Scalable and Efficient Graph Algorithms and Analysis Techniques for Modern Machines

Quanquan C. Liu
quanquan@mit.edu

Thesis Defense

August 31, 2021

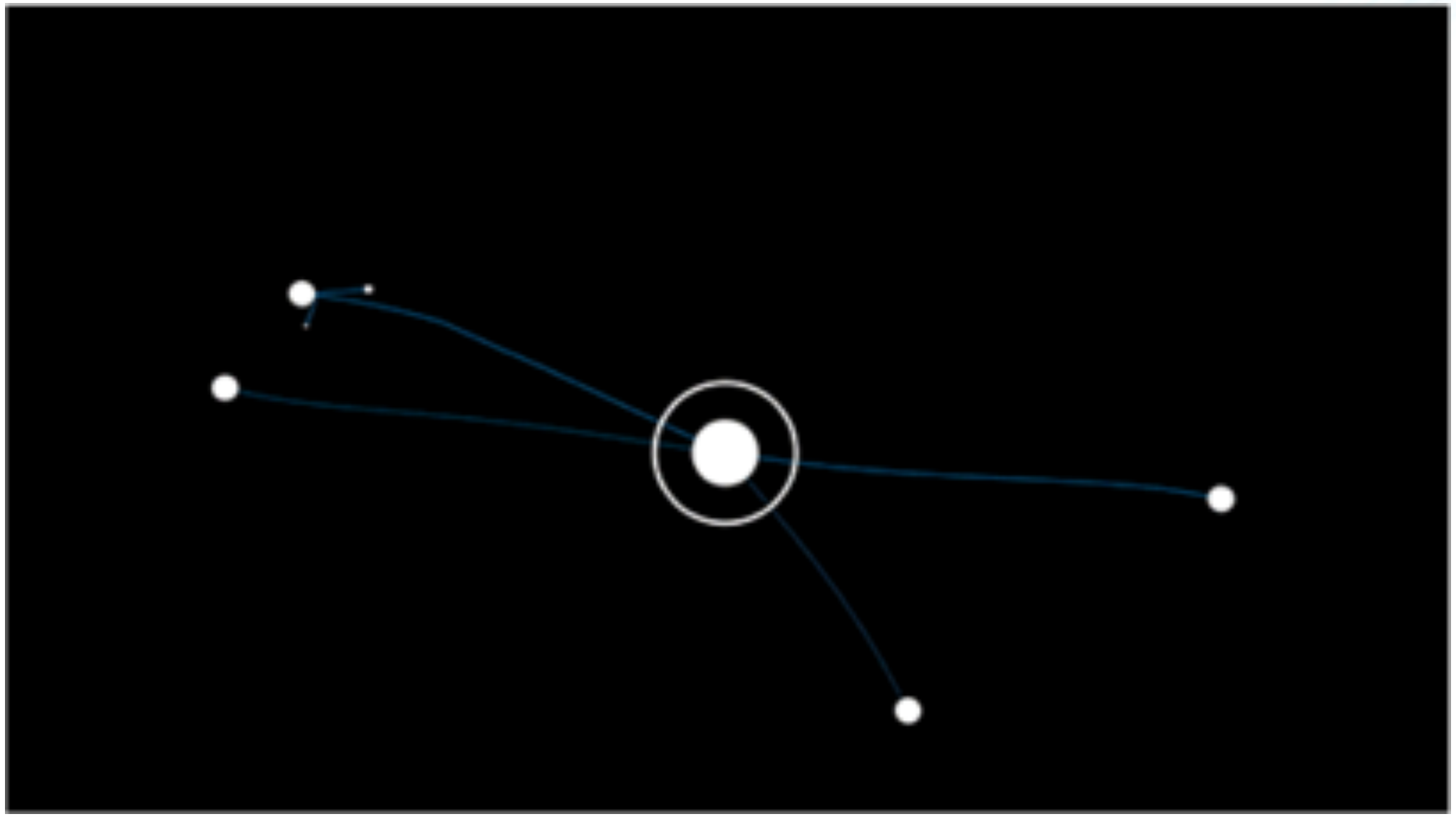
Large, Dynamic Networks

facebook ~ 92.5 million edges

friendster ~ 1.8 billion edges

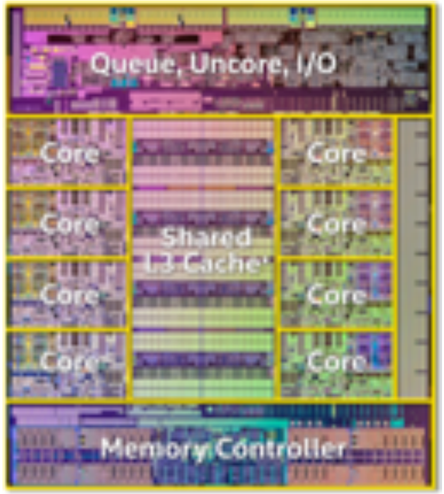
~ 2 billion edges

Common Crawl ~ 128 billion edges



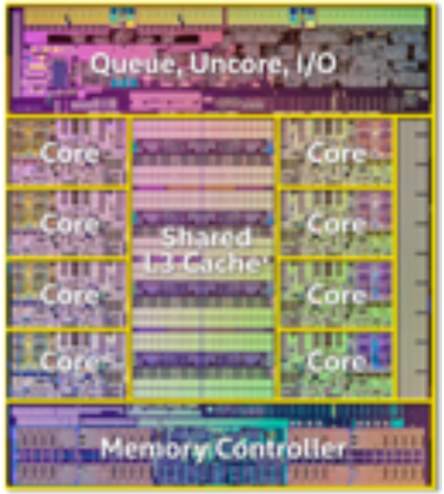
Modern Machines

- **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

E2 standard machine types

Iowa (us-central1) | Monthly | Hourly

Machine type	Virtual CPUs	Memory	Price (USD)	Preemptible price (USD)
e2-standard-2	2	8GB	$0.067006	$0.020102
e2-standard-4	4	16GB	$0.134012	$0.040204
e2-standard-8	8	32GB	$0.268024	$0.080408
e2-standard-16	16	64GB	$0.536048	$0.160816
e2-standard-32	32	128GB	$1.072096	$0.32163
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 vCPUs and memory.			

Google Cloud

aws

Instance name	On-Demand hourly rate	vCPU	Memory	Storage	[illegible]
m6g.8xlarge	$1.232	32	128 GiB	EBS Only	[illegible]
m6gd.8xlarge	$1.4464	32	128 GiB	1 x 1900 NVMe SSD	10 Gigabit
m6i.8xlarge	$1.536	32	128 GiB	EBS Only	12500 Megabit
m5.8xlarge	$1.536	32	128 GiB	EBS Only	10 Gigabit
m5a.8xlarge	$1.376	32	128 GiB	EBS Only	Up to 10 Gigabit
m5ad.8xlarge	$1.648	32	128 GiB	2 x 600 NVMe SSD	Up to 10 Gigabit
m5d.8xlarge	$1.808	32	128 GiB	2 x 600 NVMe SSD	10 Gigabit

Modern Machines

E2 standard machine types

Iowa (us-central1) Monthly Hourly

Machine type	Virtual CPUs	Memory	Price (USD)	Preemptible price (USD)
e2-standard-2	2	8GB	$0.067006	$0.020102
e2-standard-4	4	16GB	$0.134012	$0.040204
e2-standard-8	8	32GB	$0.268024	$0.080408
e2-standard-16	16	64GB	$0.536048	$0.160816
e2-standard-32	32	128GB	$1.072096	[illegible]
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 vCPUs and memory.			

Google Cloud

Instance name	On-Demand hourly rate	vCPU	Memory	Storage	
m6g.8xlarge	$1.232	32	128 GiB	EBS Only	[illegible]
m6gd.8xlarge	$1.4464	32	128 GiB	1 x 1900 NVMe SSD	10 Gigabit
m6i.8xlarge	$1.536	32	128 GiB	EBS Only	12500 Megabit
m5.8xlarge	$1.536	32	128 GiB	EBS Only	10 Gigabit
m5a.8xlarge	$1.376	32	128 GiB	EBS Only	Up to 10 Gigabit
m5ad.8xlarge	$1.648	32	128 GiB	2 x 600 NVMe SSD	Up to 10 Gigabit
m5d.8xlarge	$1.808	32	128 GiB	2 x 600 NVMe SSD	10 Gigabit

aws

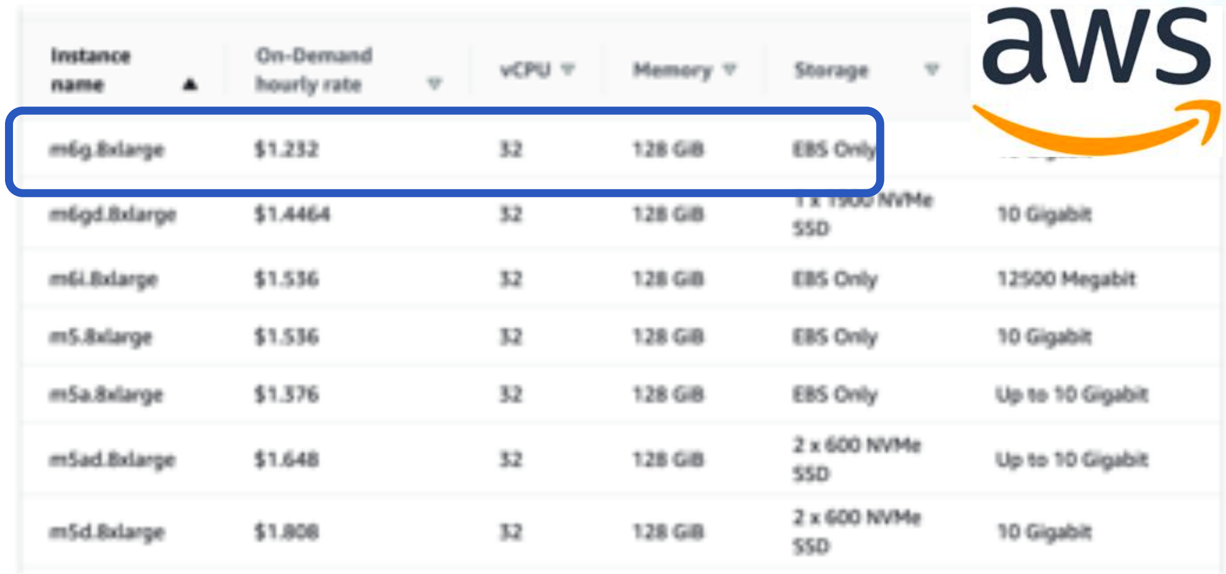
Modern Machines

E2 standard machine types

Iowa (us-central1) Monthly Hourly

Machine type	Virtual CPUs	Memory	Price (USD)	Preemptible price (USD)
e2-standard-2	2	8GB	$0.067006	$0.020102
e2-standard-4	4	16GB	$0.134012	$0.040204
e2-standard-8	8	32GB	$0.268024	$0.080408
e2-standard-16	16	64GB	$0.536048	$0.160816
e2-standard-32	32	128GB	$1.072096	$0.32163

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 vCPUs and memory.



Instance name	On-Demand hourly rate	vCPU	Memory	Storage	
m6g.8xlarge	$1.232	32	128 GiB	EBS Only	
m6gd.8xlarge	$1.4464	32	128 GiB	1 x 1900 NVMe SSD	10 Gigabit
m6i.8xlarge	$1.536	32	128 GiB	EBS Only	12500 Megabit
m5.8xlarge	$1.536	32	128 GiB	EBS Only	10 Gigabit
m5a.8xlarge	$1.376	32	128 GiB	EBS Only	Up to 10 Gigabit
m5ad.8xlarge	$1.648	32	128 GiB	2 x 600 NVMe SSD	Up to 10 Gigabit
m5d.8xlarge	$1.808	32	128 GiB	2 x 600 NVMe SSD	10 Gigabit

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

1 million companies each!

E2 standard machine types

Iowa (us-central1) Monthly Hourly

Machine type	Virtual CPUs	Memory	Price (USD)	Preemptible price (USD)
e2-standard-2	2	8GB	$0.067006	$0.020102
e2-standard-4	4	16GB	$0.134012	$0.040204
e2-standard-8	8	32GB	$0.268024	$0.080408
e2-standard-16	16	64GB	$0.536048	$0.160816
e2-standard-32	32	128GB	$1.072096	$0.32163

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 vCPUs and memory.

Google Cloud

Instance name	On-Demand hourly rate	vCPU	Memory	Storage	
m6g.8xlarge	$1.232	32	128 GiB	EBS Only	
m6gd.8xlarge	$1.4464	32	128 GiB	1 x 1900 NVMe SSD	10 Gigabit
m6i.8xlarge	$1.536	32	128 GiB	EBS Only	12500 Megabit
m5.8xlarge	$1.536	32	128 GiB	EBS Only	10 Gigabit
m5a.8xlarge	$1.376	32	128 GiB	EBS Only	Up to 10 Gigabit
m5ad.8xlarge	$1.648	32	128 GiB	2 x 600 NVMe SSD	Up to 10 Gigabit
m5d.8xlarge	$1.808	32	128 GiB	2 x 600 NVMe SSD	10 Gigabit

aws

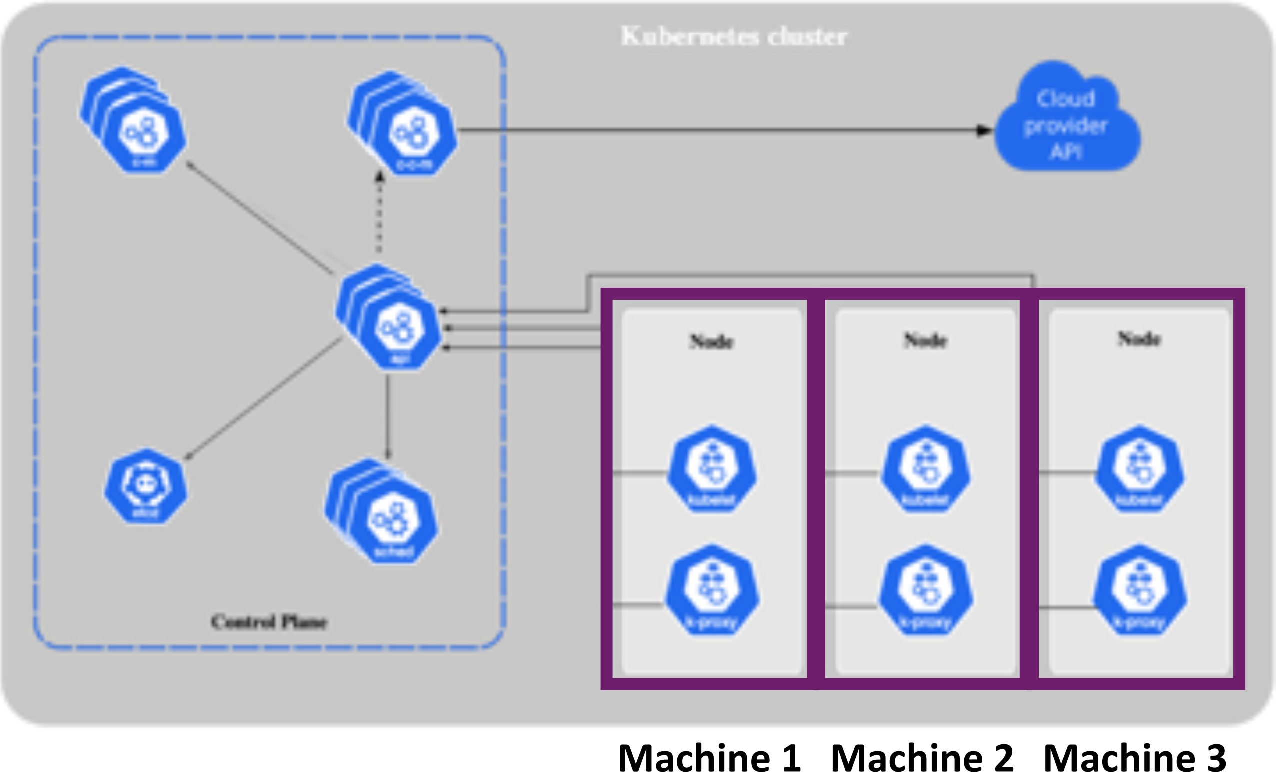
Some companies using these services: *Netflix, Twitch, LinkedIn, Facebook, BBC, Baidu, Adobe, Twitter, Coinbase, Comcast, Coursera, Disney, Expedia, Harvard Medical School, International Centre for Radio Astronomy Research, Novartis, Pfizer, Reddit, Sage, Samsung, US Department of State, USDA Food and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

Commercial Data Centers

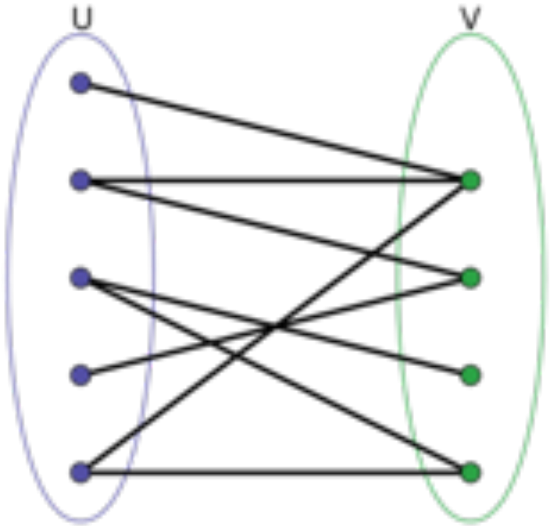
Google Kubernetes Engine

Commercial Data Centers

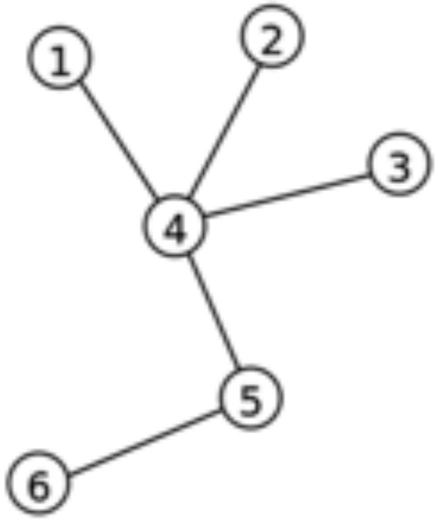
Google Kubernetes Engine



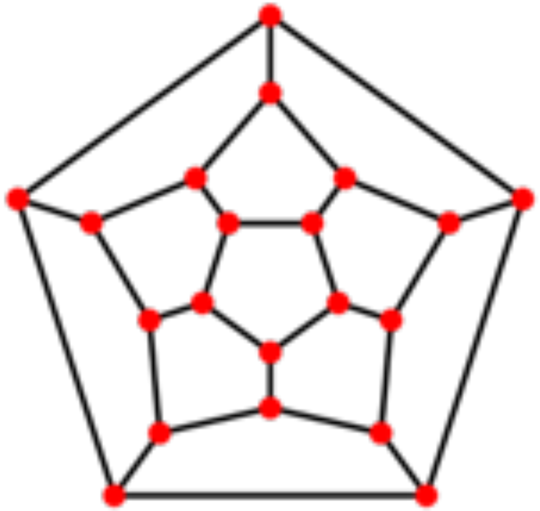
Fast Algorithms on Classes of Graphs



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

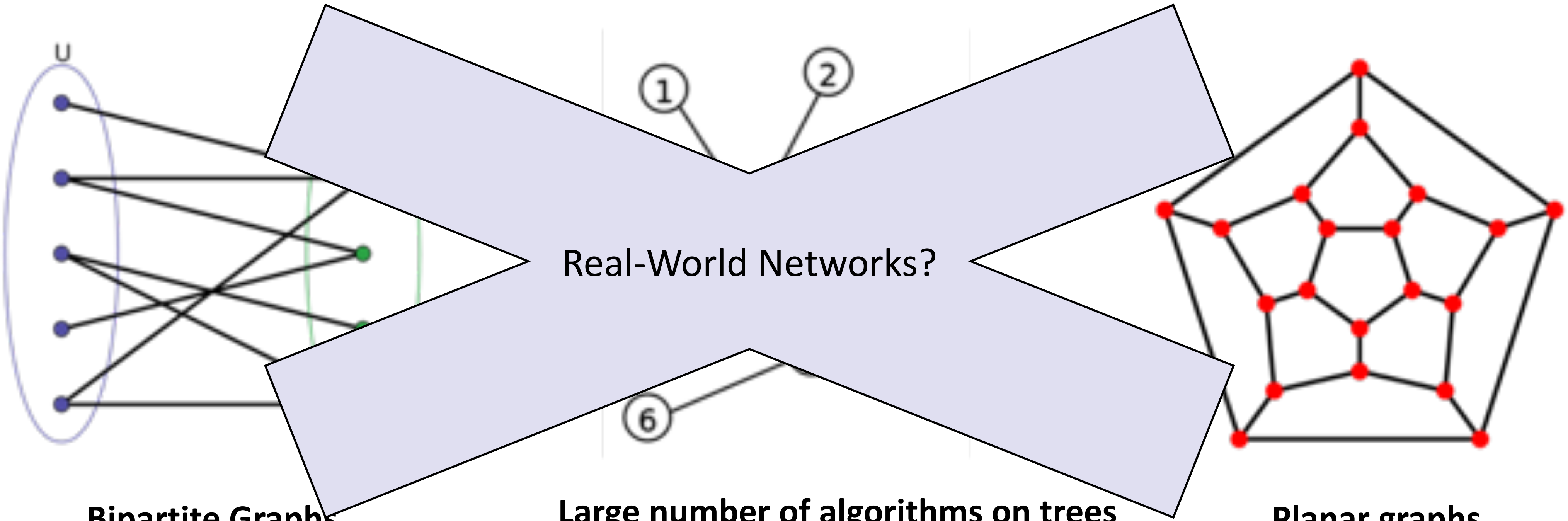


Large number of algorithms on trees
maximum independent set, subgraph isomorphism, dominating set, coloring…etc. all easy



Planar graphs
Single-source shortest path can be linear

Fast Algorithms on Classes of Graphs



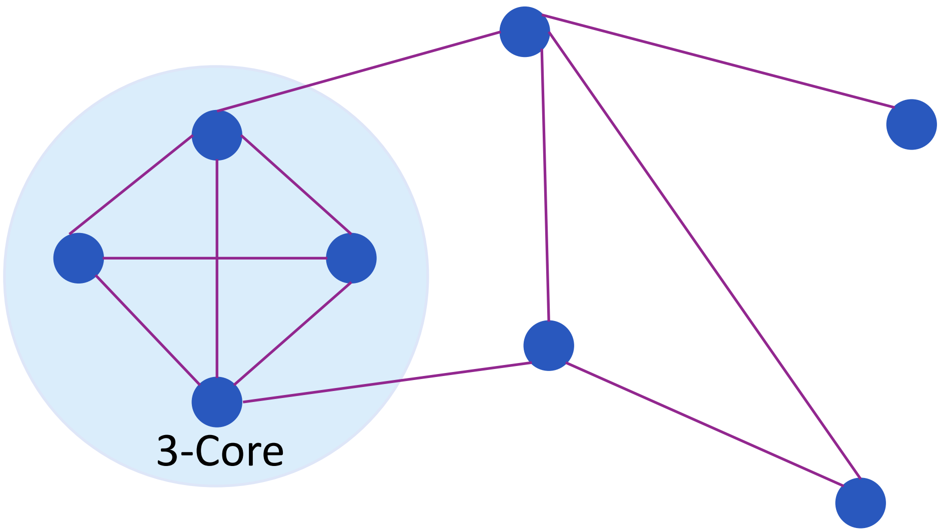
Bipartite Graphs
minimum vertex cover, NP-hard in general

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

Planar graphs
Single-source shortest path can be linear

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

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

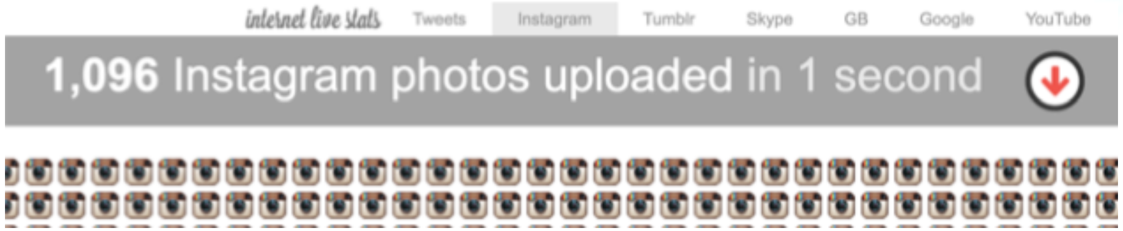
Properties of Real-World Networks

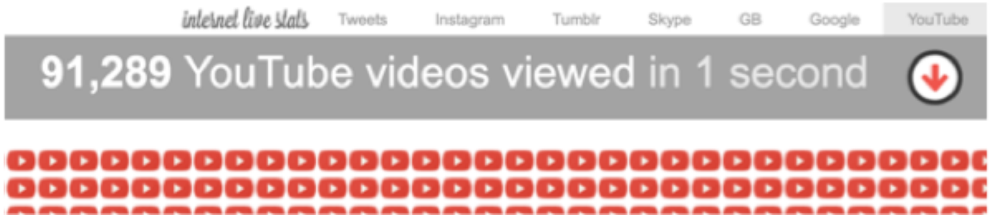
Factor of $\boxed{O(k)}$ vs. $O(n)$ or $O(m)$

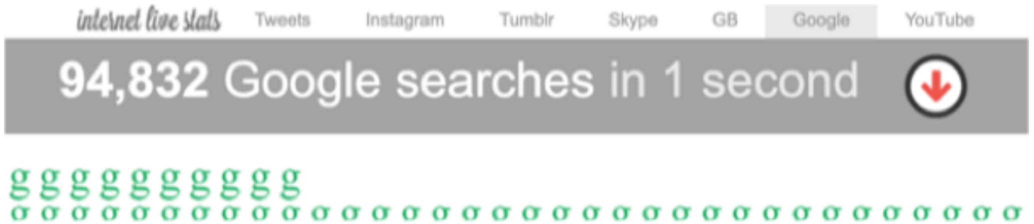
- Degeneracy: a measure of a graph's sparsity

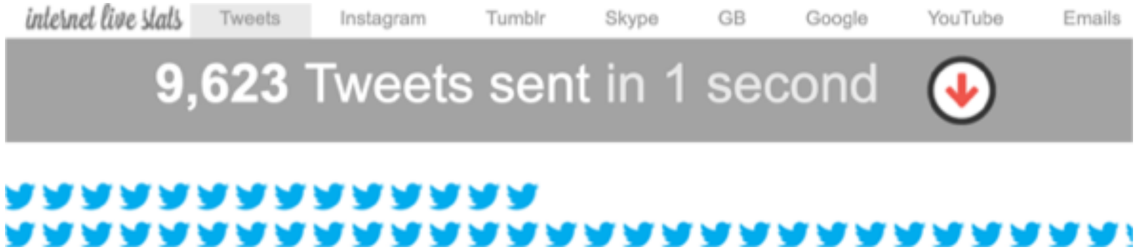
Graph	Num. Vertices	Num. Edges	Degeneracy (*k*)
dblp	425,957	2,099,732	101
brain-network	784,262	267,844,669	1200
wikipedia	1,140,149	2,787,967	124
youtube	1,138,499	5,980,886	51
stackoverflow	2,601,977	28,183,518	163
livejournal	4,847,571	85,702,474	329
orkut	3,072,627	234,370,166	253
usa-road	23,072,627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

Dynamic Graphs



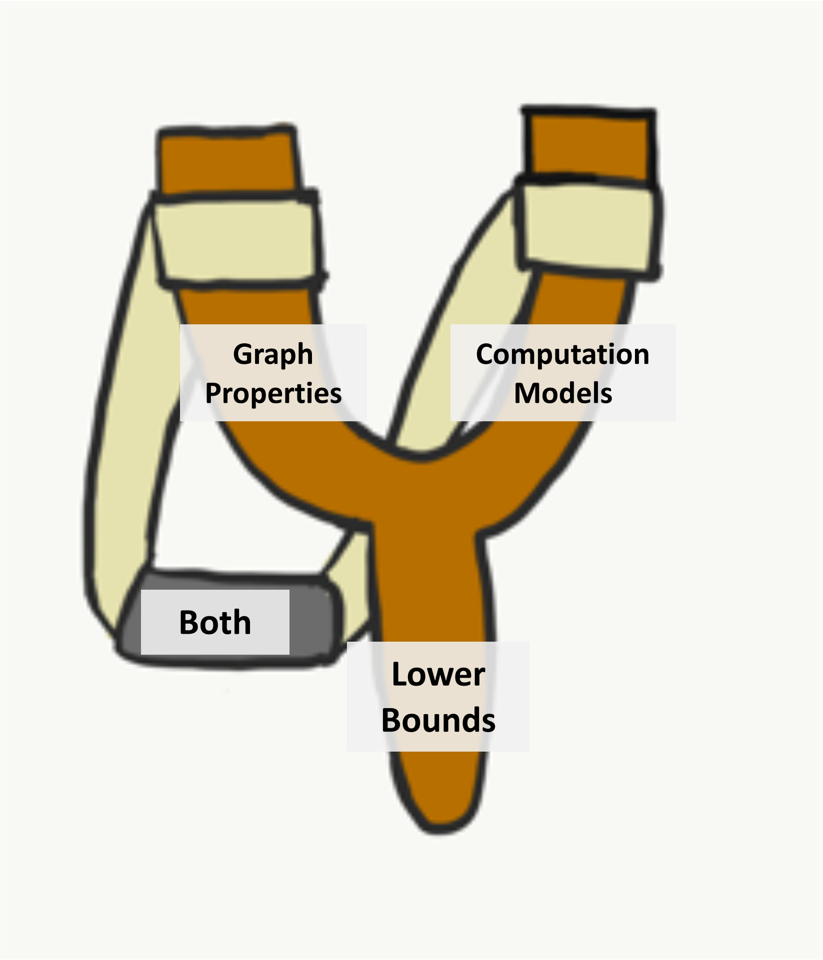






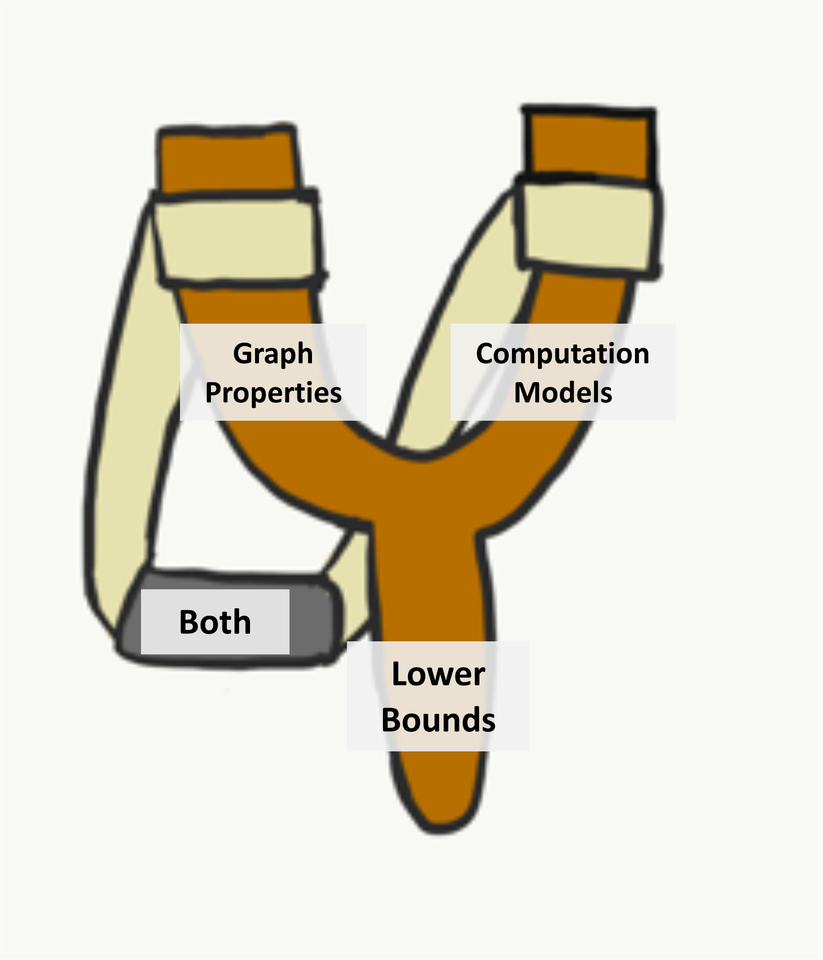
Graph Algorithms for Modern Machines

- 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

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§] Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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

Summary of Results

Static Algorithms

Structural Rounding

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§] Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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

Summary of Results

Static Algorithms

Structural Rounding

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§] Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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

Summary of Results

Static Algorithms

Structural Rounding

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§] Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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

Summary of Results

Static Algorithms

Structural Rounding

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§] Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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 Batch-Dynamic *k*-Core Decomposition

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

$\boxed{\text{Parallel}}$ Batch-Dynamic *k*-Core Decomposition

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

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

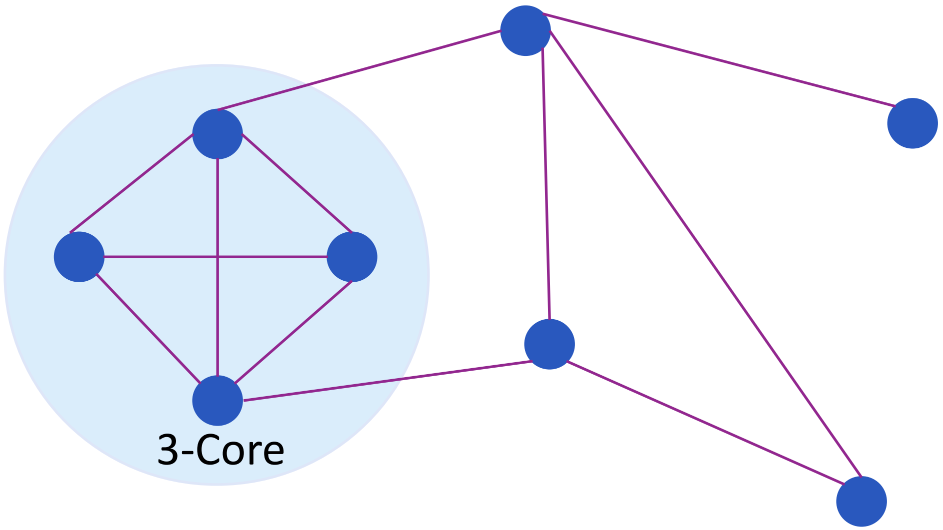
Jessica Shi

Shangdi Yu

Laxman Dhulipala

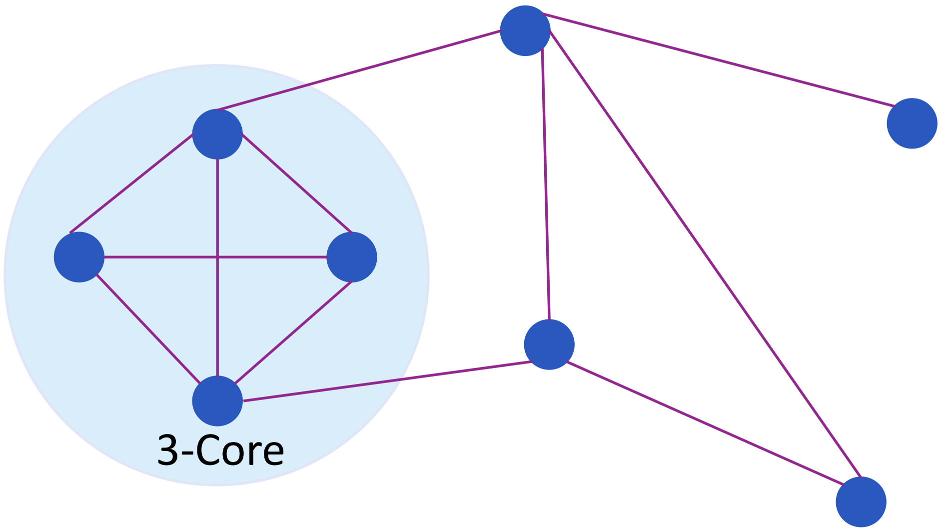
Julian Shun

k-Core Decomposition



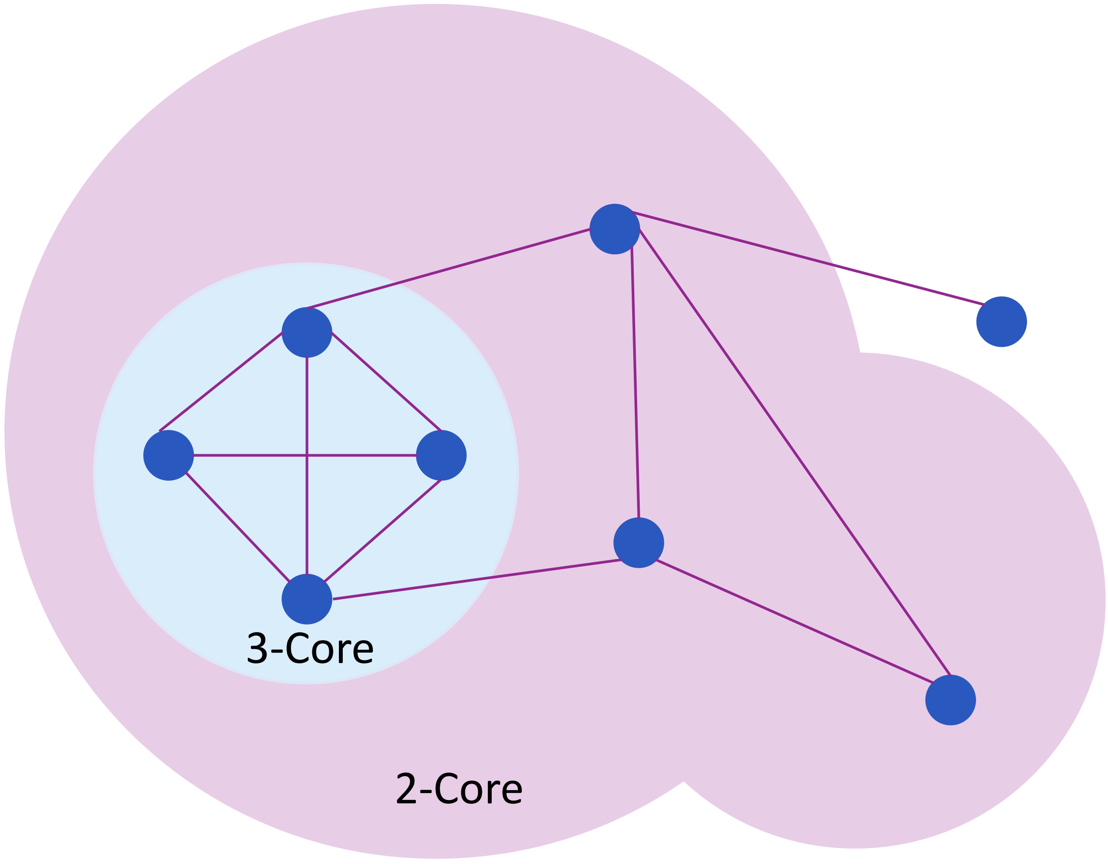
k-Core Decomposition

Coreness or Core Number of Node v: **Maximum Core Value** of a Core Containing v



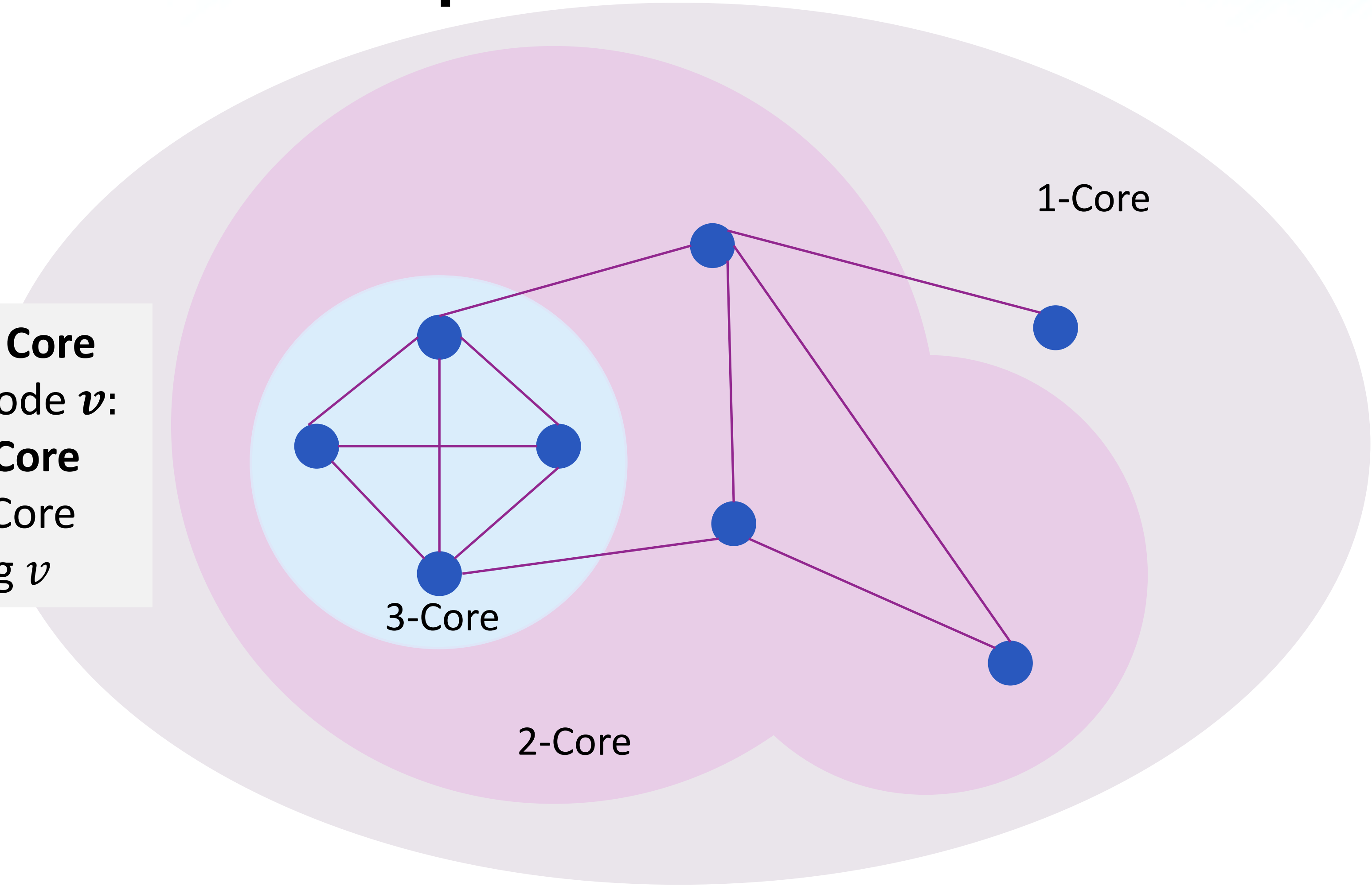
k-Core Decomposition

Coreness or Core Number of Node v: **Maximum Core Value** of a Core Containing v



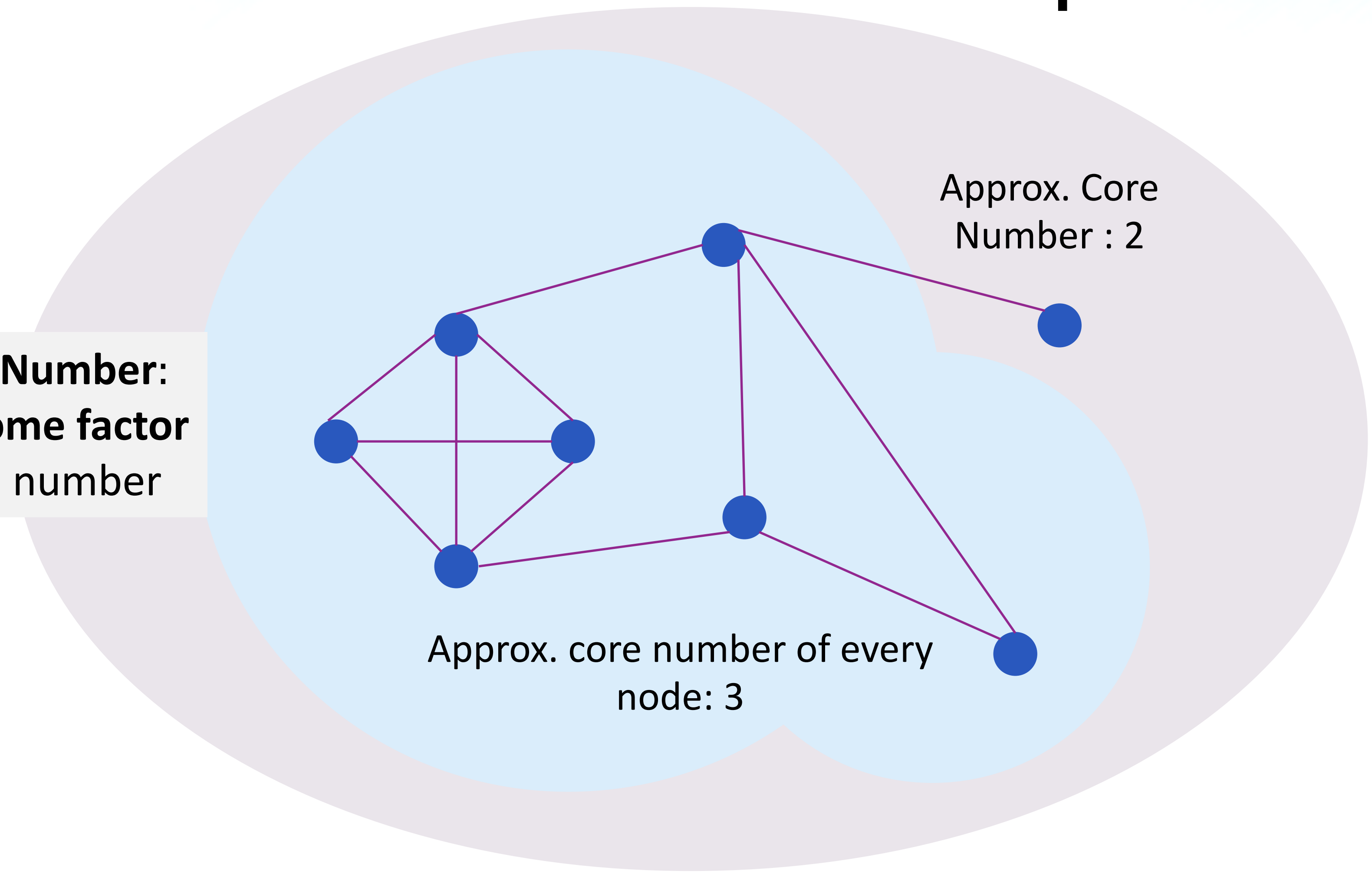
k-Core Decomposition

Coreness or Core Number of Node v: **Maximum Core Value** of a Core Containing v



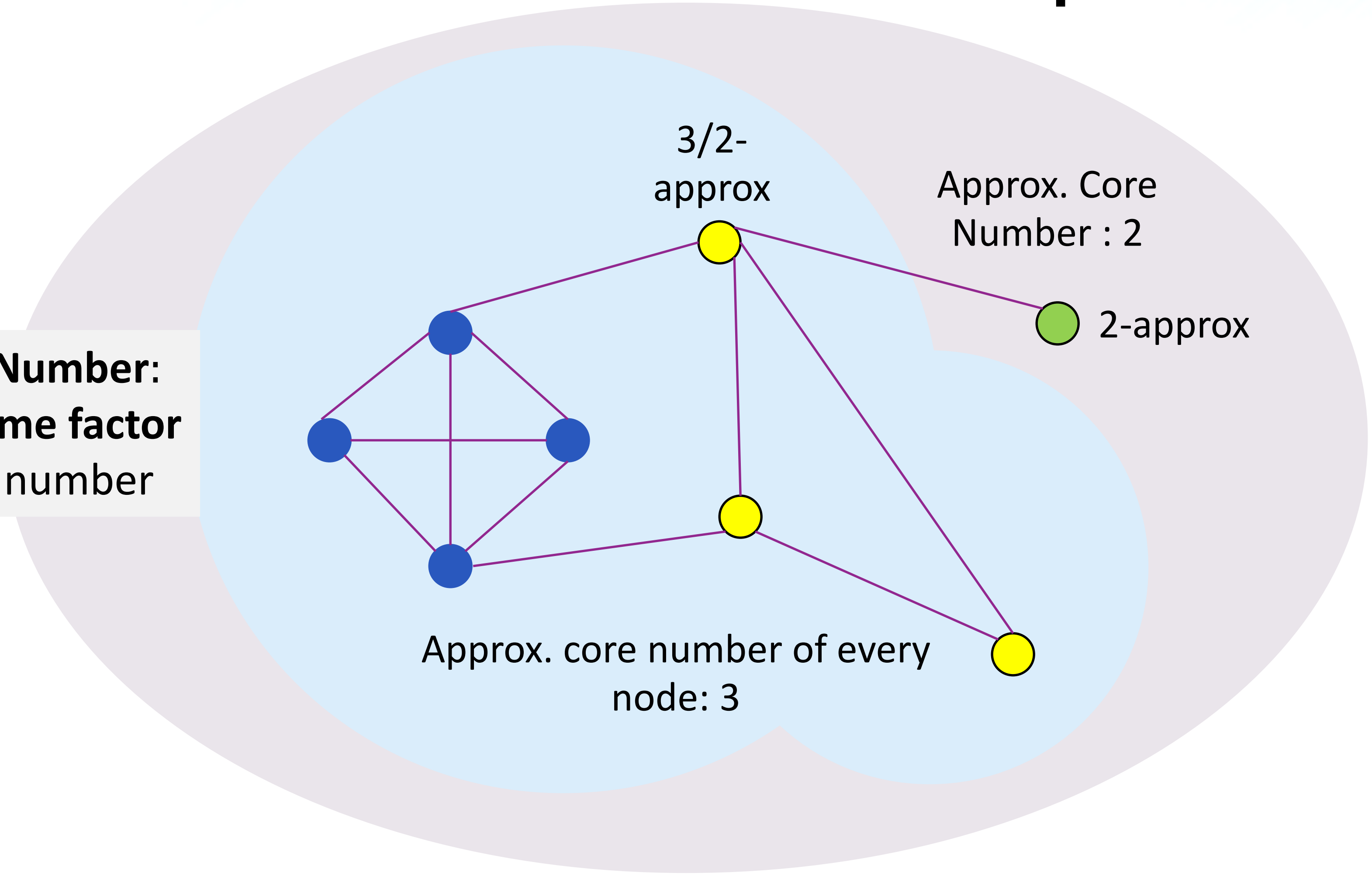
Approximate *k*-Core Decomposition

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



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

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

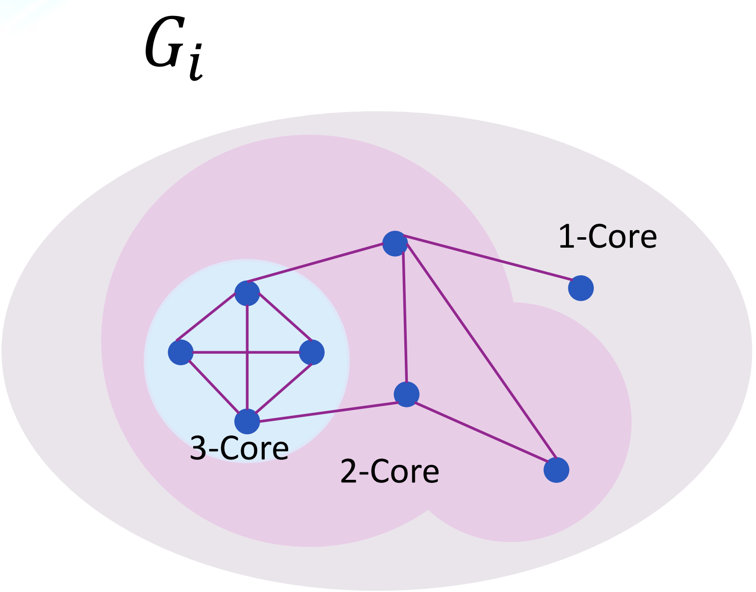
Jessica Shi

Shangdi Yu

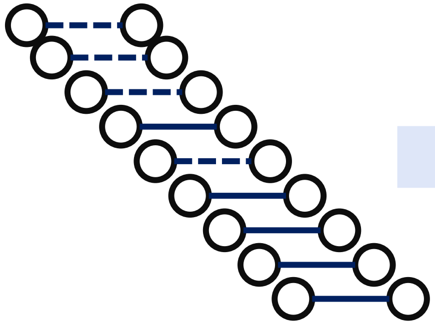
Laxman Dhulipala

Julian Shun

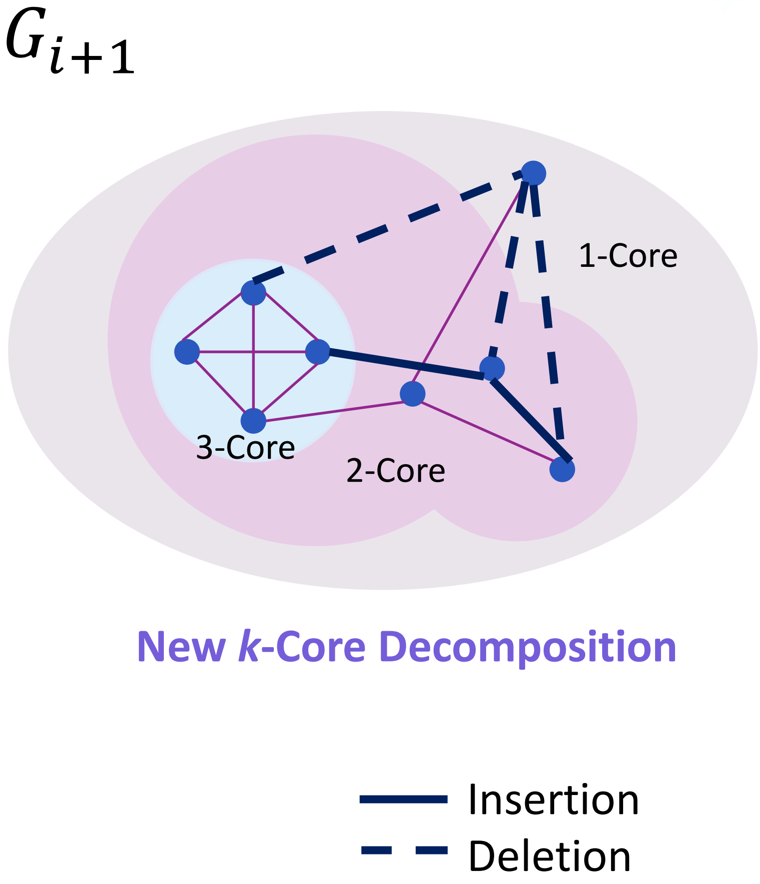
Batch-Dynamic Model Definition [BW09, DDKPSS20]



Initial *k*-Core Decomposition



***B* Edge Insertions/Deletions**



New *k*-Core Decomposition

Parallel Batch-Dynamic *k*-Core Decomposition

Parallel Batch-Dynamic k-Core Decomposition and Low Out-Degree Orientation

Quanquan C. Liu, Jessica Shi, Shangdi Yu, Laxman Dhulipala, Julian Shun

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.

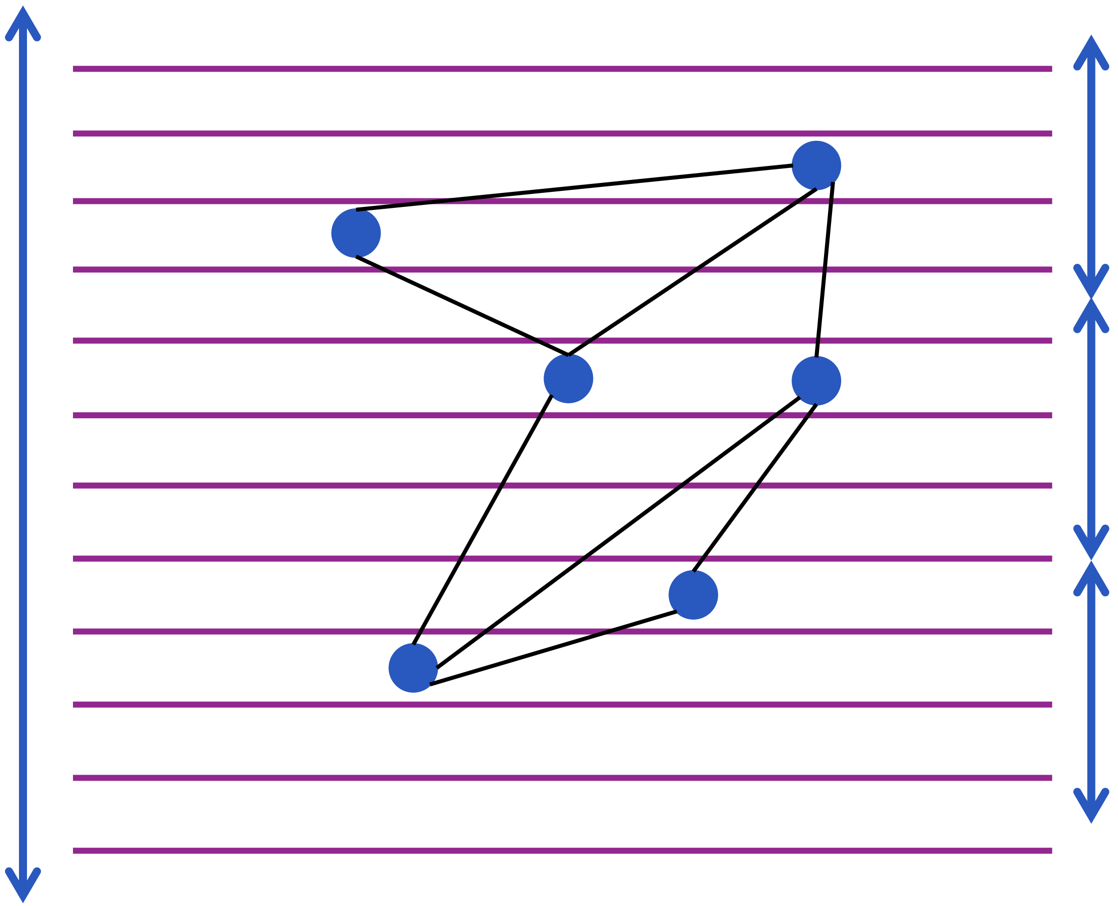
Level Data Structures

- 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)

$O(\log^2 n)$

Vertices partitioned into levels



Group of $O(\log n)$ levels

Cut-off: $(1+\epsilon)^i$

Group of $O(\log n)$ levels

Cut-off: $(1+\epsilon)^{i-1}$

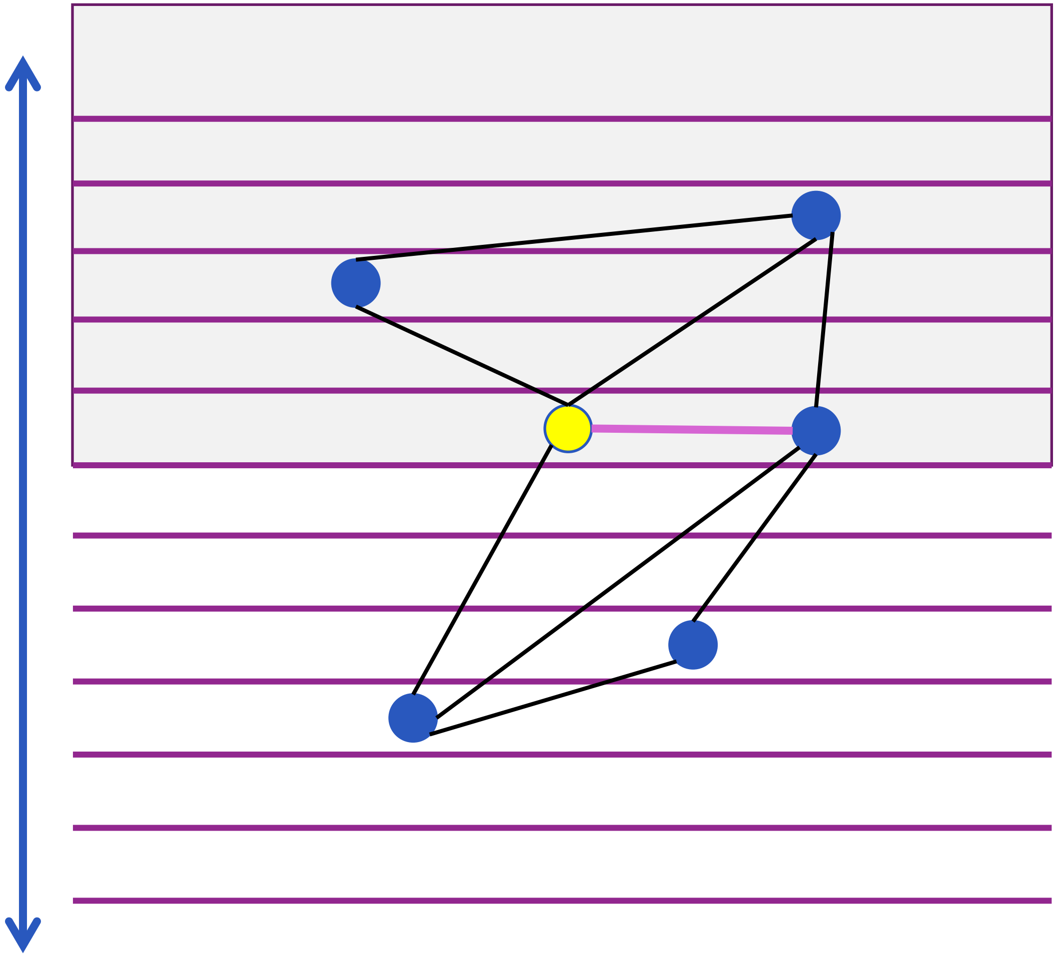
Group of $O(\log n)$ levels

Cut-off: $(1+\epsilon)^{i-2}$

...

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)



$O(\log^2 n)$

Vertices partitioned into levels

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

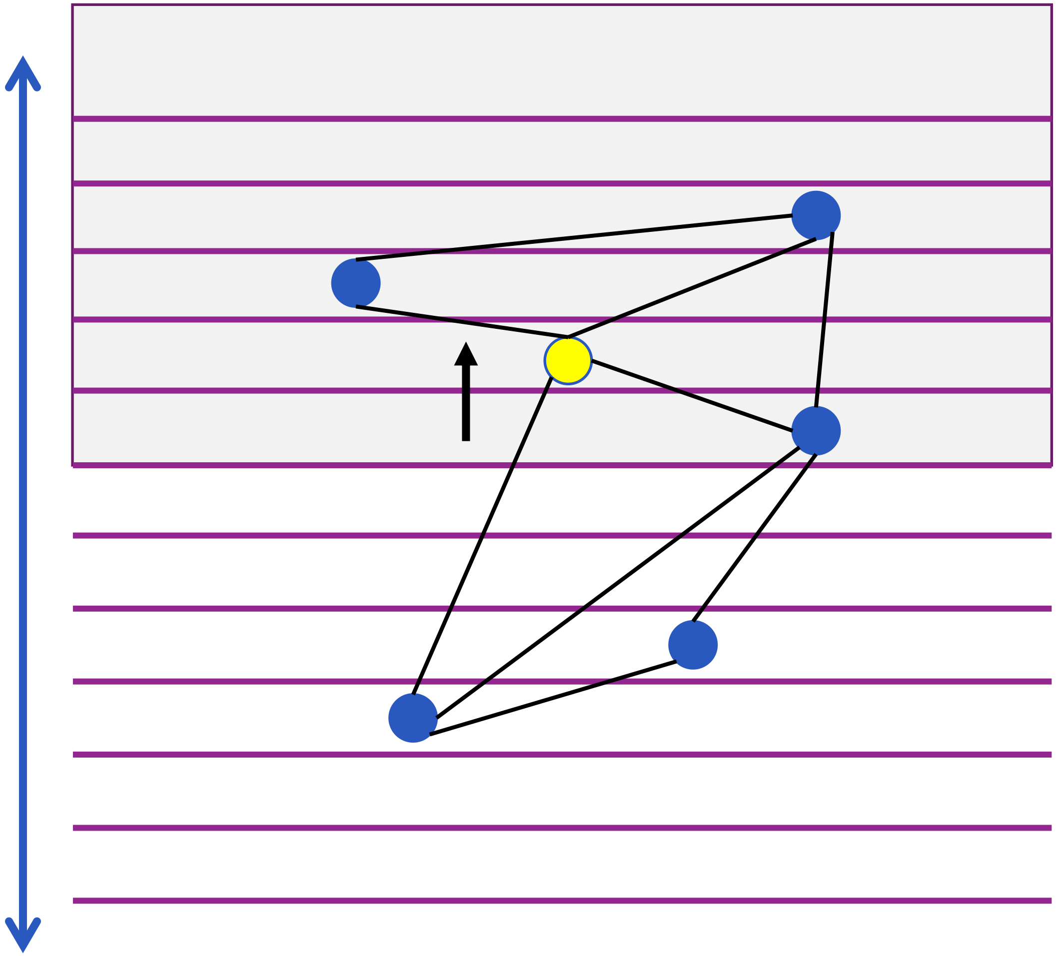
= edge insertion

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels



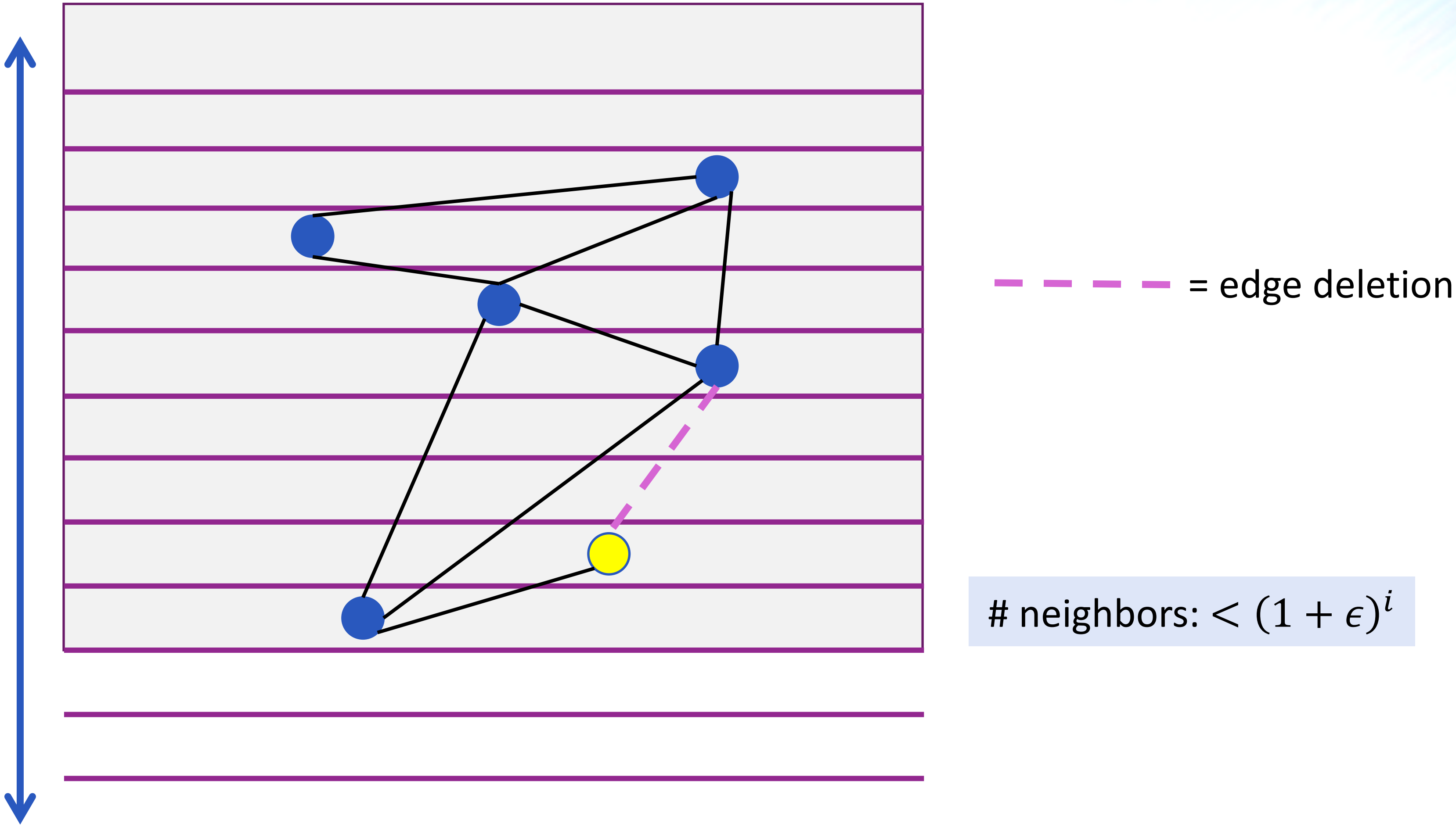
\# neighbors: $> 2.1(1+\epsilon)^i$

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

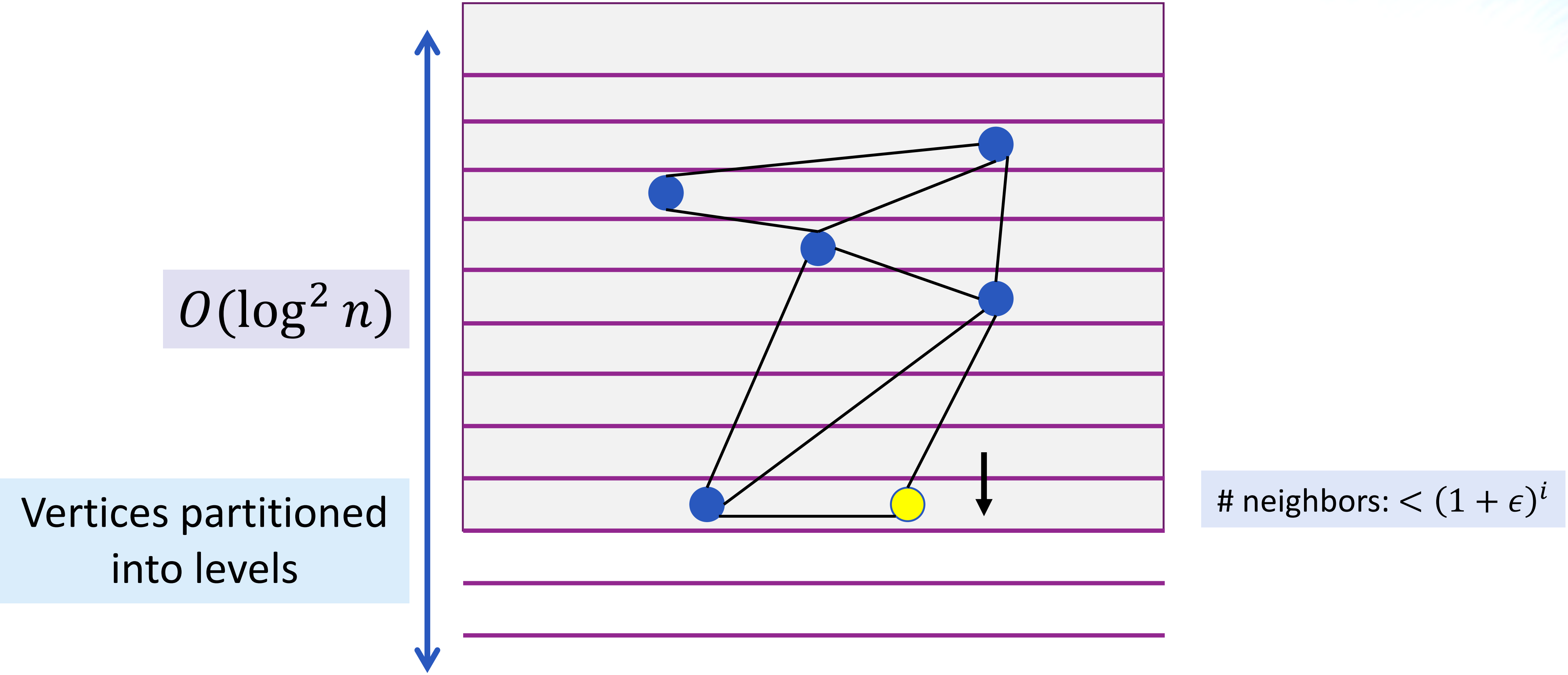
$O(\log^2 n)$

Vertices partitioned into levels



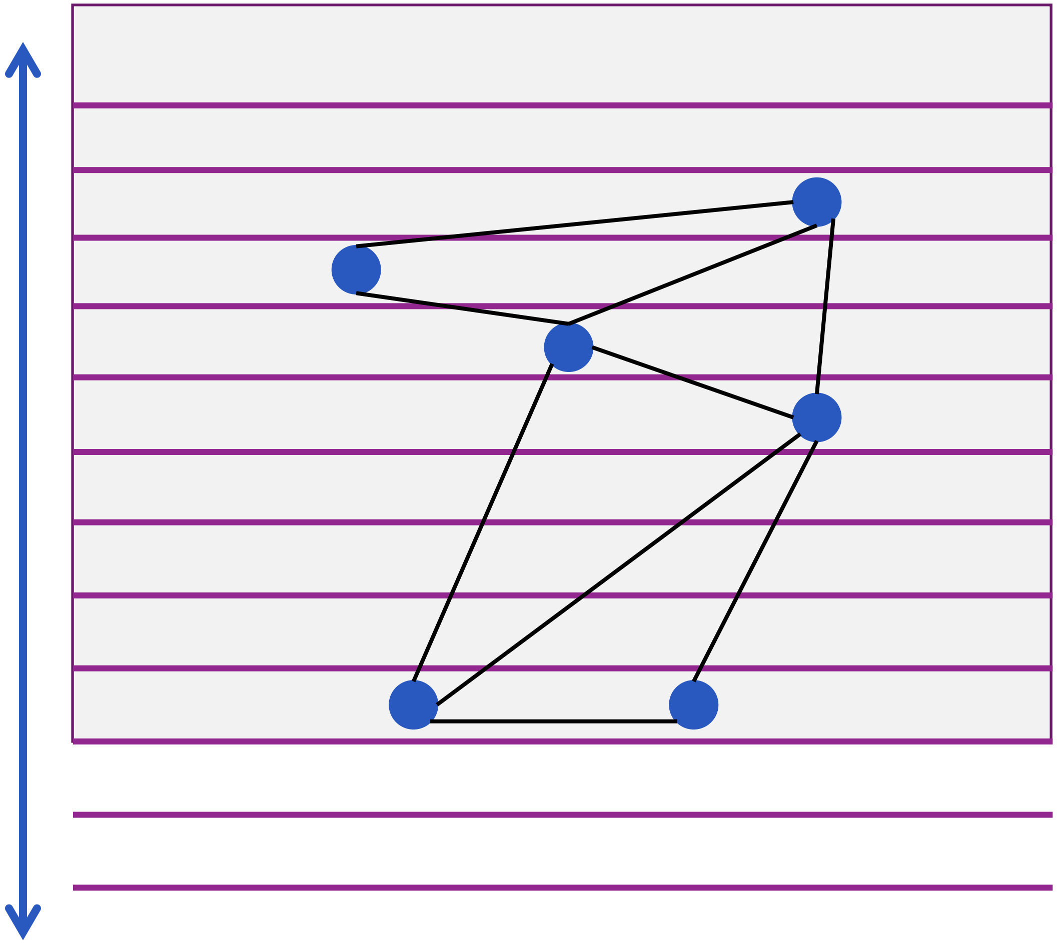
Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)



Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)



$O(\log^2 n)$

Vertices partitioned into levels

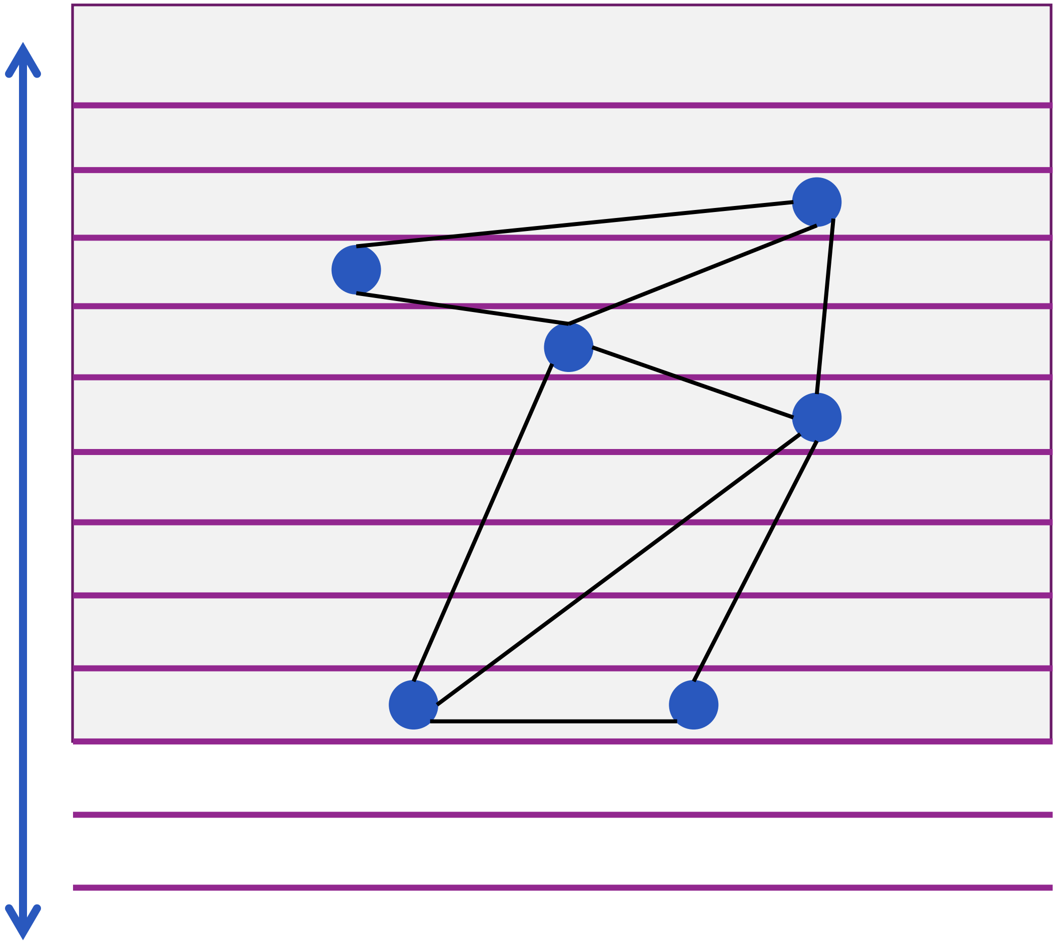
v is stored at a level where
1. Number of neighbors at or above same level is $\leq 2.1(1+\epsilon)^i$
2. Number of neighbors at or above level below v is $\geq (1+\epsilon)^i$

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

Vertices partitioned into levels

$O(\log^2 n)$



v is stored at a level where

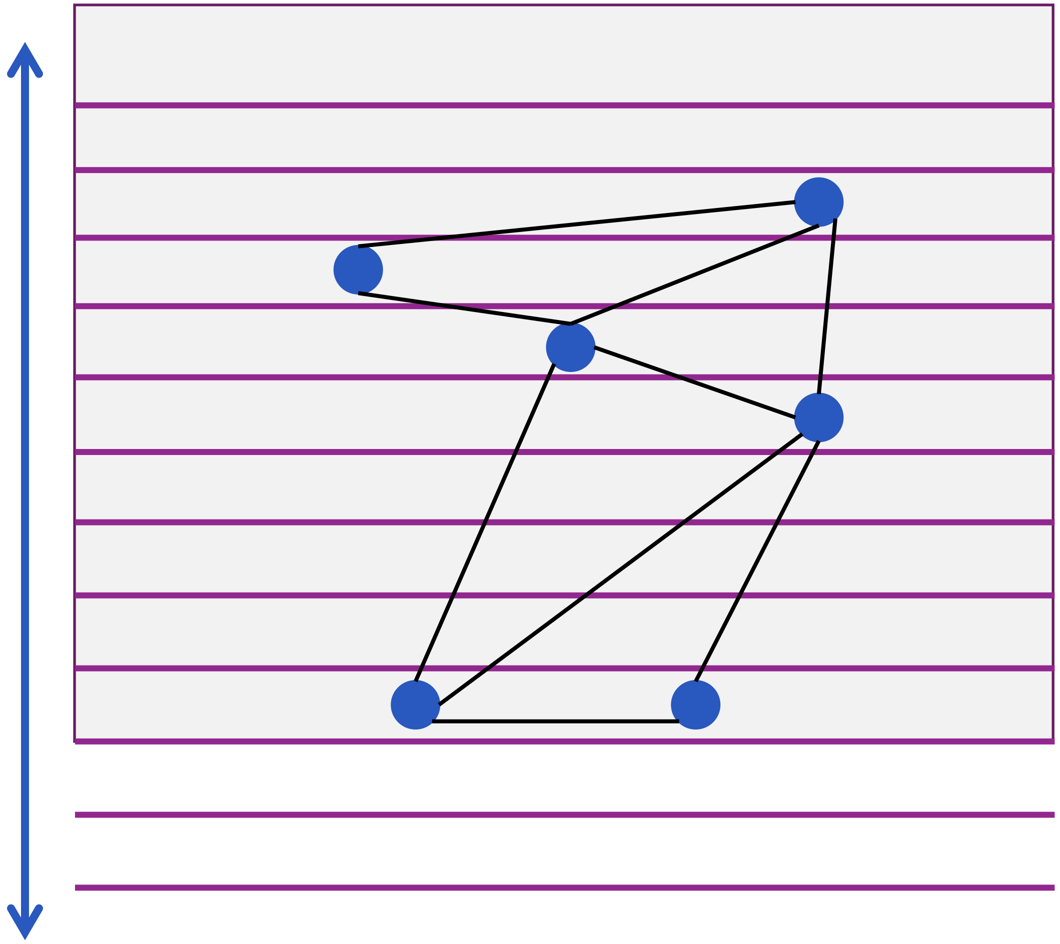
1. Number of neighbors at or above same level is $\leq 2.1(1+\epsilon)^i$
2. Number of neighbors at or above level below v is $\geq (1+\epsilon)^i$

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels

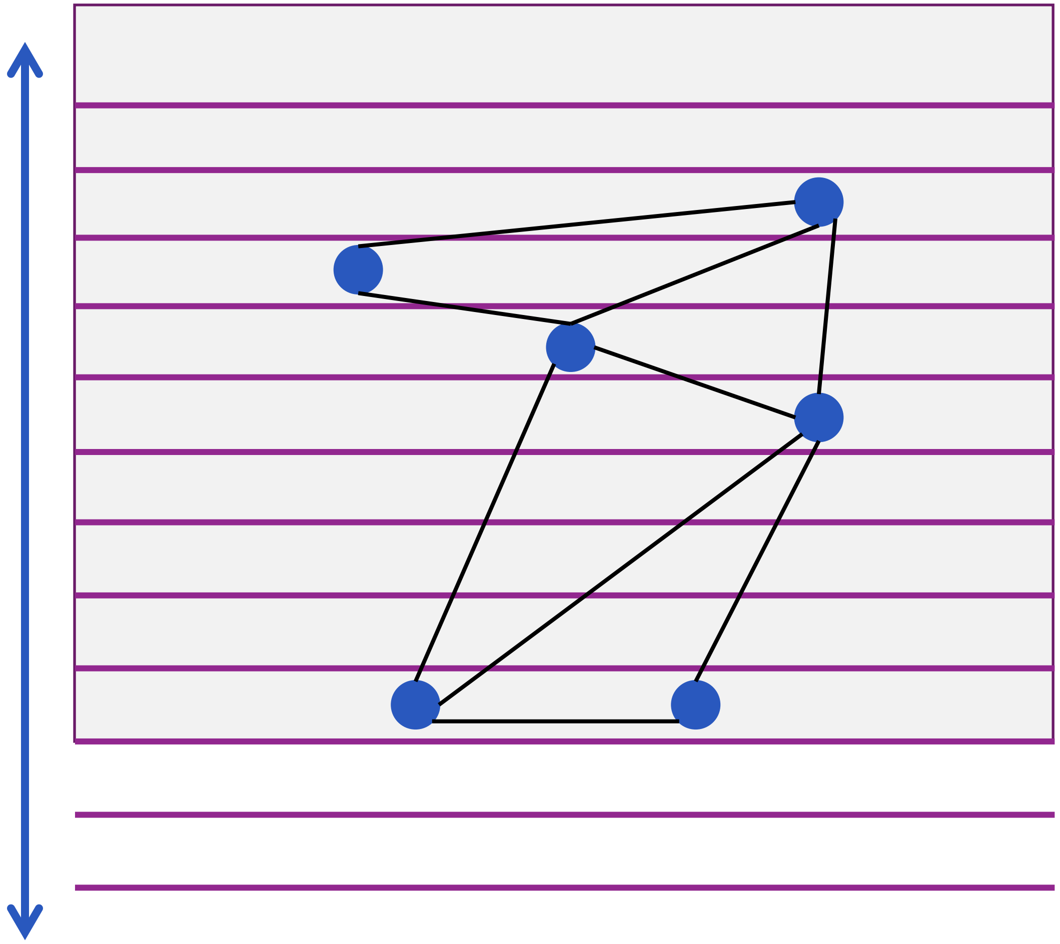


v is stored at a level where

1. Number of neighbors at or above same level is $\leq \boxed{2.1}(1+\epsilon)^i$
2. Number of neighbors at or above level below v is $\boxed{\geq (1+\epsilon)^i}$

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)



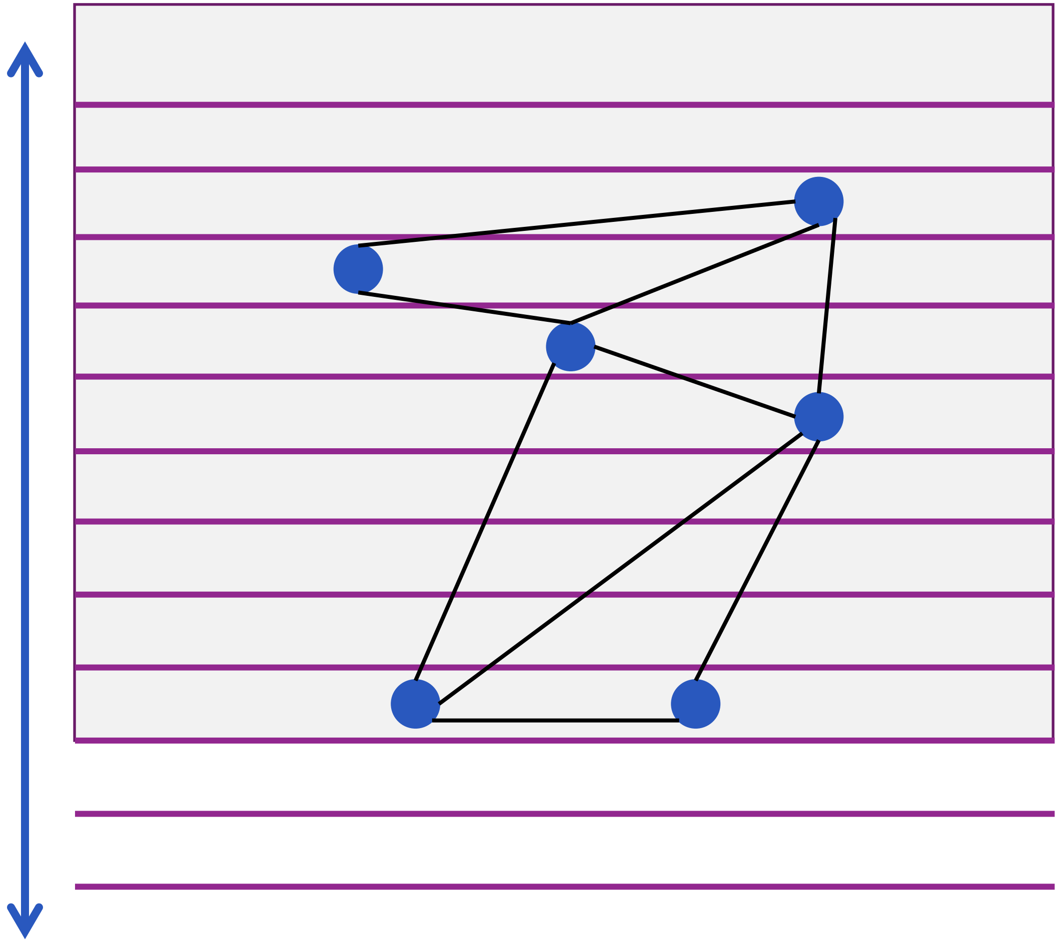
$O(\log^2 n)$

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 $\geq (1+\epsilon)^i$

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)



$O(\log^2 n)$

Vertices partitioned into levels

v is stored at a level where

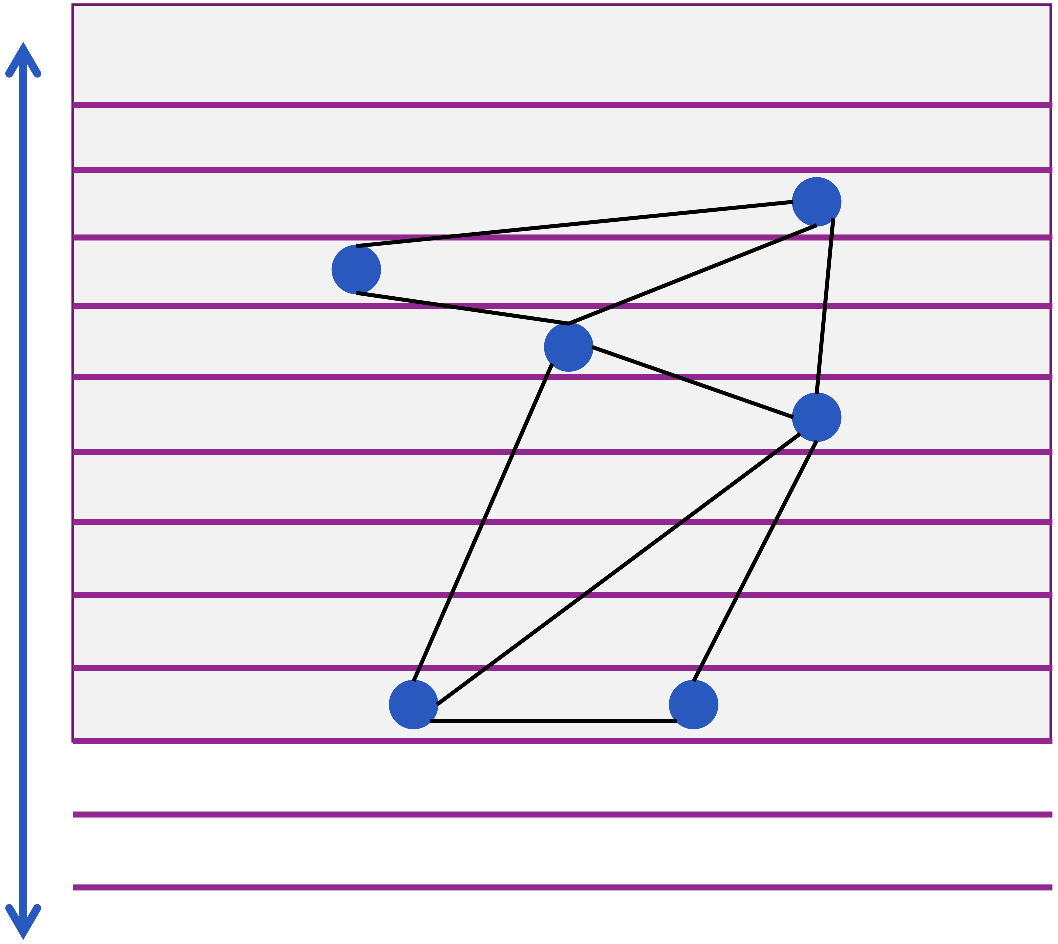
1. Number of neighbors at or above same level is $\leq 2.1\boxed{(1+\epsilon)^i}$
2. Number of neighbors at or above level below v is $\geq \boxed{(1+\epsilon)^i}$

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

$O(\log^2 n)$

Vertices partitioned into levels



v is stored at a level where

1. **Upper Bound Invariant:**

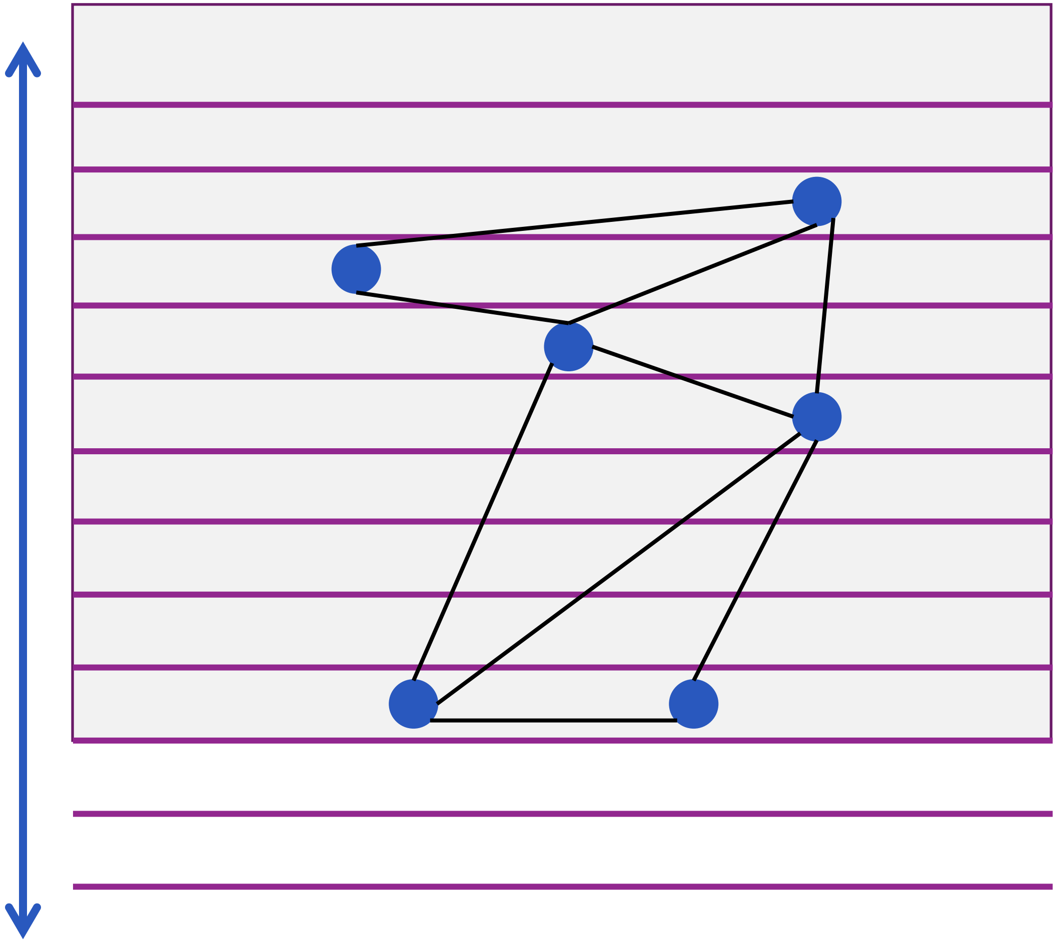
   $\leq 2.1(1+\epsilon)^i$

2. **Lower Bound Invariant:**

   $\geq (1+\epsilon)^i$

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)



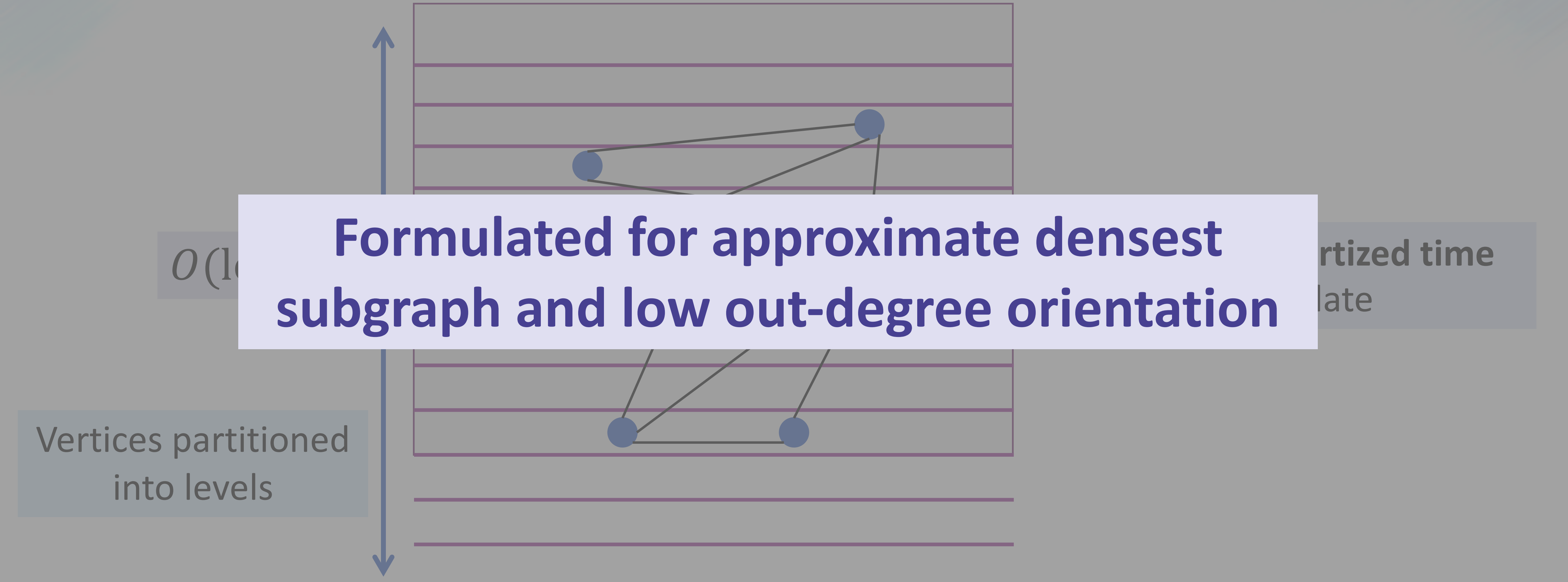
$O(\log^2 n)$

Vertices partitioned into levels

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

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Sequential Level Data Structure (LDS)

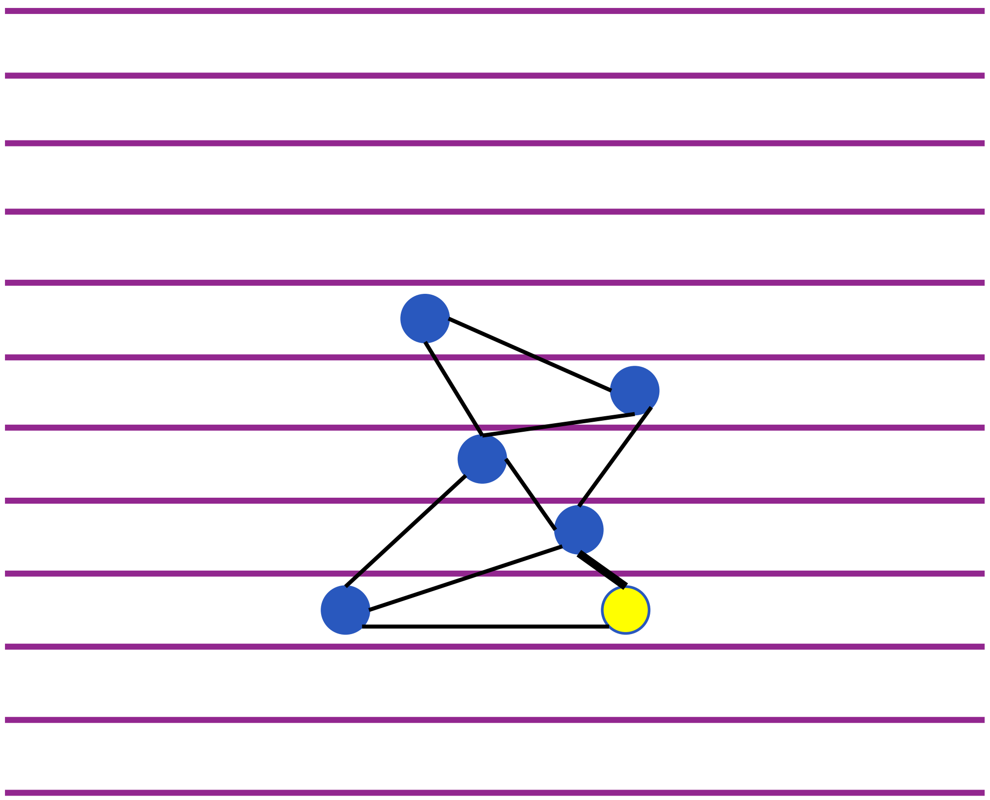


Formulated for approximate densest subgraph and low out-degree orientation

Bhattacharya-Henzinger-Nanongkai-Tsourakakis STOC 2015 and Henzinger-Neumann-Weiss 2020

Difficulties with Parallelization

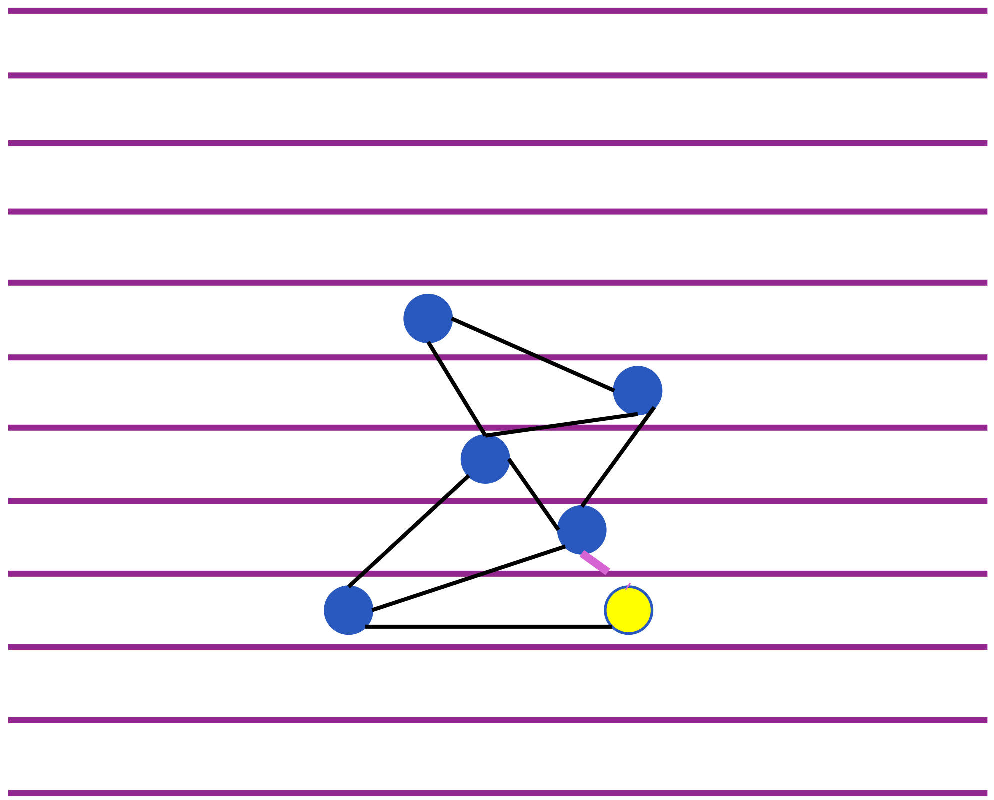
Large sequential dependencies



Large depth

Difficulties with Parallelization

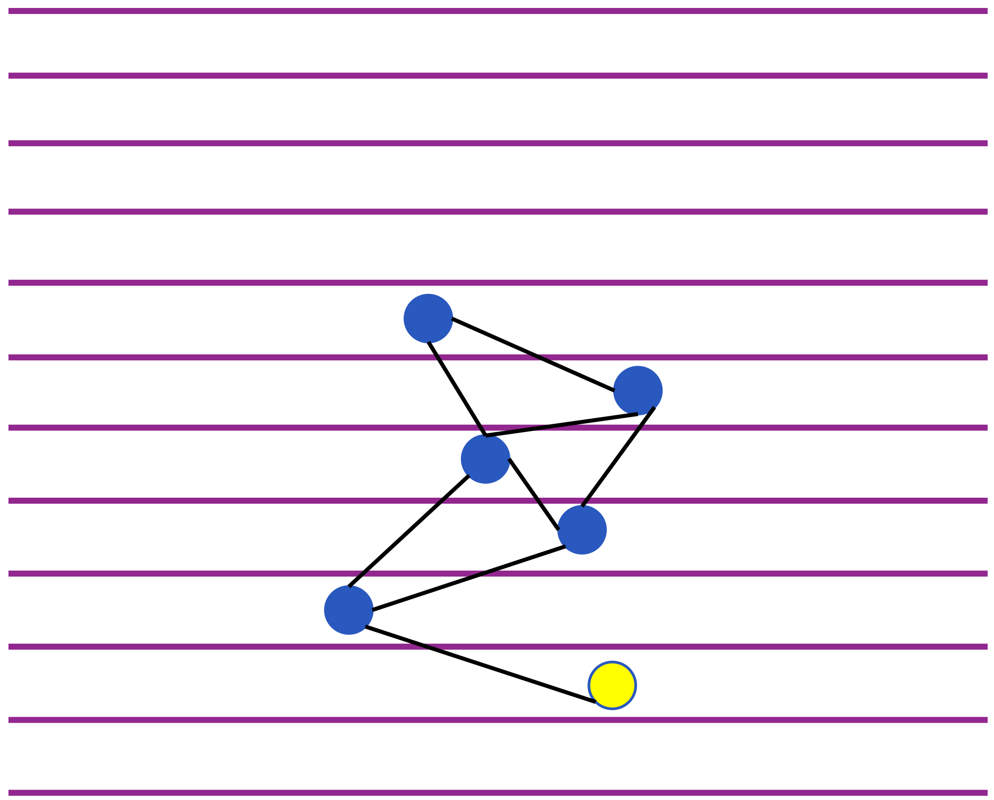
Large sequential dependencies



Large depth

Difficulties with Parallelization

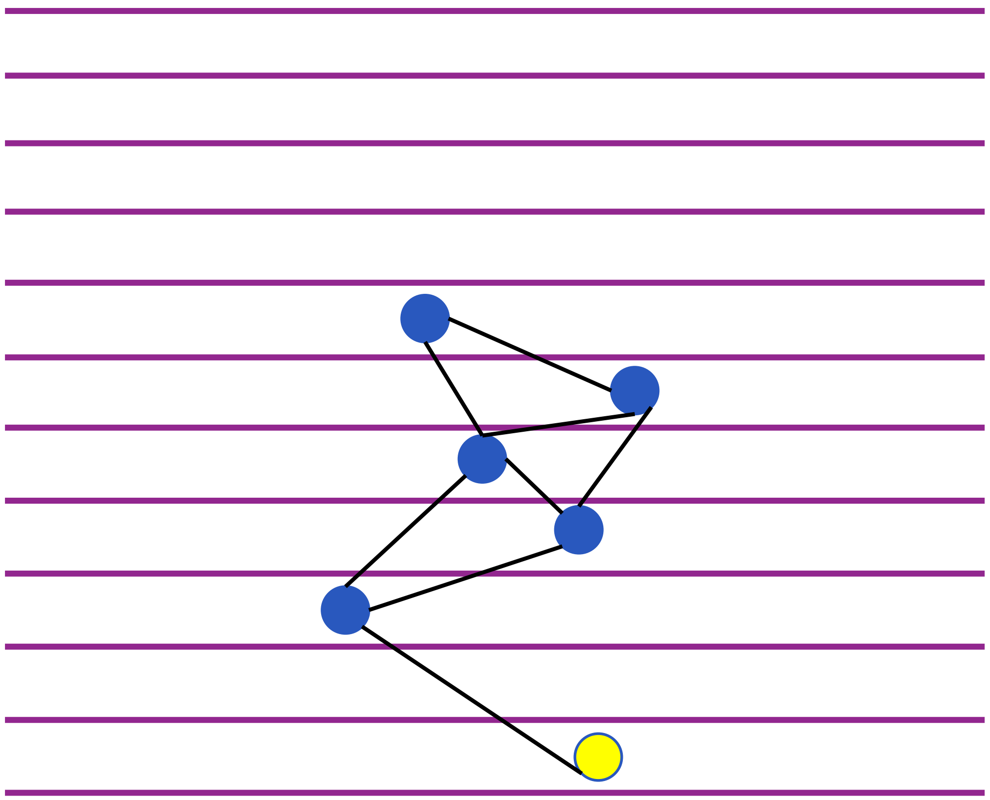
Large sequential dependencies



Large depth

Difficulties with Parallelization

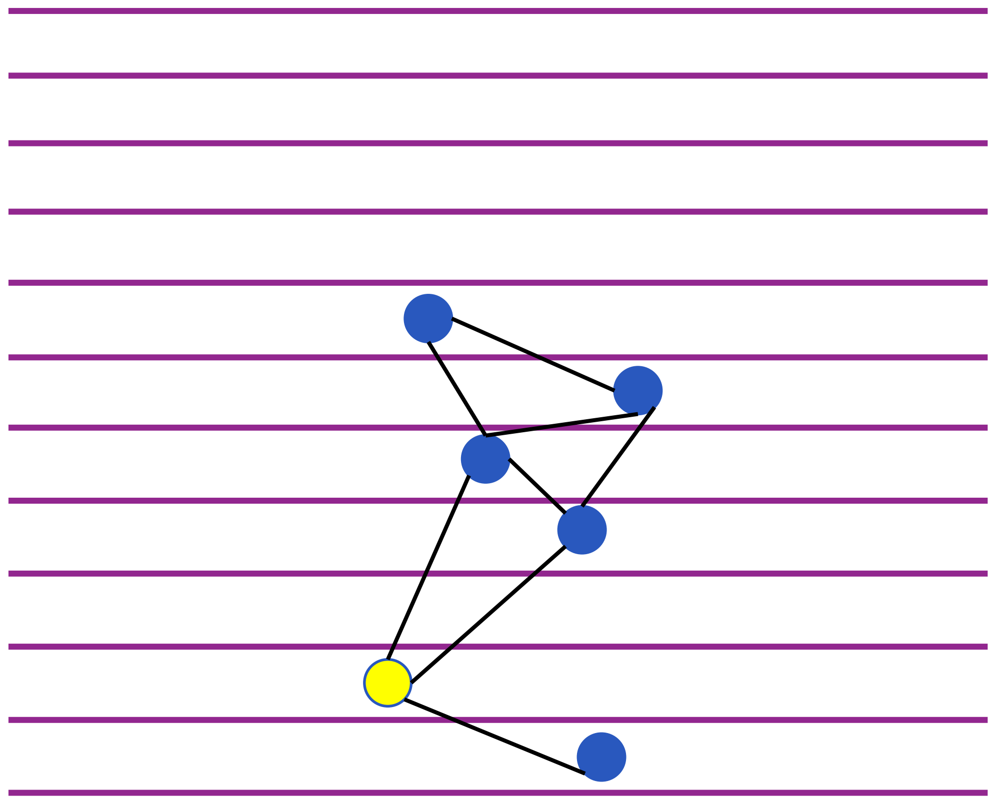
Large sequential dependencies



Large depth

Difficulties with Parallelization

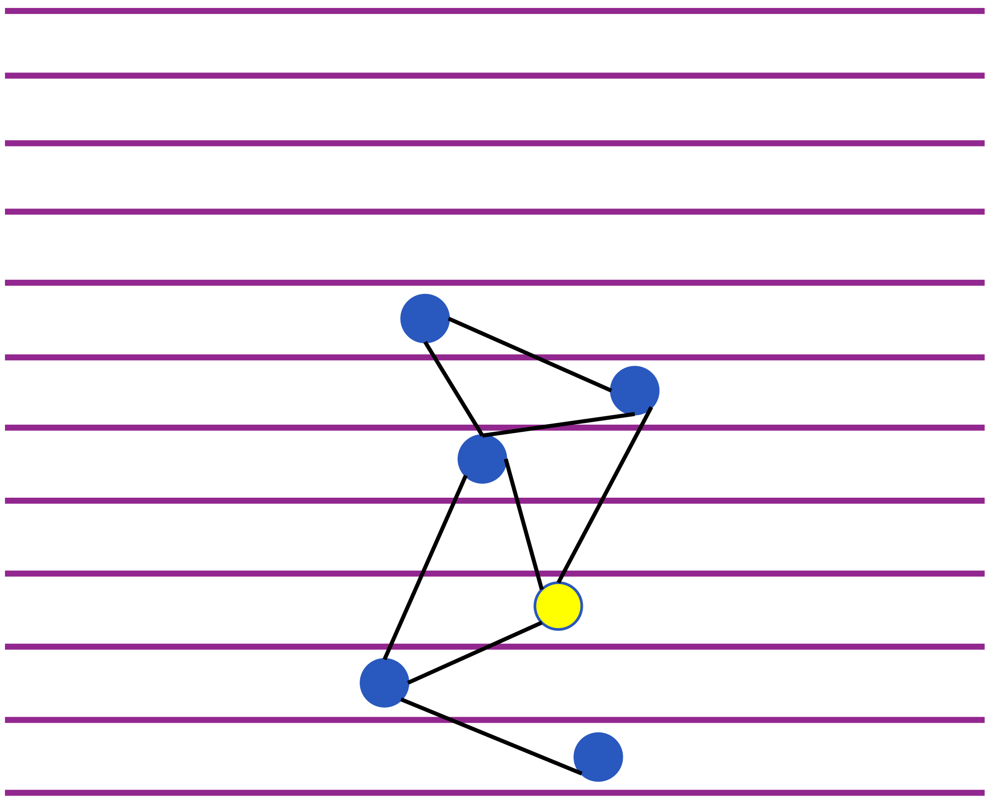
Large sequential dependencies



Large depth

Difficulties with Parallelization

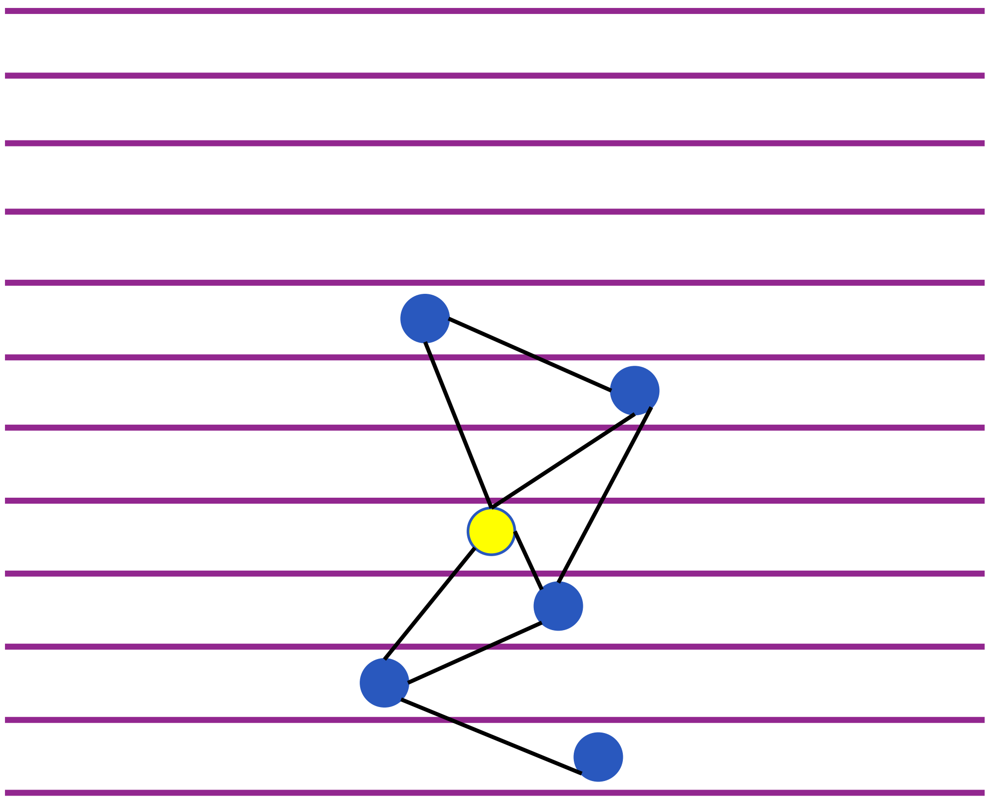
Large sequential dependencies



Large depth

Difficulties with Parallelization

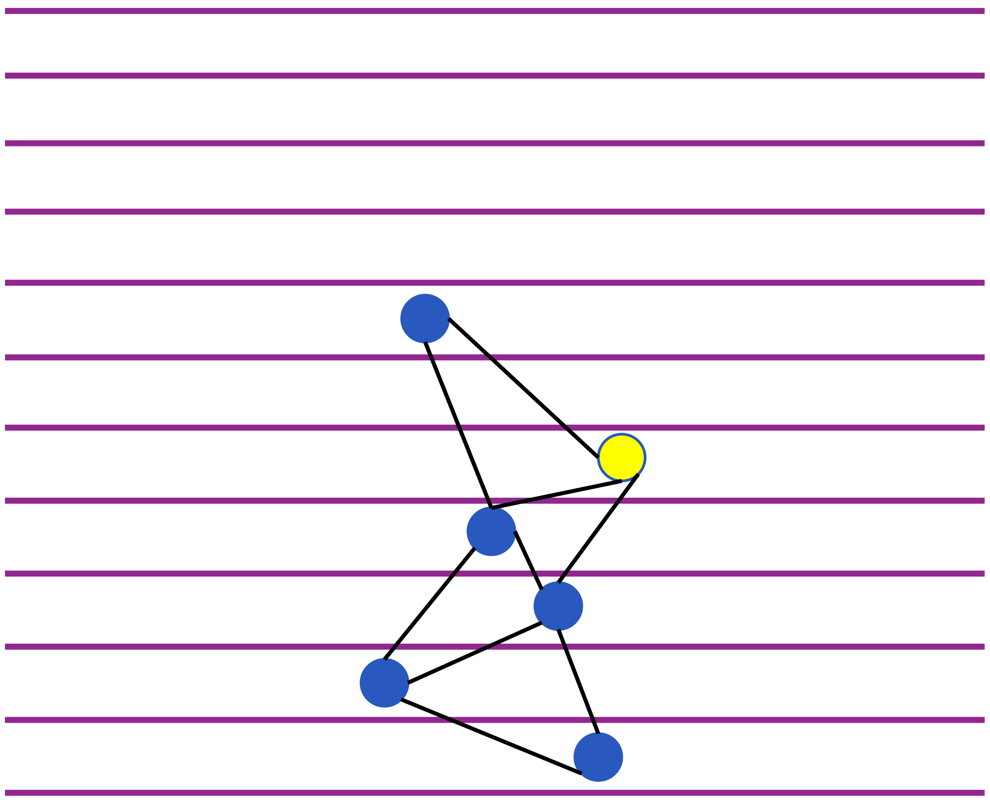
Large sequential dependencies



Large depth

Difficulties with Parallelization

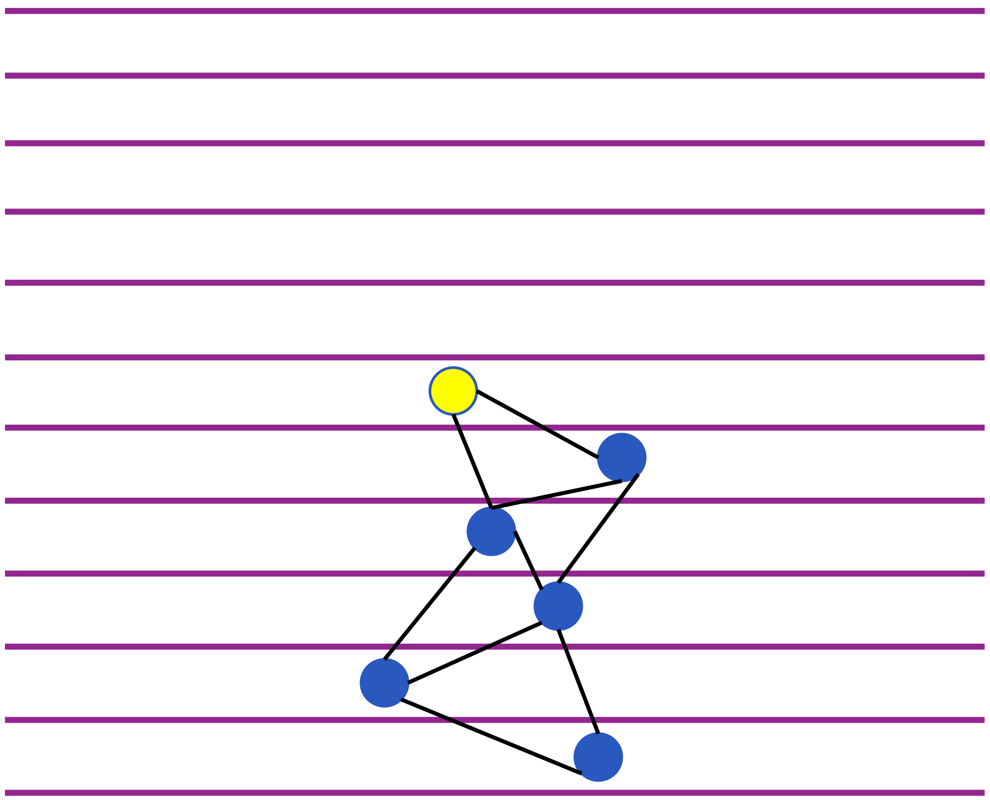
Large sequential dependencies



Large depth

Difficulties with Parallelization

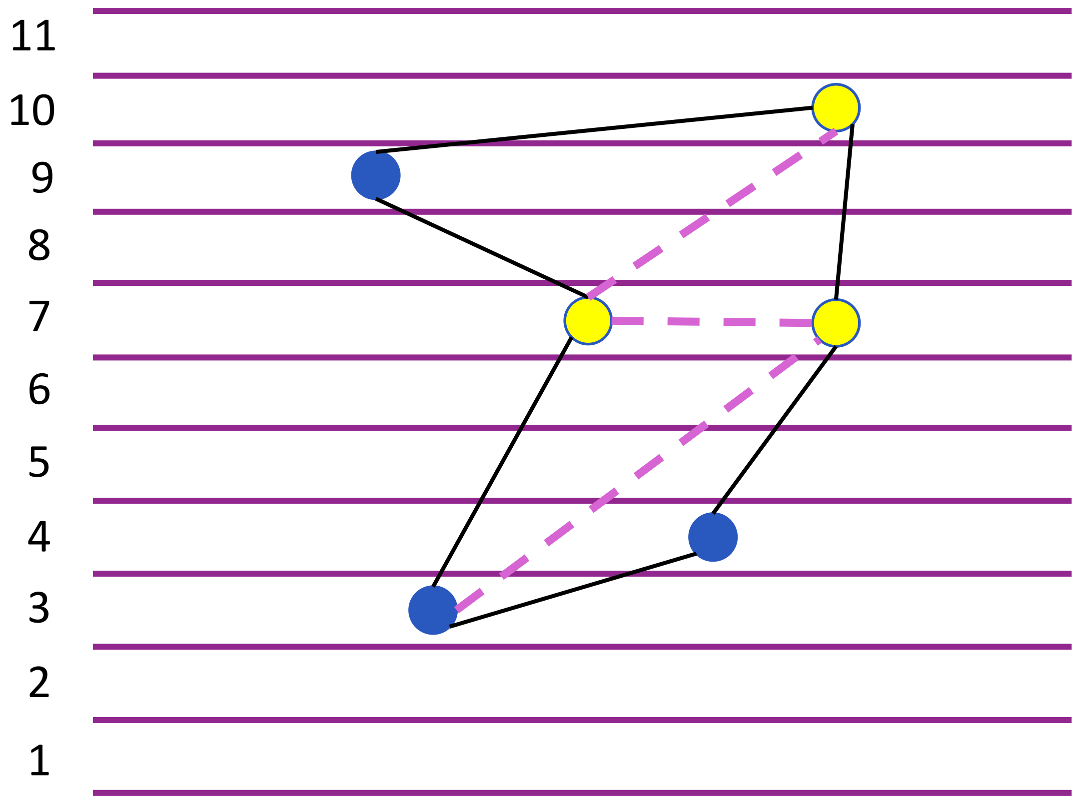
Large sequential dependencies



Large depth

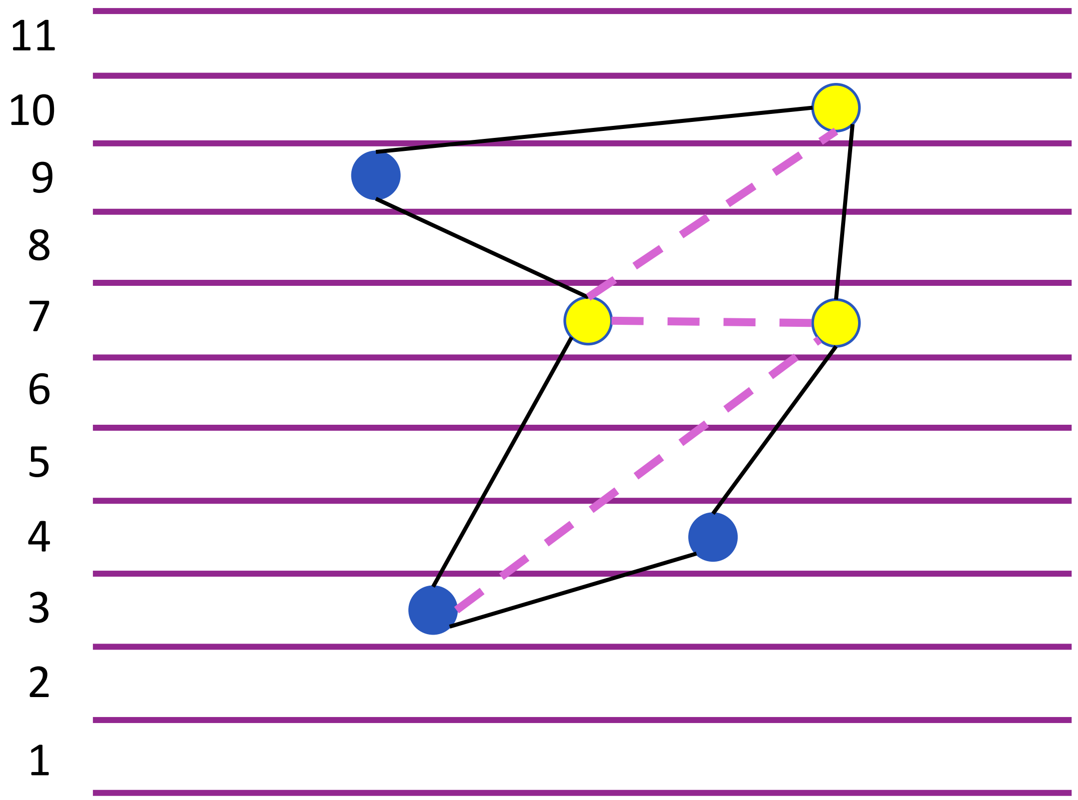
Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions



Our Parallel Batch-Dynamic Level Data Structure (PLDS)

Deletions

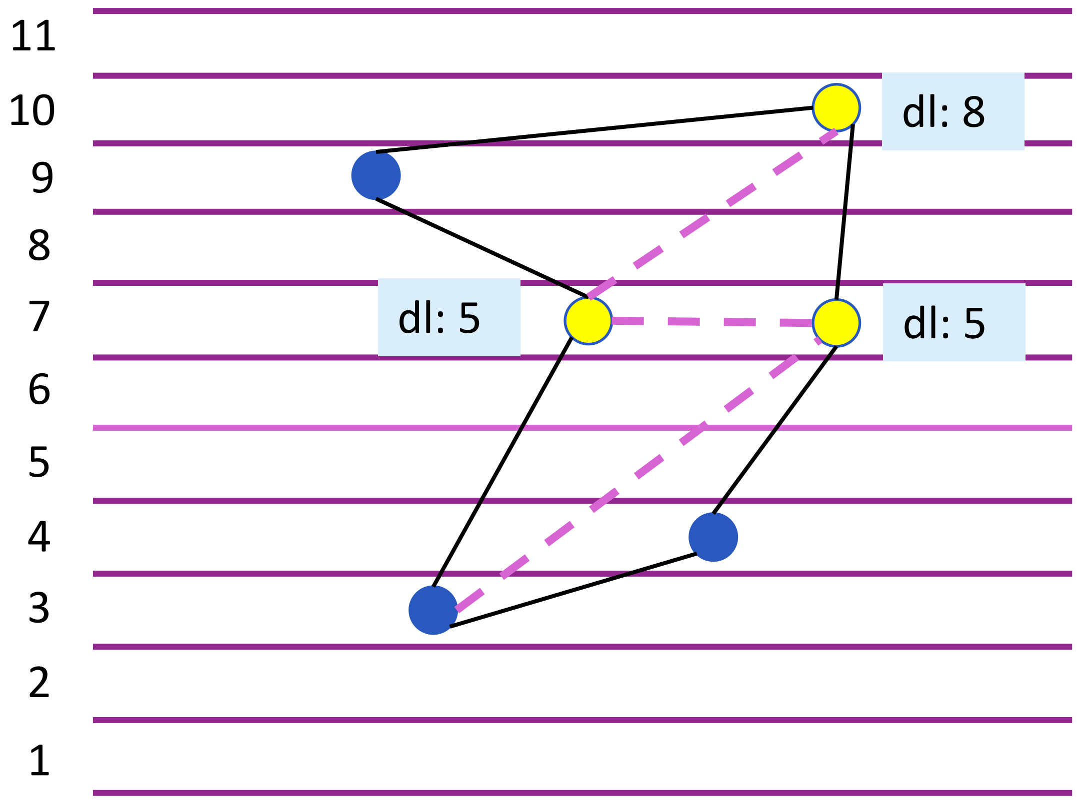


Only lower bound invariant, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure

Deletions

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

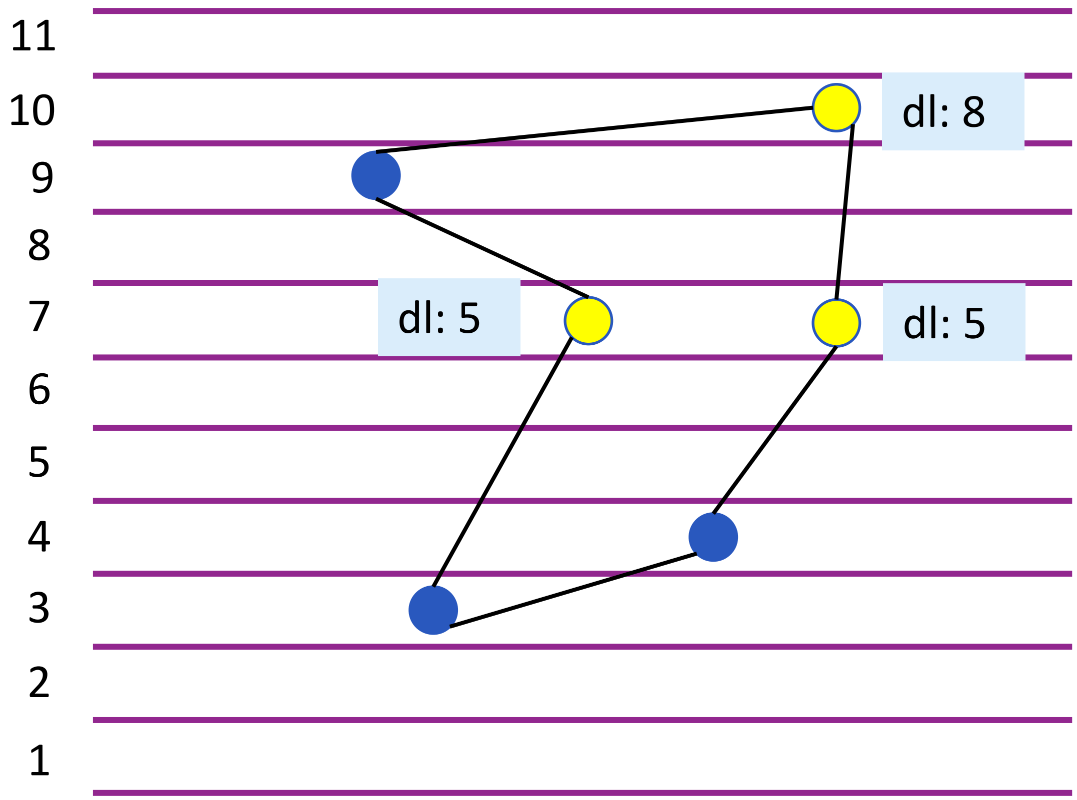


Only lower bound invariant, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure

Deletions

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



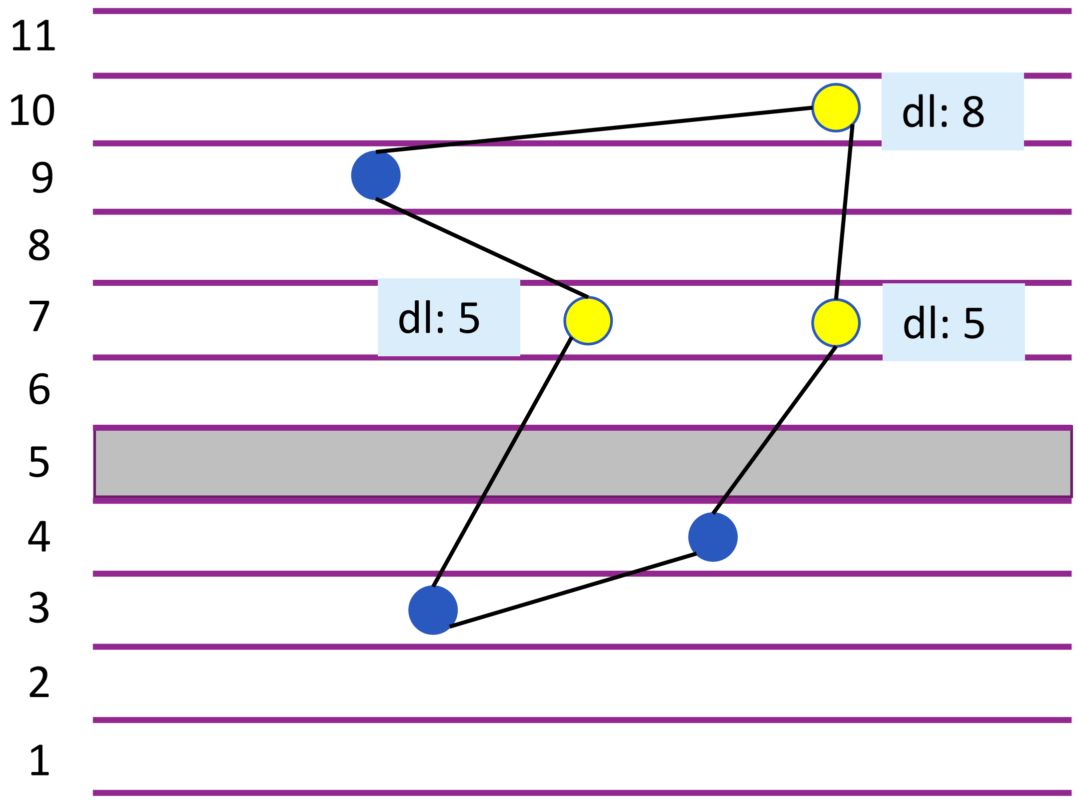
Iterate from **bottommost level to top level** and move vertices to desire-level

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

Our Parallel Batch-Dynamic Level Data Structure

Deletions

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



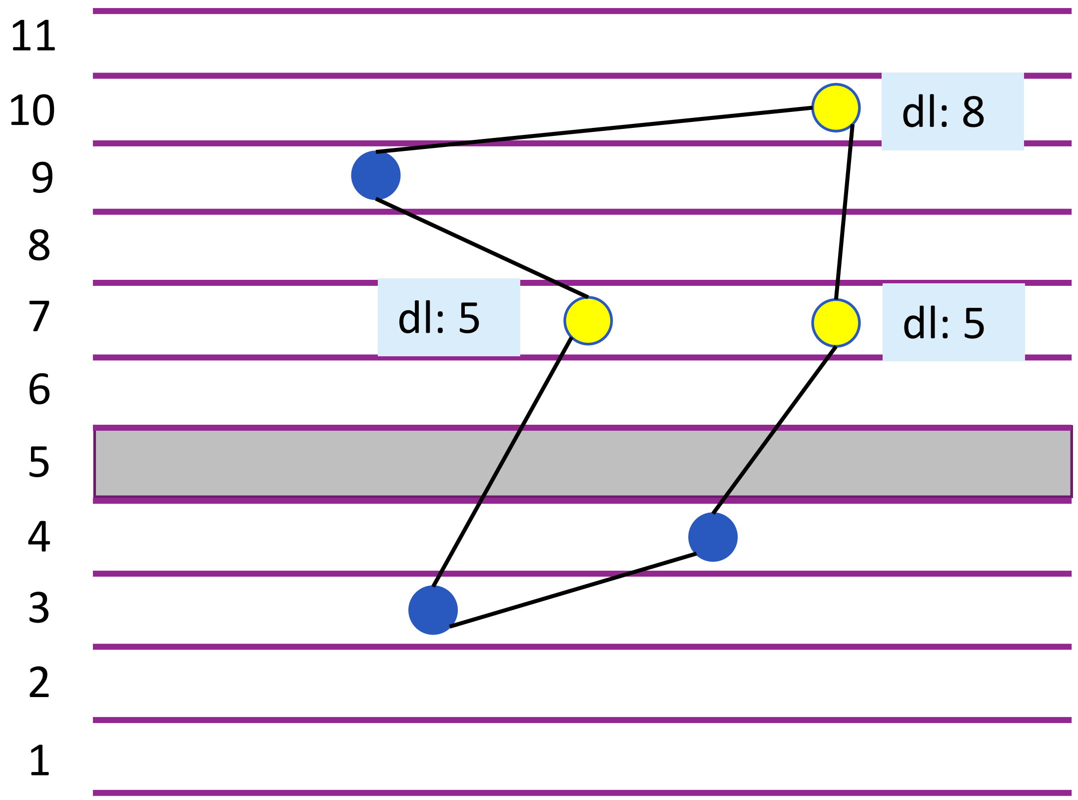
Iterate from **bottommost level to top level** and move vertices to desire-level

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**



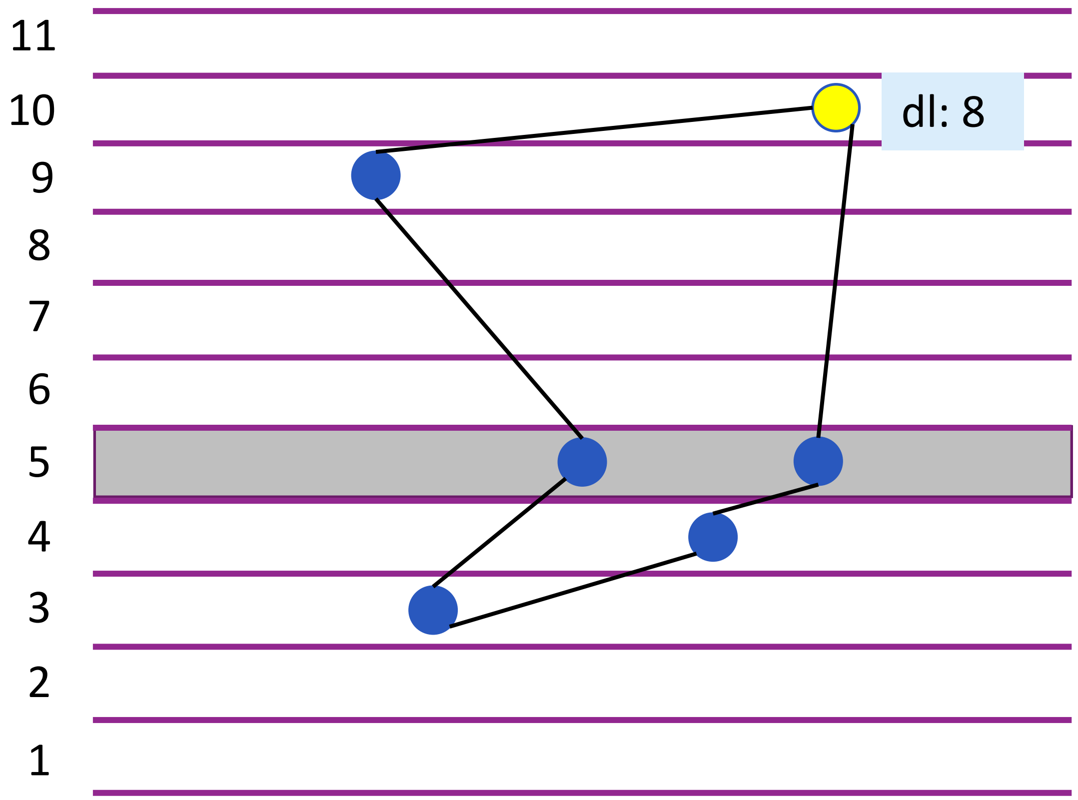
Iterate from **bottommost level to top level** and move vertices to desire-level

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**

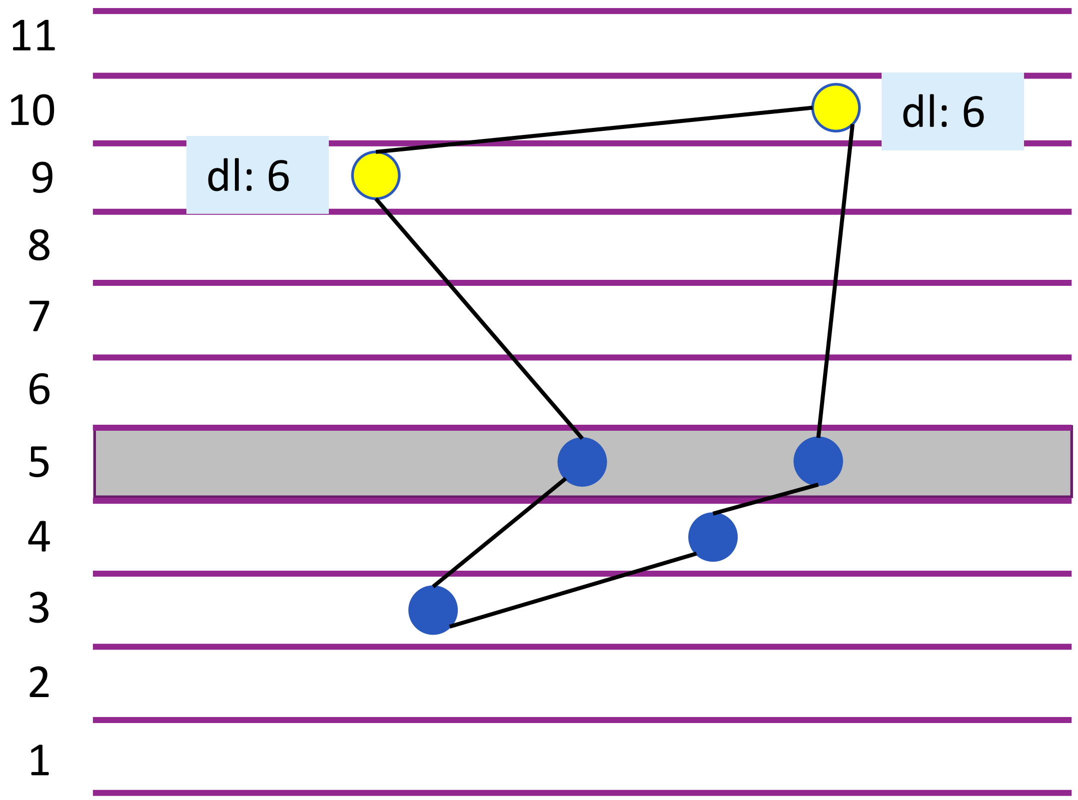


Update desire-levels of adjacent vertices at higher levels

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**

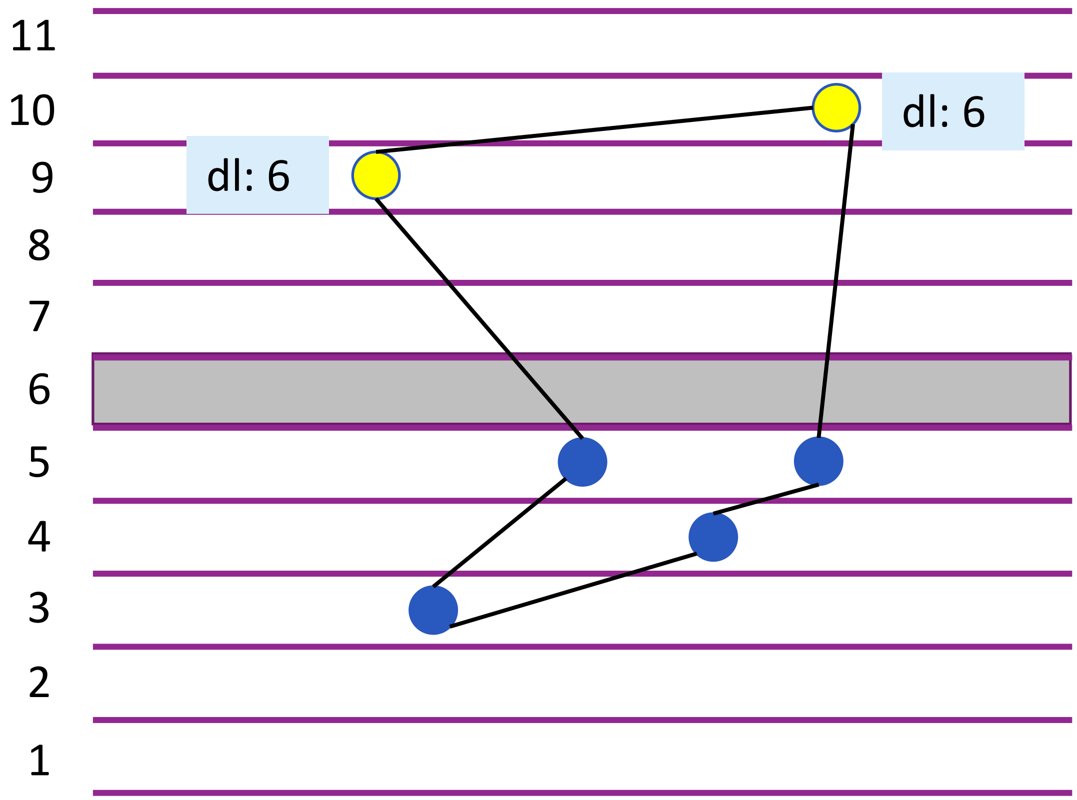
Update desire-levels of adjacent vertices at higher levels

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**



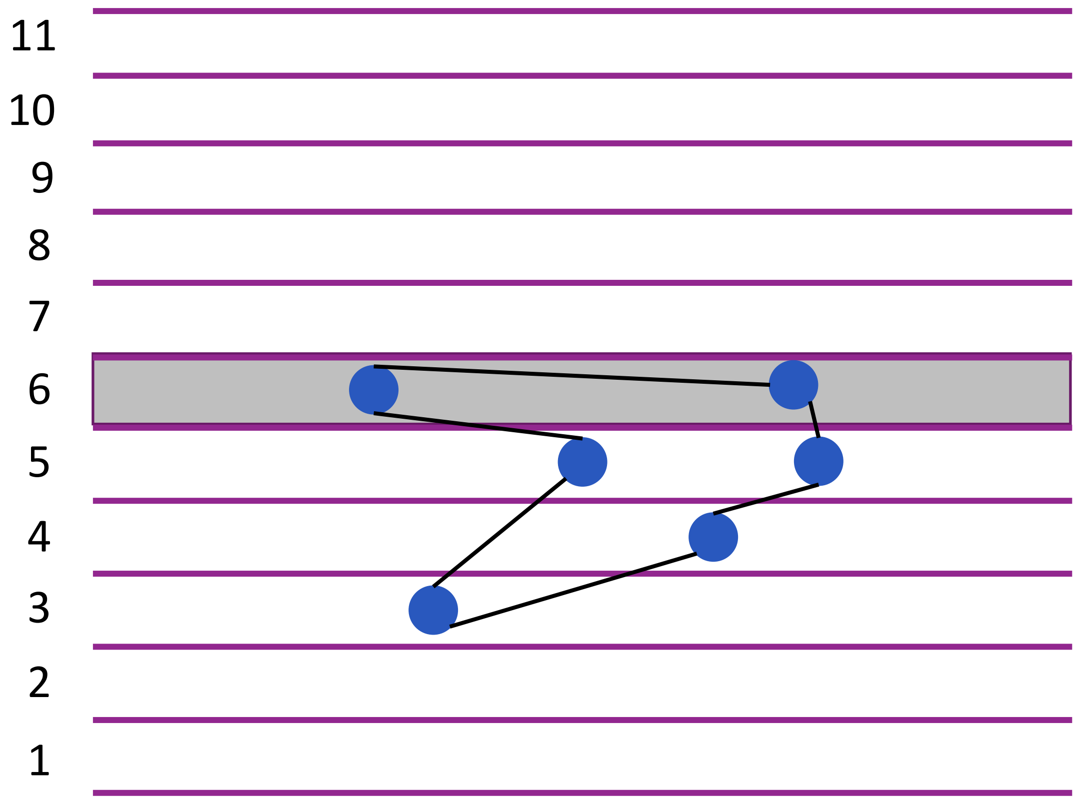
Update desire-levels of adjacent vertices at higher levels

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

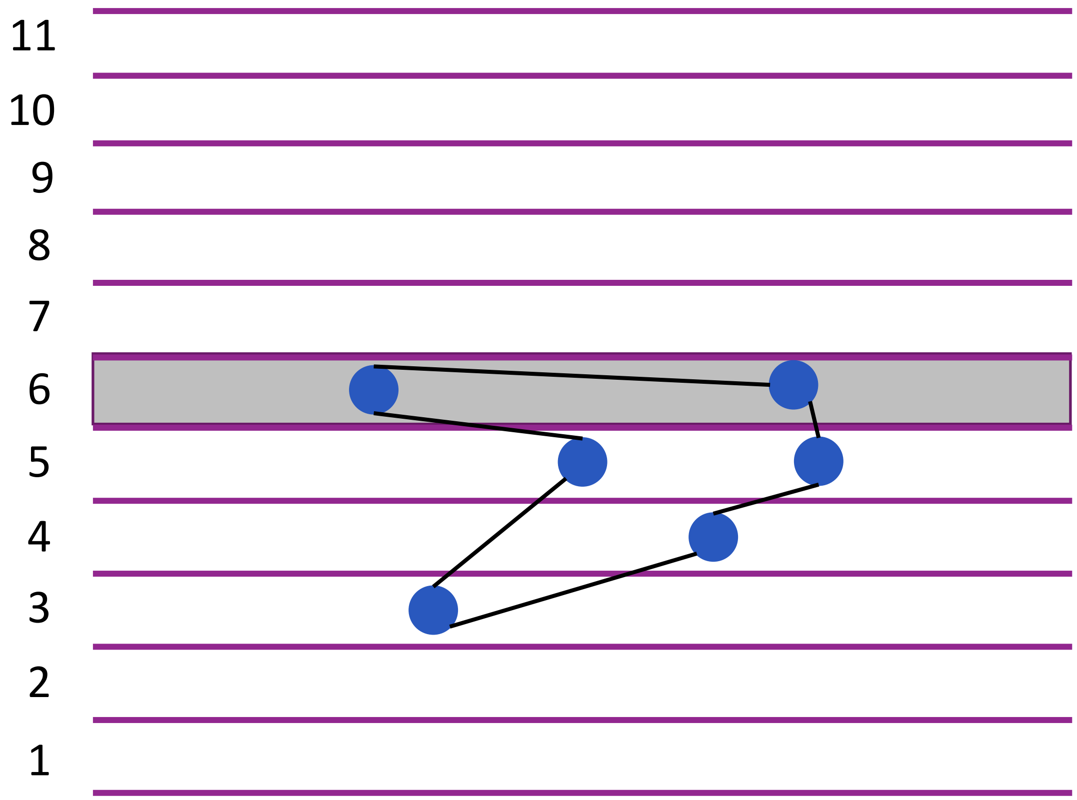
Only lower bound invariant, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure

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

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

Deletions



Update desire-levels of adjacent vertices at higher levels

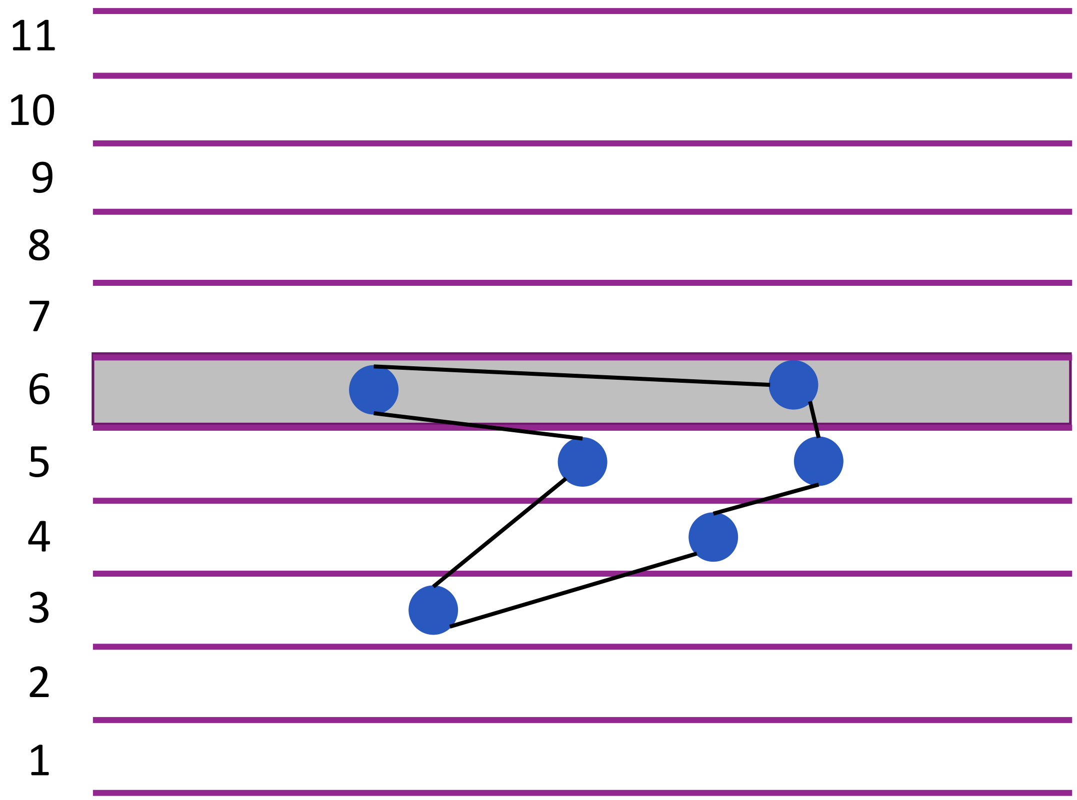
Only lower bound invariant, $(1+\epsilon)^i$, ever violated.

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

Update desire-levels of adjacent vertices at higher levels

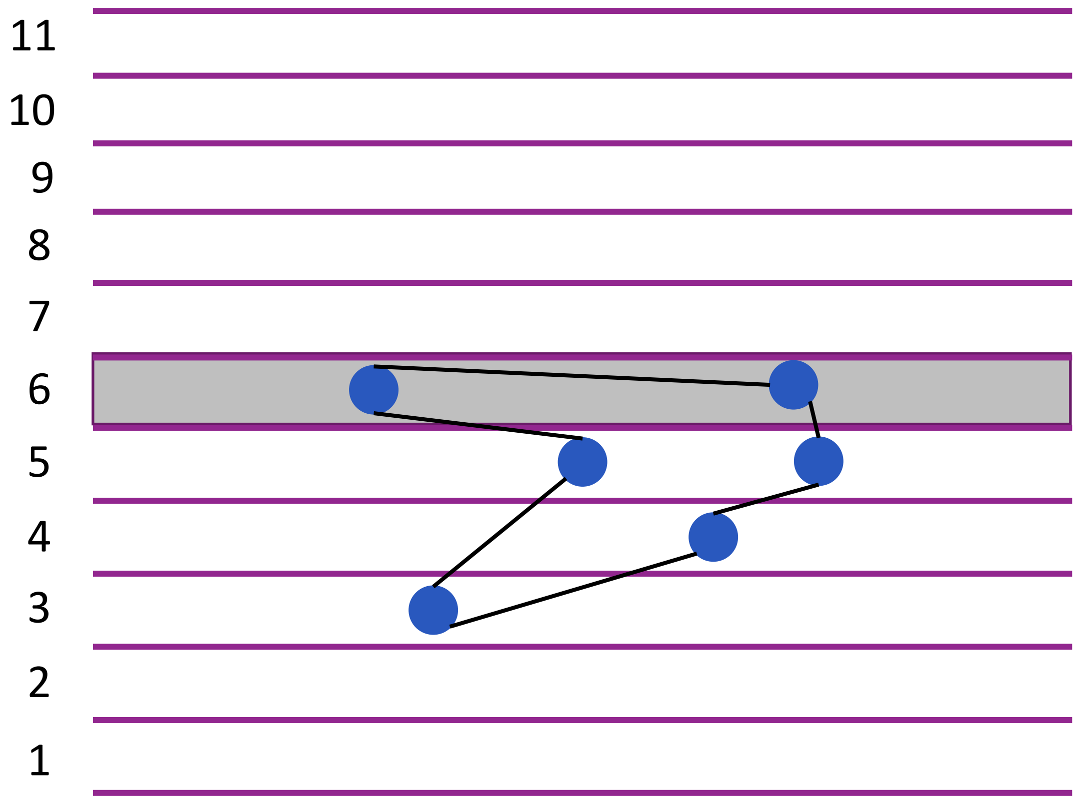
Only lower bound invariant, $(1+\epsilon)^i$, ever violated.

Our Parallel Batch-Dynamic Level Data Structure

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

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

Deletions



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

Update desire-levels of adjacent vertices at higher levels

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

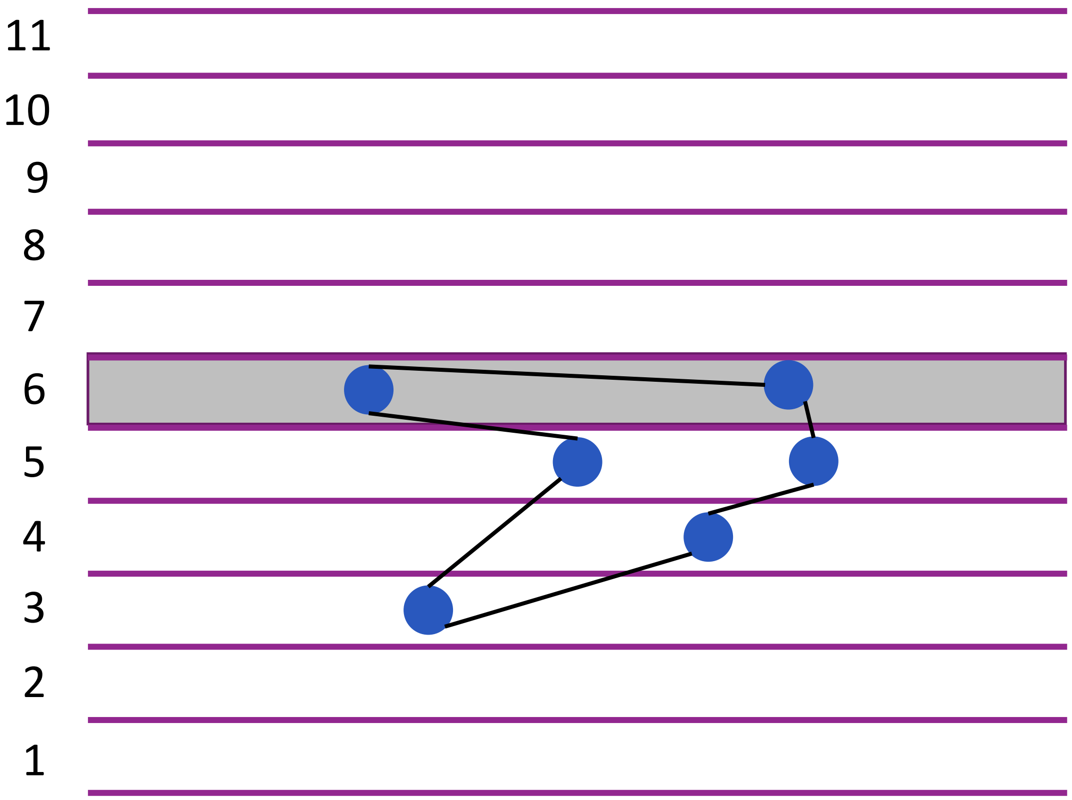
Our Parallel Batch-Dynamic Level Data Structure

Deletions

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

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

Work-efficient

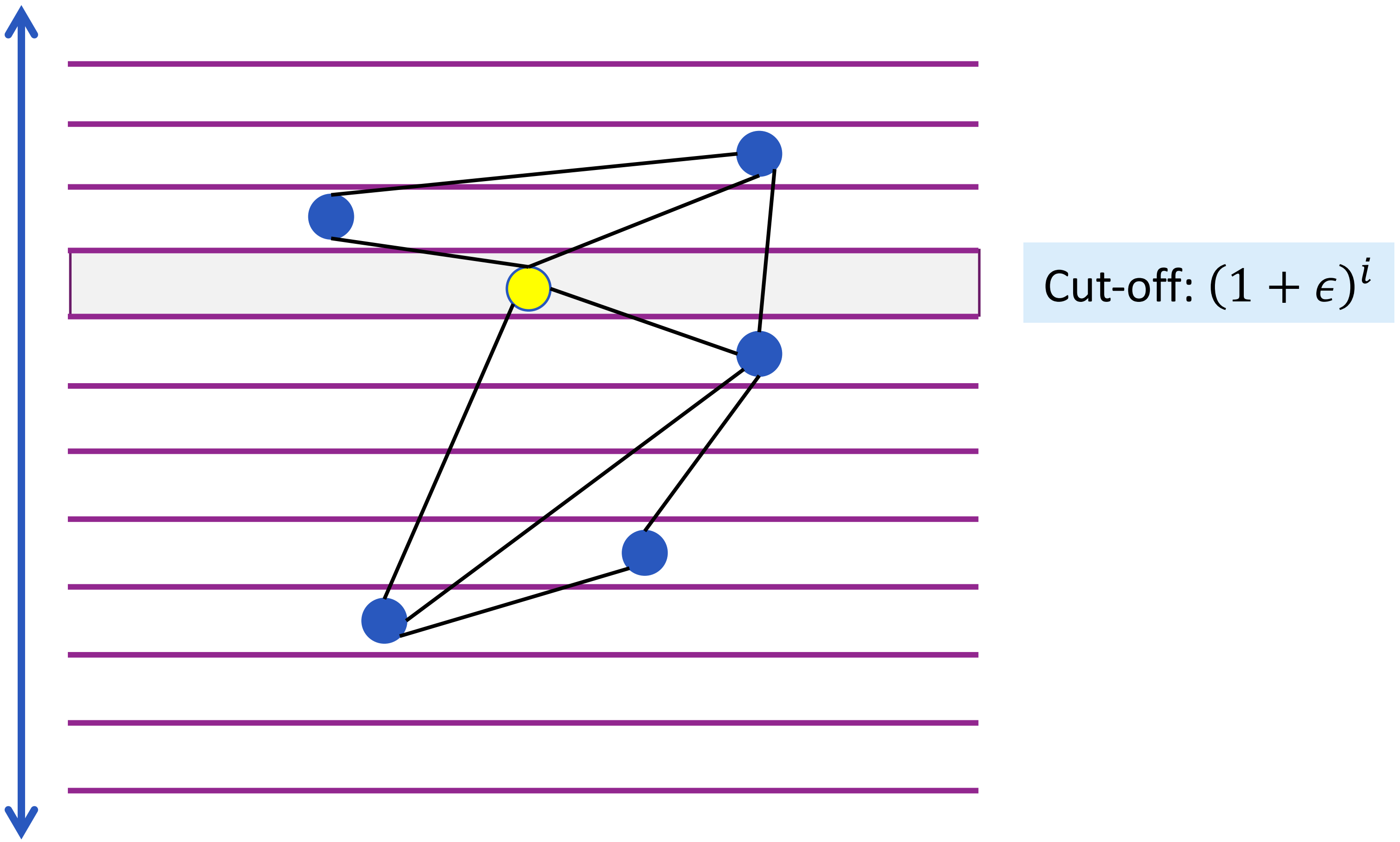


$O(\log^2 n \log\log n)$ depth w.h.p

Update desire-levels of adjacent vertices at higher levels

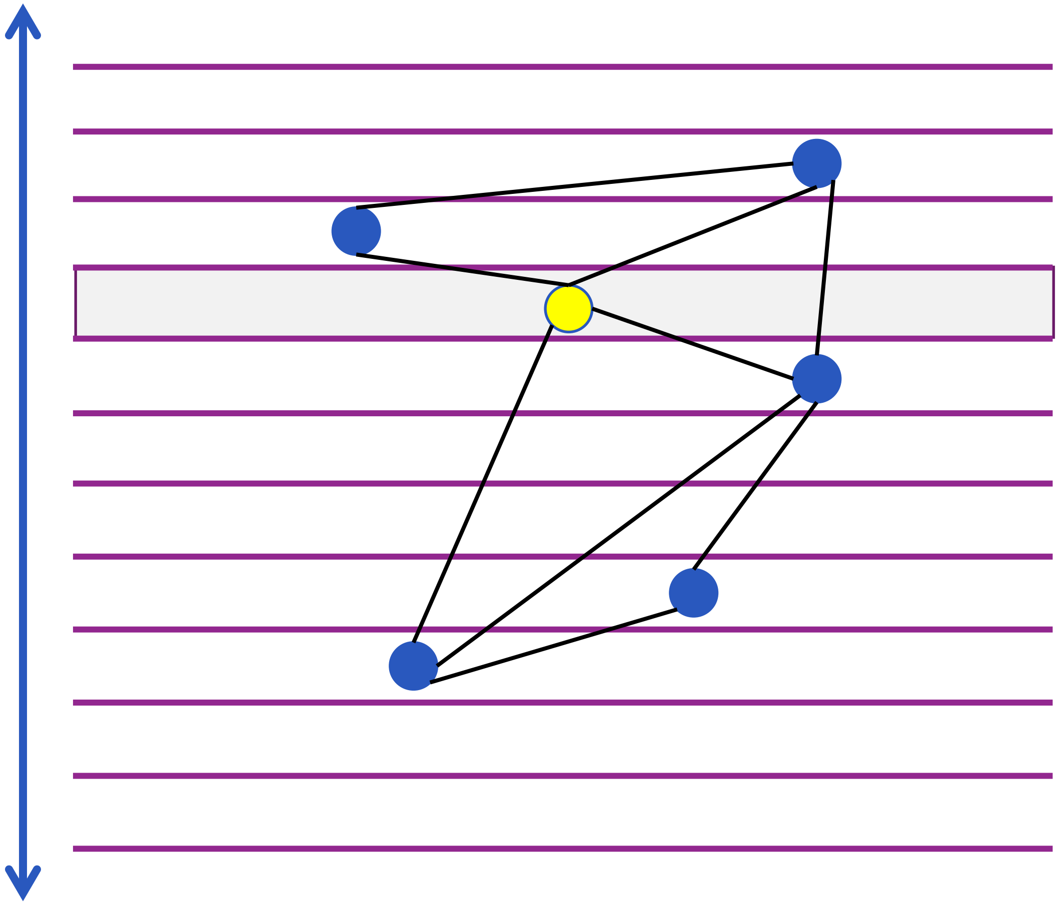
Only lower bound invariant, $(1+\epsilon)^i$, ever violated.

Proof of Our Approximation Factor



Proof of Our Approximation Factor

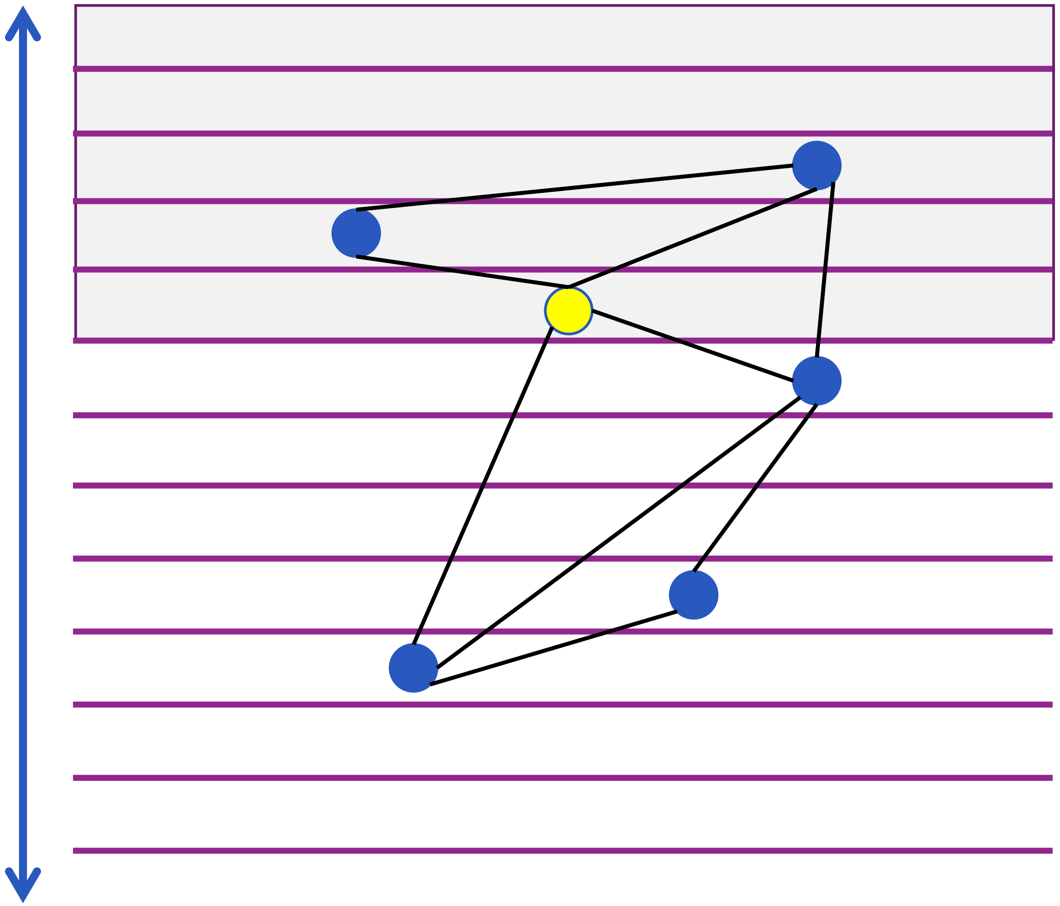
Coreness Estimate:
$(1+\epsilon)^i$



Cut-off: $(1+\epsilon)^i$

Proof of Our Approximation Factor: Upper Bound

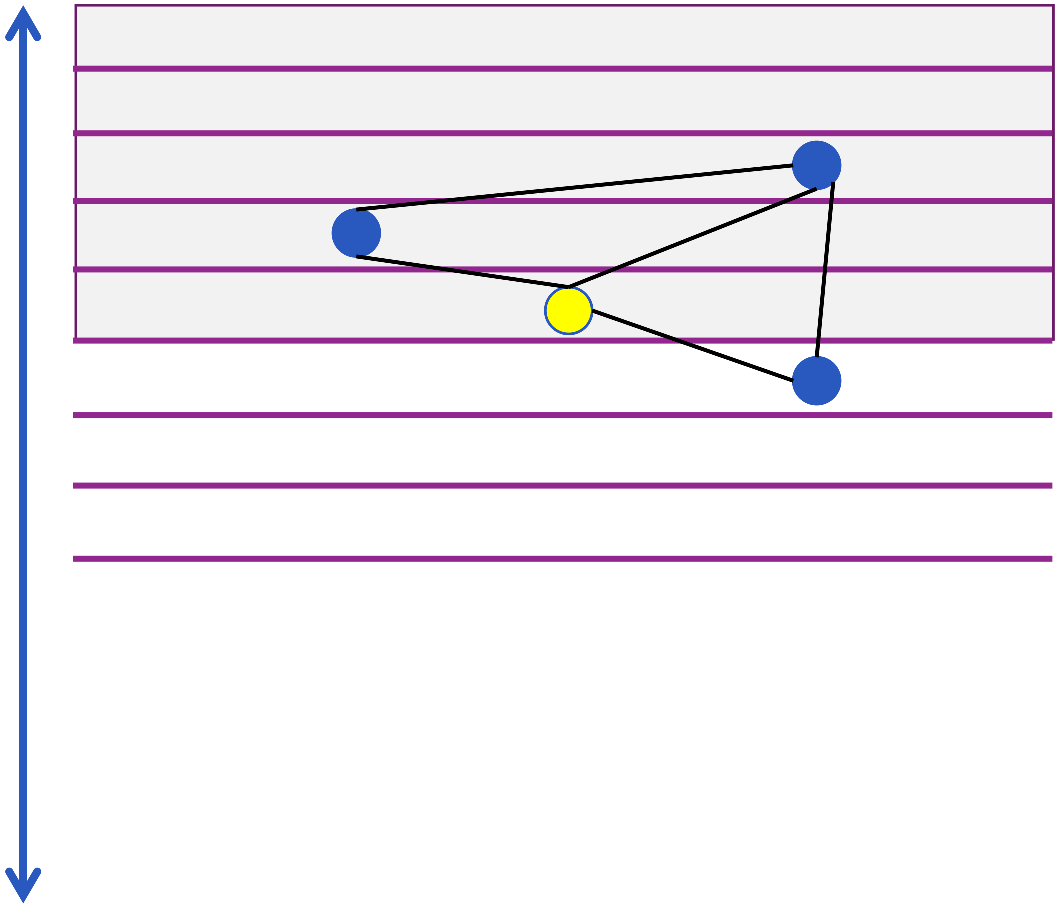
Coreness Estimate:
$(1+\epsilon)^i$



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

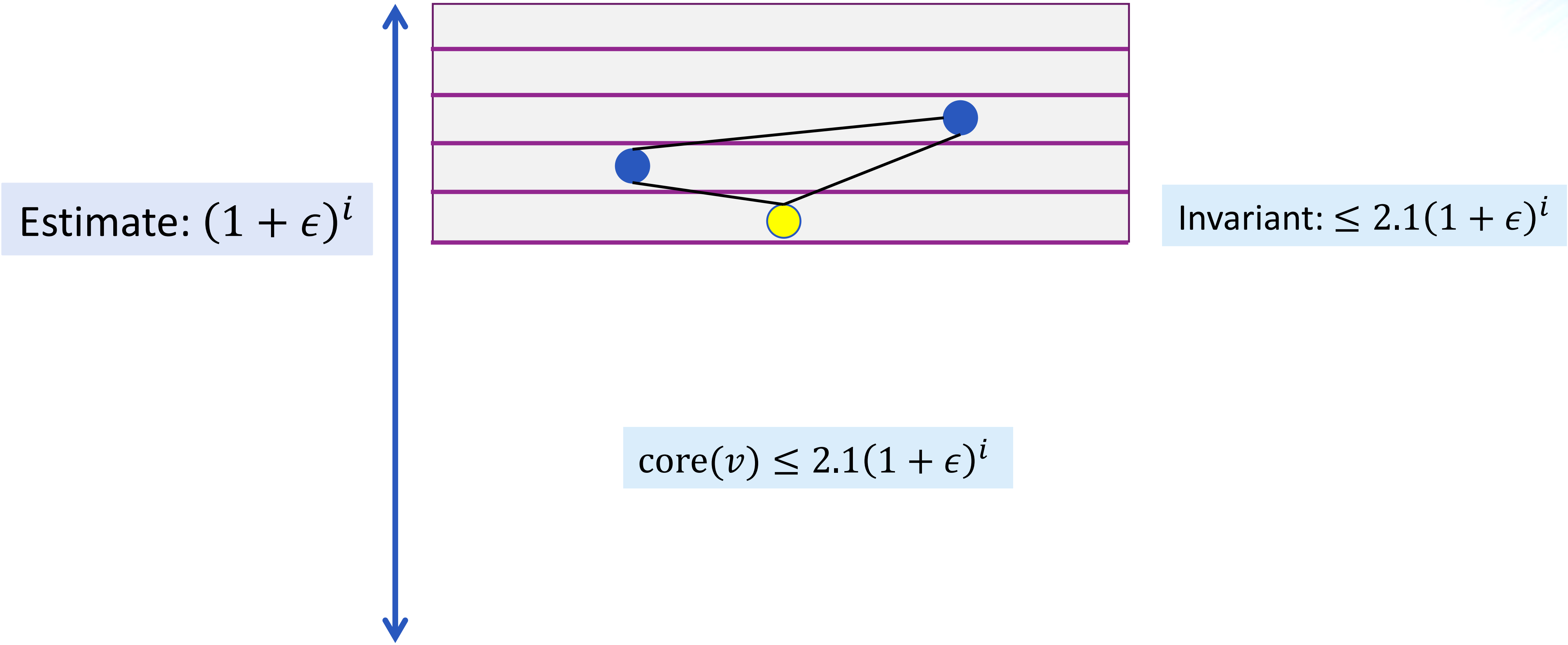
Proof of Our Approximation Factor: Upper Bound

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

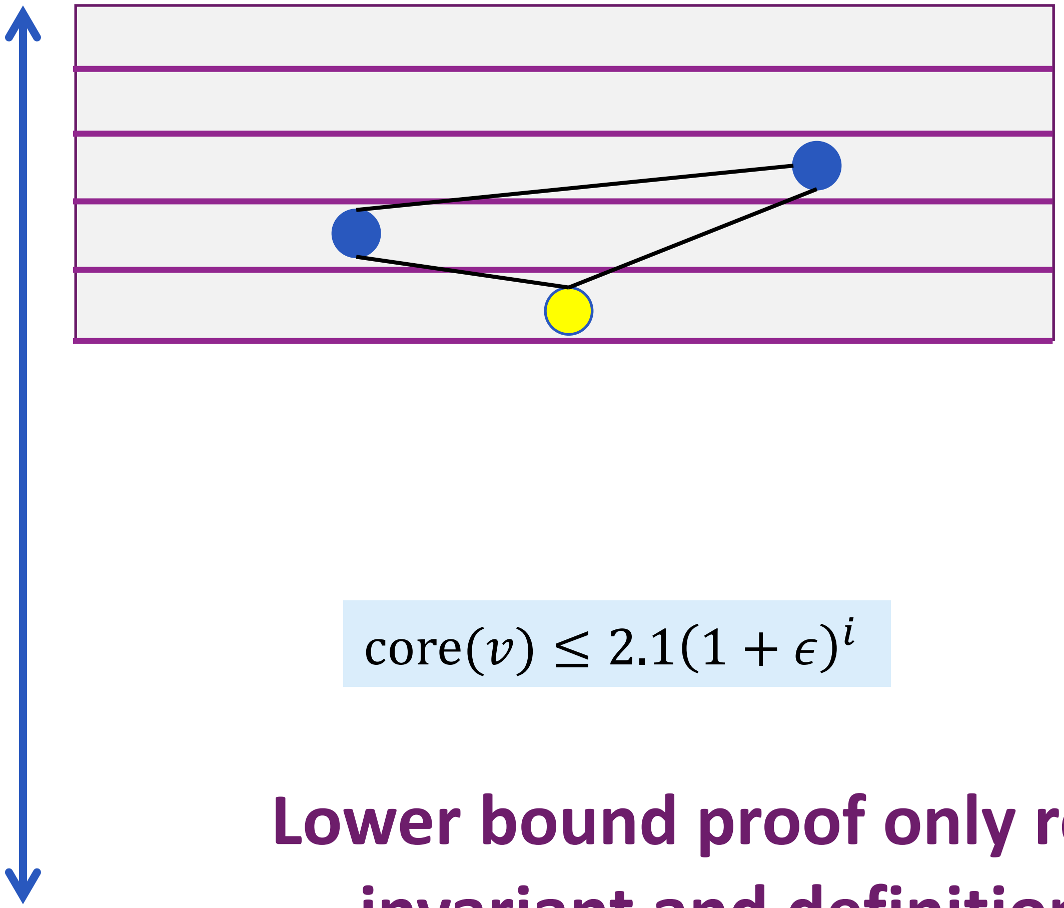


Invariant: $\leq 2.1(1+\epsilon)^i$

Proof of Our Approximation Factor: Upper Bound



Proof of Our Approximation Factor: Upper Bound



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

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

$\text{core}(v) \leq 2.1(1+\epsilon)^i$

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

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

Tested Graphs

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

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

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

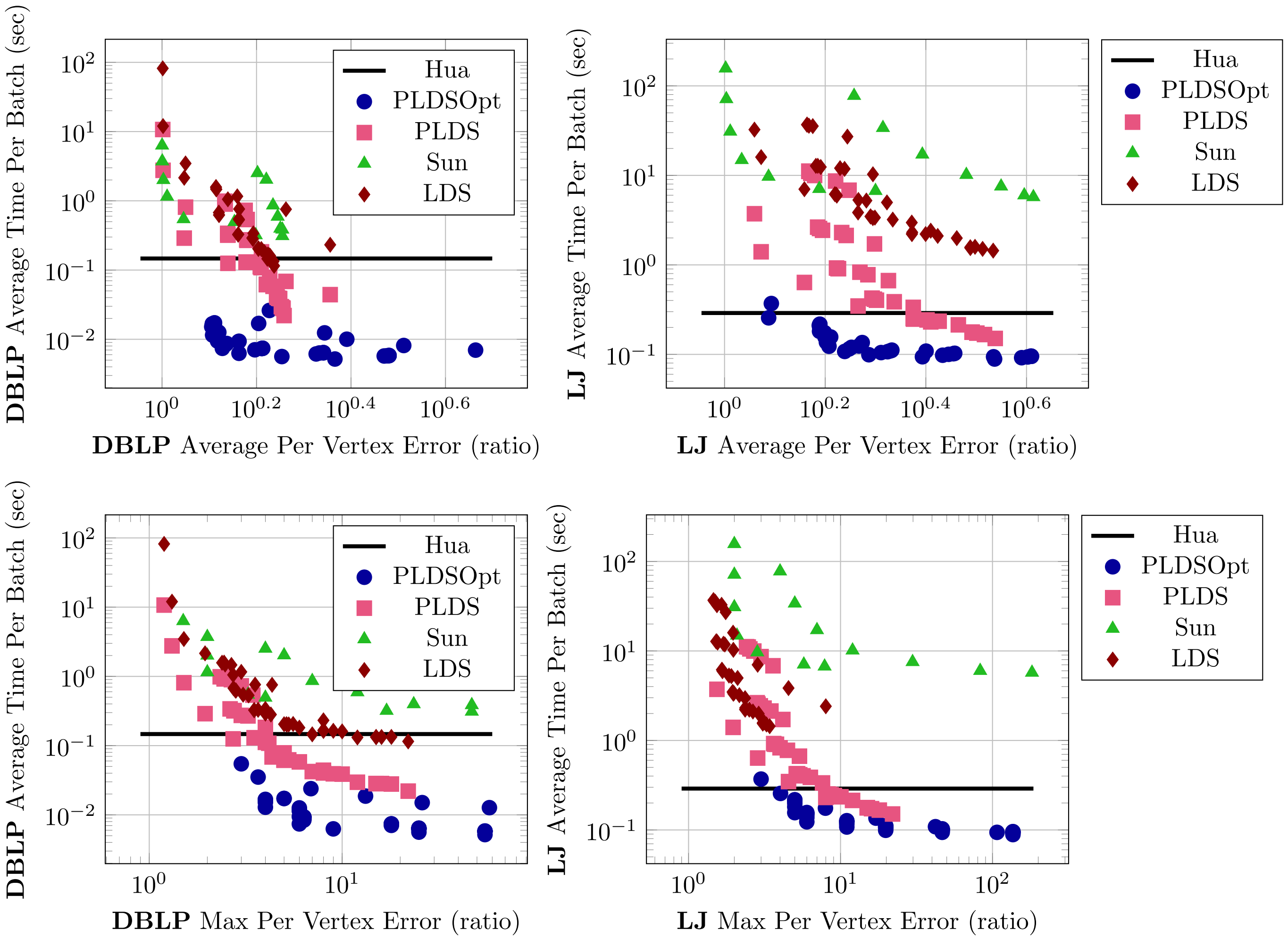
https://github.com/qqliu/batch-dynamic-kcore-decomposition

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)



Runtimes/Accuracy Against State-of-the-Art Algorithms

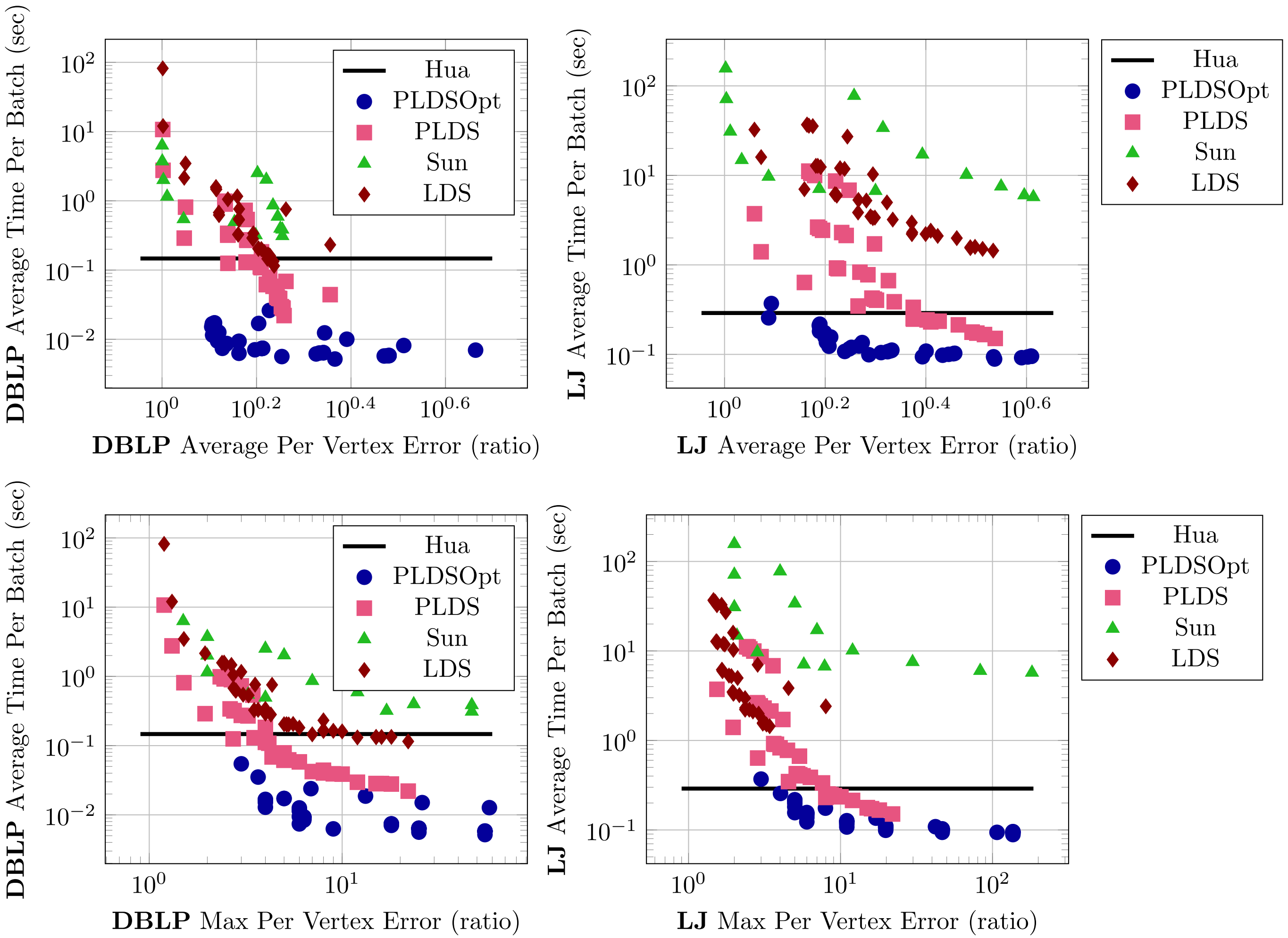
https://github.com/qqliu/batch-dynamic-kcore-decomposition

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)



Runtimes/Accuracy Against State-of-the-Art Algorithms

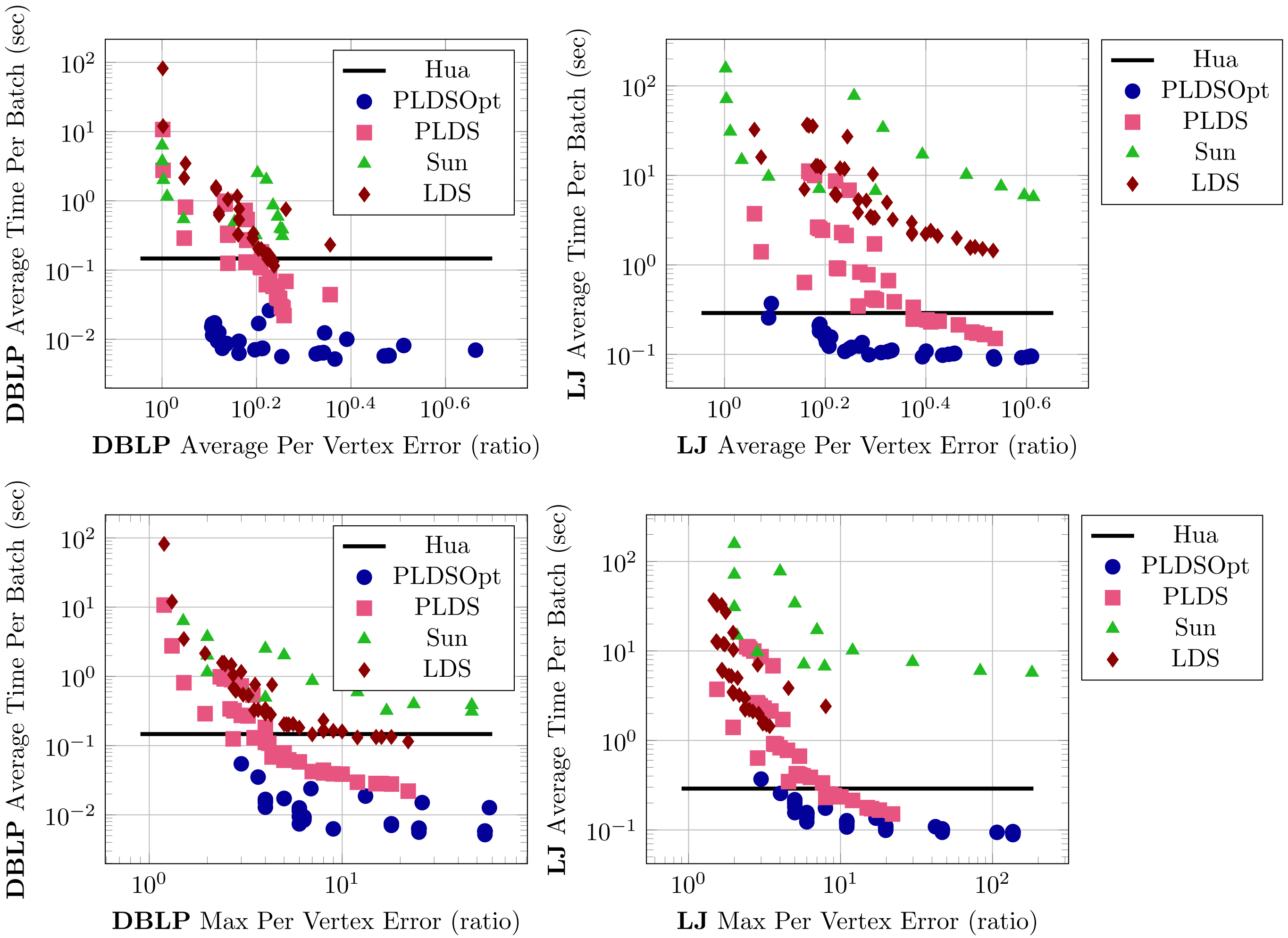
https://github.com/qqliu/batch-dynamic-kcore-decomposition

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)



Runtimes/Accuracy Against State-of-the-Art Algorithms

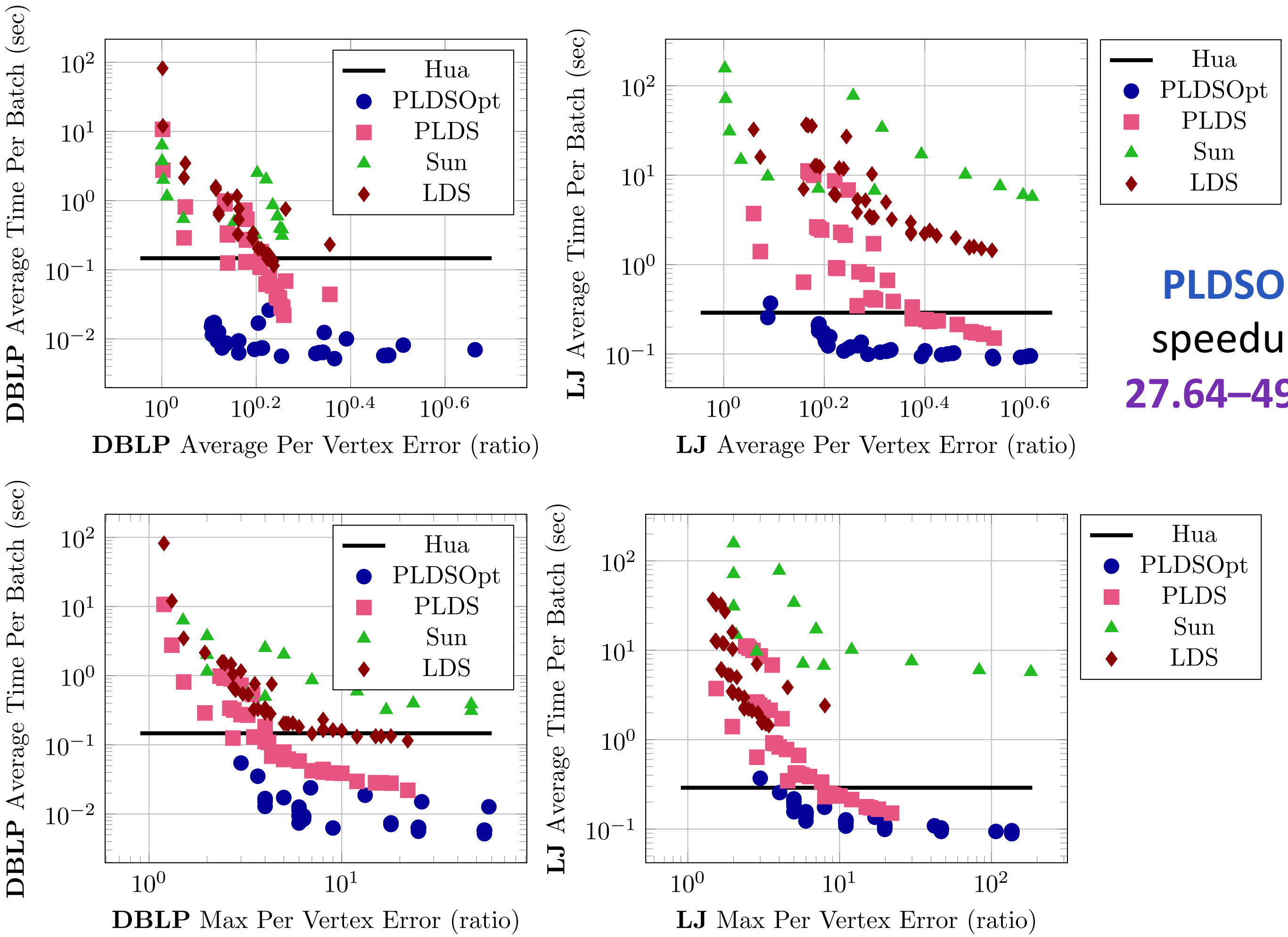
https://github.com/qqliu/batch-dynamic-kcore-decomposition

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

Runtimes/Accuracy Against State-of-the-Art Algorithms

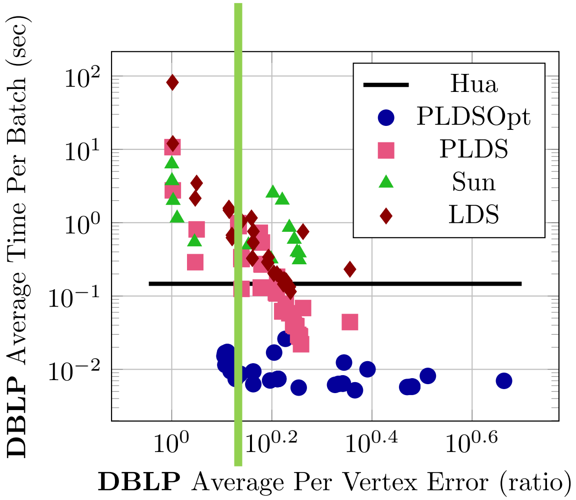
Sun et al. TKDD: current best **sequential, approx** dynamic algorithm

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

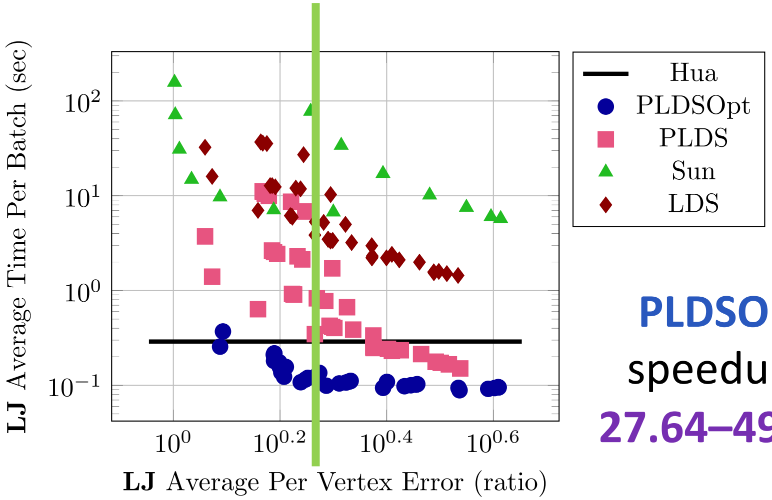
LDS: sequential LDS of Henzinger et al.

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

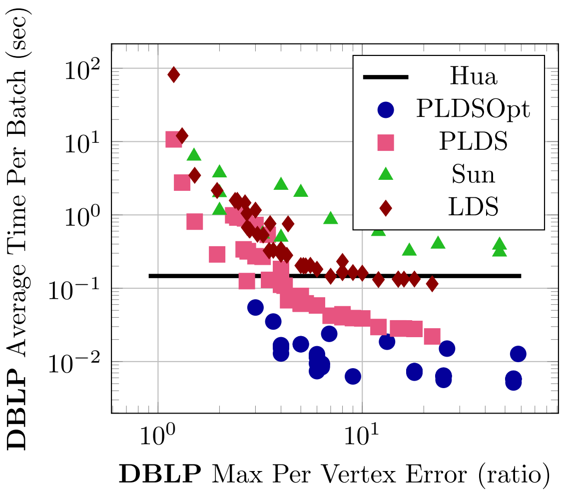
PLDSOpt: 24.5x over Hua

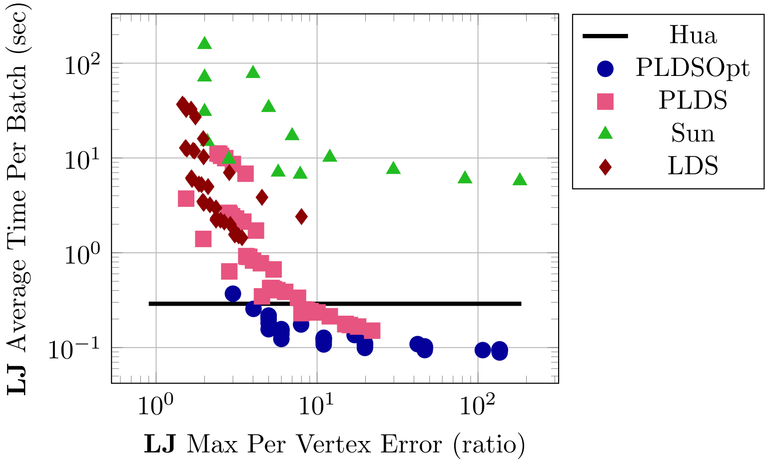


PLDSOpt: 2.68x over Hua



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



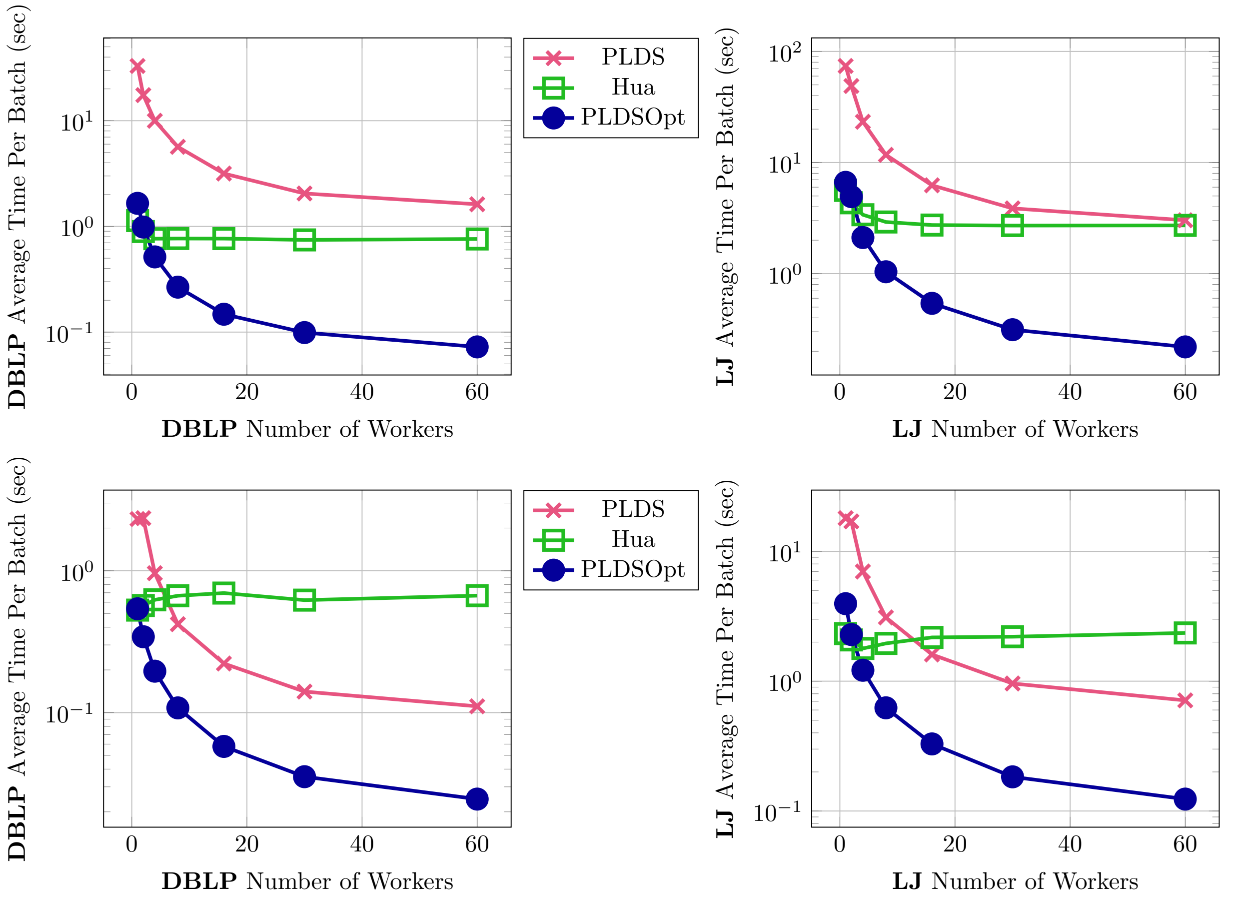


Number of Workers

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

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

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

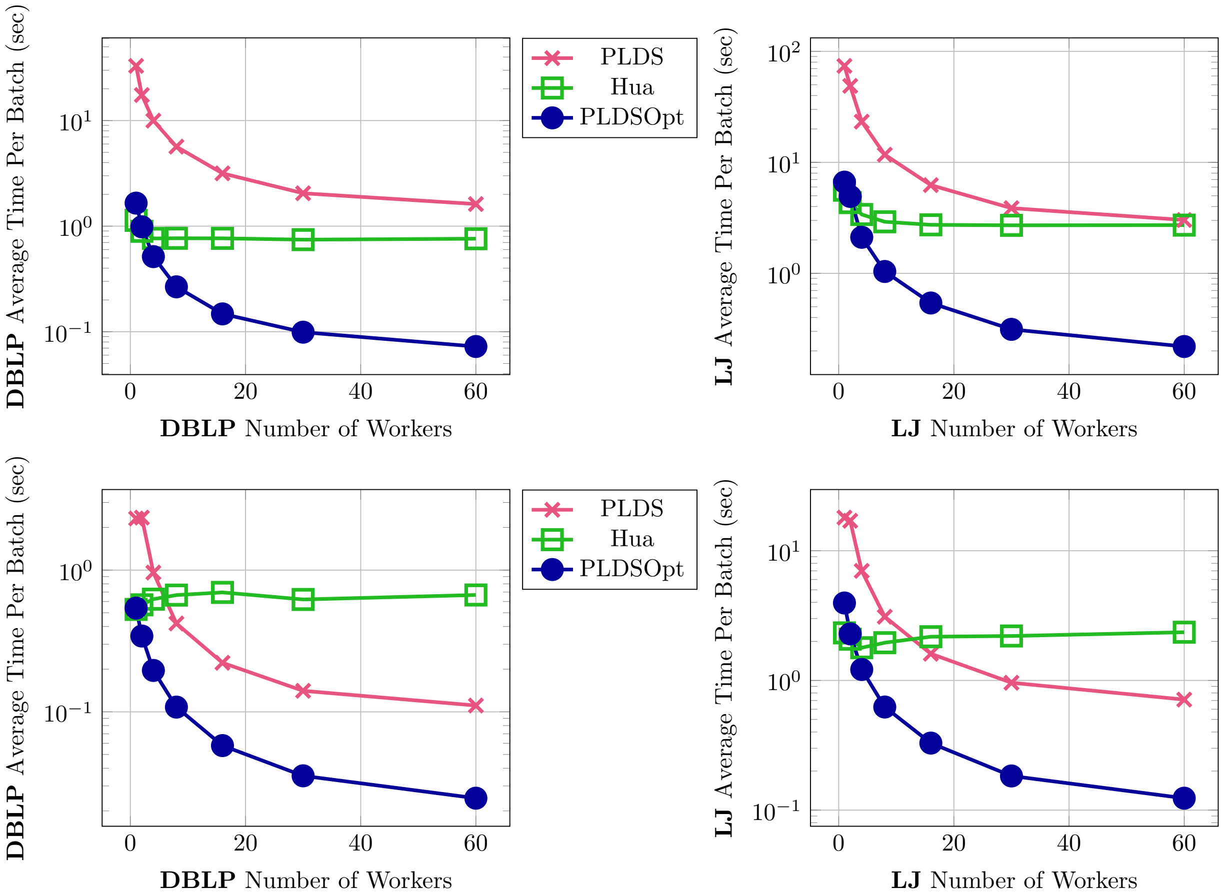


Number of Workers

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

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

On Variety of Graphs

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

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

On Variety of Graphs

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

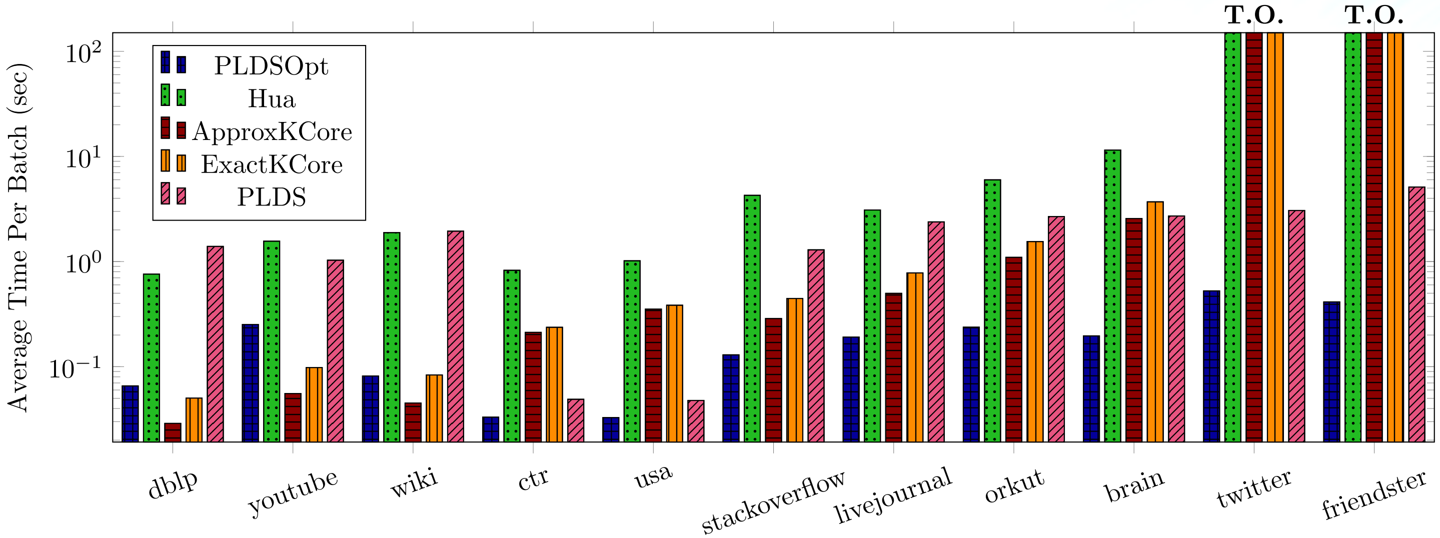
Sun et al. TKDD: current best **sequential, approx** dynamic algorithm

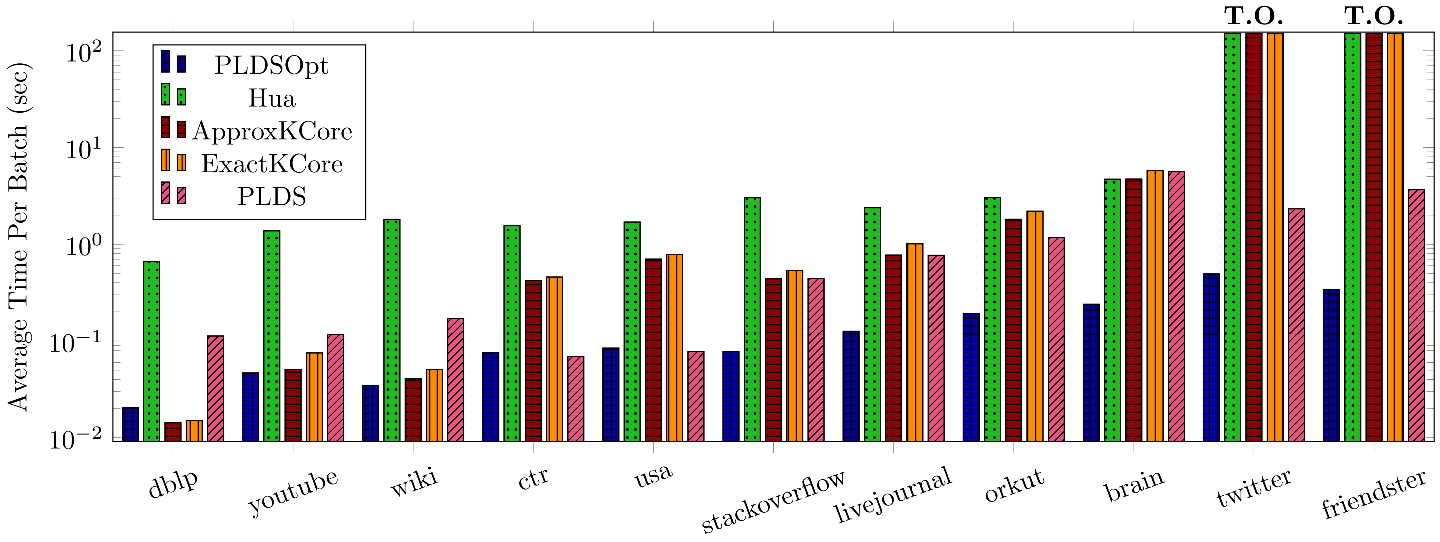
Hua et al. TPDS: current best **parallel, exact** dynamic algorithm

LDS: sequential LDS of Henzinger et al.

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

PLDSOpt: code-optimized PLDS





On Variety of Graphs

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

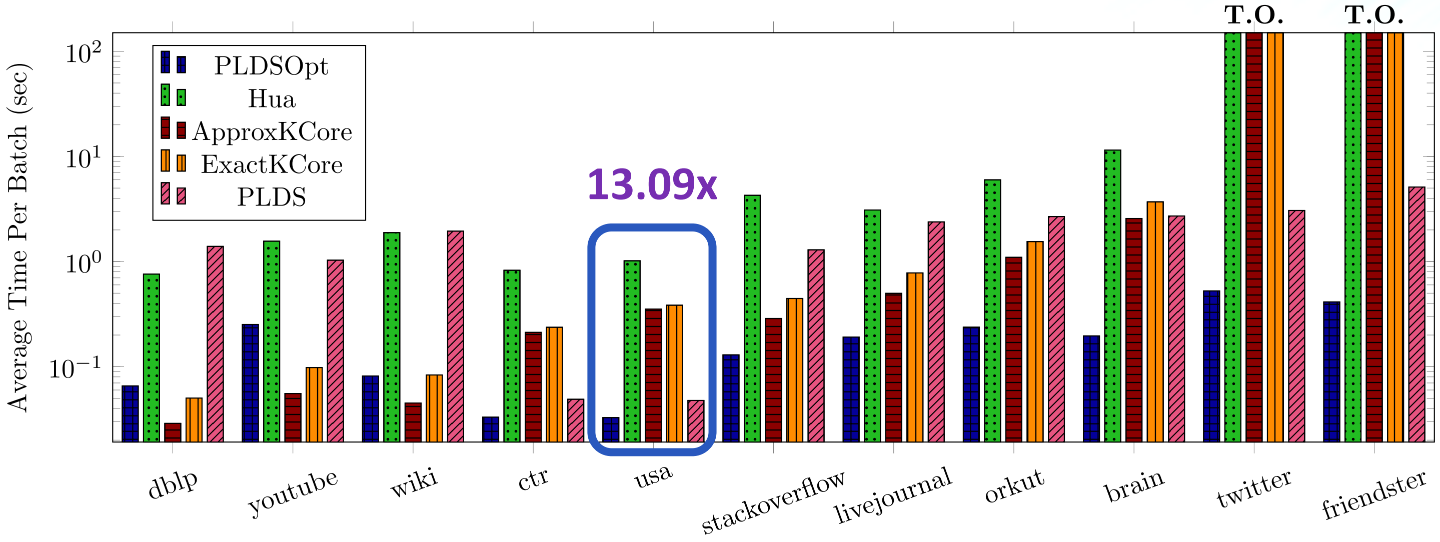
Sun et al. TKDD: current best **sequential, approx** dynamic algorithm

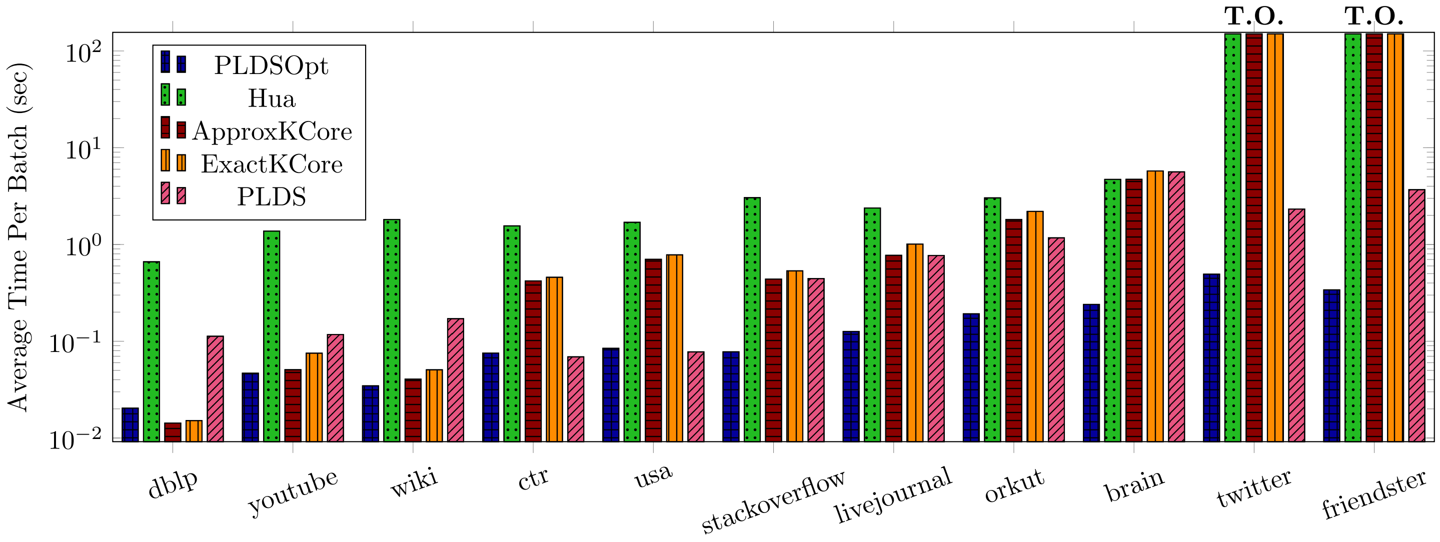
Hua et al. TPDS: current best **parallel, exact** dynamic algorithm

LDS: sequential LDS of Henzinger et al.

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

PLDSOpt: code-optimized PLDS





On Variety of Graphs

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

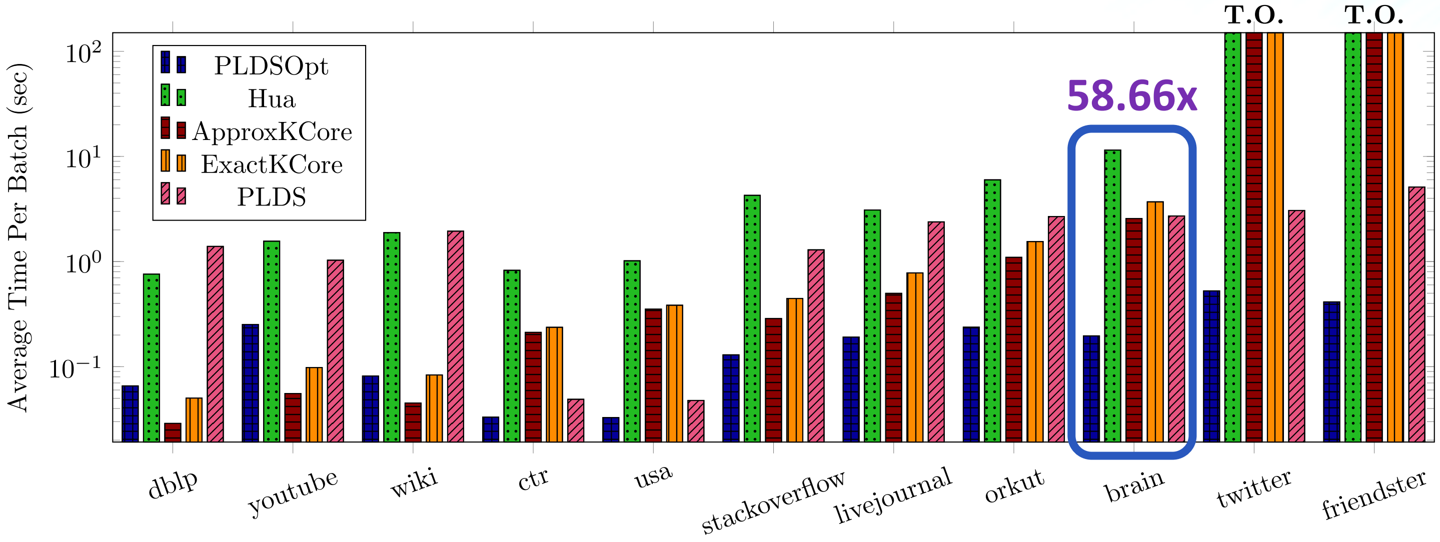
Sun et al. TKDD: current best **sequential, approx** dynamic algorithm

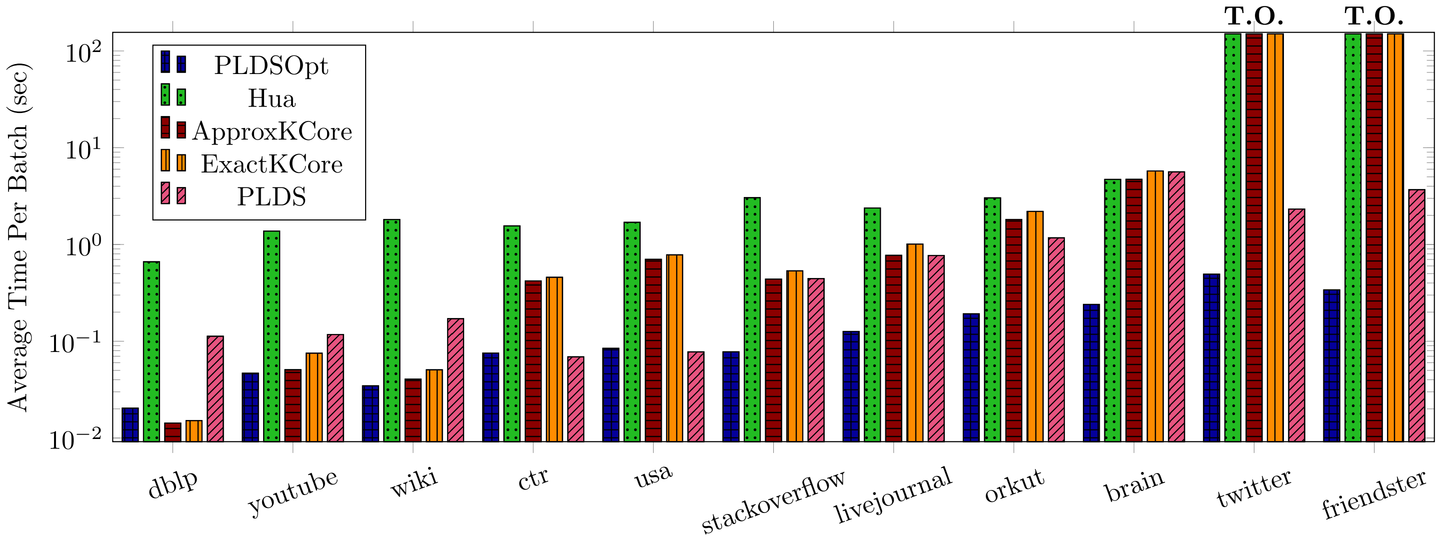
Hua et al. TPDS: current best **parallel, exact** dynamic algorithm

LDS: sequential LDS of Henzinger et al.

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

PLDSOpt: code-optimized PLDS





On Variety of Graphs

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

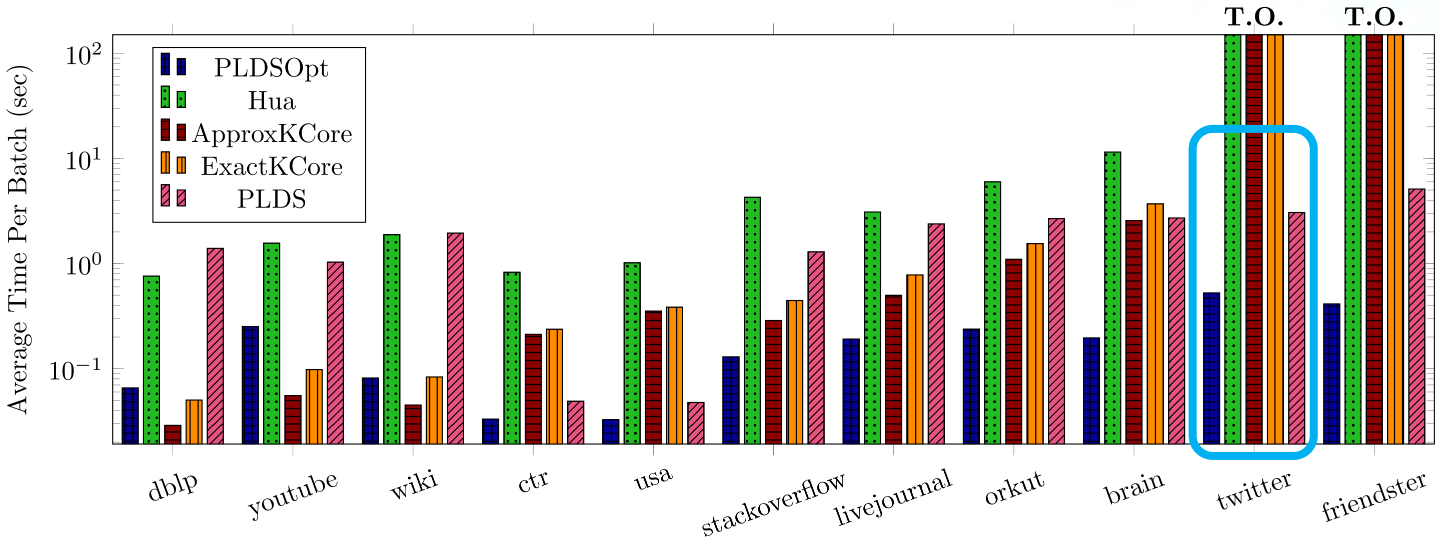
Sun et al. TKDD: current best **sequential, approx** dynamic algorithm

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

LDS: sequential LDS of Henzinger et al.

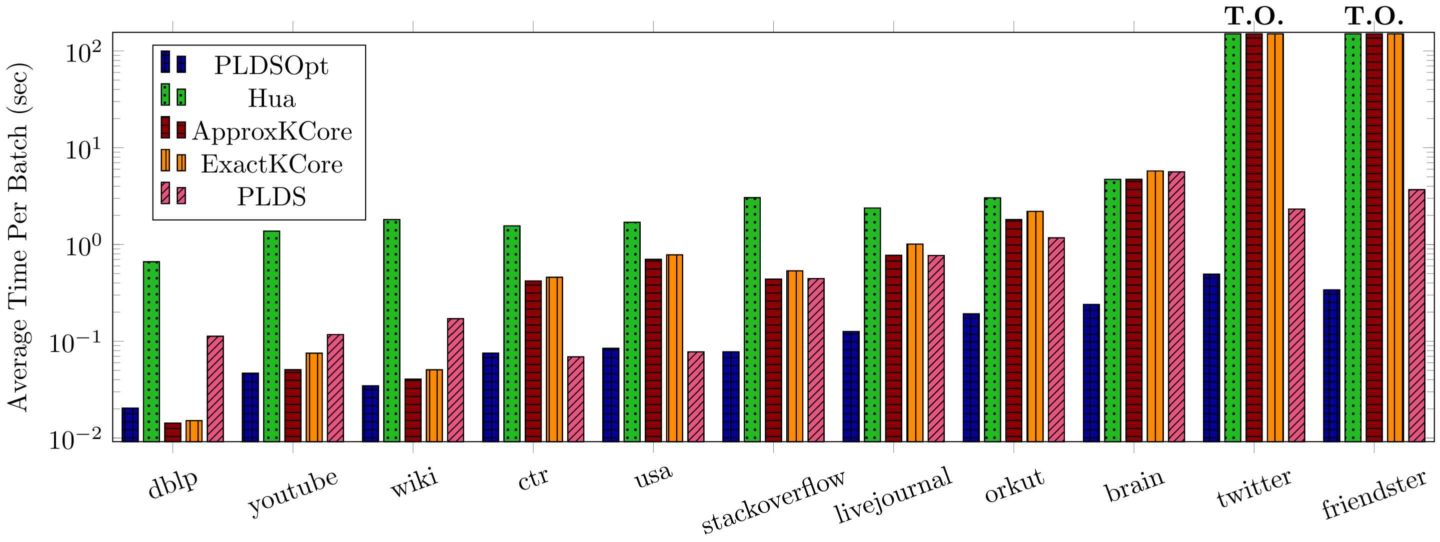
ExactKCore Dhulipala et al. SPAA: parallel, static exact algorithm

PLDSOpt: code-optimized PLDS



PLDSOpt: 0.53 sec per batch

PLDS: 3.06 sec per batch



Summary of Results

Static Algorithms

Structural Rounding

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§], Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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

Fully Dynamic $(\Delta + 1)$-Coloring in Constant Update Time

Fully Dynamic $(\Delta + 1)$-Coloring in Constant Update Time

Sayan Bhattacharya[*] Fabrizio Grandoni[†] Janardhan Kulkarni[‡] Quanquan C. Liu[§]
Shay Solomon[¶]

Sayan Bhattacharya

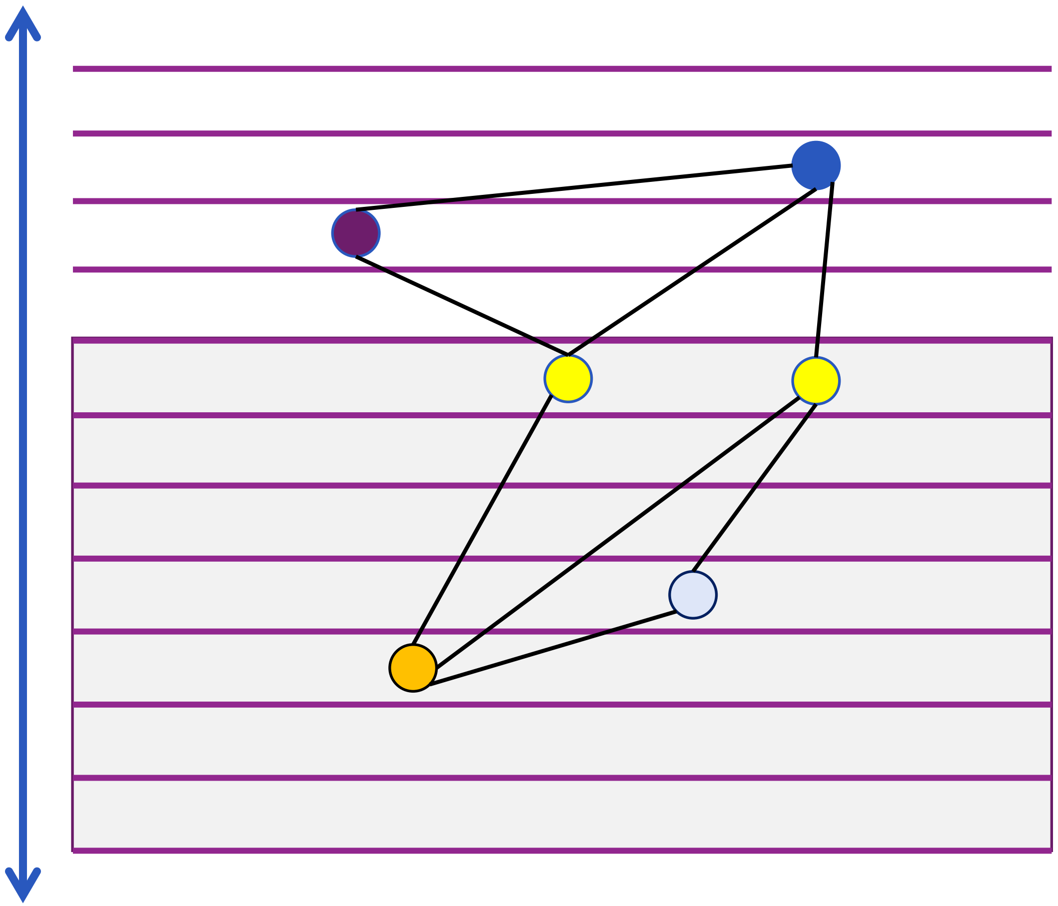
Fabrizio Grandoni

Janardhan Kulkarni

Shay Solomon

Level Data Structure

Δ := Max Degree



$O(\log \Delta)$

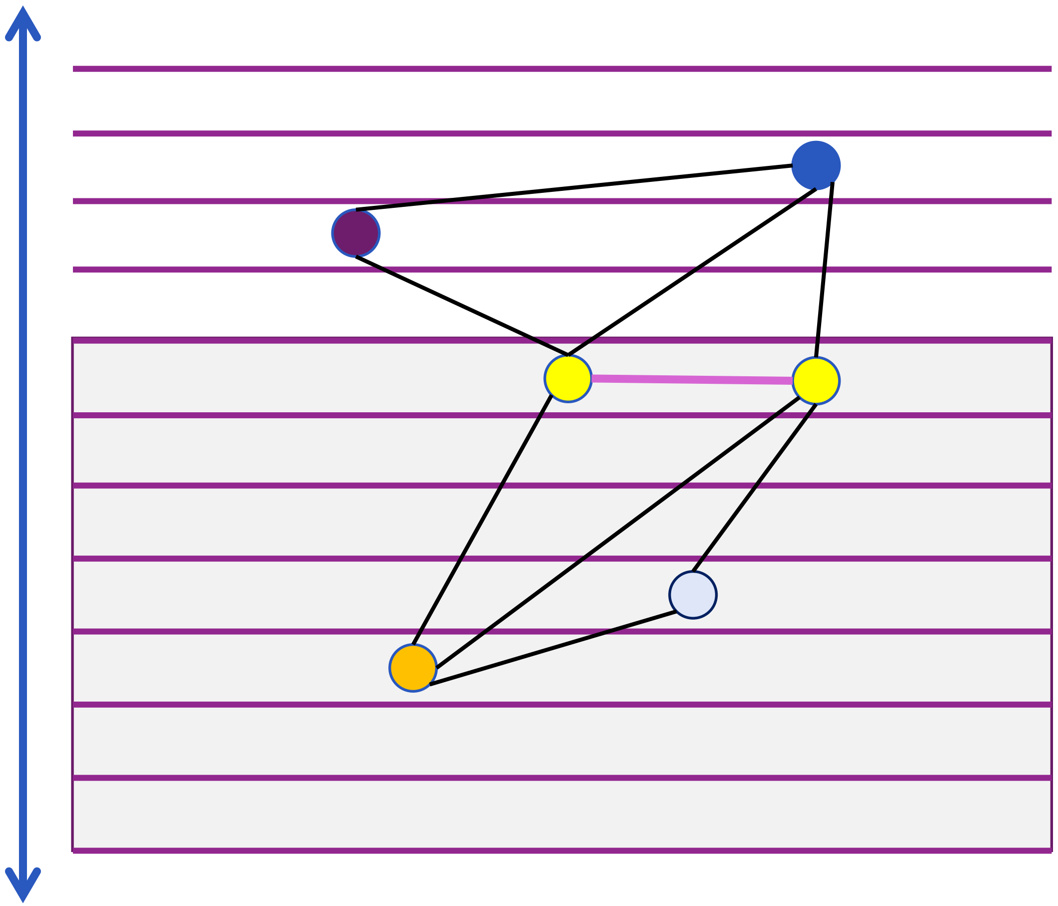
Vertices partitioned into levels

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

Level Data Structure

Δ := Max Degree

$O(\log \Delta)$



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

Vertices partitioned into levels

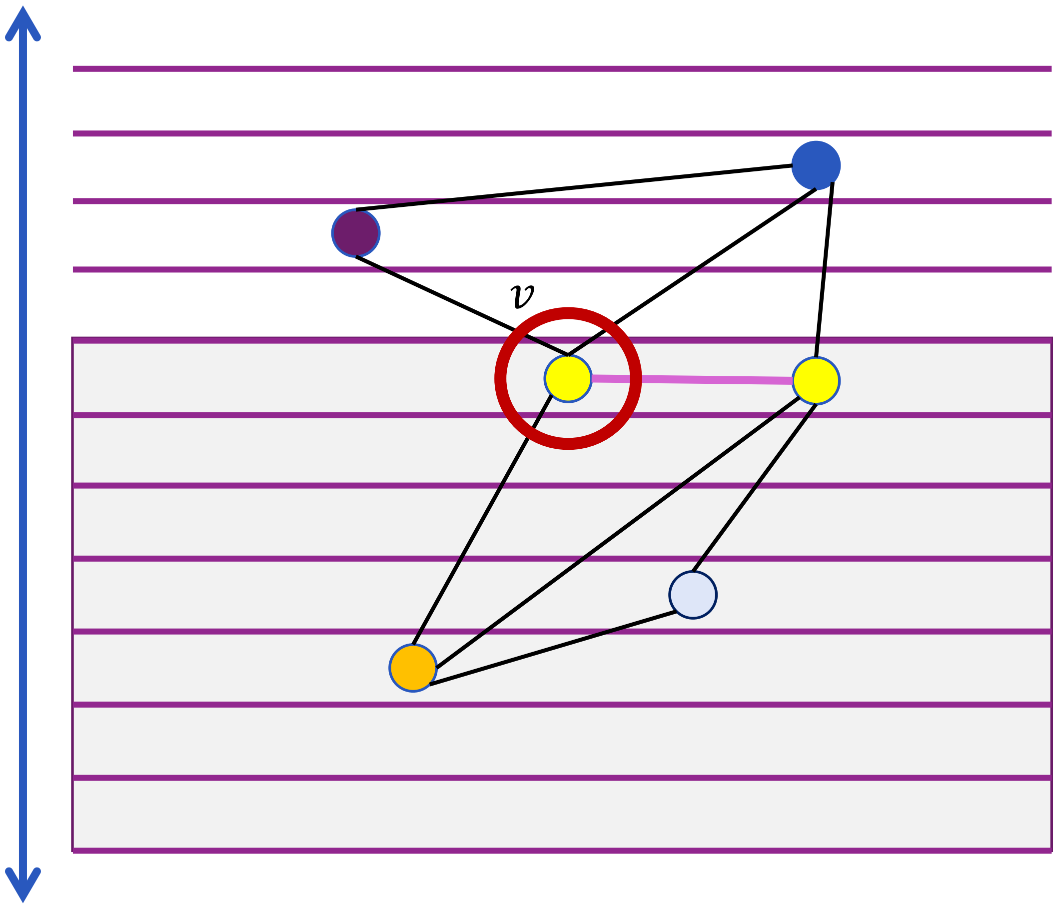
= Edge Insertion

Level Data Structure

Δ := Max Degree

$O(\log \Delta)$

Vertices partitioned into levels

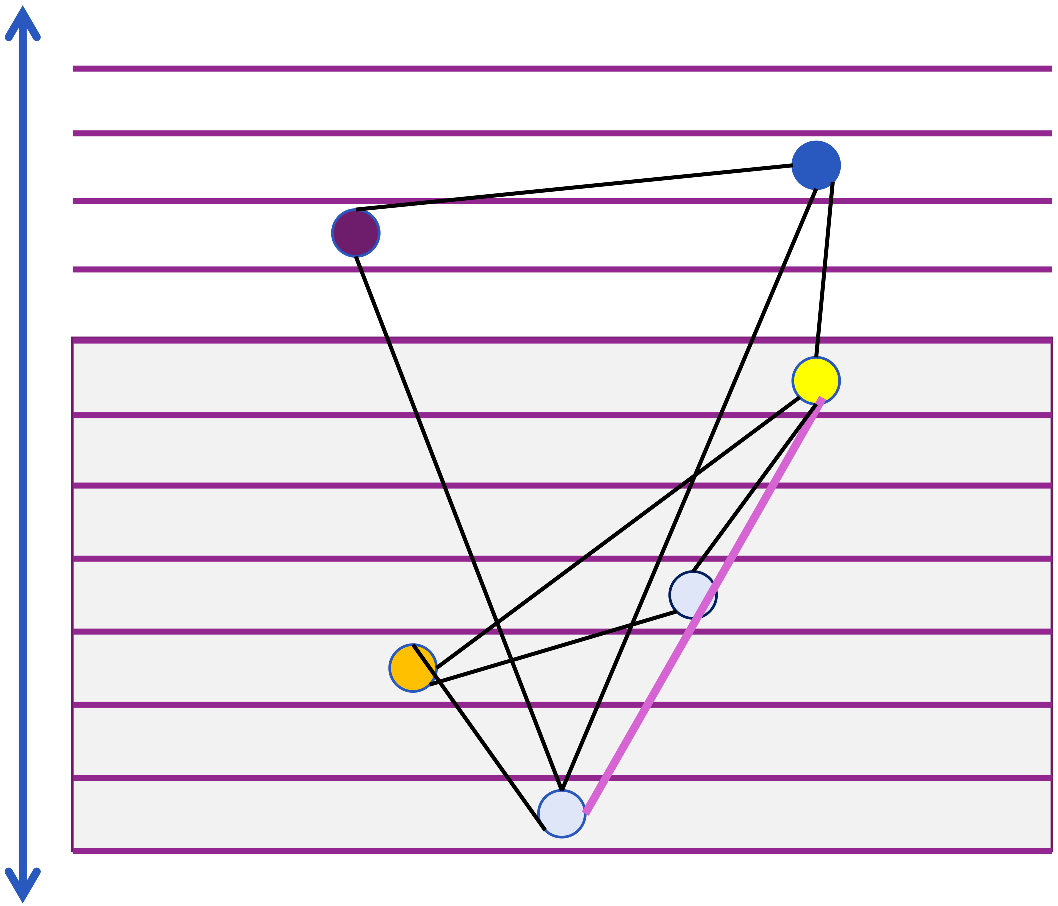


neighbors: $< 3^{\ell(v)+2}$

= Edge Insertion

Level Data Structure

Δ := Max Degree



$O(\log \Delta)$

Vertices partitioned into levels

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

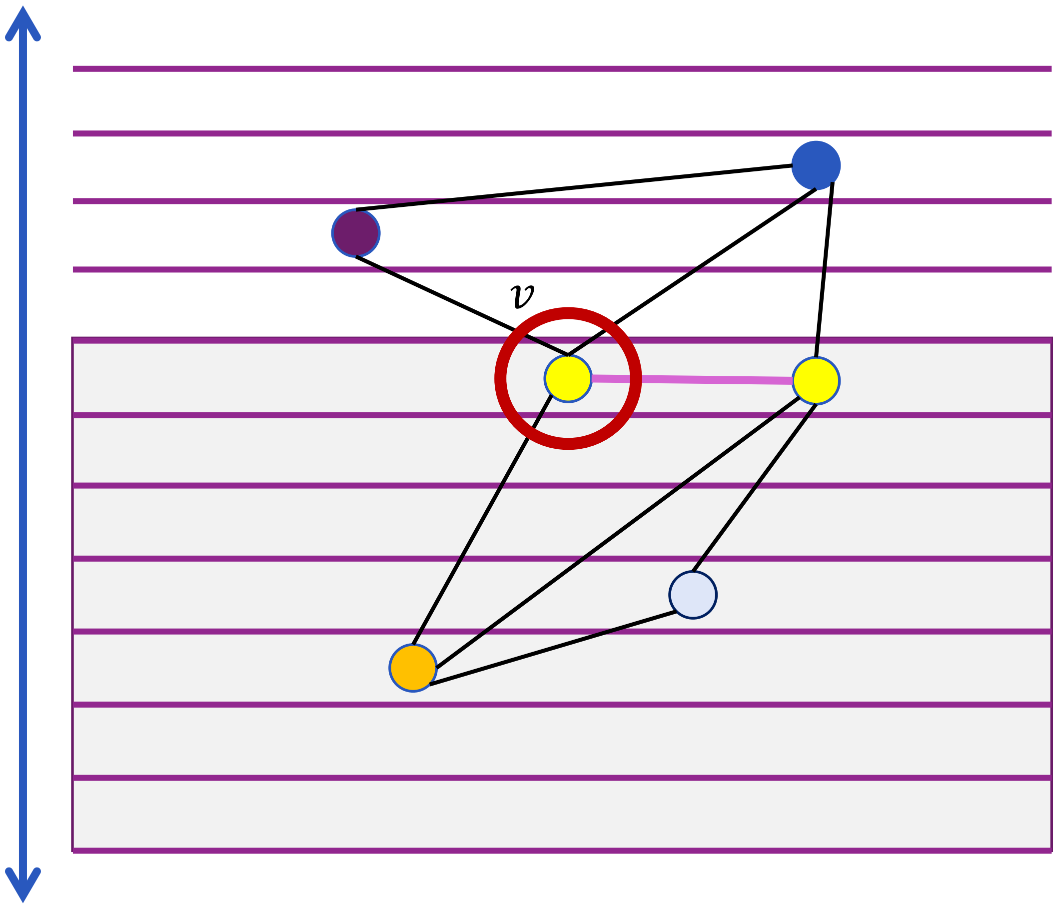
Deterministically recolor, move to bottommost level

Level Data Structure

Δ := Max Degree

$O(\log \Delta)$

Vertices partitioned into levels



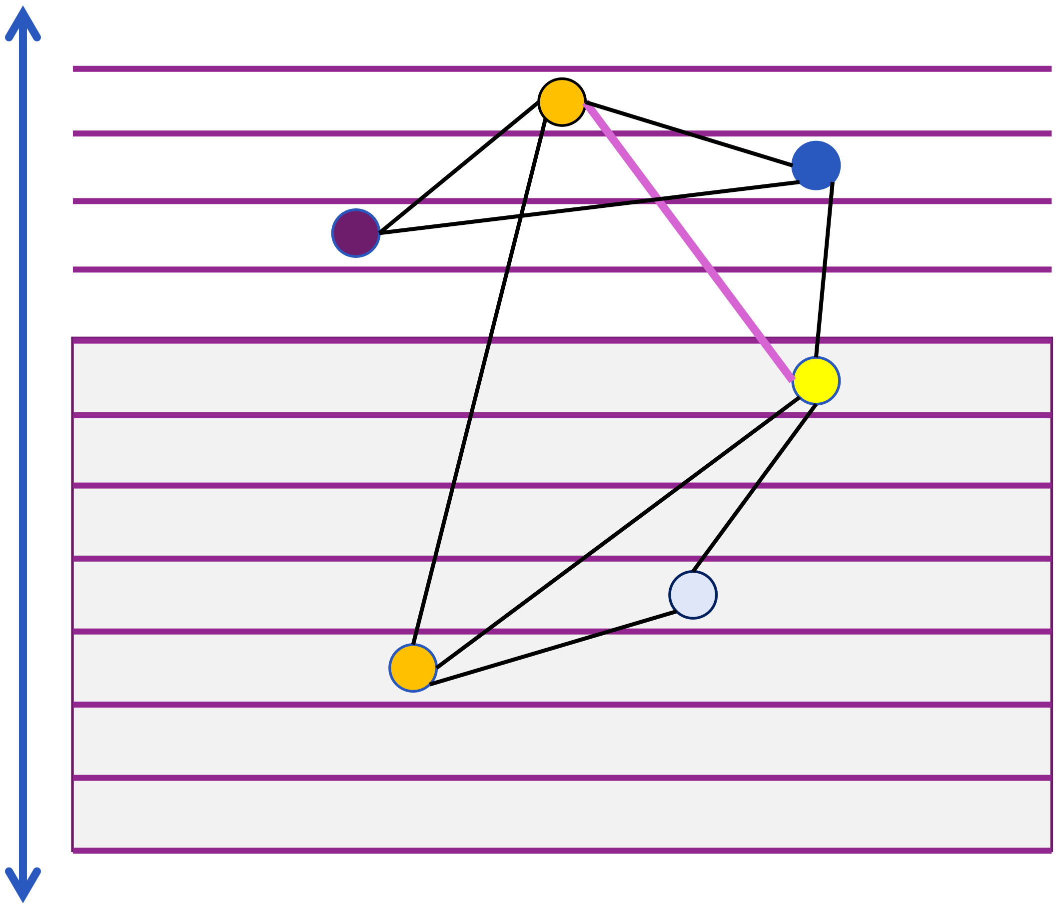
neighbors: $\geq 3^{\ell(v)+2}$

Level Data Structure

Δ := Max Degree

$O(\log \Delta)$

Vertices partitioned into levels



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

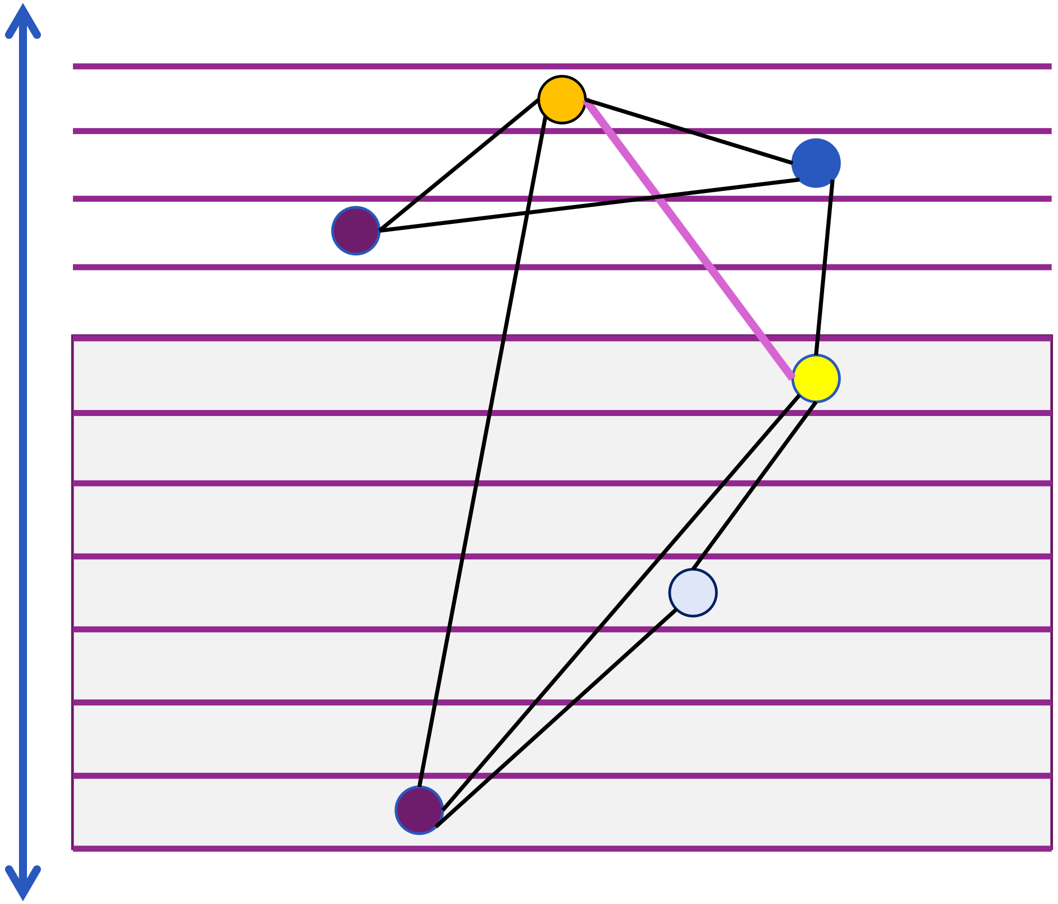
Move to lowest level where # neighbors $< 3^{\ell(v)+2}$ randomly recolor

Level Data Structure

Δ := Max Degree

$O(\log \Delta)$

Vertices partitioned into levels



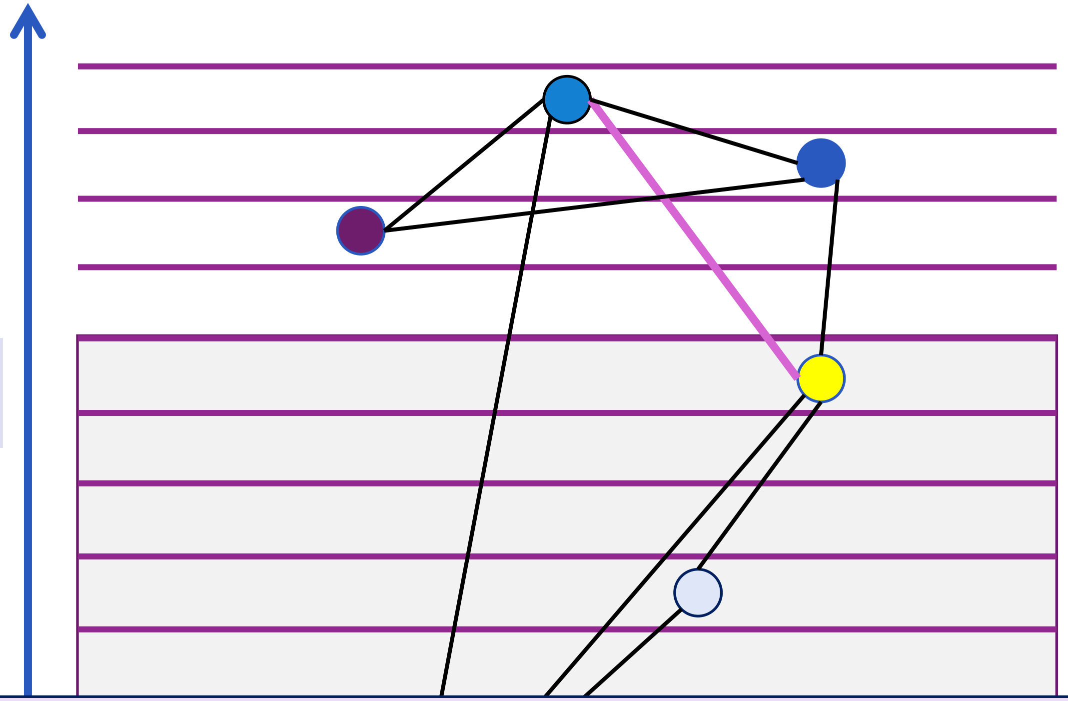
neighbors: $\geq 3^{\ell(v)+2}$

Recursively resolve new conflicts

Level Data Structure

Δ := Max Degree

$O(\log \Delta)$



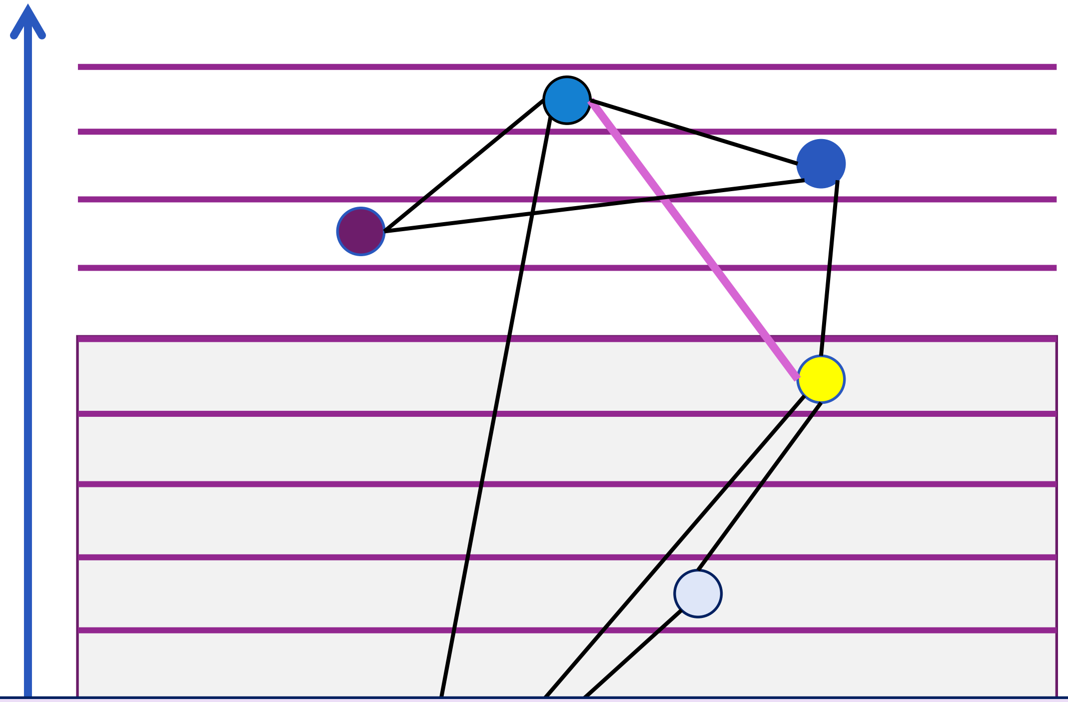
neighbors: $\geq 3^{\ell(v)+2}$

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

Level Data Structure

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

OPEN: more efficient practically?



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

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

Summary of Results

Static Algorithms

Structural Rounding

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§] Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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

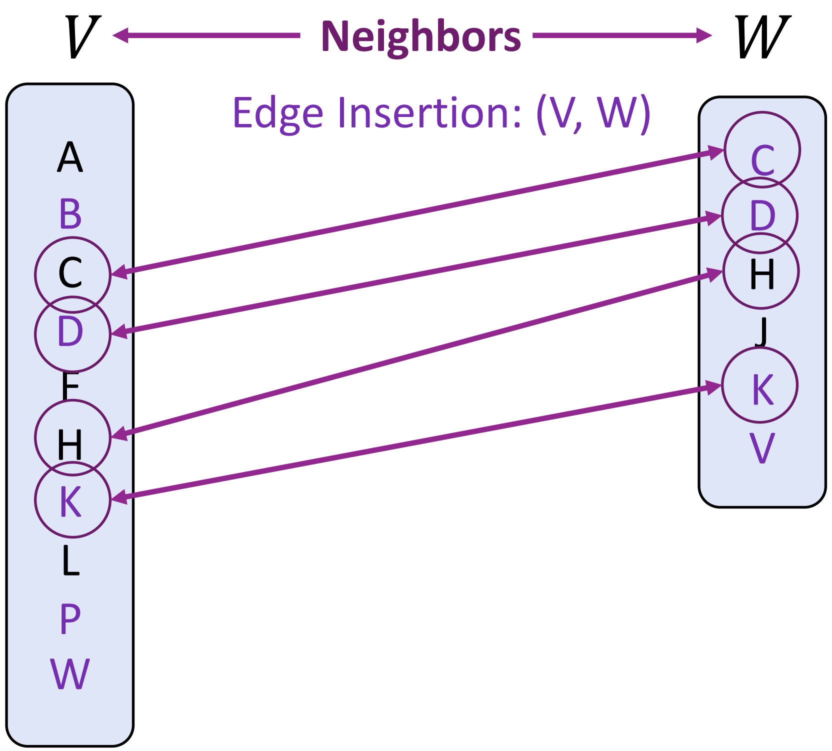
Shangdi Yu

Simple SotA Batch-Dynamic Algorithm

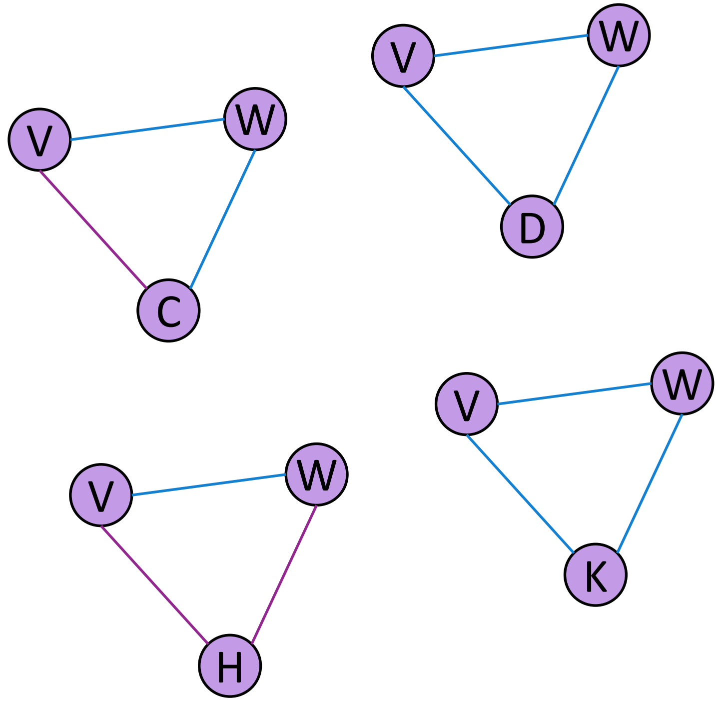
- Makkar, Bader, Green HiPC 2017

Purple: new neighbors
Black: old neighbors

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

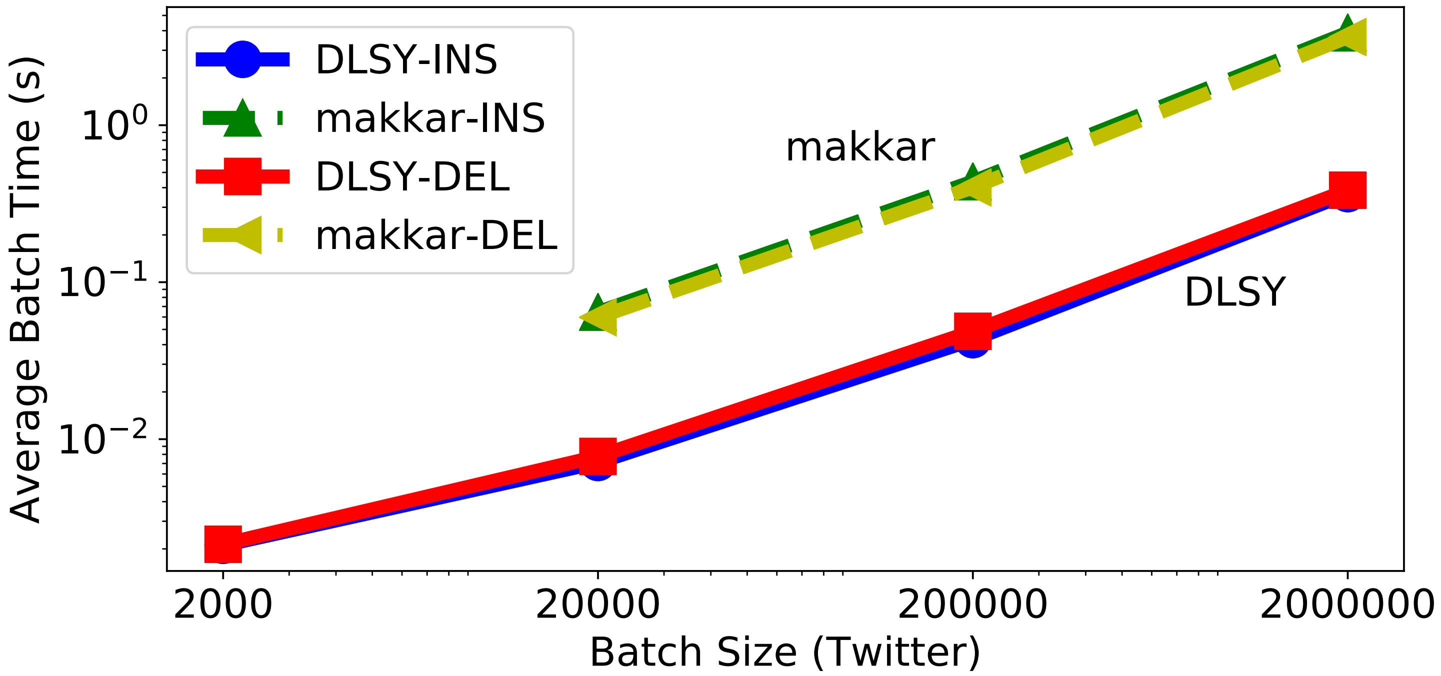


Wedges (two edges that share a vertex)



Batch-Dynamic Triangle Counting

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

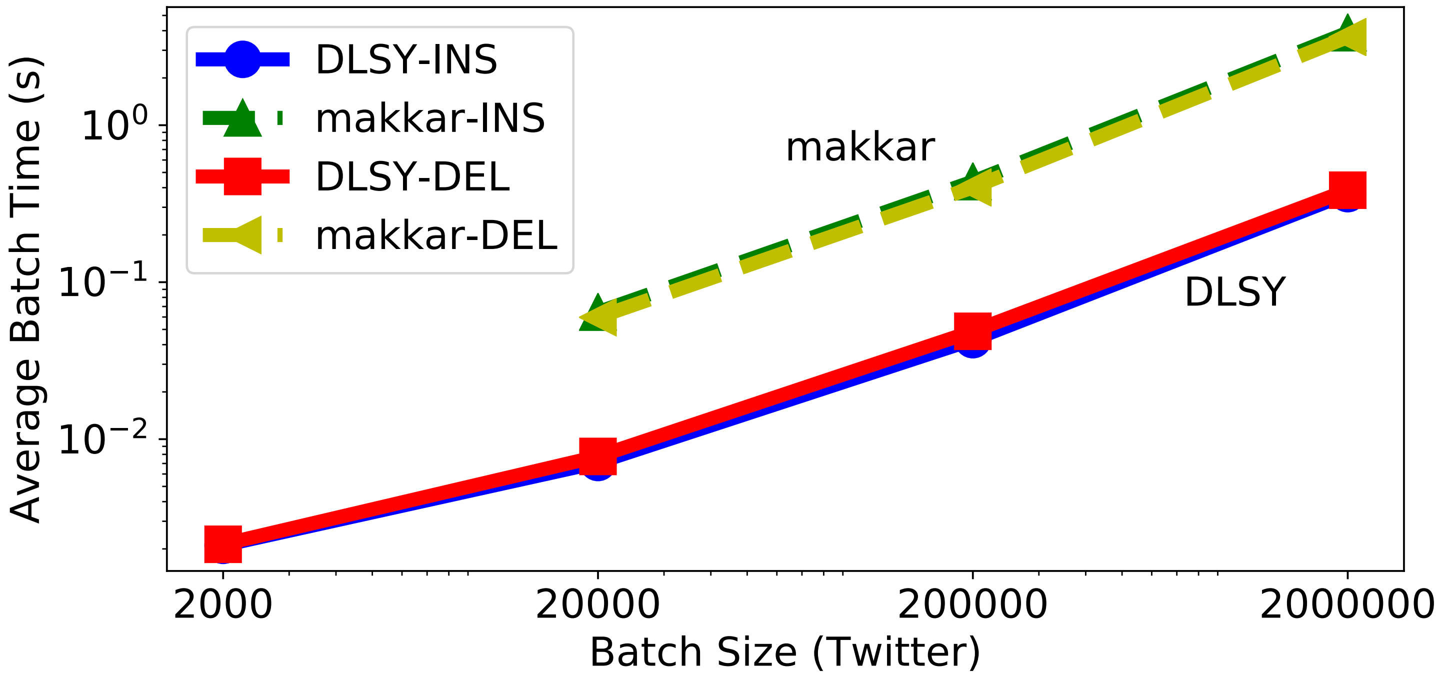


$O(\sqrt{m})$ amortized work and $O(\log n)$ depth

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

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§] Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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

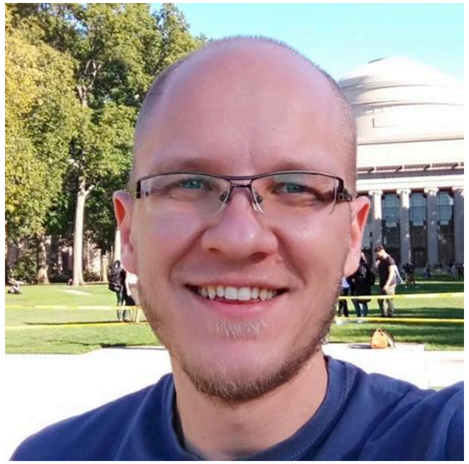
Massively Parallel Algorithms for Subgraph Counting

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

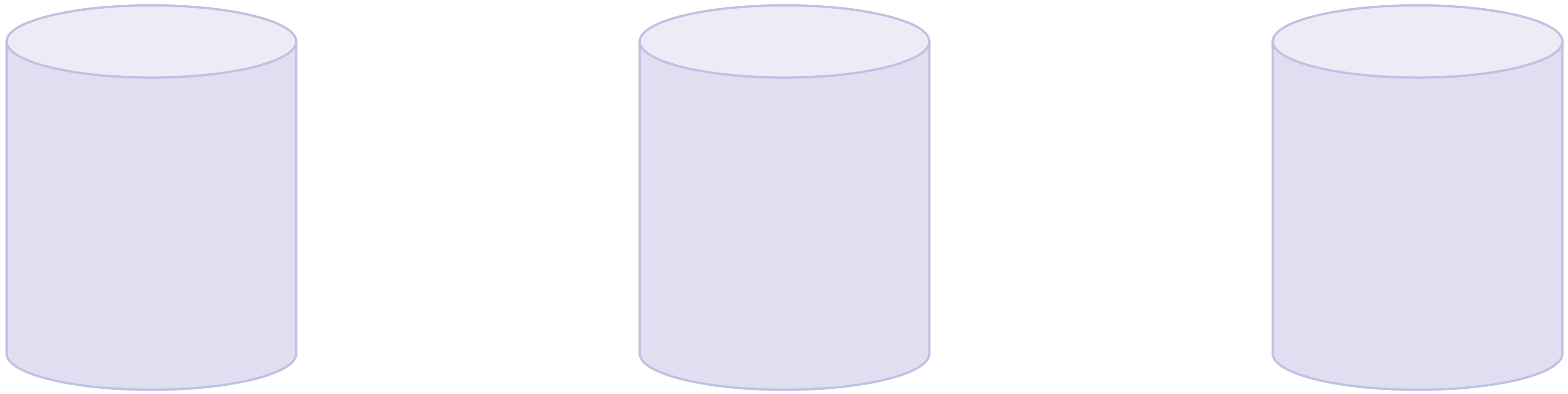
Ronitt Rubinfeld

MPC Model Definition

- *M* machines
- Synchronous rounds

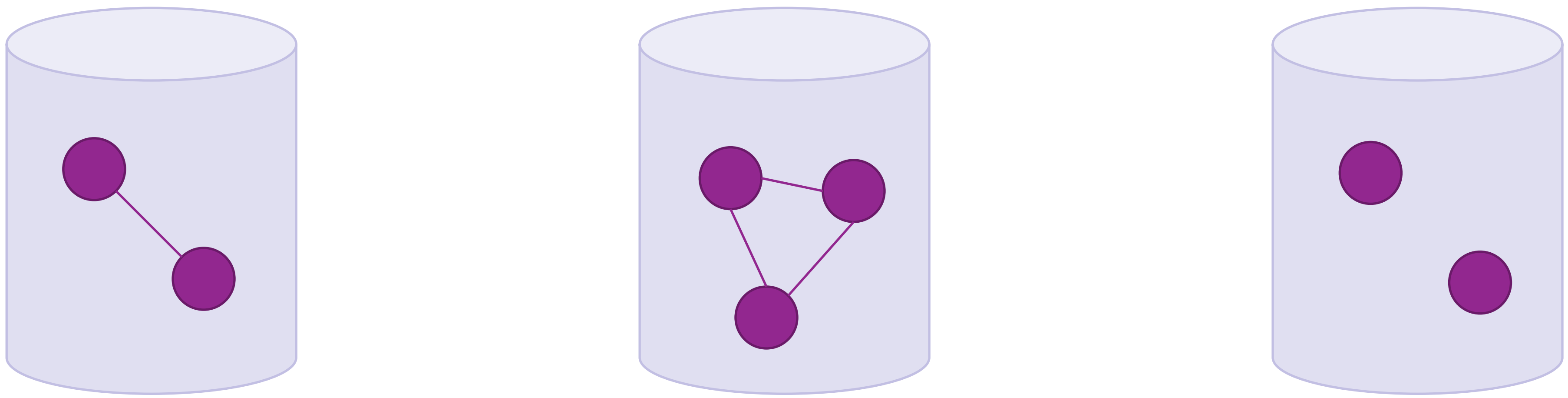
MPC Model Definition

- *M* machines
- Synchronous rounds



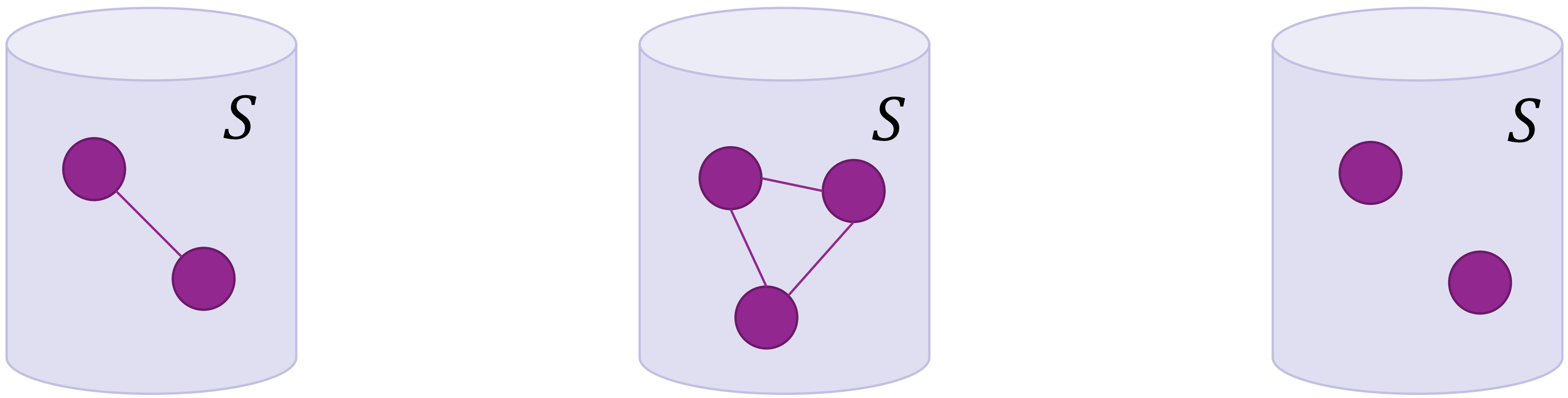
MPC Model Definition

- *M* machines
- Synchronous rounds



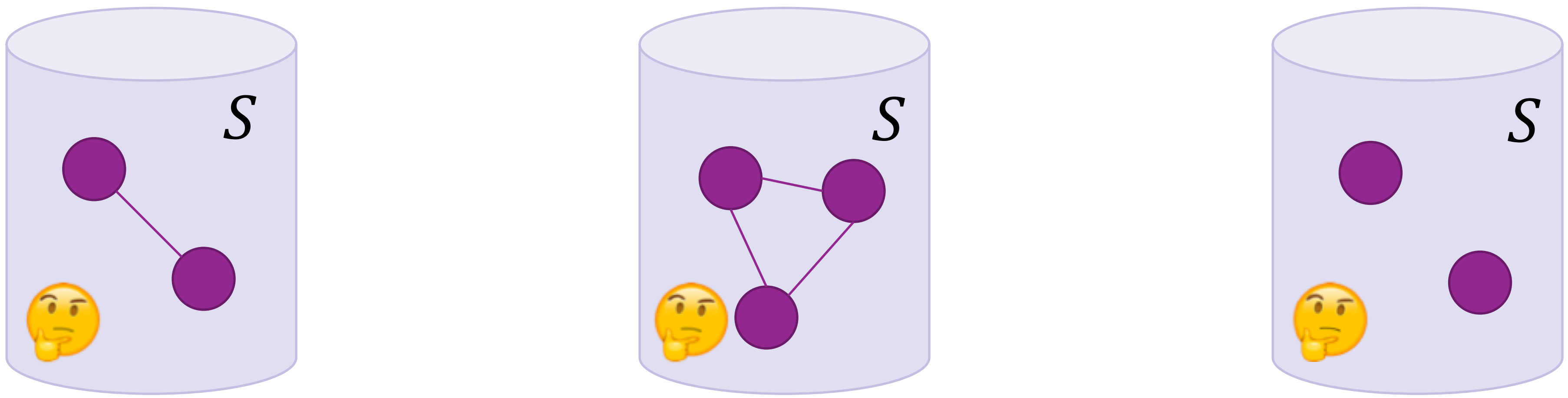
MPC Model Definition

- *M* machines
- Synchronous rounds



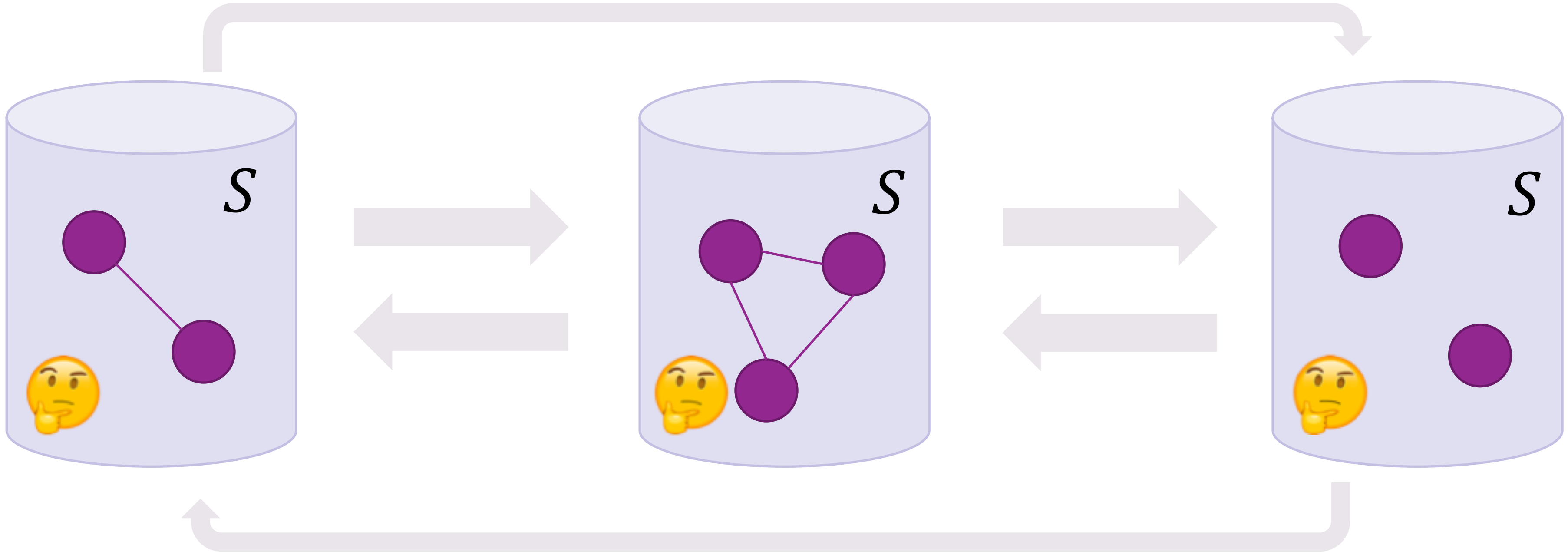
MPC Model Definition

- M machines
- Synchronous rounds



MPC Model Definition

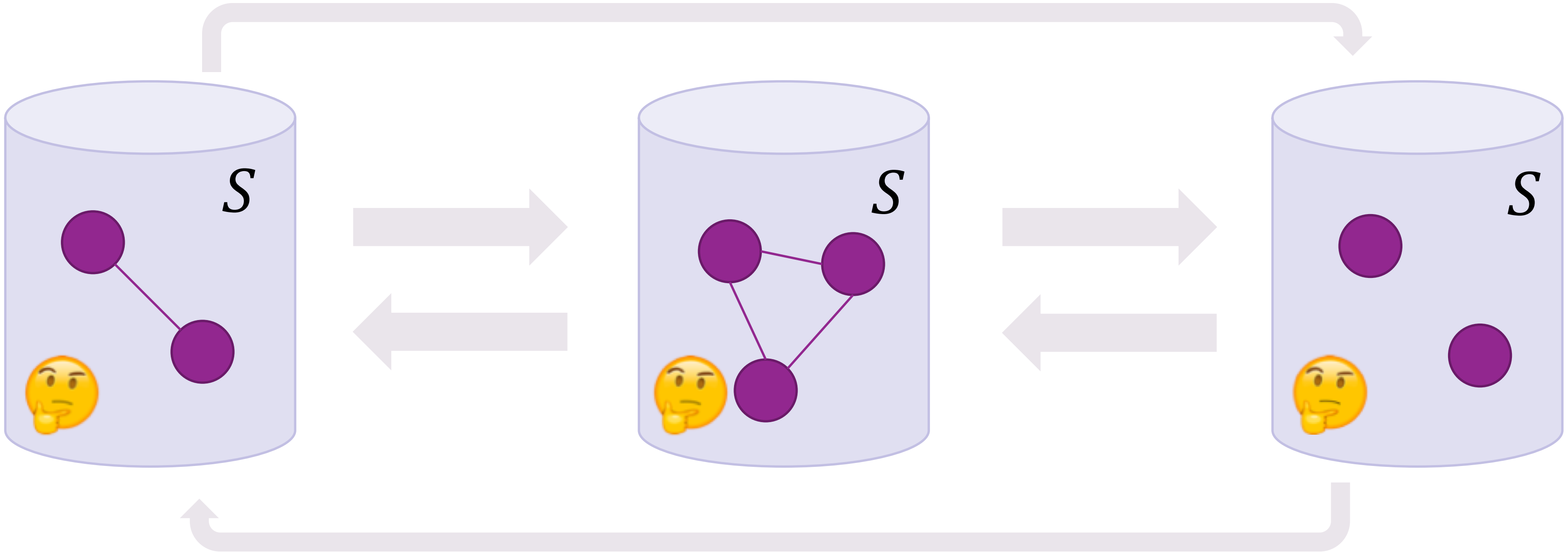
- *M* machines
- Synchronous rounds



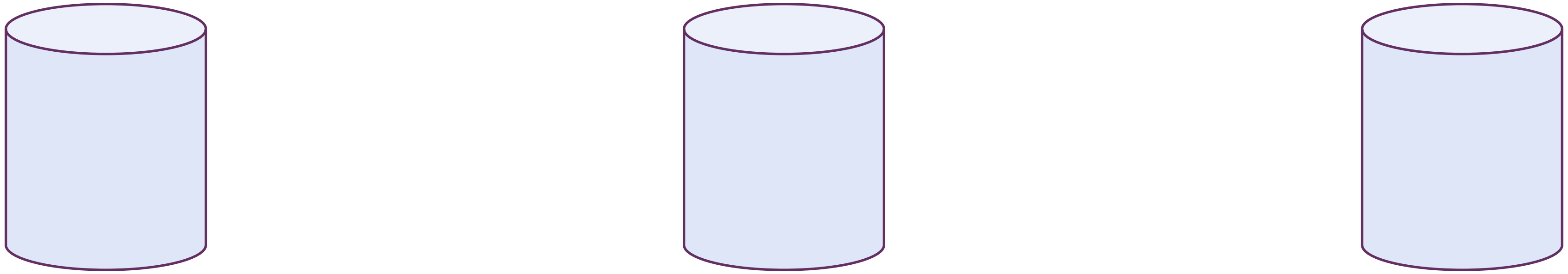
MPC Model Definition

- M machines
- Synchronous rounds

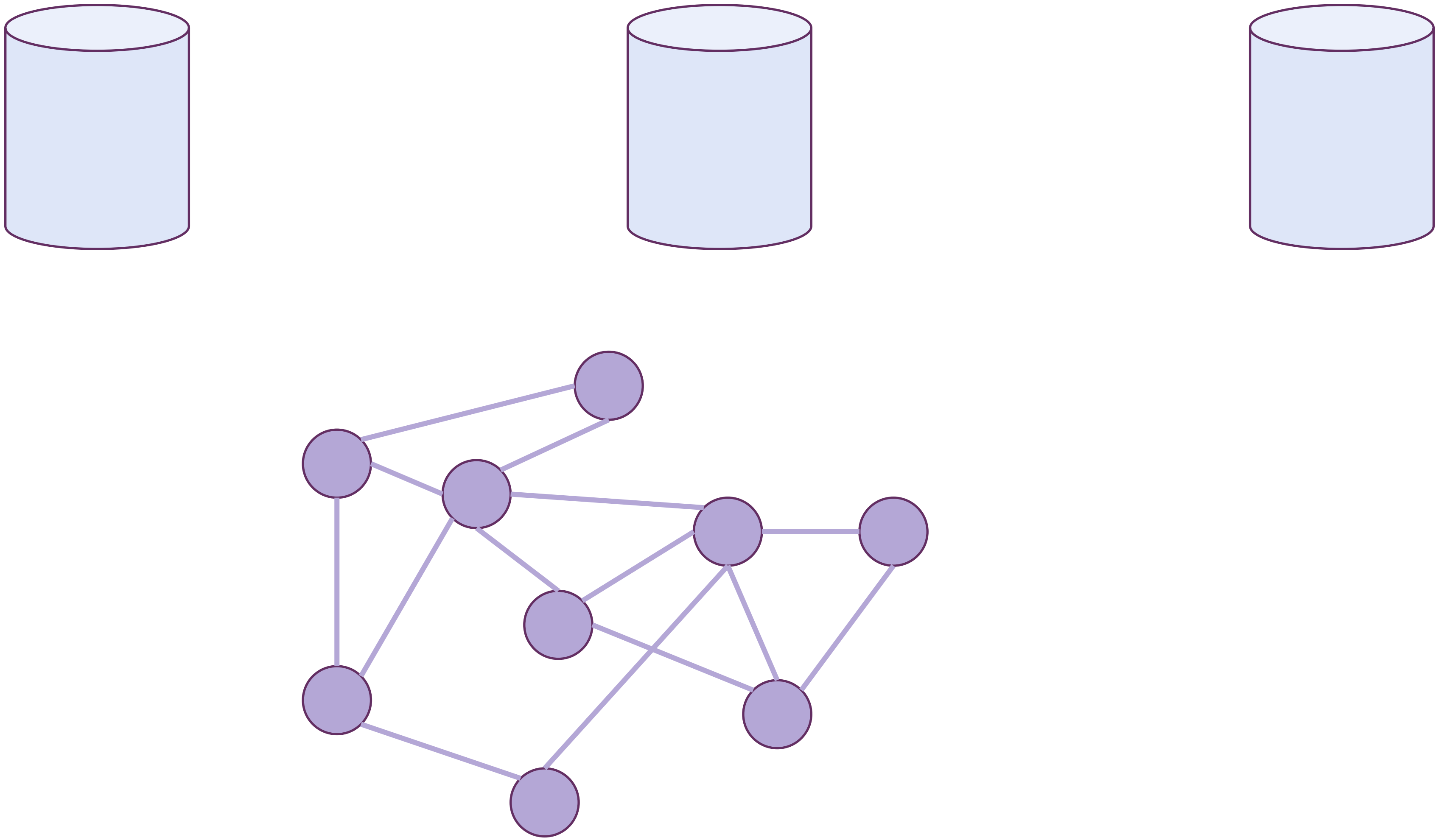
Total Space: $\boldsymbol{M} \cdot \boldsymbol{S}$



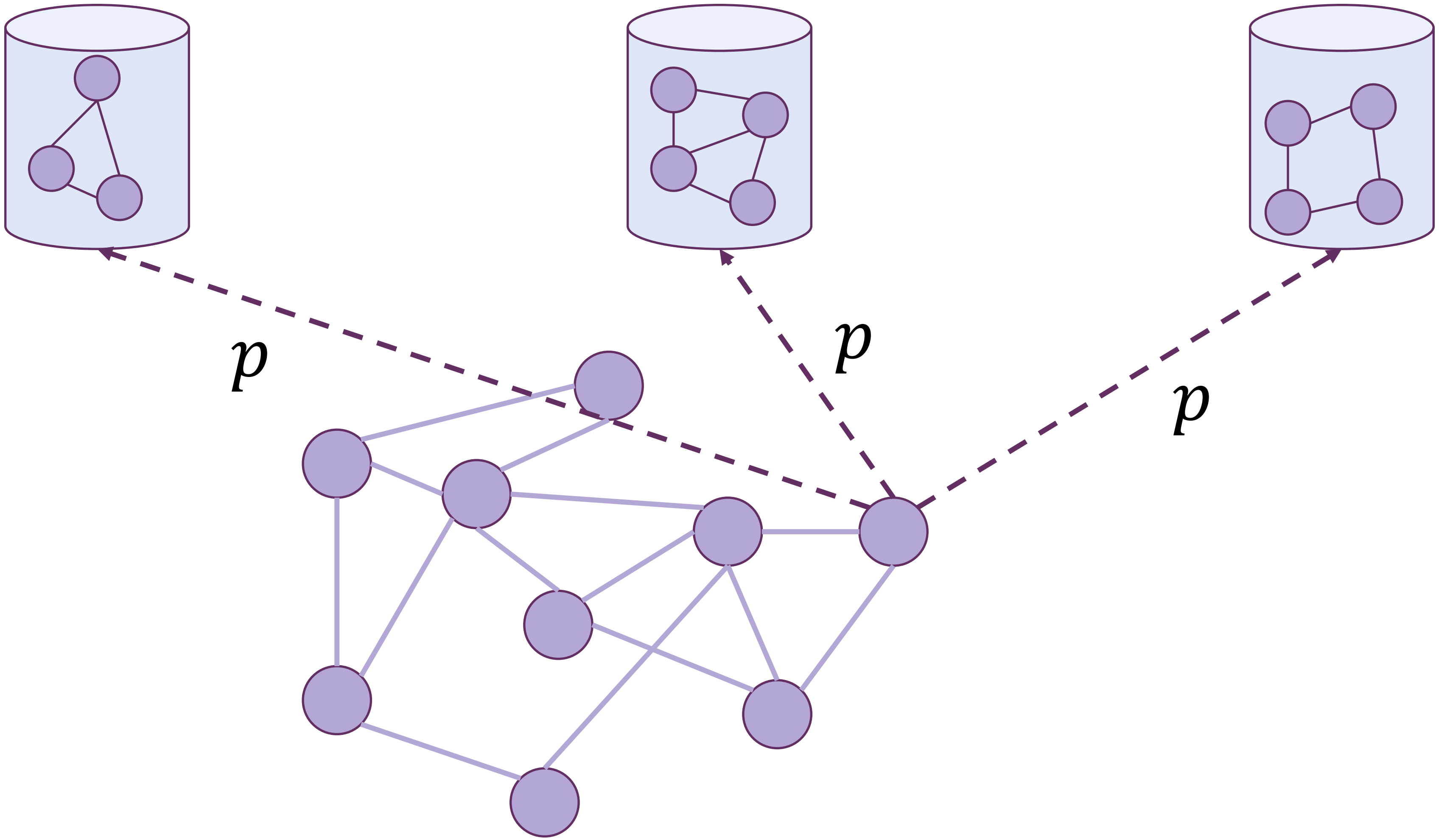
Approximate Triangle Counting



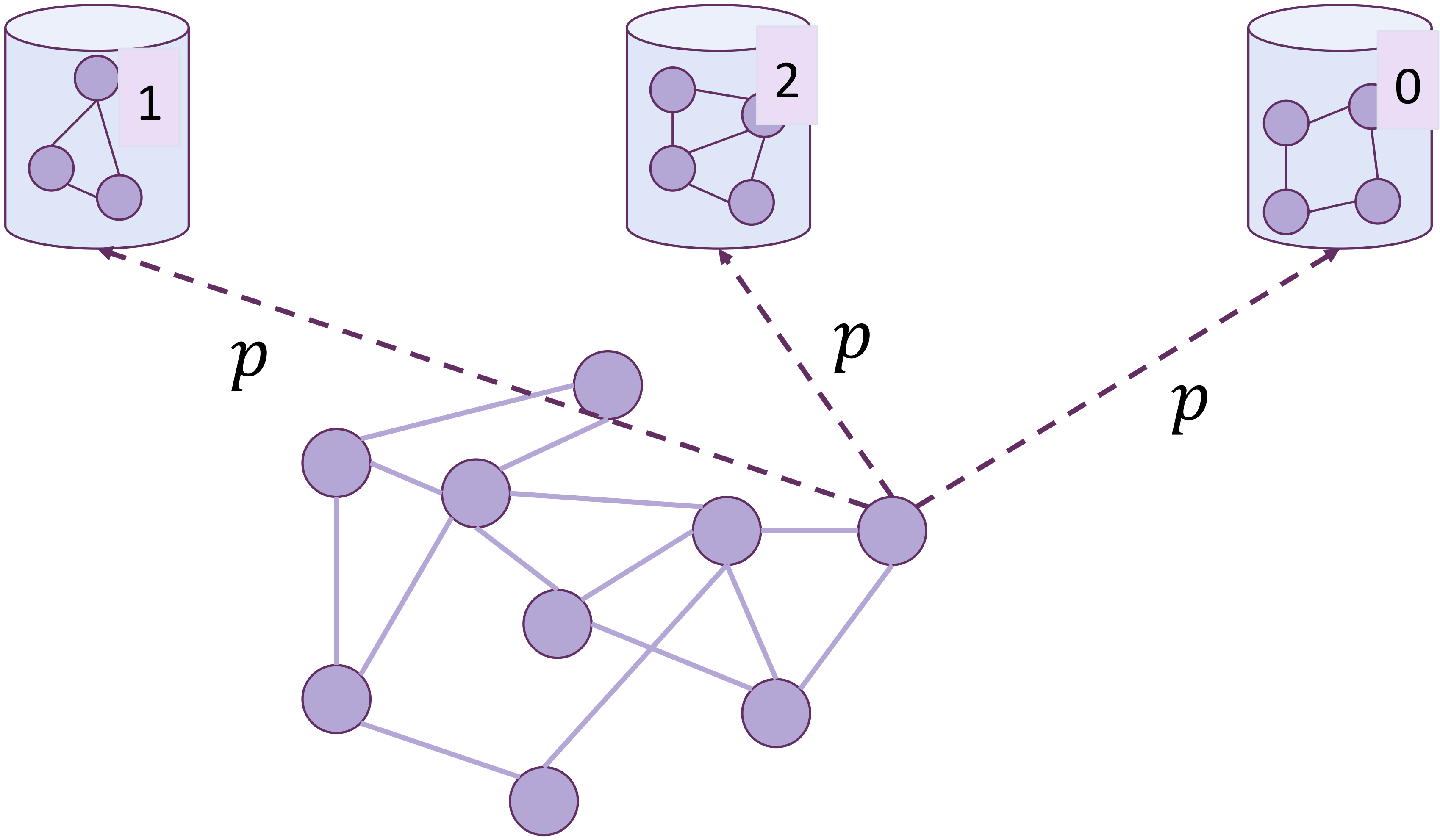
Approximate Triangle Counting



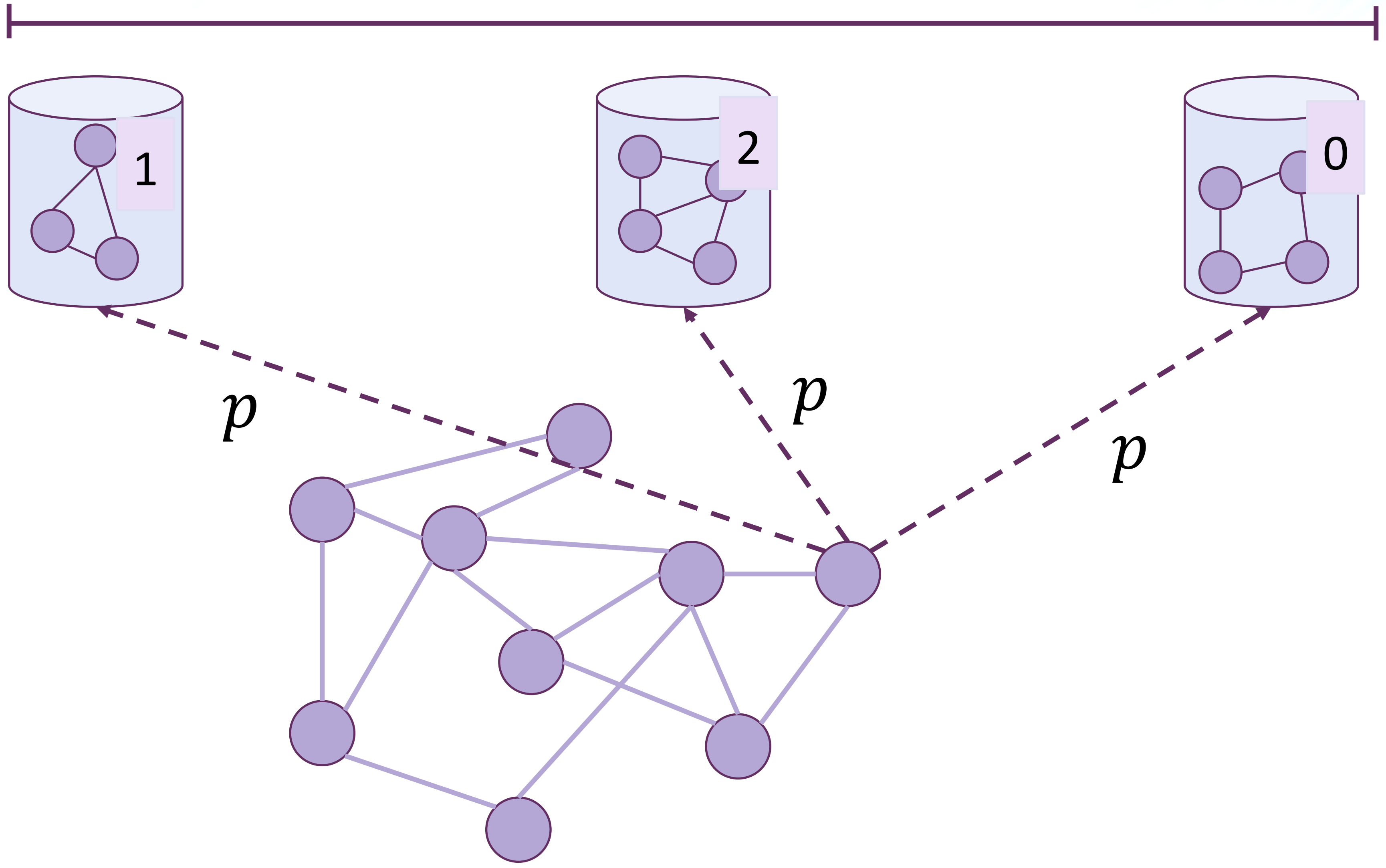
Approximate Triangle Counting



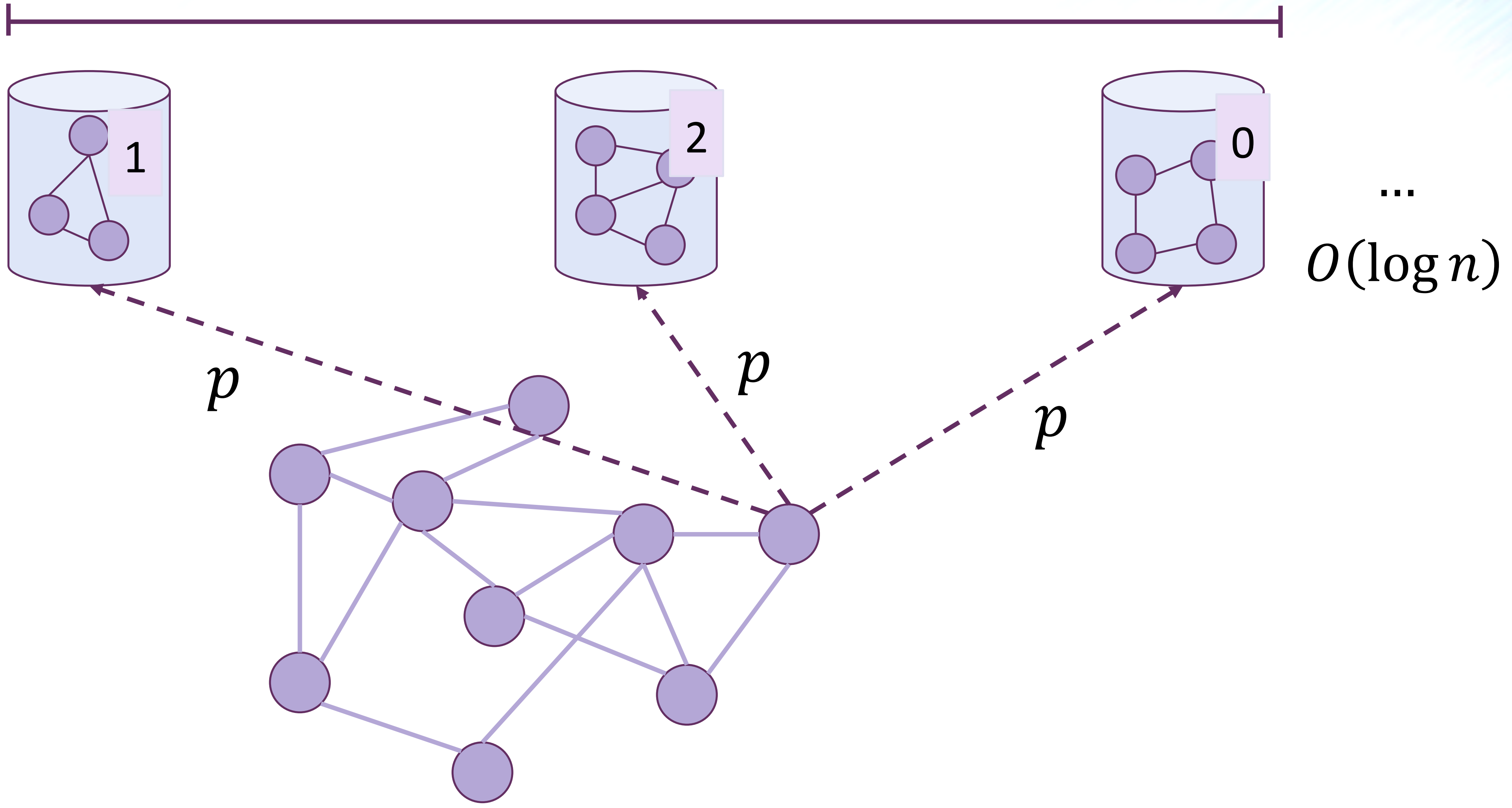
Approximate Triangle Counting



Approximate Triangle Counting



Approximate Triangle Counting



Approximate Triangle Counting

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

https://github.com/qqliu/mpc-triangle-count-exact-approx-simulations

Graphs from Stanford SNAP Database

File	δ	Partition Approximation	Our Approximation
ego-Facebook	0.5	0.62	1.31
feather-lastfm-social	0.5	5.41	1.08
ca-GrQc	0.5	4.53	1.64
ca-HepPh	0.5	0.66	1.22
ego-Facebook	0.75	0.75	0.97
ca-GrQc	0.75	5.82	0.82
ca-HepPh	0.75	5.90	0.86
musae-twitch (DE)	0.75	0.74	0.95
oregon1_010519	0.75	0.60	0.71

Better approximations on all graphs

Approximation Factor (Same Machine Memory) Compared with PT12

Summary of Results

Static Algorithms

Structural Rounding

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§] Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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

Scheduling with Communication Delay in Near-Linear Time

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, Zoya Svitkina, Erik Vee, Joshua R. Wang

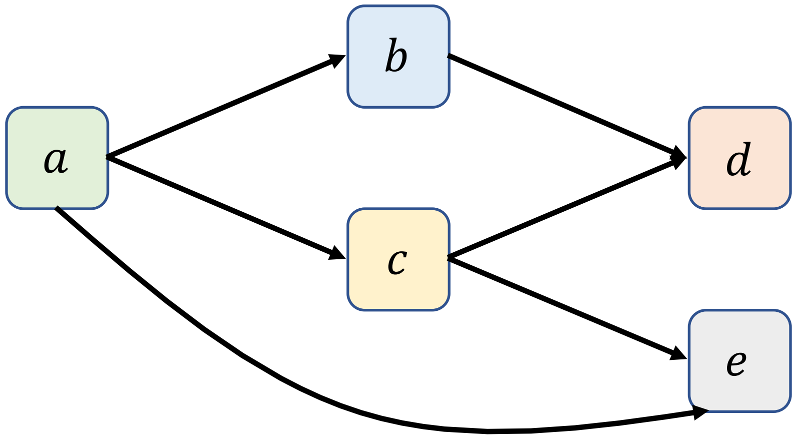
Manish Purohit

Zoya Svitkina

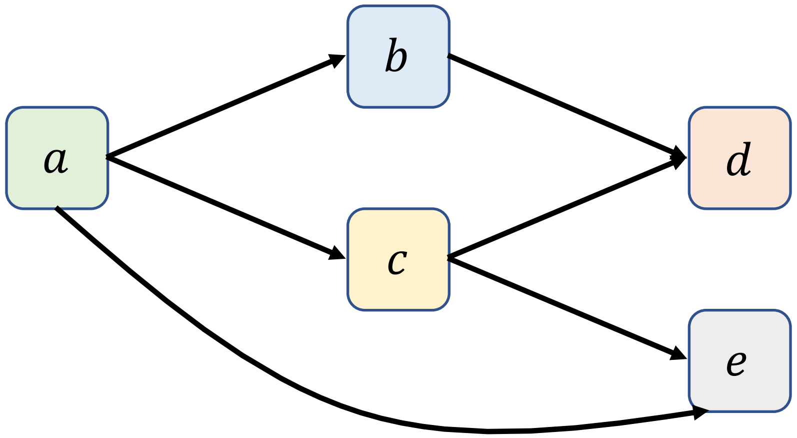
Erik Vee

Joshua R. Wang

Precedence-Constrained Jobs Modeled as DAG

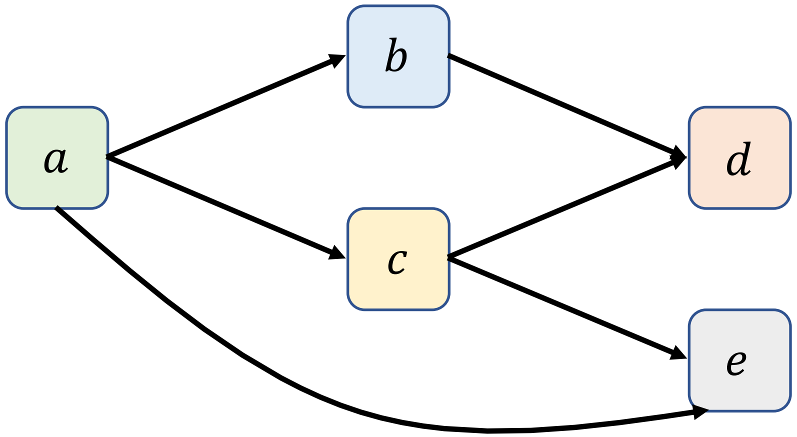


Precedence-Constrained Jobs Modeled as DAG



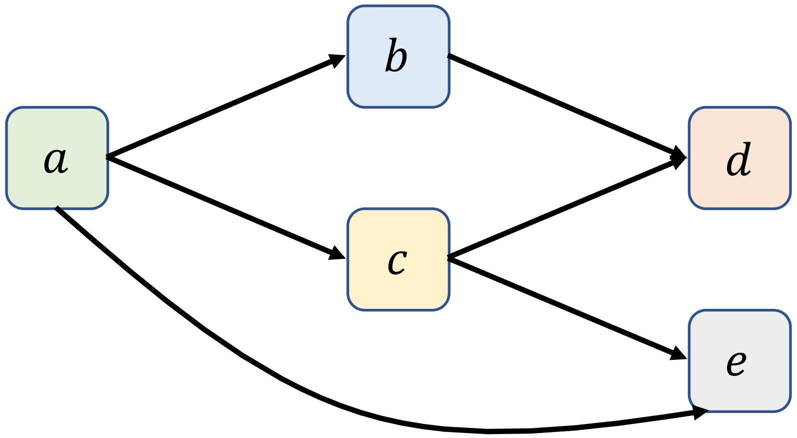
- Nodes scheduled according to some topological sort

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

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

- 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

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

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§] Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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

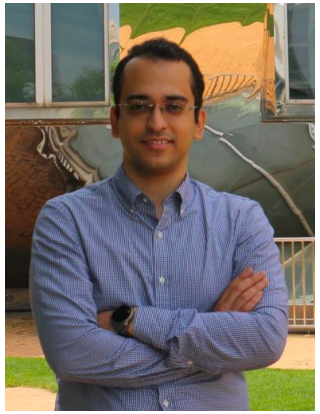
Structural Rounding

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]



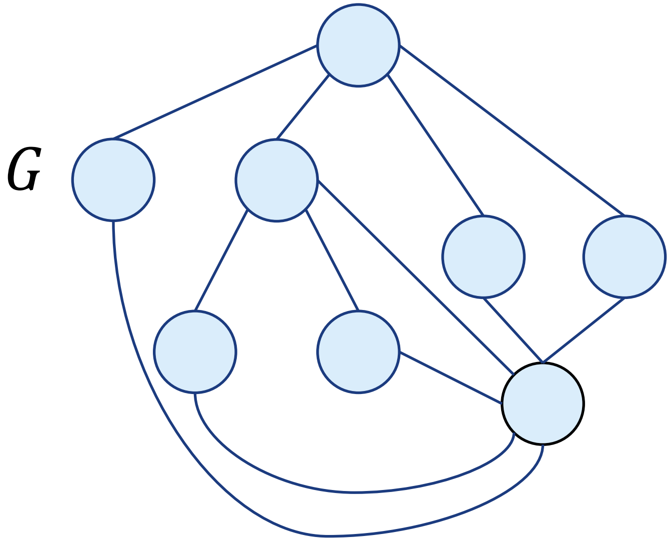



Erik D. Demaine Timothy D. Goodrich Kyle Kloster Brian Lavallee Blair D. Sullivan Ali Vakilian Andrew van der Poel

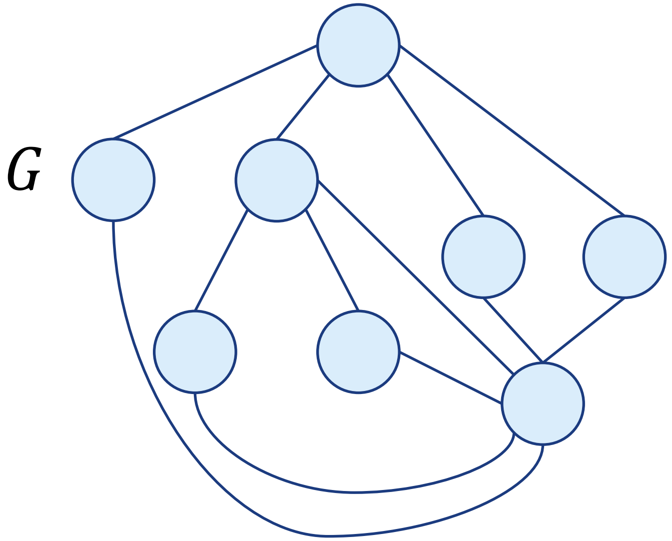
(Minimization) Problem $\mathcal{P}$ with optimal solution $OPT(G)$, e.g. vertex cover

Structural Rounding



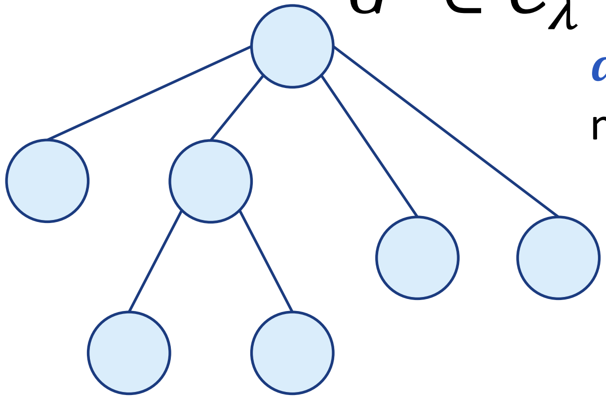
(Minimization) Problem $\mathcal{P}$ with optimal solution $OPT(G)$, e.g. vertex cover

Structural Rounding



Edit to a Structured Graph Class

(α, β)-approximation



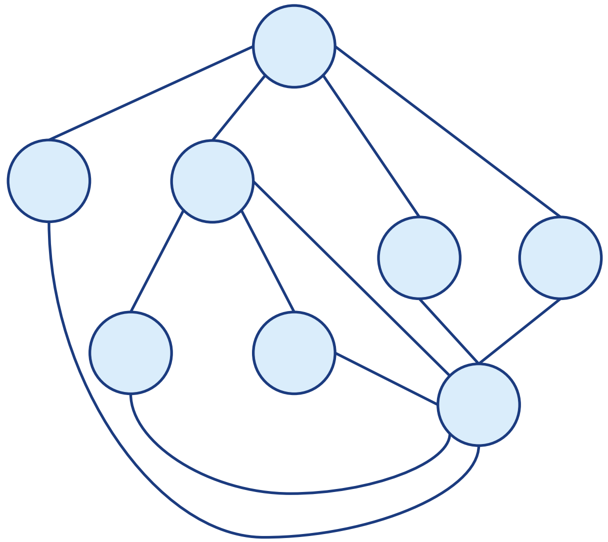
α-approx. min number of edits X

β-approx. on $\mathcal{C}_\lambda$
e.g. $\lambda \leq \beta \cdot \lambda^*$

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

Structural Rounding

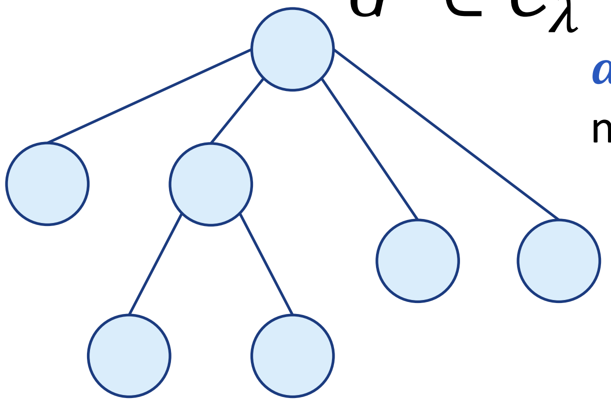
G



Edit to a Structured Graph Class

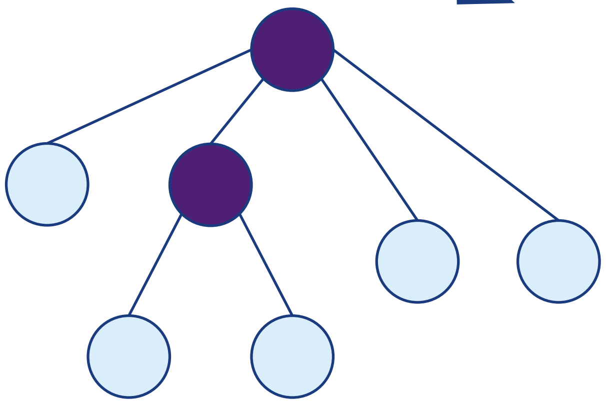
(α, β)-approximation

$G' \in \mathcal{C}_\lambda$



α-approx. min number of edits X

β-approx. on $\mathcal{C}_\lambda$ e.g. $\lambda \leq \beta \cdot \lambda^*$



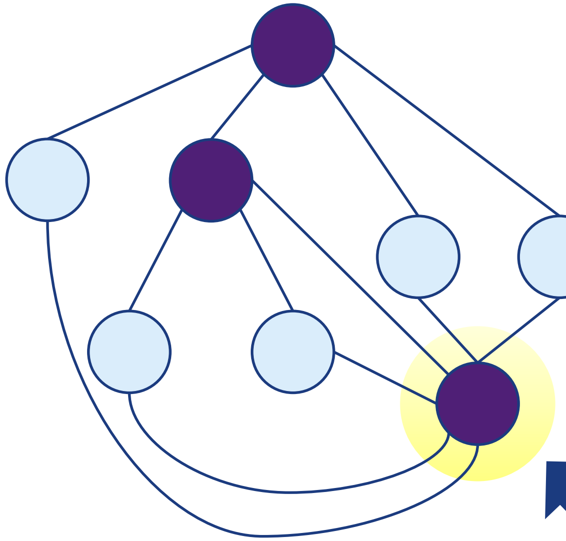
G'

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

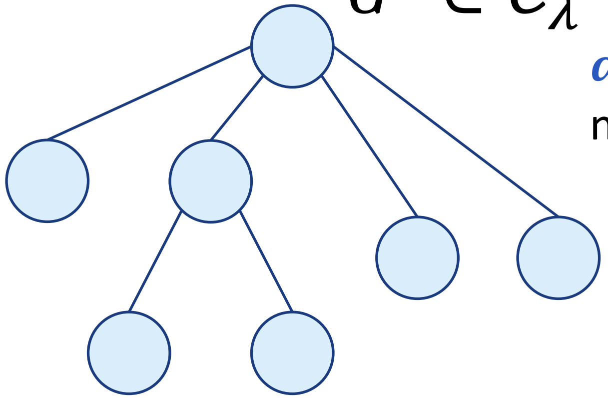
G



Edit to a Structured Graph Class

$(\boldsymbol{\alpha}, \boldsymbol{\beta})$**-approximation**

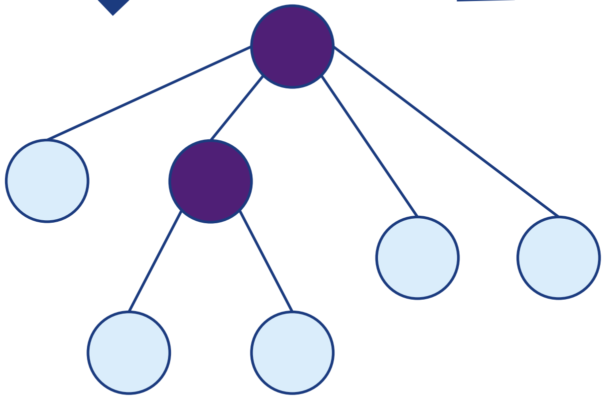
$G' \in \mathcal{C}_\lambda$



$\boldsymbol{\alpha}$**-approx.** min number of edits X

$\boldsymbol{\beta}$**-approx.** on $\mathcal{C}_\lambda$ e.g. $\lambda \leq \beta \cdot \lambda^*$

Solve the problem in the edited graph G': $OPT(G')$ where we assume $OPT(G') \leq OPT(G)$



G'

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

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

Summary of Results

Static Algorithms

Structural Rounding

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

Erik D. Demaine[1], Timothy D. Goodrich[2], Kyle Kloster[2], Brian Lavallee[2], Quanquan C. Liu[1], Blair D. Sullivan[2], Ali Vakilian[1], and Andrew van der Poel[2]

Scheduling with Communication Delay

Scheduling with Communication Delay in Near-Linear Time

Quanquan C. Liu, Manish Purohit, 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 Mitrović[*§] Ronitt Rubinfeld[¶]

Dynamic Algorithms

Parallel Dynamic k-Core Decomposition

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

Dynamic Vertex Coloring

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, 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

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

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

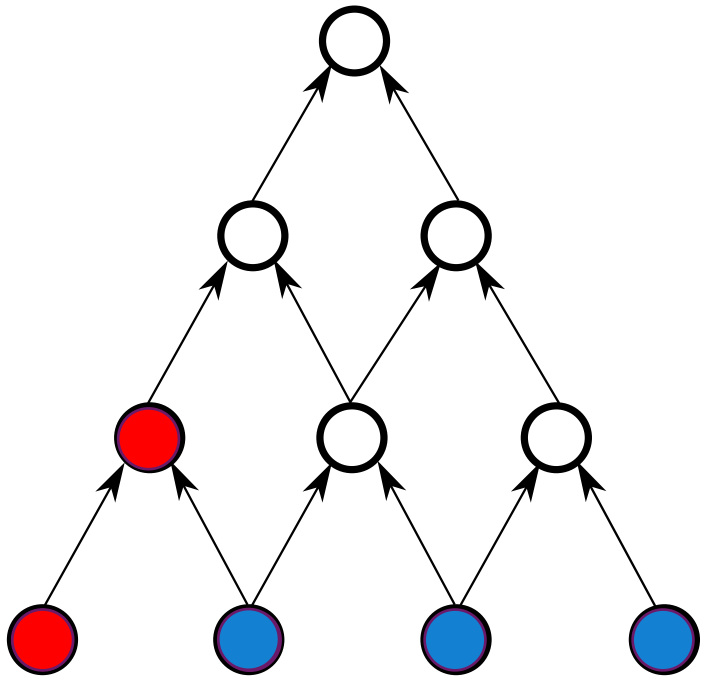
Erik D. Demaine

Thaddeus Dryja

Sunoo Park

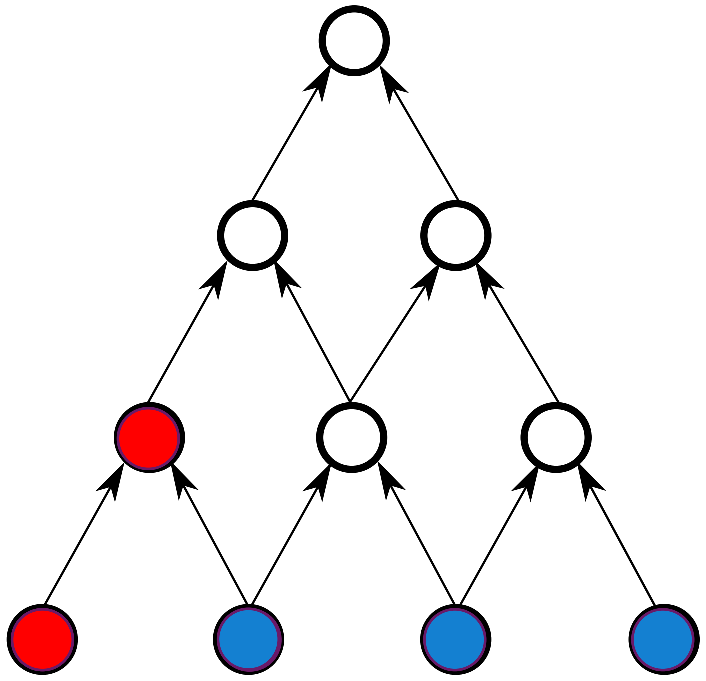
Pebble Games

- 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

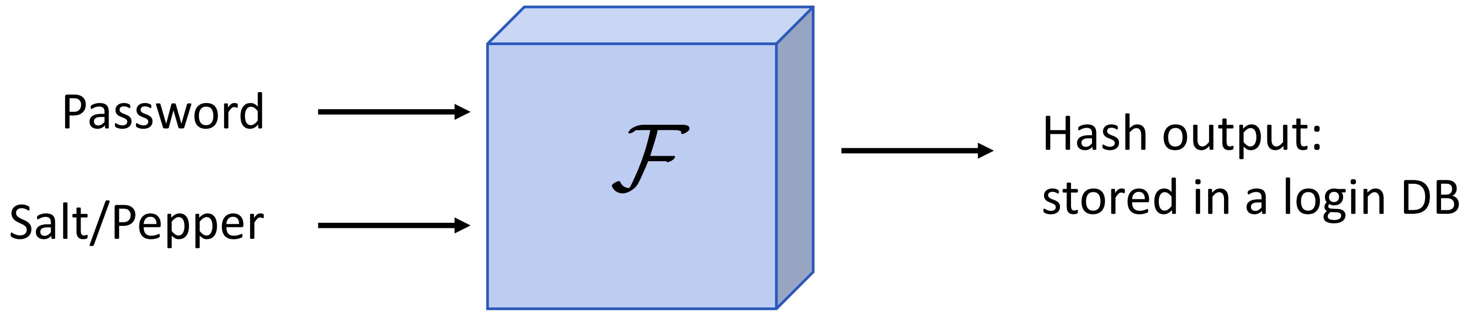


Pebble Games

- 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



Honest evaluators need only use $\mathcal{F}$ few times

Adversaries may run $\mathcal{F}$ many times (e.g. large-scale server attacks).

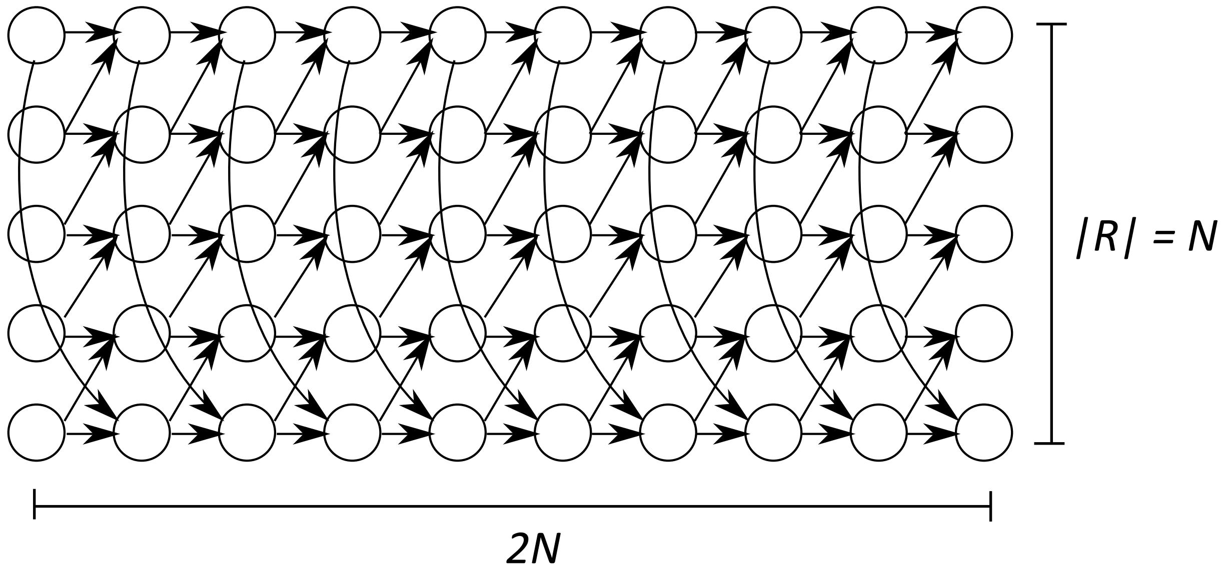
Desirable goal:

Make brute-force attacks hard by making $\mathcal{F}$ hard to compute over many hashes.

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

Candidate Constructions

- Any graph with one target node **doesn't work**
- Need at least enough target nodes so that *number hash outputs* is reasonably large
- Simple construction **cylinder graph** we implemented ($n = N^2$)



Acknowledgements

[Not included—You had to be there...]