BOĞAZIÇI UNIVERSITY

GRAPH THEORY
IE 456

Project

Authors:
Çağan Şengün
Y. Harun Kıvrıl

25 January 2021

Department of Industrial Engineering
Boğaziçi University

Contents

1 Introduction

The everyday of a human life is full of networks. From the World Wide Web to connection of neurons in a human body, social interaction of humans to food chains lots of structures are examples of networks. Since networks are very common, understanding the networks is the key to understand many structures around us. Therefore modeling the networks become an important task. The history modelling of networks starts with two scientist called Erdos and Renyi. Erdos and Renyi suggested that a network can be modelled by creating a random graph where the vertices has randomly placed edges. [6]. For many years people treated all the networks as completely random, however after investigating some important networks such as World Wide Web, social interactions etc., it is noticed that these graphs are not completely random and the degrees of the vertices follows a distribution. In most of the networks it appeared that the network is dominated by a few number of vertices and the degree distribution of these graphs follow a power law ($1/k^n$). These networks are called scale free graphs. Therefore modeling scale free graphs becomes an important task while analyzing the networks around us.

This report briefly discusses generation of scale free graphs. In the report, firstly the random graphs are briefly explained and scale free graphs are introduced. Then the graphical sequences and related theorems are presented. After that random graph generation for scale free graphs is discussed. In that section Blitzstein and Diaconis's sequential algorithm is explained including its contribution to the literature. Finally, two other algorithms selected together with sequential algorithm, implemented and run time comparison experiments are made. At the end, comparison of the run times are presented with a conclusion and discussion at the end.

2 Random Graphs

Behind the extensive scientific work focused on graph generation lies the idea that complex networks in real life could be described using graphical analogies. For example, molecules play an important role in chemical reactions such that specific types of molecules take part in the same reaction in order to form a greater entity. If we consider the molecule types that happen to be in the same reaction relatives, then we can constitute relationships between different types of molecules depending on whether they participate in the same reaction or not. A graph consisting of nodes that represent molecule types and edges the relationship between two types would describe the network structure of molecules perfectly. Another example is that spread of diseases is a result of physical contact between people and social relationships should be examined in order to gain an insight into this subject. If we represent people by nodes in a graph and put an edge between two nodes if persons represented by these nodes are likely to get into physical contact with each other, then we get a representation of the social structure which becomes grounds for the spread of a disease.

When we represent networks in real life using graphs, a natural question arises that how the nodes in a graph should be linked to each other so that the graph most closely resembles the real network. The simplest approach in the literature to this question was provided by two Hungarian mathematicians. In 1959, aiming to describe networks seen in communications and the life sciences, Erdos and Renyi suggested that such systems could be effectively modeled by connecting their nodes with randomly placed links [1]. As can be understood from the definition, each node is equally likely to be linked to any other node, that is, the probability of two nodes being connected with an edge between is equal for any pair of nodes. One can intuitively see that degrees of nodes in a graph will be very close to each other if there is a single probability of any two nodes being neighbors. Therefore, nodes with extremely low or extremely high degrees are unlikely to be observed in random graphs. However, studies show that this is not the case in real networks [1]. Considering many networks in real life, distribution of degrees is not concentrated around an average value, that is, the degrees are not distributed normally among the nodes. Real networks are observed to be dominated by a small number of nodes having extremely high degrees and other nodes in the networks are connected to each other mostly via these nodes that are called hubs. In these networks, probability of observing a node with a given degree is inversely correlated with the value of the degree. If we randomly select a node in such a network, there is a low probability of selecting a hub and the node that we select most likely will have a very small degree. Such networks are called scale-free networks. A network being free of a scale can be interpreted in different ways. Barabasi and Bonabeau say that these networks are called scale-free since they include some nodes having such high degrees that seem to be out of scale [1]. However, Caldarelli states that some networks are scale-free in the sense that they will have the same structure no matter what their size is, which means that scale-free networks are similar to fractals. Any part of a network will have the same structure with the whole network. The existence of hubs in scale-free networks could be explained with a simple analogy. Suppose that you participate in a party with the aim of meeting as many people as possible. Since meeting the ones that have largest number of people around will give you the opportunity to fulfill your aim, you are most likely to contact them first. This analogy is a good one, because it will let us notice the two basic terms that constitute the formation of hubs, namely "growth" and "preferential attachment" [1]. The party will grow when you enter and your probability of meeting a specific person will be positively correlated with that person's popularity. As new people enter the party, your popularity will also increase proportional to time and the number of people you have already met since both of these factors add up to your probability of meeting someone and you may eventually become a hub yourself if you grow highly popular. One simple observation here would be that since the probability of a node in a graph forming a new connection is positively correlated with its number of already existing connections, nodes with higher degrees will form new connections faster compared to those with lower degrees and their rate of forming new connections will increase with each connection that they form. This phenomenon has been rediscovered several times under different names in the literature, main ones being "the rich get richer" and "the Matthew effect" [4]. "The rich get richer" phenomenon may help us notice the existence of preferential attachment in real networks. People with already high amounts of capital will have the opportunity of making more investments compared to those that have relatively lower amounts of capital and each profit that they make from these investments will become a new capital for their future investments. Thus, their wealth will increase with an increasing rate in time. Although "the rich get richer" phenomenon approaches the notion of preferential attachment from a business perspective, "the Matthew effect" explains it in a social context. The main argument of "the Matthew effect" is that those that have a large number of connections will attract new members of the society to come and meet them, and their chance of forming new connections will also increase with each new member of the society that they connect. This phenomenon is named by sociologist Robert K. Merton after a passage in the New Testament: "For to every one who has will more be given, and he will have abundance; but from him who has not, even what he has will be taken away." [1].

As we mentioned earlier, degrees of nodes in a random graph follow a normal distribution, that is, we will obtain a bell-shaped curve if we plot the degrees on the y-axis and the number of nodes with the same degree (corresponding to the degrees on the y-axis) on the x-axis. Most of the nodes will have similar degrees that gather around a mean value and although there will be some variation in the degree distribution between the nodes, we are not likely to observe nodes with extremely high or extremely low degrees. However, the situation is not the same in the case of scale-free graphs. In scale-free networks, the degree distribution follows a power law in that most nodes have just a few connections (low degrees) and some have a tremendous number of links (extremely high degrees) [1]. We will get a continuously decreasing function if we plot the distribution of degrees and an interesting observation here is that we will get a straight line if we plot the distribution of degrees on a double-logarithmic scale [1].

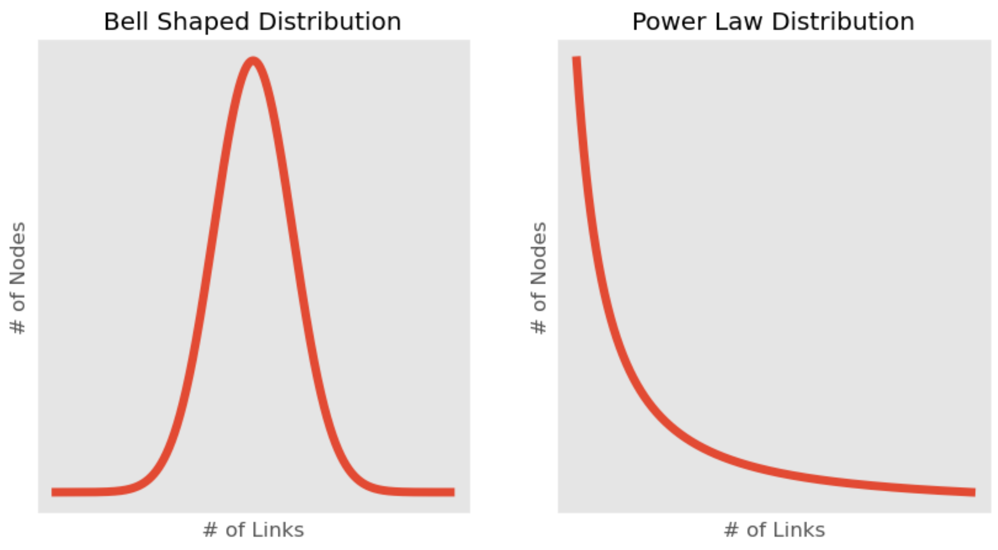


Figure 1: Plots of Bell Shaped Distribution and Power Law Distribution

Since we use graphs in order to describe the structure of real-life networks, an important research topic here is robustness of these networks against node failures. A network is called robust if it remains connected when we remove some of its nodes and the edges incident to these nodes. Random networks are vulnerable against random node failures since most of the nodes in a random network have similar degrees which concentrate around a peak value and each time a random node fails, this will most probably be a node that has quite many connections. However, scale-free networks are observed to be robust in case of random node failures [1]. This is intuitive in the sense that if we randomly select a node in a scale-free network and remove it (also remove the edges incident to this node), this will most likely be a node that has quite few connections and the network will remain connected after such random failures. Robustness of scale-free networks against accidental failures can be observed in real networks such that Internet is a scale-free network and even if as many as 80% of randomly selected Internet routers may fail, the remaining ones will remain connected [1]. Although scale-free networks are highly robust in case of random failures, they are vulnerable against coordinated attacks, that is, a scale-free network is likely to become disconnected if some of its key hubs are detected and removed [1]. This feature of scale-free networks may help prevent spread of diseases in the sense that if people with high numbers of social connections are detected and immunized in case of a fad, spread of the disease will be prevented since people that may catch the disease will have a limited number of social connections. It is easy to observe that this immunization process is equal to removal of hubs in a scale-free network, the resulting network will consist of disconnected components if the hubs are successfully detected and removed.

3 Graphical Sequences

Before introducing random graph generation, it is required to address some definitions and related theorems for better understanding. In the previous section the importance of the degree distributions of the random graphs, especially the scale-free graphs, are mentioned. The sequences created from that degree distributions are called degree sequences. In general terms a **degree sequence** is defined as any finite sequence of non-negative integers $(d_1, d_2, ..., d_n)$, $(n \geq 1)$ [3].

Given a degree sequence, it is not always possible to generate a graph using this sequence as degrees of vertices. For example a sequence d such that $\sum_{i=1}^{n} d_i = odd$ can not construct any graph since it violates the Handshaking Lemma [8]. If there exists such a graph with that degree sequence, that sequence is called **graphical** or **realizable**[3].

There are many different ways to test whether a degree sequence is realizable or not. The following theorem by Erdos-Gallai [7] is the most common criterion for testing whether a sequence is graphical or not.

Theorem (Erdos-Gallai): *Let* $d_1 \geq d_2 \geq ... \geq d_n$ *be non-negative integers with* $\sum_{i=1}^{n} d_i = even$*. Then* $d = (d_1, ..., d_n)$ *is graphical if and only if*

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

Choudum [5] proves this theorem by induction on the sum of degrees in the following way. The original form of the proof from Choudum's paper [5] can be found below.

Proof. By induction on s. The theorem holds when $s = 0$ or 2. Suppose that the theorem is true for sequences whose sum is $s - 2$ and let $\pi : d_1 \geq d_2 \geq \ldots \geq d_p$ be a sequence whose sum s is even and which satisfies (EG). There is no loss of generality in assuming $d_p \geq 1$. Let $t(\geq 1)$ be the smallest integer such that $d_t > d_{t+1}$; if π is regular then define t to be $p-1$. Consider the sequence $\pi^* : d_1 \geq \ldots \geq d_{t-1} > d_t - 1 \geq d_{t+1} \geq \ldots \geq d_{p-1} > d_p - 1$. We verify that π^* satisfies (EG). So, let k be an integer such that $1 \leq k \leq p$. We split the proof into five cases and prove in each case that π^* satisfies (EG); we use repeatedly the inequality: $\min(a,b) - 1 \leq \min(a-1,b)$.

(1) $k \geq t$.

$$\sum_{i=1}^{k} d_i - 1 \leq k(k-1) + \sum_{j=k+1}^{p} \min(d_j,k) - 1 \quad \text{[by (EG)]}$$

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

(2) $1 \leq k \leq t - 1$ and $d_k \leq k - 1$.

Clearly, $\sum_{i=1}^{k} d_i = k\, d_k \leq k(k-1) + \sum_{j=k+1}^{p} \min(d_j,k)$.

(3) $1 \leq k \leq t - 1$ and $d_k = k$.

We first observe that $d_{k+2} + \ldots + d_p \geq 2$. This is obvious if $k + 2 \leq p - 1$. If $k + 2 \geq p$, then $t = p - 1$ and so π is $(p-2)^{p-1}, d_p$. But then, $s = (p-2)(p-1) + d_p$ is even, and hence $d_p \geq 2$. So,

$$\sum_{i=1}^{k} d_i = k^2 - k + d_{k+1} \leq k^2 - k + d_{k+1} + d_{k+2} + \ldots + d_p - 2$$

$$\leq k(k-1) + \sum_{j=k+1,\neq t}^{p-1} \min(d_j,k) + \min(d_t - 1,k) + \min(d_p - 1,k).$$

(4) $1 \leq k \leq t - 1$, $d_k \geq k + 1$, and $d_p \geq k + 1$.

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{j=k+1}^{p} \min(d_j,k) \quad \text{[by (EG)]}$$

$$= k(k-1) + \sum_{j=k+1,\neq t}^{p-1} \min(d_j,k) + \min(d_t-1,k) + \min(d_p-1,k) .$$

(since, $\min(d_j,k) = \min(d_j-1,k) = k$) .

(5) $1 \le k \le t-1$, $d_k \ge k+1$ and $d_p < k+1$.

Let r be the smallest integer such that $d_{t+r+1} \le k$. If $\sum_{i=1}^{k} d_i = k(k-1) + \sum_{i=k+1}^{p} \min(d_j,k)$, then we arrive at a contradiction to (EG) as follows.

We first have,

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

So,

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

$$> (k+1)k + (t+r-k-1)(k+1) + \sum_{j=t+r+1}^{p} d_j , \text{ (since } \frac{1}{k} \sum_{j=t+r+1}^{p} d_j > 0)$$

$$= (k+1)k + \sum_{j=k+2}^{p} \min(d_j,k+1) .$$

Hence,

$$\sum_{i=1}^{k} d_i \le k(k-1) + \sum_{j=k+1}^{p} \min(d_j,k) - 1 \quad \text{[by (EG)]}$$

$$\le k(k-1) + \sum_{j=k+1,\neq t}^{p-1} \min(d_j,k) + \min(d_t-1,k) + \min(d_p-1,k) .$$

Thus in each case π^* satisfies (EG) and hence by the induction hypothesis it is graphic. Let G be a realization of π^* on the vertices $v_1,v_2,\ldots,v_p$. If $(v_t,v_p) \notin E(G)$, then $G + (v_t,v_p)$ is a realization of π . So, let $(v_t,v_p) \in E(G)$. Since $\deg_G(v_t) = d_t - 1 \le p-2$, there is a v_m such that $(v_m,v_t) \notin E(G)$. Since $\deg_G(v_m) \ge \deg_G(v_p)$, there is a v_n such that $(v_m,v_n) \in E(G)$ and $(v_n,v_p) \notin E(G)$. Deleting the edges (v_t,v_p) , (v_m,v_n) and adding the edges (v_t,v_m) , (v_n,v_p) we get a new realization G^* of π^* in which v_t and v_p are non-adjacent. Then $G^* + (v_t,v_p)$ is a realization of π .

The sum of all degrees in the sequence is indicated by s. Theorem holds when $s = 0$ or $s = 2$. We have $\pi : d_1 \geq d_2 \geq ... \geq d_p$ and $s = d_1 + d_2 + ... + d_p$, and we know that s is even. Without loss of generality, we can assume that $d_p \geq 1$. Let $t(\geq 1)$ be the smallest integer such that $d_t \geq d_{t+1}$. t is where we first observe a change (a reduction) in the degree sequence. Until and including t, all degrees are equal. We have $d_1 = d_2 = ... = d_t$. If π is regular, then we set $t = p-1$. In this case, $d_1 = d_2 = ... = d_p$. We define another degree sequence π^* by replacing the terms d_t and d_p in π with $d_t - 1$ and $d_p - 1$, respectively. We get $\pi^* : d_1 \geq ... \geq d_{t-1} > d_t - 1 \geq ... \geq d_{p-1} > d_p - 1$. In π, we have $d_{t-1} = d_t$ and in π^*, we have $d_{t-1} > d_t - 1$. In π, we have $d_{p-1} \geq d_p$ and in π^*, we have $d_{p-1} > d_p - 1$. We define an integer k such that $1 \geq k \geq p$. We will examine five different cases and prove in each case that π^* satisfies EG. In the meantime, we will repeatedly use the inequality $min(a,b) - 1 \leq min(a-1,b)$.

(1) $k \geq t$

t is at most equal to k, therefore we observe the t^{th} term in the degree sequence before the k^{th} term. d_t to $d_t - 1$ transformation takes place on the left-hand-side of the inequality $\sum_{i=1}^{k} d_i - 1 \leq k(k-1) + \sum_{j=k+1}^{p} min(d_j,k) - 1$, hence we subtract one from the left-hand-side. Since d_p to $d_p - 1$ transformation takes place on the right-hand-side of the inequality, we subtract one from the right-hand-side. The p^{th} term in the sequence is the last term in $\sum_{j=k+1}^{p} min(d_j,k)$. When we take it out of the summation, we get $min(d_p,k) - 1$. Since $min(a,b) - 1 \leq min(a-1,b)$, we can replace $min(d_p,k) - 1$ with $min(d_p - 1,k)$. Then we get $\sum_{i=1}^{k} d_i - 1 \leq k(k-1) + \sum_{j=k+1}^{p} min(d_j,k) - 1 \leq \sum_{j=k+1}^{p-1} min(d_j,k) + min(d_p - 1,k)$.

(2) $1 \leq k \leq t-1$ and $d_k \leq k-1$

The k^{th} term in the sequence comes before the t^{th} term. Therefore, $d_1 = d_2 = ... = d_k$. We know that d_k is at most equal to $k-1$. If we sum the degrees in the sequence until and including the k^{th} term, we get $k(d_k)$ since we have k terms and they all have the same value which is equal to d_k. $d_k \leq k-1$ implies that $k(d_k) \leq k(k-1)$. If we add $\sum_{j=k+1}^{p} min(d_j,k)$ to the right-hand-side, the inequality will still hold since $\sum_{k+1}^{p} min(d_j,k)$ is positive.

(3) $1 \leq k \leq t-1$ and $d_k = k$

We first observe that $d_{k+2} + ... + d_p \geq 2$. If there are at least two terms between the k^{th} term and the p^{th} term in the sequence, then the value reduction is observed in these two terms since one of them is the t^{th} term. Suppose there are two terms between the k^{th} term and the p^{th} term. Then $(k+1)^{st}$ term is the t^{th} term and $d_{k+1} > d_{k+2}$. We have two positive integers in $d_{k+2} + d_p$, therefore this sum is at least equal to 2. If there are less than two terms between the k^{th} term and the p^{th} term, then the term between those two is the t^{th} term. Except for the p^{th} term, all terms in the degree sequence have the same value d_k. Since there are less than two terms between the k^{th} term and the

p^{th} term $(k+2 \geq p)$, and $d_k = k$, we have $d_k = p-2$. In the sequence, we have $p-1$ terms with value $p-2$ and the p^{th} term. Their sum should be even, therefore we have $(p-2)(p-1)+d_p$ even. Since $(p-2)(p-1)$ is even and $d_p \geq 1$, we have $d_p \geq 2$.

Sum of the first k terms in the sequence is equal to $k(d_k)$. Since $d_k = k$, $k(d_k) = k^2$. Since $k \leq t-1$, $d_{k+1} = k$. Therefore, we have $\sum_{i=1}^{k} d_i = k^2 - k + d_{k+1}$. Since $d_{k+2} + ... + d_p \geq 2$, we have $k^2 - k + d_{k+1} \leq k^2 - k + d_{k+1} + d_{k+2} + ... + d_p - 2$. By EG, we have $\sum_{i=1}^{k} di \leq k(k-1) + \sum_{i=k+1}^{p} min(d_i,k)$. Since each of the terms $d_{k+1}, d_{k+2}, ..., d_p$ is at most equal to k, we have $d_{k+1} + d_{k+2} + ... + d_p - 2 \leq \sum_{j=k+1,\neq t}^{p-1} min(d_j,k) + min(d_t,k) - 1 + min(d_p,k) - 1$. We also have $min(d_t,k) - 1 \leq min(d_t - 1,k)$ and $min(d_p,k) - 1 \leq min(d_p - 1,k)$. Therefore, $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{j=k+1,\neq t}^{p-1} min(d_j,k) + min(d_t - 1,k) + min(d_p - 1,k)$.

(4) $1 \leq k \leq t-1$, $d_k \geq k+1$, and $d_p \geq k+1$

We have $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{j=k+1}^{p} min(d_j,k)$, and $\sum_{j=k+1}^{p} min(d_j,k) = \sum_{j=k+1,\neq t}^{p-1} min(d_j,k) + min(d_t,k) + min(d_p,k)$. Since $d_j \geq k+1$ for every j in the degree sequence, we have $min(d_j,k) = min(d_j - 1,k) = k$. Therefore, $\sum_{j=k+1}^{p} min(d_j,k) = \sum_{j=k+1,\neq t}^{p-1} min(d_j,k) + min(d_t - 1,k) + min(d_p - 1,k)$.

(5) $1 \leq k \leq t-1$, $d_k \geq k+1$, and $d_p < k+1$

We define r as the smallest integer such that $d_{t+r+1} \leq k$. We start observing reductions in the degree sequence after the t^{th} term. The $(t+r)^{th}$ term is the last term that is at least equal to $k+1$. Starting from the $(t+r+1)^{st}$ term, we observe values that are at most equal to k. Here, we try to prove that sum of the first k terms in the sequence should be at most $k(k-1) + \sum_{j=k+1}^{p} min(d_j,k) - 1$ by showing that $\sum_{i=1}^{k} d_i = k(k-1) + \sum_{i=k+1}^{p} min(d_j,k)$ leads to a contradiction.

We have $d_1 = d_2 = ... = d_k$, k terms with value d_k. We get $\sum_{i=1}^{k} d_i = k(d_k) = k(k-1) + (t+r-k)k + \sum_{j=t+r+1}^{p} d_j$. Between the k^{th} term and the $(t+r)^{th}$ term, all of the terms have value at least $k+1$, and we have $t+r-k$ terms. Therefore, $\sum_{j=k+1}^{t+r} min(d_j,k) = (t+r-k)k$. Also, $k(k-1) + (t+r-k)k = (t+r-1)k$. Therefore, we have $\sum_{i=1}^{k} d_i = k(t+r-1) + \sum_{j=t+r+1}^{p} d_j$. Since $k \leq t-1$, the $(k+1)^{st}$ term in the sequence has value d_k as well. $\sum_{i=1}^{k+1} d_i = (k+1)d_k$. This is equal to $\frac{k+1}{k} \sum_{i=1}^{k+1} d_i$. Multiplying $k(t+r-1) + \sum_{j=t+r+1}^{p} d_j$ by $\frac{k+1}{k}$, we get $(k+1)(t+r-1) + \frac{k+1}{k} \sum_{j=t+r+1}^{p} d_j$. Since $\frac{k+1}{k} > 1$ and $(k+1)(t+r-1) = (k+1)k + (t+r-k-1)(k+1)$, we have $(k+1)(t+r-1) + \frac{k+1}{k} \sum_{j=t+r+1}^{p} d_j > (k+1)k + (t+r-k-1)(k+1) + \sum_{j=t+r+1}^{p} d_j$. We have $(t+r-k-1)(k+1) + \sum_{j=t+r+1}^{p} d_j = (k+1)k + \sum_{j=k+2}^{p} min(d_j,k+1)$ since all of the terms between the $(k+1)^{st}$ term and the $(t+r)^{th}$ term have value at least k+1. However, we get a contradiction $\sum_{i=1}^{k+1} d_i > (k+1)k + \sum_{j=k+2}^{p} min(d_j,k+1)$. Since $\sum_{i=1}^{k} d_i = k(k-1) + \sum_{j=k+1}^{p} min(d_j,k)$ leads to a contradiction, $\sum_{i=1}^{k} d_i$ should be at most equal to $k(k-1) + \sum_{j=k+1}^{p} min(d_j,k) - 1$. We have $min(d_t,k) = min(d_t - 1,k) = k$ since $d_t \geq k+1$. We also have $min(d_p,k) - 1 \leq min(d_p - 1,k)$. Therefore, $\sum_{j=k+1}^{p} min(d_j,k) - 1 \leq \sum_{j=k+1,\neq t}^{p-1} min(d_j,k) + min(d_t - 1,k) + min(d_p - 1,k)$.

Checking the graphiciality using Erdos-Gallai theorem requires checking n inequalities and d must be sorted if it is not given in that form. Therefore it runs in O(n log n) time complexity. In order to not to check all of these inequalities, Havel [10] and Hakimi [9] independently come up with a recursive test which allows to terminate the test before checking all inequalities.

Theorem (Havel-Hakimi): *Let d be a degree sequence of length n ≥ 2 and let i be a coordinate with $d_i > 0$ If d does not have at least d_i positive entries other than i, then d is not graphical. Assume that there are at least d_i positive entries other than at i. Let be the degree sequence of length n-1 obtained form d by deleting coordinate i and subtracting 1 from the d_i coordinates in d of highest degree, aside from i itself. Then d is graphical if an only if is graphical. Moreover, if d is graphical, than it has a realization in which vertex i is joined to any choice of d_i highest degree vertices other than vertex i.*

The proof of Havel-Hakimi theorem is not included in this report however it can be found in the work of Blitzstain and Diaconis [3].

Applying the Havel-Hakimi theorem to a degree sequence allow us to determine whether the sequence is graphical or not. This recursive application is repeated until a sequence of zeros is found. Since, it is known that the sequence of zeros is graphical, in case of obtaining this sequence it is concluded that the given degree sequence is graphical. On the other hand if any $\tilde{d}$ is found to be not graphical during this process, it is concluded that the given degree sequence is not graphical. In addition Havel-Hakimi allows to construct a deterministic realization of a degree sequence if the sequence is graphical. By adding the edges from d_i to the vertices that 1 is subtracted a single realization can be obtained.

The process of testing a degree sequence using Havel-Hakimi theorem includes at most n iterations. It requires a initial sorting and the sorting can be maintained during the whole process. Therefore it gives a conclusion in *O(n log n)* time complexity.

4 Random Graph Generation

As we mentioned earlier, graphs can be considered models representing networks in real life. We try to model real networks using graphs in the sense that nodes in graphs correspond to members of the real network and we define a simple set of mathematical rules that would determine which nodes will be connected to each other so that these connections would best represent the relationships between the members of the real network. After we define these rules, we have to check if the resulting graph resembles the network we are trying to model. In order to achieve this aim, we can generate graphs by following previously defined mathematical rules and compare them with the real network. As for the comparison part, we can define a measurable quantity and check if the measurements in our model are consistent with the real data [4]. For example, distribution of degrees is a measurable quantity that would help check the validity of a model. Scale-free structure of some real networks (such as the Internet and the World Wide Web) was noticed when the links between members of these networks were observed to follow a power law rather than a normal distribution. Blitzstein and Diaconis state that for a generated graph to resemble a real network, it is useful to generate the graph with the degree sequence of the real network [3]. In this section, we will present main models in the literature that are used for graph generation for scale-free graphs and its more general form, generating graphs with given degree distributions.

Erdos-Renyi Model (Random Graph Model)

The simplest graph generation model is introduced by Erdos and Renyi, which is used for generation of random graphs. In this model, all possible edges in a graph are sampled and each possible edge is included in the graph with a fixed probability p [4]. There is a single probability indicating whether any pair of vertices in a graph will be linked to each other or not, that is, each vertex is equally likely to be linked to any other vertex.

Steps used in Erdos-Renyi model for graph generation are as follows:

- The number of vertices is given to the model.
- The probability of having an edge between any two vertices (p) is given to the model.
- The model samples all possible edges between the given number of vertices.
- Each edge in the sample is included in the graph with probability p.

The distribution of degrees in the resulting graph will follow a bell-shaped curve. Since the probability of having an edge between any two vertices is equal for every pair of vertices, most of the vertices will have similar degrees gathered around a mean value with some variation.

Barabasi-Albert Model

Barabasi-Albert Model is used for generation of scale-free graphs. This model introduces two terms that define the characteristics of scale-free graphs, namely “growth” and “preferential attachment” [4]. Number of vertices is not constant in this model, it increases as new vertices are introduced (growth) and the probability that a newly added vertex will be linked to each of the existing vertices is proportional to the current degrees of vertices in the graph (preferential attachment).

Steps used in Barabasi-Albert model for graph generation are as follows:

- An initial set of vertices with no edges is given to the model.
- New vertices will enter the model at any time step.

- The number of edges each newly added vertex has is given to the model.
- The model places these edges between the newly added vertex and the old vertices.
- The probability of an old vertex being linked to a newly added vertex is proportional to its current degree.

The distribution of degrees in the resulting graph will follow a power law. Most of the nodes will have very few connections and there will be a small number of nodes with exceptionally high number of connections.

Pairing Model

Pairing model is an algorithm to construct graphs with given graphical degree sequences. In case of having a degree distribution from a power law distribution it allows to construct a scale free graph. This algorithm is based on a simple logic of pairing as the name implies and it rediscovered many times by different people. The algorithm starts with generating half edges(cells) for each vertex equal to its predefined degree. Then the all of the half edges are randomly matched to obtain a random graph with initial degree sequence. This process may result in loops or multiple edges, therefore the process is repeated until a simple graph occurs. This problem of pairing process raises a question: What is the probability of having a simple graph at a trial? In order to answer this question Bender and Canfield [2] made a study that shows probability of having a simple graph as the number of graphs goes to infinity is reasonable for small d values. However for large ones too many trials in average is needed.

There also exists some modifications for pairing model. The simplest way to modify it is not letting the loops and multiple edges occur while matching the half edges one by one [3]. This method also can get stuck when there is unmatched half edges and it is not possible to match them without having a multi-graph. Also, there exists another approach to modify pairing model is made by McKay and Wormald [11] is based on making switches to avoid any loop or multiple edges.

Other algorithms

There is also a method of Tinhover [12] that uses random adjacency lists to construct graph with given degree distribution. There are also Markov chain Monte Carlo algorithms where the algorithms make switching by running a Markov Chain created based on Havel-Hakimi algorithm.

Sequential Algorithm

Knowing the drawbacks of other algorithms, Blitzsen and Diaconis has developed a new algorithm called Sequential Algorithm for graphs with given degree sequences [3]. This algorithm starts with an empty list of edges and updates the list in each iteration. Until the given degrees are all 0, it chooses least i with d_i a minimal positive entry. Then it computes a candidate list by picking all the possible edges that are not in edge list and when the edge is removed from d, d remains graphical. After that, it assigns a probability to the candidate edges proportional to the degree of the vertex other than i. In the next step, it selects an edge using the probabilities, adds the edge to the edge list and removes the edge from the degree sequence. It repeats the process of preparing a candidate list and selecting an edge until the d_i becomes 0. Then the algorithm picks a new vertex that replaces i.

Since this algorithm creates its candidate set by checking if the remaining edges constructs a graphical sequence, it does not need to run again and it always return a realization of the given graphical degree sequence. This allows to have a worst case bound in the running time. For an implementation using the Erdos-Gallai theorem to check graphicality the worst case time complexity of the algorithm is $O(n^2 \sum_{i=1}^{n} d_i)$ and the complexity gets worst for the d-regular graphs and it becomes $O(n^3 d)$ [3].

Contribution/Novelty of Sequential Algorithm

The other algorithms for generating random graphs with given degree sequence either create multiple edges or loops or get stuck. Therefore the algorithms needs to run over and over again until a simple graph is obtained. Also, the probability of having a simple graph in a trial approaches to 1 as the number of vertices and the scale of degrees grow. Therefore, for these algorithms there is no fixed bounded worst case running time. On the other hand the sequential algorithm never gets stuck due to its smart way of edge selection. As a result it has a worst case time complexity of $O(n^2 \sum_{i=1}^{n} d_i)$.

5 Experiments

In the previous sections of the report, different random graph generation algorithms are discussed. In this part, it is decided to pick three of the algorithms and compare their running times after implementing them. For this task, the deterministic graph realization from Havel-Hakimi theorem, the pairing model and the sequential algorithm is selected. These three algorithms implemented using Python programming language. After the implementation, the graph construction time of the algorithms for randomly generated graphical degree sequences is compared and the results are presented and discussed at the end.

5.1 Scale Free Degree Sequence Generation

In order to conduct experiments, some graphical degree sequences is needed. Since the main interest in on scale free graphs, a method for scale free degree sequence generation is needed. This is done by creating random numbers that follows power law distribution. The generation of these random numbers are made by first creating a uniform random variable, then putting this random variable to inverse of power law distribution's CDF. For a given number of nodes this process is repeated to obtain a degree sequence. The following block is the python code of the process.

```python
def generate_powerlaw_degreeseq(n_nodes, n=1.5):
  R = np.random.random(size=n_nodes) #Generate uniform random RVs
  X = (1/R)**(1/n) #Generate RVs from power law
  X = np.round(X) #Round the RVs to nearest integer
  return X
```

The algorithms requires graphical degree sequences for constructing graphs, therefore a method is generated to create many graphical degree sequences from the power law distribution. In this method, simply a degree sequence is created using the previous method and its graphicality is checked. This process is repeated until the number of graphical sequences reaches to the desired number of sequences.

```python
 def generate_graphical_powerlaw_seqs(n_seqs=30, n_nodes=100):
  seqs = []
  while len(seqs) < n_seqs: # While there is not enough seq
      seq = generate_powerlaw_degreeseq(n_nodes) #Generate sequence
      seq = np.sort(np.array(seq, dtype=int))[::-1] #Sort vertices
      if havel_hakimi(seq.copy()): #Check graphicality
          seqs.append((seq))

  return seqs
```

For the experiments the number of nodes are chosen as 10, 20, 50 and 100. For each number of nodes, 100 graphical sequences are created using the method above and 400 scale free graphical degree sequences are obtained.

5.2 Havel-Hakimi

The Havel-Hakimi theorem allows us to construct a graph while checking the graphicality of a degree sequence. This can be done by creating a edge from the selected vertex d_i to the vertices that 1 is subtracted.

In the implementation, a recursive function is written. This function returns True if the given degree sequences are all zero and it returns False if there is not enough positive vertices in the degree sequence. When these termination conditions not appear in the sequence, an edge between the vertex with the other is formed and the formed edges is subtracted from the degrees. Lastly the new sequence obtained from removing these edges enters the function again. According to which termination condition is hit eventually the graphicality is determined and on the way every subtracted edge constructs the graph. If the algorithm return False then the edges is set to 0. The Python implementation of the algorithm can be found in appendix.

5.3 Pairing Model

The pairing model for a given degree sequence is works by creating half edges at every vertex such that the number of half edges are equal to the degree of the vertex. Then the half edges are randomly matched to obtain a graph. This can result in loops and multiple edges, therefore after the random matching it is necessary to check whether the resulting graph is simple or not and repeat the process if it is not simple.

In the implementation for every vertex its vertex number is repeated n times and added to an array where n is the degree of the vertex. Every element in this array represents an half edges coming out of the vertices. Then this array is shuffled. After shuffling first half and the second half of the array is matched and full edges are created. Since in this matching the same edge can exists twice or the vertex can be matched with itself, it is required to check whether the full edges forms a simple graph. If the obtained graph is not simple the process is repeated.

In this algorithm probability of obtaining a simple graph decreases drastically as the number of vertices increase, therefore it is decided to set 1M iteration limit to make the experiments in a reasonable time. Also showing the number of vertices that reached 1M iteration limit allows to show the time is increasing because the iterations are more and if there are few graphs that reached 1M limit it may be possible to conclude that they might be outliers. Therefore in existence of a graph that reached 1M iteration limit means that the time is the lower bound. The Python implementation of the algorithm can be found in appendix.

5.4 Sequential Algorithm

The sequential algorithm works by adding an edge in each iteration form a candidate list with a probability proportional to its degree at that stage of the algorithm. The candidates are created by checking the graphicality of the rest of graph in absence of this edge therefore sequential algorithm gets never stuck and obtain a realization of the given degree sequence. In the implementation a recursive function is written. This function returns the constructed graph if all elements in the degree sequence is 0. Otherwise it selects the minimal positive entry vertex and until its remaining degree becomes zero, adds an edges by creating candidate list from the non-existing edges by checking graphicality using havel hakimi algorithm that implemented before. After creating the candidate list it selects an edge from that list with a probability proportional to its degree. Then the selected edge is removed from the degree sequence. After selecting all edges that comes out of the selected vertex the function returns a call of itself with the new degree sequence + the edges that constructed at that step. When the termination condition is reached the construction a realization of the graph is obtained. The Python implementation of the algorithm can be found in appendix.

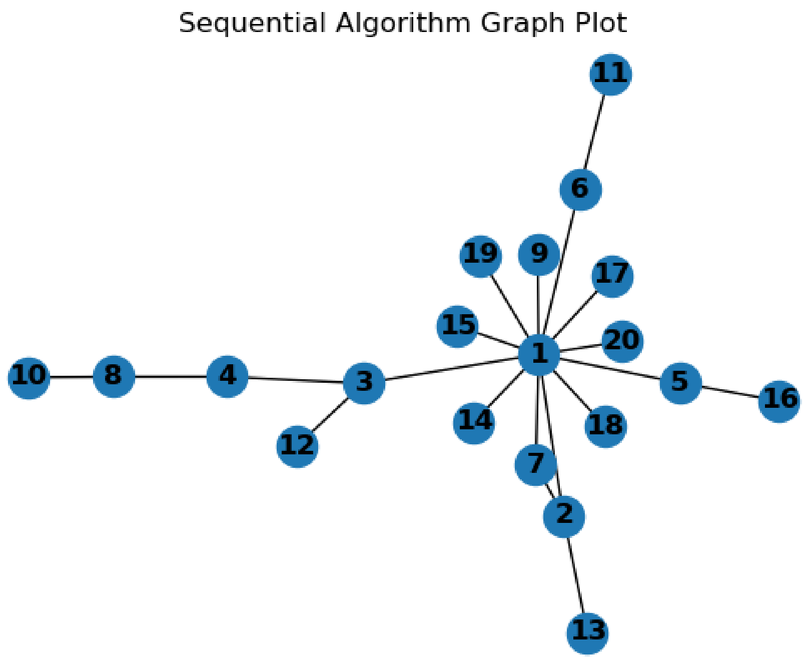
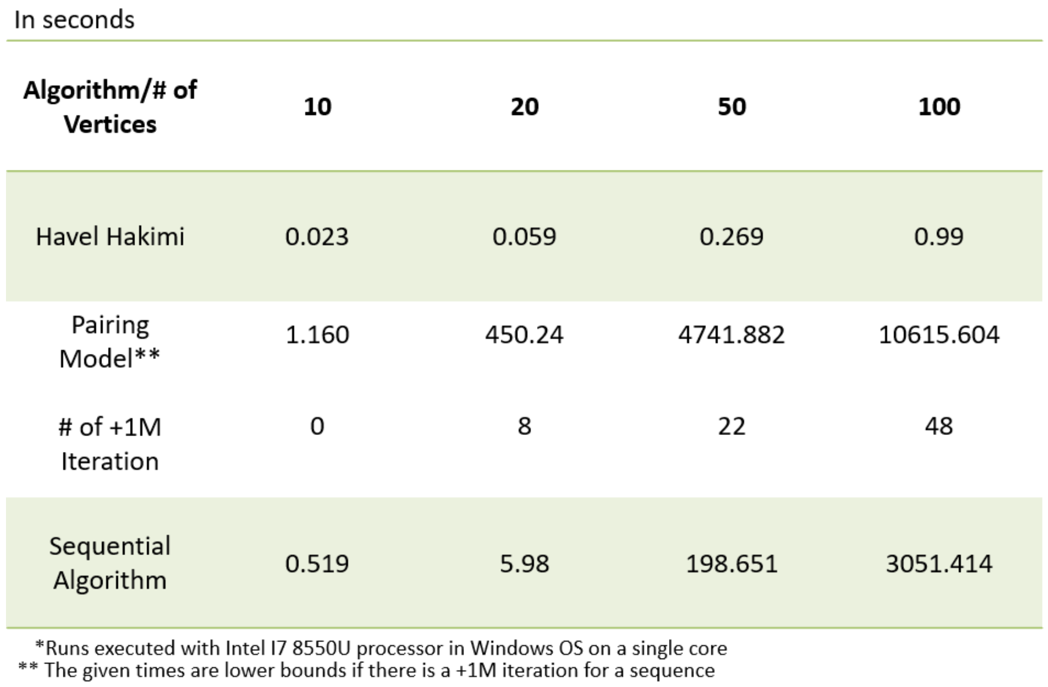


Figure 2: Example of a scale free realization obtained from sequential algorithm with 20 vertices.

5.5 Results and Discussion

The all generated scale free graphical degree sequences are fed into algorithms and total running time for 100 instances are recorded. The function written for comparison can be found in the appendix.

The runs are taken with Intel I7 8550U processor in Windows OS on a single core. After running the algorithms for total 100 sequences for each number of vertices the following table is obtained.



In seconds

Algorithm/# of Vertices	10	20	50	100
Havel Hakimi	0.023	0.059	0.269	0.99
Pairing Model**	1.160	450.24	4741.882	10615.604
# of +1M Iteration	0	8	22	48
Sequential Algorithm	0.519	5.98	198.651	3051.414

*Runs executed with Intel I7 8550U processor in Windows OS on a single core
** The given times are lower bounds if there is a +1M iteration for a sequence

Figure 3: Results obtained from experiments. (Total time of constructing 100 realizations

The results show that havel hakimi algorithm gives the results almost instantly. However. it gives a deterministic realization therefore it is not possible to construct two different graphs with the same degree sequence. For some modelling purposes having different realization of the same sequence may be important. Except from these cases, this algorithm seems the best choice in terms of time.

Pairing model performs poor when the running times are compared. Even though the iteration number is limited it took enormous time to run it. Also, the results shows an increasing number of graphs that hit the iteration limit therefore the main reason of having poor performance is getting stuck. The algorithm makes 1M million matching for these sequences and none of them comes out as simple. The sequential algorithm gives more reasonable times when it is compared to pairing model. Making an iteration in sequential algorithm requires more operations however it never gets stuck. Therefore, it is possible to say that not getting stuck in exchange of more computational cost totally worth when it compared to pairing model. However, creating 100 scale free graphs with 100 vertices still takes about 50 minutes and it may become a problem when more graphs with more vertices are tried to construct.

6 Conclusion

Generating random graphs is an important task since everyday of a human life is full of graphs and investigating, modeling these graphs around us is understanding the things around us and solve problems. Even though the first random graph generation algorithms did not covered this, all graphs are not all the same. They have different properties according to their degree distributions therefore random graphs should be generated. Scale free graphs are one of them and it is the one of the most observed graph. From world wide web to social networks, many graphs in human life follows a power law distribution which makes them scale free. Therefore, generating scale free graphs becomes an important topic.

One of the ways of generating scale free graphs are using the algorithms that produces realizations of given degree sequences. When the degree sequence is given in a scale free form the realization becomes scalee free. In the report three different algorithms for generating random graphs with given degree distribution is investigated: havel-hakimi theorem, pairing model and sequential algorithm. After investigation these algorithms are implemented and compared. As a result of comparison Havel-Hakimi gave the best time performance, however it may not be suitable for some modelling purposes since it creates a deterministic realization. On the other hand the other algorithms give non-deterministic realization. It is observed that pairing model gets stuck easily and its gets more and more stuck as the number of vertices increases. The sequential algorithm however, never gets stuck and performs much better than pairing model especially in the graphs with many vertices. Therefore, not getting stuck is an important property of sequential algorithm and it makes the sequential algorithm preferable when non-deterministic realizations are required.

7 References

[1] Albert-László Barabási and Eric Bonabeau. Scale-free networks. *Scientific american*, 288(5):60–69, 2003.

[2] Edward A Bender and E Rodney Canfield. The asymptotic number of labeled graphs with given degree sequences. *Journal of Combinatorial Theory, Series A*, 24(3):296–307, 1978.

[3] Joseph Blitzstein and Persi Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet mathematics*, 6(4):489–522, 2011.

[4] Guido Caldarelli. *Scale-free networks: complex webs in nature and technology*. Oxford University Press, 2007.

[5] S.A. Choudum. A simple proof of the erdos-gallai theorem on graph sequences. *Bulletin of the Australian Mathematical Society*, 33(1):67–70, 1986.

[6] Paul Erdös and Alfréd Rényi. On random graphs publ. *Math. debrecen*, 6:290–297, 1959.

[7] P. Erdős and T. Gallai. Gráfok előírt fokszámú pontokkal. *Matematikai Lapok*, 11:264–274.

[8] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

[9] S Louis 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*, 10(3):496–506, 1962.

[10] Václav Havel. A remark on the existence of finite graphs. *Casopis Pest. Mat.*, 80:477–480, 1955.

[11] Brendan D McKay and Nicholas C Wormald. Uniform generation of random regular graphs of moderate degree. *Journal of Algorithms*, 11(1):52–67, 1990.

[12] Gottfried Tinhofer. On the generation of random graphs with given properties and known distribution. *Appl. Comput. Sci., Ber. Prakt. Inf*, 13:265–297, 1979.

8 Appendix

Havel-Hakimi Python Code:

```python
def add_edges(di, vertex_id, vertex_ids):
    global edges
    for i in range(di):
        edges.append((vertex_id, vertex_ids[i]))

def havel_hakimi(d:list, vertex_ids:list=None, construct_graph=False) -> bool:
    """
    d: sorted descendingly and degree >= 0:list
    vertex_ids: Id of the vertices can be specified before or the function will assing
        ids
    Requires a return_graph:bool and empty edges:list if return_grah is True.
    """
    if vertex_ids is None: #Set vertex ids to construct the graph
        vertex_ids = np.array(range(1,len(d)+1))

    if sum(d) == 0: #if d all zeros
        return True

    di = d[0] #Since d always sorted i=0
    d = d[1:] #Remove di from d
    vertex_id = vertex_ids[0] #Get vertex id
    vertex_ids = vertex_ids[1:] #Remove ith vertex from vertex_ids

    if sum(d > 0) < di: #If there is not enough positive vertices
        if construct_graph:
            global edges
            edges = None #Return None for edges if graph is not realizable
        return False

    d[:di] += -1 #Remove edges between selected and others

    if construct_graph:
        add_edges(di,vertex_id, vertex_ids)

    idx = np.argsort(d)[::-1] #sorted indices of d
    d = d[idx] #Sort d using indices
    vertex_ids = vertex_ids[idx] #Also sort vertex_ids to match new d

    return havel_hakimi(d, vertex_ids, construct_graph)
```

Pairing Model Python Code:

```python

def check_simplicity(edges):
    edges = np.array(edges)
    # Make smaller vertice id appear first to compare easily
    edges = np.sort(edges, axis=1)

    # Check if is there any loops
    for edge1 in edges:
        if edge1[0] == edge1[1]:
            return False

    # Check if any multiple edges
    # Looping in python is slow. I prefer numpy functions works on lower level.
    if len(edges) > len(np.unique(edges, axis=0)):
        return False

    return True

def pairing_model(d, limit_iterations=True):
    '''
    d: graphical degree sequence
    '''
    # Raise error if accidentaly improper seq is passed.
    if sum(d) % 2 == 1:
        raise ValueError('Sum of degrees are odd.')
    if max(d) >= len(d):
        raise ValueError('There is a degree with di > n_vertices-1')

    is_simple = False

    cells = []
    for i in range(1, len(d)+1): # for every degree of every vertex add a one sided edge
         to cells
        # 3*[1] = [1,1,1] in python and [1,1] + [1] = [1,1,1]
        cells = cells + d[i-1]*[i]

    n_edges = int(sum(d)/2)

    trials = 0
    while not is_simple: # Until a simple graph is obtained
        if trials % 10000 == 0:
            print("", end=f"\rTrial: {trials}")
        #print('Trial: {}'.format(trials), end='\r')
        shuffle(cells) # Shuffle cells
        # match first half and the second to obtain edges
        edges = list(zip(cells[:n_edges], cells[n_edges:]))
        # check if edges construct a simple graph
        is_simple = check_simplicity(edges)
        trials += 1

        if trials > 1e6+1 and limit_iterations: # 1M iteration limit
            print('\nWARNING: ITERATION LIMIT EXCEDEED')
            return None
    print('\n')
    return edges
```

Pairing Model Python Code:

```
def sequential_algorithm(d):
    """
    d: Graphical degree seq.
    """
    edges = []
    # If all zero we are done
    if sum(d) == 0:
        return edges

    #Select i with minimal positive entry by replacing negatives with infinity
    idx = np.where(d > 0, d, np.inf).argmin()

    # While degree of i is not zero
    while d[idx] > 0:
        prob_weights=[]
        J=[]
        # Create candidate list
        for j in range(len(d)):
            #j not equal to i and no negative degrees allowed and the candidate not in
                edges
            if j != idx and d[j] > 0 and ((idx,j) not in edges or (j,idx) not in edges):
                d_temp = d.copy()
                d_temp[j] = d_temp[j]-1 #remove 1 from j
                d_temp[idx] = d_temp[idx]-1 #remove 1 from i
                d_temp[::-1].sort() # sort d_temp for havel-hakimi
                if havel_hakimi(d_temp): #Is candidate graphical?
                    J.append((idx+1,j+1)) #Add candidate to candidate list
                    prob_weights.append(d[j]) #Add its degree to prob weights

        probs = np.array(prob_weights)/sum(prob_weights) #Convert weights to probs
        selected_idx = np.random.choice(list(range(len(J))), size=1, p=probs)[0] #Select
             an edge with given probs
        selected_edge = J[selected_idx]
        edges.append(selected_edge) #Add selected edge to edges
        d[idx] = d[idx]-1 #Remove 1 from i
        j = selected_edge[1] -1 #Since
        d[j] = d[j] - 1 #Remove 1 from j

    return sequential_algorithm(d) + edges #Return step 2
```

Comparison Function:

```
edges = []
def compare_times(seqs):
    global edges
    results={}

    alg_start = time()
    for i , seq in enumerate(seqs):
        res = havel_hakimi(seq.copy())
        print(i, end=', ')
    results['Havel-Hakimi'] = {"total_time": time() - alg_start}

    failed = 0
    alg_start = time()
    for i, seq in enumerate(seqs):
        print(i, end=', ')
        res = pairing_model(seq.copy())
        if res is None:
            failed += 1
    results['Pairing Algorithm'] = {"total_time": time() - alg_start, "failed":failed}
```

```python

    alg_start = time()
    for i, seq in enumerate(seqs):
        print(i, end=', ')
        res = sequential_algorithm(seq.copy())
    results['Sequential Algorithm'] = {"total_time": time() - alg_start}

    return results
```