Null Models For Social Networks

Jessica Su
Department of Computer Science
Stanford University
jtysu@stanford.edu

Sen Wu
Department of Computer Science
Stanford University
senwu@stanford.edu

ABSTRACT

Social motif has widely used for clustering large social network. However, graph generation algorithms often focus on degree distribution and lack of motif distribution. In this paper, we define the motif-driven graph generation problem and present a heuristic method which focus constructing the approximate motif distribution of large network. Our experiments on 67 real social network motif network distribution show that our approach obtains the generated graph with similar motif distribution within relevant error less then 0.02 in 10 hours.

Categories and Subject Descriptors

J.4 [**Social and Behavioral Sciences**]: Miscellaneous

General Terms

Algorithms, Experimentation

Keywords

Social networks

1. INTRODUCTION

Many real-world systems can be modeled by graphs, from power grids to arXiv citations to friendships on Facebook. Early models attempted to model all systems with the same type of graph. For example, the Erdos-Renyi random graph models all networks with n participants and m connections with a graph of n nodes and m randomly placed edges. The Watts-Strogatz model models all networks by connecting nodes to their nearest neighbors, then filling the rest of the graph with randomly placed edges.

More sophisticated models account for differences in degree distribution and clustering coefficient. The preferential-attachment model generates a graph with power-law degree distribution and allows the modeler to choose the exponent in the power law. The configuration model takes as input an exact sequence of degrees and creates a random graph with that degree sequence. Newman's "triangle-edge" model generalizes this to create a random graph with specific degree sequence and clustering coefficient.

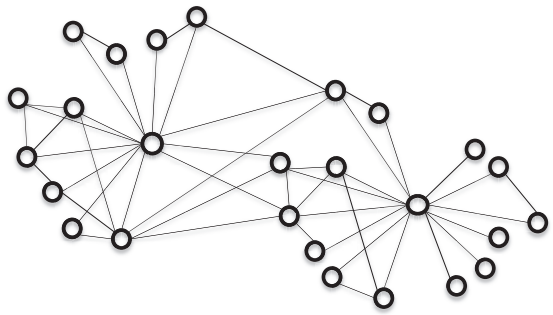


Figure 1: A sample social network.

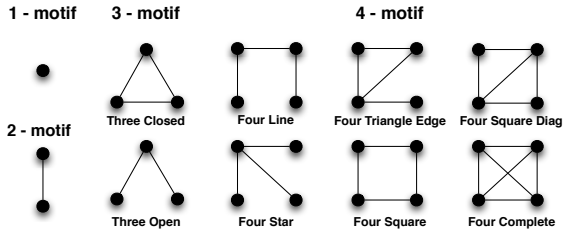


Figure 2: Graphical representation of the motifs.

In our model we create a random graph with specific degree sequence and motif counts. A motif is a small subgraph (with fewer than five vertices), so we would create a graph with specific numbers of cycles, 4-cliques, etc. While this problem does not have exact solutions in all cases, we use hill-climbing to find an approximate solution that produces good results in practice.

2. BACKGROUND

In this section, we first give some basic concepts that we use throughout the paper. Then, we formulate the problem of motif-driven graph generation.

A **graph** is a representation of a set of objects and the connections between them. It is defined as a tuple (V, E), where V is a set of $|V| = N$ vertices ("nodes") and $E \subseteq V \times V$ is a set of edges. The nodes in a graph correspond to the objects, and the edges to the connections.

A graph is called **simple** if no node has an edge to itself and no two nodes have more than one edge between them. A graph is **undirected** if whenever (v, w) is an edge, (w, v) is also an edge.

In our problem, we assume all graphs are simple and undirected.

A **subgraph** of G is a graph whose nodes and edges form subsets of the nodes and edges of the graph G. An **induced subgraph** of G on the vertices $V' \subseteq V$ is the graph consisting of the vertices V' and the edges between them.

A **motif** is a small, connected graph, often considered as a subgraph of a larger graph. In this paper, we only consider motifs within 4 vertices, which we term "1-motif", "2-motif", "3-motif" and "4-motif" respectively. Note that, 1-motif is simply a vertex, and 2-motif is simply an edge. A full list of motifs is shown in Figure 2.

The **motif distribution** of a graph G specifies how many of each motif type the graph contains. For example, in Figure 1 represents a graph with motif distribution {mtThreeClosed: 12, mtThreeOpen: 16, mfFourLine: 22, mfFourSquare: 18, mfFourStar: 30, mfFourTriangleEdge: 10, mfFourSquareDiag: 17, mfFourComplete: 2} would contain twelve triangles, two 4-cliques, and so forth.

3. PROBLEM DEFINITION

Problem: To generate a network with desired specifications.

Input: The input of our problem consists of is the motif distribution D. In this paper, we only consider motif within 4 vertices.

Output: Our goal is to generate a graph G which has motif distribution that closely approximates D.

Ideally we would generate a graph with the exact motif distribution D. In the next section we show that problem is NP-hard, so the best we can hope for is an approximation.

The problem formulation is different from other graph generation problems [2, 5, 1, 4, 3], as in this paper we focus on generating graphs based on the motif distribution. Motif distributions are frequently used for analyzing large graphs in the social sciences.

4. OUR APPROACH

In this section, we propose a heuristic model to generating the motif driven social network.

4.1 Hill climbing

Hill-climbing is a standard technique for finding good solutions to optimization problems. Start with a solution that is not particularly good. At each step, perturb the solution randomly. If the new answer is better, keep it; if not, keep the old solution. Repeat this until the algorithm converges to a good solution.

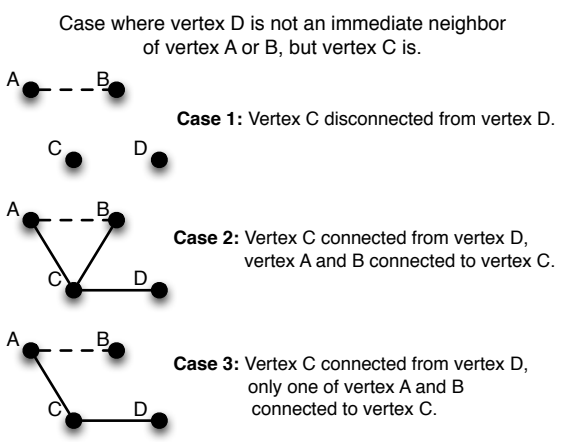
Hill-climbing does not guarantee an optimal solution, since it tends to get stuck at local maxima. Yet in practice the solutions it finds are "pretty good," especially after running the algorithm several times and picking the best solution.

In our case we minimize the error between the desired motif counts and our solution's motif counts:

$$\text{Average relative error} = \frac{1}{\ell}\sum_{i=1}^{\ell}\frac{|\text{counts}_i - \widehat{\text{counts}}_i| + 1}{|\text{counts}_i| + 1}$$

where ℓ is the number of different motif types.

We use the configuration model to generate a random graph with the required degree sequence. Then at each step we choose two edges (at random) and swap their endpoints. (We chose this transformation step since it preserves the degree sequence of the graph.) We count the motifs and compare the error with the new counts to the error with the old counts. If the new counts give a smaller error, we keep the new graph; otherwise, we discard it.



4.2 Optimizations

As written, this solution is very slow. Counting motifs is $O(|V|^4)$ and takes unacceptably long in practice. To get good results in practice, we need a large number of rewiring steps, so ideally each rewiring step should take less than a second.

We can speed this up by counting motifs incrementally. Instead of considering the whole graph on every step, we only look at the part of the graph whose motifs would be affected by the edge changes. Since we are only looking at motifs with fewer than 5 nodes, we can only consider the nodes that are one or two hops away from the nodes whose edges are being rewired. Then we can count how many motifs are being created or destroyed in the induced subgraph on those nodes, and add those to the total motif counts. Once we have the induced subgraph, we count the motifs by taking all possible sets of four vertices, and seeing which motif they form, if any.

(In our algorithm we break the rewiring step into four steps, two edge deletions and two edge creations. This way we only measure the effect of one edge creation/deletion step at a time.)

This is still fairly slow, so we apply one final optimization. We notice that for a 4-motif to be affected by the edge changes, vertices 1 and 2 of the motif must be endpoints of the edge. Vertex 3 must be an immediate neighbor of an endpoint, and vertex 4 is either an immediate neighbor or a second-degree neighbor. Ordinarily, we would have to loop through the immediate neighbors to find all possibilities for vertex 3, and perform an inner loop through the second degree neighbors to find all possibilities for vertex 4. But if vertex 4 was always an immediate neighbor, we could loop through the immediate neighbors both times, which would speed up the algorithm considerably.

Vertex 4 is not always an immediate neighbor. However, in the cases where it's not an immediate neighbor, we can do a lot less computation than we would have to otherwise. We can break these situations up into three cases:

- Vertex 3 is not connected to vertex 4. In this case, the four vertices can never form a 4-motif, regardless of whether vertices 1 and 2 are connected, and we don't have to change the motif counts.
- Vertex 3 and 4 are connected, and vertex 1 and 2 are both connected to vertex 3. In this case, connecting vertex 1 and 2 deletes a four-star motif and adds a triangle-with-edge motif.
- Vertex 3 and 4 are connected, and only one of vertex 1 and 2 are connected to vertex 3. In this case, connecting vertex 1

Dataset	#nodes	#edges
aut-as19971108	3015	5156
aut-as19990628	5322	10163
cit-scimet	3085	13474
col-ca-GrQc	5242	14484
col-netscience	1461	2742
met-HI	1424	3423
ppi-ppiall	3258	12930
ppi-ppiapms	1622	9070
pwr-power	4941	6594

Table 1: Statistics of the nine networks.

and 2 creates a four-line motif.

It is much faster to test for these cases than to do the normal computations (which would involve finding the induced subgraph on those four vertices, then testing it to see if it was either of the 4-motifs). So implementing this optimization produces an enormous speedup, allowing us to perform several thousand rewiring steps in one day.

5. RESULTS

We conduct various experiments to evaluate the our propsed method. The datasets and codes are publicly available.[1]

5.1 Experiment Setup

Data sets. We evaluate the proposed method on nine different kinds of networks: aut-as19971108, aut-as19990628, cit-scimet, col-ca-GrQc, col-netscience, met-HI, ppi-ppiall, ppi-ppiapms and pwr-power. Table 1 lists statistics of the nine networks.

Evaluation metrics. To quantitatively evaluate the proposed model, we consider using the average relative error to measure our performance:

Average relative error: the absolute error divided by the magnitude of the exact value.

$$ARE = \frac{|D - D_{approx}|}{D}$$

All codes are implemented in C++ and Python, and all the evaluations are performed on an x64 machine with E7520 1.87GHz Intel Xeon CPU (with 16 cores) and 192GB RAM. The operation system is Microsoft Windows Server 2008 R2 Enterprise.

5.2 Performance Analysis

Hill climbing produces excellent results on most networks. Table 2 shows the improvements in the error after running hill climbing for 24 hours.

Although performance varies across networks, we get at least a 50% decrease in error for all networks except aut-as19990628 and cit-scimet. For some networks we get an enormous reduction in error; pwr-power gives a 99.5% reduction and met-HI gives a 99.9% reduction. The first group of figures plots the reduction in error over time and the second group plots the probability that a rewiring step will be successful (i.e. that we will accept the changes instead of discarding them).

6. FUTURE WORK

[1]http://snap.stanford.edu/data/index.html

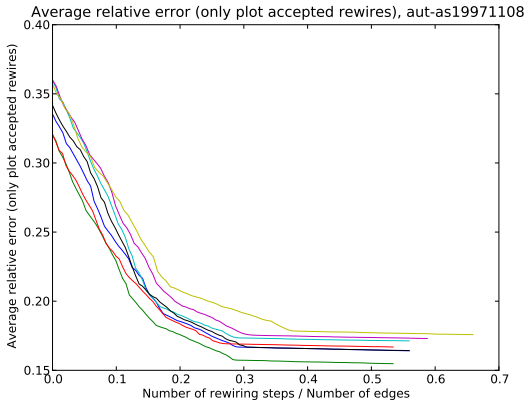


Figure 3: Error, network aut-as19971108. Only plot hill climbing steps that were successful.

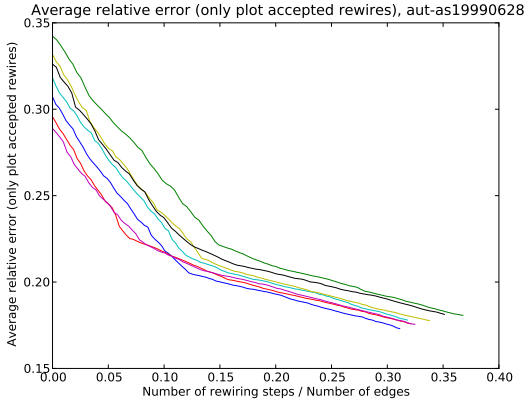


Figure 4: Error, network aut-as19990628. Only plot hill climbing steps that were successful.

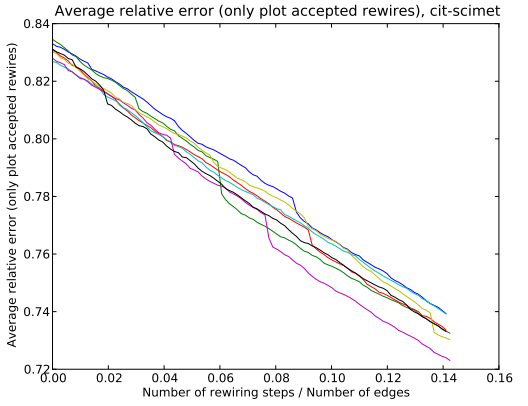


Figure 5: Error, network cit-scimet. Only plot hill climbing steps that were successful.

Network	Nodes	Edges	Initial error	Final error	Successful rewires
aut-as19971108	3015	5156	0.34216	0.16717	2951.00
aut-as19990628	5322	10163	0.31571	0.17723	3397.28
cit-scimet	3085	13474	0.83063	0.73247	1921.71
col-ca-GrQc	5242	14484	2.05549	0.93973	40583.8
col-netscience	1461	2742	3.13864	0.55110	8290.71
met-HI	1424	3423	8039.58	0.46768	5703.57
ppi-ppiall	3258	12930	1.06249	0.46058	38131.1
ppi-ppiapms	1622	9070	1.37580	0.54219	24581.7
pwr-power	4941	6594	0.57996	0.00282	9485.4

Table 2: Improvements in error after running hill climbing for 24 hours. All numbers are averaged over 7 trials.

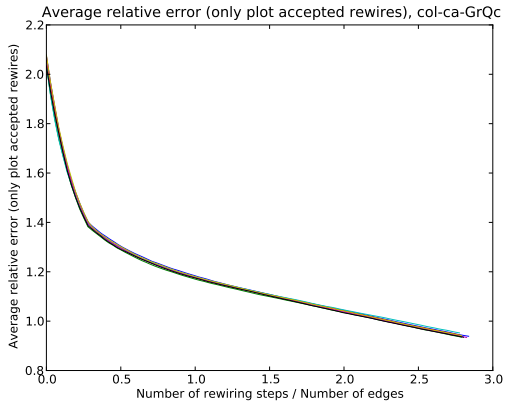


Figure 6: Error, network col-ca-GrQc. Only plot hill climbing steps that were successful.

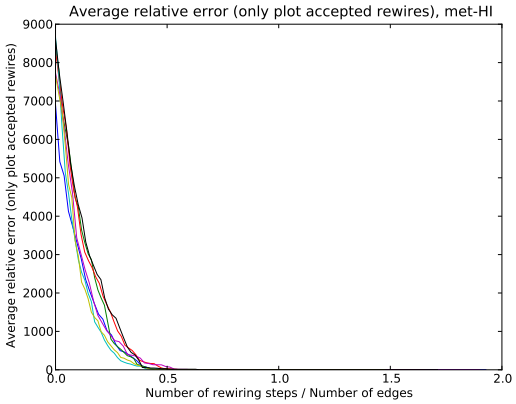


Figure 8: Error, network met-HI. Only plot hill climbing steps that were successful.

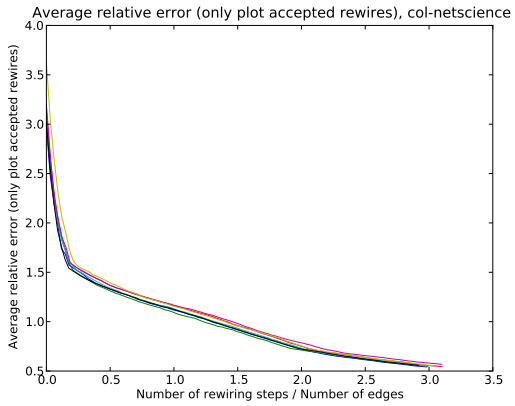


Figure 7: Error, network col-netscience. Only plot hill climbing steps that were successful.

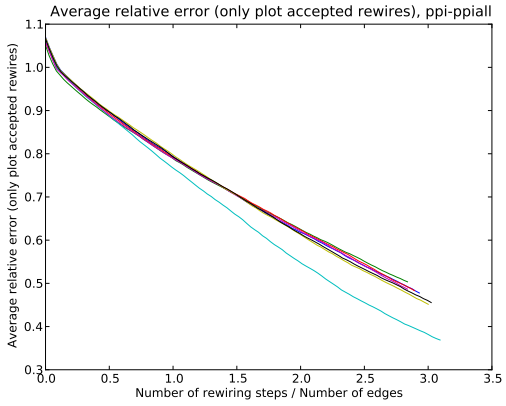


Figure 9: Error, network ppi-ppiall. Only plot hill climbing steps that were successful.

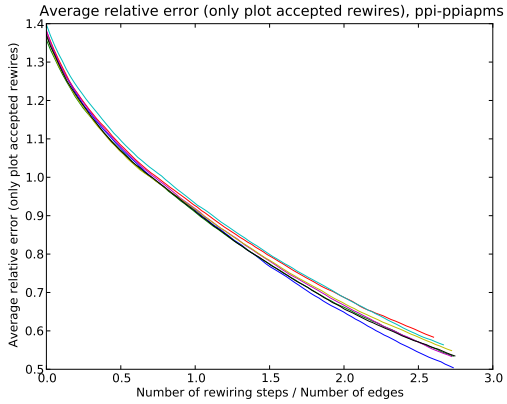


Figure 10: Error, network ppi-ppiapms. Only plot hill climbing steps that were successful.

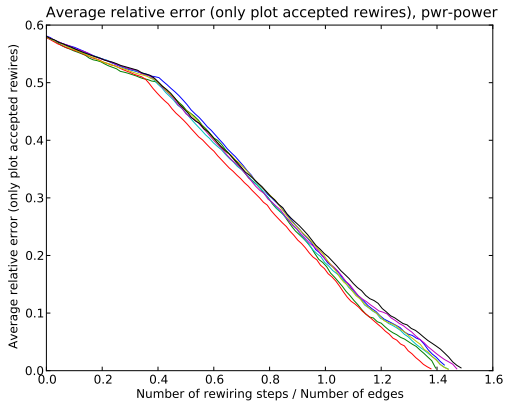


Figure 11: Error, network pwr-power. Only plot hill climbing steps that were successful.

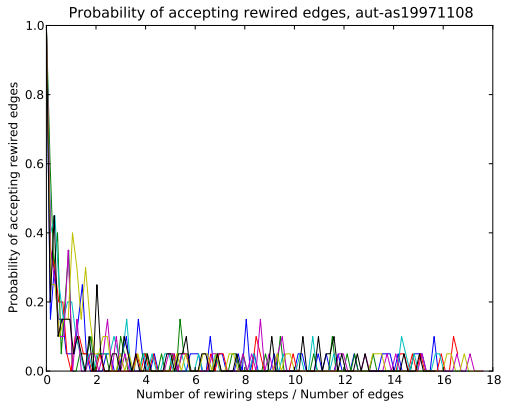


Figure 12: Probability of a rewiring step being successful, network aut-as19971108

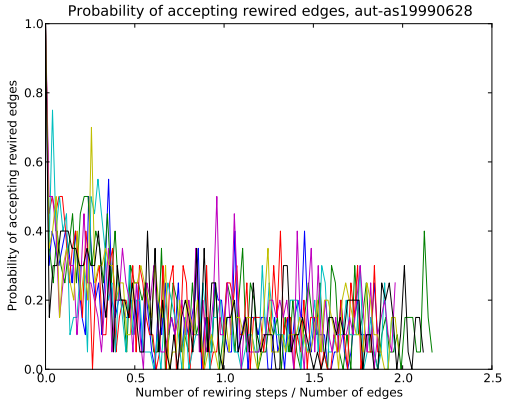


Figure 13: Probability of a rewiring step being successful, network aut-as19990628

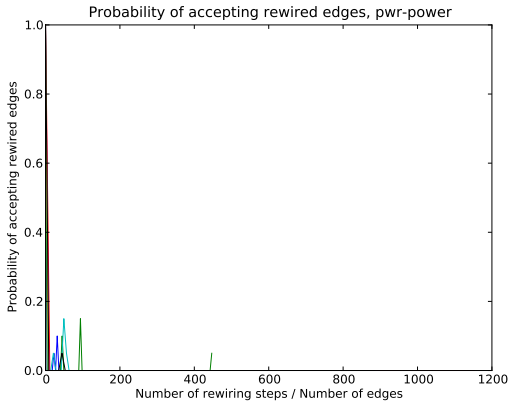


Figure 20: Probability of a rewiring step being successful, network pwr-power

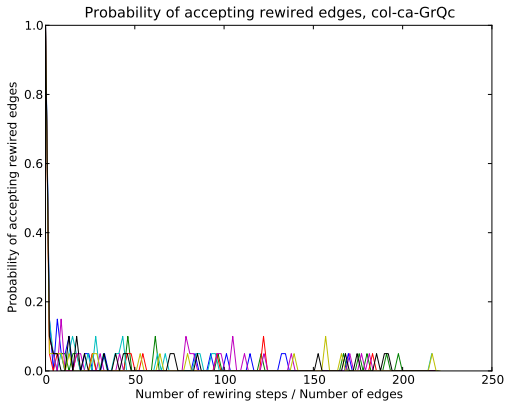


Figure 15: Probability of a rewiring step being successful, network col-ca-GrQc

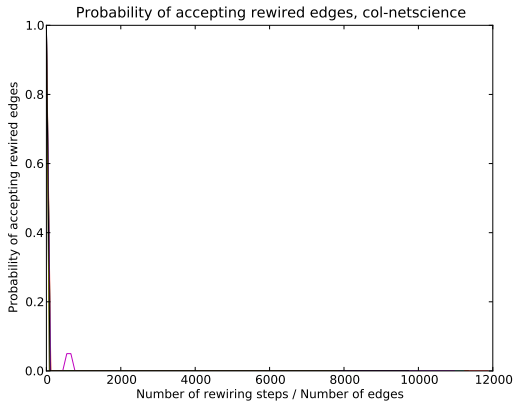


Figure 16: Probability of a rewiring step being successful, network col-netscience

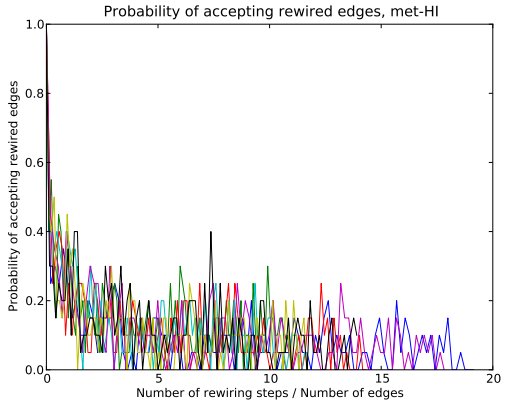


Figure 17: Probability of a rewiring step being successful, network met-HI

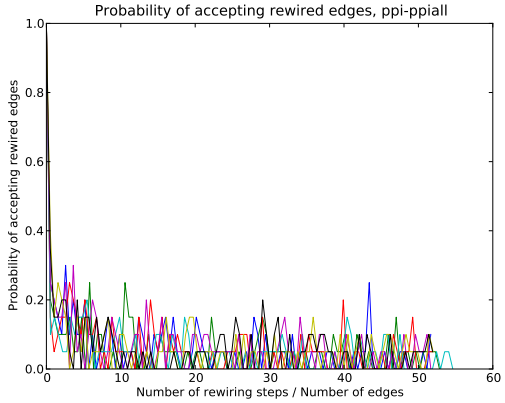


Figure 18: Probability of a rewiring step being successful, network ppi-ppiall

Our current algorithms assume that we know the degree distribution of the graph. This is fine if we have the graph of a real-world social network and we are trying to build a model to compare it to. However, it fails if we are given the motif counts only.

Fortunately, social networks tend to have power-law degree distributions, which means their distributions can be described by a single parameter α, where α is the magnitude of the exponent. (We also need a normalization constant, which can be found from the number of edges.) For the final paper we will build a model to predict α from the motif counts. We think we can do this because graphs with a lot of high-edge motifs should be denser, so the degree distribution should have a heavier tail.

Models to try include linear regression, neural networks, and stochastic gradient descent.

7. REFERENCES

[1] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[2] P. ERDdS and A. R&WI. On random graphs i. *Publ. Math. Debrecen*, 6:290–297, 1959.

[3] M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random structures & algorithms*, 6(2-3):161–180, 1995.

[4] M. E. Newman. Random graphs with clustering. *Physical review letters*, 103(5):058701, 2009.

[5] D. J. Watts and S. H. Strogatz. Collective dynamics of Śsmall-worldŠnetworks. *nature*, 393(6684):440–442, 1998.