Havel-Hakimi Algorithm for Determining Graphic Sequences

Loren Holl

SML Graph Theory - Spring 2024

The goal of this report is to understand the meaning of graphic sequences and to study the Havel-Hakimi Theorem that will allow us to show a sequence is graphic. The proof for the theorem will follow the explanation as well as an implementation of the algorithm at the end.

Let G' be a simple graph with vertices $v_i \in V$, for i = (1, 2, ... , N), and let D be the sequence of degrees of v_i such that deg(v_i) = d_i. The sequence $D = (d_1, d_2, \ldots, d_N)$ is *graphic* if $d_1 \geq d_2 \geq \ \ldots \geq \ d_N$ and there exists a graph $G = G'$. For example, consider the set of degrees $D_1 = (3, 3, 3, 3)$. In Fig 1.1, these degrees correspond to their respective vertices v_1, v_2, v_3, and v_4. Of course, we don't actually know if these vertices will make a real graph (i.e. whether D_1 is *graphic* or not) until we try to graph them out or use a theorem like the Havel-Hakimi Theorem explained later.

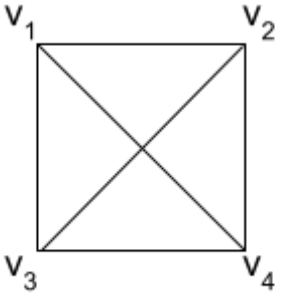

Fig. 1.1

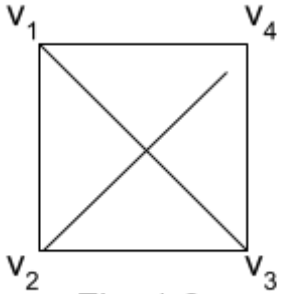

Fig. 1.2

As clearly shown in Fig 1.1, D_1 is a graphic sequence. However, consider the case of $D_2 = (3, 3, 3, 2)$ (Fig 1.2). Unlike D_1, D_2 is not a graphic sequence. This can easily be seen by trying to reconstruct the points into a graph and realizing it is not

possible. One attempt is shown in Fig 1.2, where the edge from v_3 to v_2 has to stop short, making a meaningless graph. This may lead one to find simple rules, like there must be an even number of odd-degree vertices, but we want a way to determine whether a sequence is graphic or not in any situation.

In order to do this, we need to construct an algorithm that inputs a sequence and outputs a boolean value of graphic or not graphic. This algorithm will try and connect all the points together to see if a graph can be constructed. A brief description of this algorithm is given below:

Algorithm 1.3 [Input: list of degrees *D*] (KK, pg. 30):

1.) Set isGraphic boolean to false
2.) Create temporary list D_{temp}, where $D_{temp}[i] = d_i$ for $i = 0 \sim n - 1$
3.) While $d_i \in D_{temp}$ $(i = 0, 1, \ldots, n - 1) > 0$, do the following:
    a.) Set $d = D_{temp}[i]$
    b.) If there are at least d vertices v with deg(v) > 0, then:
        i.) decrease first d degrees by 1
        ii.) set $D_{temp}[i] = 0$
    c.) Else:
        i.) return (the sequence is NOT graphic)
    d.) Increase i by 1;
3.) Set isGraphic = True and return

Essentially, if there is some degree d_i that is negative at the end, then the sequence is not graphic. An implementation of this algorithm in C is given at the end, which inputs a list of degrees D and returns a boolean value (As well as a modified list of degrees D' indirectly).

Now, I would like to introduce the concise Havel-Hakimi Theorem as a continuation of this algorithm.

Havel-Hakimi Algorithm (KK, pg. 35): *D is graphic if and only if D' is graphic.*

Proof: Let's start by assuming that D is graphic. In the algorithm, we start with D and decrease k elements in D by one each time we consider a new degree k. Essentially, we are eliminating one vertex and the number of its degree of edges

from the graph on each iteration. Since we're not altering the degree sequence of our graph the final degree sequence D' will still hold the graphic property.

Now let's assume that D' is graphic. This means that there is some V(G') = $\{u_2, u_3, \ldots, u_n\}$ and $\deg(u_i) = d_i'$ with $D' = (d_2', d_3', \ldots, d_n')$. We know that the algorithm removes the kth degree and subtracts 1 from k other degrees such that $\{d_2 - 1, d_3 - 1, \ldots, d_{d1+1} - 1, d_{d1+2}, \ldots, d_n\}$ in descending order. So if we create a new vertex u_i and join it with the other vertices of degree $d_2 - 1, d_3 - 1, \ldots, d_{d1+1} - 1$, then we can say that $\deg(u_i) = d_i$. This gives us a new degree sequence $D = (d_1, d_2, \ldots, d_n)$ that makes up a real graph, so by definition D is a graphic series.

Code Implementation:

```c
#include <stdio.h>
#include <stdlib.h>

// bubble sort
void sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] < arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

// Havel-Hakimi algorithm
int havelHakimi(int D[], int n) {

    // Sort the degree sequence in decreasing order
    sort(D, n);

    // Create a graph with n vertices
    Graph* graph = createGraph(n);
```

```
    // Construct the graph (Checks if next degree is out of bounds,
meaning the graph doesn't exist)
    for (int i = 0; i < n; i++) {

        sort(D, n);

        for (int j = 0; j < D[i]; j++) {

            if (D[i] >= n) {
                return 0;
            }

            addEdge(graph, i, i + 1 + j);
            addEdge(graph, i + 1 + j, i);
            D[i + j + 1]--;
        }
    }

    // Check if any degree is negative and return 0
    for (int i = 0; i < n; i++) {
        if (D[i] < 0) {
            return 0;
        }
    }

    return 1;
}

int main() {

    int n;

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    int D[n];

    printf("Enter the degree sequence:\n");

    for (int i = 0; i < n; i++) {
```

```
        scanf("%d", &D[i]);
    }

    if (havelHakimi(D, n)) {
        printf("Degree sequence is graphic\n");
    } else {
        printf("Degree sequence is not graphic\n");
    }

    return 0;
}
```