Correcting Output Degree Sequences in Chung-Lu Random Graph Generation

Christopher Brissette[1,2], David Liu[1,3], and George M. Slota[1,4]

[1] Rensselaer Polytechnic Institute, Troy NY
[2] brissc@rpi.edu
[3] liud9@rpi.edu
[4] slotag@rpi.edu

Abstract. Random graphs play a central role in network analysis. The Chung-Lu random graph model is one particularly popular model, which connects nodes according to their desired degrees to form a specific degree distribution in expectation. Despite its popularity, the standard Chung-Lu graph generation algorithms are susceptible to significant degree sequence errors when generating simple graphs. In this manuscript, we suggest multiple methods for improving the accuracy of Chung-Lu graph generation by computing node weights which better recreate the desired output degree sequence. We show that each of our solutions offer a significant improvement in degree sequence accuracy.

Keywords: graph theory, random graphs, graph generation

1 Introduction

Random graph generation is an important task in several fields of study, such as biology and the social sciences. Random graphs arising from graph generation algorithms have uses as null models and as algorithmic benchmarks [14, 7, 10]. Stochastic block models are random graph models in which a number of nodes is predefined and each possible edge is assigned a probability of existing [12]. A random graph can then be constructed by generating edges with respect to these assigned probabilities. Conversely, the common configuration model [3, 15] assigns to each vertex some number of stubs, equal to each vertex's desired degree, and then selects two stubs uniformly at random to create an edge. This process is repeated until all stubs are attached, and a graph exactly matching some input degree sequence can be output.

The expected pairwise degree probabilities (probability of a vertex of degree x attaching to a vertex of degree y) arising from the configuration model may be expressed as a stochastic block model via the Chung-Lu random graph model [6]. This model assigns a weight w_v to each node $v \in V$ in the graph, and then it attaches nodes $u, v \in V$ according to the probability $p_{uv} = w_u w_v / \sum_{a \in V} w_a$. If each weight is taken to be the desired degree of each given node, then this model produces a desired degree distribution, but only in expectation. This model is used as a subroutine in more complex graph generation algorithms [13, 8, 17, 18],

and the given probabilities are also implicitly used to define network measures such as modularity for graph clusters [16, 9].

Despite its popularity and theoretical importance, Chung-Lu random graph generation often results in graphs with significant degree distribution errors [5, 13, 8, 10, 19]. While this can be resolved in many cases by using an explicit graph configuration model instead of Chung-Lu, these approaches have limited room for parallelism and are not scalable, particularly when a simple graph output is desired. An appealing feature of Chung-Lu graph generation is that the method is naïvely parallelized and is additionally expedited by techniques such as edge skipping [1, 2], even when generating simple graphs. Because of the algorithm's high scalability and wide usage, some work has been done to correct and quantify these errors. In Durak et al. [8], it is shown that Chung-Lu generation often under-estimates the number of low degree nodes, and they perform an artificial inflation in the number of nodes with unit weight to account for this. Other related work has used this specific approach [13] or a similar approach where unit weight nodes are instead manually configured [17]. Alternatively, in our prior work [5], we approximate the output degree distribution of Chung-Lu generation with a matrix equation, and we solve a linear system to determine an input distribution that will best generate the desired output. We also show that the inverse of many degree sequences with respect to this matrix yield vectors with negative entries. These vectors do not have a useful interpretation with respect to the Chung-Lu generation algorithm, and the method is greatly limited because of this. This manuscript aims to remedy the issues present in previous work while utilizing the same matrix model as an important building block.

In the Chung-Lu model, each set of nodes with the same weight w may be discussed as a block in a stochastic block model, and the degrees within that block should be approximately Poisson distributed about the weight w. Therefore, for degrees $\{w_1 = 1, w_2 = 2, \cdots, w_d = d\}$, one may generate a matrix $\mathbf{P}$ given by Equation 1. Each column of the matrix represents the probability mass function of degrees within each block. The inner product of any given row r of this matrix with a degree sequence vector adds together the predicted number of nodes with degree r produced by each block. By considering the entire matrix simultaneously, the output vector predicts the output of Chung-Lu. That is, by representing an input degree sequence as a vector x, one can approximate the output degree sequence of Chung-Lu generation y as $\mathbf{P}x = y$. As presented, $\mathbf{P}$ is of infinite dimension and needs to be reduced to a finite dimension for computational use. For this purpose, the matrix is truncated by removing all rows beyond some maximum degree. This cut off can be chosen depending on the desired output distribution y, and d may also be chosen to make P square, in which case an explicit inverse is known. For details regarding this analysis, a reader may consult the original citation [5].

$$\mathbf{P} = \begin{bmatrix} | & | & & | \\ poiss(w_1) & poiss(w_2) & \cdots & poiss(w_d) \\ | & | & & | \end{bmatrix} \tag{1}$$

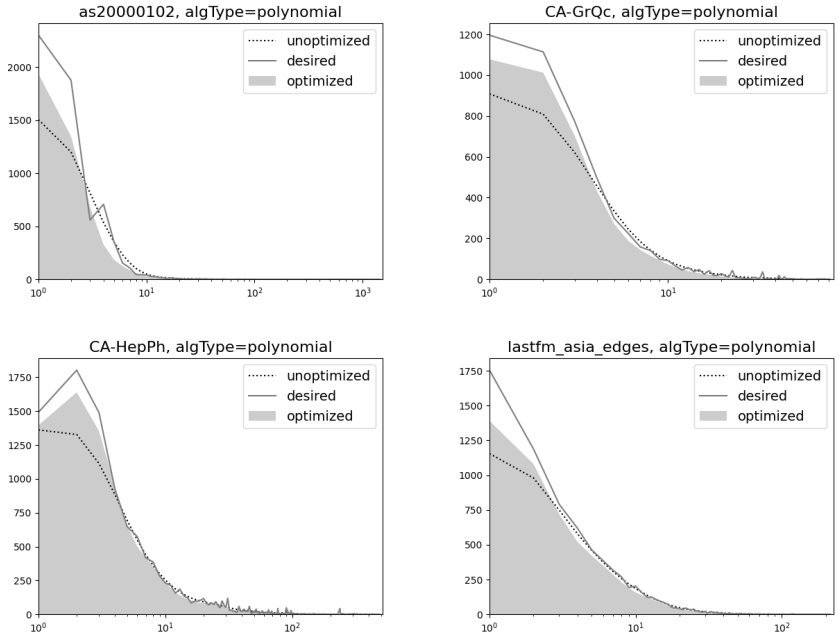

Fig. 1. Visualization of degree sequences: Comparisons of degree sequences for the as20, GrQc, HepTh, and lastfm graphs. The dotted lines denote the predicted output using standard Chung-Lu weights, the solid grey region denotes the output sequence using optimized Chung-Lu weights, and the solid line denotes the desired output sequence. Each optimization here was performed using our polynomial update method which is discussed in Section 2.

We note that this matrix can be easily generalized. By choosing a set of arbitrary positive weights $w = \{w_1, w_2, \cdots, w_d\}$, instead of simply the nodal degrees, one obtains a matrix $\mathbf{P}(w)$ where the means of each Poisson distribution correspond to the given weights. This produces a new stochastic block model.

Our Contribution: This paper focuses on determining, for a given number of weight parameters d, the set of weights such that the error between the desired output and actual output of Chung-Lu graph generation will be minimized. We develop and optimize several novel methods to minimize this error. Visualized in Figure 1 for several graphs in the Stanford Large Network Dataset Collection[5] are their degree sequences, the unoptimized output from Chung-Lu generation using these degree sequences, and the generation output after applying one of our methods. We will discuss our varying methods in Section 2 and analyze their results in Sections 3 and 4.

[5] https://snap.stanford.edu/data/

2 Methods

As we note in our prior work [5], there are numerous sequences that can not be reliably generated using naïve Chung-Lu weights. To remedy this short coming, there are two algorithm parameters which may be adjusted to alter the output. One parameter is the input sequence. This is the specific parameter studied in prior work. The other parameter is the set of weights $w = \{w_1, w_2, \cdots, w_d\}$. Conceptually, both methods are trying to approximate a distribution as a linear combination of Poisson distributions. In the former method, the Poisson distributions have means equal to the target degree classes, and the approximation is improved by altering the coefficients applied to each distribution. Alternatively, changing the weights equates to changing these means, effectively moving the Poisson distributions along the x-axis.

We present two methods incorporating weight alteration. Our first method relies on several greedy updates, where weights are chosen such that $\|\mathbf{P}(w)x - y\|$ is minimized at each step. The latter method uses maximum likelihood estimation [20] to solve for weights.

Before discussing either method, let us first formalize goals and definitions. Take $\mathbf{P}$(w) to be the square matrix given by weights $w = \{w_1, \cdots, w_d\}$ and removing both the first row and everything beyond row $d + 1$ in Equation 1. The first row is removed because it corresponds to the number of zero-degree nodes. These nodes may be ignored after generation, so removing the first row of $\mathbf{P}$(w) mathematically represents this. We call our input degree sequence vector $x = [x_1, \cdots, x_d]$ and our desired output degree sequence vector $y = [y_1, \cdots, y_d]$. Additionally, call the output of the Chung-Lu algorithm with weight set w and input vector x, $CL(w, x)$. Then, our goal is to find a combination w, x such that $\|CL(w, x) - y\|_1$ is minimized. The 1-norm is specifically considered because it can be directly interpreted as the number of nodes with incorrect degrees. A log_2-binned version of this error will additionally be considered later and is discussed in the Section 3.

2.1 Greedy updates

We first discuss the greedy update method. This is based off of a simple approximation and update loop. The basic idea is as follows. Given an input degree sequence x, determine the first k derivatives of each column of $\mathbf{P}$(w) with respect to their means and use these derivatives to approximate $\|\mathbf{P}(w + \epsilon)x - y\|$ for small perturbations in the elements of the mean-set $w + \epsilon$. Then, update the means in the optimal direction according to some minimization algorithm and repeat this process for some number of iterations.

Two update objectives are discussed in this section, which we call linear updates and polynomial updates. These objectives only differ in the number of derivatives considered. Linear updates approximate error based on the first derivative of each column in $\mathbf{P}$(w) . Alternatively, polynomial updates use an arbitrary number of k derivatives and the Taylor series to approximate error.

As is shown later, both of these methods reduce the per-node degree error significantly; however, they require different numbers of iterations. All instances of the polynomial-update variant use $k = 2$ in this manuscript. In Algorithm 1, we show a general template of the greedy method. The main difference in each of these methods comes from how our objective changes the `opt_E(·)` function. The objectives are discussed in more detail in the following subsections.

Algorithm 1 Greedy-Update (x,y,$\{w_1, \cdots, w_d\}$, δ,t)

1: $\mathbf{P} \leftarrow$ fill_P($\{w_1, \cdots, w_d\}$)
2: **for** iters $\in [1 \ldots \text{t}]$ **do**
3: $\quad \mathbf{U} = \{\mathbf{U}_1, \cdots, \mathbf{U}_k\} \leftarrow compute\_U\_set(\{w_1, \cdots, w_d\})$
4: $\quad \mathbf{E} \leftarrow$ opt_E($\mathbf{P}$, $\mathbf{U}$, x, y, δ)
5: $\quad \mathbf{P} \leftarrow$ fill_P($\{w_1 + E_{11}, \cdots, w_d + E_{dd}\}$)
6: **return** $\{w_1, \cdots, w_d\}$

Algorithm 1 is initialized with an input degree sequence vector x, a desired output degree sequence vector y, a set of initial weights $\{w_1, \cdots, w_d\}$, a maximum update step-size δ, and an iteration number t. For this paper, initial degree sequences are taken to be $x = cy$ for some positive constant $c \in \mathbb{R}^+$. Additionally, initial weights are taken to be $\{w_1 = 1, \cdots, w_d = d\}$. The iteration number and step size will vary depending on desired accuracy and whether linear, or polynomial updates are being used. The algorithm proceeds as follows. $\mathbf{P}$(w) is initialized with the input weights. Then, within the loop, a set of matrices $\mathbf{U} = \{\mathbf{U}_1, \cdots, \mathbf{U}_k\}$ is computed within the `compute_U_set(·)` function. Each matrix $\mathbf{U}_i$ corresponds to the i^{th} derivative of each column. These matrices are then used in the `opt_E(·)` function to determine how much each mean in w should change. For the purposes of this manuscript, `opt_E(·)` uses the sequential least squares minimization [4] implementation from `scipy.optimize.minimize(·)` in `Python`. Then new weights are computed and $\mathbf{P}$(w) is updated.

Linear Updates: Linear updates are the simpler of the two greedy update methods. In the linear update method, $k = 1$ and only a single $\mathbf{U}$ matrix is computed in `compute_U_set(·)`. This matrix has the same form given in Equation 2 and the columns take the form of the derivatives of the columns in $\mathbf{P}$(w) as given in Equation 3 with respect to their means. In Equation 2, μ_j corresponds to the mean of the Poisson distribution.

$$\mathbf{U} = \begin{bmatrix} | & | & & | \\ \frac{\partial}{\partial \mu_1} poiss(\mu_1, x) & \frac{\partial}{\partial \mu_2} poiss(\mu_2, x) & \cdots & \frac{\partial}{\partial \mu_m} poiss(\mu_m, x) \\ | & | & & | \end{bmatrix} \tag{2}$$

$$\frac{\partial}{\partial \mu_i} poiss(\mu_i, x) = \frac{(x - \mu_i) e^{-\mu_i} \mu_i^{x-1}}{x!} \tag{3}$$

The linear update objective function used in `opt_E(·)` takes the form of minimizing $\gamma = \|(\mathbf{P}(w) + \mathbf{UE})x - y\|_2$ with respect to the diagonal matrix $\mathbf{E}$, where each entry is bounded by δ, $|E_{jj}| \leq \delta$. Unfortunately, linear approximations lack significant accuracy, and as such, the step size δ needs to be rather small to maintain stability within each optimization step `opt_E(·)`. This ultimately leads to a method which requires many updates. This can be prohibitive for graphs with high maximum degree, since the dimensionality of our optimization problem depends on this. This issue is discussed in Section 4 at the end of the manuscript.

Polynomial Updates: The polynomial update method is very similar to the linear update method. In this method, higher order derivatives are considered in the Taylor series. This higher order error approximation is then used to predict degree sequence errors. The Taylor series approximation of the Poisson distribution is given by Equation 4.

$$poiss(z, x) = \frac{e^{-\mu}\mu^x}{x!} + \sum_{j=1}^{\infty} \left(\frac{\partial^j}{\partial \mu^j} poiss(\mu, z) \right) (z - \mu)^j \tag{4}$$

For a given number of derivatives k, a truncated series is used to make approximations. Note that the term on the left of the sum is an entry of the matrix $\mathbf{P}$(w) . Additionally, the right hand sum consists of two components, the j^{th} derivative, and a difference term. This allows us to rewrite this expression in terms of matrices as in Equation 5.

$$\mathbf{P}(w') \approx \mathbf{P}(w) + \sum_{j=1}^{k} \mathbf{U}_j \mathbf{E}_j \tag{5}$$

In Equation 5, $\mathbf{U}_j$ is the matrix corresponding to the j^{th} derivative of each column, similar to equation 2. $\mathbf{E}_j$ is a diagonal matrix with entries $E_j(a, a) = e_a^j$, corresponding to the step size in each dimension. In the polynomial update function, the error to be minimized is of the form $\gamma = \|(\mathbf{P}(w) + \sum_{j=1}^{k} \mathbf{U}_j \mathbf{E}_j)x - y\|_2$. Because of the increased accuracy of the polynomial method, a larger bound δ may be used for the step size. While we do not present bounds for this here, the size of δ can be chosen to be larger for larger instances of the number of derivatives k.

2.2 Maximum likelihood estimation

Maximum likelihood estimation (MLE) based clustering is a popular statistical method for determining probabilistic clusters for a data set [20]. Given a pre-defined type of statistical distribution (e.g. normal, binomial, Poisson, etc.) and a number of distributions m, MLE clustering determines the parameters and coefficients for those distributions such that their mixture distribution has the highest likelihood of generating the data set. While MLE is most commonly

used for clustering data, we instead use it here for function approximation. Consider the desired degree sequence y as a realization of a mixture distribution and the underlying statistical distributions as Poisson distributions. Then, the coefficients and means which are output as a mixture model from MLE may be interpreted as the input vector x and the μ values in $\mathbf{P}$(w) , respectively.

Algorithm 2 MLE-Update (m,y,d,iters)

1: x $\leftarrow [\frac{1}{m}, \cdots, \frac{1}{m}]$
2: $p_y \leftarrow \frac{y}{\|y\|_1}$
3: $\mu \leftarrow [\frac{d}{m}, \frac{2d}{m}, \cdots, d]$
4: (x,μ) $\leftarrow$ poiss_EM(x,μ,p_y,iters)
5: x $\leftarrow \|y\|_1 x$
6: **return** (x,μ)

Our MLE based method proceeds as follows, and is demonstrated in pseudocode in Algorithm 2. Begin by considering a desired output sequence y, a number of means m, and an interval $[0, d]$. Initialize a vector $x = [\frac{1}{m}, \cdots, \frac{1}{m}]$ and a vector of means $\mu = [\mu_1 = \frac{d}{m}, \mu_2 = \frac{2d}{m}, \cdots, \mu_m = d]$. Note that these means may be initialized randomly within the interval $[0, d]$, if desired. Then, normalize y to obtain a probability distribution $p_y = \frac{y}{\|y\|_1}$, from which points are sampled for maximum likelihood estimation. Maximum likelihood estimation is then run on these inputs, updating the entries of x and μ at each iteration. Once this has concluded, x is scaled by $\|y\|_1$ and each entry is rounded to the nearest natural number. This ensures that x now corresponds to the number of nodes instead of a proportion of all nodes.

As discussed earlier, there are two parameters which may be tuned when improving Chung-Lu graph generation. While our earlier work focused on changing the input sequence, and both the linear and polynomial methods focus on changing the means of Poisson distributions, Algorithm 2 simultaneously solves for both. Additionally, expectation maximization has a tune-able dimensionality. This means that one may take small samples from p_y, and consider fewer Poisson distributions to improve compute time. This is not an option that is readily available in the case of greedy linear and polynomial updates.

3 Results

In Figure 2, the three methods discussed in the previous section are compared against naïve Chung-Lu generation on a set of degree sequences from the Stanford Large Network Dataset Collection. Graph generation is performed using the `expected_degree_graph(·)` function from the `NetworkX` [11] package in `Python`.

As can be seen, each method outperforms naïve Chung-Lu by a considerable margin. However, our different methods perform better on different degree

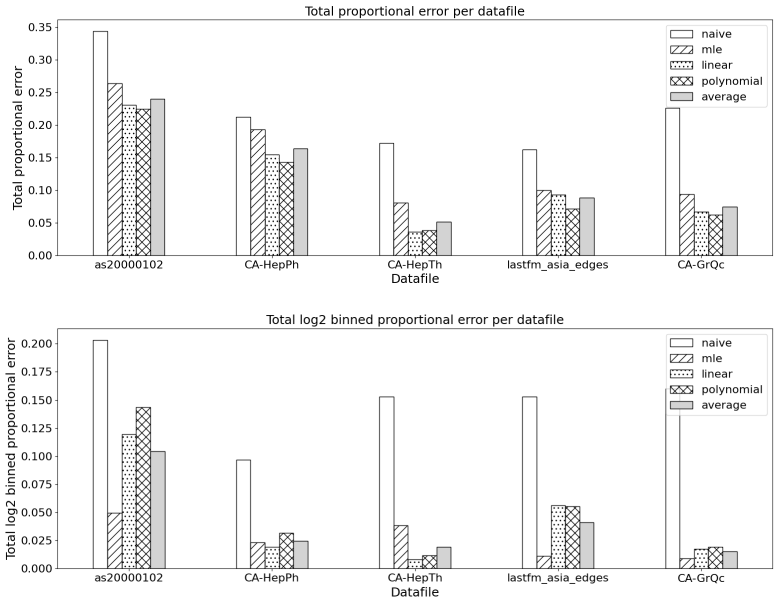

Fig. 2. Proportional errors: Degree error plots for all methods on a number of graphs. Both the proportional error (top), and log_2-binned proportional error (bottom) metrics are as described in the Results section. As is seen, every method drastically reduces the proportional L1 error of the degree sequence when compared with naïve Chung Lu. However, different methods perform better on differing degree sequences.

sequences. The exact reason for this requires further analysis. Figure 2 considers two different proportional error functions. The first one is L1 proportional error which is computed as the ratio $\|CL(w,x) - y\|_1 / \|y\|_1$. This can be directly interpreted as the proportion of nodes which have the correct degree. Additionally, one can interpret this error function as a normalized version of the total variation distance. The log_2-binned proportional error is also considered. In this case the sequences $CL(w,x)$ and y are partitioned into $b = \lceil log_2(d) \rceil$ bins, forming the sequences $\beta(CL(w,x))$ and $\beta(y)$, both of which are in $\mathbb{R}^b$. The entries of $\beta(CL(w,x))$ are $\beta(CL(w,x))_i = \sum_{j=2^{(i-1)}}^{2^{(i-1)}+2^i} CL(w,x)_j$, and the entries of $\beta(y)$ follow similarly. The proportional binned error is then computed as $\|\beta(CL(w,x)) - \beta(y)\|_1 / \|\beta(y)\|_1$. The reason for defining this error function is that there are many applications where the exact degrees are less important than simply having the correct number of "low-degree", or "high-degree" nodes. For this purpose, the log_2-binned proportional error provides a quantitative understanding of how many nodes are being generated for different "sections" of the sequence.

As is seen in Figure 3, the polynomial update method outperforms the other optimization methods in proportional error. Additionally, the MLE optimization method outperforms the others for log_2-binned proportional error. Conceptually,

this implies that the polynomial update method may be the best at matching the degrees of nodes exactly, while the MLE method is superior for approximate reproduction of sequences.

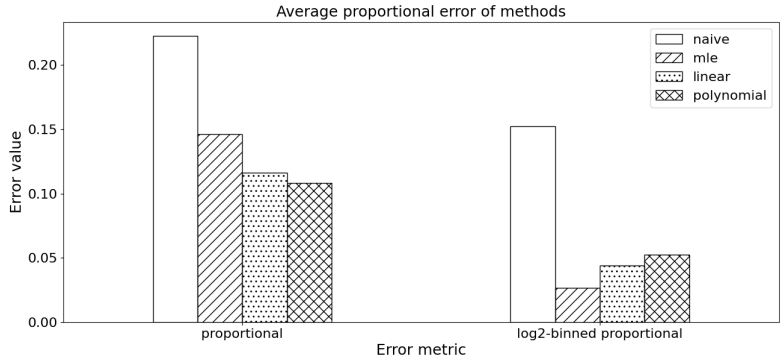

Fig. 3. Average proportional errors of methods: The proportional (left) and log2-binned proportional (right) errors are compared over all test graphs for each optimization method as well as naïve Chung-Lu. Both the proportional error, and log_2-binned proportional error metrics are as described in the Results section. As is seen, on average the polynomial update method results in the more significant reduction of proportional error, however the MLE method results in the largest reduction in log_2-binned proportional error.

4 Discussion

4.1 Parameters

When choosing parameters for Algorithm 1, a reader may be rightfully curious as to what constitutes a "good" choice. In Figure 4, a parameter search over several choices of iteration number t and constant c, such that $x = cy$ are shown for two example graphs from the Stanford Large Network Dataset Collection. As is seen, the error reaches similar levels for both the polynomial and linear update methods for different parameters. We note that $1.05 < c < 1.15$ appears to work best for both graphs. While not shown, this behavior is also seen a across many other degree sequences. Furthermore, the number of iterations required to achieve a similar error reduction with polynomial updates versus linear updates is seen to be considerably smaller. In fact, for these two graphs, a similar error reduction is seen with an order of magnitude fewer update steps.

There is significant work to be done deciding parameters. While Figure 4 suggests some best practices, it is far from definitive. Furthermore, the choice of step-size δ is currently somewhat arbitrary. In this paper, it is taken to be $0.05 \leq \delta \leq 0.2$ for linear updates, and $0.2 \leq \delta \leq 0.5$ for polynomial updates. Different step sizes drastically alter the stability and number of requisite iterations of the method. This requires further experimental and theoretical results for varying degree sequences.

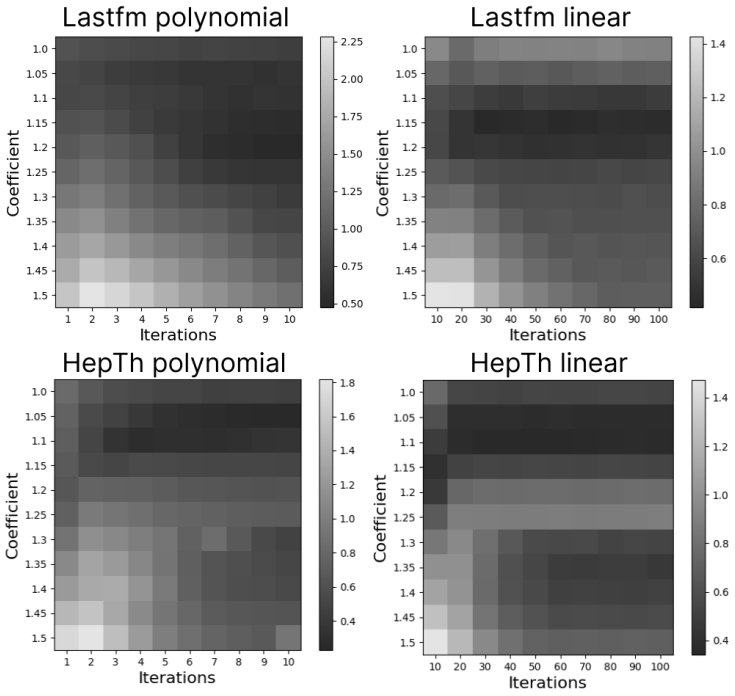

Fig. 4. A parameter search of error, varying the coefficient $c \in \mathbb{R}^+$ for $x = cy$, and the number of iterations for both the polynomial and linear update methods respectively. The polynomial update method in this case has $k = 2$. The colors indicate the proportional L1 error $\|CL(w, x) - y\|_1 / \|CL(y) - y\|_1$. As can be seen for the two sample graphs, the polynomial update method converges to a smaller proportional L1 error than the linear method does in the same number of iterations.

4.2 Timing considerations

The methods presented in this manuscript require varying times to run. The linear update method uses a miniscule step size, and as such requires many iterations to terminate. This is a significant concern when the maximum degree of the desired output is large. This is because the maximum degree controls the dimensionality of the optimization step, which must be performed at every iteration. To this end, the polynomial update method can iterate with a larger step size, requiring less iterations. However, in the case of a significantly large maximum degree, the optimization step may still not be practical. The MLE-method does not suffer from these same drawbacks, because the sample number and number of distributions may be tuned. This means the MLE method should not perform slower on larger degree sequences, given constant sample and distribution numbers.

In the case of the greedy update methods, a simple change can be made which drastically speeds up compute time. This is the method of truncation. Note that, for most real world degree sequences the vast majority of the weight

lies in the lowest degrees of the graph. Because of this, one may ignore a portion of the sequence when using either greedy update method. This drastically reduces compute time, but may introduce additional error. In our limited testing, removing the final 1% of the sequence by node count greatly improves run times and minimally affects error. Despite this, the best practice for truncation is an open problem.

5 Conclusion

In this manuscript, we presented two methods for improving the accuracy of Chung-Lu random graph generation. These methods consist of an iterative algorithm (Algorithm 1), which greedily updates the weights of nodes, and an algorithm (Algorithm 2) relying on maximum likelihood estimation. Both methods were shown to dramatically reduce degree sequence error in comparison to naïve Chung-Lu; however, they require different considerations. The greedy update methods suffer from long compute times in the case of sequences with high maximum degree, while the maximum likelihood method is significantly faster. While parameter choices for these algorithms are presented, a systematic study of their affect on resulting error is an avenue for further research.

References

1. Alam, M., Khan, M., Vullikanti, A., Marathe, M.: An efficient and scalable algorithmic method for generating large-scale random graphs. In: SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 372–383. IEEE (2016)
2. Batagelj, V., Brandes, U.: Efficient generation of large random networks. Physical Review E 71(3), 036113 (2005)
3. Bollobás, B.: A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. European Journal of Combinatorics 1(4), 311–316 (1980)
4. Bonnans, J.F., Gilbert, J.C., Lemaréchal, C., Sagastizábal, C.A.: Numerical optimization: theoretical and practical aspects. Springer Science & Business Media (2006)
5. Brissette, C., Slota, G.M.: Limitations of chung lu random graph generation. In: International Conference on Complex Networks and Their Applications. pp. 451–462. Springer (2021)
6. Chung, F., Lu, L.: The average distances in random graphs with given expected degrees. Proceedings of the National Academy of Sciences 99(25), 15879–15882 (2002)
7. Drobyshevskiy, M., Turdakov, D.: Random graph modeling: A survey of the concepts. ACM Computing Surveys (CSUR) 52(6), 1–36 (2019)
8. Durak, N., Kolda, T.G., Pinar, A., Seshadhri, C.: A scalable null model for directed graphs matching all degree distributions: In, out, and reciprocal. In: 2013 IEEE 2nd Network Science Workshop (NSW). pp. 23–30. IEEE (2013)
9. Fosdick, B.K., Larremore, D.B., Nishimura, J., Ugander, J.: Configuring random graph models with fixed degree sequences. SIAM Review 60(2), 315–355 (2018)

10. Garbus, J., Brissette, C., Slota, G.M.: Parallel generation of simple null graph models. In: The 5th IEEE Workshop on Parallel and Distributed Processing for Computational Social Systems (ParSocial) (2020)
11. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008)
12. Holland, P.W., Laskey, K.B., Leinhardt, S.: Stochastic blockmodels: First steps. Social networks 5(2), 109–137 (1983)
13. Kolda, T.G., Pinar, A., Plantenga, T., Seshadhri, C.: A scalable generative graph model with community structure. SIAM Journal on Scientific Computing 36(5), C424–C452 (2014)
14. Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., Alon, U.: Network motifs: simple building blocks of complex networks. Science 298(5594), 824–827 (2002)
15. Molloy, M., Reed, B.: A critical point for random graphs with a given degree sequence. Random structures & algorithms 6(2-3), 161–180 (1995)
16. Newman, M.E.: Modularity and community structure in networks. Proceedings of the national academy of sciences 103(23), 8577–8582 (2006)
17. Slota, G.M., Berry, J., Hammond, S.D., Olivier, S., Phillips, C., Rajamanickam, S.: Scalable generation of graphs for benchmarking HPC community-detection algorithms. In: IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC) (2019)
18. Slota, G.M., Garbus, J.: A parallel LFR-like benchmark for evaluating community detection algorithms. In: The 5th IEEE Workshop on Parallel and Distributed Processing for Computational Social Systems (ParSocial) (2020)
19. Winlaw, M., DeSterck, H., Sanders, G.: An in-depth analysis of the chung-lu model. Tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States) (2015)
20. Zaki, M.J., Meira Jr, W., Meira, W.: Data mining and analysis: fundamental concepts and algorithms. Cambridge University Press (2014)