The BFS Kernel: Applications and Implementations

Peter M. Kogge
Graph Exploration

• Common graph problem: “explore” region around some vertex

• Exploration: follow edges to see what’s reachable

• Possible outputs:
  – Identification of reachable vertices
  – “labelling” of vertices
  – Properties of reachable sub-graph

• Options:
  – Constraints on “how far”
  – Constraints on “which edges”
Major Variants of Exploration

• Depth-First: Keep jumping from vertex to vertex until stopping
  – And then back up to last vertex and see if any untried edges

• Breadth-First: Explore in waves
  – Explore all edges from current “Frontier”
  – Mark as all new vertices as “New Frontier”
  – Start over with new frontier when all current one is searched

• This kernel: Breadth-First
Example: Airline Routes

• Consider graph with
  – Vertices: airports (~17,000 in world):
    • Properties: Country, International designation, Control tower, etc
  – Edges as flights between airports (100,000/day):
    • Properties: Airline (5,000 different airlines in world), Flight number, equipment, …
    • Edges are directional
  – Note graph changes dynamically

• Possible explorations
  – What airports are reachable from some specific one
  – What if we constraint # of stops or airlines,
  – ...

BFS
Example: Six Degrees of Kevin Bacon

• IMDb Data base
  – Vertices: Multiple “classes”
    • 8.7M+ people
    • 4.8M+ titles of 10 types
  – Edges: \((u,v)\) between people and titles
    • Person \(u\) has had one of 34 roles in title \(v\)
    • Again directional

• Possible exploration:
  – Can a chain of \((u_1,t_1), (u_2,t_1), (u_2,t_2), (u_3,t_2), \ldots\) connect any one person \(u_1\) to all other people in database?
  – Kevin Bacon: 6 titles away from everyone else

• See https://oracleofbacon.org/
Other Interesting Applications

- **From:** [https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/](https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/)
  - Search for neighbors in peer-peer networks
  - Search engine web crawlers
  - Social networks – distance k friends
  - GPS navigation to find “neighboring” locations
  - Patterns for “broadcasting” in networks

  - Community Detection
  - Maze running
  - Routing of wires in circuits
  - Finding Connected components
  - Copying garbage collection, Cheney's algorithm
  - Shortest path between two nodes u and v
  - Cuthill–McKee mesh numbering
  - Maximum flow in a flow network
  - Serialization/Deserialization of a binary tree
  - Construction of the failure function of the Aho-Corasick pattern matcher.
  - Testing bipartiteness of a graph
Key Kernel: BFS - Breadth First Search

- Given a huge graph
- Start with a root, find all reachable vertices
- Performance metric: TEPS: Traversed Edges/sec

Starting at 1: 1, 0, 3, 2, 9, 5

No Flops – just Memory & Networking
Graph500: www.graph500.org

• Several years of reports on performance of BFS implementations on
  – Different size graphs
  – Different hardware configurations

• Standardized graphs for testing

• Standard approach for measuring
  – Generate a graph of certain size
  – Repeat 64 times
    • Select a root
    • Find “level” of each reachable vertex
    • Record execution time
    • TEPS = graph edges / execution time
Graph500 Graphs

- Kronecker graph generator algorithm
  - D. Chakrabarti, Y. Zhan, and C. Faloutsos, R-MAT: A recursive model for graph mining, SIAM Data Mining 2004

- Recursively sub-divides adjacency matrix into 4 partitions A, B, C, D

- Add edges one at a time, choosing partitions probabilistically
  - A = 57%, B = 19%, C = 19%, D = 5%

- # of generated edges = 16*# vertices
  - Average Vertex Degree is 2X this
## Graph Sizes

<table>
<thead>
<tr>
<th>Level</th>
<th>Scale</th>
<th>Size</th>
<th>Vertices (Billion)</th>
<th>TB</th>
<th>Bytes /Vertex</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>26</td>
<td>Toy</td>
<td>0.1</td>
<td>0.02</td>
<td>281.8048</td>
</tr>
<tr>
<td>11</td>
<td>29</td>
<td>Mini</td>
<td>0.5</td>
<td>0.14</td>
<td>281.3952</td>
</tr>
<tr>
<td>12</td>
<td>32</td>
<td>Small</td>
<td>4.3</td>
<td>1.1</td>
<td>281.472</td>
</tr>
<tr>
<td>13</td>
<td>36</td>
<td>Medium</td>
<td>68.7</td>
<td>17.6</td>
<td>281.4752</td>
</tr>
<tr>
<td>14</td>
<td>39</td>
<td>Large</td>
<td>549.8</td>
<td>141</td>
<td>281.475</td>
</tr>
<tr>
<td>15</td>
<td>42</td>
<td>Huge</td>
<td>4398.0</td>
<td>1,126</td>
<td>281.475</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Average</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>281.5162</td>
</tr>
</tbody>
</table>

\[
\text{Scale} = \log_2(\# \text{ vertices})
\]
Notional Sequential Algorithm

• Top-Down search: Keep a “frontier” of new vertices that have been “touched” but not “explored”
  – Explore them and repeat

• Bottom-up search: look at all “untouched vertices” and see if any of their edges lead to a touched vertex
  – If so, mark as touched, and repeat

• Special considerations
  – Vertices that have huge degrees
Algorithm 1 Top-Down BFS:
V is the set of vertices; E a set of edges

1: procedure TOP DOWN-BFS(G, ROOT)
2:    Touched ← {root}
3:    Frontier ← {root}
4:    Labels ← N-vector of a large integer
5:    Label[root] ← 0
6:    Level ← 0
7:    while Frontier not empty do
8:        Level + = 1
9:        TopDown - Pass(Frontier, Touched, Level)
10:     return
11: procedure Top Down Pass(Touched, Level)
12:    Next ← {}
13:    for u in Frontier do
14:        for all edges (u, v) in E do
15:            if v not in Touched then
16:                Touched ← Touched ∪ {v}
17:                Next ← Next ∪ {v}
18:                Label[v] ← Level
19:            Frontier ← Next
20:     return

Notional Complexity: O(M)
Algorithm 2 BottomUp BFS:

V is the set of vertices; E a set of edges

1: procedure BOTTOMUP-BFS(G,ROOT)
2: \begin{align*}
    & Touched \leftarrow \{\text{root}\} \\
    & Labels \leftarrow \text{N-vector of a large integer} \\
    & Label[\text{root}] \leftarrow 0 \\
    & Level \leftarrow 0 \\
    & TouchedFlag \leftarrow True
\end{align*}
3: \begin{align*}
    & \text{while } TouchedFlag \text{ do} \\
    & \text{Level} += 1 \\
    & \text{TouchedFlag} \leftarrow \text{BackwardPass}(\text{Touched, Level})
\end{align*}
4: return

5: procedure BOTTOMUPPASS(INOUT Touched, Level)
6: \begin{align*}
    & TouchedFlag \leftarrow False \\
    & \text{for } v \text{ not in } Touched \text{ do} \\
    & \text{for all edges } (u, v) \text{ in } E \text{ do} \\
    & \quad \text{if } u \text{ in } Touched \text{ then} \\
    & \quad \quad \text{TouchedFlag} \leftarrow True \\
    & \quad \text{Touched} \leftarrow Touched \cup v \\
    & \quad \text{Label}[v] \leftarrow \text{Level}
\end{align*}
8: return TouchedFlag

Notional Complexity: \( O(NM) \)
Key Observation

• Forward direction requires investigation of every edge leaving a frontier vertex
  – Each edge can be done in parallel

• Backwards direction can stop investigating edges as soon as 1 vertex in current frontier is found
  – If search edges sequentially, potentially significant work avoidance

• In any case, can still parallelize over vertices in frontier
By this level, most vertices now touched, so edges explored mostly point backward.

Going backwards from untouched vertices, and stopping on first touch, reduces # edges covered to near-optimal (optimal is 1 edge per vertex).

Few nodes in early levels mean few edges.

Fig. 5: Graph properties at each exploration level.

Checconi and Petrini, “Traversing Trillions …”
Beamer’s Hybrid Algorithm

• Switch between forward & backward steps
  – Use forward iteration as long as In is small
  – Use backward iteration when Vis is large

• Advantage: when
  – # edges from vertices in !Vis
  – are less than # edges from vertices in In
  – then we follow fewer edges overall

• Estimated savings if done optimally: up to 10X reduction in edges

• http://www.scottbeamer.net/pubs/beamer-sc2012.pdf
Hybrid Algorithm

- $N_f$ = # vertices in current frontier
- $M_f$ = # outgoing edges from current frontier
- $M_u$ = # incoming edges to current untouched
- $\alpha$ = edge reduction factor in bottom-up pass
- $\beta$ = vertex reduction factor when going from bottom-up to top-down

Switch from top-down to bottom-up when:

$$M_f < \frac{M_u}{\alpha}$$

Switch back from bottom-up to top-down when

$$N_f < \frac{N}{\beta}$$

BFS