Null Models For Social Networks

Su, Jessica
jtysu@stanford.edu

Wu, Sen
senwu@stanford.edu

October 16, 2013

Abstract

It is often useful to compare social networks to models. By doing this we can tell if observed properties are peculiar to the specific network, or if they are common to all networks of that type. We can also generate synthetic networks to quickly test hypotheses about network structure.

One way to characterize a social graph is by specifying its subgraphs. For example, a graph might have ten triangles, or seven 4-cliques. Our goal is to generate random graphs that satisfy these properties. This improves upon standard models since it accounts for features specific to social networks, like triadic closure.

1 Related work

1.1 Milo et al., On the uniform generation of random graphs with prescribed degree sequences [1]

This paper presents algorithms for generating random graphs with certain degree sequences.

One method they proposed was to start with a given graph, then perform random transpositions on the edges. That is, choose two edges at random, then exchange their ends to produce new edges. This is interesting because it leaves both the number of edges and the degree distribution constant. Therefore we can start with a graph with the right degree distribution, then perturb it until we get the right number of motifs. We can use hill climbing

or Markov Chain Monte Carlo to converge to the correct solution. However, this hasn't worked on our graphs since real graphs have tens of thousands of edges and changing two edges at a time leads to very slow convergence.

Another method is the "matching algorithm," in which vertices of degree k are assigned k spokes. Then the spokes are joined at random to form edges. Unfortunately, the paper proposes discarding the entire network if a multiple edge is created. This means the algorithm may never run to completion. An alternative strategy is to discard the multiple edge and create a new one, but the paper claims this produces a biased sample of random graphs. If we don't care that the sample is slightly biased, we can use a variant on this to solve our problem. This is partially described in Karrer and Newman [2] and we discuss it later.

To solve problems associated with the matching algorithm, the authors propose the "go-with-the-winners" algorithm, which is less efficient but somewhat more accurate. In this algorithm, we start with M graphs, and proceed as in the matching algorithm. If a multiple edge is created in one of the graphs, that entire graph is discarded. Each of the surviving graphs are periodically cloned so we don't lose all our graphs by discarding them. This is slow but the resulting sample is unbiased. The transposition algorithm also produces an unbiased sample, but wouldn't work for our problem.

1.2 Karrer and Newman, Random graphs containing arbitrary distributions of subgraphs [2]

This is an interesting paper since it seems to completely solve our problem. The paper proposes an algorithm for generating graphs with arbitrary distributions of subgraphs, then uses fancy math to prove properties of the resulting networks.

The algorithm is similar to the "matching algorithm" from the Milo paper. Instead of creating edge stubs for each vertex, we create "motif stubs" that indicate which subgraphs a vertex is part of. They also have to indicate which role a vertex plays in the subgraph. (For example, in the subgraph with three vertices and two edges, there are two different roles: the "end" vertex and the "corner" vertex.) Then we connect matching motif stubs at random to create the random graph.

There are two main problems with this. The first is that we need to be told exactly what subgraphs each vertex is part of and which roles they play.

This is not obvious if we are only given the motif counts. The second is that connecting certain motif stubs may automatically connect other motif stubs - for example, if two vertices are part of a 4-clique, they are also part of an edge. In order to implement this algorithm we must correct for these issues.

The paper goes on to calculate the size of the giant connected component and the point at which the component appears. This part does not have direct application to my project.

1.3 Ugander et al., Subgraph Frequencies: Mapping the Empirical and Extremal Geography of Large Graph Collections [3]

This paper proposes a model called the "Edge Formation Random Walk." It proves that you can produce an Erdos-Renyi random graph by doing a random walk on a graph. On each step you choose an edge randomly and add it with probability γ. You can also delete an edge with probability δ. Under this model, the fractions of 3- and 4-motifs should eventually reach a stationary point, and at that point they should be equal to the expected frequencies in the Erdos-Renyi random graph.

Ugander modifies this model to incorporate triadic closure, a property specific to social networks. Triadic closure is the property that if person A is friends with person B, and person B is friends with person C, then person A is more likely to be friends with person C. In network models this manifests as an abundance of triangles. In the Ugander paper, instead of adding an edge with probability γ, we add the edge with probability $\gamma + k\lambda$, where k is the number of triangles that would be formed by adding the edge.

We can use the equilibrium condition to analytically calculate the stationary points of the random walk. At equilibrium, the number of triangles should be constant, so the number of open triads that become triangles (in one timestep) should be the same as the number of triangles that lose an edge. Therefore we can write equations like (Number of open triads) $(\gamma + \lambda)$ = (Number of triangles) (δ). If we do this for the 3- and 4-motifs we can solve the linear system for the equilibrium quantities of each subgraph.

We minimized the error over the parameters and discovered that at the optimal parameters, the theoretical error is pretty poor. Indeed, the Erdos-Renyi random graph model produces better error than this model. On larger graphs, the error is considerably worse (over 50%), although we shouldn't

compare the errors because I changed the error function in the middle of the analysis. The error gets even worse when I actually try to generate the random graphs instead of just solving for the equilibrium solution.

There are at least four reasons why this might happen. First, there may be a bug in the code. In fact, I thought of a bug in the course of writing this proposal. Second, k might be large in large networks, so that $\gamma + k\lambda$ is greater than 1. In the simulation, the edge creation probability is strictly capped at 1, which produces different results than in the model. I attempted to correct for this by making the edge creation probability a logistic function of the number of triangles created instead of a linear function, but this made the theoretical error substantially worse.

Third, the model does not consider the subgraphs as part of a larger graph. When calculating the stationary solution we assume that the number of triangles created is (Number of open triads) $(\gamma + \lambda)$. But actually the probability of creating a triangle should depend on the other triangles that are created simultaneously.

Lastly, although this model has been tested empirically, it's only been tested on "ego networks," which are networks centered around one person. An example of an ego network would be the connections between someone's Facebook friends. The networks we are modeling are different since we have the entire network and no particular vertex is at the center.

2 Project proposal

2.1 Problem statement

It is often useful to compare social networks to models. By doing this we can tell if observed properties are peculiar to the specific network, or if they are common to all networks of that type. We can also generate synthetic networks to quickly test hypotheses about network structure.

One way to characterize a social graph is by specifying its subgraphs ("motifs"). For example, a graph might have ten triangles, or seven 4-cliques. Our goal is to generate random graphs that satisfy these properties. Specifically, we must make sure that each satisfactory graph is equally likely to be chosen.

For simplicity we assume that each motif has fewer than 5 vertices.

2.2 Proposal

I propose to follow the stub matching algorithm in the Karrer paper [2]. While most of the algorithm has been handed to us, we have to change two things to make it work. First of all, in order to follow this algorithm we have to be able to generate stub sequences from motif counts. We also need to take into account the fact that filling certain stubs will also automatically fill other stubs (so we need to use fewer stubs than the stub sequences indicate).

In order to truly get a random graph (as opposed to just any graph), we need to make sure each graph with the given motif counts is equally likely to be chosen. That means for each stub sequence, we need to count how many graphs can be generated from it. Then we must make a list of all possible stub sequences and choose them with probability proportional to that number.

The first task is a counting problem and we can find a closed-form solution using elementary methods.

The second task is harder. Unfortunately, we need to actually generate all the stub sequences so we can find the solution to the counting problem. We could certainly use recursion or a more sophisticated technique to generate all the stub sequences. But I get the feeling this would take a lot of time and space, and we might not be able to store all the stub sequences in memory. My proposal is to ask people for advice and go from there.

2.3 Evaluation of method

First we have to check if the graph we generated has the given motif counts. We already have a program to generate the motif counts of a graph, so this should not be a problem. If the graph doesn't have those motif counts, we can check how far off we are by using the root-mean-square of the relative error.

Once we know this, we should check that the graph is randomly generated. We can look at small graphs where we can enumerate all possibilities by hand. Then we can run the algorithm repeatedly and make sure these possibilities appear equally often.

References

[1] Milo et al. *On the uniform generation of random graphs with prescribed degree sequences*. (2004) arXiv:cond-mat/0312028

[2] Karrer and Newman. *Random graphs containing arbitrary distributions of subgraphs*. (2010) arXiv:1005.1659 [cond-mat.stat-mech]

[3] Ugander et al. *Subgraph Frequencies: Mapping the Empirical and Extremal Geography of Large Graph Collections*. (2013) WWW.