Volume: 15 Issue: 01 Pages: 5781 - 5785 (2023) ISSN: 0975-0290

Finding All Graphic Sequences and Generating Random Graphs

Brijendra Kumar Joshi
Indore Institute of Science and Technology, Indore - 453331
Email: brijendrajoshi@yahoo.com

ABSTRACT

Graphs are basic requirement of research in many fields like electrical circuits, computer networks, Genome sequencing, traffic flow, compiler design and cryptography to name a few. There are three fundamental issues one has to address in respect of an undirected graph. First, given a number of nodes, how many unique degree sequences can be there to handle? Second, out of these unique degree sequences, how many are graphic? And third, how to generate random graphs for a degree sequence? This paper presents working solutions for all these issues. An efficient algorithm to find all unique degree sequences for a given number of nodes is presented. An algorithm based on Havel-Kasami algorithm is presented which is more efficient than Erdos-Gallai algorithm to test the connectivity. Finally, an algorithm to generate random graphs is also presented.

Keywords – **Adjacency list, connected graph, degree sequence, Erdos-Gallai, Havel-Kasami.**

Date of Submission: April 30, 2023 Date of Acceptance: June 03, 2023

I. INTRODUCTION

The first proven application of graphs dates back to as early as 1736. It was Euler who used graphs to solve the classical Kownigsberg bridge problem. Since then, the applications of graphs have exploded exponentially in number. It would not be an exaggeration to say that among all mathematical structures, graphs are most cited and used. Deo [1] and Harary [2] have given excellent treatment to the subject of graphs. One can refer to Choudham [3] for a very basic treatment.

Formally, a graph G can be defined as an ordered pair (V, E) i.e., $G = (V, E)$ where V is a finite set of vertices or nodes or points and E is a set of edges or arcs. More formally,

$$V = \{v_0, v_1, \ldots, v_{n-1}\},\ 1 \le |V| < \infty$$

and

$$E = \{(v_i, v_j): v_i, v_j \in V \text{ and } i \neq j\},\ 0 \le |E| < \infty$$

Here, |V| and |E| mean the number of elements in respective sets. If $i = j$ then the edge is called a self-loop. Such graphs and the graphs in which a pair of vertices has more than one edge between them are beyond the scope of this paper.

A graph is said to be a *directed graph* if its edges (v_i, v_j) are ordered pairs i.e. (v_i, v_j) is not same as (v_j, v_i). The first vertex is known as the tail of the edge and the second vertex is known as the head of the edge. There is an arrow mark on the edge (v_i, v_j) indicating the direction of the edge. In case of un-directed graphs, the pair (v_i, v_j) is un-ordered i.e., it is same as (v_j, v_i) and there is no arrow mark on the edge. This paper considers only undirected graphs.

Degree of a vertex is the total number of edges incident on it. For undirected graphs, degree of a vertex is the number of distinct edges that either emanate from or terminate on the vertex.

A *path* is a sequence of distinct edges between two distinct vertices v_i and v_j in which neither any edge nor any vertex is repeated. By definition, every edge is also a path between participating vertices.

A graph is said to be *connected graph* if there is at least one path from every vertex to every other vertex. In other words, it is a graph in which we can reach from any vertex to any other vertex. We can also use the term *spanning graph* for the purpose. This paper deals with only connected graphs.

When we write the degrees of all vertices of a graph in the form of a sequence of non-negative integers, that sequence is referred to as the *degree sequence*.

Obtaining the degree sequence of a graph is straightforward but the reverse is not as simple. The problem could be that the given sequence of integers may not result into a connected graph or may not be realizable into a graph at all. For example, the degree sequence 1, 2, 3 would definitely be impossible to realize into an undirected and connected graph without any self-edges. The sequence of non-negative integers that represents the degree sequence of an undirected and connected graph is known as the *graphic sequence*.

To know whether a degree sequence is unique or not, and if it is, to know whether it graphic or not, and if it is, to find at least one random graph of such a degree sequence is the problem of great interest. Such a large collection of random graphs would be very useful in testing graph isomorphism algorithms. For example, suppose there are three degree sequences, (1, 2, 2), (2, 2, 1) and (2, 1, 2). It is obvious that all the three are identical sequences (written differently), and hence if any one of them is graphic, so would be other two. So, basically there are three problems in one i.e.

- Given the number of nodes, find out how many unique degree sequences are possible and find all of them.
- Given unique degree sequences, find out how many of them are graphic and find all such sequences.
- Given a graphic sequence, find at least one random graph to realize the sequence.

II. Related Work

Though the degree sequences and graphs thereof have been attempted by many but the most important work on characterization of graphic sequences is by Erdos and Gallai [4]. The degree sequence (d_1, d_2, ... , d_n) is graphic if and only if it satisfies the following conditions:

1) The sum of the degrees is even i.e.

$$\left(\sum_{i=1}^{n} d_i\right) \% 2 = 0$$

2) If the sequence is in nonincreasing order then it must follow:

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

We would refer to these conditions in this paper as EG condition (1) and EG condition (2).

Tripathi and Tyagi [5] give a simpler criterion for degree sequences to be graphic.

Many people have given proof of Erdos-Gallai theorem but the proof given by Tripathi, Venugopal and West [6] is short, constructive and concise.

Suppose we know that the degree sequence given is graphic, then the next problem is to find a simple graph for it. Havel did it in 1955 [7] and later independently by Hakimi in 1962 [8]. The algorithms given by Havel and Hakimi are jointly known as Havel-Hakimi theorem / algorithm / method etc.

III. Finding All Unique Degree Sequences

The first problem at hand is to find all unique degree sequences that may or may not be graphic for a given number of nodes. Suppose the number of nodes N is 3, then there are 8 possible degree sequences. They are, (1, 1, 1), (1, 1, 2), (1, 2, 1), (1, 2, 2), (2, 1, 1), (2, 1, 2), (2, 2, 1) and (2, 2, 2). Out of these only four are unique. (1, 1, 1) is unique. (1, 1, 2), (1, 2, 1) and (2, 1, 1) are same i.e., permutations of (1, 1, 2). (1, 2, 2), (2, 1, 2) and (2, 2, 1) are same i.e., permutations of (1, 2, 2). Finally, (2, 2, 2) is unique. So we get only 4 unique degree sequences out of 8 possible degree sequences for a given number of nodes i.e., 3 which is $(N-1)^N$. It is trivial to show that out of these 4, only 3 are graphic as (1, 1, 1) is not graphic.

It is quite obvious that if a degree sequence is graphic, then all of its permutations will also be graphic, similarly, if the sequence is not graphic then so will be all of its permutations. Therefore, it is important to find all unique degree sequences for a given number of vertices so that later we can test whether a degree sequence is graphic or not.

III.1. *UDS: An Algorithm for Unique Degree Sequences*

The algorithm UDS heavily depends on two class templates (of C++) Digit and myvector [9]. Though these classes are generic, only their integer form is used. In algorithm 1, *Digit* is a class which has three attributes, *value* which is an integer representing the degree of corresponding vertex, *lower_bound* which represents the integer below which the *value* cannot go and *upper_bound* above which the *value* cannot go.

The names of the functions are self-explanatory. The algorithm is invoked with a vector of length N where N is the number of vertices and this vector is initialized with the degree sequence (1, 1, …, 1) with N 1s. The algorithm is repeated until the output is a vector of length N with the degree sequence ($N-1$, $N-1$, …, $N-1$).

Algorithm 1 UDS: An Algorithm for Unique Degree Sequences

```
Input: N (the number of vertices/nodes), vector = the degree sequence to start with
Output: vector = next unique degree sequence
size = number of vertices
vector = a vector of objects of class Digit of length size
adjust = false
change_flag = false
d = object of class Digit which is at the end of vector
d++
vector[size-1] = d
if (d.GetValue() = = d.GetLowerBound()) then
  change_flag = true
  Adjust = true
end if
for (i = size-2; i ≥ 0 && change_flag = = true; i- -) do
  d = vector[i]
  d++
  vector[i] = d
  if (d.GetValue() = = d.GetLowerBound()) then
    change_flag = true
  else
    change_flag = false
  end if
end for
maxvalue = an object of class Digit
if (adjust) then
  maxindex = index of the biggest element of vector
  maxvalue.SetValue(maxvalue.GetValue())
  for (i = maxindex+1; i < size; i++) do
    vector[i].SetValue(maxvalue.GetValue())
  end for
end if
return vector // The next unique degree sequence
```

Table captions appear centered above the table in upper and lower case letters. When referring to a table in the text, no abbreviation is used and "Table" is capitalized.

IV. FINDING ALL GRAPHIC SEQUENCES

After finding all unique degree sequences for a given number of vertices, the next problem is to find all possible unique degree sequences that are graphic. The EG condition 2 given in Equation (2) is a bit difficult to understand and implement. It also does not guarantee that if a sequence passes both EG condition (1) and EG condition (2) then it would be graphic. For example, the degree sequence (1, 1, 1, 1) passes the first condition that sum of degrees is even, here it is 1+1+1+1 = 4. The *min()* function of EG condition (2) will always result in 1 as the minimum value of k is 1 and all degrees in the sequence under consideration are 1.

The values of k will be 1, 2, 3 and 4. It is to be noted that for second sum in EG condition (2), the value of $k = 4$ results in $i = 5$ whereas d_5 is not defined. Therefore, the maximum value of k cannot be n. So, for correct understanding, the range of k must be mentioned in EG condition (2) as $1 \leq k < n$.

Now, for $k = 1$; EG condition (2) results in

$$1 \leq 1\times0 + (1+1+1) \text{ i.e. } 1 \leq 3$$

for $k = 2$;

$$(1+1) \leq 2\times(2-1) + (1+1) \text{ i.e. } 2 \leq 4$$

and finally for $k = 3$;

$$(1+1+1) \leq 3\times(3-1) + (1) \text{ i.e. } 3 \leq 7.$$

It means that the degree sequence (1, 1, 1, 1) passes both the tests but we cannot construct an undirected graph from it. Fig. 1 shows an undirected graph that has the degree sequence (1, 1, 1, 1) and satisfies both the EG conditions and still not connected.

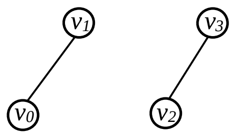

Figure 1: Disconnected graph passing Erdos-Gallai tests

IV.1. Potentially Connected Degree Sequence

We know that Erdos-Gallai algorithm does not guarantee the connectedness of graph resulting from corresponding degree sequence. Algorithm 2 checks whether the degree sequence input is potentially connected or not. By *potentially connected* we mean that the degree sequence passes various tests. Algorithm 2 also does not guarantee the connectedness but it tests for more conditions on the degree sequence.

The algorithm returns *true* if the degree sequence is potentially connected else it returns *false*.

Algorithm 2 PCDS: An Algorithm for Potentially Connected Degree Sequence

Input: vect = the degree sequence in nondecreasing order
Output: true or false

```
1:  sum = Sum of degrees of vect
2:  size = length of vect
3:  if (vect[size-1] ≥ size) then
4:    return false
5:  end if
6:  if (sum % 2 != 0 || sum < 2*(size-1)) then
7:    return false
8:  end if
9:  max_degree = number of times degree is (size-1) in vect
10: if (max_degree = = 2 && size = = 2) then
11:   return true
12: end if
13: if (max_degree = = size) then
14:   return true
15: end if
16: if (max_degree > 0 && vect[0] < max_degree) then
17:   return false
18: end if
19: return true
```

IV.2. Finding All Graphic Sequences

We know that Havel-Hakimi algorithm can be used to obtain a simple graph if the given degree sequence is known to be graphic. Algorithm 3 depicts an algorithm that tests whether the potentially connected degree sequence is graphic or not by constructing an adjacency list representation of the graph of the degree sequence. If the adjacency list is empty then that means the degree sequence is not graphic.

Algorithm 3 GS: An Algorithm for Obtaining a Graphic Sequence

Input: vector<node> wnl. It is potentially connected degree sequence in nondecreasing order of degrees
Output: adjacency_list. It is the representation of the graphic sequence

```
1:  node wn1, wn2
2:  graph_node tgn
3:  vector<node> tnl
4:  while (wnl is not empty) do
5:    wn1 = head of wnl
6:  tgn.label = wn1.label
7:  remove and delete head of wnl
8:  if (wn1.degree > size of wnl) then
9:    delete all elements of adjacency_list
10:   return adjacency_list
11: end if
12: while (wn1.degree) do
13:   wn2 = tail of wnl
14:   delete tail of wnl
15:   decrement degree of wn2 by 1
16:   add wn2 to the tail of list_of_nodes of tgn
17:   decrement degree of wn1 by 1
18:     if (degree of wn2 not equal to 0) then
19:       add wn2 to the tail of tnl
20:     end if
21:   end while
22:   add tgn to the tail of adjacency_list
23:   delete all elements of list_of_nodes of tgn
24:   add tnl at the end of wnl
```

```
25:   delete all elements of tnl
26:   sort wnl on degrees
27: **end while**
28: **return** adjacency_list
```

A *node* is a structure that has a label and its degree. It looks like as given in Fig. 2. The *adjacency_list* is a vector of *graph_node*, which looks like as shown in Fig. 2. An example of a five node connected graph is given in Fig. 3 and its corresponding adjacency list is given in Fig. 4.

```
struct node
{
  string label;
  int degree;
 };

struct graph_node
{
  string label;
  vector<node> list_of_nodes;
 };
```

Figure 2: Structure of node and graph_node

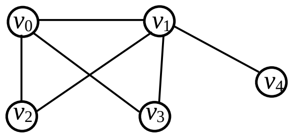

Figure 3: An example of a connected graph

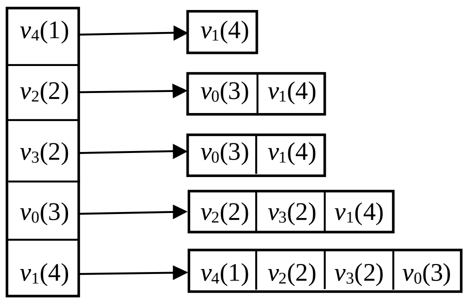

Figure 4: Adjacency list of graph of Fig. 3

It is to be noted that in the adjacency list of Fig.4 4, v_4 is at index 0 of the vector, v_2 at index 1 and so on. This algorithm has an advantage that if the degree sequence input is graphic then it produces an adjacency list of it also whereas in case of Erdos-Gallai algorithm it only tells whether the sequence is graphic or not that too without any guarantee.

For example, Algorithm 1 was used to generate a unique degree sequence (1, 2, 2, 3, 4). This sequence passed the test of Algorithm 2 and then it was applied to Algorithm 3 as input which resulted in the adjacency list of Fig.4 whose graph depiction is in Fig. 3.

V. Generating Random Graphs

To generate random connected graphs, we can use Algorithm 3 with minor modifications. In step number 5, instead of head of the list, generate a random integer number in closed interval [0, n-2] and use it as index into *wnl*. If size of the degree sequence is *n*, then the last index of *wnl* would be (*n* – 1) which must not be the location of *wn*1 as the highest degree term would be at the end.

In step number 7, remove and delete element at index generated in step 5.

Obviously, for small (in length) degree sequence, there will be less options for random graphs. With each iteration of *while* loop of step 4, the node removed would be random and its range would be [0, n – 2] for current shape and size of *wnl*. With each iteration of this *while* loop, the length of *wnl* would decrease by 1.

VI. Experimental Results

All the three algorithms presented in this paper were implemented using C++ language and were compiled using g++ under Ubuntu 20.04 LTS. Two different machines were used for the experimentation

1) Intel Core 4th generation *i*5 – 4690T @2.50 GHz, no hyperthreading => so 4 cores. 8 GB RAM, 6144 KB Cache.

2) Intel Xeon W – 2155, @3.30 GHz, 10 cores, hyperthreading => so 20 cores, 64 GB RAM, 14080 KB Cache.

For testing Algorithm 1, only first machine was used. The program was executed 10 times for each number of nodes from 1 through 20 and the results were recorded. The average of all 10 iterations are given in Table 1.

Table 1: Unique Degree Sequences (Time in seconds).

Nodes	Sequences	Time	Nodes	Sequences	Time
1	1	0	11	167960	0.01821
2	1	0	12	646646	0.05917
3	4	5e-06	13	2496144	0.19969
4	15	9e-06	14	9657700	0.78440
5	56	2e-05	15	37442160	3.01379
6	210	4.4e-05	16	145422675	11.9661
7	792	7.3e-05	17	565722720	48.1107
8	3003	0.00083	18	2203961430	190.614
9	11440	0.00299	19	8597496600	756.847
10	43758	0.00667	20	33578000610	3047.64

Erdos-Gallai algorithm for nondecreasing sequences and Algorithm 3 were executed on both machines. The averages of 10 iterations are given in Table 2 and Table 3. For small number of nodes i.e. up to 8 nodes, performance of both the algorithms are almost same. From 8 nodes onwards Algorithm 3 outperforms Erdos-Gallai algorithm. As the number of nodes further increases, the difference in performance is even more significant.

I have tested these algorithms on 23 different hardware configurations with same operating system and compiler and in each case the trend is same except time in seconds. On all machines from 8 nodes onwards Algorithm 3 outperforms Erdos-Gallai.

Table 2: Time taken by Algorithm 3 and Erdos-Gallai algorithm on machine 1. Time in seconds.

Nodes	Graphic Sequences	Algorithm 3	Erdos-Gallai
2	1	5.2e-05	6e-06
3	2	6.3e-05	1.4e-05
4	6	0.00012	6e-05
5	19	0.000887	0.000782
6	68	0.008733	0.004152
7	236	0.066023	0.065221
8	863	1.23314	1.31442
9	3133	33.6472	36.4811
10	11636	896.286	970.565
11	43306	26500.7	28875.0

Table 3: Time taken by Algorithm 3 and Erdos-Gallai algorithm on machine 2. Time in seconds.

Nodes	Graphic Sequences	Algorithm 3	Erdos-Gallai
2	1	4e-05	4e-06
3	2	4.6e-05	9e-05
4	6	0.000104	5.1e-05
5	19	0.000482	0.000419
6	68	0.006042	0.006024
7	236	0.05979	0.053845
8	863	1.15867	1.19833
9	3133	30.7943	32.419
10	11636	821.43	876.951
11	43306	18127.4	21312.5

VII. Conclusion

Three algorithms related to graphic sequences are presented in this paper. Algorithm 1 generates all unique degree sequences for a given number of vertices. Algorithm 2tests whether the degree sequence input is potentially connected or not. It is better than Erdos-Gallai algorithm in testing the connectivity. Finally, Algorithm 3 presents the case of obtaining graphic sequences as well as their adjacency list representation in far too less time as compared to the Erdos-Gallai algorithm.

The algorithms were tested up to 11 nodes on different hardware to support the claim. The algorithms can be easily used for generating simple random connected graphs. This work will definitely be very helpful in graph isomorphism problems.

References

[1] N. Deo, *Graph Theory with Applications to Engineering and Computer Science* (New Delhi: Prentice-Hall of India, 1974).

[2] F. Harary, *Graph Theory* (New Delhi: Narosa Publishing House, 1987).

[3] S. A. Choudham, *A First Course in Graph Theory* (Madras: MacMillan India, 1987).

[4] P. Erdos and T. Gallai, Graphs with prescribed degrees of vertices, *Mat. Lapok, 11*, 1960, 264-274.

[5] A. Tripathi and H. Tyagi, A simple criterion on degrees of vertices, *Discrete Applied Mathematics, 156*, 2008, 3516-3517.

[6] A. Tripathi, S. Venugopalan, and D. B. West, A short constructive proof of the Erdos-Gallai characterization of graphic lists, *Discrete Mathematics, 310*, 2010, 843-844.

[7] V. Havel, A remark on existence of finite graphs, *Casopis propestovani matematiky (in Czech), 80*, 1955, 477-480.1962, 496-506.

[8] S. L. Hakimi, On the realizability of a set of integers as degrees of the vertices of a graph, *SIAM Journal on Applied Mathematics, 10*, 1962, 496-506.

[9] B. K. Joshi, Digit and Vector classes for graph applications, unpublished.

Author Biography

Dr Brijendra Kumar Joshi is currently working as Professor and Dean (Academics) at Indore Institute of Science and Technology, Indore. He retired as Professor of Electronics & Telecommunication and Computer Engineering at Military College of Telecommunication Engineering, MHOW (MP), India. He obtained BE in Electronics and Telecommunication Engineering from Govt Engineering College, Jabalpur; ME in Computer Science and Engineering from IISc, Bangalore, PhD in Electronics and Telecommunication Engineering from Rani Durgavati University, Jabalpur, and MTech in Digital Communication from MANIT, Bhopal. He has more than 38 years of teaching experience. His research interests are programming languages, compiler design, digital communications, mobile ad-hoc and wireless sensor networks, image processing, software engineering and formal methods. He has number of research publications to his credit. He has supervised 12 Ph D theses and currently supervising 2 research scholars. He has authored two books on Data Structures and Algorithms in C/C++ published by Tata McGraw-Hill, New Delhi.