A Parallel Algorithm for Generating a Random Graph with a Prescribed Degree Sequence

Hasanuzzaman Bhuiyan
Department of Computer Science
Network Dynamics and Simulation Science Laboratory (NDSSL)
Biocomplexity Institute of Virginia Tech
Blacksburg, VA, USA
mhb@vt.edu

Maleq Khan
Department of Electrical Engineering and Computer Science
Texas A&M University–Kingsville
Kingsville, TX, USA
maleq.khan@tamuk.edu

Madhav Marathe
Department of Computer Science
Network Dynamics and Simulation Science Laboratory (NDSSL)
Biocomplexity Institute of Virginia Tech
Blacksburg, VA, USA
mmarathe@vt.edu

***Abstract*—Random graphs (or networks) have gained a significant increase of interest due to its popularity in modeling and simulating many complex real-world systems. Degree sequence is one of the most important aspects of these systems. Random graphs with a given degree sequence can capture many characteristics like dependent edges and non-binomial degree distribution that are absent in many classical random graph models such as the Erdős-Rényi graph model. In addition, they have important applications in uniform sampling of random graphs, counting the number of graphs having the same degree sequence, as well as in string theory, random matrix theory, and matching theory. In this paper, we present an OpenMP-based shared-memory parallel algorithm for generating a random graph with a prescribed degree sequence, which achieves a speedup of 20.5 with 32 cores. We also present a comparative study of several structural properties of the random graphs generated by our algorithm with that of the real-world graphs and random graphs generated by other popular methods. One of the steps in our parallel algorithm requires checking the Erdős-Gallai characterization, i.e., whether there exists a graph obeying the given degree sequence, in parallel. This paper presents a non-trivial parallel algorithm for checking the Erdős-Gallai characterization, which achieves a speedup of 23 with 32 cores.**

***Index Terms*—graph theory, random graph generation, degree sequence, Erdős-Gallai characterization, parallel algorithms**

I. Introduction

Random graphs are widely used for modeling many complex real-world systems such as the Internet [1], biological [2], social [3], and infrastructure [4] networks to understand how the systems work through obtaining rigorous mathematical and simulation results. Many random graph models such as the Erdős-Rényi [5], the Preferential Attachment [6], the small-world [7], and the Chung-Lu [8] models have been proposed to capture various characteristics of real-world systems. Degree sequence is one of the most important aspects of these systems and has been extensively studied in graph theory [9–11]. It has significant applications in a wide range of areas including structural reliability and communication networks because of the strong ties between the degrees of vertices and the structural properties of and dynamics over a network [12].

Random graphs with given degree sequences are widely used in uniform sampling of random graphs as well as in counting the number of graphs having the same degree sequence [13–16]. For example, in an epidemiology study of sexually transmitted diseases [17], anonymous surveys collect data about the number of sexual partners of an individual within a given period of time, and then the problem reduces to generating a network obeying the degree sequence collected from the survey, and studying the disease dynamics over the network. Other examples include determining the total number of structural isomers of chemical compounds such as alkanes, where the valence of an atom is the degree. Moreover, the random graphs with given degree sequences can capture many characteristics such as dependent edges and non-binomial degree distribution that are absent in many classical models such as the Erdős-Rényi [5] graph model. They also have important applications in string theory, random matrix theory, and matching theory [10].

The problem of generating a random graph with a given degree sequence becomes considerably easier if self-loops and parallel edges are allowed. Throughout this paper, we consider simple graphs with no self-loops or parallel edges. Most prior work on generating random graphs involves sequential algorithms, and they can be broadly categorized in two classes: (***i***) edge swapping and (***ii***) stub-matching. Edge swapping [18–20] uses the Markov chain Monte Carlo (MCMC) scheme on a given graph having the degree sequence. An edge swap operation replaces two edges $e_1 = (a, b)$ and $e_2 = (c, d)$, selected uniformly at random from the graph, by new edges $e_3 = (a, d)$ and $e_4 = (c, b)$, i.e., the end vertices of the selected edges are swapped with each other. This operation is repeated either a given number of times or until a specified criterion is satisfied. It is easy to see that the degree of each vertex remains invariant under an edge swap process. Unfortunately, very little theoretical results have been rigorously shown about the mixing time [18, 22] of the edge swap process and they are ill-controlled. Moreover, most of the results are heuristic-based.

On the other hand, among the swap-free stub-matching methods, the *configuration* or *pairing* method [23] is very popular and uses a direct graph construction method. For each vertex, it creates as many stubs or "dangling half-edges" as of its degree. Then edges are created by choosing pairs of

vertices randomly and connecting them. This approach creates parallel edges, which are dealt with by restarting the process. Unfortunately, the probability of restarting the process approaches 1 for larger degree sequences. Many variants [24–26] of the configuration models have been studied to avoid parallel edges for the regular graphs. By using the Havel-Hakimi method [27], a deterministic graph can be generated following a given degree sequence. Bayati et al. [15] presented an algorithm for counting and generating random simple graphs with given degree sequences. However, this algorithm does not guarantee to always generate a graph, and it is shown that the probability of not generating a graph is small for a certain bound on the maximum degree, which restricts many degree sequences. Genio et al. [16] presented an algorithm to generate a random graph from a given degree sequence, which can be used in sampling graphs from the graphical realizations of a degree sequence. Blitzstein et al. [14] also proposed a sequential importance sampling [28] algorithm to generate random graphs with an exact given degree sequence, which can generate every possible graph with the given degree sequence with a non-zero probability. Moreover, the distribution of the generated graphs can be estimated, which is a much-desired result used in sampling random graphs.

A deterministic parallel algorithm for generating a simple graph with a given degree sequence has been presented by Arikati et al. [29], which runs in $\mathcal{O}(\log n)$ time using $\mathcal{O}(n+m)$ CRCW PRAM [30] processors, where n and m denote the number of vertices and edges in the graph, respectively. From a given degree sequence, the algorithm first computes an appropriate bipartite sequence (degree sequence of a bipartite graph), generates a deterministic bipartite graph obeying the bipartite sequence, applies some edge swap techniques to generate a symmetric bipartite graph, and then reduces the symmetric bipartite graph to a simple graph having the given degree sequence. Another parallel algorithm, with a time complexity of $\mathcal{O}\left(\log^4 n\right)$ using $\mathcal{O}\left(n^{10}\right)$ EREW PRAM processors, has been presented in [31], where the maximum degree is bounded by the square-root of the sum of the degrees, which restricts many degree sequences. A parallel algorithm for generating a random graph with a given *expected* degree sequence has been presented in [32]. However, there is no existing parallel algorithm for generating random graphs following an exact degree sequence, which can provably generate each possible graph, having the given degree sequence, with a positive probability. In this paper, we present an efficient parallel algorithm for generating a random graph with an exact given degree sequence. We choose to parallelize the sequential algorithm by Blitzstein et al. [14] because of its rigorous mathematical and theoretical results, and the algorithm supports all of the important and much-desired properties below, whereas the other algorithms are either heuristic-based or lack some of the following properties:

- It can construct a random simple graph with a prescribed degree sequence.
- It can provably generate each possible graph, obeying the given degree sequence, with a positive probability.
- It can be used in importance sampling by explicitly measuring the weights associated with the generated graphs.
- It is guaranteed to terminate with a graph having the prescribed degree sequence.
- Given a degree sequence of a tree, a small tweak while assigning the edges allows the same algorithm to generate trees uniformly at random.
- It can be used in estimating the number of possible graphs with the given degree sequence.

Our Contributions. In this paper, we present an efficient shared-memory parallel algorithm for generating random graphs with exact given degree sequences. The dependencies among assigning edges to vertices in a particular order to ensure the algorithm always successfully terminates with a graph, the requirement of keeping the graph simple, maintaining an exact stochastic process as that of the sequential algorithm, and concurrent writing by multiple cores in the global address space lead to significant challenges in designing a parallel algorithm. Dealing with these requires complex synchronization among the processing cores. Our parallel algorithm achieves a maximum speedup of 20.5 with 32 cores. We also present a comparative study of various structural properties of the random graphs generated by the parallel algorithm with that of the real-world graphs. One of the steps in our parallel algorithm requires checking the graphicality of a given degree sequence, i.e., whether there exists a graph with the degree sequence, using the Erdős-Gallai characterization [33] in parallel. We present here a novel parallel algorithm for checking the Erdős-Gallai characterization, which achieves a speedup of 23 using 32 cores.

Organization. The rest of the paper is organized as follows. Section II describes the preliminaries and notations used in the paper. Our main parallel algorithm for generating random graphs along with the experimental results are presented in Section III. We present a parallel algorithm for checking the Erdős-Gallai characterization of a given degree sequence accompanied by the performance evaluation of the algorithm in Section IV. Finally, we conclude in Section V.

II. Preliminaries

Below are the notations, definitions, and computation model used in this paper.

Notations. We use $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ to denote a simple graph, where $\mathbb{V}$ is the set of vertices and $\mathbb{E}$ is the set of edges. A *self-loop* is an edge from a vertex to itself. *Parallel edges* are two or more edges connecting the same pair of vertices. A *simple graph* is an undirected graph with no self-loops or parallel edges. We are given a *degree sequence* $\mathbb{D} = (d_1, d_2, \ldots, d_n)$. There are a total of $n = |\mathbb{V}|$ vertices labeled as $1, 2, \ldots, n$, and d_i is the degree of vertex i, where $0 \leq d_i \leq n-1$. For a degree sequence $\mathbb{D}$ and distinct $u, v \in \{1, 2, \ldots, n\}$, we define $\ominus_{u,v}^{\mathbb{D}}$ to be the degree sequence obtained from $\mathbb{D}$ by

TABLE I: Notations used frequently in the paper.

Symbol	Description	Symbol	Description
$\mathbb{D}$	Degree sequence	d_i	Degree of vertex i
$\mathbb{V}$	Set of vertices	n	Number of vertices
$\mathbb{E}$	Set of edges	m	Number of edges
$\mathcal{P}$	Number of cores	$\mathbb{P}_k$	Core with rank k
$\mathbb{C}$	Candidate set	$\mathcal{C}$	Corrected Durfee number
$\mathbb{G}$	Graph	K	Thousands
M	Millions	B	Billions

subtracting 1 from each of d_u and d_v. Let d'_j be the degree of vertex j in the degree sequence $\ominus^{\mathbb{D}}_{u,v}$, then

$$d'_j = \begin{cases} d_j - 1 & \text{if } j \in \{u, v\}, \\ d_j & \text{otherwise.} \end{cases} \tag{1}$$

If there is a simple graph $\mathbb{G}$ having the degree sequence $\mathbb{D}$, then there are $m = |\mathbb{E}|$ edges in $\mathbb{G}$, where $2m = \sum_i d_i$. The terms graph and network are used interchangeably throughout the paper. We use K, M, and B to denote thousands, millions, and billions, respectively; e.g., 1M stands for one million. For the parallel algorithms, let $\mathcal{P}$ be the number of processing cores, and $\mathbb{P}_k$ the core with rank k, where $0 \leq k < \mathcal{P}$. A summary of the frequently used notations (some of them are introduced later for convenience) is provided in Table I.

Residual Degree. During the course of a graph generation process, the *residual degree* of a vertex u is the remaining number of edges incident on u, which have not been created yet. From hereon, we refer to the degree d_u of a vertex u as the residual degree of u at any given time, unless otherwise specified.

Graphical Sequence. A degree sequence $\mathbb{D}$ of non-negative integers is called *graphical* if there exists a labeled simple graph with vertex set $\{1, 2, \ldots, n\}$, where vertex i has degree d_i. Such a graph is called a *realization* of the degree sequence $\mathbb{D}$. Note that there can be several graphs having the same degree sequence. Eight equivalent necessary and sufficient conditions for testing the graphicality of a degree sequence are listed in [34]. Among them, the Erdős-Gallai characterization [33] is the most famous and frequently used criterion. Another popular recursive test for checking a graphical sequence is the Havel-Hakimi method [27].

Erdős-Gallai Characterization [33]. Assuming a given degree sequence $\mathbb{D}$ is sorted in non-increasing order, i.e., $d_1 \geq d_2 \geq \ldots \geq d_n$, the sequence $\mathbb{D}$ is graphical if and only if $\sum_{i=1}^{n} d_i$ is even and

$$\text{for each } k \in \{1, 2, \ldots, n\}, \sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i). \tag{2}$$

For example, $\mathbb{D}_1 = (3, 3, 2, 2, 2)$ is a graphical sequence and there is a realization of $\mathbb{D}_1$ as it satisfies the Erdős-Gallai characterization, whereas $\mathbb{D}_2 = (4, 3, 2, 1)$ is not a graphical sequence and there is no simple graph realizing $\mathbb{D}_2$, as shown in Figs. 1 and 2.

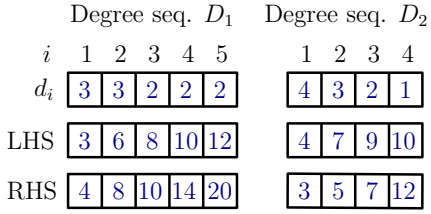

Fig. 1: Graphicality check for the degree sequences $\mathbb{D}_1 = (3, 3, 2, 2, 2)$ and $\mathbb{D}_2 = (4, 3, 2, 1)$ using the Erdős-Gallai characterization, where LHS and RHS denote the left hand side and right hand side values of Eq. (2), respectively.

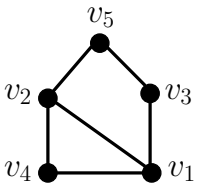

Fig. 2: A simple graph realizing the degree sequence $\mathbb{D}_1 = (3, 3, 2, 2, 2)$.

Computation Model. We develop algorithms for shared-memory parallel systems. All the cores can read from and write to the global address space. In addition, each core can have its own local variables and data structures.

III. Generating Random Graphs with Prescribed Degree Sequences

We briefly discuss the sequential algorithm in Section III-A. Then we present our parallel algorithm in Section III-B and the experimental results in Section III-C.

A. Sequential Algorithm

Blitzstein et al. [14] presented a sequential importance sampling [28] algorithm for generating random graphs with exact prescribed degree sequences. This approach first creates all edges incident on the vertex having the minimum degree in the sequence, then moves to the next vertex having the minimum degree to create its incident edges and so on. To create an edge incident on a vertex u, a candidate list $\mathbb{C}$ is computed using the Erdős-Gallai characterization such that, after adding an edge by connecting u to any candidate vertex v from the list $\mathbb{C}$, the residual degree sequence remains graphical and the graph remains simple. Then an edge (u, v) is assigned by choosing v from the candidate list $\mathbb{C}$ with a probability proportional to the degree of v. This process is repeated until all edges incident on vertex u are assigned.

For example, for a given degree sequence $\mathbb{D} = (3, 3, 2, 2, 2)$, the algorithm starts by assigning edges incident on vertex v_3. It computes the candidate list $\mathbb{C} = \{v_1, v_2, v_4, v_5\}$. Say it chooses the vertex v_5 from $\mathbb{C}$ and assigns the edge (v_3, v_5). Then the new degree sequence is $\mathbb{D} = (3, 3, 1, 2, 1)$, and the new candidate list for assigning the remaining edge incident on vertex v_3 is $\mathbb{C} = \{v_1, v_2\}$. Say the algorithm selects v_1 from $\mathbb{C}$ and assigns the edge (v_3, v_1). Now the new degree sequence is $\mathbb{D} = (2, 3, 0, 2, 1)$, and the algorithm will proceed to assign edges incident on vertex v_5 and so on. One possible sequence of degree sequences is

$$(3, 3, 2, 2, 2) \to (3, 3, 1, 2, 1) \to (2, 3, 0, 2, 1) \to (2, 2, 0, 2, 0) \to (1, 2, 0, 1, 0) \to (0, 1, 0, 1, 0) \to (0, 0, 0, 0, 0),$$

with the corresponding edge set

$$\mathbb{E} = \{(v_3, v_5), (v_3, v_1), (v_5, v_2), (v_1, v_4), (v_1, v_2), (v_2, v_4)\}.$$

1: $\mathbb{E} \leftarrow \emptyset$ ▷ initially empty set of edges
2: **while** $\mathbb{D} \neq$ **0 do**
3: Select the least u such that d_u is a minimal positive degree in $\mathbb{D}$
4: **while** $d_u \neq 0$ **do**
5: $\mathbb{C} \leftarrow \{v \neq u : (u,v) \notin \mathbb{E} \bigwedge \ominus_{u,v}^{\mathbb{D}}$ is graphical$\}$
6: $v \leftarrow$ a random candidate in $\mathbb{C}$ where probability of selecting v is proportional to d_v
7: $\mathbb{E} \leftarrow \mathbb{E} \cup \{(u,v)\}$
8: $\mathbb{D} \leftarrow \ominus_{u,v}^{\mathbb{D}}$
9: Output $\mathbb{E}$

Fig. 3: A sequential algorithm [14] for generating a random graph with a given degree sequence.

The corresponding graph is shown in Fig. 2. Note that during the assignment of incident edges on a vertex u, a candidate at a later stage is also a candidate at an earlier stage. The pseudocode of the algorithm is shown in Fig. 3. Since a total of m edges are generated for the graph $\mathbb{G}$ and computing the candidate list (Line 5) for each edge takes $\mathcal{O}\left(n^2\right)$ time, the time complexity of the algorithm is $\mathcal{O}\left(mn^2\right)$.

Unlike many other graph generation algorithms, this method never gets stuck, i.e., it always terminates with a graph realizing the given degree sequence (proof provided in Theorem 3 in [14]) or creates loops or parallel edges through the computation of the candidate list using the Erdős-Gallai characterization. The algorithm can generate every possible graph with a positive probability (proof given in Corollary 1 in [14]). For additional details about the importance sampling and estimating the number of graphs for a given degree sequence, see Sections 8 and 9 in [14]; and we omit the details in this paper due to space constraints.

B. Parallel Algorithm

To design an exact parallel version by maintaining the same stochastic process (in order to retain the same theoretical and mathematical results) as that of the sequential algorithm, the vertices are considered (to assign their incident edges) in the same order in the parallel algorithm, i.e., in ascending order of their degrees. Hence, we emphasize parallelizing the computation of the candidate list $\mathbb{C}$, i.e., Line 5 of the sequential algorithm in Fig. 3. For computing the candidate list to assign edges incident on a vertex u, we need to consider all other vertices v with non-zero degrees d_v as potential candidates; and we parallelize this step. While considering a particular vertex v as a candidate, we need to check whether $\ominus_{u,v}^{\mathbb{D}}$ is a graphical sequence using the Erdős-Gallai characterization. If $\ominus_{u,v}^{\mathbb{D}}$ is graphical, then v is added to the candidate list $\mathbb{C}$. The time complexity of the best known sequential algorithm for testing the Erdős-Gallai characterization is $\mathcal{O}(n)$ [14, 35]. Thus to have an efficient parallel algorithm for generating random graphs, we need to use an efficient parallel algorithm for checking the Erdős-Gallai characterization. In Section IV, we present an efficient parallel algorithm for checking the Erdős-Gallai characterization that runs in $\mathcal{O}\left(\frac{n}{\mathcal{P}} + \log \mathcal{P}\right)$ time. The parallel algorithm for the Erdős-Gallai characterization returns *TRUE* if the given degree sequence is graphical and *FALSE* otherwise.

1: $\mathbb{E} \leftarrow \emptyset$ ▷ initially empty set of edges
▷ Assign the edges until the degree of each vertex reduces to 0
2: **while** $\mathbb{D} \neq$ **0 do**
3: Select the least u such that d_u is a minimal positive degree in $\mathbb{D}$
4: $\mathbb{C} \leftarrow \emptyset$ ▷ candidate list
▷ Assign all d_u edges incident on u
5: **while** $d_u \neq 0$ **do**
6: **if** $\mathbb{C} = \emptyset$ **then**
7: $\mathbb{F} \leftarrow \{v \neq u : (u,v) \notin \mathbb{E} \bigwedge d_v > 0\}$
8: **else**
9: $\mathbb{F} \leftarrow \mathbb{C}$
10: $\mathbb{C} \leftarrow \emptyset$
▷ Compute the candidate list
11: **for** *each* $v \in \mathbb{F}$ ***in parallel*** **do**
12: $flag \leftarrow$ PARALLEL-ERDŐS-GALLAI $\left(\ominus_{u,v}^{\mathbb{D}}\right)$
13: **if** $flag = TRUE$ **then**
14: $\mathbb{C} \leftarrow \mathbb{C} \cup \{v\}$ ▷ v is a candidate
15: **if** $d_u = |\mathbb{C}|$ **then**
16: **for** *each* $v \in \mathbb{C}$ ***in parallel*** **do**
17: $\mathbb{E} \leftarrow \mathbb{E} \cup \{(u,v)\}$
18: $\mathbb{D} \leftarrow \ominus_{u,v}^{\mathbb{D}}$
19: **break**
▷ Assign an edge (u,v) from $\mathbb{C}$
20: $v \leftarrow$ a random candidate in $\mathbb{C}$ where probability of selecting v is proportional to d_v
21: $\mathbb{E} \leftarrow \mathbb{E} \cup \{(u,v)\}$
22: $\mathbb{D} \leftarrow \ominus_{u,v}^{\mathbb{D}}$
23: $\mathbb{C} \leftarrow \mathbb{C} - \{v\}$
24: Output $\mathbb{E}$ ▷ final set of edges

Fig. 4: A parallel algorithm for generating a random graph with a prescribed degree sequence.

Once the candidate list is computed, if the degree of u is equal to the cardinality $|\mathbb{C}|$ of the candidate list, then new edges are assigned between u and all candidate vertices v in the candidate list $\mathbb{C}$ in parallel. Otherwise, like the sequential algorithm, a candidate vertex v is chosen randomly from $\mathbb{C}$, a new edge (u,v) is assigned, the degree sequence $\mathbb{D}$ is updated by reducing the degrees of each of u and v by 1, and the process is repeated until d_u is reduced to 0. After assigning all edges incident on vertex u, the algorithm proceeds with assigning edges incident on the next vertex having the minimum positive degree in $\mathbb{D}$ and so on. We present the pseudocode of our parallel algorithm for generating random graphs in Fig. 4.

Theorem 1. *The parallel algorithm for generating random graphs maintains an exact stochastic process as that of the sequential algorithm and preserves all mathematical and theoretical results of the sequential algorithm.*

Proof. The parallel algorithm always selects the vertex u with the minimum degree in the sequence (Line 3), assigns d_u edges incident on u (Lines 5-23), and then proceeds with the next vertex in the sequence as the sequential algorithm would do. While assigning the first edge incident on a vertex u, all vertices in the sequence that do not create self-loops

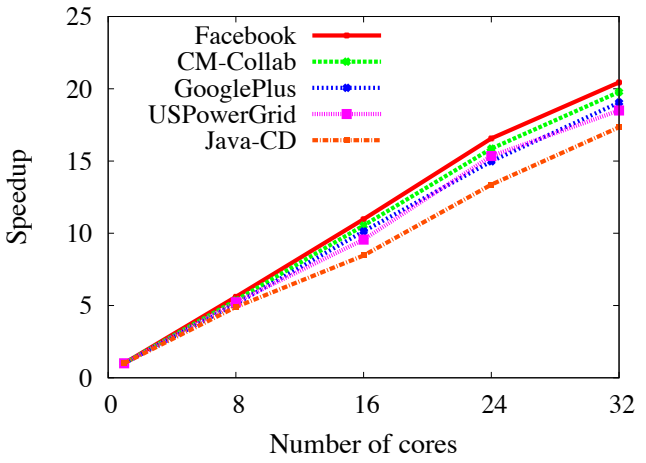

Fig. 5: Strong scaling of our parallel algorithm for generating random graphs on different data sets.

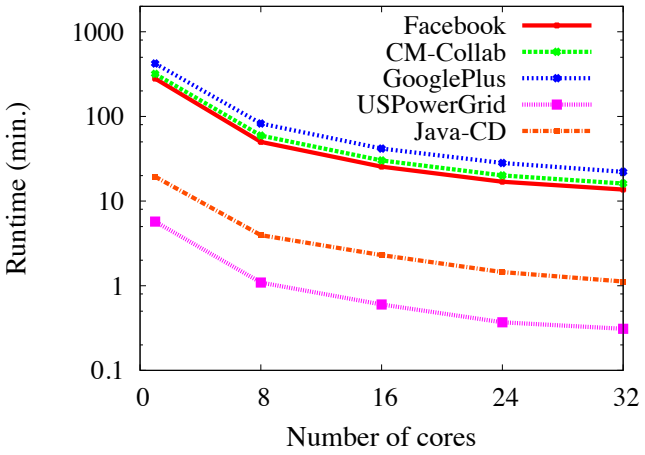

Fig. 6: Runtime of our parallel algorithm for generating random graphs on different data sets.

or parallel edges are considered as potential candidates $\mathbb{F}$ (Line 7), whereas for assigning the subsequent edges incident on u, the candidates $\mathbb{C}$ in an earlier stage are considered as the potential candidates $\mathbb{F}$ (Line 9) in a later stage. The candidate list is then computed in parallel by checking whether an edge can be assigned between u and a potential candidate v in $\mathbb{F}$ by checking whether the residual degree sequence $\ominus^{\mathbb{D}}_{u,v}$, if an edge (u, v) is assigned, is a graphical sequence by using the parallel algorithm for the Erdős-Gallai characterization (Lines 11-14). If the cardinality of the candidate list is equal to the degree d_u of vertex u, then edges are assigned between u and all vertices v in the candidate list $\mathbb{C}$ in parallel (Lines 15-19). Although this step is not explicitly mentioned in the sequential algorithm, this is obvious since the sequential algorithm would assign all d_u edges incident on u and there are no additional candidates other than the d_u candidates in $\mathbb{C}$. We parallelize this step to improve the performance of the algorithm. If the candidate list $\mathbb{C}$ has more than d_u candidates, then a vertex v is selected randomly from $\mathbb{C}$ with probability proportional to d_v, and an edge (u, v) is assigned, as the sequential algorithm would do. Hence, the parallel algorithm maintains an exact stochastic process as that of the sequential algorithm. As a consequence, all mathematical and theoretical results (except the time complexity) of the sequential algorithm are applicable to our parallel algorithm as well. ■

Theorem 2. *The time complexity of each of the core $\mathbb{P}_k$ in the parallel algorithm for generating random graphs is $\mathcal{O}\left(mn\left(\frac{n}{\mathcal{P}} + \log \mathcal{P}\right)\right)$.*

Proof. The parallel algorithm assigns m edges one by one. To assign an edge incident on a vertex u, it computes the candidate list in parallel (Line 11). Whether a vertex v is a candidate is computed using the parallel algorithm for the Erdős-Gallai characterization (Line 12), which has a time complexity of $\mathcal{O}\left(\frac{n}{\mathcal{P}} + \log \mathcal{P}\right)$. Hence, the time complexity of the parallel algorithm for generating random graphs is $\mathcal{O}\left(mn\left(\frac{n}{\mathcal{P}} + \log \mathcal{P}\right)\right)$. ■

TABLE II: Data sets used in the experiments, where n, m, and $\frac{2m}{n}$ denote the no. of vertices, no. of edges, and average degree of the networks, respectively. K denotes thousands.

Network	Type	n	m	$\frac{2m}{n}$
Facebook [36]	Social contact	6.6K	250K	75.50
GooglePlus [36]	Social contact	23.6K	39.2K	3.32
USPowerGrid [37]	Power grid	4.94K	6.6K	2.67
Java-CD [37]	Dependency	6.12K	50.3K	16.43
CM-Collab [39]	Collaboration	16.3K	47.6K	5.85

Theorem 3. *The space complexity of the parallel algorithm for generating random graphs is $\mathcal{O}(m + n)$.*

Proof. Storing the degree sequence and the edges take $\mathcal{O}(n)$ and $\mathcal{O}(m)$ space, respectively, making a space requirement of $\mathcal{O}(m + n)$. ■

C. Experimental Results

In this section, we present the data sets used in the experiments and the strong scaling and runtime of our parallel algorithm for generating random graphs.

Experimental Setup. We use a 32-core Haswell-EP E5-2698 v3 2.30GHz (3.60GHz Turbo) dual processor node with 128GB of memory, 1TB internal hard drive, and QLogic QDR InfiniBand adapter. We use OpenMP version 3.1 and GCC version 4.7.2 for implementation.

Data Sets. We use degree sequences of five real-world networks for the experiments. A summary of the networks is given in Table II. Facebook [36] is an anonymized Facebook friendship network of the students of CMU. GooglePlus [36] is an online social contact network of GooglePlus. The US-PowerGrid [37] network represents a high-voltage power grid in the western states of the USA. Java-CD [37] is a Java class dependency network of JUNG 2.0.1 [38]. CM-Collab [39] is a scientific collaboration network on the condensed matter topic.

TABLE III: A comparison of some structural properties of the random networks generated (from the degree sequences of the real-world networks) by our parallel algorithm with that of the real-world networks and random networks generated by swapping 100% edges of the real-world networks. We use average values of 20 experiments.

		Network structural properties							
							Average vertex value		
Network	**Network model**	**Triangles**	**Cliques**	**Connected component**	**Avg. shortest path length**	**Diameter**	**Betweenness centr. (x10^{-4})**	**Closeness centrality**	**Clustering coefficient**
Facebook	Real-world	2.31M	1.24M	1	2.74	8	2.63	0.37	0.28
	Our algo.	0.57M	0.40M	1	2.50	6	2.27	0.40	0.04
	Edge swap	0.54M	0.39M	1	2.49	5	2.26	0.41	0.04
GooglePlus	Real-world	18.22K	31.09K	4	4.03	8	1.28	0.25	0.17
	Our algo.	163.7K	21.96K	1.6K	3.20	5	0.69	0.24	0.22
	Edge swap	99.95K	1.27M	637	3.13	9	0.81	0.29	0.19
USPowerGrid	Real-world	651	5.69K	1	18.99	46	36.42	0.05	0.08
	Our algo.	8	6.58K	74	8.48	20	14.07	0.11	0.0008
	Edge swap	2	6.59K	88	8.49	22	13.91	0.11	0.0003
Java-CD	Real-world	0.18M	31.89K	1	2.11	7	1.82	0.48	0.68
	Our algo.	0.29M	21.34K	1	2.10	5	1.80	0.48	0.66
	Edge swap	0.19M	55.79K	1	2.00	4	1.64	0.50	0.69
CM-Collab	Real-world	68K	10.49K	726	6.63	18	2.51	0.11	0.64
	Our algo.	272	47.09K	25	4.91	12	2.39	0.20	0.0012
	Edge swap	264	47.09K	31	4.91	14	2.39	0.20	0.0010

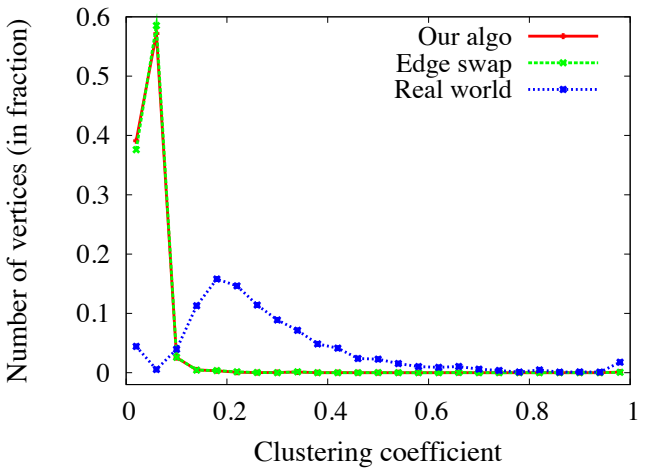

Fig. 7: A comparison of clustering coefficient distribution of the real-world Facebook network and random networks generated by our parallel algorithm and by swapping edges of the real-world graph. The distributions of the random networks almost completely overlap with each other.

Strong Scaling. The strong scaling and runtime of the parallel algorithm are shown in Figs. 5 and 6, respectively. The speedups increase almost linearly with the increase in the number of cores, and we achieve a maximum speedup of 20.5 with 32 cores on the Facebook network.

Structural Properties of the Generated Graphs. A comparison of the structural properties of the random graphs generated by our parallel algorithm with that of the real-world graphs and random graphs generated by swapping edges is given in Table III. To generate random graphs by swapping edges, 100% edges of the real-world graphs are swapped (see [21] for details). We use average values of 20 experiments. We study the number of triangles, cliques, connected components, average shortest path length, diameter, average betweenness centrality, average closeness centrality, and average local clustering coefficient of the networks. We observe that in many cases the properties of the random graphs are far away than that of the real-world graphs. The structural properties of the random graphs generated by our algorithm and by swapping edges are very close to each other in most of the cases. For example, the clustering coefficient distribution, as shown in Fig. 7, of the random graphs generated by these two methods almost completely overlap with each other, and it is difficult to distinguish them in the figure, whereas both of them lie far away than that of the real-world graph.

IV. Parallel Algorithm for Checking the Erdős-Gallai Characterization

Many variants of the Erdős-Gallai characterization have been developed and proofs have been given (see [14] and [34] for a good discussion). Such a useful result has been presented in Theorem 3.4.1 in [34], which defines the *corrected Durfee number* $\mathcal{C}$ of the degree sequence $\mathbb{D} = (d_1, d_2, \ldots, d_n)$ (sorted in non-increasing order) as

$$\mathcal{C} = |\{j : d_j \geq j - 1\}| \tag{3}$$

and showed that $\mathbb{D}$ is graphical if and only if it satisfies the first $\mathcal{C}$ inequalities of the Erdős-Gallai test. The corrected

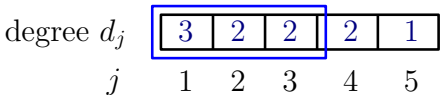

Fig. 8: For the degree sequence $\mathbb{D} = (3, 2, 2, 2, 1)$, the corrected Durfee number $\mathcal{C} = 3$.

Durfee number $\mathcal{C}$ is often significantly smaller than the number of vertices n. For example, for the degree sequence $\mathbb{D} = (3, 2, 2, 2, 1)$, the corrected Durfee number $\mathcal{C}$ is 3, as shown in Fig. 8; hence, it is sufficient to check only the first three Erdős-Gallai inequalities instead of checking all five inequalities of Eq. (2).

The sequential algorithm for checking the Erdős-Gallai characterization is quite straightforward and has a time complexity of $\mathcal{O}(n)$ [35]. A parallel algorithm for the problem has been presented in [31], which has a runtime of $\mathcal{O}(\log n)$ using $\mathcal{O}\left(\frac{n}{\log n}\right)$ EREW PRAM processors. In addition to generating random graphs, the Erdős-Gallai characterization has important applications in many other graph theory problems as well. For example, Iványi et al. [40] applied the sequential algorithm for checking the Erdős-Gallai characterization to enumerate the distinct degree sequences of simple graphs in parallel. In this section, we present a shared-memory parallel algorithm for checking the Erdős-Gallai characterization of a given degree sequence with a time complexity of $\mathcal{O}\left(\frac{n}{\mathcal{P}} + \log \mathcal{P}\right)$ using $\mathcal{P}$ processing cores. First we briefly review the current state-of-the-art sequential algorithm.

```
   ▷ Compute the corrected Durfee number
 1: i ← 1
 2: while i <= n and d_i ≥ i − 1 do
 3:     C ← i                    // corrected Durfee number
 4:     i ← i + 1

   ▷ Compute the prefix sum of the degrees
 5: H_0 ← 0
 6: for i = 1 to n do
 7:     H_i ← H_{i−1} + d_i

   ▷ Check the parity
 8: if H_n is odd then
 9:     return FALSE    // not a graphical sequence

   ▷ Compute the weights
10: d_0 ← n − 1
11: for i = 1 to n do
12:     if d_i < d_{i−1} then
13:         for j = d_{i−1} downto d_i + 1 do
14:             w_j ← i − 1
15:         w_{d_i} ← i
16: for j = d_n downto 1 do
17:     w_j ← n

   ▷ Check the Erdős-Gallai inequalities
18: for i = 1 to C do
19:     if i ≤ w_i then
20:         if H_i > i(i − 1) + i(w_i − i) + H_n − H_{w_i} then
21:             return FALSE
22:     else if H_i > i(i − 1) + H_n − H_i then
23:         return FALSE

24: return TRUE              // a graphical sequence
```

Fig. 9: A sequential algorithm [35] for checking the Erdős-Gallai characterization.

A. Sequential Algorithm

The sequential algorithm [35] is quite simple and consists of the following steps: (***i***) compute the corrected Durfee number $\mathcal{C}$, (***ii***) compute the prefix sum of the degrees, (***iii***) check the parity, i.e., whether the sum of the degrees is even or odd, (***iv***) compute the weights, which are useful in computing the right hand side of Eq. (2) in linear time, and (***v***) check the first $\mathcal{C}$ Erdős-Gallai inequalities. If the sum of the degrees is even and all the inequalities are satisfied, then the degree sequence $\mathbb{D}$ is graphical; otherwise, $\mathbb{D}$ is not graphical. The pseudocode of the sequential algorithm is given in Fig. 9.

B. Parallel Algorithm

Based on the sequential algorithm presented in Fig. 9, we present a parallel algorithm for checking the Erdős-Gallai characterization. Below we describe the methodology to parallelize the steps of the sequential algorithm.

• ***Step 1*: Compute the Corrected Durfee Number.** The corrected Durfee number can be computed in parallel in a round robin fashion, as shown in Fig. 10. Each core $\mathbb{P}_k$ computes its local corrected Durfee number $\mathcal{C}_k$. Then all the cores synchronize and the maximum value of all $\mathcal{C}_k$ is reduced as the corrected Durfee number $\mathcal{C}$.

• ***Step 2*: Compute the Prefix Sum of the Degrees.** We use a parallel version [41] of computing the prefix sum, as shown in Fig. 11. Each core $\mathbb{P}_k$ works on a chunk of size $\lceil \frac{n}{\mathcal{P}} \rceil$ of the degree sequence. First, the sum s_k of the degrees in the chunk is computed (Line 4) and then a prefix sum S_k

```
1: k ← core id

   ▷ Each core P_k executes the following in
   parallel:
2: i ← k + 1
3: C_k ← 0
4: while i <= n and d_i ≥ i − 1 do
5:     C_k ← i   // local corrected Durfee number
6:     i ← i + P

   ▷ Reduce the corrected Durfee number
7: C ← REDUCE-MAX_k C_k
```

Fig. 10: Compute the corrected Durfee number in parallel.

```
1: k ← core id

   ▷ Each core P_k executes the following in
   parallel:
2: x ← k⌈n/P⌉ + 1
3: y ← min{(k + 1)⌈n/P⌉, n}
4: s_k ← Σ_{i=x}^{y} d_i
5: In Parallel: S_k ← Σ_{j=0}^{k−1} s_j
6: Q ← S_k                    // note that S_0 = 0
7: for i = x to y do
8:     H_i ← Q + d_i
9:     Q ← H_i
```

Fig. 11: Compute the prefix sum of the degrees in parallel.

of the s_j $(0 \leq j \leq k-1)$ is computed in parallel (Line 5). Finally, each core gives a pass to the chunk and uses the value of S_k to compute the final prefix sum (Lines 6-9).

• ***Step 3*: Check the Parity.** The master core checks whether the sum of the degrees is even. If the sum is odd, then the degree sequence is not graphical. Otherwise, the algorithm proceeds to the next step.

• ***Step 4*: Compute the Weights.** The pseudocode of computing the weights in parallel is shown in Fig. 14. We first initialize (Lines 2-3) the weight array w in parallel. Then the actual weights are computed inside a `for` loop (Lines 4-10) in parallel. Due to the simultaneous nature of the parallel algorithm, there is a possibility that the same weight w_j may be updated by multiple cores in an order different than that of the sequential algorithm. To deal with this difficulty, we add two additional `if` conditions (Lines 7 and 9) as the values of w_j are only updated with larger values in the sequential algorithm. These two conditions ensure the correctness of the weight values as well as allow simultaneous parallel computation of them. Finally, the larger weights are computed in parallel in the last `for` loop (Lines 11-12).

• ***Step 5*: Check the Erdős-Gallai Inequalities.** The Erdős-Gallai inequalities can be checked in parallel in a round robin fashion. We have to check only the first $\mathcal{C}$ inequalities instead of checking all the n inequalities. This significantly improves the performance of the algorithm since $\mathcal{C} << n$ in many degree sequences, as shown later in Table IV. If any of the inequalities is dissatisfied, then the degree sequence is not graphical; otherwise, it is a graphical sequence. The pseudocode of the algorithm is presented in Fig. 15.

Theorem 4. *The time complexity of each of the core $\mathbb{P}_k$ in the parallel algorithm for checking the Erdős-Gallai characterization is $\mathcal{O}\left(\frac{n}{\mathcal{P}} + \log \mathcal{P}\right)$.*

Proof. The `while` loop in Lines 4-6 in Fig. 10 takes $\mathcal{O}\left(\frac{\mathcal{C}}{\mathcal{P}}\right)$ time, where $\mathcal{C}$ is the corrected Durfee number and $\mathcal{P}$ is the number of cores. The `reduction` in Line 7 takes $\mathcal{O}(\log \mathcal{P})$ time. Hence, the corrected Durfee number in Step 1 can be computed in $\mathcal{O}\left(\frac{\mathcal{C}}{\mathcal{P}} + \log \mathcal{P}\right)$ time. Lines 4, 5, and 7-9 in Fig. 11 take $\mathcal{O}\left(\frac{n}{\mathcal{P}}\right)$, $\mathcal{O}(\log \mathcal{P})$, and $\mathcal{O}\left(\frac{n}{\mathcal{P}}\right)$ time, respectively, where n is the number of vertices. So, the prefix sum of the degrees in Step 2 can be computed in $\mathcal{O}\left(\frac{n}{\mathcal{P}} + \log \mathcal{P}\right)$ time [41]. Checking the parity in Step 3 can be done in $\mathcal{O}(1)$ time. Each of the three `for` loops (Lines 2-3, 4-10, and 11-12) in Fig. 14 takes $\mathcal{O}\left(\frac{n}{\mathcal{P}}\right)$ time. Although the two `for` loops in Lines 4 and 6 are nested, the total number of weights updated are $\mathcal{O}(n)$. Thus computing the weights in Step 4 takes $\mathcal{O}\left(\frac{n}{\mathcal{P}}\right)$ time. The `while` loop (Lines 4-10) in Fig. 15 takes $\mathcal{O}\left(\frac{\mathcal{C}}{\mathcal{P}}\right)$ time. Therefore, the Erdős-Gallai inequalities in Step 5 are tested in $\mathcal{O}\left(\frac{\mathcal{C}}{\mathcal{P}}\right)$ time. Thus, the time complexity of the algorithm is $\mathcal{O}\left(\frac{n}{\mathcal{P}} + \frac{\mathcal{C}}{\mathcal{P}} + \log \mathcal{P}\right) = \mathcal{O}\left(\frac{n}{\mathcal{P}} + \log \mathcal{P}\right)$. ■

Theorem 5. *The space complexity of the parallel algorithm for checking the Erdős-Gallai characterization is $\mathcal{O}(n)$.*

Proof. Storing the degree sequence and the prefix sum of the degrees take $\mathcal{O}(n)$ space. ■

```
1: d_0 ← n − 1

   ▷ Initialize the weight array
2: for i = 1 to n in parallel do
3:     w_i ← 0

   ▷ Compute the weight values
4: for i = 1 to n in parallel do
5:     if d_i < d_{i−1} then
6:         for j = d_{i−1} downto d_i + 1 in parallel do
7:             if i − 1 > w_j then
8:                 w_j ← i − 1
9:         if i > w_{d_i} then
10:            w_{d_i} ← i

   ▷ Compute the larger weight values
11: for j = d_n downto 1 in parallel do
12:    w_j ← n
```

Fig. 14: Compute the weights in parallel.

```
1: k ← core id
2: flag ← TRUE                    // shared variable

   ▷ Each core P_k executes the following in
   parallel:
3: i ← k + 1
4: while i <= C and flag = TRUE do
5:     if i ≤ w_i then
6:         if H_i > i(i − 1) + i(w_i − i) + H_n − H_{w_i} then
7:             flag ← FALSE
8:     else if H_i > i(i − 1) + H_n − H_i then
9:         flag ← FALSE
10:    i ← i + P

11: OMP-BARRIER
12: return flag
```

Fig. 15: Check the Erdős-Gallai inequalities in parallel.

C. Performance Evaluation

In this section, we present the data sets used in the experiments and the strong scaling and runtime of our parallel algorithm for checking the Erdős-Gallai characterization. We use the same experimental setup as described before in Section III-C.

Data Sets. We use degree sequences of both artificial and real-world networks for the experiments. A summary of the

TABLE IV: Data sets used in the experiments, where n, m, $\frac{2m}{n}$, and $\mathcal{C}$ denote the number of vertices, number of edges, average degree, and the corrected Durfee number of the networks, respectively. `M` and `B` denote millions and billions, respectively.

Network	Type	n	m	$\frac{2m}{n}$	$\mathcal{C}$
Friendster [42]	Social	65.6M	1.8B	55.06	2959
Twitter [42]	Social	40.56M	667.7M	32.93	6842
Los Angeles (LA) [43]	Contact	16.23M	459.3M	56.59	380
New York (NY) [43]	Contact	17.88M	480.1M	53.70	387
LiveJournal (LJ) [42]	Social	4.80M	42.85M	17.68	990
SmallWorld (SW) [7]	Random	4.80M	48.00M	20.00	31

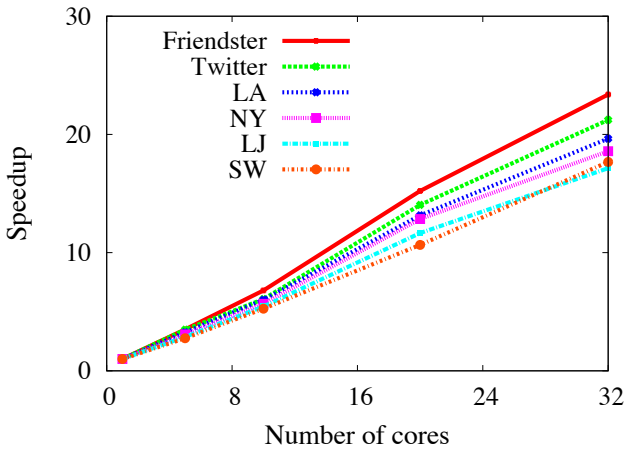

Fig. 12: Strong scaling of the parallel algorithm for checking the Erdős-Gallai characterization.

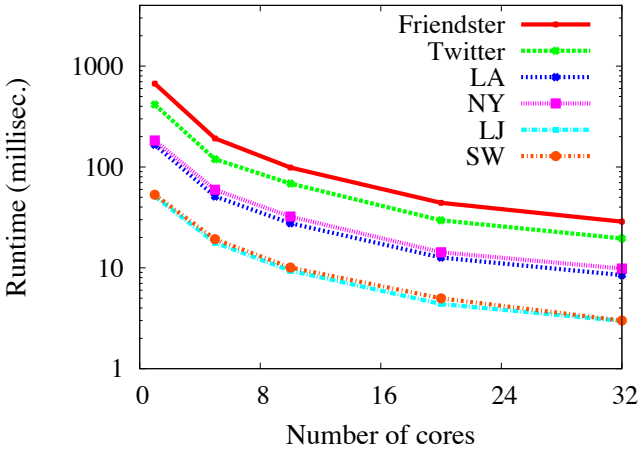

Fig. 13: Runtime of the parallel algorithm for checking the Erdős-Gallai characterization.

networks is given in Table IV. Friendster, Twitter, and LiveJournal (LJ) are real-world online social networks [42]. New York (NY) and Los Angeles (LA) are synthetic, yet realistic social contact networks [43]. The SmallWorld random network follows the Watts-Strogatz small world network model [7]. Table IV also shows that the corrected Durfee number $\mathcal{C}$ is significantly smaller than the number of vertices n for all six networks.

Strong Scaling. The strong scaling and runtime of the parallel algorithm are illustrated in Figs. 12 and 13, respectively. The speedup increases almost linearly with the increase in the number of cores. We observe better speedups for the degree sequences of larger graphs and achieve a maximum speedup of 23 with 32 cores on the Friendster graph.

V. Conclusion

We presented an efficient parallel algorithm for generating random graphs with prescribed degree sequences. It can be used in studying various structural properties of and dynamics over a network, sampling graphs uniformly at random from the graphical realizations of a given degree sequence and estimating the number of possible graphs with a given degree sequence. The algorithm never gets stuck, can generate every possible graph with a positive probability, and exhibits good speedup. We also compared several important structural properties of the random graphs generated by our parallel algorithm with that of the real-world graphs and random graphs generated by the edge swapping method. In addition, we developed an efficient parallel algorithm for checking the Erdős-Gallai characterization of a given degree sequence. This algorithm can be of independent interest and prove useful in parallelizing many other graph theory problems. We believe the parallel algorithms will contribute significantly in analyzing and mining emerging complex systems and understanding interesting characteristics of such networks.

References

[1] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos, "Power laws and the AS-level internet topology," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 4, pp. 514–524, 2003. (page 1)

[2] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002. (page 1)

[3] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015. (page 1)

[4] V. Latora and M. Marchiori, "Vulnerability and protection of infrastructure networks," *Physical Review E*, vol. 71, no. 1, p. 015103, 2005. (page 1)

[5] B. Bollobás, *Random Graphs*. Springer, 1998. (page 1)

[6] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999. (page 1)

[7] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998. (pages 1, 8, and 9)

[8] F. Chung and L. Lu, "The average distances in random graphs with given expected degrees," *Proceedings of the National Academy of Sciences*, vol. 99, no. 25, pp. 15 879–15 882, 2002. (page 1)

[9] C. Berge and E. Minieka, *Graphs and Hypergraphs*. North-Holland Publishing Company, 1973. (page 1)

[10] L. Lovász and M. D. Plummer, *Matching Theory*. American Mathematical Society, 2009, vol. 367. (page 1)

[11] S. L. Hakimi and E. F. Schmeichel, "Graphs and their degree sequences: A survey," in *Proceedings of the Theory and Applications of Graphs*. Springer, 1978, pp. 225–235. (page 1)

[12] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang, "Complex networks: Structure and dynamics," *Physics Reports*, vol. 424, no. 4, pp. 175–308, 2006. (page 1)

[13] B. Cloteaux, "Fast sequential creation of random realizations of degree sequences," *Internet Mathematics*, vol. 12, no. 3, pp. 205–219, 2016. (page 1)

[14] J. Blitzstein and P. Diaconis, "A sequential importance sampling algorithm for generating random graphs with prescribed degrees," *Internet Mathematics*, vol. 6, no. 4, pp. 489–522, 2011. (pages 1, 2, 3, 4, and 6)

[15] M. Bayati, J. H. Kim, and A. Saberi, "A sequential algorithm for generating random graphs," *Algorithmica*, vol. 58, no. 4, pp. 860–910, 2010. (pages 1 and 2)

[16] C. I. Del Genio, H. Kim, Z. Toroczkai, and K. E. Bassler, "Efficient and exact sampling of simple graphs with given arbitrary degree sequence," *PLoS One*, vol. 5, no. 4, p. e10012, 2010. (pages 1 and 2)

[17] F. Liljeros, C. R. Edling, L. A. N. Amaral, H. E. Stanley, and Y. Aberg, "The web of human sexual contacts," *Nature*, vol. 411, pp. 907–908, 2001. (page 1)

[18] T. Feder, A. Guetz, M. Mihail, and A. Saberi, "A local switch Markov chain on given degree graphs with application in connectivity of peer-to-peer networks," in *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2006, pp. 69–76. (page 1)

[19] C. Gkantsidist, M. Mihail, and E. Zegura, "The Markov chain simulation method for generating connected power law random graphs," in *Proceedings of the 5th Workshop on Algorithm Engineering and Experiments*, vol. 111. SIAM, 2003, pp. 16–25. (page 1)

[20] H. Bhuiyan, J. Chen, M. Khan, and M. V. Marathe, "Fast parallel algorithms for edge-switching to achieve a target visit rate in heterogeneous graphs," in *Proceedings of the 43rd International Conference on Parallel Processing*. IEEE, 2014, pp. 60–69. (page 1)

[21] H. Bhuiyan, M. Khan, J. Chen, and M. Marathe, "Parallel algorithms for switching edges in heterogeneous graphs," *Journal of Parallel and Distributed Computing*, vol. 104, pp. 19–35, 2017. (page 6)

[22] C. Cooper, M. Dyer, and C. Greenhill, "Sampling regular graphs and a peer-to-peer network," in *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2005, pp. 980–988. (page 1)

[23] M. E. Newman, "The structure and function of complex networks," *SIAM Review*, vol. 45, no. 2, pp. 167–256, 2003. (page 1)

[24] N. C. Wormald, "Models of random regular graphs," *London Mathematical Society Lecture Note Series*, pp. 239–298, 1999. (page 2)

[25] A. Steger and N. C. Wormald, "Generating random regular graphs quickly," *Combinatorics, Probability and Computing*, vol. 8, no. 04, pp. 377–396, 1999. (page 2)

[26] J. Kim and V. Vu, "Sandwiching random graphs: Universality between random graph models," *Advances in Mathematics*, vol. 188, no. 2, pp. 444–469, 2004. (page 2)

[27] S. L. Hakimi, "On realizability of a set of integers as degrees of the vertices of a linear graph. I," *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, no. 3, pp. 496–506, 1962. (pages 2 and 3)

[28] J. S. Liu and R. Chen, "Sequential Monte Carlo methods for dynamic systems," *Journal of the American Statistical Association*, vol. 93, no. 443, pp. 1032–1044, 1998. (pages 2 and 3)

[29] S. R. Arikati and A. Maheshwari, "Realizing degree sequences in parallel," *SIAM Journal on Discrete Mathematics*, vol. 9, no. 2, pp. 317–338, 1996. (page 2)

[30] R. M. Karp, "A survey of parallel algorithms for shared-memory machines," 1988. (page 2)

[31] A. Dessmark, A. Lingas, and O. Garrido, "On parallel complexity of maximum f-matching and the degree sequence problem," in *Proceedings of the 19th International Symposium on Mathematical Foundations of Computer Science*. Springer, 1994, pp. 316–325. (pages 2 and 7)

[32] M. Alam and M. Khan, "Parallel algorithms for generating random networks with given degree sequences," *International Journal of Parallel Programming*, vol. 45, no. 1, pp. 109–127, 2017. (page 2)

[33] P. Erdős and T. Gallai, "Gráfok előírt fokú pontokkal (Graphs with points of prescribed degrees, in Hungarian)," *Mat. Lapok*, vol. 11, pp. 264–274, 1961. (pages 2 and 3)

[34] N. V. Mahadev and U. N. Peled, *Threshold Graphs and Related Topics*. Elsevier, 1995, vol. 56. (pages 3 and 6)

[35] A. Iványi and L. Lucz, "Erdős-Gallai test in linear time," *Combinatorica*, 2011. (pages 4 and 7)

[36] R. A. Rossi and N. K. Ahmed, "An interactive data repository with visual analytics," *ACM SIGKDD Explorations Newsletter*, vol. 17, no. 2, pp. 37–41, 2016. (page 5)

[37] S. E. Abdelhamid, R. Alo, S. Arifuzzaman, P. Beckman, M. H. Bhuiyan, K. Bisset, E. A. Fox, G. C. Fox, K. Hall, S. S. Hasan *et al.*, "CINET: A cyberinfrastructure for network science," in *Proceedings of the 8th International Conference on E-Science*. IEEE, 2012, pp. 1–8. (page 5)

[38] JUNG, "Java universal network/graph framework," http://jung.sourceforge.net/, [Online; accessed 18-Aug-2017]. (page 5)

[39] M. E. Newman, "The structure of scientific collaboration networks," *Proceedings of the National Academy of Sciences*, vol. 98, no. 2, pp. 404–409, 2001. (page 5)

[40] A. Iványi, L. Lucz, P. Sótér, and S. Pirzada, "Parallel Erdős-Gallai algorithm," *Central European Journal of Operations Research*, 2011. (page 7)

[41] S. Aluru, "Teaching parallel computing through parallel prefix," in *The International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012. (pages 7 and 8)

[42] J. Leskovec and A. Krevl, "SNAP datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, [Online; accessed 18-Aug-2017]. (pages 8 and 9)

[43] C. L. Barrett, R. J. Beckman, M. Khan, V. Anil Kumar, M. V. Marathe, P. E. Stretz, T. Dutta, and B. Lewis, "Generation and analysis of large synthetic social contact networks," in *Proceedings of the 2009 Winter Simulation Conference (WSC)*, 2009, pp. 1003–1014. (pages 8 and 9)