LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-678723

A Clustering Graph Generator

M. Winlaw, H. DeSterck, G. Sanders

October 28, 2015

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

A Clustering Graph Generator*

Manda Winlaw, Hans De Sterck and Geoffrey Sanders

August 27, 2015

1 Introduction

In very simple terms a network can be defined as a collection of points joined together by lines. Thus, networks can be used to represent connections between entities in a wide variety of fields including engineering, science, medicine, and sociology. Many large real-world networks share a surprising number of properties, leading to a strong interest in model development research and techniques for building synthetic networks have been developed, that capture these similarities and replicate real-world graphs. Modeling these real-world networks serves two purposes. First, building models that mimic the patterns and properties of real networks helps to understand the implications of these patterns and helps determine which patterns are important. If we develop a generative process to synthesize real networks we can also examine which growth processes are plausible and which are not. Secondly, high-quality, large-scale network data is often not available, because of economic, legal, technological, or other obstacles [7]. Thus, there are many instances where the systems of interest cannot be represented by a single exemplar network. As one example, consider the field of cybersecurity, where systems require testing across diverse threat scenarios and validation across diverse network structures. In these cases, where there is no single exemplar network, the systems must instead be modeled as a collection of networks in which the variation among them may be just as important as their common features. By developing processes to build synthetic models, so-called graph generators, we can build synthetic networks that capture both the essential features of a system and realistic variability. Then we can use such synthetic graphs to perform tasks such as simulations, analysis, and decision making. We can also use synthetic graphs to performance test graph analysis algorithms, including clustering algorithms and anomaly detection algorithms.

One network property of particular interest is the network community structure. In a network, a community can broadly be defined as a collection of nodes that share a large number of connections internally with many fewer connections to nodes outside the collection [27]. The size and number of communities in a network can provide insight into the underlying network. As well, each community itself can be examined for patterns and properties and since the communities are smaller than the overall network this makes the analysis easier. However, the study of network community structure can often be difficult since a community can be quantified in many different ways [27]. One way to determine if a network has community structure is to look at the number of triangles in a graph and in particular the number of triangles each node is a part of. Thus, there are a number of graph generators that try

*This work was supported in part by NSERC of Canada and by the Scalable Graph Factorization LDRD Project, 13-ERD-072, under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-665572.

to replicate the triangle structure of real networks including the models developed by Seshadhri, Kolda and Pinar [34, 18] and Gutfraind, Meyers and Safro [14]. To preserve the community structure, we are interested in using a clustering algorithm as part of a graph generator. By using a clustering algorithm on a real-world network we can then use these clusters an input as well as the degree information to design a graph generator that better preserves the community structure that a graph generator focused only on the triangle structure. In our graph generator, we separate out the between cluster and within cluster behavior and use techniques to generate random graphs with prescribed degree sequences to build our synthetic networks. We then look at the properties of the synthetic networks generated from our algorithm using inputs from real networks and compare these properties to the real network properties as well as to the properties of some popular existing graph generators. Before presenting our graph generating algorithm and our numerical results, we first discuss some of the interesting properties common to many networks of interest. Then we introduce some of the prevailing graph generators and present the clustering algorithm, the Louvain algorithm, we will be using to generate clusters. We conclude by looking at avenues for future research.

2 Network Properties

We refer to the networks we are interested in as "interaction networks" [34]. This describes a wide variety of networks including social networks, citation networks, collaboration networks, computer traffic networks, and gene regulation networks. Despite their diverse nature these interaction networks share many similarities, including degree distribution, diameter, clustering coefficient and assortativity coefficient. We consider each of these properties in turn and describe the similarities across different interaction networks. In the discussion that follows we only consider undirected graphs (i.e. networks where none of the edges have a direction).

In interaction networks, one of the most commonly examined properties is the degree distribution. The degree of a node or vertex in an undirected graph is the number of edges connected to it, thus the degree distribution simply describes the distribution of degrees in the graph. The degree distribution of a graph follows a power law distribution if the number of nodes N_k with degree k is given by

$$N_k \propto k^{-\gamma} \tag{1}$$

where $\gamma > 0$ is called the power law exponent. So, any graph that has a power-law degree distribution will have a lot of nodes with small degree and only a small number of nodes with large degree. Most real-world networks have a power law degree distribution. Power laws have been found in the Internet [12], the Web [6, 16], citation graphs [33], online social networks [8] and many others. Thus, any graph generator that we develop should have a power-law degree distribution.

Perhaps one of the most well known properties common to many real-world networks is known as the "small-world" property. One way to measure this property is to use the diameter of a graph. A graph has diameter D if every pair of nodes can be connected by a path of length at most D edges. The diameter of most real-world graphs is relatively small suggesting that most nodes in the graph are relatively close to one another hence the "small-world" description. However, the diameter can be affected by outliers so we define the integer effective diameter to measure the pair-wise distance between nodes. The integer effective diameter is the minimum number of steps in which some fraction of all connected pairs of nodes can reach each other. The effective diameter has been found to be small for large real-world graphs, like the Internet, the Web, and online social networks [2, 21, 22].

Since we are particularly interested in the community structure of networks we examine one of the measures most commonly used to determine community structure, the clustering coefficient. However, as mentioned previously there are many ways to quantify a community in a network and thus there are many measures of community structure in a network. We present the clustering coefficient since it is one of the most popular measures for determining community structure. The clustering coefficient is defined as

$$C = \frac{3 \times \text{Total Number of Triangles}}{\text{Total Number of Wedges}}, \tag{2}$$

where a *wedge* is a path of length 2 and $0 \leq C \leq 1$. If connections between vertices are made at random such that the probability of an edge occurring is the same for all edges, then the clustering coefficient takes the value

$$C = \frac{1}{n} \frac{\left[\langle k^2 \rangle - \langle k \rangle\right]^2}{\langle k \rangle^3} \tag{3}$$

where $\langle k^m \rangle = \frac{1}{n} \sum_{i=1}^{n} k_i^m$ and k_i is the degree of node i [27]. If $\langle k^2 \rangle$ and $\langle k \rangle$ have fixed finite values the clustering coefficient becomes small as $n \to \infty$, thus we can expect the clustering coefficient to be very small in large graphs assuming the connections between vertices are made at random. However, the typical value for C is between 0.1 and 0.5 in many real-world networks [34] which is much larger than suggested by Equation (3).

We can also define the local clustering coefficient for each node as

$$C_i = \frac{3 \times \text{Number of Pairs of Neighbors of } i \text{ that are Connected}}{\text{Number of Pairs of Neighbors of } i}, \tag{4}$$

The local clustering coefficient is often dependent on degree, where vertices with higher degree have a lower local clustering coefficient on average [27].

Another popular metric used to analyze networks is known as assortative mixing and in particular assortative mixing by degree which is the tendency for vertices to connect to other vertices with similar degrees as their own. One way to measure assortative mixing is through the assortativity coefficient defined as:

$$r = \frac{\sum_{ij} \left(A_{ij} - k_i k_j / 2m\right) k_i k_j}{\sum_{ij} \left(k_i \delta_{ij} - k_i k_j / 2m\right) k_i k_j}, \tag{5}$$

where k_i is the degree of node i, $A_{ij} = 1$ if nodes i and j are connected and $A_{ij} = 0$ if nodes i and j are not connected, m is the number of edges in the graph, and $\delta_{ij} = 1$ if $A_{ij} = 1$. Note that r is in fact a Pearson correlation coefficient so the values of r range from -1 to 1. The network has perfect assortative mixing if $r = 1$, is non-assortative if $r = 0$ and is completely dissortative (i.e. vertices connect to others with very different degrees) if $r = -1$. As noted in [27] social networks tend to have positive r values while technological, information and biological networks tend to have negative r values.

As with the clustering coefficient we can look at assortative mixing at the local level. Before defining local assortativity we first define $q(k)$, the remaining degree distribution. It is derived from the degree distribution $p(k)$ and defined as $q(k) = \frac{p(k+1)}{\sum_{j \geq 1} p(j)}$ where $p(k+1)$ is the probability of a node occurring with degree $k+1$. The local assortativity [31], can then be defined as

$$\rho_i = \frac{j(j+1)(\overline{k} - \mu_q)}{2m\sigma_q^2}, \tag{6}$$

where j is the excess degree of node i, $\overline{k}$ is the average excess degree of node i's neighbors, μ_q is the mean of the remaining degree distribution $q(k)$ and σ_q^2 is its variance.

The properties highlighted above are often targeted when building graph generators. In the next section we outline some of the current graph generators being used to replicate real-world graphs.

3 Existing Graph Generators

The most well-known graph generating model is widely attributed to Erdős and Rényi [11] and is often referred to as the "Erdős-Rényi model" or "Erdős-Rényi random graph". In the Erdős-Rényi (ER) graph each pair of nodes has an identical, independent probability of sharing an edge. The ER graph provably violates the degree distribution power law found in real-world networks. In fact, it can be shown that the degree distribution follows a Poisson distribution which is why this graph is also referred to as the "Poisson random graph". For this reason, Erdős-Rényi graphs are rarely used to model real-world networks.

In the ER graph model every edge has an equal probability of occurring and the expected degree of each node is the same. To overcome this limitation, Chung and Lu [9, 10] introduced a graph model whereby the expected degree distribution can be specified. Suppose we have the following expected degree sequence $\mathbf{k} = (k_1, k_2, ..., k_n)$ where k_i is the expected degree of node i and suppose the edge between node i and node j is chosen independently with probability p_{ij} where

$$p_{ij} = \frac{d_i d_j}{2s}, \tag{7}$$

and $s = \sum k_i/2$ is number of edges in the graph with degree sequence $\mathbf{k}$. Then the expected degree of node i in the Chung-Lu random graph is k_i. Thus, if we want to match the degree distribution of a real-world network using the Chung-Lu graph model we can do so by using the degree distribution of the real-world as an input. So, Chung-Lu graphs can produce graphs with power law degree distributions. However, the generated graphs have small clustering coefficients relative to real-world networks. We should also note that the ER graph model is equivalent to the Chung-Lu graph model with degree distribution $\mathbf{k} = (pn, pn, \ldots, pn)$ where p is the probability of nodes sharing an edge in the ER model and n is the number of nodes in the graph.

Similar to the Chung-Lu model we have the graph model that consists of the ensemble of all graphs with a prescribed degree sequence [3, 23, 25, 26, 28]. In the Chung-Lu model, only the expected degree of a node is equal to the given degree and in any particular instance, the degree of a node may actually differ from the given degree, however, in the model consisting of the ensemble of all graphs with a prescribed degree sequence, in every instance, the node degree matches the given node degree. Two common algorithms for generating instances of this model are the switching algorithm [24, 29, 32, 15] and the matching algorithm [25, 28, 24]. We use both algorithms in our graph generator and we will discuss these algorithms in more detail when we present our graph generator.

Next, we look at some of the more complex graphs used to model interaction networks. The Stochastic Kronecker Graph Model [8] also known as R-MAT has proved to be one of the most successful generative strategies for modeling real-world networks and is used as the graph generator for the Graph 500 Supercomputer Benchmark [13]. R-MAT generates a graph by recursively partitioning the adjacency matrix. There are typically four partitions with the probabilities of an edge falling into one of these partitions given by a, b, c, d where $a + b + c + d = 1$. Typically, $a \geq b$, $a \geq c$, $a \geq d$ and often $b = c$. Figure 1 shows a simple example of how the R-MAT model works. The figure shows the adjacency matrix for a given graph. Initially the matrix is divided into four parts. Partition b is then chosen and divided into

four parts. From there partition a is chosen. This process continues recursively until we actually reach the finest level of the adjacency matrix at which point we can add an edge to the graph by setting the value in the partition to 1. In the above explanation of the R-MAT model the probabilities a, b, c and d are the same for each partition. In the actual R-MAT algorithm an element of randomness is introduced so that these values vary slightly for each partition. The randomness is introduced to better match the properties of real-world networks and while the R-MAT model can generate graphs with power-law degree distributions, it does not generate graphs with high clustering coefficients [30].

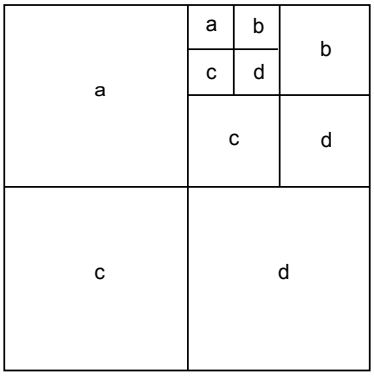


Figure 1: The R-MAT Model

The Block Two-Level Erdős-Rényi (BTER) model [34, 18] is designed to capture the underlying community structure in real-world networks. BTER graphs contain a scale free collection of dense ER subgraphs which the authors suggest is the underlying structure of most real-world networks. The BTER graph takes as an input the degree distribution and the clustering coefficient distribution and is able to match both the degree distribution and the local and global clustering coefficients of real-world networks fairly well. However, in BTER graphs nodes of degree d have neighbors with degree $\approx d$. Yet, there is more variation in the joint degree distribution of real-world graphs. Communities in real-world networks are not as homogeneous as depicted by the BTER graph.

There are many more graph generators than mentioned above. Chakrabarti and Faloutsos [7] provide a survey of some other models. They include preferential attachment [1], small-world models [36], copying models [19] and forest fire [20]. All of the complex models we have discussed thus far have been generative models. The other class of network generators, are known as graph editing models. In these models, we start with a given network and randomly change its components until the network becomes sufficiently different from the original network. Thus, we are able to introduce variability while preserving key structural properties. One notable graph editing method is known as MUSKETEER [14]. The method uses a multilevel approach to build graphs that preserve some of the original network structural properties while introducing realistic variability and is able to retain many of the real-world network properties.

4 The Louvain Clustering Algorithm

To build a graph generator via clustering we must first choose an algorithm to cluster our real-world networks. While our graph generating algorithm is designed to use clusters from any clustering algorithm we choose to use the Louvain algorithm [4] as our initial clustering method. The Louvain algorithm finds

community structure using a heuristic modularity optimization technique. Since communities within any network can loosely be defined as densely connected nodes that share few connections with nodes outside the community any community detection or clustering algorithm will attempt to partition a network into clusters or communities of densely connected nodes where the number of connections between clusters is small. One measure of partitioning success is defined by modularity, $Q \in [0, 1]$:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j), \tag{8}$$

where A_{ij} represents the edge weight between node i and j, $k_i = \sum_j A_{ij}$ is the sum of edge weights adjacent to vertex i, c_i is the community node i belongs to, the δ-function $\delta(c_i, c_j)$ is 1 if nodes i and j belong to the same community and 0 otherwise and $m = \frac{1}{2} \sum_{ij} A_{ij}$ is the sum of all the edge weights. Not only is modularity used to compare the quality of partitions obtained by different clustering algorithms, but it can also be used as an objective function to optimize in designing a community detection algorithm. However, exact modularity optimization is a computationally hard problem [5]; thus, many algorithms, including the Louvain algorithm, use a heuristic approach to modularity optimization.

The Louvain algorithm can be divided into two steps that are repeated iteratively to build hierarchical clusters. In the first step of the algorithm, each node in the graph is assigned to its own community. Each node is then considered sequentially and for each node i we calculate the change in modularity from moving node i to each of its neighboring communities. A neighboring community is any community that contains a neighbor of node i. Thus if node i has m neighbors we must calculate at most m changes in modularity. Once all of the modularity calculations have been completed we determine whether or not to move node i to a new community based on the largest of these calculations. If the largest change in modularity is positive we move node i to the community with the largest modularity change; otherwise, node i remains in its original cluster. Once all of the nodes have been considered we repeat this process until no nodes can be moved into new communities and thus no gains in modularity can be made. It should be noted that the order in which nodes are considered does change the algorithm's output; however, in practice, it does not seem to significantly affect the modularity obtained [4].

After the first step of the algorithm has been completed we move to the second step which consists of building a new network. Each node in the new network now represents a cluster found during the first step of the algorithm. The edges within a cluster are represented by self-loops where the edge weight is determined by the sum of the within cluster edge weights. The edges between nodes in the new graph are determined by the edges between clusters and the edge weights are determined by the corresponding between cluster edge weight sums. Once we have constructed the new graph we can repeat these two steps and the process can be repeated until no new clusters can be formed. At this point the algorithm stops.

5 The Clustering Graph Generator

As mentioned previously, our graph generator is built using the clusters generated from a clustering algorithm where we use the Louvain algorithm described above to generate clusters. We generate edges between clusters differently than edges within clusters, where we use more information to build the graph within a cluster than between clusters. Between clusters we use a modified version of the matching algorithm [25, 28, 24] also known as the configuration model. In the original matching algorithm each node is assigned a set of "stubs", which are sawn-off ends of the edges, according to the desired degree sequence. Then, pairs of stubs are chosen at random to create the edges of the network. If a self-edge or multiple edge is chosen then the entire network is discarded and the entire process starts over.

Alternatively, the method can be modified so that if a multiple edge or self-edge is drawn then the network is not discarded, we simply draw another pair of stubs. The network is only discarded if the final pair is a self-edge or a multiple-edge. To better illustrate how we use a modified matching algorithm to create edges between clusters we begin by considering the graph in Figure 2. Figure 2 depicts the original graph, and the hierarchical clusters generated by the Louvain algorithm where in the original graph the numbers represent node labels and in the level 1 and coarsest level graphs the numbers represent edge weights. For our purposes we are only interested in the original graph and the level 1 graph. The first step in our between clusters algorithm is to calculate the external degree for each node. The external degree for node i is the number of edges between node i and node j where node j does not belong to the same cluster as node i. For example, node 12 has external degree is 4. Once the external degree for each node has been calculated we use it to create a list of stubs for each cluster where each node appears in the list with multiplicity equal to their external degree. If we let the green nodes represent cluster 1, the black nodes cluster 2, the red nodes cluster 3 and the light blue nodes cluster 4, then we have the following lists: $C_1 = \{1, 2, 3, 5, 6, 6\}, C_2 = \{4, 7, 7, 8, 8\}, C_3 = \{9, 11, 11, 11\}, C_4 = \{12, 12, 12, 12, 14\}$. We can use these lists to create the edges but before we add an edge we must first decide what clusters any given edge will be between. To do this we use the level 1 graph given in Figure 2. The weights on the edges in this graph indicate how many edges there are between the clusters and we use these weights to calculate the probabilities of placing an edge between cluster I and J. The probability that an edge will be between cluster I and J is given as follows

$$P(I, J) = \frac{\text{Number of edges between clusters } I \text{ and } J}{\text{Number of external edges}} \tag{9}$$

In our example graph, $P(1, 2) = \frac{4}{10}$. Once we have chosen which clusters to place an edge between then we can randomly choose a value from C_I and C_J. If no edge exists between this pair of nodes then we add this edge to our generated graph and remove these values from C_I and C_J. For example, if on our first draw we chose to place an edge between clusters 1 and 2 and then chose 6 from C_1 and 7 from C_2 we would add edge (6,7) to our generated graph and C_1 and C_2 we now be $C_1 = \{1, 2, 3, 5, 6\}, C_2 = \{4, 7, 8, 8\}$. We continue to add edges in this way. Ideally we would only have to repeat this process , times where m is equal to the number of external edges but because we can get duplicate edges we complete $alpha * m$ draws where $alpha >= 1$. As well, we note that there is always the possibility that when we complete this procedure there are still values in the C_I lists although the number should be small. Algorithm **??** summarizes the algorithm for adding edges between clusters described above.

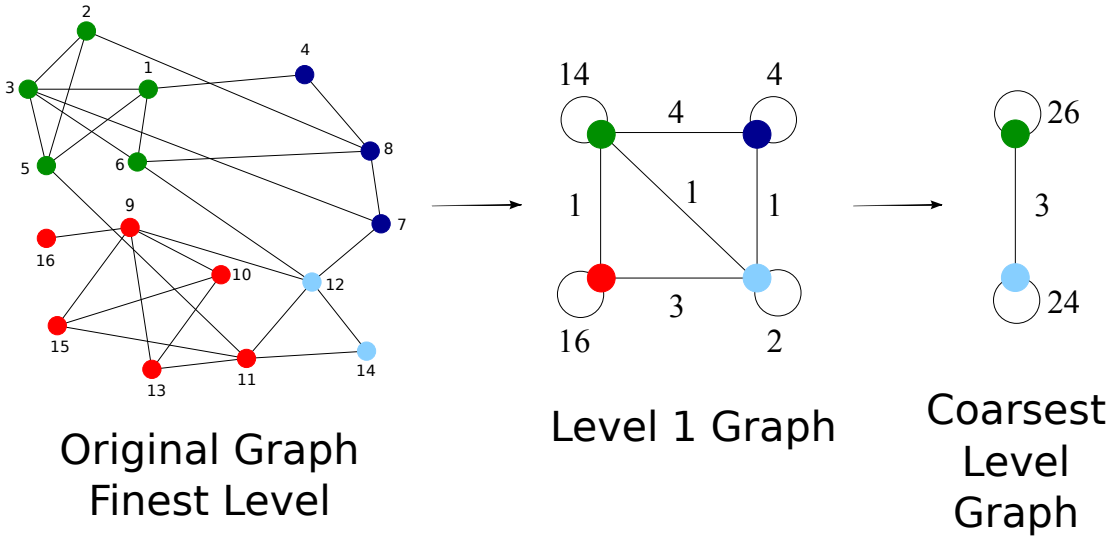


Figure 2: Simple Graph

Algorithm 1: Between Clusters Algorithm

for $I = 1$ *to* $nClusters$ **do**
  Create a list of stubs, C_I, based on the external degree of each node in the cluster;
  for $J = I + 1$ *to* $nClusters$ **do**
    Calculate $P(I, J) = \frac{\text{Number of edges between clusters } I \text{ and } J}{\text{Number of external edges}}$;
  end
end
for $i = 1$ *to* $\alpha \cdot nExternalEdges$ **do**
  Choose the two clusters I, and J to put an edge between with probability $P(I, J)$;
  Randomly choose stub p from C_I;
  Randomly choose stub k form C_J;
  if $p \neq k$ *and edge* (p, k) *isn't in the graph* **then**
    Add edge (p, k) to the graph;
    Remove p from C_I;
    Remove k from C_J;
  end
end

The algorithm for adding edges within a cluster is based on the switching algorithm [24, 29, 32, 15]. For each cluster, we create a graph from the cluster nodes and internal cluster edges and apply the switching algorithm to this graph. The switching algorithm uses a Markov chain to generate a random graph with a given degree sequence. To describe the switching algorithm suppose we have a graph G. We start with the original graph, G, and proceed to carry out a series of Monte Carlo switching steps. In each step a pair of edges (i, j), and (k, l) is selected at random and the ends of the edges are switched to give the edges

(i, l), and (k, j). If this switch does not lead to self-edges or multiple edges then the switch is performed; otherwise it is not performed. The entire process is performed $\alpha * m$ times where m is the number of edges in G and α is chosen large enough so that the Markov chain shows good mixing. This switching algorithm is performed on each of the clusters to generate all the edges within clusters. Algorithm 2 describes the above algorithm. Our graph generator combines Algorithms 1 and 2 to generate synthetic networks.

Algorithm 2: Within Clusters Algorithm

```
for I = 1 to nClusters do
    Create a graph, G_I, from the nodes and internal edges of cluster I;
    for i = 1 to α · nInternalEdges do
        Choose two edges (i, j), and (k, l) randomly from G_I;
        if i ≠ l and k ≠ j and edge (i, l) and (k, j) are not in G_I then
            Remove edges (i, j), and (k, l) from G_I;
            Add edges (i, l), and (k, j) to G_I;
        end
    end
end
```

6 Experimental Studies

We test our graph generator on various networks including including two collaboration networks (ca-GrQc, ca-AstroPh), one citation networks (cit-HepPh), and a technological network (as-735)[35]. All four networks are treated as undirected graphs. Table 1 lists the basic attributes of the graphs including the number of clusters generated by the Louvain algorithm.

Graph Name	Nodes	Edges	Number of Clusters
ca-GrQc	4158	13422	868
ca-AstroPh	18772	198080	2061
cit-HepPh	34546	210789	895
as-735	6474	12572	964

Table 1: Basic Attributes of the Networks

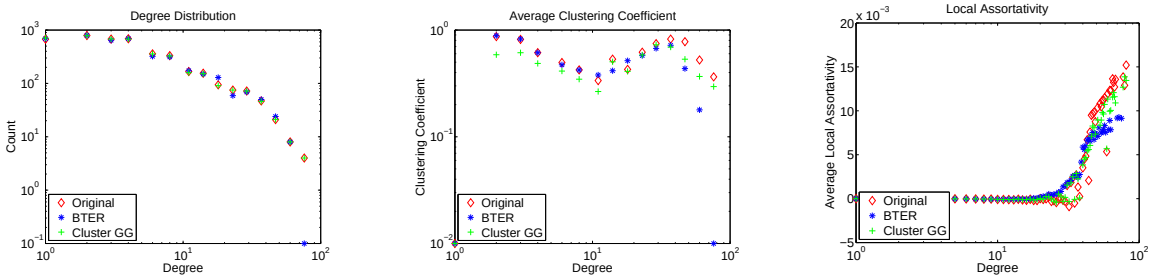
We compare our graph generator using an instance generated from our clustering graph generator algorithm (CGG). We compare the properties of the clustering generated graph to the original network and to an instance generated based on the BTER model using the code from the feastpack package [17]. The two properties we are most interested in comparing are the clustering coefficient and the assortativity coefficient. Tables 2 and 3 list the global clustering coefficient and the assortativity coefficient for each of the models, respectively. Figures 3, 4, 5, and 6 plot the degree distribution, the local clustering coefficient distribution and the local assortativity distribution for the models for each of the networks under study. From Table 2 we can see that the CGG tends to underestimate the global clustering graph generator. However, the GCC does a much better job of matching the assortativity coefficient which can be seen from Table 3. This can also be seen in Figures 3c,4c,5c and 6c. In the case of the citation network, cit-HepPh, GCC is able to match the sign of the assortativity coefficient whereas BTER is not.

Graph Name	Original	CGG	BTER
ca-GrQc	0.629	0.536	0.531
ca-AstroPh	0.318	0.148	0.326
cit-HepPh	0.146	0.059	0.154
as-735	0.01	0.016	0.06

Table 2: Global Clustering Coefficient of Each Model.

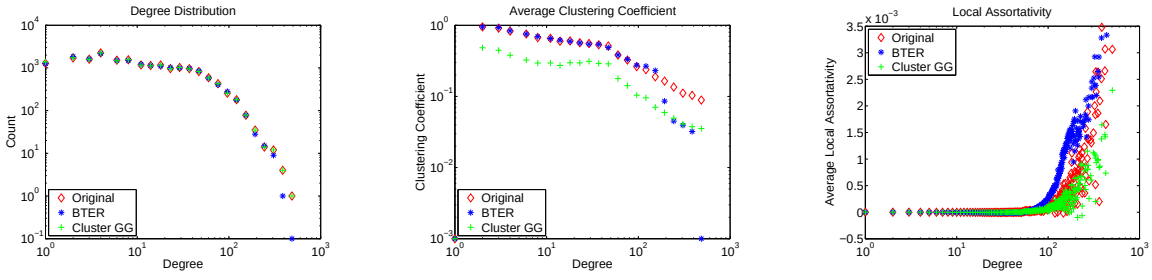
Graph Name	Original	CGG	BTER
ca-GrQc	0.63919	0.64715	0.59470
ca-AstroPh	0.20513	0.14322	0.46496
cit-HepPh	-0.00629	-0.00221	0.28205
as-735	-0.18176	-0.16292	-0.08315

Table 3: Assortativity Coefficient of Each Model.



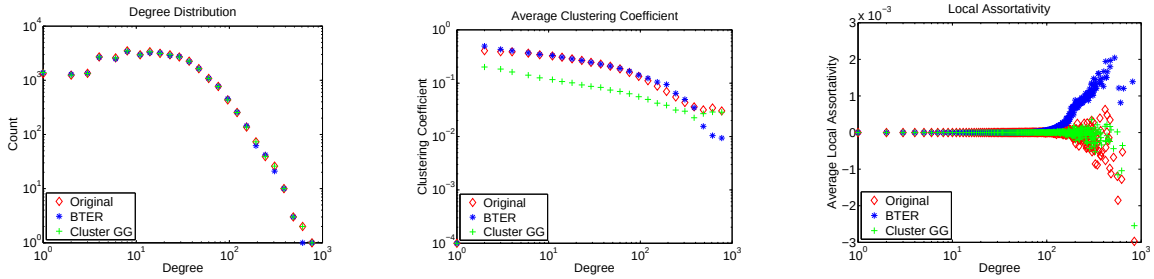
(a) Degree Distribution (b) Average Clustering Coefficient (c) Local Assortativity

Figure 3: Comparison of Properties of Various Models for the ca-GrQc Network.



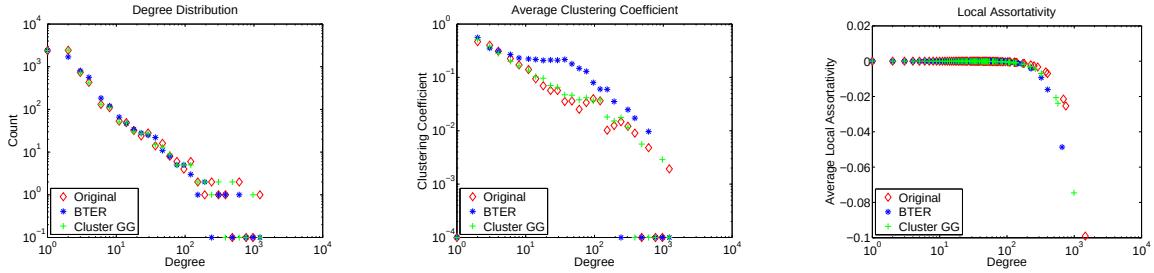
(a) Degree Distribution (b) Average Clustering Coefficient (c) Local Assortativity

Figure 4: Comparison of Properties of Various Models for the ca-AstroPh Network.



(a) Degree Distribution (b) Average Clustering Coefficient (c) Local Assortativity

Figure 5: Comparison of Properties of Various Models for the cit-HepPh Network.



(a) Degree Distribution (b) Average Clustering Coefficient (c) Local Assortativity

Figure 6: Comparison of Properties of Various Models for the as-735 Network.

7 Conclusion

In this paper, we presented an algorithm for generating a network based on clusters generated using a clustering algorithm on the original network. We used numerical experiments to compare our graph generator to the original network and another popular graph generator, BTER, designed to match the clustering coefficient distribution. We looked at two different measures of community structure, the clustering coefficient and assortative mixing both at the global and local level. We found the our graph generator performed well. While the global clustering coefficient was generally lower for our graph generator than in the original graph, our graph generator was much better able to match the assortativity coefficient and unlike BTER was always able to match the sign of the assortativity coefficient. Thus, building synthetic graphs based on clusters can help preserve the community structure in the network.

One possible area for future research, is to extend the graph generator to bipartite graphs. In a bipartite graph, nodes can either be classified as belonging to class 1 (black) or class 2 (red) where edges only exist between a node of class 1 and a node of class 2. The clustering graph generator should work they same in the bipartite case. However, we will have to ensure that black-black and red-red connections do not occur. In the first step of the algorithm when we use the matching model to create edges between the clusters this just involves adding a check to make sure that when we choose two nodes they do not belong to the same class. If a trial edge does contain two nodes of the same class then we simply do not accept the trial

edge as an actual edge. In the second part of the algorithm where we switch edges within a cluster, again we can check to ensure we are not forming an edge that has nodes from the same class, when we perform a switch. Alternatively, we can list all the edges with nodes of class one (red) first and nodes of class 2 (black) second. Then when we perform any switch we cannot form edges with two nodes of the same class. This would generate bipartite networks with the same community structure as the original network.

References

[1] Réka Albert and Albert-László Barabási. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[2] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[3] Edward A. Bender and E. Rodney Canfield. The asymptotic number of labeled graphs with given degree sequences. *Journal of Combinatorial Theory, Series A*, 24:296–307, 1978.

[4] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008:P10008, 2008.

[5] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Grke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On finding graph clusterings with maximum modularity. In Andreas Brandstdt, Dieter Kratsch, and Haiko Mller, editors, *Graph-Theoretic Concepts in Computer Science*, volume 4769 of *Lecture Notes in Computer Science*, pages 121–132. Springer, Berlin / Heidelberg, 2007.

[6] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. In *Proceedings of the 9th International World Wide Web Conference on Computer Networks : The International Journal of Computer and Telecommunications Networking*, pages 309–320, 2000.

[7] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1), June 2006.

[8] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. In *Fourth SIAM International Conference on Data Mining*, 2004.

[9] Fan Chung and Linyuan Lu. The average distance in random graphs with given expected degrees. *Proceedings of National Academy of Science*, 99:15879–15882, 2002.

[10] Fan Chung and Linyuan Lu. Connected components in a random graph with given degree sequences. *Annals of Combinatorics*, 6:125–145, 2002.

[11] P. Erdős and A. Rényi. On the evolution of random graphs. In *Publication of The Mathematic Institute of The Hungarian Academy of Sciences*, pages 17–61, 1960.

[12] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '99, pages 251–262, 1999.

[13] Graph500. Graph 500 benchmark. `http://www.graph500.org`.

[14] Alexander Gutfraind, Lauren Ancel Meyers, and Ilya Safro. Multiscale network generation. *CoRR*, abs/1207.4266, 2012.

[15] J. M. Roberts Jr. Simple methods for simulating sociomatrices with given marginal totals. *Social Networks*, 22:273–283, 2000.

[16] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: measurements, models, and methods. In *Proceedings of the 5th Annual International Conference on Computing and Combinatorics*, COCOON '99, pages 1–17, 1999.

[17] Tamara G. Kolda, Ali Pinar, et al. FEASTPACK v1.1. `http://www.sandia.gov/~tgkolda/feastpack/`, January 2014.

[18] Tamara G. Kolda, Ali Pinar, Todd Plantenga, and C. Seshadhri. A scalable generative graph model with community structure. *SIAM Journal on Scientific Computing*, 36(5):C424–C452, September 2014.

[19] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 57–65, Washington, DC, USA, 2000. IEEE Computer Society.

[20] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007.

[21] Jurij Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.

[22] Stanley Milgram. The small world problem. *Psychology Today*, pages 60–67, 1967.

[23] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. *Arxiv preprint cond-mat/0312028*, 2004.

[24] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network Motifs: Simple Building Blocks of Complex Networks. *Science*, 298:824–827, 2002.

[25] Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random Structures and Algorithms*, 6:161–179, 1995.

[26] Michael Molloy and Bruce Reed. The size of the giant component of a random graph with a given degree sequence. *Combinatorics, Probababilty and Computing*, 7:295–305, 1998.

[27] M. E. J. Newman. *Networks: An Introduction*. Oxford University Press, 2010.

[28] M. E. J. Newman, S. H. Strogatz, and D. J. Watts. Random graphs with arbitrary degree distributions and their applications. *Physical Review E*, 64:026118, 2001.

[29] Mark E. Newman. Assortative mixing in networks. *Physical Review Letters*, 89:208701, 2002.

[30] Ali Pinar, C. Seshadhri, and Tamara G. Kolda. The similarity between stochastic Kronecker and Chung-Lu graph models. In *SDM12: Proceedings of the Twelfth SIAM International Conference on Data Mining*, pages 1071–1082, April 2012.

[31] M. Piraveenan, M. Prokopenko, and A. Y. Zomaya. Local assortativeness in scale-free networks. *Europhysics Letters*, 84:28002, 2008.

[32] A. Ramachandra Rao, Rabindranath Jana, and Suraj Bandyopadhyay. A markov chain monte carlo method for generating random (0, 1)-matrices with given marginals. *Sankhy: The Indian Journal of Statistics, Series A (1961-2002)*, 58:225–242, 1996.

[33] S. Redner. How Popular is Your Paper? An Empirical Study of the Citation Distribution. *The European Physical Journal B*, 4:131–134, 1998.

[34] C. Seshadhri, Tamara G. Kolda, and Ali Pinar. Community structure and scale-free collections of Erdős-Rényi graphs. *Physical Review E*, 85, May 2012.

[35] SNAP. SNAP: Standford network analysis project. `http://snap.stanford.edu/`.

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