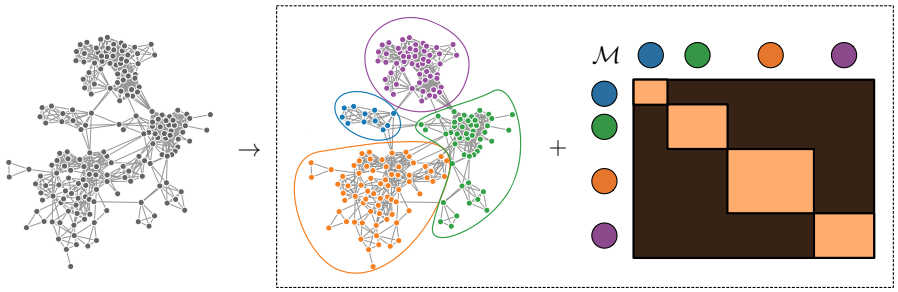
1 Modular Networks: Inference

The process of decomposing a particular network G into "clusters" of nodes and "bundles" of edges is called **community detection**. Typically, we use this process to obtain a description of the network's structure that falls between the level of local (node-level) and global (network-level) statistics, by identifying groups of nodes that connect to other groups in similar ways.[1] This kind of "graph clustering" can be used for many downstream purposes, from describing a network's architecture in terms of its modules to better predict missing information.

The input to community detection is always a network of nodes and their connections. In some cases, we can also take advantage of various forms of auxiliary information, e.g., node attributes or edge weights, to produce a better decomposition of a network into groups. The output is always a partition $\vec{z}$ of the network, i.e., an assignment of nodes into communities (or modules, or compartments, or just groups). Given that partition, we can then derive a community mixing matrix $\mathcal{M}$ that tells us how these communities fit together to construct the whole.



There are now hundreds of different algorithms for detecting communities in networks. At the end of this lecture, we will discuss some of their differences, and what it means that different algorithms can produce different partitions of the same network. In the meantime, we will learn about one of the more powerful and statistically principled approaches for community detection, and see how to use the stochastic block model (SBM) to detect communities.

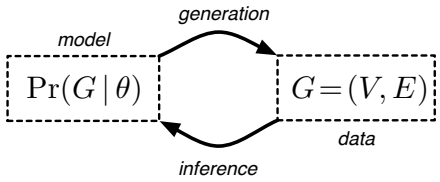
2 Statistical inference

Each of the random graph models in our toolbox is a *probabilistic generative model*, so called because it defines a parameterized probability distribution over graphs $\Pr(G \,|\, \theta)$, where θ encodes the

[1] This wording precisely recapitulates our definition of a "community," which generalizes many other notions of group-level structure in a network. These include assortative and disassortative communities (edges inside or between the groups), ordered communities (a linear hierarchy of groups), core-periphery structure (a dense core with a sparse periphery), and mixtures thereof. See Lecture 5.

structural assumptions that define the particular model.

If we ourselves choose some value of θ, we can then *generate* a network instance G by flipping a set of coins whose biases are derived from θ. For instance, recall how we generated graphs from the Erdős-Rényi model with $\theta = p$, the edge density (Lecture 3), or from the stochastic block model, with $\theta = (c, \vec{z}, \mathcal{M})$, the community partition and mixing matrix (Lecture 5). The reverse of this process is called *inference*, in which we ask, for some observed network G and model $\Pr(G \,|\, \theta)$, what is the best choice of θ for generating G? Together, generation and inference form a virtuous circle for modeling network data:



Given some data G, we can infer a model[2] $\Pr(G \,|\, \hat{\theta})$, and from the model, we can generate new (synthetic) data. If the parameters are interpretable, i.e., they have some understandable meaning for how edges are generated, inferring them can give us direct insight about the network's organization and why some edges exist but others do not. For instance, an inferred mixing matrix $\hat{\mathcal{M}}$ provides this kind of insight about how groups interact. A parameterized model also allows us to generate synthetic networks, which we can use as a substrate for simulation experiments (as in Lecture 5), or to check the validity of the inferred model, e.g., by comparing network measures of the generated networks against those of the empirical network, as in a null model (see Lecture 3).

Generative models have a variety of attractive features that partly explain why they represent the state-of-the-art in community detection. For instance, they make our assumptions about modular structure explicit, and their parameters θ typically relate to specific structural hypotheses (e.g., degrees and randomness, etc.).[3] And, they make it easier to compare competing models of a network's structure, because each can be phrased as a probability distribution, and statistics provides many ways of comparing two distributions. And finally, they are easily adapted to tasks of missing attribute and link prediction, based on a partial or past set of observations. These benefits do come at some cost, as either specifying a new model or fitting it to data can seem more complicated than less principled heuristics. In practice, however, the benefits for science are very much worth the cost. In the rest of this lecture, we will focus on developing the tools of statistical inference for the stochastic block model (SBM) and its degree-corrected variant (the DC-SBM).

[2] A "hatted" symbol $\hat{\theta}$ means a parameter value we estimated with the benefit of data. An unhatted symbol θ means one we choose without data, i.e., we chose it ourselves to create synthetic data.

[3] Most generative models for networks are what you might call "edge generative," meaning that they do not consider networks with different numbers of vertices, only networks of fixed size with different patterns of edges.

Optimize or sample? Selecting the best choice of θ for a model requires writing down some function $f(G, \theta)$ of both the data and the parameters that scores the quality of each choice. We then would either *search* the parameter space of this function for a single high-scoring choice $\hat{\theta}$, ideally the global optimum but often we have to settle for merely a local one,[4] or *sample* the parameter space for a set of high-scoring choices, in much the same way we used a sampler to generate configuration model random graphs (in Lecture 3). Both approaches are useful, depending on the circumstances. Below, we mainly consider optimization approaches.

2.1 The likelihood of a SBM

Recall that to specify a stochastic block model, we need to choose the number of communities c, the partition of nodes into groups $\vec{z}$, and the stochastic block matrix $\mathcal{M}$. From these, we can write down an explicit function of the probability or **likelihood** $\mathcal{L}$ that we generate exactly (i) the set of observed edges $(i, j) \in E$ *and* (ii) the set of observed non-edges $(i, j) \notin E$ of a network G:

$$\begin{aligned}
\mathcal{L}(G \,|\, z, \mathcal{M}) &= \prod_{i,j} \Pr(i \to j \,|\, z, \mathcal{M}) \\
&= \left(\prod_{(i,j)\in E} \Pr(i \to j \,|\, z, \mathcal{M}) \right) \left(\prod_{(i,j)\notin E} 1 - \Pr(i \to j \,|\, z, \mathcal{M}) \right) \\
&= \left(\prod_{(i,j)\in E} \mathcal{M}_{z_i, z_j} \right) \left(\prod_{(i,j)\notin E} 1 - \mathcal{M}_{z_i, z_j} \right) \quad , \qquad (1)
\end{aligned}$$

where we drop the c parameter for notational convenience.

When G is a simple graph (undirected edges, no self-loops), Eq. (1) gives the probability of observing exactly the set of $\binom{n}{2}$ values in the adjacency matrix, both 1s and 0s. The second line above separates the product over pairs i, j into two parts, the total probability of the 1s and the total probability of the 0s. In this way, every pair i, j still appears in the likelihood function exactly once, and the function contains $\Theta(n^2)$ terms.

Eq. (1) is sufficient for scoring different choices of θ for the SBM, but it is not a particularly efficient function. A key feature of the SBM is that two nodes i and j in the same community $z_i = z_j$ are *stochastically equivalent*, meaning that each of their pairwise probabilities are equal, i.e., $\Pr(i \to k) = \Pr(j \to k)$. These symmetries imply that many of the terms in Eq. (1) are the same, and we can group them together to write down a simpler likelihood function.

[4] Because it's usually hard to guarantee that we have found the global maximum of $f(G, \theta)$. It's easier to guarantee that for some choice θ, the nearby points in the parameter space are no better than $f(G, \theta)$.

Before we write down such a function, we first define two auxiliary variables

$$e_{rs} = \sum_{i,j} A_{ij}\,\delta_{r,z_j}\delta_{s,z_j} \qquad \text{and} \qquad n_{rs} = \sum_{i,j} \delta_{r,z_j}\delta_{s,z_j} \ , \tag{2}$$

which count the actual number of edges that connect nodes in group r to nodes in group s, and the number of possible such connections,[5] respectively, where δ_{ab} is the Kronecker delta function. (If G is undirected without self-loops, then i, j here is shorthand for $1 \leq i < j \leq n$, etc.)

With these in hand, we can rewrite the likelihood of G as a product series over pairs of groups r, s:

$$\mathcal{L}(G \,|\, z, \mathcal{M}) = \prod_{r,s} (\mathcal{M}_{rs})^{e_{rs}} \, (1 - \mathcal{M}_{rs})^{n_{rs} - e_{rs}} \ , \tag{3}$$

where the first term $(\mathcal{M}_{rs})^{e_{rs}}$ is the likelihood of generating exactly e_{rs} edges (the 1s in the adjacency matrix), each with probability $\mathcal{M}_{rs}$, and the second term is the likelihood of generating exactly $n_{rs} - e_{rs}$ non-edges (0s in the adjacency matrix), each with probability $1 - \mathcal{M}_{rs}$. Crucially, just like in Eq. (1), every pair i, j is accounted for in Eq. (3) (do you see why?). Notably, the form of Eq. (3) is similar to that of a Binomial distribution, where the bias on a coin depends on the group labels r and s, e_{rs} specifies the number of "successes" and n_{rs} is the number of attempts.

This version of the SBM likelihood function takes $\Theta(c^2)$ time to compute, a significant improvement, if we have precomputed the edge bundle matrix e_{rs} and group size array n_r. Given a partition z, both variables can be computed in $\Theta(n^2)$ time. If we had to recompute them for each choice of θ, Eq. (3) would not be more efficient than Eq. (1). Hence, we can only be more efficient if we pay that cost once, at initialization, and then efficiently update them for each subsequent choice of θ.

2.1.1 The maximum likelihood choice

A common strategy for sorting among different parameterizations is to prefer the ones that maximize the likelihood function. Subject to mild regularity conditions, maximum likelihood is asymptotically *consistent*, meaning that if we generate some data using a parameter θ, the *maximum likelihood estimate* (MLE) of that parameter $\hat{\theta}$ will converge on θ in the limit of large data sets. The stochastic block model is a model where this property holds.[6] Hence, because the probability

[5] The matrix e is similar to the mixing matrix $\mathcal{M}$ in the DC-SBM, which counts edges between pairs of groups. Note that in order to make G a simple graph, when $r = s$, $n_{rr} = \binom{n_r}{2}$, and otherwise $n_{rs} = n_r n_s$. This implies that e_{rr} should count edges, rather than stubs.

[6] Somewhat amazingly, not all network models have this property. For network parameter estimates to be consistent, edges must be conditionally independent, which is true in the SBM and DC-SBM: edges are independent, conditioned on their group labels. For more information about network models that lack this property, see Shalizi & Rinaldo, *Ann. Statist.* **41**(2), 508–535 (2013), `https://projecteuclid.org/euclid.aos/1366980556` .

that i, j are connected in G is a Bernoulli trial with parameter $\mathcal{M}_{z_i z_j}$, the MLE of $\mathcal{M}_{rs}$ is

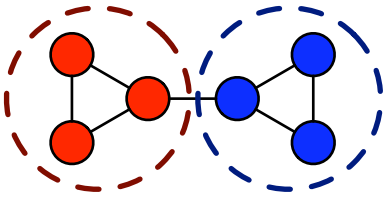
$$\hat{\mathcal{M}}_{rs} = e_{rs} \,/ n_{rs} \quad . \tag{4}$$

That is, the best choice for a mixing parameter is the observed fraction of groupwise connections.

Plugging Eq. (4) into Eq. (3), and then taking the log of both sides yields a *log-likelihood function*:

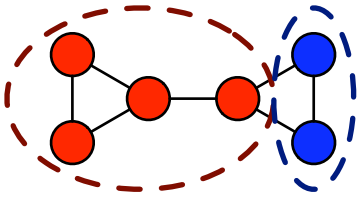
$$\ln \mathcal{L} = \sum_{r,s} e_{rs} \ln \frac{e_{rs}}{n_{rs}} + (n_{rs} - e_{rs}) \ln \left(\frac{n_{rs} - e_{rs}}{n_{rs}} \right) \quad . \tag{5}$$

which depends only on the actual e_{rs} and possible n_{rs} number of edges in the r, s edge bundle, both of which can be derived using only the partition choice z and graph G.[7] Log-likelihood functions are significantly easier to work with on computers because the logarithm converts multiplication (okay) into addition (easy), and allows us to represent extremely small numbers without worrying about underflow errors. And, crucially, the parameter choice $\hat{\theta}$ that maximizes $\mathcal{L}$ also maximizes $\ln \mathcal{L}$ (do you see why?).



$\mathcal{L}_{\text{good}} = 0.043304\ldots$
$\ln \mathcal{L}_{\text{good}} = -3.1395\ldots$

e_{rs}/n_{rs}	red	blue
red	3/3	1/9
blue	1/9	3/3



$\mathcal{L}_{\text{bad}} = 0.000244\ldots$
$\ln \mathcal{L}_{\text{bad}} = -8.3178\ldots$

e_{rs}/n_{rs}	red	blue
red	4/6	2/8
blue	2/8	1/1

2.1.2 An example

To see how the SBM likelihood function sorts different partitions, consider a simple $n = 6$ node and $m = 7$ edge network composed of two triangles linked by a single edge, and two possible $c = 2$ partitions. In the "good" partition, each triangle is placed into its own group, and $n_{\text{red}} = n_{\text{blue}} = 3$. In the "bad" partition, we place both nodes on either end of the "bridge" edge into the same group as one of the triangles. If Eq. (5) is to sort partitions well, then clearly the good partition should have a higher likelihood than the bad partition.

[7] In both the SBM here, and the DC-SBM described below, we define $0^0 = 1$. This choice prevents the numerical calculation from failing when either $e_{rs} = 0$ or $n_{rs} - e_{rs} = 0$.

The $\mathcal{M}$ stochastic block matrix in each case has been populated with the corresponding MLE fractions e_{rs}/n_{rs}. Then, applying Eq. (5) to the corresponding matrices yields:

$$\ln \mathcal{L}_{\text{good}} = \left(3 \ln \frac{3}{3}\right) + \left(1 \ln \frac{1}{9} + 8 \ln \frac{8}{9}\right) + \left(3 \ln \frac{3}{3}\right) = -3.1395\ldots$$
$$\ln \mathcal{L}_{\text{bad}} = \left(4 \ln \frac{4}{6} + 2 \ln \frac{2}{6}\right) + \left(2 \ln \frac{2}{8} + 6 \ln \frac{6}{8}\right) + \left(1 \ln \frac{1}{1}\right) = -8.3178\ldots \quad .$$

We can measure how much better the "good" partition is than the "bad" one by computing $\mathrm{e}^{\Delta \ln \mathcal{L}}$, which is their likelihood ratio (do you see why?). Plugging in our results shows that the good partition is $\exp(\ln \mathcal{L}_{\text{good}} - \ln \mathcal{L}_{\text{bad}}) = 177$ times more likely to generate the observed data than the "bad" partition. In other words, the good partition is a much better model of the data.

2.2 The likelihood of a DC-SBM

Recall that to specify a degree-corrected SBM, we need to choose the number of communities c, the partition of nodes into groups $\vec{z}$, the expected degree sequence $\vec{k}$, and the mixing matrix $\mathcal{M}$.

Under the DC-SBM, the value we assign to any particular adjacency A_{ij} is a Poisson-distributed random variable with mean $\gamma_i \gamma_j \mathcal{M}_{z_i z_j}$, where each γ_i is a node-specific model parameter that quantifies the fraction of the group z_i's total degree that belongs to node i (see Lecture 5). Hence, the likelihood function is

$$\begin{aligned}\mathcal{L}(G \,|\, z, \gamma, \mathcal{M}) &= \prod_{i,j} \text{Poisson}(\gamma_i \gamma_j \mathcal{M}_{z_i z_j}) \\ &= \prod_{i<j} \frac{\left(\gamma_i \gamma_j \mathcal{M}_{z_i z_j}\right)^{A_{ij}}}{A_{ij}!} \exp\left(-\gamma_i \gamma_j \mathcal{M}_{z_i z_j}\right) \times \prod_i \frac{\left(\frac{1}{2}\gamma_i^2 \mathcal{M}_{z_i z_j}\right)^{A_{ii}/2}}{(A_{ii}/2)!} \exp\left(-\frac{1}{2}\gamma_i^2 \mathcal{M}_{z_i z_j}\right) \quad (6)\end{aligned}$$

where the two parts are the likelihoods of the between-group and within-group adjacencies, respectively. These two parts appear because we assume an undirected network, and in the DC-SBM, we count stubs within groups but edges between groups (see Lecture 5).

Although it may seem rather complicated, we can substantially simplify Eq. (6) by first factoring out and collecting the various constants that depend only on the adjacency matrix,[8] and then substituting maximum likelihood estimators for γ and $\mathcal{M}$:

$$\hat{\gamma}_i = \frac{k_i}{\kappa_{z_i}} \qquad \text{and} \qquad \hat{\mathcal{M}}_{rs} = e_{rs} \ , \quad (7)$$

[8] Which we may collect in a term $1/C = \prod_{i<j} A_{ij}! \prod_i 2^{A_{ii}/2} (A_{ii}/2)!$.

where e_{rs} is the same as in Eq. (2), except that when $r = s$, we sum over $i \neq j$ (which counts the within-group stubs rather than within-group edges), and

$$\kappa_r = \sum_s \hat{\mathcal{M}}_{rs} = \sum_i k_i \delta_{z_i,r} \tag{8}$$

is the "degree" of group r.

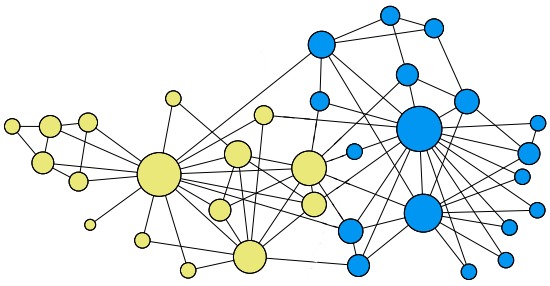
Taking the logarithm and then dropping the collected constants, which do not depend on the parameter choice, yields a nicely compact expression that depends only on the edge bundle counts induced by the partition choice z on the graph G:

$$\ln \mathcal{L} = \sum_{rs} e_{rs} \ln \frac{e_{rs}}{\kappa_r \kappa_s} \quad , \tag{9}$$

where the sum runs over all the elements of the e_{rs} matrix in order to account for both ends of every edge. Notice that the form of Eq. (9) is similar to that of the SBM's log-likelihood function in Eq. (5), but where we replace the node-based count n_{rs} with an edge-based count $\kappa_r \kappa_s$.

2.2.1 An example

The canonical example for how the SBM and DC-SBM can produce different good partitions of the same network is in their application to the Zachary karate club social network.[9] This network represents the $m = 78$ friendships among $n = 34$ members of a karate club at a U.S. university in the 1970s. During the course of Zachary's study, the club split into $c = 2$ similar-sized groups, centered around the club's president and instructor (dashed line in the figures below). Hence, this network is simple graph in which nodes are annotated with a categorical attribute that represents a "social partition," i.e., which of the two groups the person went with after the split.



The $c = 2$ partition that maximizes the SBM log-likelihood in Eq. (5) places the five highest-degree nodes into one group (including both the president and the instructor), and the other 29 nodes

[9] See W. W. Zachary, *J. Anthropol. Res.* **33**, 452–473 (1977), `http://www.jstor.org/stable/3629752`. There's also this tumblr: `http://networkkarate.tumblr.com`.

into the other. The tables below give the corresponding e_{rs} and n_{rs} matrices, and log-likelihoods, for both partitions. This partition is very different from the social partition, but it's not less meaningful—instead, divides the network into what we might call leaders and followers. Under the SBM, the leader-follower partition is an astounding $\exp(196.29 - 179.39) \approx 10^{7.3}$ more likely to generate the observed network than is the social partition.

e_{rs}/n_{rs}	A (16)	B (18)
A (16)	33/120	10/288
B (18)	—	35/153

$\ln \mathcal{L}_{\text{SBM}} = -196.29$ (social)

e_{rs}/n_{rs}	A (5)	B (29)
A (5)	5/10	54/145
B (29)	—	19/406

$\ln \mathcal{L}_{\text{SBM}} = -179.39$ (leader-follower)

In contrast, the $c = 2$ partition that maximizes the DC-SBM log-likelihood in Eq. (9) presents a very different picture: the social partition is now a far better model of the edges than the leader-follower partition. The tables below give the corresponding e_{rs} matrix and κ_r values, and the log-likelihoods, for the two partitions under this model.

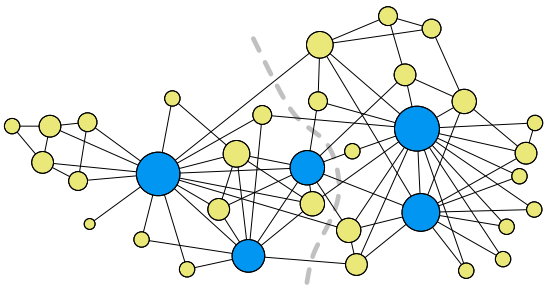
e_{rs}	A (16)	B (18)	κ_r
A (16)	66	10	76
B (18)	10	70	80

$\ln \mathcal{L}_{\text{DC-SBM}} = -739.43$ (social)

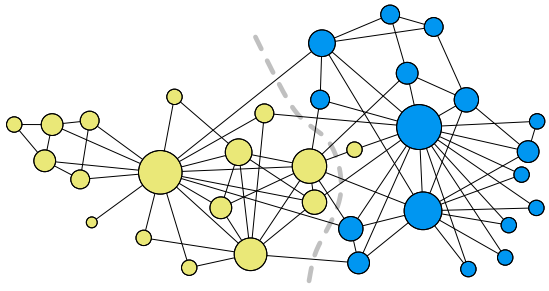
e_{rs}	A (5)	B (29)	κ_r
A (5)	10	54	64
B (29)	54	38	92

$\ln \mathcal{L}_{\text{DC-SBM}} = -772.28$ (leader-follower)

Although close, the social partition is not the globally best DC-SBM partition for this network. That partition moves one node between the groups, and yields a log-likelihood of $\ln \mathcal{L} = -739.39$, which about 4% more likely to generate the observed network than the social partition.[10]



best $\hat{z}$, SBM ($c = 2$)



best $\hat{z}$, DC-SBM ($c = 2$)

[10] Here's a fun, possibly apocryphal story about the one "misclassified" node, which has an equal number of connections to each group. This person apparently had a karate exam coming up soon, and when the club split, they chose to go with the instructor's faction, instead of the president's, in order to be better prepared for the exam.

2.3 Choosing the number of groups c

The likelihood functions for both the SBM and the DC-SBM keep the number of groups c fixed, and neither provides a way to choose its value. Instead, we have to choose some c and then search for the best c-way partition of the network. This is not an unreasonable requirement, but why can't we choose c automatically, somehow?

We can, but doing so via statistical inference requires account for the fact that changing c changes the "complexity" or flexibility of the model, because $\mathcal{M}$ has $\Theta(c^2)$ entries. Likelihoods for different choices of c cannot be compared fairly because the "larger" model, with more parameters, will naturally tend to yield higher likelihood models of the network. In the extreme case of $c = n$, the model can simply "memorizes" the adjacency matrix via the $\Theta(n^2)$ parameters in the mixing matrix $A = \mathcal{M}$, placing every node is in a group by itself.

A popular approach to making different choices of c comparable is to *penalize* or *regularize* the likelihood by some function $f(c)$ that is grows with c. That is, we impose a cost to using additional parameters, and then search for the parameterization that balances this cost against the improved fit to the network. There are many choices for $f(c)$, and these go by names like Bayesian marginalization, Bayes factors, various information criteria (BIC, AIC, etc.), minimum description length (MDL) approaches, and more. We will not cover any of these techniques here.

2.4 Finding good partitions

The preceding sections have reduced the overall task of community detection, first, into one parameter estimation via statistical inference, and second, into one of searching over partitions all $\{\vec{z}\}$ to find one $\vec{z}$ that maximizes the SBM or DC-SBM log-likelihood of a network.

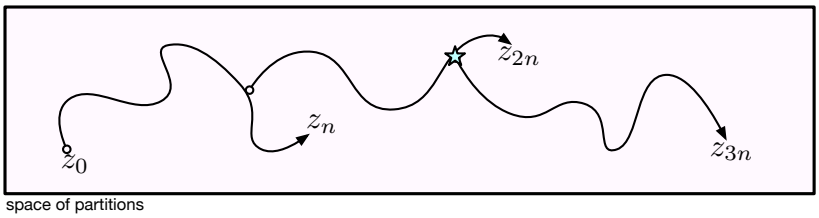
There are many ways we could perform such a search over partitions. These include powerful methods like Markov chain Monte Carlo, expectation-maximization, belief propagation, among many others. Here, we will learn about a **locally greedy heuristic**, which is a kind of generalization of the Kernighan-Lin algorithm developed in 1970 for solving the minimum-cut graph partitioning problem.[11] Like Kernighan-Lin, the locally greedy heuristic explores a sequence of partitions $z_0, z_1, \ldots, z_t$ by moving nodes one-at-a-time between groups, in a series of phases, until an entire phase passes when no improvements are made.[12]

The locally greedy heuristic proceeds in phases. In each phase, we move each node exactly once, in a greedy fashion, to whichever other group results in the most positive (least negative) change in the log-likelihood. (Moving some nodes may *decrease* the log-likelihood. This is fine.) At the end

[11] Kernighan & Lin, *Bell System Technical Journal* **49**, 291 (1970), `https://archive.org/details/bstj49-2-291`.
[12] Described well in Karrer & Newman, *Phys. Rev. E* **83**, 016107 (2011), `https://arxiv.org/abs/1008.3926`.

of each phase, we then examine each of the previous n partitions, and initialize the next phase to begin at the one with the best (most positive) log-likelihood. However, if each of the n partitions had a lower log-likelihood than the partition that initialized the previous phase, we halt and return that partition.

Basically, this heuristic performs a sequence of local perturbations to an initial partition (represented as a circle-to-arrow line below). In this way, it explores nearby partitions. It then repeats this process, each time starting a new sequence from whichever was the best partition in the previous sequence. As a result, it exploits the best partition it traversed. If an entire new sequence fails to find a better partition than that phase's initial partition (e.g., the blue star in the figure), then the algorithm has converged on local optimum, and it returns that partition.



We can make this verbal description more concrete by writing it out in pseudocode:

`function local-greedy`(G):

1. `initialization`
   for each node i: assign a random group label $z_{0,i} = \text{Uniform}(1, c)$,
   record the log-likelihood as $w_0 = \ln \mathcal{L}(z_0)$
   set $t = 0$
2. `proceed in phases`
   while not converged:
   (a) for each node i,
      i. `copy previous partition`
      set $t = t + 1$
      set $z_t = z_{t-1}$
      let $s = z_{t,i}$ denote i's current group
      ii. `consider all moves`
      compute $\Delta \ln \mathcal{L}$ for each possible move of i to a *different* group $r \neq s$
      iii. `make the greedy choice`
      among these $c - 1$ possibilities, identify the group r that would produce the most positive change $\Delta \ln \mathcal{L}$ (or least negative, if no changes are positive)
      iv. `update this partition`
      move i by setting $z_{t,i} = r$
      set $w_t = \ln \mathcal{L}(z_t)$

(b) (evaluate the sequence)
among the log-likelihoods $w_0, \ldots, w_n$ produced in this phase, identify the largest w_{t*},
(halt) if $w_{t*} \leq w_0$, then halt and return z_0
(new phase) otherwise, set $z_0 = z_{t*}$ and $w_0 = w_{t*}$

Left unspecified in step (a) is the order in which we consider each node i. Two simple choices would be to either consider nodes (i) in alphanumeric order $1, 2, \ldots, n$, or (ii) in a different random order in each phase. Regardless, this locally greedy heuristic will, at some point, converge on a local optimum, but it is not guaranteed to be the global optimum. We can mitigate this problem to some degree by simply running the heuristic a number of times, each with with a different random initial partition and then taking the best partition over all the runs.

2.4.1 An example

To illustrate how this locally greedy heuristic works in practice, we'll apply it to a synthetic network with $n = 30$ nodes, connected according to a DC-SBM planted partition of $c = 3$ equal-sized and assortative groups (see Lecture 5), where every node has the same expected degree. In this experiment, we'll track both the log-likelihood (given by Eq. (9)), and a measure of the distance between the current and planted partitions called the variation of information (VI).[13]

Before showing the results of the experiment, let's first define the variation of information (VI), which measures the "distance" between two partitions X and Y. The VI has the nice property that it obeys the triangle inequality, which means its a distance metric. For our purposes, we'll define X to be the planted partition we used to generate the network, which means the VI provides a complementary measure for quantifying how well the locally greedy heuristic finds good partitions.

For two partitions X and Y, the **variation of information** is defined as

$$\begin{aligned} \mathrm{VI}(X,Y) &= H(X) + H(Y) - 2I(X,Y) && (10) \\ &= -\frac{1}{n}\sum_{r,s} d_{rs}\left[\log\left(\frac{d_{rs}}{p_r}\right) + \log\left(\frac{d_{rs}}{q_s}\right)\right] \ , && (11) \end{aligned}$$

where Eq. (10) is expressed in terms of the standard entropy and mutual information functions $H(.)$ and $I(.,.)$, while Eq. (11) is expressed in terms of simple counts

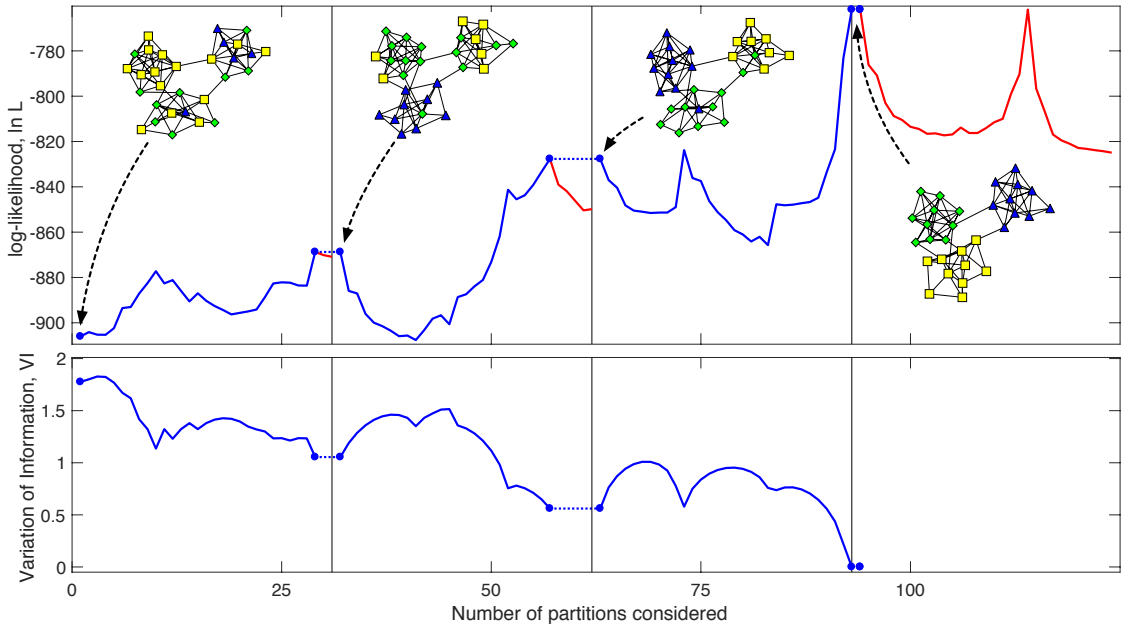
$$d_{rs} = |X_r \cap Y_s| \qquad \text{and} \qquad p_r = |X_r| \qquad q_r = |Y_r| \ , \tag{12}$$

where d_{rs} counts the number of nodes that are in group r in partition X and in group s in partition Y, and p_r and q_r are the group sizes in X and Y.[14] In the latter formulation, calculating the

[13] Introduced in Meilă, *J. Multivariate Analysis* **98**, 873–895 (2007), `https://bit.ly/2qdunKl`.
[14] As in both the SBM and DC-SBM log-likelihoods, we must define $0^0 = 1$ in practice.

VI takes time $\Theta(n)$ from the partitions X and Y. In addition to being a metric, the variation of information has the property $\mathrm{VI}(X, X) = 0$ (do you see why?), and is bounded $\mathrm{VI}(X, Y) \leq 2 \log c$. In our experiment, we will set X equal to the planted partition, and Y will be the current partition being considered. If the locally greedy heuristic is working, the log-likelihood will go up, and the VI will go down.

Starting with a random initial partition of our $n = 30$ node network, the figure below illustrates the locally greedy heuristic's trajectory through the partition space. Within each phase (demarcated by a black vertical line), the highest log-likelihood partition is marked, and a dotted line connects it to the initialization of the next phase. In the upper panel, the lower-quality partitions explored after the best-in-phase partition is found are indicated in a different color, which reflect subsequent node moves that worsen the quality of the partition. In this run of the algorithm, the heuristic finds the planted partition at the end of the third phase, at which point the distance drops to $\mathrm{VI} = 0$ in the lower panel.[15]



[15] This run was particularly lucky in finding a short path from the random initial partition to the planted partition. More typically, the heuristic gets stuck in a local optimum, which is why the right strategy is to run the heuristic many times, each from a different random initial partition, and take the best output across these runs.

2.5 Caveats and limitations of community detection

Generally, the task of community detection is the network analog of clustering data in a vector space. While clustering aims to find groups of points that clump together, community detection aims to find groups of nodes in a network with similar connectivity patterns.

The general problem of clustering is notoriously slippery, and cannot, in fact, be solved universally.[16] Ultimately, *which* clustering is optimal depends on what we intend to do with it later. Crucially, a similar fact holds for community detection. It's a remarkably useful tool in our toolbox, but its output should always be interpreted with some skepticism. Good community detection methods, like the SBM and the DC-SBM, can be powerful tools for exploratory data analysis, able to uncover a wide variety of such patterns in real networks. They can also be used to predict missing links or missing node attributes, as null models, or to test specific network hypotheses. But, the particular output of any algorithm may not tell us the full story about a network's modular structure, as we will see below.

2.5.1 Competitive local optima

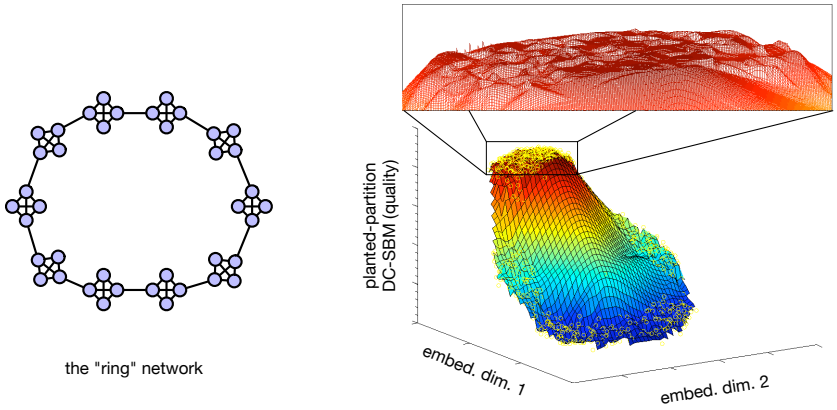
A common feature across many of the most popular and most powerful ways of decomposing a network into its constituent communities is that the *objective function* (the likelihood function above) is not convex. If it were, then it would be relatively easy to pursue some kind of "gradient ascent" strategy in order to reliably find the globally best partition under a given model. Instead, these objective functions are often very "rugged", with a large number of local optima in which search algorithms tend to get stuck. For instance, below is a visualization of the landscape of the likelihood function (right-hand figure) for a planted-partition version of the DC-SBM, applied to a "ring" network of n/d cliques, each of size d, arranged in a ring (left-hand figure). The zoomed-in portion illustrates just how rugged the surface function is—the global maximum is in there somewhere, but one would be hard pressed to tell which peak it is.[17]

For this reason, the output of nearly any community detection algorithms cannot be considered the *best possible* partition under a given model. Rather, it is likely to be merely a *pretty good* solution. Furthermore, as the figure's inset illustrates, many of the local optima are also *competitive* in quality with the global optimum, and mathematically, there tend to be an exponential number of them.[18] In a very real sense, the prevalence of these merely good solutions is why so many different

[16] For a truly excellent explanation of this fact, see von Luxburg et al. *J. Mach. Learn. Res.* **27**, 65–79 (2012). `http://proceedings.mlr.press/v27/luxburg12a/luxburg12a.pdf`, and Kleinberg *Adv. Neural Inf. Process. Syst.* 463–470 (2003), `https://www.cs.cornell.edu/home/kleinber/nips15.pdf`.

[17] Reproduced from Good et al., *Phys. Rev. E* **81**, 046106 (2010), `http://arxiv.org/abs/0910.0165`. Embedding dimensions are constructed by computing pairwise VI distances among a set of sampled partitions, and then embedding those points in a 2-dimensional space that preserves those pairwise distances, .

[18] That might sound like a big number, but they represent a vanishingly small fraction of the total space of all



algorithmic strategies for optimizing this or that objective function work relatively well—it's not hard to find a decent local optimum, and, in truth, finding even one can be useful if we are mainly doing data exploration.

However, if we have a scientific application in mind, e.g., we want to use the output partition $\vec{z}$ to conduct additional analyses, modeling, or hypothesis testing, then we must ask whether it matters *which* local optimum an algorithm converges on. If most good solutions are similar to each other in how the decompose the network, then any one will be representative of the set and we need not worry which we found. But, if good solutions are structurally dissimilar to each other, then there's no guarantee that our algorithm of choice has found the one that is scientifically meaningful. What are we to do in such situations? A reasonable strategy is to switch from optimization to sampling to instead obtain a set of good partitions $\{\vec{z}\}$ (e.g., running the algorithm of Section 2.4 many times) and then check whether they are similar or different before proceeding with any downstream analyses.

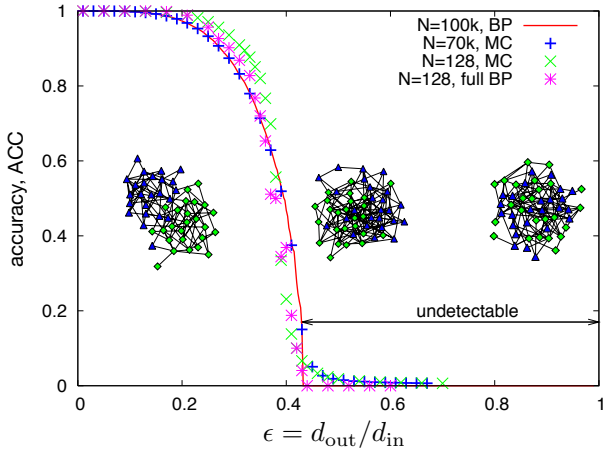
2.5.2 Detectability limits

In addition to not necessarily finding the best partition possible, community detection is not guaranteed to find the actual communities in a network, if the boundaries between them are insufficiently sharp. This is true not just on real-world networks, where the underlying community structure is not known, but also true on synthetic networks with known structure, and it holds for all algorithms.

To illustrate this fact, recall the planted partition model for the SBM, in which we have $c=2$ equal-sized communities and where the sharpness of their boundary is governed by a single parameter ϵ. In seminal work, Decelle et al. (2011) proved mathematically that for this simple setting, there exists a regime of ϵ beyond which *no algorithm* can successfully recover the planted partition, and

partitions, which is super-exponential in size.

that this "undetectable" regime emerges well before the communities are formally equivalent. The figure below, from their paper, shows a sharp phase transition in the recoverability of the planted communities. (In their formulation, ϵ is defined as the ratio of out-of-group edges to in-group edges, but the model is otherwise the same as the one we saw in Lecture 5.)[19]



These "detectability limits" have since been extended to networks that vary over time, to multilayer networks, and to networks with node metadata that we might exploit to identify underlying communities (e.g., if the metadata labels correlate with the community structure pattern of edges). The implication of these limits is that we should be mildly skeptical of the output of any community detection algorithm, as it may not have been able to detect the actual underlying structure.

2.5.3 No ground truths and No Free Lunches

Beyond not finding the best partition and not necessarily detecting the underlying communities, there is no guarantee that community detection can recover the correct "ground truth" and, on average, there is no "best" community detection algorithm.

Many networks have nodes annotated with metadata $\vec{x}$, and it's common to treat the partition these labels induce as being the thing that a good community detection algorithm should find, using only the edges. But, node metadata are not really "ground truth" for network communities; they're just more data. It's possible, sometimes even likely, that $\vec{x}$ correlates with the community structure of the network, e.g., if the probability that an edge (i, j) exists is conditioned in some way on x_i and x_j. But without a clear model of that relationship, metadata always have an uncertain relationship with ground truth.

[19] Reproduced in part from Decelle, et al., *Phys. Rev. E* **84**, 066106 (2011), `https://arxiv.org/abs/1109.3041`.

Think of it this way: if an algorithm fails to find a good division $\vec{z}$ that correlates with some node metadata $\vec{x}$, there are four possible reasons:

(i) (no correlation) the metadata are irrelevant to the network's structure,

(ii) (they're different) the detected communities and the metadata capture different aspects of the networks structure,

(iii) (no correlation) the network contains no communities (as in a non-modular random graph) or its communities fall below the detectability threshold, or

(iv) the community detection algorithm performed poorly.

In practice, we often cannot tell which of these cases is more or less likely, and the presence of measurement errors or noise in the network's connectivity or metadata only increases that uncertainty.

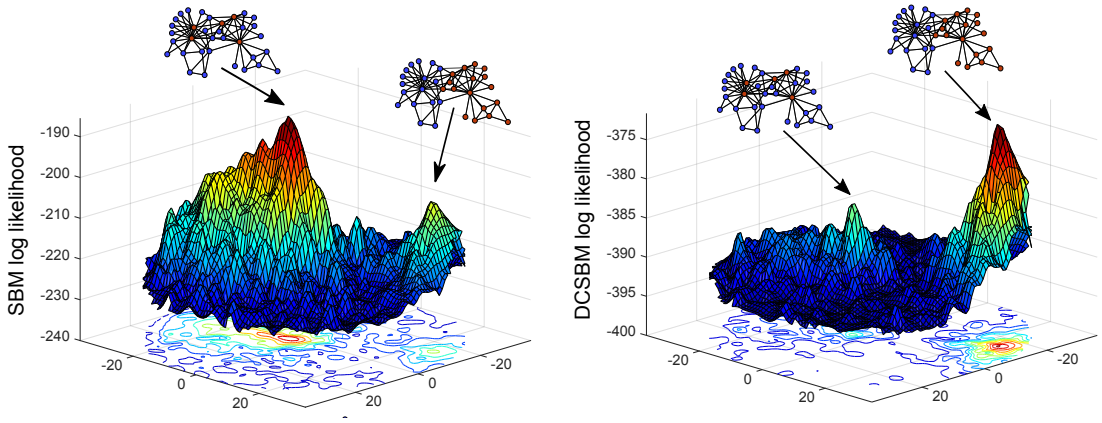
There is no ground truth. Formally, community detection is a kind of inverse problem. We imagine generating a network G by applying a generative process g to a "ground truth" set of communities $\mathcal{T}$, i.e., we compute $G = g(\mathcal{T})$. In community detection, we want an algorithm g^{-1} that can reliably identify the $\mathcal{T}$ that produced G, i.e., community detection aims to compute $\mathcal{T} = g^{-1}(G)$.

Now suppose we have a network G that can be produced by multiple, distinct generative processes, each with its own ground truth. That is, we have $G = g_1(\mathcal{T}_1) = g_2(\mathcal{T}_2)$, but where $(g_1, \mathcal{T}_1) \neq (g_2, \mathcal{T}_2)$. If this is possible, and it is, then no community detection algorithm method can uniquely solve the problem for every network, or even just for one arbitrary network. In other words, the community detection problem is not invertible because there is not a bijection between networks G and ground truths $\mathcal{T}$. Or, more succinctly, there is no (identifiable) ground truth.

We have already seen one such network: the Zachary karate club can be generated using either the SBM with the leader-follower partition, or the DS-SBM with something like the social partition. The figure below illustrates this point using the likelihood surfaces for both models, in which we have labeled the local maxima for both partitions.

There are No Free Lunches. In machine learning, the famous No Free Lunch (NFL) theorem proves that, subject to mild constraints on the evaluation criteria, no learning algorithm has an *a priori* advantage over any other across all possible inputs. In other words, for a set of cases that a particular method $\mathcal{A}_1$ outperforms $\mathcal{A}_2$, the NFL theorem implies that there must exist a set of cases where $\mathcal{A}_2$ outperforms $\mathcal{A}_1$ in such a way that, on average, no algorithm performs better than any other. This theorem also holds for the task of community detection.[20]

[20] See Peel et al., *Science Advances*, **3**(5), e1602548 (2017). Formally, the NFL holds for community detection algorithms evaluated using the standard measure of adjusted mutual information (AMI), which is closely related to the variation of information in Eq. (10). Notably, there do exist areas of statistical learning where the NFL doesn't hold. Identifying these narrow settings is an area of active research.



The lack of generally better, or worse, performance by different community detection algorithms does not mean community detection itself if pointless. Rather, the theorem implies that if the inputs of interest represent a restricted subset of cases, e.g., all networks with realistic structure, or specific groups of networks like food webs or online social networks, then there may indeed be a method that out-performs others *within the confines of that subset*. In other words, an algorithm can be better on some groups of inputs only if it is worse on enough other inputs to compensate. Hence, if we intend to use community detection algorithms in practice, we are more likely to obtain good solutions if we match our beliefs about the underlying data generating process g with the particular assumptions of an algorithm $\mathcal{A}$. That is, we should favor specialized algorithms whenever we know something about what kind of network we're analyzing.

3 At home

1. Read Chapters 14.1 and 14.4 in *Networks*