Null Models For Social Networks

Jessica Su
Group #3
Stanford University
jtysu@stanford.edu

Sen Wu
Group #3
Stanford University
senwu@stanford.edu

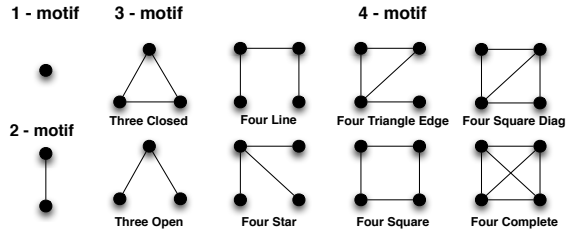


Figure 1: Graphical representation of the motifs.

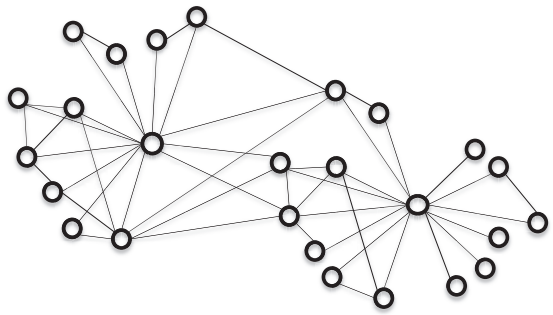


Figure 2: A network with motif distribution {numNodes: 25, numEdges: 44, mtThreeClosed: 21, mtThreeOpen: 151, mfFourLine: 134, mfFourSquare: 1, mfFourStar: 305, mfFourTriangleEdge: 155, mfFourSquareDiag: 26, mfFourComplete: 2}.

ABSTRACT

Networks with similar motif distributions tend to have similar functions in the real world [7]. In this paper, we present a heuristic method to construct graphs with specific motif distributions. By constructing graphs with specific motif distributions, we hope to build models that will imitate the functionality of specific networks.

Our experiments on 9 networks show substantial improvement over random graphs. On the pwr-power network, the average relative error between the motif counts of our model and the network is 0.003.

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 [2][3] models all networks with n participants and m connections with a graph of n nodes and m randomly placed edges. The Watts-Strogatz model [8] 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 [1] 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 [5] generalizes this to create a random graph with specific degree sequence and clustering coefficient.

Our objective is to create a random graph with specific degree sequence and motif counts. (See Figure 2 for an example of motif counts.) A motif is a small, connected subgraph of a larger graph, and we only consider motifs with four or fewer vertices in our analysis (Figure 1). Motif counts are useful for distinguishing different types of real-world graphs. A network like Twitter (when converted to an undirected graph) might have many Four Star motifs, since a small fraction of users have large numbers of followers who do not follow each other. By contrast, a network like Facebook would have a disproportionate number of Four Complete motifs, since a user's Facebook friends often have many mutual friends with the user himself. Finding the motif balance allows us to distinguish "celebrity-based" networks from "friendship-based" networks. (There are also many other types of networks; email for-

warding chains would have a large number of line graphs, as would product recommendation graphs.)

These structures are not fully captured by models that look only at 3-motifs or clustering coefficients. Knowing only the number of triangles (Three Closed) and triads (Three Open), we could not tell the difference between two strangers with one mutual friend and two strangers with two mutual friends. The first might correspond to two people in different social circles who are both connected to a social "hub," while the second might correspond to two people in the same class who know several of the same classmates but have not yet met. Using 4-motifs gives us a better sense of the structure of the graph. While we might obtain better results by using 5- and 6-motifs, this is more computationally intensive and we relegate it to future work.

This intuition was formalized by Chuanqi Shen, who built a classifier to cluster networks according to their motif counts. He found that the clusters corresponded closely to real-life functionality [7]. Therefore, by generating graphs with similar motif counts, we hope to build graphs with similar function to real-world networks.

2. PROBLEM DEFINITION

Our goal is to generate a random graph with a given motif distribution D, where each graph with correct motif counts is chosen with equal probability.

This is a difficult problem, so we solve the easier problem of generating an arbitrary graph with motif distribution D. Given a solution to this problem, we may be able to solve the original problem by finding transformations that preserve the motif distribution D, then showing that every graph with motif distribution D can be obtained through a sequence of such transformations. However, this is very challenging and we relegate it to future work.

Finding an arbitrary graph with motif distribution D is not always possible and may be NP-hard. Therefore, we try to find a graph with a motif distribution that closely approximates D, where "closeness" is defined by the average relative error between the graph's motif counts and the desired motif counts (Equation 1).

Input: A motif distribution D, where each motif has at most 4 vertices.

Output: A graph G with a motif distribution that closely approximates D.

We first focus on an easier problem, where we are given both a motif distribution and the real-life network it corresponds to. This problem is easier since it allows us to use the degree distribution of the original network. It also guarantees that a solution exists, which is not true for all motif distributions. (For example, there are no graphs with two nodes and three triangles.) In solving this problem we should not just return the original graph, since that method cannot be easily extended to solve the previous problems.

If our purpose is to compare null models to real data sets, it suffices to solve the easier problem, since we have the data required to create the model. If we want to solve the harder problem, we can assume that the degree distribution follows a power law and use the motif counts to predict the exponent. Then we can use the solution to the easier problem to solve the harder problem. We predict the exponent in Section 6.

3. RELATED WORK

Our approach is inspired by Milo's approach for generating a random graph with prescribed degree sequence. [6] He begins by generating an arbitrary graph with that degree sequence. (This can be done by giving each node a certain number of "half-edges," then connecting them randomly to form the edges of the graph.) One this graph is generated, he chooses pairs of edges at random and swaps the endpoints, repeating this step until the graph is sufficiently "randomized."

Generate graph $G = (V, E)$ with required degree sequence
while *Markov chain displays insufficient mixing* **do**
  Choose $e_1, e_2 \in E$ at random
  Add edges $(e_1.Src, e_2.Dst)$, $(e_2.Src, e_1.Dst)$
  Delete edges $(e_1.Src, e_1.Dst)$, $(e_2.Src, e_2.Dst)$
end

Algorithm 1: Milo's approach for generating random graphs with prescribed degree sequences.

This "rewiring" step is innovative because it preserves both the number of edges and the degree of each node. For this reason we use the same rewiring step in our algorithm.

4. OUR APPROACH

In this section, we present a approach framework for generating the graph. At the high level, our proposed framework consists of two stage.

- **Degree distribution generation.** First, given a certain motif distribution, we train a gradient boosting model using the motif distribution. And then we predict the degree distribution from motif distribution.
- **Candidate graph generation.** Second, with the degree distribution, we generate a random graph based on degree distribution as candidate graph for our problem.
- **Graph refinement.** Second, the random graph is fed to a heuristic model to refine the graph. The model incorporates the several strategies to refine the graph satisfied the given motif distribution.

4.1 Degree distribution generation

For a given motif distribution, we use the motif distribution to predict degree distribution. The idea of this problem is using the motif distribution as a feature to train a model, and then predict a degree distribution for the given motif distribution. More precisely, we define a gradient boosting model to train a model. Gradient Boosting Decision Tree [4] is a useful machine learning method for regression problems, which also an ensemble method with combine multiple weak prediction models. It makes the model in a stage-wise fashion and generalizes the model by optimization of the arbitrary differentiable loss function which is same as logistic regression.

==========

In this section, we propose a heuristic method for generating the required graph.

4.2 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:

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

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 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.

```
Initialize graph G = (V, E) with prescribed degree sequence
motifCounts <- CountMotifs(G)
repeat
    Choose e1, e2 ∈ E at random
    Add edges (e1.Src, e2.Dst), (e2.Src, e1.Dst)
    Delete edges (e1.Src, e1.Dst), (e2.Src, e2.Dst)
    newMotifCounts <- CountMotifs(G)
    if error(newMotifCounts) < error(motifCounts) then
        motifCounts <- newMotifCounts
    else
        Return graph to original state
    end
until computation time limit exceeded;
```

Algorithm 2: Naive approach

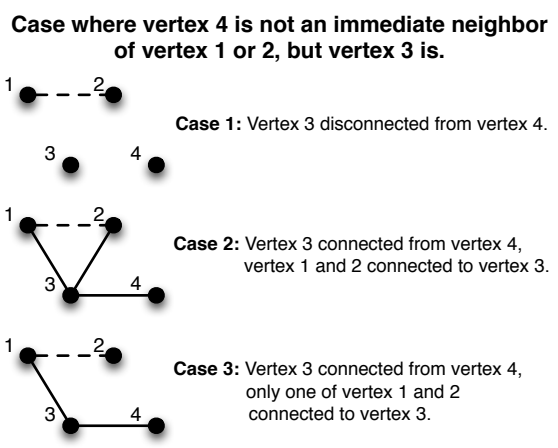
4.3 Optimizations

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 of vertex 1 or vertex 2. However, when it's not an immediate neighbor of either endpoint, 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 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. Each rewiring step is $O((d_1 + d_2)^2)$, where d_1 is the degree of vertex v_1, d_2 is the degree of vertex v_2, and $d_1 + d_2$ is an upper bound on the number of first-degree neighbors of v_1 and v_2.

4.4 Random restarts

Hill climbing is an imperfect solution since it is easy to get stuck at local minima. We fix this with the method of random restarts. When we reach a local minimum, we save the graph and begin rewiring edges randomly. After this we run hill climbing until we reach another local minimum, then repeat the process. At the end of the computation we return the graph with lowest error.

We assume the algorithm has reached a local minimum if we have gone through $|E|$ consecutive rewiring steps without changing the graph. At this point, we rewire $|E|/8$ edge pairs randomly and proceed with hill climbing. Preliminary results for this approach are presented in Section 5.

5. EXPERIMENTAL RESULTS

We conduct various experiments to evaluate our proposed method.

5.1 Experiment 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.

Data sets. We evaluate the proposed method on nine different 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. All data was obtained from Jure Leskovec at Stanford University.

Evaluation metrics. To quantitatively evaluate the proposed model, we use the average relative error between the model's motif counts and the motif counts of the real network. The average relative error is defined in equation 1.

All code is implemented in C++ and Python, and all the evaluations are performed on an x64 machine with Xeon E7-8837 CPU (with 64 cores) and 1024GB RAM. The operating system is CentOS 6.

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 error over time and the second group plots the probability that a rewiring step will be successful (i.e. that the changes are accepted instead of discarded).

The error graphs for aut-as19971108 (Figure 3) and col-netscience (Figure 7) converge to an asymptote. This indicates that running the procedure longer would not improve the error and the final error is a good reflection of the algorithm's performance. Since the errors approach nonzero values (and we know a perfect solution exists), this implies that our hill climbing algorithm has found a local minimum.

The graph for pwr-power (Figure 11) rapidly decreases to zero. This means we have found a perfect or near-perfect solution. The graph for met-HI (Figure 8) also appears to decrease to zero, but that's just because the error started out very high. (The initial error was 8039.58 and the final error was 0.46768.)

The other error graphs have not yet converged and are steadily decreasing 24 hours after the computation. This indicates that we can reduce the final error by increasing the computation time and our current results are not a hard limit on the algorithm's performance.

5.2 Experiment 2

Experimental setup. In our second experiment we made three changes.

1. We ran the simulation for 3.66 days instead of 24 hours.
2. We used a slower library function to generate the initial graph. The original function generated a random graph with approximately the same degree distribution as the real-world graph, but the new function uses the exact degree distribution.
3. We plotted the relative error for each motif, which we defined as
$$\frac{|count - \widehat{count}|}{count}.$$
For brevity, only a few of these plots are shown here.

Performance analysis. Although performance varies across networks (Table 3), we get at least a 58% decrease in error for each of the networks, and at least an 84% decrease for all networks except ppi-ppiapms. The first group of figures plots the error over time, where only successful rewiring steps are plotted.

The error for ppi-ppiapms (Figure 15) does not approach a horizontal asymptote, indicating that increasing the computation time would improve the error.

The errors for col-ca-GrQc (Figure 12) and col-netscience (Figure 13) do approach a horizontal asymptote, which means the algorithm has reached a local minimum. This means we should use other techniques to improve the error, such as simulated annealing, MCMC, or the method of random restarts.

The error for pwr-power (Figure 16) converges to 0.006. Although this error was nonzero and probably a local minimum, we can still say the algorithm was successful because the error was very small.

On the met-HI network (Figure 14), the motif counts of the generated graph were identical to the desired motif counts. The error is 0.4 because we added 1 to the numerator and denominator of the error function to avoid dividing by zero (Equation 1). In Experiment 3, we changed the error function to avoid this issue (Equation 2).

Error for each motif. We plot the per-motif errors for pwr-power in Figures 17, 18, 19, 20, 21, 22, 23, and 24. In this network most motif errors steadily decrease to zero. However, the Four Line motif error decreases to zero and then rises again (Figure 18). The rise appears concurrently with changes in the other plots. At this point, the Four Square Diag error becomes zero (Figure 20) and the Four Star curve becomes a lot flatter. We suspect that the rise marks a single rewiring step that greatly affected the topology of the graph. This might have happened if an edge involved was connected with many motifs.

5.3 Preliminary results: random restarts

In our third experiment we implemented the method of random restarts. Due to time constraints we only ran the method on the small graph pro-polbooks. Since our old error function did not produce zero error on correct motif counts, we changed the error function to

$$\text{Average relative error} = \frac{1}{\ell}\sum_{i=1}^{\ell} \text{error}_i$$
$$\text{error}_i = |\text{counts}_i - \widehat{\text{counts}_i}| \text{ if counts}_i = 0 \quad (2)$$
$$\text{error}_i = \frac{|\text{counts}_i - \widehat{\text{counts}_i}|}{\text{counts}_i} \text{ if counts}_i \neq 0$$

where ℓ is the number of different motif types.

In Table 4 we see this result produces 0% improvement over the baseline. The baseline was calculated by returning the error at the first local minimum. In this case the first local minimum was the best one. In the future we will try this on more networks and hope for better results.

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.

Network	Nodes	Edges	Initial error	Final error	Successful rewires
col-ca-GrQc	5242	14484	5.13635	0.83224	63925
col-netscience	1461	2742	7.63883	0.59894	8048
met-HI	1424	3423	19129.09	0.40010	5109
ppi-ppiapms	1622	9070	1.57338	0.67426	30160
pwr-power	4941	6594	0.58720	0.00596	14324

Table 3: Improved hill climbing and ran it for 3.66 days.

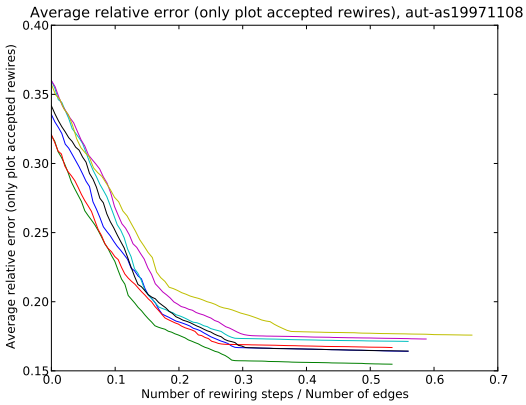


Figure 3: Error, network aut-as19971108. Only plot hill climbing steps that were successful.

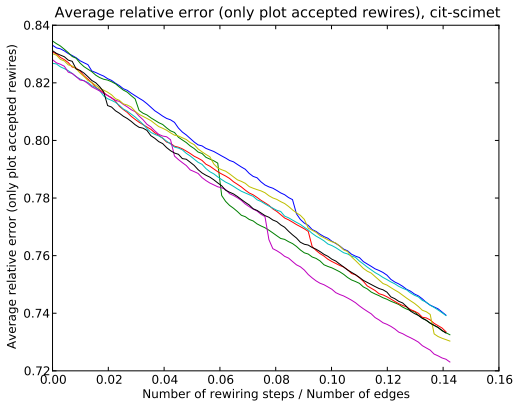


Figure 5: Error, network cit-scimet. Only plot hill climbing steps that were successful.

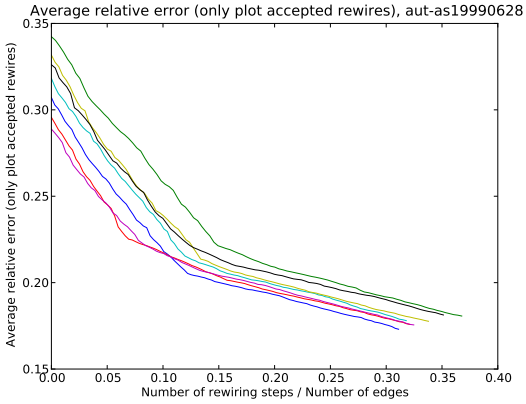


Figure 4: Error, network aut-as19990628. Only plot hill climbing steps that were successful.

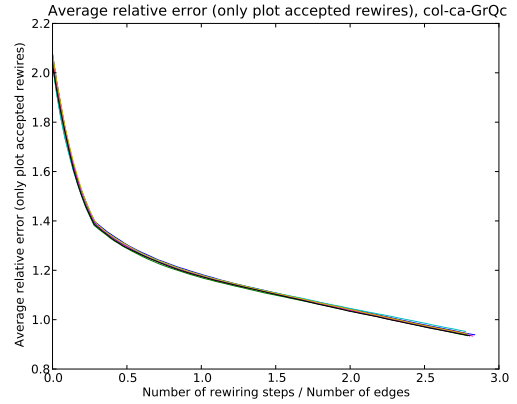


Figure 6: Error, network col-ca-GrQc. Only plot hill climbing steps that were successful.

Dataset	#Nodes	#Edges	Successful rewires	Initial error	Best error	Baseline error	Percent improvement
pro-polbooks	105	441	190656	0.50658	0.01129	0.01129	0.00000

Table 4: Method of random restarts.

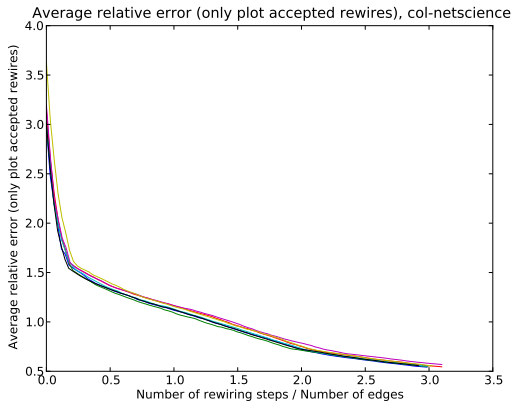


Figure 7: Error, network col-netscience. Only plot hill climbing steps that were successful.

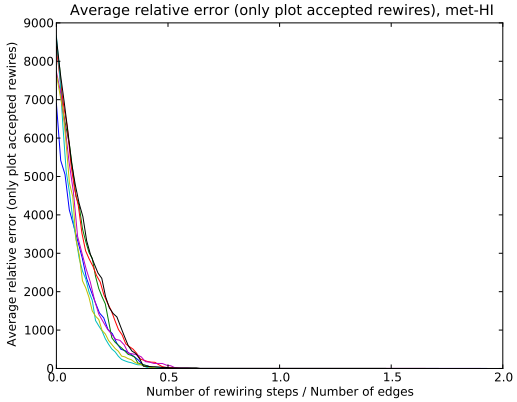


Figure 8: Error, network met-HI. Only plot hill climbing steps that were successful.

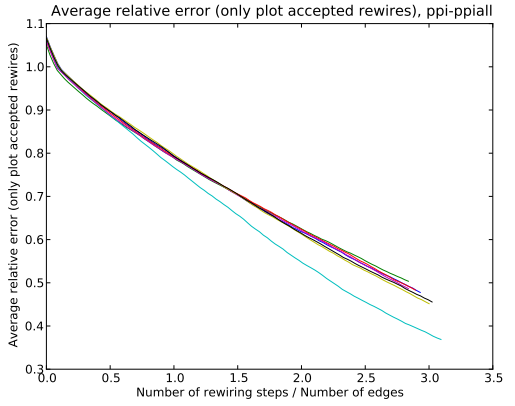


Figure 9: Error, network ppi-ppiall. Only plot hill climbing steps that were successful.

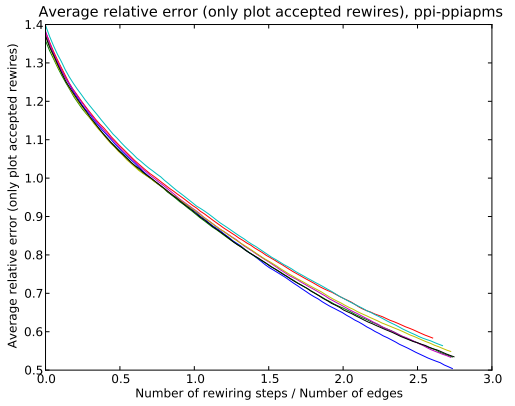


Figure 10: Error, network ppi-ppiapms. Only plot hill climbing steps that were successful.

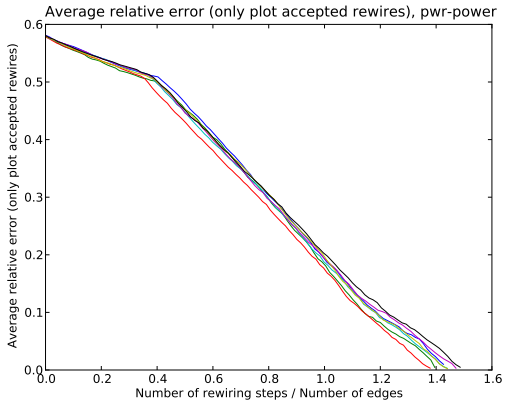


Figure 11: Error, network pwr-power. Only plot hill climbing steps that were successful.

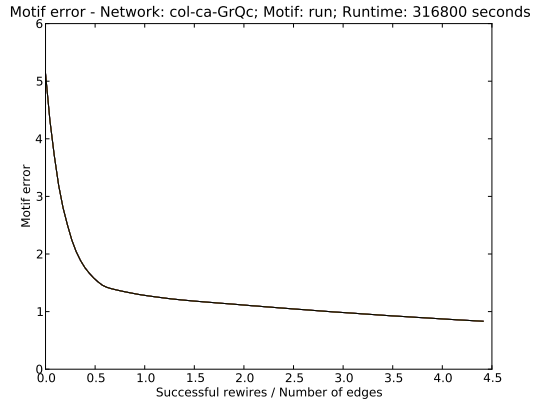


Figure 12: Experiment 2, network col-ca-GrQc.

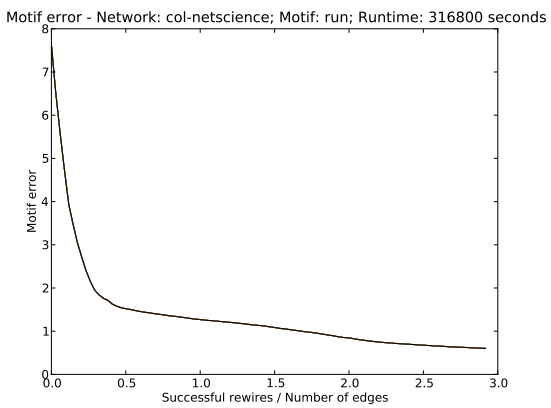


Figure 13: Experiment 2, network col-netscience.

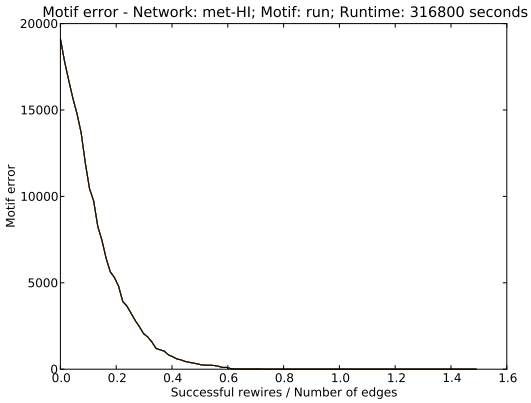


Figure 14: Experiment 2, network met-HI.

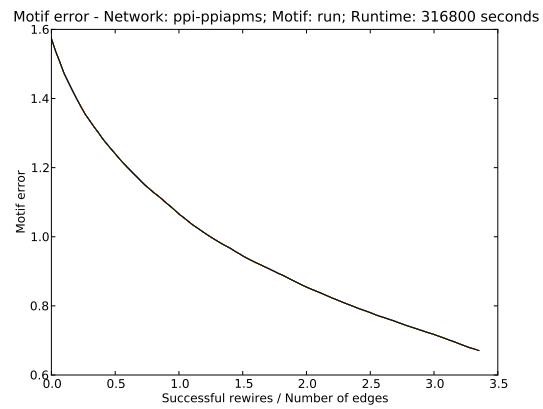


Figure 15: Experiment 2, network ppi-ppiapms.

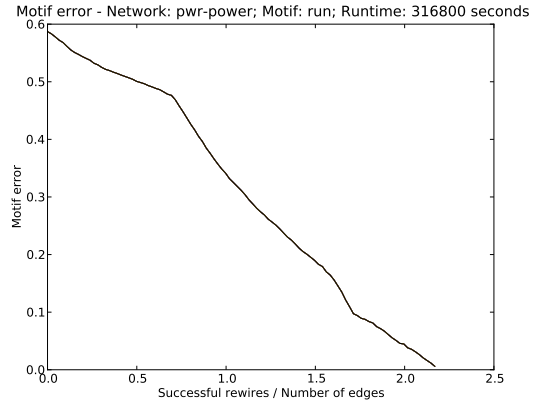


Figure 16: Experiment 2, network pwr-power.

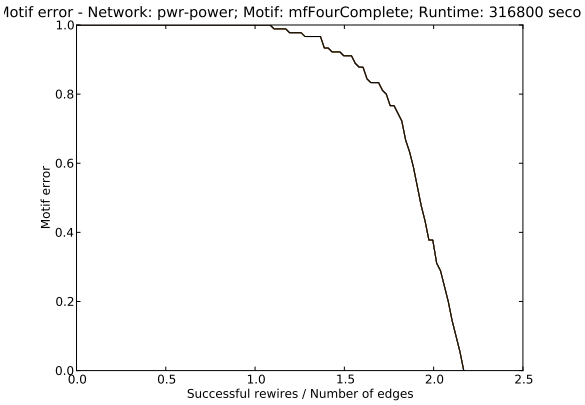


Figure 17: Experiment 2, network pwr-power, motif mfFourComplete.

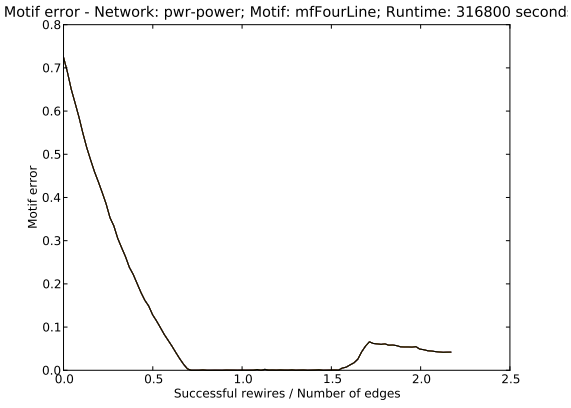


Figure 18: Experiment 2, network pwr-power, motif mfFourLine.

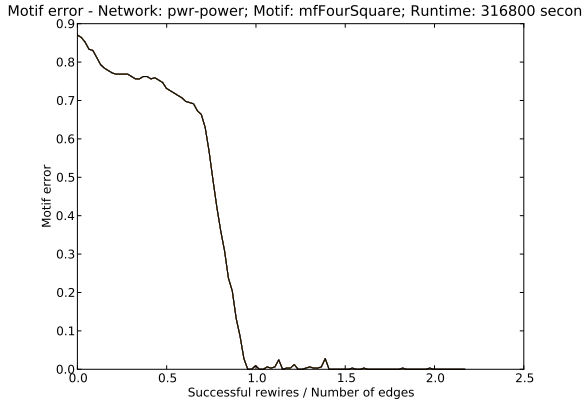


Figure 19: Experiment 2, network pwr-power, motif mfFourSquare.

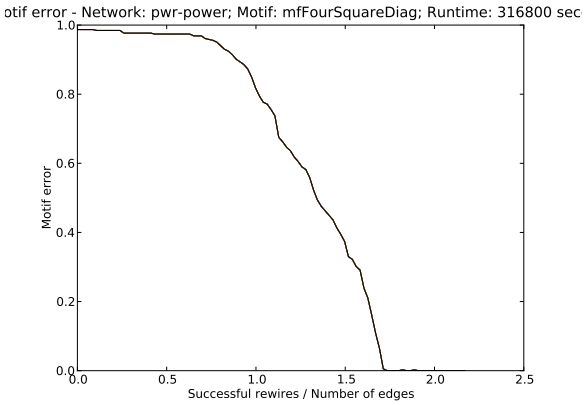


Figure 20: Experiment 2, network pwr-power, motif mfFourSquareDiag.

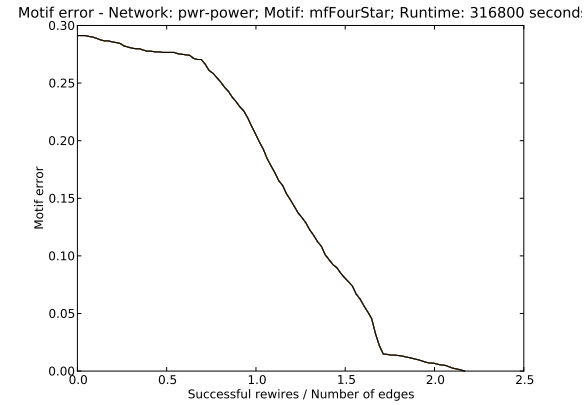


Figure 21: Experiment 2, network pwr-power, motif mfFourStar.

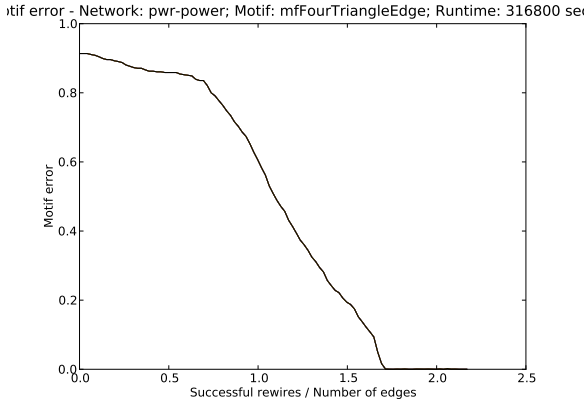


Figure 22: Experiment 2, network pwr-power, motif mfFourTriangleEdge.

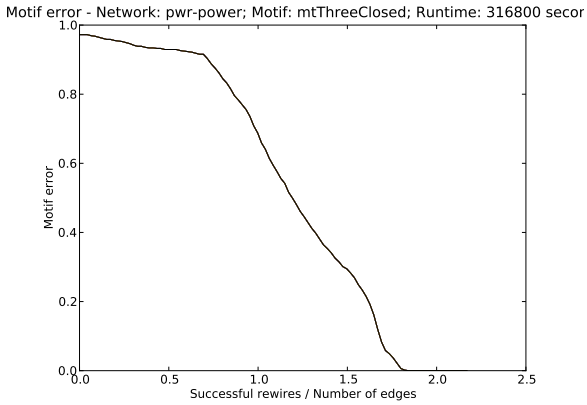


Figure 23: Experiment 2, network pwr-power, motif mtThreeClosed.

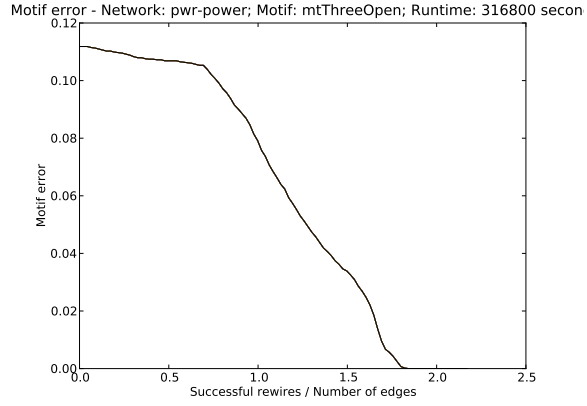


Figure 24: Experiment 2, network pwr-power, motif mtThreeOpen.

6. PREDICTING DEGREE DISTRIBUTION FROM MOTIF COUNTS

7. FUTURE WORK

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.

8. REFERENCES

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

[2] P. Erdos and A. Renyi. On random graphs i. *Publ. Math. Debrecen*, 6:290–297, 1959.

[3] P. Erdos and A. Renyi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutato Int. Koezl*, 1960.

[4] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

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

[6] S. I. M. N. R. Milo, N. Kashtan and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. *arXiv:cond-mat/0312028v2*, 2004.

[7] C. Shen. Curis project on clustering social networks. 2013.

[8] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, 1998.