Direction-Optimizing BFS

Jason Priest

Traditional BFS

```
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

```
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

Small World Phenomenon

- Avg degree of 16
- Frontier balloons rapidly
- Heavily revisiting edges
- Nearly all edge visits fail

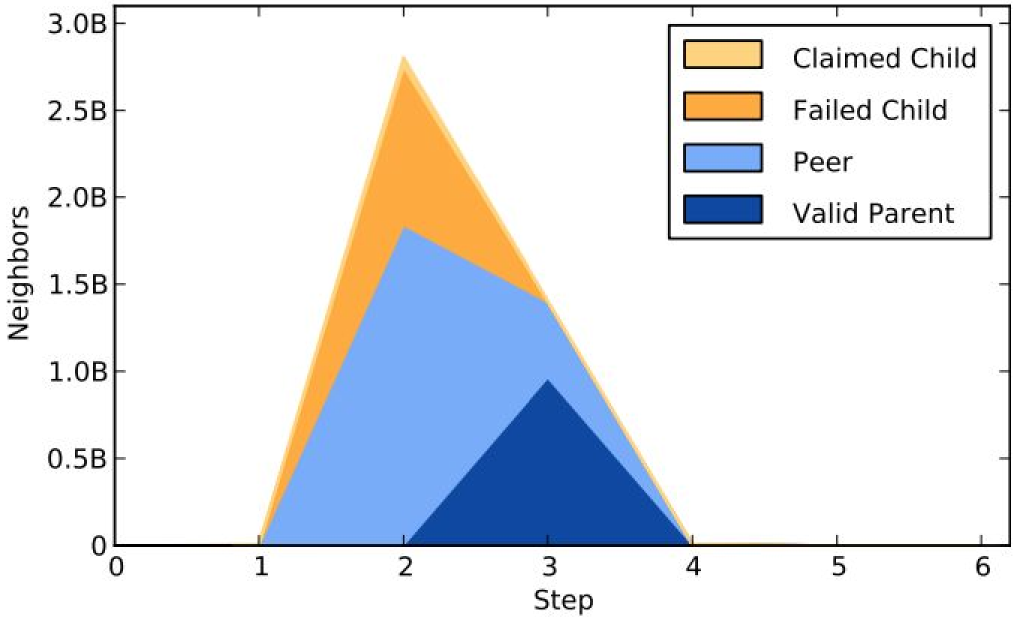

Fig. 3. Breakdown of edges in the frontier for a sample search on kron27 (Kronecker generated 128M vertices with 2B undirected edges) on the 16-core system.

Small World Phenomenon

- After first few steps, nearly all edge visits fail

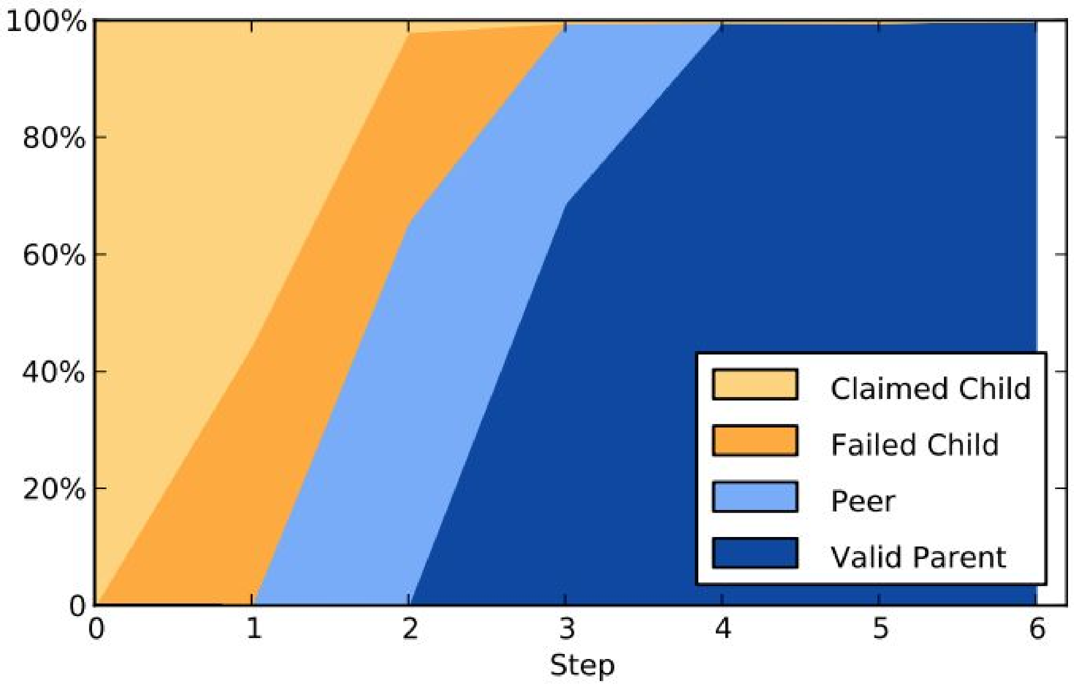

Fig. 4. Breakdown of edges in the frontier for a sample search on `kron27` (Kronecker generated 128M vertices with 2B undirected edges) on the 16-core system.

The Improvement

- Easy to parallelize by partitioning vertices, no longer requires atomic operations
- Requires inverse graph, with large memory overhead, in case of directed graphs

```
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

Hybrid Approach

- Bottom-up is most effective with large frontier
- Bottom-up requires checking all vertices to see if they remain unvisited, so a lot of unnecessary work if the graph is multiple components
- Best to switch techniques

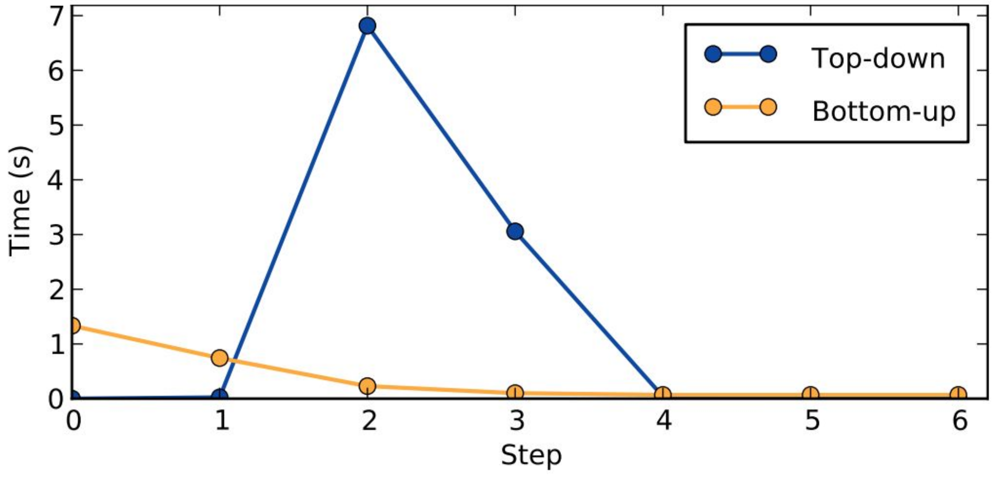

Fig. 6. Sample search on kron27 (Kronecker 128M vertices with 2B undirected edges) on the 16-core system.

Heuristic Thresholds

- switch to Bottom-Up
- switch to Top-down

m_f = edges adjacent to frontier
m_u = edges adjacent to unvisited nodes
n_f = vertices in frontier
n = total vertices

alpha compensates for bottom-up finishing before examining all of m_u

Beta compensates for

$$m_f > \frac{m_u}{\alpha} = C_{TB}$$

$$n_f < \frac{n}{\beta} = C_{BT}$$

Optimizing Alpha

- Chose alpha = 14
- Much larger does not impact which step transition occurs on

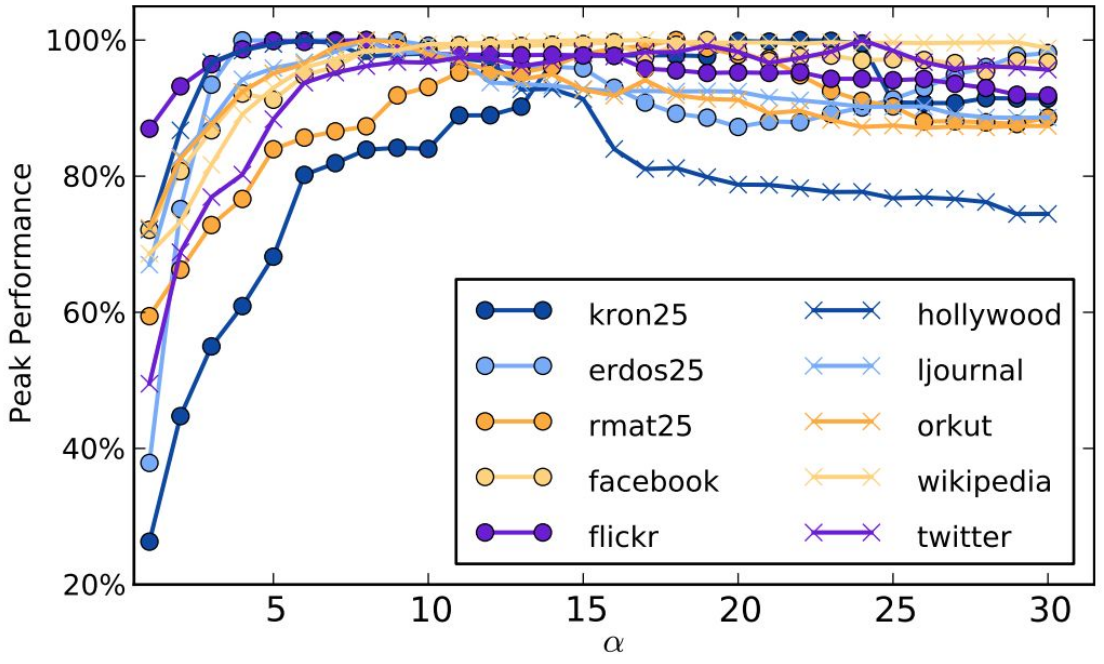

Fig. 8. Performance of *hybrid-heuristic* on each graph relative to its best on that graph for the range of α examined.

Optimizing Beta

- Chose beta = 24
- Minor variance has little effect because switching back to Top-Down at the very end is inconsequential because majority of work has already been done

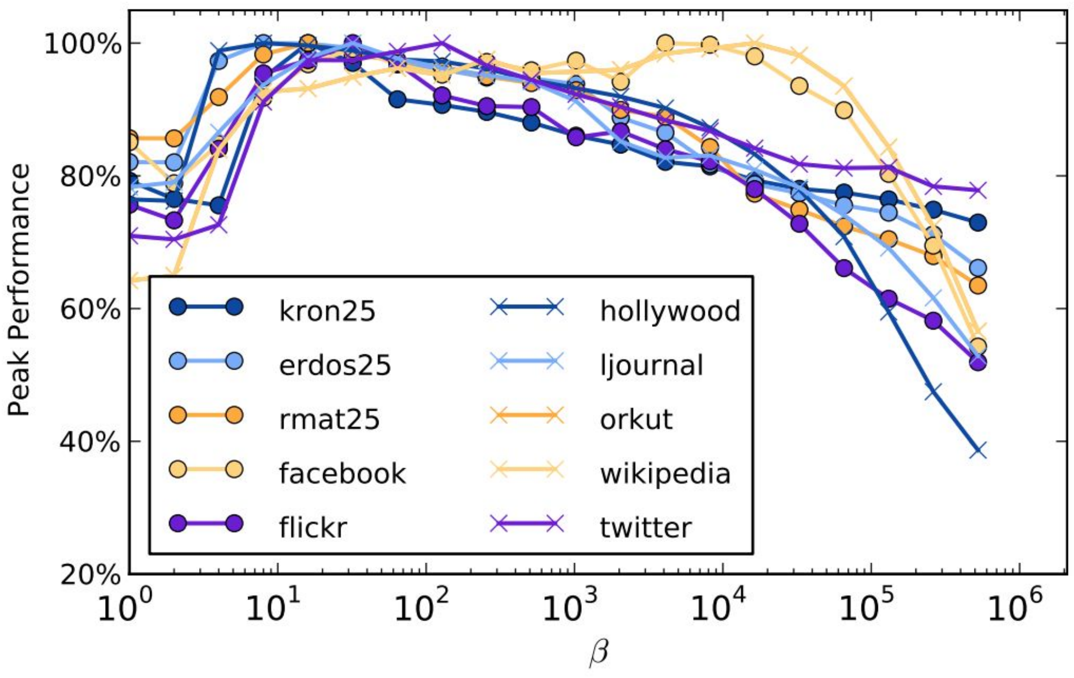

Fig. 9. Performance of *hybrid-heuristic* on each graph relative to its best on that graph for the range of β examined.

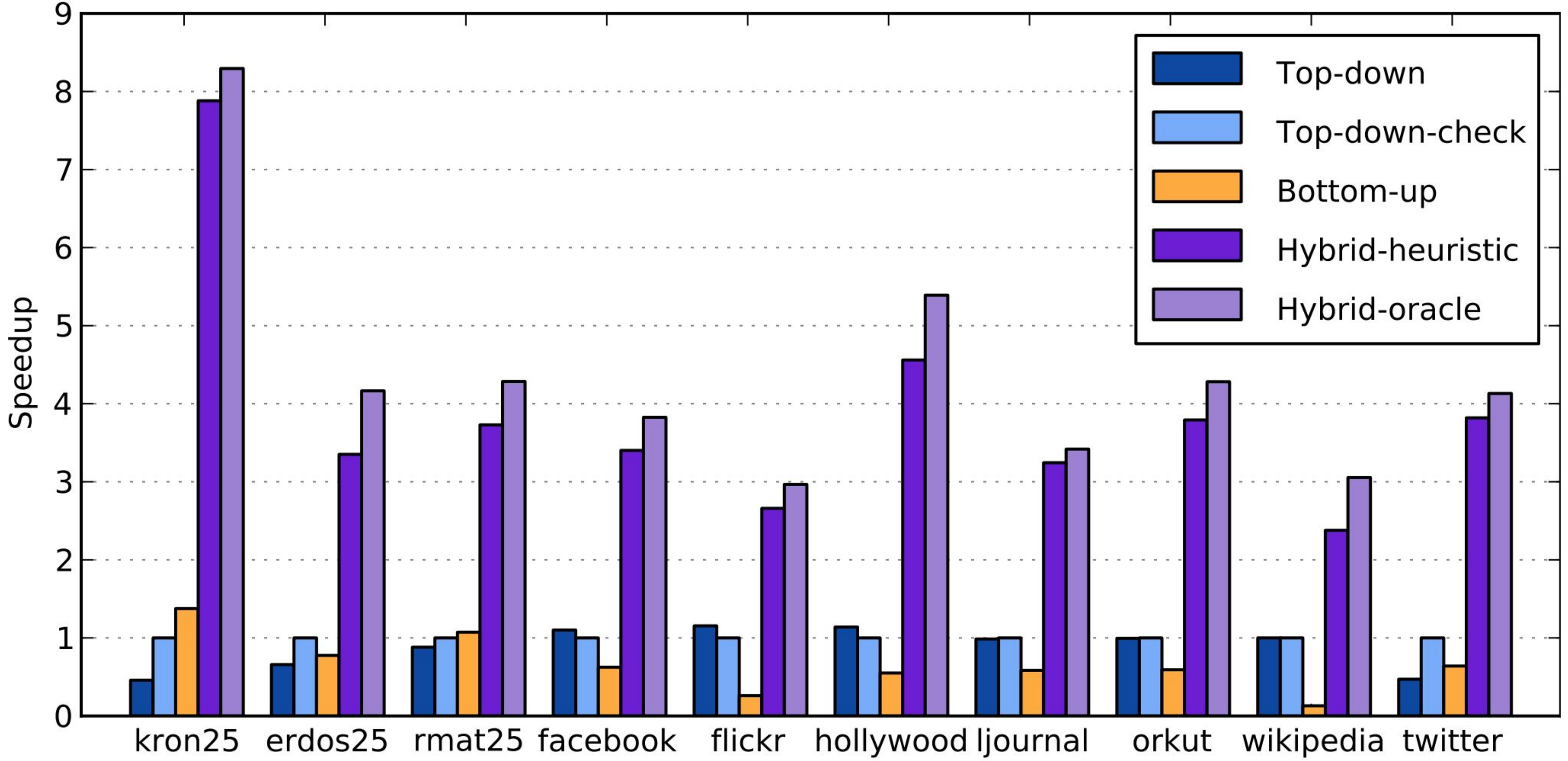

Fig. 10. Speedups on the 16-core machine relative to *Top-down-check*.

Effect of Degree

- Measured in terms of effective number of edges traversed per second
- Dense graphs benefit greatly

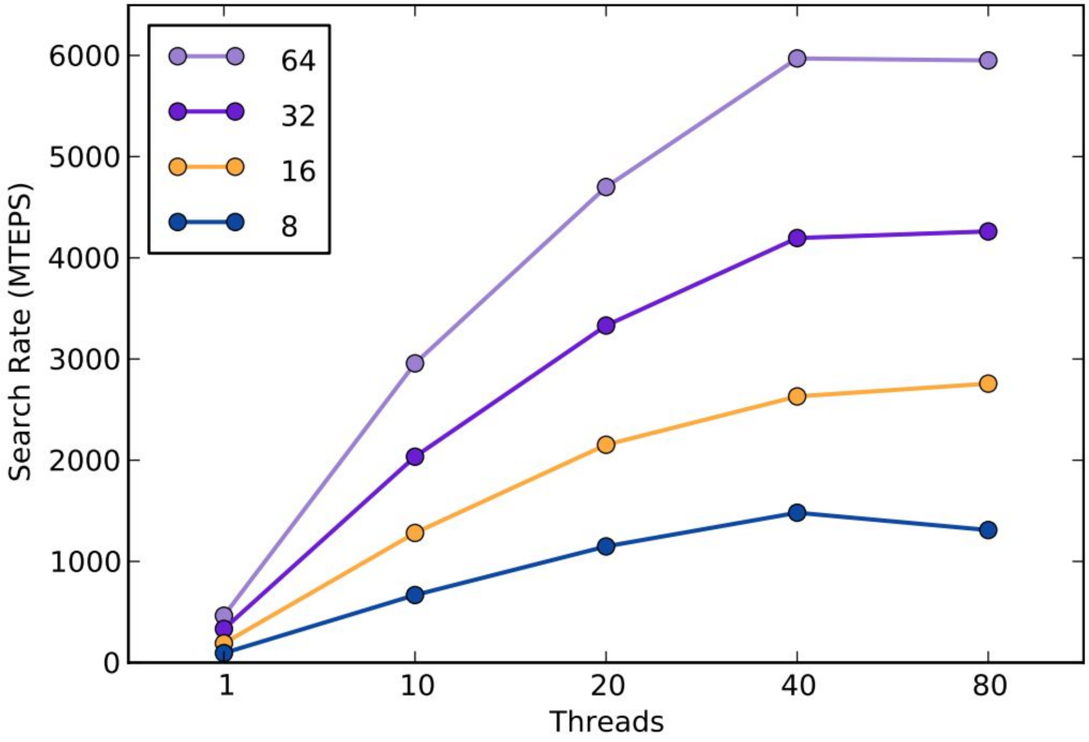

Fig. 14. Parallel scaling of *Hybrid-heuristic* on the 40-core system for an RMAT graph with 16M vertices and varied degree.

Related Works

- Efficient Breadth-First Search on the Cell/BE Processor[1]
- A Scalable Distributed Parallel Breadth-First Search Algorithm on BlueGene/L[2]
- Designing Multithreaded Algorithms for Breadth-First Search and st-connectivity on the Cray MTA-2[3]
- Topologically adaptive parallel breadth-first search on multicore processors[4]