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.
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).

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.


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.”

We provide Algorithm 1 and our method is inspired by Milo’s [11]. Algorithm 1: Milo’s approach for generating random graphs with prescribed degree sequences.

| Input: | A motif distribution $D$, where each motif has at most 4 vertices. |
| Output: | A graph $G$ with a motif distribution that closely approximates $D$. |

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
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:

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

where \(\ell\) 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 motifs 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.

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:

- **Case 1:** Vertex 3 disconnected from vertex 4.
- **Case 2:** Vertex 3 connected from vertex 4, vertex 1 and 2 connected to vertex 3.
- **Case 3:** Vertex 3 connected from vertex 4, only one of vertex 1 and 2 connected to vertex 3.

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.
was 8039.58 and the final error was 0.46768.)

The error graphs for aut-as19971108 (Figure 3) and col-netscience (Figure 7) converge to an asymptote. This indicates that running 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).

The 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
<table>
<thead>
<tr>
<th>Network</th>
<th>Nodes</th>
<th>Edges</th>
<th>Initial error</th>
<th>Final error</th>
<th>Successful rewires</th>
</tr>
</thead>
<tbody>
<tr>
<td>aut-as19971108</td>
<td>3015</td>
<td>5156</td>
<td>0.34216</td>
<td>0.16/17</td>
<td>2951.00</td>
</tr>
<tr>
<td>aut-as19990628</td>
<td>5322</td>
<td>10163</td>
<td>0.31371</td>
<td>0.17723</td>
<td>3397.28</td>
</tr>
<tr>
<td>cit-scimt</td>
<td>3085</td>
<td>13474</td>
<td>0.83063</td>
<td>0.73247</td>
<td>8290.71</td>
</tr>
<tr>
<td>col-ca-GrQc</td>
<td>5242</td>
<td>14484</td>
<td>2.05549</td>
<td>0.93973</td>
<td>40583.8</td>
</tr>
<tr>
<td>col-netscience</td>
<td>1461</td>
<td>2742</td>
<td>3.13864</td>
<td>0.55110</td>
<td>1921.71</td>
</tr>
<tr>
<td>met-HI</td>
<td>1424</td>
<td>3423</td>
<td>8039.58</td>
<td>0.46768</td>
<td>5703.57</td>
</tr>
<tr>
<td>ppi-ppial</td>
<td>3258</td>
<td>12930</td>
<td>1.06249</td>
<td>0.46058</td>
<td>38131.1</td>
</tr>
<tr>
<td>ppi-ppiapms</td>
<td>1622</td>
<td>9070</td>
<td>1.37580</td>
<td>0.54219</td>
<td>24581.7</td>
</tr>
<tr>
<td>pwr-power</td>
<td>4941</td>
<td>6594</td>
<td>0.57996</td>
<td>0.00282</td>
<td>9485.4</td>
</tr>
</tbody>
</table>

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

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

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

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

\[
\text{error}_i = \begin{cases} 
|\text{counts}_i - \hat{\text{counts}}_i| & \text{if counts}_i = 0 \\
\frac{|\text{counts}_i - \hat{\text{counts}}_i|}{\text{counts}_i} & \text{if counts}_i \neq 0
\end{cases}
\]

where \( \ell \) 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 \( \alpha \). Given \( \alpha \) 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
<table>
<thead>
<tr>
<th>Network</th>
<th>#Nodes</th>
<th>#Edges</th>
<th>Initial error</th>
<th>Best error</th>
<th>Baseline error</th>
<th>#Local minima</th>
<th>Rewires per local minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>aut-as19971108</td>
<td>3015</td>
<td>5156</td>
<td>0.42023</td>
<td>0.01708</td>
<td>0.01719</td>
<td>1.00000</td>
<td>4410.28</td>
</tr>
<tr>
<td>aut-as19990628</td>
<td>5322</td>
<td>10163</td>
<td>0.58112</td>
<td>0.03523</td>
<td>0.03380</td>
<td>1.00000</td>
<td>4963.14</td>
</tr>
<tr>
<td>col-netscience</td>
<td>1461</td>
<td>2742</td>
<td>5.45962</td>
<td>0.70892</td>
<td>0.61568</td>
<td>2143.14</td>
<td>1347.88</td>
</tr>
<tr>
<td>met-HI</td>
<td>1424</td>
<td>3423</td>
<td>17931.1</td>
<td>0.00000</td>
<td>0.00000</td>
<td>24.00000</td>
<td>2405.95</td>
</tr>
<tr>
<td>ppi-ppiapms</td>
<td>1622</td>
<td>9070</td>
<td>1.64981</td>
<td>0.67485</td>
<td>0.67251</td>
<td>1.00000</td>
<td>32969.1</td>
</tr>
<tr>
<td>pwr-power</td>
<td>4941</td>
<td>6594</td>
<td>0.60097</td>
<td>0.10435</td>
<td>0.01059</td>
<td>19630.5</td>
<td>1385.99</td>
</tr>
</tbody>
</table>

Table 4: Method of random restarts.

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

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

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

Figure 8: Error, network met-HI. Only plot hill climbing steps that were successful.
Figure 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.
Motif error - Network: ppi-ppiapms; Motif: run; Runtime: 316800 seconds

Figure 15: Experiment 2, network ppi-ppiapms.

Motif error - Network: pwr-power; Motif: run; Runtime: 316800 seconds

Figure 16: Experiment 2, network pwr-power.

Motif error - Network: pwr-power; Motif: mfFourComplete; Runtime: 316800 seconds

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

Motif error - Network: pwr-power; Motif: mfFourLine; Runtime: 316800 seconds

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

Motif error - Network: pwr-power; Motif: mfFourSquare; Runtime: 316800 seconds

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

Motif error - Network: pwr-power; Motif: mfFourSquareDiag; Runtime: 316800 seconds

Figure 20: Experiment 2, network pwr-power, motif mfFourSquareDiag.
Figure 21: Experiment 2, network pwr-power, motif mf-FourStar.

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

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

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

\[ \mathcal{L} = (f(x) - y)^2 \]  

(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 \( f(x) \) denote the prediction score of sample \( x \), 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 \( \beta \) 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.

```
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.

Input: Data set \( D = \{(x_1, y_1), ..., (x_n, y_n)\} \), learning rate \( \beta \), \#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

        \( p_k \leftarrow p_k - \beta T_i(x_k) \)

    end

end

\( T \leftarrow \beta \sum_{i=1}^{M} T_i \)

return \( T \)
```

**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 \( \alpha \) 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 restart method by increasing the number of unsuccessful rewires required for a random restart.
Table 5: Predicted degree distribution exponents.

<table>
<thead>
<tr>
<th>Network</th>
<th>α</th>
<th>̂α</th>
<th>Relative error</th>
</tr>
</thead>
<tbody>
<tr>
<td>aut-as19971108.txt</td>
<td>2.53315</td>
<td>2.52657</td>
<td>0.00259</td>
</tr>
<tr>
<td>aut-as19990628.txt</td>
<td>2.52159</td>
<td>2.54907</td>
<td>-0.01090</td>
</tr>
<tr>
<td>cit-scim.met.txt</td>
<td>1.95277</td>
<td>1.93871</td>
<td>0.00720</td>
</tr>
<tr>
<td>col-ca-GrQc.txt</td>
<td>1.88493</td>
<td>1.86740</td>
<td>0.00930</td>
</tr>
<tr>
<td>col-netscience.txt</td>
<td>1.89612</td>
<td>1.89536</td>
<td>0.00039</td>
</tr>
<tr>
<td>met-HI.txt</td>
<td>1.88824</td>
<td>1.87993</td>
<td>0.00439</td>
</tr>
<tr>
<td>ppi-ppiall.txt</td>
<td>1.89357</td>
<td>1.88467</td>
<td>0.00469</td>
</tr>
<tr>
<td>ppi-ppiapms.txt</td>
<td>1.89285</td>
<td>1.87544</td>
<td>0.00919</td>
</tr>
<tr>
<td>pwr-power.txt</td>
<td>1.78450</td>
<td>1.78343</td>
<td>0.00059</td>
</tr>
</tbody>
</table>

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


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