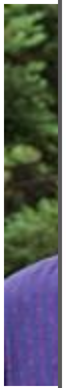
Direction-optimizing Breadth-First Search

Paper Review // 6.5060 Algorithm Engineering

Direction-Optimizing Breadth-First Search (IEEE, 2012)



Professor David Patterson sets the APA RAW California State Record

Posted on April 20, 2013 by Boban Zarkovich

On Saturday April 20 I participated in the American Powerlifting Association California Championships in Sacramento.

It was a "RAW" competition, which means no gadgets to help you lift the weights (no supportive suits).

I thought I could just enter the bench press, but it was a three-event contest so I had to also learn how to do a squat and a dead lift while competing (photos included).

You get 3 chances per lift, and if you fail all 3 you are out of the contest.

My old wrestling buddy Rich Byrne showed up after the first lift failure to give me just-in-time suggestions.

I failed two each round to make it exciting, but got one right each round to make it to the final lift, where I only failed once.

I ending up winning my age group and weight class (see photo), although I surely failed more lifts than anyone else at the event.

The total for all three lifts was 620 pounds, which for my age and which for my age and weight group (inadvertently) set the APA RAW California State Record.

Some background on parallel algorithms for BFS


Designing Multithreaded Algorithms for Breadth-First Search and
st-connectivity on the Cray MTA-2
David A. Bader Kamesh Madduri
College of Computing
Georgia Institute of Technology, Atlanta GA 30332
Abstract
1 Introduction

Designing Multithreaded Algorithms for Breadth-First Search and st-connectivity on the Cray MTA-2
Bader and Madduri

1. All vertices at a given *level* in the graph can be processed simultaneously, instead of just picking the vertex at the head of the queue (step 7 in Alg. 1)
2. The adjacencies of each vertex can be inspected in parallel (step 9 in Alg. 1).

Some background on parallel algorithms for BFS

Designing Multithreaded Algorithms for Breadth-First Search and *st*-connectivity on the Cray MTA-2

David A. Bader Kamesh Madduri
College of Computing
Georgia Institute of Technology, Atlanta GA 30332

Designing Multithreaded Algorithms for Breadth-First Search and st-connectivity on the Cray MTA-2
Bader and Madduri

Input: $G(V, E)$, source vertex s

Output: Array $d[1..n]$ with $d[v]$ holding the length of the shortest path from s to $v \in V$, assuming unit-weight edges

1 **for** *all* $v \in V$ ***in parallel*** **do**
2   $d[v] \leftarrow -1$;
3 $d[s] \leftarrow 0$;
4 $Q \leftarrow \phi$;
5 *Enqueue* $s \leftarrow Q$;
6 **while** $Q \neq \phi$ **do**
7   **for** *all* $u \in Q$ ***in parallel*** **do**
8   *Delete* $u \leftarrow Q$;
9   **for** *each* v *adjacent to* u ***in parallel*** **do**
10   **if** $d[v] = -1$ **then**
11         $d[v] \leftarrow d[u] + 1$;
12   *Enqueue* $v \leftarrow Q$;

Algorithm 1: Level-synchronized Parallel BFS

Initialize [-1,...,-1] parent array

Put source into queue Q

While Q not empty

For each vertex in queue (**parallel**)

For each neighbor leading out of queue (**parallel**), if valid child, add to queue

1. All vertices at a given *level* in the graph can be processed simultaneously, instead of just picking the vertex at the head of the queue (step 7 in Alg. 1)
2. The adjacencies of each vertex can be inspected in parallel (step 9 in Alg. 1).

Designing parallel BFS to scale on massively parallel architectures (Cray MTA-2)

- Every vertex and its neighbours at subsequent levels of the graph is visited *simultaneously* (no level-level dependency)
- Main contribution is mapping onto hardware-specific primitives (#pragma) (not shown here but in paper)

Challenges in designing parallel algorithms for BFS

- Maintain a set of frontier values
- **Parallelism:** collect all *next* frontier values in parallel (must remove duplicates)

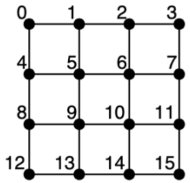
Challenges

- Inefficiencies: in worst case, every m edge is *always* visited; every n node is visited (O(n+m))
- Standard (TD) BFS almost always takes worst case time

ALGORITHM: BFS(s, G)

```
FRONT := [s]
TREE := distribute(−1, |G|)
TREE[s] := s
while (|FRONT| ≠ 0)
   E := flatten({{(u, v) : u ∈ G[v]} : v ∈ FRONT})
   E′ := {(u, v) ∈ E | TREE[u] = −1}
   TREE := TREE ← E′
   FRONT := {u : (u, v) ∈ E′ | v = TREE[u]}
return TREE
```

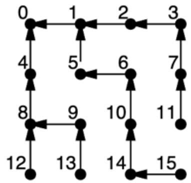
Parallel Algorithms
Blelloch & Maggs



(a)

Step	Frontier
0	[0]
1	[1, 4]
2	[2, 5, 8]
3	[3, 6, 9, 12]
5	[7, 10, 13]
6	[11, 14]
7	[15]

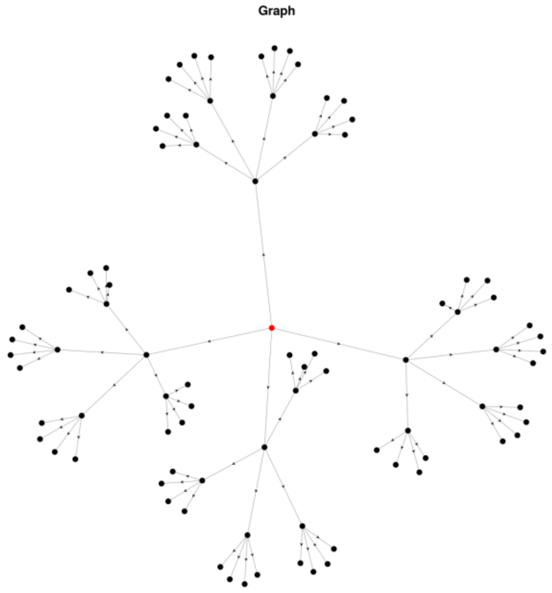
(b)



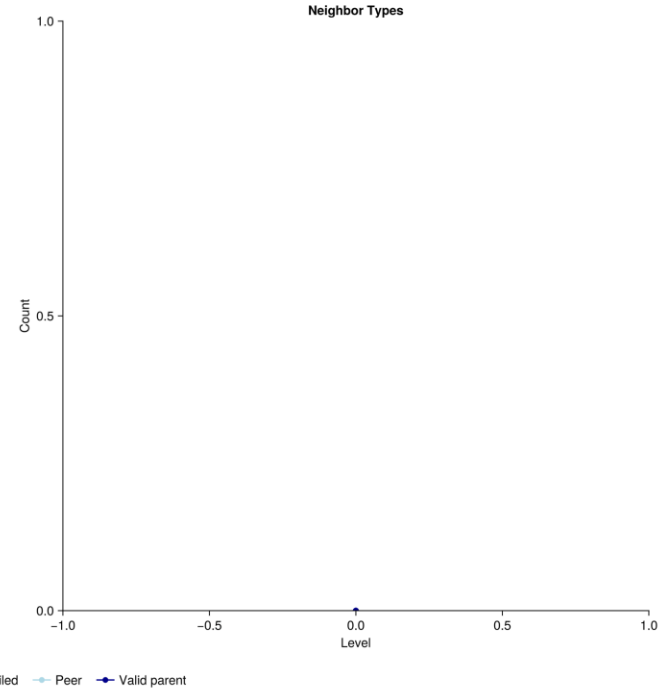
(c)

Figure 8: Example of Parallel Breadth First Search. (a) A graph G. (b) The frontier at each step of the BFS of G with $s = 0$. (c) A BFS tree.

Top-down BFS K-ary Tree Graph (low-degree, high-depth)



Own animation, K-ary tree graph




```
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
```

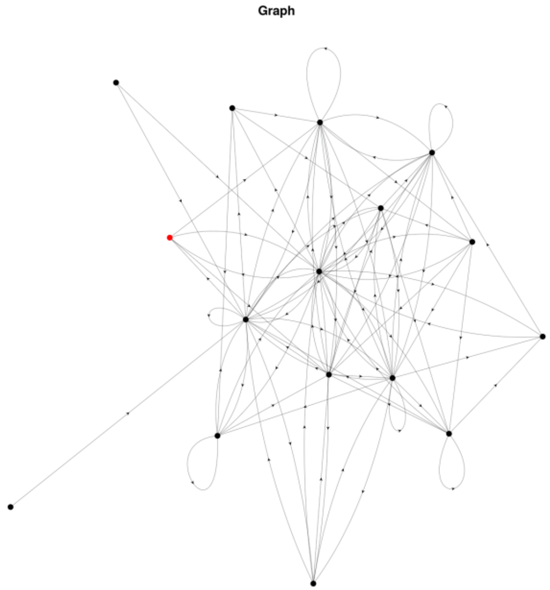
Fig. 1. Conventional BFS Algorithm

```
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
```

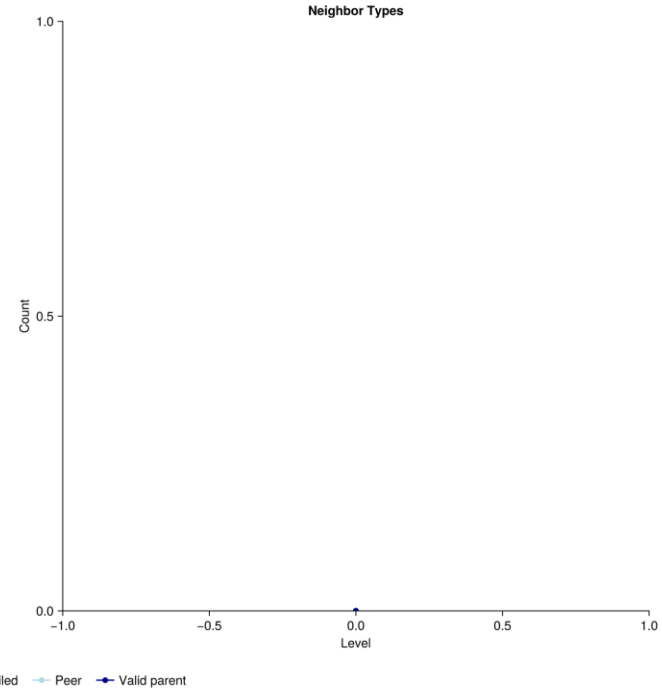
Fig. 2. Single Step of Top-Down Approach

Beamer et al.

Top-down BFS Kronecker Graph (high-degree, low-depth)



Own animation, Kronecker graph




```
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
```

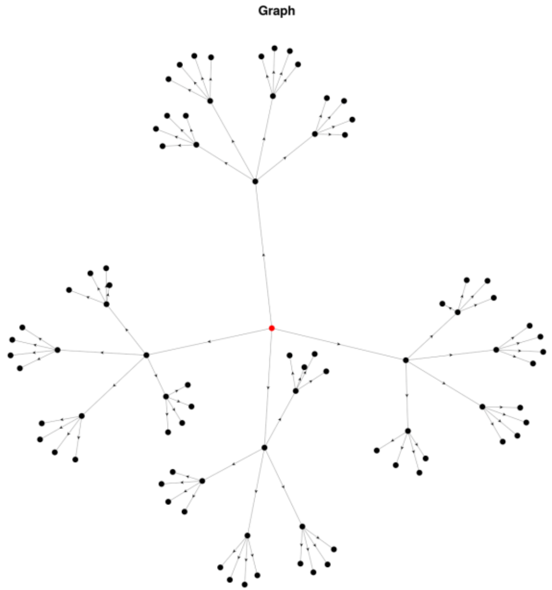
Fig. 1. Conventional BFS Algorithm

```
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
```

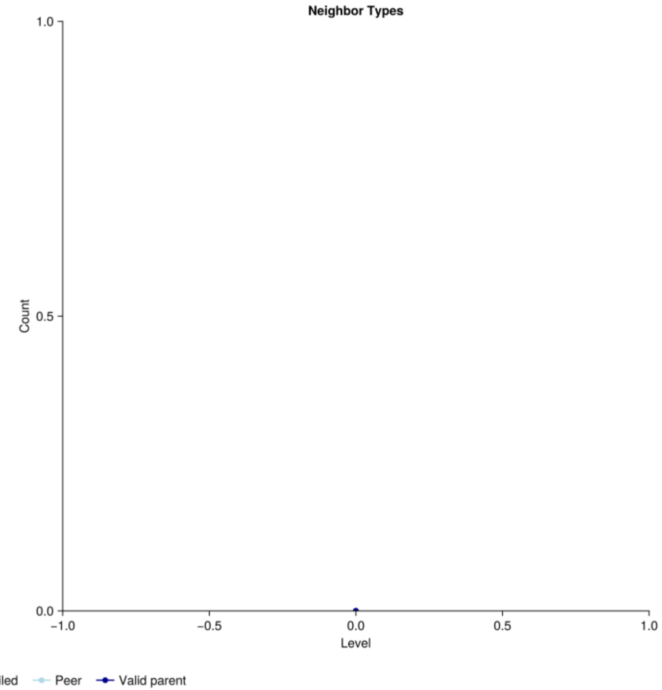
Fig. 2. Single Step of Top-Down Approach

Beamer et al.

Bottom-up BFS K-ary Tree Graph (low-degree, high-depth)



Own animation, K-ary tree graph




```
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

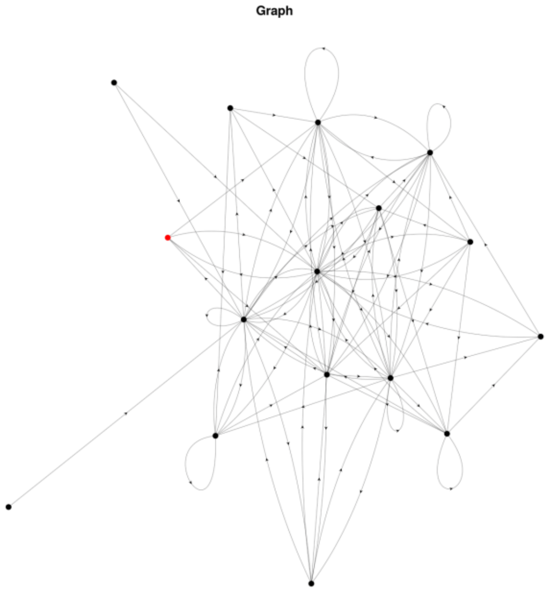
```
function bottom-up-step(vertices, frontier, next, parents)
    for v ∈ vertices do
        if parents[v] = -1 then
            for n ∈ neighbors[v] do
                if n ∈ frontier then
                    parents[v] ← n
                    next ← next ∪ {v}
                    break
         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

Fig. 5. Single Step of Bottom-Up Approach

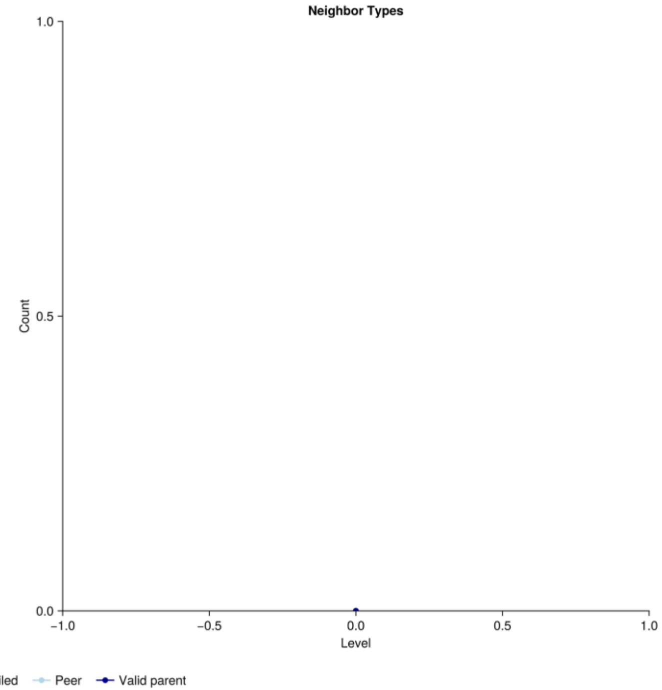
Beamer et al.

Bottom-up BFS Kronecker Graph (high-degree, low-depth)

Watch the pink edges!



Own animation, Kronecker graph




```
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 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

Beamer et al.

Parallelization is possible for both TD and BU

```
function top-down-step(vertices, frontier, next, parents)
    for v ∈ frontier do                 in parallel?!!
        for n ∈ neighbors[v] do         in parallel?!!
            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

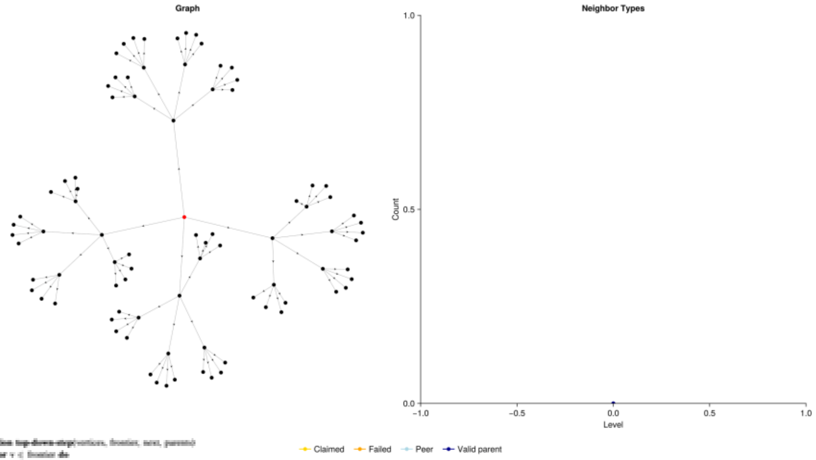
```
function bottom-up-step(vertices, frontier, next, parents)
    for v ∈ vertices do                 in parallel?!!
        if parents[v] = -1 then
            for n ∈ neighbors[v] do     in parallel?!!
                if n ∈ frontier then
                    parents[v] ← n
                    next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

Fig. 5. Single Step of Bottom-Up Approach

Beamer et al.

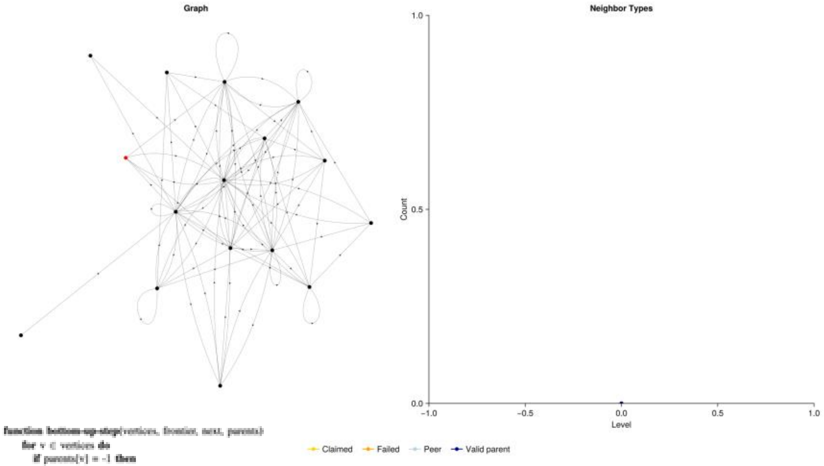
Searching over all neighbours simultaneously*

* only outgoing neighbours in TD, and ingoing neighbours in BU for directional graphs




```
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



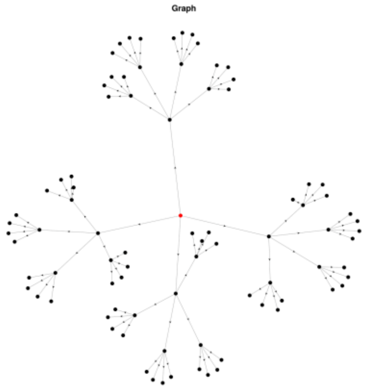

```
function bottom-up-step(vertices, frontier, next, parents)
  for v ∈ vertices do
    if parents[v] = -1 then
      for n ∈ neighbors[v] do
        if n ∈ frontier then
          parents[v] ← n
          next ← next ∪ {v}
          break
        end if
      end for
    end if
  end for
```

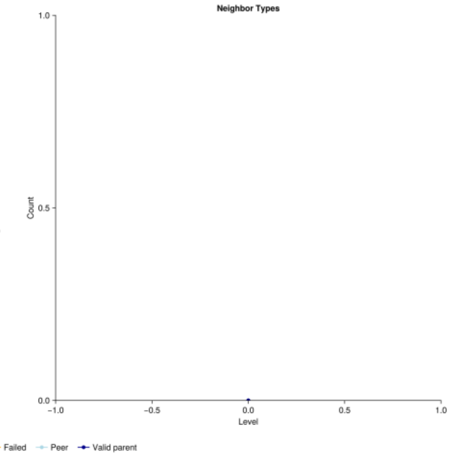
Fig. 3. Single Step of Bottom-Up Approach

Beamer et al.

Searching over all vertices and neighbours simultaneously*

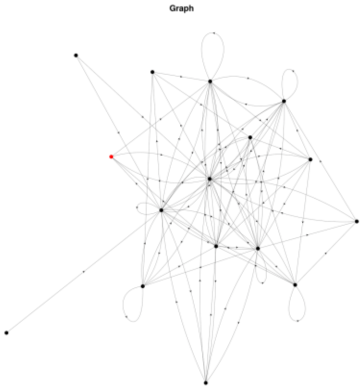
* only outgoing neighbours in TD, and ingoing neighbours in BU for directional graphs

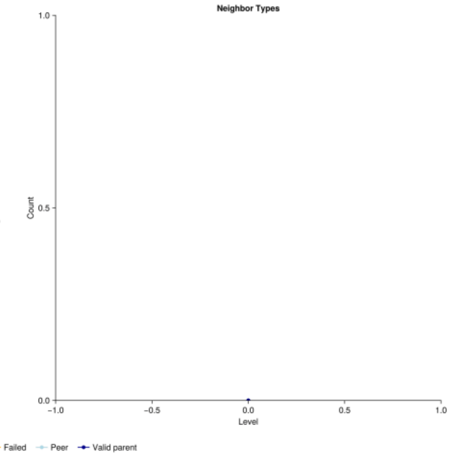





```
function top-down-step(vertices, frontier, next, parents)
  for v ∈ frontier do
    for n ∈ neighbors[v] do
      if parents[n] = -1 then
        parents[n] ← v
        next ← next ∪ {n}
      end if
    end for
  end for
```

Fig. 2. Single Step of Top-Down Approach






```
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. 3. Single Step of Bottom-Up Approach

Beamer et al.

Directional BFS proposes a hybrid approach

Start with top-down

Better for small frontier

Where:

n_f = *n vertices in frontier*

m_f = *m edges to check from frontier*

m_u = *m edges to check from unexplored vertices*

alpha = tuning parameter to adjust from upper bound m_u

beta = tuning parameter to adjust from upper bound m

Directional BFS proposes a hybrid approach

Start with top-down

Better for small frontier

↓

At each level

Where:

n_f = *n vertices in frontier*

m_f = *m edges to check from frontier*

m_u = *m edges to check from unexplored vertices*

alpha = tuning parameter to adjust from upper bound m_u

beta = tuning parameter to adjust from upper bound m

Directional BFS proposes a hybrid approach

Start with top-down

Better for small frontier

At each level

Switch to bottom-up if:

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

Frontier is large relative to
total count of unexplored edges

Where:
n_f = n vertices in frontier
m_f = m edges to check from frontier
m_u = m edges to check from unexplored vertices
alpha = tuning parameter to adjust from upper bound m_u
beta = tuning parameter to adjust from upper bound m

Directional BFS proposes a hybrid approach

Start with top-down

Better for small frontier

↓

At each level

Switch to bottom-up if:

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

Frontier is large relative to total count of unexplored edges

Switch to top-down if:

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

Frontier is small relative to the total number of nodes

Where:

n_f = n vertices in frontier
m_f = m edges to check from frontier
m_u = m edges to check from unexplored vertices
alpha = tuning parameter to adjust from upper bound m_u
beta = tuning parameter to adjust from upper bound m

Directional BFS proposes a hybrid approach

Start with top-down

Better for small frontier

At each level

Switch to bottom-up if:

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

Frontier is large relative to total count of unexplored edges

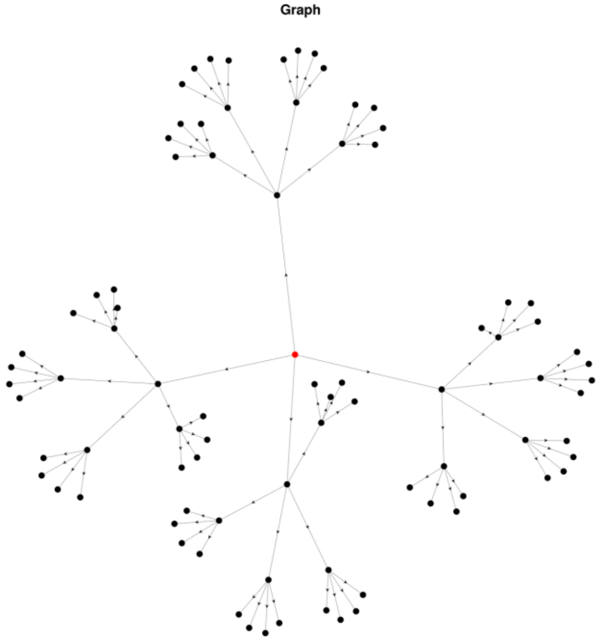
Switch to top-down if:

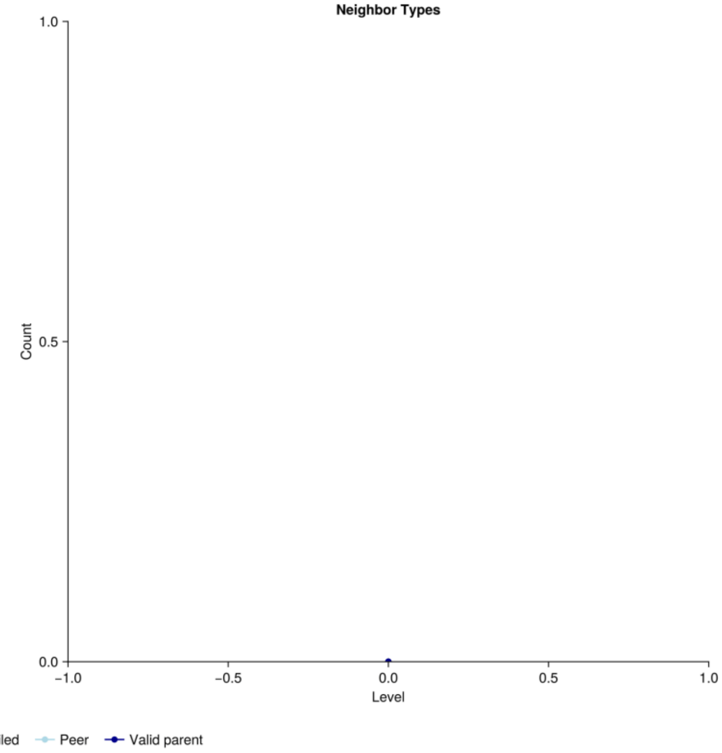
$$n_f < \frac{n}{\beta} = C_{BT}$$

Frontier is small relative to the total number of nodes

Where:
n_f = n vertices in frontier
m_f = m edges to check from frontier
m_u = m edges to check from unexplored vertices
alpha = tuning parameter to adjust from upper bound m_u
beta = tuning parameter to adjust from upper bound m

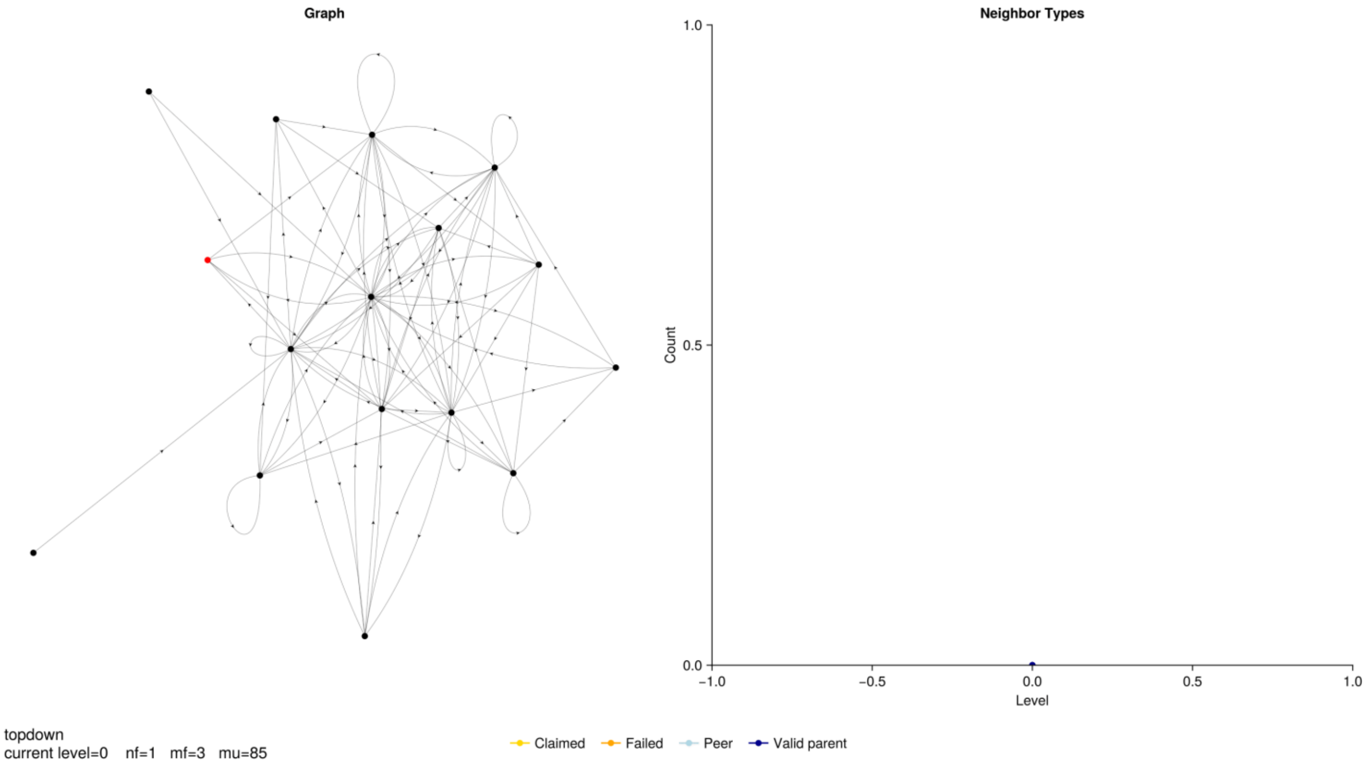
Hybrid approach for k-tree graph





topdown
current level=0 nf=1 mf=4 mu=80

Hybrid approach for Kronecker graph



Methodology

Graphs evaluated for experiments

Abbreviation	Graph	# Vertices (M)	# Edges (M)	Degree	Diameter	Directed	References
kron25	Kronecker	33.554	536.870	16.0	6	N	[14, 17]
erdos25	Erdős–Réyni (Uniform Random)	33.554	268.435	8.0	8	N	[3, 13]
rmat25	RMAT	33.554	268.435	8.0	9	Y	[3, 9]
facebook	Facebook Trace A	3.097	28.377	9.2	9	N	[26]
flickr	Flickr Follow Links	1.861	22.614	12.2	15	Y	[21]
hollywood	Hollywood Movie Actor Network	1.140	57.516	50.5	10	N	[6, 7, 12, 23]
ljournal	LiveJournal Social Network	5.363	79.023	14.7	44	Y	[21]
orkut	Orkut Social Network	3.073	223.534	72.8	7	N	[21]
wikipedia	Wikipedia Links	5.717	130.160	22.8	282	Y	[25]
twitter	Twitter User Follow Links	61.578	1,468.365	23.8	15	Y	[16]

TABLE I
GRAPHS USED FOR EVALUATION

Three systems: 8-core, 16-core, 40-core

Methodology

Graphs evaluated for experiments

Abbreviation	Graph	# Vertices (M)	# Edges (M)	Degree	Diameter	Directed	References
kron25	Kronecker	33.554	536.870	16.0	6	N	[14, 17]
erdos25	Erdős–Réyni (Uniform Random)	33.554	268.435	8.0	8	N	[3, 13]
rmat25	RMAT	33.554	268.435	8.0	9	Y	[3, 9]
facebook	Facebook Trace A	3.097	28.377	9.2	9	N	[26]
flickr	Flickr Follow Links	1.861	22.614	12.2	15	Y	[21]
hollywood	Hollywood Movie Actor Network	1.140	57.516	50.5	10	N	[6, 7, 12, 23]
ljournal	LiveJournal Social Network	5.363	79.023	14.7	44	Y	[21]
orkut	Orkut Social Network	3.073	223.534	72.8	7	N	[21]
wikipedia	Wikipedia Links	5.717	130.160	22.8	282	Y	[25]
twitter	Twitter User Follow Links	61.578	1,468.365	23.8	15	Y	[16]

TABLE I
GRAPHS USED FOR EVALUATION

Three systems: 8-core, 16-core, 40-core

To evaluate parallel scalability up to 80 threads

Most representative of compute nodes for clusters

Methodology

Graphs evaluated for experiments

Abbreviation	Graph	# Vertices (M)	# Edges (M)	Degree	Diameter	Directed	References
kron25	Kronecker	33.554	536.870	16.0	6	N	[14, 17]
erdos25	Erdős–Réyni (Uniform Random)	33.554	268.435	8.0	8	N	[3, 13]
rmat25	RMAT	33.554	268.435	8.0	9	Y	[3, 9]
facebook	Facebook Trace A	3.097	28.377	9.2	9	N	[26]
flickr	Flickr Follow Links	1.861	22.614	12.2	15	Y	[21]
hollywood	Hollywood Movie Actor Network	1.140	57.516	50.5	10	N	[6, 7, 12, 23]
ljournal	LiveJournal Social Network	5.363	79.023	14.7	44	Y	[21]
orkut	Orkut Social Network	3.073	223.534	72.8	7	N	[21]
wikipedia	Wikipedia Links	5.717	130.160	22.8	282	Y	[25]
twitter	Twitter User Follow Links	61.578	1,468.365	23.8	15	Y	[16]

TABLE I
GRAPHS USED FOR EVALUATION

Three systems: 8-core, 16-core, 40-core
Implementation: C++, OpenMP, CSR format

Experiment results: 2.4-7.6x speedup!

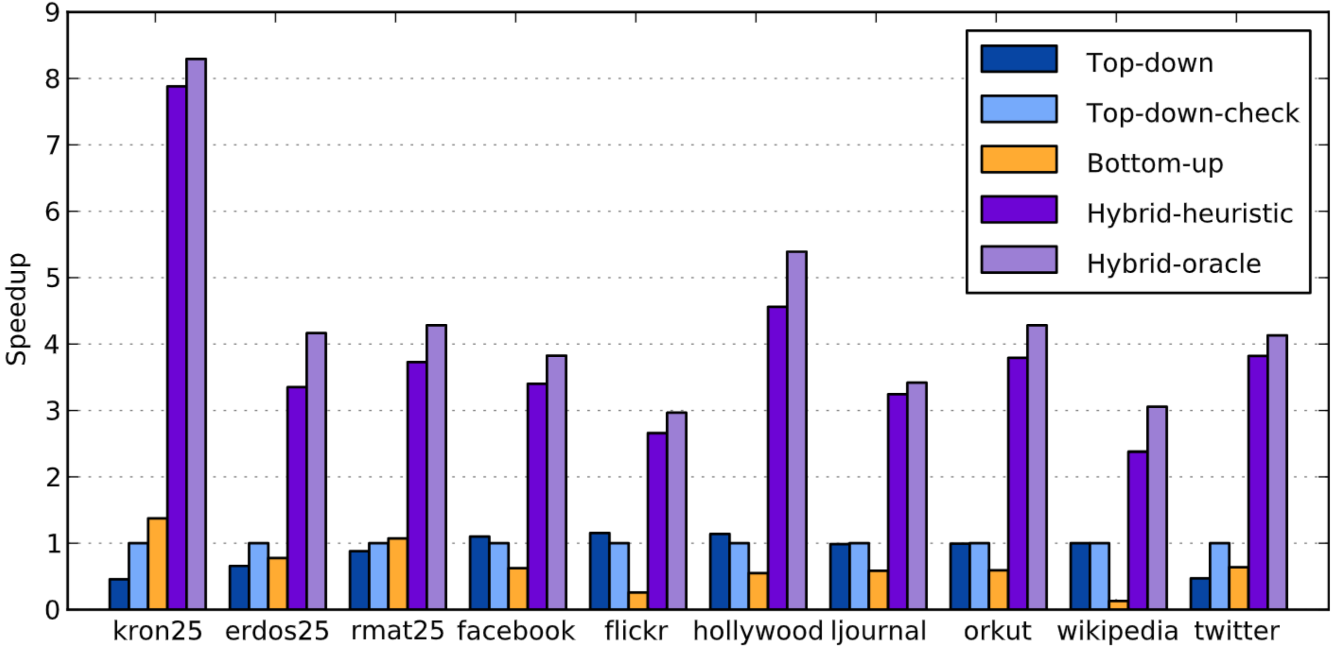


Fig. 10. Speedups on the 16-core machine relative to *Top-down-check*.

Bitmap check for completed nodes

Picks best switch locations across all possible iterations

Results interpretation

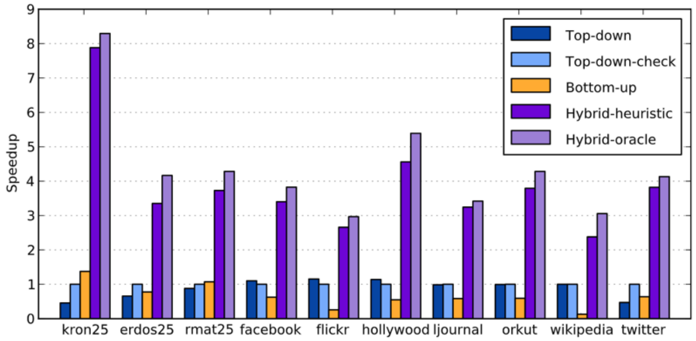


Fig. 10. Speedups on the 16-core machine relative to *Top-down-check*.

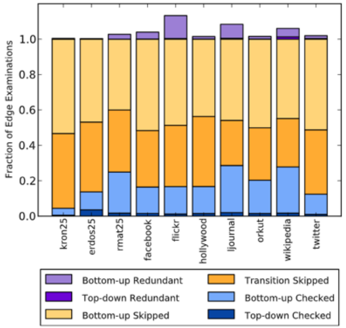


Fig. 11. Breakdown of edge examinations.

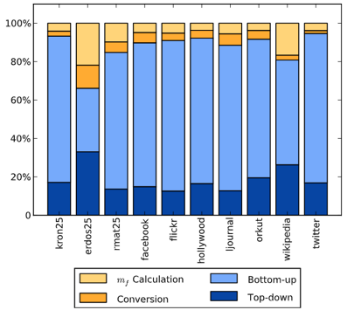


Fig. 12. Breakdown of time spent per search.

Why?

- Modulating between TD and BU bypasses their respective weaknesses (TD checks all edges, BU checks all vertices)

Takeaways

- Least effective on higher effective diameter graphs (twitter, Wikipedia) so more edges *must* be checked
- Most time spent in BU (used for more iterations due to edge-skipping)
- Constant degree graphs (e.g. Erdos-Reyni) will have more top-down steps (slower frontier growth) i.e. benefits **mostly small-world graphs**

Next steps?

What are the issues with Bottom-Up?

- Need fast frontier membership tests… frontier is too large to store in each processor’s memory
- Checking for a parent *has* to be sequential

2d-graph partitioning

Distributed Memory Breadth-First Search Revisited: Enabling Bottom-Up Search (2013)
Beamer, Buluç, Asanović, Patterson

Systolic shifts (rhythmic flows of data through processors)
In top-down:

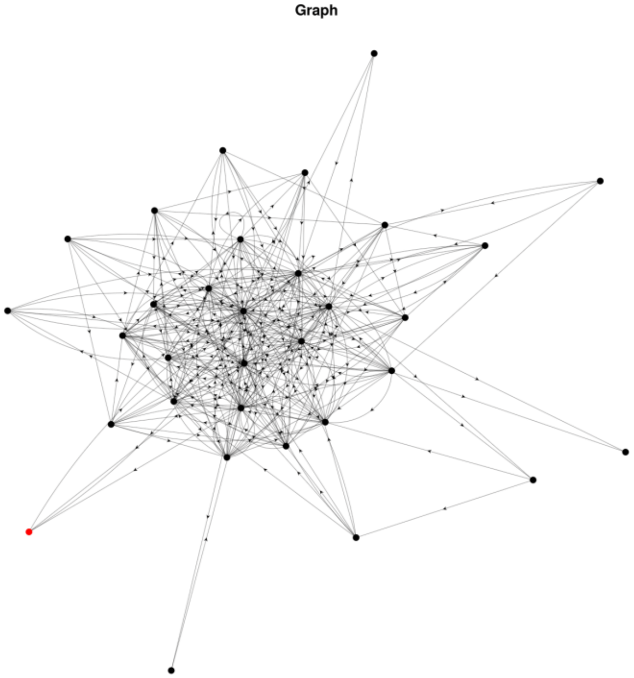
- Each processor ”proposes” parents for the partitioned subset of the graph
- The proposals are evaluated serially

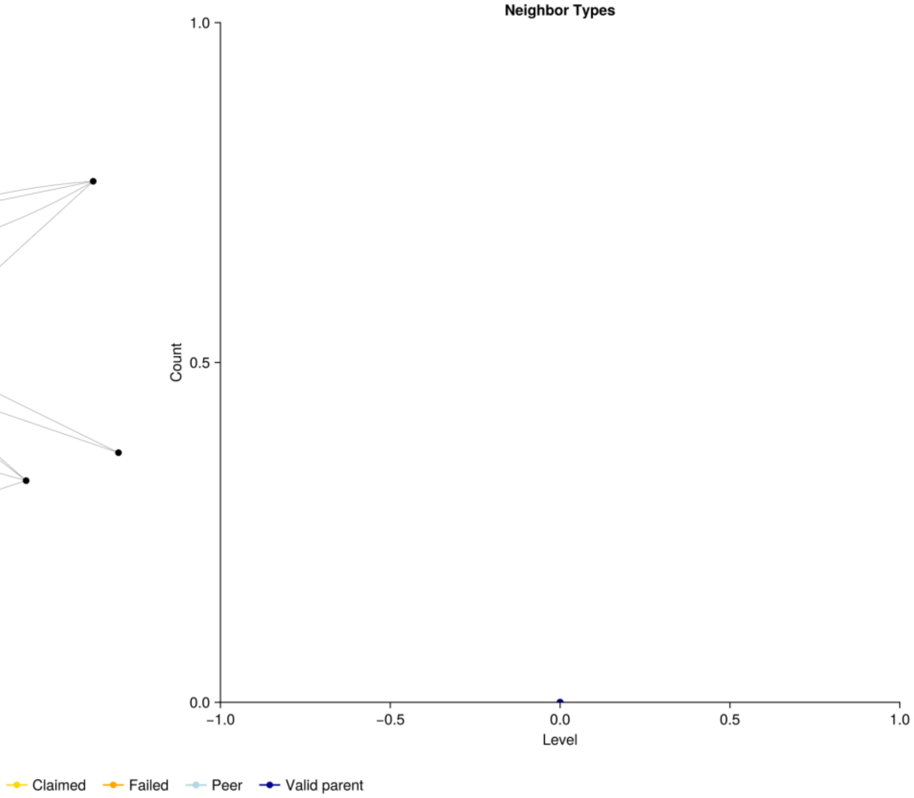
In bottom-up:

- Division of work into p_c substeps, during each of which $1/p_c$ vertices in the processor row are examined, after which the vertices are passed on to the next processor and the current processor accepts new ones
- If a parent is found, the next processor will skip over that vertex

Improve alpha/beta tuning heuristics (manual and a little arbitrary)

Bottom-up uses in-neighbours vs. out-neighbours used by top-down. Doubly represented directed graph? Maybe not very memory efficient.





topdown
current level=0 nf=1 mf=1 mu=311