Chapter 4

This is not the chapter you're looking for [handwave]

The plan of this course was to have 4 parts, and the fourth part would be half on analyzing random graphs, and looking at different models, and half on other issues in social networks such as diffusion (disease spread, or information) and graph partitioning (as with the Karate club). Well, we just finished the third part with one lecture left, so needless to say we won't get that far. I hope to reteach this course again in the future as a year-long course, and at that point I can expand and revise these notes further (and maybe actually include a proper reference section).

Here's just a little taste of things.

4.1 Two random results

To analyze random graphs, one typically proves (or at least heurstically argues) that some property is true for most graphs, possibly by proving it in the limit.

Proposition 4.1.1. *For $n \geq 4$, most graphs on n vertices are connected.*

This is a cute result motivated by a MathOverflow post I came across recently. Here "most" just means more than half, and it is valid if we are counting labelled graphs or unlabelled graphs. For $n = 1$ it is all graphs, and for $n = 2$ or $n = 3$ it is exactly half the graphs.

Proof. Suppose $G = (V, E)$ is disconnected. Then I claim the complement, $\bar{G} = (V, \bar{E})$, is connected. Here $\bar{E} = \{(u, v) : u \neq v, (u, v) \notin E\}$ is the set complement of E inside $V \times V - \Delta V = \{(u, v) : u, v \in V, u \neq v\}$. In other words, all edges of G are not edges off $\bar{G}$, and vice versa. Put another way, the adjacency matrix $\bar{A}$ of $\bar{G}$ is formed by changing all non-diagonal entries of A from 0's to 1's or 1's to 0's.

To prove the claim, we want to show any $u, v \in V$ lie in the same component in $\bar{G}$. If u, v are in different components in G, then they must be connected by an edge in $\bar{G}$, hence in the same component in $\bar{G}$. If they are in the same component in G, take $w \in V$ in a different component in G, so u and v are connected to w in $\bar{G}$, and again in the same component. This shows the claim.

If there are m disconnected graphs on n vertices (labelled or unlabelled, the argument is the same), taking complements gives m connected graphs (whose complement is disconnected, as the complement of the complement of G is simply G). Now we just observe that for $n \geq 4$, there

are other connected graphs, i.e., there are connected graphs whose complements are connected. We can, for example, just take the path graph on n vertices (exercise—check its complement is connected for $n \geq 4$). □

Proposition 4.1.2. *Fix $0 < p \leq 1$. In the random $G(n,p)$ model, the probability of having an isolated node goes to 0 as $n \to \infty$.*

Proof. Let G be a $G(n,p)$ graph. The probability a given vertex i is isolated is simply $(1-p)^{n-1}$. (Vertex i pairs with $n-1$ other vertices, and all of these pairs are non-edges with independent probability $1-p$.) If A_i denotes the event that vertex i is isolated, the probability that G has at least one isolated node is

$$P(A_1 \cup \cdots \cup A_n) \leq P(A_1) + \cdots + P(A_n) = n(1-p)^{n-1}.$$

But this goes to 0 as $n \to \infty$. □

This says that for large n, we don't expect isolated nodes.

These results hint at the idea that random graphs tend to be fairly well connected, at least if n is large. Indeed, one can prove under certain conditions on n, p statements like: almost all random $G(n,p)$ graphs have a *giant component*, or almost all random $G(n,p)$ graphs are connected, or almost all random $G(n,p)$ graphs have a clique of size m.

4.2 Spectral graph partitioning

Take a graph G with transition matrix T. Assume for simplicity that T has a basis of eigenvectors $v_1, \ldots, v_n$ with corresponding eigenvalues $\lambda_1, \ldots, \lambda_n$. As before, we order these so that $1 = \lambda_1 \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$. Further assume $1 > |\lambda|_2 > |\lambda_3|$.

Take a random walk on G with initial state v_0. Write

$$v_0 = c_1 v_1 + \cdots c_n v_n.$$

Then the t-th state of the random walk is

$$\begin{aligned} v_t = T^t v_0 &= c_1 \lambda_1^t v_1 + c_2 \lambda_2^t v_2 + \cdots + c_n \lambda_n^t v_n \\ &= c_1 v_1 + c_2 \lambda_2^t v_2 + \cdots + c_n \lambda_n^t v_n. \end{aligned}$$

For t of moderate size, λ_i^t are really small for $i \geq 3$, so

$$v_t \approx c_1 v_1 + c_2 \lambda_2^t v_2. \tag{4.1}$$

There is a lot packed into this approximation. We've seen already that v_1 gives a centrality ranking of vertices and λ_2 measures the rate of convergence of the random walk, i.e., network flow. We might call λ_2 the second dominant eigenvector and v_2 the second dominant eigenvector. This is because λ_2 and v_2 are the second most important things (after $\lambda_1 = 1$ and v_1) in determining the behavior of the random walk. Namely $|\lambda_2|$ measures how fast the random walk converges, and v_2 tells us *how* it actually converges.

One way to see this is to look at

$$v_{t+1} - v_t \approx c_2 \lambda_2^t (\lambda_2 - 1) v_2. \tag{4.2}$$

So in some sense, v_2 is essentially a "directional derivative" of this random walk. (Apologies again for the conflicting notation of v_2 and v_t—v_2 is not v_t with $t = 2$. Remind me to change this in the next version of these notes.)

Example 4.2.1. *Let G be the path graph on 2 vertices. Then we can take $\lambda_1 = 1$, $\lambda_2 = -1$, $v_1 = (1, 1)$ and $v_2 = (1, -1)$. (Okay, here $\lambda_1 \not> \lambda_2$, but this is the simplest example possible, and I think it is somewhat illustrative anyway.) Any random walk will just flip back and forth between vertices 1 and 2. From the derivative point of view, we can think of*

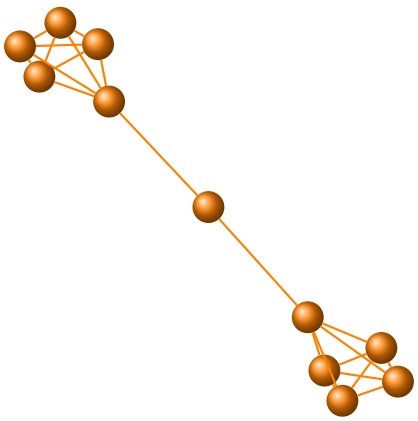
$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = c\lambda_2^t v_2 = (-1)^t c \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

as representing the approxmate change in v_t at time t. If $y_1 > 0$ then this means we're more likely to be at vertex 1 at the next step, and if $y_1 < 0$ this means we're less likely to be at vertex 1 at the next step. Similarly the sign of y_2 tells us if we're more or less likely to be at vertex 2 at the next step.

Now let's engage in a though experiment with the above example: suppose we had $1 > \lambda_2 > 0$. Then the signs of y_i's don't change at each step, and the random walk won't keep flipping back and forth between vertices. Say $y_1 < 0$ and $y_2 > 0$. This would mean that at every step we're getting more and more likely to be at vertex 2 and less likely to be at vertex 1. In other words, the random walk is getting "pulled" towards vertex 2.

This suggests that the signs of the entries of v_2 can tell us if a random walk is converging towards or away from from certain vertices. Note that the signs of the entries of v_2 are not determined uniquely, since $-v_2$ is also an eigenvector with eigenvalue λ. What will be important for us is which vertices have the same sign (assuming v_2 is real)—this is uniquely determined.

Consider the following graph, which is two copies of K_5 connected by a single vertex with two edges.



If we start a random walk on the left copy of K_5, it will wander around there for awhile, but it will gradually get pulled to the other side.

In Sage, I computed the transition matrix (the single "bottleneck" or "bridge" vertex in the middle is the last vertex in this ordering), the eigenvalues and dominant and second dominant eigenvectors.

Sage 6.1

```
sage: T
[  0 1/4 1/4 1/4 1/4   0   0   0   0   0 1/2]
```

```
[1/5   0 1/4 1/4 1/4   0   0   0   0   0   0]
[1/5 1/4   0 1/4 1/4   0   0   0   0   0   0]
[1/5 1/4 1/4   0 1/4   0   0   0   0   0   0]
[1/5 1/4 1/4 1/4   0   0   0   0   0   0   0]
[  0   0   0   0   0   0 1/4 1/4 1/4 1/4 1/2]
[  0   0   0   0   0 1/5   0 1/4 1/4 1/4   0]
[  0   0   0   0   0 1/5 1/4   0 1/4 1/4   0]
[  0   0   0   0   0 1/5 1/4 1/4   0 1/4   0]
[  0   0   0   0   0 1/5 1/4 1/4 1/4   0   0]
[1/5   0   0   0   0 1/5   0   0   0   0   0]
sage: T.eigenvalues()
[1, -1/4, -1/4, -1/4, -1/4, -1/4, -1/4, -0.2086308764964376?, 0.9586308764964376?,
-0.5319705149024926?, 0.2819705149024927?]
sage: ev = T.eigenvectors_right()
sage: ev[0][1] # dominant eigenvector
[
(1, 4/5, 4/5, 4/5, 4/5, 1, 4/5, 4/5, 4/5, 4/5, 2/5)
]
sage: ev[3][1] # next dominant eigenvector
[(1, 0.9586308764964376?, 0.9586308764964376?, 0.9586308764964376?,
0.9586308764964376?, -1, -0.9586308764964376?, -0.9586308764964376?,
-0.9586308764964376?, -0.9586308764964376?, 0)]
```

Here we see $\lambda_2 \approx 0.9586$ is large, i.e., the spectral gap is small, i.e., the network flow is bad, and this is due to the bottleneck as discussed in the last chapter. In the second dominant eigenvector v_2, all signs for the left hand K_5 are positive, while all signs for the right hand K_5 are negative, and the vertex in the middle has value 0.

This means in a random walk, when t is of moderate size so the behavior is dominated by v_2, we have $v_t \approx c_1 v_1 + \lambda_2^t c_2 v_2$. If we started our random walk on the left hand K_5, then c_2 will be negative, so the signs of the entries of v_2 will be negative for the left hand K_5 and positive for the right-hand K_5. This means the random walk is getting "pulled" towards the right hand K_5. That is, if we start on the left, for quite some time we are much more likely to be on the left, but we gradually become more likely to be on the right hand K_5. On the other hand, if we started on the right hand K_5, then c_2 would be positive and the random walk would get pulled to the left.

We can use this idea to partition graphs. Let's just think about the problem of bipartitioning, i.e., splitting partitioning a graph into two subsets V_1 and V_2. Roughly the idea is to partition so that the subgraphs corresponding to V_1 and V_2 should be well connected, but there shouldn't be too many connections between V_1 in V_2. Hence if we start a random walk in V_1, it will stay in V_1 for awhile, then eventually get pulled towards V_2, and vice versa. Thus if we look at the second dominant eigenvector v_2, the signs of the entries for the V_1 vertices should be (say) positive, while for V_2 will be (say) negative. If the value of the entry is 0 (or maybe close to 0), we can say that vertex is "on the fence." Note if we apply this algorithm to the above example of two K_5's connected to a single vertex, it does perfectly—the signs of the entries of v_2 partition the graph into 3 pieces: the two K_5's and the middle vertex which has value 0.

To close the semester, let's revisit the Karate club graph.

Sage's innate linear algebra functions are not so fast (Sage's focus is on precision, not speed), so we'll use `numpy`, a numerical method package for Python.

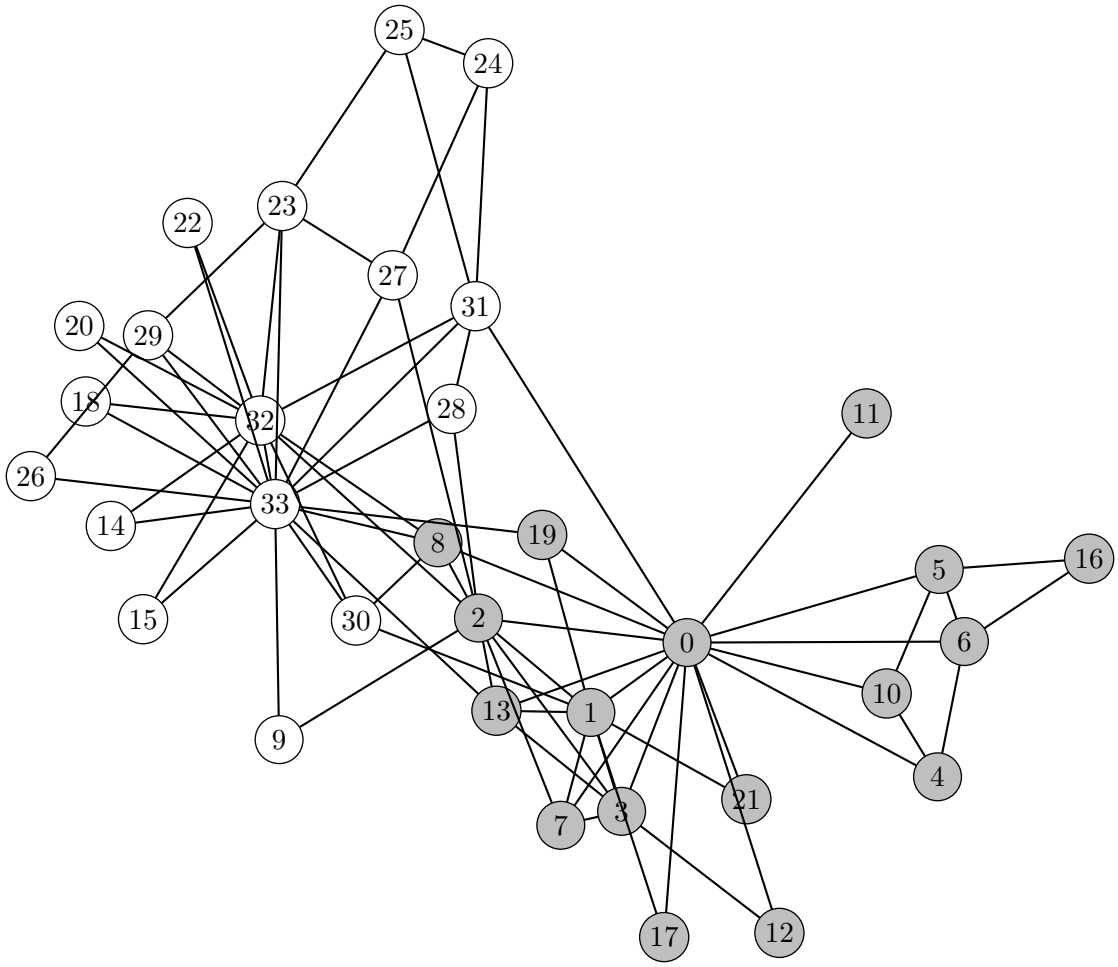


Figure 4.1: The Karate Club, actual split

Sage 6.1

```
sage: from numpy.linalg import eig
sage: w, v = eig(T)
sage: w # eigenvalues
array([  1.00000000e+00 +0.00000000e+00j,
         8.67727671e-01 +0.00000000e+00j,
         7.12951015e-01 +0.00000000e+00j,
         6.12686767e-01 +0.00000000e+00j,
        -7.14611347e-01 +0.00000000e+00j,
         3.87769460e-01 +0.00000000e+00j,
         3.51007053e-01 +0.00000000e+00j,
         2.92791798e-01 +0.00000000e+00j,
         2.60042011e-01 +0.00000000e+00j,
         2.29089383e-01 +0.00000000e+00j,
         1.77057148e-01 +0.00000000e+00j,
         1.35167055e-01 +0.00000000e+00j,
         9.31839984e-02 +0.00000000e+00j,
        -1.05380839e-01 +0.00000000e+00j,
```

```
        -1.59299956e-01 +0.00000000e+00j,
        -2.68023547e-01 +0.00000000e+00j,
        -6.11909588e-01 +0.00000000e+00j,
        -5.69506603e-01 +0.00000000e+00j,
        -3.51778259e-01 +0.00000000e+00j,
        -3.93104541e-01 +0.00000000e+00j,
        -4.16915851e-01 +0.00000000e+00j,
        -4.48579382e-01 +0.00000000e+00j,
        -4.97030113e-01 +0.00000000e+00j,
        -5.83333333e-01 +0.00000000e+00j,
        -8.49622483e-17 +0.00000000e+00j,
        -1.17901012e-16 +0.00000000e+00j,
        -5.32777279e-17 +0.00000000e+00j,
         3.97274886e-17 +1.75808521e-17j,
         3.97274886e-17 -1.75808521e-17j,
         3.63910315e-17 +0.00000000e+00j,
        -5.74878926e-18 +0.00000000e+00j,
         2.32016470e-18 +1.32924556e-17j,
         2.32016470e-18 -1.32924556e-17j,   1.52951722e-17 +0.00000000e+00j])

sage: v[:,0] # dominant eigenvector v_1
array([ 0.45958799+0.j,  0.25851825+0.j,  0.28724249+0.j,  0.17234550+0.j,
        0.08617275+0.j,  0.11489700+0.j,  0.11489700+0.j,  0.11489700+0.j,
        0.14362125+0.j,  0.05744850+0.j,  0.08617275+0.j,  0.02872425+0.j,
        0.05744850+0.j,  0.14362125+0.j,  0.05744850+0.j,  0.05744850+0.j,
        0.05744850+0.j,  0.05744850+0.j,  0.05744850+0.j,  0.08617275+0.j,
        0.05744850+0.j,  0.05744850+0.j,  0.05744850+0.j,  0.14362125+0.j,
        0.08617275+0.j,  0.08617275+0.j,  0.05744850+0.j,  0.11489700+0.j,
        0.08617275+0.j,  0.11489700+0.j,  0.11489700+0.j,  0.17234550+0.j,
        0.34469099+0.j,  0.48831224+0.j])

sage: v[:,1] # second dominant eigenvector v_2
array([ 0.48059815+0.j,  0.13792141+0.j, -0.01149982+0.j,  0.11431394+0.j,
        0.18758378+0.j,  0.28082530+0.j,  0.28082530+0.j,  0.07290804+0.j,
       -0.04788581+0.j, -0.03189334+0.j,  0.18758378+0.j,  0.03461614+0.j,
        0.05657271+0.j,  0.04233999+0.j, -0.06450152+0.j, -0.06450152+0.j,
        0.16181649+0.j,  0.05227675+0.j, -0.06450152+0.j,  0.02170870+0.j,
       -0.06450152+0.j,  0.05227675+0.j, -0.06450152+0.j, -0.17767894+0.j,
       -0.09509451+0.j, -0.10191495+0.j, -0.07308313+0.j, -0.10937613+0.j,
       -0.05632550+0.j, -0.14756600+0.j, -0.05787797+0.j, -0.12720276+0.j,
       -0.35334004+0.j, -0.45092074+0.j])
```

This says the second dominant eigenvalue $\lambda_2 \approx 0.8677$. By looking at the signs of the entries of v_2, we are led to the partition of the Karate club graph in Figure 4.2.

Note vertices 8 and 2 are classified as going with 32 and 33, rather than with 0 as actually happened. However this makes sense from an algorithmic point of view: 8 is friends with 30, 32 and 33 which, so will more likely go with those; then 2 is friends with 32, 28, 8 and 9 and may go with them. It in fact makes more sense for at least 8 to go with 33 than with 0, based solely on the information in the graph. (Factors that may have played a role, such as strength of ties, convenience and beliefs are not accounted for in this graph.)

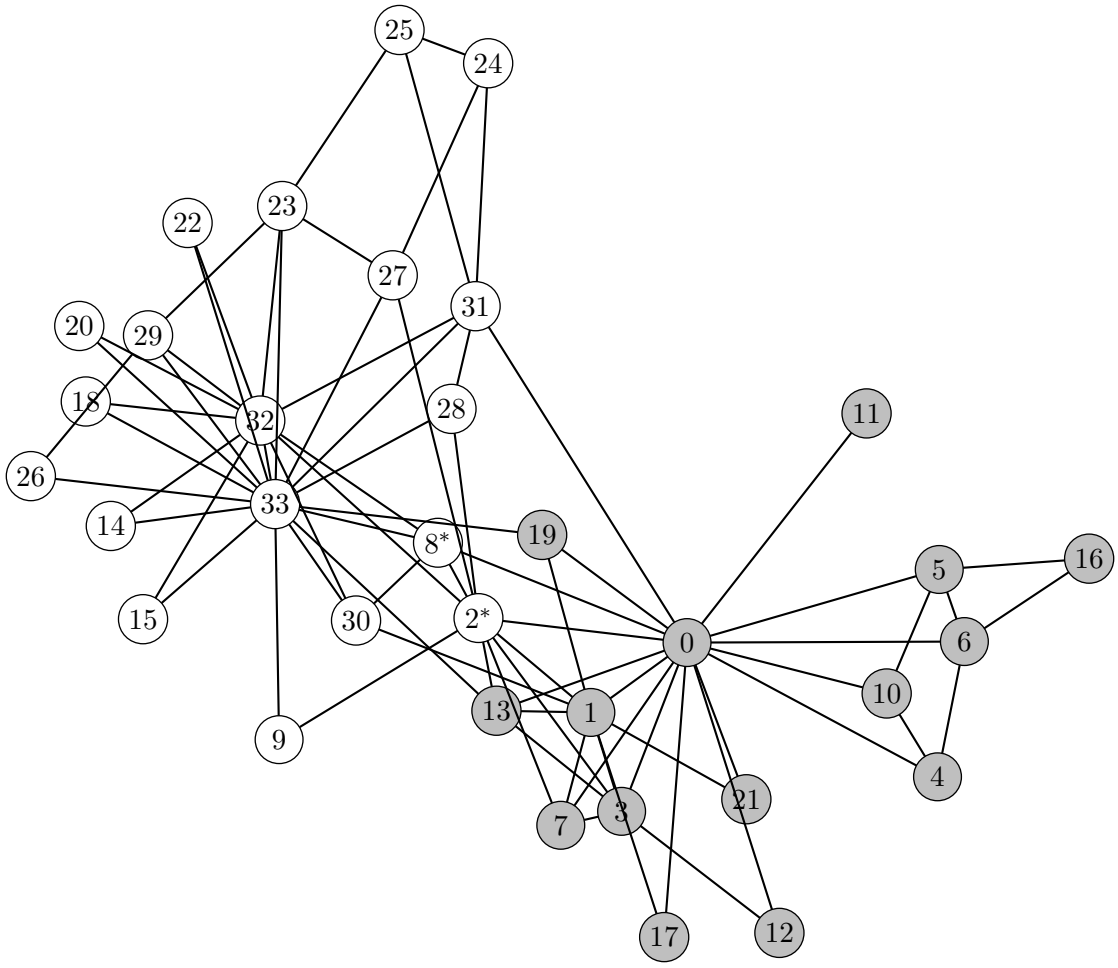


Figure 4.2: The Karate Club, spectral partition (starred vertices mispartitioned)

However, it is true that the values of v_2 for vertices 2 and 8, -0.0114... and -0.04788... are quite close to 0, so they're "on the fence" from this spectral partitioning point of view (along with some other vertices like 9 and 13).

This technique also falls under the heading of spectral clustering, as it divides the graph into 2 "clusters" (or more with repeated application). However, if $\lambda_2 < 0$, one won't get a partition into 2 clusters but into 2 whatever-the-opposite-of-clusters-are. Namely if your graph is close to bipartite, e.g., one can partition the vertices into V_1 and V_2 such that most connections are from V_1 to V_2, rather than within V_1 and within V_2, this spectral partitioning method will separate V_1 and V_2. If G actually is bipartite (and connected) with partition $V_1 \cup V_2$, then $\lambda_2 = -1$ and all entries of v_2 for V_1 will have the same sign, while V_2 will have the opposite sign, so this method will split V_1 from V_2, as in Example 4.2.1.

Well, that's all. Have a good life. If there's one thing you take away from this course, make it this: random walks are f**king awesome.