	13683	0.002482	0.013900	2
58	Grey Worm	79	10416	0.015164	0.010989	2
104	Barristan Selmy	35	6514	0.000000	0.007237	2
9	Drogon	84	6396	0.092749	0.007395	2
164	Daario Naharis	32	5370	0.000000	0.005907	2
15	Rhaegal	49	3932	0.125322	0.004902	2
137	Khal Drogo	14	3575	0.000000	0.004687	2

g

Illustration: GoT hypergraph



Yellow clique: Braavos theater group.

Packages for hypergraphs

With Python:
github.com/pnnl/HyperNetX

github.com/xgi-org/xgi

With Julia:
github.com/pszufe/SimpleHypergraphs.jl

Other applications

There are several topics we did not touch such as:

- overlapping community detection
- embedding edges, whole graphs
- semi-supervised learning
- anomaly detection
- graph robustness
- road networks
- dynamic graphs

Thanks for your attention!

theberge@ieee.org

Slides and notebooks:
github.com/ftheberge/CMS2023_MiniCourse

CRC textbook: www.torontomu.ca/mining-complex-networks

companion notebooks:
github.com/ftheberge/GraphMiningNotebooks

More references:
github.com/ftheberge/Graph_and_Hypergraph_References