1 Random graphs with heterogeneous degrees

Suppose that instead of setting the density of edges p—akin to choosing the mean degree $\langle k \rangle$—we fixed the entire degree sequence $\vec{k}$ of a random graph. The **configuration model** defines such an object: a random graph with a specified degree structure. These networks are richer in their structural variety than Erdős-Rényi random graphs, because they have more realistic degree structures—indeed, we can even set $\vec{k}$ to be exactly the degree sequence of a real-world network A, to obtain a more realistic null model: one that matches A on its degree structure.

As an undirected adjacency matrix, such a random graph essentially says

$$\forall_{i>j} \qquad A_{ij} = A_{ji} = \begin{cases} 1 & \text{with probability} \propto k_i k_j \\ 0 & \text{otherwise} \end{cases} ,$$

meaning that the probability that i, j are connected depends on their joint degrees k_i and k_j, rather than the constant value $p = 2m/n(n-1)$ of an ER graph.

And, just as the Erdős-Rényi model allows us to test whether some descriptive statistic's value can be explained by the density of edges alone (plus randomness), the configuration model lets us say whether or not it can be explained by the degree distribution alone (plus randomness).

A configuration model is defined as a random graph with a specified degree sequence, i.e., we set the exact degrees of each node in the network, and then draw uniformly at random from the set of graphs with that property. A closely related random graph model, called the Chung-Lu model, also uses a specified degree sequence, but nodes in the corresponding random graphs have the specified degree is in expectation, with Poisson noise around that value.

There are, in fact, *four* configuration models for undirected networks,[1] depending on the target graph properties; specifically, (i) self-loops, yes or no, and (ii) multi-edges, yes or no. If neither are permitted, then we are dealing with the space of random simple graphs. We note that nearly all of the statements in this lecture also hold for a configuration model with directed edges, where we separately specify $\vec{k}^{\text{out}}$ and $\vec{k}^{\text{in}}$, but for simplicity we focus on the undirected case.

1.1 Properties of random graphs with specified degree structure

Configuration models generate more realistic networks because at least they can capture the heavy-tailed nature of real-world degree distributions, which we saw previously. For this reason, they find uses in a variety of network analyses and models. The two main uses are (i) as a mathematical

[1]The subtleties of these distinctions, and their implications for generating and using configuration model random graphs are discussed thoroughly in Fosdick et al., *SIAM Review* **60** (2018) `https://arxiv.org/abs/1608.00607`.

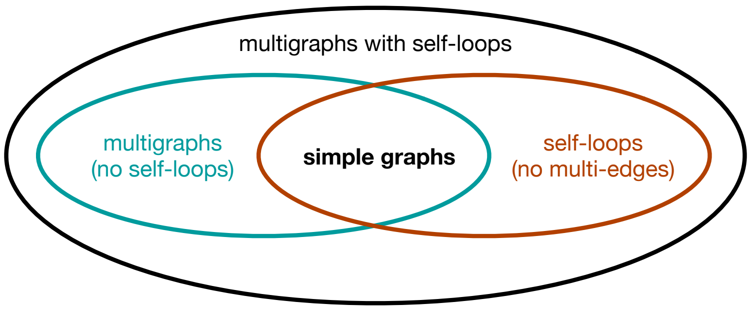


Figure 1: The different "graph spaces" of the configuration model, and their relationships.

models or substrates for questions about network dynamics, and (ii) as "null" models for constructing reference distributions for empirical quantities.

At a high level, configuration model random graphs have the following properties, which are similar in many ways to those of an ER graph:

- Four flavors: *simple graph* or multigraph, both with or without self-loops.
- Connectivity is *specified* by the degree sequence parameter $\vec{k}$, and many patterns vary with the mean $\langle k \rangle$ and variance $\langle k^2 \rangle$ of this sequence.
- The diameter and mean geodesic distance are typically $O(\log n)$, making configuration model graphs "small-world-like," but can shrink further if $\vec{k}$ is extremely heavy-tailed.
- Motif frequencies tend to be $O(1/n)$ when the network is sparse, but can be larger as either $\langle k \rangle$ or $\langle k^2 \rangle$ increases.
- The size of the largest connected component (LCC) tends to be $O(n)$ when G is not too sparse, but a giant component can emergence in sparser settings if either $\langle k \rangle$ or $\langle k^2 \rangle$ is large.

A *configuration model* can be denoted as $G(n, \vec{k})$ where $\vec{k} = \{k_i\}$ is a *degree sequence* on n vertices, with k_i being the degree of vertex i. This model denotes a uniform distribution over all graphs with n vertices, conditioned on their having the $\vec{k}$ degree sequence.

Here are some examples of configuration models:

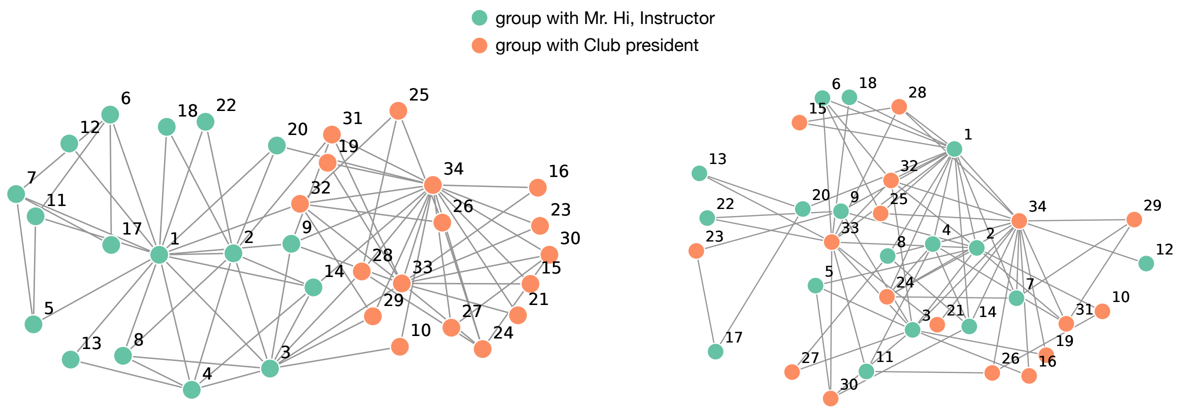
- Fix all the degrees to be the some constant k. This choice defines a probability distribution over all k-regular graphs.
- Choose a set of n values drawn iid from some parametric degree distribution $\Pr(k)$, e.g., a Poisson distribution, a geometric distribution, a log-normal distribution, or even a power-law distribution. If we choose a Poisson with mean $c/(n-1)$, then we recover the $G(n, p)$ model

if c is small. If we choose a power law, then the model is sometimes called a "power-law random graph."

- Choose the degree sequence of an empirical network.

If the degree distribution $\Pr(k)$ we use to generate the degree sequence has a mathematically simple form, we can often analytically calculate certain properties of the corresponding ensemble of graphs, much like we did with Erdős-Rényi random graphs. This specification provides us with a rich family of models, and is a common strategy for mathematical modeling in network science.

On the other hand, setting the input degree sequence $\vec{k}$ equal to that of some empirical network we are studying turns the configuration model into a null model, a powerful tool for exploring the structure of real-world networks. The figures below show such an application, to the Zachary karate club network: on the left is the empirical network, and on the right a single draw from the corresponding configuration model. Immediately noticeable is that while the degree structure has been preserved, the original group structure—clearly visible on the left—has been randomized.



1.2 Generating a configuration model network

Unlike an ER graph, generating a configuration model network is mildly non-trivial. This additional complexity is nearly always worthwhile because of how practically useful they are.

Two facts constrain how we generate a random graph with specified degree structure $\vec{k}$.

Constraint 1: which graph space? First, we must decide which graph space the generated graph will come from, which then tells us which algorithm we use to generate it:

- multigraphs with self-loops: use the Molloy-Reed stub-matching algorithm

- multigraphs with no self-loops: use Fosdick et al. MCMC
- self-loops but no multi-edges: use Fosdick et al. MCMC
- simple graphs: use Fosdick et al. MCMC (specified degrees) or Chung-Lu (expected degrees).

Constraint 2: is it graphical? Second, given a graph space, the degree sequence must be *graphical* within that graph space, meaning that we must be able to pair every "stub" of every edge to some other appropriate stub. What constitutes an appropriate stub depends on that graph space we want the graph to come from.

For instance, the sequence $(1, 1, 10)$ is not graphical as a simple graph, because the degree 10 node will have 8 unmatched "stubs" after connecting to the two degree 1 nodes. On the other hand, this sequence is graphical as a multigraph with self-loops.

Given an arbitrary sequence, we can use the Havel-Hakimi algorithm[2] to determine if there exists any simple graph G with $\vec{k}$. More practically, every real-world network A is an existence proof that its degree sequence is graphical, in whatever graph space A comes from. Hence, if we are using a configuration model as a null model, then this constraint is typically automatically satisfied.

1.2.1 Fosdick et al. MCMC: sampling to generate

The most general way to generate a configuration model network is using the Fosdick et al. MCMC,[3] which is a *Markov chain* algorithm that *samples* networks with a specified degree sequence, using a "double-edge swap" to move around in a given graph space.

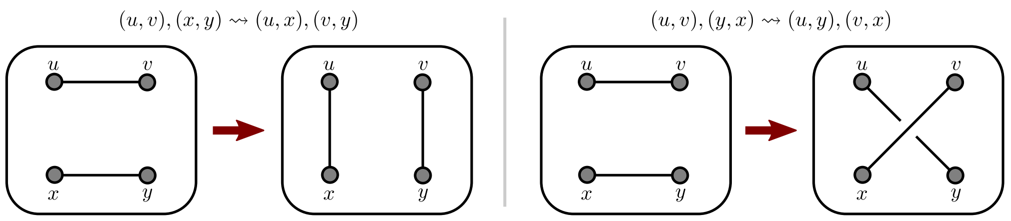
To start the chain, we must first construct *one* G_0 that has the target graph properties and specified degree sequence $\vec{k}$. If $\vec{k}$ is not graphical, we will know at this step, and can stop. Once we have this initial graph, the algorithm then repeatedly applies a simple function $g : G_t \to G_{t+1}$ to create a sequence of graphs $G_0, G_1, G_2, \ldots, G_t$, for any t we like. If g has the right properties, this graph sequence (the "chain") will be a random walk within the target set $\mathcal{S}$ that eventually visits every member with the same probability. For a sufficiently long chain, the structure of G_t will be independent of G_0 and can we output G_t as the generated random graph, or, if we run the chain sufficiently long between draws, produce as many such random graphs as we like.

[2]Havel-Hakimi goes as follows. Sort the nodes in decreasing order of their degree. Without loss of generality, let that list be $L = (v_n, v_{n-1}, \ldots, v_1)$, with degrees $\vec{k} = (k_n, k_{n-1}, \ldots, k_1)$. Then, recursively apply the following steps. Take the largest degree vertex v_i and connect it to the k_i vertices immediately to its right in L, i.e., vertices $v_{i-1}, \ldots, v_{i-k_i}$; set $k_i = 0$ and decrement the degrees of each of the right-hand vertices i connected to; finally, remove all vertices from L and $\vec{k}$ that have degree 0. If $\vec{k}$ is graphical, then we will apply this rule at most $n - 1$ times. If at any time $k_i > i - 1$, then there are not enough "stubs" left for i to connect to, and the sequence is not graphical. See also `https://en.wikipedia.org/wiki/Havel-Hakimi_algorithm`.

[3]A complete Python implementation is here `https://github.com/UpasanaDutta98/ConfigModel_MCMC`

Given some $G_t \in \mathcal{S}$, we can generate another $G_{t+1} \in \mathcal{S}$ using a simple *double edge swap* procedure, as follows:

- choose a uniformly random pair of edges $(u, v), (x, y)$
- create G_{t+1} by replacing these edges with either $(u, x), (v, y)$ or $(u, y), (v, x)$ (equal probability)
- finally, if $G_{t+1} \notin \mathcal{S}$ (the sequence exited the target set), revert to G_t and pick new edges.[4]


$(u,v),(x,y) \rightsquigarrow (u,x),(v,y)$
$(u,v),(y,x) \rightsquigarrow (u,y),(v,x)$
u
v
x
y

The idea here is that the double edge swap incrementally randomizes which nodes connect to each other, and so applying it repeatedly can transform any $G_i \in \mathcal{S}$ to any other $G_j \in \mathcal{S}$. Crucially, the double edge swap is *degree preserving*, meaning that the degrees k_u, k_v, k_x, k_y before and after the swap remain the same. Hence, every graph G_t in the sequence will have the same, specified degree sequence $\vec{k}$. The third step of the procedure guarantees that each G_{t+1} will also have the target set of graph properties, and hence so will the final output graph.

Convergence and sampling. In the sequence of graphs $G_0, G_1, G_2, \ldots, G_t$ traversed by the Fosdick et al. MCMC algorithm, the first graph G_0 is almost always *not* a uniformly random choice—it is either the particular output of Havel-Hakimi or an empirical network we used as a starting point. So how large should t be before G_t *is* a uniformly random graph with the target properties?

This question is one of *convergence*: how long do we run a Markov chain before G_t is independent of G_0, and hence has "converged" on a uniform distribution over $\mathcal{S}$? And then also, how many steps η_0 should we go between "draws" of graphs from the chain? When the input degree sequence is defined mathematically, it may be possible to derive mathematical bounds for these questions; when the input is empirical, we must instead rely on heuristics, albeit statistically principled ones.

Empirical studies of running the Fosdick et al. MCMC on hundreds of real-world networks of varying size and origin suggests that for most networks,[5] convergence occurs after $t^* \approx 20m$ double-edge

[4]Depending on the chosen graph space, there can be other conditions under which a swap is rejected. Fosdick et al., *SIAM Review* (2018) explains.

[5]These heuristics work best for networks in which the edge density ρ and the maximum degree are not too high. If a network violates these technical constraints, the Markov chain tends to behave in unusual ways, which requires a direct estimation of an appropriate sampling gap η_0.

swaps, and that consecutive "draws" can be made about every $\eta_0 = 2m$ steps thereafter.[6]

1.2.2 Chung-Lu model: simple graphs from flipping coins

Much of the complexity of generating configuration model random graphs stems from requiring that G have *exactly* the specified degree sequence $\vec{k}$. If we relax this constraint, requiring only that G have a degree sequence close to $\vec{k}$, then we can use the Chung-Lu model[7] to generate a simple random graph in much the same way as we generate an ER graph. [8]

Suppose that we fix a choice of nodes i and j, with degrees k_i and k_j. What is the probability that i, j are connected? For now, let $k_i = 1$, i.e., i only has one edge to connect anywhere. From i's perspective, it has k_j chances for its one edge "stub" to connect to j. And, across the entire network, there are $2m - k_i$ possible stubs where its edge could land (we subtract off k_i because we prohibit self-loops). Hence, the probability that i connects to j is $k_j/(2m - k_i)$. This argument holds independently for each of i's stubs,[9] implying that a probability that any of i's stubs connects to any of j's stubs

$$\text{if} \quad \max_i k_i^2 < 2m \quad \text{then} \qquad p_{ij} = k_i \left(\frac{k_j}{2m - k_i} \right) \approx \frac{k_i k_j}{2m} \ . \tag{1}$$

The constraint on the left-hand side in Eq. (1) ensures that $p_{ij} \leq 1$ always.

The Chung-Lu model calculates this probability for each pair i, j and then adds each edge (i, j) to G with probability p_{ij}:

$$\forall_{i>j} \qquad A_{ij} = A_{ji} = \begin{cases} 1 & \text{with probability } p_{ij} \\ 0 & \text{otherwise} \ . \end{cases}$$

The Chung-Lu model does not produce exactly $\vec{k}$, and instead generates networks whose degrees are $\vec{k}$ in expectation, only. In fact, the degree of a node i will be a Poisson-distributed random variable, with mean equal to the specified degree k_i, if the network is sparse and the maximum degree is not too large. And, because we flip one coin for each pair of nodes, Chung-Lu graphs are simple, and take the same amount of time to generate as a similarly sized ER graph. They also generalize naturally to directed networks, when both $\vec{k}^{\text{in}}$ and $\vec{k}^{\text{out}}$ are specified, by setting $p_{ij} \approx k_i^{\text{out}} k_j^{\text{in}}/m$, and dropping the requirement that $A_{ij} = A_{ji}$.

[6]See Dutta, Fosdick and Clauset, "Sampling random graphs with specified degree sequences." Preprint, arXiv:2105.12120 (2022). `https://arxiv.org/abs/2105.12120`

[7]Chung and Lu, *Annals of Combinatorics* **6**, 125–145 (2002) `http://math.ucsd.edu/~fan/wp/conn.pdf`

[8]It's possible to draw from the Chung-Lu model in only $O(n + m)$ time, using an algorithm described by Miller and Hagberg, *Proc. Workshop on Algorithms and Models for the Web Graph* (WAW 2011) `https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-11-01482`.

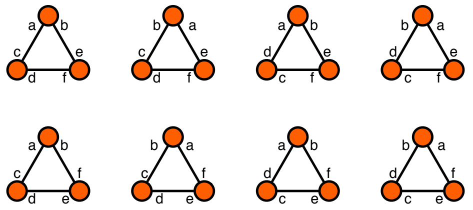
[9]Such an independence assumption is approximately valid for sparse and large networks, because the probability of multiple connections between i and j is vanishingly small. Hence, the difference between this argument and the rigorous one is negligible in most cases of interest.

1.2.3 Molloy-Reed model: multigraphs from random matchings

If we are working in the graph space of "loopy multigraphs," or we are doing asymptotic mathematical analysis, we can use the Molloy-Reed model to generate a configuration model network. Molloy-Reed works by choosing a uniformly random matching on the degree "stubs" (half edges) of the specified degree sequence. Unlike in the Chung-Lu model, which only generates simple graphs by design, or the Fosdick et al. MCMC, this "stub matching" method will typically produce some number of self-loops and multi-edges. In practice, these deviations from a simple graph represent an asymptotically small fraction of all edges.[10]

Given a degree sequence $\vec{k} = \{k_1, k_2, \ldots, k_n\}$, we say that each vertex i has a number of "stubs" equal to its degree. Every matching on these stubs, in which we repeatedly choose an unmatched stub on some vertex i and connect it with some unmatched stub on vertex j, represents a network. Under this method of generating a graph, we will choose such a matching uniformly at random from among all such matchings. Each possible matching thus occurs with equal probability; however, each network with the specified degree sequence does not occur with equal probability under this model, as some matchings produce the same network.

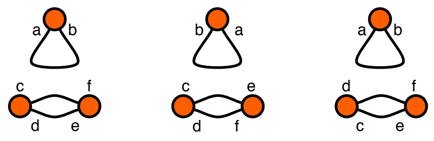
To illustrate this idea, consider the set of matchings on three vertices, each with degree 2, that result in a triangle. The following figure shows the distinct labelings, and hence distinct matchings, that form a triangle. In the configuration model, we choose each of these with equal probability.



However, these are not the only possible matchings on these six edge stubs. The following figure shows three other distinct matchings, which produce non-simple networks, i.e., networks with self-loops and/or multi-edges.

In mathematical analyses of Molloy-Reed networks, because the fraction of edges involved in either self-loops or multi-edges is vanishingly small in the large-n limit, it is common to ignore them.

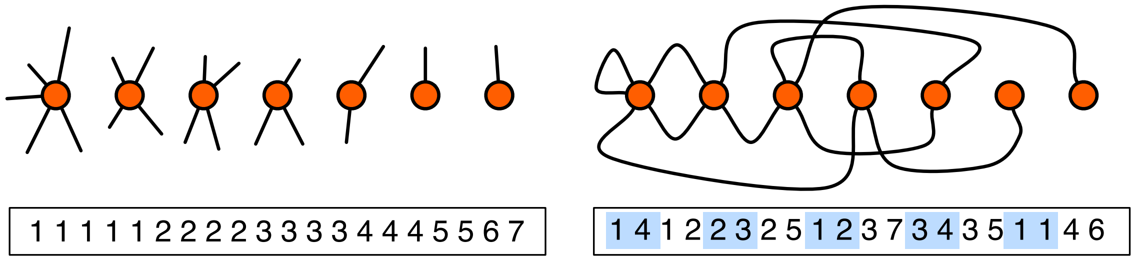
[10]Warning: one can always "simplify" a loopy multigraph by removing self-loops and collapsing multi-edges, and potentially also discarding disconnected components. This procedure is *not* recommended, because it is biased in how it changes the resulting degree distribution: because both self-loops and multi-edges are more likely to be attached to high-degree nodes, removing them tends to suppress the degree distribution's tail, which can have correspondingly large impact on many network quantities. In some cases, doing this can even *invert* the conclusions one might draw, had one analyzed a properly generated simple graph with the right degree sequence (see Fosdick et al. (2018)).



Once a degree sequence $\vec{k}$ has been chosen, e.g., by taking the degree sequence in some empirical network or by drawing a sequence from a probability distribution over the positive integers, to generate a Molloy-Reed loopy multigraph, we simply need an efficient method by which to choose a uniformly random matching on the $\sum_i k_i$ stubs. Happily, this is straightforward and efficient.

Let v be an array of length $2m$ and let us write the index of each vertex i exactly k_i times in the vector v. Each of these entries will represent a single edge stub attached to vertex i. Then, we take a random permutation of the entries of v and read the contents of the array in order, in pairs. For each pair that we read, we add the corresponding edge to the network. (Once we have taking a random permutation of the stubs, we could choose the pairs in any other way, but reading them in order allows us to write down the full network with only a single pass through the array.)

For instance, the figure below shows an example of this "stub matching" construction of a configuration model random graph. On the left is shown both the vertices with their stubs, which shows the degree sequence graphically, and the initial contents of the array v. On the right is shown the wired up network defined by the in-order sequence of pairs given in the array, which has been replaced with a random permutation of v. In this case, the random permutation produces both one self-loop and one multi-edge.



Standard mathematical libraries often provide the functionality to select a uniformly random permutation on the contents of v. However, it is straightforward to do it manually, as well: to each entry v_i, associate a uniformly random variable $r_i \sim U(0, 1)$ (which most good pseudorandom number generators will produce). Sorting the r_i values produces a random permutation[11] on the associated v_i values, which can be done using QuickSort in time $O(m \log m)$, or you can generate a random permutation directly in $O(m)$ time.

1.3 Graph space complications: vertex vs. edge labels

Earlier, we stated that there were four distinct types of configuration models. In fact, however, there is another dimension of variation that complicates this view: do edge stubs in the network have identities? To answer this question, we introduce two definitions:

- Definition: *A vertex-labeled graph is a graph in which each vertex has a distinct label.*

  For example, in a social network, each node is a person, and people are distinct and unique, but there is no implicit labeling of a person's friends—there is no "1st" friend or "kth" friend.

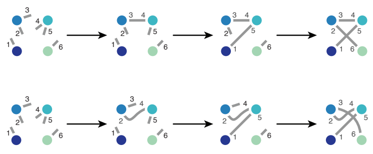
- Definition: *A stub-labeled graph is a graph in which each half-edge (stub) has a distinct label, and thus each edge has a pair of distinct labels.*

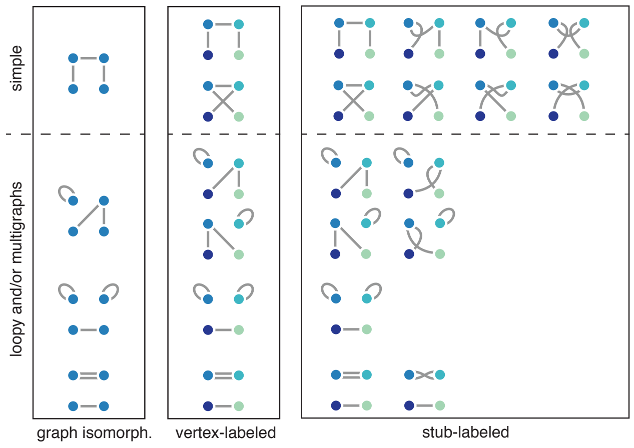
  For instance, in a sexual contact network, connects are ordered (by time), and so it can matter whether j is the 1st or kth contact with i. Or, in an employment network, where companies connect to the people they hire, jobs are usually distinct, implying that each permutation of employees across the job-stubs of a company i represents a distinct employment graph.

Note that a stub-labeled graph also has implicitly labeled vertices, since each vertex is distinctly labeled by the set of labeled stubs attached to it. These distinctions are important because, depending on whether the graph space being considered is stub-labeled or merely vertex-labeled, the sizes and compositions of the sets will differ. In other words: for a fixed $\vec{k}$, a uniform distribution over a stub-labeled space is not necessarily a uniform distribution over a vertex-labeled space. This is illustrated in the figure below for $\vec{k} = \{1, 2, 2, 1\}$, where the graph space for the stub-labeled graph is far larger than for the vertex-labeled version.

Deciding whether to use a vertex- or stub-labeled space depends on the definition of a node and the definition of an edge, and depends mainly on whether there is a meaningful distinction between the connections, with respect to the underlying system the network models.

[11] Each of these permutations occurs with probability equal to $1/n!$, and because there are $n!$ such permutations, we are choosing uniformly from among them. If we choose the r_i values uniformly at random, then the probability that any particular element v_i has the smallest r_i is $1/n$. Similarly, the probability that v_i has the ith smallest value is $1/(n-i+1)$. By induction, the probability of choosing any particular ordering is $\prod_{i=1}^{n}(n-i+1)^{-1} = 1/n!$. It is possible to choose a random permutation in $O(m)$ time using an in-place randomizer. Instead of sorting the uniform deviates, we instead loop from $i = 1$ to n within v and swap v_i with a uniformly randomly chosen element v_j where $i \leq j \leq n$. The proof that this produces a random permutation follows the proof above.





1.3.1 Counting configurations

Given a simple and stub-labeled graph, how many other simple graphs are there under permutations of the stub labels? For vertex i with degree k_i, the stubs can be permuted in $k_i!$ ways. Therefore, the number of stub-labeled permutations across the graph is

$$q_{\text{simple}}(G) = \prod_{i=1}^{n} k_i! \; . \tag{2}$$

Note that this depends only on the degree sequence, and not on the particular configuration. This means that for each simple graph is the vertex-labeled space, there are exactly $q_{\text{simple}}(G)$ "isomorphic" graphs in the stub-labeled space.

What about self loops and multi-edges? Let w_{ij} be the integer number of edges between vertices i and j. For a single self loop, let $w_{ii} = 1$. First, consider a single self-loop on a node i. This reduces the total number of stub-labeled configurations by a factor of 2 since an edge (i_1, i_2) is equivalent to (i_2, i_1). If there are multiple self loops, the number of configurations drops by an additional factor of $w_{ii}!$. This means that the number of configurations drops by a factor of $\prod_{i=1}^{n} w_{ii}!(2^{w_{ii}})$. Using similar arguments for networks with multi-edges, the following relationships can be derived.

$$\begin{aligned} q_{\text{loopy}}(G) &= q_{\text{simple}}(G) \frac{1}{\prod_{i=1}^{n} w_{ii}!(2^{w_{ii}})} \\ q_{\text{multi}}(G) &= q_{\text{simple}}(G) \frac{1}{\prod_{i<j} w_{ij}!} \\ q_{\text{loopy multi}}(G) &= q_{\text{simple}}(G) \frac{1}{\prod_{i=1}^{n} w_{ii}!(2^{w_{ii}})} \times \frac{1}{\prod_{i<j} w_{ij}!} \; . \end{aligned} \tag{3}$$

Based on the equations above, one could therefore, in principle, draw networks from any stub-labeled space using the stub-matching procedure, and then weight accordingly. However, the conversion factors in the equations above can be enormous, illustrating that the graphs that are prevalent in one distribution can be extremely different from those that are prevalent in the other distribution. As a result, a conversion between stub-labeled and vertex-labeled spaces is an infeasible approach to sampling from the less easily sampled space.

1.4 Mathematical properties and descriptive statistics

The precise mathematical properties for the configuration model depend on the choice of degree sequence. In the version of the model where we draw the degree sequence from some distribution, we may often calculate properties of the configuration model ensemble analytically (often using powerful techniques called generating functions).[12]

Under the random matching approach for constructing an instance of the model, for a particular stub attached to vertex i, there are k_j possible stubs, out of $2m-1$ (excluding the stub on i under consideration), attached to j to which it could connect. And, there are k_i chances that this could happen. Thus, the probability that i and j are connected is

$$\begin{aligned} p_{ij} &= \frac{k_i k_j}{2m-1} \\ &\simeq \frac{k_i k_j}{2m} \ , \end{aligned} \tag{4}$$

where the second form holds in the limit of large m. Notice that this immediately implies that the higher the degrees are of i and j, the greater the probability that they connect under the configuration model.

1.4.1 Expected number of multi-edges

Eq. (4) gives the probability that one edge appears between i and j. A closely related quantity is the probability that a second edge appears between i and j, and this quantity allows us to calculate the expected number of multi-edges in the entire network. The construction is very similar to that above, except that we must update our counts of stubs to account for the existence of the first edge between i and j.

The probability that a second edge appears is $(k_i-1)(k_j-1)/2m$, because we have used one stub from each of i and j to form the first edge. Thus, the probability of both a first and a second edge

[12] For a good introduction to this technique, see Wilf *generatingfunctionology*, AK Peters (2006).

appearing is $k_i k_j (k_i - 1)(k_j - 1)/(2m)^2$. Summing this expression over all distinct pairs gives us the expected number of multi-edges in the entire network:

$$\begin{aligned}
\sum_{\text{distinct } i,j} \left(\frac{k_i k_j}{2m}\right)\left(\frac{(k_i-1)(k_j-1)}{2m}\right) &= \frac{1}{2}\frac{1}{(2m)^2}\left(\sum_{i=1}^{n} k_i(k_i-1)\sum_{j=1}^{n} k_j(k_j-1)\right) \\
&= \frac{1}{2\langle k\rangle^2 n^2}\left(\sum_i k_i^2 - k_i\right)\left(\sum_j k_j^2 - k_j\right) \\
&= \frac{1}{2\langle k\rangle^2}\left(\frac{1}{n}\sum_i k_i^2 - \frac{1}{n}\sum_i k_i\right)\left(\frac{1}{n}\sum_j k_j^2 - \frac{1}{n}\sum_j k_j\right) \\
&= \frac{\big(\langle k^2\rangle - \langle k\rangle\big)^2}{2\langle k\rangle^2} \\
&= \frac{1}{2}\left[\frac{\langle k^2\rangle - \langle k\rangle}{\langle k\rangle}\right]^2 \quad . \qquad (5)
\end{aligned}$$

In this derivation, we used several identities: $2m = \langle k\rangle n$, which relates the number of edge stubs to the mean degree and number of vertices, and $\langle k^m\rangle = \frac{1}{n}\sum_i k_i^m$, which is the mth (uncentered) moment of the degree sequence.

The result, Eq. (5), is a compact expression that depends only on the first and second moments of the degree sequence, and not on the size of the network. Thus, the expected number of multi-edges is a constant[13] implying a vanishingly small $O(1/n)$ fraction of all edges in the large-n limit.

1.4.2 Expected number of self-loops

This argument works almost the same for self-loops, except that the number of pairs of possible connections is $\binom{k_i}{2}$ instead of $k_i k_j$. Thus, the probability of a self-loop is $p_{ii} = k_i(k_i - 1)\,/4m$, and the expected number of self-loops is

$$\frac{\langle k^2\rangle - \langle k\rangle}{2\langle k\rangle} \ ,$$

which is a constant depending only on the first and second moments of the degree sequence. Thus, just as with multi-edges, self-loops are a vanishingly small $O(1/n)$ fraction of all edges in the large-n limit when $\langle k^2\rangle$ is finite.

[13]So long as the first and second moments of the distribution producing $\vec{k}$ are finite, which is not the case if the degree distribution follows a power law with $\alpha < 3$. We are also ignoring the fact that we treated the $i = j$ case, i.e., self-loops, identically to the $i \neq j$ case, but this difference is small, as the next section shows.

1.4.3 Expected number of common neighbors

Given a pair of vertices i and j, with degrees k_i and k_j, how many common neighbors n_{ij} do we expected them to have?

For some ℓ to be a common neighbor of a pair i and j, both the (i, ℓ) edge and the (j, ℓ) edges must exist. As with the multi-edge calculation above, the correct calculation must account for the reduction in the number of available stubs for the (j, ℓ) edge once we condition on the (i, ℓ) edge existing. Thus, the probability that ℓ is a common neighbor is the product of the probability that ℓ is a neighbor of i, which is given by Eq. (4), and the probability that ℓ is a neighbor of j, given that the edge (i, ℓ) exists, which is also given by Eq. (4) except that we must decrement the stub count on ℓ.

$$\begin{aligned} n_{ij} &= \sum_{\ell} \left(\frac{k_i k_\ell}{2m} \right) \left(\frac{k_j (k_\ell - 1)}{2m} \right) \\ &= \left(\frac{k_i k_j}{2m} \right) \sum_{\ell} \frac{k_\ell (k_\ell - 1)}{\langle k \rangle n} \\ &= p_{ij} \frac{\langle k^2 \rangle - \langle k \rangle}{\langle k \rangle} \quad . \end{aligned} \tag{6}$$

Thus, the probability that i and j have a common neighbor is proportional to the probability that they themselves are connected (where the constant of proportionality again depends on the first and second moments of the degree sequence).

1.4.4 The excess degree distribution

Many quantities about the configuration model, including the clustering coefficient, can be calculated using something called the *excess degree distribution*, which gives the degree distribution of a randomly chosen neighbor of a randomly chosen vertex, excluding the edge followed to get there. This distribution also shows us something slightly counterintuitive about configuration model networks.

Let p_k be the fraction of vertices in the network with degree k, and suppose that following the edge brings us to a vertex of degree k. What is the probability that event? To have arrived at a vertex with degree k, we must have followed an edge attached to one of the $n\, p_k$ vertices of degree k in the network. Because edges are a random matching conditioned on the vertex's degrees, the end point of every edge in the network has the same probability $k/2m$ (in the limit of large m) of connecting to one of the stubs attached to our vertex.

Thus, the degree distribution of a randomly chosen neighbor is

$$\begin{aligned} p_{\text{neighbor has } k} &= \frac{k}{2m} n\, p_k \\ &= \frac{k\, p_k}{\langle k \rangle} \quad . \end{aligned} \tag{7}$$

Although the excess degree distribution is closely related to Eq. (7), there are a few interesting things this formula implies that are worth describing.

From this expression, we can calculate the average degree of such a neighbor, as $\langle k_{\text{neighbor}} \rangle = \sum_k k\, p_{\text{neighbor has } k} = \langle k^2 \rangle / \langle k \rangle$, which is strictly greater than the mean degree itself $\langle k \rangle$ (do you see why?). Counterintuitively, this means that your neighbors in the network tend to have a greater degree than you do. This happens because high-degree vertices have more edges attached to them, and each edge provides a chance that the random step will choose them.

Returning to the excess degree distribution, note that because we followed an edge to get to our final destination, its degree must be at least 1, as there are no edges we could follow to arrive a vertex with degree 0. The excess degree distribution is the probability of the number of other edges attached to our destination, and thus we substitute $k+1$ for k in our expression for the probability of a degree k. This yields

$$q_k = \frac{(k+1)p_{k+1}}{\langle k \rangle} \quad . \tag{8}$$

1.4.5 Expected clustering coefficient

The clustering coefficient C is the average probability that two neighbors of a vertex are themselves neighbors of each other, which we can calculate now using Eq. (8). Given that we start at some vertex v (which has degree $k \geq 2$), we choose a random pair of its neighbors i and j, and ask for the probability that they themselves are connected. The degree distribution of i (or j), however, is exactly the excess degree distribution, because we chose a random vertex v and followed a randomly chosen edge.

The probability that i and j are themselves connected is $k_i k_j / 2m$, and the clustering coefficient is given by this probability multiplied by the probability that i has excess degree k_i and that j has excess degree k_j, and summed over all choices of k_i and k_j:

$$\begin{aligned}
C &= \sum_{k_i=0}^{\infty}\sum_{k_j=0}^{\infty} q_{k_i} q_{k_j} \frac{k_i k_j}{2m} \\
&= \frac{1}{2m}\left[\sum_{k=0}^{\infty} q_k k\right]^2 \\
&= \frac{1}{2m\langle k\rangle^2}\left[\sum_{k=0}^{\infty} k(k+1)p_{k+1}\right]^2 \\
&= \frac{1}{2m\langle k\rangle^2}\left[\sum_{k=0}^{\infty} k(k-1)p_{k}\right]^2 \\
&= \frac{1}{2m\langle k\rangle^2}\left[\sum_{k=0}^{\infty} k^2\, p_{k} - \sum_{k=0}^{\infty} k\, p_k\right]^2 \\
&= \frac{1}{n}\frac{\left[\langle k^2\rangle - \langle k\rangle\right]^2}{\langle k\rangle^3} \quad . \qquad (9)
\end{aligned}$$

where we have used the definition of the mth moment of a distribution to reduce the summations. Like the expression we derived for the expected number of multi-edges, the expected clustering coefficient is a vanishing fraction $O(1/n)$ in the limit of large networks, so long as the second moment of the degree distribution is finite.

1.4.6 Expected clustering coefficient (alternative)

It should also be possible to calculate the expected clustering coefficient under the configuration model by starting with the expected number of common neighbors n_{ij} for some pair i, j, which we derived in Eq. (6). In particular, given the result derived in Eq. (9), we can express the clustering coefficient in terms of n_{ij} and p_{ij}:

$$C = \frac{1}{2m}\left(\frac{n_{ij}}{p_{ij}}\right)^2 \quad . \qquad (10)$$

(Can you explain why this formula is correct?)

1.4.7 The giant component, and network diameter

Just as with the Erdős-Rényi random graph model, the configuration model also exhibits a phase transition for the appearance of a giant component. The most compact calculation uses generating functions, and is given in Chapter 13.8 in *Networks*. The result of these calculations is a simple formula for estimating when a giant component will exist, which, like all of our other results, depends only on the first and second moments of the degree distribution:

$$\langle k^2 \rangle - 2\langle k \rangle > 0 \ . \tag{11}$$

Unlike our previous results, however, this equation works even when the second moment of the distribution is infinite. In that case, the requirement is trivially true.

A corollary of the existence of the giant component in this model is the implication that the diameter of the network grows logarithmically with n, when a giant component exists. As with $G(n, p)$, the configuration model is locally tree-like (which is consistent with the vanishingly small clustering coefficient derived above), implying that the number of vertices within a distance ℓ of some vertex v grows exponentially with ℓ, where the rate of this growth again depends on the first two moments of the degree distribution (which are themselves related to the number of first- and second-neighbors of v).

1.5 Directed random graphs

All of these results can be generalized to the case of directed graphs, and the intuition we built from the undirected case generally carries over to the directed case, as well. There are, of course, small differences, as now we must concern ourselves with both the in- and out-degree distributions, and the results will depend on second moments of these. (The first moments of the in- and out-degree distributions must be equal. Do you see why?)

Generating directed random graphs using the configuration model can be done using directed analogs of the construction techniques described above. In the double-edge swap algorithm, edges now have directionality, and so the swap must respect both node degree and edge direction. In the Chung-Lu model, the value p_{ij} is different from p_{ji}. And in the Molloy-Reed model, instead of maintaining a single array v containing the names of the stubs, we maintain two arrays, v_{in} and v_{out}, each of length m, which contain the in- and out-stubs respectively. The uniformly random matching we choose is then between these arrays, with the beginning of an edge chosen from v_{out} and the ending of an edge chosen from v_{in}.

2 Random graphs as null models

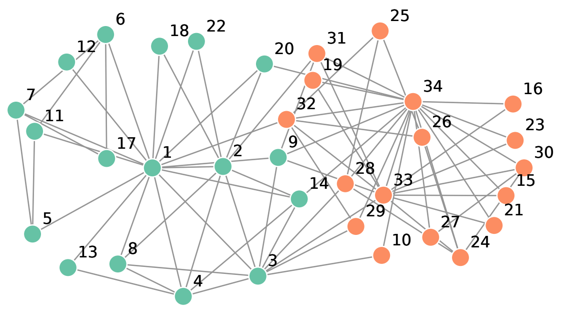
The configuration model is remarkably useful for constructing reference distributions, or null models, for evaluating different structural patterns in a real network A. In particular, it allows us to answer the question of

Can some observed pattern be explained by the network's degree structure alone?

The configuration model defines a probability distribution over graphs $\Pr(G \,|\, \vec{k})$ with the same degrees as the original network A. If f is a function we can compute on A, e.g., any network-level or node-level descriptive statistic we've seen previously, we can compute the same function on a synthetic graph G drawn from A's configuration model $f(G)$. Because G is a random variable, applying f to each of many draws from the configuration model $\{G_1, G_2, \dots\}$ provides a practical estimate of a reference distribution $\Pr(f(G) \,|\, \vec{k})$ against which we can evaluate whether $f(A)$ is typical, unusual, or an outlier, given A's degree structue.

2.1 Zachary's karate club

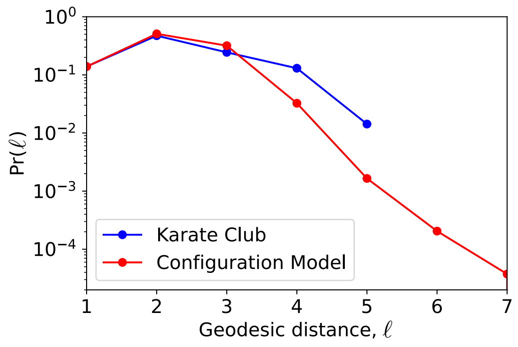
As a first example, we'll examine the structure of Zachary's karate club ($n = 34$, $m = 77$) to ask how well different descriptive statistics of the empirical network are "explained" by the network's specific degrees.

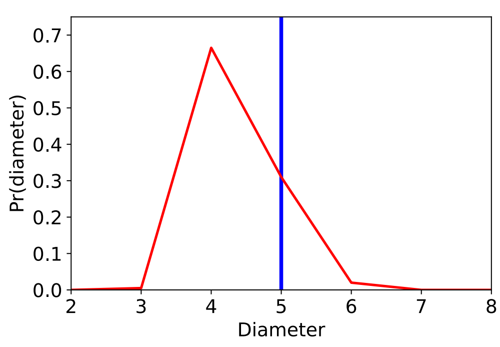


Because the karate club is a simple graph, we use the Fosdick et al. MCMC to construct the reference distribution. Typically, 1000 synthetic networks is sufficient to get a good approximation of that distribution, although for functions f that produce highly variable outputs, more might be needed to get a smooth curve. For this example, we'll examine both network-level statistics like the distribution of geodesic path lengths $\Pr(\ell)$ or the network diameter $\ell_{\max}$, and node-level statistics such as measures of node "centrality."

Tabulating the distribution of geodesic path lengths $\Pr(\ell)$ and the diameter $\Pr(\ell_{\max})$ from the synthetic networks reveals that the values of both of these descriptive statistics align closely with what the configuration model produces. Hence, neither empirical pattern or value is particularly interesting—they are very close to what we would expect, given the network's degree structure (and randomness). Except, perhaps, that the karate club exhibits slightly more very long paths than expected ($\ell = 4$ or 5).

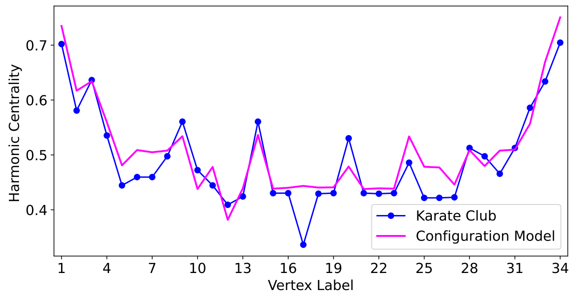
Harmonic centrality is a node-level measure of how "close" a particular node i is to the "center"

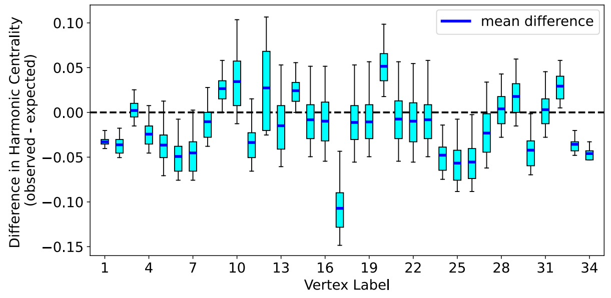




of a network, and is defined as $h_i = \frac{1}{n-1}\sum_{j\neq i}\ell_{ij}^{-1}$, where ℓ_{ij} is the length of a geodesic path from the focal node i to every other node j. Calculating h_i for each node in a synthetic network, and then tabulating the distribution $\Pr(h_i)$ across networks, reveals that the average h_i under the configuration model is often very close to the observed value in the karate club (left figure below).

But, not all of these values are explained by the network's degree structure. Taking the difference between a node's empirical centrality ("observed") and its simulated centralities ("expected") gives us a distribution of deviations for node i. This node-level reference distribution (right hand panel) lets us assess more closely whether a node's centrality can be explained by the degree structure of the network alone, i.e., if the reference distribution includes $\Delta = 0$. If a node's reference distribution is mostly above this line, then i is more central than we would expect based on degrees alone, while if it is below the line, node i is less central.

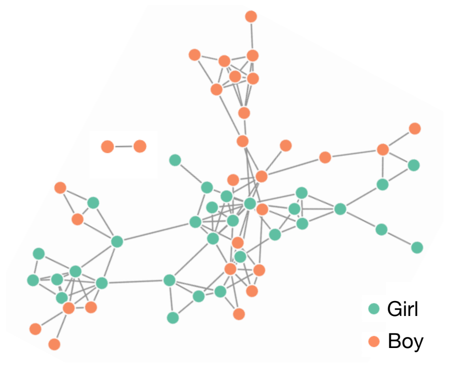


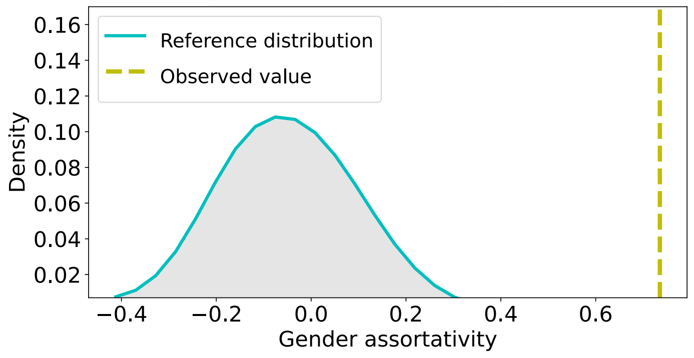


Examining the reference distributions (shown as box-and-whisker plots), we see that the main vertices (1 and 34, the president and instructor) are somewhat less central than we would expect just based on the degree structure in the network—this should make sense because these nodes are not connected to each other, and the strong two-group structure of the network pushes them away from the nodes in the center of the graph. (What other nodes are more (or less) central than expected? And, given their location in the network, can you formulate a hypothesis as to why?)

2.2 Dutch high school

As a second example, we'll examine the structure of the Dutch high school network ($n = 73$, $m = 51$; left panel below), where nodes are students, and edges represent the provision of emotional support from either node to the other. In this network, nodes are also annotated with the student's gender, either boy or girl. The question we seek to answer is whether the provision of emotional support is gendered.





First, we must rephrase this question as a question about the structure of the network. We can quantify the degree to which the gender of connected pairs of students is the same using the *assortativity coefficient*, which is a kind of Pearson correlation coefficient between the attributes on either end of an edge. We'll formally define assortativity in a later lecture; for now, it's a function f of the graph. The gender assortativity on this network is $r = 0.74$. The question of whether emotional support is gendered is then a question of whether this value is about what we would expect, given the degree structure (and node genders) of the observed network alone.

Because this graph is simple, we again use the Fosdick et al. MCMC to generate synthetic networks with the same degree sequence as the original network, but now we also maintain each node's original gender label. Tabulating the reference distribution of gender assortativities, we find that the observed value is far outside the reference distribution (right hand panel); in fact, the probability of generating a value of r at least as large as the observed value is $p < 10^{-4}$. This is a formal hypothesis test, which we constructed using the configuration model, the results of which indicate that emotional support in this network is highly gendered.

3 Taking stock of our random graph models

The configuration model is certainly an improvement over the simple random graph model in that it allows us to specify its degree structure. As a null model, this property is often sufficient for us to use the model to decide whether some other property of a network could be explained by its degree structure alone.

More generally, the configuration model shares many properties with the simple random graph model. For instance, in the sparse regime and when the degree distribution is well behaved (i.e., when it has a finite second moment) configuration model networks have locally tree-like structure. This property implies its diameter is $O(\log n)$, it has a $O(1/n)$ clustering coefficient, and $O(1/n)$ reciprocity (where the precise values of these properties depend on the structure of the degree distribution, as we saw above). The Table below summarizes these properties and compares them with the simple random graph model. As we develop more sophisticated random-graph models throughout the semester, we will expand this table.

network property	real-world	Erdős-Rényi	configuration
degree distribution	heavy tailed	Poisson($\langle k \rangle$)	specified
diameter	"small" ($\propto \log n$)	$O(\log n)$	$O(\log n)$
clustering coefficient	social: moderate non-social: low	$O(1/n)$	$O(1/n)$
reciprocity	high	$O(1/n)$	$O(1/n)$
giant component	very common	$\langle k \rangle > 1$	$\langle k^2 \rangle - 2\langle k \rangle > 0$

4 At home

1. Read Chapter 13.0–13.11 (pages 369–433) in *Networks* (2nd ed.)