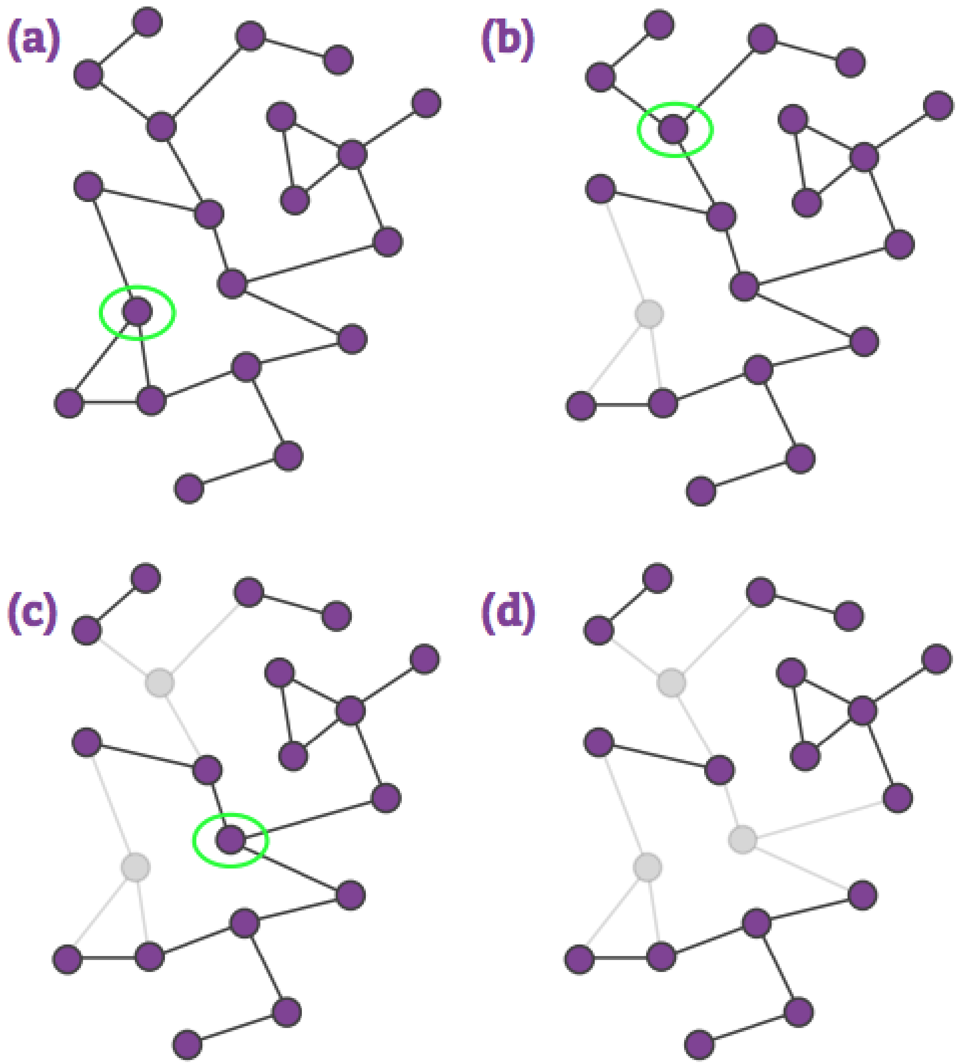
NETWORK ROBUSTNESS

Famim Talukder

INTRODUCTION

Network Robustness – the network's structure plays a role in its ability to survive

- Random failures
- Deliberate attacks



MOTIVATIONS

Biology: some mutations lead to diseases while others do not

Ecology: failure of an ecosystem based on human activity

Engineering: communication systems, power grids, and component failures

VULNERABILITY METRICS

Centrality Metrics

Degree

Closeness

Centroid

Eccentricity

Betweenness

Eigenvector

Robustness Measure

Average path length

Efficiency (power grids)

Largest connected component

PSEUDOCODE – BRANDES ALGORITHM

Algorithm 1: Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + σ[v]/σ[w] · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

Uses BFS for unweighted graphs

Runtime: $O(m \times n)$ on unweighted

Space: $O(m+n)$

Weighted networks:

Runtime: $O(m*n+n^2\log(n))$

LARGEST CONNECTED COMPONENT

Algorithm 2: Largest Connected Component

Data: Graph $G = (V, E)$
Result: Largest Connected Component of unweighted graphs
$Vis[v] \leftarrow 0, v \in V$;
for $v \in V$ **do**
  if $Vis[v] == 0$ **then**
    $DFSuntil(v)$;
  end
end

Data: Node v, $Vis[]$, Graph $G = (V, E)$
$v == 1$ **for** u *adjacent to* v **do**
  if $Vis[u] == 0$ **then**
    DFSuntil$(u, Vis[], G)$;
  end
end

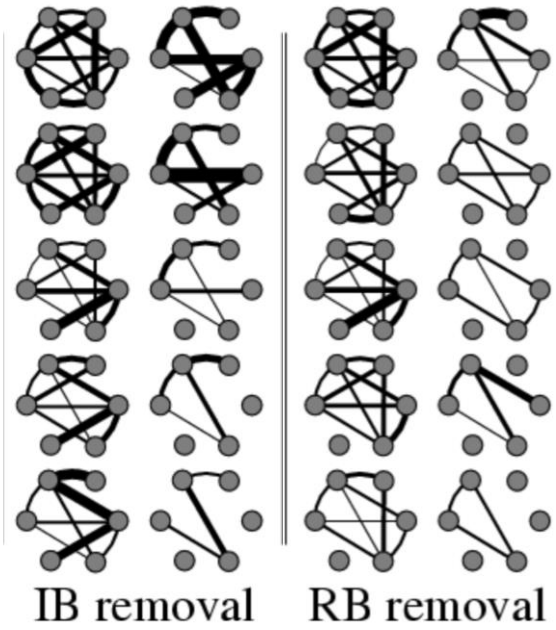
Can be performed with either BFS or DFS

Runtime: O(m+n)

BETWEENNESS CENTRALITY – ROBUSTNESS

Pipeline

- Calculate Betweenness Centrality
- Remove nodes/edges
- Measure robustness
- Compare with initial measure



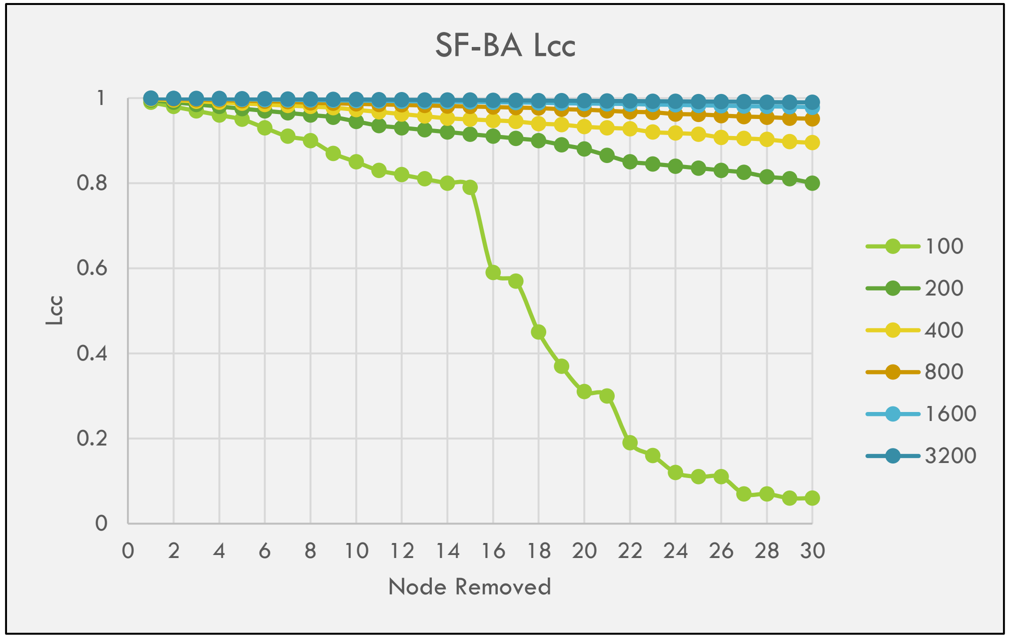
SEQUENTIAL IMPLEMENTATION

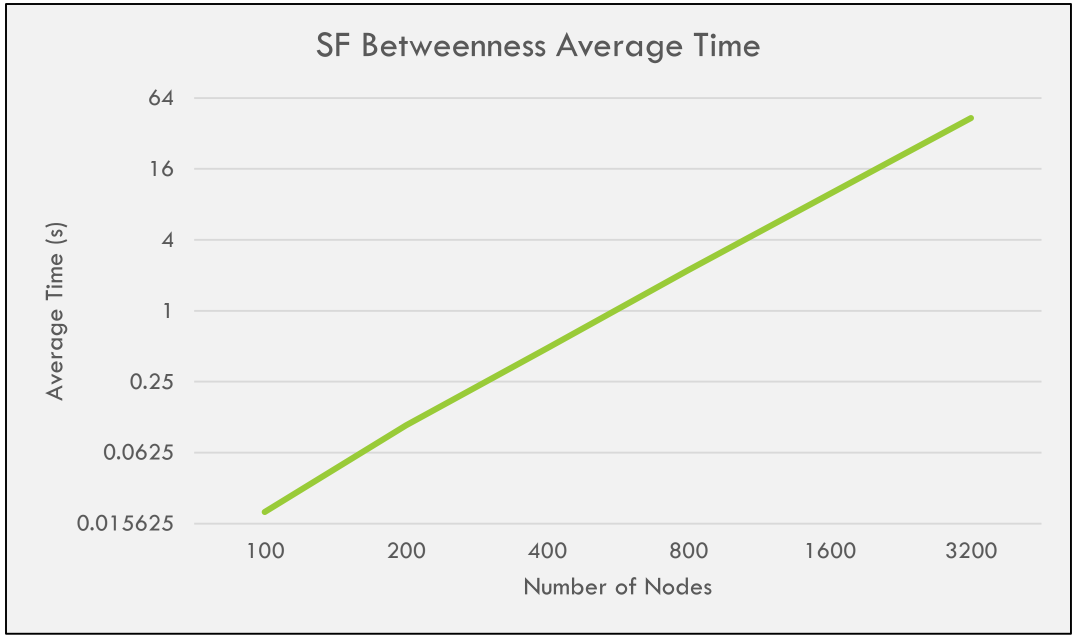
```python
for times in range(0,stop):
    # add a column with NaN
    node_betweenness[col] = np.NaN

    # calculate the betweeness of all the nodes
    time_start = time.time()
    all_betweenness = nx.betweenness_centrality(G, normalized=True)
    time_end = time.time()
#     print("Betweenness Time:", time_end-time_start)

    # insert the betweenness into the panda
    for node, bet_val in all_betweenness.items():
        n = int(node)
        node_betweenness.at[n,col] = bet_val

    # remove the largest betweenness node
    max_betweenness = node_betweenness[col].idxmax()
    max_betweenness_val = node_betweenness[col].max()
    G.remove_node((max_betweenness))
    node_remove_list.append(max_betweenness)
    bet_remove_list.append(max_betweenness_val)
#     print("removed:", max_betweenness, "| val:", max_betweenness_val)

    # find the num of nodes/edges in each subgraph
    all_sungraphs = list(nx.connected_component_subgraphs(G))
    num_nodes_edges_subgraphs = list()
    for k in range(0,len(all_sungraphs)):
        subgraph_nodes = all_sungraphs[k].number_of_nodes()
        subgraph_edges = all_sungraphs[k].number_of_edges()
        num_nodes_edges_subgraphs.append([subgraph_nodes,subgraph_edges])

    # find the largest connected component
    Gcc = max(nx.connected_component_subgraphs(G), key=len)
    Gcc_list.append(Gcc.number_of_nodes())
    col = col + 1
    temp = pd.DataFrame([[time_end-time_start, max_betweenness, max_betweenness_val, \
                          Gcc.number_of_nodes()/num_nodes, G.number_of_nodes(), \
                          G.number_of_edges(), num_nodes_edges_subgraphs]], \
                          columns=['Time', 'Node Rem', 'Bet Value', 'LCC', 'G_nodes', 'G_edges', 'subgraphs (N,E)'])
    df_SF = df_SF.append(temp, ignore_index=True)
```

Python

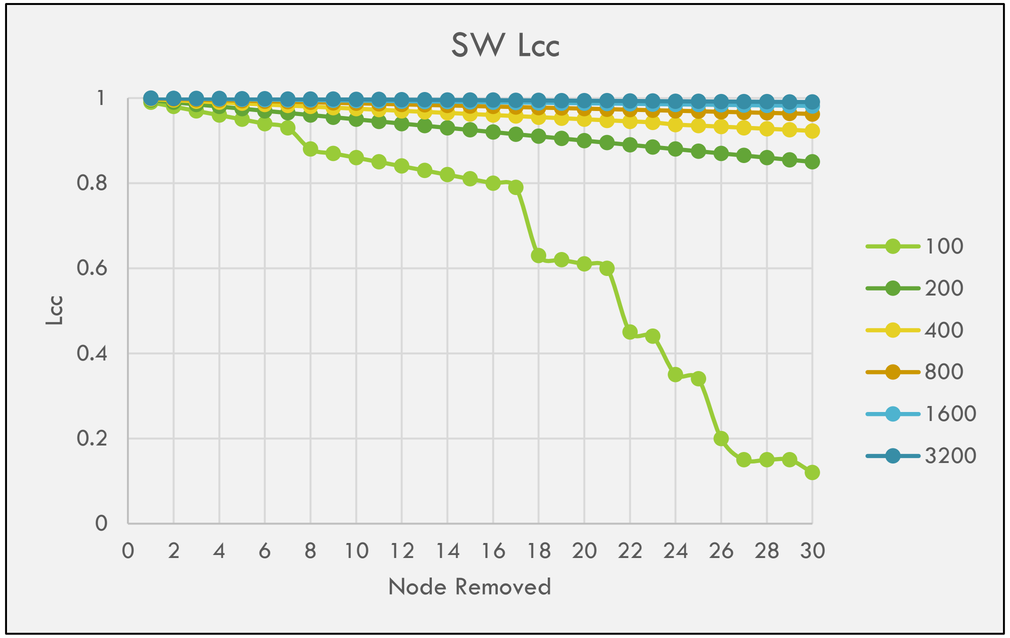
- Networkx
- Uses Brandes Algorithm

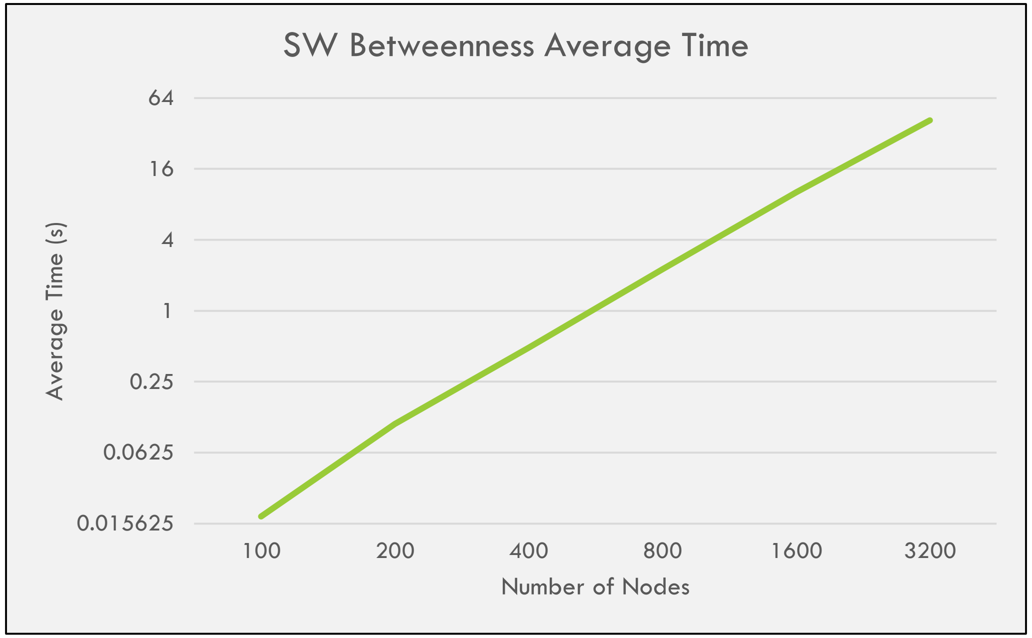
SCALE-FREE MODEL



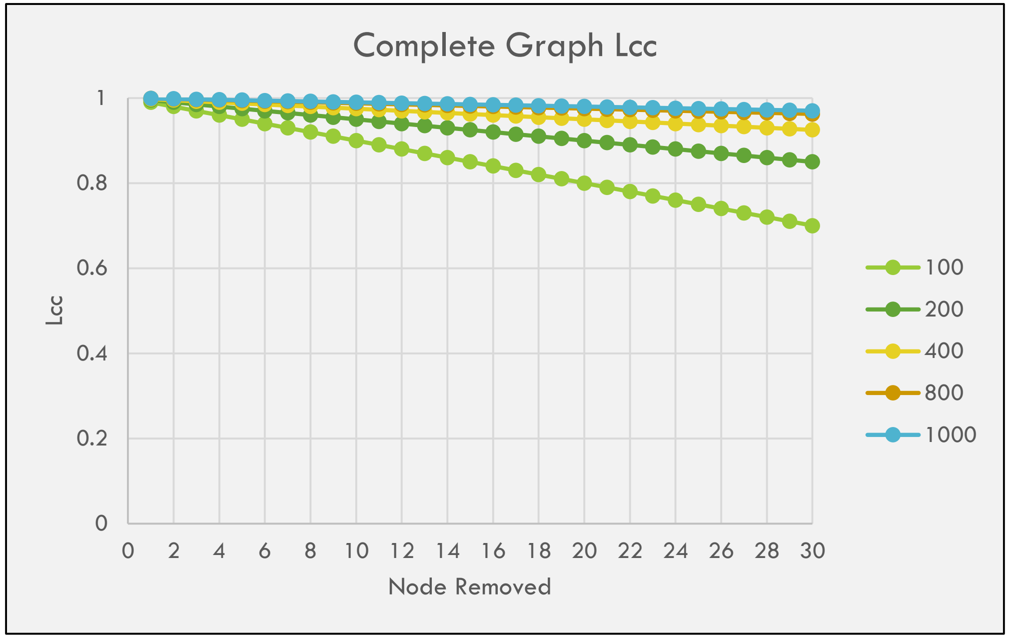


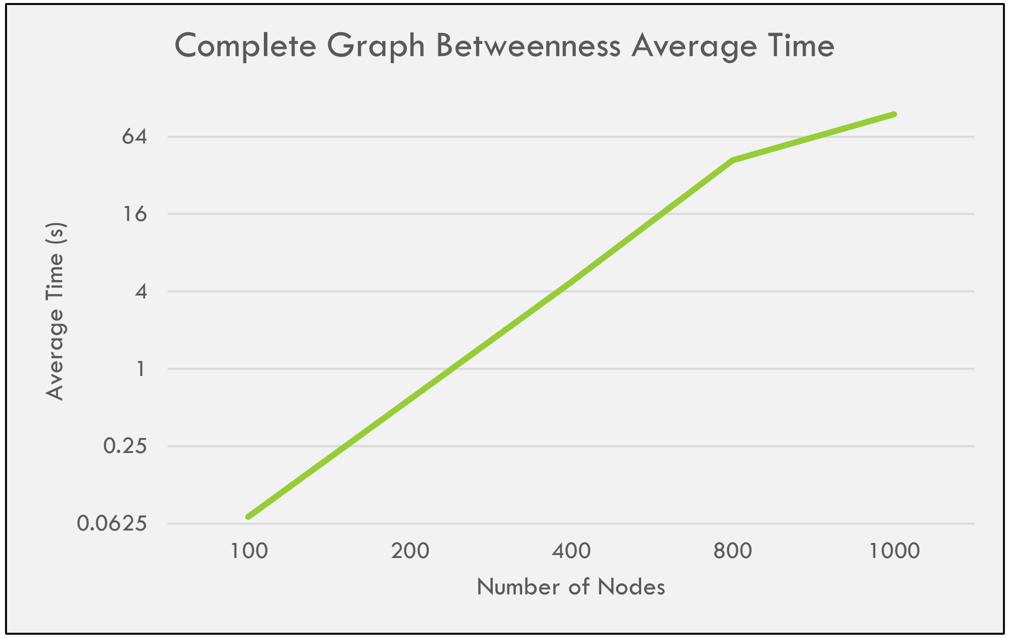
SMALL WORLD MODEL





COMPLETE GRAPH MODEL





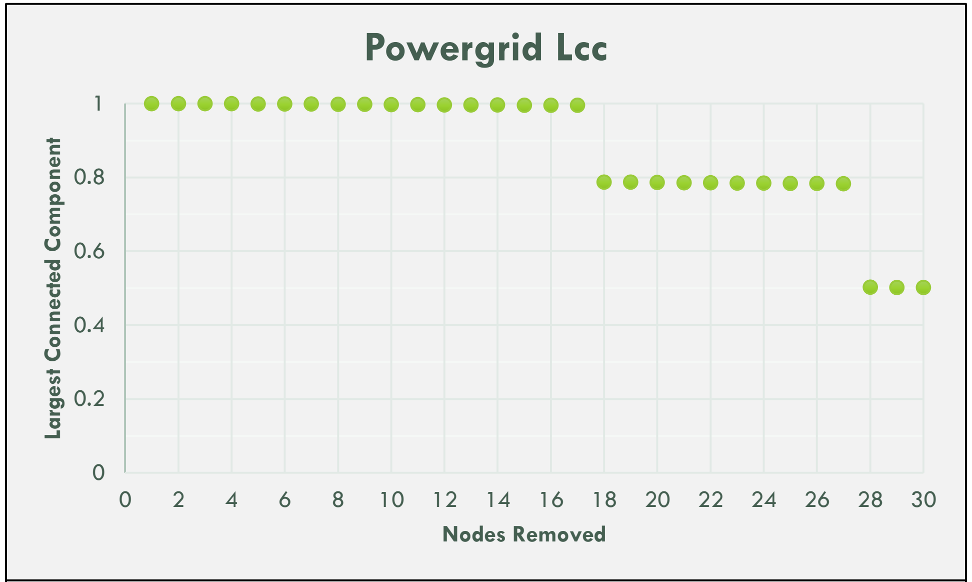
RESULTS

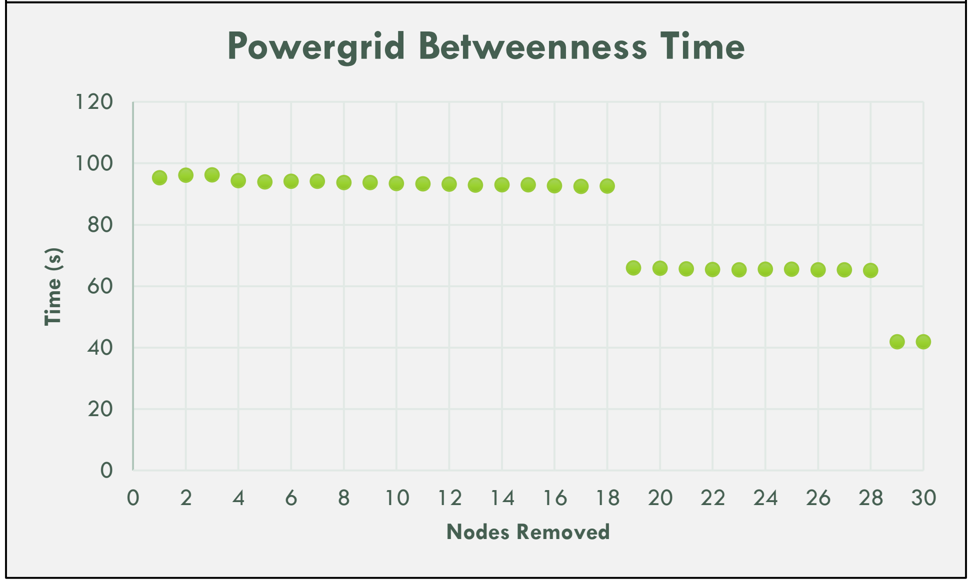
5000 nodes

6500 edges

90-Percentile Diameter: 28.17

Average Shortest Path: 20.09





FUTURE WORK

- Parallelize
- Streaming calculation
- Comparison of IB to RB
- Introduce cascading failure

QUESTIONS/COMMENTS

BACKUP SLIDES