Null Models For Social Networks

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

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

ABSTRACT

Networks with similar motif distributions tend to have similar functions in the real world [6]. 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 [7] 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 [4] generalizes this to create a random graph with specific degree sequence and clustering coefficient.

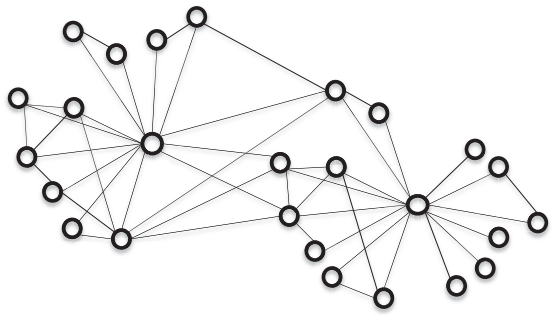


Figure 1: A sample social network.

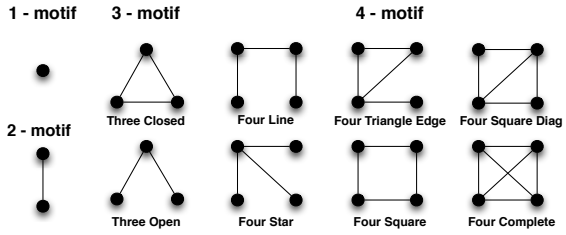


Figure 2: Graphical representation of the motifs.

In our model we create a random graph with specific degree sequence and motif counts. Algorithms that classify social networks according to their motif counts find clusters that correspond closely to real-life functionality [6]. Therefore, by generating graphs with similar motif counts, we hope to build graphs that function similarly to real-life social networks.

While it is difficult to find exact solutions, we use hill-climbing to find approximate solutions that produce 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 **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 {numNodes: 25, numEdges: 44, mtThreeClosed: 21, mtThreeOpen: 151, mfFourLine: 134, mfFourSquare: 1, mfFourStar: 305, mfFourTriangleEdge: 155, mfFourSquareDiag: 26, mfFourComplete: 2} would contain 25 nodes, 44 edges, 21 triangles, two 4-cliques, and so forth.

3. PROBLEM DEFINITION

Our problem is to generate a network with a specific motif distribution.

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.

In this report we 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.)

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. However, in the final report, we will extend our solution to solve the harder problem.

4. RELATED WORK

Our approach is inspired by Milo's approach for generating a random graph with prescribed degree sequence. [5] 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.

5. OUR APPROACH

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

5.1 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:

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

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 $e_1, e_2 \in E$ at random
  Add edges $(e_1.Src, e_2.Dst)$, $(e_2.Src, e_1.Dst)$
  Delete edges $(e_1.Src, e_1.Dst)$, $(e_2.Src, e_2.Dst)$
  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

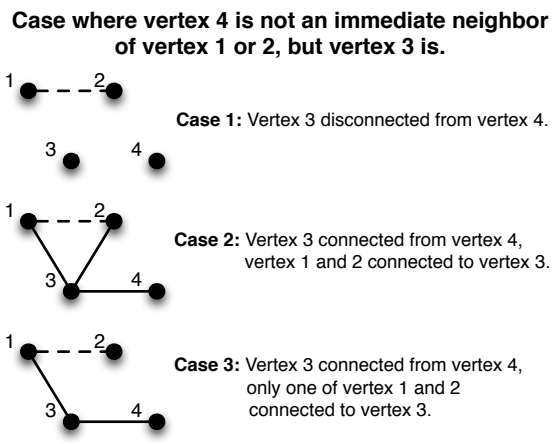
5.2 Optimizations

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 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.

6. EXPERIMENTAL RESULTS

We conduct various experiments to evaluate our proposed method. The datasets are publicly available at the SNAP website.[1]

6.1 Experimental Setup

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.

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.

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

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.

6.2 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 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).

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

[5] 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.

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

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

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.

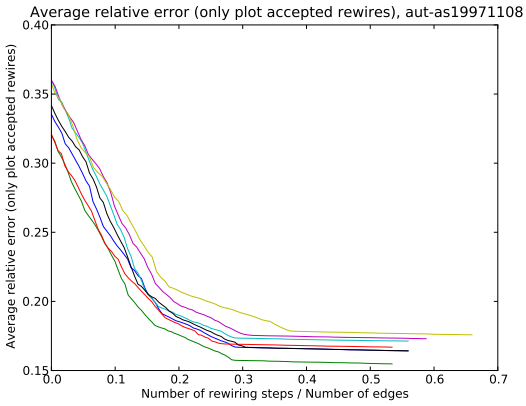


Figure 3: Error, network aut-as19971108. Only plot hill climbing steps that were successful.

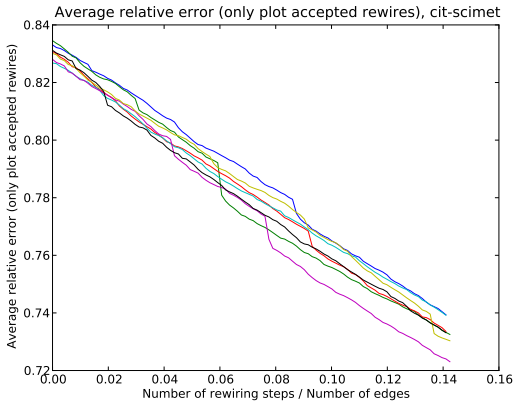


Figure 5: Error, network cit-scimet. Only plot hill climbing steps that were successful.

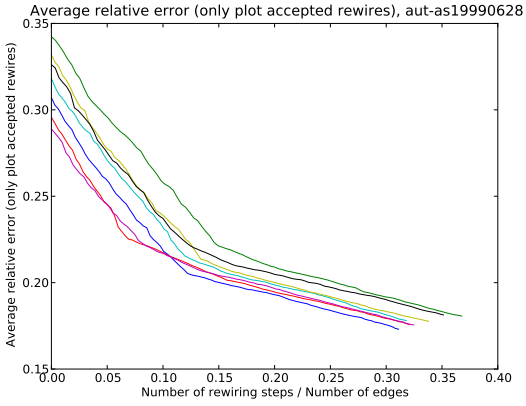


Figure 4: Error, network aut-as19990628. Only plot hill climbing steps that were successful.

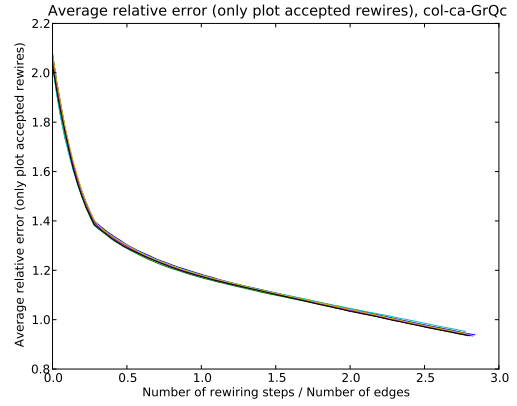


Figure 6: Error, network col-ca-GrQc. Only plot hill climbing steps that were successful.

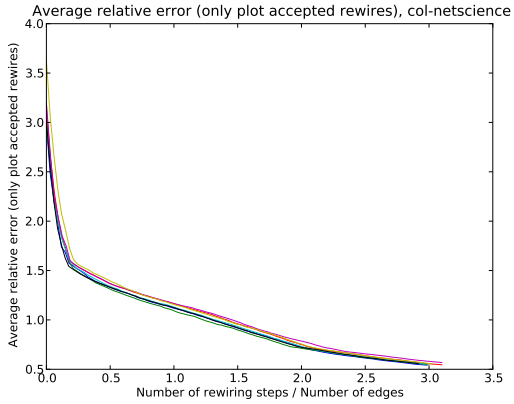


Figure 7: Error, network col-netscience. Only plot hill climbing steps that were successful.

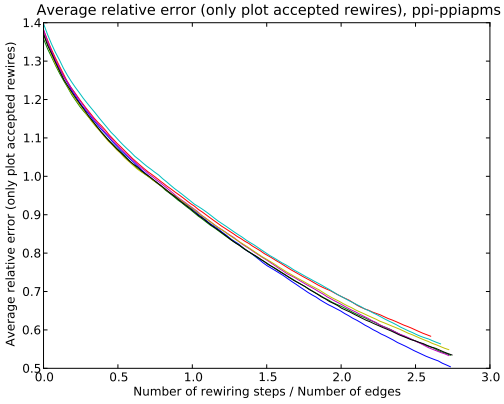


Figure 10: Error, network ppi-ppiapms. Only plot hill climbing steps that were successful.

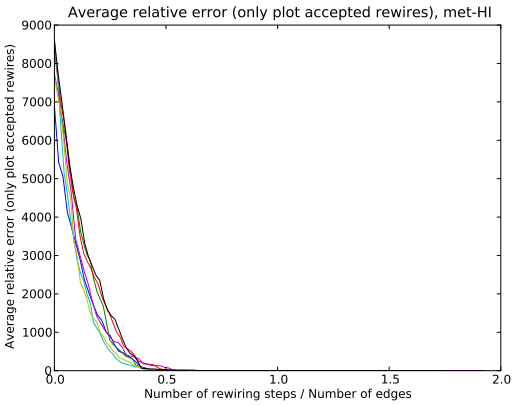


Figure 8: Error, network met-HI. Only plot hill climbing steps that were successful.

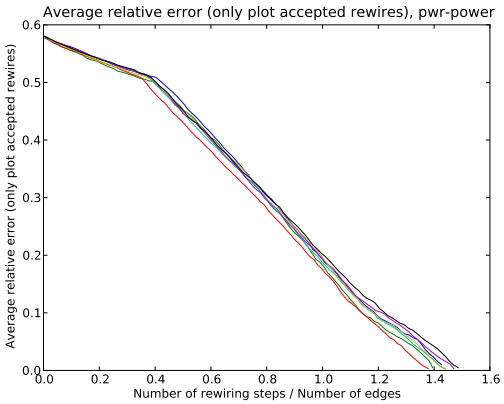


Figure 11: Error, network pwr-power. Only plot hill climbing steps that were successful.

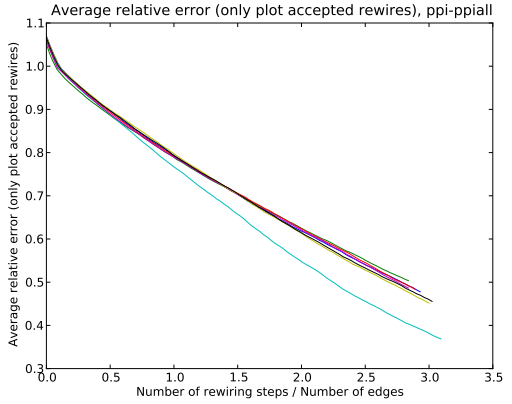


Figure 9: Error, network ppi-ppiall. Only plot hill climbing steps that were successful.

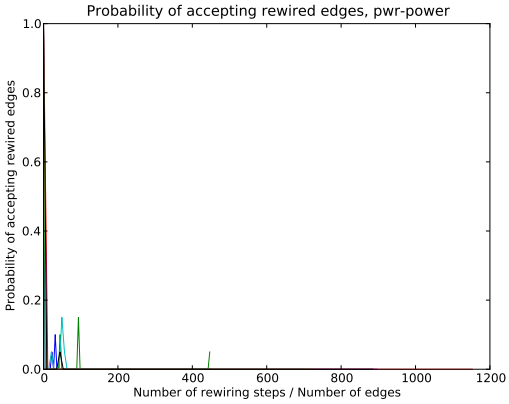


Figure 20: Probability of a rewiring step being successful, network pwr-power

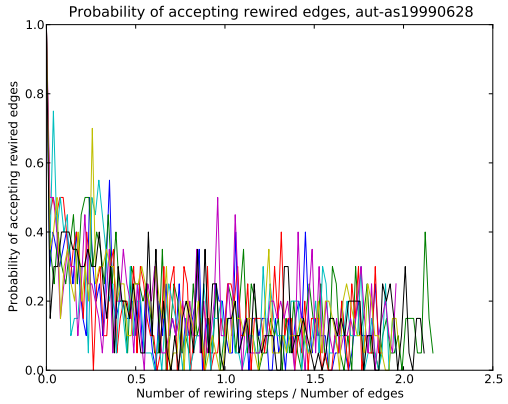


Figure 13: Probability of a rewiring step being successful, network aut-as19990628

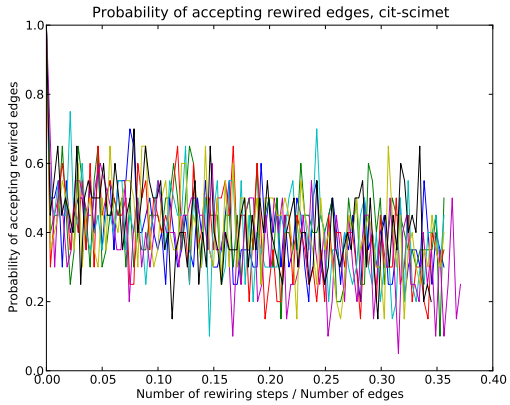


Figure 14: Probability of a rewiring step being successful, network cit-scimet

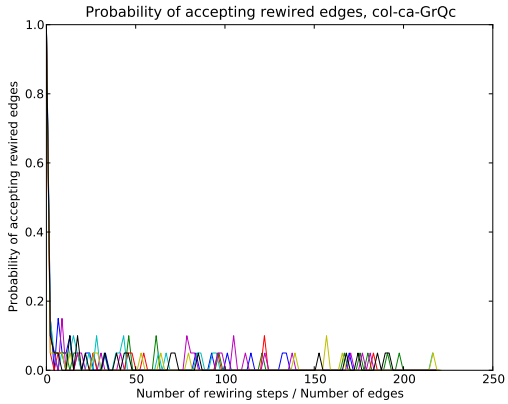


Figure 15: Probability of a rewiring step being successful, network col-ca-GrQc

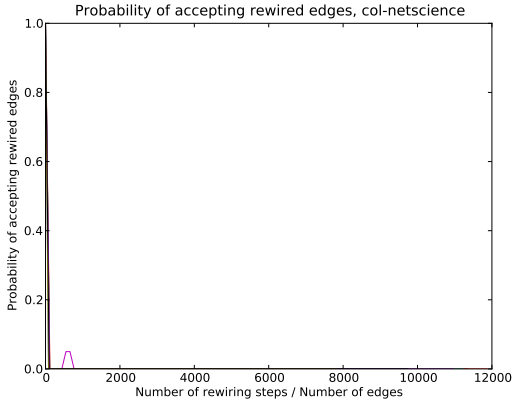


Figure 16: Probability of a rewiring step being successful, network col-netscience

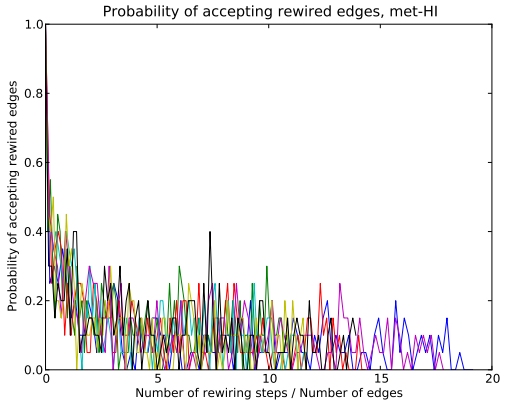


Figure 17: Probability of a rewiring step being successful, network met-HI

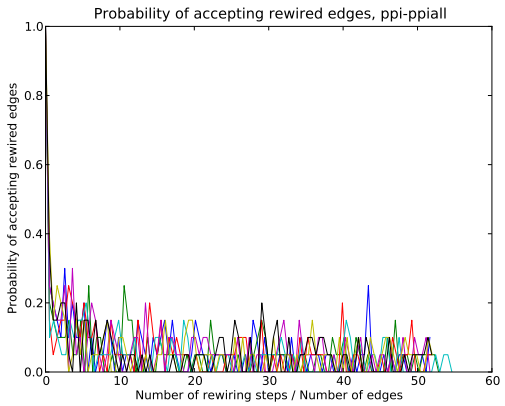


Figure 18: Probability of a rewiring step being successful, network ppi-ppiall