

## Lecture 8: Degree sequences

September 9, 2021

Kennesaw State University

## 1 The graphic sequence algorithm

### 1.1 What the algorithm does

Let's summarize the algorithm from the previous lecture which tests if a sequence is graphic.

It begins by a cycle of steps that make the sequence shorter and shorter. Given a sequence of degrees, we:

1. **Sort** the sequence, so that it's ordered  $d_1, d_2, \dots, d_n$  with  $d_1 \geq d_2 \geq \dots \geq d_n$ .
2. **Delete** the first term  $d_1$ , and subtract 1 from the next  $d_1$  terms, leaving the sequence  $d_2 - 1, d_3 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n$ .

From here, there are several options:

- This sequence has  $n - 1$  terms. If they are not all in the range 0 to  $n - 2$ , we immediately **give up**: the sequence we got is not graphic.
- If we get to a sequence which we know is graphic, we can **draw a graph** with that degree sequence. (It's enough for the algorithm to work if you do this when the sequence is  $0, 0, \dots, 0$ , in which case we draw the empty graph.)
- Most of the time, we go back and **repeat** this operation.

Once we've simplified the sequence enough, we can draw a graph of that degree sequence. We will draw the graph so it has the degrees we want in order, from left to right.<sup>2</sup> Then we undo all the **sort** and **delete** steps we did above, in reverse order:

1. When we **undelete** a term  $d$ , we add a new vertex to the left of all the vertices, and add  $d$  edges from that vertex to the next  $d$  vertices in order.
2. When we **unsort** the graph, we rearrange our drawing so that the vertices have the degrees we want in the order we wanted them.

The goal: given a graph that realizes the length- $(n - 1)$  degree sequence, the **undelete** and **unsort** operations should give us a graph that realizes the length- $n$  degree sequence.

Then, we repeat these two steps until we get back to the original degree sequence.

---

<sup>1</sup>This document comes from the Math 3322 course webpage: <http://facultyweb.kennesaw.edu/mlavrov/courses/3322-fall-2021.php>

<sup>2</sup>The graph doesn't care how we draw it, but the algorithm assumes we're doing this.

## 1.2 Guarantees about the algorithm

Given a sequence, the algorithm either gives up, or finds a graph with that degree sequence. The only worry is: will the algorithm ever give up, even though we started with a graphic degree sequence?

The algorithm only gives up when the current, shorter degree sequence is obviously not graphic. So another way to restate our worry is: do we ever go from a length- $n$  graphic sequence to a length- $(n - 1)$  sequence which is no longer graphic?

Our **delete** operation was based on imagining a realization of our sequence in which the vertex with degree  $d_1$  is adjacent to the vertices with degrees  $d_2, d_3, \dots, d_{d_1+1}$ . To prove that this is always valid, we want the Havel–Hakimi theorem:

**Theorem 1.1** (Havel–Hakimi). *Let  $G$  be a graph with vertex set  $v_1, v_2, \dots, v_n$  such that  $\deg(v_1) \geq \deg(v_2) \geq \dots \geq \deg(v_n)$ . Let  $d = \deg(v_1)$  (which is also the maximum degree  $\Delta(G)$ ).*

*Then there is a graph  $H$  with the same vertex set  $v_1, v_2, \dots, v_n$  and the same degrees of each vertex (with  $\deg_G(v_i) = \deg_H(v_i)$  for all  $i$ ) in which  $v_1$  is adjacent to  $v_2, v_3, \dots, v_{d+1}$ .*

In other words: if the sequence  $d_1, d_2, \dots, d_n$  is graphic to begin with (if it has a graph  $G$  realizing it) then it also has a graph  $H$  realizing it. The conclusion of Theorem 1.1 is set up exactly so that when we do the **delete** step, the resulting sequence is the degree sequence of  $H - v_1$ . So we always go from a graphic sequence to another graphic sequence.

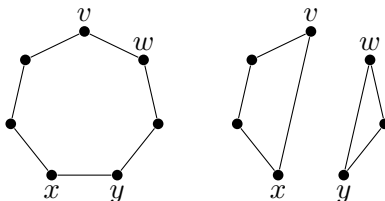
This makes sure that our algorithm always works: if the sequence is graphic, it will realize it, and if not, then it will give up eventually.

## 2 Proof of the Havel–Hakimi theorem

As in the statement of Theorem 1.1, let  $G$  be a graph with vertex set  $v_1, v_2, \dots, v_n$  such that  $\deg(v_1) \geq \deg(v_2) \geq \dots \geq \deg(v_n)$ . Let  $d = \deg(v_1)$ .

For convenience, let  $S = \{v_2, v_3, \dots, v_{d+1}\}$ : our goal is to make these vertices be neighbors of  $v_1$ .

We will proceed by an operation called an **edge swap**, which changes a graph without changing its degree sequence. An edge swap is the following operation: whenever we find vertices  $v, w$  and  $x, y$  such that  $vw$  and  $xy$  are edges, but  $vx$  and  $wy$  are not, we delete the edges  $vw$  and  $xy$  then add new edges  $vx$  and  $wy$ . Here is an example of the “before” and “after” of this operation:



First, here is a lemma that essentially says “we can always use edge swaps to make progress toward what we want”:

**Lemma 2.1.** *If not all vertices of  $S$  are adjacent to  $v_1$ , we can perform an edge swap to increase the number of vertices in  $S$  which are adjacent to  $v_1$ .*

*Proof.* Suppose that  $v_i \in S$  is not adjacent to  $v_1$ . Since  $|S| = d$ , and  $\deg(v_1) = d$ , in order for  $v_1$  to have  $d$  neighbors, it must also be adjacent to some  $v_j \notin S$ .

To do an edge swap that replaces edge  $v_1v_j$  by edge  $v_1v_i$ , we need to find a fourth vertex: a vertex  $w$  such that  $v_i$  is adjacent to  $w$ , but  $v_j$  is not.

Here is where the order of degrees in  $G$  comes in. Since we sorted the degrees in descending order, the vertices in  $S$  have higher degrees than the vertices not in  $S$  (except for  $v_1$ ). In particular,  $\deg(v_i) \geq \deg(v_j)$ . Moreover,  $v_j$  already has one neighbor that  $v_i$  does not have: the vertex  $v_1$ .

Therefore it is impossible for  $v_j$  to be adjacent to every neighbor of  $v_i$ , and also to  $v_1$ : then, we'd have  $\deg(v_i) < \deg(v_j)$ . So we can find a neighbor  $w$  of  $v_i$  such that  $v_j$  is not adjacent to  $w$ .

Now perform the edge swap which replaces  $v_1v_j$  and  $wv_i$  by  $v_1v_i$  and  $wv_j$ . After this switch,  $v_1$  is adjacent to one more vertex of  $S$ .  $\square$

I put Lemma 2.1 first because from here, the proof of Theorem 1.1 is essentially just an application of the lemma, together with the extremal principle. It is a good exercise to try to do this yourself before reading ahead.

*Proof of Havel–Hakimi.* Of all graphs with the same vertex set and vertex degrees as  $G$ , let  $H$  be chosen to maximize the number of neighbors of  $v_1$  in  $S$ .

Then actually, all vertices of  $S$  must be adjacent to  $v_1$ . If not, we could use Lemma 2.1 to increase the number of neighbors of  $v_1$  in  $S$ . But  $H$  was chosen to have the largest possible number of such neighbors, so this can't happen.

So we have the graph we wanted.  $\square$

### 3 More on degree sequences

We only used edge swaps in the proof of the Havel–Hakimi theorem, but they have other applications. To begin with, we can show the following theorem:

**Theorem 3.1.** *If two graphs  $G$  and  $H$  have the same vertex set  $V$ , and  $\deg_G(v) = \deg_H(v)$  for all  $v \in V$ , then we can turn  $G$  into  $H$  by doing edge swaps.*

*Proof.* We prove this by induction on the number of vertices in  $V$  (the vertex set of both  $G$  and  $H$ ).

When  $|V| = 1$ , there is nothing to show: there is only one possible graph on 1 vertex.

Assume this is possible for all pairs of  $(n - 1)$ -vertex graphs, and let  $G$  and  $H$  both have  $n$  vertices. Let  $v$  be a vertex with the highest degree (in both  $G$  and  $H$ ) and let  $S$  be the  $\deg(v)$  vertices with the highest degrees after  $v$ . (This is the same  $S$  as in our proof of the Havel–Hakimi theorem and the edge swap lemma.)

By what we've already shown today, we can do edge swaps on  $G$  to get a graph  $G'$  in which  $v$ 's neighbors are the vertices in  $S$ . We can do the same to  $H$ , getting a graph  $H'$  (and so there is also a sequence of edge swaps that turn  $H'$  into  $H$ , by reversing those edge swaps).

By induction, there is a sequence of edge swaps that turns  $G' - v$  into  $H' - v$  (both  $(n - 1)$ -vertex graphs on the same vertex set). We can perform these edge swaps on  $G'$  instead, and they will work equally well, turning  $G'$  into  $H'$ . Finally, we know that we can turn  $H'$  into  $H$ .

This shows that we can turn  $G$  into  $H$ , and so by induction, this works when  $G$  and  $H$  have any number of vertices.  $\square$

Why is this theorem useful? Well, we know how to answer two possible questions about potential degree sequences...

1. Is this sequence graphic? (Is it the degree sequence of a graph?)
2. How is this sequence graphic? (Find a graph with this degree sequence.)

... but there is a third, harder question we can ask:

3. What is a typical graph with this degree sequence?

That's deliberately vague. However, Theorem 3.1 can, surprisingly, help us answer this question.

What we can do is start with *some* realization of a degree sequence, then do lots of edge swaps at random, getting a random graph with this degree sequence.<sup>3</sup> We can approximately answer questions like "are any graphs with this degree sequence bipartite" or "what is the average diameter of a graph with this degree sequence" by sampling lots of random graphs and doing statistics with the results.

We can even approximately answer questions like "how many different graphs have this degree sequence?" To do this, just sample lots of graphs, then count how many repeats you get. Compare this to how many repeats you'd expect, if there were  $N$  possible graphs total. This should let you estimate the most likely value of  $N$ .

The phrase "different graphs" is a bit vague, though: what exactly makes two graphs "different" in this case? We will see how to answer this in the next lecture, when we talk about graph isomorphism, and say a bit about the different  $r$ -regular graphs there are for some values of  $r$ .

---

<sup>3</sup>Fine print: this does not sample a graph uniformly at random. Essentially, it samples a graph with probability proportional to how many edge swaps are possible to do in it. But that's a known bias we can account for.