IE456-PROJECT

REPORT

Eda BİTARAF – 2019402024

Yusuf Utku GÜLDÜR – 2019402198

Table of Content

1.Introduction

Random graphs have gained currency thanks to their ability to model complex real-life problems such as the structure of the internet, social interactions, biological networks, transportation problems, etc. The need for random graphs arose in these problems that have high unpredictability and lack of information in their nature, because modeling such systems with non-random graphs is not a good model choice and it is almost impossible. Although non-random graphs are good at representing networks with specific structures; as the networks get more complex, and some unknown components are included, non-random graph representations fall short in modeling the complex structures since we do not know every possible relationship in the network. Hence generating random graphs that have similar features to real-life problems and analyzing them to understand the behavior of the real system is more useful in complex problems.

The emergence of such random graph algorithms started with Erdős–Rényi. They constructed the random graphs in such a way that for the given graph G with the number of vertices equal to N, the probability of having an edge between two vertices is p for all pairs of vertices [1]. Hence, each vertex's degree follows the binomial distribution [2]. However, this model excludes many graph models that don't have the same degree distribution among their vertices or the probability of having an edge between two vertices is not the same. An example of such a graph model is power law graphs where some nodes can have many connections (higher degree) while others have very few connections (lower degree).

The graphs that have the mentioned power law degree distribution are called scale-free graphs. Suppose graph G has N vertices where each vertex has degree of k and the probability of having degree k is as follows [3],

$$P(k) = k^{-\gamma}$$

These graphs arise in complex structures like web and biological networks and since classical Erdős–Rényi cannot represent random graphs that have power law distribution many other models are constructed. Examples of such models that uses scale-free graph notion are Barabási –Albert, Chung-Lu, R-MAT, exponential random graph, HOT, etc [3]. According to their approaches to build a scale-free graph these models can be divided into two categories that are growth models and configuration models [4].

In this report, the aim is to generate random graphs with given degree sequences by using Havel Hakimi and Sequential algorithm and compare the algorithms by conducting an experiment that gives the time to generate the graphs. In section two, the algorithms for degree sequences are given and the sufficient condition for generating graphical degree sequences is proved by induction. In section three, random graphs are generated with the prescribed degree sequences that were generated in section two. In section four, the graph connectivity is examined by using the depth-first search algorithm, and if the graph is disconnected pairwise interchanges are used to make the graph a connected one. Lastly, experiments that aim to give an idea about the time complexity of generated graphs are done and a comparison of algorithms is made.

2.Generating degree sequences

While generating degree sequences, we had two main constraints. First, sequences were needed to be able to generate connected graphs since our aim is to have connected graphs after the pairwise interchanges which will be analyzed in the later of the report. To be able to have connected graphs the degrees must hold minimally connected condition which is $m \geq n - 1$ where m is the number of edges and n is the number of vertices. If degree sequence does not hold this condition, it is impossible to generate connected graphs after the pairwise interchanges hence it is added to our degree sequence generator as a constraint. Second and the most important constraint was being graphical. Two basic conditions for graphicality are not having a vertex having a degree more than the total number of vertices and sum of degrees being even. However, these conditions are not sufficient to ensure that degree sequence is graphical hence following Erdős–Gallai theorem is used in degree generation since it is a sufficient condition for graphicality.

Theorem 1 (Erdős–Gallai). Let d1 ≥ d2 ≥ · · · ≥ dn be nonnegative integers with $\sum_{i=1}^{n} di$ even. Then d = (d1, . . . , dn) is graphical if and only if;

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

If the degree sequence holds this condition or not is checked with a function called is graphical in the implementation and sequences that are graphical are used in further study.

The given theorem can be proven by using the proof by induction technique. The theorem holds for degree sequences where their sum is equal to 0 and 2. Since the first two even possible sums

holds for the theorem proof by induction can work in this case. Hence, let's assume that a degree sequence whose sum is even and equal to *s* satisfies the Erdős–Gallai condition. One can deduce that if the assumption is correct then Erdős–Gallai theorem is proven. The proof starts with choosing a k value in which $d_k \geq k$. And the theorem checks if sum of the first k degrees is not higher than the maximum possible edges between first k vertices plus the maximum number of possible edges that can be constructed between these k vertices and the remaining n-k vertices. This idea is repeated in [5] by trying the idea with the equation for different cases when $d_k = k$, $d_k \leq k - 1$, etc. Since each possibility is tried and all of them fulfills the requirements for the Erdős–Gallai therom, the sufficiency of the theorem is proven.

Another proof was made by Berge in his book called "Graphs and Hypergraphs [6]" to prove the Erdős–Gallai theorem. Berge proved the same idea by using the network flows and giving 3 equivalent conditions which are as follows;

(1) There exists a simple graph G whose vertices x_i satisfy $d_G(x_i) = d_i$

(2) $\sum_{i=1}^{k} di \leq \sum_{i=1}^{k} di^{\wedge}$ where $di^{\wedge}$ denote its corrected conjugate sequence

(3) $\sum_{i=1}^{k} di \leq k(k-1) + \sum_{i=k+1}^{n} min(k, di)$ for each k ∈ {1, . . . , n}.

He shows why (1) implies (2), (2) implies (1), (2) implies (3) and (3) implies (2). Since Berge' proof includes many other theorems and intensive graph knowledge the detailed explanation won't be included in the report.

After getting the degree sequence, we realized that the degree sequence was not that much random. When many degree sequences were generated, there was a resemblance among them because all their degrees were distributed according to the same degree distribution which is uniform. The algorithm generated degree sequences for uniform random graph model since the same probability is given to each possible number from 1 to n-1. Hence the generated degree sequence was only for a subset of graphs where degree distribution follows a uniform distribution. Since we need to consider other graph types as well in our experiment, we implemented three-degree sequence generation function one of which is power-law degree sequence generator since scale-free graphs that are used in modeling many complex problems has degree's that follows power-law distribution. Secondly, we add binomial degree sequence generation function to generate the degree sequence of previously mentioned Erdős–Rényi graphs. Also, degree sequences where its degrees follow a normal distribution is constructed. to examine another subset of the graphs and enhance the content of the report.

Lastly, one more condition is checked in the degree sequence generation part to fasten the process by reducing the number of inequalities to check in the Gallai theorem. A theorem in [8] suggests that it is sufficient to check only the first m inequalities in Gallai theorem and this m is called the corrected Durfee number of the degree sequence [7].

$$m = |\{j: dj \geq j - 1\}|$$

3. Implementation of algorithms and random graph generation

3.1 Havel Hakimi Algorithm

To implement this algorithm Havel Hakimi theorem which finds whether a given sequence is graphical or not is used. The theorem says that for a given degree sequence *d* of length n, let *d'* be the degree sequence of length n-1 which is generated by choosing first the vertex with highest degree d_i and delete it, and subtracting 1 from d_i many of vertices that has the highest degree. Then theorem says that *d* is graphical only if *d'* is graphical. By using this theorem four different algorithms are generated.

3.1.1 Havel Hakimi-Highest Degree Vertex First, distributed to the Highest Degree

The theorem uses a recursive method in which it reduces the graph into smaller one by deleting a vertex and all the corresponding edges that vertex have in each step until all degrees are zero (assume after deleting the vertex its degree is zero), if not sequence is not graphical. Since we work on graphical sequences, we won't consider the later case. To construct the graph the Havel Hakimi approach will be used but rather than deleting and subtracting edges and vertices, we are going to add them on our graphs one by one recursively. First, we have an empty graph and the graphical sequence that we generated in section 2. Then, the highest degree vertex that has a degree of d_i will be placed on the graph and d_i many of edges will be constructed. Since each edge contains two end points d_i many of vertices are generated as well. Since we add vertices to the graph and each of them have 1 edge, their degrees will be decreased by one. The algorithm will keep working accordingly until all vertices and edges are placed on the graph.

3.1.2 Havel Hakimi-Random Vertex Selection, distributed to the Highest Degree

The only thing changed in this algorithm is the selection of the vertex to put on the graph. Compared to the first algorithm, it chooses the vertex randomly without considering the vertex degrees. Other than that implementation is the same. This algorithm is different from the first one since it puts some randomness to the algorithm. Normally, Havel Hakimi algorithm is deterministic hence for a given degree sequence it generates the same graph for every instance. Since our aim is to generate random graphs random vertex selection is preferable.

3.1.3 Havel Hakimi-Smallest Degree Vertex First, distributed to the Highest Degree

In this algorithm construction of the graph starts from the smallest degree vertex and it builds and edge starting from the highest degree vertex. This algorithm is also a deterministic one.

3.1.4 Havel Hakimi-Smallest Degree Vertex First, distributed to the Smallest Degree

The algorithm starts with choosing the vertex with smallest degree and it selects among remaining vertices according to their degrees and it selects the vertex with smallest degree. However, this causes problems during the implementation of the graph. For example, if graph has a vertex whose degree is equal to n-1 and a vertex with degree 1, algorithm stuck and cannot construct a graph although the sequence is graphical. So, we can say that especially in sparse graphs the algorithm cannot work. Another subset of graphs the algorithm cannot work most of the time are scale free graphs since in their degree sequences there are vertices with very high degrees and vertices with low degrees. If we start from smallest and distributed to the smallest the vertex with high degree won't be able to construct its edges. However, there are also some examples of graphs where the method works like completed graphs.

3.2 Sequential Algorithm

We started the implementation of this algorithm with a generated graphical sequence and an empty list of edges. First the minimum degree entry is chosen and to decide with which vertex it will construct an edge first a candidate list of edges is constructed such a way that neither of the edges in the edge set violates the graphical conditions of the remaining graph. From the list an edge is chosen with a probability proportional to its degree. After adding the selected edge degree sequence is updated and steps are repeated until all degrees reduce to zero. Note that after selecting a vertex its degree in the sequence doesn't reduce to zero but it decreased by 1 since in each step, we just add one edge not the all-possible edges of the vertex like in the Havel-Hakimi algorithm.

Also, to increase the time efficiency of the sequential algorithm some improvements are made. In the basic algorithm for each choice of a vertex the graph is searched to construct a candidate list. However, for a vertex the candidate list in a later stage is also a candidate list in an earlier stage hence contrary to the basic algorithm, we use the previous candidate list for the same choice of a vertex [7]. Another improvement was made when the minimum degree of residuals is equal to the length of the candidate list. In sequential algorithm, normally after choosing the minimum degree, the candidate list is searched to find the proper candidate. However, if the minimum degree is equal to the length of the candidate list, it is certain that it will choose everything from the candidate list after some steps. By writing a condition for this specific case in the code for the sequential algorithm we eliminate these steps and pair the minimum degree vertex with the whole candidate list when their degree is equal to the length of the candidate list.

The question arises after implementing different algorithms for the same purpose: why do we need to have another algorithm like sequential when we already have Havel-Hakimi algorithms to generate graphs with given degree sequences? This has two answers, one is that the sequential algorithm never gets stuck [7], and secondly this algorithm is more flexible because compared to other algorithms sequential doesn't choose the vertices to distribute the edges with a deterministic approach (like distributing to the highest).

4. Graph Connectivity

After generating the graphs, their connectivity was examined by applying depth first search. For the graphs that resulted as disconnected after the search, an algorithm to convert the disconnected graphs to connected ones is generated. One way to implement this is to implement pairwise edge interchanges by randomly choosing two edges and change the edge connections between them and repeat this until the graph becomes connected. However, choosing random 2 edge from the graph is not efficient hence another algorithm is thought in which we use connected subgraphs of a graph. By choosing two edges from two different connected subgraphs and changing the edges between them the graph can be converted into a connected one. However, we used a slightly different approach which we think gives faster results.

Our algorithm is composed of connected subgraphs. By using the idea of cycles and bridges, the algorithm can convert disconnected graphs into connected ones efficiently. In the algorithm that will be explained explicitly, edges are chosen from the cycles of a subgraph if there are any since the edges of a cycle cannot be a bridge. Because if one chose an edge from a bridge, the connected subgraph would turn into a disconnected one. By taking an edge from a subgraph that has cycle and an edge from a subgraph that has no cycles and swapping them, interchanges will be conducted. At the end of the algorithm when there is no acyclic graph left, an edge from each subgraph that forms cycle will be chosen and with respect to their index number they will swap their edges and form a cycle which result in a connected graph. In last step since we choose an edge from each subgraph, there might be a problem regarding changing degree sequence however this problem is solved by respecting the index which explained in step 5. The step for the algorithm is as follows:

1- Find the subgraphs of the given graph, by using the implemented depth first search algorithm.
2- Label them cyclic if there is a cycle in the subgraphs, otherwise label them as acyclic. Also hold the number of cycles
3- Choose an edge from an acyclic subgraph and interchange this edge with an edge from the cyclic subgraph that has the maximum number of cycles.
4- After updating the graph go to step 1, if there is no acyclic component go to step 5.
5- Choose an edge that belongs to a cycle from each subgraph. Form a cycle among them by interchanging selected edges. To avoid changing degree sequences or multiple edges,

after choosing and removing the edges(i,j) i's and j's will be put on different lists with their index number. And lastly, first i will be connected with the (i-1) index of the j list.

Since in the last step of our algorithm the process is not exactly same with the normal pairwise interchange, in the outputs we didn't record the number of swaps rather we recorded total number of changes happened during the pairwise interchange.

5.Experiement and Comparison of models.

After implementation of the algorithms, we generated many graphs with different number of vertices and edges by using different algorithms. As we mentioned in the beginning of the report, we generated degree sequences with respect to 3 probability distributions. We collected the data of the graphs that have binomial, power-law, and normal degree distribution. Small parts from datasets are given in Figure1, 2, and 3 for readers to understand better the data we are working on.

	vertecies	degrees	edge	density	connection	pairwise	time(ms)	algo
0	10	[8, 7, 7, 6, 6, 4, 4, 3, 2, 1]	24	0.53	1	0	0	HH_high_to_high
1	10	[8, 7, 7, 6, 6, 4, 4, 3, 2, 1]	24	0.53	1	0	0	HH_small_to_high
2	10	[8, 7, 7, 6, 6, 4, 4, 3, 2, 1]	24	0.53	1	0	0	HH_random_to_high
3	10	[8, 7, 7, 6, 6, 4, 4, 3, 2, 1]	24	0.53	1	0	1	sequential
4	10	[8, 7, 7, 6, 6, 5, 4, 4, 3, 2]	26	0.57	1	0	0	HH_high_to_high
5	10	[8, 7, 7, 6, 6, 5, 4, 4, 3, 2]	26	0.57	1	0	0	HH_small_to_high
6	10	[8, 7, 7, 6, 6, 5, 4, 4, 3, 2]	26	0.57	1	0	0	HH_random_to_high
7	10	[8, 7, 7, 6, 6, 5, 4, 4, 3, 2]	26	0.57	1	0	0	sequential
8	10	[8, 8, 8, 8, 8, 7, 6, 6, 6, 5]	35	0.77	1	0	0	HH_high_to_high
9	10	[8, 8, 8, 8, 8, 7, 6, 6, 6, 5]	35	0.77	1	0	0	HH_small_to_high
10	10	[8, 8, 8, 8, 8, 7, 6, 6, 6, 5]	35	0.77	1	0	0	HH_random_to_high
11	10	[8, 8, 8, 8, 8, 7, 6, 6, 6, 5]	35	0.77	1	0	1	sequential

Figure 1. First 12 rows of the dataset for the graphs with normal distribution

	vertecies	degrees	edge	density	connection	pairwise	time(ms)	algo
200	60	[56, 54, 53, 53, 53, 53, 52, 52, 52, 51, 48, 4...	973	0.54	1	0	2	HH_high_to_high
201	60	[56, 54, 53, 53, 53, 53, 52, 52, 52, 51, 48, 4...	973	0.54	1	0	1	HH_small_to_high
202	60	[56, 54, 53, 53, 53, 53, 52, 52, 52, 51, 48, 4...	973	0.54	1	0	0	HH_random_to_high
203	60	[56, 54, 53, 53, 53, 53, 52, 52, 52, 51, 48, 4...	973	0.54	1	0	1530	sequential
204	60	[57, 56, 53, 53, 52, 52, 51, 51, 50, 50, 49, 4...	915	0.51	1	0	1	HH_high_to_high
205	60	[57, 56, 53, 53, 52, 52, 51, 51, 50, 50, 49, 4...	915	0.51	1	0	1	HH_small_to_high
206	60	[57, 56, 53, 53, 52, 52, 51, 51, 50, 50, 49, 4...	915	0.51	1	0	1	HH_random_to_high
207	60	[57, 56, 53, 53, 52, 52, 51, 51, 50, 50, 49, 4...	915	0.51	1	0	1032	sequential
208	60	[58, 55, 54, 54, 53, 52, 52, 50, 50, 49, 49, 4...	984	0.55	1	0	1	HH_high_to_high
209	60	[58, 55, 54, 54, 53, 52, 52, 50, 50, 49, 49, 4...	984	0.55	1	0	0	HH_small_to_high
210	60	[58, 55, 54, 54, 53, 52, 52, 50, 50, 49, 49, 4...	984	0.55	1	0	1	HH_random_to_high
211	60	[58, 55, 54, 54, 53, 52, 52, 50, 50, 49, 49, 4...	984	0.55	1	0	1494	sequential

Figure 2. Rows of the dataset for the graphs with normal distribution with n = 60.

	vertecies	degrees	edge	density	connection	pairwise	time(ms)	algo
388	100	[96, 95, 94, 94, 94, 91, 90, 88, 88, 87, 86, 8...	2476	0.50	1	0	3	HH_high_to_high
389	100	[96, 95, 94, 94, 94, 91, 90, 88, 88, 87, 86, 8...	2476	0.50	1	0	3	HH_small_to_high
390	100	[96, 95, 94, 94, 94, 91, 90, 88, 88, 87, 86, 8...	2476	0.50	1	0	2	HH_random_to_high
391	100	[96, 95, 94, 94, 94, 91, 90, 88, 88, 87, 86, 8...	2476	0.50	1	0	13345	sequential
392	100	[98, 98, 96, 96, 95, 95, 93, 89, 87, 87, 86, 8...	2336	0.47	1	0	1	HH_high_to_high
393	100	[98, 98, 96, 96, 95, 95, 93, 89, 87, 87, 86, 8...	2336	0.47	1	0	3	HH_small_to_high
394	100	[98, 98, 96, 96, 95, 95, 93, 89, 87, 87, 86, 8...	2336	0.47	1	0	2	HH_random_to_high
395	100	[98, 98, 96, 96, 95, 95, 93, 89, 87, 87, 86, 8...	2336	0.47	1	0	6349	sequential
396	100	[98, 97, 96, 96, 96, 96, 94, 93, 92, 92, 91, 9...	2634	0.53	1	0	2	HH_high_to_high
397	100	[98, 97, 96, 96, 96, 96, 94, 93, 92, 92, 91, 9...	2634	0.53	1	0	3	HH_small_to_high
398	100	[98, 97, 96, 96, 96, 96, 94, 93, 92, 92, 91, 9...	2634	0.53	1	0	3	HH_random_to_high
399	100	[98, 97, 96, 96, 96, 96, 94, 93, 92, 92, 91, 9...	2634	0.53	1	0	10166	sequential

Figure 3. Last 12 rows of the dataset for the graphs with normal distribution

5.1 Connectivity comparison

To examine the algorithms behavior about connectivity, we plot the number of connected and disconnected graphs generated by different algorithms which we found by depth first search algorithm. After analyzing the data and plots, we have concluded that Havel Hakimi algorithm that choses smallest degree first and distributes to the highest, gives the best connectivity result for both degree distribution and highest to highest version of HH algorithm generate disconnected graphs more often rather than other algorithms. These results were expected by the definitions of the algorithms. For example, when degrees are distributed highest to highest, first higher degree vertices make connections between each other hence for the remaining small degree vertices the probability of being disconnected increases. Also, from

Figure 4. we can deduce that graphs whose degree sequence follow a binomial distribution are mostly connected graphs with high probabilities.

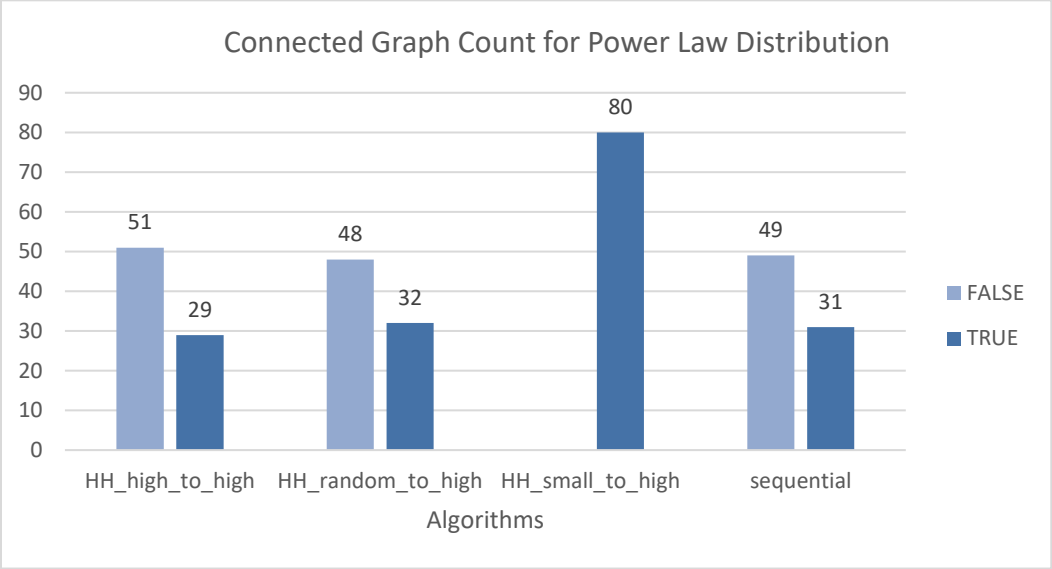

Figure 4 .

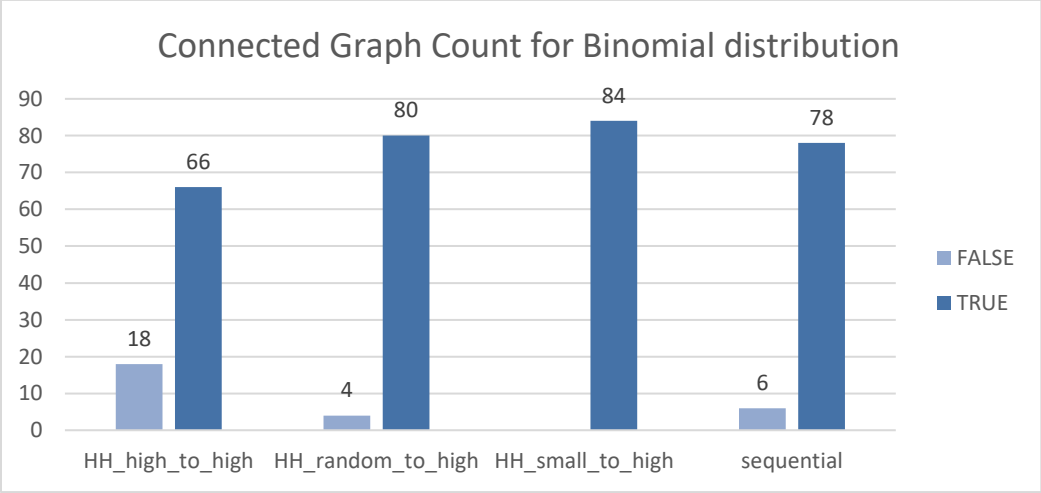

Figure 5.

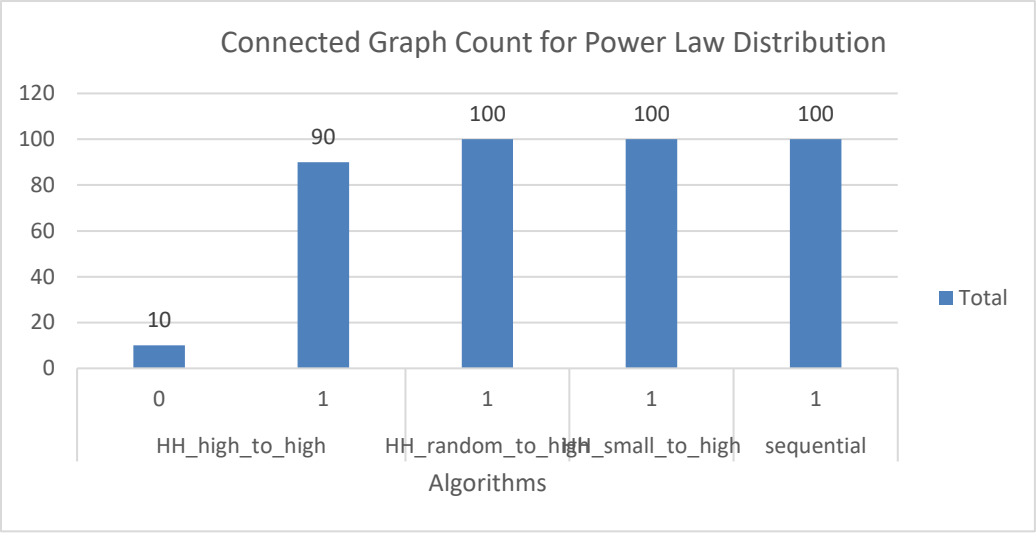

Figure 6.

Another observation was about the number of pairwise interchanges needed for an algorithm. To examine this behavior, graphs that have same degree sequence and all resulted as disconnected after generated by the algorithms high to high HH, random to high HH, and sequential are chosen. Small to high HH is skipped since it does not produce disconnected graphs. For the chosen graphs the number of pairwise interchange needed is plotted in Figure 7. From the figure one can deduce that, although the degree sequences are same, for a disconnected graph number of pairwise interchange needed is highest for high-to-high HH algorithm because this algorithm produces graphs that have many disconnected components.

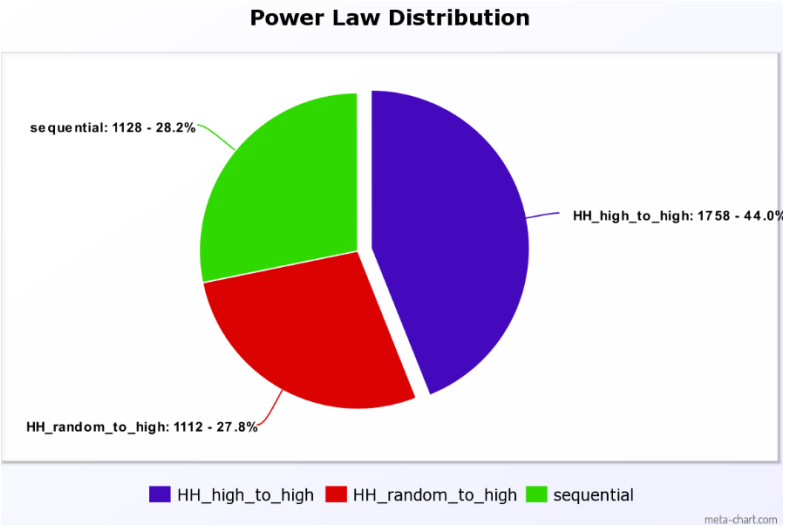

Figure 7.

5.2 Graph Density Analysis

Since it is intuitive that denser the graph, the higher the probability of being connected, we calculated the graph density and its correlation with the graph's connectivity.

The correlation for algorithms in power-law distribution:

0.8134693083441139	HH_high_to_high
0.8451034048552435	HH_random_to_high
0	HH_small_to_high
0.8071202490898047	Sequential

The correlation for algorithms in binomial distribution:

0.46705649457370174	HH_high_to_high
0.16437212271763796	HH_random_to_high
0	HH_small_to_high
0.16219239641233965	Sequential

The correlation for algorithms in normal distribution:

0.280093	HH_high_to_high
0	HH_random_to_high
0	HH_small_to_high
0	Sequential

Since binomial and normal distribution has independent edges where in power-law distribution edges are dependent according to degrees of vertices, binomial and normal distribution shows low correlation with graph density. For normal last three correlation is zero since normal distribution generated only connected graphs.

Another important remark is about the relationship between graph density and the probability of Erdős–Rényi graphs. When the graphs that are constructed with the binomial degree distribution (Erdős–Rényi graphs) are examined, it can be seen that there is a relationship between the probability p of having an edge between two vertices and graph's density. Graph density is always around the value of p and when the n gets larger the deviation

from p is getting smaller. Shortly, when number of vertices increases graph density becomes closer to the p value of the binomial distribution (see Figure 6).

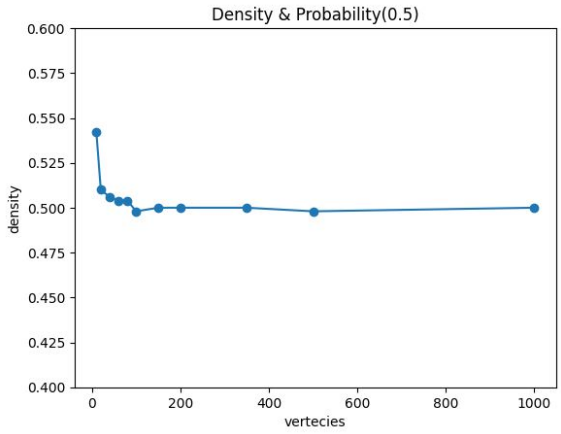

Figure 8. Graph density and its relationship with ER graphs probability

5.3 Time analysis and comparison of algorithms

To compare how fast and efficient the algorithms are working, we started collecting the time before generating the graph. We end the timing after the graph generation, search for connectivity, and pairwise interchange is completed. In Figure 5, one can see that compared to sequential algorithm, all Havel Hakimi algorithms have timing that are close to zero. Since sequential algorithm shows an exponential increase the time effect of other algorithms cannot be seen from figure 5, 6, and 7.

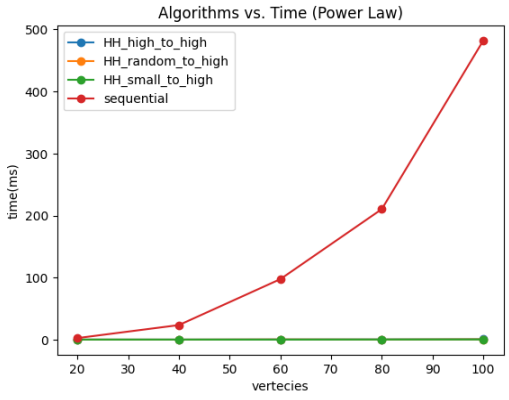

Figure 9. Algorithm time comparison sequential vs. others with power-law distribution

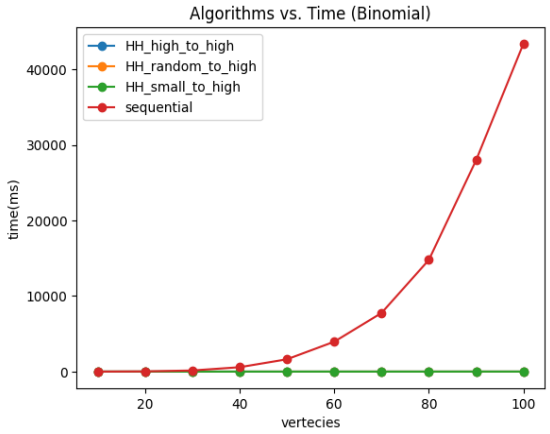

Figure 10. Algorithm time comparison sequential vs. others with binomial distribution

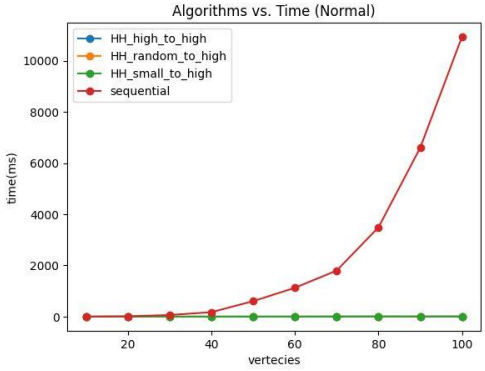

Figure 11. Algorithm time comparison sequential vs. others with normal distribution

To compare the Havel Hakimi algorithms, Figures 6 and 8 are generated. Since Havel Hakimi with smallest vertex distributed to smallest algorithm mostly causes errors, we didn't include them in the comparison since we didn't generate any graph with the algorithm. From Figure 6, one can see that for the graphs with power law distribution high to high Havel Hakimi algorithm is the slowest among the HH algorithms where small to high HH is the fastest. This occurs since in power law distribution, high to high HH constructs graphs with more disconnected parts which causes higher pairwise interchange time. Similarly small to high HH produces connected graphs hence no pairwise interchange occurs which results in slower time. For the normal and binomial distributions, time needed for the experiment does not differ significantly (see Figure 9, Figure 10) since in binomial distribution all algorithms produces connected graphs most of the time.

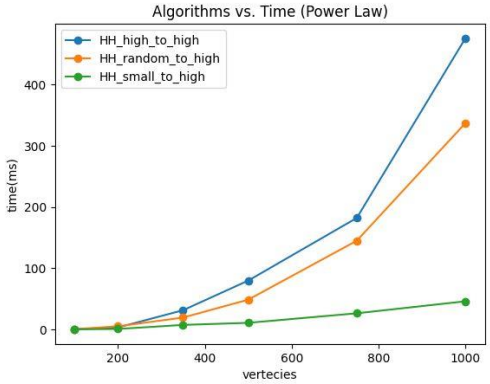

Figure 12. Algorithm time comparison for different Havel Hakimi algorithms with power-law distribution

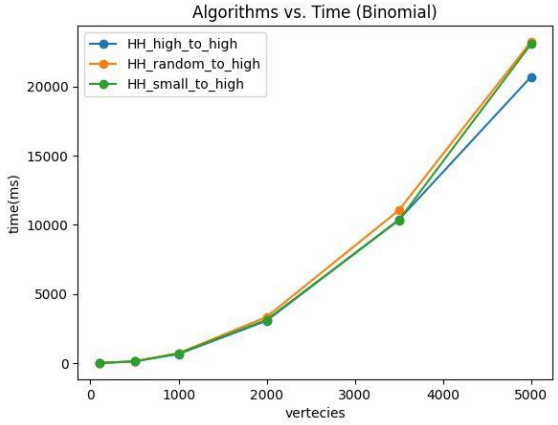

Figure 13. Algorithm time comparison for different Havel Hakimi algorithms with binomial distribution

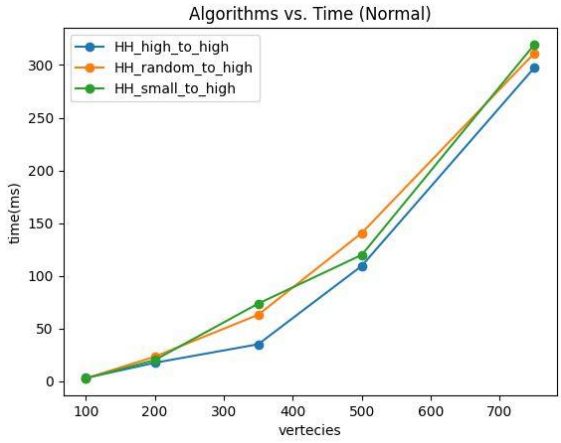

Figure 14. Algorithm time comparison for different Havel Hakimi algorithms with normal distribution

REFERENCES

[1] Sindey Render, Daniel ben- Avraham, Vishal Sood, «First-passage properties of the Erd´o´s–Renyi random,» *Journal Of Physics A: Mathematical And General,* pp. 109-123, 2005.

[2] Charles Taylor, Leonid Bogachev Boshra Alarfaj, «The joint node degree distribution in the,» 9 March 2023. [Çevrimiçi]. Available: https://arxiv.org/pdf/2303.05138.pdf. [Erişildi: 2 May 2023].

[3] Maleq Khan, Madhav V. Marathe, Maksudul Alam, «Distributed-Memory Parallel Algorithms for Generating Massive Scale-free Networks Using Preferential Attachment Model,» p. 1, 2013.

[4] Micheal Small, Kevin Judd, Linjun Zhang, «Exactly scale-free scale-free networks,» Physica A, 2014.

[5] Sheshayya A. Choudum, «A Simple Proof Of The Erdos-Gallai Theorem On Graph Sequences,» Bull. Austral. Math. Soc., cilt 33, pp. 67-70, 1986.

[6] Claude Berge, Graphs and Hypergr aphs, Paris: North Holland Publishing Company , 1970.

[7] Joseph Blitzen,Persi Diaconis, «A Sequential Importance Sampling Algorithm For Generating Random Graphs With Prescribed Degrees» June 2006.

[8] Mahadev, N. V. R. and Peled, Threshold graphs and related topics. Annals of Discrete, Amsterdam: North-Holland Publishing Co., 1995.