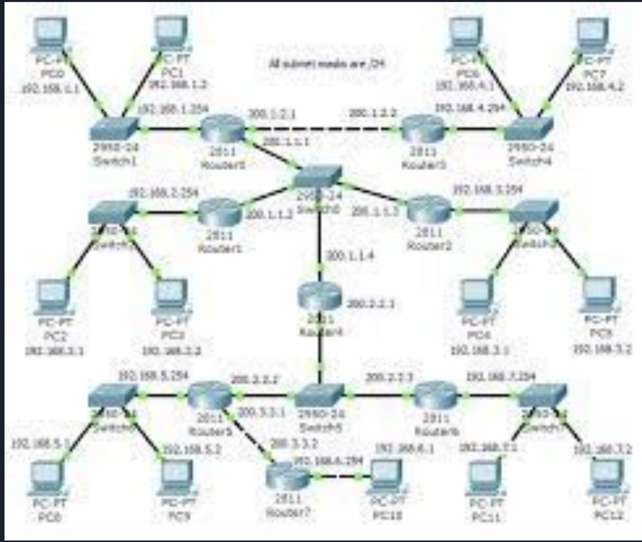
Parallelization of Pathfinding Graph Algorithms

By Kyle Manke and Devin Rosenthal

Outline

1. Network Routing
2. Planned Algorithms
3. Parallel Approaches
4. Code Overview
5. Datasets
6. Results
7. Continuing On

Network Routing



- Routers maintain a routing table in order to send packets in efficient paths.
- Network graphs are useful as they allow for straightforward graph algorithms to perform the routing.

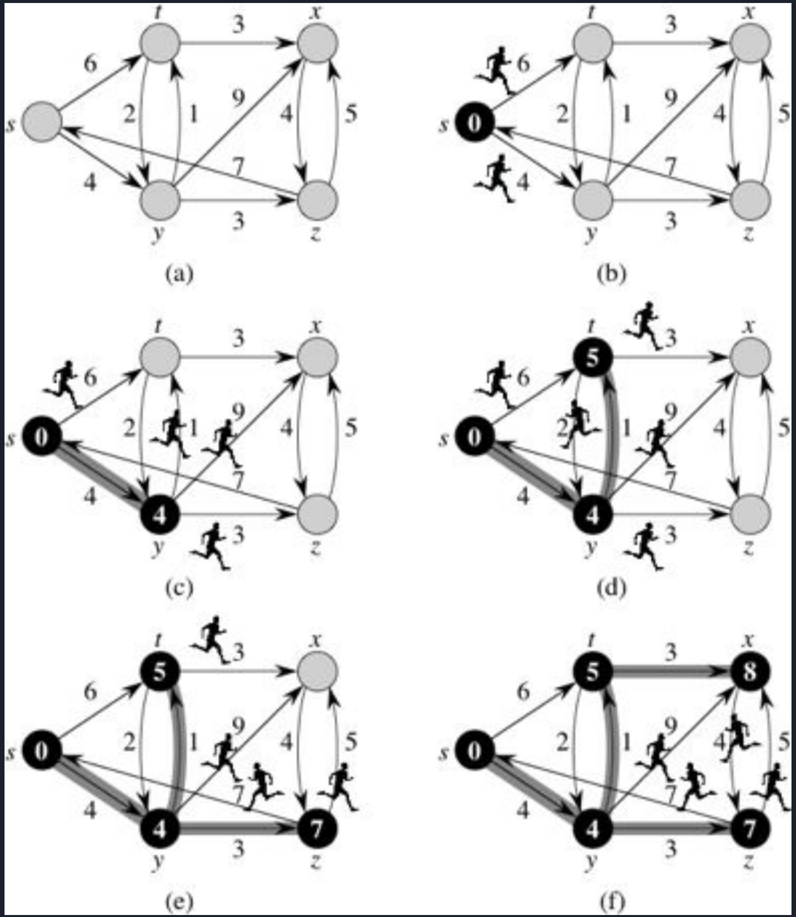
Our Goals

1. Implement an efficient parallelized version of Dijkstra’s and Bellman-Ford with OpenMP.

2. Evaluate the speedup and scalability of the algorithms relative to each other and their serial versions.

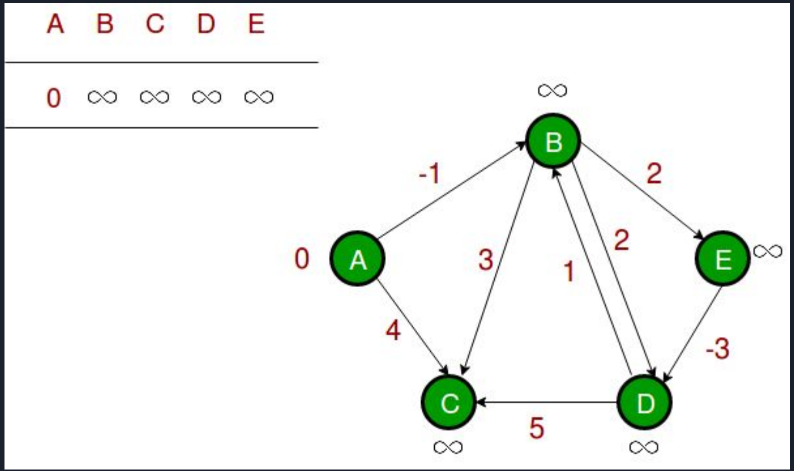
Dijkstra’s

- Main Idea: Very similar to BFS but uses a priority queue vs. a standard queue
- Allows for termination upon visiting the destination
- O((m+n)log(n)) for n nodes, m vertices



Bellman-Ford

- Main Idea: Use dynamic programming to build the shortest path in a bottom up manner
- May terminate early after no distances change
- O(nm) for n nodes, m edges



Parallel Approach: Dijkstra's

Simple Approach: Relies on previous results, but could parallelize the loops within each iteration.

- Con: Creates a high overhead with little benefit on sparse graphs

Advanced Approach: Each processor handles a subset S of graph G to find eventually map the path through each cluster

Parallel Approach: Bellman-Ford

Algorithm 1: BELLMAN-FORD

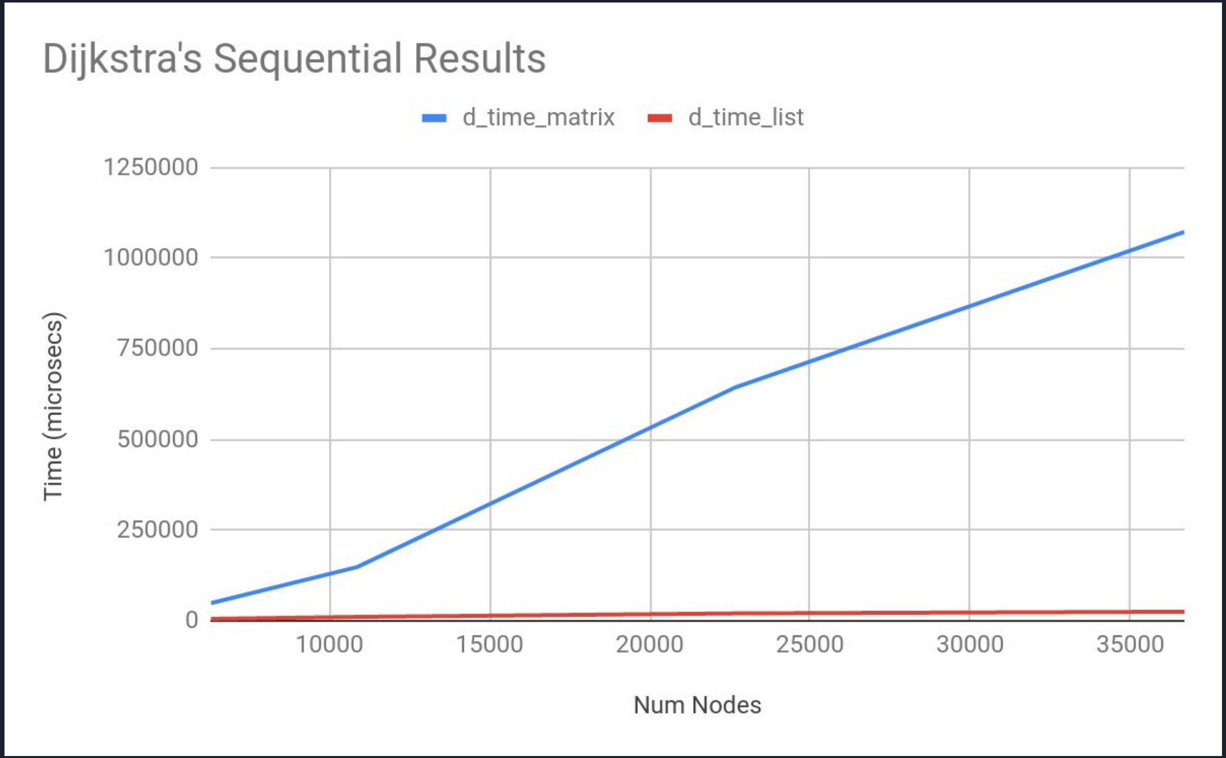
Input : Graph G =(V,E), source vertex
Output : shortest distance & predecessor arrays

```
Initialize distance array to ∞
Initialize predecessor array to −1
distance[source] := 0
predecessor[source] := -1
for i ← 1 to |V-1| do
    for each edge (u, v) with weight w in E do
        if distance[u] + w < distance[v] then
            distance[v] := distance[u] + w
            predecessor[v] := u
        end
    end
end
return distance, predecessor
```

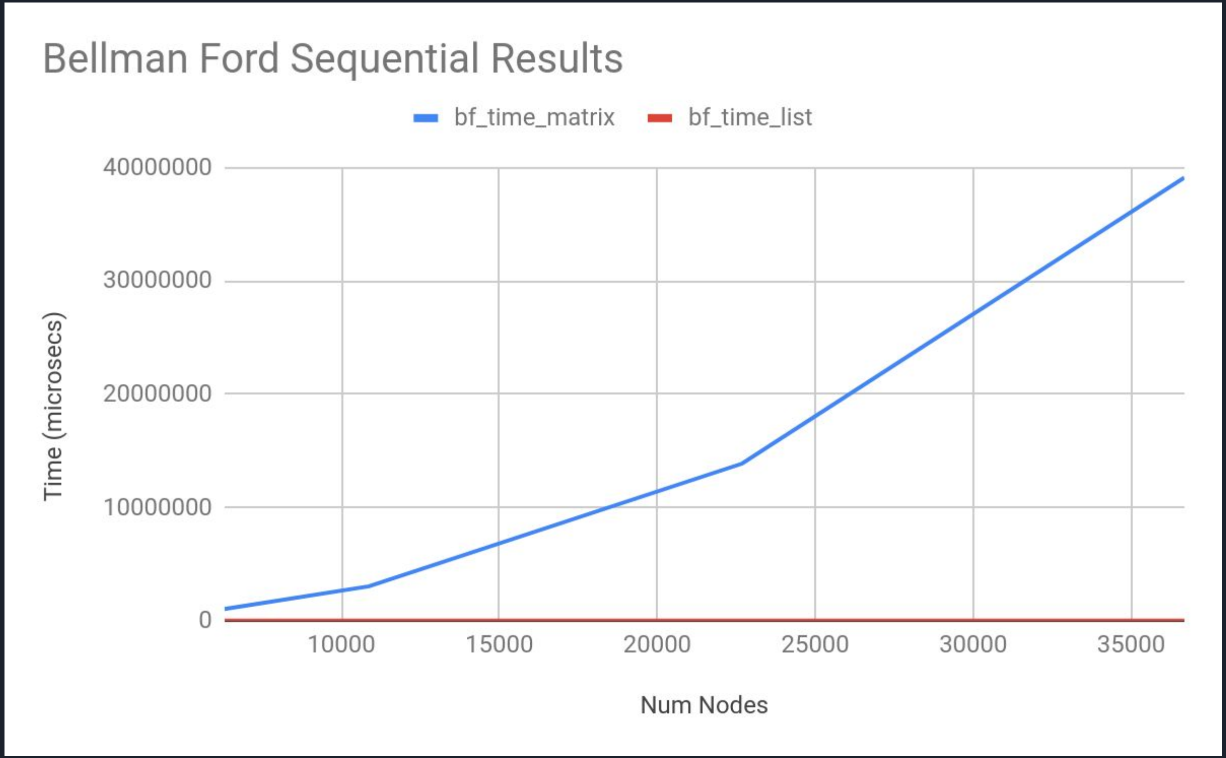
Dataset's

- Using the Stanford Large Network Dataset
- Consists of various sized peer-to-peer network models
    - Undirected and unweighted
    - Very sparse graphs
- In order to create weighted edges, edges are assigned a random weight between 0-255 during the initial parse of the graph

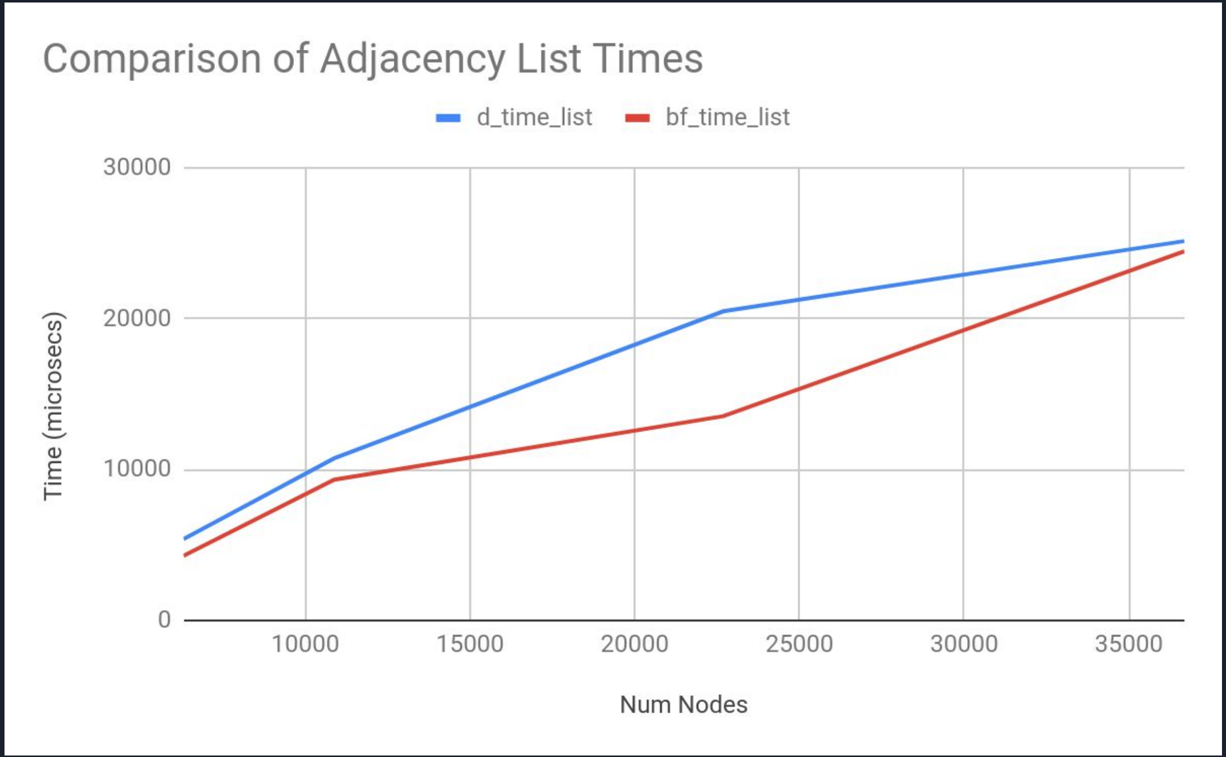
Sequential Results



Sequential Results



Sequential Results



Parallel Results

- Still working on ironing out the parallel versions of Dijkstra’s and Bellman-Ford for the report
- Currently Bellman-Ford has proven much easier to implement
    - Dijkstra’s is a lot less straightforward
- Final step before the final paper

Thank you!