Direction-Optimizing BFS

Beamer, Asanovic, Patterson

Graph terminology

Diameter: maximum (shortest) distance between two nodes

Small-world graph: diameter proportional to log(|V|)

Scale-free graph: # nodes with degree k is proportional to k^{-}[illegible]

Social networks usually satisfy both properties (facebook, twitter, wikipedia, hollywood)

BFS recap

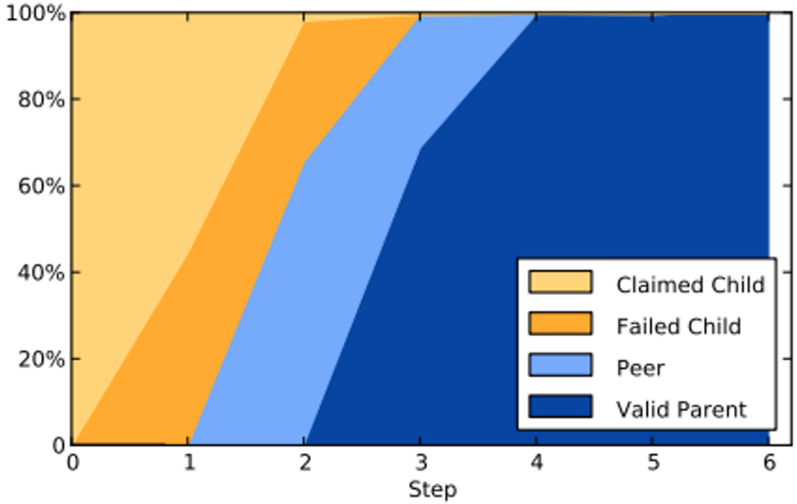
```
function breadth-first-search(vertices, source)
    frontier ← {source}
    next ← {}
    parents ← [-1,-1,... -1]
    while frontier ≠ {} do
        top-down-step(vertices, frontier, next, parents)
        frontier ← next
        next ← {}
    end while
    return tree
```

```
function top-down-step(vertices, frontier, next, parents)
    for v ∈ frontier do
        for n ∈ neighbors[v] do
            if parents[n] = -1 then
                parents[n] ← v
                next ← next ∪ {n}
            end if
        end for
    end for
```

Top-down step (expanding frontier)

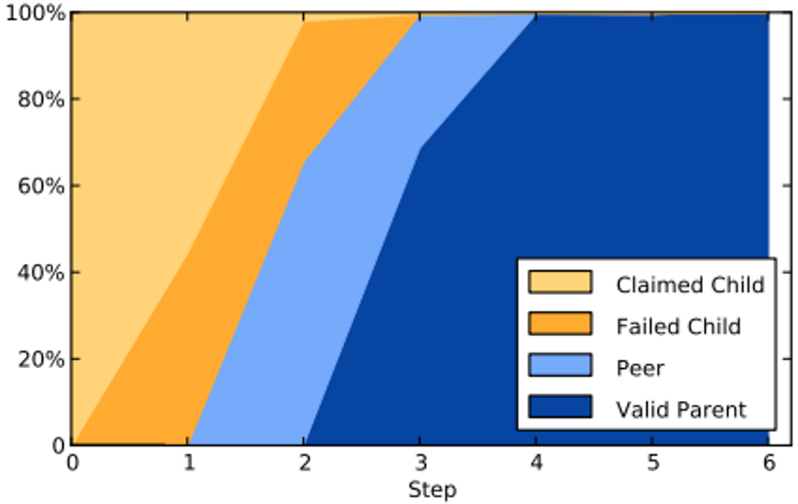
4 outcomes when we check an edge

1. claimed child (good)
2. failed child
3. peer
4. valid parent



Top-down BFS analysis

1. Initially, lots of claimed children, since most of graph is unvisited
2. More peers as frontier grows
3. Finally, valid parents when most of graph is visited



As more nodes are visited, fewer edge checks result in claimed children.

Bottom-up BFS

Look at all the unvisited nodes and check if it's adjacent to frontier

```
function bottom-up-step(vertices, frontier, next, parents)
    for v ∈ vertices do
        if parents[v] = -1 then
            for n ∈ neighbors[v] do
                if n ∈ frontier then
                    parents[v] ← n
                    next ← next ∪ {v}
                    break
                end if
            end for
        end if
    end for
```

Small side effect: one writer per node, no need for CAS in parallel implementation

Bottom-up BFS downsides

Bottom-up BFS downsides

Wastes a lot of work if most of the graph is unvisited

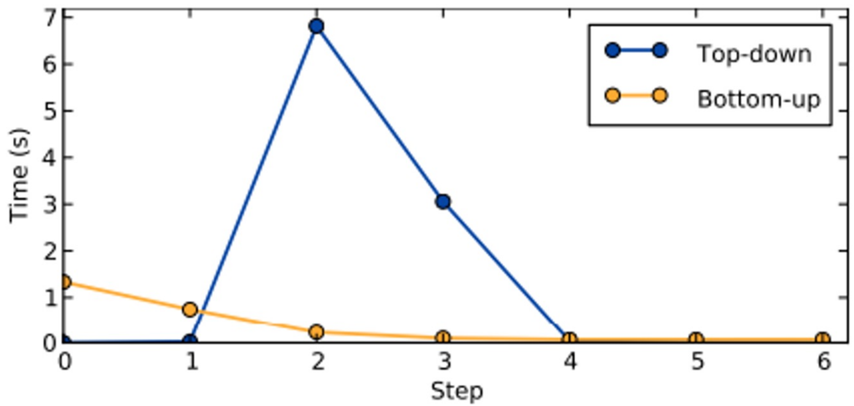
Overhead from nodes not connected to the source

Loops over all the nodes to find unvisited ones

Hybrid BFS

Intuitively:

- When frontier is small: use top-down BFS
- When frontier is large: use bottom-up BFS



Hybrid BFS

n_f = number of nodes in frontier

m_f = number of edges to check in the frontier (exact)

m_u = number of edges to check from unvisited nodes (upper bound)

Since m_u is a upper bound, use a heuristic to determine when to switch to bottom-up

Due to bottom-up overhead, switch to top-bottom when frontier is small

Hybrid BFS

Switch from top-down to bottom-up when:

$$m_f > \frac{m_u}{\alpha}$$

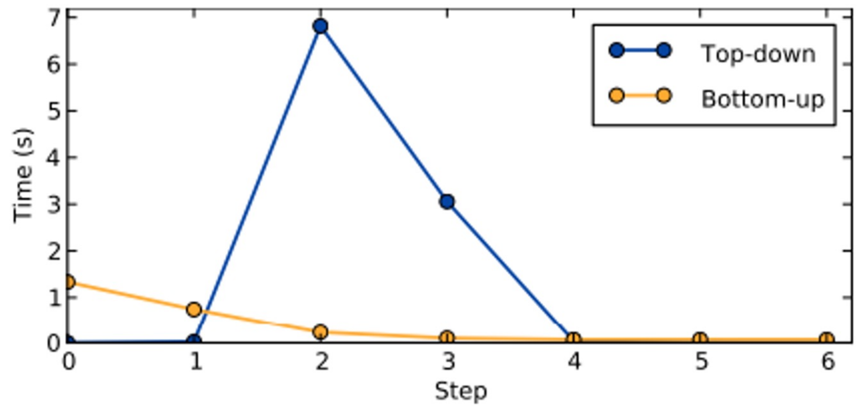
Switch from bottom-up to top-down when:

$$n_f < \frac{n}{\beta}$$

α and β are hyperparameters

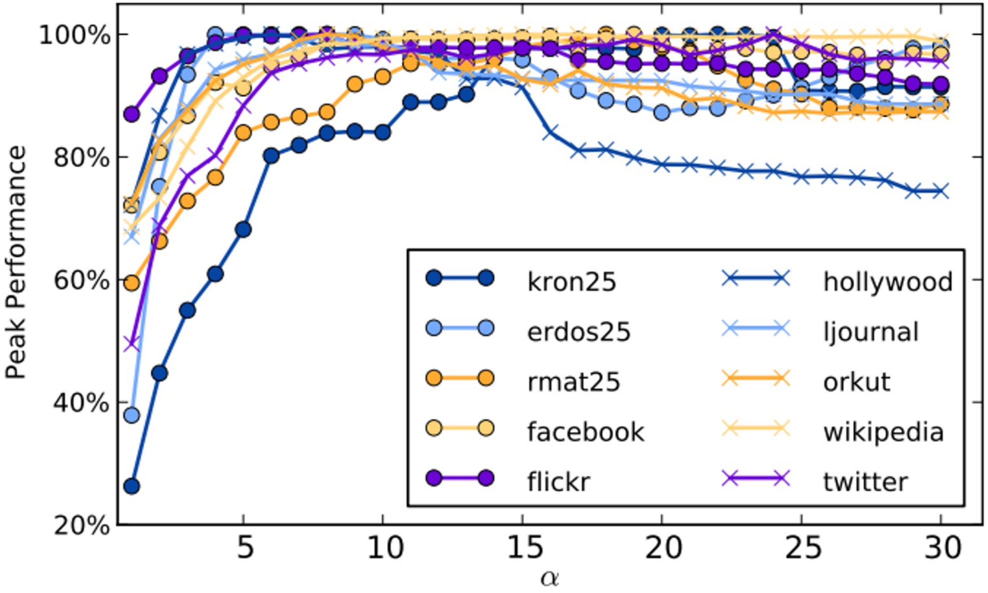
Parameter Tuning

Runtime is dominated by middle steps when the frontier is large, so tuning ▯ is more important than ▯.



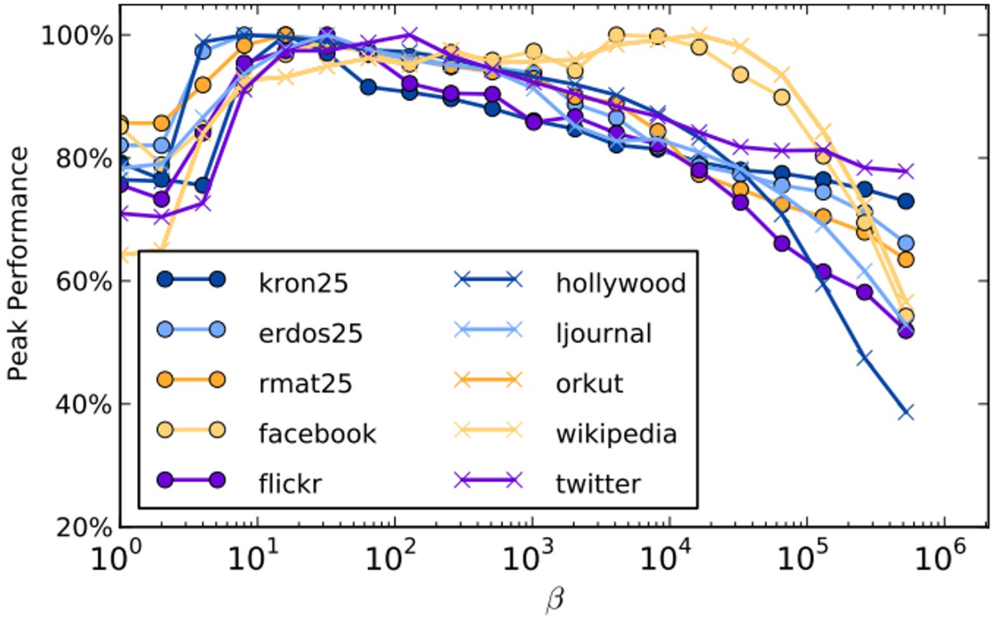
Parameter Tuning: α

How large is the frontier (edgewise), compared to unvisited edges, before we switch to bottom-up?



Parameter Tuning: β

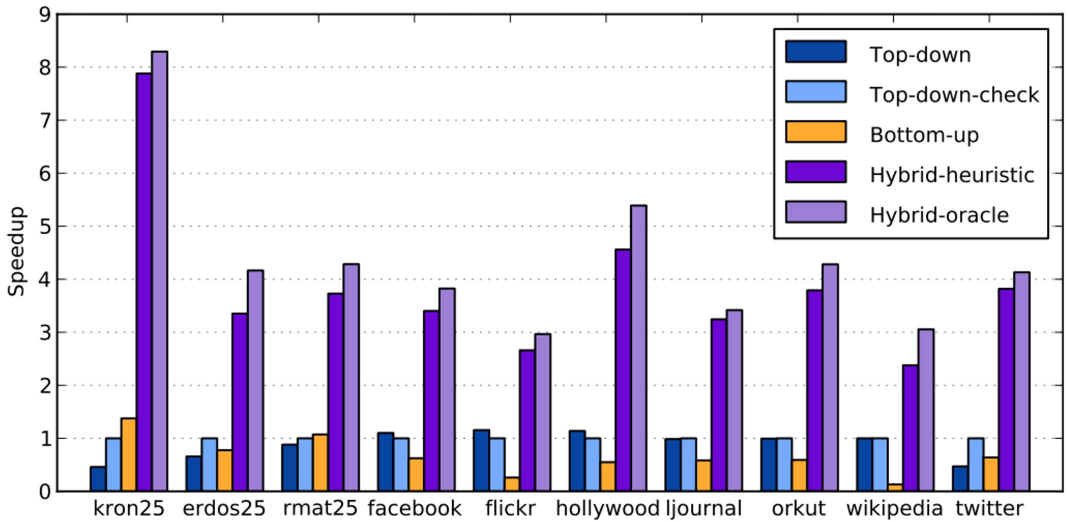
How small is the frontier (nodewise), before we switch back to top-down?



Evaluation

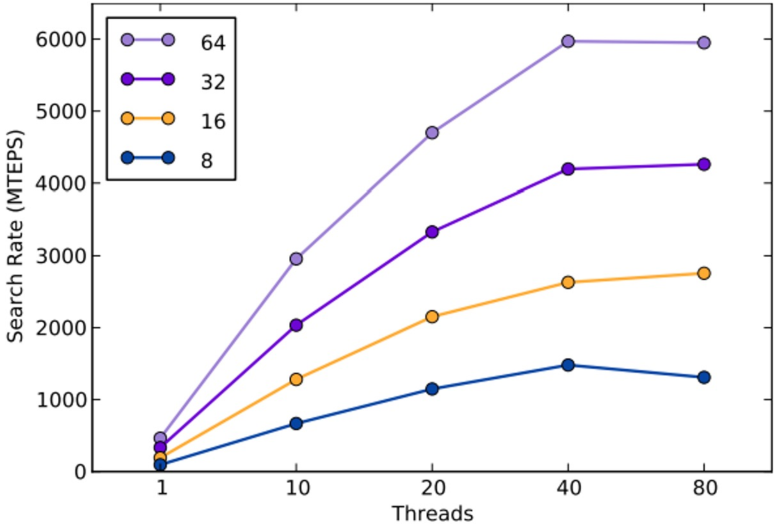
Compare against regular top-down BFS

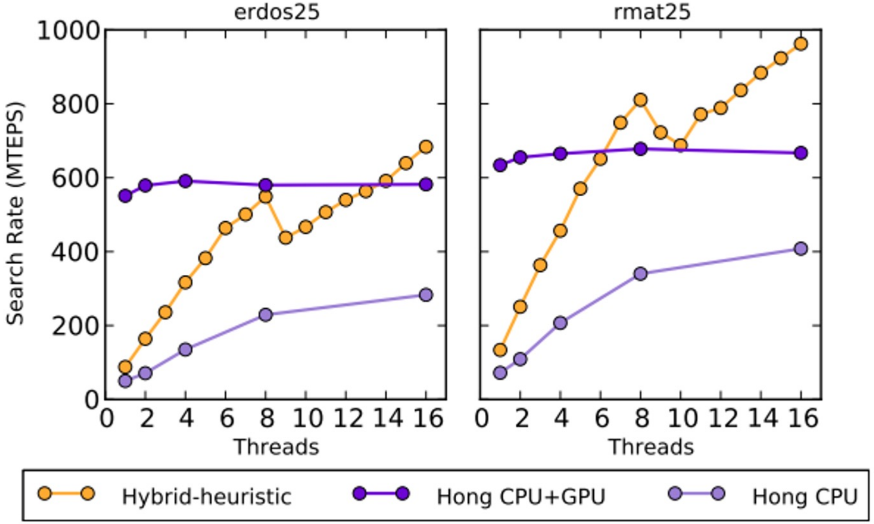
Also, compare against optimal hybrid: “offline” oracle



Parallelism

MTEPS = millions of edges traversed per second (more on Hong et al next slide)





Related Work

Hong et al. also uses a hybrid-heuristic approach. Instead of switching algorithms, they switch between CPU and GPU

Most of the other related papers are on optimizing memory utilization and parallelism