LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-678729

An In-Depth Analysis of the Chung-Lu Model

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.

An In-Depth Analysis of the Chung-Lu Model[1]

Manda Winlaw, Hans De Sterck, and Geoffrey Sanders

August 26, 2015

[1]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.

Chapter 1

An In-Depth Analysis of the Chung-Lu Model

1.1 Introduction

In the classic Erdős Rényi random graph model [5] each edge is chosen with uniform probability and the degree distribution is binomial, limiting the number of graphs that can be modeled using the Erdős Rényi framework [10]. The Chung-Lu model [1, 2, 3] is an extension of the Erdős Rényi model that allows for more general degree distributions. The probability of each edge is no longer uniform and is a function of a user-supplied degree sequence, which by design is the expected degree sequence of the model. This property makes it an easy model to work with theoretically and since the Chung-Lu model is a special case of a random graph model with a given degree sequence, many of its properties are well known and have been studied extensively [2, 3, 13, 8, 9]. It is also an attractive null model for many real-world networks, particularly those with power-law degree distributions and it is sometimes used as a benchmark for comparison with other graph generators despite some of its limitations [12, 11]. We know for example, that the average clustering coefficient is too low relative to most real world networks. As well, measures of affinity are also too low relative to most real-world networks of interest. However, despite these limitations or perhaps because of them, the Chung-Lu model provides a basis for comparing new graph models.

To compare the Chung-Lu model to other graph generators we often use instances of the models we are interested in. However, the standard algorithm used to generate instances of the Chung-Lu model has a runtime that is $\mathcal{O}(n^2)$ where n is the number of nodes in the original graph. Thus, if we are dealing with a very large graph this algorithm can be prohibitively expensive. Miller and Hagberg [7] introduce an algorithm with expected runtime that is $\mathcal{O}(n+m)$ where m is the expected number of edges in any instance of the model. They argue that for graphs with finite average degree $n = \mathcal{O}(m)$ and the expected runtime is $\mathcal{O}(n)$. As part of their Block Two-Level Erdős Rényi graph generator, Kolda et al. [12, 6] introduce a Fast Chung-Lu algorithm with an expected runtime of $\mathcal{O}(m)$ where m, in this case, is the number of edges in the original graph. In [6], the authors note that duplicate edges and self-edges may be created by their algorithm, but that they simply discard these edges, stating that in their experiments the practical impact was small. In this paper, we are interested in examining the impact of discarding

these edges. By removing these duplicate edges and self-edges we find that the model described by the Fast Chung-Lu algorithm is in fact an approximation to the true Chung-Lu model. We are interested in examining the size of the approximation and determining how different factors can affect it. Since the original Chung-Lu model assumes that there are self-edges in the graph we are interested in examining the approximation error as a result of removing self-edges separately from the impact of removing duplicate edges. As well, certain restrictions are placed on the input degree sequence that must hold for the properties of the Chung-Lu model to hold. Often the real-world networks we are interested in studying violate this constraint, however, these degree sequences can still be used as inputs into the Fast Chung-Lu algorithm and we are interested in studying how different the Fast Chung-Lu model and the true Chung-Lu model are in this case. Since the Fast Chung-Lu model is only an approximation to the original model it is important that when we generate instances using the Fast Chung-Lu algorithm we are aware of the degree of approximation especially when we are using these instances to compare different graph generators. Note that Kolda et al. [4], also introduce an algorithm for directed graphs that is based on the Fast Chung-Lu algorithm, however, our discussion will be restricted to undirected, connected graphs although much of the analysis does not require the graph to be connected.

In this paper, we explore the Chung-Lu model in detail and we look at how certain relaxations of the model can lead to new models with different properties from the original Chung-Lu model. We begin our discussion by presenting the basic Chung-Lu model, which we will refer to as the Bernoulli Chung-Lu model for reasons that will become apparent in the next section, and we describe the standard $\mathcal{O}(n^2)$ algorithm that can be used to generate instances of this model. We calculate the expected degree for each node and using the degree sequence from a real world network we use instances of the model to show that for each node, the degree average approaches the expected degree as the number of instances grows, as is predicted by the law of large numbers. In Section 1.3, we present the model described by the Fast Chung-Lu algorithm which we will also refer to as the $\mathcal{O}(|Edges|)$ or $\mathcal{O}(m)$ Chung-Lu model. We begin by allowing self-edges in the model (i.e. we do not remove self-edges in the algorithm, only duplicate edges). As mentioned previously, instances of this model can be generated much more efficiently than the Bernoulli Chung-Lu model however, they only approximate the Bernoulli Chung-Lu model. We calculate the expected degree in this model and compare it to the Bernoulli Chung-Lu model. We also use this model to generate instances of a real network and show that for each node the degree average approaches the expected degree as the number of instances grows and show how the expected degree differs from the expected degree under the Bernoulli Chung-Lu model. In Section 1.4 we explore how removing self-edges from both the Bernoulli Chung-Lu and $\mathcal{O}(|Edges|)$ Chung-Lu models impact the properties of the model. As we mentioned above, there is a constraint placed on the input degree sequence to ensure that the properties of the model hold. In Section 1.5 we explore how the properties of both the Bernoulli Chung-Lu and $\mathcal{O}(|Edges|)$ Chung-Lu models change when this constraint is violated. In Section 1.6 we focus on another network property that is often important when modeling networks, the triangle count and the related clustering coefficient. We calculate the expected triangle count for the various Chung-Lu models we have presented and show how these values are different from the values found in real networks.

1.2 Bernoulli Chung-Lu Model

In the Chung-Lu model, as previously mentioned, each edge is chosen with a different probability. As such, each edge can be represented as a random variable and we can formally define the probability distribution for each edge. Edges between two distinct nodes can be represented as Bernoulli random variables. Written

in terms the adjacency matrix, we have

$$\mathbf{A}_{ij} \sim \text{Bernoulli}(p_{ij}), \;\; 1 \leq i < j \leq n, \; \mathbf{A}_{ji} = \mathbf{A}_{ij}. \tag{1.1}$$

For self-edges, the probability distribution is more general although it is very similar to the Bernoulli distribution. Written in terms of the adjacency matrix, self-edges, $\mathbf{A}_{ii}$, $i = 1, \ldots, n$, can be defined as random variables with support, $S(\mathbf{A}_{ii}) = \{0, 2\}$, and probability mass function given by the following

$$P(\mathbf{A}_{ii} = a_{ii}) = \begin{cases} 1 - p_{ii} & a_{ii} = 0, \\ p_{ii} & a_{ii} = 2, \end{cases} \tag{1.2}$$

where $i = 1, \ldots, n$. Why is the support of each self-edge equal to $\{0, 2\}$ and not $\{0, 1\}$? Essentially, this is to ensure that all edges are counted equally. In general, if there is an edge in an undirected graph between two distinct nodes i and j then both a_{ij} and a_{ji} are 1. If edges are to be counted equally, then self-edges should also appear in the adjacency matrix twice but since there is only one diagonal matrix element, a_{ii}, both appearances need to be recorded in the same element. Hence, $a_{ii} = 2$ if node i has a self-edge and thus the support of the random variable $\mathbf{A}_{ii}$ is $\{0, 2\}$.

We have not yet defined the probabilities, $p_{ij}, \;\; i, j = 1, \ldots, n$, in the probability distributions, (1.1) and (1.2). In the Chung-Lu model they have a very specific form and are based on the only input into the Chung-Lu model, a sequence of degrees, $k = (k_1, \ldots, k_n)$, where k_i is the degree of node i and n is the number of nodes. Often, this degree sequence is taken from a real network we are interested in studying. In the Chung-Lu model, $p_{ij}, \;\; i, j = 1, \ldots, n$, has the following form

$$p_{ij} = \begin{cases} \frac{k_i k_j}{2m} & i \neq j, \\ \frac{k_i^2}{4m} & i = j, \end{cases} \tag{1.3}$$

where $m = \frac{1}{2} \sum_i k_i$ is the number of edges in the graph and we assume $k_i^2 \leq 2m \;\forall\; i$ which is sufficient condition to ensure that $p_{ij} \leq 1 \;\forall\; i, j$.

One of the central properties of the Chung-Lu model is that the expected degree sequence is actually given by the input degree sequence, k. In other words, the expected degree of node i is equal to k_i. To show this we note that in the model, the degree of node i is a random variable $\mathbf{D}_i$, $i = 1, \ldots, n$, given by

$$\mathbf{D}_i = \sum_j \mathbf{A}_{ij}, \;\; i = 1, \ldots, n. \tag{1.4}$$

Since the degree of node i, $\mathbf{D}_i$, is a random variable defined as the sum of random variables its expected value will be equal to the sum of the expected values of its components. More formally, we have

$$E(\mathbf{D}_i) = E\left(\sum_j \mathbf{A}_{ij} \right) = \sum_j E(\mathbf{A}_{ij}). \tag{1.5}$$

For distinct edges, $\mathbf{A}_{ij}$ is a Bernoulli random variable and thus its expected value is given by

$$E(\mathbf{A}_{ij}) = p_{ij} = \frac{k_i k_j}{2m}, \;\; i \neq j. \tag{1.6}$$

For self-edges we can use the probability mass function given in (1.2) to determine the expected value

$$\begin{aligned} E(\mathbf{A}_{ii}) &= \sum a_{ii} P(\mathbf{A}_{ii} = a_{ii}) \\ &= 0 \cdot P(\mathbf{A}_{ii} = 0) + 2 \cdot P(\mathbf{A}_{ii} = 2) \\ &= 2p_{ii} = \frac{k_i^2}{2m}. \end{aligned} \tag{1.7}$$

Thus, the expected degree of node i is given by

$$\begin{aligned} E(\mathbf{D}_i) &= \sum_j E(\mathbf{A}_{ij}) = \sum_{j \neq i} E(\mathbf{A}_{ij}) + E(\mathbf{A}_{ii}) \\ &= \sum_{j \neq i} \frac{k_i k_j}{2m} + \frac{k_i^2}{2m} \\ &= \sum_j \frac{k_i k_j}{2m} = \frac{k_i}{2m} \sum_j k_j = k_i \frac{2m}{2m} \\ &= k_i. \end{aligned} \tag{1.8}$$

This is the central property of the Chung-Lu model. Since $\mathbf{D}_i$ is a random variable we can also define its variance,

$$\mathrm{Var}(\mathbf{D}_i) = \mathrm{Var}\left(\sum_j \mathbf{A}_{ij} \right) = \sum_j \mathrm{Var}(\mathbf{A}_{ij}). \tag{1.9}$$

where we can write $\mathrm{Var}\left(\sum_j \mathbf{A}_{ij}\right) = \sum_j \mathrm{Var}(\mathbf{A}_{ij})$ because the random variables $\mathbf{A}_{ij}$, $i, j = 1, \ldots, n$, are independent. As mentioned previously, for distinct edges, $\mathbf{A}_{ij}$ is a Bernoulli random variable and thus its variance is given by

$$\mathrm{Var}(\mathbf{A}_{ij}) = p_{ij}(1 - p_{ij}) = \frac{k_i k_j}{2m} \left(1 - \frac{k_i k_j}{2m} \right), \; i \neq j. \tag{1.10}$$

For self-edges we can use the probability mass function given in (1.2) to determine the variance

$$\begin{aligned} \mathrm{Var}(\mathbf{A}_{ii}) &= \sum (a_{ii} - E(\mathbf{A}_{ii}))^2 P(\mathbf{A}_{ii} = a_{ii}) \\ &= (0 - 2p_{ii})^2 \cdot P(\mathbf{A}_{ii} = 0) + (2 - 2p_{ii})^2 \cdot P(\mathbf{A}_{ii} = 2) \\ &= 4p_{ii}^2(1 - p_{ii}) + (2 - 2p_{ii})^2 p_{ii} \\ &= 4p_{ii}^2(1 - p_{ii}) + 4(1 - p_{ii})^2 p_{ii} \\ &= 4(1 - p_{ii})(p_{ii}^2 + p_{ii} - p_{ii}^2) \\ &= 4(1 - p_{ii}) p_{ii} \\ &= \frac{k_i^2}{m} \left(1 - \frac{k_i^2}{4m} \right). \end{aligned} \tag{1.11}$$

Thus, the variance of the degree of node i is given by

$$\begin{aligned}\text{Var}(\mathbf{D}_i) &= \sum_j \text{Var}(\mathbf{A}_{ij}) = \sum_{j\neq i} \text{Var}(\mathbf{A}_{ij}) + \text{Var}(\mathbf{A}_{ii}) \\ &= \sum_{j\neq i} \frac{k_i k_j}{2m}\left(1 - \frac{k_i k_j}{2m}\right) + \frac{k_i^2}{m}\left(1 - \frac{k_i^2}{4m}\right).\end{aligned} \tag{1.12}$$

Although the variance of $\mathbf{D}_i$ does not simplify nicely it is still useful in certain error bound calculations described below.

In practice, model properties are often examined by generating instances of the model. In any given instance of the Bernoulli Chung-Lu model the node degree will not match the expected degree; however, by the law of large numbers as we increase the number of instances the average degree for each node should approach the expected degree for each node. We test this theory using a small real network. Consider the arXiv general relativity collaboration network which is an author collaboration network based on data from the general relativity section of the arXiv preprint server and has 4158 nodes and 13422 edges. The largest degree node has 81 neighbors. Thus, $k_i^2 \leq 2m \; \forall \; i$, since the maximum value of k_i^2 is 6561 and $2m$ is equal to 26844. If we use the Chung-Lu model to generate a collection of synthetic graphs then the average degree for each node should approach the expected degree as the collection size increases. Algorithm 1 describes an algorithm that can be used to generate instances of the Chung-Lu model. Given that we will present several different versions of the Chung-Lu model throughout this paper, we need to provide different names to each version. We refer to the Chung-Lu model presented in this section as both the Bernoulli Chung-Lu model, a name based on the edge probability distributions and Model I. We will also often refer to this model as the original Chung-Lu model since it was the model originally proposed by Chung and Lu [2, 3]. Using the degree sequence from the general relativity collaboration network to define the probabilities, p_{ij}, $i, j = 1, \ldots, n$ in Equation (1.3) we use Algorithm 1 to generate 10000 instances of the Chung-Lu model. Figure 1.1 plots the difference between the expected degree of each node and the average degree of each node where we have ordered the nodes from smallest degree to largest degree based on their degree size in the original graph. The expected degree of each node is of course equal to the actual degree of each node in the original graph as shown by Equation (1.8). The average degree of each node is calculated using l instances of the Chung-Lu model where we vary l from 10 to 10000. We can see from the plots that as we increase the number of instances in our average degree calculation the difference between the average and the expected degree decreases where the largest difference between the two values is seen in large degree nodes. Also note that the total number of possible graphs is $2^{\frac{n(n+1)}{2}} \approx 2^{8 \text{ million}}$, so we are generating only a small fraction of the total number of graphs for our average degree calculations, however we are still able to obtain fairly accurate approximations to the expected degree. In fact, we can use the law of large numbers and Chebyshev's inequality to put a bound on this difference. For any node i, let $\mathbf{D}_i^{(j)}$ be the random degree variable from instance j, $j = 1, \ldots, l$. Then $\mathbf{D}_i^{(1)}, \mathbf{D}_i^{(2)}, \ldots, \mathbf{D}_i^{(l)}$ are independent, identically, distributed random variables with mean $E(\mathbf{D}_i^{(j)}) = k_i$ and $\text{Var}(\mathbf{D}_i^{(j)}) = \sigma^2$, $j = 1, \ldots, l$ where σ^2 is given by Equation (1.12). If we define the sample average as

$$\overline{\mathbf{D}}_i = \frac{\mathbf{D}_i^{(1)} + \mathbf{D}_i^{(2)} + \cdots + \mathbf{D}_i^{(l)}}{l}, \tag{1.13}$$

then using Chebyshev's inequality on $\overline{\mathbf{D}}_i$ we get

$$P(|\overline{\mathbf{D}}_i - k_i| \geq \epsilon) \leq \frac{\sigma^2}{l\epsilon^2} \tag{1.14}$$

or

$$P(|\overline{\mathbf{D}}_i - k_i| < \epsilon) \geq 1 - \frac{\sigma^2}{l\epsilon^2} \tag{1.15}$$

So the difference between the sample average and the expected value is a function of ϵ, the number of instances, l, and the variance of $\mathbf{D}_i$. In the general relativity graph, we noted that the difference between the average degree and the expected degree was largest for large degree nodes. For the largest degree node if we let $\epsilon = 0.2$ then

$$P(|\overline{\mathbf{D}}_i - k_i| < 0.2) \geq 1 - \frac{\sigma^2}{l\epsilon^2} = 1 - \frac{1921.24}{l}, \tag{1.16}$$

which implies that l has to be at least 1922 to get this probability to be less than 1 and if $l = 10000$ then this probability is approximately 0.808.

Additionally, we can also compare the expected number of edges in the graph with the average number of edges, calculated from l instances of the model. Like the degree of node i, $\mathbf{D}_i$, we can define the random variable $\mathbf{M}$ as the number of edges in the model where $\mathbf{M}$ is defined as

$$\mathbf{M} = \frac{1}{2}\sum_i \mathbf{D}_i = \frac{1}{2}\sum_i \sum_j \mathbf{A}_{ij} \tag{1.17}$$

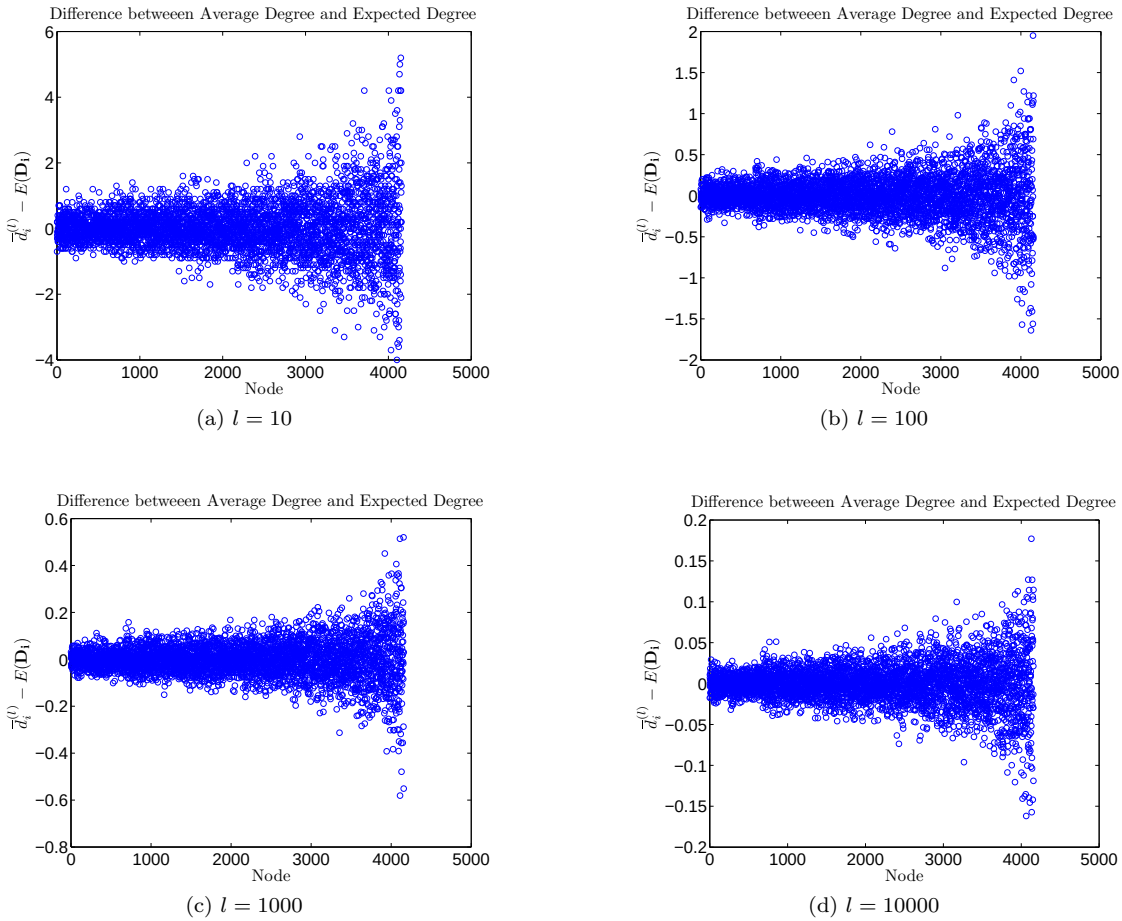
Then the expected value of $\mathbf{M}$ can be calculated as follows

$$\begin{aligned} E(\mathbf{M}) &= \frac{1}{2}\sum_i E(\mathbf{D}_i) \\ &= \frac{1}{2}\sum_i k_i \\ &= \frac{1}{2}(2m) \\ &= m. \end{aligned} \tag{1.18}$$

For $l = 10000$ the average number of edges is 13421.12. Compare this to 13422 which is the expected number of edges in the graph (and the actual number of edges in the original graph). Table 1.1 lists the average number of edges for $l = 10, 100, 1000, 5000$ and 10000.

Algorithm 1: Bernoulli Chung-Lu Algorithm

```
for i = 1 to n do
    for j = i to n do
        Draw a random number from the uniform distribution on [0,1];
        if The random number is less than or equal to p_ij then
            /* Add edge (i, j) to the graph */
            if i ≠ j then
                a_ij = a_ji = 1;
            else
                a_ii = 2;
            end
        end
    end
end
```



(a) $l = 10$ (b) $l = 100$

(c) $l = 1000$ (d) $l = 10000$

Figure 1.1: General Relativity Collaboration Network: Comparing Average Degree and Expected Degree from Model I.

1.3 $\mathcal{O}(|Edges|)$ Chung-Lu Model

One of the issues with the Bernoulli Chung-Lu model and its implementation using Alogirthm 1 is that Algorithm 1 requires $\mathcal{O}(n^2)$ operations. We must visit every edge and decide with probability p_{ij} if the edge should be included in the graph. This algorithm can become prohibitively expensive as the size of the graphs we consider grows. However, we can devise an algorithm where the probability that an edge will appear in the graph is approximately the same as under the previous model and that requires only $\mathcal{O}(m)$ operations where $m = \frac{1}{2}\sum_i k_i$. Suppose we place all the elements of the adjacency matrix a_{ij}, $i, j = 1, \ldots, n$ in a bag. We are no longer treating the elements as random variables so we write the elements as a_{ij} instead of $\mathbf{A}_{ij}$. For now we will assume that the elements representing self-edges, a_{ii}, $i = 1, \ldots, n$, are in the bag. We create our graph by drawing m elements of the adjacency matrix from the bag. After each draw we return the elements to the bag so that we allow elements to be drawn more

Table 1.1: Average Number of Edges for l Instances of the Bernoulli Chung-Lu Model

Number of Instances (l)	10	100	1000	5000	10000
Average Number of Edges	13442.5	13420.31	13421.65	13421.21	13421.12

than once. If the element a_{ij} is drawn, a_{ij} and a_{ji} are both set equal to 1. If an element has already been drawn the values of a_{ij} and a_{ji} remain equal to 1. If the element a_{ii} is drawn then $a_{ii} = 2$ and this value doesn't change if a_{ii} is drawn subsequently. We begin by considering the case where $i \neq j$. We want to know what the probability of an edge between vertices i and j is after m draws. In other words, we want to know the probability of having an edge between vertices i and j is in our graph. This is not simply equal to the probability of drawing a_{ij} or a_{ji} on a single draw. To calculate the probability of having an edge between vertices i and j in the graph we note that an edge between nodes i and j appears in the graph because either a_{ij} or a_{ji} is drawn at least once. More formally, let us define three mutually exclusive and exhaustive events that can occur on a single draw, $\mathbf{E}_1 = \{a_{ij} \text{ is drawn}\}$, $\mathbf{E}_2 = \{a_{ji} \text{ is drawn}\}$, and $\mathbf{E}_3 = \{\text{Another element of the adjacency matrix is drawn}\}$. Let p_{ij} be the probability that event $\mathbf{E}_1$ happens, p_{ji} be the probability that event $\mathbf{E}_2$ happens and $1 - p_{ij} - p_{ji}$ be the probability that event $\mathbf{E}_3$ happens. On m independent draws let $\mathbf{X}_i$ be the number of occurrences of event $\mathbf{E}_i$, $i = 1, 2, 3$. Then the vector $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2)$ has a multinomial distribution with joint pdf given by

$$f(x_1, x_2) = \frac{m!}{x_1!x_2!x_3!} p_1^{x_1} p_2^{x_2} p_3^{x_3} = \frac{m!}{x_1!x_2!x_3!} p_{ij}^{x_1} p_{ji}^{x_2} (1 - p_{ij} - p_{ji})^{x_3}, \tag{1.19}$$

where $x_3 = m - x_1 - x_2$ and $p_3 = 1 - p_1 - p_2$. In fact, we could define the vector $\widehat{\mathbf{X}} = (\mathbf{X}_{11}, \mathbf{X}_{12}, \ldots, \mathbf{X}_{ij}, \ldots, \mathbf{X}_{n(n-1)})$ where $\mathbf{X}_{ij}$ is the number of occurrences of event $\mathbf{E}_{ij} = \{a_{ij} \text{ is drawn}\}$, $i, j = 1, \ldots, n$, on m independent draws. Then $\widehat{\mathbf{X}}$ has a multinomial distribution with joint pdf given by

$$f(x_{11}, x_{12}, \ldots, x_{ij}, \ldots, x_{n(n-1)}) = \frac{m!}{x_{11}!x_{12}! \cdots x_{ij}! \cdots x_{n(n-1)}!x_{nn}!} p_{11}^{x_{11}} p_{12}^{x_{12}} \cdots p_{ij}^{x_{ij}} \cdots p_{n(n-1)}^{x_{n(n-1)}} p_{nn}^{x_{nn}} \tag{1.20}$$

where $\sum_i \sum_j p_{ij} = 1$. The vector $\mathbf{X}$ is simply a condensed version of $\widehat{\mathbf{X}}$ where we have simply condensed the $n^2 - 2$ events where we draw an adjacency matrix element that is not a_{ij} or a_{ji} into one event. As noted $\sum_i \sum_j p_{ij} = 1$. On a single draw, drawing one element of the adjacency matrix precludes the drawing of any other element. In this sense, there is some dependence amongst the edges in the graph. Returning to our original goal of calculating the probability of an edge between nodes i and j in the graph, we can use the joint pdf in (1.19) to calculate this value. We want to know the probability that either $\mathbf{X}_1$ or $\mathbf{X}_2$ is at least 1. This is equivalent to one minus the probability that both are zero. Written formally we have

$$\begin{aligned} p_{ij}(m) &:= P(\text{There is an edge between nodes } i \text{ and } j \text{ in the graph after } m \text{ draws}) \\ &= 1 - f(\mathbf{X}_1 = 0, \mathbf{X}_2 = 0) \\ &= 1 - (1 - p_{ij} - p_{ji})^m, \ i \neq j. \end{aligned} \tag{1.21}$$

If we assume that $p_{ij} = p_{ji}$ we can use the binomial theorem to simplify the formula for $p_{ij}(m)$:

$$(1 - 2p_{ij})^m = \sum_{k=0}^{m} \binom{m}{k} (-1)^k (2p_{ij})^k. \tag{1.22}$$

This give us

$$\begin{aligned} p_{ij}(m) &= 1-(1-2p_{ij})^m \\ &= 1-\sum_{k=0}^{m}\binom{m}{k}(-1)^k(2p_{ij})^k \\ &= \sum_{k=1}^{m}\binom{m}{k}(-1)^{k+1}(2p_{ij})^k \\ &= 2mp_{ij}+\mathcal{O}(p_{ij}^2),\ i\neq j. \end{aligned} \tag{1.23}$$

Returning to the case where $i=j$, we note that a self-edge appears in the graph at node i if A_{ii} is drawn at least once. So to calculate the probability of a self-edge appearing in the graph, we define two mutually exclusive and exhaustive events that can occur on a single draw, $\mathbf{E}=\{a_{ii} \text{ is drawn}\}$, and $\overline{\mathbf{E}}=\{\text{Another element of the adjacency matrix is drawn}\}$. Let p_{ii} be the probability that event $\mathbf{E}$ happens, and $1-p_{ii}$ be the probability that event $\overline{\mathbf{E}}$ happens. On m independent draws let $\mathbf{X}$ be the number of occurrences of event $\mathbf{E}$. Then the vector $\mathbf{X}$ has a binomial distribution with probability distribution function given by

$$f(x)=\binom{m}{x}p_{ii}^x(1-p_{ii})^{m-x} \tag{1.24}$$

We want to know the probability that the value of $\mathbf{X}$ is greater than or equal to 1. This is equal to $1-P(\mathbf{X}=0)$. Using the binomial distribution in Equation (1.24) we get

$$\begin{aligned} p_{ii}(m) &:= P(\text{There is a self-edge at node } i \text{ in the graph after } m \text{ draws}) \\ &= 1-f(\mathbf{X}=0) \\ &= 1-(1-p_{ii})^m. \end{aligned} \tag{1.25}$$

Using the binomial theorem to simplify we get

$$\begin{aligned} p_{ii}(m) &= 1-(1-p_{ii})^m \\ &= 1-\sum_{k=0}^{m}\binom{m}{k}(-1)^k(p_{ii})^k \\ &= \sum_{k=1}^{m}\binom{m}{k}(-1)^{k+1}(p_{ii})^k \\ &= mp_{ii}+\mathcal{O}(p_{ii}^2). \end{aligned} \tag{1.26}$$

We can combine the two cases to get the probability of edge (i,j) being in the graph after m draws as

$$p_{ij}(m)=\begin{cases} 1-(1-2p_{ij})^m \approx 2mp_{ij} & i\neq j, \\ 1-(1-p_{ij})^m \approx mp_{ii} & i=j, \end{cases} \tag{1.27}$$

where we have assumed that $p_{ij}=p_{ji}$. Suppose we assume that $p_{ij}=\frac{k_ik_j}{(2m)^2}\ \forall\ i,j$ so that $\sum_i\sum_j p_{ij}=1$ then

$$p_{ij}(m)=\begin{cases} 1-(1-2\frac{k_ik_j}{(2m)^2})^m=\frac{k_ik_j}{2m}+\mathcal{O}(p_{ij}^2) & i\neq j, \\ 1-(1-\frac{k_i^2}{(2m)^2})^m=\frac{k_i^2}{4m}+\mathcal{O}(p_{ii}^2) & i=j. \end{cases} \tag{1.28}$$

If we ignore the higher order terms then we get

$$p_{ij}(m) \approx \begin{cases} \frac{k_i k_j}{2m} & i \neq j, \\ \frac{k_i^2}{4m} & i = j. \end{cases} \tag{1.29}$$

Thus, the probability of having an edge between nodes i and j in the graph is approximately equal to the value given by Equation (1.3), the probability in our original $\mathcal{O}(n^2)$ algorithm. The following algorithm outlines the above graph generator.

Algorithm 2: Approximate Chung-Lu Algorithm

for $k = 1$ *to* m **do**
  Draw element a_{ij} with probability $\frac{k_i k_j}{(2m)^2}$;
  `/* Add edge` (i, j) `to the graph */`
  if $i \neq j$ **then**
    $a_{ij} = a_{ji} = 1$;
  else
    $a_{ii} = 2$;
  end
end

The way that Algorithm 2 is written requires us to perform $\mathcal{O}(n^2)$ calculations to determine the probabilities of drawing each edge on a single draw (i.e. the probability of drawing a_{ij}), so in fact this algorithm is still $\mathcal{O}(n^2)$. However, suppose on each draw, instead of drawing the edges themselves we independently draw the nodes of each edge. So, on a single draw, we draw node i with probability $\frac{k_i}{2m}$ and we draw node j with probability $\frac{k_j}{2m}$. Since the two events are independent, the probability of drawing the pair of nodes (i, j), or edge (i, j), is equal to $\frac{k_i k_j}{(2m)^2}$. This only requires us to calculate n probabilities and since $m \geq n - 1$ in any connected graph, our algorithm is now $\mathcal{O}(m)$. This algorithm is described below and is the $\mathcal{O}(m)$ algorithm we use to generate instances of the $\mathcal{O}(m)$ Chung-Lu model. It is the algorithm referred to in [12, 6] known as the Fast Chung-Lu algorithm. Note, we refer to the model described by Algorithm 3 as the $\mathcal{O}(m)$ or $\mathcal{O}(|\text{Edges}|)$ Chung-Lu model and as Model II.

Algorithm 3: $\mathcal{O}(|\text{Edges}|)$ Chung-Lu Algorithm

for $k = 1$ *to* m **do**
  Draw node i with probability $\frac{k_i}{2m}$;
  Draw node j with probability $\frac{k_j}{2m}$;
  `/* Add edge` (i, j) `to the graph */`
  if $i \neq j$ **then**
    $a_{ij} = a_{ji} = 1$;
  else
    $a_{ii} = 2$;
  end
end

Note that we have defined $p_{ij}(m)$ in the $\mathcal{O}(m)$ Chung-Lu model as the probability of edge (i, j) being in the graph. Equivalently, $p_{ij}(m)$ can be defined as the probability of choosing edge (i, j) to be in the

graph. Given this interpretation, we can then represent each edge as a random variable like we did in the Bernoulli Chung-Lu model. Edges can again be represented as elements of the adjacency matrix where edges between distinct nodes follow a Bernoulli distribution as in (1.1) and self-edges are random variables with the probability mass function given by (1.2). The difference between the Bernoulli Chung-Lu model and the $\mathcal{O}(m)$ Chung-Lu algorithm is in the probability p_{ij}. In the Bernoulli Chung-Lu model p_{ij} is defined in Equation (1.3) and in the $\mathcal{O}(m)$ Chung-Lu algorithm p_{ij} is defined as $p_{ij}(m)$, Equation (1.28). In other words, we could use Algorithm 1 with $p_{ij}(m)$ given by the exact values in Equation (1.28) in place of p_{ij} and this would be equivalent to Algorithm 3 (i.e. the underlying model is the same). We can use this equivalence to calculate the expected degree of each node in the $\mathcal{O}(m)$ Chung-Lu model. As noted in the discussion of the Bernoulli Chung-Lu model, since the adjacency matrix elements are random variables so too is the degree of each node and the expected degree of each node is given by

$$E(\mathbf{D}_i) = E\left(\sum_j \mathbf{A}_{ij}\right) = \sum_j E(\mathbf{A}_{ij}). \tag{1.30}$$

For the $\mathcal{O}(m)$ Chung-Lu model, $E(\mathbf{A}_{ij}) = p_{ij}(m)$ if $i \neq j$ and $E(\mathbf{A}_{ii}) = 2p_{ii}(m)$. If we use the approximation in Equation (1.29) for $p_{ij}(m)$ then $E(\mathbf{D}_i) = k_i$. However if we include the higher order terms in $p_{ij}(m)$ we get the following

$$\begin{aligned} E(\mathbf{D}_i) &= E\left(\sum_j \mathbf{A}_{ij}\right) = \sum_j E(\mathbf{A}_{ij}) \\ &= \sum_{j\neq i} p_{ij}(m) + 2p_{ii}(m) \\ &= \sum_{j\neq i} 1 - (1-2p_{ij})^m + 2(1-(1-p_{ii})^m) \\ &= n + 1 - \sum_{j\neq i}(1 - \frac{k_i k_j}{2m^2})^m - 2(1-\frac{k_i^2}{4m^2})^m, \end{aligned} \tag{1.31}$$

which should be approximately equal to k_i. We should note however that the expected degree in Equation (1.31) will always be less than k_i. To see this, we examine the probabilities in Equation (1.28):

$$p_{ij}(m) = \begin{cases} 1 - (1 - 2\frac{k_i k_j}{(2m)^2})^m = \frac{k_i k_j}{2m} + \sum_{l=2}^m \binom{m}{l}(-1)^{l+1}(\frac{k_i k_j}{2m^2})^l & i \neq j, \\ 1 - (1 - \frac{k_i^2}{(2m)^2})^m = \frac{k_i^2}{4m} + \sum_{l=2}^m \binom{m}{l}(-1)^{l+1}(\frac{k_i^2}{4m^2})^l & i = j. \end{cases} \tag{1.32}$$

Since $\frac{k_i k_j}{2m^2} < 1$ this implies that alternating series, $\sum_{l=2}^m \binom{m}{l}(-1)^{l+1}(\frac{k_i k_j}{2m^2})^l$ is monotonically decreasing and since the initial term is negative, this implies that the series will be negative. This implies that the maximum value for $p_{ij}(m)$ is strictly less than $\frac{k_i k_j}{2m}$ for $i \neq j$. For self-edges, since $\frac{k_i^2}{4m^2} < 1$ this implies that alternating series, $\sum_{l=2}^m \binom{m}{l}(-1)^{l+1}(\frac{k_i^2}{4m^2})^l$ is monotonically decreasing and since the initial term is negative, this implies that the series will be negative. This implies that the maximum value for $p_{ii}(m)$ is strictly less than $\frac{k_i^2}{4m}$ for $i \neq j$. Given that $E(\mathbf{A}_{ij}) = p_{ij}(m)$ for $i \neq j$ and $E(\mathbf{A}_{ii}) = 2p_{ii}(m)$ this implies that $E(\mathbf{D}_i) < k_i$.

We can once again explore the performance of our model using the general relativity collaboration network. Using Algorithm 3 to generate $l = 10000$ instances of the graph we can examine the difference between the average degree for each node and the expected degree. Figure 1.2 plots the difference between the average degree and the expected degree, given by Equation (1.31), for $l = 10, 100, 1000$ and 10000. We can see from the figures that as the number of instances of the graph increase the difference between the average and the expected degree decreases. We can once again bound this difference using Chebyshev's inequality. To do this we need the variance of $\mathbf{D}_i$ which is given by

$$\begin{aligned}\text{Var}(\mathbf{D}_i) &= \sum_j \text{Var}(\mathbf{A}_{ij}) = \sum_{j\neq i} \text{Var}(\mathbf{A}_{ij}) + \text{Var}(\mathbf{A}_{ii}) \\ &= \sum_{j\neq i} p_{ij}(m)\left(1 - p_{ij}(m)\right) + 4p_{ii}(m)\left(1 - p_{ii}(m)\right).\end{aligned} \tag{1.33}$$

where $p_{ij}(m)$ and $p_{ii}(m)$ are defined in Equation (1.32). There is no guarantee that this value will be less than or greater than the variance of $\mathbf{D}_i$ given by Equation (1.12). To see this, we note that $\text{Var}(\mathbf{A}_{ij}) = p_{ij}(m)\left(1 - p_{ij}(m)\right)$. We know that $p_{ij}(m) < \frac{k_i k_j}{2m}$, however,in some cases this will imply that $p_{ij}(m)left(1 - p_{ij}(m) > \frac{k_i k_j}{2m}\left(1 - \frac{k_i k_j}{2m}\right)$ and in other cases it implies that $p_{ij}(m)left(1 - p_{ij}(m) < \frac{k_i k_j}{2m}\left(1 - \frac{k_i k_j}{2m}\right)$ since $f(x) = x(1-x)$, $x \in [0, 1]$ is an upside-down parabola centered at 0.5. In the case of the general relativity graph, for the largest degree node we have

$$P(|\overline{\mathbf{D}}_i - k_i| < 0.2) \geq 1 - \frac{\sigma^2}{l\epsilon^2} = 1 - \frac{1921.24}{l}, \tag{1.34}$$

and if $l = 10000$ this probability is approximately 0.811 which is slightly higher than in the Bernoulli Chung-Lu model case.

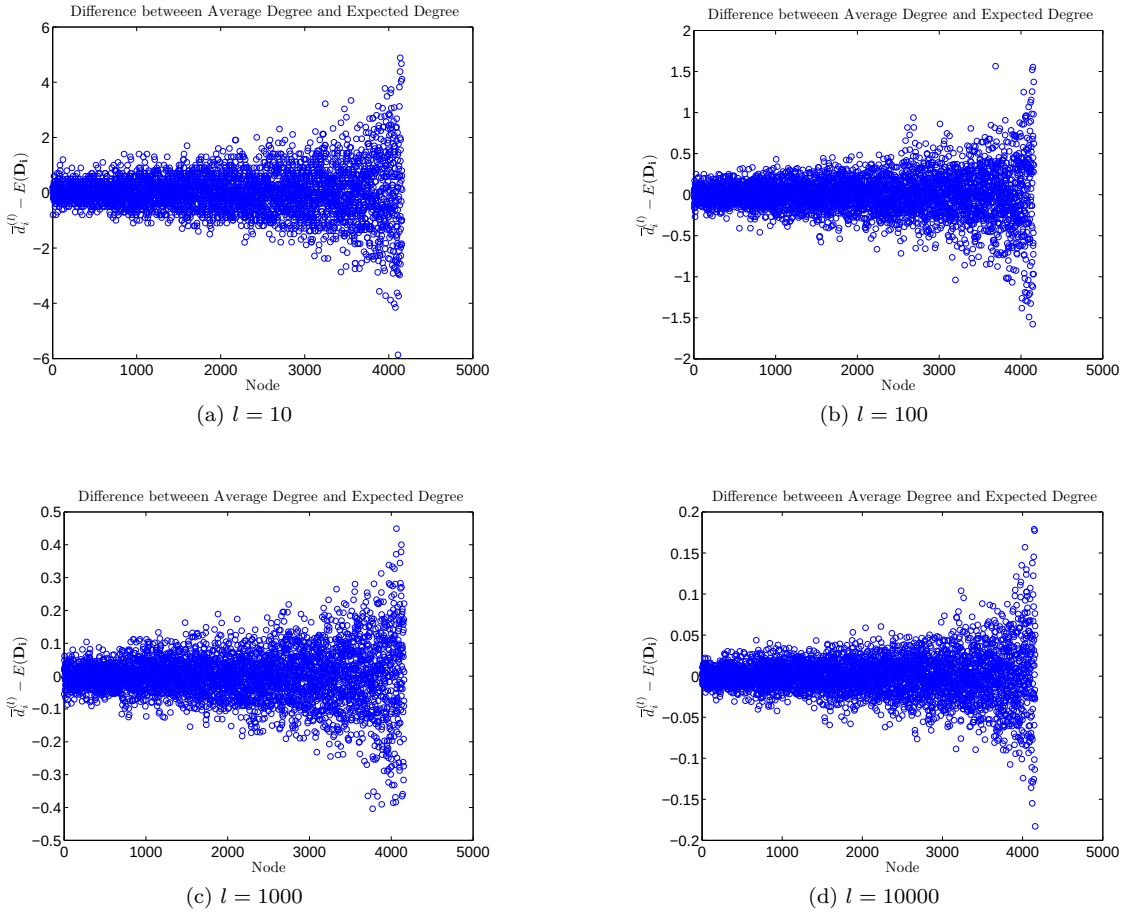
We can also use the general relativity collaboration network to explore the difference between the expected degree given by Equation (1.31) and the the expected degree under the Bernoulli Chung-Lu model, which is k_i, $\forall\, i$. Figure 1.3 plots the difference between the expected degree and the actual degree.

1.4 Excluding Self-Edges

1.4.1 Bernoulli Chung-Lu

Most of the real-world networks we are interested in studying do not have self-edges. So we would like to build models that do not include self-edges. This often involves taking existing models and modifying them to exclude self-edges. To do this in the Bernoulli Chung-Lu model is fairly straight forward. We still represent each edge as a random variable, where distinct node edges have a Bernoulli distribution and self-edges have the probability mass function given in (1.2). However, to remove the possibility of self-edges we simply set the probability of choosing a self-edge to 0. More formally, we re-define the probabilities in (1.3) as

$$p_{ij} = \begin{cases} \frac{k_i, k_j}{2m} & i \neq j, \\ 0 & i = j, \end{cases} \tag{1.35}$$



(a) $l = 10$ (b) $l = 100$

(c) $l = 1000$ (d) $l = 10000$

Figure 1.2: General Relativity Collaboration Network: Comparing Average Degree and Expected Degree from Model II.

where we still assume $k_i^2 \leq 2m \; \forall \; i$ to ensure $p_{ij} \leq 1 \; \forall \; i, j$. This is, of course, a sufficient but not necessary condition to ensure that the probabilities are well defined. Note that by removing self-edges from the graph we no longer have the nice property that $E(\mathbf{D}_i) = k_i$. The expected degree is now given by the following

$$\begin{aligned} E(\mathbf{D}_i) &= E\left(\sum_j \mathbf{A}_{ij}\right) = \sum_j E(\mathbf{A}_{ij}) = \sum_{j \neq i} \frac{k_i k_j}{2m} \\ &= \sum_{j \neq i} \frac{k_i k_j}{2m} = \frac{k_i}{2m} \sum_{j \neq i} k_j = k_i \frac{2m - k_i}{2m} \\ &= k_i - \frac{k_i^2}{2m}, \end{aligned} \tag{1.36}$$

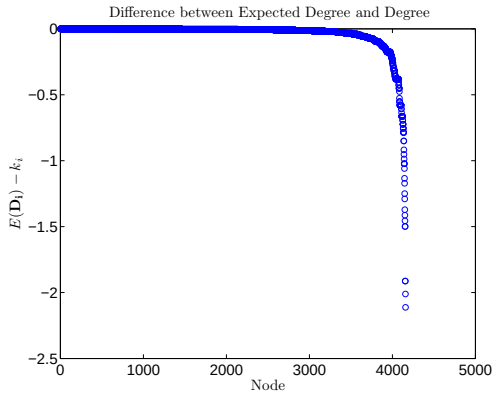


Figure 1.3: General Relativity Collaboration Network: Comparing Expected Degree from Model II and Actual Degree.

where $E(\mathbf{A}_{ii}) = 0 \cdot 1 + 2 \cdot 0 = 0,\ i = 1, \ldots, n$. We have assumed that $\frac{k_i^2}{2m} \leq 1$ which means that in most cases $E(\mathbf{D}_i) \approx k_i$. We use the general relativity collaboration network to illustrate the difference between the original Bernoulli Chung-Lu model and the version of the model with no self-edges which we label Model III. Figure 1.4 plots the difference between the expected degree, Equation (1.36) and the degree in the original graph. Note that the difference is equal to $E(\mathbf{D}_i) - k_i = -\frac{k_i^2}{2m}$. From Figure 1.4 we can see that all the values are less than one in magnitude as expected.

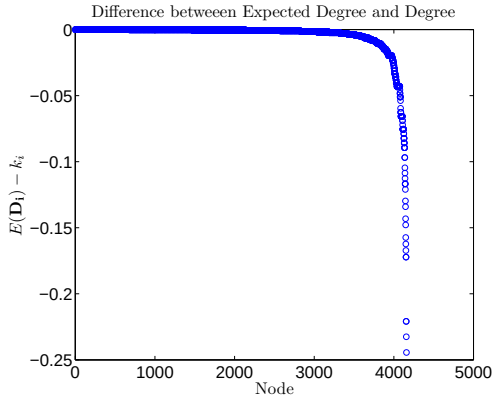


Figure 1.4: General Relativity Collaboration Network: Comparing Expected Degree from Model III and Actual Degree.

1.4.2 $\mathcal{O}(m)$ Chung-Lu

As we previously mentioned, the Bernoulli Chung-Lu algorithm is computationally expensive. Even if we remove self-edges from the model the algorithm is still $\mathcal{O}(n^2)$. In the case with self-edges, the $\mathcal{O}(m)$

Chung-Lu algorithm proved an attractive alternative since it is an approximation to the Bernoulli Chung-Lu model but it is much less computationally expensive. We would like to find a similar algorithm for the case without self-edges, this involves modifying Algorithm 3. We could write an algorithm based on drawing elements of the adjacency matrix from a bag that doesn't include the self-edges (i.e. We remove the a_{ii} elements to begin with and only draw the a_{ij} elements). This would modify p_{ij}, the probability of drawing edge (i, j) on a single draw, since $\sum_{i,j} p_{ij}$ must equal 1 and now $p_{ii} = 0 \; \forall \; i$. This would of course modify $p_{ij}(m)$, the probability of edge (i, j) being in the graph after m draws. However, what makes the $\mathcal{O}(m)$ algorithm so attractive is the ability to draw the nodes of an edge independently as opposed to drawing the edge itself. This is what makes the algorithm $\mathcal{O}(m)$ as opposed to $\mathcal{O}(n^2)$. Once we require that self-edges not be included in the graph then we can no longer draw edge nodes independently since once we draw the first node the probability of the second node changes (i.e. the probability of drawing node i if it has already been drawn is 0). So this rules out the idea of building a $\mathcal{O}(m)$ algorithm by drawing edges from a bag where the self-edges are simply not included. However, a much simpler alternative exists. Returning to Algorithm 3, we can exclude self-edges by simply doing nothing if $i = j$ (i.e. a_{ii} remains equal to 0). This new algorithm is given by Algorithm 4.

Algorithm 4: $\mathcal{O}(|\text{Edges}|)$ Chung-Lu Algorithm without Self-Edges.

for $k = 1$ *to* m **do**
  Draw node i with probability $\frac{k_i}{2m}$;
  Draw node j with probability $\frac{k_j}{2m}$;
  `/* Add edge` (i, j) `to the graph */`
  if $i \neq j$ **then**
    $a_{ij} = a_{ji} = 1$;
  end
end

In this case, the probability on a single draw of drawing edge (i, j) is still given by the following

$$p_{ij} = \begin{cases} \frac{k_i k_j}{4m^2} & i \neq j, \\ \frac{k_i^2}{4m^2} & i = j, \end{cases} \tag{1.37}$$

but the probability of having edge (i, j) in the graph after m draws is now given by

$$p_{ij}(m) = \begin{cases} \frac{k_i k_j}{2m} + \mathcal{O}(p_{ij}^2) & i \neq j, \\ 0 & i = j. \end{cases} \tag{1.38}$$

So $p_{ij}(m)$ remains the same for $i \neq j$, while by design $p_{ii}(m)$ now equals 0 for all i. This model is equivalent to the Bernoulli Chung-Lu model with the probabilities in Equation (1.3) replaced with the probabilities given in Equation (1.38) and we refer to this model as Model IV. Of course, by not including self-edges in the model we have also changed the expected degree for all nodes. If we ignore the higher order terms in

Equation (1.38) then the expected degree is given by

$$\begin{aligned} E(\mathbf{D}_i) &= E(\sum_j \mathbf{A}_{ij}) = \sum_j E(\mathbf{A}_{ij}) = \sum_{j\neq i} E(\mathbf{A}_{ij}) \\ &= \sum_{j\neq i} p_{ij}(m) = \sum_{j\neq i} \frac{k_i k_j}{2m} \\ &= k_i - \frac{k_i^2}{2m}, \end{aligned} \tag{1.39}$$

which is the same as in Model III, the Bernoulli Chung-Lu model with no self-edges. However, if we don't ignore the higher order terms in $p_{ij}(m)$ then the expected degree is calculated as follows

$$\begin{aligned} E(\mathbf{D}_i) &= \sum_{j\neq i} p_{ij}(m) = \sum_{j\neq i} (1 - (1 - 2\frac{k_i k_j}{(2m)^2})^m) \\ &= \sum_{j\neq i} (1 - (1 - \frac{k_i k_j}{2m^2})^m), \end{aligned} \tag{1.40}$$

where $E(\mathbf{D}_i) < k_i - \frac{k_i^2}{2m}$ since we showed in Section 1.3 that $1 - (1 - 2\frac{k_i k_j}{(2m)^2})^m < \frac{k_i k_j}{2m}$. We return to the general relativity collaboration network to examine how these model properties differ in a real network. We note that there are no self-edges in the original network. Figure 1.5 plots the difference between the expected degree in Model IV and the upper bound on the expected degree, Equation (1.39), which is also the expected degree in Model III. We also note that the expected number of edges in Model IV for the general relativity network is 13333.55 which is approximately 100 edges fewer than in the actual graph, which is 13422, and where the expected number of edges in Model III is 13413.01.

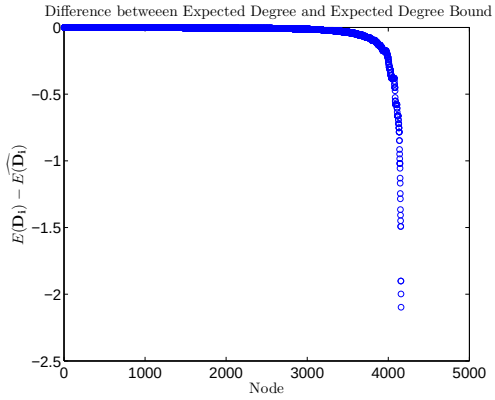


Figure 1.5: General Relativity Collaboration Network: Comparing Expected Degree from Model IV and Expected Degree Upper Bound, the Expected Degree from Model III.

		Probability of Edge (i,j)	Probability of Edge (i,i)	Expected Degree $E(\mathbf{D}_i)$
Bernoulli Chung-Lu	Self-Edges: Model I	$\frac{k_i k_j}{2m}$	$\frac{k_i^2}{4m}$	k_i
	No Self-Edges: Model III	$\frac{k_i k_j}{2m}$	0	$k_i - \frac{k_i^2}{2m}$
$\mathcal{O}(m)$ Chung-Lu	Self-Edges: Model II	$1-(1-2\frac{k_i k_j}{4m^2})^m$ $< \frac{k_i k_j}{2m}$	$1-(1-\frac{k_i^2}{4m^2})^m$ $< \frac{k_i^2}{4m}$	$\sum_{j\neq i} 1-(1-2\frac{k_i k_j}{4m^2})^m$ $+1-(1-\frac{k_i^2}{4m^2})^m < k_i$
	No Self-Edges: Model IV	$1-(1-2\frac{k_i k_j}{4m^2})^m$ $< \frac{k_i k_j}{2m}$	0	$\sum_{j\neq i} 1-(1-2\frac{k_i k_j}{4m^2})^m$ $< k_i - \frac{k_i^2}{2m}$

Table 1.2: Model Summary

1.4.3 Summary

We have looked at two different models, the Bernoulli Chung-Lu model and the $\mathcal{O}(m)$ Chung-Lu model and modified both to exclude self-edges. The following table summarizes the models presented where we refer to the probability of edge (i,j) as the probability of edge (i,j) being in the graph after the algorithm has been executed. We next look at what happens to these models when the constraint $k_i^2 \leq 2m \ \forall \ i$ is violated.

1.5 Constraint Violation

1.5.1 Bernoulli Chung-Lu Model

Suppose we use as an input into either the Bernoulli Chung-Lu model or the $\mathcal{O}(m)$ Chung-Lu model a degree sequence $k = (k_i, \ldots, k_n)$ where the constraint $k_i^2 \leq 2m \ \forall \ i$ is violated. To see why this constraint is important we revisit the edge probability formulas from the Bernoulli Chung-Lu model which are given as follows

$$p_{ij} = \begin{cases} \frac{k_i k_j}{2m} & i \neq j, \\ \frac{k_i^2}{4m} & i = j. \end{cases} \tag{1.41}$$

The constraint is important because it provides a sufficient condition to guarantee that these probabilities are well-defined (i.e. $0 \leq p_{ij} \leq 1 \ \forall \ i,j$). Consider the case where $k_i > 2m$ for only one node. Then as

long as $k_i^2 \leq 4m$, all the probabilities in Equation (1.41) are still well-defined. However, if $k_i^2 > 4m$ then $p_{ii} > 1$ and the probability of this self-edge is not well-defined. However, the larger issue is when $k_i^2 > 2m$ for two or more nodes. Suppose we have two nodes i and j where $k_i^2 > 2m$ and $k_j^2 > 2m$. This implies that $k_i k_j > 2m$ and $p_{ij} > 1$. In fact, for all pairs of nodes i and j with $k_i^2 > 2m$, and $k_j^2 > 2m$ the probability of edge (i, j) being in the graph, p_{ij}, is greater than 1. In order for the probabilities to be well-defined we need to redefine them. In the case where the constraint $k_i^2 \leq 2m \; \forall \; i$ is violated the probability of choosing and edge between nodes i and j is given by

$$p_{ij} = \begin{cases} \min\left\{\frac{k_i k_j}{2m}, 1\right\} & i \neq j, \\ \min\left\{\frac{k_i^2}{4m}, 1\right\} & i = j. \end{cases} \tag{1.42}$$

If the probabilities are defined as above then $0 \leq p_{ij} \leq 1 \; \forall \; i, j$. If the constraint $k_i^2 \leq 2m \; \forall \; i$ holds then the probabilities reduce to those in Equation (1.41). Thus, Model I is imbedded in this model, which we refer to as Model V. We can calculate the expected degree for Model V as

$$\begin{aligned} E(\mathbf{D}_i) = E\left(\sum_j \mathbf{A}_{ij}\right) &= \sum_j E(\mathbf{A}_{ij}) = \sum_{j \neq i} \min\{\frac{k_i k_j}{2m}, 1\} + 2\min\{\frac{k_i^2}{4m}, 1\} \\ &= \sum_{j \neq i} \min\{\frac{k_i k_j}{2m}, 1\} + \min\{\frac{k_i^2}{2m}, 2\}, \end{aligned} \tag{1.43}$$

where $E(\mathbf{D}_i) \leq k_i$ and if the constraint $k_i^2 \leq 2m \; \forall \; i$ holds then $E(\mathbf{D}_i) = k_i$. The difference between k_i and $E(\mathbf{D}_i)$ can be written as

$$k_i - E(\mathbf{D}_i) = \frac{k_i^2}{2m} - \min\left\{\frac{k_i^2}{2m}, 2\right\} + \sum_{j \neq i}\left(\frac{k_i k_j}{2m} - \min\left\{\frac{k_i k_j}{2m}, 1\right\}\right) \tag{1.44}$$

where we use the following to rewrite k_i,

$$k_i = \sum_j \frac{k_i k_j}{2m} = \sum_{j \neq i} \frac{k_i k_j}{2m} + \frac{k_i^2}{2m}. \tag{1.45}$$

From this equation we can see that for $i \neq j$, it is the difference between $\frac{k_i k_j}{2m}$ and 1, when $\frac{k_i k_j}{2m} > 1$, that contributes to the difference between k_i and $E(\mathbf{D}_i)$. For self-edges, it is the difference between $\frac{k_i^2}{2m}$ and 2, when $\frac{k_i^2}{2m} > 2$, that contributes to the difference between k_i and $E(\mathbf{D}_i)$.

To explore this difference empirically we now consider a network that violates the constraint $k_i^2 \leq 2m \; \forall \; i$. The network we will examine is an autonomous system graph which maps the traffic flow across the internet. It has 6474 nodes and 12572 edges. The largest degree node has 1458 neighbors, thus $k_i^2 > 2m$ for at least 1 node and in fact, for this network, k_i^2 is greater than $2m$ for 491 nodes. This implies that there are 120295 probabilities, $p_{ij}, \; i \neq j$, that need to be redefined and set equal to 1. This amounts to less than 1% of all probabilities, $p_{ij}, \; i \neq j$. However, these violations can have a noticeable impact on the model and its expected degree. Figures 1.6 and 1.7 compare the expected degree to k_i.

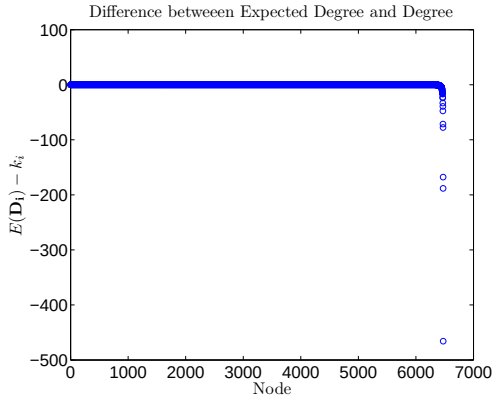


Figure 1.6: Autonomous System Network: Comparing Expected Degree from Model V and Actual Degree.

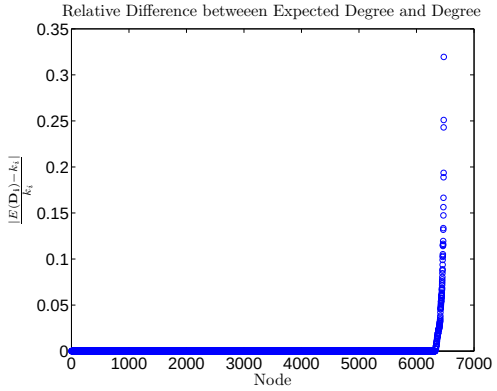


Figure 1.7: Autonomous System Network: Relative difference between Expected Degree Bound from Model V and Actual Degree.

1.5.2 ($\mathcal{O}(|Edges|)$ Chung-Lu Model

In the $\mathcal{O}(m)$ Chung-Lu model all the probabilities are still well-defined even if the constraint $k_i^2 \leq 2m \; \forall \; i$ is violated. To see this, we use Equation (1.28), which defines the probability of choosing edge (i, j) and which we include here for ease of exposition:

$$p_{ij}(m) = \begin{cases} 1 - (1 - \frac{k_i k_j}{2m^2})^m & i \neq j, \\ 1 - (1 - \frac{k_i^2}{4m^2})^m & i = j. \end{cases} \tag{1.46}$$

Even if $k_i^2 > 2m$ for more than one node i, it is still true that both $0 \leq 1 - \frac{k_i k_j}{2m^2} \leq 1$ and $0 \leq 1 - \frac{k_i^2}{4m^2} \leq 1$ which implies that $0 \leq p_{ij}(m) \leq 1, \; \forall \; i, j$. This implies that our model does not change and that the expected degree for each node is still well defined and is given by Equation (1.31). However, as noted in

Section 1.3, the expected degree in the $\mathcal{O}(m)$ Chung-Lu model can be much different from k_i, the expected degree in the original Bernoulli Chung-Lu model, and this difference is exacerbated when the constraint is violated. As we noted previously, the maximum value of $p_{ij}(m)$ is $\frac{k_i k_j}{2m}$ for $i \neq j$ and $\frac{k_i^2}{4m}$ for self-edges. However, when the constraint $k_i^2 \leq 2m \ \forall \ i$ is violated, and $\frac{k_i k_j}{2m} > 1$ for some $i \neq j$, since p_{ij} is still a well-defined probability, we know that 1 is in fact a lower upper bound. The same is true if $\frac{k_i^2}{4m} > 1$ for some node i; p_{ii} still has an upper bound of 1. Using this knowledge we can actually calculate a tighter upper bound on the expected degree for Model II when the degree constraint is violated. Recall from Section 1.3 that the upper bound on $E(\mathbf{D}_i)$ was k_i when $k_i^2 \leq 2m \ \forall \ i$. To calculate our new upper bound, we note that for nodes i and j where $\frac{k_i k_j}{2m} \leq 1$, the upper bound on $p_{ij}(m)$ is still $\frac{k_i k_j}{2m}$, and if $\frac{k_i k_j}{2m} > 1$ then the upper bound on $p_{ij}(m)$ is 1. For self-edges, if $\frac{k_i^2}{4m} \leq 1$, then upper bound on $p_{ii}(m)$ is still $\frac{k_i^2}{4m}$, and if $\frac{k_i^2}{4m} > 1$ then the upper bound on $p_{ii}(m)$ is 1. Combining this information we get the upper bound on the expected degree of node i as

$$\widehat{E(\mathbf{D}_i)} = \min\left\{\frac{k_i^2}{2m}, 2\right\} + \sum_{j \neq i} \min\left\{\frac{k_i k_j}{2m}, 1\right\} \tag{1.47}$$

which is equal to the expected degree, Equation (1.43), from Model V. So when the constraint $k_i^2 \leq 2m \ \forall \ i$ is violated, the expected degree of node i in the $\mathcal{O}(m)$ Chung-Lu model, Model II, is further from k_i than if the constraint held. Empirically, we can study this model using a real-world network to generate instances of the model. We use the degree sequence from the same autonomous system graph that was used with Model V. Figure 1.8 plots the difference between the expected degree and the degree, which is the expected degree in the original Chung-Lu model, Model I. Figures 1.9 and 1.10 break this difference down by plotting the difference between the expected degree and the upper bound on the expected degree and the difference between the upper bound on the expected degree and the degree, respectively. We can see for some large degree nodes this difference can be quite large. This is also reflected in the number of edges. The expected number of edges for this graph can be calculated from the expected degree of each node and is equal to 11445.185225. The upper bound on the expected number of edges can be calculated as

$$\widehat{E(\mathbf{M})} = \frac{1}{2}\sum_i \widehat{E(\mathbf{D}_i)}, \tag{1.48}$$

where $\widehat{E(\mathbf{D}_i)}$ is given by Equation (1.47). For the autonomous system graph, $\widehat{E(\mathbf{M})} = 11836.836939$. Note that the total number of edges in the original autonomous system graph is 12572. Thus there are approximately 1000 fewer edges expected in the graph than are in the original graph which is approximately 9% of the total number of edges.

1.5.3 Removing Self-Edges

Sections 1.5.1 and 1.5.2 analyze what happens to Models I and II when the constraint $k_i^2 \leq 2m \ \forall \ i$ is violated, respectively. We are still left to analyze Models III and IV in the case of the degree constraint violation. Model III is simply Model I without self-edges, thus our analysis of what happens to Model III when the degree constraint is violated will be similar to our analysis in Section 1.5.1. Similarly, Model IV is simply Model II without self-edges and thus our analysis of what happens to Model IV when the degree

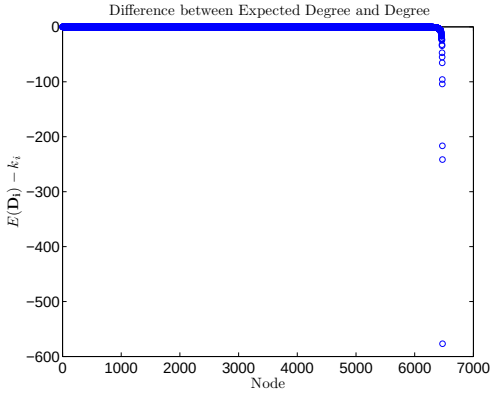


Figure 1.8: Autonomous System Network: Comparing Expected Degree from Model II and Actual Degree.

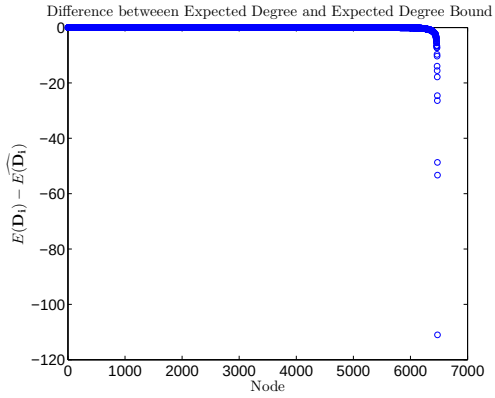


Figure 1.9: Autonomous System Network: Comparing Expected Degree and Expected Degree Bound from Model II.

constraint is violated will be similar to our analysis from Section 1.5.2. In Model III, the probability of choosing edge (i, j) is given by

$$p_{ij} = \begin{cases} \frac{k_i k_j}{2m} & i \neq j, \\ 0 & i = j. \end{cases} \tag{1.49}$$

If the degree constraint is violated, then not all the probabilities are necessarily well-defined. Note, as we have previously mentioned, $k_i^2 \leq 2m \; \forall \; i$, is a sufficient but not necessary condition to guarantee that all the probabilities in Equation (1.49) are well-defined. For our analysis of Model III we are assuming that $k_i^2 > 2m$ for at least 2 nodes, otherwise p_{ij} as defined in Equation (1.49) remains well-defined. So if we suppose that not all the probabilities in Equation (1.49) are well-defined we need to redefine the probabilities. In the case where the degree constraint is violated we define the probability of choosing edge

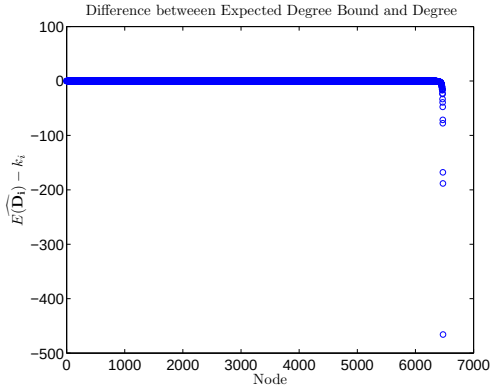


Figure 1.10: Autonomous System Network: Comparing Expected Degree Bound from Model II and Actual Degree

(i, j) as follows

$$p_{ij} = \begin{cases} \min\left\{\frac{k_i k_j}{2m}, 1\right\} & i \neq j, \\ 0 & i = j, \end{cases} \tag{1.50}$$

and where we refer to the model defined by the probabilities in Equation (1.50) as Model VI. We can calculate the expected degree for Model VI as

$$\begin{aligned} E(\mathbf{D}_i) &= E\left(\sum_j \mathbf{A}_{ij}\right) = \sum_j E(\mathbf{A}_{ij}) \\ &= \sum_{j \neq i} \min\left\{\frac{k_i k_j}{2m}, 1\right\} \end{aligned} \tag{1.51}$$

where $E(\mathbf{D}_i) \leq k_i - \frac{k_i^2}{2m}$ and if the constraint $k_i^2 \leq 2m \ \forall \ i$ holds then $E(\mathbf{D}_i) = k_i - \frac{k_i^2}{2m}$. The difference between k_i and $E(\mathbf{D}_i)$ can be written as

$$k_i - E(\mathbf{D}_i) = \frac{k_i^2}{2m} + \sum_{j \neq i} \left(\frac{k_i k_j}{2m} - \min\left\{\frac{k_i k_j}{2m}, 1\right\}\right) \tag{1.52}$$

We once again use the autonomous system graph to explore this model numerically. The expected degree is given by Equation (1.51) and in Figure 1.11 we plot the difference between the expected degree and the expected degree bound, which is given by $\widehat{E(\mathbf{D}_i)} = k_i - \frac{k_i^2}{2m}$. Figure 1.12 plots the difference between $E(\mathbf{D}_i)$ and k_i, where k_i is the expected degree value in Model I. Note that $E(\mathbf{D}_i) \leq \widehat{E(\mathbf{D}_i)} < k_i$ and the difference between $\widehat{E(\mathbf{D}_i)}$ and k_i can be quite large, as can be seen in Figure 1.13. Contrast this to the case where $k_i^2 \leq 2m \ \forall \ i$, where the maximum difference between k_i and $k_i - \frac{k_i^2}{2m}$ is 1 since $\frac{k_i^2}{2m} \leq 1 \ \forall \ i$. However, the majority of the difference between $E(\mathbf{D}_i)$ and k_i is in the difference between $E(\mathbf{D}_i)$ and $\widehat{E(\mathbf{D}_i)}$ as can be seen in Figure 1.11.

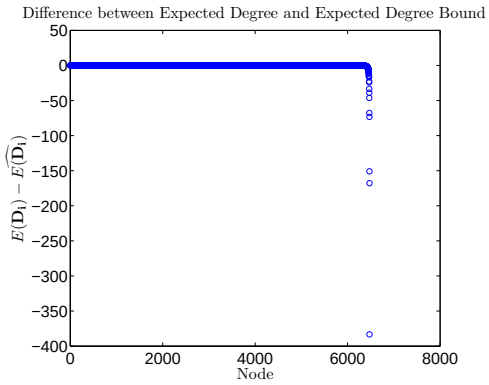


Figure 1.11: Autonomous System Network: Comparing Expected Degree and Expected Degree Bound from Model VI.

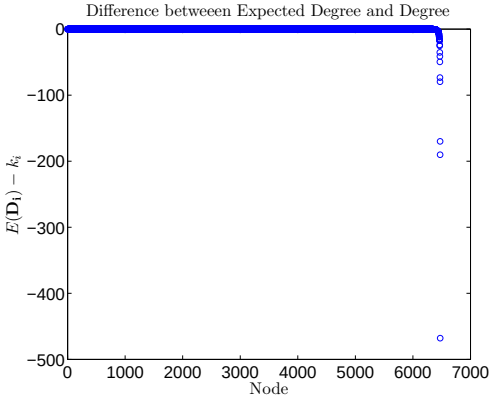


Figure 1.12: Autonomous System Network: Comparing Expected Degree from Model VI and Actual Degree.

Finally, we consider Model IV, under the assumption that the constraint $k_i^2 \leq 2m \; \forall \; i$ is violated. The probability of choosing edge (i, j) in this model is given by

$$p_{ij}(m) = \begin{cases} 1 - (1 - \frac{k_i k_j}{2m^2})^m & i \neq j, \\ 0 & i = j. \end{cases} \tag{1.53}$$

When the constraint $k_i^2 \leq 2m \; \forall \; i$ is violated, from Section 1.5.2 we know that $p_{ij}(m)$ is still well-defined for $i \neq j$ and since $p_{ii}(m) = 0 \; \forall \; i$, p_{ij} is still well-defined for all i, j. Thus, our model does not change and the expected degree is still given by Equation (1.40). As in Section 1.5.2, we can put an upper bound on the expected degree and compare this to the expected degree from Model I. As we noted in Section 1.5.2 the maximum value for $p_{ij}(m)$ is $\min\{1, \frac{k_i k_j}{2m}\}$ for $i \neq j$. Thus, the upper bound on the expected degree of

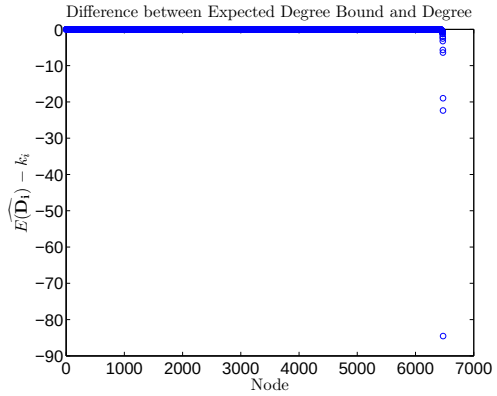


Figure 1.13: Autonomous System Network: Comparing Expected Degree Bound from Model VI and Actual Degree.

node i is

$$\widehat{E(\mathbf{D}_i)} = \sum_{j \neq i} \min\left\{\frac{k_i k_j}{2m}, 1\right\}, \tag{1.54}$$

which is equivalent to $E(\mathbf{D}_i)$ from Model VI. This upper bound will be strict and thus the expected degree from Model VI will always be greater than the expected degree from Model IV. To see this we note that for nodes $i \neq j$ where $\frac{k_i k_j}{2m} > 1$, the probability of edge (i, j) being in Model VI is 1 and in Model IV is $1 - (1 - \frac{k_i k_j}{2m^2})^m < 1$. For nodes $i \neq j$ where $\frac{k_i k_j}{2m} \leq 1$, the probability of edge (i, j) being in Model VI is $\frac{k_i k_j}{2m}$ and in Model IV is $1 - (1 - \frac{k_i k_j}{2m^2})^m < \frac{k_i k_j}{2m}$. Note that we have the following relationship for Model IV,

$$E(\mathbf{D}_i) < \widehat{E(\mathbf{D}_i)} \leq k_i - \frac{k_i^2}{2m} < k_i \tag{1.55}$$

where $k_i - \frac{k_i^2}{2m}$ is the expected degree from Model III and k_i is the expected degree from Model I. To see how large these differences can be empirically we again use the autonomous system graph. Figure 1.14 plots the difference between the expected degree and the expected degree bound, while Figure 1.15 plots the difference between the expected degree bound and the degree. Figure 1.16 summarizes these differences by plotting the difference between the expected degree and the degree in the original graph which is quite large for some large degree nodes.

1.5.4 Summary

We have examined both the Bernoulli and $\mathcal{O}(m)$ Chung-Lu models with and without self-edges when the constraint $k_i^2 \leq 2m \ \forall \ i$ is violated. When the constraint is violated in the Bernoulli Chung-Lu model, both with and without self-edges, we actually need to define new models. However, when the constraint is violated in the $\mathcal{O}(m)$ Chung-Lu model, both with and without self-edges, the models do not change. The following table summarizes the models presented when the constraint $k_i^2 \leq 2m \ \forall \ i$ is violated and note that for all four models, when the constraint is violated, the degree of certain nodes, particularly large degree nodes, in any given instance of the model can be much less than the degree in the original graph.

		Probability of Edge (i,j)	Probability of Edge (i,i)	Expected Degree $E(\mathbf{D}_i)$	Expected Degree Bound $\widehat{E(\mathbf{D}_i)}$
Bernoulli Chung-Lu	Self-Edges Model V	$\min\left\{\frac{k_i k_j}{2m}, 1\right\}$	$\min\left\{\frac{k_i^2}{4m}, 1\right\}$	$\sum_{j\neq i} \min\left\{\frac{k_i k_j}{2m}, 1\right\}$ $+ \min\left\{\frac{k_i^2}{2m}, 2\right\}$	k_i
	No Self-Edges Model VI	$\min\left\{\frac{k_i k_j}{2m}, 1\right\}$	0	$\sum_{j\neq i} \min\left\{\frac{k_i k_j}{2m}, 1\right\}$	$k_i - \frac{k_i^2}{2m}$
$\mathcal{O}(m)$ Chung-Lu	Self-Edges Model II	$1-(1-2^{\frac{k_i k_j}{4m^2}})^m$ $< \min\left\{\frac{k_i k_j}{2m}, 1\right\}$	$1-(1-\frac{k_i^2}{4m^2})^m$ $< \min\left\{\frac{k_i^2}{4m}, 1\right\}$	$\sum_{j\neq i} 1-(1-2^{\frac{k_i k_j}{4m^2}})^m$ $+1-(1-\frac{k_i^2}{4m^2})^m$	$\sum_{j\neq i} \min\left\{\frac{k_i k_j}{2m}, 1\right\}$ $+ \min\left\{\frac{k_i^2}{2m}, 2\right\}$
	No Self-Edges Model IV	$1-(1-2^{\frac{k_i k_j}{4m^2}})^m$ $< \min\left\{\frac{k_i k_j}{2m}, 1\right\}$	0	$\sum_{j\neq i} 1-(1-2^{\frac{k_i k_j}{4m^2}})^m$	$\sum_{j\neq i} \min\left\{\frac{k_i k_j}{2m}, 1\right\}$

Table 1.3: Model Summary – Degree Constraint Violated

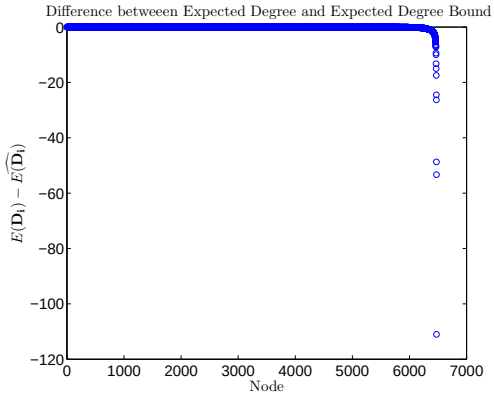


Figure 1.14: Autonomous System Network: Comparing Expected Degree and Expected Degree Bound from Model IV.

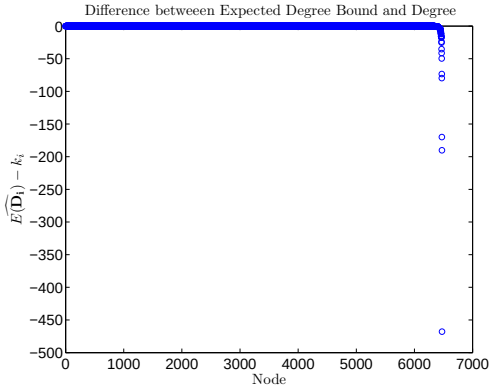


Figure 1.15: Autonomous System Network: Comparing Expected Degree Bound from Model IV and Actual Degree.

1.6 Triangle Counts and Clustering Coefficients

One of the nice properties of the Chung-Lu model, as originally proposed, is the fact that $E(\mathbf{D}_i) = k_i \; \forall \; i$. This makes it an attractive model, since it can be used to model networks with any degree sequence. However, researchers are often interested in building models that match other network properties, including the network community structure. The network community structure is often measured by the clustering coefficient which can be defined for each node i as follows

$$C_i = \frac{\text{Number of pairs of neighbors of } i \text{ that are connected}}{\text{Number of pairs of neighbors of } i} \tag{1.56}$$

or

$$C_i = \frac{\text{Number of unique triangles node } i \text{ belongs to}}{\text{Number of pairs of neighbors of } i}. \tag{1.57}$$

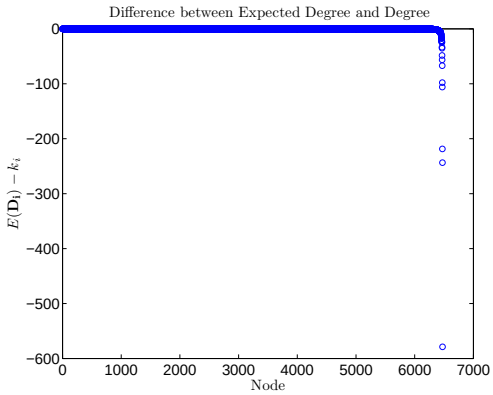


Figure 1.16: Autonomous System Network: Comparing Expected Degree from Model IV and Actual Degree.

The clustering coefficient can be written as a random variable, $\mathbf{C}_i$, where $\mathbf{C}_i$ is a function of the random variables $\mathbf{A}_{ij}$, $j = 1, \ldots, n$. We can then compare the realizations of this random variable from our model instances and its expected value with the clustering coefficients from the real network. Unfortunately, the number of triangles node i belongs to is not independent of the number of pairs of neighbors of node i and thus the expected value of the clustering coefficient, $E(\mathbf{C}_i)$, is not easily determined. If these two variables where independent then we could write the following

$$E(\mathbf{C}_i) = \frac{E(\text{Number of unique triangles node } i \text{ belongs to})}{E(\text{Number of pairs of neighbors of } i)}. \tag{1.58}$$

Despite the fact that $E(\mathbf{C}_i)$ cannot be written as above we note that in the Bernoulli Chung-Lu model, Model I, $E(\mathbf{D}_i) = k_i$, and thus on average the number of pairs of neighbors of node i is the same in any instance of Model I as in the real network. It has been noted empirically that the Chung-Lu model generates instances that under-estimate the clustering coefficient. Thus, if $\mathbf{C}_i = c_i$ is much less on average in the model instances then it is due to the lack of triangles for node i. In all the other models we have introduced, $E(\mathbf{D}_i) < k_i$, so the number of pairs of neighbors of node i is on average less in a given model instance than in the real network. Thus, for $\mathbf{C}_i = c_i$ to be less on average for a given model instance relative to the real-network, the number of triangles for node i must be less on average in the model instance relative to the real network by a greater proportion than the number of pairs of neighbors. Not only does the number of triangles for each node play an important role in determining the clustering coefficient but it can also be used as a measure of community structure independent of the clustering coefficient. Hence, we focus on determining its value in our models as a way to look at community structure and to examine the difference between our models and real networks. We begin by noting that the number of triangles for a given node i in the original Chung-Lu model, Model I, is a random variable that can be defined as

$$\mathbf{T}_i = \sum_{j \neq i} \sum_{k \neq i, k \neq j} \mathbf{A}_{ij}\mathbf{A}_{ik}\mathbf{A}_{jk}, \ \forall\, i, \tag{1.59}$$

where a triangle occurs at node i if there is an edge between nodes i and j, between nodes i and k and between nodes k and j where the nodes i, j and k are all unique nodes. Note that every triangle is counted

twice in the above sum since A_{jk} and A_{kj} both appear. Since $\mathbf{T}_i$ is a random variable we can compute the expected value of $\mathbf{T}_i$ as

$$\begin{aligned} E(\mathbf{T}_i) &= \sum_{j\neq i}\sum_{k\neq i,j} \mathbf{A}_{ij}\mathbf{A}_{ik}\mathbf{A}_{jk} \\ &= \sum_{j\neq i}\sum_{k\neq i,j} E(\mathbf{A}_{ij}\mathbf{A}_{ik}\mathbf{A}_{jk}) \\ &= \sum_{j\neq i}\sum_{k\neq i,j} E(\mathbf{A}_{ij})E(\mathbf{A}_{ik})E(\mathbf{A}_{jk}) \\ &= \sum_{j\neq i}\sum_{k\neq i,j} p_{ij}p_{ik}p_{jk}, \end{aligned} \tag{1.60}$$

where $E(\mathbf{A}_{ij}\mathbf{A}_{ik}\mathbf{A}_{jk}) = E(\mathbf{A}_{ij})E(\mathbf{A}_{ik})E(\mathbf{A}_{jk})$ because each edge is drawn independently. Equation (1.3) defines $p_{ij}, i, j = 1, \ldots, n$, in Model I which allows us to simplify $E(\mathbf{T}_i)$ as follows

$$\begin{aligned} E(\mathbf{T}_i) &= \sum_{j\neq i}\sum_{k\neq i,j} p_{ij}p_{ik}p_{jk} \\ &= \sum_{j\neq i}\sum_{k\neq i,j} \left(\frac{k_i k_j}{2m}\right)\left(\frac{k_i k_k}{2m}\right)\left(\frac{k_j k_k}{2m}\right) \\ &= \frac{k_i^2}{(2m)^3}\sum_{j\neq i} k_j^2 \sum_{k\neq i,j} k_k^2 \\ &= \frac{k_i^2}{(2m)^3}\sum_{j\neq i} k_j^2 \left(\mathcal{K}_2 - k_i^2 - k_j^2\right) \\ &= \frac{k_i^2}{(2m)^3}\sum_{j\neq i} \mathcal{K}_2 k_j^2 - k_i^2 k_j^2 - k_j^4 \\ &= \frac{k_i^2}{(2m)^3}\left(\mathcal{K}_2(\mathcal{K}_2 - k_i^2) - k_i^2(\mathcal{K}_2 - k_i^2) - (\mathcal{K}_4 - k_i^4)\right) \\ E(\mathbf{T}_i) &= \frac{k_i^2}{(2m)^3}\left((\mathcal{K}_2 - k_i^2)^2 - (\mathcal{K}_4 - k_i^4)\right), \end{aligned} \tag{1.61}$$

where we define $\mathcal{K}_2 = \sum_{i=1}^n k_i^2$ and $\mathcal{K}_4 = \sum_{i=1}^n k_i^4$. It is interesting to note that the expected value of the number of triangles at node i is a function of the degrees and is not related to the number of triangles in the network. If we were interested in better matching the expected number of triangles at node i to the actual number of triangles at node i, t_i, then a possible strategy would be to make p_{ij} a function of t_i and t_j. However, $E(\mathbf{D}_i)$ would then be a function of t_i and would no longer be equal to k_i. Also note that random variable $\mathbf{T}_i$ is not a function of any self-edges which means that neither is $E(\mathbf{T}_i)$ and thus for Model III, $E(\mathbf{T}_i)$ is also given by Equation (1.61). For Models II and IV, we can determine $E(\mathbf{T}_i)$ if

we replace p_{ij} in Equation (1.60) with p_{ij}, $i \neq j$ from Equation (1.28) as follows

$$\begin{aligned} E(\mathbf{T}_i) &= \sum_{j \neq i} \sum_{k \neq i,j} p_{ij} p_{ik} p_{jk} \\ &= \sum_{j \neq i} \sum_{k \neq i,j} p_{ij}(m) p_{ik}(m) p_{jk}(m) \\ &= \sum_{j \neq i} \sum_{k \neq i,j} \left(1 - \left(1 - 2\frac{k_i k_j}{(2m)^2})\right)^m\right) \left(1 - \left(1 - 2\frac{k_i k_k}{(2m)^2}\right)^m\right) \left(1 - \left(1 - 2\frac{k_j k_k}{(2m)^2}\right)^m\right), \end{aligned} \tag{1.62}$$

which can not be easily simplified. However, as we have noted before the probability of an edge between distinct nodes in Models II and IV is lower than in Models I and IV. This implies that $E(\mathbf{T}_i)$ will be lower in Models II and IV than in Models I and III. To better quantify the differences between $E(\mathbf{T}_i)$ in our models and in a real network we revisit the general relativity collaboration network. Figure 1.17 plots the difference between the expected number of triangles in Models I and III and the actual number of triangles. This difference is quite large for some nodes. Figure 1.18 plots the relative difference between the expected number of triangles in Models I and III and the actual number of triangles for each node with at least one triangle in the real network. For Models II and IV, Figure 1.19 plots the difference between the expected number of triangles and the actual number of triangles and Figure 1.20 plots the relative difference between the expected number of triangles and the actual number of triangles. From these figures it is hard to determine if the expected number of triangles in Models I and III is closer to the actual triangle count or if the expected number of triangles in Models II and IV is closer to the actual triangle count. Figure 1.21 plots the difference between the expected number of triangles in Models II and IV and the upper bound which is the expected number of triangles in Models I and III. From the figure we can see that the expected number of triangles in Models I and III is higher than in Models II and IV and thus the expected number of triangles in Models I and III is closer to the true triangle count. In general, since $E(\mathbf{T}_i)$ is not a function of T_i we can not draw any conclusions about whether $E(\mathbf{T}_i)$ will be closer to T_i in Models I and III or in Models II and IV.

In the above analysis we have assumed that the constraint, $k_i^2 \leq 2m \ \forall \ i$, holds. If this constraint no longer holds, then our above analysis needs to be revised to account for the constraint violation. As we mentioned in Section 1.5, if the constraint, $k_i^2 \leq 2m \ \forall \ i$, is violated then instead of using Models I and III we use Models V and VI, respectively. For both Model V and VI, $E(\mathbf{T}_i)$ is the same, and can be determined by using Equation (1.42) to define p_{ij}, $i \neq j$, and substituting this into Equation (1.60). This gives us the following

$$\begin{aligned} E(\mathbf{T}_i) &= \sum_{j \neq i} \sum_{k \neq i,j} p_{ij} p_{ik} p_{jk} \\ &= \sum_{j \neq i} \sum_{k \neq i,j} \min\left\{\frac{k_i k_j}{2m}, 1\right\} \min\left\{\frac{k_i k_k}{2m}, 1\right\} \min\left\{\frac{k_j k_k}{2m}, 1\right\}. \end{aligned} \tag{1.63}$$

Equation (1.63) will be less than Equation (1.61). However, this does not imply that when the constraint is violated $E(\mathbf{T}_i)$ will be further from the true triangle count since $E(\mathbf{T}_i)$ is not a function of the number of the triangles in the original network. Equation (1.61) only provides an upper bound on Equation (1.63).

For the $\mathcal{O}(m)$ Chung-Lu models, Models II and IV when the constraint, $k_i^2 \leq 2m \ \forall \ i$, is violated the models do not change and hence $E(\mathbf{T}_i)$ is still given by Equation (1.62). However, if the constraint is

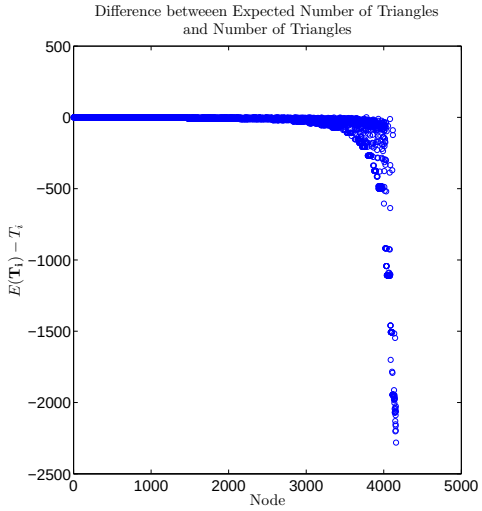


Figure 1.17: General Relativity Collaboration Network: Comparing Expected Number of Triangles from Models I and III and Actual Number of Triangles.

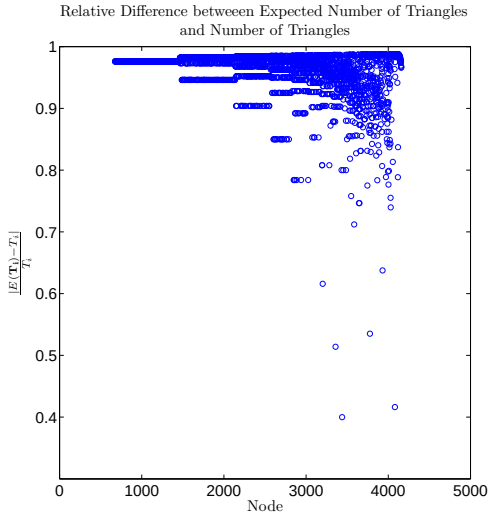


Figure 1.18: General Relativity Collaboration Network: Relative difference between Expected Number of Triangles from Models I and III and Actual Number of Triangles.

violated then the upper bound on $p_{ij},\ i \neq j$, is given by

$$p_{ij} < \min\left\{1, \frac{k_i k_j}{2m}\right\} \tag{1.64}$$

This implies that upper bound on $E(\mathbf{T}_i)$ is given by Equation (1.63) a tighter upper bound than Equation (1.61). Note that if the constraint, $k_i^2 \leq 2m\ \forall\ i$, holds then Equations (1.63) and (1.61) are the same.

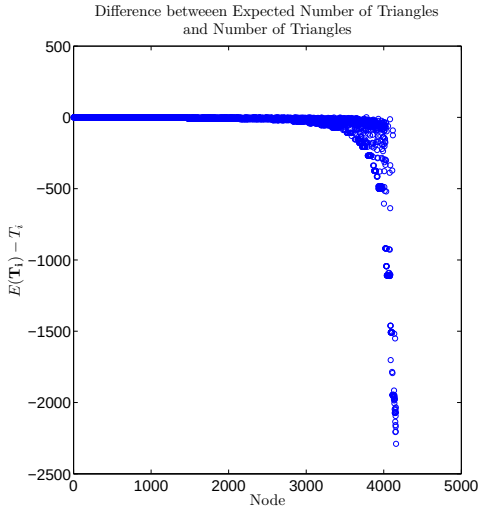


Figure 1.19: General Relativity Collaboration Network: Comparing Expected Number of Triangles from Models II and IV and Actual Number of Triangles.

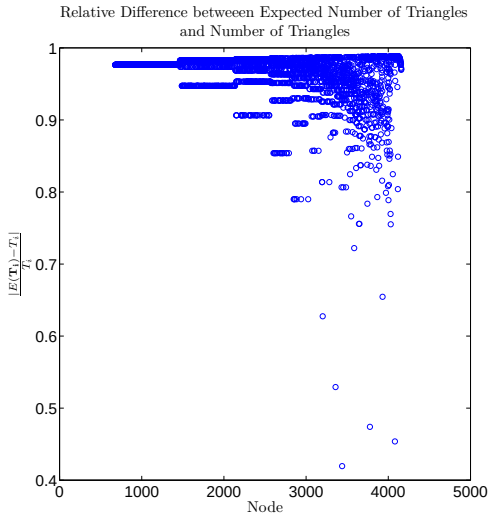


Figure 1.20: General Relativity Collaboration Network: Relative difference between Expected Number of Triangles from Models II and IV and Actual Number of Triangles.

Numerically, we can examine the affect of the constraint violation on the number of triangles using the autonomous system network we have examined previously. Figures 1.22 and 1.23 plot the difference between $\mathbf{E}(T_i)$ and T_i, the actual number of triangles node i belongs to, for Models V and VI. Notice how for this network $E(\mathbf{T}_i) > T_i$ for a number of nodes whereas for the general relativity network, $E(\mathbf{T}_i) < T_i$ for all node i. Note that for the general relativity network, which does not violate the constraint, $k_i^2 \leq 2m, \ \forall \ i$,

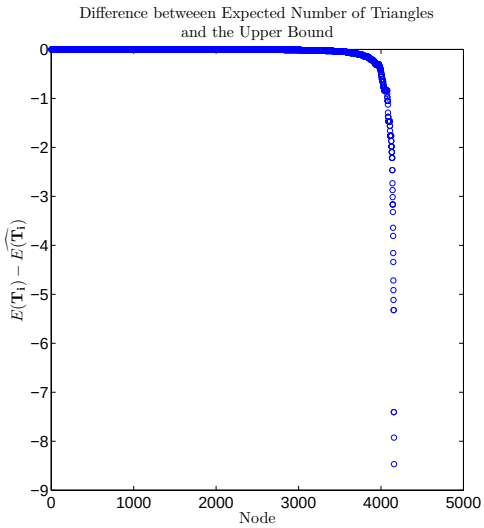


Figure 1.21: General Relativity Collaboration Network: Comparing Expected Number of Triangles from Models II and IV and the Upper Bound.

Models I and III are equivalent to Models V and VI, respectively. In general, since $E(\mathbf{T}_i)$ is not a function of T_i we can not draw any conclusions about whether $E(\mathbf{T}_i)$ will be greater or less than T_i. Figures 1.24 and 1.25 plot the difference between between $\mathbf{E}(T_i)$ and T_i for Models II and IV. Note from the figures that Models II and IV are actually better able to match the number of triangles than Models V and VI although no conclusion can be drawn in general about which set of models will better match the triangle count data.

1.7 Conclusion

The original Chung-Lu model is a simple model that has the nice property that the expected degree is given by the input degree sequence. However, we have shown that when changes are made to the original model (i.e. excluding self-edges) and the constraints of the model do not hold then this property no longer holds. In the case where we exclude self-edges this difference can be quite small. However, when the degree sequence violates the constraint this difference can be quite large. We have also examined in-depth an algorithm that approximates the original Chung-Lu model and shown how this approximation performs both under ideal circumstances and less than ideal circumstances. It is important to know how much this approximation can effect any instance created using either Model II or Model IV. In particular, we must be aware that when the degree constraint is violated then the expected degree can in fact be quite different than the original degree sequence. We also looked at another important network property, triangle counts, and noted that the Chung-Lu model is not designed to match this property and typically underestimates the number of triangles in networks we are interested in. In the next Chapter we will look at some ways to try and improve the $\mathcal{O}(m)$ Chung-Lu model and decrease the approximation error.

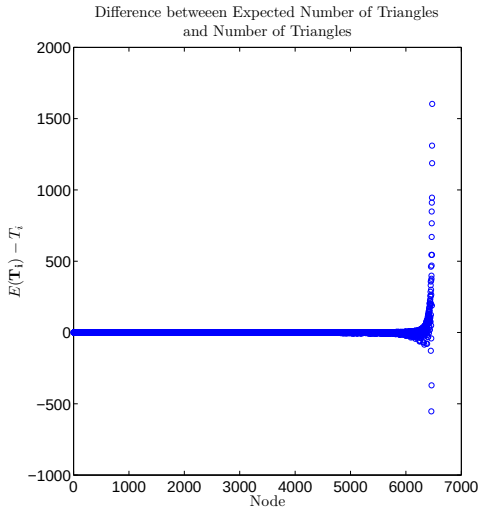


Figure 1.22: Autonomous System Network: Comparing Expected Number of Triangles from Models V and VI and Actual Number of Triangles.

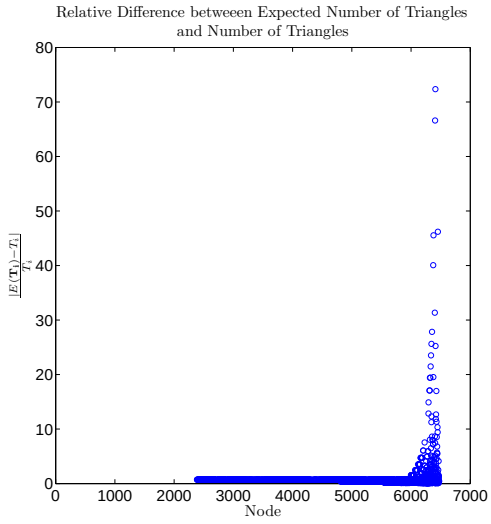


Figure 1.23: Autonomous System Network: Relative difference between Expected Number of Triangles from Models V and VI and Actual Number of Triangles.

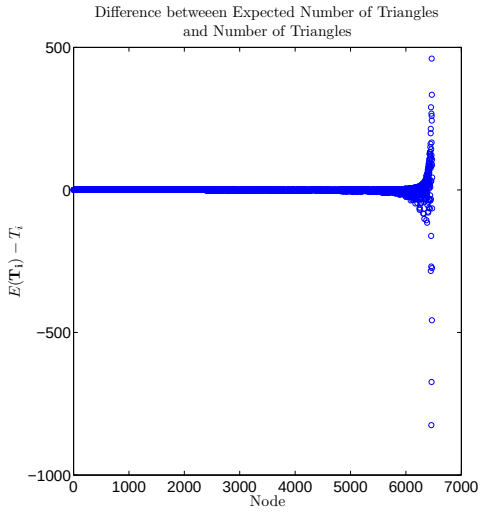


Figure 1.24: Autonomous System Network: Comparing Expected Number of Triangles from Models II and IV and Actual Number of Triangles.

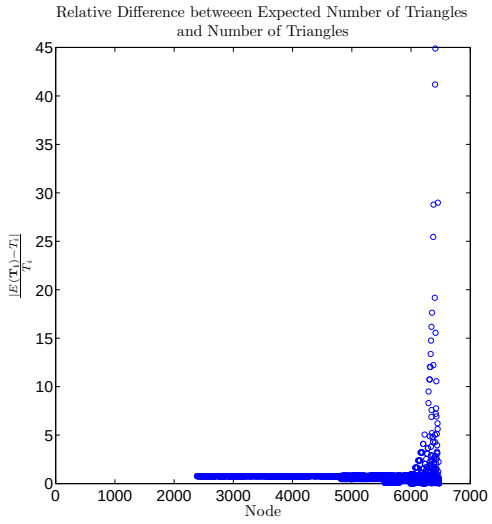


Figure 1.25: Autonomous System Network: Relative difference between Expected Number of Triangles from Models II and IV and Actual Number of Triangles.

Chapter 2

Improving the $\mathcal{O}(m)$ Chung-Lu model

2.1 Introduction

Consider the $\mathcal{O}(m)$ Chung-Lu model without self-edges, Model IV, as described in Section 1.4.2. Algorithm 4 is used to generate instances of this model and often these instances are referred to as instances of the original Chung-Lu model, Model I. As we have mentioned previously, in the original Chung-Lu model, the expected degree of node i is equal to the input degree, k_i, and the expected number of of edges is equal to m, the edge count in the original network. However, as we saw in the previous Chapter, the expected degree of node i and the expected edge count can be much different under Model IV than in Model I, and thus the instances that are referred to as representations of the original Chung-Lu model can have fairly different properties than what is assumed. Given this, the next step in our analysis is to look at simple ways of modifying Model IV to better approximate the original Chung-Lu model, Model I. In our discussion we only consider Model IV since this is the model most used in practice (i.e. Algorithm 4 is most often used to generate instances of the "Chung-Lu" model). In Section 2.2, we examine how increasing the number of draws can change the model. In Section 2.3 we create an algorithm that first uses Algorithm 4 and then looks at the resulting instance to determine any additional edges needed. In Section 2.4 we examine what happens if we let the probabilities in Equation (1.40) have a more general form (i.e. $p_{ij} \neq \frac{k_i k_j}{(2m)^2}$). This results in a constrained optimization problem that is difficult to solve and we turn to fixed point methods for an approximate solution. In Sections 2.2, 2.3 and 2.4, we assume that the degree constraint is not violated. In Section 2.5 we look at the improvements introduced in the previous sections when the constraint $k_i^2 < 2m \; \forall \; i$ is violated.

2.2 Drawing Additional Edges

As we mentioned previously, we are interested in modifying Model IV to improve the approximation error. For Model IV, the probability of edge (i, j) being in the graph after m draws is given by

$$p_{ij}(m) = \begin{cases} 1 - (1 - \frac{k_i k_j}{4m^2})^m & i \neq j, \\ 0 & i = j, \end{cases} \tag{2.1}$$

where $p_{ij}(m) \approx \frac{k_i k_j}{2m}$ but $p_{ij}(m)$ is strictly less than $\frac{k_i k_j}{2m}$. For now let us assume that the input degree sequence does not violate the constraint, $k_i^2 < 2m \ \forall \ i$, and thus $\frac{k_i k_j}{2m} \leq 1$. If we could increase the probability, $p_{ij}(m)$ such that it was closer to $\frac{k_i k_j}{2m}$ then $E(\mathbf{D}_i)$ would be be closer to k_i and $E(\mathbf{M})$ would better approximate m. One way to increase this probability is to draw more edges. So instead of drawing m edges we draw ω edges and Algorithm 4 becomes the following:

Algorithm 5: $\mathcal{O}(\omega)$ Chung-Lu Algorithm without Self-Edges.

```
for k = 1 to ω do
    Draw node i with probability k_i/(2m);
    Draw node j with probability k_j/(2m);
    /* Add edge (i, j) to the graph */
    if i ≠ j then
        a_ij = a_ji = 1;
    end
end
```

The probability of edge (i, j) being in the graph after ω draws is given by

$$p_{ij}(\omega) = \begin{cases} 1 - \left(1 - \frac{k_i k_j}{2m^2}\right)^\omega & i \neq j, \\ 0 & i = j. \end{cases} \tag{2.2}$$

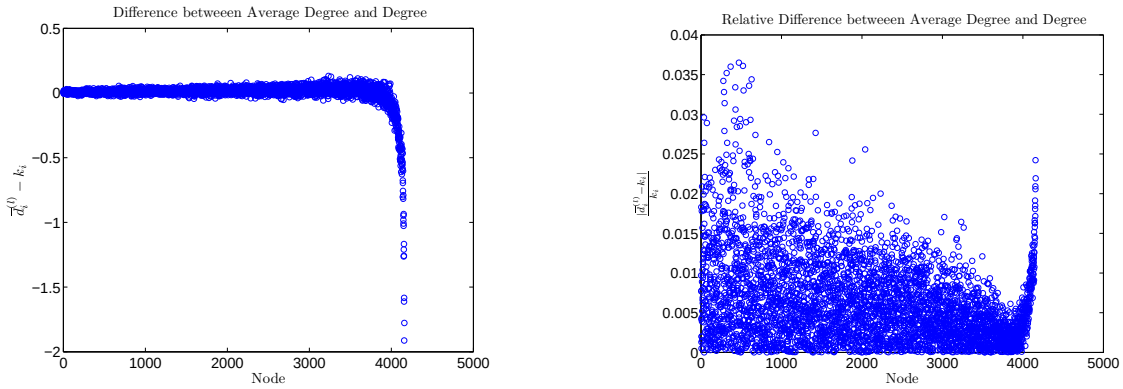
If we replace m with ω in Equation (1.23) and substitute in $p_{ij} = \frac{k_i k_j}{(2m)^2}$, then we get that $p_{ij}(\omega) \approx \frac{w}{m} \cdot \frac{k_i k_j}{2m}$ with $p_{ij}(\omega)$ strictly less than $\frac{w}{m} \cdot \frac{k_i k_j}{2m}$. Since $\omega > m$ this implies that $p_{ij}(\omega)$ can equal $\frac{k_i k_j}{2m}$ if we choose the correct ω; however, we have only one parameter, ω, and we want to match $\frac{n(n-1)}{2}$ probabilities. In fact, for certain values of ω, $p_{ij}(\omega)$ may be larger than $\frac{k_i k_j}{2m}$ which implies that $E(\mathbf{D}_i)$ may be greater than k_i. So, how many edges do we draw? One solution is to draw as many edges as necessary to get m edges in the resulting instance of the graph. This process is described in Algorithm 6.

Algorithm 6: A Matching Edge Count Chung-Lu Algorithm (No Self-Edges).

$\delta \leftarrow 1$;
while $\delta \leq m$ **do**
  Draw node i with probability $\frac{k_i}{2m}$;
  Draw node j with probability $\frac{k_j}{2m}$;
  `/* Add edge` (i, j) `to the graph */`
  if $i \neq j$ *and* $a_{ij} = a_{ji} = 0$ **then**
    $a_{ij} = a_{ji} = 1$;
    $\delta \leftarrow \delta + 1$;
  end
end

Note that if we use Algorithm 6 to generate an instance of our network and it requires γ draws (γ iterations of the while loop) to get m edges then the underlying model can be described by Equation (2.2) with $\omega = \gamma$ and this instance could have been created using Algorithm 5 with $\omega = \gamma$. Thus, if we use Algorithm 6 to generate a series of instances of the network each instance may actually represent a different model since each instance may be generated with different values of ω and each value of ω represents a different model. Although, if ω is approximately the same for all instances then we can assume each instance is drawn from approximately the same model. Although the underlying model may not be exactly the same for all instances if we use Algorithm 6 to generate our synthetic graphs these instance do have the nice property that they all have the same number of edges as in the original graph. Given this we use the general relativity network and Algorithm 6 to generate $l = 10000$ synthetic graphs. The average value of ω is 13 502.5 and the standard deviation is 9.01. Figure 2.1 plots the difference and the relative difference between the average node degree and the actual degree. Figure 2.2 plots the difference between the average node degree and the expected node degree from Model IV. We can see from these figures that for large degree nodes adding edges moves the average degree closer to the actual degree relative to Model IV; however, the average degree still remains below the actual degree. To better capture these differences we assume that these instances were all generated using Algorithm 5 with ω equal to the average across all 10000 instances (i.e. $\omega = 13502.5$). For this model, we plot the difference between the expected degree and the actual degree and the difference between the expected degree and the expected degree from Model IV in Figures 2.3 and 2.4, respectively. The same general pattern is seen in these Figures as in Figures 2.1a and 2.2. We should note that for approximately 90% of the nodes the expected degree under this model is actually higher than the actual degree.

Algorithm 6 has the nice property that ensures that every instance has exactly m edges but every instance can potentially be generated from a different model. As an alternative, we could use Algorithm



(a) $l = 10000$ (b) $l = 1000$

Figure 2.1: General Relativity Collaboration Network: Difference and Relative difference between Average Degree from Algorithm 6 and Actual Degree.

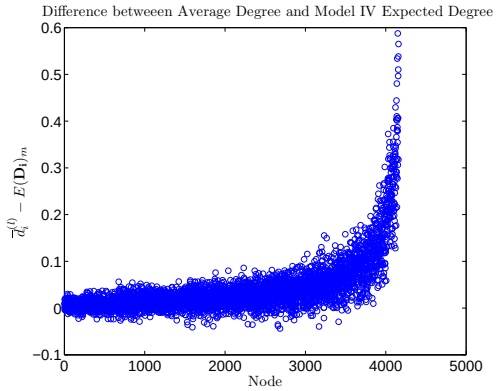


Figure 2.2: General Relativity Collaboration Network: Comparing Average Degree from Algorithm 6 and Expected Degree from Model IV.

5 with ω chosen so that $E(\mathbf{M}) = m$. The formula for $E(\mathbf{M})$ is given by the following

$$\begin{aligned} E(\mathbf{M}) &= \frac{1}{2}\sum_i E(\mathbf{D}_i) \\ &= \frac{1}{2}\sum_i \sum_{j\neq i} E(\mathbf{A}_{ij}) \\ &= \frac{1}{2}\sum_i \sum_{j\neq i} p_{ij}(\omega) \\ &= \frac{1}{2}\sum_i \sum_{j\neq i} 1 - \left(1 - \frac{k_i k_j}{2^2}\right)^{\omega} \end{aligned} \tag{2.3}$$

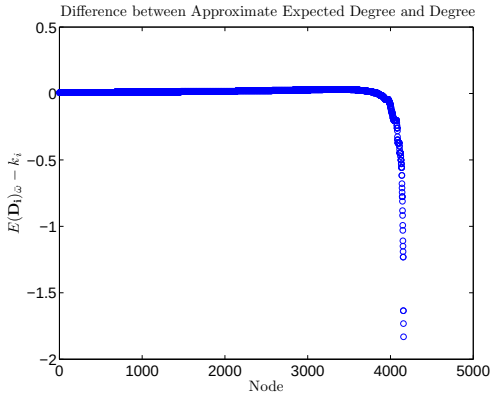


Figure 2.3: General Relativity Collaboration Network: Comparing Expected Degree from Algorithm 5 with $\omega = 13502.5$ and Actual Degree.

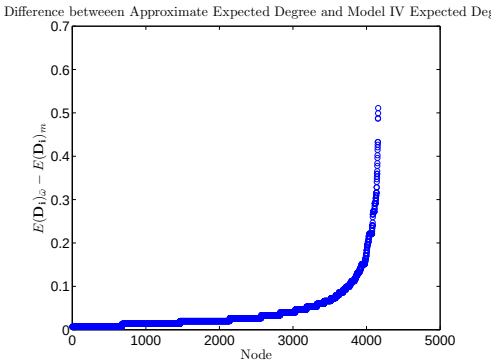


Figure 2.4: General Relativity Collaboration Network: Comparing Expected Degree from Algorithm 5 with $\omega = 13502.5$ and Expected Degree from Model IV.

We can set $E(\mathbf{M})$ equal to m and solve for ω numerically. This gives $\omega = 13511.56 \approx 13512$. As an alternative, we could choose ω to match the degree of the highest degree node. The formula for $E(\mathbf{D}_i)$ is given by

$$\begin{aligned} E(\mathbf{D}_i) &= \sum_{j \neq i} E(\mathbf{A}_{ij}) \\ &= \sum_{j \neq i} p_{ij}(\omega) \\ &= \sum_{j \neq i} 1 - \left(1 - \frac{k_i k_j}{2m^2}\right)^{\omega} \end{aligned} \tag{2.4}$$

We can set $E(\mathbf{D}_i)$ equal to k_i where i is chosen to be the largest degree node and solve for ω numerically. For the general relativity network this gives $\omega = 13832.54 \approx 13833$. Finally, we consider matching all

$\frac{n(n-1)}{2}$ probabilities (i.e. Find ω such that $p_{ij}(\omega) = \frac{k_i k_j}{2m}$, $i = 1, \ldots, n,\ j = i, \ldots, n.$) In this case we have $\frac{n(n-1)}{2}$ equations and only one unknown. So the resulting ω is the least-squares solution to this overdetermined system. Since probabilities are given by

$$p_{ij}(\omega) = 1 - \left(1 - \frac{k_i k_j}{2m^2}\right)^{\omega}, \tag{2.5}$$

where $i = 1, \ldots, n,\ j = i \ldots, n$, our system of equations is described by

$$\begin{aligned} \frac{k_i k_j}{2m} &= 1 - \left(1 - \frac{k_i k_j}{2m^2}\right)^{\omega} \\ \left(1 - \frac{k_i k_j}{2m^2}\right)^{\omega} &= 1 - \frac{k_i k_j}{2m} \\ \omega \ln\left(1 - \frac{k_i k_j}{2m^2}\right) &= \ln\left(1 - \frac{k_i k_j}{2m}\right), \end{aligned} \tag{2.6}$$

for $i = 1, \ldots, n,\ j = i \ldots, n$ where the following must hold: $\frac{k_i k_j}{2m} < 1\ \forall\ i, j \neq i$. If we place all $\frac{n(n-1)}{2}$, $\ln\left(1 - \frac{k_i k_j}{4m^2}\right)$ terms in the vector $\mathbf{a}$ and all $\frac{n(n-1)}{2}$, $\ln\left(1 - \frac{k_i k_j}{2m}\right)$ terms in the vector $\mathbf{b}$ we can write the system as $\mathbf{a}\omega = \mathbf{b}$ and the least squares solution is $\omega = \frac{\mathbf{a}^T\mathbf{b}}{\mathbf{a}^T\mathbf{a}}$. The least squares solution for the general relativity network is $\omega = 13720.66 \approx 13721$. For the three different ω outlined above, we use the general relativity network to plot the difference between the expected degree and the actual degree and the difference between the expected degree from Model IV in Figures 2.5 to 2.10. We can see from these figures that as we increase ω in order to match the node degree for the higher degree nodes we overshoot on the node degrees for the smaller degree nodes.

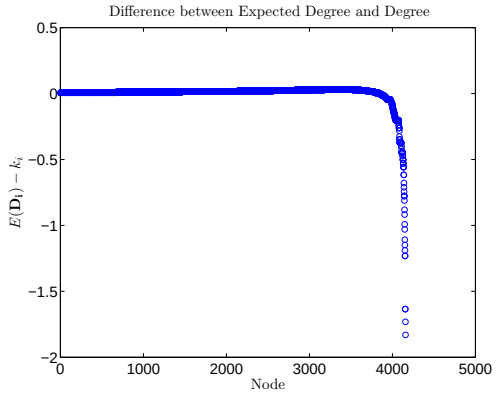


Figure 2.5: General Relativity Collaboration Network: Comparing Expected Degree from Algorithm 5 with $\omega = 13511.56$ and Actual Degree.

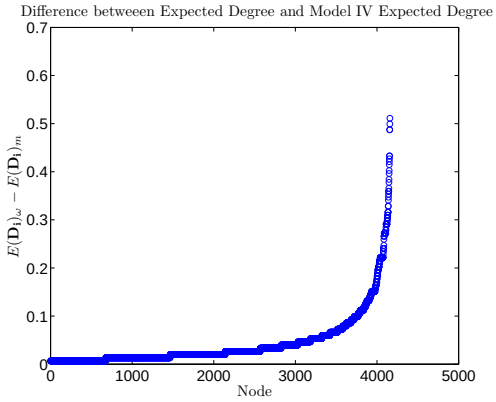


Figure 2.6: General Relativity Collaboration Network: Comparing Expected Degree from Algorithm 5 with $\omega = 13511.56$ and Expected Degree from Model IV.

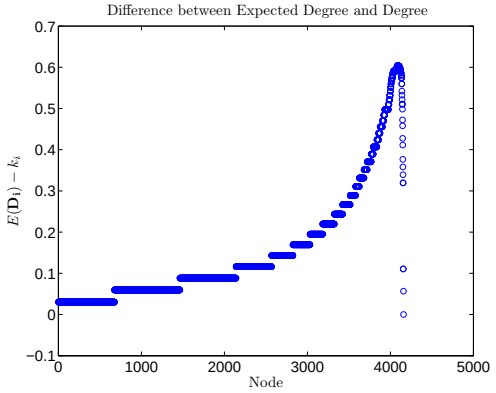


Figure 2.7: General Relativity Collaboration Network: Comparing Expected Degree from Algorithm 5 with $\omega = 13832.54$ and Actual Degree.

2.3 Two Distributions Algorithm

As the above analysis indicates, simply drawing more edges to add to our graph doesn't necessarily get us any closer to matching the degree distribution. Given this, we consider an alternative approach. We begin by using Algorithm 4 to build an initial instance of the graph. Then using this instance we create a "remaining degree" distribution for each node based on the remaining number of edges needed to match the degree of each node. So, for each node we calculate the following

$$r_i = k_i - d_i, \tag{2.7}$$

where k_i is the degree of node i in the original graph and d_i is the degree of node i after running Algorithm 4. For the nodes where $r_i < 0$, we set those values equal to zero, so in fact the formula for r_i is given by

$$r_i = \max\{k_i - d_i, 0\}, \tag{2.8}$$

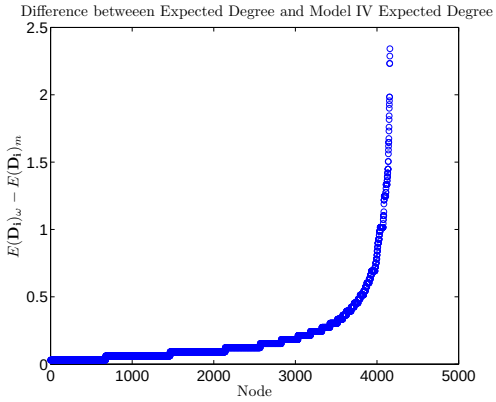


Figure 2.8: General Relativity Collaboration Network: Comparing Expected Degree from Algorithm 5 with $\omega = 13832.54$ and Expected Degree from Model IV.

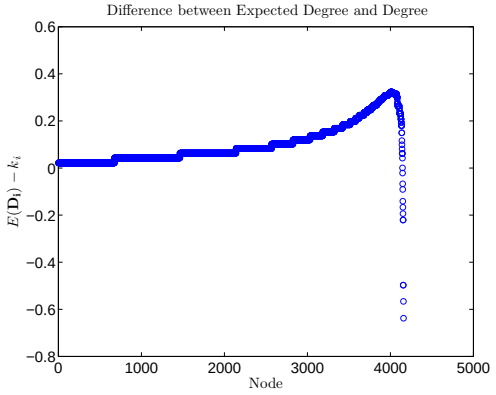


Figure 2.9: General Relativity Collaboration Network: Comparing Expected Degree from Algorithm 5 with $\omega = 13720.66$ and Actual Degree.

and we define $m_r = \frac{1}{2}\sum_i r_i$. We then run Algorithm 4 again but this time with node i drawn with probability $\frac{r_i}{2m_r}$ and node j drawn with probability $\frac{r_j}{2m_r}$ and the number of draws equal to m_r. For this model, we have

$$p_{ij}(m_r) = 1 - (1 - \frac{r_i r_j}{2m_r^2})^{m_r} \approx \frac{r_i r_j}{2m_r} \tag{2.9}$$

If we assume that $p_{ij}(m_r) = \frac{r_i r_j}{2m_r}$ then the expected degree of node i in this model, the remaining degree model, is given by

$$E(\mathbf{D}_i^r) = r_i - \frac{r_i^2}{2m_r} \approx r_i = k_i - d_i. \tag{2.10}$$

If these two graphs were independent then the expected degree of node i in the combined graph would just be the sum of the expected degrees however these graphs are not independent. An edge in Model IV can appear in the remaining degree model and vice versa. Thus the expected degree is not simply the

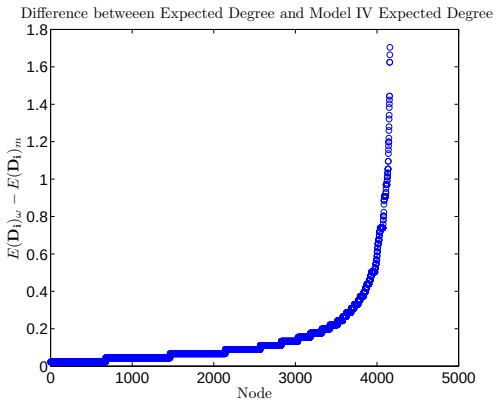


Figure 2.10: General Relativity Collaboration Network: Comparing Expected Degree from Algorithm 5 with $\omega = 13720.66$ and Expected Degree from Model IV.

sum of expected degrees in the two models and cannot be easily determined. However, given this lack of independence, we do know that the expected degree in the combined graph will be less than the sum of the expected degrees from the two models and thus is still strictly less than k_i. Thus, unlike the case of just adding edges the expected degree can not overshoot the actual degree. We should note that the remaining degree distribution is based on the particular instance of the graph created using Algorithm 4 and thus will be different every time Algorithm 4 is run. The entire algorithm is summarized in Algorithm 7. Given the lack of independence of the two graph models in Algorithm 7 and the dependence of the remaining degree distribution on the realization of the first graph model this makes it difficult to analyze the actual model underlying any instance created using Algorithm 7 (i.e. finding the probability of edge (i, j) being in the graph after running the algorithm) thus we analyze Algorithm 7 by using instances created from real networks. Again, we use the general relativity network and generate $l = 10000$ instances of the graph using Algorithm 7. Figure 2.11 plots the difference between the average degree from $l = 10000$ instances created using Algorithm 7 and the actual degree, while 2.12 plots the difference between the average degree from $l = 10000$ instances created using Algorithm 7 and the expected degree under Model IV. From 2.12 we can see that Algorithm 7 improves upon Model IV however it does not do as good as just simply adding edges.

Algorithm 7: Two Distributions Chung-Lu Algorithm without Self-Edges.

for $k = 1$ *to* m **do**
  Draw node i with probability $\frac{k_i}{2m}$;
  Draw node j with probability $\frac{k_j}{2m}$;
  `/* Add edge` (i, j) `to the graph */`
  if $i \neq j$ **then**
    $a_{ij} = a_{ji} = 1$;
  end
end
for $i = 1$ *to* n **do**
  Calculate $d_i = \sum_i a_{ij}$;
  Calculate $r_i = \max\{k_i - d_i, 0\}$;
end
Calculate $m_r = \frac{1}{2}\sum_i r_i$;
for $k = 1$ *to* m_r **do**
  Draw node i with probability $\frac{r_i}{2m_r}$;
  Draw node j with probability $\frac{r_j}{2m_r}$;
  `/* Add edge` (i, j) `to the graph */`
  if $i \neq j$ **then**
    $a_{ij} = a_{ji} = 1$;
  end
end

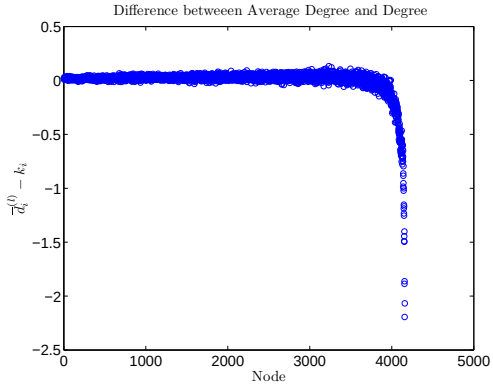


Figure 2.11: General Relativity Collaboration Network: Comparing Average Degree from Algorithm 7 and Actual Degree.

2.4 Optimal Probabilities

In Model IV, networks are generated by independently drawing m edges. The probability of drawing edge (i, j) on a single draw is denoted p_{ij} where $\sum_i \sum_j p_{ij} = 1$. The probability of an edge occurring between

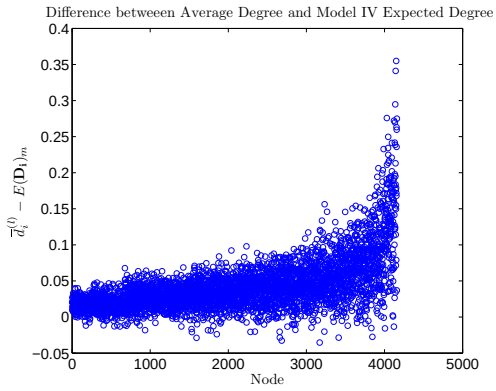


Figure 2.12: General Relativity Collaboration Network: Comparing Average Degree from Algorithm 7 and Expected Degree from Model IV.

nodes i and j where $i \neq j$ in the resulting graph is given by

$$\begin{aligned} p_{ij}(m) &:= P(\text{There is an edge between nodes } i \text{ and } j \text{ in the graph after } m \text{ draws}) \\ &= 1 - (1 - p_{ij} - p_{ji})^m, \ i \neq j, \end{aligned} \tag{2.11}$$

and the probability of a self-edge is given by

$$\begin{aligned} p_{ii}(m) &:= P(\text{There is a self-edge at node } i \text{ in the graph after } m \text{ draws}) \\ &= 0, \end{aligned} \tag{2.12}$$

by design. The expected degree of each node is computed as follows,

$$E(\mathbf{D}_i) = E\left(\sum_j \mathbf{A}_{ij}\right) = \sum_j E(\mathbf{A}_{ij}), \tag{2.13}$$

where $E(\mathbf{A}_{ij}) = p_{ij}(m)$ if $i \neq j$ and $E(\mathbf{A}_{ii}) = 2p_{ii}(m)$. This gives us

$$\begin{aligned} E(\mathbf{D}_i) = E\left(\sum_j \mathbf{A}_{ij}\right) &= \sum_j E(\mathbf{A}_{ij}) \\ &= \sum_{j \neq i} p_{ij}(m) \\ &= \sum_{j \neq i} 1 - (1 - p_{ij} - p_{ji})^m. \end{aligned} \tag{2.14}$$

In Model IV, we assume that on each edge draw, the probability of choosing edge (i, j) is equal to $\frac{k_i k_j}{4m^2}$ and $p_{ij} = p_{ji}$. However, what if we assumed a more general form for this probability (i.e we no longer assume it is a function of node degrees). In order for the algorithm to remain $\mathcal{O}(m)$ we need the nodes to be drawn independently so we assume a general form for the probability of choosing edge (i, j) where this

assumption holds. Let λ_i be the probability of choosing node i and let λ_j be the probability of drawing node j where $\sum_i \lambda_i = 1$ then $p_{ij} = \lambda_i \lambda_j$ and the expected degree can be written as

$$E(\mathbf{D}_i) = \sum_{j \neq i} 1 - (1 - 2\lambda_i \lambda_j)^m. \tag{2.15}$$

How do we determine λ_i, $i = 1, \ldots, n$? One way is to match the expected degree of each node to the degree in the actual network where k_i is the degree of node i in the actual network. Thus we want to find λ_i, $i = 1, \ldots, n$ such that

$$\begin{gathered} k_i = \sum_{j \neq i} 1 - (1 - 2\lambda_i \lambda_j)^m, \;\; i = 1, \ldots, n \\ \sum_i \lambda_i = 1, \;\; 0 \leq \lambda_i \leq 1 \;\; \forall \, i \end{gathered} \tag{2.16}$$

where n is the number of nodes in the network. If we ignore the constraint, $\sum_i \lambda_i = 1$, the we can write the above as a system of n nonlinear equations and n unknowns:

$$F(\boldsymbol{\lambda}) = \mathbf{0}, \tag{2.17}$$

where

$$F_i(\boldsymbol{\lambda}) = k_i - \sum_{j \neq i} 1 - (1 - 2\lambda_i \lambda_j)^m, \;\; i = 1, \ldots, n \tag{2.18}$$

Note that we can include the constraint in the nonlinear system by defining $\lambda_1 = 1 - \sum_{i=2}^n \lambda_i$ and then substituting out λ_1. Then we have a nonlinear system of n equations and $n - 1$ unknowns, an overdetermined system.

We can also set this up as a constrained optimization problem:

$$\min_{\boldsymbol{\lambda}} \|F(\boldsymbol{\lambda})\| \;\; s.t. \;\; \sum_i \lambda_i = 1, \;\; 0 \leq \lambda_i \leq 1 \;\; \forall \, i \tag{2.19}$$

where $\|F(\boldsymbol{\lambda})\|^2 = \sum_{i=1}^n |F_i(\boldsymbol{\lambda})|^2$.

Unfortunately, the optimization problem in 2.19 can be difficult to solve. So our goal is not to find the solution to the optimization problem. Instead, we develop an iterative method to find λ_i such that $\|F(\boldsymbol{\lambda})\|$ is lower than $\|F(\boldsymbol{\lambda})\|$ with $\lambda_i = \frac{k_i}{2m} \; \forall \, i$. We begin by developing a fixed point method to solve $F(\boldsymbol{\lambda}) = 0$.

Consider the equation for a single node, $F_i(\boldsymbol{\lambda}) = 0$, which can be re-written as follows,

$$\begin{aligned}
F_i(\boldsymbol{\lambda}) &= 0 \\
k_i - \sum_{j \neq i} 1 - (1 - 2\lambda_i\lambda_j)^m &= 0 \\
k_i - \sum_{j \neq i} \left[1 - \sum_{p=0}^{m} \binom{m}{p} (1)^p (-2\lambda_i\lambda_j)^{m-p} \right] &= 0 \\
k_i - \sum_{j \neq i} \left[1 - \left(1 + m(-2\lambda_i\lambda_j) + \sum_{p=2}^{m} \binom{m}{p} (1)^p (-2\lambda_i\lambda_j)^{m-p} \right) \right] &= 0 \\
k_i - \sum_{j \neq i} \left[2m\lambda_i\lambda_j - \sum_{p=2}^{m} \binom{m}{p} (1)^p (-2\lambda_i\lambda_j)^{m-p} \right] &= 0 \\
k_i - 2m\lambda_i \sum_{j \neq i} \lambda_j - \sum_{j \neq i} \sum_{p=2}^{m} \binom{m}{p} (1)^p (-2\lambda_i\lambda_j)^{m-p} &= 0 \\
2m\lambda_i \sum_{j \neq i} \lambda_j &= k_i - \sum_{j \neq i} \sum_{p=2}^{m} \binom{m}{p} (1)^p (-2\lambda_i\lambda_j)^{m-p} \\
\lambda_i &= \frac{k_i - \sum_{j \neq i} \sum_{p=2}^{m} \binom{m}{p} (1)^p (-2\lambda_i\lambda_j)^{m-p}}{2m \sum_{j \neq i} \lambda_j},
\end{aligned} \tag{2.20}$$

and from which we can get the following fixed point method

$$\lambda_i^{(q+1)} = \frac{k_i - \sum_{j<i} \sum_{p=2}^{m} \binom{m}{p} (1)^p (-2\lambda_i^{(q)}\lambda_j^{(q+1)})^{m-p} - \sum_{j>i} \sum_{p=2}^{m} \binom{m}{p} (1)^p (-2\lambda_i^{(q)}\lambda_j^{(q)})^{m-p}}{2m \left(\sum_{j<i} \lambda_j^{(q+1)} + \sum_{j>i} \lambda_j^{(q)} \right)}, \tag{2.21}$$

where we can force $\lambda_i^{(q+1)} = \lambda_i^{(q)}$ if the $\lambda_i^{(q+1)}$ given by Equation (2.21) violates the constraint $0 \leq \lambda_i^{(q+1)} \leq 1$. Using Equation (2.21) along with the constraints on λ_i, we propose the method outlined in Algorithm 8 to find an improved solution for $\boldsymbol{\lambda}$. If the constraints on $\lambda_i^{(q+1)}$ are never violated then Algorithm 8 will find a solution to $F(\boldsymbol{\lambda}) = \mathbf{0}$, however, we have no guarantee that $\sum_i \lambda_i = 1$ and by normalizing at the end we no longer have $F(\boldsymbol{\lambda}) = \mathbf{0}$; however, the idea is that we should get a better solution (i.e. smaller $|||F(\boldsymbol{\lambda})||$) than if $\lambda_i = \frac{k_i}{2m}$.

Algorithm 8: Algorithm For Finding Optimal $\boldsymbol{\lambda}$

Input: $\boldsymbol{\lambda}^{(0)}$
$q = 0$;
Evaluate $\|F(\boldsymbol{\lambda}^{(0)})\|$;
while $\|F(\boldsymbol{\lambda}^{(q)})\| > \epsilon$ **do**
  for $i = 1$ *to* n **do**
    $\lambda_i^{(q+1)} = \frac{k_i - \sum_{j<i}\sum_{p=2}^{m}\binom{m}{p}(1)^p(-2\lambda_i^{(q)}\lambda_j^{(q+1)})^{m-p} - \sum_{j>i}\sum_{p=2}^{m}\binom{m}{p}(1)^p(-2\lambda_i^{(q)}\lambda_j^{(q)})^{m-p}}{2m\left(\sum_{j<i}\lambda_j^{(q+1)} + \sum_{j>i}\lambda_j^{(q)}\right)}$;
    if $\lambda_i^{(q+1)} < 0$ *or* $\lambda_i^{(q+1)} > 1$ **then**
      $\lambda_i^{(q+1)} = \lambda_i^{(q)}$;
    end
  end
  $q = q + 1$;
end
$\sigma_{\boldsymbol{\lambda}} = \sum_i \lambda_i^{(q)}$;
for $i = 1$ *to* n **do**
  $\overline{\lambda}_i = \frac{\lambda_i^{(q)}}{\sigma_{\boldsymbol{\lambda}}}$;
end

To test our algorithm we once again use the general relativity graph. We set $\lambda_i^{(0)} = \frac{k_i}{2m}\ \forall\ i$ and $\epsilon = 10^{-7}$. The algorithm requires 12 iterations to reach the desired accuracy and the constraints on λ_i are never violated. However, before we normalize at the end of the algorithm we find $\sum_i \lambda_i = 1.0034$. Let $\overline{\boldsymbol{\lambda}}$ denote the optimal solution. We find that $\|F(\overline{\boldsymbol{\lambda}})\| = 4.656$ and $\|F(\boldsymbol{\lambda}^{(0)})\| = 9.876$. The maximum value of F_i, which is the difference between the actual degree of node i and the expected degree of node i goes from 2.342 to 0.534. However, the minimum value of F_i actually increases from $3.72e^4$ to $6.78e^{-3}$ and if we examine the expected number of edges then we find that with $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{(0)}$ the expected number of edges is 13333.55 but with $\boldsymbol{\lambda} = \overline{\boldsymbol{\lambda}}$ the expected number of edges is 13331.57 where the actual number of edges in the graph is 13422. It appears that choosing a different $\boldsymbol{\lambda}$ is better able to match the expected degree to the actual degree for higher degree nodes but at the cost of matching the expected degree to the actual degree for lower degree nodes. Figure 2.13 plots the difference between the expected degree using $\overline{\boldsymbol{\lambda}}$ and the actual degree while Figure 2.14 plots the difference between the expected degree with $\lambda_i = \frac{k_i}{2m}\ \forall\ i$ and the actual degree.

2.5 Constraint Violation

We now turn to the case where the constraint, $k_i^2 \leq 2m,\ \forall i$, is violated and look at how the different methods to improve the $\mathcal{O}(m)$ Chung-Lu model perform in this case. We begin by revisiting Algorithm 6. Using Algorithm 6 we generate $l = 10000$ instances of the autonomous system network. Note that this network has several nodes that violate the degree constraint. Figure 2.15 plots the difference and the relative difference between the average node degree and the actual degree. Figure 2.16 plots the difference between the average node degree and the expected node degree from Model IV. As with the general relativity network, where all the degree constraints hold, increasing the number of draws to match

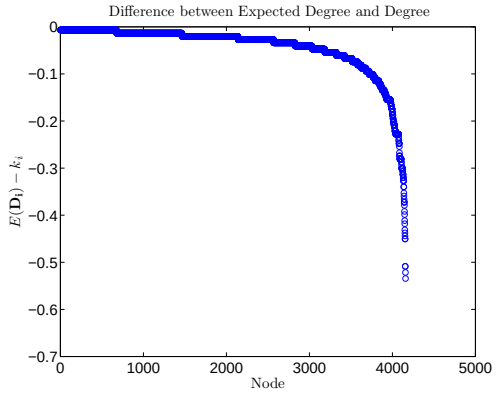


Figure 2.13: General Relativity Network: Comparing Expected Degree with $\overline{\boldsymbol{\lambda}}$ and Actual Degree.

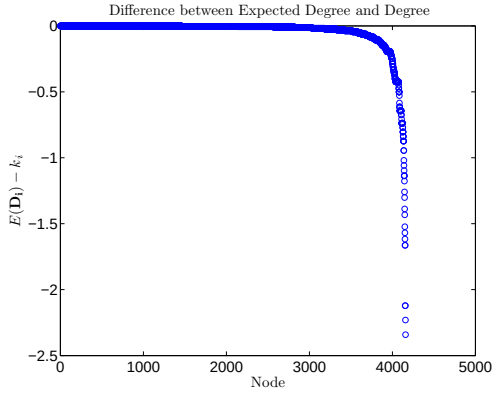


Figure 2.14: General Relativity Collaboration Network: Comparing Expected Degree from Model IV $\left(\lambda_i = \frac{k_i}{2m}\ \forall\ i\right)$ and Actual Degree.

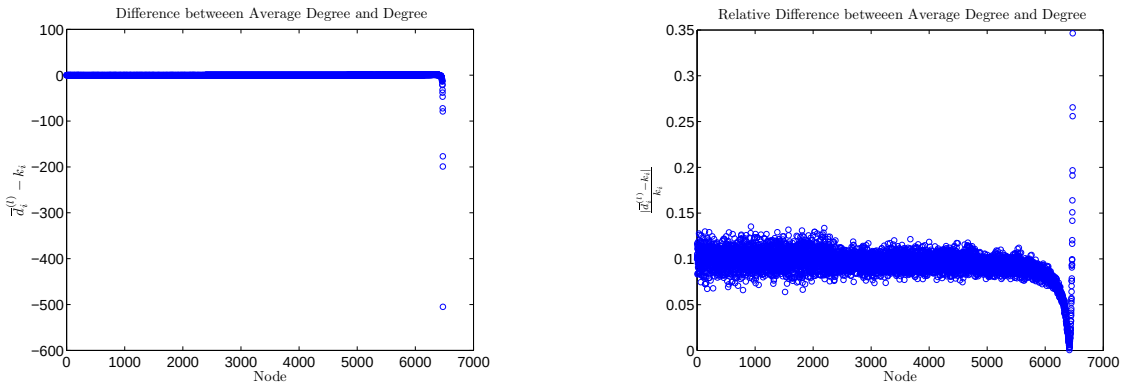
the edge count in the original network improves the degree distribution relative to Model IV, however, in this case the difference between the average degree distribution and the actual degree distribution is still quite large. To discuss why this may be the case we revisit Algorithm 5, since each instance of the network generated by Algorithm 6 could be generated from Algorithm 5 with the appropriate ω. Note that in the model described by Algorithm 5, the probability of edge (i, j) being in the graph after ω draws is given by Equation (2.2) even if the degree constraint is violated. We mentioned previously that $p_{ij}(\omega)$ is strictly less than $\frac{\omega}{m}\frac{k_i k_j}{2m}$. In networks where the degree constraint is violated, we can refine this as $p_{ij}(\omega) = \min\{1, \frac{\omega}{m}\frac{k_i k_j}{2m}\}$. For each node the expected degree is just the sum of these probabilities (i.e. $E(\mathbf{D}_i) = \sum_{j \neq i} p_{ij}(\omega)$). Ideally, we want $p_{ij}(\omega)$ to contribute $\frac{k_i k_j}{2m}$ to the expected degree (Note that in the case where self-edges are not allowed, we actually want this contribution to be larger). However, if $\frac{k_i k_j}{2m} > 1$ then the contribution from $p_{ij}(\omega)$ is 1 and the other $p_{ij}(\omega)$ must make up this difference. If this difference is large this may require ω to be large however we don't want to increase ω too much as that will increase the total number of edges in the graph. In the autonomous system graph, the largest degree

node has degree 1458. For this node, if we look at all the nodes j where $\frac{k_i k_j}{2m} > 1$ and look at the sum of the difference between $\frac{k_i k_j}{2m}$ and 1 we get

$$\sum_{j \in J} \left(\frac{k_i k_j}{2m} - 1 \right) = 641.83 \tag{2.22}$$

where J is the set of all nodes where $\frac{k_i k_j}{2m} > 1$. This sum is over a third of the total node degree for this node and thus the remaining $p_{ij}(\omega)$ have to contribute this amount to $E(\mathbf{D}_i)$ which explains why even increasing the number of draws may not be enough to match the degree distribution. Also note that by increasing $p_{ij}(\omega)$ to match the expected degree for some nodes will increase the expected degree above the actual degree for other nodes.

Returning to the autonomous system network we note that the instances generated using Algorithm 6 had an average ω of 13890.12 and a standard deviation of 41.875. Compare this with the actual number of edges in the network which is 12572. So for this network, on average we have to draw over 1000 more edges to match the total number of edges. Figures 2.17 and 2.18 plot the results from Algorithm 5 with $\omega = 13890.12$ and highlight the patterns found in Figures 2.15 and 2.16.



(a) $l = 10000$ (b) $l = 1000$

Figure 2.15: Autonomous System Network: Difference and Relative difference between Average Degree from Algorithm 6 and Actual Degree.

Our next step is to analyze the autonomous system network and Algorithm 5 with ω chosen to match certain expected values. We begin by finding ω to match the expected number of edges with the actual number of edges in the network ($E(\mathbf{M}) = m$). For the autonomous system network, we use the formula for $E(\mathbf{M})$ given by Equation (2.3) and set it equal to m. We then solve this equation numerically to find $\omega = 13889.77 \approx 13890$. This is approximately the same ω as the average ω calculated from the instances generated by Algorithm 6. Thus, Figures 2.17 and 2.18 can be used to show the difference between the expected degree and the real network and the expected degree and the expected degree from Model IV. We do note that under this model, the expected degree of the highest degree node is approximately 879, well under the actually degree of 1458. Given this we could also choose ω to match the expected degree of the largest degree node. In this case, for the autonomous system network, we find $\omega = 23937.66 \approx 23938$. This is almost double the number of edges in the original graph and the expected number of edges under

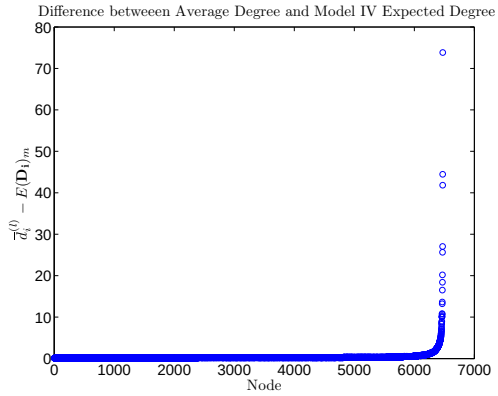


Figure 2.16: Autonomous System Network: Comparing Average Degree from Algorithm 6 and Expected Degree from Model IV.

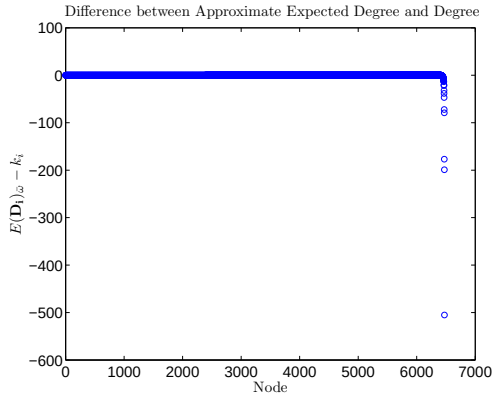


Figure 2.17: Autonomous System Network: Comparing Expected Degree from Algorithm 5 with $\omega = 13890.12$ and Actual Degree.

this model is 21085.51. Figures 2.19 and 2.20 plot the difference between the expected degree and the actual degree and the difference between the expected degree and the expected degree from Model IV, respectively. With $\omega = 23937.66$, the expected degree for a large number of nodes is much greater than the actual degree. This makes it an unattractive choice for ω in practice. The remaining case for ω that we previously examined is the case where ω is chosen to match the probabilities. The ω we found was the least squares solution to a system of equations. However, when the degree constraint is violated, $\ln(1 - \frac{k_i k_j}{2m})$ is not defined for some nodes i and j since $1 - \frac{k_i k_j}{2m} < 1$ for some nodes i and j. Thus we can not calculate this ω for the autonomous system network or any network where the degree constraint is violated.

Next, we use the autonomous system network to examine the two distributions model described by Algorithm 7. Using Algorithm 7, we generate $l = 10000$ instances of the network. Since we don't know the underlying model (i.e. we don't know the probability of edge (i, j) begin in the graph after running Algorithm 7), we can't analyze the model analytically. However, we can look at the average values for

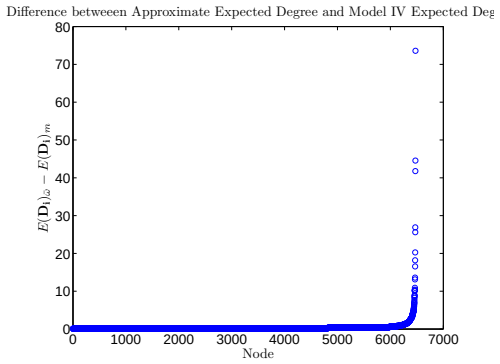


Figure 2.18: Autonomous System Network: Comparing Expected Degree from Algorithm 5 with $\omega = 13890.12$ and Expected Degree from Model IV.

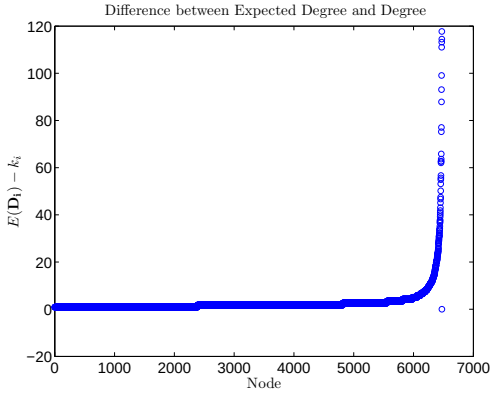


Figure 2.19: Autonomous System Network: Comparing Expected Degree from Algorithm 5 with $\omega = 23937.66$ and Actual Degree.

different properties calculated from the $l = 10000$ instances. Figure 2.21 plots the difference between the average degree and the actual degree and Figure 2.22 plots the difference between the average degree and the expected degree from Model IV. While, the average degree is still much lower than the actual degree for the larger degree nodes Algorithm 7 performs better than Algorithm 6 which can be seen by comparing Figures 2.22 and 2.18. Also note that the average number of edges calculate from the $l = 10000$ instances is 12412 which implies that unlike under Algorithm 6 we can still add edges after running Algorithm 7.

Finally, we use Algorithm 8 on the autonomous system network to find probabilities λ_i which will better match the expected degree to the actual degree. We set $\lambda_i^{(0)} = \frac{k_i}{2m}$ $\forall$ i and $\epsilon = 10^{-7}$. For the autonomous system network, the algorithm requires 58 iterations to reach the desired accuracy; however, once again the constraints on λ_i are never violated. Before we normalize at the end of the algorithm we find $\sum_i \lambda_i = 1.12$. For this network, $\|F(\boldsymbol{\lambda}^{(0)})\| = 694.25$ and $\|F(\overline{\boldsymbol{\lambda}})\| = 342.42$. The maximum value of F_i is given by the largest degree node and goes from 578.84 to 233.35 but the minimum value of F_i actually increases from 0.00327 to 0.2035. As well, we see the same pattern with the expected number

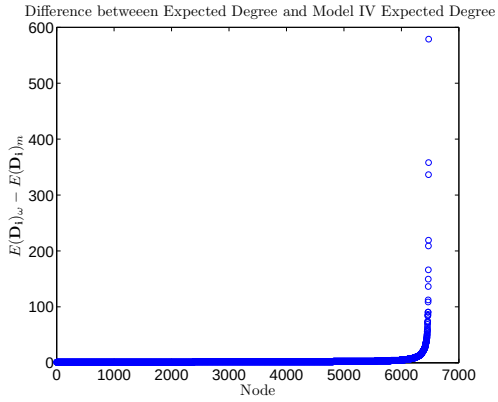


Figure 2.20: Autonomous System Network: Comparing Expected Degree from Algorithm 5 with $\omega = 23937.66$ and Expected Degree from Model IV.

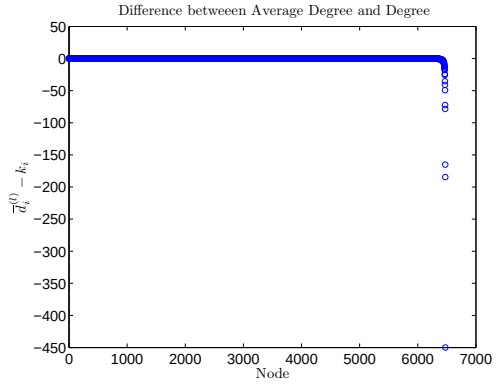


Figure 2.21: Autonomous System Network: Comparing Average Degree from Algorithm 7 and Actual Degree.

of edges as we saw with the general relativity network. The expected number of edges with $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{(0)}$ is 11429.50 but with $\boldsymbol{\lambda} = \overline{\boldsymbol{\lambda}}$ the expected number of edges is 10166.16 where the actual number of edges in the graph is 12572. Again, it appears that choosing a different $\boldsymbol{\lambda}$ is better able to match the expected degree to the actual degree for higher degree nodes comes at the cost of matching the expected degree to the actual degree for lower degree nodes. Figure 2.23 plots the difference between the expected degree using $\overline{\boldsymbol{\lambda}}$ and the actual degree while Figure 2.24 plots the difference between the expected degree with $\lambda_i = \frac{k_i}{2m}\ \forall\ i$ and the actual degree. Figure 2.25 plots the difference between the expected degree using $\overline{\boldsymbol{\lambda}}$ and the expected degree with $\lambda_i = \frac{k_i}{2m}$ which is the expected degree from Model IV. From this figure we can see that Algorithm 8 performs better than both Algorithm 7 and 6; however there is still a large difference between the expected degree and the actual degree for some nodes.

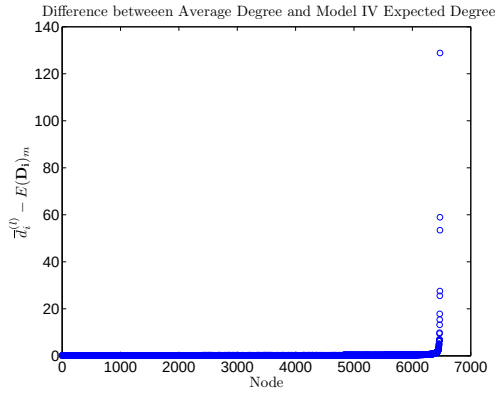


Figure 2.22: Autonomous System Network: Comparing Average Degree from Algorithm 7 and Expected Degree from Model IV.

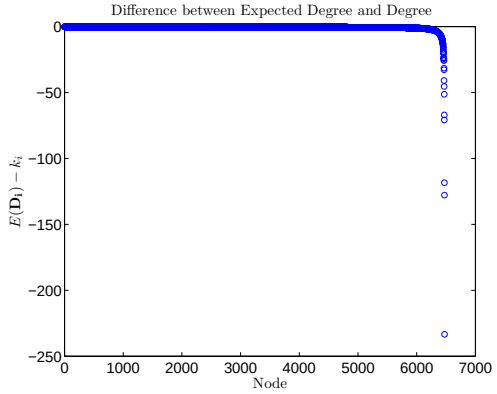


Figure 2.23: Autonomous System Network: Comparing Expected Degree with $\overline{\boldsymbol{\lambda}}$ and Actual Degree.

2.6 Conclusion

We examined some simple ways of improving the $\mathcal{O}(m)$ Chung-Lu model. In the case where the degree constraint is not violated, the $\mathcal{O}(m)$ Chung-Lu model, Model IV, may not be a bad approximation to begin with but by drawing more edges we can improve the degree distribution for some nodes although this comes at the expense of overestimating the degree distribution for other nodes. Using the two distributions algorithm, Algorithm 7, also improves the distribution, albeit by less than by simply adding edges for the network we examined, but does not overestimate the distribution of other nodes. When the degree constraint is violated drawing more edges again improves the degree distribution but is still well under the desired value for large degree nodes. Using Algorithm 7 in this case does a better job of matching the distribution and still underestimates the number of edges which allows for the possibility of adding more edges. Algorithm 8 underestimates the number of edges and performs better Algorithm 7 in both the base where the degree constraints hold and when they are violated. However, in the case where the constraint

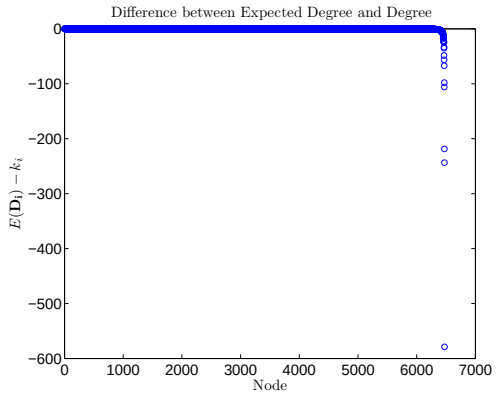


Figure 2.24: Autonomous System Network: Comparing Expected Degree from Model IV $\left(\lambda_i = \frac{k_i}{2m} \; \forall \; i\right)$ and Actual Degree.

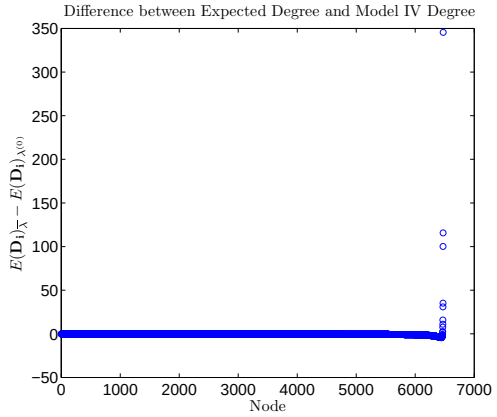


Figure 2.25: Autonomous System Network: Comparing Expected Degree with $\overline{\boldsymbol{\lambda}}$ and Expected Degree from Model IV.

on the input degree sequence is violated the modifications to the $\mathcal{O}(m)$ Chung-Lu model still leave large differences between the expected degree and the actual degree for some nodes. This suggests along with all our previous analysis that the Chung-Lu model, either the approximate model or the exact, either with or without self-edges, is not a good choice for a null model.

Bibliography

[1] William Aiello, Fan Chung Graham, and Linyuan Lu. A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66, 2001.

[2] 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.

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

[4] Nurcan Durak, Tamara G. Kolda, Ali Pinar, and C. Seshadhri. A scalable null model for directed graphs matching all degree distributions: In, out, and reciprocal. In *NSW 2013: Proceedings of IEEE 2013 2nd International Network Science Workshop*, pages 23–30, April 2013.

[5] 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.

[6] 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.

[7] Joel C. Miller and Aric A. Hagberg. Efficient generation of networks with given expected degrees. In Alan M. Frieze, Paul Horn, and Pawel Pralat, editors, *WAW*, volume 6732 of *Lecture Notes in Computer Science*, pages 115–126. Springer, 2011.

[8] Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random Struct. Algorithms*, 6(2-3):161–180, March 1995.

[9] Michael Molloy and Bruce Reed. The size of the giant component of a random graph with a given degree sequence. *COMBIN. PROBAB. COMPUT*, 7:295–305, 2000.

[10] M. E. J. Newman. *Networks: An Introduction*. Oxford University Press, New York, NY, USA, 2010.

[11] 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.

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

[13] Nicholas C. Wormald. The asymptotic connectivity of labelled regular graphs. *Journal of Combinatorial Theory, Series B*, 31(2):156 – 167, 1981.