1 Random graphs with specified degrees

Recall that a vertex's degree under the random graph model $G(n, p)$ follows a Poisson distribution in the sparse regime, while most real-world graphs exhibit heavy-tailed degree distributions. This difference is one reason we say that $G(n, p)$ is an unrealistic model. In this lecture, we will learn about random graph models that have a more flexible degree structure.

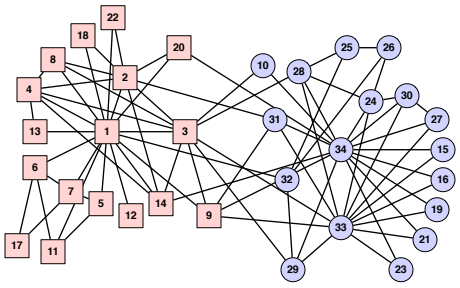
There are two main flavors of these models: those that generate random graphs with a specified degree sequence (a "configuration model") or with a specified degree distribution. These random-graph models are typically defined in terms of undirected and unweighted graphs, but they are straightforward to generalize to bipartite networks, or directed graphs. Extending them to multi-layer, temporal, or weighted networks requires additional assumptions.

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 represents a uniform distribution over all graphs with n vertices, conditioned on their having the $\vec{k}$ degree sequence. So long as $\sum k_i$ is even (do you see why?), we are free to choose $\vec{k}$ in any way we like. It then only remains to define how we will *sample* individual networks from this distribution.[1]

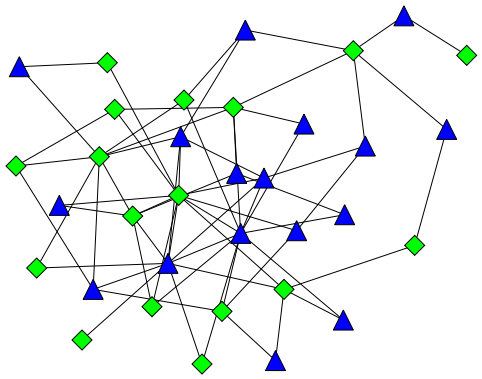
Here are some specific examples of specified configuration models. If we fix all the degrees to be the some constant k, we have defined a probability distribution over all k-regular graphs. Or, we could choose a set of n values drawn iid from some degree distribution $\Pr(k)$. If that distribution is Poisson with mean $c/(n-1)$, then we recover the $G(n, p)$ model. Or, we could draw the degrees from some other distribution, like an exponential, a log-normal or even a power law; the latter case is sometimes called a "power-law random graph." If the degree distribution that generates the degree sequence has a mathematically simple form, like the distributions just mentioned, we can often compute exactly certain properties of the corresponding ensemble of graphs, much like we did with Erdős-Rényi random graphs. This provides us with a rich family of models to study.

In practice, however, we are interested in a particular network and its degree structure, and thus a very common step in network analysis is to choose $\vec{k}$ as the empirically observed degree sequence of a real-world network. For instance, consider the karate club again. The pair of figures below show the original network, and a single example of a network drawn from the ensemble defined by the configuration model, parameterized by n and the empirical $\vec{k}$. Immediately noticeable is that while the degree structure has been preserved, the original group structure has been randomized.

[1] How we sample from the specified set of graphs turns out to be somewhat subtle, and it can matter whether one uses a hacky method or a provably correct way of doing it. These and related questions are discussed at length in a tour-de-force article B. K. Fosdick et al., "Configuring Random Graph Models with Fixed Degree Sequences." Preprint, `arxiv:1608.00607` (2016).



karate club



configuration model

1.1 A null model for network analysis

The configuration model can serve as a null model for investigating the structure of a real network A. That is, it allows us to quantitatively answer the question of

How much of some observed pattern is driven by the degrees alone?

The configuration model defines a probability distribution over graphs $\Pr(G \,|\, \vec{k})$ that has the same degrees as the original network A. Thus, if we can compute a function f on A, we can compute the same function on a graph drawn from this configuration model $f(G)$. And, because G is a random variable, we can compute the entire distribution $\Pr(f(G) \,|\, \vec{k})$. For simple functions and simple specifications of the configuration model, we can often compute these distributions analytically.

For more complicated functions or for a configuration model specified with an empirical degree sequence, we can compute $\Pr(f(G) \,|\, \vec{k})$ numerically, by drawing many graphs $\{G_1, G_2, \dots\}$ from the model, computing f on each, and tabulating the results. If the empirical value $f(A)$ is unusual relative to this distribution, we can conclude that it is a property of A that is not well explained by the degrees alone. We will revisit this idea later in this lecture.

1.2 Generating networks using the configuration model

The two most common methods for generating a random graph with particular degree structure are associated with the Chung-Lu model for random simple graphs and the Molloy-Reed model for random multigraphs.

1.2.1 Simple graphs from flipping coins

The central mathematical property of all random-graph models is the probability that two vertices i and j are connected. In the random graph models we consider here, this probability depends only on the degrees k_i and k_j of that pair. Thus, from the perspective of i, the probability that one of its edges connects to j is equal to the fraction of the m total edges we choose that point to j. Because we have chosen j's degree, this fraction is exactly $k_j/2m$. And, because we have also chosen i's degree, this event has k_i chances to occur and the probability that (i, j) exists is

$$p_{ij} = k_i \left(\frac{k_j}{2m} \right) = \frac{k_i k_j}{\sum_{\ell=1}^{n} k_\ell} \quad . \tag{1}$$

The Chung-Lu model takes this probability as a parameter and simply flips a single coin for each of the pairs i, j to generate a simple graph:

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

where p_{ij} is given by Eq. (1). Just as with generating Erdős-Rényi graphs, each pair is considered only once; hence, this process produces a simple graph, with no self-loops and no multi-edges. (In contrast, the Molloy-Reed model produces a random multigraph, which may have multi-edges and self-loops.) This method can also be used to generate directed networks by first specifying the in-degree and out-degree sequences, subject to the requirement that $\sum_i k_i^{\text{in}} = \sum_j k_j^{\text{out}}$. We then choose $p_{i \to j} = k_i^{\text{out}} k_j^{\text{in}}/m$ and drop the requirement that $A_{ij} = A_{ji}$.

As a result of this form, the degree of each vertex i under this method of generation equals the specified value k_i only in expectation (and similarly for the in- and out-degrees in the directed version). The observed degree for node i in the Chung-Lu ensemble is a Poisson distribution with mean k_i (do you see why?). Hence, deviations from the expected value are generally small, when the graph is sparse and the maximum degree is $\ll \sqrt{n}$.

Notably, drawing random graphs from the Chung-Lu model is computationally expensive, especially for large n, as we need to flip $\Theta(n^2)$ coins, one for each possible pair of vertices $i, j \in V$. This cost is one reason that the Molloy-Reed model is more commonly used for large empirical studies (but see Fosdick et al. [2018]).

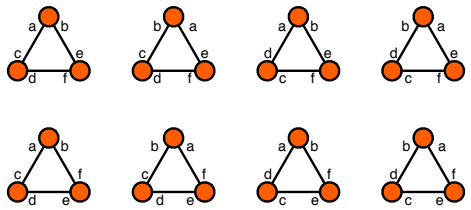
1.2.2 Multigraphs from random matchings

The standard method for generating a Molloy-Reed random multigraph is to choose a uniformly random matching on the degree "stubs" (half edges) of the specified degree sequence. Unlike in the Chung-Lu model described above, which only generates simple graphs by design, this "stub matching" method will typically produce some number of self-loops and multi-edges. In practice,

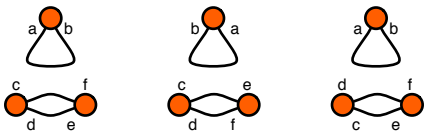
these deviations from a simple graph represent an asymptotically small fraction of all edges, and we can "simplify" the network by discarding self-loops and collapsing multi-edges, and potentially also discarding disconnected components.[2]

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.

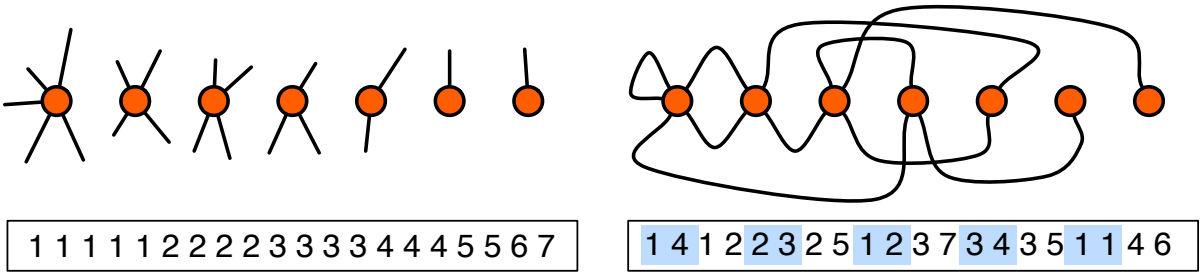


In practice, the fraction of edges involved in either self-loops or multi-edges is vanishingly small in the large-n limit, and thus we may generally ignore them without much impact. (However, there are applications in which these features are important, and so it is worth remembering that they exist.)

[2]These procedures do change the graph structure slightly, and a safer approach is to use Fosdick et al.'s (2018) methods to sample directly from the simple graph ensemble.

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 degree distribution, to draw a network from the corresponding configuration model, we simply need an efficient method by which to choose a uniformly random matching on the $\sum_i k_i$ stubs.

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 above 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[3] on the

[3]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

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 Complications: graph spaces

The Chung-Lu model creates simple graphs with degree sequence $\vec{k}$ *in expectation*, but in practice, the realized degrees of a Chung-Lu network may differ from that expectation. The matching method described directly above creates networks with degree sequence exactly $\vec{k}$, but with the possibility of self-loops and multiedges. What should be done if we wish to generate networks that are, say, simple, but have degree sequence *exactly* $\vec{k}$?

Rather than writing down a solution for only simple graphs, we'll write down more generic solutions for entire classes of graphs. Note that graphs may have self loops, multiedges, both, or neither. For precision, we'll call these *loopy* graphs, *multigraphs*, *loopy multigraphs*, or *simple* graphs.

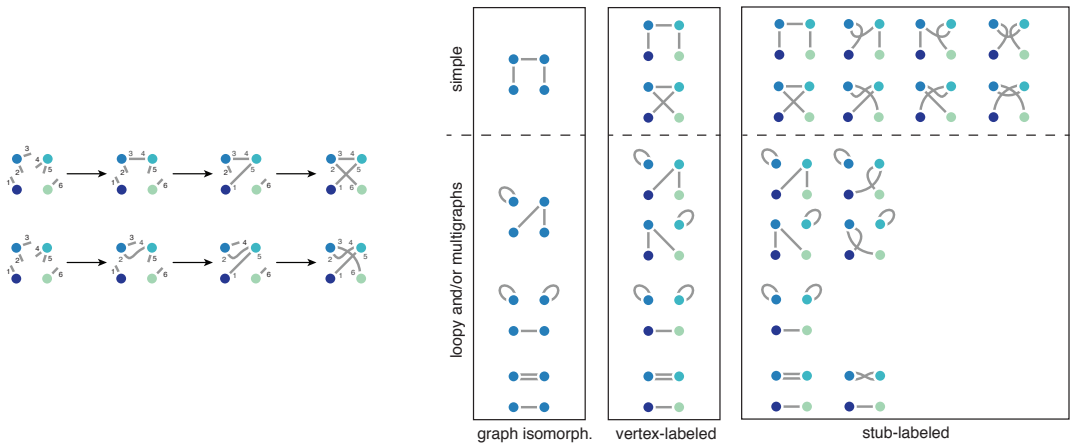
But things get more complicated. Recall that the configuration model is a model for *random* networks with a fixed degree sequence, meaning that, of the set of possible networks created by the model, we wish for each network to be drawn from the set with equal probability. In other words, the configuration model can be thought of as a uniform distribution over the set of networks with a fixed degree sequence. This means that simply drawing a network is insufficient if we cannot also guarantee that it was a uniform random draw, i.e. that each of the n members of the set has the same associated probability $1/n$. Without such guarantees, the configuration model, as a null model, is useless due to bias.

It turns out that, beyond self-loops and multiedges, one additional question needs to be considered: do the stubs of the network have identities? This motivates two definitions. Definition (vertex-labeled graph): A *vertex-labeled graph* is a graph in which each vertex has a distinct label. Definition (stub-labeled graph). 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. 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 $\vec{k} = \{1, 2, 2, 1\}$.

Having decided whether to draw from a vertex- or stub-labeled space of { simple graphs, loopy graphs, multigraphs, loopy multigraphs}—see [4] for guidance on this decision—how does one actu-

$i \leq j \leq n$. The proof that this produces a random permutation follows the proof above.

[4] "Configuring random graph models with fixed degree sequences," Fosdick, Larremore, Nishimura, and Ugander.



ally draw a network uniformly at random from such a space? One intuitive solution for stub-labeled spaces is to use standard stub-matching, which draws from the space of stub-labeled loopy multigraphs, and to then discard all graphs that do not belong in the space. This is a form of rejection sampling, and may be efficient or inefficient, depending on the probability of generating multiedges and self-loops (see section below).
Rejection sampling can also be used for vertex-labeled spaces by *up-weighting* graphs with multiedges or self-loops, depending on the exact counts of each. We now turn to counting graph configurations.

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 multiedges? 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

SIAM Review **60** (2) 315-355.

(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.

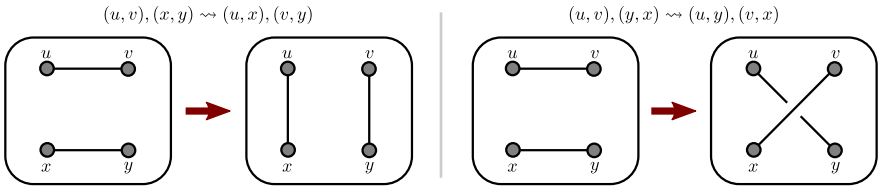
$$\begin{aligned} q_{\text{loopy}}(G) &= q_{\text{simple}}(G) \frac{1}{\prod_{i=1}^{n} w_{ii}!(2^{w_{ii}})} \\ q_{\text{multi}}(G) &= q_{\text{simple}}(G) \frac{1}{\prod_{i<j} w_{ij}!} \\ q_{\text{loopy multi}}(G) &= q_{\text{simple}}(G) \frac{1}{\prod_{i=1}^{n} w_{ii}!(2^{w_{ii}})} \times \frac{1}{\prod_{i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

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.

1.3.2 Alternative to stub-matching: MCMC and the double-edge swap

Rather than trying to assemble a network uniformly at random, an alternative approach is to engineer a process that will wander over the set of networks in the desired space, which can be sampled from time to time. This process, called Markov chain Monte Carlo (MCMC) involves creating a stochastic process that will provably (i) reach all graphs in the space and (ii) spend an equal amount of time on each graph.

The engine of this process is the *double-edge swap*, pictured below. In it, two edges are cut in half, and the network is rewired in either of the two possible ways. As a consequence, the graph itself changes, but the degree sequence is preserved. By repeating this process over and over, drawing the two edges uniformly at random from the edge set, the network's edges may be randomized. This process is a Markov chain. Each state of the chain is a configuration, and transitions between configurations occur via these double-edge swaps.



If edges are chosen uniformly at random, and all proposed swaps are accepted, the Markov chain will uniformly explore the space of stub-labeled loopy multigraphs. To sample from a stub-labeled

space that excludes self-loops, multiedges, or both, when one of the offending edge configurations is selected by a double-edge swap, the existing network (without the offending swap) is resampled in its place—effectively creating a "self loop" in the Markov chain. To sample from vertex-labeled spaces, adjustments to the transition probabilities allow the process to uniformly sample the vertex-labeled space instead of the stub-labeled space. For the case of loopy graphs, with no multiedges, some additional adjustments need to be made so that all networks are sampled, eventually. [5] These issues and algorithms are provided in Fosdick 2018.

1.4 Mathematical properties

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).[6]

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.

[5] See "Swap connectivity for two graph spaces between simple and pseudo graphs and disconnectivity for triangle constraints" 2017 on the arXiv, by Joel Nishimura.

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

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

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

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[7] 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_ik_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

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

[7] 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.

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.

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:

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

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}:

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

(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).

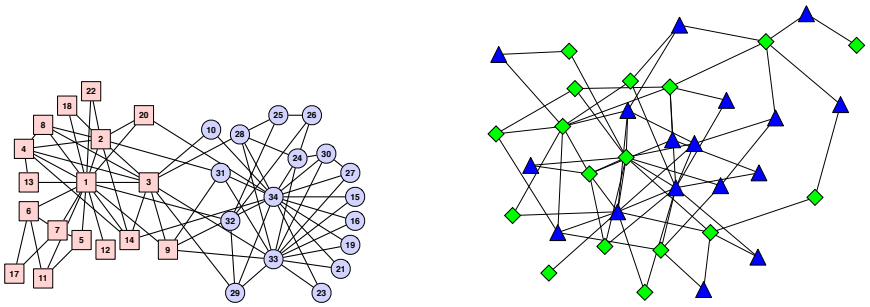
1.5 Directed random graphs

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?)

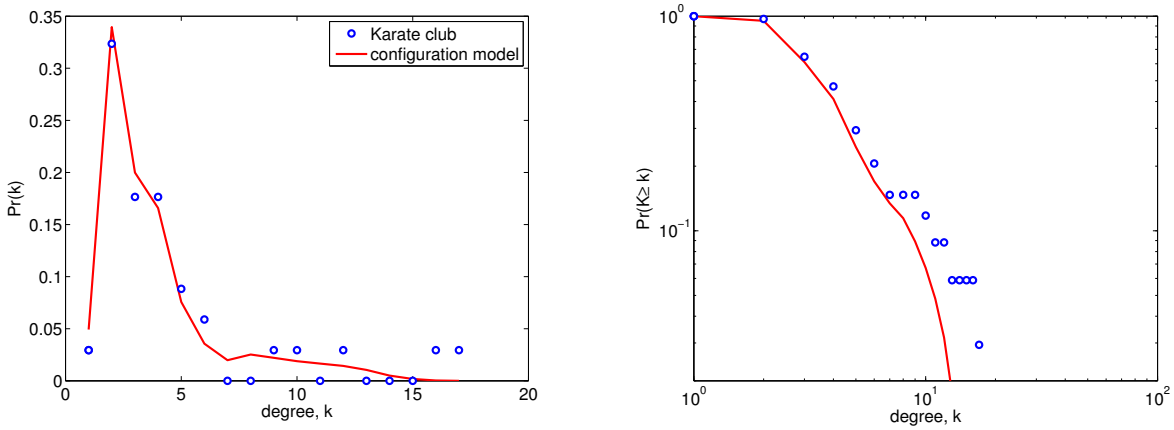
Constructing directed random graphs using the configuration model is also analogous, but with one small variation. Now, instead of maintaining a single array v containing the names of the stubs, we must maintain two arrays, v_{in} and v_{out}, each of length m, which contain the in- and out-stubs respectively. 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 A null model for empirical networks

The most common use of the configuration model in analyzing real-world networks is as a null model, i.e., as an expectation against which we measure deviations. Recall from the last lecture our example of the karate club, and an instance drawn from the corresponding configuration model. Using the configuration model to generate many such instances, we can use each network as input to our structural measures. This produces a distribution of measures, which we can then compare directly to the empirical values. Each of the mini-experiments below used 1000 instances of the configuration model, and where multi-edges were collapsed and self-loops discarded.



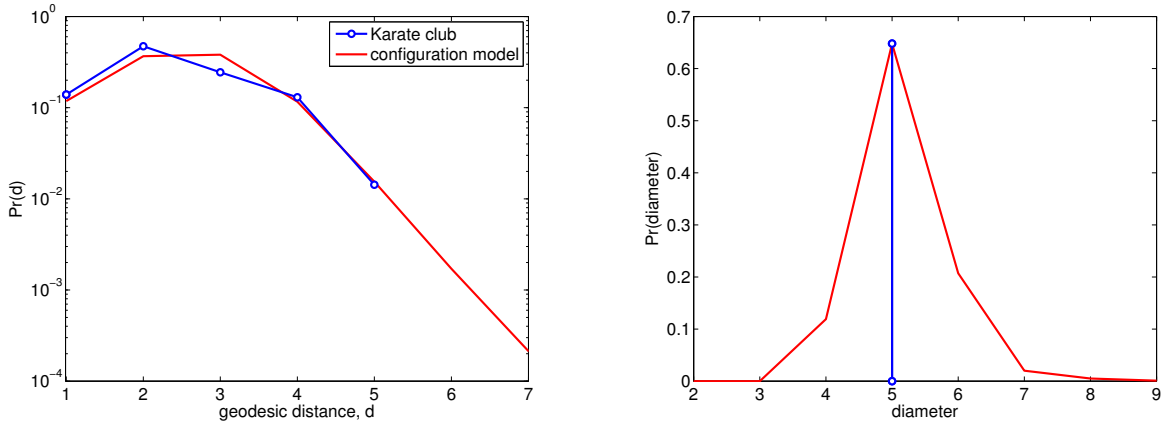
The degree distribution (shown below as both pdf and ccdf) is very similar, but with a few notable differences. In particular, there the highest-degree vertices in the model have slightly lower degree values than observed empirically. This is reflects the fact that both multi-edges and self-loops have a higher probability of occurring if k_i is large, and thus converting the generated network into a simple network tends to remove edges attached to these high-degree vertices. Otherwise, the generated degree distribution is very close to the empirical one, as we expect.[8]



Both the distribution of pairwise geodesic distances and the network's diameter are accurately reproduced under the configuration model, indicating that neither of these measures of the network are particularly interesting as patterns themselves. That is, they are about what we would expect

[8]It is possible to change the configuration model slightly in order to eliminate self-loops and multi-edges, by flipping a coin for each pair i, j (where $i \neq j$) with bias exactly $k_i k_j / 2m$. In this model, the expected degree we generate is distributed as $\hat{k}_i \sim \text{Poisson}(k_i)$, which assumes the specified value in expectation.

for a random graph with the same degree distribution. One nice feature of the configuration model's pairwise distance distribution is that it both follows and extends the empirical pattern out to geodesic distances beyond what are observed in the network itself.



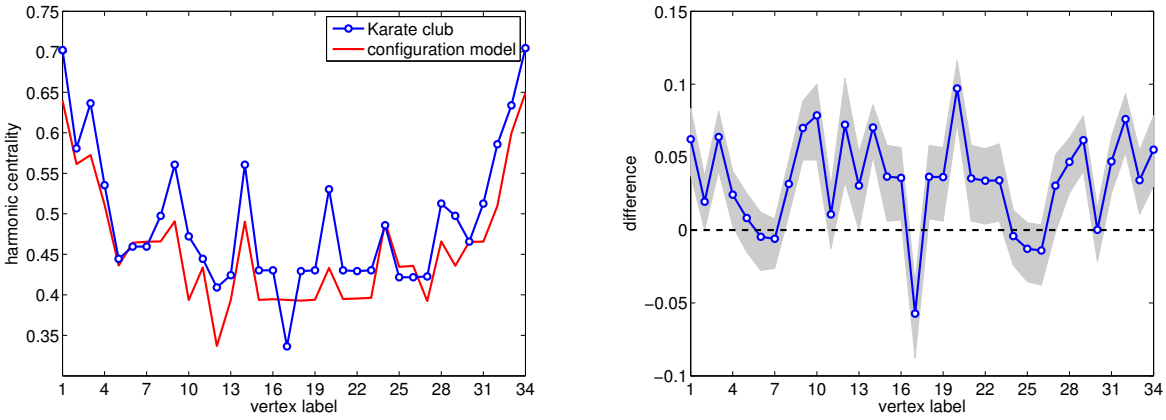
We may also examine vertex-level measures, such as measures of centrality. From the geodesic distances used in the previous figures, we may also estimate the mean harmonic centrality of each vertex. The first figure below plots both the empirical harmonic centralities (in order of vertex label, from 1 to 34) and the mean values under the configuration model. The various centrality scores are now placed in context, showing that their scores are largely driven by the associated vertex degree, as demonstrated by the similar overall pattern seen in the configuration model networks.[9]

But, not all of the values are explained by degree alone. The second figure plots the difference between the observed and expected centrality scores Δ, where the line $\Delta = 0$ indicates no difference between observed and expected values. If an observed value is above this line, then it is more central than we would expect based on degree alone, while if it is below the line, it is less central.

When making such comparisons, however, it is important to remember that the null model defines a distribution over networks, and thus the difference is also a distribution. Fortunately, however, computing the expected centrality scores by drawing many instances from the configuration model also produces the distribution of centrality scores for each vertex, which provides us with a quantitative notion of how much variance is in the configuration model value. The grey shaded region shows the 25 and 75% quantiles on the distribution of centrality scores for each vertex. When the $\Delta = 0$ line is outside of this range, we may claim with some confidence that the observed value is

[9]Recall also that the Pearson correlation coefficient for harmonic centrality and degree was large $r^2 = 0.83$, a fact that reinforces our conclusion here.

different from the expected value.



This analysis shows that the main vertices (1 and 34, the president and instructor) are somewhat more central than we would expect just based on their degree alone. In fact, most vertices are more central than we would expect, one is less central than we expect, and about a third of the vertices fall in line with the expectation.

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 428–483) in *Networks*