Analysis Algorithms for Large-Scale Networks

Dan Meehan
meehan.49@osu.edu
Table of Contents

- Algorithm analysis overview
- Degree distribution
- Characteristic path length
- Betweenness centrality
  - Exact
  - Approximate
Algorithm Analysis Overview
Asymptotic Complexity

- Algorithms measured on time complexity and space complexity
  - Time complexity - how long an algorithm takes to complete
  - Space complexity - how much memory is needed for computation

<table>
<thead>
<tr>
<th>$O(n)$</th>
<th>$\Omega(n)$</th>
<th>$\Theta(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time grows at least as fast as $n$</td>
<td>Run time grows at most as fast as $n$</td>
<td>Run time grows exactly as fast as $n$</td>
</tr>
</tbody>
</table>
### Graph Representations

- **Adjacency matrix**
  - Space: $\Theta(V^2)$
  - Element query: $\Theta(1)$

- **Adjacency list**
  - Space: $\Theta(V + E)$
  - Element query: $\Theta(\text{degree}(V))$
Degree Distribution
Degree Distribution

- Set of all degrees in a network [5]
  - Mean degree used as a measure of density of the network
Degree Distribution

● Intuitively, will need to visit each vertex v and count edges incident on v
  ○ Can be performed in O(V + E) with an adjacency list
    ■ Loop through adjacency list
    ■ At each vertex, count the number of edges

● Alternatively, use a variant of breadth-first or depth-first search, both of which are O(V + E)
Degree Distribution

- Intuitive approach
Characteristic Path Length
Characteristic Path Length

- Average length of shortest paths between all pairs of vertices in a graph [7]
- Can also look at diameter - longest shortest path

![Graph Diagram]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$L = 1.12$
Characteristic Path Length

- Need to solve all-pairs-shortest-path problem with an unweighted graph
  - Given a graph $G$, find the minimum distance $d_G(s, t)$ for all $s, t \in V$

<table>
<thead>
<tr>
<th></th>
<th>0→3→1</th>
<th>1→3→0</th>
<th>2→3→0</th>
<th>3→0</th>
<th>4→3→0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0→3→2</td>
<td>1→2</td>
<td>2→1</td>
<td>3→1</td>
<td>4→2→1</td>
<td></td>
</tr>
<tr>
<td>0→3</td>
<td>1→3</td>
<td>2→3</td>
<td>3→2</td>
<td>4→3→1</td>
<td></td>
</tr>
<tr>
<td>0→3→4</td>
<td>1→2→4</td>
<td>2→4</td>
<td>3→4</td>
<td>4→2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1→3→4</td>
<td></td>
<td>4→3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Characteristic Path Length

● Naive approach
  ○ Breadth-first search repeated for each vertex
  ○ BFS runs in $O(E + V)$ for one source node, so overall runtime is $O(EV + V^2)$
  ○ If graph is dense, this approaches $O(V^3)$

● Faster method
  ○ Reduce to matrix multiplication [6]
  ○ Runtime: $O(V^{2.376}\log V)$

● Even better method
  ○ dynamic programming - iteratively optimize a $V^2$ matrix of shortest path lengths [3]
  ○ Runtime: $O(V^2\log V)$
  ○ Space: $O(V^2)$
Betweenness Centrality
Betweenness Centrality

- Probability that a given vertex falls on a randomly-selected shortest path between two other vertices in the network [2]

\[ C_B(3) = \frac{4}{7} \]
Betweenness Centrality - exact

- Basic approach [1]
  1. Compute length and number of shortest paths between all pairs
     - Variation of all-pairs-shortest-path problem
  2. Sum all pair-dependencies
     - Pair-dependency - ratio of shortest paths between s and t containing v

- Takes $O(V^3)$ time to sum all pair-dependencies, and $O(V^2)$ space to store shortest paths
Betweenness Centrality - exact

- Faster method [1]
  - Runtime: $O(VE)$ on unweighted graphs
    $O(VE + V^2 \log V)$ on weighted graphs
  - Space: $O(V + E)$
  - Based on BFS for unweighted graphs or Djikstra’s algorithm for weighted graphs
  - Use the fact that $v$ is a predecessor of $w$ to calculate a partial sum for dependency of $s$ on $v$
  - Adding these partial sums together over all predecessors of $w$ yields the pair-dependencies needed to calculate betweenness centrality
Betweenness Centrality - exact

Algorithm 1: Betweenness centrality in unweighted graphs

```plaintext
C_B[v] = 0, v ∈ V;
for s ∈ V do
  S ← empty stack;
P[s] ← empty list, w ∈ V;
σ[t] ← 0, t ∈ V; σ[e] ← 1;
d[t] ← -1, t ∈ V; d[e] ← 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
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[w] ← δ[v] + σ[w] * (1 + δ[w]);
  If w ≠ s then C_B[w] ← C_B[w] + δ[w];
end
```

- Run modified BFS from source s
  1. Compute shortest path lengths and predecessor lists from s to v ∈ V
  2. Update betweenness centrality values for all v ∈ V based on dependency of s on v
- Repeat for all s ∈ V
Betweenness Centrality - exact

- Results on random, undirected, unweighted graphs for size 100-2000 vertices and density 10%-90% of all possible edges
Betweenness Centrality - approximate

- LINERANK algorithm [4]
  - Measure the importance of a node by summing the importance score of its incident edges
  - Importance score of an edge is the probability that a random walker traversing edges via nodes (with random restarts) will stay at the edge
    - Defined using a directed line graph

Original graph

Directed line graph
Betweenness Centrality - approximate

- LINERANK runtime: $O(kE)$
  - Run for $k$ iterations
  - Each iteration improves the accuracy of the estimate, but reasonable accuracy can be achieved after only a few iterations

- LINERANK space: $O(E)$
  - Algorithm uses two incidence matrices, which hold only non-zero elements of the directed line graph, of which there are $E$ elements
Questions?
Bibliography


