IE 456 Term Project Final Report Fall 20/21

Instructor: *Tınaz Ekim Aşıcı**

Algorithmic Efficiency of Random Graph Generation Algorithms for Given Degree Sequences

Orkun İrsoy & Şanser Güz

INTRODUCTION

The generation of random graphs has been an attractive topic due to its various usages in daily life. Upon correct implementation, graphical models of popular topics of study such as social, biological, and internet networks prove to be very useful in analyzing miscellaneous forms of issues in network structures to deduce insights about their properties (Riordan & Spencer, 2001; Bhuiyan et al., 2017). These issues can range from the analysis of the neural transmission system of the brain to the spread of a virus, to name a few (Barabási, 2009).

However, generating random graphs may prove to be a challenging task for complex structures with many vertices and "difficult" degree sequences. Also, the difficulty is further elevated since the objective is to create simple graphs with no multiple edges or loops. The presence of these computational drawbacks proposes two main complications regarding the efficiency of algorithms. First, the algorithm may get stuck. Second, the running time of the algorithm may be too long to compute depending on the sequence of operations. Algorithms with these complications are inefficient to use in the generation of large complex networks mentioned earlier.

As the centerpiece of this study lies the sequential algorithm proposed by Blitzstein and Diaconis (Blitzstein & Diaconis, 2006). They argue that most of the previous algorithms on this topic, such as the pairing model, get stuck or produce wrong outputs, and restart until a feasible output is obtained, which significantly prolongs the running time. They claim that their algorithm never gets stuck, does not produce multiple edges or loops, and have the worst-case running time of $O(n^3d)$ for d-regular graphs.

In this paper, we aimed to implement the three algorithms presented by Blitzstein and Draconis: Havel Hakimi, the Pairing Model, and the Sequential Algorithm; and compare their computational efficiencies in random graph generation, in terms of the running time and the accuracy of the output graphs (presence of multiple edges/loops, loyalty to given degree sequence). Additionally, we made several improvements to the original sequential algorithm to increase their efficiency, by following the remarks made by the authors (Blitzstein & Diaconis, 2006).

The paper is organized as follows. In the following section, we will provide a background on the subjects; the Erdös-Rényi model, ways of generating scale-free graphs, and their practical use. The background will be followed by a proof of Erdös-Gallai graphicality condition on

*Industrial Engineering Department, Boğaziçi University

graph sequences (Erdös & Gallai, 1960; Choudum, 1986). In Section 3, we will present the algorithms in pseudo-code and a guide to our C++ code; which will be followed by an analysis of results in Section 4. The final section of the paper is devoted to our conclusion.

BACKGROUND

Graphs in general can be used to build an analogy between a real-life issue and its mathematical counterpart. If the analogy is maintained well, the various properties of the real-life issue can be analyzed or a given problem can be solved, using the graphical model. In this sense, random graph generation can be utilized in many ways. One such school of study is Agent-Based Simulation which heavily uses network structures in its models. Problems such as virus or information spread with probabilistic nature can be modeled using ABM. The analysis of such problems requires the simulation to run on different networks, thus requiring the generation of random graphs with certain properties analogous to the system under the lens.

The study on the generation of random graphs has a history reaching back to the mid-20th century. One of the major breakthroughs in the field was introduced by mathematicians Erdös and Rényi when they proposed their random graph model in 1959 (Erdös & Rényi, 1959). In their model, they generate a random graph by selecting N edges (out of possible [(n:2): N] edge candidates) at random for a graph of n vertices. Each pair of nodes are linked by an edge with an equal probability, and a random graph is formed. In their paper, they further analyze the generation process and the properties of their output regarding the connectedness. Their study formed as a solid spine to further studies to come in the field.

As the analysis on graphs and networks grew, the need for additional models raised, such a case is scale-free property. Scale-free graphs are defined as graphs that follow scale-free power-law, meaning that they preserve their node characteristics independent of the size of the graphs (Barabási & Albert, 1999). It has been argued that many natural networks in real life have scale-free property, including biological and social networks (Cohen et al., 2003). Due to such applicability, the generation of scale-free graphs has also been a topic of interest in graph theory and network science.

One famous model of scale-free graph generation is the preferential attachment model of Barabasi and Albert (Barabási & Albert, 1999). In their model, an initial simple graph is constructed randomly, and new nodes are added in each step, with a given number of desired edges. Newly introduced nodes are more inclined to connect with the well-connected vertices, they have "preferences”. The model yields a scale-free output graph, with several hubs (central pivot vertices).

However, recently an article on Nature argued that empirically, scale-free network structures are rare in real life (Broido & Clauset, 2019). Authors claimed that the assumption of their wide-ranged applicability is not well-grounded, and the studies should use them cautiously. Having supporters and challengers, it is safe to assume that the topic remains controversial.

In this paper, we aimed to review the existing algorithms in the literature that includes Havel Hakimi, the Pairing Model, and the Sequential Algorithm, and compare their algorithmic efficiencies (Hakimi, 1962; Havel, 1955; Bollobás, 1980; Blitzstein & Diaconis, 2006). In this

regard, we will run the algorithms with several randomly generated graphical degree sequences of different properties and measure their total running times. The outputs will be reviewed with hand-written functions to check whether they satisfy the desired conditions of the inexistence of multiple edges or loops and to check their commitment to the given degree sequence. Also, we tried to implement some changes on the existing sequential algorithm to observe the effects of remarked improvements (by Blitzstein and Diaconis) on algorithmic efficiency.

Proof of Erdös-Gallai Graphicality Criterion

A major subject to be tackled in graph generation studies is the graphicality of the input degree sequence. Erdös-Gallai proposed their criterion for graphicality in 1960 (Erdös & Gallai, 1960). If a given degree sequence of size n, having the sum of degrees even, satisfies n number of inequalities, then the sequence is graphical. The criterion is also presented by Blitzstein and Diaconis in their paper, as Theorem 1 (Blitzstein & Diaconis, 2006). The criterion follows:

$$Let\ d{:}\{d_i, i\ \epsilon\ [1,n]\}\ consist\ of\ nonnegative\ integers\ d_1 \geq d_2 \geq \cdots \geq d_p$$

$$\sum_{i=1}^{p} d_i = s,\ \ s\ even. \quad d\ is\ graphical\ if\ and\ only\ if{:}$$

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{p} min\ (d_i, k), \qquad for\ every\ 1 \leq k \leq p$$

An extensive proof by induction of the theorem is made by Chodum (Choudum, 1986). Proof takes an initial degree sequence satisfying the criterion, removes two degrees and shows that the remaining degree sequence still satisfies the criterion. Proof by Chodum is explained as follows.

Chodum's proof induction hypothesis on s, sum of degrees. First, show that the theorem is satisfied for degree sequences with s = 0 and s = 2.

<u>For s = 0</u>

$$\sum_{i=1}^{k} d_i = 0, \qquad k(k-1) \geq 0 \qquad \sum_{i=k+1}^{p} min\ (d_i, k) = 0$$

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{p} min\ (d_i, k), \qquad for\ every\ 1 \leq k \leq p$$

- d is found to be graphical for s=0

<u>For s = 2</u>

$$d_1 = d_2 = 1, \quad d_i = 0 \quad for\ 2 < i \leq p$$

$$\sum_{i=1}^{k} d_i = 1 \qquad k = 1 \qquad ; \sum_{i=1}^{k} d_i = 2 \qquad k \geq 2$$

$$\sum_{i=k+1}^{p} \min(d_i, k) = 1 \quad for\ k = 1 \quad ; \quad \sum_{i=k+1}^{p} \min(d_i, k) = 0 \quad for\ k \geq 2$$

$$k(k-1) = 0 \quad k = 1 \quad ; \quad k(k-1) \geq 2 \quad k \geq 2$$

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{p} \min(d_i, k), \quad for\ every\ 1 \leq k \leq p$$

- d is found to be graphical for s=0

Now, next step is to try for larger sequences, having even degree s. Assume π: $d_1 \geq d_2 \geq \cdots \geq d_p$ be a sequence satisfying the criterion, having sum s. For the smallest integer t that satisfies $d_t > d_{t+1}$, set t to p-1. Define a new degree sequence π^*: $\{d_i^*\}$ which can be shown as follows.

$$\pi^*: d_1 \geq d_2 \geq \cdots \geq d_{t-1} > d_t - 1 \geq \cdots \geq d_{p-1} > d_p - 1$$

$$\sum_{i=1}^{p} d_i^* = s - 2$$

If the new sequence is also graphical, criterion should be satisfied for each $1 \leq k \leq p$. Chodum presents the 5 possible cases and prove Graphicality for each of them.

$$(a, b) - 1 \leq (a - 1, b) \qquad (\text{א})$$

Case 1: $k \leq t$

$$\pi: \sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{p} \min(d_i, k), \quad for\ every\ 1 \leq k \leq p$$

$$\pi^*: \sum_{i=1}^{k} d_i^* = \sum_{i=1}^{k} d_i - 1 \quad since \quad d_t^{\pi^*} = d_t^{\pi} - 1 \ is\ the\ only\ differing\ entry$$

$$\pi^*: \sum_{i=k+1}^{p} \min(d_i^*, k) = \sum_{i=k+1}^{p} \min(d_i, k) - 1 \quad since\ d_p^{\pi^*} = d_p^{\pi} - 1$$

$$\pi^*: \sum_{i=1}^{k} d_i - 1 \leq k(k-1) + \sum_{i=k+1}^{p} \min(d_i, k) - 1, \quad for\ every\ t \leq k \leq p$$

End of Case 1

π^* is found to be graphical for $k \leq t$

Case 2: $1 < k \leq t - 1 \ \ and \ \ d_k \leq k - 1$

$$\pi^*: \sum_{i=1}^{k} d_i^* = k * d_k, \quad since\ k \leq t - 1$$

$$\pi^*: \quad k * d_k \leq k(k-1), \qquad since\ d_k \leq k-1$$

$$\pi^*: \sum_{i=k+1}^{p} min\ (d_i, k) \geq 0$$

$$\pi^*: \sum_{i=1}^{k} d_i^* \leq k(k-1) + \sum_{i=k+1}^{p} min\ (d_i^*, k), \qquad for\ every\ 1 \leq k \leq t-1\ \&\ d_k \leq k-1$$

End of Case 2

π^* is found to be graphical for $1 < k \leq t-1$ *and* $d_k \leq k-1$

Case 3: $1 \leq k \leq t-1$ *and* $d_k = k$

Case 3.1: $k+2 \leq p-1$

$$\sum_{i=k+2}^{p} d_i^* \geq 2 \qquad as \quad d_i^* \geq 1$$

C**ase 3.2:** $k+2 \geq p$

$$k = p-2, \ \ t = p-1 \quad as \quad k < t < p$$

$$\sum_{i=1}^{p} d_i^* = k * (k+1) + d_p = s \qquad If\ s\ is\ even, d_p\ should\ be\ even\ as\ well$$

$$\sum_{i=k+2}^{p} d_i^* \geq 2$$

End of Case 3.1 and 3.2, continuing Case 3

$$\pi^*: \sum_{i=1}^{k} d_i^* = k^2, \qquad since\ k \leq t-1\ \&\ d_k = k$$

$$k^2 - k + d_{k+1} \leq k^2 - k + d_{k+1} + \sum_{i=k+2}^{p} d_i^* - 2 \ \ since \sum_{i=k+2}^{p} d_i^* \geq 2$$

$$= k^2 - k + d_{k+1} + d_{k+2} + \cdots d_t - 1 \ + \cdots d_p - 1$$

$$\leq k(k-1) + \sum_{i=k+1, i\neq t}^{p-1} (d_i, k) \ + (d_t - 1, k) \ + \left(d_p - 1, k\right)\ from\ \aleph$$

$$\pi^*: \sum_{i=1}^{k} d_i^* \leq k(k-1) + \sum_{i=k+1}^{p} min\ (d_i^*, k), \qquad for\ every\ 1 \leq k \leq t-1\ \&\ d_k = k$$

End of Case 3

π^* is found to be graphical for $1 < k \leq t-1 \;\; and \;\; d_k = k$

Case 4: $1 < k \leq t-1 \;\; and \;\; d_k \geq k \;\; and \;\; d_p \geq k+1$

$$(d_j, k) = (d_j - 1, k) = k \quad as \;\; d_p \geq k+1$$

$$\sum_{i=k+1}^{p} min\,(d_i, k) = \sum_{j=k+1}^{p} min\,(d_j^*, k) \;\; no\ changes\ on\ the\ right\ hand\ side$$

$$\pi^*: \quad \sum_{i=1}^{k} d_i^* \leq k^2 - k + \sum_{j=k+1}^{p} min\,(d_j^*, k) \quad for \;\; 1 \leq k \leq t-1$$

End of Case 4

π^* is found to be graphical for $1 < k \leq t-1 \;\; and \;\; d_k \geq k \;\; and \;\; d_p \geq k+1$

Case 5: $1 < k \leq t-1 \;\; and \;\; d_k \geq k+1 \;\; and \;\; d_p < k+1$

$$\pi: \sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{j=k+1}^{p} min\,(d_j, k) \;\; since\ we\ assume\ \pi\ holds$$

$$\sum_{j=k+1}^{p} min\,(d_j, k) - 1$$

$$= \sum_{j=k+1}^{p} min\,(d_j^*, k) \quad then, \pi^*\ satisfies\ EG\ only\ if\ \pi\ is\ not\ an\ equality$$

$$\sum_{i=1}^{k} d_i = k(k-1)$$

$$+ \sum_{j=k+1}^{p} min\,(d_j, k)\ could\ be\ only\ problematic\ case.\ Assume\ it\ holds:$$

$$r: smallest\ integer\ s.t. \qquad d_{t+r-1} \leq k$$

$$\sum_{j=k+1}^{p} min\,(d_j, k) = (t+r-k)k + \sum_{j=t+r+1}^{p} d_j \;\; by\ the\ definition\ of\ r\ and\ t\ then;$$

$$\sum_{i=1}^{k} d_i = k * d_k = k(k-1) + (t+r-k)k + \sum_{j=t+r+1}^{p} d_j$$

$$= k(t+r-1) + \sum_{j=t+r+1}^{p} d_j \quad Multiplying\ both\ sides\ with\ \frac{k+1}{k}$$

$$(k+1) * d_k = (k+1)(t+r-1) + \frac{k+1}{k} \sum_{j=t+r+1}^{p} d_j$$

$$Also \sum_{i=1}^{k+1} d_i = (k+1)d_k \quad then$$

$$\sum_{i=1}^{k+1} d_i = (k+1)(t+r-1) + \frac{k+1}{k} \sum_{j=t+r+1}^{p} d_j$$

$$(k+1)(t+r-1) + \frac{k+1}{k} \sum_{j=t+r+1}^{p} d_j > (k+1)k + (t+r-k-1)(k+1) + \sum_{j=t+r+1}^{p} d_j \ (since \frac{k+1}{k} > 1)$$

$$(k+1)k + (t+r-k-1)(k+1) + \sum_{j=t+r+1}^{p} d_j = (k+1)k + \sum_{j=k+2}^{p} (d_j, k+1)$$

$$\sum_{i=1}^{k+1} d_i > (k+1)k + \sum_{j=k+2}^{p} (d_j, k+1) \ which\ is\ a\ contradiction\ \pi\ satisfying\ EG$$

$$Hence;\ \pi^*: \sum_{i=1}^{k} d_i^* \leq k(k-1) + \sum_{j=k+1}^{p} min\ (d_j^*, k)$$

End of Case 5

π^* is found to be graphical for $1 < k \leq t-1 \ \ and \ \ d_k \geq k+1 \ \ and \ \ d_p < k+1$

End of Cases

$$\therefore \pi^*\ is\ graphical\ for\ all\ 1 < k \leq t$$

Thus, it can be argued that graphicality is not lost upon removal of one degree each from two elements of π (here t and p). Repeat the process many times -pick out two vertices at each iteration and the remaining sequence is also graphical- till the sequence is reduced to the zero vector. If we let pairs of degrees removed at each iteration to represent an edge, the recursive application of the process would yield a graph (not necessarily a simple graph). Hence, graphicality is proved.

ALGORITHMIC MODELS

As previously stated, three different graph generating algorithms (and an additional improved version of the last algorithm) are implemented on randomly generated degree sequences to compare their efficiencies. In the following sections, models and their implementations will be elaborated in plain language accompanied by their pseudo-codes. For the algorithms, C++ programming language is used as the medium. The entire C++ code is submitted as an additional material. A brief guide to our coding is provided at the end of this section.

4.a. Recursive Application of Havel-Hakimi Theorem:

The recursive test that was founded by Havel (Havel, 1955) and Hakimi (Hakimi, 1962) separately, can be also used to derive the realization of a sequence. The recursive test starts with selecting the maximum degree in the sequence. Then, that vertex is connected with the highest order vertices until the degree requirement is satisfied. If the resulting degree sequence is graphical, then the initial degree sequence is also graphical. When applied recursively, the algorithm stops when the final degree sequence becomes zero and a realization of the sequence is obtained. Further information about the proof can be found in (Blitzstein & Diaconis, 2006).

Implementation of the test and the graph generation algorithms performed separately.

Pseudo Code for Havel-Hakimi Test:

Input: Degree sequence

Output: Boolean (Graphicality)

1. Sort the degree sequence in descending order.
2. Pick the first element as d_{max}. If d_{max}=0 then stop. (The sequence is graphical.)
3. If d_{max} > number of positive elements in the sequence, then stop. (The sequence is not graphical.)
4. Erase the first element of the sequence and remove 1 from the first d_{max} elements of the sequence. Go to Step 1.

Pseudo Code for Havel-Hakimi Graph Generation:

Input: Degree Sequence (with size n)

Output: A simple graph with the given degrees.

1. Create n nodes and initialize their desired degrees with the degree sequence.
2. Sort nodes with respect to their desired degrees in descending order.
3. Pick the desired degree of the first node as d_{max}. If d_{max} = 0 then stop. (The sequence is graphical, return the adjacency lists of vertices.)
4. If d_{max} > number of nodes with positive desired degree, then stop. (The sequence is not graphical.)
5. Link the first node with the first d_{max} nodes after itself. (update desired degree, current degree, and adjacent vertex list.) Go to 1.

Since the algorithm chooses one vertex at a time and creates links until the desired degree of that vertex is satisfied, the existence of multiple edges is prohibited. Moreover, it always picks among the vertices different from the node with the desired degree d_{max} which does not allow loops. Sorting is used due to the easier implementation of the algorithms. For the test "quicksort" is used whereas sorting of nodes with respect to desired degrees is performed by "bubble sort".

4.b. The Pairing Model for General Degree Sequences

Another model to be examined is known as the pairing model. The original pairing model is used to generate a regular graph for a given number of vertices n and a uniform degree d for each vertex. At each stage of the iteration, the algorithm pairs two unmatched vertices with residual degrees larger than zero and links them (Bollobás, 1980; Blitzstein & Diaconis, 2006). There are several algorithms based on the pairing model. One such improvement allows the generation of simple graphs for general degree sequences (Steger & Wormald, 1999).

Our implementation takes a general degree sequence as input. At each step, available vertices are listed, and one vertex is selected randomly. Selected vertex and a randomly selected potential neighbor are paired and linked. The process continues until residual degrees of all vertices are reduced to zero.

Pseudo Code for The Pairing Model for General Degree Sequences

Input: Degree Sequence (with size n)

Output: A simple graph with the given degrees.

1. Initialize the nodes with the given degree sequence.
2. Generate a set of available vertices, A with desired degrees greater than zero.
3. Select a random vertex v in A, and remove v from A.
4. Remove all vertices in A that are already connected with v.
5. If A is an empty set, restart the algorithm.
6. Select a random vertex h in A.
7. Link v to h (update desired degree, current degree, and adjacent vertex list.).
8. If the sum of desired degrees of all vertices is zero, stop (return the adjacency lists of vertices).
9. If not, go back to step 2.

One major insight from the sequence of events is that the algorithm restarts in step 5 when the selected random vertex v has no available neighbor candidates. This does not propose much of a problem for small-sized problems, but for higher degree levels, the probability of getting stuck significantly increases which makes the algorithm stuck and restart over and over again.

Since selected vertex v is immediately removed from the list of available vertices in step 3, the algorithm would never produce loops. Moreover, each link to be established also considers the existing links in step 4, multiple edges are prohibited. Therefore, we can safely conclude that the output will be a simple graph.

4.c. The Sequential Algorithm of Blitzstein and Diaconis

Blitzstein and Diaconis (2006) propose a sequential algorithm to construct graphs with a given degree sequence. Detailed pseudo-code of the algorithm can be found in their paper. Simply, the algorithm starts with the vertex having the minimum positive residual degree and creates a set of vertices by making a graphicality check. Vertices to link have a probability of selection proportional to their residual degrees. Graphicality check at every iteration ensures that the

algorithm never stuck thus providing a significant advantage compared to algorithms having a positive probability to get stuck (Blitzstein & Diaconis, 2006).

The initial version of the model built from the pseudo-code of Blitzstein and Diaconis turned out to be less efficient than the previous models described in this section in terms of running time. We then implemented several remarks for improvement they presented in their paper in order to decrease the algorithm's computational complexity and consequently the running time.

Improvement 1: Erdös-Gallai graphicality

In the original method, we used the Havel-Hakimi graphicality test to test the graphicality of the candidate vectors. Following the remark made by the authors, we then switched to use the Erdös-Gallai graphicality test. A mathematical model of the Erdös-Gallai graphicality criterion can be found at the end of the Background Section. We present our pseudo-code for the test as follows:

Pseudo Code for Erdös-Gallai Graphicality Test:

Input: Degree sequence (d)

Output: Boolean (Graphicality)

1. Sort the degree sequence d in descending order.
2. If the sum of the degree sequence is odd; return False.
3. For all integers k in [1,n], do;
    a. If $\sum_{i=1}^{k} \quad d_i > k(k-1) + \sum_{i=k+1}^{n} \quad min(k, d_i)$, return False
4. Return True.

We found that the switch to the Erdös-Gallai test from the Havel-Hakimi improved the running time and updated the original algorithm accordingly.

Improvement 2: Corrected Durfee Number

Mahadev and Peled show that if the degree sequence satisfies the first durfee number (m) many of the Erdös-Gallai inequalities, it is graphical (Mahadev & Peled, 1995). Test for graphicality is repeatedly used for every iteration of the algorithm. Thus, a further improvement of graphicality testing can be very beneficial for easing computational difficulty.

In this regard, the Erdös-Gallai Graphicality test is revised to test only the first m of n inequalities in step 3. Updated pseudo-code is as follows:

Pseudo Code for Improved Erdös-Gallai Graphicality Test:

Input: Degree sequence (d)

1. Sort the degree sequence d in descending order.
2. If the sum of the degree sequence is odd; return False.
3. Compute the largest m that satisfies $d_m > m - 1$
4. For all integers k in [1,m], do;
    a. If $\sum_{i=1}^{k} \quad d_i > k(k-1) + \sum_{i=k+1}^{n} \quad min(k, d_i)$, return False
5. Return True.

Improvement 3: Randomly accepting first graphical pair

The last improvement was randomly accepting the candidate vertices as long as they are consistent with the graphicality test instead of assigning probabilities to candidates using their residual degrees. Moreover, Blitzstein and Diaconis (2006) state that for a specific index, a candidate at a later stage must be presented at the earlier stages. Using this property, the number of graphicality checks have been reduced. The pseudo-code of the final version of the algorithm

Pseudo Code for Improved Sequential Algorithm

Input: A graphical degree sequence

Output: A simple graph with the given degrees

1. If the sum of all degrees in the sequence equals 0 stop.
2. Find the degree with the minimal positive entry (d_{min}).
3. While $d_{min} \neq 0$
    a. Find a non-zero element in the degree sequence other than d_{min} that satisfies the Improved Erdös-Gallai Graphicality Test (d_{select})
    b. Create an edge between the node with d_{min} and the node with d_{select}.
    c. Update the degree sequence.
4. Go to Step 1.

Remarks on C++ Implementation

To build graphs, the Node class is introduced with attributes

- ID (int): ID of the current node
- currentdeg (int): Number of links that the node currently has, initially equals to zero.
- desireddeg (int): Residual degree of the node, initially equals to degree determined by the given sequence.
- adj_ver (vector<int>): Integer vector that stores the current adjacent vertices.

While constructing links, for both nodes, IDs are added to the adjacent vertices vector, current degrees are increased, and desired degrees are decreased by one. Some additional global functions are used to perform basic operations:

- CreateNodes: Initialize the nodes with the given degree sequence.
- ResetNodes: Deletes the existing nodes and initializes new nodes with the given degree sequence.
- DeleteNodes: Deletes the existing nodes.
- SortNodesDegree: Sort nodes with respect to their desired degree in descending order. (Bubble Sort)
- SortNodesID: Sort nodes with respect to their IDs. (Bubble Sort)
- Sort: Sort a given vector in descending order. (Bubble Sort)
- quicksort: Sorts a given vector in descending order using recursion.

- Vectorsum: Return the sum of the elements in a given vector.
- minVectorsum: In Erdös-Gallai Test return the sum$\sum_{i=k+1}^{n} \; min(k, d_i)$ for given vector d_i and k,n values.
- CountPositive: Returns the number of positive elements in a given vector.
- CountPositiveDegree: Returns the number of nodes with positive desired degree in a given node vector.

To perform tests three functions are aggregated in one function:

- CheckLoop: Returns true if any vertex is added in its own adjacency vector.
- CheckMultipleEdges: Returns true if any adjacency vector contains duplicate values.
- CheckDegreeSeq: Returns true if the degree sequence of the created graph is the same with the given degree sequence.

RESULTS

This section is devoted to the comparison of the performance of four algorithms for different degree sequences. Outcomes of measure will be the running time of each algorithm, and the accuracy of the output to the following desired properties: no multiple edges are allowed, no loops are allowed, output graph should have the same degree sequence as the given input.

As the input degree sequences, we have used three specific lines of sequences:

1. Regular Graph Sequences: Given (n, d) a degree sequence of size n is created with each degree equals to d.
2. Random Graph Sequences: Given (n, p) a degree sequence of size n is created by adding a new element at each iteration. Every time an element is added, for every possible pair with the existing elements in the sequence, there is a probability p that results in a unit increase for the pair. (The Erdös-Rényi G(n,p) Model)
3. Scale-free Graph Sequences: Given n,k a negative exponential function is used so that the degree distribution will result in scale-free property. (The Barabási-Albert Model)

Graphical degree sequences for each specific type are generated using hand-written functions, for sizes 25, 50, 75 and 100. For each regular graph, a low and a high degree sequence are used. For random graph sequences, we selected two different connection probabilities, 0.1 and 0.2; while for the scale-free graph sequences rates 1.5 and 2 are utilized. The number of generated graphical degree sequences totals 24. Sequences are provided as input files of .txt format.

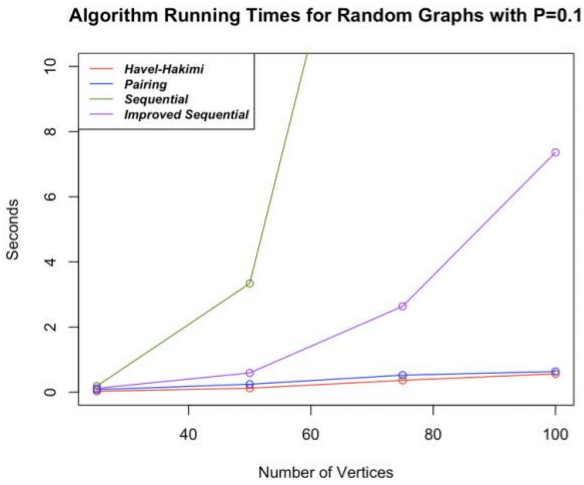
Havel-Hakimi, Pairing, Sequential, and Improved Sequential Algorithms are then run for the generated degree sequences. Running times of algorithms for each case are presented in the following table.

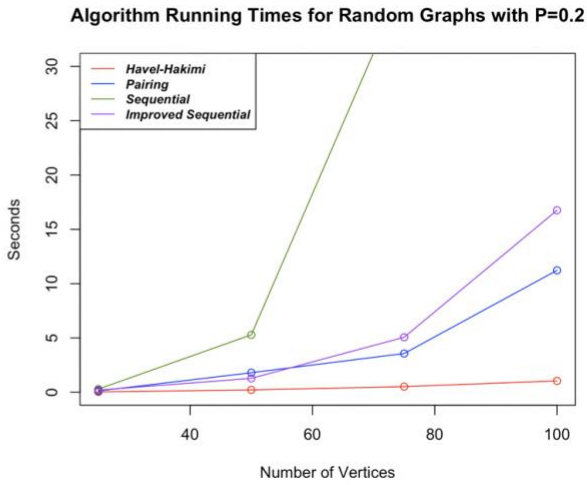
Degree Sequence Properties				Algorithm Running Times				Average Number of Restarts for Pairing Model
Degree Seqence	Size	Type	Property	Havel-Hakimi	Pairing*	Sequential	Improved Seq	
1	25	Regular	d = 6	0.0388551	0.103637	0.409863	0.130503	3.1
2	25	Regular	d = 10	0.0618314	0.160716	0.740072	0.246077	6.5
3	25	Random	p = 0.1	0.0283148	0.0787171	0.188847	0.119358	3.2
4	25	Random	p = 0.2	0.0333145	0.11839	0.287628	0.192977	8.8
5	25	Scale-Free	rate = 2	0.0207659	0.097397	0.137265	0.0908613	31.3
6	25	Scale-Free	rate = 1.5	0.0298142	0.71821	0.162758	0.115912	600.8
7	50	Regular	d = 10	0.189461	0.270819	6.98165	1.42989	2.5
8	50	Regular	d = 15	0.429515	0.563664	9.75081	2.68971	10.7
9	50	Random	p = 0.1	0.12229	0.244626	3.33919	0.592267	10.6
10	50	Random	p = 0.2	0.210338	1.79601	5.28161	1.27721	48
11	50	Scale-Free	rate = 2	0.12244	0.146086	1.01545	0.368914	3.3
12	50	Scale-Free	rate = 1.5	0.128054	1.20187	1.16299	0.367721	318.3
13	75	Regular	d = 10	0.392955	0.40418	30.2664	3.94385	3.3
14	75	Regular	d = 20	0.80031	1.7004	60.5996	8.74489	17.5
15	75	Random	p = 0.1	0.365453	0.522807	22.0974	2.63709	14
16	75	Random	p = 0.2	0.514326	3.5647	37.7456	5.06017	78.8
17	75	Scale-Free	rate = 2	0.183979	0.213583	4.56749	0.954837	5
18	75	Scale-Free	rate = 1.5	0.188961	1.24656	5.73102	1.02605	200.8
19	100	Regular	d = 15	1.16826	0.939436	148.6	12.5025	5.9
20	100	Regular	d = 25	1.38992	3.83908	239.403	23.7906	23.9
21	100	Random	p = 0.1	0.563357	0.634866	87.9589	7.36224	5.7
22	100	Random	p = 0.2	1.04326	11.226	168.91	16.7548	113.8
23	100	Scale-Free	rate = 2	0.278481	0.261741	10.1756	1.38302	3.8
24	100	Scale-Free	rate = 1.5	0.283392	2.0309	19.2088	2.00646	200.6
All runs yielded accurate simple graphs with the given degree sequence								
** 10 replications were taken for the Pairing model, average running times are reported*								

For all cases, all algorithms produced consistent outputs. That is, simple graphs with no multiple edges or loops, that are consistent with the given degree sequences. Depending on the degree sequence and random seed, the number of restarts Pairing Algorithm went through changes significantly. Thus, 10 replications were taken for the pairing model. Average running times are reported in the respective column. Additionally, average numbers of restarts are also reported for each case, as seen in the last column.

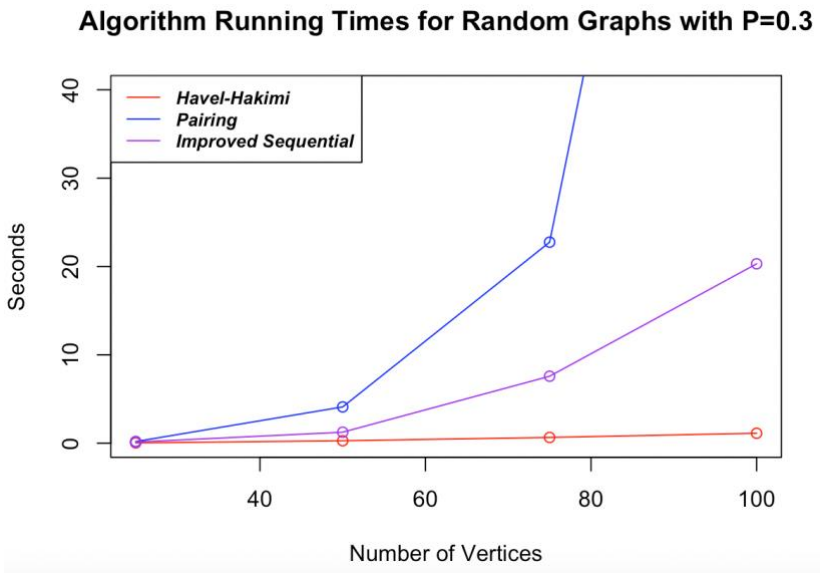
To compare the running times in a given experimental design, each running time row is colored on a white to red scale where the red represents the longer computational time. The first conclusion from the table is that the improvements to the sequential algorithm create a significant increase in computational efficiency. Except for the scale-free sequence with a high degree of size 25, the Sequential Algorithm performs the worst in terms of computational time among the four algorithms. However, the Improved Sequential Algorithm usually results in closer running times to the Pairing Model and often beats it as in the case of dense scale-free sequences. The Havel-Hakimi model is consistent among different degree sequences and always provides the shortest running time.

In order to analyze the performance of each algorithm for different cases, resulting running times are plotted for each line of sequence.



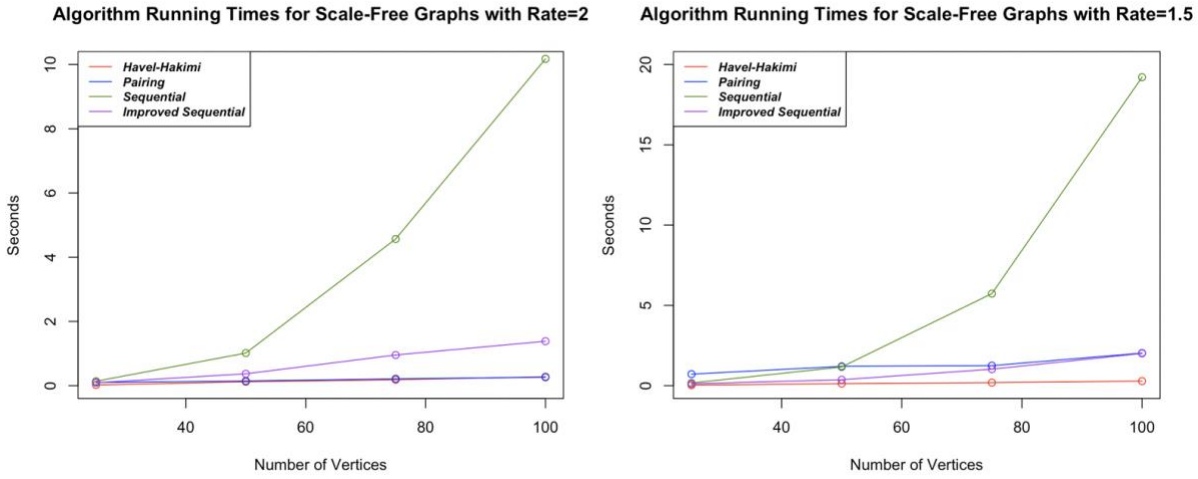


For random graphs, the Sequential Algorithm quickly diverges as the size of the sequence increases. The Improved Sequential Algorithm, on the other hand, shows high order polynomial increase with respect to sequence size which is consistent with the running time computation of the authors (Blitzstein & Diaconis, 2006). The Pairing Algorithm and the Havel-Hakimi Algorithm exhibit a linear increase in sequence size for the low degree random graphs while the behavior of the former changes as the average degree of the sequence increases. For the high degree case, the Pairing Algorithm results with a high order polynomial time and a significant increase in the average number of trials (see Table 1) which indicates the algorithm is sensitive to the average number of degrees. To investigate this claim, additional runs (Sequences 25, 26, 27, 28) are taken for random graphs with a higher average degree.

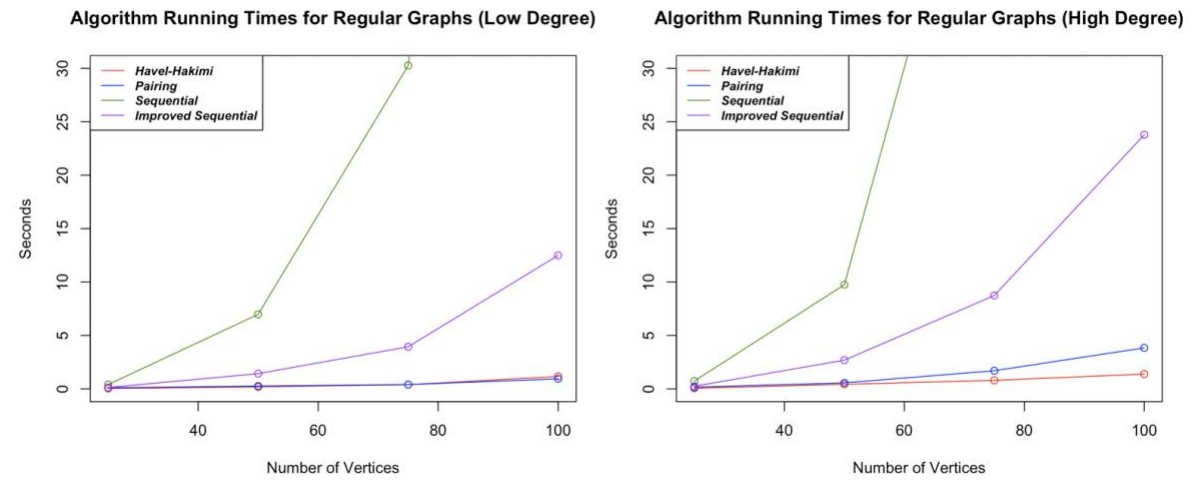


As observed in the above figure, Pairing Algorithm diverges as the size grows for higher degree ($p = 0.3$) random graphs, leaving the Improved Sequential Algorithm as the second-best in terms of running time.

The disparity between the improved and the initial version of the Sequential Algorithm is persistent in the scale-free sequences as well. Parallel to the observation in random graphs, the efficiency of the Pairing Algorithm is reduced as the density increases. The Havel-Hakimi Algorithm produces almost the same results for different rates and minuscule increases as the size grows. Similarly, different rates do not create significantly different running times for the Improved Sequential Algorithm whereas the increasing size produces a linear increase in running time. Due to the decreased efficiency of the Pairing Algorithm for high-density sequences, the Improved Sequential Algorithm often performs better while the Havel-Hakimi always provides the best computational time.



The last type of degree sequence is the regular graph sequence. The authors state that the worst-case scenario for the running time of the Sequential Algorithm is $O(dn^3)$ for the regular graphs (Blitzstein & Diaconis, 2006) which is consistent with the results. Running times grow exponentially as graph sizes increase. The improved sequential algorithm shows a sub-exponential growth with behavior more flattened, and manageable for moderate sample sizes. Pairing model and Havel-Hakimi is clearly superior to both versions of sequential algorithms, sailing way below two exponential curves, with running times not exceeding 3 seconds.



CONCLUSION

Based on our experimentation, the recursive application of Havel-Hakimi proved to be the most efficient method in generating random simple graphs for any graph type within the experimentation. As the graph size increases, the running time of the algorithm increases polynomially, yielding consistent outputs.

The efficiency of the pairing model significantly decreases as the graph degrees increase. Also, depending on the input degree sequence, the standard deviation of the running times between replications for the same case can be very high. However, overall the algorithm yields consistent results within a moderate amount of time.

Though its output is consistent, the original sequential algorithm is found to be the least efficient method for any type of graph. The improved version obtained after following the remarks made by the authors is much more efficient than the original algorithm. It can compete with the pairing model and the Havel-Hakimi in terms of running time. This shows that modifications on the type of the graphicality test (Improvement 1), number of inequalities to check (Improvement 2), and way of edge selection (Improvement 3) are significantly influential on algorithmic efficiency, effectively reducing running times.

Overall, Havel-Hakimi advances as the best method without any disadvantages compared to other algorithms. The Pairing model works fine but should not always be considered as the second-best method, since for random graphs with higher degrees, the algorithm is stuck with a high probability over and over again. For those cases, the improved sequential algorithm performs much better.

REFERENCES

Barabási, A. L. (2009). Scale-free networks: A decade and beyond. *Science*, *325*(5939), 412–413. https://doi.org/10.1126/science.1173299

Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*. https://doi.org/10.1126/science.286.5439.509

Bhuiyan, H., Khan, M., & Marathe, M. (2017). A parallel algorithm for generating a random graph with a prescribed degree sequence. *Proceedings - 2017 IEEE International Conference on Big Data, Big Data 2017*, *2018-January*(August), 3312–3321. https://doi.org/10.1109/BigData.2017.8258316

Blitzstein, J., & Diaconis, P. (2006). *A SEQUENTIAL IMPORTANCE SAMPLING ALGORITHM FOR GENERATING RANDOM GRAPHS WITH PRESCRIBED DEGREES By Joseph Blitzstein and Persi Diaconis ↽ Stanford University*. *June*, 1–35.

Bollobás, B. (1980). A Probabilistic Proof of an Asymptotic Formula for the Number of Labelled Regular Graphs. *European Journal of Combinatorics*. https://doi.org/10.1016/S0195-6698(80)80030-8

Broido, A. D., & Clauset, A. (2019). Scale-free networks are rare. *Nature Communications*, *10*(1), 1–10. https://doi.org/10.1038/s41467-019-08746-5

Choudum, S. A. (1986). A simple proof of the Erdös-Gallai theorem on graph sequences. *Bulletin of the Australian Mathematical Society*, *33*(1), 67–70. https://doi.org/10.1017/S0004972700002872

Cohen, R., Havlin, S., & Ben-Avraham, D. (2003). *Handbook of Graphs and Networks: From the Genome to the Internet (Chapter 4 - Structural Properties of Scale-Free Networks)*. *January*.

Erdös, P., & Gallai, T. (1960). Gráfok előírt fokú pontokkal. *Matematikai Lapok*, *11*, 264–274.

Erdös, P., & Rényi, A. (1959). On random graphs I. *Publicationes Mathematicae*.

Hakimi, S. L. (1962). 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*. https://doi.org/10.1137/0110037

Havel, V. (1955). A remark on the existence of finite graphs. *Časopis pro Pěstování Matematiky*. https://doi.org/10.21136/cpm.1955.108220

Mahadev, N. V. R., & Peled, U. N. (1995). Threshold Graphs and Related Topics. *Annals of Discrete Mathematics*. https://doi.org/10.1016/S0167-5060(13)71063-X

Riordan, O., & Spencer, J. (2001). *Random Graph Process*. 279–290.

Steger, A., & Wormald, N. C. (1999). Generating Random Regular Graphs Quickly. *Combinatorics Probability and Computing*. https://doi.org/10.1017/S0963548399003867