Parallel Generation of Simple Null Graph Models

Jack Garbus and Christopher Brissette and George M. Slota

Rensselaer Polytechnic Institute, Troy, NY
garbuj@rpi.edu, brissc@rpi.edu, slotag@rpi.edu

March 15, 2020

***Abstract*—The generation of uniformly-random graphs is a key analytic tool for hypothesis testing throughout social network analysis. This work specifically optimizes the generation of large-scale *simple* uniform random graphs. We consider the separate but related problems of generating such a graph from an existing edge list and the problem of generating a graph from only a degree distribution. To address these problems, we implement an efficient parallel Markov Chain Monte Carlo process for *double-edge swapping*, a fast and parallel method for *edge-skipping* edge list generation, and a novel method to solve for valid inputs to our edge-skipping generator. Our double-edge swapping procedure is considerably faster than prior parallel methods, our edge generator uses state-of-the-art methods, and the algorithmic approach we have to solve for valid edge-skipping inputs addresses the often significant shortcomings of current approaches.**

***Index Terms*—graph generation; random graphs; null graph models**

I. Introduction

Uniformly-random *null graph models* find wide applicability within social network analytics and other related fields. Notable applications include: motif finding for subgraph-based analytics [23], where a *motif* is a subgraph that appears more frequently relative to in uniformly random graph; modularity maximization as a common approach for community detection applications [6], where *modularity* measures the "clustering" of a network relative to what's randomly expected; other measurements such as *assortativity* [26] are similarly calculated relative to an assumed null model.

There exists several different *spaces* for null graph models [16]. For this work, we specifically consider the category of *simple graphs* with undirected edges that match a degree distribution, though our results can be extrapolated to directed graphs with certain considerations [14], [15]. To facilitate graph studies, applications often use a null model from one space within the analytical context of a graph in a separate space. E.g., calculations for modularity and assortativity often use what we'll refer to as "Chung-Lu" [11] attachment probabilities[1]. While these pairwise probabilities fit analytically for *non-simple graphs* (graphs with multiple edges between the same vertex pair and self loops), they are still often applied in the simple graph space. We'll discuss this more in Section II.

A. The Problems with Current Methods: The naïve application of Chung-Lu probabilities to the simple graph space can have **extreme error** in approximating edge attachment probabilities or realizing a graph of a given degree distribution. See Figures 1 and 2, where we consider an autonomous systems interaction network (AS-733 from SNAP [20]). In Figure 1, we plot the Chung-Lu attachment probabilities of the largest degree vertex versus the same pairwise probabilities as sampled over 100 generated uniform random graphs. The given approximation fails dramatically in capturing the empirical curve. In fact, for a majority of pairwise degrees, the attachment probability as calculated exceeds 1. Likewise, using the same probabilities to generate a simple graph (e.g., via an *erased model* – to be discussed), also results in error in the final degree distribution. We show the error in output distribution versus degree in Figure 2.

While the probabilities between simple and non-simple graphs converge under certain assumptions (i.e., the probabilities of multi-edges and self loops approaches zero as the graph size increases to infinity), we and others have observed this assumption dramatically fails for real-world instances [16], [36]. This is especially apparent on small and relatively dense graphs with skewed degree distributions, such as tightly clustered social networks, biological interaction networks, and subgraphs or communities within larger networks. Researchers have in particular noticed that

[1]What we refer to as Chung-Lu probabilities show up equivalently in several other models, such as the configuration model [24]. In this instance, the Chung-Lu model and configuration model for non-simple graphs are equivalent in output but generally differentiated in practice by how edges are generated. We'll stick to referencing "Chung-Lu graphs" and "Chung-Lu probabilities" within this paper for simplicity.

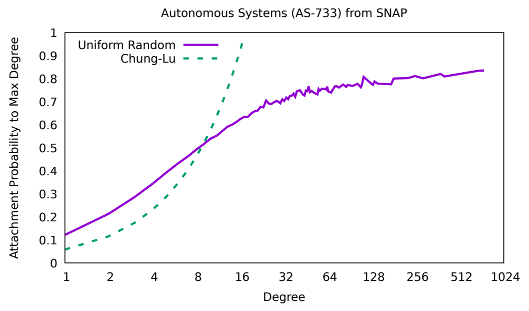

Fig. 1. Approximate (Chung-Lu) and empirical (Uniform Random) attachment probabilities between the largest degree vertex and all other vertex degrees for a null graph model generated from the AS-733 degree distribution.

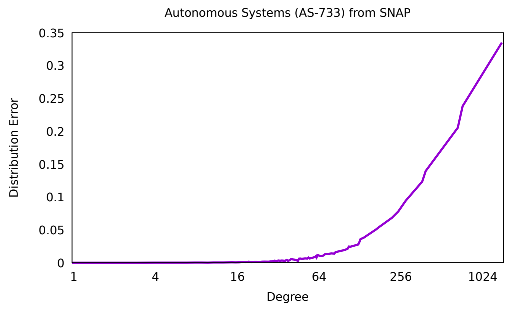

Fig. 2. Output error in the degree distribution when attempting to generate a null graph model using an erased configuration-based approach.

careful consideration is required when generating null graph models for motif finding [22].

B. Our Solutions: This work considers two closely related but still distinct problems. Here, a *uniformly random simple graph* is defined as a graph uniformly and randomly sampled from all possibly topologies within the simple graph space of a given degree distribution. The two distinct problems are as follows:

1) Generating a uniformly random simple graph from an existing edge list.
2) Generating a uniformly random simple graph from a degree distribution.

Our solution for the first problem involves the development of an efficient parallel algorithm to perform *double-edge swaps* while retaining edge simplicity. To our knowledge, no prior shared-memory parallel algorithm exists for this problem. Recent algorithms [5] take hours at the scale we consider. We engineer an algorithm for this purpose and experimentally validate its scalability and correctness.

Our solution for the latter problem involves the additional step to first generate an initial simple edge list. This edge list is **not** uniformly random[2], so we perform subsequent swaps on this edge list to retrieve an appropriately sampled graph. Our challenge here was to come up with *some* set of attachment probabilities that, while not representative of a uniformly random graph, will output *some* edge list that empirically matches our target input degree distribution. We develop a heuristical method to generate these probabilities, which can be input into a parallel *edge-skipping*-based [1], [4], [21] generator.

II. Background

A. Definitions: In this work, *simple* graphs refer to graphs containing vertices with no self loops or multiple edges (*multi-edges*) between any given vertex pair. *Non-simple* graphs can have such self loops or multi-edges. A *random graph* is a graph with randomly wired edges, usually matching some parameters such as a number of vertices and/or edges, a degree distribution, and/or some specified generative process. A *null graph model* is a random graph selected uniformly from the space of all possible configurations that could arise from the specified generative process or the set of parameters. The *edge list* of a graph is simply a listing of its edges each defined by an i, j vertex pair. A graph's *degree distribution* is defined by the number of vertices having a given degree over all unique degrees within the graph.

B. Related Work: There are many ways described in the literature to create uniform random graphs with a given degree distribution [2], [14], [16], [22], [24], [28]. While the generation of non-simple graphs is straightforward, simple graph generation is considerably more challenging. For configuration models, one can simply continue to re-generate graphs from scratch until a simple graph is output (*repeated configuration model*). However, as we'll discuss, the expected number of multi-edges for skewed graphs often exceeds one, which makes the probability of selecting such a simple graph low. As we demonstrated, one can also discard any multi-edges and self loops (*erased configuration model* [8]) at a cost to accuracy in the output degree distribution; naïvelyre-generating or attempting to re-configure discarded edges introduces inaccuracy in the output degree distribution, bias in attachment probabilities, and might not even be feasible under the existing configuration [36].

The only current practical method for generating uniformly random simple graphs is by taking an existing edge list and performing some number of *double edge swaps* [2]. Double edge swaps are defined as taking two

[2]Methods for directly generating such an edge list have no closed-form solutions for attachment probabilities, and tend to be computationally hard. We know of no existing implementations.

edges $e = \{u, v\}, f = \{x, y\}$ and swapping endpoints to create $g = \{u, x\}, h = \{v, y\}$ or $g = \{u, y\}, h = \{v, x\}$. Such a swap will retain the existing degree distribution, and graph simplicity can be maintained if each new edge is evaluated against the existing edge list. Performing some number of randomly selected swaps as a Markov Chain Monte Carlo process produces a uniformly sampled graph after sufficient *mixing*. While there currently exist no known bounds on the number of swaps necessary for uniform mixing in the general case [13], empirical studies [27] and this work show that mixing time is often not computationally prohibitive.

C. Generating Simple Chung-Lu-like Graphs: The goal of this work is to efficiently generate large-scale uniformly random simple graphs given some existing edge list or an input degree distribution. We avoid configuration-based approaches as they are difficult to parallelize. Random edge-generative models such as Chung-Lu have existing and straightforwardly parallel approaches for non-simple graphs, though don't exactly match an input degree distribution. Winlaw et al. [36] has a fairly exhaustive analysis of some of the computational difficulties encountered when attempting to use Chung-Lu as a generator for simple graphs. We summarize the Chung-Lu method and these difficulties below.

The Chung-Lu model itself considers some list W of n vertex weights corresponding to n vertices in some target graph as

$$W = \{w_1, w_2, \ldots, w_n\}$$

with the attachment probabilities of an edge between vertex i with weight w_i and vertex j with weight w_j as

$$P_{i,j} = \frac{w_i w_j}{2m}$$

where $2m$ is the weight sums

$$2m = \sum_{i}^{n} w_i$$

To generate a uniformly random loopy multi-graph with some target degree distribution, one can set each weight to a corresponding vertex degree and make $2m$ biased draws with replacement from $i = 1 \ldots n$ based on these weights. The resultant list of draws

$$E = \{r_1, r_2, \ldots r_{2m}\}$$

can be considered as an edge list of m edges when taking each pair of $\{r_i, r_{i+1}\}$ as a single undirected edge. This is known to be the $O(m)$ model, and it can be computed in an embarrassingly parallel fashion. The aforementioned challenges of creating a simple graph from this approach apply.

Correspondingly, as with Erdős-Rényi graphs, there also exists what is termed as a Bernoulli or $O(n^2)$ model. In this model, all $\frac{n(n-1)}{2}$ undirected edge pairs between all possible i and j vertices are evaluated based on the $P_{i,j}$ given above and a coin flip. As each edge is only considered once, this model is guaranteed to output a simple graph. This can also be implemented in an embarrassingly parallel fashion, though with quadratic work complexity. However, the technique of *edge-skipping* [4], [21] allows the Bernoulli model to be implemented with $O(m)$ work complexity. We use a recent parallel implementation of edge-skipping [33] for this purpose and discuss it in Section IV-B.

As was demonstrated in Figures 2 and 1, considerable error in an output degree distribution occurs for dense or skewed graphs, where $w_i \times w_j > 2m$ for multiple i, j vertex pairs. As such, Winlaw et al. [36] and numerous others [8], [30], [35] have looked at making "corrections" to these probabilities via adjusting the weights. Unfortunately, even with expensive fixed point methods to compute some optimal set of corrected weights, the probabilities are still not representative of a uniformly random or properly mixed graph. For many degree distributions, there does not even exist a set of weights that will optimally solve the problem [36]. Generalized random graphs have a similar interpretation [29] but, again, solving for a set of valid weights is deceptively non-trivial.

Luckily, the idea of the Bernoulli model with edge skipping can be generalized, where we don't need to assume the closed form for $P_{i,j}$ as a function of two vertex weights. We can simply compute some $P_{i,j}$ for each i, j pair. This allows us to use these methods to output a biased graph that will match the target degree distribution in expectation. By uniformly mixing the edges of this graph via double-edge swaps, we can then achieve our desired end product of a graph randomly selected from the simple null graph space of the specified degree distribution. We'll discuss our approach in detail in the next section.

D. Other Models: The study of null graph models and random graphs with fixed degree sequences has been a theoretical research area for many decades. For space considerations, we are unable to properly summarize the field here. Instead, we point the interested reader to the recent and relatively comprehensive review by Fosdick et al. [16].

III. Generating from an Existing Edge List

We first consider the problem of generation a null graph model from an existing edge list. We take an existing edge list and perform some number of parallel double edge swaps while retaining edge simplicity and ensuring the output is a suitable random sample. As noted, it is unknown exactly how many swaps are required to ensure proper mixing of a MCMC double edge swap process, though in practice we observe attachment probabilities close to a "steady-state" once all edges have successfully swapped at least once by chance. The challenge of creating a parallel double-edge swapping procedure is primarily an algorithmic engineering one, and we give our approach below.

A. Parallel Double-edge Swaps: We create a fully parallel double-edge swapping procedure by randomly permuting a given edge list, attempting to swap adjacent edge pairs in the permuted list, while checking against existing edges via a hash table which allows thread-safe edge-wise insertions. We give our double-edge swap algorithm in Algorithm III.1.

ALGORITHM III.1. Parallel Double-edge Swaps.

```
1: procedure SwapEdges($E$)
2:     for some number of iterations do
3:         $T \leftarrow \emptyset$                                  ▷ Hash Table
4:         for all $e = \{u, v\} \in E$ do in parallel
5:             TestAndSet($T, e$)                        ▷ Thread-safe insertion
6:         Permute($E$)                                  ▷ Parallel Permutation
7:         for $i = 1 \ldots |E|$, $i$ is even do in parallel
8:             $\{u, v\} \leftarrow e = E(i)$
9:             $\{x, y\} \leftarrow f = E(i+1)$
10:                                          ▷ Randomly select swap partners
11:            if rand(0, 1) $< 0.5$ then
12:                $g \leftarrow \{u, x\}$
13:                $h \leftarrow \{v, y\}$
14:            else
15:                $g \leftarrow \{u, y\}$
16:                $h \leftarrow \{v, x\}$
17:            if TestAndSet($T, g$) $=$ false and
18:                TestAndSet($T, h$) $=$ false and
19:                $g$ and $h$ not self loops then
20:                $E(i), E(i+1) \leftarrow g, h$
21:            else
22:                $E(i), E(i+1) \leftarrow e, f$                  ▷ Swap failed
23:        clear($T$)
24:    return $E$
```

To permute the edge list, we use an algorithm from Shun et al. [32]. Their approach allows the extraction of parallel execution from procedures that are otherwise serial, with little overhead and high efficiency. We observe an order-of-magnitude speedup using the approach of Shun et al. over other existing libraries and implementations for random permutations on an array, e.g. [3].

In order to track existing edges and check edge simplicity in a scalable way, we require a hash table that allows fast thread-safe insertions of edges defined by $\{u, v\}$ vertex pairs. We adapt a hash table from prior work [33] for this purpose. The hash table uses an efficient hashing function combined with linear (or quadratic) probing. Edges defined by two `32-bit` integers are packed into a single `64-bit` key, with a paired value of `true` or `false` defining the key's existence in the table. The table generally requires only a single atomic operation per insertion, while threads only require further blocking if there is a "collision" between two or more threads attempting to insert into the same index of the key-value array. Though we note that collisions are rather rare as each key is initially guaranteed to be unique. The "TestAndSet" function in Algorithm III.1 performs insertions, and returns `true` if the key is already in the table and `false` otherwise.

We have validated that our procedure produces a minimally-biased uniform sample by repeating several variations of an experiment from prior work [22]. These experiments demonstrate that a sample of graphs produced from repeated swaps matches an analytically expected sample. We further discuss the number of swap iterations versus empirical mixing in our results.

IV. Generating from a Degree Distribution

The second problem we consider is the generation a null graph model from only a degree distribution. Algorithm IV.1 gives an overview of our method, where we take as input a degree distribution $\{D, N\} = \{(d_1, n_1), \ldots, (d_{max}, n_{max})\}$ (d_1 is the first degree; n_1 is the number of vertices with degree d_1). The graph that we output will in expectation match this distribution. We have three primary phases to our algorithm. First, we generate a set of degree i, j pairwise probabilities P for edge-skipping. Inputting these probabilities to the second phase, edge-skipping itself, will output an edge list E with vertices of degrees that will in expectation match the original distribution N. We then perform double-edge swaps to get our final edge list E'.

ALGORITHM IV.1. Overview of Method to Generate Simple Uniformly Random Graphs.

```
1: procedure GenerateGraph($\{D, N\}$)
2:     $P \leftarrow$ GenerateProbabilities($\{D, N\}$)          ▷ Section IV-A
3:     $E \leftarrow$ GenerateEdges($P, \{D, N\}$)              ▷ Section IV-B
4:     $E' \leftarrow$ SwapEdges($E$)                           ▷ Section III-A
5:     return $E'$
```

A. Adaptation of Probabilities: In order for a Bernoulli Chung-Lu generator to output a graph having in expectation a given degree distribution, the pairwise probabilities must solve the following system of equations:

$$\begin{aligned}
1 &= \left(\sum_{i \in D} n_i \times P_{1,i}\right) - P_{1,1} \\
2 &= \left(\sum_{i \in D} n_i \times P_{2,i}\right) - P_{2,2} \\
&\cdots \\
d_{max} &= \left(\sum_{i \in D} n_i \times P_{d_{max},i}\right) - P_{d_{max},d_{max}}
\end{aligned}$$

where D is a list of unique degrees within a given distribution and $P_{i,j}$ is the pairwise edge probability between degrees i and j (obviously: $0 \leq P_{i,j} \leq 1$ and $P_{i,j} = P_{j,i}$). This system states that for any vertex of degree j, the sum of the attachment probabilities for all other vertex degrees ($i \in D$) times the number of vertices with the other degree (n_i) must equal i. We must subtract the final $P_{j,j}$ as there are $(n_j - 1)$ other vertices of degree j that the vertex of degree j can match to. This system has $|D|$ equations and $\frac{|D|(|D|-1)}{2}$ unknowns, making it very underdetermined. There exist many viable methods to calculate some valid solution to the system, but our aim is to do so as fast as possible; with subsequent generation and edge swaps we remove any bias our probability selection creates.

We focus here on a heuristic quadratic work $O(|D|^2)$ algorithm for generating these probabilities. Taking our set of unique degrees D we separate the nodes of our network into equivalence classes

$$[i] = \{v \in G : |\mathcal{N}(v)| = i\}$$

where $\mathcal{N}(v)$ denotes the neighbors of v. We then order these degree classes by expected degree. Each class will have $i \times n_i$ stubs connected to it, where a stub is an edge connected at only one end. We then iterate through our list conducting preferential inter-class attachment. For attachment we estimate the number of edges $e_{i,j}$ from $[i]$ to $[j]$. Using this estimate we calculate the temporary probability of a node in $[i]$ connecting to a node in $[j]$ by taking $p_{i,j}$ to be

$$p_{i,j} = \frac{e_{i,j}}{(2 \times n_i \times n_j)}$$

where $e_{i,j}$ is the number of stubs being connected between the communities. Call the number of free-stubs for community $[k]$ at a given iteration $FE(k)$. Then

$$e_{i,j} = \text{Min}\big(\frac{FE(i) \times FE(j)}{\sum FE(k) - FE(i)},\, n_i \times n_j,\, FE(j)\big)$$

which respectively correspond to the naive number of paired stubs based on uniform stub sampling, the maximum number of possible connected edges between the communities for a simple graph, and the total number of free stubs for class $[j]$ at the current iteration. Choosing the minimum of these will ensure we do not violate the simplicity condition of our network. A second component of $p_{i,j}$ that needs to be addressed is the factor of $\frac{1}{2}$. Note that we calculate both a $p_{i,j}$ and a $p_{j,i}$ associated to every pair of classes $[i], [j]$. This comes from the fact that we iterate through all classes in D removing stubs at every step. This implies our final edge probability between classes is:

$$P_{i,j} = P_{j,i} = p_{i,j} + p_{j,i}$$

This directly implies the necessity for our factor of $\frac{1}{2}$. Intuitively, this means we are "connecting" half as many stubs in each step as we intended. To make up for this discrepancy we also end up doubling all values in our initial free-stub array FE. This ensures we obtain degrees approximating our desired distribution as opposed to half our desired distribution.

As for bounding error we note that the number of free-stubs a class $[j]$ has at step $[i]$ follows a simple relation:

$$\forall i \neq j+1 : FE(i,j) = FE(i-1,j)(1 - p_{(i-1),j})$$

For the case, $i = j+1$ we can express our free stubs at family $[j]$ as:

$$FE(j+1,j) = FE(j,j)(1 - \sum_{k}^{d_{max}} p_{j,k})$$

Thus we express our number of available stubs in the final step as:

$$FE(d_{max},j) = FE(1,j) \times (1 - \sum_{k}^{d_{max}} p_{j,k}) \times \Pi_{k=1}^{d_{max}}(1 - p_{k,j})$$

One can see that $FE(d_{max},j) \in \big[0, FE(1,j)\big)$. Unfortunately this upper bound can't be minimized in general. Despite this, empirical evidence suggests this error is small for non-contrived networks.

B. Edge Generation: We use a parallel edge-skipping implementation adapted from prior work [33] to generate edges. We give a condensed implementation in Algorithm IV.2. We include a short description of edge-skipping below, but for a more in-depth discussion of edge-skipping and its implementation details, we refer the reader to Miller and Hagberg [21] and Batagelj and Brandes [4].

The basic idea of edge skipping is to consider all possible undirected edges in some graph as an ordered space and to iteratively traverse and select from that space approximately m edges. Consider space X for all possible edges as u, v pairs as:

$$X = \{e_1, e_2, \ldots, e_{end}\}$$

Instead of, as in the Bernoulli model, flipping a coin for each possible e_i, we iteratively sample some number of *skip lengths*, given as l in Algorithm IV.2. We use these skip lengths to traverse the space X, where we can determine at each step some u, v pair represented by our current index x in the space. With a graph having equal edge probabilities between all vertex pairs (e.g., the Erdős-Rényi model), we only need to consider one single space for the entire graph. When probabilities differ (e.g., the Chung-Lu model), we need to consider a separate space ($X_{1,1}, X_{1,2}, \ldots, X_{d_{max},d_{max}}$), the size of each space (*end* in Algorithm IV.2), and how we get u, v for each differing probability. We note that the u, v we calculate are offsets within a given space; global identifiers can be retrieved based on prefix sums of N if we order vertex identifiers by degree.

ALGORITHM IV.2. Parallel Edge-Skipping.

```
1: procedure GenerateEdges($P, \{D, N\}$)
2:     $E \leftarrow \emptyset$
3:     $I \leftarrow$ ParallelPrefixSums($N$)
4:     for $k \leftarrow 1 \ldots |D| \times |D|$ do in parallel
5:         $i \leftarrow D(\frac{k}{|D|})$                          ▷ First degree
6:         $j \leftarrow D(k \mod |D|)$                     ▷ Second degree
7:         if $j > i$ then
8:             continue
9:         $p \leftarrow P(i,j)$                          ▷ Edge probabilities
10:        if $i = j$ then
11:            $end \leftarrow \frac{N(i) \times (N(i)-1)}{2}$
12:        else
13:            $end \leftarrow N(i) \times N(j)$
14:        $x \leftarrow 0$
15:        while $x < end$ do                       ▷ Can be parallelized
16:            $r \leftarrow$ Rand(0, 1)
17:            $l \leftarrow$ Floor($\frac{\log(r)}{\log(1-p)}$)
18:            $x \leftarrow x + (l + 1)$
19:            if $i = j$ then
20:                $u \leftarrow$ Ceil($\frac{-1+\sqrt{1+8*x}}{2}$)
21:                $v \leftarrow x - (u \times \frac{u-1}{2}) - 1$
22:            else
23:                $u \leftarrow$ Floor($\frac{x-1}{N(j)}$)
24:                $v \leftarrow (x - 1)$
25:            $E \leftarrow \{(I(i) + u), (I(j) + v)\}$
26:    return $E$
```

Parallelization can be performed over the entirety of X, where each thread determines some initial start and end offset pair within the space on which to calculate skip-lengths and output edges. As with the method in general, such an approach is provably equivalent to a general Bernoulli process of flipping a coin on each possible edge.

V. Complexity Discussion

The work complexity of generating probabilities is $O(|D|^2)$ with parallel time of $O(|D|)$, as only the attachment probabilities for a given degree can all be computed naïvelyin parallel; carried dependencies prevent full parallelization. We require $O(|D|^2)$ space to hold the probabilities in memory. We note that for a majority of real-world degree distributions and all of our test instances $|D| \ll d_{max} < |D|^2 \ll m$. Edge-skipping has a known optimal work complexity of $O(m)$ with a parallel time of $O(1 + |D|)$ using fine-grained parallelism on m processors. We need $O(\log(n))$ time to compute prefix sums for our vertex identifiers. Each iteration of double-edge swapping requires approximately $O(m \log(m))$ work and $\log(m)$ parallel time for permutation and $O(m)$ work and $O(1)$ time for the actual swaps. For space, we require an additional $O(m)$ for the hash table to ensure edge simplicity when swapping and $O(m)$ for the edge list itself. In total, our full end-to-end procedure for generating a graph from a degree distribution requires $O(|D|^2 + m\log(m))$ work and $O(|D| + \log(m) + \log(n))$ parallel time. Our memory requirements are $O(|D|^2 + m + n)$ in total.

VI. Hierarchical Network Generation

This work can also be utilized for the generation of random hierarchical networks. A popular benchmark for community detection algorithms is the Lancichinetti-Fortunato-Radicchi benchmark [19], colloquially known as LFR. An LFR graph is comprised of some number of clusters (or communities) of various sizes matching some power-law size distribution. The vertices comprising the global graph also match some power-law degree distribution. A given vertex's degrees are distributed between internal and external neighbors, with regards to its assigned community, such that the global average ratio of external to total edges is equal to some *mixing parameter* μ. As μ increases, the performance of some community detection algorithm is expected to drop, as the communities become less well-defined. We can directly use our method to create LFR-like graphs by layering random graphs created from splitting the degrees for each vertex into distinct *internal* and *external* degrees, relative to its assigned community and the global μ [34]. We've observed that existing Chung-Lu methods are unable to accurately capture the degree distributions of the large number of small skewed communities.

This two-level approach can be further generalized to any number of hierarchical or overlapping levels, similar to hierarchical random graphs [12] or overlapping communities [37]. Each level in a given hierarchy would have some number of subgraphs having assigned some subset (or all) vertices in the graph. For each subgraph, we include a value λ_i which is the share of the degree for each vertex that is assigned to the given subgraph i. The only restriction is that the λ values in the subgraphs for which vertex is assigned must sum to 1.0. Under this, arbitrary hierarchical and overlapping network structures can be generated in a straightforward manner while retaining a global degree distribution. We reserve further discussion of these applications for future work.

VII. Experimental Setup

Our experimental system is the *DRP* testbed cluster at the Center for Computational Innovations at RPI. Each node has 256 GB DDR and two 8-core 2.6 GHz Intel Xeon E5-2650 processor and 64 nodes are networked via 56 Gb FDR Infiniband. We parsed degree distributions from well-known large-scale graph datasets. The graph properties and sources are listed in Table I. The first four graphs have extremely skewed distributions, while the latter four are for scalability testing.

All of our algorithms are written in `C++` using `OpenMP` for parallelism. We've positively validated our outputs versus serial where applicable. Our code will be released into the `HPCGraphAnalysis` repository[3], pending copyright

[3]https://github.com/HPCGraphAnalysis/

TABLE I
TEST GRAPH CHARACTERISTICS. # VERTICES (n), # EDGES (m), AVERAGE (d_{avg}) AND MAX (d_{max}) VERTEX DEGREES, # UNIQUE DEGREES $|D|$, AND SOURCE. $B = \times10^9$, $M = \times10^6$, $K = \times10^3$.

Network	n	m	d_{avg}	d_{max}	$\|D\|$	Source
Meso	1.8 K	3.1 K	3.4	401	31	[31]
as20	6.5 K	12.5 K	3.9	1.5 K	83	[20]
WikiTalk	2.4 M	4.7 M	3.9	100 K	1.8 K	[20]
DBPedia	6.7 M	193 M	5.8	7.3 M	4.9 K	[25]
LiveJournal	4.1 M	27 M	13	2.0 K	945	[20]
Friendster	40 M	1.8 B	90	5.2 K	3.1 K	[20]
Twitter	39 M	1.4 B	73	56 K	18 K	[10]
uk-2005	30 M	728 M	49	41 K	5.2 K	[7]

approvals.

VIII. Results

We focus our results on our method for generating from a degree distribution, but results for generating from an edge list can be inferred as they fully utilize the swap phase of the former's approach. We compare our method against a baseline Chung-Lu model ($O(m)$), an *erased* Chung-Lu model ($O(m)$ simple), and a Bernoulli Chung-Lu model ($O(n^2)$ edgeskip). All methods are equivalently parallelized. Note that while the $O(m)$ model produces multi-edges, some number of double-edge swap iterations will result in the graph being "simplified". We look at execution time, how closely the output matches the input degree distribution, and how closely the pairwise edge probabilities match a uniformly random sample. We generate a uniformly random graph via Havel-Hakimi generation and 128 full iterations of double-edge swaps for comparison [22]. We report parallel timings running on all 16 cores of a single node of *DRP*.

A. Quality Comparisons: We first examine output quality in terms of matching the degree distribution by examining the error in the number of edges output, the maximum vertex degree, and degree skew via the Gini coefficient [9]. We plot the averaged percentage error for the different generators in Figure 3. We note that on average the $O(m)$ model output most closely matches the input distribution except for with the Gini coefficient, where we more closely match the lower degree part of the distribution due to how we handle our attachment probabilities for lower-degree vertices. Out of all of the simple graph generators, we note that our solution to attachment probabilities help us accurately match the distribution's maximum degree and number of total edges while using edge-skipping. This is the primary advantage of our method relative to existing approaches.

We also consider how quickly the pairwise edge probabilities from each of the models converge towards the uniform random sample. We plot in Figure 4 the error over all probabilities as the L1-norm of matrices $P_{Gen} - P_{Base}$, defined where $P_{i,j}$ is the pairwise probability between degrees i and j, P_{Gen} is generator output, and P_{Base} is the baseline Havel-Hakimi generator with swaps. We plot this error averaged over several tests and versus the number of swap iterations.

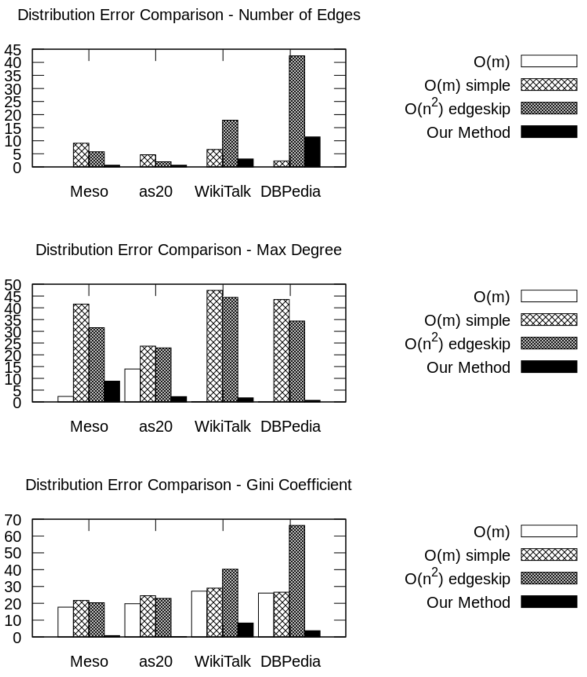

Fig. 3. Error comparison in # edges (top), d_{max} (middle), and Gini coefficient (bottom).

We note that the probabilities for the $O(m)$ model are initially the worst but eventually converge. This is due to the large number of multi-edges and the subsequent number of swaps which "fail" on each iteration . For all graphs, about two dozen or so swap iterations is sufficient to eliminate all multi-edges with the $O(m)$ approach. We also observe that 10 or so swap iterations is also sufficient to ensure that all edges are randomly swapped with all methods. As swapping is the most expensive part of generating from a degree distribution, we state that our described method is preferable to naïve $O(m)$ edge list generation for performance reasons. The exact number of iterations depends on the degree distribution and is related to the average probability of creating multi-edges, which is proportional to the "heaviness" of the distribution tail. All of the *simple* methods converge quite quickly, with about 5 iterations resulting in under 1% error on most tests. This is despite the fact that none of the edge generators used exactly match the input degree sequence. We note slightly slower convergence for our method relative to the other

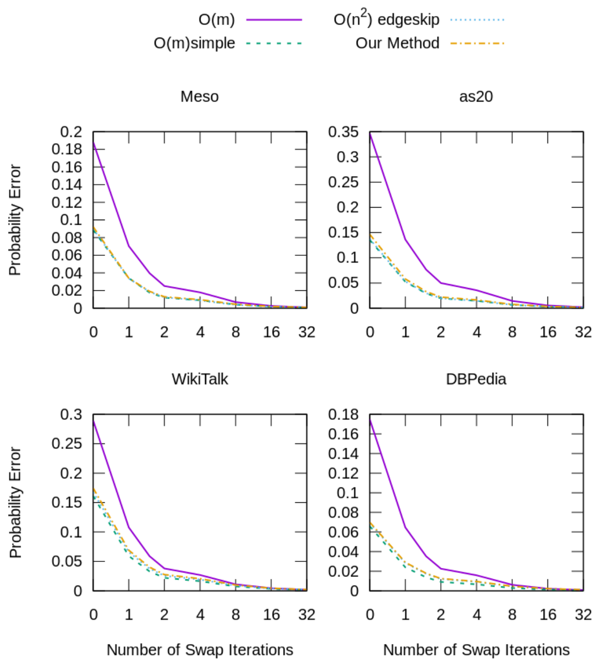

Fig. 4. Error comparison in pairwise attachment probabilities relative to a uniformly random sample.

two simple approaches due to bias introduced with our probability generation; however, as per Figure 3, we better match in expectation the input degree distribution.

B. Timing Comparisons: We also compare the end-to-end times for generation from a degree distribution with the various methods in Figure 5. We consider only a single iteration of double-edge swaps for consistency, as we've noted that mixing time is graph-dependent. We observe that at the smaller scale most methods are approximately the same. Our additional probability calculation step results in slower execution times in certain instances. However, as the test sizes scale up, as with DBPedia, we observe a considerable benefit of edge-skipping, as sampling for the $O(m)$ and erased model are done on a weighted list, requiring $O(log(n))$ time for a binary search for each sampled vertex. For the larger instances, the $O(m)$ methods are approximately twice as slow.

We next consider the per-phase time costs for our method. We plot in Figure 6 the average time required for probability computation, edge generation, and edge swapping over all eight test instances. We observe that while probability generation has quadratic complexity and linear time, in practice the small relative size of $|D|$ versus d_{max} makes its computation proportionally quick. Even for our largest instances, our end-to-end generation time takes less than two minutes to complete, giving us an edge generation rate of about 1 billion edges per second on only 16 cores.

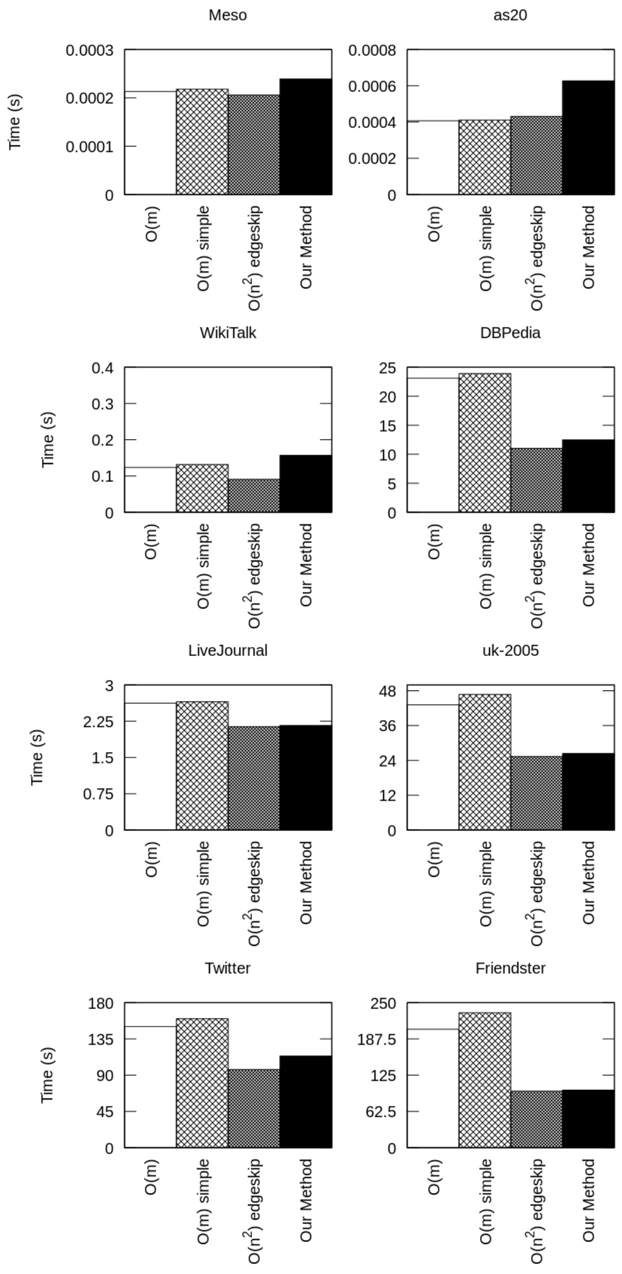

Fig. 5. Shared-memory end-to-end times for the various generators.

C. Comparisons to Related Work: Little other prior work seeks to optimize the end-to-end generation procedure for unbiased uniformly-random simple graphs. Recent work has looked at parallel algorithms for edge swaps [5], though they consider the problem in distributed memory. They report in serial a time of about 300 seconds to successfully swap all edges in LiveJournal and about 20 seconds on 64 processors. We report a time of 15 seconds in serial and 3 seconds on 16 cores processors to achieve the same (3 swap

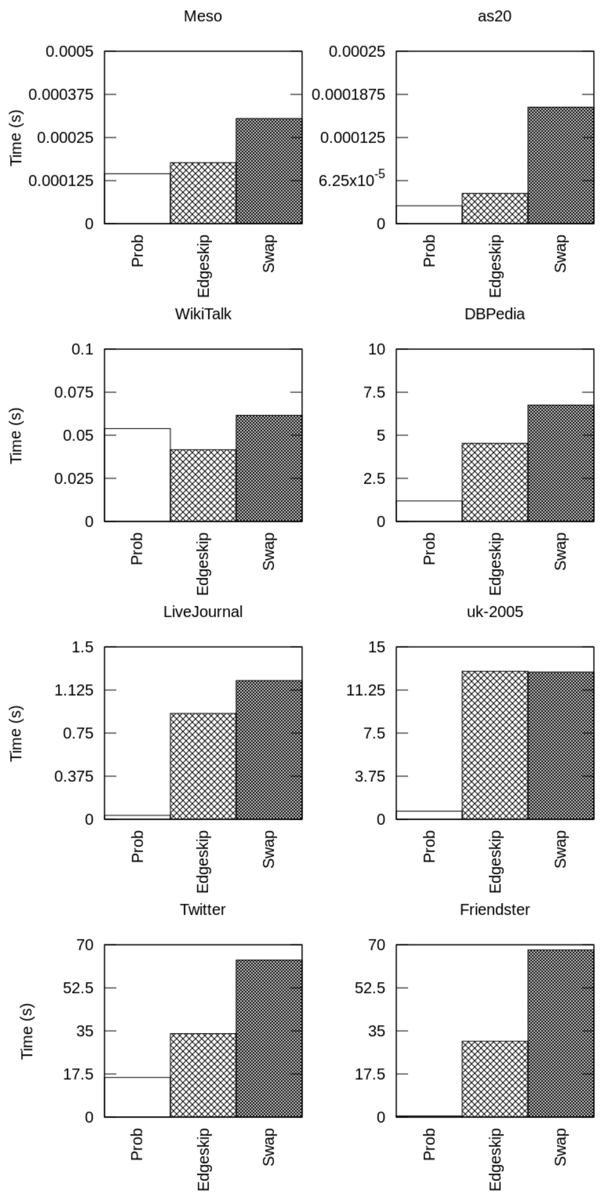

Fig. 6. Per-phase execution time.

iterations). We can do a single swap iteration in 1 second in parallel, which we observe to swap 99.9% of edges. We add the disclaimer that these results again aren't directly comparable, as we currently only consider the problem in shared memory. Other work has sought to optimize the generation of various types of random graphs, including community detection benchmarking graphs [33], edge-skipping generation for Chung-Lu and similar graphs [1], and massive-scale generation for a variety of other network models [17]. We note that the parallel performance of these generators can be quite impressive, scaling to trillion-edge graph generation, though the graph models they consider aren't directly comparable to what's considered in this current work.

IX. Discussion and Future Work

Ideally, there would exist a direct solution for some set of $P_{i,j}$ edge probabilities that, when run with a Bernoulli Chung-Lu edge generator, would output a simple uniform random graph. However, such a solution is not known to exist. In our research, we have derived a combinatorial approximation for some set of probabilities. However, the expected complexity is $O(n^2 d_{max}^2)$ and implementation at even a modest scale poses numerical challenges due to the combinatorially large numbers involved. In addition, we note that Chung-Lu generators in general have error in matching the low degree of the distribution. This work and others [18], [33] only utilize heuristics to address this issue. We reserve for future work an analytical or algorithmic solution using both probabilities and degree distribution modifications that would allow us to minimize this error.

We make one final note that a more formal validation of uniform randomness per mixing time is required, as current analytical bounds appear rather loose relative to empirical observations. We make the assumption that, in practice, the number of swap iterations required is proportional to the chance of an unsuccessful swap; this chance is relatable to graph density and degree skew. Similarly, we also observe that uniform mixing appears to be achieved after a sufficient number of iterations where each edge has been successfully swapped, regardless of graph scale. A more in-depth empirical and analytical study might help reinforce these notions and give more practical bounds.

X. Acknowledgements

We thank the Center for Computational Innovations at RPI for supplying and supporting the systems used in this paper. This research was partially funded through the Laboratory Directed Research and Development (LDRD) program at Sandia. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

[1] M. Alam, M. Khan, A. Vullikanti, and M. Marathe, *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, IEEE, 2016, pp. 372–383.

[2] Y. Artzy-Randrup and L. Stone, *Generating uniformly distributed random networks*, Physical Review E, 72 (2005), p. 056708.

[3] A. Bacher, O. Bodini, A. Hollender, and J. Lum-

BROSO, *Mergeshuffle: A very fast, parallel random permutation algorithm*, arXiv preprint arXiv:1508.03167, (2015).

[4] V. BATAGELJ AND U. BRANDES, *Efficient generation of large random networks*, Physical Review E, 71 (2005), p. 036113.

[5] H. BHUIYAN, M. KHAN, J. CHEN, AND M. MARATHE, *Parallel algorithms for switching edges in heterogeneous graphs*, Journal of parallel and distributed computing, 104 (2017), pp. 19–35.

[6] V. D. BLONDEL, J.-L. GUILLAUME, R. LAMBIOTTE, AND E. LEFEBVRE, *Fast unfolding of communities in large networks*, Journal of statistical mechanics: theory and experiment, 2008 (2008), p. P10008.

[7] P. BOLDI AND S. VIGNA, *The WebGraph framework I: Compression techniques*, in Proc. of the Thirteenth International World Wide Web Conference (WWW 2004), Manhattan, USA, 2004, ACM Press, pp. 595–601.

[8] T. BRITTON, M. DEIJFEN, AND A. MARTIN-LÖF, *Generating simple random graphs with prescribed degree distribution*, Journal of Statistical Physics, 124 (2006), pp. 1377–1397.

[9] L. CERIANI AND P. VERME, *The origins of the gini index: extracts from variabilità e mutabilità (1912) by corrado gini*, The Journal of Economic Inequality, 10 (2012), pp. 421–443.

[10] M. CHA, H. HADDADI, F. BENEVENUTO, AND K. P. GUMMADI, *Measuring user influence in Twitter: The million follower fallacy*, in Proc. Int'l. Conf. on Weblogs and Social Media (ICWSM), 2010.

[11] F. CHUNG AND L. LU, *The average distances in random graphs with given expected degrees*, Proceedings of the National Academy of Sciences, 99 (2002), pp. 15879–15882.

[12] A. CLAUSET, C. MOORE, AND M. E. NEWMAN, *Hierarchical structure and the prediction of missing links in networks*, Nature, 453 (2008), p. 98.

[13] C. COOPER, M. DYER, AND C. GREENHILL, *Sampling regular graphs and a peer-to-peer network*, Combinatorics, Probability and Computing, 16 (2007), pp. 557–593.

[14] N. DURAK, T. G. KOLDA, A. PINAR, AND C. SESHADHRI, *A scalable null model for directed graphs matching all degree distributions: In, out, and reciprocal*, in 2013 IEEE 2nd Network Science Workshop (NSW), IEEE, 2013, pp. 23–30.

[15] P. L. ERDOS, I. MIKLÓS, AND Z. TOROCZKAI, *A simple havel-hakimi type algorithm to realize graphical degree sequences of directed graphs*, Electronic Journal of Combinatorics, 17 (2010), p. R66.

[16] B. K. FOSDICK, D. B. LARREMORE, J. NISHIMURA, AND J. UGANDER, *Configuring random graph models with fixed degree sequences*, SIAM Review, 60 (2018), pp. 315–355.

[17] D. FUNKE, S. LAMM, P. SANDERS, C. SCHULTZ, D. STRASH, AND M. VON LOOZ, *Communication-free massively distributed graph generation*, in International Parallel & Distributed Processing Symposium (IPDPS), 2018.

[18] T. G. KOLDA, A. PINAR, T. PLANTENGA, AND C. SESHADHRI, *A scalable generative graph model with community structure*, SIAM Journal on Scientific Computing, 36 (2014), pp. C424–C452.

[19] A. LANCICHINETTI, S. FORTUNATO, AND F. RADICCHI, *Benchmark graphs for testing community detection algorithms*, Physical Review E, 78 (2008), pp. 1–5, https://doi.org/10.1103/PhysRevE.78.046110, http://link.aps.org/doi/10.1103/PhysRevE.78.046110.

[20] J. LESKOVEC AND A. KREVL, *SNAP Datasets: Stanford large network dataset collection*. http://snap.stanford.edu/data, jun 2014.

[21] J. C. MILLER AND A. HAGBERG, *Efficient generation of networks with given expected degrees*, in International Workshop on Algorithms and Models for the Web-Graph, Springer, 2011, pp. 115–126.

[22] R. MILO, N. KASHTAN, S. ITZKOVITZ, M. E. NEWMAN, AND U. ALON, *On the uniform generation of random graphs with prescribed degree sequences*, arXiv preprint cond-mat/0312028, (2003).

[23] R. MILO, S. SHEN-ORR, S. ITZKOVITZ, N. KASHTAN, D. CHKLOVSKII, AND U. ALON, *Network motifs: simple building blocks of complex networks*, Science, 298 (2002), pp. 824–827.

[24] M. MOLLOY AND B. REED, *A critical point for random graphs with a given degree sequence*, Random structures & algorithms, 6 (1995), pp. 161–180.

[25] M. MORSEY, J. LEHMANN, S. AUER, AND A.-C. N. NGOMO, *Dbpedia sparql benchmark–performance assessment with real queries on real data*, in International semantic web conference, Springer, 2011, pp. 454–469.

[26] M. E. NEWMAN, *Assortative mixing in networks*, Physical review letters, 89 (2002), p. 208701.

[27] M. E. NEWMAN, *Mixing patterns in networks*, Physical Review E, 67 (2003), p. 026126.

[28] M. E. NEWMAN, S. H. STROGATZ, AND D. J. WATTS, *Random graphs with arbitrary degree distributions and their applications*, Physical review E, 64 (2001), p. 026118.

[29] J. PARK AND M. E. NEWMAN, *Statistical mechanics of networks*, Physical Review E, 70 (2004), p. 066117.

[30] J. J. PFEIFFER III, T. LA FOND, S. MORENO, AND J. NEVILLE, *Fast generation of large scale social networks with clustering*, arXiv preprint arXiv:1202.4805, (2012).

[31] Y. SHIMODA, S. SHINPO, M. KOHARA, Y. NAKAMURA, S. TABATA, AND S. SATO, *A large scale analysis of protein–protein interactions in the nitrogen-fixing bacterium mesorhizobium loti*, DNA research, 15 (2008), pp. 13–23.

[32] J. SHUN, Y. GU, G. E. BLELLOCH, J. T. FINEMAN, AND P. B. GIBBONS, *Sequential random permutation, list contraction and tree contraction are highly parallel*, in Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2015, pp. 431–448.

[33] G. M. SLOTA, J. BERRY, S. D. HAMMOND, S. OLIVIER, C. PHILLIPS, AND S. RAJAMANICKAM, *Scalable generation of graphs for benchmarking hpc community-detection algorithms*, in IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2019.

[34] G. M. SLOTA AND J. GARBUS, *A parallel LFR-like benchmark for evaluating community detection algorithms*, in IEEE Workshop on Parallel and Distributed Processing for Computational Social Systems (ParSocial 2020), 2020.

[35] R. VAN DER HOFSTAD, *Critical behavior in inhomogeneous random graphs*, Random Structures & Algorithms, 42 (2013), pp. 480–508.

[36] M. WINLAW, H. DESTERCK, AND G. SANDERS, *An in-depth analysis of the chung-lu model*, tech. report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2015.

[37] J. YANG AND J. LESKOVEC, *Community-affiliation graph model for overlapping network community detection*, in 2012 IEEE 12th international conference on data mining, IEEE, 2012, pp. 1170–1175.