Scalable and Efficient Graph Algorithms and Analysis Techniques for Modern Machines

Quanquan C. Liu
quanquan@mit.edu
Thesis Defense
August 31, 2021
Large, Dynamic Networks

- Facebook: ~ 92.5 million edges
- Friendster: ~ 1.8 billion edges
- Twitter: ~ 2 billion edges
- Common Crawl: ~ 128 billion edges
Modern Machines

- **Multicore systems**: shared-memory (virtual) parallel machines since the early 2000s
  - Multiprocessing systems and multi-core processors
Modern Machines

- **Multicore systems**: shared-memory (virtual) parallel machines since the early 2000s
  - Multiprocessing systems and multi-core processors
- **Massively parallel systems**
  - Distributed cluster of multiple machines
Modern Machines
Modern Machines

<table>
<thead>
<tr>
<th>Machine Type</th>
<th>Virtual CPUs</th>
<th>Memory</th>
<th>Price (USD)</th>
<th>Preemptible Price (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>e2-standard 2</td>
<td>2</td>
<td>8GB</td>
<td>$0.067096</td>
<td>$0.020102</td>
</tr>
<tr>
<td>e2-standard 4</td>
<td>4</td>
<td>16GB</td>
<td>$0.134912</td>
<td>$0.042354</td>
</tr>
<tr>
<td>e2-standard 8</td>
<td>8</td>
<td>32GB</td>
<td>$0.268024</td>
<td>$0.080408</td>
</tr>
<tr>
<td>e2-standard 16</td>
<td>16</td>
<td>64GB</td>
<td>$0.530648</td>
<td>$0.168014</td>
</tr>
<tr>
<td>e2-standard 32</td>
<td>32</td>
<td>128GB</td>
<td>$1.072096</td>
<td></td>
</tr>
</tbody>
</table>

### AWS

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>On-Demand Hourly Rate</th>
<th>vCPU</th>
<th>Memory</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>m6g.8xlarge</td>
<td>$1.232</td>
<td>32</td>
<td>128GB</td>
<td>EBS Only</td>
</tr>
<tr>
<td>m6gd.8xlarge</td>
<td>$1.4464</td>
<td>32</td>
<td>128GB</td>
<td>SSD</td>
</tr>
<tr>
<td>m6i.8xlarge</td>
<td>$1.536</td>
<td>32</td>
<td>128GB</td>
<td>EBS Only</td>
</tr>
<tr>
<td>m5.8xlarge</td>
<td>$1.536</td>
<td>32</td>
<td>128GB</td>
<td>EBS Only</td>
</tr>
<tr>
<td>m5a.8xlarge</td>
<td>$1.376</td>
<td>32</td>
<td>128GB</td>
<td>EBS Only</td>
</tr>
<tr>
<td>m5ad.8xlarge</td>
<td>$1.648</td>
<td>32</td>
<td>128GB</td>
<td>SSD</td>
</tr>
<tr>
<td>m5d.8xlarge</td>
<td>$1.808</td>
<td>32</td>
<td>128GB</td>
<td>SSD</td>
</tr>
</tbody>
</table>

### Google Cloud

- Custom machine type
  - If your ideal machine shape is in between two predefined types, using a custom E2 machine type could save you as much as 40%. For more information, see E2 custom shape and pricing.
Modern Machines

Some companies using these services: Netflix, Twitch, LinkedIn, Facebook, BBC, Baidu, Adobe, Twitter, Coinbase, Comcast, Coursera, Disney, Expedia, Harvard Medical School, International Centre for Radio Astronomy Research, Novartis, Pfizer, Reddit, Sage, Samsung, US Department of State, USDA Food and Nutrition Service, Verizon, Lyft, State of Arizona, Topcoder, Autodesk, Georgetown University, many others
Modern Machines

Some companies using these services: Netflix, Twitch, LinkedIn, Facebook, BBC, Baidu, Adobe, Twitter, Coinbase, Comcast, Coursera, Disney, Expedia, Harvard Medical School, International Centre for Radio Astronomy Research, Novartis, Pfizer, Reddit, Sage, Samsung, US Department of State, USDA Food and Nutrition Service, Verizon, Lyft, State of Arizona, Topcoder, Autodesk, Georgetown University, many others
Commercial Data Centers

Google Kubernetes Engine
Commercial Data Centers

Google Kubernetes Engine

Machine 1  Machine 2  Machine 3
Fast Algorithms on Classes of Graphs

Bipartite Graphs
minimum vertex cover easy, NP-hard in general

Large number of algorithms on trees
maximum independent set, subgraph isomorphism, dominating set, coloring...etc. all easy

Planar graphs
Single-source shortest path can be linear
Fast Algorithms on Classes of Graphs

- Bipartite Graphs
  - minimum vertex cover, NP-hard in general

- Large number of algorithms on trees
  - maximum independent set, subgraph isomorphism, dominating set, coloring...etc.

- Planar graphs
  - Single-source shortest path can be linear

Real-World Networks?
Degeneracy

• A **k-core** is a maximal subgraph where each vertex has degree at least $k$

• The **degeneracy** of a graph is equal to the maximum $k$ for which a k-core exists in a graph
Properties of Real-World Networks

- Degeneracy: a measure of a graph’s sparsity

<table>
<thead>
<tr>
<th>Graph</th>
<th>Num. Vertices</th>
<th>Num. Edges</th>
<th>Degeneracy ($k$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp</td>
<td>425,957</td>
<td>2,099,732</td>
<td>101</td>
</tr>
<tr>
<td>brain-network</td>
<td>784,262</td>
<td>267,844,669</td>
<td>1200</td>
</tr>
<tr>
<td>wikipedia</td>
<td>1,140,149</td>
<td>2,787,967</td>
<td>124</td>
</tr>
<tr>
<td>youtube</td>
<td>1,138,499</td>
<td>5,980,886</td>
<td>51</td>
</tr>
<tr>
<td>stackoverflow</td>
<td>2,601,977</td>
<td>28,183,518</td>
<td>163</td>
</tr>
<tr>
<td>livejournal</td>
<td>4,847,571</td>
<td>85,702,474</td>
<td>329</td>
</tr>
<tr>
<td>orkut</td>
<td>3,072,627</td>
<td>234,370,166</td>
<td>253</td>
</tr>
<tr>
<td>usa-road</td>
<td>23,072,627</td>
<td>28,854,312</td>
<td>3</td>
</tr>
<tr>
<td>twitter</td>
<td>41,652,231</td>
<td>1,202,513,046</td>
<td>2484</td>
</tr>
<tr>
<td>friendster</td>
<td>65,608,366</td>
<td>1,806,067,135</td>
<td>304</td>
</tr>
</tbody>
</table>
Properties of Real-World Networks

- Degeneracy: a measure of a graph’s sparsity

<table>
<thead>
<tr>
<th>Graph</th>
<th>Num. Vertices</th>
<th>Num. Edges</th>
<th>Degeneracy (k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp</td>
<td>425,957</td>
<td>2,099,732</td>
<td>101</td>
</tr>
<tr>
<td>brain-network</td>
<td>784,262</td>
<td>267,844,669</td>
<td>1200</td>
</tr>
<tr>
<td>wikipedia</td>
<td>1,140,149</td>
<td>2,787,967</td>
<td>124</td>
</tr>
<tr>
<td>youtube</td>
<td>1,138,499</td>
<td>5,980,886</td>
<td>51</td>
</tr>
<tr>
<td>stackoverflow</td>
<td>2,601,977</td>
<td>28,183,518</td>
<td>163</td>
</tr>
<tr>
<td>livejournal</td>
<td>4,847,571</td>
<td>85,702,474</td>
<td>329</td>
</tr>
<tr>
<td>orkut</td>
<td>3,072,627</td>
<td>234,370,166</td>
<td>253</td>
</tr>
<tr>
<td>usa-road</td>
<td>23,072,627</td>
<td>28,854,312</td>
<td>3</td>
</tr>
<tr>
<td>twitter</td>
<td>41,652,231</td>
<td>1,202,513,046</td>
<td>2484</td>
</tr>
<tr>
<td>friendster</td>
<td>65,608,366</td>
<td>1,806,067,135</td>
<td>304</td>
</tr>
</tbody>
</table>

Factor of $O(k)$ vs. $O(n)$ or $O(m)$
Dynamic Graphs
Graph Algorithms for Modern Machines

• Two-pronged approach:
  • Computation models that represent modern computing environments
  • Graph properties exhibited by real-world networks
• Experiments for new algorithms
• Lower bounds
Graph Algorithms for Modern Machines

- Two-pronged approach:
  - Computation models that represent modern computing environments
  - Graph properties exhibited by real-world networks
- Experiments for new algorithms
- Lower bounds
Summary of Results

Static Algorithms
- Structural Rounding
- Scheduling with Communication Delay
- MPC Algorithms for Subgraph Counting

Dynamic Algorithms
- Parallel Dynamic k-Core Decomposition
- Dynamic Vertex Coloring
- Parallel Dynamic k-Clique Counting

Lower Bounds/Constructions
- Hardness from Pebbling
- Static-Memory-Hard Hash Functions

MIT PhD Thesis Defense 2021
Summary of Results

Static Algorithms
- Structural Rounding
  - Scheduling with Communication Delay
  - MPC Algorithms for Subgraph Counting

Dynamic Algorithms
- Parallel Dynamic k-Core Decomposition
  - Dynamic Vertex Coloring
  - Parallel Dynamic k-Clique Counting

Lower Bounds/Constructions
- Hardness from Pebbling
  - Static-Memory-Hard Hash Functions
Summary of Results

Static Algorithms
- Structural Rounding

Dynamic Algorithms
- Parallel Dynamic k-Core Decomposition
- Dynamic Vertex Coloring
- Parallel Dynamic k-Clique Counting

Lower Bounds/Constructions
- Hardness from Pebbling
- Static-Memory-Hard Hash Functions

Scheduling with Communication Delay
- Scheduling with Communication Delay in Near-Linear Time

MPC Algorithms for Subgraph Counting
- Parallel Algorithms for Small Subgraph Counting

Structural Rounding: Approximation Algorithms for Graphs Near an Algorithmically Tractable Class
Erik D. Demaine\textsuperscript{1}, Timothy D. Goodrich\textsuperscript{2}, Kyle Kloster\textsuperscript{3}, Brian Lasalle\textsuperscript{3}, Quanquan C. Liu\textsuperscript{1}, Blair D. Sullivan\textsuperscript{4}, Ali Vakilian\textsuperscript{5}, and Andrew van der Poel\textsuperscript{6}

Parallel Batch-Dynamic k-Core Decomposition

Parallel Batch-Dynamic k-Clique Counting
Summary of Results

**Static Algorithms**

- Structural Rounding
- Scheduling with Communication Delay
- MPC Algorithms for Subgraph Counting

**Dynamic Algorithms**

- Parallel Dynamic $k$-Core Decomposition
- Dynamic Vertex Coloring
- Parallel Dynamic $k$-Clique Counting

**Lower Bounds/Constructions**

- Hardness from Pebbling
- Static-Memory-Hard Hash Functions
Summary of Results

**Static Algorithms**
- Structural Rounding
  - Scheduling with Communication Delay
  - MPC Algorithms for Subgraph Counting

**Dynamic Algorithms**
- Parallel Dynamic $k$-Core Decomposition
  - Dynamic Vertex Coloring
  - Parallel Dynamic $k$-Clique Counting

**Lower Bounds/Constructions**
- Hardness from Pebbling
- Static-Memory-Hard Hash Functions
Parallel Batch-Dynamic $k$-Core Decomposition

Quanquan C. Liu  
MIT CSAIL  
quanquan@mit.edu

Jessica Shi  
MIT CSAIL  
jeshi@mit.edu

Shangdi Yu  
MIT CSAIL  
shangdiy@mit.edu

Laxman Dhulipala  
MIT CSAIL  
laxman@mit.edu

Julian Shun  
MIT CSAIL  
jshun@mit.edu

Jessica Shi  
Shangdi Yu  
Laxman Dhulipala  
Julian Shun
Parallel Batch-Dynamic $k$-Core Decomposition

Quanquan C. Liu
MIT CSAIL
quanquan@mit.edu

Jessica Shi
MIT CSAIL
jeshi@mit.edu

Shangdi Yu
MIT CSAIL
shangdiy@mit.edu

Laxman Dhulipala
MIT CSAIL
laxman@mit.edu

Julian Shun
MIT CSAIL
jshun@mit.edu

Jessica Shi
Shangdi Yu
Laxman Dhulipala
Julian Shun
Work-Depth Model

- **Work:**
  - Total time of performing all operations executed by algorithm
Work-Depth Model

- **Work:**
  - Total time of performing all operations executed by algorithm
  - **Work-efficient:** work asymptotically the same as best-known sequential algorithm
Work-Depth Model

• **Work:**
  - Total time of performing all operations executed by algorithm
  - **Work-efficient:** work asymptotically the same as *best-known sequential algorithm*

• **Depth:**
  - Longest chain of sequential dependencies in algorithm
Work-Depth Model

• **Work:**
  • Total time of performing all operations executed by algorithm
  • *Work-efficient:* work asymptotically the same as *best-known sequential algorithm*

• **Depth:**
  • Longest chain of sequential dependencies in algorithm

• **Shared Memory:**
  • Processes can concurrently read/write to the same shared memory
Parallel Batch-Dynamic $k$-Core Decomposition

Quanquan C. Liu  
MIT CSAIL  
quanquan@mit.edu

Jessica Shi  
MIT CSAIL  
jeshi@mit.edu

Shangdi Yu  
MIT CSAIL  
shangdiy@mit.edu

Laxman Dhulipala  
MIT CSAIL  
laxman@mit.edu

Julian Shun  
MIT CSAIL  
jshun@mit.edu

Jessica Shi  
Shangdi Yu  
Laxman Dhulipala  
Julian Shun
$k$-Core Decomposition
$k$-Core Decomposition

Coreness or Core Number of Node $\nu$: Maximum Core Value of a Core Containing $\nu$
$k$-Core Decomposition

Coreness or Core Number of Node $\nu$: Maximum Core Value of a Core Containing $\nu$
**$k$-Core Decomposition**

Coreness or Core Number of Node $\nu$: Maximum Core Value of a Core Containing $\nu$
Approximate $k$-Core Decomposition

Approx. Core Number: Value within some factor of actual core number

Approx. core number of every node: 3

Approx. Core Number: 2
Approximate $k$-Core Decomposition

Approx. Core Number: Value within some factor of actual core number
Applications of $k$-Core Decomposition

- Graph clustering
- Community detection
- Graph visualizations
- Protein network analysis
- Approximating network centrality measures
- Much interest in the machine learning, database, graph analytics, and other communities
Parallel Batch-Dynamic $k$-Core Decomposition

Quanquan C. Liu
MIT CSAIL
quanquan@mit.edu

Jessica Shi
MIT CSAIL
jeshi@mit.edu

Shangdi Yu
MIT CSAIL
shangdiy@mit.edu

Laxman Dhulipala
MIT CSAIL
laxman@mit.edu

Julian Shun
MIT CSAIL
jshun@mit.edu

Jessica Shi
Shangdi Yu
Laxman Dhulipala
Julian Shun
Batch-Dynamic Model Definition [BW09, DDKPSS20]

Initial $k$-Core Decomposition

$G_i$

$B$ Edge Insertions/Deletions

New $k$-Core Decomposition

$G_{i+1}$
There exists a parallel batch-dynamic algorithm that outputs a \((2 + \epsilon)\)-approximation for coreness of each node in \(O(B \cdot \log^2 n)\) amortized work and \(O(\log^2 n \log \log n)\) depth w.h.p., batch size \(B\).
Level Data Structures

• Maximal Matching [Baswana-Gupta-Sen 18, Solomon 16]
• \((\Delta + 1)\)-Coloring [Bhattacharya-Chakrabarty-Henzinger-Nanongkai 18, Bhattacharya-Grandoni-Kulkarni-Liu-Solomon 19]
• Clustering [Wulff-Nilsen 12]
• Low out-degree orientation [Solomon-Wein 20, Henzinger-Neumann-Weiss 20]
• Densest subgraph [Bhattacharya-Henzinger-Nanongkai-Tsourakakis 15]
Sequential Level Data Structure (LDS)

Vertices partitioned into levels

$O(\log^2 n)$

Group of $O(\log n)$ levels
Cut-off: $(1 + \epsilon)^i$

Group of $O(\log n)$ levels
Cut-off: $(1 + \epsilon)^{i-1}$

Group of $O(\log n)$ levels
Cut-off: $(1 + \epsilon)^{i-2}$

...
Sequential Level Data Structure (LDS)

Vertices partitioned into levels

$O(\log^2 n)$

# neighbors: $> 2.1(1 + \epsilon)^i$

Sequential Level Data Structure (LDS)

Vertices partitioned into levels

$O(\log^2 n)$

# neighbors: $> 2.1(1 + \epsilon)^i$

Sequential Level Data Structure (LDS)

Vertices partitioned into levels

$O(\log^2 n)$

$\# \text{ neighbors: } < (1 + \epsilon)^i$

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels

# neighbors: $< (1 + \epsilon)^i$

Sequential Level Data Structure (LDS)

Vertices partitioned into levels

\[ O(\log^2 n) \]

\( \nu \) is stored at a level where
1. Number of neighbors at or above same level is
   \[ \leq 2.1(1 + \epsilon)^i \]
2. Number of neighbors at or above level below \( \nu \) is
   \[ \geq (1 + \epsilon)^i \]

Sequential Level Data Structure (LDS)

Vertices partitioned into levels

\[ O(\log^2 n) \]

\( \nu \) is stored at a level where
1. Number of neighbors at or above same level is
   \[ \leq 2.1(1 + \epsilon)^i \]
2. Number of neighbors at or above level below \( \nu \) is
   \[ \geq (1 + \epsilon)^i \]

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels

$v$ is stored at a level where
1. Number of neighbors at or above same level is
   \[ \leq 2.1(1 + \epsilon)^i \]
2. Number of neighbors at or above level below $v$ is
   \[ \geq (1 + \epsilon)^i \]

Sequential Level Data Structure (LDS)

Vertices partitioned into levels

\(O(\log^2 n)\)

\(\nu\) is stored at a level where
1. Number of neighbors at or above same level is \(\leq 2.1(1 + \epsilon)^i\)
2. Number of neighbors at or above level below \(\nu\) is \(\geq (1 + \epsilon)^i\)

Sequential Level Data Structure (LDS)

Vertices partitioned into levels

$O(\log^2 n)$

$v$ is stored at a level where
1. Number of neighbors at or above same level is
   \[ \leq 2.1(1 + \epsilon)^i \]
2. Number of neighbors at or above level below $v$ is
   \[ \geq (1 + \epsilon)^i \]

Sequential Level Data Structure (LDS)

Vertices partitioned into levels

$O(\log^2 n)$

$v$ is stored at a level where

1. Number of neighbors at or above same level is
   $\leq 2.1(1 + \epsilon)^i$

2. Number of neighbors at or above level below $v$ is
   $\geq (1 + \epsilon)^i$

Sequential Level Data Structure (LDS)

Vertices partitioned into levels

$O(\log^2 n)$ amortized time per update

Sequential Level Data Structure (LDS)

Vertices partitioned into levels

$O(1/e)$ amortized time per update

Formulated for approximate densest subgraph and low out-degree orientation

Difficulties with Parallelization

Large sequential dependencies

Large depth
Difficulties with Parallelization

- Large sequential dependencies
- Large depth
Difficulties with Parallelization

- Large sequential dependencies
- Large depth
Difficulties with Parallelization

- Large sequential dependencies
- Large depth
Difficulties with Parallelization

- Large sequential dependencies
- Large depth
Difficulties with Parallelization

Large sequential dependencies

Large depth
Difficulties with Parallelization

- Large sequential dependencies
- Large depth
Difficulties with Parallelization

- Large sequential dependencies
- Large depth
Difficulties with Parallelization

Large sequential dependencies

Large depth
Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions
Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

Only lower bound invariant, $(1 + \varepsilon)^i$, ever violated.
Our Parallel Batch-Dynamic Level Data Structure

Calculate *desire-level*: closest level that satisfies invariants

Only lower bound invariant, \((1 + \epsilon)^i\), ever violated.
Our Parallel Batch-Dynamic Level Data Structure

Deletions

Calculate *desire-level*: closest level that satisfies invariants

Iterate from *bottommost level to top level* and move vertices to desire-level

Only lower bound invariant, $(1 + \epsilon)^i$, ever violated.
Our Parallel Batch-Dynamic Level Data Structure

Calculate *desire-level*: closest level that satisfies invariants

Deletions

Iterate from *bottommost level to top level* and move vertices to desire-level

Only lower bound invariant, $(1 + \epsilon)^i$, ever violated.
Our Parallel Batch-Dynamic Level Data Structure

Deletions

Calculate *desire-level*: closest level that satisfies invariants

Iterate from **bottommost level to top level** and move vertices to desire-level

Only lower bound invariant, \((1 + \epsilon)^i\), ever violated.
Our Parallel Batch-Dynamic Level Data Structure

Deletions

Calculate desire-level: closest level that satisfies invariants

Update desire-levels of adjacent vertices at higher levels

Only lower bound invariant, $(1 + \epsilon)^i$, ever violated.
Our Parallel Batch-Dynamic Level Data Structure

Deletions

Calculate *desire-level*: closest level that satisfies invariants

Update desire-levels of adjacent vertices at higher levels

Only lower bound invariant, $(1 + \epsilon)^i$, ever violated.
Our Parallel Batch-Dynamic Level Data Structure

Deletions

Calculate *desire-level*: closest level that satisfies invariants

Update *desire-levels* of adjacent vertices at higher levels

Only lower bound invariant, \((1 + \epsilon)^i\), ever violated.
Our Parallel Batch-Dynamic Level Data Structure

Deletions

Calculate *desire-level*: closest level that satisfies invariants

Update desire-levels of adjacent vertices at higher levels

Only lower bound invariant, $(1 + \epsilon)^i$, ever violated.
Our Parallel Batch-Dynamic Level Data Structure

Deletions

Calculate desire-level: closest level that satisfies invariants

No vertices need to be moved to a lower level than the current level!

Update desire-levels of adjacent vertices at higher levels

Only lower bound invariant, \((1 + \epsilon)^i\), ever violated.
Our Parallel Batch-Dynamic Level Data Structure

Deletions

No vertices need to be moved to a lower level than the current level!

Calculate desire-level: closest level that satisfies invariants

$O(\log^2 n \log \log n)$ depth w.h.p

Only lower bound invariant, $(1 + \epsilon)^i$, ever violated.

Update desire-levels of adjacent vertices at higher levels

Calculate desire-level: closest level that satisfies invariants
Our Parallel Batch-Dynamic Level Data Structure

Deletions

No vertices need to be moved to a lower level than the current level!

Calculate desire-level: closest level that satisfies invariants

Update desire-levels of adjacent vertices at higher levels

Only lower bound invariant, $(1 + \epsilon)^i$, ever violated.

$O(\log^2 n \log \log n)$ depth w.h.p
Our Parallel Batch-Dynamic Level Data Structure

Calculate desire-level: closest level that satisfies invariants

Deletions

No vertices need to be moved to a lower level than the current level!

Work-efficient

$O(\log^2 n \log \log n)$ depth w.h.p

Update desire-levels of adjacent vertices at higher levels

Only lower bound invariant, $(1 + \epsilon)^i$, ever violated.
Proof of Our Approximation Factor

Cut-off: $(1 + \epsilon)^i$
Proof of Our Approximation Factor

Coreness Estimate: $(1 + \epsilon)^i$

Cut-off: $(1 + \epsilon)^i$
Proof of Our Approximation Factor: Upper Bound

Coreness Estimate: \((1 + \epsilon)^i\)

Invariant: \(\leq 2.1(1 + \epsilon)^i\)
Proof of Our Approximation Factor: Upper Bound

Coreness Estimate: $(1 + \epsilon)^i$

Invariant: $\leq 2.1(1 + \epsilon)^i$
Proof of Our Approximation Factor: Upper Bound

Estimate: \((1 + \epsilon)^i\)

Invariant: \(\leq 2.1(1 + \epsilon)^i\)

\[\text{core}(v) \leq 2.1(1 + \epsilon)^i\]
Proof of Our Approximation Factor: Upper Bound

Estimate: \((1 + \epsilon)^i\)

Invariant: \(\leq 2.1(1 + \epsilon)^i\)

Lower bound proof only requires lower invariant and definition of \(k\)-core.
## Tested Graphs

Graphs from Stanford SNAP database, DIMACS Shortest Paths challenge, and Network Repository—including some temporal

<table>
<thead>
<tr>
<th>Graph</th>
<th>Num. Vertices</th>
<th>Num. Edges</th>
<th>Degeneracy ($k$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp</td>
<td>425,957</td>
<td>2,099,732</td>
<td>101</td>
</tr>
<tr>
<td>brain-network</td>
<td>784,262</td>
<td>2,099,732</td>
<td>1200</td>
</tr>
<tr>
<td>wikipedia</td>
<td>1,140,149</td>
<td>2,787,967</td>
<td>124</td>
</tr>
<tr>
<td>youtube</td>
<td>1,138,499</td>
<td>5,980,886</td>
<td>51</td>
</tr>
<tr>
<td>stackoverflow</td>
<td>2,601,977</td>
<td>28,183,518</td>
<td>163</td>
</tr>
<tr>
<td>livejournal</td>
<td>4,847,571</td>
<td>85,702,474</td>
<td>329</td>
</tr>
<tr>
<td>orkut</td>
<td>3,072,627</td>
<td>234,370,166</td>
<td>253</td>
</tr>
<tr>
<td>usa-central</td>
<td>14,081,816</td>
<td>16,933,413</td>
<td>2</td>
</tr>
<tr>
<td>usa-road</td>
<td>23,072,627</td>
<td>28,854,312</td>
<td>3</td>
</tr>
<tr>
<td>twitter</td>
<td>41,652,231</td>
<td>1,202,513,046</td>
<td>2484</td>
</tr>
<tr>
<td>friendster</td>
<td>65,608,366</td>
<td>1,806,067,135</td>
<td>304</td>
</tr>
</tbody>
</table>
## Tested Graphs

Graphs from Stanford SNAP database, DIMACS Shortest Paths challenge, and Network Repository—including some temporal

<table>
<thead>
<tr>
<th>Graph</th>
<th>Num. Vertices</th>
<th>Num. Edges</th>
<th>Degeneracy ($k$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp</td>
<td>425,957</td>
<td>2,099,732</td>
<td>101</td>
</tr>
<tr>
<td>brain-network</td>
<td>784,262</td>
<td>267,844,669</td>
<td>1200</td>
</tr>
<tr>
<td>wikipedia</td>
<td>1,140,149</td>
<td>2,787,967</td>
<td>124</td>
</tr>
<tr>
<td>youtube</td>
<td>1,138,499</td>
<td>5,980,886</td>
<td>51</td>
</tr>
<tr>
<td>stackoverflow</td>
<td>2,601,977</td>
<td>28,183,518</td>
<td>163</td>
</tr>
<tr>
<td>livejournal</td>
<td>4,847,571</td>
<td>85,702,474</td>
<td>329</td>
</tr>
<tr>
<td>orkut</td>
<td>3,072,627</td>
<td>234,370,166</td>
<td>253</td>
</tr>
<tr>
<td>usa-central</td>
<td>14,081,816</td>
<td>16,933,413</td>
<td>2</td>
</tr>
<tr>
<td>usa-road</td>
<td>23,072,627</td>
<td>28,854,312</td>
<td>3</td>
</tr>
<tr>
<td>twitter</td>
<td>41,652,231</td>
<td>1,202,513,046</td>
<td>2484</td>
</tr>
<tr>
<td>friendster</td>
<td>65,608,366</td>
<td>1,806,067,135</td>
<td>304</td>
</tr>
</tbody>
</table>
Experiments

- c2-standard-60 Google Cloud instances
  - 30 cores with two-way hyper-threading
  - 236 GB memory
- m1-megamem-96 Google Cloud instances
  - 48 cores with two-way hyperthreading
  - 1433.6 GB memory
- Timeout: 3 hours
- Insertion/Deletion batches generated from random permutation, regular intervals

Graph Based Benchmark Suite: https://github.com/ParAlg/gbbs
Runtimes/Accuracy Against State-of-the-Art Algorithms

**Sun et al. TKDD:** current best **sequential, approx** dynamic algorithm

**Hua et al. TPDS:** current best **parallel, exact** dynamic algorithm

**LDS:** sequential LDS of Henzinger et al.

**PLDSOpt:** code-optimized PLDS (theoretical approx. may not hold for all parameters)

https://github.com/qqliu/batch-dynamic-kcore-decomposition
Runtimes/Accuracy Against State-of-the-Art Algorithms

**Sun et al. TKDD**: current best sequential, approx dynamic algorithm

**Hua et al. TPDS**: current best parallel, exact dynamic algorithm

**LDS**: sequential LDS of Henzinger et al.

**PLDSOpt**: code-optimized PLDS (theoretical approx. may not hold for all parameters)

https://github.com/qqliu/batch-dynamic-kcore-decomposition
Runtimes/Accuracy Against State-of-the-Art Algorithms

**Sun et al. TKDD:** current best sequential, approx dynamic algorithm

**Hua et al. TPDS:** current best parallel, exact dynamic algorithm

**LDS:** sequential LDS of Henzinger et al.

**PLDSOpt:** code-optimized PLDS (theoretical approx. may not hold for all parameters)

https://github.com/qqliu/batch-dynamic-kcore-decomposition
Run times/Accuracy Against State-of-the-Art Algorithms

Sun et al. TKDD: current best sequential, approx dynamic algorithm

Hua et al. TPDS: current best parallel, exact dynamic algorithm

LDS: sequential LDS of Henzinger et al.

PLDSOpt: code-optimized PLDS (theoretical approx. may not hold for all parameters)

PLDSOpt: 22.35–195.82x speedup over Sun on dblp, 27.64–497.63x speedup on LJ

https://github.com/qqliu/batch-dynamic-kcore-decomposition
Runtimes/Accuracy Against State-of-the-Art Algorithms

**Sun et al. TKDD:** current best sequential, approx dynamic algorithm

**Hua et al. TPDS:** current best parallel, exact dynamic algorithm

**LDS:** sequential LDS of Henzinger et al.

**PLDSOpt:** code-optimized PLDS (theoretical approx. may not hold for all parameters)

PLDSOpt: **24.5x** over Hua

PLDSOpt: **2.68x** over Hua

PLDSOpt: **22.35–195.82x** speedup over Sun on dblp, **27.64–497.63x** speedup on LJ

MIT PhD Thesis Defense 2021
Number of Workers

Hua et al. TPDS: current best parallel, exact dynamic algorithm

PLDS Opt: code-optimized PLDS (theoretical approx. may not hold for all parameters)

https://github.com/qqliu/batch-dynamic-kcore-decomposition
**Number of Workers**

Hua et al. TPDS: current best **parallel, exact** dynamic algorithm

**PLDS Opt**: code-optimized PLDS (theoretical approx. may not hold for all parameters)

**PLDSOpt**: 30.28x self-relative speedup

**PLDS**: 26.46x

**Hua**: 2.07x

https://github.com/qqliu/batch-dynamic-kcore-decomposition
On Variety of Graphs

<table>
<thead>
<tr>
<th>Graph Dataset</th>
<th>PLDSOpt Avg.</th>
<th>PLDSOpt Max</th>
<th>PLDS Avg.</th>
<th>PLDS Max</th>
<th>Approx KCore Avg.</th>
<th>Approx KCore Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp</td>
<td>1.9345</td>
<td>3</td>
<td>2.635</td>
<td>4</td>
<td>1.15</td>
<td>3.875</td>
</tr>
<tr>
<td>brain</td>
<td>1.834</td>
<td>3</td>
<td>3.363</td>
<td>4.193</td>
<td>1.315</td>
<td>4.305</td>
</tr>
<tr>
<td>wiki</td>
<td>1.590</td>
<td>3</td>
<td>1.780</td>
<td>4.172</td>
<td>1.010</td>
<td>3</td>
</tr>
<tr>
<td>youtube</td>
<td>1.359</td>
<td>3</td>
<td>1.593</td>
<td>4</td>
<td>1.1283</td>
<td>3.75</td>
</tr>
<tr>
<td>stackoverflow</td>
<td>1.826</td>
<td>3</td>
<td>2.272</td>
<td>4.067</td>
<td>1.048</td>
<td>3.875</td>
</tr>
<tr>
<td>livejournal</td>
<td>1.660</td>
<td>3</td>
<td>2.321</td>
<td>4.175</td>
<td>1.165</td>
<td>4.2</td>
</tr>
<tr>
<td>orkut</td>
<td>1.926</td>
<td>3</td>
<td>3.115</td>
<td>4.175</td>
<td>1.204</td>
<td>4.2</td>
</tr>
<tr>
<td>ctc</td>
<td>1.601</td>
<td>3</td>
<td>1.683</td>
<td>3</td>
<td>1.374</td>
<td>3</td>
</tr>
<tr>
<td>usa</td>
<td>1.826</td>
<td>3</td>
<td>1.683</td>
<td>3</td>
<td>1.379</td>
<td>3</td>
</tr>
<tr>
<td>twitter</td>
<td>2.118*</td>
<td>3*</td>
<td>T.O.</td>
<td>T.O.</td>
<td>T.O.</td>
<td>T.O.</td>
</tr>
<tr>
<td>friendster</td>
<td>1.851*</td>
<td>3*</td>
<td>T.O.</td>
<td>T.O.</td>
<td>T.O.</td>
<td>T.O.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph Dataset</th>
<th>PLDSOpt Avg.</th>
<th>PLDSOpt Max</th>
<th>PLDS Avg.</th>
<th>PLDS Max</th>
<th>Approx KCore Avg.</th>
<th>Approx KCore Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp</td>
<td>1.187</td>
<td>6</td>
<td>1.507</td>
<td>2</td>
<td>1.236</td>
<td>3.0</td>
</tr>
<tr>
<td>brain</td>
<td>1.575</td>
<td>6</td>
<td>1.943</td>
<td>4.186</td>
<td>1.315</td>
<td>5.0</td>
</tr>
<tr>
<td>wiki</td>
<td>1.423</td>
<td>4</td>
<td>1.494</td>
<td>4</td>
<td>1.013</td>
<td>3.875</td>
</tr>
<tr>
<td>youtube</td>
<td>1.268</td>
<td>4</td>
<td>1.317</td>
<td>4</td>
<td>1.137</td>
<td>3.706</td>
</tr>
<tr>
<td>stackoverflow</td>
<td>1.630</td>
<td>6</td>
<td>1.792</td>
<td>4.172</td>
<td>1.045</td>
<td>3.908</td>
</tr>
<tr>
<td>livejournal</td>
<td>1.613</td>
<td>6</td>
<td>1.704</td>
<td>4.14</td>
<td>1.167</td>
<td>3.984</td>
</tr>
<tr>
<td>orkut</td>
<td>1.681</td>
<td>6</td>
<td>1.913</td>
<td>4.175</td>
<td>1.205</td>
<td>4.2</td>
</tr>
<tr>
<td>ctc</td>
<td>1.243</td>
<td>3</td>
<td>1.257</td>
<td>3</td>
<td>1.524</td>
<td>3.0</td>
</tr>
<tr>
<td>usa</td>
<td>1.253</td>
<td>3</td>
<td>1.278</td>
<td>3</td>
<td>1.522</td>
<td>3.0</td>
</tr>
<tr>
<td>twitter</td>
<td>1.893*</td>
<td>4*</td>
<td>T.O.</td>
<td>T.O.</td>
<td>T.O.</td>
<td>T.O.</td>
</tr>
<tr>
<td>friendster</td>
<td>1.685*</td>
<td>3*</td>
<td>T.O.</td>
<td>T.O.</td>
<td>T.O.</td>
<td>T.O.</td>
</tr>
</tbody>
</table>
On Variety of Graphs

**Sun et al. TKDD**: current best sequential, approx dynamic algorithm

**Hua et al. TPDS**: current best parallel, exact dynamic algorithm

**LDS**: sequential LDS of Henzinger et al.

**ExactKCore Dhulipala et al. SPAA**: parallel, static exact algorithm

**PLDSOpt**: code-optimized PLDS

https://github.com/qqliu/batch-dynamic-kcore-decomposition
On Variety of Graphs

Sun et al. TKDD: current best sequential, approx dynamic algorithm

Hua et al. TPDS: current best parallel, exact dynamic algorithm

LDS: sequential LDS of Henzinger et al.

ExactKCore Dhulipala et al. SPAA: parallel, static exact algorithm

PLDSOpt: code-optimized PLDS

https://github.com/qqliu/batch-dynamic-kcore-decomposition
On Variety of Graphs

Sun et al. TKDD: current best sequential, approx dynamic algorithm

Hua et al. TPDS: current best parallel, exact dynamic algorithm

LDS: sequential LDS of Henzinger et al.

ExactKCore Dhulipala et al. SPAA: parallel, static exact algorithm

PLDSOpt: code-optimized PLDS

https://github.com/qqliu/batch-dynamic-kcore-decomposition
On Variety of Graphs

Sun et al. TKDD: current best sequential, approx dynamic algorithm

Hua et al. TPDS: current best parallel, exact dynamic algorithm

LDS: sequential LDS of Henzinger et al.

ExactKCore Dhulipala et al. SPAA: parallel, static exact algorithm

PLDSOpt: code-optimized PLDS

https://github.com/qqliu/batch-dynamic-kcore-decomposition

PLDSOpt: 0.53 sec per batch

PLDS: 3.06 sec per batch
Summary of Results

Static Algorithms
- Structural Rounding
- Scheduling with Communication Delay
- MPC Algorithms for Subgraph Counting

Dynamic Algorithms
- Parallel Dynamic k-Core Decomposition
- Dynamic Vertex Coloring
- Parallel Dynamic k-Clique Counting

Lower Bounds/Constructions
- Hardness from Pebbling
- Static-Memory-Hard Hash Functions
Fully Dynamic $(\Delta + 1)$-Coloring in Constant Update Time

Sayan Bhattacharya†  Fabrizio Grandoni‡  Janardhan Kulkarni¶  Quanquan C. Liu§
Shay Solomon§
Level Data Structure

\[ \Delta := \text{Max Degree} \]

\[ O(\log \Delta) \]

Vertices partitioned into levels

# neighbors: \( < 3^{\ell(v)+2} \)
Level Data Structure

\[ \Delta := \text{Max Degree} \]

\[ O(\log \Delta) \]

Vertices partitioned into levels

\[ \# \text{neighbors: } < 3^{\ell(v)} + 2 \]

= Edge Insertion
Level Data Structure

$\Delta := \text{Max Degree}$

$O(\log \Delta)$

Vertices partitioned into levels

# neighbors: $< 3^{\ell(v)} + 2$

= Edge Insertion
Level Data Structure

$\Delta := \text{Max Degree}$

$O(\log \Delta)$

Vertices partitioned into levels

# neighbors: $< 3^{\ell(v)+2}$

Deterministically recolor, move to bottommost level
Level Data Structure

$\Delta := \text{Max Degree}$

$\mathcal{O}(\log \Delta)$

Vertices partitioned into levels

$\# \text{neighbors: } \geq 3^{\ell(v)+2}$
Level Data Structure

Δ := Max Degree

Vertices partitioned into levels

$O(\log \Delta)$

# neighbors: $\geq 3\ell(v)+2$

Move to lowest level where # neighbors < $3\ell(v)+2$
randomly recolor
Level Data Structure

\[ \Delta := \text{Max Degree} \]

\[ O(\log \Delta) \]

Vertices partitioned into levels

\# neighbors: \( \geq 3^{\ell(v)+2} \)

Recursively resolve new conflicts
Level Data Structure

$\Delta := \text{Max Degree}$

$O(\log \Delta)$

There exists a $O(1)$ amortized update time in expectation and with high probability against an oblivious adversary.

# neighbors: $\geq 3^{\ell(v)+2}$
Level Data Structure

There exists a $O(1)$ amortized update time in expectation and with high probability against an oblivious adversary.

Using techniques from PLDS, can probably parallelize in $O(\log \Delta)$ depth

OPEN: more efficient practically?

# neighbors: $\geq 3^\ell(v)+2$
Summary of Results

**Static Algorithms**

- **Structural Rounding**
  - Structural Rounding: Approximation Algorithms for Graphs Near an Algorithmically Tractable Class
    - Erik D. Demaine, Timothy D. Goodrich, Kyle Kloster, Brian Laszlo, Quanquan C. Liu, Blair D. Sullivan, Ali Vakilian, and Andrew van der Poel

- **Scheduling with Communication Delay**
  - Scheduling with Communication Delay in Near-Linear Time
    - Quanquan C. Liu, Manish Parohit, Zoya Svitkina, Erik Vee, Joshua R. Wang

- **MPC Algorithms for Subgraph Counting**
  - Parallel Algorithms for Small Subgraph Counting
    - Amartya Shankha Biswas, Talya Eden, Quanquan C. Liu, Slobodan Mitrovic, Ronitt Rubinfeld

**Dynamic Algorithms**

- **Parallel Dynamic k-Core Decomposition**
  - Fully Dynamic $(\Delta + 1)$-Coloring in Constant Update Time
    - Sayan Bhattacharya, Fabrizio Grandoni, Janardhan Kulkarni, Quanquan C. Liu, Shay Solomon

- **Parallel Dynamic k-Clique Counting**
  - Parallel Batch-Dynamic $k$-Clique Counting
    - Laxman Dhulipala, Quanquan C. Liu, Julian Shan, Shangdi Yu

**Lower Bounds/Constructions**

- **Hardness from Pebbling**
  - Red-Blue Pebble Game: Complexity of Computing the Trade-Off between Cache Size and Memory Transfers
    - Erik D. Demaine, Quanquan C. Liu

- **Static-Memory-Hard Hash Functions**
  - Static-Memory-Hard Functions and Nonlinear Space-Time Tradeoffs via Pebbling
    - Thaddeus Dryja, Quanquan C. Liu, Sunoo Park
Parallel Dynamic $k$-Clique Counting

Parallel Batch-Dynamic $k$-Clique Counting

Laxman Dhulipala  
MIT CSAIL  
laxman@mit.edu

Quanquan C. Liu  
MIT CSAIL  
quanquan@mit.edu

Julian Shun  
MIT CSAIL  
jshun@mit.edu

Shangdi Yu  
MIT CSAIL  
shangdiy@mit.edu

Laxman Dhulipala  
Julian Shun  
Shangdi Yu
Simple SotA Batch-Dynamic Algorithm

- Makkar, Bader, Green HiPC 2017

Purple: new neighbors
Black: old neighbors

Worst-case: $\Omega(n)$ amortized work per edge update

Edge Insertion: $(V, W)$

Neighbors

Wedges (two edges that share a vertex)
Batch-Dynamic Triangle Counting

https://github.com/ParAlg/gbbs/tree/master/benchmarks/TriangleCounting/DhulipalaLiuShunYu20

$O(\sqrt{m})$ amortized work and $O(\log n)$ depth

up to 3.31x speedup for all batch sizes
Batch-Dynamic Triangle Counting

https://github.com/ParAlg/gbbs/tree/master/benchmarks/TriangleCounting/DhulipalaLiuShunYu20

\( O(\sqrt{m}) \) amortized work and \( O(\log n) \) depth

up to 3.31x speedup for all batch sizes
Summary of Results

**Static Algorithms**

- **Structural Rounding**
- **Scheduling with Communication Delay**
- **MPC Algorithms for Subgraph Counting**

**Dynamic Algorithms**

- **Parallel Dynamic k-Core Decomposition**
- **Dynamic Vertex Coloring**
- **Parallel Dynamic k-Clique Counting**

**Lower Bounds/Constructions**

- **Hardness from Pebbling**
- **Static-Memory-Hard Hash Functions**
Massively Parallel Algorithms for Subgraph Counting

Parallel Algorithms for Small Subgraph Counting

Amartya Shankha Biswas*†  Talya Eden*‡  
Quanquan C. Liu*  Slobodan Mitrović*§  Ronitt Rubinfeld¶

Amartya Shankha Biswas  Talya Eden  Slobodan Mitrović  Ronitt Rubinfeld
MPC Model Definition

- $M$ machines
- *Synchronous* rounds
MPC Model Definition

- $M$ machines
- Synchronous rounds
MPC Model Definition

- $M$ machines
- Synchronous rounds
MPC Model Definition

- $M$ machines
- Synchronous rounds
MPC Model Definition

- $M$ machines
- Synchronous rounds

![Diagram showing three machines connected in a network within S cylinders, each with a sad face emoji.]
MPC Model Definition

- $M$ machines
- Synchronous rounds
MPC Model Definition

- $M$ machines
- **Synchronous** rounds

Total Space: $M \cdot S$
Approximate Triangle Counting
Approximate Triangle Counting
Approximate Triangle Counting
Approximate Triangle Counting
Approximate Triangle Counting
Approximate Triangle Counting

$O(\log n)$
Approximate Triangle Counting

There exists a MPC algorithm that outputs a \((1 + \epsilon)\)-approximation for the number of triangles if the number of triangles \(T \geq \sqrt{d_{avg}}\) and uses \(\tilde{O}(m)\) total space and \(\tilde{\Theta}(n)\) space per machine, \(O(1)\) MPC rounds.

Massively Parallel Algorithms for Small Subgraph Counting
Amartya Shankha Biswas, Talya Eden, Quanquan C. Liu, Slobodan Mitrovic, Ronitt Rubinfeld
[arxiv.org/2002.08299]

Previous: \(T \geq d_{avg}\) [Pagh and Tsourakakis ‘12]
# MPC Simulation Experiments

Graphs from Stanford SNAP Database

<table>
<thead>
<tr>
<th>File</th>
<th>$\delta$</th>
<th>Partition Approximation</th>
<th>Our Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ego-Facebook</td>
<td>0.5</td>
<td>0.62</td>
<td>1.31</td>
</tr>
<tr>
<td>feather-lastfm-social</td>
<td>0.5</td>
<td>5.41</td>
<td>1.08</td>
</tr>
<tr>
<td>ca-GrQc</td>
<td>0.5</td>
<td>4.53</td>
<td>1.64</td>
</tr>
<tr>
<td>ca-HepPh</td>
<td>0.5</td>
<td>0.66</td>
<td>1.22</td>
</tr>
<tr>
<td>ego-Facebook</td>
<td>0.75</td>
<td>0.75</td>
<td>0.97</td>
</tr>
<tr>
<td>ca-GrQc</td>
<td>0.75</td>
<td>5.82</td>
<td>0.82</td>
</tr>
<tr>
<td>ca-HepPh</td>
<td>0.75</td>
<td>5.90</td>
<td>0.86</td>
</tr>
<tr>
<td>musae-twitch (DE)</td>
<td>0.75</td>
<td>0.74</td>
<td>0.95</td>
</tr>
<tr>
<td>oregon1_010519</td>
<td>0.75</td>
<td>0.60</td>
<td>0.71</td>
</tr>
</tbody>
</table>

Better approximations on all graphs

Approximation Factor (Same Machine Memory) Compared with PT12

Summary of Results

**Static Algorithms**
- Structural Rounding

**Dynamic Algorithms**
- Parallel Dynamic $k$-Core Decomposition
- Parallel Dynamic $k$-Clique Counting

**Lower Bounds/Constructions**
- Hardness from Pebbling
- Static-Memory-Hard Hash Functions
Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, Zoya Svitkina, Erik Vee, Joshua R. Wang
Precedence-Constrained Jobs Modeled as DAG
Precedence-Constrained Jobs Modeled as DAG

- Nodes scheduled according to some topological sort

![Diagram of a DAG with nodes a, b, c, d, and e connected in a specific order]
Precedence-Constrained Jobs Modeled as DAG

- Nodes scheduled according to some topological sort
- Near-linear time algorithm:
  - Identical machines,
  - Uniform communication delay,
  - Allow duplication of jobs
Precedence-Constrained Jobs Modeled as DAG

- Nodes scheduled according to some topological sort
- Near-linear time algorithm:
  - Identical machines,
  - Uniform communication delay,
  - Allow duplication of jobs
- Approximation factor matches best by Lepere-Rapine [02]
Precedence-Constrained Jobs Modeled as DAG

- Nodes scheduled according to some topological sort
- Near-linear time algorithm:
  - Identical machines,
  - Uniform communication delay,
  - Allow duplication of jobs
- Approximation factor matches best by Lepere-Rapine [02]

Main challenge: determine intersection between ancestor sets in $o(n^2)$ time
Precedence-Constrained Jobs Modeled as DAG

**Main challenge:** determine intersection between ancestor sets in $o(n^2)$ time

**Solution:** size-estimation via sketching, sampling and pruning, and work charging argument

**Open:** can we achieve same result without duplication?

- Nodes scheduled according to some topological sort
- Near-linear time algorithm:
  - Identical machines,
  - Uniform communication delay,
  - Allow duplication of jobs
- Approximation factor matches best by Lepere-Rapine [02]
Summary of Results

Static Algorithms

- **Structural Rounding**
  - Scheduling with Communication Delay
  - MPC Algorithms for Subgraph Counting

Dynamic Algorithms

- Parallel Dynamic k-Core Decomposition
  - Dynamic Vertex Coloring
  - Parallel Dynamic k-Clique Counting

Lower Bounds/Constructions

- Hardness from Pebbling
  - Static-Memory-Hard Hash Functions
Structural Rounding

**Structural Rounding:** Approximation Algorithms for Graphs Near an Algorithmically Tractable Class

Erik D. Demaine¹, Timothy D. Goodrich², Kyle Kloster², Brian Lavallee², Quanquan C. Liu¹, Blair D. Sullivan², Ali Vakilian¹, and Andrew van der Poel²

Erik D. Demaine Timothy D. Goodrich Kyle Kloster Brian Lavallee Blair D. Sullivan Ali Vakilian Andrew van der Poel
(Minimization) Problem $\mathcal{P}$ with optimal solution $OPT(G)$, e.g. vertex cover
(Minimization) Problem $\mathcal{P}$ with optimal solution $OPT(G)$, e.g. vertex cover

Structural Rounding

$G$ $\rightarrow$ $G' \in C_\lambda$

$\alpha$-approx. min number of edits $X$

$\beta$-approx. on $C_\lambda$
eq e.g. $\lambda \leq \beta \cdot \lambda^*$

$G'$ is a structured graph class

$\alpha$, $\beta$-approximation

Edit to a Structured Graph Class
(Minimization) Problem $\mathcal{P}$ with optimal solution $OPT(G)$, e.g. vertex cover

Structural Rounding

$G' \in C_\lambda$

$\alpha$-approx. min number of edits $X$

$\beta$-approx. on $C_\lambda$
eq \lambda \leq \beta \cdot \lambda^*$

$G$

Edit to a Structured Graph Class

$(\alpha, \beta)$-approximation

Solve the problem in the edited graph $G'$: $OPT(G')$

where we assume $OPT(G') \leq OPT(G)$
Structural Rounding

(Minimization) Problem $\mathcal{P}$ with optimal solution $OPT(G)$, e.g. vertex cover

$S \leq OPT(G') + c \cdot \alpha \cdot |X|$

Lift the solution to the original graph with loss $c$ i.e. need to add $c \cdot |X|$ to the solution

Edit to a Structured Graph Class $G' \in C_\lambda$

$\alpha$-approx. min number of edits $X$

$\beta$-approx. on $C_\lambda$ e.g. $\lambda \leq \beta \cdot \lambda^*$

Solve the problem in the edited graph $G'$: $OPT(G') \leq OPT(G)$
Summary of Results

Static Algorithms

- Structural Rounding

Dynamic Algorithms

- Parallel Dynamic k-Core Decomposition
- Dynamic Vertex Coloring
- Parallel Dynamic k-Clique Counting

Lower Bounds/Constructions

- Hardness from Pebbling
- Static-Memory-Hard Hash Functions

Scheduling with Communication Delay

MPC Algorithms for Subgraph Counting

Parallel Algorithms for Small Subgraph Counting

- Amartya Shankha Biswas
- Talya Eden
- Quanquan C. Liu
- Slobodan Mitrovic
- Ronitt Rubinfeld
Hardness from Pebbling

Red-Blue Pebble Game: Complexity of Computing the Trade-Off between Cache Size and Memory Transfers

Erik D. Demaine
Quanquan C. Liu

Static-Memory-Hard Functions and Nonlinear Space-Time Tradeoffs via Pebbling

Thaddeus Dryja
Quanquan C. Liu
Sunoo Park

Erik D. Demaine
Thaddeus Dryja
Sunoo Park
Pebble Games

- Combinatorial game to model:
  - Register allocation
  - Rematerialization/gradient checkpointing in ML
  - Input/output complexity
- Hardness for obtaining optimum schedule in external-memory model [DL18]
  - PSPACE-complete
  - Fixed-parameter intractable
Pebble Games

• Combinatorial game to model:
  • Register allocation
  • Rematerialization/gradient checkpointing in ML
  • Input/output complexity
• Hardness for obtaining optimum schedule in external-memory model [DL18]
  • PSPACE-complete
  • Fixed-parameter intractable
Password Hashing

Desirable goal:
Make brute-force attacks hard by making $F$ hard to compute over many hashes.

(Not implied by traditional hash function guarantees like collision-resistance.)

Honest evaluators need only use $F$ few times

Adversaries may run $F$ many times (e.g. large-scale server attacks.)
Candidate Constructions

• Any graph with one target node \textit{doesn’t work}
• Need at least enough target nodes so that \textit{number hash outputs} is reasonably large
• Simple construction \textit{cylinder graph} we implemented \((n = N^2)\)
Acknowledgements

[Not included—You had to be there...]