

Direction-optimizing Breadth-First Search

Paper Review // 6.5060 Algorithm Engineering

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

Direction-Optimizing Breadth-First Search

Scott Branson Kostas Anagnostou David Patterson
Electrical Engineering and Computer Sciences Department
University of California, Berkeley
{sbranson, kostas, patt}@eecs.berkeley.edu

Abstract—Breadth-First Search is an important kernel used by many graph-processing applications. In many of these emerging applications of BFS, such as analyzing social networks, the input graphs are low-diameter and scale-free. We propose a hybrid approach that is advantageous for low-diameter graphs, which combines a conventional top-down algorithm along with a novel bottom-up algorithm. The bottom-up algorithm can dramatically reduce the number of edges examined, which in turn accelerates the search as a whole. On a multi-modal set of standard synthetic graphs and spanning of 2.4-4.6 on graphs from real social networks when compared to a strong baseline. We also typically double the performance of prior leading shared-memory (multicore and GPU) implementations.

1. INTRODUCTION

Graph algorithms are becoming increasingly important, with applications covering a wide range of scales. Warehouse-scale computers run graph algorithms that reason about vast amounts of data, with applications including analytics and recommendation systems [1], [2]. On mobile devices, graph algorithms are important components of navigation and machine-learning applications [3], [4].

Unfortunately, due to a lack of locality, graph applications are often memory-bound on shared-memory hardware or communication-bound on clusters. In particular, Breadth-First Search (BFS), an important building block in many other graph algorithms, has low computational locality, which exacerbates the lack of locality and results in low overall performance. To accelerate BFS, there has been significant prior work to change the algorithm and data structures, in some cases by adding additional computational work, to increase locality and boost overall performance [5], [6], [7], [8]. However, none of these previous schemes attempt to reduce the number of edges examined.

In this paper, we present a hybrid BFS algorithm that combines a conventional top-down approach with a novel bottom-up approach. By examining substantially fewer edges, the new algorithm achieves speedups of 3.3-7.8 on synthetic graphs and 2.4-4.6 on real social network graphs. In the top-down approach, nodes in the active frontier search for an unvisited child, while in our new bottom-up approach, unvisited nodes search for a parent in the active frontier. In general, the bottom-up approach will yield speedups when the active frontier is a substantial fraction of the total graph, which commonly occurs in small-world graphs such as social networks.

The bottom-up approach is not always advantageous, so we combine it with the conventional top-down approach, and use a simple heuristic to dynamically select the appropriate approach to use in each step of BFS. We show that our dynamic on-line heuristic achieves performance within 25% of the optimum possible using an off-line oracle. Our hybrid implementation also provides typical speedups of 1 or greater over prior state-of-the-art for multicores [9], [10], [11] and GPUs [12] when utilizing the same graphs and the same or similar hardware. An early version of this algorithm [2] running on a stock quad-core Intel server was ranked 17th in the GoogleIO November 2011 rankings [14], achieving the fastest single-node implementation and the highest per-core processing rate, and outperforming specialized architectures and clusters with more than 150 nodes.

II. GRAPH PROPERTIES

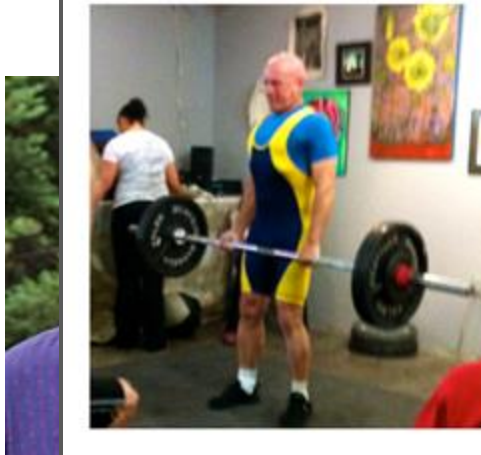
Graphs are a powerful and general abstraction that allow a large number of problems to be represented and solved using the same algorithmic machinery. However, there is often substantial performance to be gained by optimizing algorithms for the types of graph present in a particular target workload.

We can characterize graphs using a few metrics, in particular their diameter and degree distribution. Graphs representing networks used for physical simulations typically have very high diameter and degree bounded by a small constant. As a result, they are amenable to graph partitioning when mapping to parallel systems, as they have mostly local communication and a relatively constant amount of work per vertex, which simplifies load balancing.

In contrast, graphs taken from social networks tend to be both small-world and scale-free. A small-world graph's diameter grows only proportionally to the logarithm of the number of nodes, resulting in a low effective diameter [15]. The effect is caused by a subset of edges connecting parts of the graph that would otherwise be distant. A scale-free graph's degree distribution follows a power law, resulting in a few, very high-degree nodes [16]. These properties complicate the parallelization of graph algorithms to efficiently analyze social networks. Graphs with the small-world property are often hard to partition because the low diameter and cross edges make it difficult to reduce the size of a cut. Moreover, scale-free graphs are challenging to load-balance because the amount of work per node is often proportional to the degree which can vary by several orders of magnitude.

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 ended 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 Kenneth Madduri
College of Computing
Georgia Institute of Technology, Atlanta GA 30332
{bader, karmahs}@cc.gatech.edu

Abstract

Graph algorithms are *extremely* used in understanding about other challenging computational problems in science, technology and many domains. They have particularly suited themselves to shared systems for optimizing increasingly large scale networks. In this paper we present fast parallel implementations of three fundamental graph theory problems: Breadth First Search, st-connectivity and Shortest Paths for unweighted graphs, on multithreaded architectures such as the Cray MTA-2. The architectural features of the MTA-2 and the design of simple, scalable and high-performance graph algorithms. We use our high-dimensional and sparse regular graph structures on their multi-processor, both for algorithmic innovation and more practical performance. For instance, Breadth First Search on a scale free graph of 400 million vertices and 2 billion edges takes less than 3 seconds on 40 processors (20% of peak) on the physical number of 40 processors. This is a significant result in parallel context due to prior implementations of parallel graph algorithms, which were limited or so specific on regular and sparse graphs where compared to the first sequential implementation.

1 Introduction

Graph theoretic and combinatorial problems arise in several traditional and emerging scientific disciplines such as VLSI Design, Optimization, Database and Computational Biology. Some examples include phylogeny reconstruction [14], [22], protein-protein interaction networks [44], [45], [46], [47], and design of VLSI chips [23], data mining [22], [23], [24] and clustering in scientific fields. Graph algorithms also have finding increasing relevance in the domain of large-scale network analysis [22], [23]. Empirical studies show that many social and economic interactions tend to organize

themselves in complex network structures. These networks may contain millions of vertices with degrees ranging from small constants to thousands [22]. The observed social communication networks, transportation and power distribution networks also share this property. The two-layer characteristics realized in these networks are connectivity (which nodes in the graph are first connected to others, or have the most influence) and conversely these nodes are connected to no others. Further metrics to analyzing these networks, the betweenness centrality [22], [23], are important graph structure graph statistics like Breadth First Search (BFS) and shortest paths.

In recognition of the importance of graph structures in solving large-scale problems, in 1998 Performance Computing (PPC) system, several researchers have proposed graph theoretic computational techniques. For instance, the recently announced WS-DBSCAN implementation [14] Challenge calls to request a finding shortest paths in graphs. The widely High Performance Computing Forum (HPCF) [22] program has developed a variety of graph theory benchmark called HPCF-23.12, which is a collection of four kernels operating on a large-scale, directed multi-graph. The directed set implementation of BFS/DFS on directed, unweighted graphs.

Graph theoretic problems are typically memory intensive and the memory access on the parallel and highly irregular. The search to great performance on cache-based systems. The distributed memory structure, the parallel graph algorithms require the two separate implementation due to long memory latency and high communication cost. Parallel graph memory access is a more supportive platform. They offer higher memory bandwidth and lower latency than others, as the global-based memory access is essential in managing parallel algorithms. Performance is dependent on the cache performance of the algorithms and scalability is limited to some extent. While it may be possible to improve the cache performance to a certain degree for some classes of graphs, there are no known algorithms and techniques for cache optimization because the memory

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 Karnesh Madduri
College of Computing
Georgia Institute of Technology, Atlanta GA 30332
bader, karnesh1990@gmail.com

Acknowledgments

Graph algorithms are extensively used to understand and solve challenging computational problems in various scientific and engineering domains. They have particularly gained prominence in recent years for applications such as social network analysis, recommendation systems, and intelligent implementations of three fundamental graph theory problems, Breadth-First Search, connectivity and shortest paths. In this paper, we present a novel graph processing framework such as the C++ M2S-2. The architectural features of the M2S-2 is *the design of simple, scalable and high-performance graph algorithms. We test our implementation on a wide range of real-world graphs, including instances, and report impressive results, both for graph traversal time and parallel performance. For instance, we achieve a speedup of 10x for a graph with 1.5 billion vertices and 2 billion edges taking less than 3 seconds on a 40-processor M2S-2 system with an absolute speedup of close to 30. This is a significant result in parallel computing. We also report the performance of our framework on graph partitioning, report very limited or no speedup on irregular and sparse graphs, where compared to the three sequential implementations*

themselves in complex network structures. These networks may contain billions of vertices and edges ranging from small-scale networks [22] to large-scale networks [23]. The communication networks, transportation and power distribution networks also share this property. The many-body characteristics studied in these networks are *centrality* (which nodes in the graph are best connected to others, or have the most influence) and *connectivity* (how nodes are connected to one another). Popular metrics for analyzing these networks, like *betweenness centrality* [24, 25], are computed using fundamental graph algorithms like Breadth-First Search (BFS) and shortest paths.

In the last few years, there have been a number of efforts for solving large-scale problems on High Performance Computing (HPC) systems, several communities have proposed graph theoretic computational challenges. For instance, the recently announced HP-DMACS Implementation Challenge [26] is a graph theoretic challenge on the graph. The DAMP High Productivity Computer Systems (HPCC) [27] program has developed a synthetic graph benchmark called SISCAR [28, 29] which is composed of four kernels operating on a large-scale, directed multi-graph. (We describe our implementation of SISCAR in

1 Introduction

Graph-theoretic and combinatorial problems arise in several traditional and emerging scientific disciplines such as VLSI Design, Optimization, Databases and Computational Biology. Some examples include phylogeny reconstruction [136, 207], protein-protein interaction networks [442], placement and layout in VLSI chips [212], data mining [212], clustering and clustering in semantic webs. Graph abstractions are also finding increasing relevance in the domain of large-scale network analysis [138, 205]. Empirical studies show that many social and economic interactions tend to organize

Proceedings of the 2006 International Conference on Parallel Processing (ICPP'06)
0-7395-2638-6/06 \$20.00 © 2006 IEEE. For limited to MIT. Downloaded on September 04, 2009 at 14:51:57 UTC from IEEE Xplore. Restrictions apply.

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    $d[s] \leftarrow 0$ ;           Initialize [-1,...,-1] parent array
4    $Q \leftarrow \phi$ ;
5   Enqueue  $s \leftarrow Q$ ;    Put source into queue Q
6   while  $Q \neq \phi$  do      While Q not empty
7     for all  $u \in Q$  in parallel do For each vertex in queue
8       Delete  $u \leftarrow Q$ ;      (parallel)
9     for each  $v$  adjacent to  $u$  in parallel do
10      if  $d[v] = -1$  then         For each neighbor leading
11         $d[v] \leftarrow d[u] + 1$ ; out of queue (parallel), if valid
12        Enqueue  $v \leftarrow Q$ ; child, add to queue

```

Algorithm 1: Level-synchronized Parallel BFS

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: $\text{BFS}(s, G)$

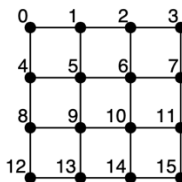
```

1  FRONT := [s]
2  TREE := distribute(-1, |G|)
3  TREE[s] := s
4  while (|FRONT| ≠ 0)
5    E := flatten({{(u, v) : u ∈ G[v]} : v ∈ FRONT})
6    E' := {(u, v) ∈ E | TREE[u] = -1}
7    TREE := TREE ← E'
8    FRONT := {u : (u, v) ∈ E' | v = TREE[u]}
9  return TREE

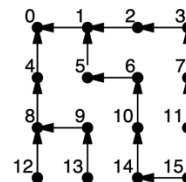
```

Parallel Algorithms
Blelloch & Maggs

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]



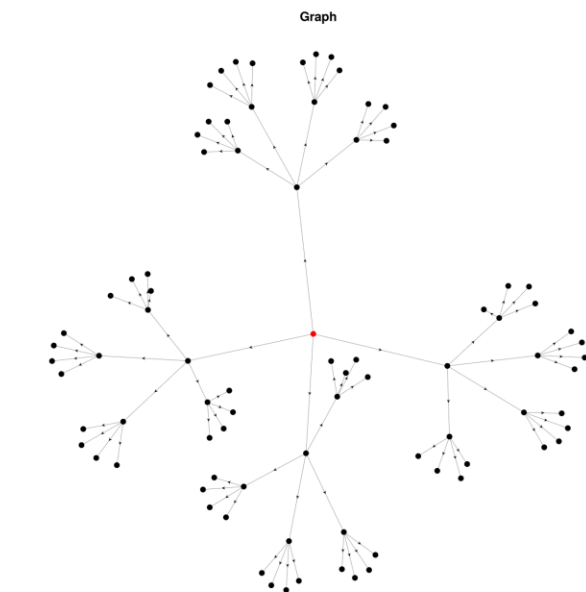
(a)



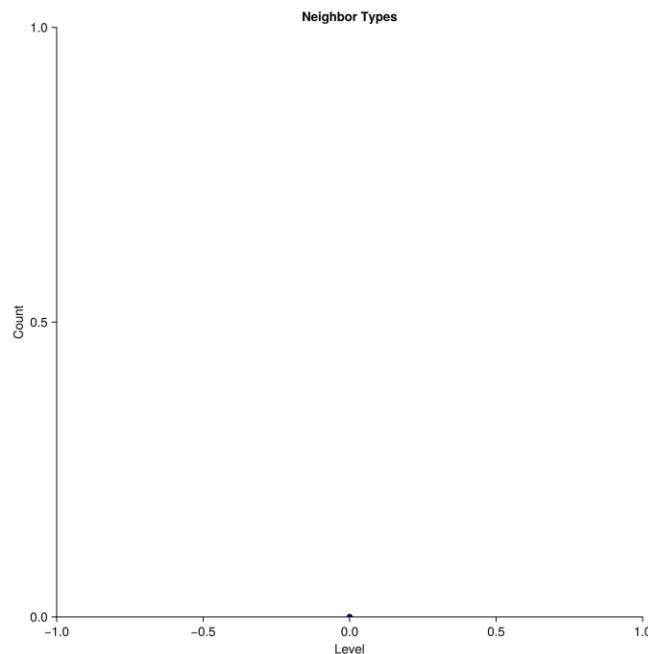
(c)

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



● Claimed
 ● Failed
 ● Peer
 ● Valid parent

```

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

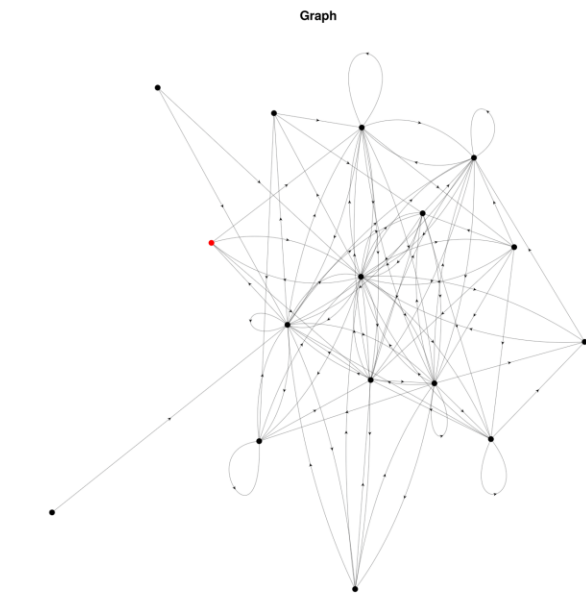
```

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
end function
  
```

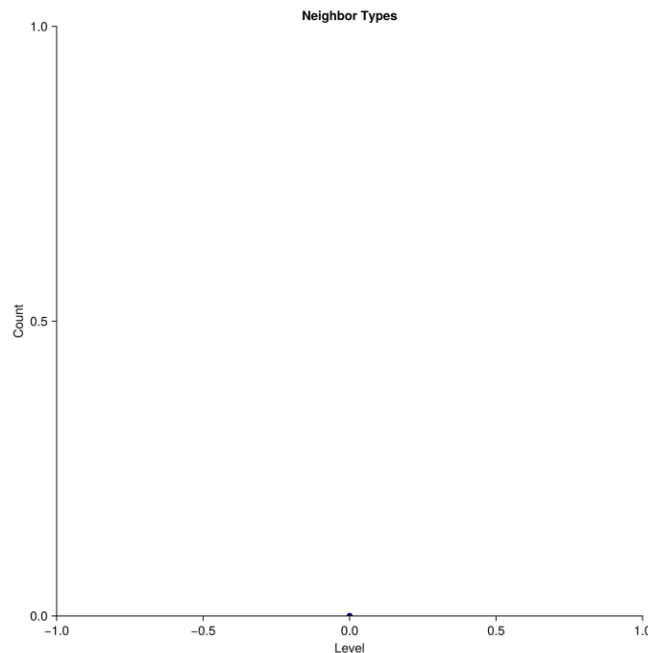
Fig. 2. Single Step of Top-Down Approach

Beamer et al.

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



Own animation, Kronecker graph



● Claimed
 ● Failed
 ● Peer
 ● Valid parent

```

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

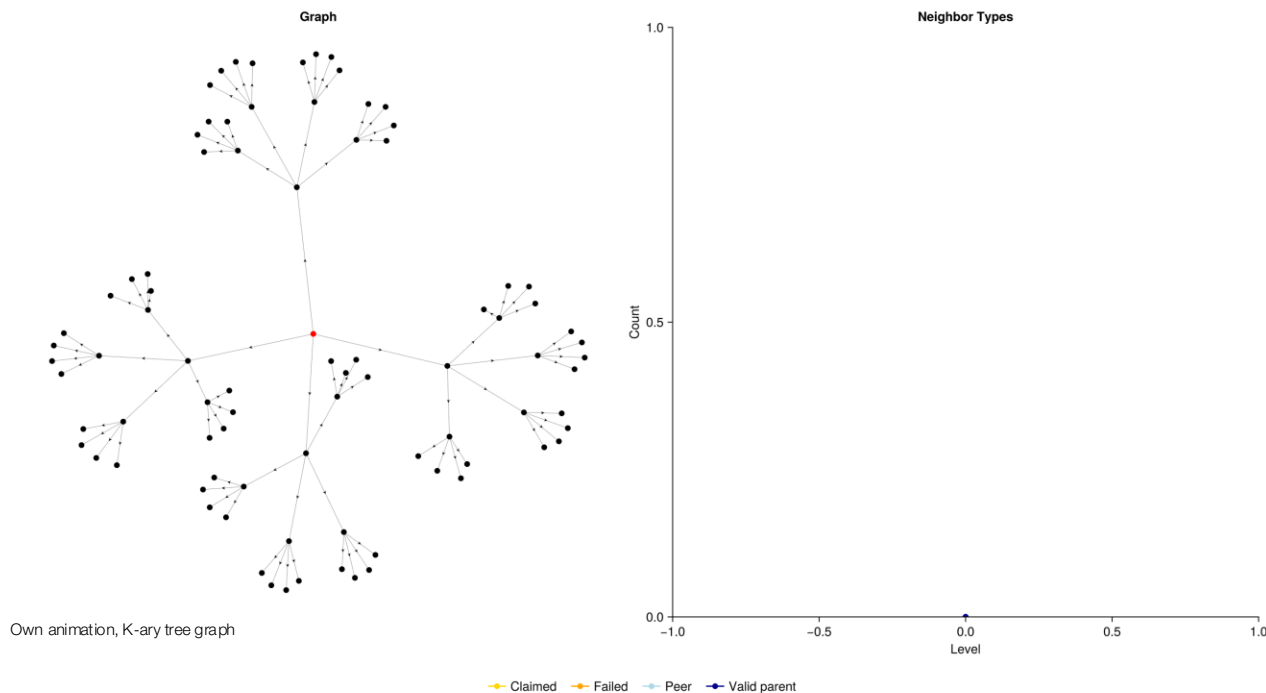
```

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)



```

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
end function

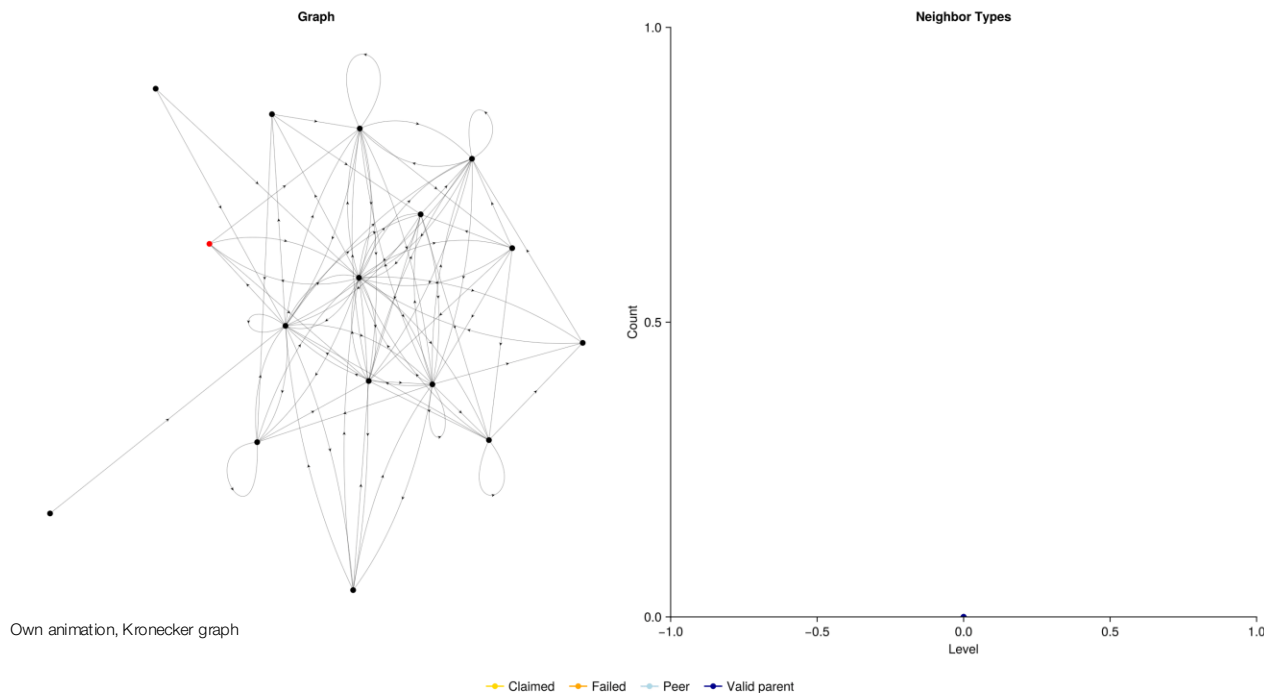
```

Fig. 5. Single Step of Bottom-Up Approach

Beamer et al.

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

Watch the pink 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 bottom-up-step(vertices, frontier, next, parents)
  for v ∈ vertices do
    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
  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

Searching over all neighbours simultaneously*

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

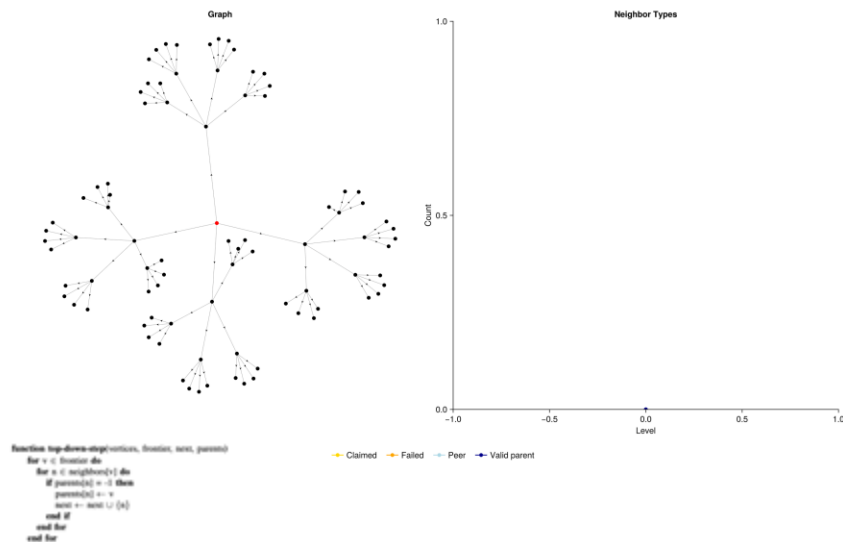


Fig. 3. Single Step of Top-Down Approach

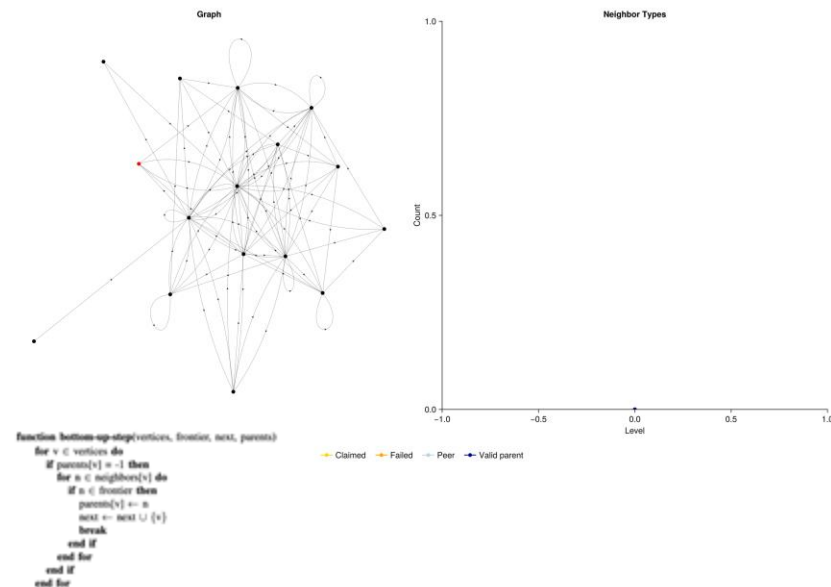


Fig. 5. Single Step of Bottom-Up Approach

Searching over all vertices and neighbours simultaneously*

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

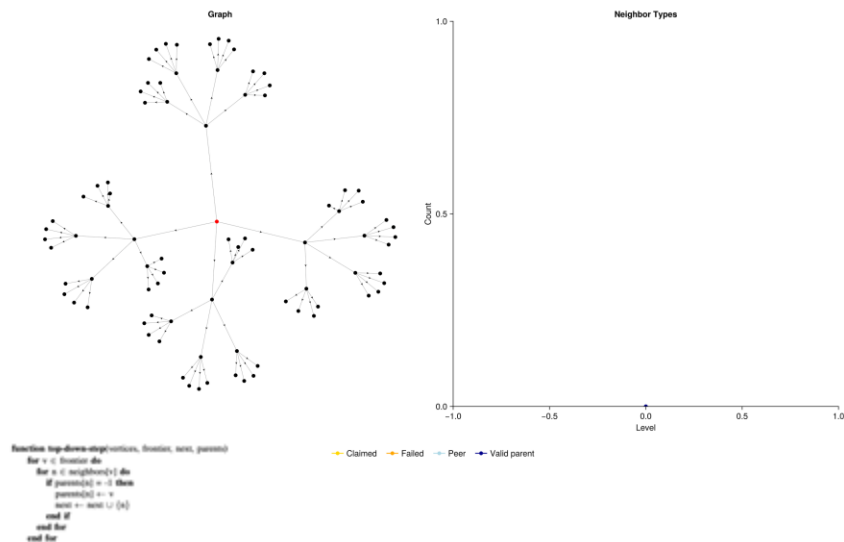


Fig. 3. Single Step of Top-Down Approach

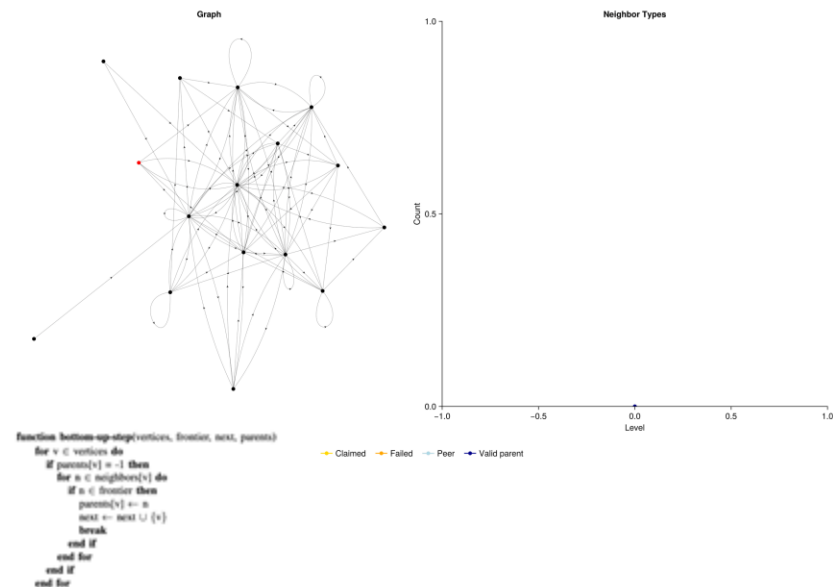


Fig. 5. Single Step of Bottom-Up Approach

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

α = tuning parameter to adjust from upper bound m_u

β = 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

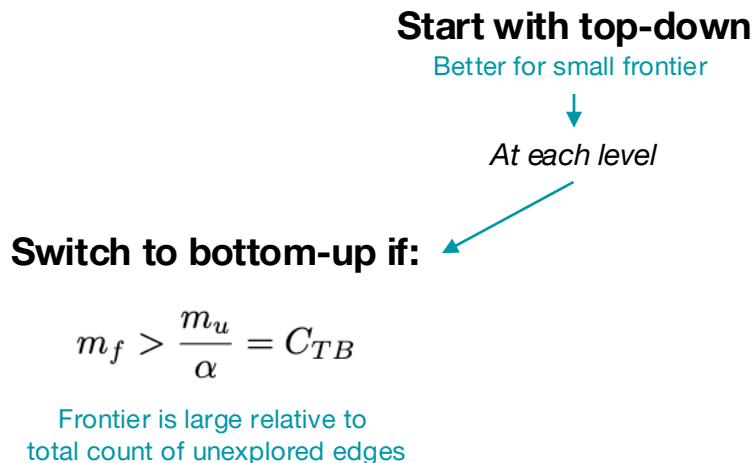
m_f = m edges to check from frontier

m_u = m edges to check from unexplored vertices

α = tuning parameter to adjust from upper bound m_u

β = tuning parameter to adjust from upper bound m

Directional BFS proposes a hybrid approach



Where:

n_f = n vertices in frontier

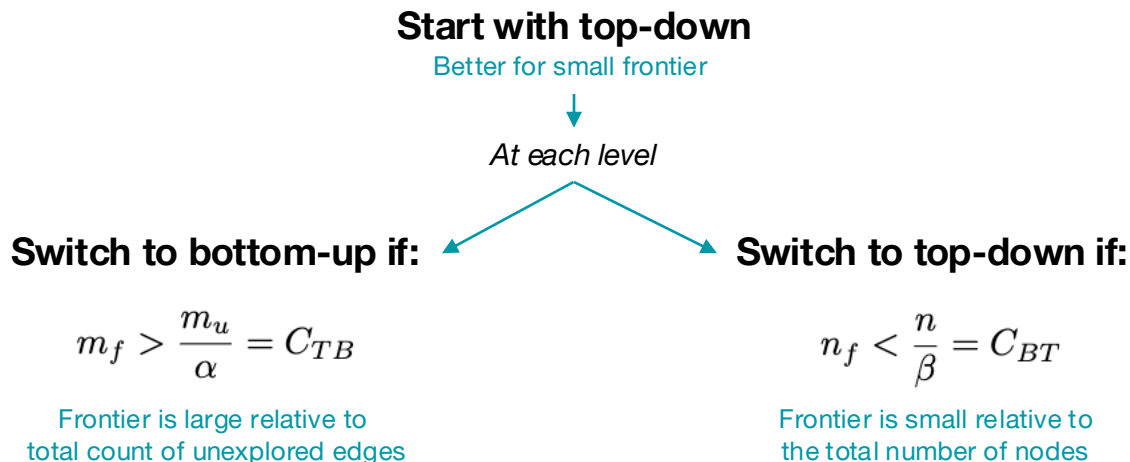
m_f = m edges to check from frontier

m_u = m edges to check from unexplored vertices

α = tuning parameter to adjust from upper bound m_u

β = tuning parameter to adjust from upper bound m

Directional BFS proposes a hybrid approach



Where:

n_f = n vertices in frontier

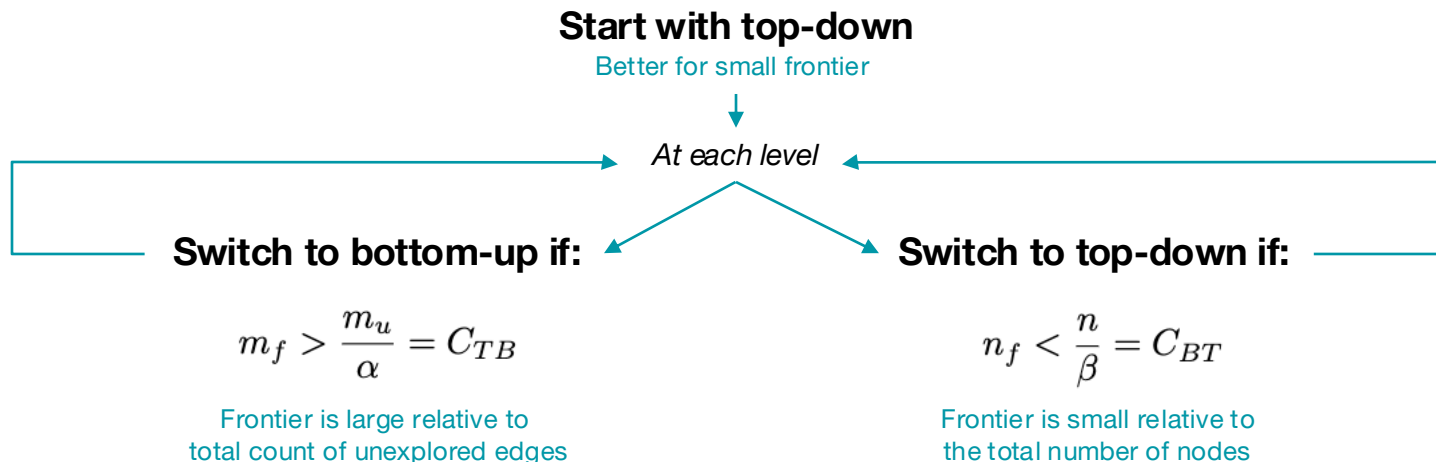
m_f = m edges to check from frontier

m_u = m edges to check from unexplored vertices

α = tuning parameter to adjust from upper bound m_u

β = tuning parameter to adjust from upper bound m

Directional BFS proposes a hybrid approach



Where:

n_f = n vertices in frontier

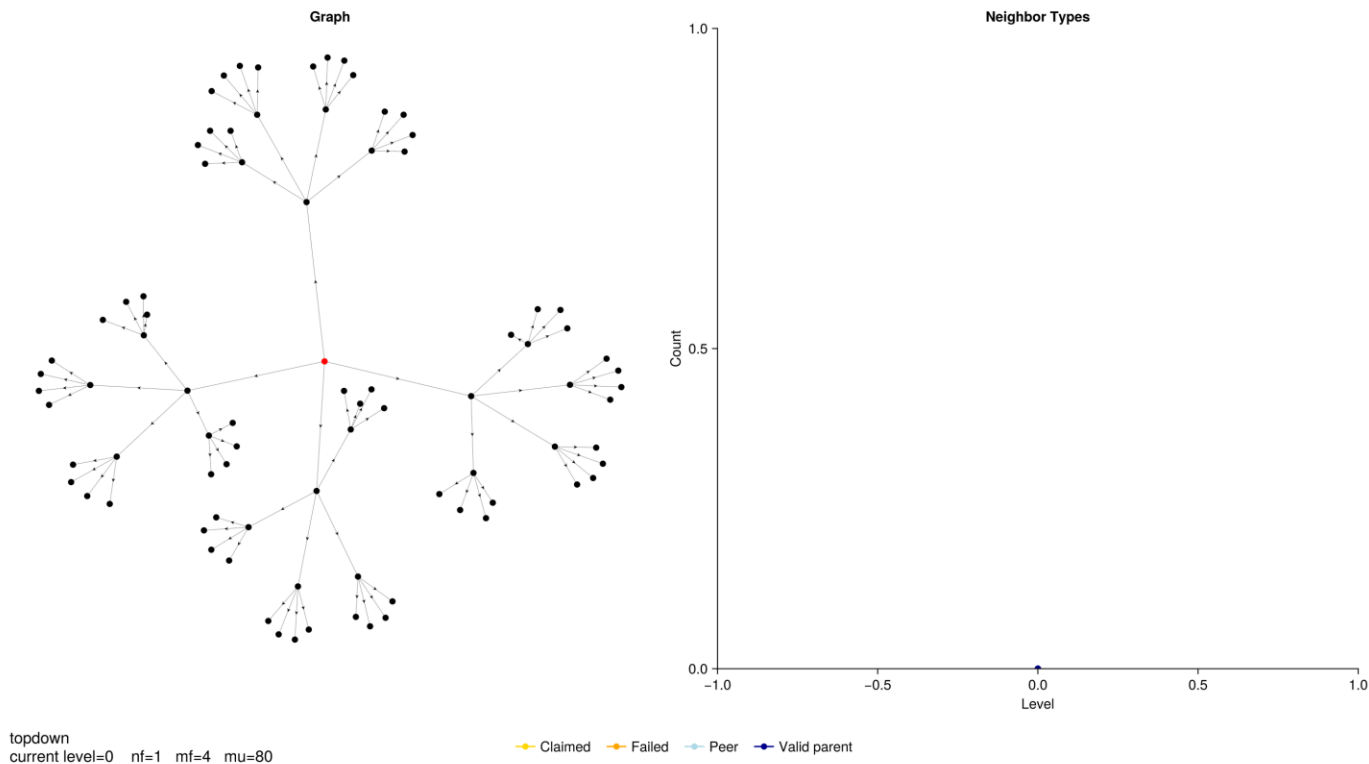
m_f = m edges to check from frontier

m_u = m edges to check from unexplored vertices

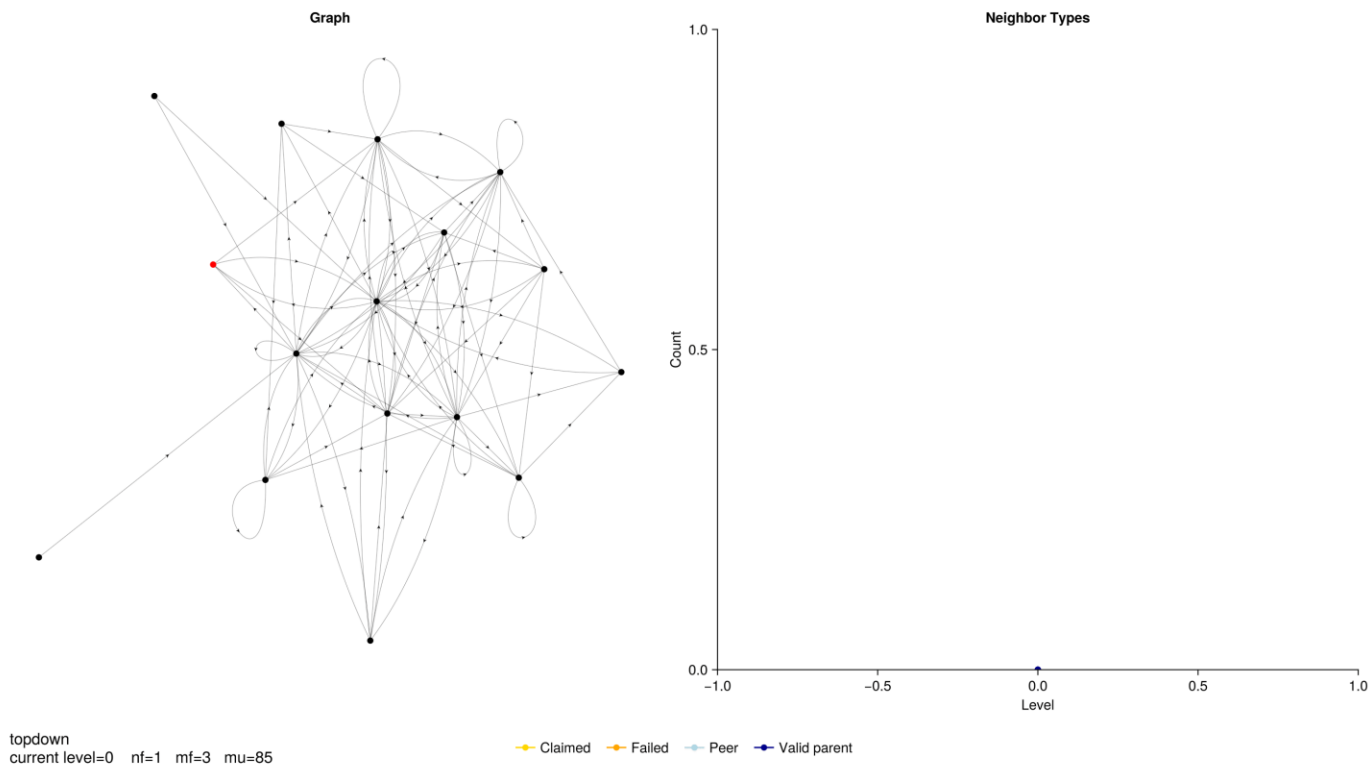
α = tuning parameter to adjust from upper bound m_u

β = tuning parameter to adjust from upper bound m

Hybrid approach for k-tree graph



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–Rényi (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–Rényi (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



 Most representative of compute nodes for clusters

 To evaluate parallel scalability up to 80 threads

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–Rényi (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!

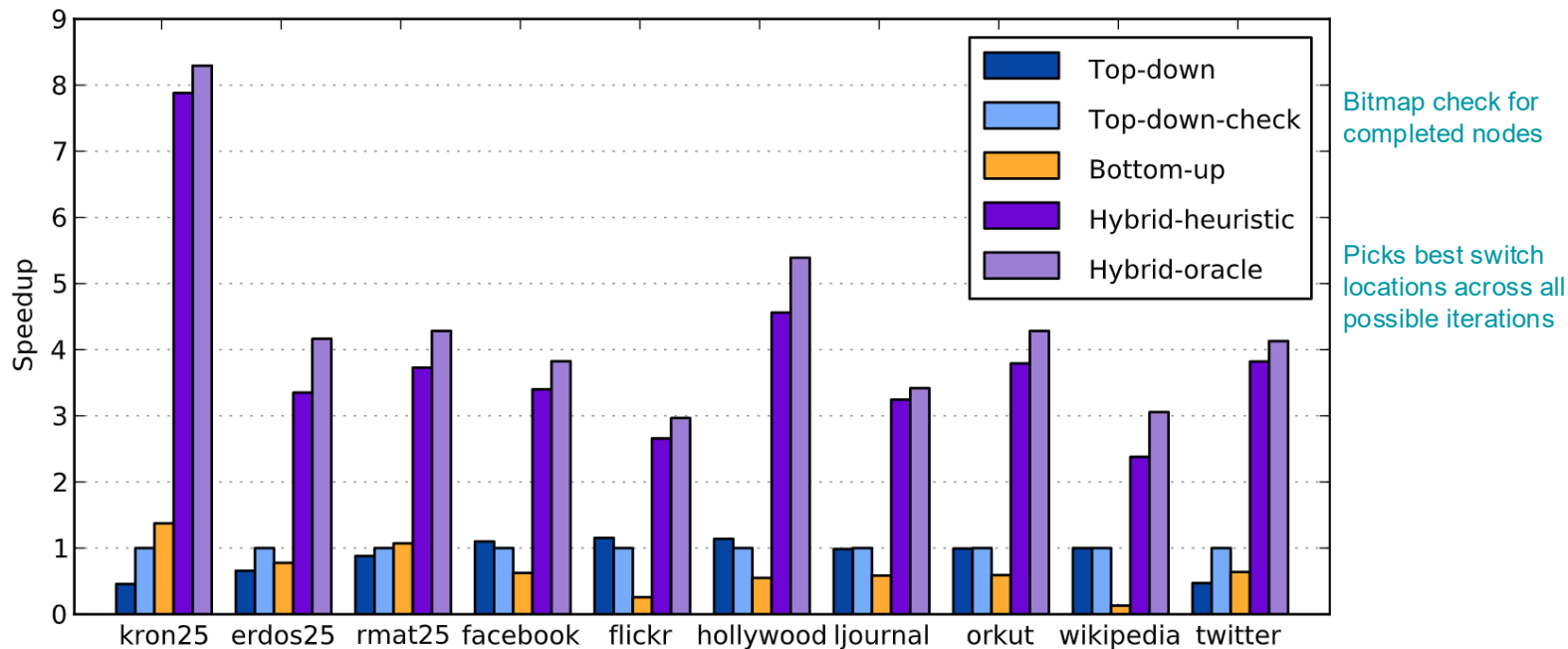


Fig. 10. Speedups on the 16-core machine relative to *Top-down-check*.

Results interpretation

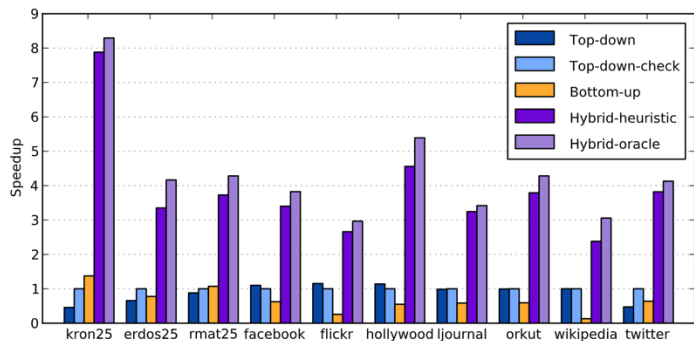
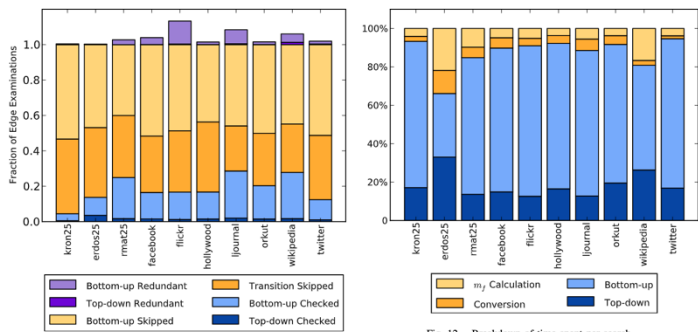
Fig. 10. Speedups on the 16-core machine relative to *Top-down-check*.

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

**Distributed Memory Breadth-First Search Revisited:
Enabling Bottom-Up Search (2013)**

Beamer, Buluç, Asanović, Patterson

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.

2d-graph partitioning

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 n substeps, during each of which $1/p$ 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

