Null Models For Social Networks

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

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

ABSTRACT

In this paper, we present a heuristic method to generate graphs with specific motif counts. Motif counts imply many structural properties of a graph, such as clustering coefficient and degree distribution (Section 6), and can be used to cluster networks into meaningful real-world categories [13]. By constructing graphs with specific motif counts, we hope to build models that will imitate the functionality of specific networks.

Our method is based on hill-climbing. We perform successive transformations on an initial graph, keeping the ones that reduce error and discarding the ones that don't. Running this algorithm on 9 real-world networks shows substantial improvement over the baseline. On our metabolic network, our model's motif counts were identical to the motif counts of the original network. On our power grid network, the average relative error between the motif counts of our model and the network was 0.006. All networks saw at least a 58% decrease in average relative error, with most networks seeing much better performance.

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 [5][6] models all networks with n participants and m connections with a graph of n nodes and m randomly placed edges. The Watts-Strogatz model [15] models all networks by connecting nodes to their nearest neighbors, then filling the rest of the graph with randomly placed edges.

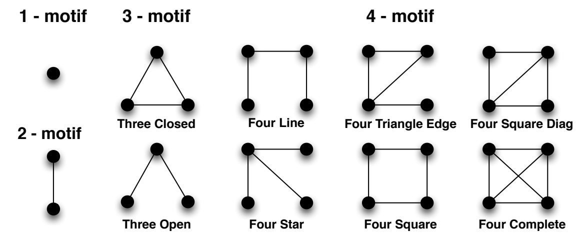


Figure 1: Graphical representation of the motifs.

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 [10] 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 forwarding 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.

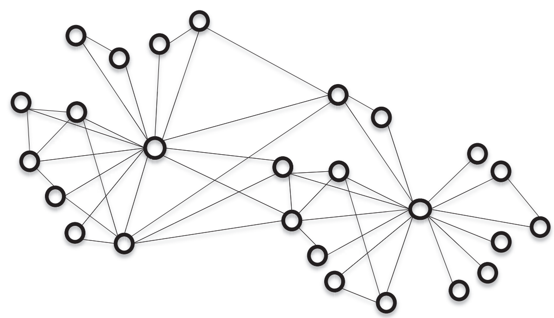


Figure 2: A network with motif distribution {numNodes: 25, numEdges: 44, mtThreeClosed: 21, mtThreeOpen: 151, mfFourLine: 134, mfFourSquare: 1, mfFourStar: 305, mfFourTriangleEdge: 155, mfFourSquareDiag: 26, mfFourComplete: 2}.

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 [13]. Therefore, by generating graphs with similar motif counts, we hope to build graphs with similar function to real-world networks.

The rest of the paper is organized as follows: Section 2 formulates the problem; Section 3 discusses related work; Section 4 describes the algorithm for generating random graphs; Section 5 presents the experimental results; Section 6 describes our method for predicting the degree distribution from the motif counts, and Section 7 describes our future work.

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

Motifs. Graph motifs are an important local property which are defined as recurrent and statistically significant subgraphs or patterns. Many researchers have studied graph motifs. Milo et al. [8] uses network motifs to uncover structural design principles in complex networks. Shen-Orr et al. [14] systematically detects network motifs in one of the best-characterized regulation networks, that of direct transcriptional interactions in Escherichia coli. Alon et al. [2] reviews network motifs and their functions, with an emphasis on experimental studies. Recently, Bhuiyan et al. [4] proposes a method called *GUISE*, which uses a Markov Chain Monte Carlo (MCMC) sampling method for constructing the approximate motif distribution of a large network.

Generating random graphs. Generating random graphs is an important problem in social network analysis.

Most work in this area has involved degree distributions. Molloy et al. [9] provides a model for generating random graphs with a given degree sequence. Rao et al. [12] proposes an MCMC based model using switches along alternating cycles for generating random graphs. Bayati et al. [3] presents a nearly-linear time algorithm for counting and randomly generating simple graphs with a given degree sequence in a certain range.

Our approach is inspired by Milo et al. [11], who proposes a method of generating a random graph with a prescribed degree sequence. 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.) Once 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 propose a heuristic method for generating a graph given its motif counts and degree distribution.

4.1 Hill climbing

Initialize graph $G = (V, E)$ with prescribed degree sequence
motifCounts $\leftarrow$ 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 $\leftarrow$ CountMotifs(G)
  if *error(newMotifCounts) < error(motifCounts)* **then**
    motifCounts $\leftarrow newMotifCounts$
  else
    Return graph to original state
  end
until *computation time limit exceeded*;

Algorithm 2: Naive approach

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:

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

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.

4.2 Optimizations

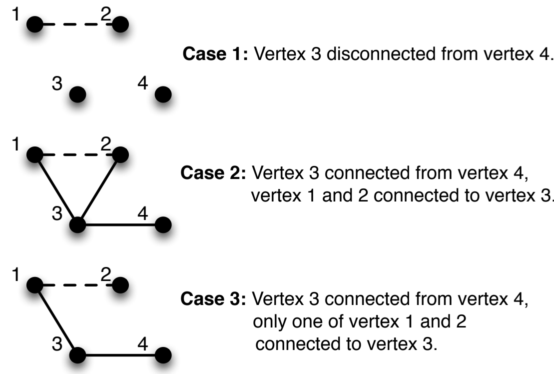
As written, this solution is very slow. Counting motifs is $O(|V|^4)$ and takes unacceptably long in practice. To get good results in practice, we need a large number of rewiring steps, so ideally each rewiring step should take less than a second.

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 (i.e. a neighbor of a 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.

Case where vertex 4 is not an immediate neighbor of vertex 1 or 2, but vertex 3 is.



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. This is because the vertex pairs that require the most computation are ones where both vertices are first-degree neighbors. To consider all possible pairs we must iterate through the first degree neighbors twice, hence the bound $O((d_1 + d_2)^2)$.

4.3 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.

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.

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. 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

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 Random restarts

In our third experiment we implemented the method of random restarts. Since our old error function did not produce zero error on correct motif counts, we changed the error function to

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.

$$\text{Average relative error} = \frac{1}{\ell}\sum_{i=1}^{\ell} \text{error}_i$$

$$\text{error}_i = |\text{counts}_i - \widehat{\text{counts}_i}| \text{ if counts}_i = 0 \quad (2)$$

$$\text{error}_i = \frac{|\text{counts}_i - \widehat{\text{counts}_i}|}{\text{counts}_i} \text{ if counts}_i \neq 0$$

where ℓ is the number of different motif types.

In Table 4 we run the random restart algorithm for 432000 seconds (5 days), running 7 trials for each network. We see this result produces negative improvement over the baseline. The baseline was calculated by running three trials of the previous algorithm for 432000 seconds and averaging the results.

On the networks aut-as19971108, aut-as19990628, and ppi-ppiapms, our algorithm never reached a "local minimum," so the random restart performance was similar to the baseline. On met-HI (which reached 2406 local minima on average), both random restarts and baseline achieved perfect performance. On the other networks performance was actually worse. We think this is because we started randomly rewiring edges before a local minimum was reached. In our next experiment we will tweak parameters to account for this situation.

6. PREDICTING DEGREE DISTRIBUTION FROM MOTIF COUNTS

In previous sections we assumed that we knew the degree distribution of the graph. Now we relax this assumption and predict the degree distribution from the motif counts. In the future we will use this to generate graphs given only their motif counts.

We assume that all the graphs we generate have a power law degree distribution. We use the motif counts as features to predict the power law exponent α. Given α and a normalization constant, producing an actual distribution is straightforward.

In this paper, we use Gradient Boosted Regression Trees (GBRT) [7] as the main regression model. Gradient Boosted Regression Trees is a useful machine learning method for regression problems, which is also an ensemble method that combines multiple weak prediction models. It constructs the model in a stage-wise fashion and

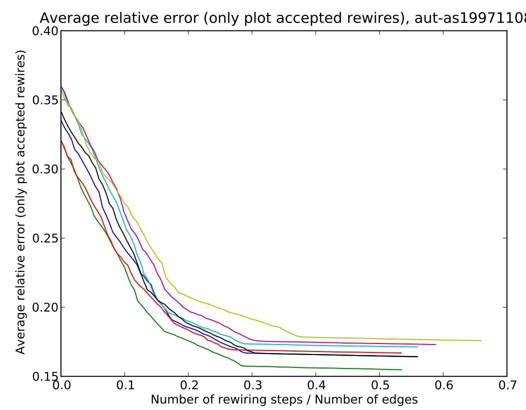


Figure 3: Error, network aut-as19971108. Only plot hill climbing steps that were successful.

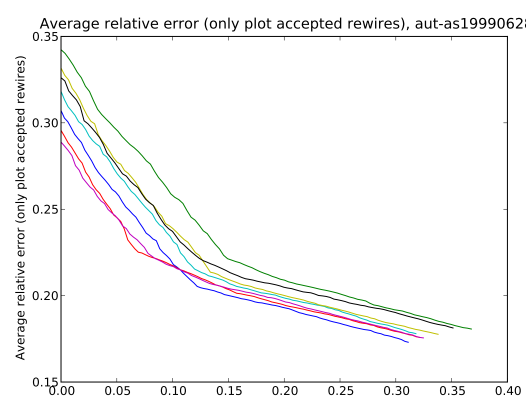


Figure 4: Error, network aut-as19990628. Only plot hill climbing steps that were successful.

Network	#Nodes	#Edges	Initial error	Best error	Baseline error	#Local minima	Rewires per local minimum
aut-as19971108	3015	5156	0.42023	0.01708	0.01719	1.00000	4410.28
aut-as19990628	5322	10163	0.58112	0.03523	0.03380	1.00000	4963.14
col-netscience	1461	2742	5.45962	0.70892	0.61568	2143.14	1347.88
met-HI	1424	3423	17931.1	0.00000	0.00000	24.0000	2405.95
ppi-ppiapms	1622	9070	1.64981	0.67485	0.67251	1.00000	32969.1
pwr-power	4941	6594	0.60097	0.10435	0.01059	19630.5	1585.99

Table 4: Method of random restarts.

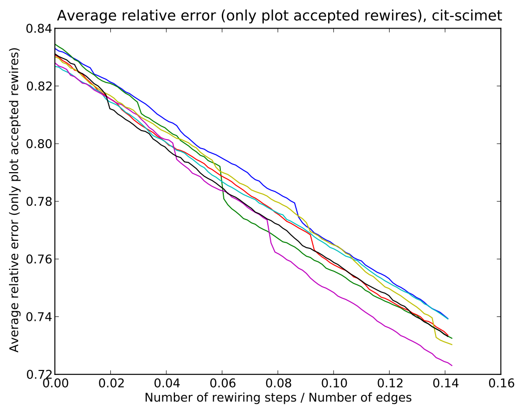


Figure 5: Error, network cit-scimet. Only plot hill climbing steps that were successful.

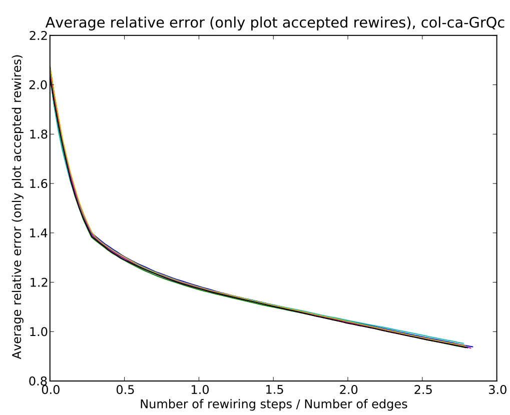


Figure 6: Error, network col-ca-GrQc. Only plot hill climbing steps that were successful.

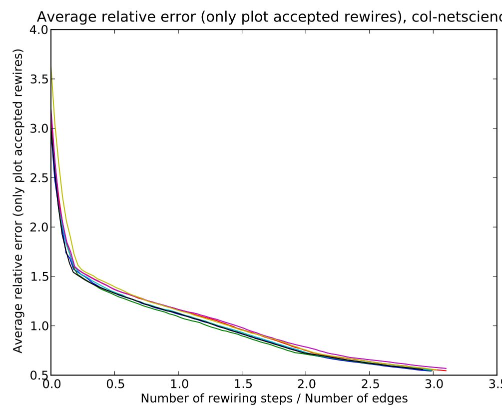


Figure 7: Error, network col-netscience. Only plot hill climbing steps that were successful.

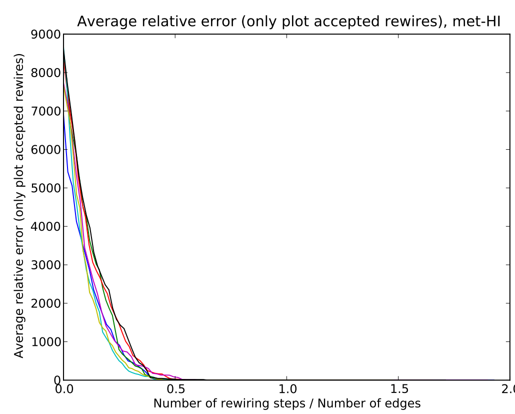


Figure 8: Error, network met-HI. Only plot hill climbing steps that were successful.

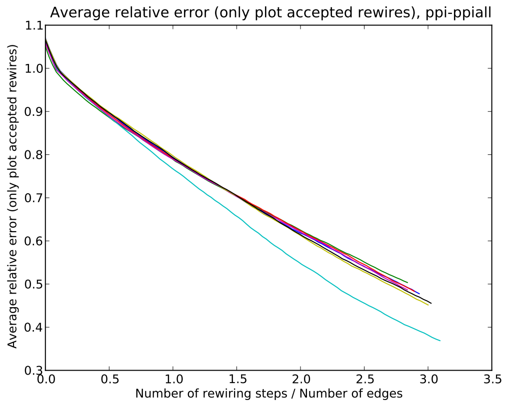


Figure 9: Error, network ppi-ppiall. Only plot hill climbing steps that were successful.

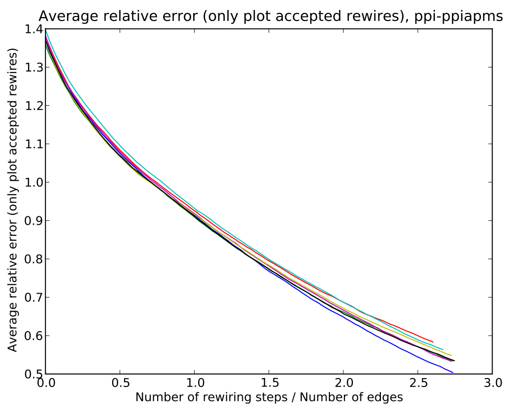


Figure 10: Error, network ppi-ppiapms. Only plot hill climbing steps that were successful.

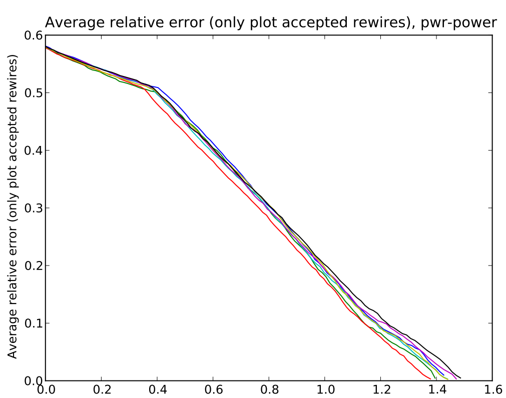


Figure 11: Error, network pwr-power. Only plot hill climbing steps that were successful.

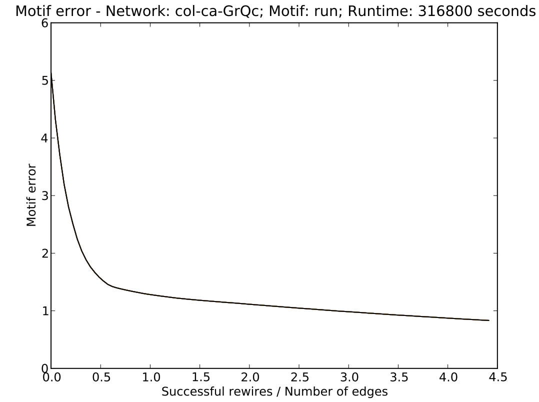


Figure 12: Experiment 2, network col-ca-GrQc.

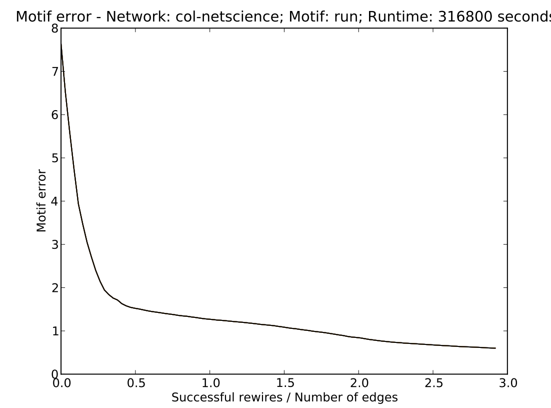


Figure 13: Experiment 2, network col-netscience.

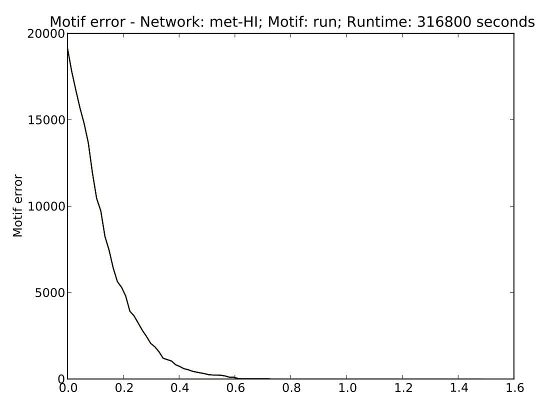


Figure 14: Experiment 2, network met-HI.

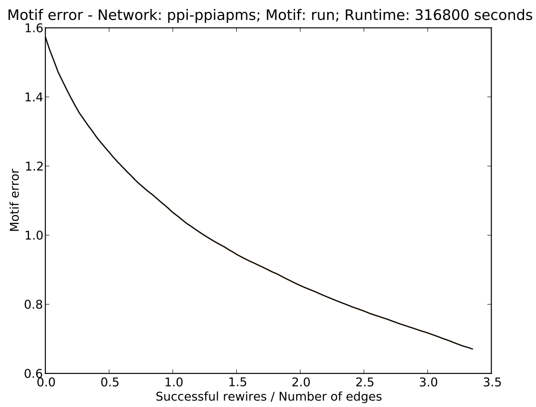


Figure 15: Experiment 2, network ppi-ppiapms.

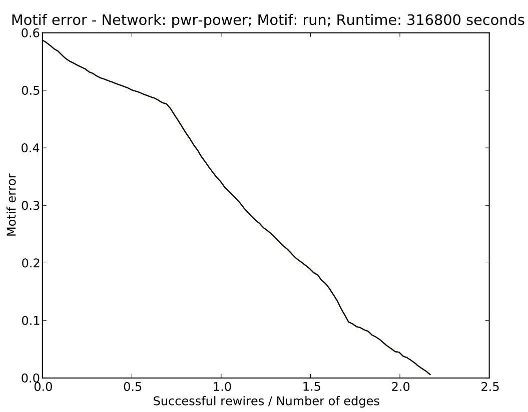


Figure 16: Experiment 2, network pwr-power.

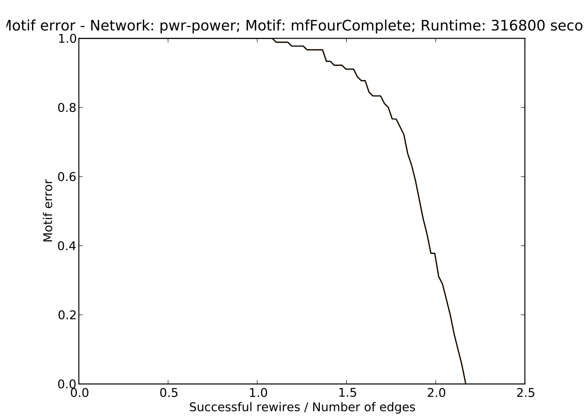


Figure 17: Experiment 2, network pwr-power, motif mfFour-Complete.

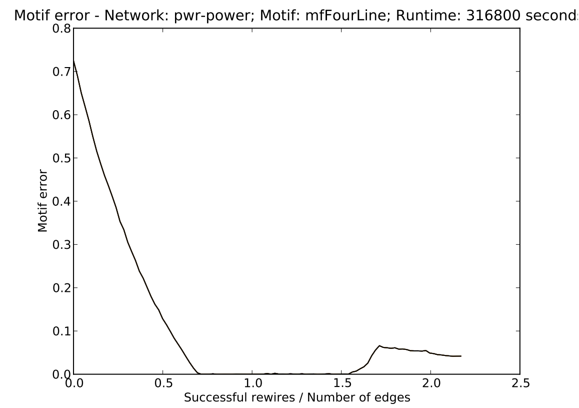


Figure 18: Experiment 2, network pwr-power, motif mf-FourLine.

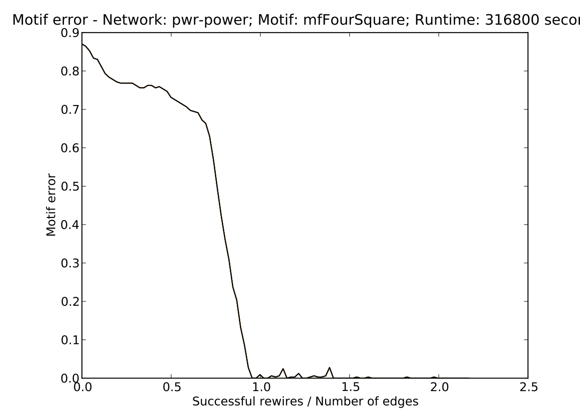


Figure 19: Experiment 2, network pwr-power, motif mf-FourSquare.

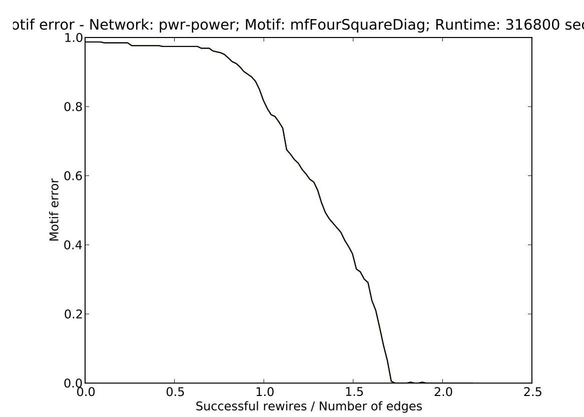


Figure 20: Experiment 2, network pwr-power, motif mf-FourSquareDiag.

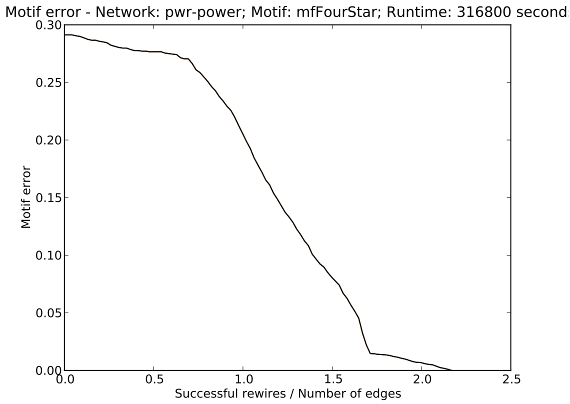


Figure 21: Experiment 2, network pwr-power, motif mfFourStar.

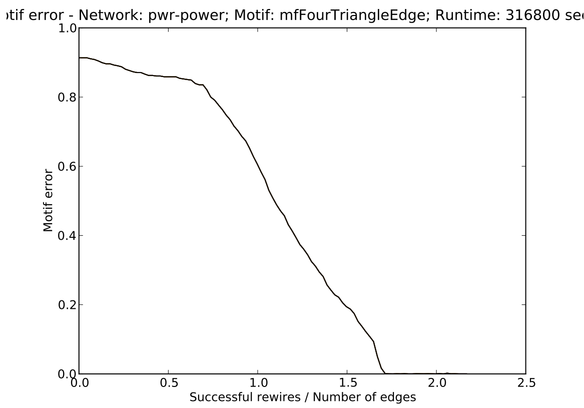


Figure 22: Experiment 2, network pwr-power, motif mfFourTriangleEdge.

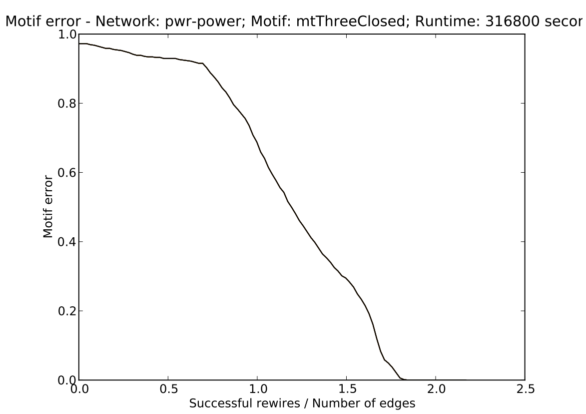


Figure 23: Experiment 2, network pwr-power, motif mtThreeClosed.

generalizes the model by optimizing an arbitrary differentiable loss function which in our case is the square loss function

$$\mathcal{L} = (f(x) - y)^2 \tag{3}$$

where y is the label and $f(x)$ is our prediction for y.

More precisely, GBRT trains a lot of small tree regression models (the depth of each tree is 5), each with high bias. Instead of using a uniform weight of each model to prevent overfitting, GBRT focus on adding new trees to minimize the current remaining regression error at each iteration.Let $f(x_i)$ denote the prediction score of sample x_i and $\mathcal{L}(f(x_1), ..., f(x_n))$ as the loss function of the model, which reaches a minimum if $f(x_i) = y_i$ for all x_i. For each new tree T_i, that is added into the existing classifier and the current prediction $P(x_i)$, we use the following step:

$$P(x_i) \leftarrow P(x_i) + \beta \frac{\mathcal{L}}{P(x_i)}$$

where β is the learning rate, and the gradient $\frac{\mathcal{L}}{P(x_i)}$ is approximated with the prediction score of regression tree [16]. Algorithm 3 shows the details of GBRT.

Input: Data set $\mathcal{D} = \{(x_1, y_1), ..., (x_n, y_n)\}$, learning rate β, #Trees M, depth d
Output: Combined tree model T
Initialization: $\forall i, p_i \leftarrow y_i$
for $i = 1$ *to* M **do**
  $T_i \leftarrow DT(\{(x_1, p_1), ..., (x_n, p_n)\}, f, d)$
  for $j = 1$ *to* n **do**
    $p_i \leftarrow p_i - \beta T_i(x_i)$
  end
end
$T \leftarrow \beta \sum_{i=1}^{M} T_i$
return T

Algorithm 3: Gradient Boosted Regression Trees (GBRT). DT indicates the decision tree model which has three parameters, data D, features f and the depth d of tree.

Implementation note. In the Decision Tree model, we train $M = 1000$ trees and for each tree randomly select $f = \#\text{features}/10$ and set the depth to 5. If M is too big, the algorithm starts overfitting. As for learning rate, we use $\beta = 0.001$.

Evaluation measures. To quantitatively evaluate the performance of our model, we conducted experiments with different networks. For each network, we evaluated the approach in terms of average relative error (ARE).

Prediction performance. We use 168 different networks as input data to evaluate the proposed model, with 10-fold cross validation. In order to avoid bias, we test the data 10 times, and get the predicted α for each network. This model achieves average relative error 0.005521 with standard deviation 0.00316. Some sample predictions are shown in Table 6.

7. FUTURE WORK

So far we know how to (approximately) generate graphs with given motif counts, assuming we know the degree distribution of the graph. We can also predict properties of the degree distribution from the motif counts. The next logical step is to use the predicted degree distribution to generate a graph.

We also hope to improve the random restarts method by increasing the number of unsuccessful rewires required for a random restart.

Table 5: Predicted degree distribution exponents.

Network	α	$\hat{\alpha}$	Relative error
aut-as19971108.txt	2.53315	2.52657	0.00259
aut-as19990628.txt	2.52159	2.54907	-0.01090
cit-scimet.txt	1.95277	1.93871	0.00720
col-ca-GrQc.txt	1.88493	1.86740	0.00930
col-netscience.txt	1.89612	1.89536	0.00039
met-HI.txt	1.88824	1.87993	0.00439
ppi-ppiall.txt	1.89357	1.88467	0.00469
ppi-ppiapms.txt	1.89285	1.87544	0.00919
pwr-power.txt	1.78450	1.78343	0.00059

We will also tweak the number of random rewirings that happen at each random restart. Currently those numbers are $|E|$ and $|E|/8$, but we intend to run simulations for several different parameter values.

8. REFERENCES

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

[2] U. Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450–461, 2007.

[3] M. Bayati, J. H. Kim, and A. Saberi. A sequential algorithm for generating random graphs. *Algorithmica*, 58(4):860–910, 2010.

[4] M. A. Bhuiyan, M. Rahman, M. Rahman, and M. Al Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 91–100. IEEE, 2012.

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

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

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

[8] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[9] 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.

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

[11] 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.

[12] A. R. Rao, R. Jana, and S. Bandyopadhyay. A markov chain monte carlo method for generating random (0, 1)-matrices with given marginals. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 225–242, 1996.

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

[14] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature Genetics*, 31(1):64–68, 2002.

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

[16] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun. A general boosting method and its application to learning ranking functions for web search. In *Advances in neural information processing systems*, pages 1697–1704, 2007.

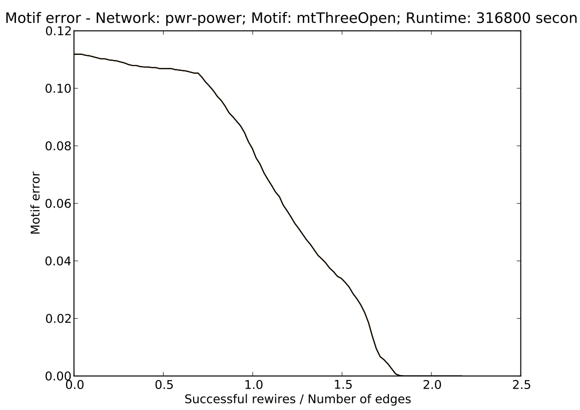


Figure 24: Experiment 2, network pwr-power, motif mtThree-Open.