Direction-Optimizing Breadth-First Search

Authors: Scott Beamer, Krste Asanović, David Patterson

Rahul Yesantharao

02/11/2020

Context

- Graphs can represent an immense range of problem classes, but it is hard to write high-performance graph algorithms in the most general sense
- By focusing on specific types of graphs, we can write much higher performance graph algorithms than would otherwise be possible
- Breadth-First Search is a fundamental but important piece of many graph-processing programs
- One specific application domain for BFS is the analysis of social networks, which are often represented by very large graphs

Social Network Graphs

- Low-diameter: Graphs that have small diameters, relative to the graph size
  - Diameter: Maximum (shortest) distance between any two vertices
  - Small-World: The diameter grows logarithmically in the number of nodes
- Scale-Free: The degree distribution follows a power law
  - There are a few nodes with very high degrees, and many more with geometrically smaller degrees
- Difficult to parallelize because the small diameter makes it difficult to partition them equitably, and the geometric degree distribution causes the work per node to vary drastically

Breadth-First Search

- Common algorithm for exploring a graph
- Executed in a stepwise fashion
- At each step, maintains a “frontier” of nodes that will be explored at that step
  - Starts with the single source vertex in the frontier
- At each step, expands the “frontier” by traversing all of the “frontier edges” (edges with at least one end-vertex in the frontier) and adding all the unvisited vertices to the next frontier
- This traversal can happen in two directions

Breadth-First Search

```
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
```

Fig. 1. Conventional BFS Algorithm

Top-Down BFS Step

```
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
```

Fig. 2. Single Step of Top-Down Approach

Bottom-Up BFS Step

```
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
```

Fig. 5. Single Step of Bottom-Up Approach

BFS on Social Network Graphs

- Because of the structure of social network graphs, BFS has a distinctive execution pattern.
- Low-Diameter: The BFS completes in a relatively small number of steps, and thus visits a large fraction of vertices on each of the first few steps
- Scale-Free: Some vertices will have a geometrically larger degree than others, so the frontier growth will be significantly faster than implied by the average degree
- Overall, the frontier size grows and then decreases exponentially within the first few steps

BFS Work

- The majority of the work comes from exploring all of the frontier edges
- In social-network graphs, after the first few steps, most of these explorations are wasted
- They can be placed in four categories
  - Valid parent: A neighbor from the previous frontier
  - Peer: A neighbor currently in the same frontier
  - Failed child: A new neighbor that has already been visited by another node in the same frontier
  - Claimed child: A successful edge traversal

BFS Work

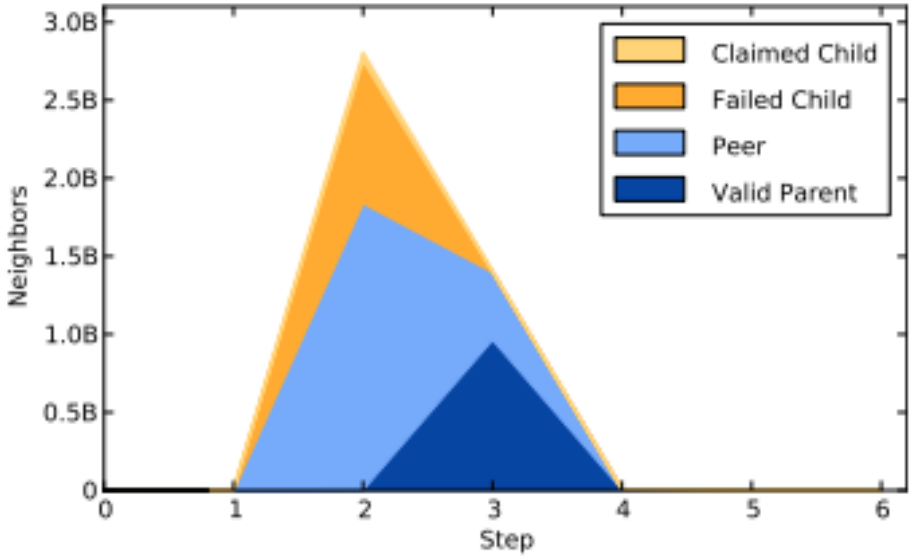

Fig. 3. Breakdown of edges in the frontier for a sample search on `kron27` (Kronecker generated 128M vertices with 2B undirected edges) on the 16-core system.

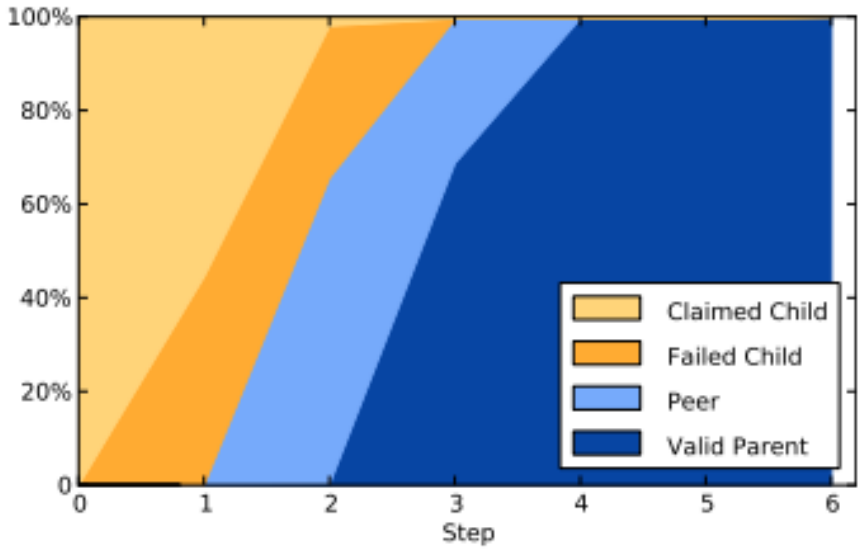

Fig. 4. Breakdown of edges in the frontier for a sample search on `kron27` (Kronecker generated 128M vertices with 2B undirected edges) on the 16-core system.

Hybrid BFS Approach

- The conventional top-down approach results in the edge distribution from before, which is very wasteful after a few steps
- The bottom-up approach by itself would be very poor in the first few steps, because the number of unclaimed vertices is huge, and so we would be exploring a massive number of edges searching for very few frontier vertices
- Thus, we combine the two approaches, and use the appropriate exploration method at each step
- Different frontier representations
  - Top-Down: FIFO Queue
  - Bottom-Up: Bitmap

Hybrid BFS Approach

- In general, the first few steps are faster with top-down, the intermediate steps are better with bottom-up, and the final steps are better with top-down (the tail of the geometric distribution)
- Choosing when to switch is a key decision rule that this paper introduces

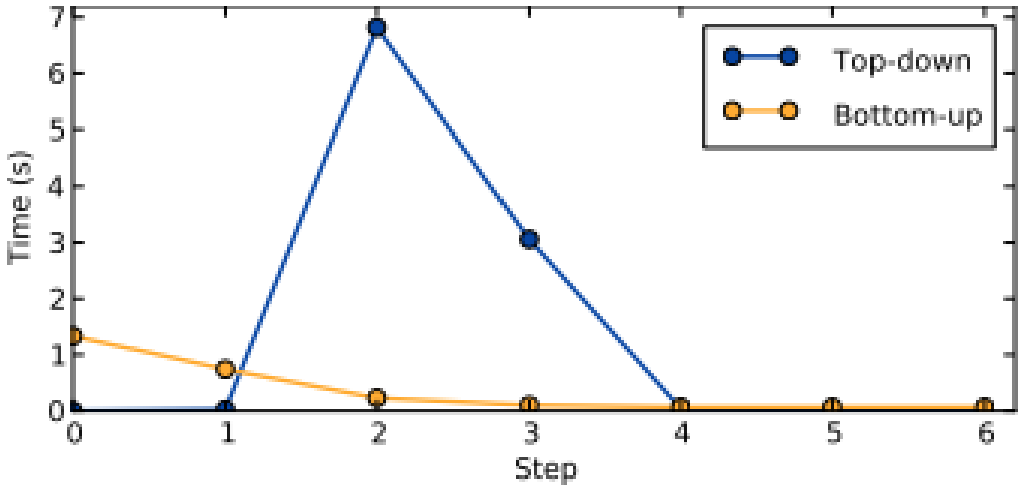

Fig. 6. Sample search on `kron27` (Kronecker 128M vertices with 2B undirected edges) on the 16-core system.

Hybrid BFS Approach

- n_f: number of vertices on the frontier
- n: number of vertices
- m_f: number of frontier edges
- m_u: number of edges from unexplored vertices
- α, β: tunable parameters

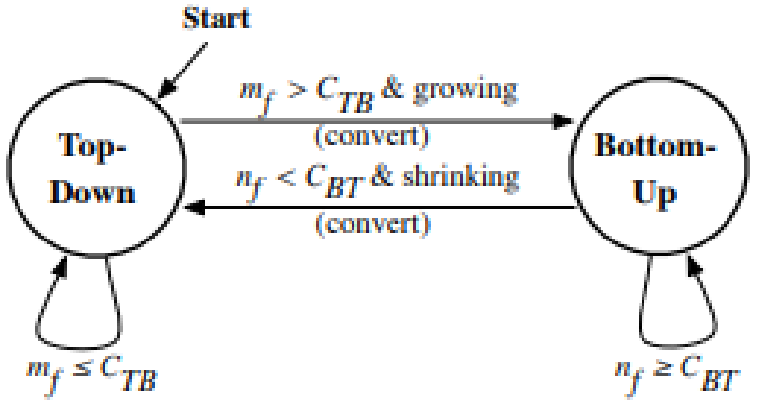

Fig. 7. Control algorithm for hybrid algorithm. (convert) indicates the frontier must be converted from a queue to a bitmap or vice versa between the steps. Growing and shrinking refer to the frontier size, and although they are typically redundant, their inclusion yields a speedup of about 10%.

$$\frac{m_u}{\alpha} = C_{TB} \qquad \frac{n}{\beta} = C_{BT}$$

Tuning Parameters

- Parameter tuning is done by sweeping the two parameters over a wide range of possible values for several test graphs and choosing the values that result in the best average and minimum performance values
- $\alpha = 14, \beta = 24$

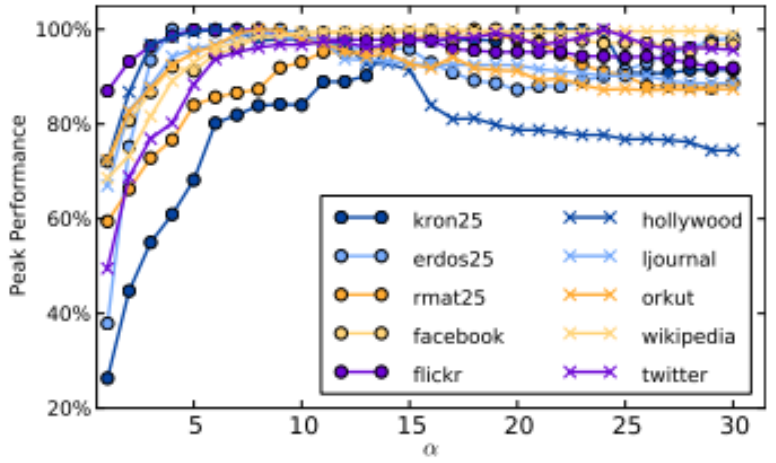

Fig. 8. Performance of *hybrid-heuristic* on each graph relative to its best on that graph for the range of α examined.

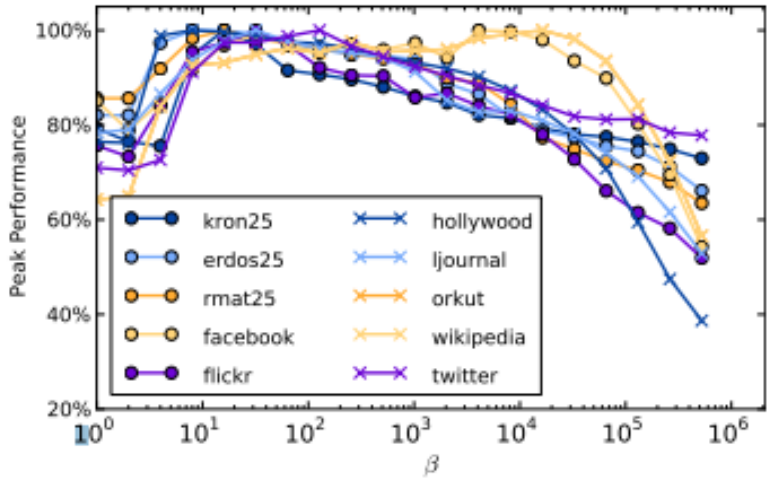

Fig. 9. Performance of *hybrid-heuristic* on each graph relative to its best on that graph for the range of β examined.

Related Work

- Bader and Madduri: Parallel BFS that parallelizes across vertices and edges, specifically on an MTA-2 (shared memory, no caching)
- Agarwal, et al: Optimize memory locality and memory traffic by pinning threads to specific sockets and minimizing messages between sockets
- Hong, et al: Use both CPUs and GPUs, with a decision rule to switch between them (always top-down) and specific data structures to optimize locality in each case.
- Merrill, et al: Optimize GPU performance by using prefix sums and bitmaps; considered the fastest shared memory BFS
- Chhugani, et al: Many optimizations to memory usage, locality, and socket communications

Experimental Results

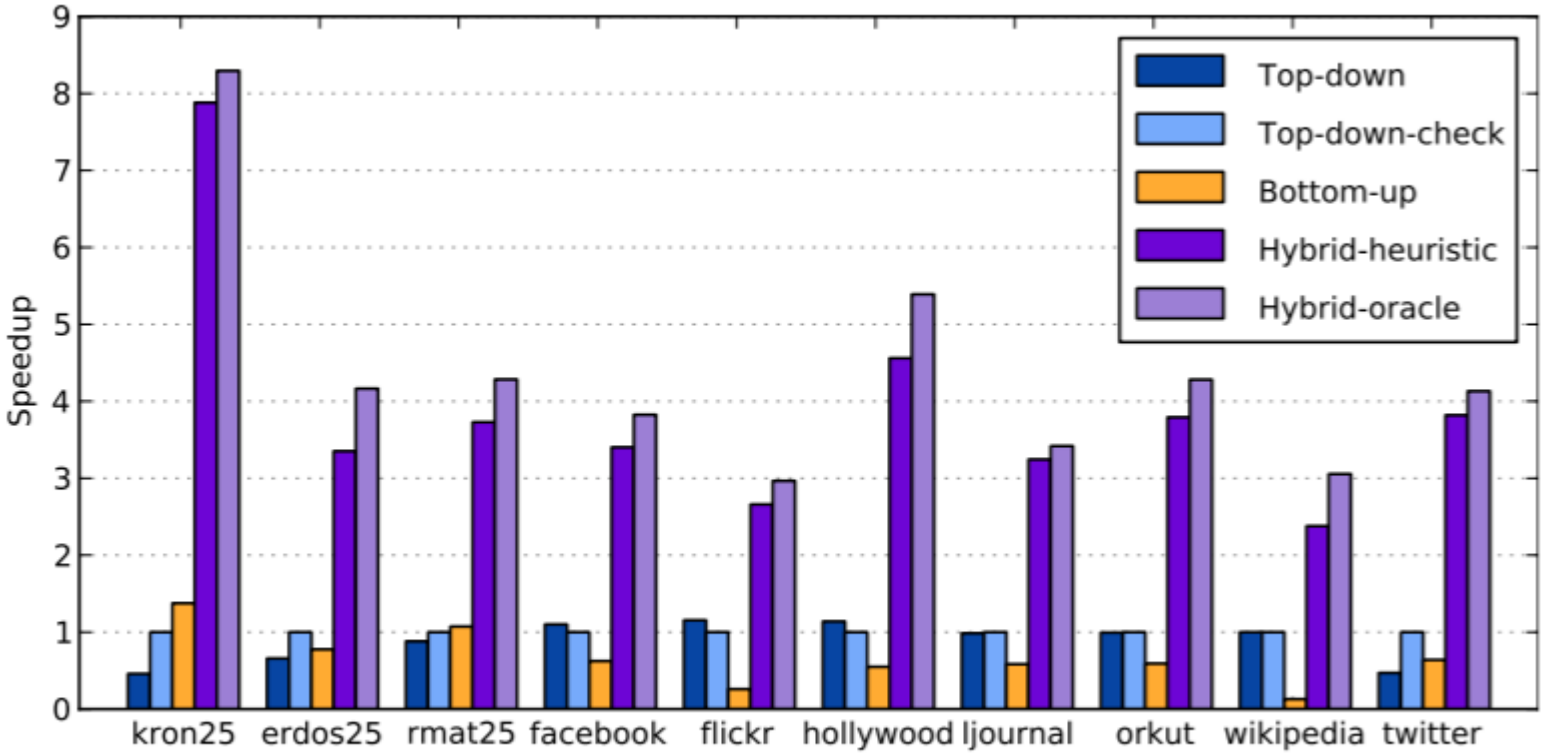

Fig. 10. Speedups on the 16-core machine relative to *Top-down-check*.

Experimental Results

System	kron_ g500-logn20	random. 2Mv.128Me	rmat. 2Mv.128Me
GPU results from Merrill et al. [20]			
Single-GPU	1.25	2.40	2.60
Quad-GPU	3.10	7.40	8.30
Hybrid-heuristic results on multicore			
8-core	7.76	6.75	6.14
16-core	12.38	12.61	10.45
40-core	8.89	9.01	7.14

TABLE IV
Hybrid-heuristic ON MULTICORE SYSTEMS IN THIS STUDY COMPARED TO GPU RESULTS FROM MERRILL ET AL. [20] (IN GTEPS).

	rmat-8	rmat-32	erdos-8	erdos-32	orkut	facebook
Prior	750	1100	590	1010	2050	920
8-core	1580	4630	850	2250	4690	1360

TABLE III
PERFORMANCE IN MTEPS OF *Hybrid-heuristic* ON THE 8-CORE SYSTEM COMPARED TO CHHUGANI ET AL. [10]. SYNTHETIC GRAPHS ARE ALL 16M VERTICES, AND THE LAST NUMBER IN THE NAME IS THE DEGREE.

Experimental Results

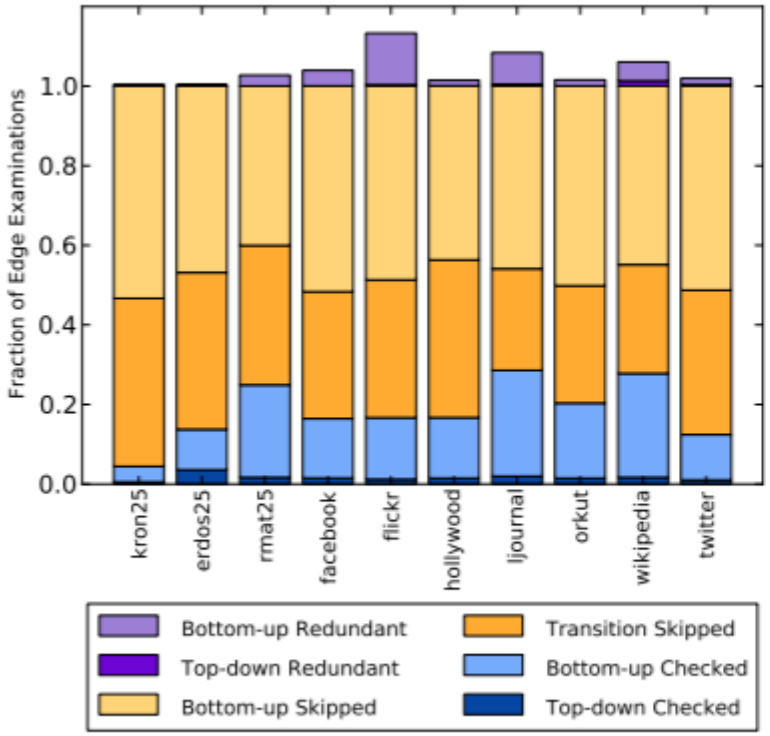

Fig. 11. Breakdown of edge examinations.

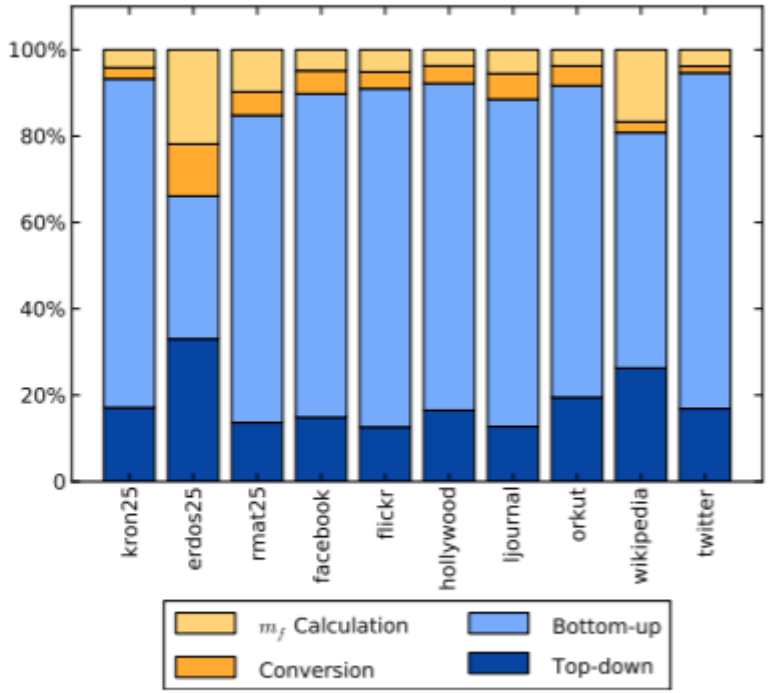

Fig. 12. Breakdown of time spent per search.

Parallelization

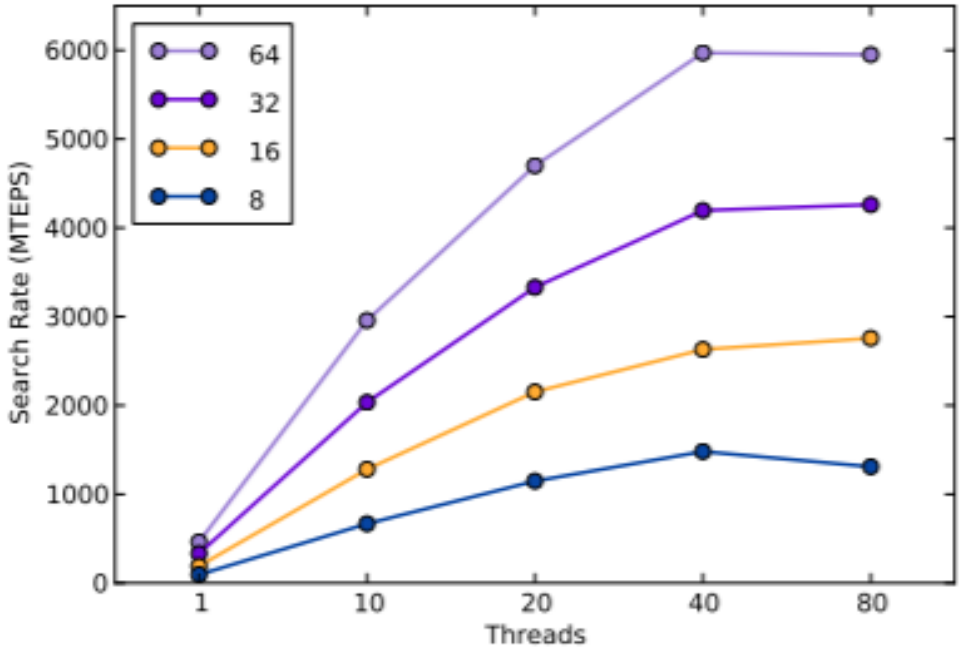

Fig. 14. Parallel scaling of *Hybrid-heuristic* on the 40-core system for an RMAT graph with 16M vertices and varied degree.

Thoughts

- Strengths
  - Achieves speedup over many existing BFS implementations
  - Is empirically competitive with the offline optimal oracle
  - Focuses on reducing the total number of edges traversed rather than speeding up conventional BFS
  - Manages to parallelize the algorithm pretty well
- Weaknesses
  - Does not focus too heavily on tuning its parameters, using a simple grid search over a rather small set of training data
  - Does not use a particularly sophisticated decision rule for switching

Discussion

- What are ideas for extending this algorithm?
- How can we extend the decision rules?
- What other features seem relevant to the switching thresholds?