Modularity and Stochastic Block Modelling

Enrico Caprioglio
University of Sussex

Network Science Lecture 6, Spring 2025

Some Key Objectives

1. why null models are used (last week lecture's recap)
2. learn about modularity and the Stochastic Block Model
3. compare community detection approaches
4. introduction to Network Inference

Last week's recap

Why do we use network Null Models?

Last week's recap

Why do we use network Null Models?

"Null models are a flexible tool to statistically benchmark the presence or magnitude of features of interest, by selectively preserving specific architectural properties of (brain) networks while systematically randomizing others." [1]

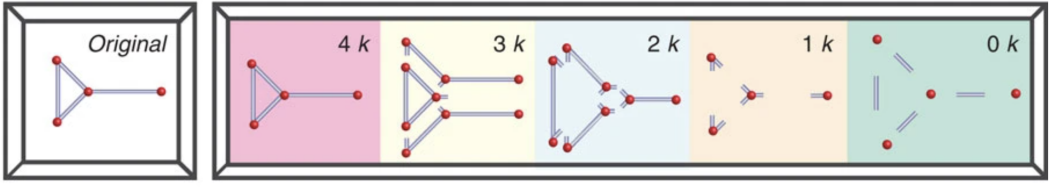
Why do we use network Null Models?

This is at the core of network science:

- we accept the complexity of a system
- we try to represent this complexity as a network
- we hypothesize that a specific collective property plays a role in the function of the system
- we use network null models to test if the property of interest is non-trivial!

Why do we use network Null Models?

Recall **D-k Randomization** [2]



Why do we use network Null Models?

Recall **D-k Randomization** [2]

"A common belief is that a self-organizing system should evolve to a network structure that makes these dynamical processes, or network functions, efficient. If this is the case, then given a real network, we may 'reverse engineer' it by showing that its structure optimizes its function. In that respect the problem of interdependency between different network properties becomes particularly important" [2]

Last week's recap

Why do we use network Null Models?

In other words,

we use null models to make sure that the property we are studying is nontrivial!

we will shortly see some good examples of this . . . [1]

[1] At some point I will try to trick you!

Conceptual Summary (*i.e.*, no maths)

So far, we have seen:

- **Random networks models** (Gilbert and Erdős–Rényi models)

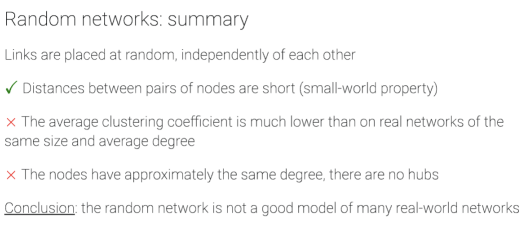


Figure: Luc's summary

Conceptual Summary (*i.e.*, no maths)

So far, we have seen:

- **Random networks models** (Gilbert and Erdős–Rényi models)
- **Small-world models** (Watts-Strogatz model)

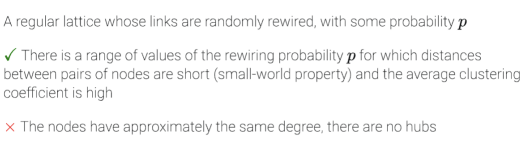


Figure: Luc's summary

Conceptual Summary (*i.e.*, no maths)

So far, we have seen:

- **Random networks models** (Gilbert and Erdős–Rényi models)
- **Small-world models** (Watts-Strogatz model)
- **Preferential atttachment models** (Polya's urn and variations of Barabási-Albert models)

Conceptual Summary (*i.e.*, no maths)

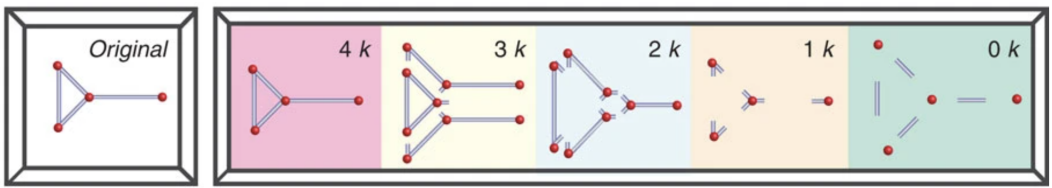
So far, we have seen:

- **Random networks models** (Gilbert and Erdős–Rényi models)
- **Small-world models** (Watts-Strogatz model)
- **Preferential atttachment models** (Polya's urn and variations of Barabási-Albert models)
- **Configuration model**
- **Exponential random graphs** (ERG), we will get back to this later

The models discussed so far (except for ERG), focus on "local" network properties.

The models discussed so far (except for ERG), focus on "local" network properties.

The D-k randomization technique is also a good example of this:



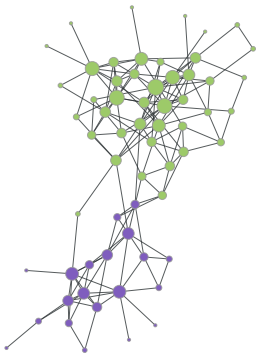
The models discussed so far (except for ERG), focus on "local" network properties.

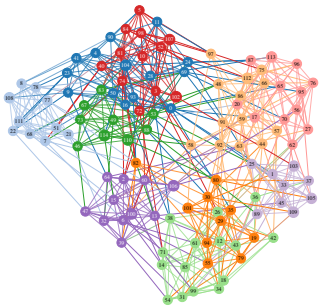
What is a simple example of "non-local" network property?

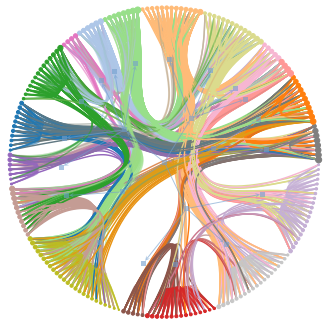
The models discussed so far (except for ERG), focus on "local" network properties.

What is a simple example of "non-local" network property?

Modularity







Modularity and Community Detection

Modularity seems like an intuitive concept, but it is not so easy to define.
One possible "general" definition for modules (or clusters or communitites) is:

Modules

groups of vertices that have higher probability of being connected to each other than to other vertices [3]

and a network is said to have a modular structure if it has these modules.

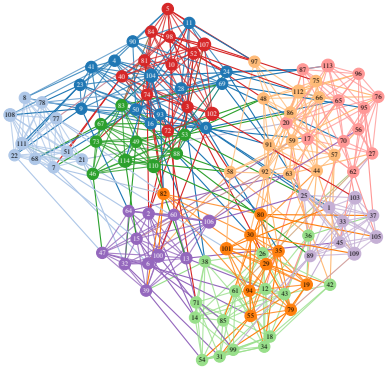


Figure: Network of American football teams [4]

Examples

Most real world networks display a modular structure:

- Sociology $\longrightarrow$ Homophily Principle [5, 6]

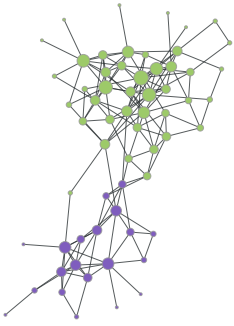


Figure: Dolphin's social network [7]

Examples

Most real world networks display a modular structure:

- Sociology $\longrightarrow$ Homophily Principle [5, 6]
- Cybernetics $\longrightarrow$ "The architecture of Complexity " [8]

"Hierarchy, I shall argue, is one of the central structural schemes that the architect of complexity uses. ([. . .])
There once were two watchmakers, named Hora and Tempus, . . . " [8]

Examples

Most real world networks display a modular structure:

- Sociology $\longrightarrow$ Homophily Principle [5, 6]
- Cybernetics $\longrightarrow$ "The architecture of Complexity " [8]
- Neuroscience [9], biology and so on

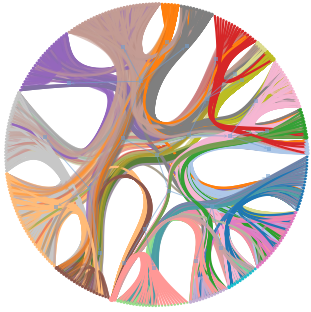


Figure: Hierarchically Modular Structure of the c-elegans brain

Examples

Most real world networks display a modular structure:

- Sociology $\longrightarrow$ Homophily Principle [5, 6]
- Cybernetics $\longrightarrow$ "The architecture of Complexity " [8]
- Neuroscience [9], biology and so on

How to identify modules:

$\longrightarrow$ Community Detection

1. **Modularity Optimization**
2. **Statistical (Bayesian) Inference**

Examples

Most real world networks display a modular structure:

- Sociology $\longrightarrow$ Homophily Principle [5, 6]
- Cybernetics $\longrightarrow$ "The architecture of Complexity " [8]
- Neuroscience [9], biology and so on

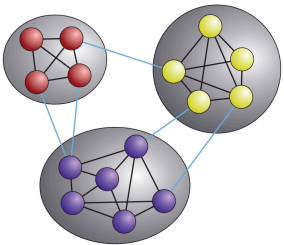
How to identify modules:

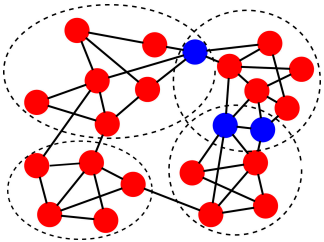
$\longrightarrow$ Community Detection

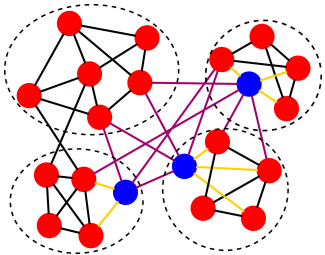
1. **Modularity Optimization**
2. **Statistical (Bayesian) Inference**
3. *Spectral Methods*
4. *Dynamical Methods*
5. *And many more [10]*

Strong and Weak Communities [10] (and Ref. within)

- **Strong community**: a subgraph G_r such that the internal degree k^{in} of each vertex i is greater than its external degree k^{out} $\quad k_i^{\text{in}}(G_r) > k_i^{\text{out}}(G_r)$
- **Weak community**: the internal degree of the subgraph exceeds its external degree $\quad \sum_{i \in G_r} k_i^{\text{in}}(G_r) > \sum_{i \in G_r} k_i^{\text{out}}(G_r)$







Strong and Weak Communities [10] (and Ref. within)

UNIVERSITY OF SUSSEX

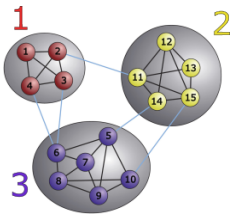
Modern Definitions:

- **Strong community**: a subgraph each of whose vertices has a higher probability to be linked to every vertex of the subgraph than to any other vertex of the graph
- **Weak community**: a subgraph such that the average edge probability of each vertex with the other members of the group exceeds the average edge probability of the vertex with the vertices of any other group

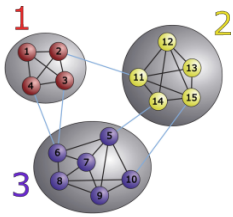
Assuming we have a good definition for the probability of an edge

Modularity Index

- *Assume* we have found a **partition** of the network $\mathbf{b} = \{1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2\}$
- Recall the **configuration model**, our degree sequence is $\{k_i\} = \{3, 4, 4, 4, 5, 5, 4, 4, 4, 4, 5, 4, 3, 4, 5\}$.

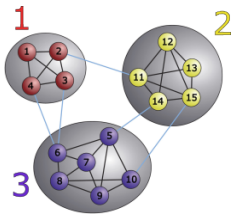


Modularity Index



- *Assume* we have found a **partition** of the network $\mathbf{b} = \{1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2\}$
- Recall the **configuration model**, our degree sequence is $\{k_i\} = \{3, 4, 4, 4, 5, 5, 4, 4, 4, 4, 5, 4, 3, 4, 5\}$.
- the number of edges between nodes in the same group is $\frac{1}{2}\sum_{i,j} A_{ij}\delta_{b_i,b_j}$
- the total expected number of edges between nodes i and j in the same group is $\frac{1}{2}\sum_{ij} \frac{k_i k_j}{2L}\delta_{b_i,b_j}$, where L is the total number of edges (note the double $1/2$).

Modularity Index



- *Assume* we have found a **partition** of the network $\mathbf{b} = \{1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2\}$
- Recall the **configuration model**, our degree sequence is $\{k_i\} = \{3, 4, 4, 4, 5, 5, 4, 4, 4, 4, 5, 4, 3, 4, 5\}$.
- the number of edges between nodes in the same group is $\frac{1}{2}\sum_{i,j} A_{ij}\delta_{b_i,b_j}$
- the total expected number of edges between nodes i and j in the same group is $\frac{1}{2}\sum_{ij} \frac{k_i k_j}{2L}\delta_{b_i,b_j}$, where L is the total number of edges (note the double $1/2$).

Modularity Index

$$Q = \frac{1}{2L}\sum_{ij}\left[A_{ij} - \frac{k_i k_j}{2L}\right]\delta_{b_i,b_j}$$

For the network above $Q \approx 0.5$
(rule of thumb: $Q > 0.3$ is considered modular)

Modularity Index (Python Example)

UNIVERSITY OF SUSSEX

```
def _my_modularity(A, b):

    N = np.shape(A)[0]
    L = np.sum(A) / 2
    k_seq = _my_degree_sequence(A)

    Q = 0
    for i in range(N):
        for j in range(N):
            if b[i] == b[j]: # \delta_{b_{i}}\delta_{b_{j}} = 1
                if b[i] == b[j]
                 Q += A[i,j] - k_seq[i]*k_seq[j]/(2*L)

    return np.sum(Q) / (2*L)
```

Modularity Index (Python Example)

UNIVERSITY OF SUSSEX

or more simply, using networkx

```
from networkx.algorithms.community.quality import modularity

communities = [np.where(b == i)[0] for i in range(len(np.unique(
    b)))]

modularity(g, communities)
```

NOTE: networkx uses a list of lists, in which each list specifies the nodes (labels) within a module, rather than b. Above, I converted these two equivalent descriptions.

Community Detection: Modularity Maximisation

Goal: find the partition $\boldsymbol{b}$ that maximises the modularity index Q (Similar concept to k-means clustering.)

Brute force approach example:

- loop through each possible partition of a network
- compute Q for each partition
- select the partition with the highest Q

easy right?

Goal: find the partition $\boldsymbol{b}$ that maximises the modularity index Q (Similar concept to k-means clustering.)

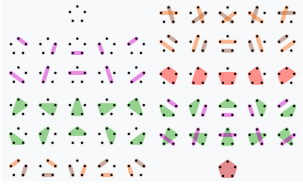
Brute force approach example:

- loop through each possible partition of a network
- compute Q for each partition
- select the partition with the highest Q

easy right?

This approach is indeed crazy:

the number of partitions of a network of size N is given by the Bell number a_n.



For instance, $a(20) = 51724158235372\ldots$

Community Detection: Modularity Maximisation

There are of course better ways to do this (greedy optimization):

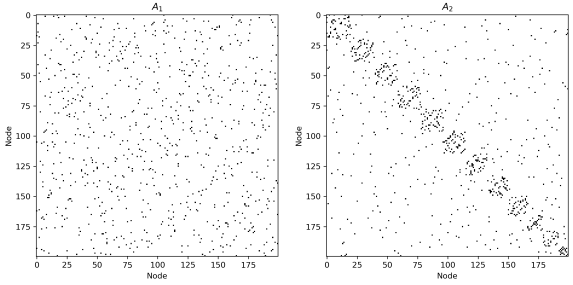
- Clauset, Newman and Moore greedy optimization algorithm: [11]

```
from networkx.algorithms.community import
    greedy_modularity_communities
```

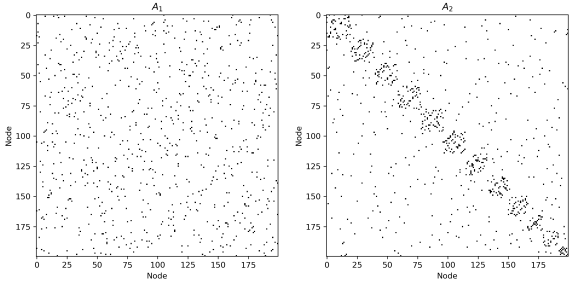
- Louvain Algorithm [12] (similar but faster):

```
from networkx.algorithms.community import
    louvain_communities
```

Community Detection: Modularity Maximisation Example

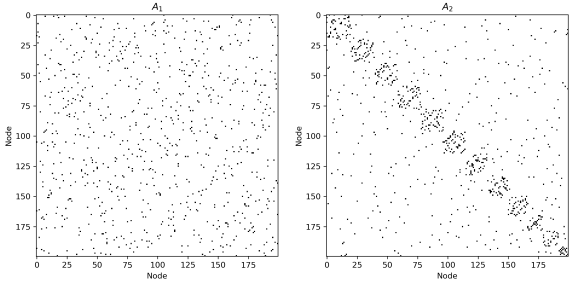


Community Detection: Modularity Maximisation Example



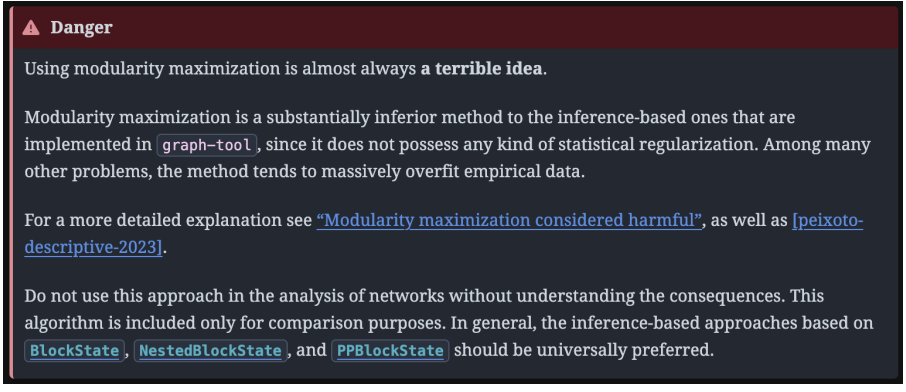
Using Louvain, best partition gives $Q = 0.5003 \longrightarrow$ for both of them!

Community Detection: Modularity Maximisation Example



Using Louvain, best partition gives $Q = 0.5003 \longrightarrow$ for both of them!
They are actually the same exact network, generate using a **configuration model** with degree sequence $k_i = 4, \forall i$ (same as a **regular random graph** with $d = 4$). The trick is to sort nodes using the partition from the Louvain algorithm! (revisit slide no. 3)

Community Detection: Modularity Maximisation Example



Danger

Using modularity maximization is almost always **a terrible idea**.

Modularity maximization is a substantially inferior method to the inference-based ones that are implemented in `graph-tool`, since it does not possess any kind of statistical regularization. Among many other problems, the method tends to massively overfit empirical data.

For a more detailed explanation see "Modularity maximization considered harmful", as well as [peixoto-descriptive-2023].

Do not use this approach in the analysis of networks without understanding the consequences. This algorithm is included only for comparison purposes. In general, the inference-based approaches based on `BlockState`, `NestedBlockState`, and `PPBlockState` should be universally preferred.

Figure: From graph-tool's ModularityState documentation

New Goal:
Community detection without overfitting

New Goal:
Community detection without overfitting

Possible Solution:

- define a null model for modular networks: the **Stochastic Block Model** (SBM).
- do **Bayesian inference**

Stochastic Block Model [13, 14]

Intuition: think of modular networks as particular representations of a latent similarity space.

SBM:

- given the partition of N nodes into B blocks:

$$\boldsymbol{b} = \{b_1, b_2, \ldots, b_B\}$$

- we construct a generative model that generates networks with a probability

$$P(A|\boldsymbol{b})$$

In practice, there are two ways to do this. Both start by defining the number of blocks B and the number of nodes in each block $\boldsymbol{n} = \{n_1, n_2, \ldots, n_B\}$

In practice, there are two ways to do this. Both start by defining the number of blocks B and the number of nodes in each block $\boldsymbol{n} = \{n_1, n_2, \ldots, n_B\}$

(i): Using the **expected** number of edges between block r and block s

$$\{e_{rs}\} \quad B \times B \text{ matrix}$$

$$e_{rs} = \sum_{ij} A_{ij} \delta_{b_i,r} \delta_{b_j,s}$$

Convention is that if $r = s$ then e_{rs} is twice the number of edges

Stochastic Block Model [13, 14]

In practice, there are two ways to do this. Both start by defining the number of blocks B and the number of nodes in each block $\boldsymbol{n} = \{n_1, n_2, \ldots, n_B\}$

(i): Using the **expected** number of edges between block r and block s

$\{e_{rs}\}$ $B \times B$ matrix

$$e_{rs} = \sum_{ij} A_{ij} \delta_{b_i,r} \delta_{b_j,s}$$

Convention is that if $r = s$ then e_{rs} is twice the number of edges

(ii): Using the probability (density) of connections between blocks

$\{p_{rs}\}$ $B \times B$ matrix

Stochastic Block Model [13, 14]

In practice, there are two ways to do this. Both start by defining the number of blocks B and the number of nodes in each block $\boldsymbol{n} = \{n_1, n_2, \ldots, n_B\}$

(i): Using the **expected** number of edges between block r and block s

$$\{e_{rs}\} \quad B \times B \text{ matrix}$$

$$e_{rs} = \sum_{ij} A_{ij} \delta_{b_i,r} \delta_{b_j,s}$$

Convention is that if $r = s$ then e_{rs} is twice the number of edges

(ii): Using the probability (density) of connections between blocks

$$\{p_{rs}\} \quad B \times B \text{ matrix}$$

$$\text{if } r \neq s \quad p_{rs} = \frac{e_{rs}}{n_r \times n_s}$$

$$\text{if } r = s \quad p_{rs} = \frac{1}{2} \frac{e_{rs}}{\binom{n_r}{2}}$$

Stochastic Block Model: Example

UNIVERSITY OF SUSSEX

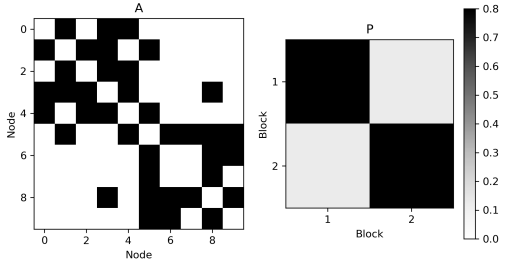
```
import math
random.seed(8) # so you can replicate this exactly
n = np.array([5,5])
N = sum(n) # total no. of nodes
e = np.array([[16,3],[3,16]]) # double counting within block edges
p = _p_from_e(e,n)
g = nx.stochastic_block_model(n, p)
```

```
# just one way to do this
def _p_from_e(e,n):
    p = np.zeros(e.shape)
    for i in range(len(n)):
        for j in range(i,len(n)):

            if j == i:
                L_max = math.comb(n[i], 2) * 2
            else:
                L_max = n[i] * n[j]

            p[i,j] = e[i,j] / L_max
            p[j,i] = p[i,j]

    return p
```



Stochastic Block Model: Example

We can of course generate much larger and interesting graphs:

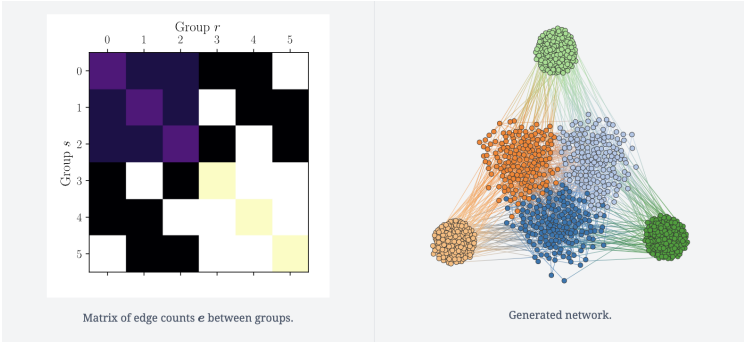
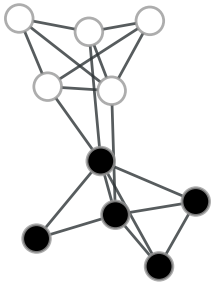


Figure: Example using graph-tool (from graph-tool documentation)

Stochastic Block Model: Example

Let's count the number of possible networks Ω that satisfy our constraints.

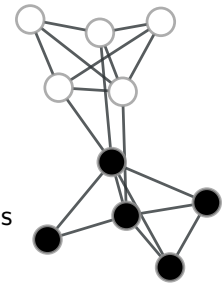


Stochastic Block Model: Example

Let's count the number of possible networks Ω that satisfy our constraints.
To do this we can compute

- the number of possible edge configurations **within** each block is

$$\Omega_{11} = \Omega_{22} = \binom{\binom{5}{2}}{8} = 45$$



- the number of possible edge configurations **between** each block is

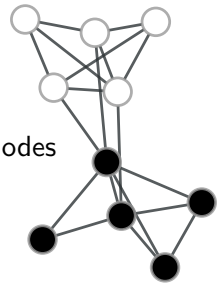
$$\Omega_{12} = \Omega_{21} = \binom{5 \times 5}{3} = 2300$$

Stochastic Block Model: Example

Let's count the number of possible networks Ω that satisfy our constraints.

- The total number is then

$$\Omega = \Omega_{11} \times \Omega_{12} \times \Omega_{22} = 4657500$$



- compare with the number of networks with the same number of nodes ($N = 10$) and edges ($E = 19$)

$$\Omega^\star = \binom{\binom{N}{2}}{19} = 2438362177020$$

6 orders of magnitude higher!

Stochastic Block Model: Example

The point is...

we need to think probabilistically

Extra [15]

Formally, we can use Ω to define the entropy S of a SBM, $S = \ln[\Omega]$, and obtain the log-likelihood function $\mathcal{L} = \ln \mathcal{P}$. If all networks are equally likely $\mathcal{P} = 1/\Omega \Rightarrow \mathcal{L} = -S$. Then, we can use this to infer the most likely block structure that matches our data.

Earlier, we generated modular networks using our generative model $P(A|\boldsymbol{b})$.
Now we wish to learn the best partition $\boldsymbol{b}$ after observing a real world network A, *i.e.*, $P(\boldsymbol{b}|A)$.

Using, $P(X \wedge Y) = P(X|Y)P(Y) = P(Y|X)P(X)$

Bayes' Rule

$$P(\boldsymbol{b}|A) = \frac{P(A|\boldsymbol{b})P(\boldsymbol{b})}{P(A)}$$

- $P(A|\boldsymbol{b})$ is the likelihood
- $P(\boldsymbol{b})$ is our prior
- $P(A)$ is the evidence

Bayesian Inference: Example using ER random graphs

Assume that we have just observed a network, and all we know (for sure) is that each edge is independent and has the same probability[2].

Goal: infer the value of p.

ER graphs can be generated by tossing a biased coin for each possible link amongst N nodes, so we have the likelihood $P(A|p)$:

$$P(A|p) = p^L(1-p)^{\binom{N}{2}-L}$$

To find the most likely p, we want to maximise $P(p|A)$ using by Bayes' rule.

[2] is this realistic?

The evidence $P(A)$ is a normalization constant, so we don't worry about it.

In this particular case, we can safely assume that our prior $P(p)$ is uniform and constant, so we don't worry about it.

$\rightarrow$ maximization of $P(p|A)$ is equivalent to the maximisation of $P(A|p)$

Differentiating with respect to p and setting to zero try it! we obtain $p = L/\binom{N}{2}$.

Extra

Note, we could have also maximised $\mathcal{L} = \log(P(A|p))$, since the $\log$ is a monotone increasing function of its argument. In the previous extra box we showed how this is related to the entropy of our stochastic block model. See also principle of maximum entropy. Since exponential distributions maximise the entropy, this should give you some intuition behind exponential random graphs and why they are useful! Indeed, the stochastic block model is part of this family of graphs.

Bayesian Inference: SBM

Goal: infer the most likely partition $\boldsymbol{b}$ of the observed network $A \longrightarrow$ maximise $p(\boldsymbol{b}|A)$, **while avoiding overfitting**.

Goal: infer the most likely partition $\boldsymbol{b}$ of the observed network $A \longrightarrow$ maximise $p(\boldsymbol{b}|A)$, **while avoiding overfitting**.

To do this we take very seriously the *principle of maximum indifference*, *i.e.*, we want to choose the most uninformative priors and a generative model (likelihood) with maximal entropy.

Minimum Description Length Σ

$$P(\boldsymbol{b}|A) = \frac{P(A|\boldsymbol{b})P(\boldsymbol{b})}{P(A)} = \frac{\exp(-\Sigma)}{P(A)}$$

where $\Sigma = -\ln P(A|\boldsymbol{b}) - \ln P(\boldsymbol{b})$

Goal: infer the most likely partition $\boldsymbol{b}$ of the observed network $A \longrightarrow$ maximise $p(\boldsymbol{b}|A)$, **while avoiding overfitting**.

The SBM $P(A|\boldsymbol{b})$ is the maximum entropy model for a network A with partition $\boldsymbol{b}$. By maximising the entropy $S = -\sum_A P(A|\boldsymbol{b}) \ln P(A|\boldsymbol{b})$ we can obtain (not shown[2])

$$P(A|p, \boldsymbol{b}) = \prod_{i<j} p_{b_i b_j}^{A_{ij}} (1 - p_{b_i b_j})^{1-A_{ij}}$$

This should make sense, remember our Stochastic Block Model: Example in slide 19.

[2] you're welcome! if you know the method of Lagrange multipliers check [13]

Goal: infer the most likely partition $\boldsymbol{b}$ of the observed network $A \longrightarrow$ maximise $p(\boldsymbol{b}|A)$, **while avoiding overfitting**.

Choosing the most uninformative prior $P(\boldsymbol{b})$ is a bit tricky[2].

- We require that the number of blocks B (hyperparameter) is uniform $P(B) = 1/N$ (hyperprior)
- that the number of nodes in each block n_r (hyperhyperparameter!) is also sampled uniformly $P(\boldsymbol{n}|B) = \binom{N-1}{B-1}^{-1}$ (hyperhyperprior!)
- Finally, we sample $\boldsymbol{b}$ uniformly $P(\boldsymbol{b}|\boldsymbol{n}) = \prod_r n_r!/N!$

[2]The uniform distribution $P(\boldsymbol{b}) = \frac{1}{a_N}$ won't work (recall slide 12) since most partitions have high B

Goal: infer the most likely partition $\boldsymbol{b}$ of the observed network $A \longrightarrow$ maximise $p(\boldsymbol{b}|A)$, **while avoiding overfitting**.

Putting everything together we obtain:

$$P(\boldsymbol{b}) = P(\boldsymbol{b}|\boldsymbol{n})P(\boldsymbol{n}|B)P(B)$$

$$P(\boldsymbol{b}) = \frac{\prod_r n_r!}{N!}\binom{N-1}{B-1}^{-1} N^{-1}$$

Bayesian Inference: Monte Carlo Markov Chain MCMC[a]

[a] See Q. 3 of Lab class 5!

Problem: We have written all the equations and now we can't solve them ...[2]

Best solution: we can employ MCMC

Idea: Sample from $P(\boldsymbol{b}|A)$ by starting with some $\boldsymbol{b}_0$ and then making new proposals $\boldsymbol{b} \to \boldsymbol{b}'$ with probability $P(\boldsymbol{b}'|\boldsymbol{b})$. We accept the change with probability

$$\min\left(1, \frac{P(\boldsymbol{b}'|A)P(\boldsymbol{b}|\boldsymbol{b}')}{P(\boldsymbol{b}|A)P(\boldsymbol{b}'|\boldsymbol{b})}\right)$$

[2] it's okay, now you see that the community detection problem can be formalized in a nice, principled way!

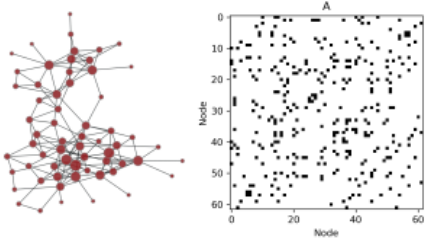
In practice: we use graph-tool

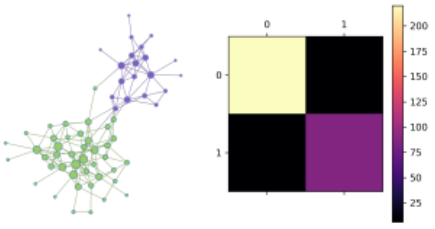
```
import graph_tool.all as gt
# load adjacency matrix
sparse_dolphin_A = scipy.io.mmread('<your path>/soc-dolphins.mtx')
# create graph
dolphin_G = gt.Graph(sparse_dolphin_A, directed=False)
# stochastic block modelling
dolphine_state = gt.minimize_blockmodel_dl(dolphin_G) # by default degree corrected, otherwise:
     state_args=dict(deg_corr=False)
for i in range(100):
    dolphine_state.multiflip_mcmc_sweep(beta=np.inf, niter=10)
```

Where I used the dolphin's dataset [7] from here.

The above is a "fancier" version of the *Metropolis-Hastings acceptance-rejection MCMC* algorithm (we used simulated annealing by setting $\beta = \infty$). To do what we described in the previous slide we use

```
    gt.mcmc_sweep(beta=1.0, niter=1000)
```





My plot settings will be available on canvas!

Bayesian Inference: graph-tool


```
g = gt.collection.data["lesmis"]

state = gt.BlockState(g)    # This automatically initializes the state with a partition
                            # into one group. The user could also pass a higher number
                            # to start with a random partition of a given size, or pass a
                            # specific initial partition using the 'b' parameter.

# Now we run 1,000 sweeps of the MCMC. Note that the number of groups
# is allowed to change, so it will eventually move from the initial
# value of B=1 to whatever is most appropriate for the data.

dS, nattempts, nmoves = state.multiflip_mcmc_sweep(niter=1000)

print("Change in description length:", dS)
print("Number of accepted vertex moves:", nmoves)
```

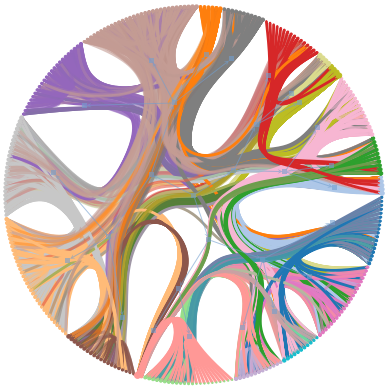
```
Change in description length: -80.840379...
Number of accepted vertex moves: 72451
```

Figure: Example directly from graph-tool documentation

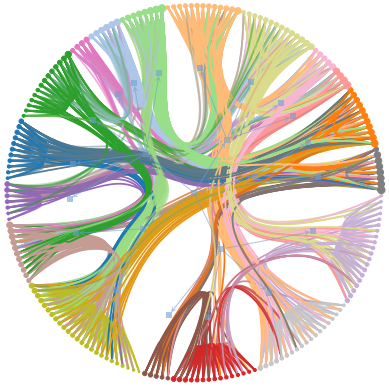
Bayesian Inference: graph-tool

graph-tool offers many extensions of what we studied today, in particular nested stochastic block modelling:

```
worm_G = gt.collection.data["celegansneural"]
worm_state = gt.minimize_nested_blockmodel_dl(
    worm_G)
# description length from quick likelihood
    minimization:
S0 = worm_state.entropy()
# let's decrease the description length:
for i in range(1000):
    worm_state.multiflip_mcmc_sweep(beta=np.inf,
        niter=10)
# description length after after sampling from
    the posterior:
S = worm_state.entropy()
worm_state.draw(ecmap = (plt.cm.inferno, .6),
    edge_pen_width = 1) # output = "c-elegans-
    network.pdf",
worm_state.print_summary()
print('\nImprovement after sampling from the
    posterior:', S - S0)
```



Bayesian Inference: graph-tool



Bibliography I

1. Váša F and Mišić B. Null models in network neuroscience. Nature Reviews Neuroscience 2022; 23 (8):493–504. DOI: 10.1038/s41583-022-00601-9
2. Orsini C, Dankulov MM, Colomer-de-Simón P, Jamakovic A, Mahadevan P, Vahdat A, Bassler KE, Toroczkai Z, Boguñá M, Caldarelli G, Fortunato S, and Krioukov D. Quantifying randomness in real networks. Nature Communications 2015; 6 (1). DOI: 10.1038/ncomms9627
3. Fortunato S. Community detection in graphs. Physics Reports 2010; 486 (3–5):75–174. DOI: 10.1016/j.physrep.2009.11.002
4. Evans TS. Clique graphs and overlapping communities. Journal of Statistical Mechanics: Theory and Experiment 2010; 2010 (12):P12037. DOI: 10.1088/1742-5468/2010/12/p12037

Bibliography II

5. Granovetter MS. The strength of weak ties. American journal of sociology 1973; 78 (6):1360–80
6. McPherson M, Smith-Lovin L, and Cook JM. Birds of a Feather: Homophily in Social Networks. Annual Review of Sociology 2001; 27 (1):415–44. DOI: 10.1146/annurev.soc.27.1.415
7. Lusseau D. The emergent properties of a dolphin social network. Proceedings of the Royal Society of London. Series B: Biological Sciences 2003; 270 (suppl_2):S186–S188
8. Simon HA. The architecture of complexity. *The Roots of Logistics*. Springer, 2012 :335–61

Bibliography III

9. Meunier D. Hierarchical modularity in human brain functional networks. Frontiers in Neuroinformatics 2009; 3. DOI: 10.3389/neuro.11.037.2009

10. Fortunato S and Hric D. Community detection in networks: A user guide. Physics Reports 2016; 659:1–44. DOI: 10.1016/j.physrep.2016.09.002

11. Clauset A, Newman MEJ, and Moore C. Finding community structure in very large networks. Physical Review E 2004; 70 (6). DOI: 10.1103/physreve.70.066111

12. Blondel VD, Guillaume JL, Lambiotte R, and Lefebvre E. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008; 2008 (10):P10008. DOI: 10.1088/1742-5468/2008/10/p10008

Bibliography IV

13. Peixoto TP. Bayesian Stochastic Blockmodeling. 2019. DOI: 10.1002/9781119483298.ch11

14. Peixoto TP. Descriptive vs. Inferential Community Detection in Networks: Pitfalls, Myths and Half-Truths. Cambridge University Press, 2023. DOI: 10.1017/9781009118897

15. Peixoto TP. Entropy of stochastic blockmodel ensembles. Physical Review E—Statistical, Nonlinear, and Soft Matter Physics 2012; 85 (5):056122