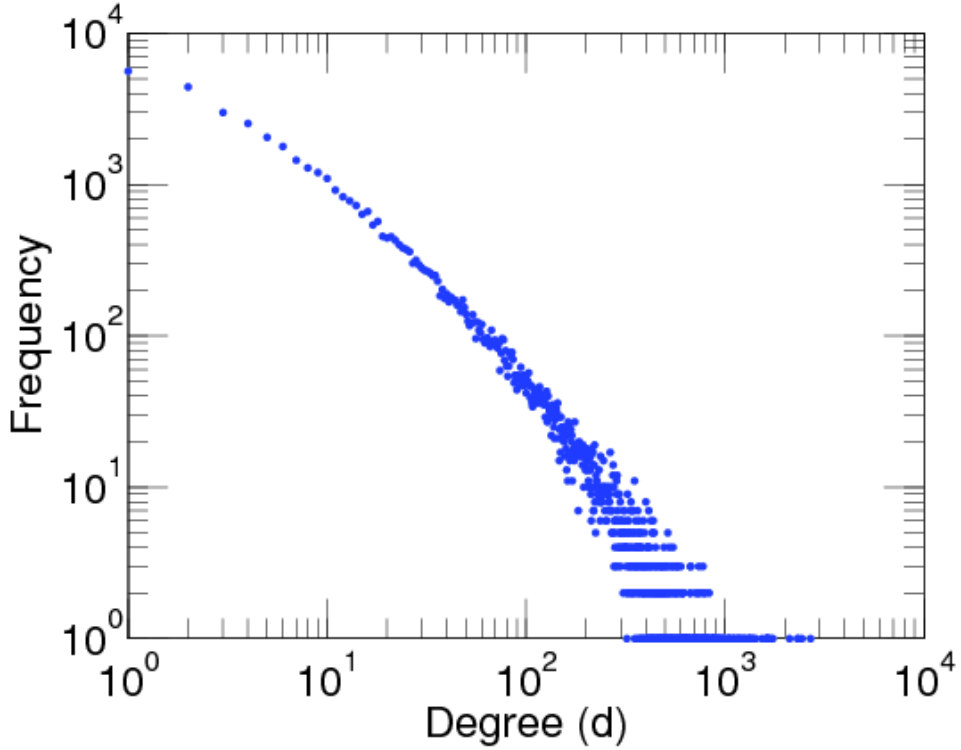
Direction Optimizing breadth-first search

Authors: Scott Beamer, Krste Asanović, David Patterson

Presenter: Patrick Insinger

Applied Breadth First Search

- BFS returns the shortest path for unweighted graphs, very useful for analyzing social networks
- Social network graphs are typically low-diameter and scale-free
  - Low-diameter (small-world) → the maximum distance between any two nodes is low
  - Scale-free → the degree distribution follows a power law



KONECT Facebook wall post degree distribution

Difficulties of Optimizing Breadth First Search

- Lack of spatial locality
- For large graphs, finding neighbors is essentially a random access
- Performance is memory-bound on individual machines, communication-bound on clusters
- Instead of trying to do additional work optimizing locality, this paper tries to do less work inspecting edges.

```
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

Understanding Edge Checks

- Paper has four edge check classifications
  - Valid Parent – neighbor at depth d-1 of a vertex at depth d
  - Peer – neighbor at same depth
  - Failed child – any neighbor at depth d+1 of a vertex at depth d, already visited
  - Claimed child – any neighbor at depth d+1 of a vertex at depth d, not visited
- Florentine Families Example

Understanding Edge Checks

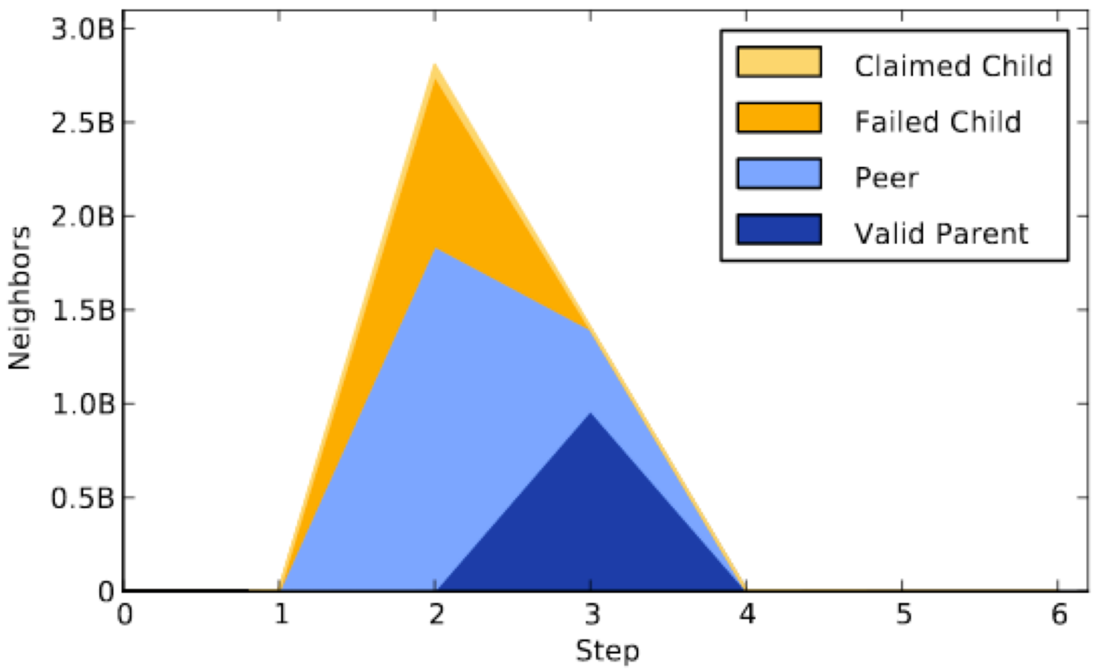


Fig. 3. Breakdown of edges in the frontier for a sample search on `kron27` (Kronecker generated 128M vertices with 2B undirected edges) on the 16-core system.

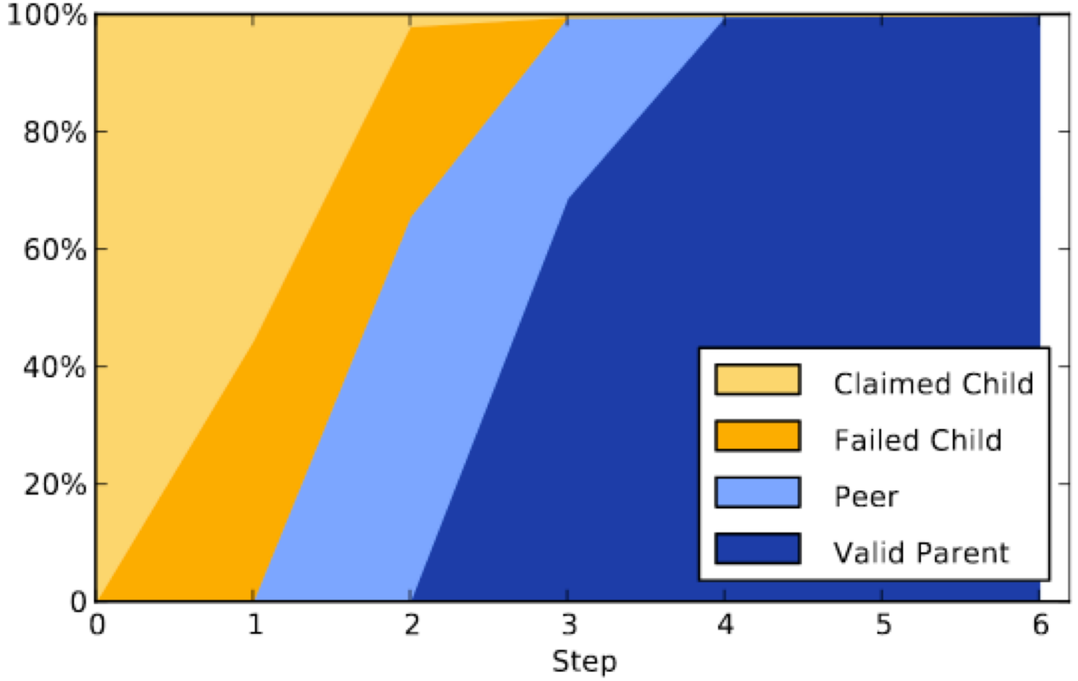


Fig. 4. Breakdown of edges in the frontier for a sample search on `kron27` (Kronecker generated 128M vertices with 2B undirected edges) on the 16-core system.

Idea: Bottom Up Search

"Instead of each vertex in the frontier attempting to become the parent of *all* of its neighbors, each unvisited vertex attempts to find *any* parent among its neighbors."

Idea: Bottom Up Search

- Works well when the frontier is large.
- Doesn't need mutual exclusion for parallelization!
- Needs frontier conversion:
  - Top-Down: FIFO-queue
  - Bottom-Up: bitmap

```
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 Algorithm

- Idea:
  - Small frontier → top-down
  - Large frontier → bottom-up

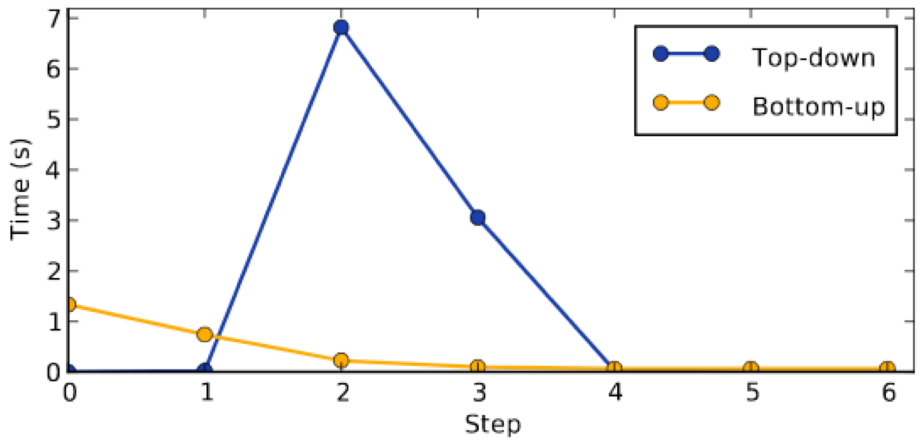


Fig. 6. Sample search on kron27 (Kronecker 128M vertices with 2B undirected edges) on the 16-core system.

Hybrid Algorithm

- State machine determines algorithm
- Switch from top-down to bottom-up when # of edges to explore from frontier > 1/alpha * (# of edges to explore from unvisited nodes)
- Switch from bottom-up to top-down when # of vertices in frontier < 1/beta * (# of vertices)

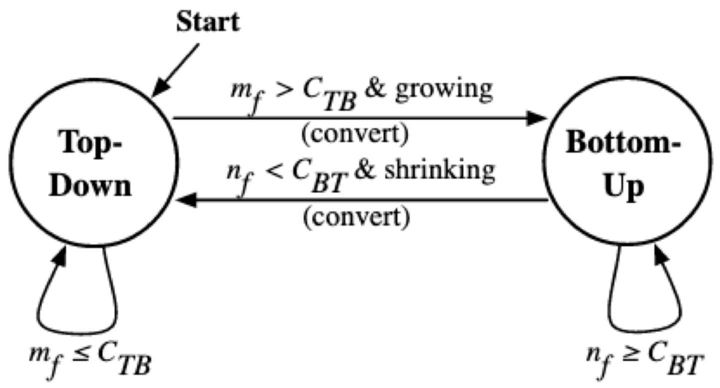


Fig. 7. Control algorithm for hybrid algorithm. (convert) indicates the frontier must be converted from a queue to a bitmap or vice versa between the steps. Growing and shrinking refer to the frontier size, and although they are typically redundant, their inclusion yields a speedup of about 10%.

Tuning

- Greedy tuning, alpha (14) then beta (24)

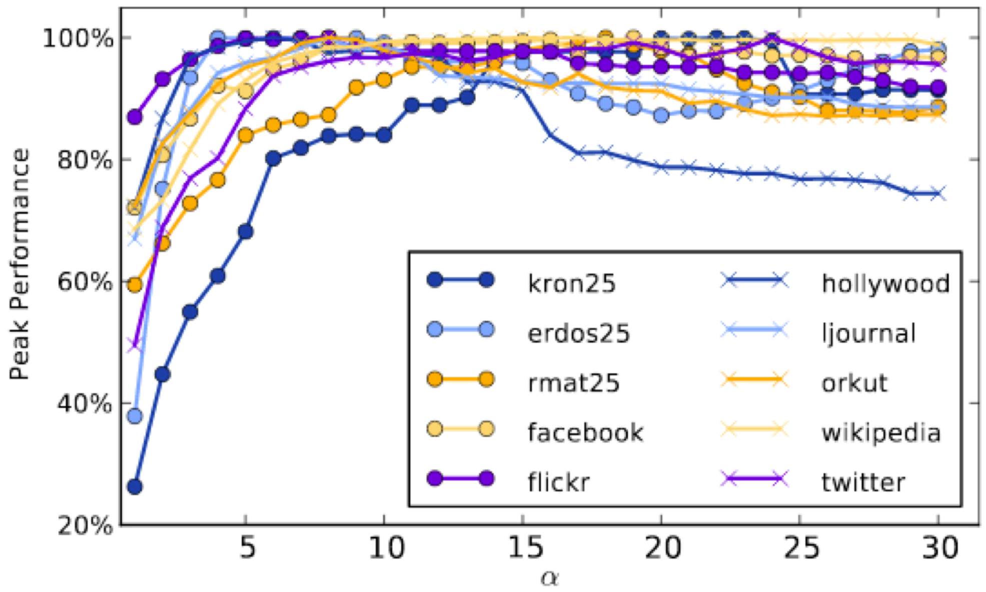


Fig. 8. Performance of *hybrid-heuristic* on each graph relative to its best on that graph for the range of α examined.

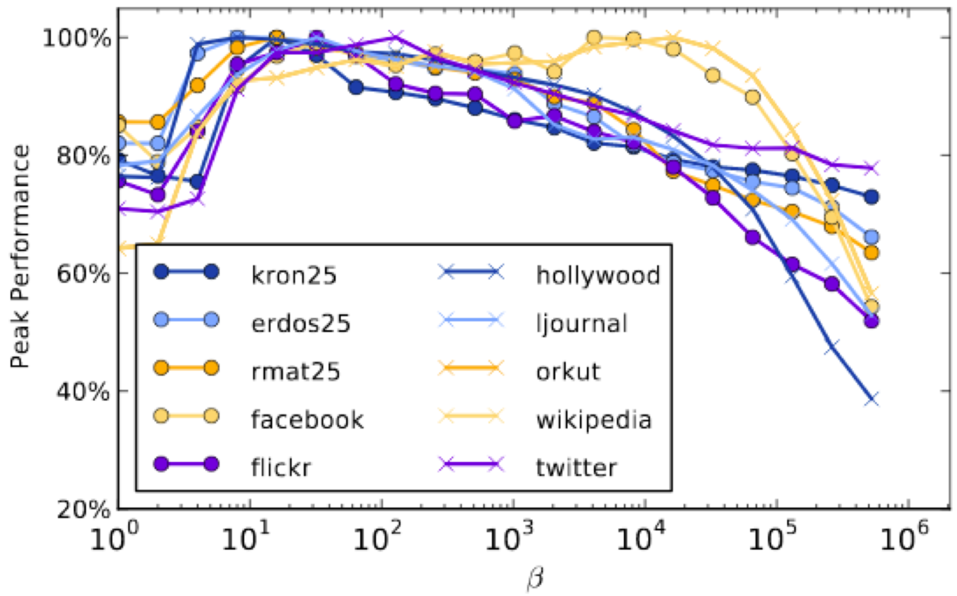


Fig. 9. Performance of *hybrid-heuristic* on each graph relative to its best on that graph for the range of β examined.

Hybrid Control Evaluation

- Empirically, within 25% of optimal

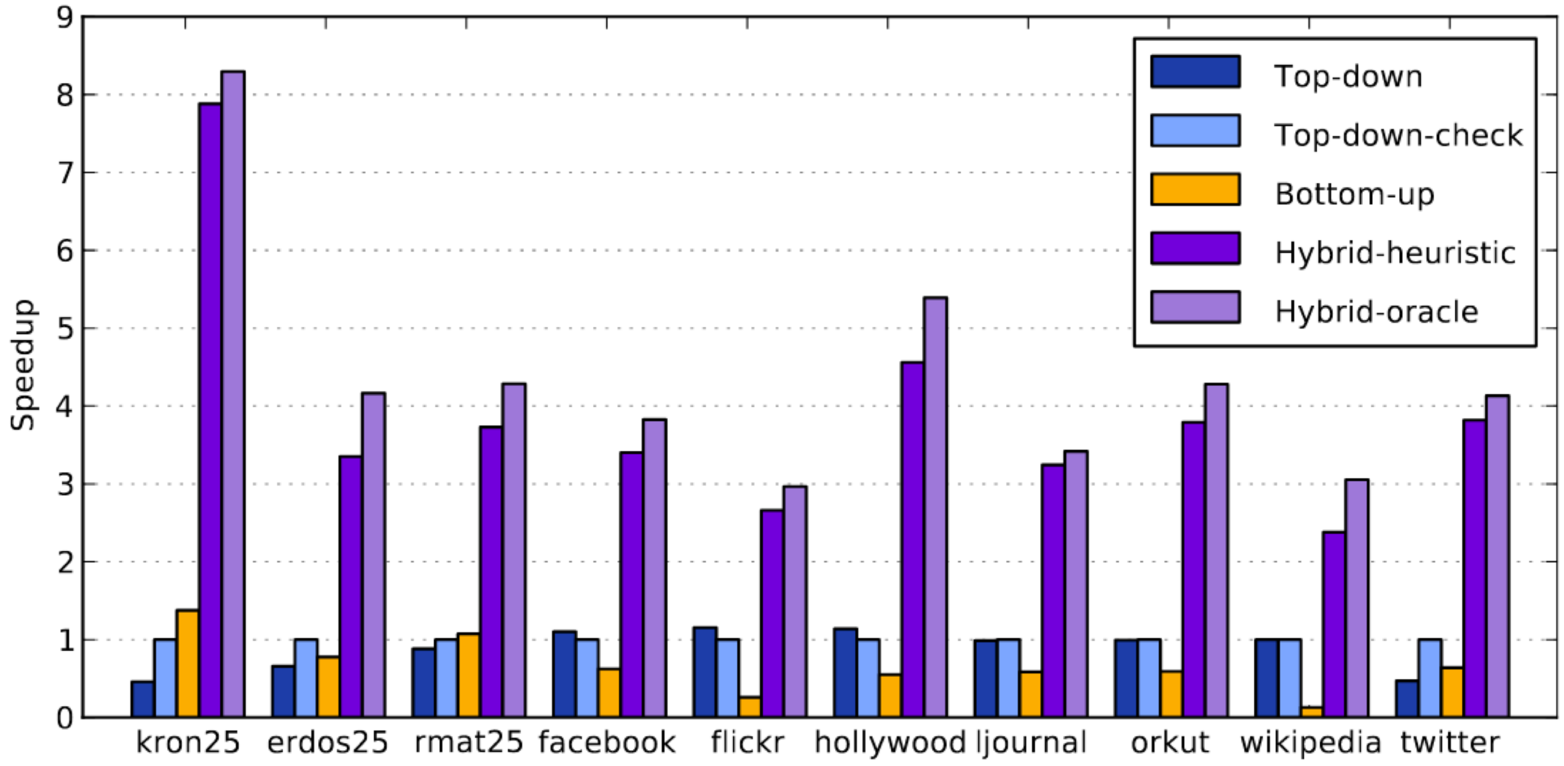


Fig. 10. Speedups on the 16-core machine relative to *Top-down-check*.

Performance Evaluation

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.

TEPS – Edges traversed per second = (# of edges in graph)/(runtime)

Time breakdown

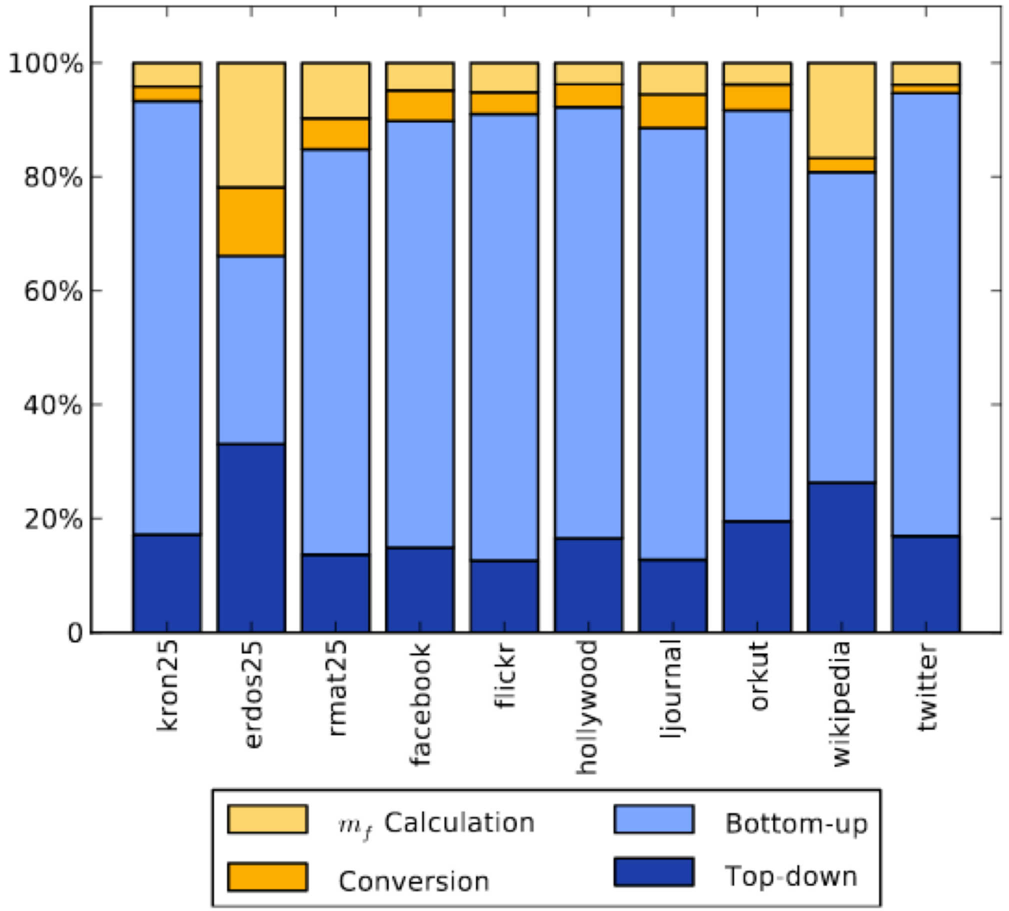


Fig. 12. Breakdown of time spent per search.

Parallel Performance Evaluation

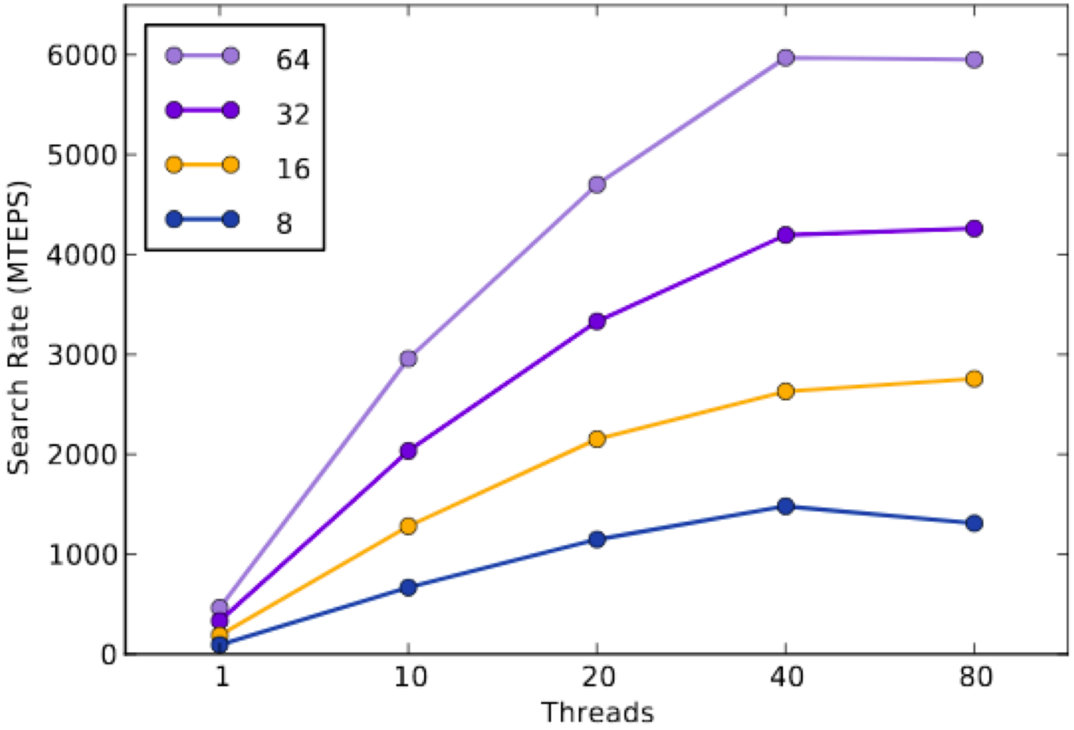


Fig. 14. Parallel scaling of *Hybrid-heuristic* on the 40-core system for an RMAT graph with 16M vertices and varied degree.