IE456 – Project Paper

Generating Simple Graphs Uniformly at Random with Prescribed Degree Sequence

Group Members:

Ahmet Buğra Taksuk-2017402168

Dorukhan Kılınç – 2017402093

Table of Contents

1. Introduction

A graph is a set of vertices and a set of edges between these vertices which are often used to study complex networks and relationships between the elements in them. Graph theory has wide-spread applications from studying the relationships of languages to making the evolution map of species as well as computer science and information systems. For big systems, studying different graphs and comparing the observations is a way to explore their properties. A common approach to do so is to generate random graphs and to work on them. By doing so, we can statistically test the properties of the graphs such as the expected number of separate components or average number of minimum edges to cross to reach the other vertices from a fixed vertex. Not only that, but also we can see how these systems behave if the components of the systems(i.e. vertices and edges) increase by generating random graphs with greater number of vertices and/or edges.

For example, let's consider the minimum coloring problem. The minimum coloring problem asks the minimum number of colors needed such that all the edges in a graph are colored(or labeled) and no pair of edges sharing a vertex as an endpoint are the same color. It has many applications such as job scheduling problems where we represent the jobs as vertices and we put an edge whenever two jobs conflict. Solving the minimum coloring problem in this graph would give us the minimum number of slots(slots can be time periods or locations) we need to handle these jobs. Assume that we want to study how the solution to the scheduling problem changes as we play with the connections between the vertices without changing the number of edges each vertex has, which is called the degree of a vertex. Such a study would show us if there is a vertex limiting the solution. As proposed by Blitzstein-Diaconis[3], we can study this by generating random graphs with a given set of integers representing the degree of vertices in the graph then solving the problem and taking the statistics of these solutions. Such a set of integers is called a degree sequence. Now, there are several unwanted conditions in a random graph which can disrupt this study. Firstly, it is meaningless for a random graph to have more than one edge between a pair of vertices because it doesn't have any additional meaning, two edges also mean there is a conflict . Secondly, a self loop, which is an edge whose endpoints are the same vertex, creates the same problem because what do we do when a job is in conflict with itself? We would want to study graphs which don't contain multiple edges and self-loops. Such a graph is called a simple graph. Simple graphs are used to represent various structures such as social networks or road maps in addition to job scheduling. Returning back to the

scheduling problem, assume further that instead of studying only one degree sequence, we want to see the solution for different degree sequences as well. Then, we would need to create random degree sequences before generating the realizations of them. We simply construct a set containing n randomly generated integers from 0 to n-1 to create a degree sequence. However, we encounter different problems in the creation of graphs. Assume that we have the degree sequence {1,1,1} denoting 3 vertices each having 1 edge. Notice that we can not draw such a graph because a vertex always ends up with 2 degrees. This means that there is no graph with this degree sequence, in other words this degree sequence is not graphical. Then, the question rises: when is a given degree sequence graphical? An immediate observation would be that for a degree sequence to be graphical, the sum of degrees must be an even number. This is because each edge is counted exactly twice, one for each endpoint. But is it enough? Apart from that, another question would be that does the degree of each vertex tend to increase as the number of vertices in a system increases? This question is crucial in the implementation of random graph generation. As Blitzstein-Diaconis[2] claims, it takes much longer to implement more complex random graphs and we want to reduce the run times of the algorithms. Changing the distribution of the vertex degrees might help this matter. But when and how should this be implemented? Lastly, another thing to consider is that a degree sequence itself is not enough to construct a graph since it doesn't specify which vertex is adjacent to(connected with an edge) which vertices, i.e. we need methods to generate graph realizations from them.

The aim of this paper is to examine how to create random simple graphs mainly with a given degree sequence when it is graphical and efficiently implement and compare different realization methods such that they are applicable in real life. Section 2 will introduce the literature review regarding random graph generation methods, theorems to check if a degree sequence is graphical and algorithms to have realizations of such methods. Section 3 will give some basic definitions and the theorems that will be implemented with their proofs. In sections 4 and 5, we will give the algorithms to be implemented and share the results. Lastly, we will conclude our paper with a discussion of our results in section 6.

2. Literature Review

For a given set of vertices, we can generate a random graph by assigning probabilities to form an edge between two vertices for all possible pairs of vertices and then by generating random numbers to find a set of edges. Newman[8] says that there are several choices to assign these probabilities and emphasizes the choice that whether or not selecting two different edges to be in the graph are independent events.

Erdos-Renyi[5] proposes a special way to generate random graphs from a given number of vertices and edges. Their approach is to start with a vertex set with 0 edges, and to choose a random edge among all the possible edges with each edge having the same probability of being chosen. Here, an edge is selected by selecting a pair of vertices. Then we proceed by selecting another edge among the remaining edges. We continue this process until the number of edges we desire is acquired. This method allows us to generate random graphs uniformly, i.e. no edge is biased to be chosen and the probabilities of choosing 2 different edges at a given step are always independent. We can also implement this method by giving a probability instead of the number of edges since all the edges have the same probability of being chosen. Further, it gives a realization, i.e. we have all the information we need to draw the graph. However, notice that this method would give us different degree sequences every time we use it.

The most natural way to get a degree sequence is to generate random integers, as discussed in the introduction. However, recall that they are not guaranteed to be graphical. Fortunately, there are 2 methods proposed to check this. First one is proposed by Havel[6] and Hakimi[7] at different times. Havel-Hakimi theorem proposes a recursive algorithm to check whether a given degree sequence is graphical, which will be given in the next section. In addition, using the Havel-Hakimi theorem, we can get a deterministic realization of a given degree sequence[3]. Secondly, Erdos-Gallai[4] proposes another theorem to check whether a degree sequence is graphical. This theorem requires the degree sequence to be in a nonincreasing order and gives a list of inequalities to be satisfied. Erdos-Gallai theorem and its proof will also be given in the next section.

Once we have a graphical degree sequence, now comes to find a realization of it. Above, we said that the Havel-Hakimi theorem can be used[3]. However, it doesn't give us a random graph. To obtain a random graph from a given degree sequence, we will consider 2 algorithms. First

one is the pairing model, given by Wormald[9]. Pairing model suggests creating multiple copies of each vertex equal to their degrees and then finding random pairings among these new vertex sets, each pairing corresponding to an edge. The probability of pairing two vertices is equally distributed. However, to get a simple graph with this method, we sometimes need to run the algorithm multiple times since it doesn't guarantee that there will be no loops or multiple edges(for example, two vertices which are the duplicates of the same vertex in the original graph can be chosen). The second one is the sequential algorithm, from Blitzstein-Diaconis[2]. This algorithm works by choosing the vertex with minimum non zero degree and constructs a candidate vertices list for this vertex to have an edge with. A vertex is put into the candidate list if putting an edge between them does not make the rest of the degree sequence not graphical and currently they are not connected with an edge. Then a vertex from this candidate list is randomly chosen by assigning probabilities of being chosen as proportional to their degrees. Then the algorithm updates the degree sequence and runs until the sequence is all zeros. They both give opportunities to uniformly generate random graphs, which will be discussed in section 4. In addition, compared to similar studies, Blitzstein-Diaconis[2] gives an algorithm that is implemented with a prescribed degree sequence while there are many algorithms suggested by other publications that fail to directly do it such as [4],[5],[8].

Lastly, considering the generation of the random degree sequence, there are several propositions. Graphs generated by using Erdos-Renyi[5] method, which we will refer as Erdos-Renyi graphs, have a special degree distribution among the vertices, which is binomial. As the number of vertices in an Erdos-Renyi graph increases, this distribution converges to a poisson distribution. Secondly, Barabasi-Albert[1] claims that in a complex graph with a large number of vertices, the probability of a vertex having k edges is proportional to $k^{-\gamma}$ where γ is a characteristic constant for the system. For example, in a graph in which scientific publications are represented as vertices and the citations are as edges, the distribution of vertex degrees are in accordance to their claim and with γ = 3. Graphs representing the systems with such a distribution of vertex degrees are called scale-free graphs, since an increase in the number of vertices would not affect the distribution of the vertex degrees in such a graph. Scale-free graphs are often useful to describe complex system as discussed above. Barabasi-Albert[1] further claims that this γ should be generalized as around 3. We will use these 2 methods and uniform distribution to generate random degree sequences and compare them according to the proportion of the graphical sequences and algorithm implementation times.

3. Havel-Hakimi and Erdos-Gallai Theorems

Before giving the theorems, we first give some useful terminology:

Definition: A degree sequence S is said to be **graphical** if a graph G with |S| many vertices can be drawn such that each entry in S corresponds to the degree of one of the vertices.
Example:
The degree sequence {2, 2, 2, 2} is graphical since a rectangle whose corners are represented by vertices and edges by edges corresponds to a graph having vertices as this sequence.
The degree sequence {1,1,1} is not graphical as discussed in the introduction.

Theorem(Havel[6]-Hakimi[7]): Let d be a degree sequence with $|d| \geq 2$, all entries of d nonnegative. For d to be graphical, the number of nonzero entries of d must be greater than all the entries of d. Assume so. Let $\overline{d}$ be the degree sequence we get by deleting the $d_{max} \in d$ where d_{max} is the maximum of d and subtracting one from the d_{max} remaining highest entries. Then, d is graphical if and only if $\overline{d}$ is graphical. If so, d has a realization in which each edge is between a deleted vertex and a vertex whose degree is decreased by 1 in that step.

Proof(by Blitzstein-Diaconis[2]): First part is trivial since for a vertex to have the given degree, there must be at least that many vertices with nonzero degrees. Considering the second part, it is also trivial that if $\overline{d}$ is graphical, so is d since we only add an edge to $\overline{d}$ to get d, and similar logic applies to the converse. Lastly, assume we have a graph G with such a degree sequence. We need to show that we can get the graph prescribed in the theorem by switching some edges in G without changing the degree sequence. We only need to do it for the vertex of highest degree since we can recursively do the switchings by removing that vertex and the corresponding edges from the graph and applying the same algorithm until we end up with an empty set. We first start by considering the vertex with maximum degree in G, call it v. If it is only connected to the vertices of highest degree other than v, then we are done. If not, then v is connected to an unwanted vertex, call it w. Similarly, there is a vertex that we want v to be connected to and they are not connected at the moment, call it u. Delete the edge between v and w and connect v and u. Right now v has the same degree but u has one more and w has one less. Take care of this by finding a vertex y different from v which is connected to u but not to w. Remove the edge between u and y and add one between w and y. We are done.

Theorem(Erdos-Gallai[4]): Let d be a degree sequence whose sum is an even number with all entries nonnegative and in nonincreasing order. Then d is graphical if and only if

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{j=k+1}^{n} min(k, d_j) \quad \forall k \in \{1,\dots,n\},\ n = |d|$$

Proof(by Choudum [3]): Informally stating, the theorem says that the sum of degrees in a set of vertices must be at most equal to the case at which all the vertices belonging to this set are connected to all the other vertices in the graph. The term $k(k-1)$ denotes the connections in the set and the summation term is to count the edges to the vertices outside of the set. [3]

For a more concrete proof, let s denote the sum of the entries of d. We will prove the theorem by induction on s. The theorem holds for $s = 0$ and $s = 2$. Further, notice that if the theorem holds for a sequence, it will also hold for the sequence we get by appending zeros to the end of it. This is not only apparent from the above inequality since only the right hand side increases but also makes sense because addition of an isolated vertex would not make any difference. Similarly, we can work with a sequence whose last entry is nonzero without loss of generality. So, let $d_n > 0$ and the d satisfies the conditions of the theorem. Now, let t be de smallest integer such that $d_t > d_{t+1}$, i.e. t is the first index we observe a decrease in the degree sequence. If d consists of all equal entires, let $t = n-1$. Consider the sequence we get by subtracting one from the entries d_t and d_n, i.e. $d' = \{d_1, d_2,\dots, d_t - 1,\dots, d_{n-1}, d_n - 1\}$. We will show that the conditions given in the theorem are satisfied for d' as well.
Let k be an integer with $1 \leq k \leq n$. We consider the following conditions:

(1) $k \geq t$.

$$\sum_{i=1}^{k} d_i - 1 \leq k(k-1) + \sum_{j=k+1}^{n} min(k, d_j) - 1 \quad \text{by the assumption on } d$$

$$\leq k(k-1) + \sum_{j=k+1}^{n-1} min(k, d_j) + min(k, d_n - 1) \quad \text{using } min(a,b) - 1 \leq min(a, b-1)$$

So, the conditions given in the theorem are satisfied for all $k \geq t$ in d'.

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

The inequality is clearly satisfied for k by $d_k \leq k - 1$.

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

Notice that there exists at least 2 terms whose indices are greater than k (we know for sure that there are t and $t+1$ by our choice of t). If there are more than 2 terms, clearly $\sum_{i=k+2}^{n} d_i \geq 2$. If there are exactly 2 terms, than $t = n - 1$. By the definition that $t+1$ is the first decrease in our sequence, we get $s = (n-1)(n-2) + d_n$. For this sum to be an even number d_n must be even as well, which means that it must be at least 2 by our assumption that all the degrees are nonzero. So, we have

$\sum_{i=1}^{k} d_i = k^2 - k + d_{k+1}$ (by $d_{k+1} = k$ as well)

$\leq k^2 - k + d_{k+1} + d_{k+2} + \ldots + d_n - 2$

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

Which shows that the theorem holds for such k. In this condition, we used that the sum of degrees after k being greater than 2 to use it as minus ones on t and n, which is exactly what we wanted to find.

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

$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{j=k+1}^{n} min(k, d_j)$ (by the theorem)

Using the fact that the last term has a degree greater than k, we have

$min(d_i, k) = min(d_i - 1, k) \quad \forall i$ which implies

$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{j=k+1, \neq t}^{n-1} min(k, d_j) + min(d_t - 1, k) + min(d_n - 1, k)$

We are done.

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

Assume for a contradiction that we have

$k(k+1) \geq \sum_{i=1}^{k} d_i > k(k-1) + \sum_{j=k+1}^{n} min(k, d_j)$ in d'

Which implies

$$2k > \sum_{j=k+1}^{n} min(k, d_j) = \sum_{j=k+1,\neq t}^{n-1} min(k, d_j) + min(d_t - 1, k) + min(d_n - 1, k)$$

$min(d_t - 1, k) = min(d_t, k)$ by $d_t > k$ and $min(d_n, k) = min(d_n, k) - 1$ by $d_n \leq k$.

Plugging these, we get

$$2k > \sum_{j=k+1,\neq t}^{n-1} min(k, d_j) + min(d_t, k) + min(d_n, k) - 1$$

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

Which contradicts the assumption that d satisfies the conditions given in the theorem by:

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{j=k+1}^{n} min(k, d_j) < k(k-1) + k + 1 = k^2 - 1$$

However, by $k < t$ we have:

$\sum_{i=1}^{k} d_i = (k+1)(k-1) < k^2 - 1$. Contradiction.

By proving that the theorem holds for the above 5 conditions, we have completed all cases, i.e. the proof is complete.

4. Implementations

4.1 Implementation of Havel-Hakimi Theorem to Check if a Degree Sequence is Graphical or not

To implement the Havel-Hakimi theorem to check if a given degree sequence is graphical, we start by checking some basic conditions. Although they are not alone sufficient for a degree sequence to be graphical, they offer a good starting point. (Consider {3,3,3,1} for an example. It is not graphical though it satisfies the basic conditions)

- Firstly, if the sequence consists of negative integers, it is obviously not graphical.
- Secondly, a degree sequence is not graphical if the sum of degrees in this sequence is an odd number, as discussed in the introduction.

- Lastly, for each entry in a given degree sequence, there must be at least that many positive entries other than the degree given by this entry because each degree d requires d vertices.

We now proceed to implementation of the algorithm. The above 3 conditions will be referred to as basic conditions.

Algorithm:

- Input: A degree sequence
- Output: True if the degree sequence is graphical, False otherwise
- Algorithm:

1. If the sequence is the 0 vector, return True, terminate the algorithm.
2. If the sequence does not provide the basic conditions, return False, terminate the algorithm.
3. Sort this sequence, call it s'.
4. Delete first entry of s', let d be the degree corresponding to the degree of this deleted element, subtract 1 from the first d entries. Call this sequence s''.
5. Call the algorithm by providing s''.

4.2 Implementation of Havel-Hakimi Theorem to Find a Realization of a Given Degree Sequence

Only addition in this algorithm to the one above is that we start with an empty edge set and whenever we delete a vertex v (which is the vertex of highest degree at each step of the Havel-Hakimi algorithm) and subtract 1 from the degree of a vertex w (that means deg(w) is one of the largest deg(v) vertices in the corresponding step of the Havel-Hakimi algorithm), we add (v,w) to the edge set.

4.3 Pairing Model

Although the steps of the pairing model are pretty similar to that explained by Wormald[9], we added a few control mechanisms to reduce the computation time. Before that, let us define some terms to refer further in the explanation:

Vertex Array: The array of vertices gotten by taking each vertex with label i degree(i) times. Here, the labels are the indices of the entries in the original degree sequence.

Neighborhood: Neighborhood of a vertex v is a set containing all the vertices which are connected to v with an edge.

Remaining Degree Sequence: The degree sequence we get by adding the edge (v,w), i.e. subtracting 1 from the entries v and w in the degree sequence.

Adjacency List: A list consisting of lists denoting the neighborhood for each vertex. For example, 2nd entry of the adjacency list is a list denoting the neighborhood of the 2nd vertex.

Added Control Mechanisms:

- Firstly, we only consider the pairings which do not correspond to a loop or an existing edge by taking a vertex v from the vertex array and then limiting the second choice to the part of the vertex array which does not correspond to any vertex in the neighborhood of v. This doesn't change anything in the original algorithm since we are guaranteed to rerun the algorithm if the second vertex is not from this set. We will call this subset of vertex array as the valid vertex array.
- Secondly, we add the edge if the remaining degree sequence is still graphical.(using havel-hakimi theorem)
- Thirdly, instead of choosing two vertices at random, we first choose the vertex with the highest degree as the first vertex and then choose the second vertex at random from the valid vertex array. This tweak reduces the run time significantly since the probability of rerunning the algorithm increases as we leave dealing with the vertices with highest degrees later.
- Lastly, we restart the algorithm whenever in the remaining degree sequence and the adjacency list there is a vertex for which the set of second chances is an empty set, i.e. we are guaranteed to end up with a loop or an edge which is already formed. This reduces the run time by removing unnecessary iterations.

Algorithm:

- Input: A graphical degree sequence
- Output: Adjacency list
- Steps:

1. Create a vertex array from the input degree sequence, a degree sequence equal to the input degree sequence and an empty adjacency list.
2. If the desired number of edges is acquired, return adjacency list and terminate the algorithm.
3. If the vertex array and adjacency list definitely correspond to a multigraph, go to step 1.
4. Choose the vertex with highest degree, call it v, and construct the valid vertex array.
5. Choose a vertex at random from the valid vertex array, call it w. Update the adjacency list, and delete the vertices of this edge from the vertex array. Update the degree sequence by subtracting one to the corresponding edges. Go to step 2.

4.5 Sequential Algorithm

- Input: A graphical degree sequence, an empty edge set
- Output: An edge set
- Algorithm:

1. If the sequence is the empty sequence, return the edge set, terminate the algorithm.
2. Let v be the vertex implied by the minimum nonzero entry of the sequence.
3. Create an empty candidate list.
4. Iterate over the sequence:
5. If (w,v), w !=v is not in the edge list and the remaining degree sequence is graphical: add w to the candidate list
6. Assign probabilities to the candidate vertices proportional to their degrees.
7. Randomly choose a candidate according to the given candidate probabilities, call u.
8. Return to step 1 by updating the sequence such that the vertices v and u has 1 less degree and the edge set by adding the edge (u,v)

4.6 Random Degree Sequence Generation

We propose 3 different methods to create degree sequences:

1. Uniform method: We generate n vertices whose degrees vary from 0 to n-1 and each degree is equally likely to be chosen for all vertices.

2. Erdos-Renyi method: We generate n vertices where the degree distribution follows a poisson distribution with a parameter λ denoting the mean number of edges for a vertex

we get by taking the limit of the binomial distribution suggested by Erdos-Renyi[5]. Here we'll choose different λ's to compare the performances.

3. Scale-free method: We generate n vertices each having the probability of having a degree k proportional to $k^{-\gamma}$. We will take $\gamma = 3$ as Barabasi-Albert[1] suggests.

5. Results

We implemented these algorithms in python using libraries numpy to perform sequence operations, scipy to implement poisson distribution and pandas for output tables.

5.1 About the Degree Sequence Generation Algorithms

To test the performances of the methods given in section 4.6, we generated 100 sequences for different conditions and here below is the table for the mean statistics for each case:

			Ratio of Graphical Sequences	Average Degree	Average Time Elapsed During Sequence Generation
Method	**Parameter**	**Number of Vertices**			
Erdos-Renyi	**4**	**100**	0.46	3.97520	0.008264
		250	0.51	3.97564	0.019454
		500	0.50	3.97952	0.039425
		1000	0.53	3.98777	0.078765
	5	**100**	0.46	4.97380	0.008140
		250	0.48	4.97088	0.019507
		500	0.54	4.97800	0.039844
		1000	0.52	4.98794	0.079095
	6	**100**	0.57	5.96610	0.008076
		250	0.40	5.96608	0.019532
		500	0.46	5.97352	0.039876
		1000	0.55	5.98547	0.080457
scale-free	**3**	**100**	0.54	1.34680	0.000096
		250	0.39	1.35804	0.000101
		500	0.48	1.35560	0.000136
		1000	0.53	1.36455	0.000174
uniform	**0**	**100**	0.11	49.36090	0.000036
		250	0.04	124.82216	0.000036
		500	0.03	249.68264	0.000043
		1000	0.04	498.98622	0.000048

Table 1

Notice that for both of the Erdos-Renyi and scale-free methods, the average degree of a vertex is independent of the size of the graph, i.e. the total number of vertices. For Erdos-Renyi case, notice further that the statistics agree with our parameters. From this table, we would conclude that the uniform case is not suitable for modeling complex networks because the average number of degrees for a vertex increases and the ratio of graphical sequences decreases as the size gets larger. We will move further with sequences generated by other 2 methods.

5.2 About the Graph Realization Algorithms

We created 280 degree sequences with 25, 50 and 75 vertices using scale-free method with $\gamma = 3$ and Erdos-Renyi method with $\lambda = 4, 5, 6$. In this section, we will give some summary statistics. See the appendix for the whole table.

Method Used to Generate	Number of Vertices	Parameter	Is graphical
Erdos-Renyi	**25**	**4**	0.56
		5	0.44
		6	0.48
	50	**4**	0.52
		5	0.32
		6	0.52
	75	**4**	0.35
		5	0.50
		6	0.45
Scale-free	**25**	**3**	0.40
	50	**3**	0.32
	75	**3**	0.40

Table 2.

Here above is the table denoting the ratio of graphical sequences generated by the corresponding methods. We generated 25 sequences for each scenario and kept the number of

vertices at these levels because of computational obstacles. Now we proceed with the algorithm run times.

Method Used to Generate	Number of Vertices	Parameter	HH Runtime	PM Runtime	SA Runtime
Erdos-Renyi	**25**	**4**	0.000274	0.072441	0.510239
		5	0.000303	0.130379	0.707886
		6	0.000323	0.145336	0.945809
	50	**4**	0.000594	0.404299	6.471008
		5	0.000657	0.481223	9.263134
		6	0.000691	0.682561	12.066090
	75	**4**	0.000961	1.067426	30.977562
		5	0.000999	1.691729	41.703024
		6	0.001080	1.807936	53.933346
Scale-free	**25**	**3**	0.000205	0.059521	0.166788
	50	**3**	0.000338	0.462274	2.024249
	75	**3**	0.000536	0.605054	9.243680

Table 3.

As can be seen above, the Havel-Hakimi algorithm has significantly less mean run times. This is simply because it is a deterministic algorithm while the other two has randomness. Other than that, notice that the run times for each algorithm increases as the number of vertices or edges(edges are implemented by λ in Erdos-Renyi method) because the algorithms need more iterations to deal with them. However, the run time for sequential algorithm increases significantly as these numbers increase. The average run time for sequential algorithm for a graph with 75 vertices with each having around 6 edges is around 54 seconds, which would mean a huge run time if we wanted to examine sequences with hundreds of vertices. However, we can say that the scale free algorithm offers a better chance to work with such sequences. Lastly, notice that the pairing model beats the sequential algorithm in terms of run times. This is thanks to the intermediary control steps and reductions of the possible scenarios we implemented during the implementation of pairing model.

6. Conclusion and Further Remarks

Graphs are utilized frequently to model the complex systems whose elements have relations to each other. One way to perform experiments in these systems is to generate random graphs and to work on them. Although there are many methods to generate random graphs[1], Erdos-Renyi[5] and Barabasi-Albert[1] suggests two methods that can be implied to these systems via generation of degree sequences. However, not all degree sequences represent can be used to construct a graph, which requires us to have a method of deciding it. For this matter, we use two theorems given by Havel[6]-Hakimi[7] and Erdos-Gallai[4] and their proofs [2][3] to check this condition. Then, we proceed by introducing various algorithms given by [2][6][7][9]. We further make some improvements to the pairing model via introducing control steps. Lastly, we implement these algorithms in python to compare their performances. Scale-free algorithm looks like the best algorithm to generate a random degree sequence which is applicable to real systems and the pairing model seems like the most efficient algorithm to generate realizations of the random degree sequences.

For further improvements, more computational power can be utilized to test the performances of sequential algorithm and pairing model. We had restricted time to implement them, so had only realizations for 25, 50 and 75 vertices. Secondly, the algorithms may be tested to see how they work with real data sets instead of synthetic sequences if such data sets can be accessed.

7. References

[1] A.-L. Barabasi and R. Albert, Emergence of scaling in random networks, Science 286 (1999), 509–512.

[2] Blitzstein, Joseph & Diaconis, Persi. (2010). A Sequential Importance Sampling Algorithm for Generating Random Graphs with Prescribed Degrees. Internet Mathematics. 6. 10.1080/15427951.2010.557277.

[3] Choudum, S. A. (1986). A simple proof of the Erdös-Gallai theorem on graph sequences. Bull.Austral. Math. Soc. 33, 1, 67–70

[4] Erdös, P. and Gallai, T. (1960). Graphen mit punkten vorgeschriebenen grades. Mat. Lapok 11, 264–274.

[5] Erdös, P. and Rényi, A.. "On the evolution of random graphs". The Structure and Dynamics of Networks, Princeton: Princeton University Press, 2011, pp. 38-82.

[6] Havel, V. (1955). A remark on the existence of finite graphs. ˇCasopis Pˇest. Mat. 80, 477–480.
[7] Hakimi, S. L. (1962). On realizability of a set of integers as degrees of the vertices of a linear graph. I. J. Soc. Indust. Appl. Math. 10, 496–506.
[8] Newman, M. E. J., Strogatz, S., and Watts, D. (2001). Random graphs with arbitrary degree distributions and their applications. Physical Review E 64, 026118.
[9] Wormald, N. C. (1999). Models of random regular graphs. In Surveys in combinatorics, 1999 (Canterbury). London Math. Soc. Lecture Note Ser., Vol. 267. Cambridge Univ. Press, Cambridge, 239–298.