Longest Path

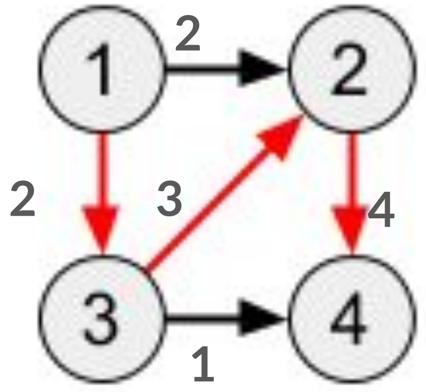
Nate Baker, Ellona Macmillan, Will Keppel, Ben Glass

Exact Solution

Ben Glass

Problem description and Practical Applications

- Given any weighted, directed graph, find the longest (heaviest) path in the graph.
- The path must be legal:
  - Each vertex connects to the last
  - No repeated vertices
- Practical applications:
  - Find the longest possible route on a map.
  - Find the way to use the most amount of a resource
  - Find the longest set of connections in a system:
    - DNA sequencing
    - Public transportation



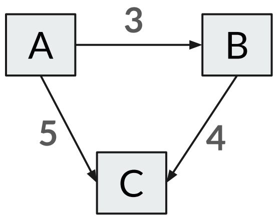
Details on Inputs/Outputs

Input:

- A graph
- First line: The number of vertices and edges in the graph (integer: number_V, integer: number_E)
- Every line after: An edge in the graph, (character: u, character: v, integer: w)

Example:

```
3 3
a b 3
b c 4
a c 5
```

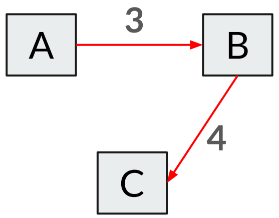


Output:

- First line: The integer weight of the largest path.
- Second Line: The largest path as a sequence of space-separated characters.

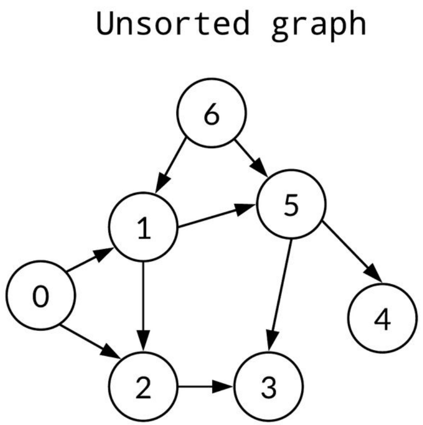
Example:

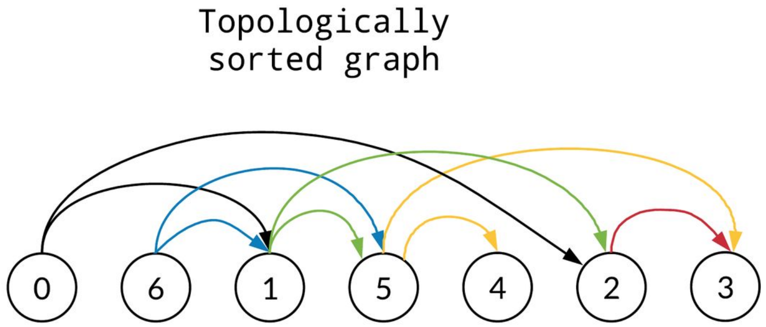
```
7
a b c
```



A restriction on the problem that makes it possible to compute the answer in polynomial time

- Restrict the input to weighted DAGs only
- A cycle-free graph can be sorted in order (topologically) O(V+E)
- For each vertex, track its current distance and predecessor O(V)
- Find the longest path to each vertex by relaxing tense edges (opposite of SSSP tense) O(V+E)
  - If dist(v) < dist(u) + weight from u→v, then it is tense
- Find the vertex with the longest distance, and its path backwards until it cannot go backwards O(V+E)
- Total: O(V+E)





Reduction from a known NP-Hard problem

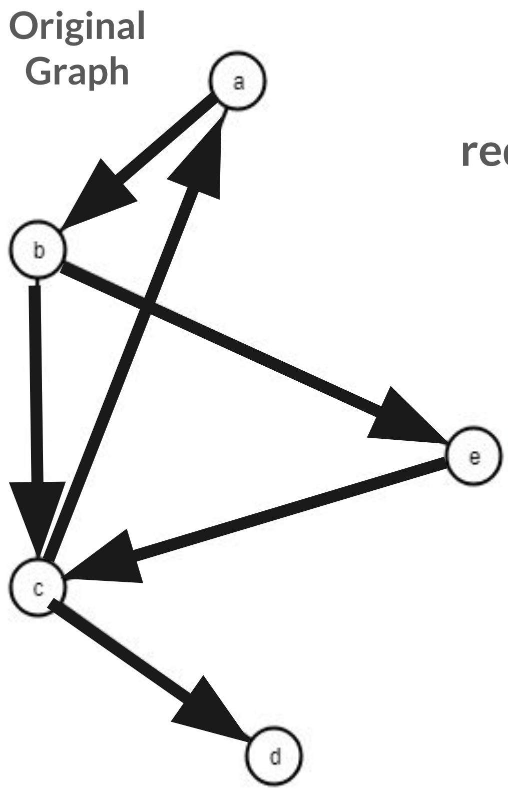
Reduce the **Directed Hamiltonian Path** problem to The **Longest Path** problem.

- Goal: Given a directed graph, determine if there is a hamiltonian path within it.
- Polynomial-time reduction:
  - Take in the graph.
  - Set every edge weight to 1.   O(E)
  - Track the number of vertices in the graph.
  - Feed the graph into the longest path algorithm.
  - If longest path returns a path with a weight **equal to V-1**, then there is a hamiltonian path within the graph.
- This works because longest path and hamiltonian path both only allow simple paths.
- Total time for the reduction step is O(E)

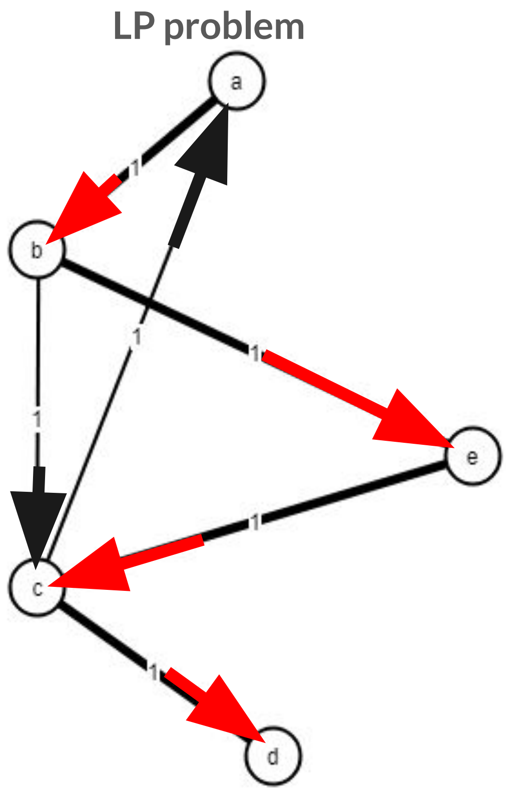
Correctness argument:

- If G has a Hamiltonian path:
  - A hamiltonian path has exactly n-1 edges, therefore, the weight = n-1.
  - This is the maximum possible weight of any simple path in G.
  - The Longest Path algorithm must return a path of weight n -1
- If the algorithm returns a path of weight n-1:
  - Weight = number of edges (all weighted 1).
  - Having n-1 edges means it visits all n vertices and is by definition a Hamiltonian path.
  - Therefore, the original graph has a hamiltonian path.
- The Longest Path algorithm outputs a Hamiltonian path **iff** one exists.
- Since, the Hamiltonian path problem is known to be NP-Hard, computing the longest simple path in a directed graph is NP-Hard.

Reduction from a known NP-Hard problem visual



Main reduction step:



V - 1 == 4
AND
LP == 4
Therefore
Hamiltonian
Path

Approach for solving

High level algorithm steps:

1. Read in the graph as an adjacency list, maintain a list of every edge.
2. Use itertools to get a permutation of all edges.
3. Check every single combination of edges for legality:
    a. The paths connect and are in sequential order (check by remembering the endpoint of the previous edge)
    b. The path does not repeat vertices (maintain a set of visited vertices and reject any path with repeats)
4. If the path is legal and bigger than the current maximum length path, it becomes the new max.
5. Traverse the resulting longest path and output its contents.

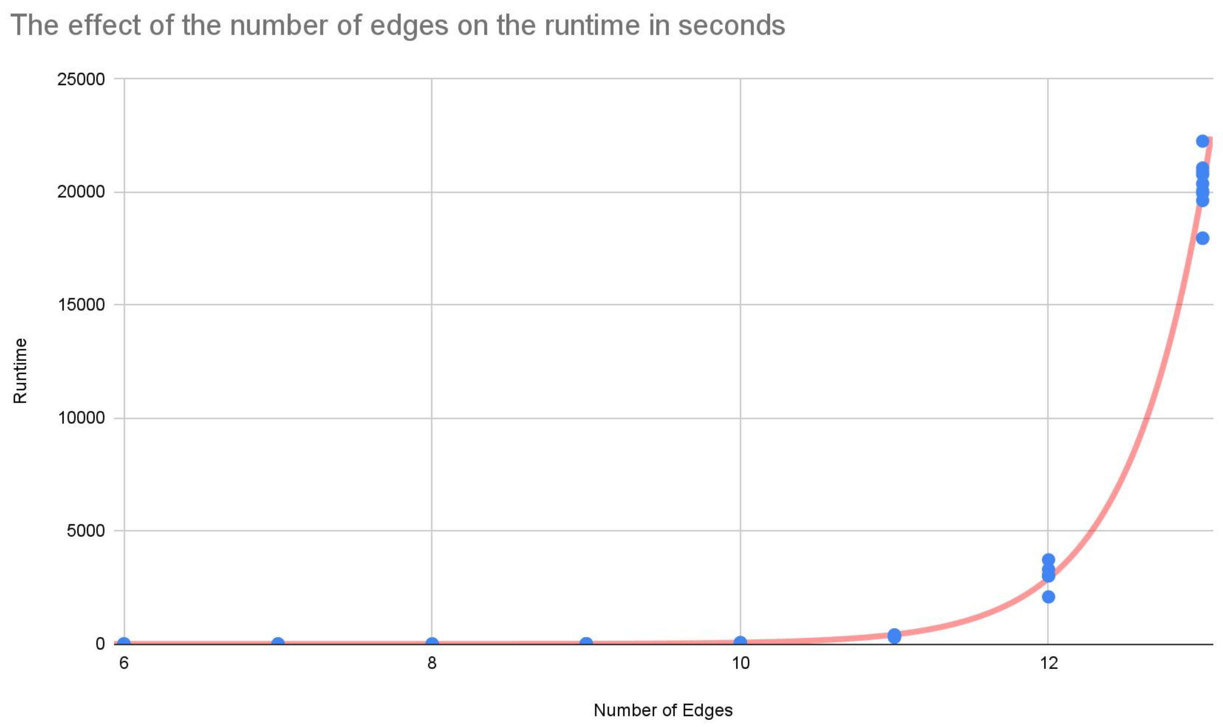
Analytical runtime

- Build the graph from input (graph size dependent on number of edges): O(E).
- Permute every edge combination of every length: O(E!)
  - Check each edge combination for legality (could be up to E long): O(E)
  - Check if the combination is the biggest so far: O(1)
- Traverse the resulting path (could be up to E long): O(E)
- Result:
  - The permutation and legality check combine for O(E * E!)
  - The permutation and legality check dominate the runtime.
  - Total runtime: O(2E + E * E!).
  - The algorithm's runtime simplifies to O(E * E!)

Can changing the input order make it run faster?

- Since every run permutes every possible edge, it is very challenging to optimize the algorithm based on ordering.
- Since it always looks at every permutation before deciding the longest path, changing the input order would not make it run faster.
- The algorithm would have to be fundamentally changed for the input order to affect its runtime.

Runtime plot



Approximation 1

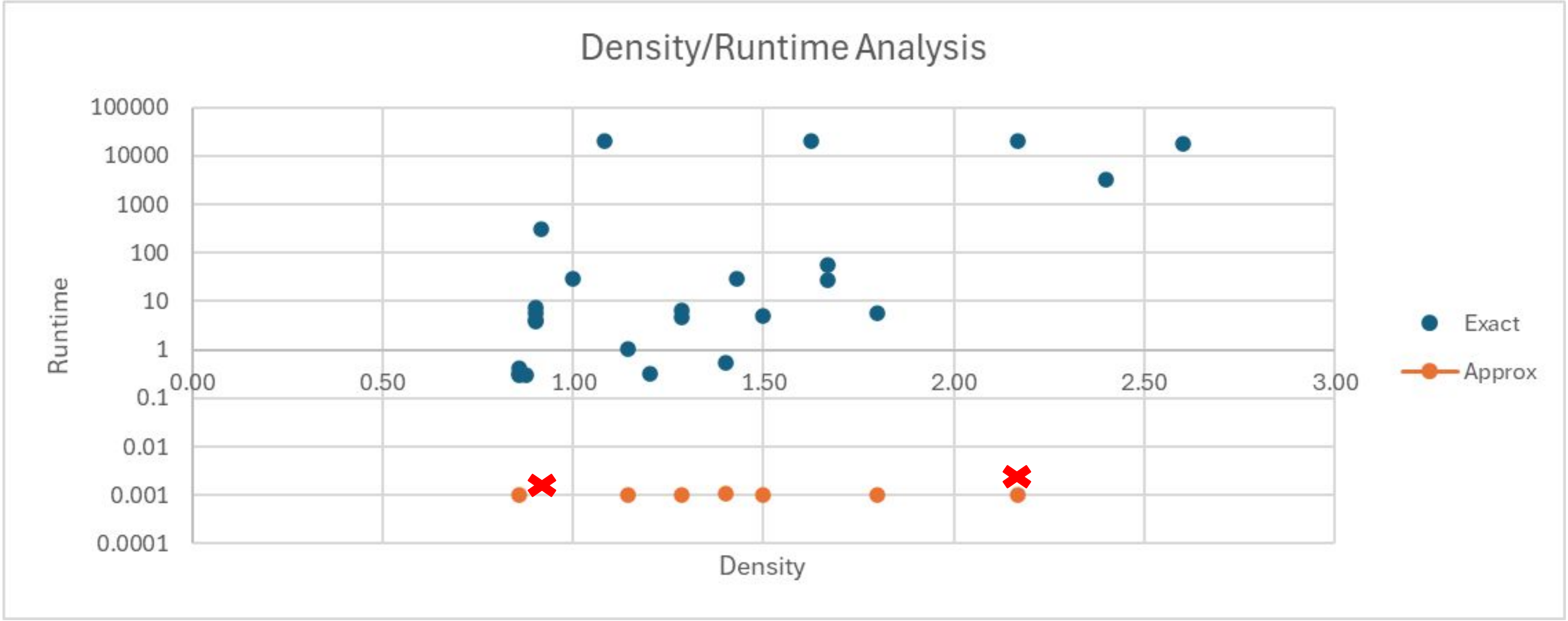
Will Keppel

Basic Implementation -> $O(V^2)$ + some nonsense

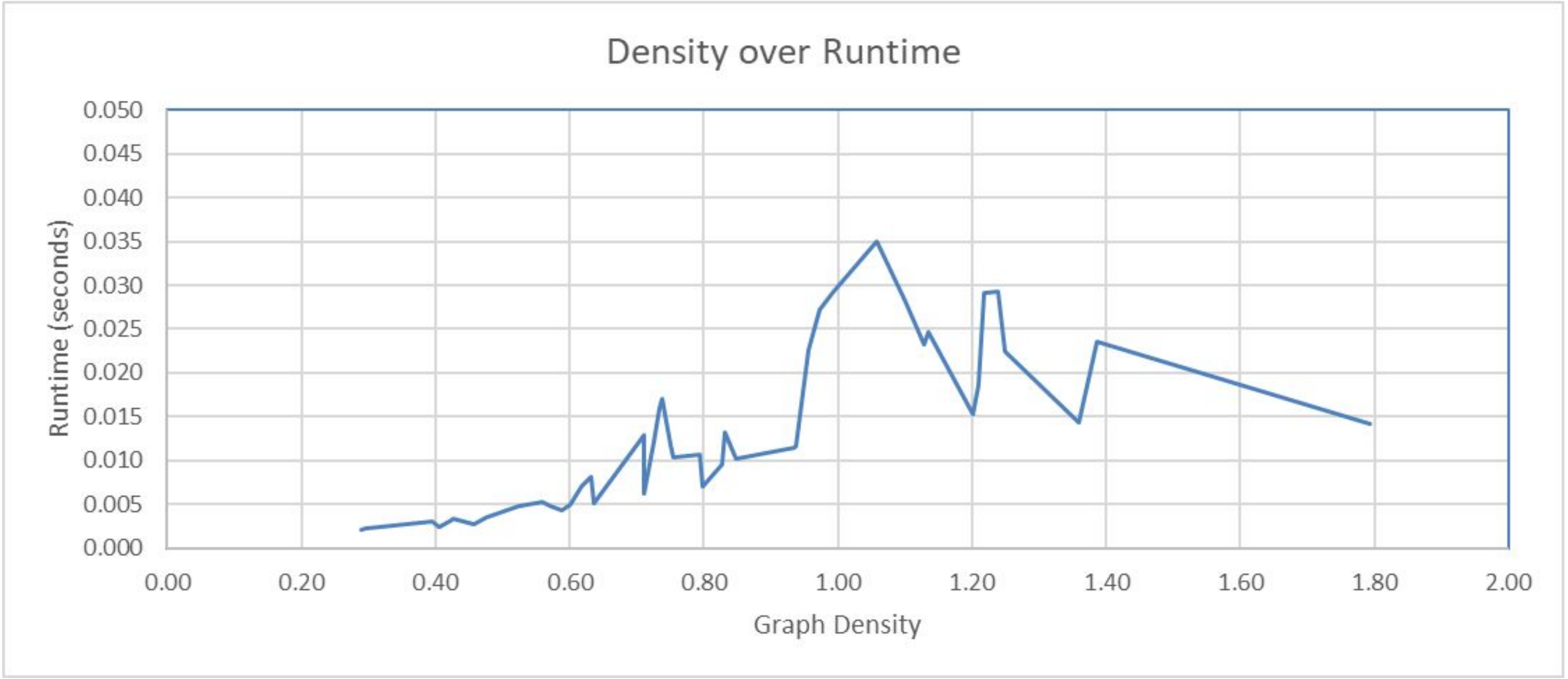
- Create Graph ***O(V + E)***
- Sort Edges by weight ***O(ElogE)***
- Make Max Spanning Tree using Modified Boruvka's ***O(EV)***
    - Add the max weight that remains acyclic
    - Could be really bad for dense graphs (O(V^3) worst case of every node connected(but almost never kinda like quicksort))
- DFS with Max Weight Heuristic (Greedy) ***$O(V^2)$***
    - For every node, check all of its paths
    - Keep best overall
        - random() < 0.5 Tie breaker
    - If time runs too long, returns None, breaking the dfs stack

Graphs

✖ Denotes Wrong Answer



Graphs


Density over Runtime
Runtime (seconds)
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0.00
0.20
0.40
0.60
0.80
1.00
1.20
1.40
1.60
1.80
2.00
Graph Density

My Approximation

Lower Bounds - The edge with the highest weight

+ The longest path must include the heaviest edge or something worse

Upper Bounds - The sum of the heaviest V-1 edges

+ The longest path super path should include all nodes, and all heaviest edges

Approximation 2

Nate Baker

Reduction and Bounds

Ellona Macmillan