Min Vertex Cover Approximate Solution

By: Mason and Chingiz

Our Approximate solution

- We decided on using a greedy approach in our approximate solution
- We sort the graphs by the number of edges incident to each vertex
- The rest of our code follows the same structure as our exact solution
    - We loop through a subset of vertices removing all edges until the graph is empty
    - We only check a single subset of vertices (sorted by edge #)

```python
def vertex_cover(graph):
    min_cover = []

    sort_graph = sorted(graph, key=lambda key: len(graph[key]), reverse=True)

    for v in sort_graph:
        # print(graph)
        if v not in graph:
            continue
        for u in graph[v]:
            graph[u].remove(v)
            if graph[u]:
                graph.pop(u)
        graph.pop(v)
        min_cover.append(v)
        if not graph:
            break
```

Our Approximate Solution Runtime

- Our solution's runtime is $O(n^2)$
    - In the worst case our graph is complete
    - When the graph is complete we need to remove up to n edges from n vertices

```python
def vertex_cover(graph):
    min_cover = []

    sort_graph = sorted(graph, key=lambda key: len(graph[key]), reverse=True)

    for v in sort_graph:
        # print(graph)
        if v not in graph:
            continue
        for u in graph[v]:
            graph[u].remove(v)
            if graph[u]:
                graph.pop(u)
        graph.pop(v)
        min_cover.append(v)
        if not graph:
            break
```

Analyzing the run-time

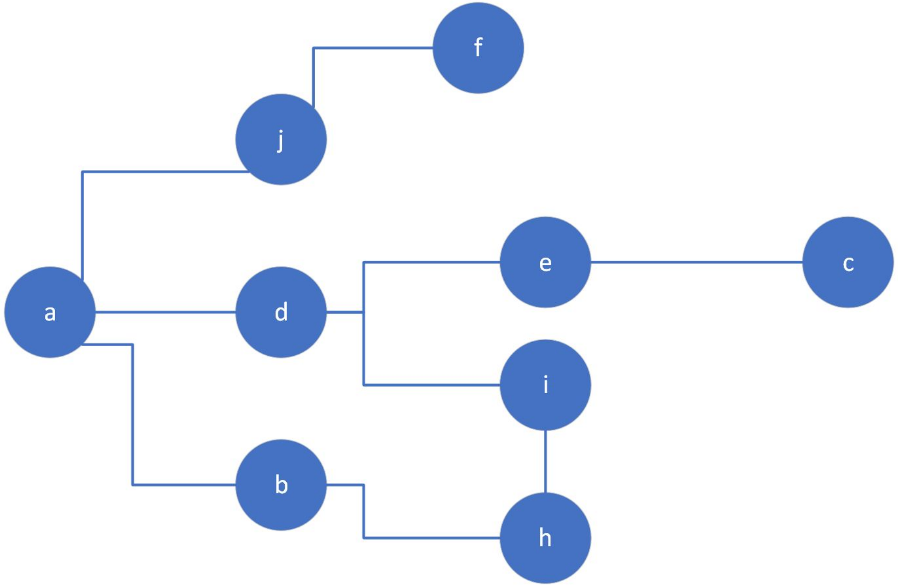
- The most amount of time was spent on making the graph and using the set add and setDefault
- And we can see the difference with exact solution where we had to call minvertexcover 12 million times
- Additionally the runtime here is for the largest graph whereas the exact solution was for the smallest one

```
         2715 function calls in 0.001 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.001    0.001 {built-in method builtins.exec}
        1    0.000    0.000    0.001    0.001 cs412_minvertexcover_approx.py:1(<module>)
        1    0.000    0.000    0.001    0.001 cs412_minvertexcover_approx.py:8(main)
        1    0.000    0.000    0.000    0.000 cs412_minvertexcover_approx.py:21(vertex_cover)
      326    0.000    0.000    0.000    0.000 {built-in method builtins.input}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.print}
      650    0.000    0.000    0.000    0.000 {method 'add' of 'set' objects}
      650    0.000    0.000    0.000    0.000 {method 'setdefault' of 'dict' objects}
      325    0.000    0.000    0.000    0.000 {method 'split' of 'str' objects}
      351    0.000    0.000    0.000    0.000 {built-in method builtins.len}
      325    0.000    0.000    0.000    0.000 {method 'remove' of 'set' objects}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.sorted}
       26    0.000    0.000    0.000    0.000 cs412_minvertexcover_approx.py:24(<lambda>)
        1    0.000    0.000    0.000    0.000 cp1252.py:22(decode)
        1    0.000    0.000    0.000    0.000 {built-in method _codecs.charmap_decode}
        1    0.000    0.000    0.000    0.000 <frozen codecs>:281(getstate)
       26    0.000    0.000    0.000    0.000 {method 'pop' of 'dict' objects}
       25    0.000    0.000    0.000    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'join' of 'str' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

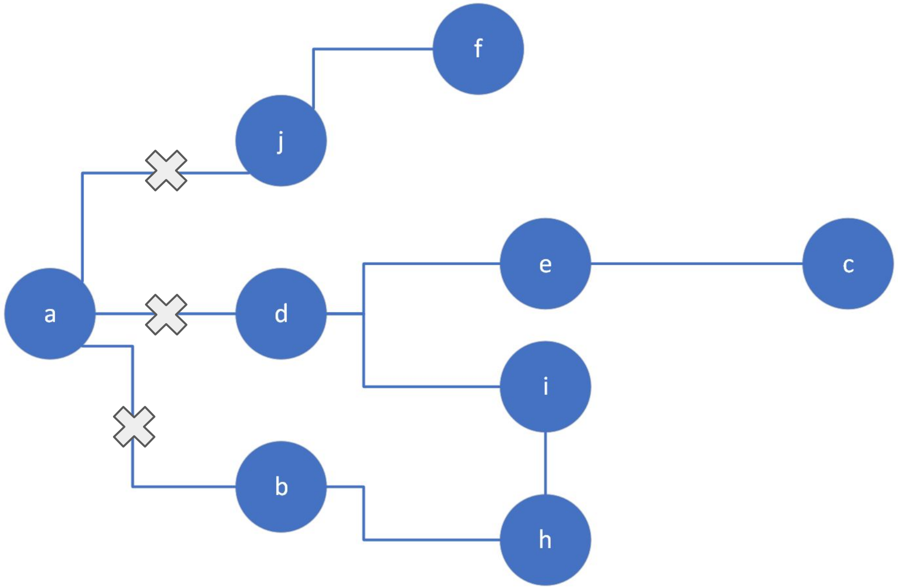
Non-Optimal Solution



Approx Output: a d b j h e

Exact Output: a j d e h

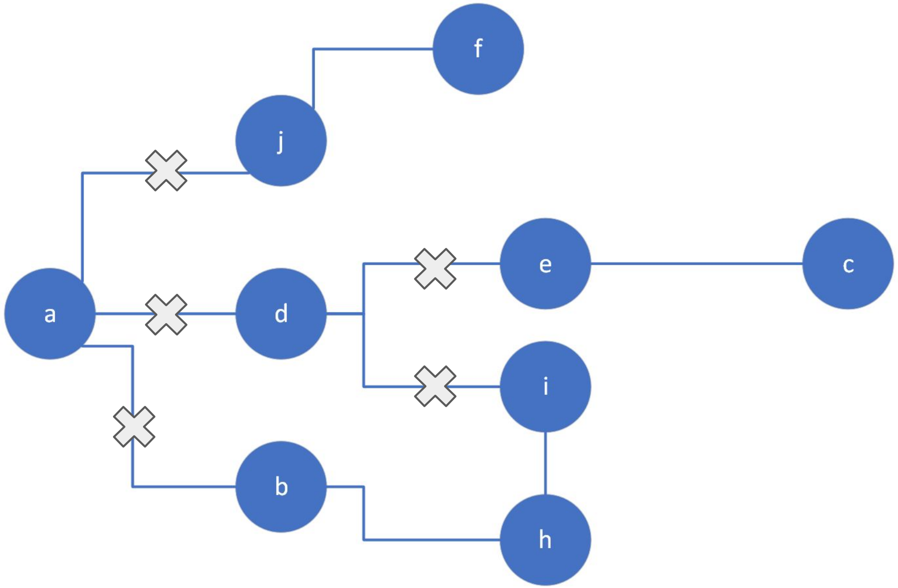
Non-Optimal Solution



Approx Output: **a** d b j h e

Exact Output: a j d e h

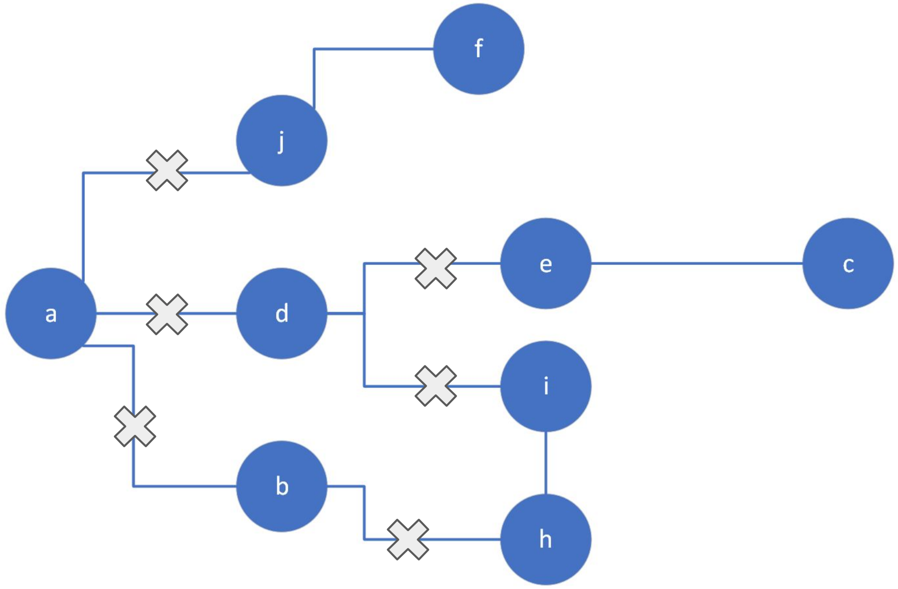
Non-Optimal Solution



Approx Output: a **d** b j h e

Exact Output: a j d e h

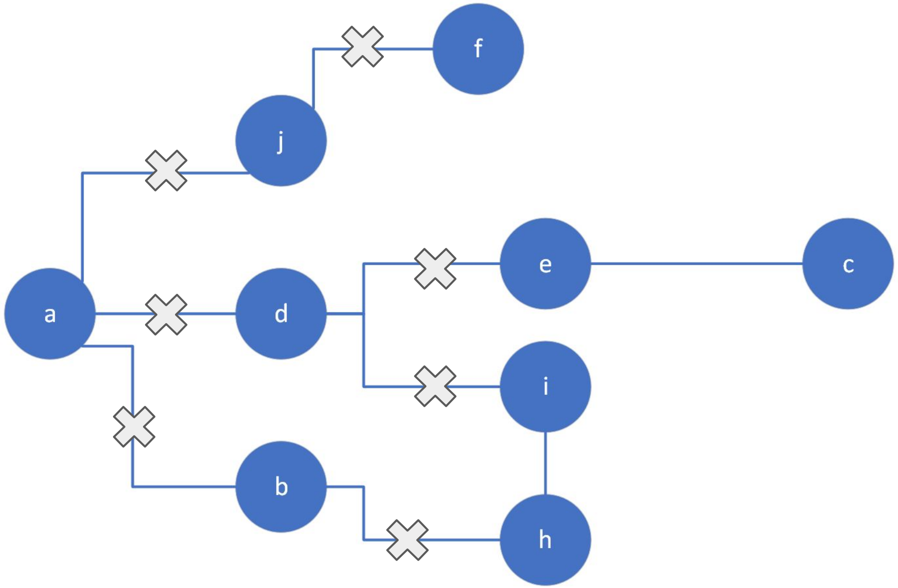
Non-Optimal Solution



Approx Output: a d **b** j h e

Exact Output: a j d e h

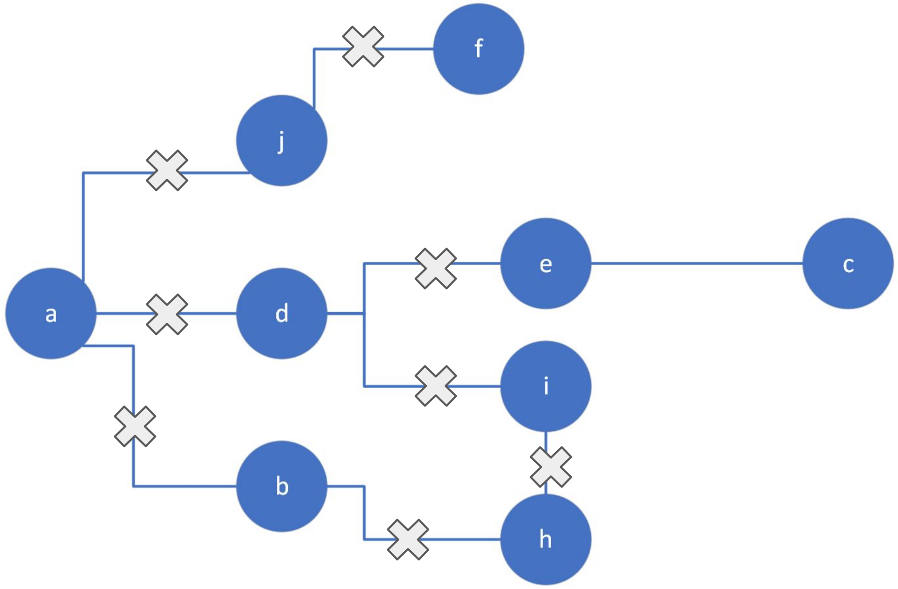
Non-Optimal Solution



Approx Output: a d b **j** h e

Exact Output: a j d e h

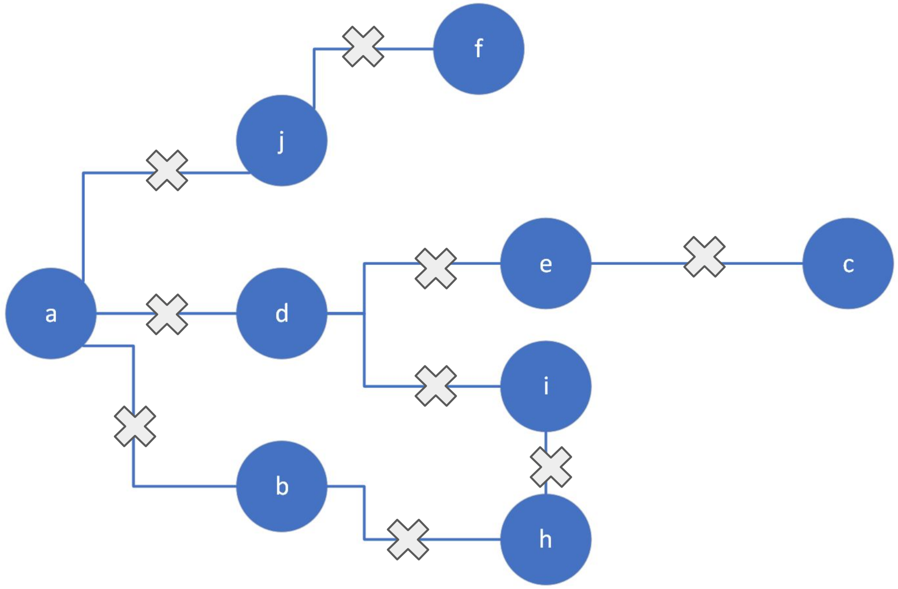
Non-Optimal Solution



Approx Output: a d b j **h** e

Exact Output: a j d e h

Non-Optimal Solution

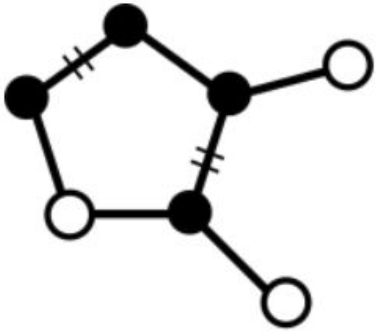


Approx Output: a d b j h **e**

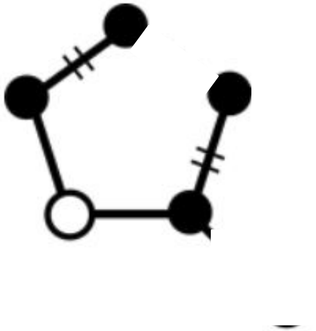
Exact Output: a j d e h

Lower Bound Analysis 1/2

- A lower bound for the minimum vertex cover is given by a maximal matching.
  - A matching is a subgraph in which no 2 edges share a common vertex
  - The maximal matching isn't a subset of any other matching of G (no additional edges can be added).
  - There must be at least one vertex in the vertex cover for each edge in the maximal matching
  - Therefore, our lower bound is the number of edges in our maximal matching



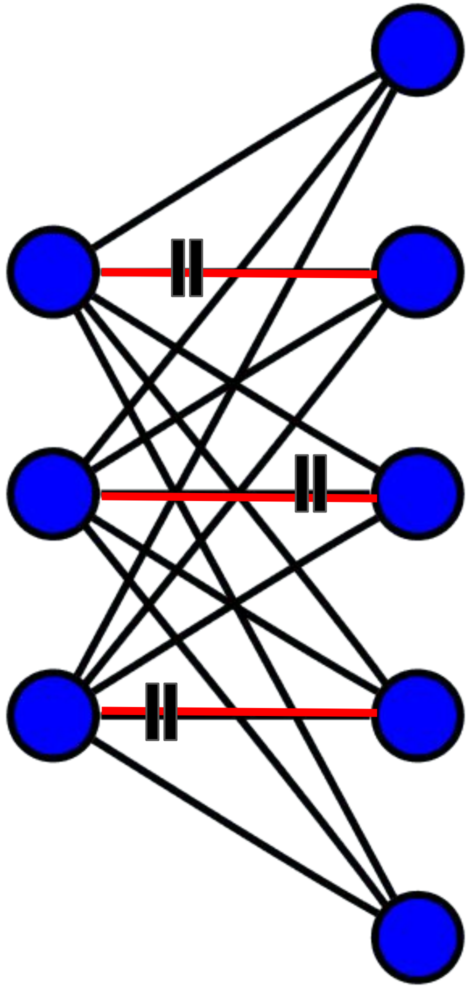
maximal



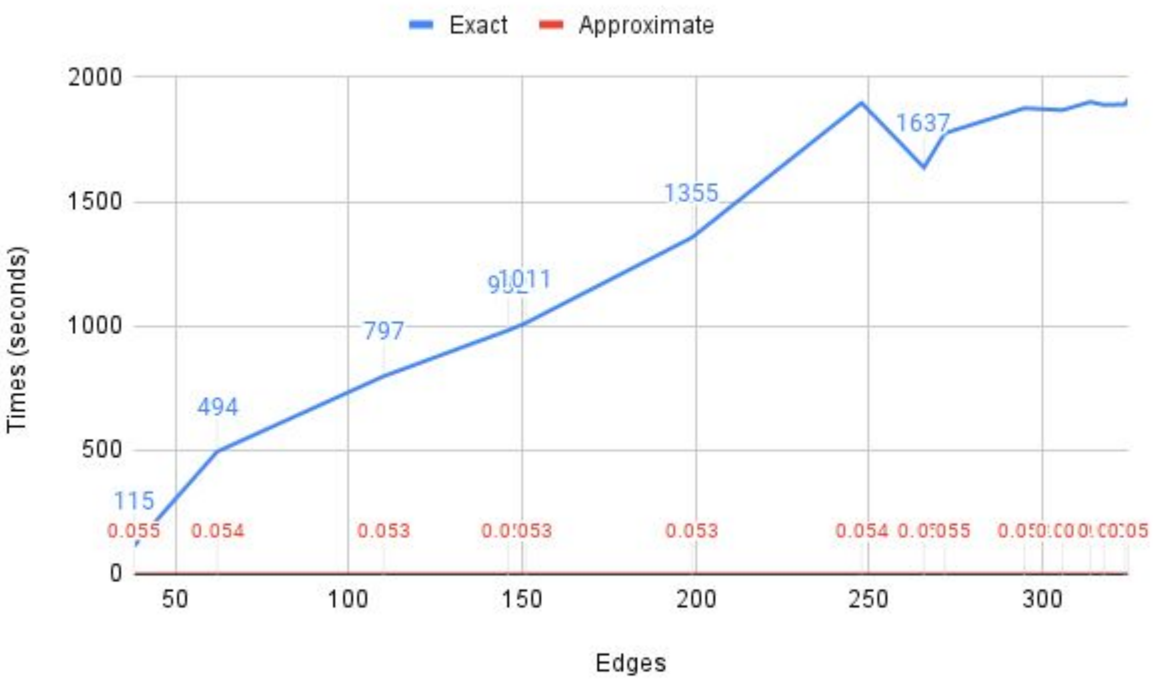
maximal

Lower Bound Analysis 2/2

- Consider a complete bipartite graph where every vertex of the first set is connected to every vertex of the second set
  - Our approximate solution outputs vertices from a maximal matching
  - The optimal vertex cover only needs vertices, from one side of the partition.



Runtime with exact and approximate solutions



Sources

https://ocw.mit.edu/courses/18-433-combinatorial-optimization-fall-2003/8d2afe77ec0c0ac4a22f5e203f65dd17_l2122.pdf

https://www.math.cmu.edu/~mradclif/teaching/301F15/Matchings.pdf

https://depth-first.com/articles/2019/04/02/the-maximum-matching-problem/