Degree Sequences, Havel–Hakimi Theorem and Algorithm

Hanbyul Lee - Applied Mathematics PhD Student

MATH 6404 - Applied Graph Theory

Final Project Presentation

May 5, 2025

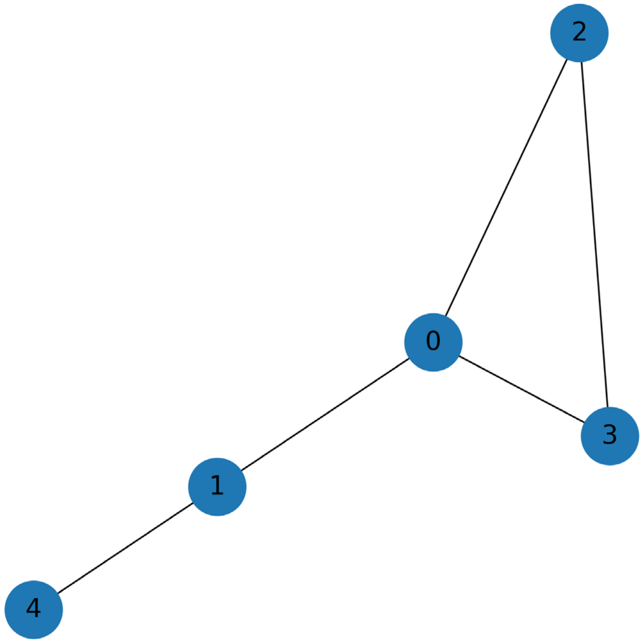
Denver

Outline

- Degree Sequences
- 2-Switch Operation (Simple Graphs)
- Havel–Hakimi Theorem
- Havel–Hakimi Algorithm
- Python Implementation
- Conclusions

Degree Sequences

- List ($d_1 \geq \cdots \geq d_n$) of vertex degrees.
- Example graph realizes degree list (3,2,2,2,1).
- Graphic $\Leftrightarrow$ realizable by a simple graph.



2-Switch Operation (Simple Graphs)

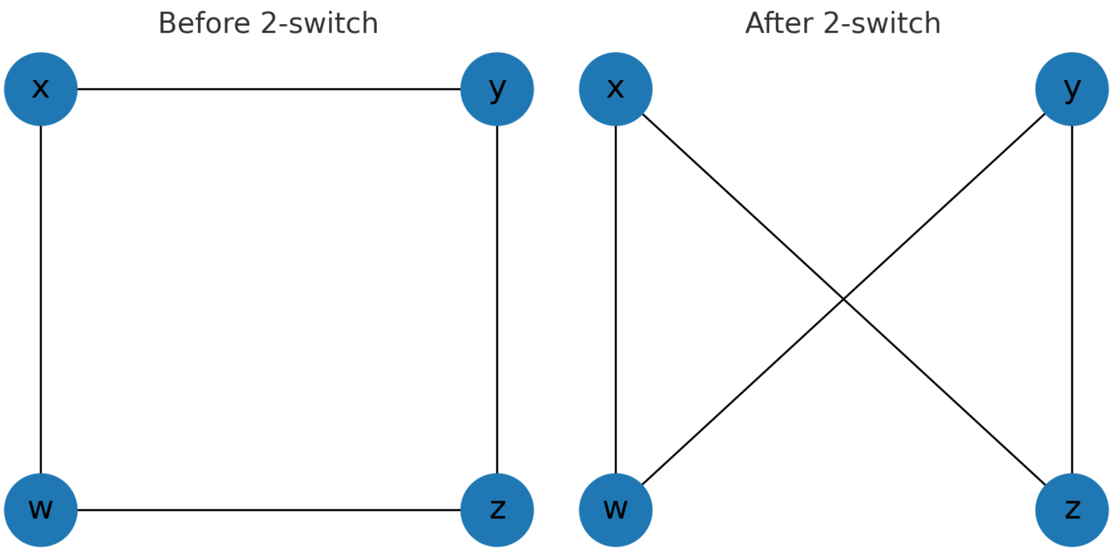
1. Select four distinct vertices x, y, z, w.

2. Edges xy and zw are present, while xz and yw are absent.

3. Replace xy, zw with xz, yw (the 2-switch).

4. Each vertex involved loses one edge and gains one edge, so degrees remain unchanged.

5. Because xz and yw were not previously present and all four vertices are distinct, the resulting graph is still simple (no loops, no multiple edges).



Denver

Havel–Hakimi Theorem [1962]

- A nonnegative integer list d of size $n>1$ is graphic if and only if the list d′, formed by removing the largest element Δ of d and subtracting 1 from the next Δ largest elements, is also graphic.

- d graphic $\Leftrightarrow$ d' (remove Δ, subtract 1 from next Δ) graphic.

Example walkthrough:

- d = (3, 2, 2, 1, 1, 1) (sorted)
- Choose $\Delta = 3$ (degree of vertex w).
- Delete the first ‘3’: $\rightarrow$ (2, 2, 1, 1, 1)
- Subtract 1 from the next Δ=3 entries: (2, 2, 1) $\rightarrow$ (1, 1, 0)
- Combine and re-sort: d′ = (1, 1, 1, 1, 0).
- If d′ is graphic, re-attaching w to its 3 neighbor's reconstructs a graph for d.

Proof Sketch (Necessity Condition)

Proof Sketch – Necessity ($d \Rightarrow d'$)

1. Choose a maximal-degree vertex

Let G realize d and pick $w \in V(G)$ with $\deg_G(w) = \Delta = \max d$.

2. Define the target neighbour set

S = the Δ vertices ($\neq w$) having the largest degrees in G.

3. Correct w's adjacency via 2-switches

If $w \nsim x \in S$ or $w \sim \alpha \notin S$, perform

$(\alpha w,\ x\beta) \to (xw,\ \alpha\beta)$.

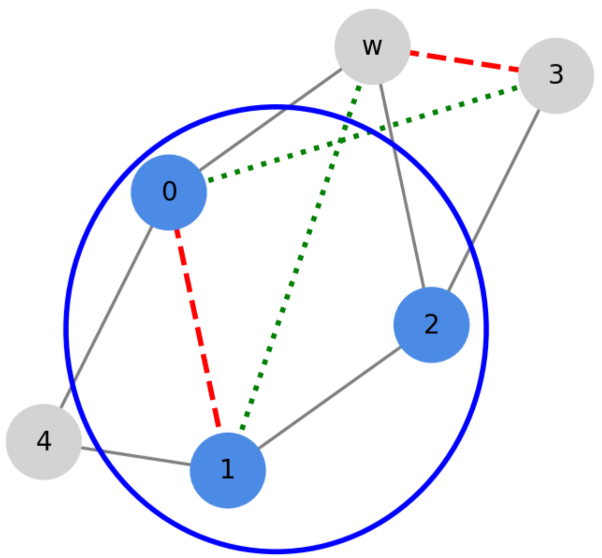
Each switch raises $|N_G(w) \cap S|$; after $\leq \Delta$ steps we have $N_G(w) = S$.

4. Delete w

Removing w decreases each $v \in S$ by 1 and leaves others unchanged,

giving $d' = (d_2 - 1,\ \ldots, d_{\Delta+1} - 1, d_{\Delta+2},\ \ldots, d_n)$, which is graphic. ■

Necessity 2-switch (d = 3,3,3,3,2,2)



d=(3,3,3,3,2,2) ⟶ d′=(2,2,2,2,2)

Proof Sketch (Sufficiency Condition)

Proof Sketch – Sufficiency ($d' \Rightarrow d$)

1. Assume d' is graphic

Let G' realize d' on vertices $v_1, \ldots, v_{n-1}$.

2. Identify the affected vertices

T = the Δ vertices whose degrees were reduced by 1

when forming d' from d (the next Δ largest entries).

3. Add a new vertex w

Insert w into G' as a fresh vertex.

4. Connect w to every vertex in T

Introduce exactly Δ new edges $w \sim T$.

5. Verify the degrees

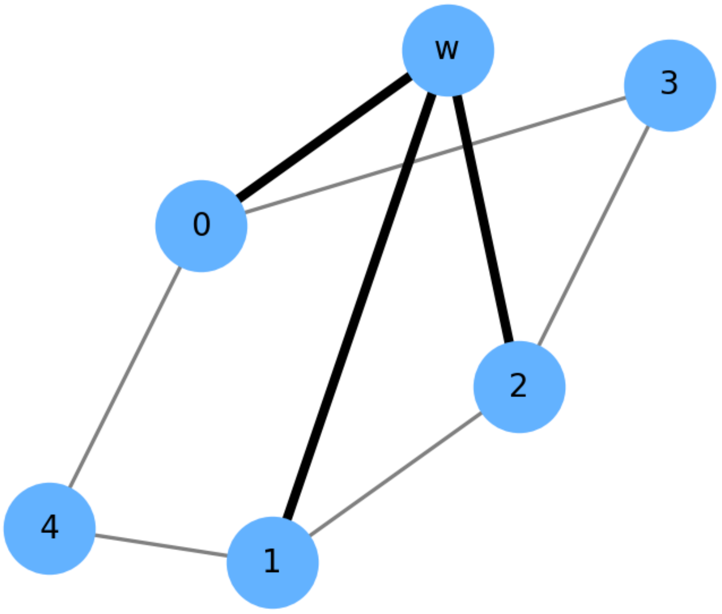
$\deg_G(w) = \Delta$; each $v \in T$ regains the lost edge,

and all other vertices keep their degrees.

6. Conclusion

The resulting graph G realizes d, so d is graphic. ∎

Sufficiency: Reconstructed Graph (d = 3,3,3,3,2,2)



d′=(2,2,2,2,2) ⟶ d=(3,3,3,3,2,2)

Havel–Hakimi Algorithm

Algorithm ⇔ Proof Directions

Necessity $d \Rightarrow d'$

- One loop step removes Δ and decrements next Δ entries.
- Mirrors the construction proving d graphic ⇒ d′ graphic.

Necessity fails

Early exits: Δ greater than remaining list length or negative degree.
⇒ degree list is not graphic.

Sufficiency $d' \Rightarrow d$

When loop terminates with all zeros, replay stored decrement history to reconnect removed vertices, giving a graph for the original d.

Denver

Python Implementation

```
def havel_hakimi(seq: list[int]) -> bool:
    seq = sorted(seq, reverse=True)          # initial order
    while seq:
        seq = [d for d in seq if d]          # drop isolated vertices (history saved)
        if not seq:                          # ← Sufficiency terminus (all zeros)
            return True                      #    ⇒ original d is graphic
        Δ = seq.pop(0)                       # largest degree
        if Δ > len(seq):                     # ↱ Necessity breach: too few vertices
            return False                     # ↳   list not graphic
        for i in range(Δ):                   # ↱ Necessity step: build reduced d′
            seq[i] -= 1                      # |  decrement next Δ entries
            if seq[i] < 0:                   # |  negative ⇒ breach
                return False                 # ↳
```

Tested in Python ≥ 3.9

Denver

Havel–Hakimi Algorithm

- **What can we do with it?**

✓ Decide in $O(n^2)$ time whether a list is graphic.

✓ Build a concrete graph by keeping track of decremented vertices.

✓ Quickly generate synthetic networks with prescribed degrees.

✓ Serve as starting point for random degree-preserving rewiring.

Conclusions

Havel–Hakimi Theorem

A degree list d is graphic $\Leftrightarrow$ the reduced list d' (remove largest Δ and subtract 1 from the next Δ entries) is graphic.

Havel–Hakimi Algorithm

- Iterative realisation / graphic test in $O(n^2)$ time ($O(n \log n)$ with a max-heap).
- Constructs an explicit graph and seeds synthetic-network generation.

Theorem 2 (Fulkerson–Hoffman–McAndrew 1965; Berge 1970)

Let G and H be graphs on the same vertex set V. Then G can be transformed into H by a finite sequence of 2-switches iff $d_G(v) = d_H(v)$ for every $v \in V$.

Reference: https://math.ucdenver.edu/~sborgwardt/wiki/index.php/Degree_Sequence

Denver

Q & A

Contact: hanbyul.lee@ucdenver.edu

Thank you for listening!

CU Denver