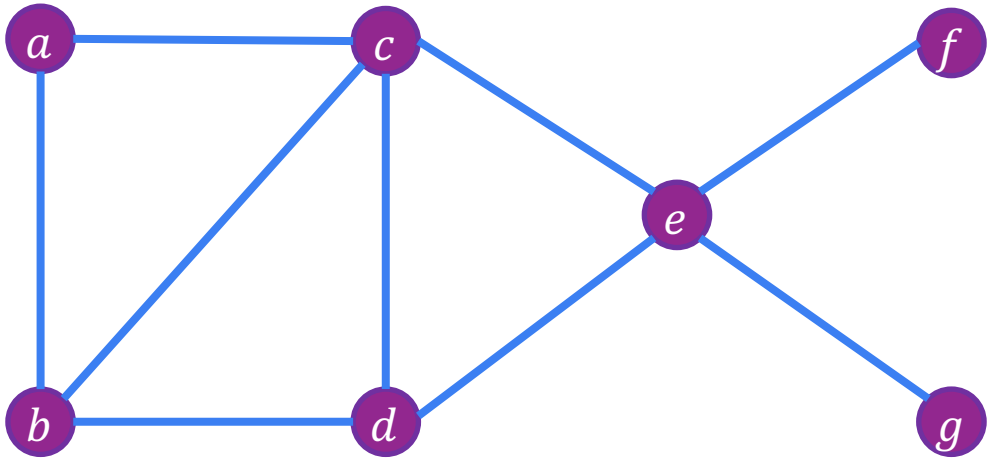
Practical Parallel Algorithms for Near-Optimal Densest Subgraphs on Massive Graphs

Pattara Sukprasert, Quanquan C Liu, Laxman Dhulipala, Julian Shun

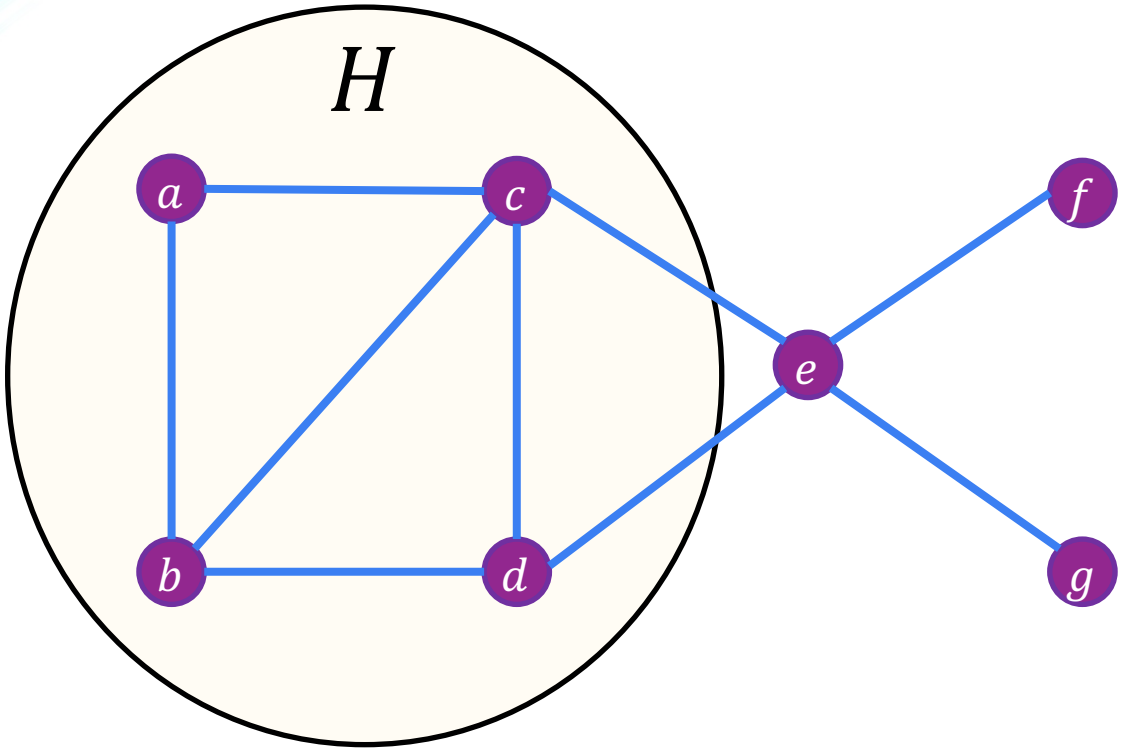
Densest Subgraph

Density of Entire Graph: $\boldsymbol{\rho(G) = \frac{|E|}{|V|} = \frac{9}{7}}$



V = set of vertices, E = set of edges

Densest Subgraph

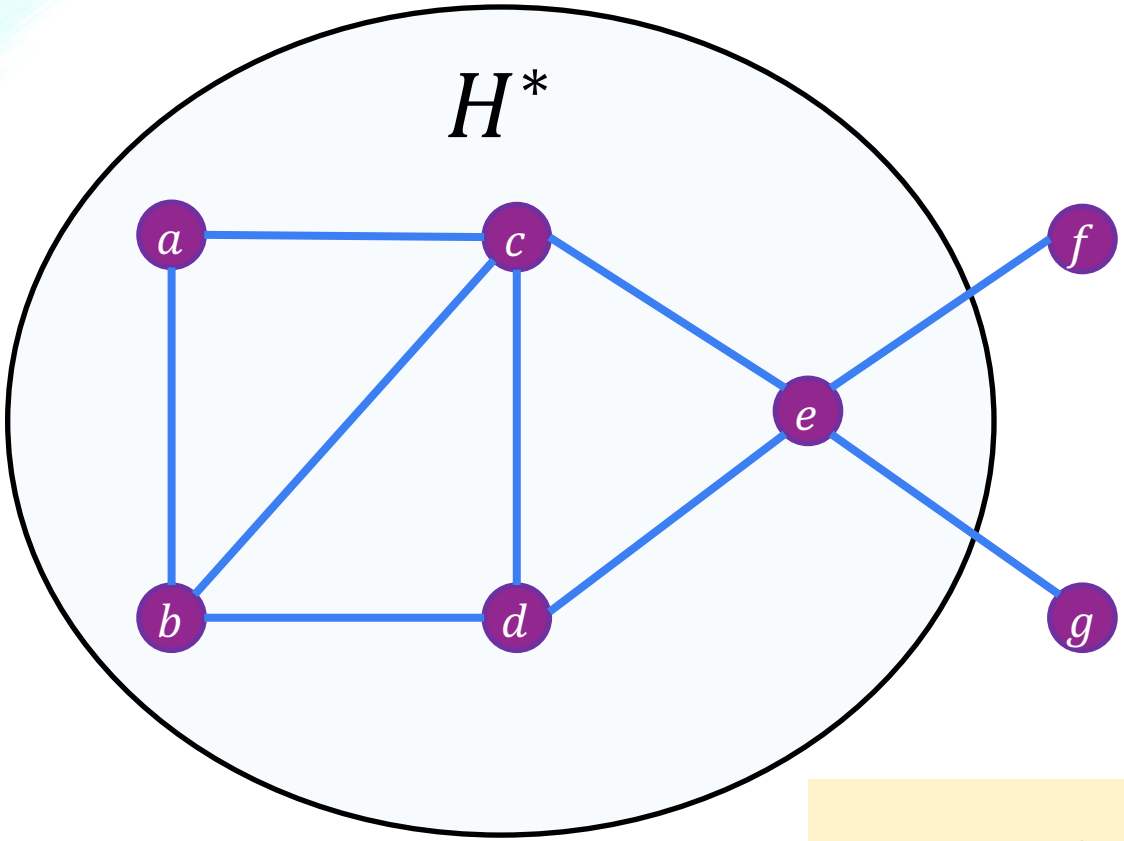


Density of Entire Graph: $\rho(G) = \frac{|E|}{|V|} = \frac{9}{7}$

Density of Subgraph H: $\rho(H) = \frac{5}{4}$

Goal: Find the maximal subgraph H^* that maximizes $\rho(H^*)$

Densest Subgraph



Density of Entire Graph: $\rho(G) = \frac{|E|}{|V|} = \frac{9}{7}$

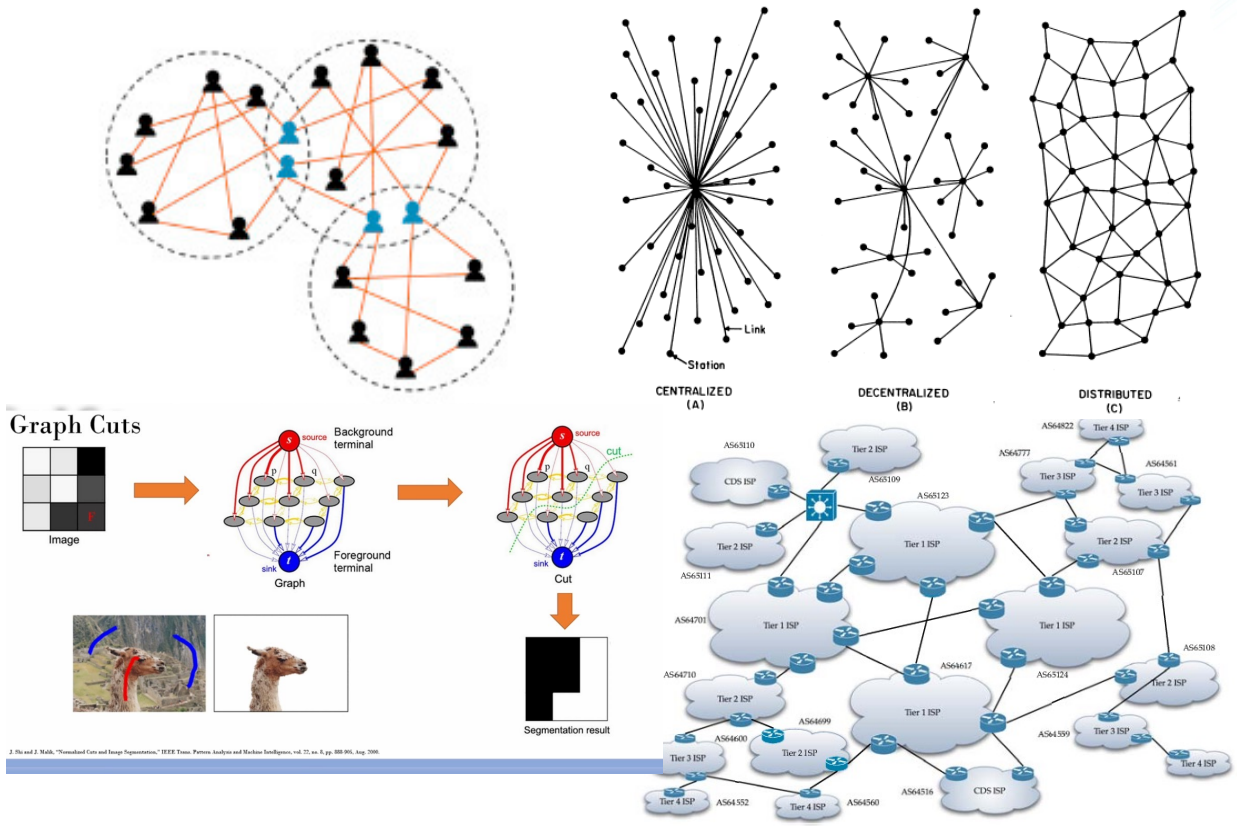
Density of Subgraph H: $\rho(H) = \frac{5}{4}$

Goal: Find the maximal subgraph H^* that maximizes $\rho(H^*)$

Density of Densest Subgraph H^*: $\rho(H^*) = \frac{7}{5}$

Applications and Related Problems

- Community detection in social networks
- Image processing
- Finding weak spots in data and communication networks
- IP traffic routing
- Spam and fraud detection
- Ecological network analysis
- Recommendation systems



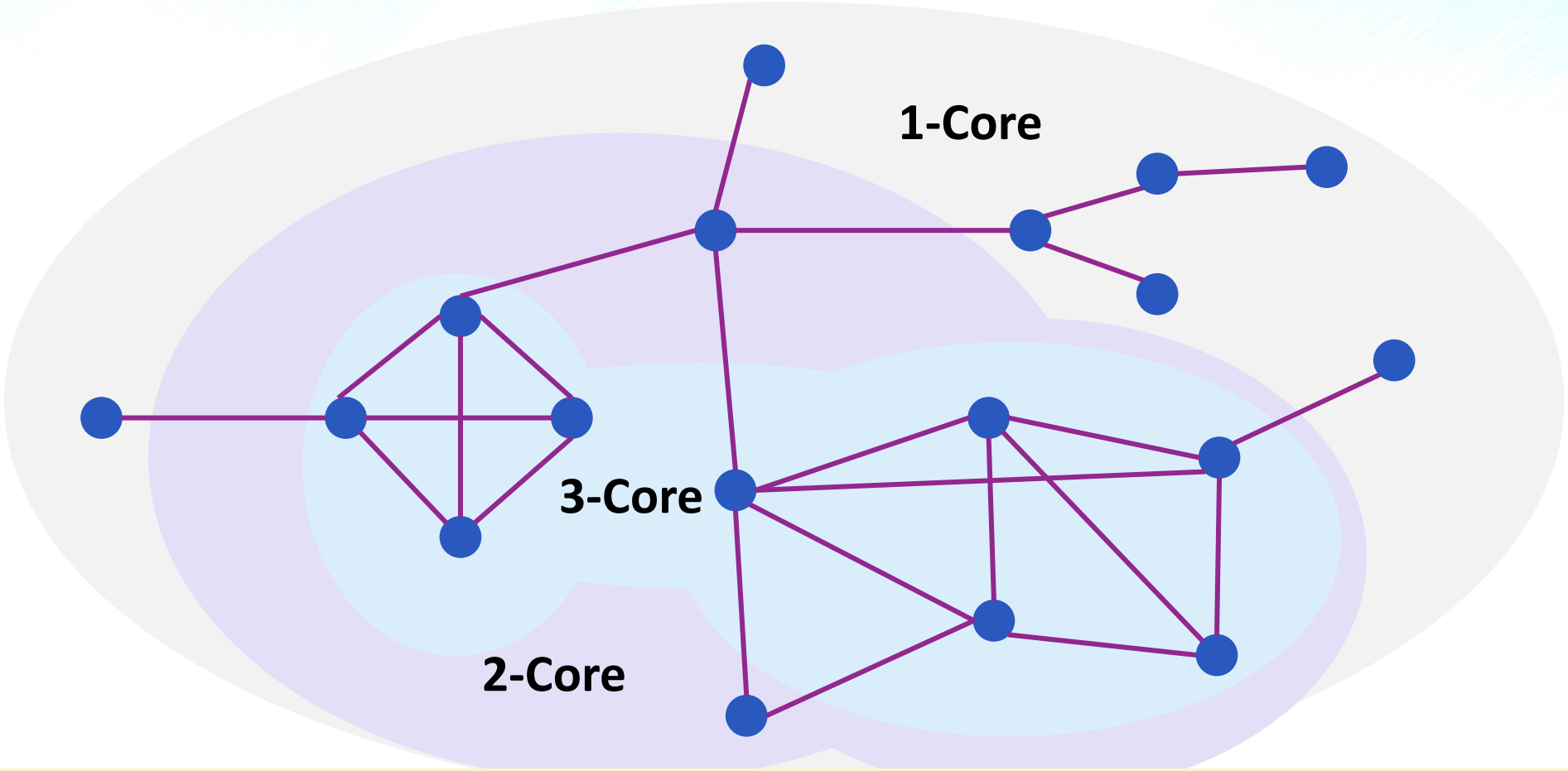
Applications and Related Problems

- Community detection in social networks
- Image processing
- Finding weak spots in data and communication networks
- IP traffic routing
- Spam and fraud detection
- Ecological network analysis
- Recommendation systems

Closely related to:

k-Core Decomposition

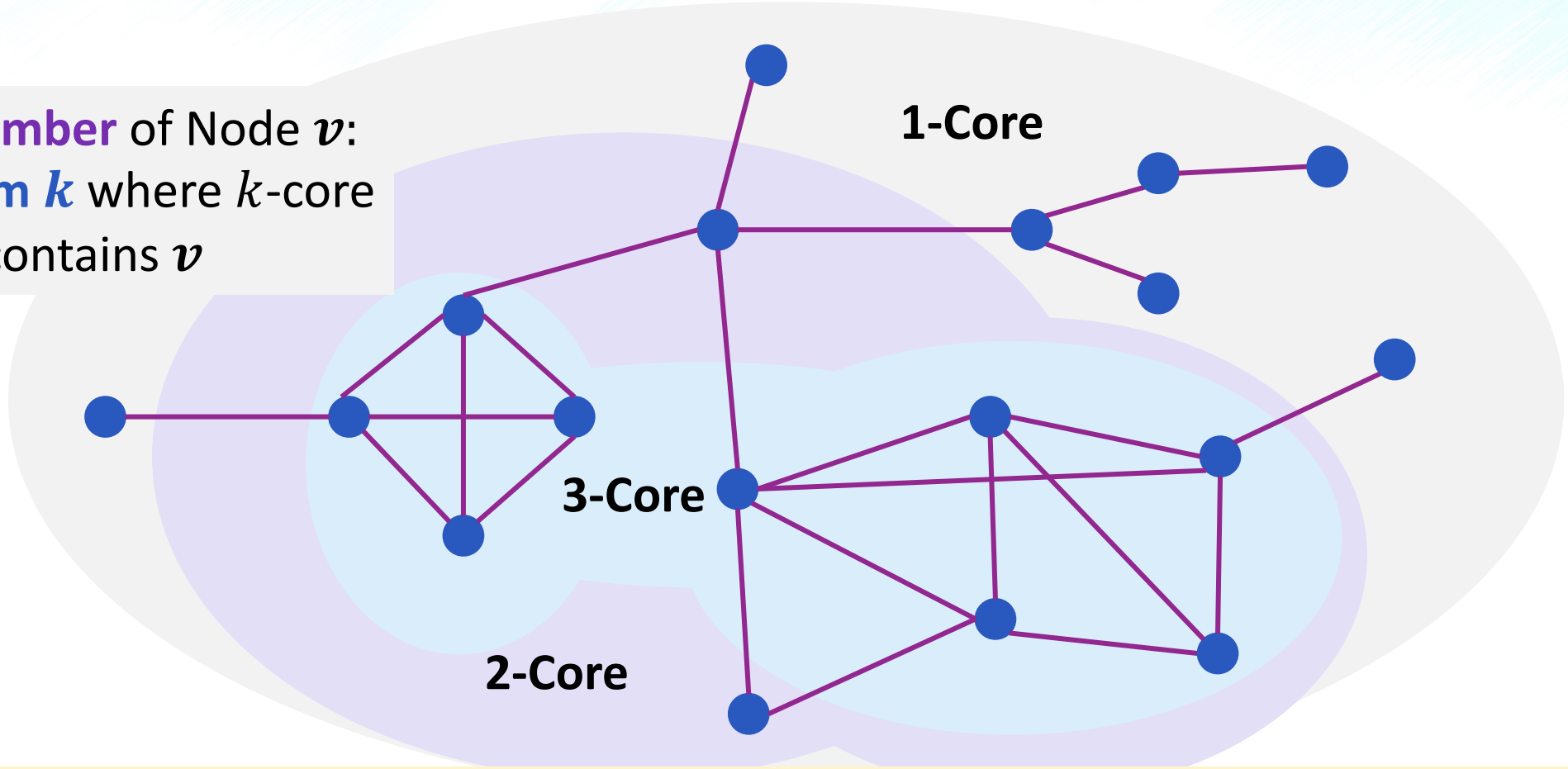
k-Core Decomposition



k-Core: Maximal induced subgraph where each vertex has degree at least k

k-Core Decomposition

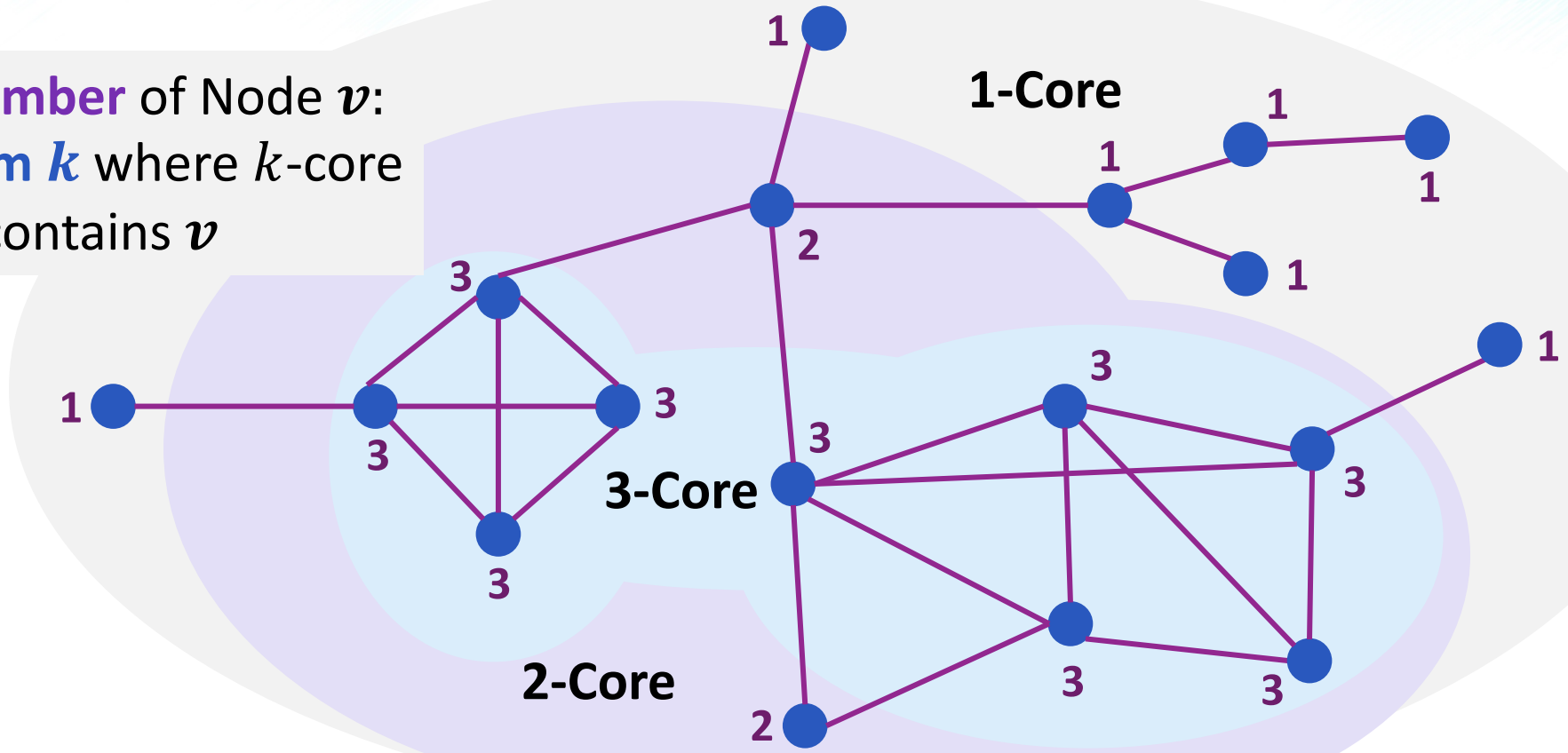
Core Number of Node v:
Maximum k where k-core contains v



k-Core: Maximal induced subgraph where each vertex has degree at least k

k-Core Decomposition

Core Number of Node v:
Maximum k where k-core contains v



k-Core: Maximal induced subgraph where each vertex has degree at least k

Densest Subgraph and k-Core Decomposition

- Let G^* be the densest subgraph, ρ^* be its density, $k_{\max}$ be the maximum core number, and $G_{k'}$ be the k'- core

Densest Subgraph and k**-Core Decomposition**

- Let G^* be the densest subgraph, ρ^* be its density, $k_{\max}$ be the maximum core number, and $G_{k'}$ be the k'- core
- Folklore:

$$\frac{k_{\max}}{2} \leq \rho^* \leq k_{\max}$$

and

$$G^* \in G_{k_{\max}/2}$$

Densest Subgraph and k-Core Decomposition

- Let G^* be the densest subgraph, ρ^* be its density, $k_{\max}$ be the maximum core number, and $G_{k'}$ be the k'- core
- Folklore:

$$\frac{k_{\max}}{2} \leq \rho^* \leq k_{\max}$$

and

$$G^* \in G_{k_{\max}/2}$$

$$G^* \in G_k \text{ for any } k \leq \lceil \rho^* \rceil$$

Parallel Framework

Find the $k_{\max}/2$-core and **prune** all vertices not in the core

Parallel Framework

Find the $k_{\max}/2$-core and **prune** all vertices not in the core

Refine the estimate of the density of the densest subgraph

Parallel Framework

Find the $k_{\max}/2$-core and **prune** all vertices not in the core

Refine the estimate of the density of the densest subgraph

Prune again

Parallel Framework

Find the $k_{\max}/2$-core and **prune** all vertices not in the core

Refine the estimate of the density of the densest subgraph

Prune again

Output **best density**

Parallel Framework

Find the $k_{\max}/2$-core and **prune** all vertices not in the core

Removes large part of graph very fast

Refine the estimate of the density of the densest subgraph

Prune again

Output **best density**

Parallel Framework

Find the $k_{\max}/2$-core and **prune** all vertices not in the core

Removes large part of graph very fast

Refine the estimate of the density of the densest subgraph

Prune again

Slower but more accurate on much smaller graph

Output **best density**

Parallel Framework

Find the $k_{\max}/2$-core and **prune** all vertices not in the core

Existing **Fast** and **parallel** approximate k-core decomposition alg

Refine the estimate of the density of the densest subgraph

Prune again

Output **best density**

Parallel Framework

Find the $k_{\max}/2$-core and **prune** all vertices not in the core

Existing **Fast** and **parallel** approximate k-core decomposition alg

Refine the estimate of the density of the densest subgraph

Prune again

Fast and **almost optimally parallel** approximate MWU densest subgraph alg

Output **best density**

Parallel Framework

Find the $k_{\max}/2$-core and **prune** all vertices not in the core

Existing **Fast** and **parallel** approximate k-core decomposition alg

[Dhulipala-Blelloch-Shun '17]
[Liu et al. '22]

Refine the estimate of the density of the densest subgraph

Fang et al. VLDB '19
Xu et al. SIGMOD '23

Prune again

Fast and **almost optimally parallel** approximate MWU densest subgraph alg

Output **best density**

Parallel Framework

Refine the estimate of the density of the densest subgraph

Prune again

Fast and **almost optimally parallel** approximate MWU densest subgraph alg

Output **best density**

2-Approx Peeling Algorithm [Charikar ‘00]

2-Approx Peeling Algorithm [Charikar ‘00]

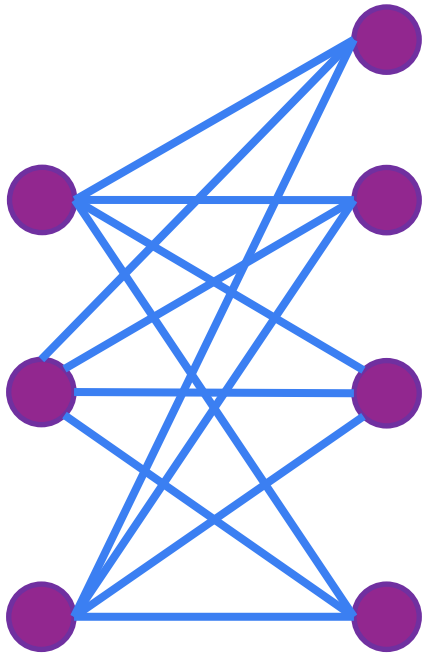
- Keep peeling vertex with **lowest degree**

2-Approx Peeling Algorithm [Charikar ‘00]

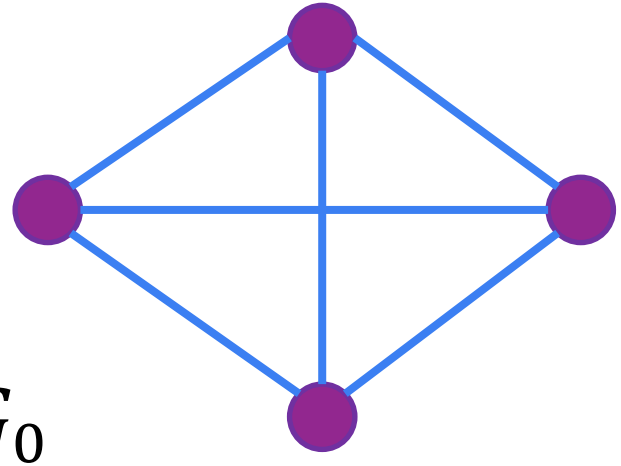
- Keep peeling vertex with **lowest degree**
- Return **largest density** anytime during the peeling

2-Approx Peeling Algorithm [Charikar '00]

- Keep peeling vertex with **lowest degree**
- Return **largest density** anytime during the peeling



$\rho(G_0) \approx 1.6$

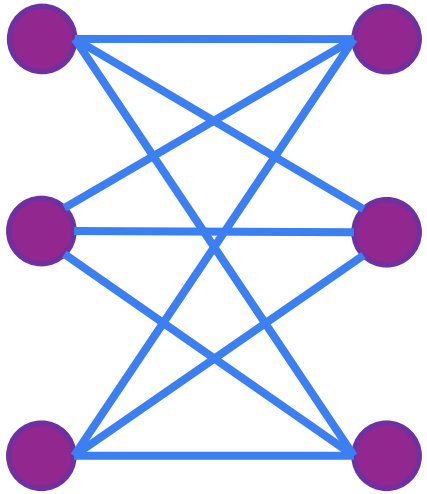


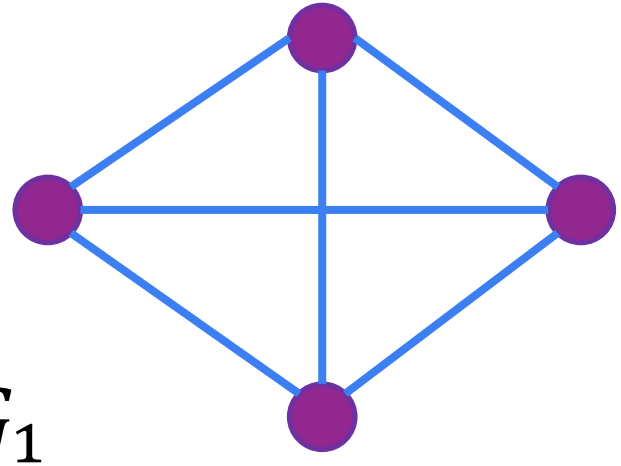
G_0

2-Approx Peeling Algorithm [Charikar '00]

- Keep peeling vertex with **lowest degree**
- Return **largest density** anytime during the peeling

$$\rho(G_0) \approx 1.6 \qquad \rho(G_1) = 1.5$$



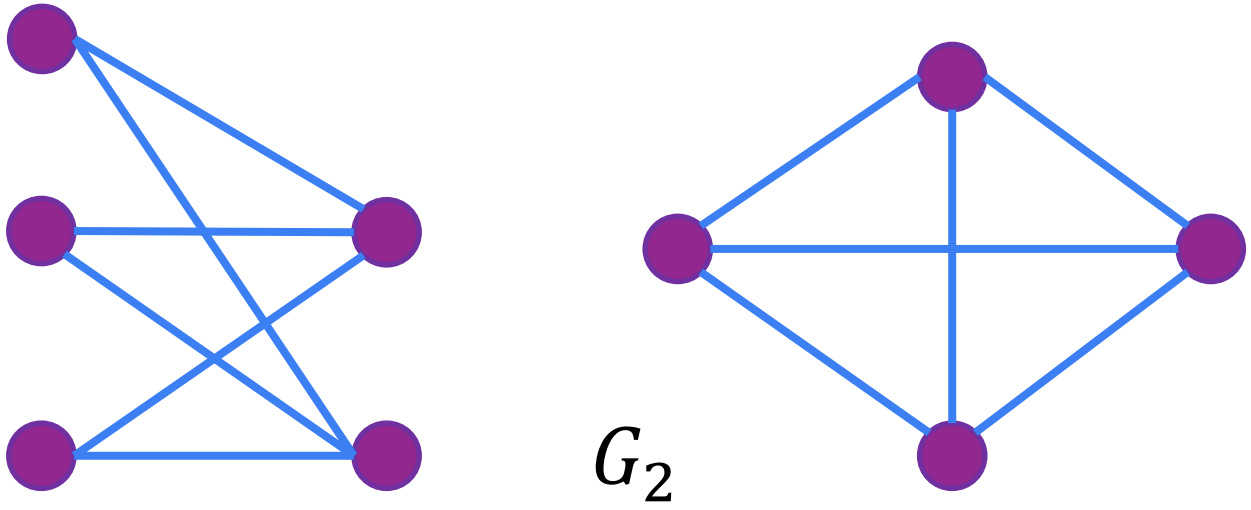


G_1

2-Approx Peeling Algorithm [Charikar '00]

- Keep peeling vertex with **lowest degree**
- Return **largest density** anytime during the peeling

$$\rho(G_0) \approx 1.6 \qquad \rho(G_1) = 1.5 \qquad \rho(G_2) \approx 1.3$$

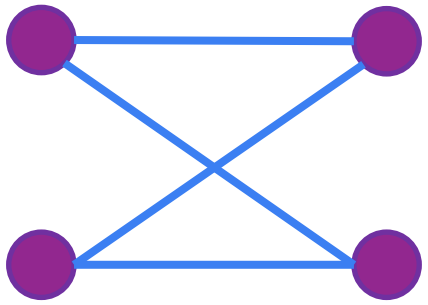


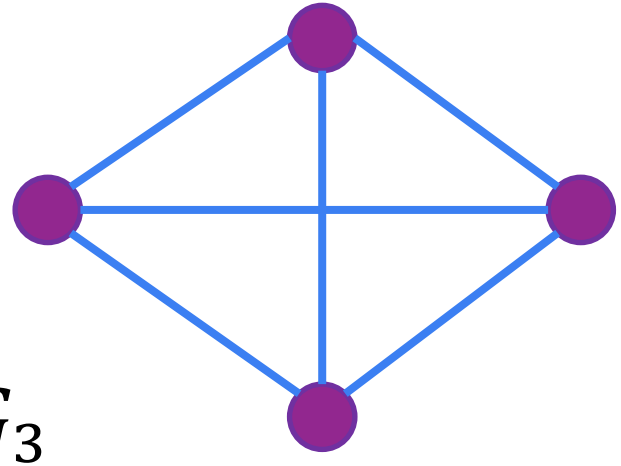
G_2

2-Approx Peeling Algorithm [Charikar '00]

- Keep peeling vertex with **lowest degree**
- Return **largest density** anytime during the peeling

$$\rho(G_0) \approx 1.6 \qquad \rho(G_1) = 1.5 \qquad \rho(G_2) \approx 1.3$$





G_3

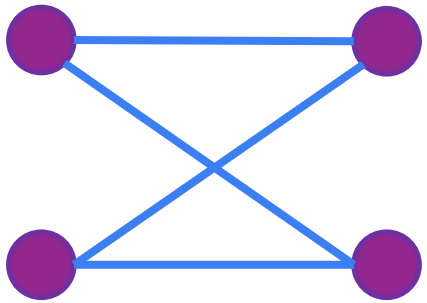
$$\rho(G_3) \approx 1.25$$

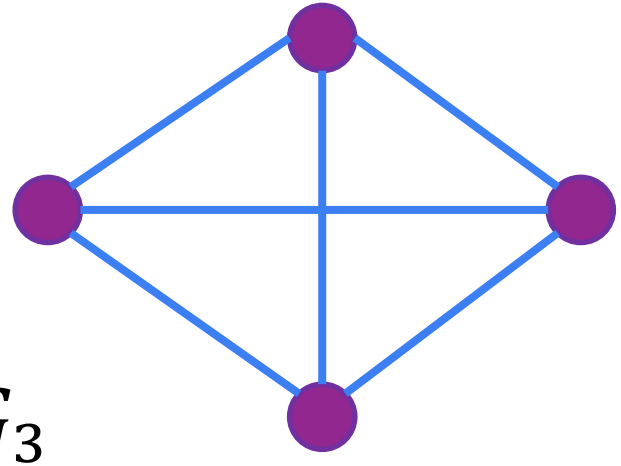
...

2-Approx Peeling Algorithm [Charikar '00]

- Keep peeling vertex with **lowest degree**
- Return **largest density** anytime during the peeling

$\rho(G_0) \approx 1.6$ $\rho(G_1) = 1.5$ $\rho(G_2) \approx 1.3$





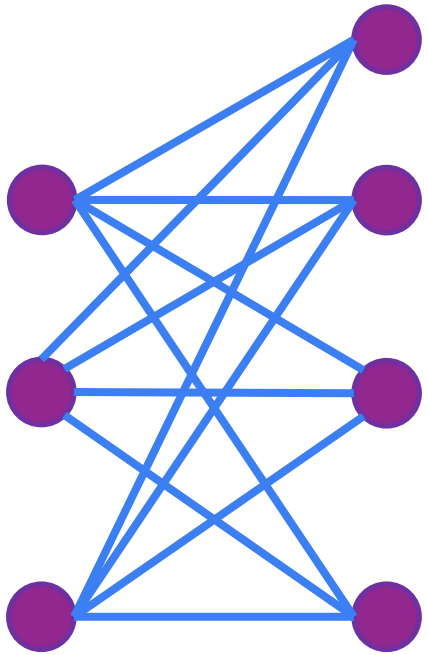
G_3

$\rho(G_3) \approx 1.25$

...

2-Approx Peeling Algorithm [Charikar '00]

- Keep peeling vertex with **lowest degree**
- Return **largest density** anytime during the peeling



$$\rho(G_1) = 1.5$$

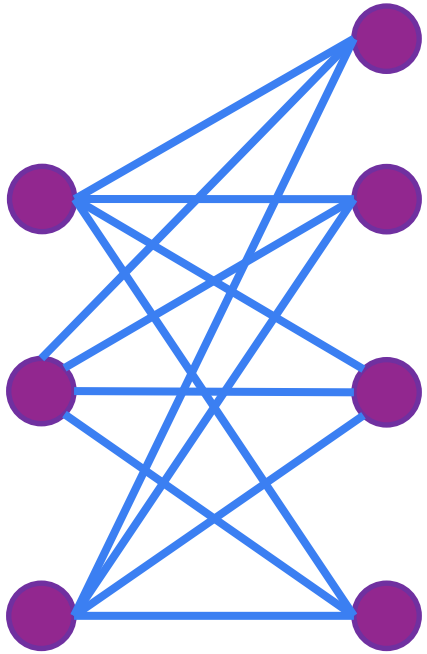
Densest subgraph: $\rho^* \approx 1.7$

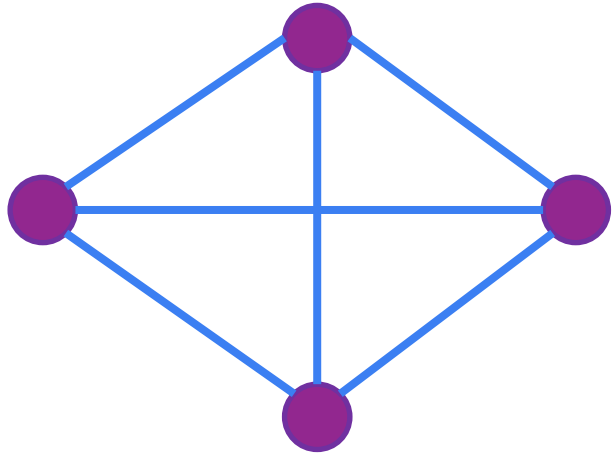
$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**

$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees

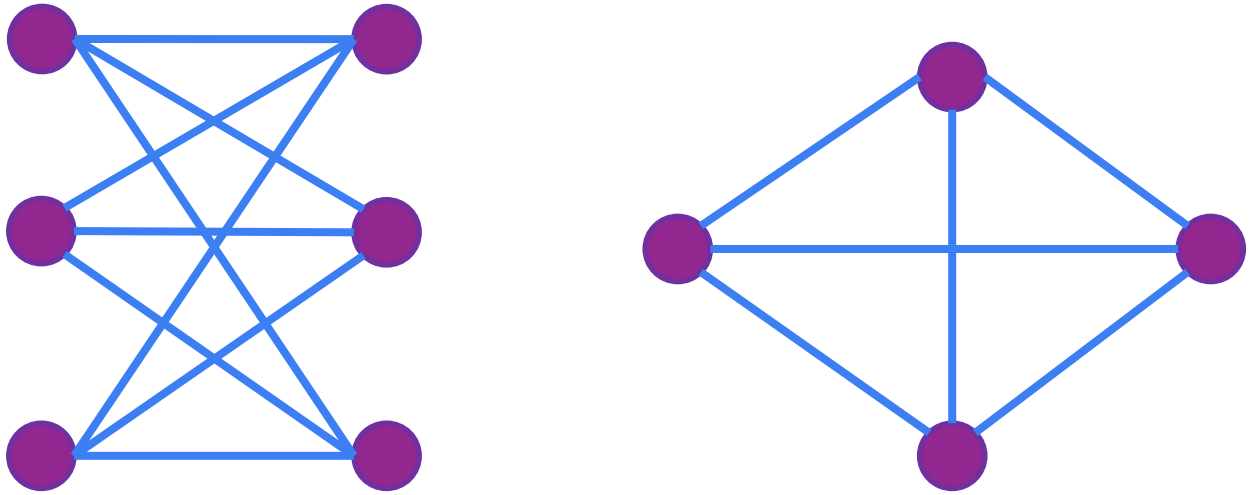




$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

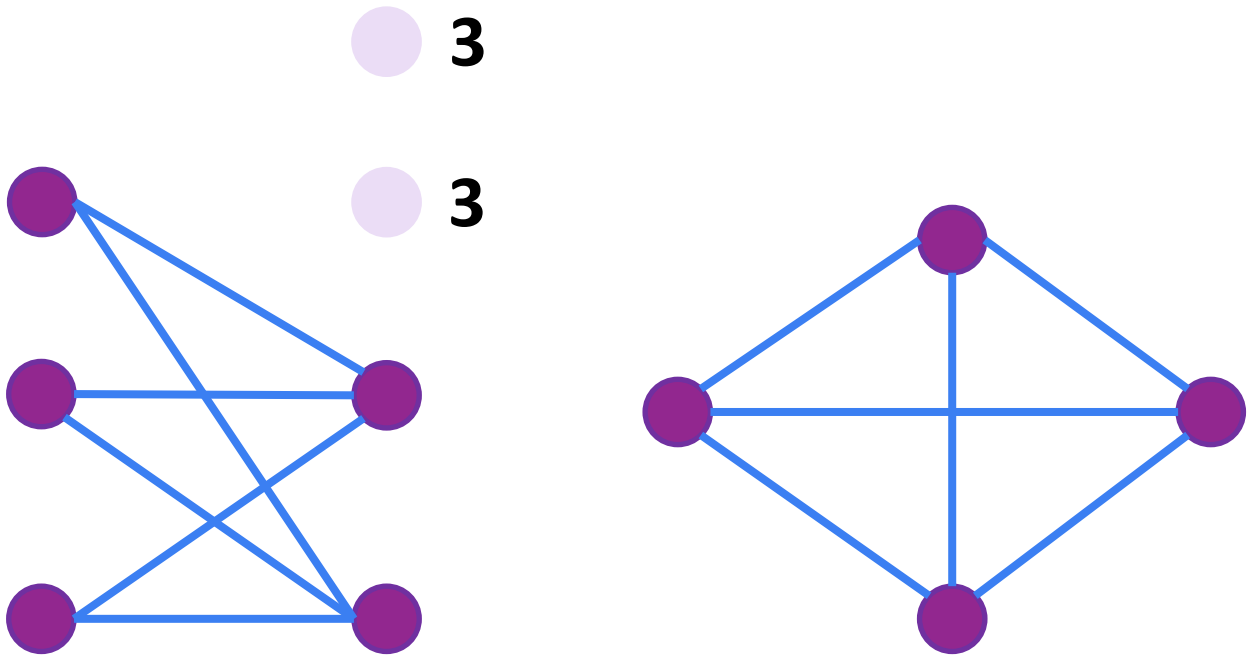
- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees

3



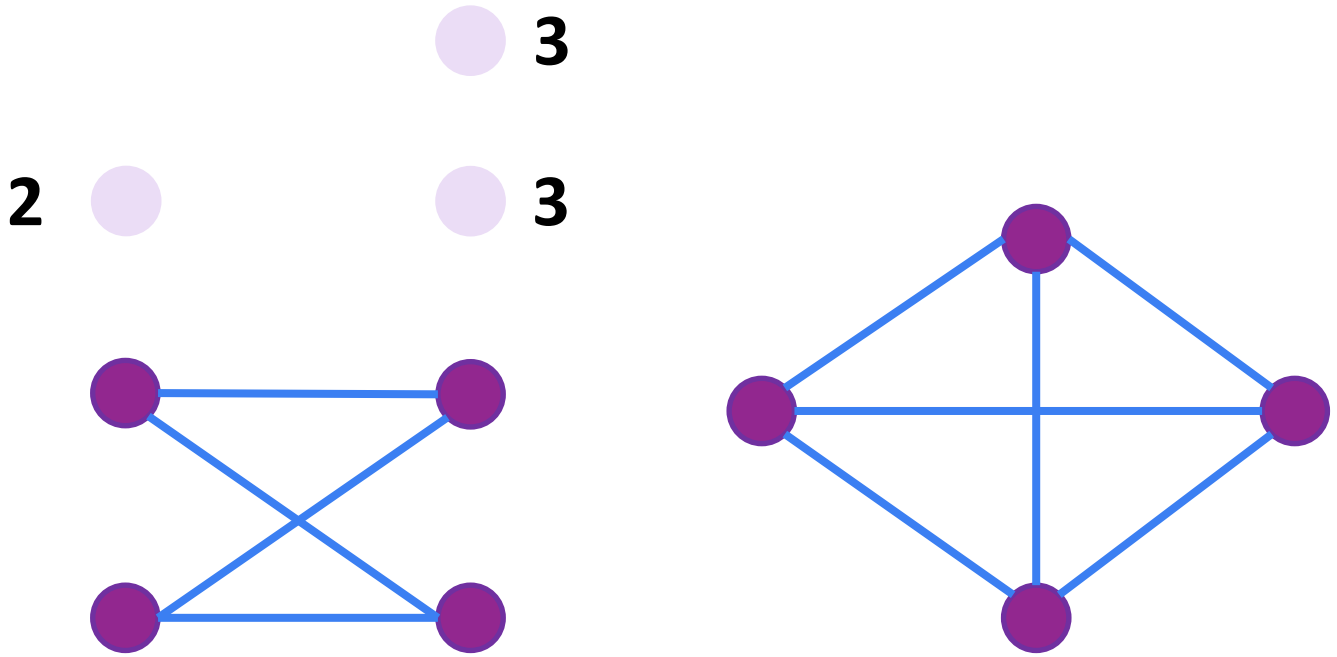
$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees



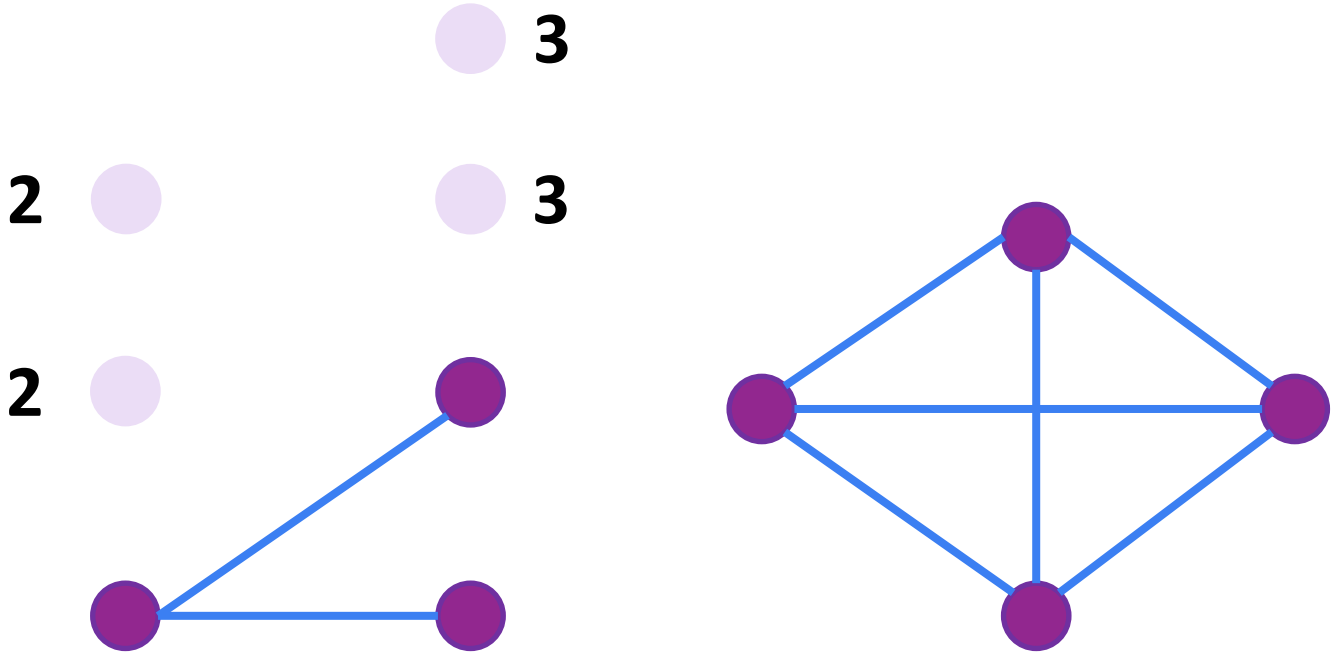
$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees



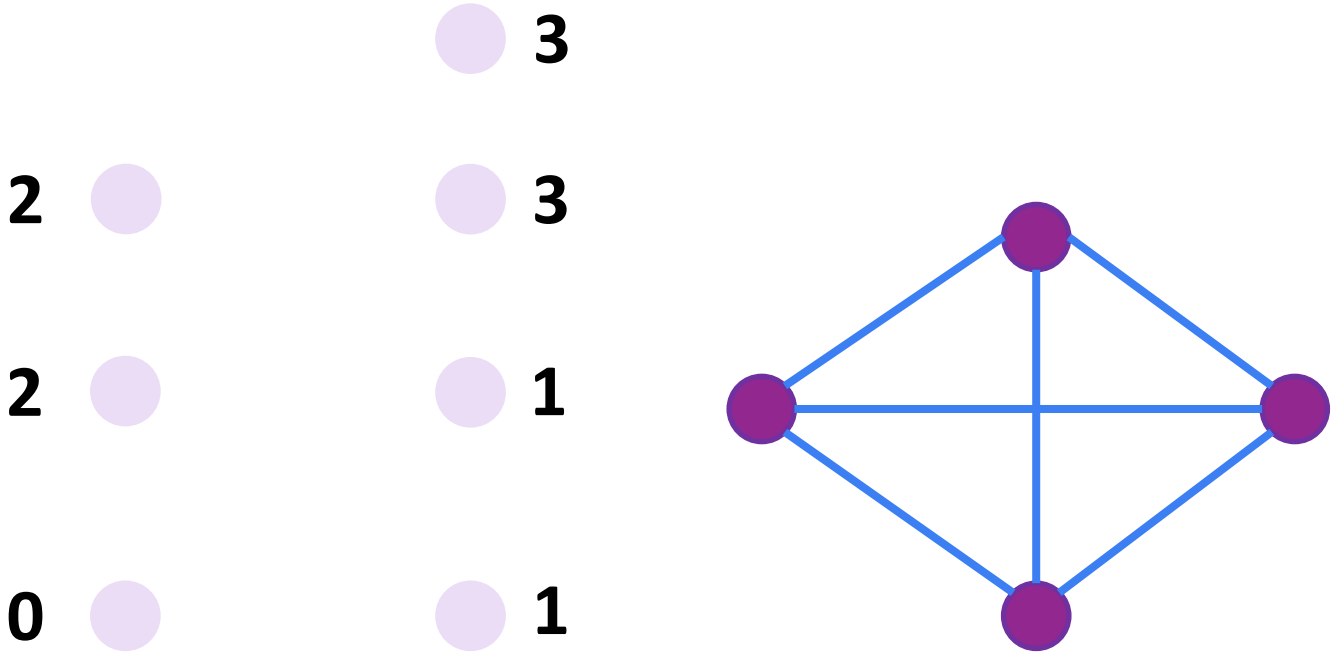
$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees



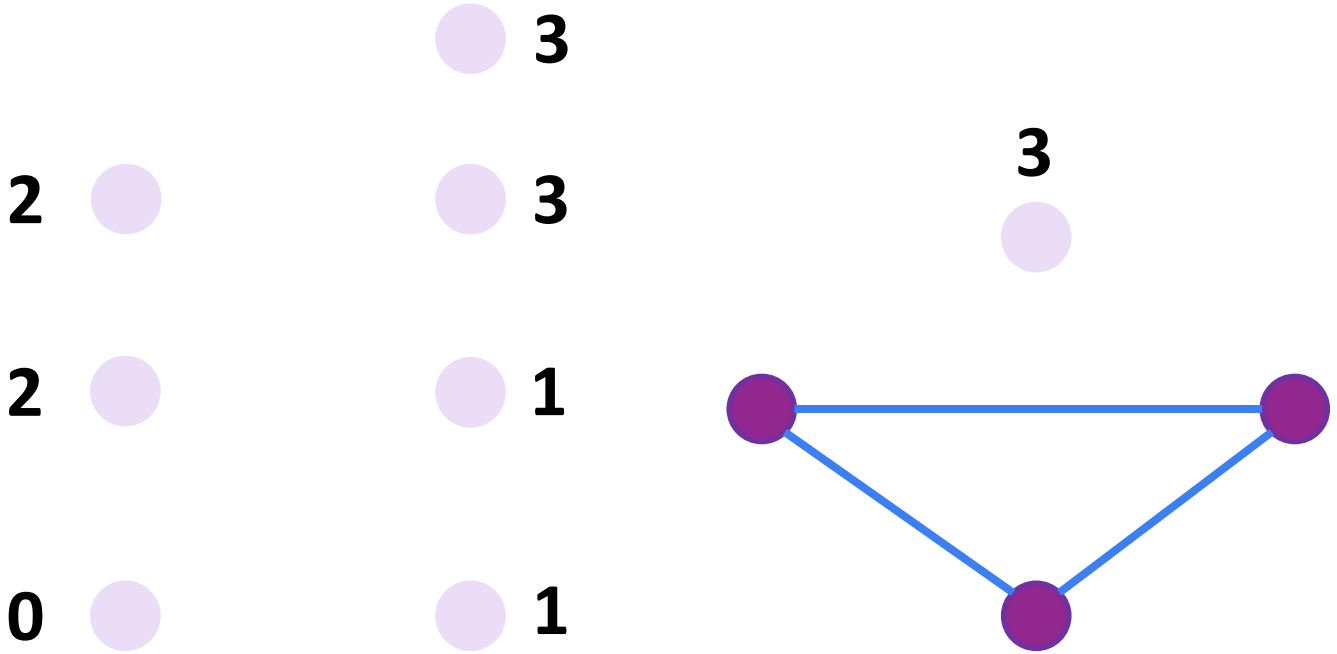
$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees



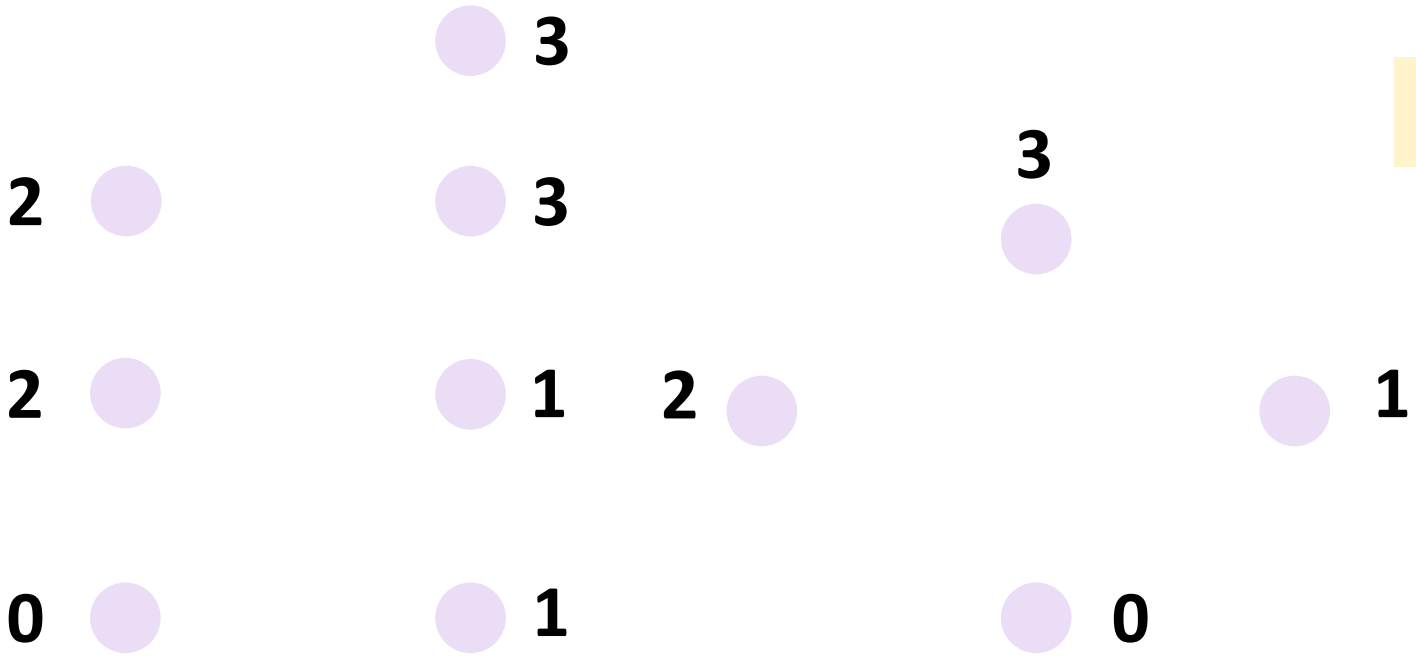
$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees



$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

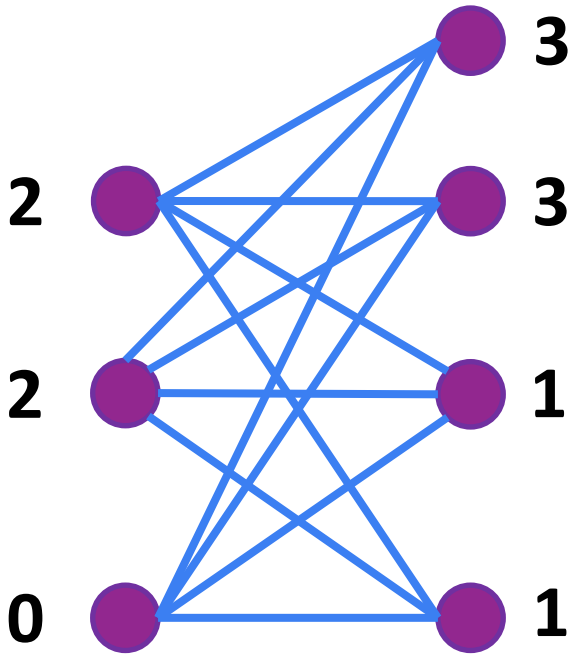
- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees

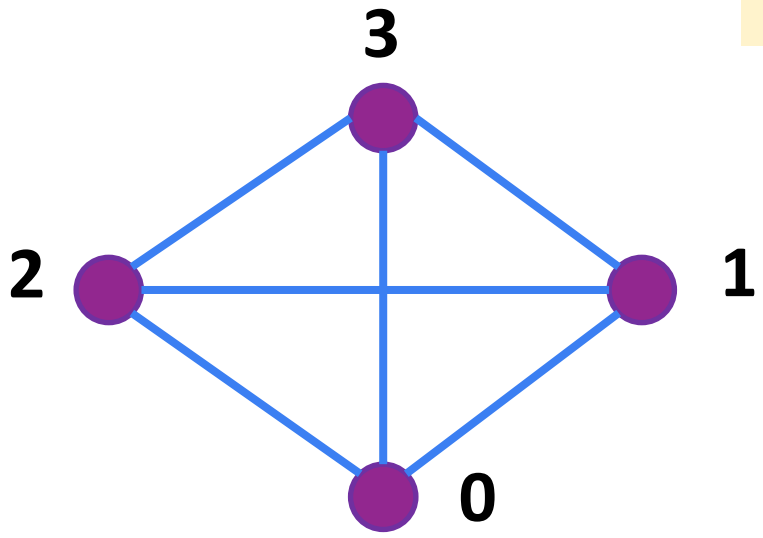


Max Density Round 1: 1.6

$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees

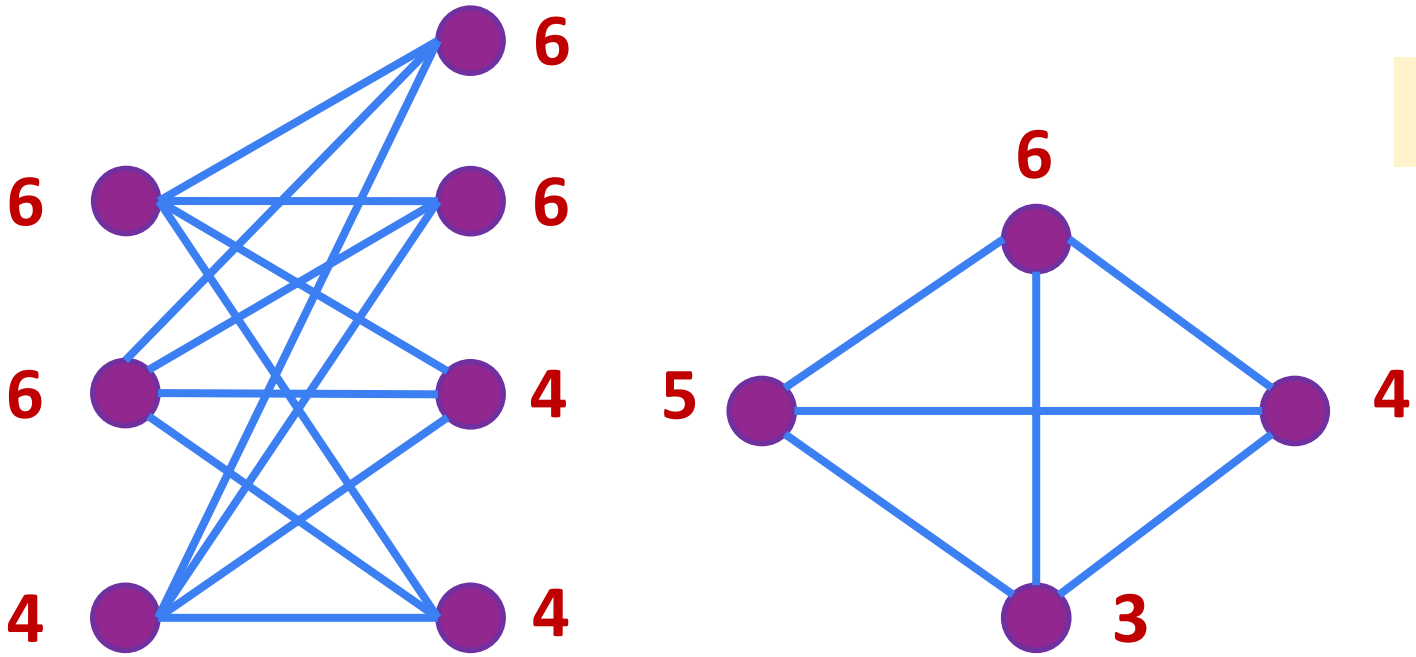




Max Density Round 1: 1.6

$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees

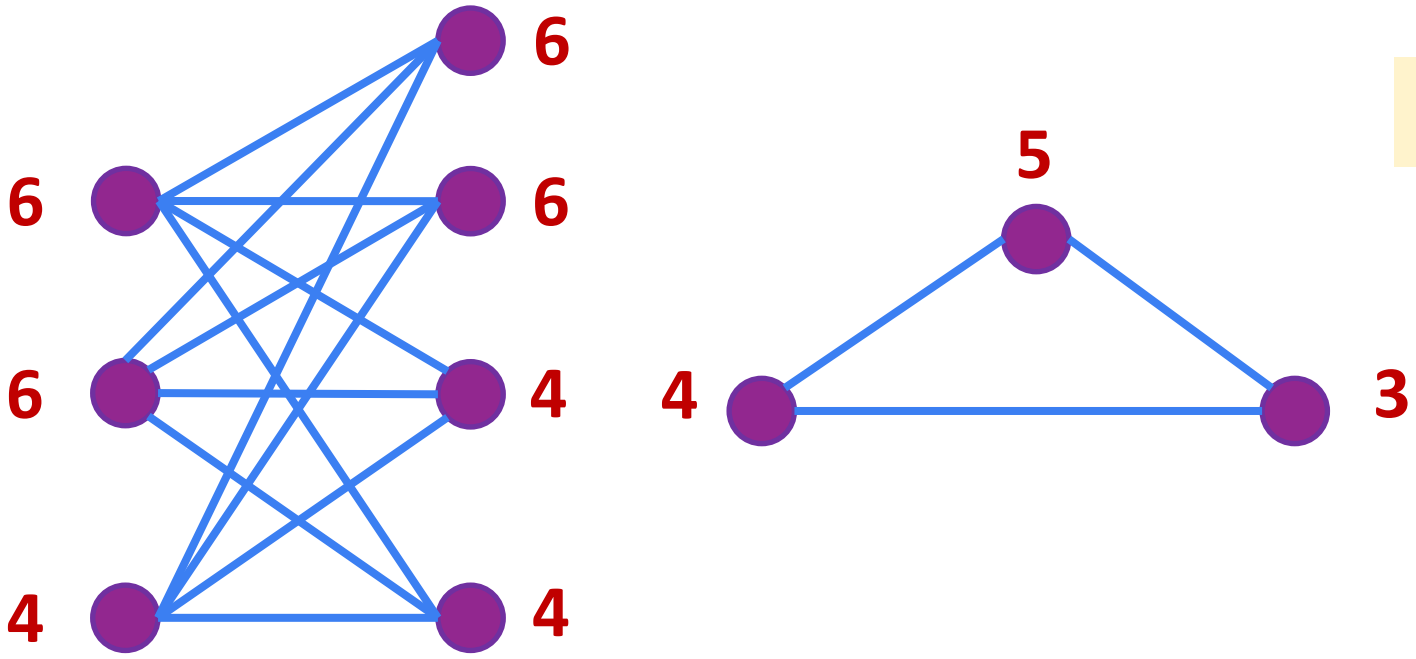


Max Density Round 1: 1.6

Compute Load + Degree

$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

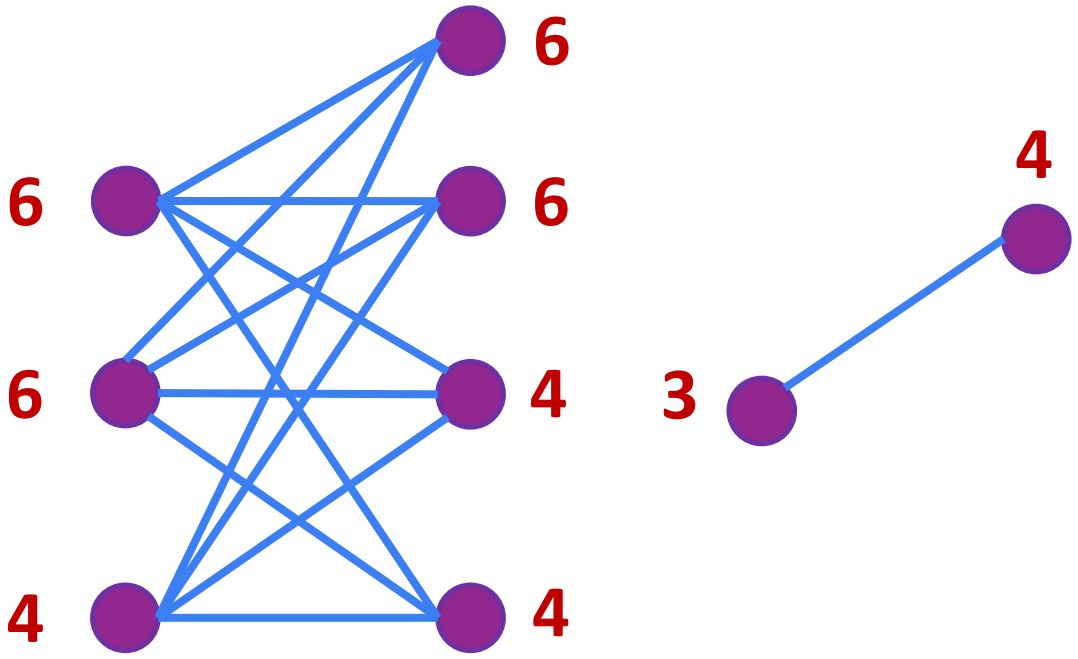
- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees



Max Density Round 1: 1.6

$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

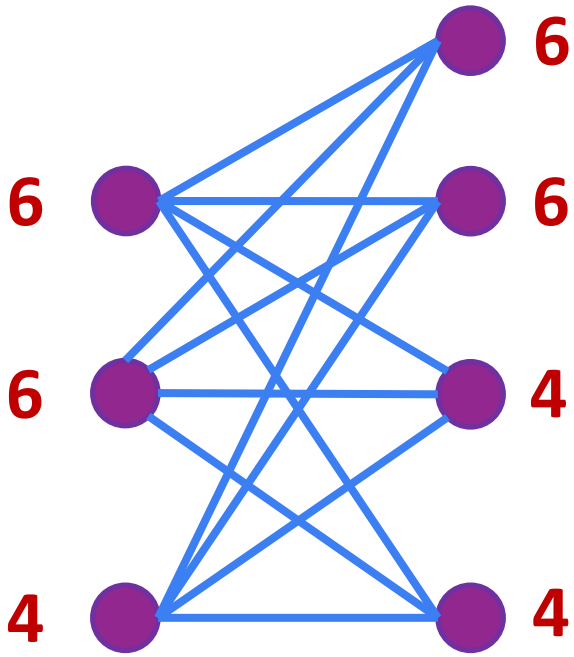
- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees



Max Density Round 1: 1.6

$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

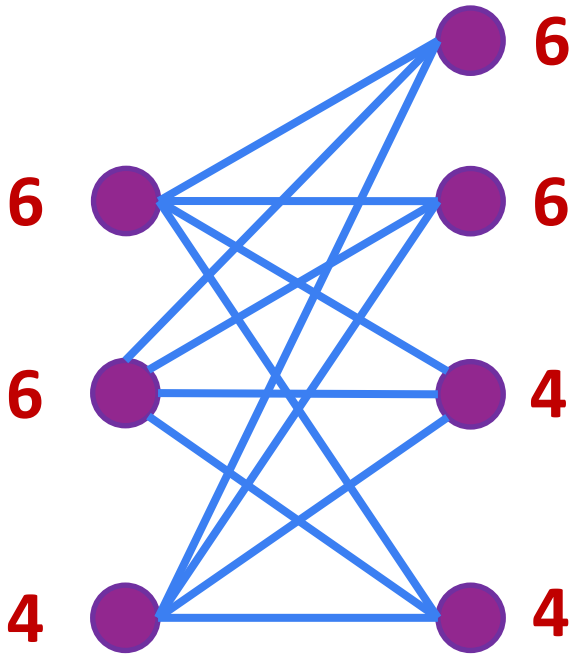
- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees



Max Density Round 1: 1.6

$(1+\varepsilon)$-Approx Greedy++ [Boob et al. '20]

- Run peeling for multiple **rounds**
  - Each round: peel node with smallest **degree + load**
  - **Load: sum** of previous peeled degrees



$(1+\epsilon)$-approx.
in $\tilde{O}\left(\frac{\Delta}{\rho^* \epsilon^2}\right)$ rounds
[Chekuri-Quanrud-Torres '22]

Max Density Round 1: **1.6**

Max Density Round 2: **1.7**

$(1+\varepsilon)$-Approx GreedySorting++

For $t = 1 \dots T$
  while G is not empty
    $v_{min} \leftarrow argmin_v \deg(v) + \ell(v)$
    $\ell(v_{min}) \leftarrow \ell(v_{min}) + \deg(v_{min})$
    $G \leftarrow G \setminus v_{min}$
    $\rho^* \leftarrow \max(\rho^*, \rho(G))$
  reset G
return ρ^*

Greedy++

$(1+\varepsilon)$-Approx GreedySorting++

Highly sequential

For $t = 1 \dots T$
  while G is not empty
    $v_{min} \leftarrow argmin_v \mathbf{deg}(\boldsymbol{v}) + \ell(v)$
    $\ell(v_{min}) \leftarrow \ell(v_{min}) + \deg(v_{min})$
    $G \leftarrow G \setminus v_{min}$
    $\rho^* \leftarrow \max(\rho^*, \rho(G))$
  reset G
return ρ^*

Greedy++

$(1+\varepsilon)$-Approx GreedySorting++

Highly sequential

For $t = 1 \dots T$
  while G is not empty
    $v_{min} \leftarrow argmin_v \mathbf{deg}(v) + \ell(v)$
    $\ell(v_{min}) \leftarrow \ell(v_{min}) + \deg(v_{min})$
    $G \leftarrow G \setminus v_{min}$
    $\rho^* \leftarrow \max(\rho^*, \rho(G))$
  reset G
return ρ^*

Greedy++

For $t = 1 \dots T$
  while G is not empty
    $\boldsymbol{v_{min} \leftarrow argmin_v \ell(v)}$
    $\ell(v_{min}) \leftarrow \ell(v_{min}) + \deg(v_{min})$
    $G \leftarrow G \setminus v_{min}$
    $\rho^* \leftarrow \max(\rho^*, \rho(G))$
  reset G
return ρ^*

GreedySorting++

$(1+\varepsilon)$-Approx GreedySorting++

GreedySorting++
very parallelizable

For $t = 1 \dots T$
  while G is not empty
    $v_{min} \leftarrow \boldsymbol{argmin}_v \ell(v)$
    $\ell(v_{min}) \leftarrow \ell(v_{min}) + \deg(v_{min})$
    $G \leftarrow G \setminus v_{min}$
    $\rho^* \leftarrow \max(\rho^*, \rho(G))$
  reset G
return ρ^*

GreedySorting++

$(1+\varepsilon)$-Approx GreedySorting++

GreedySorting++
very parallelizable

$(1+\epsilon)$-approx.
in $\tilde{\boldsymbol{O}}\left(\frac{\Delta}{\rho^* \epsilon^2}\right)$ rounds
[Chekuri-Quanrud-Torres '22]

For $t = 1 \dots T$
  while G is not empty
    $v_{min} \leftarrow \boldsymbol{argmin}_v \ell(v)$
    $\ell(v_{min}) \leftarrow \ell(v_{min}) + \deg(v_{min})$
    $G \leftarrow G \setminus v_{min}$
    $\rho^* \leftarrow \max(\rho^*, \rho(G))$
  reset G
return ρ^*

GreedySorting++

$(1+\varepsilon)$-Approx GreedySorting++ Complexity

- We analyze algorithms in **work-depth** model
  - **Work** total time of performing all operations executed by algorithm
  - **Depth** longest chain of sequential dependencies in algorithm

$(1+\varepsilon)$-Approx GreedySorting++ Complexity

- We analyze algorithms in **work-depth** model
  - **Work** total time of performing all operations executed by algorithm
  - **Depth** longest chain of sequential dependencies in algorithm

Greedy++ has $\Omega(n)$ **depth**
GreedySorting++ has $\boldsymbol{O(\log n)}$ **depth**

Experimental Setup

https://github.com/PattaraS/gbbs/tree/ALENEX

- c2-standard-60 Google Cloud instances
  - 30 cores with two-way hyper-threading, 236 GB RAM
- m1-megamem-96 Google Cloud instances
  - 48 cores with two-way hyper-threading, and 1433.6 GB RAM
- GBBS [DBS17] and Parlay [BAD20] libraries
  - GBBS: https://github.com/ParAlg/gbbs
  - Parlay: https://github.com/ParAlg/parlaylib
- Terminate experiments that take longer than 1 hour wall-clock

Benchmarks

https://github.com/PattaraS/gbbs/tree/ALENEX

- $(1+\varepsilon)$-approximate sequential algorithms
  - **CoreExact** and **CoreApp** [Fang et al. '19]
  - **Greedy++** [Boob et al. '20]
  - **cCoreG++** [Xu et al. '23]
- Parallel $(1+\varepsilon)$-approximate algorithms
  - **FISTA** [Harb-Quanrud-Chekuri '22]
  - **Frank-Wolf** [Danisch-Chan-Sozio '17]
  - **MWU** [Bahmani-Goel-Munagala '14]
- Parallel 2-approximation algorithms based on k-core
  - **PKMC** [Luo et al. '23]
  - **Julienne** [Dhulipala-Blelloch-Shun '18]

Dataset Information

Graph Dataset	Original Graph Num. Vertices	Original Graph Num. Edges	k_{max}	$core(G, \lceil \frac{k_{max}}{2} \rceil)$ Num. Vertices	$core(G, \lceil \frac{k_{max}}{2} \rceil)$ Num. Edges	Vertex Ratio	Edge Ratio
closecliques	3,230	95,400	59	3,230	95,400	1.000	1.000
cahepth	9,877	25,973	31	96	2,306	0.0097	0.088
ascaida	26,475	106,762	22	208	6,244	0.007	0.048
hepph	28,094	3,148,447	410	6,304	1,562,818	0.224	0.494
dblp	317,080	1,049,866	101	280	13,609	0.001	0.010
brain	784,262	267,844,669	1,200	187,494	137,354,946	0.239	0.512
wiki	1,094,018	2,787,967	124	3,807	344,553	0.034	0.090
youtube	1,138,499	2,990,443	51	12,836	439,678	0.011	0.110
stackoverflow	2,584,164	28,183,518	163	41,651	5,709,796	0.016	0.187
livejournal	4,846,609	42,851,237	329	6,090	1,054,941	0.001	0.022
orkut	3,072,441	117,185,083	253	71,507	13,469,722	0.023	0.113
twitter	41,652,230	1,202,513,046	2,484	24,480	36,136,023	0.001	0.029
friendster	65,608,366	1,806,067,135	304	1,474,236	271,902,207	0.022	0.146
clueweb	978,408,098	37,372,179,311	4,244	91,874	132,549,663	9.39e-05	0.003
hyperlink2012	3,563,602,789	112,920,331,616	10,565	250,477	1,046,929,322	7.02e-05	0.009

Dataset Information

Graph Dataset	Original Graph Num. Vertices	Original Graph Num. Edges	k_{max}	$core(G, \lceil \frac{k_{max}}{2} \rceil)$ Num. Vertices	$core(G, \lceil \frac{k_{max}}{2} \rceil)$ Num. Edges	Vertex Ratio	Edge Ratio
closecliques	3,230	95,400	59	3,230	95,400	1.000	1.000
cahepth	9,877	25,973	31	96	2,306	0.0097	0.088
ascaida	26,475	106,762	22	208	6,244	0.007	0.048
hepph	28,094	3,148,447	410	6,304	1,562,818	0.224	0.494
dblp	317,080	1,049,866	101	280	13,609	0.001	0.010
brain	784,262	267,844,669	1,200	187,494	137,354,946	0.239	0.512
wiki	1,094,018	2,787,967	124	3,807	344,553	0.034	0.090
youtube	1,138,499	2,990,443	51	12,836	439,678	0.011	0.110
stackoverflow	2,584,164	28,183,518	163	41,651	5,709,796	0.016	0.187
livejournal	4,846,609	42,851,237	329	6,090	1,054,941	0.001	0.022
orkut	3,072,441	117,185,083	253	71,507	13,469,722	0.023	0.113
twitter	41,652,230	1,202,513,046	2,484	24,480	36,136,023	0.001	0.029
friendster	65,608,366	1,806,067,135	304	1,474,236	271,902,207	0.022	0.146
clueweb	978,408,098	37,372,179,311	4,244	91,874	132,549,663	9.39e-05	0.003
hyperlink2012	3,563,602,789	112,920,331,616	10,565	250,477	1,046,929,322	7.02e-05	0.009

Dataset Information

Graph Dataset	Original Graph Num. Vertices	Original Graph Num. Edges	k_{max}	$core(G, \lceil \frac{k_{max}}{2} \rceil)$ Num. Vertices	$core(G, \lceil \frac{k_{max}}{2} \rceil)$ Num. Edges	Vertex Ratio	Edge Ratio
closecliques	3,230	95,400	59	3,230	95,400	1.000	1.000
cahepth	9,877	25,973	31	96	2,306	0.0097	0.088
ascaida	26,475	106,762	22	208	6,244	0.007	0.048
hepph	28,094	3,148,447	410	6,304	1,562,818	0.224	0.494
dblp	317,080	1,049,866	101	280	13,609	0.001	0.010
brain	784,262	267,844,669	1,200	187,494	137,354,946	0.239	0.512
wiki	1,094,018	2,787,967	124	3,807	344,553	0.034	0.090
youtube	1,138,499	2,990,443	51	12,836	439,678	0.011	0.110
stackoverflow	2,584,164	28,183,518	163	41,651	5,709,796	0.016	0.187
livejournal	4,846,609	42,851,237	329	6,090	1,054,941	0.001	0.022
orkut	3,072,441	117,185,083	253	71,507	13,469,722	0.023	0.113
twitter	41,652,230	1,202,513,046	2,484	24,480	36,136,023	0.001	0.029
friendster	65,608,366	1,806,067,135	304	1,474,236	271,902,207	0.022	0.146
clueweb	978,408,098	37,372,179,311	4,244	91,874	132,549,663	9.39e-05	0.003
hyperlink2012	3,563,602,789	112,920,331,616	10,565	250,477	1,046,929,322	7.02e-05	0.009

Most graphs have **less than 15%** of edges in $k_{\max}/2$ core!

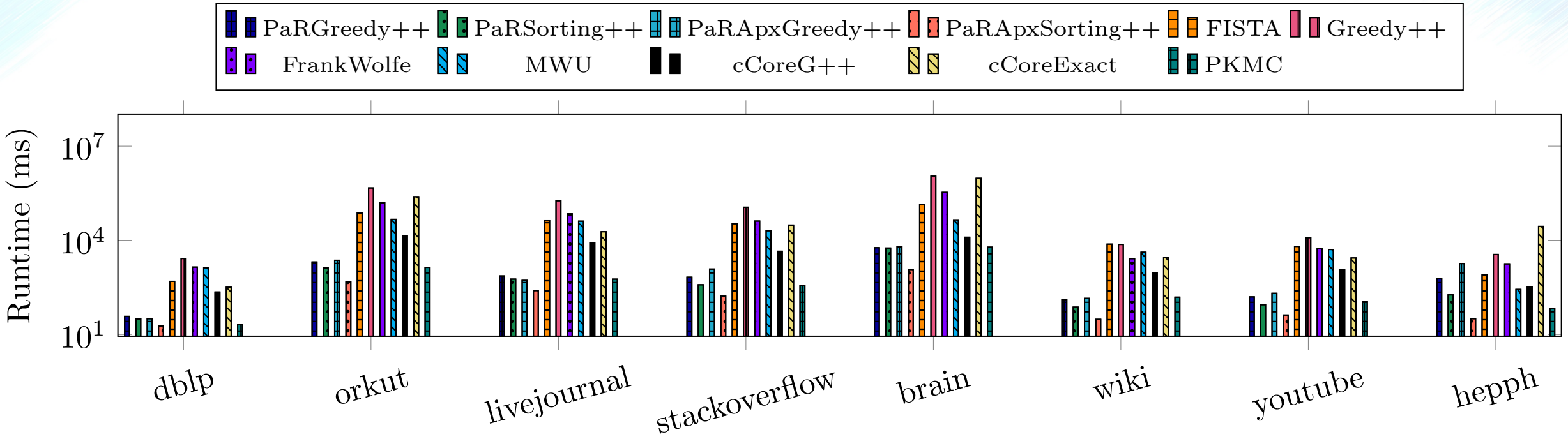
Dataset Information

Core of max found density

Graph Dataset	k_{max}	$core(G, \lceil \frac{k_{max}}{2} \rceil)$ Num. Vertices	$core(G, \lceil \frac{k_{max}}{2} \rceil)$ Num. Edges	$\tilde{\rho}$	$core(G, \lceil \tilde{\rho} \rceil)$ Num. Vertices	$core(G, \lceil \tilde{\rho} \rceil)$ Num. Edges	Vertex Ratio	Edge Ratio
closecliques	59	3,230	95,400	29.55665	3,230	95,400	1.000	1.000
cahepth	31	96	2,306	15.5	96	2,306	1.000	1.000
ascaida	22	208	6,244	17.53409	90	1,578	0.432	0.259
hepph	410	6304	1,562,818	265.969	3,786	988,734	0.600	0.633
dblp	101	280	13,609	56.56522	280	13,609	1.000	1.000
brain	1,200	187,494	137,354,946	1,057.458	8,993	9,509,717	0.048	0.069
wiki	124	3,807	344,553	108.5877	1,379	149,739	0.362	0.434
youtube	51	12,836	439,678	45.59877	2,269	103,342	0.176	0.233
stackoverflow	163	41,651	5,709,796	181.5867	5,877	1,067,149	0.141	0.187
livejournal	329	6,090	1,054,941	229.8459	3,393	610,864	0.557	0.579
orkut	253	71,507	13,469,722	227.874	26,670	6,077,055	0.372	0.451
twitter	2,484	24,480	36,136,023	1643.301	11,619	17,996,107	0.474	0.498
friendster	304	1,474,236	271,902,207	273.5187	49,370	1,353,583	0.033	0.005
clueweb	4,244	91,874	132,549,663	2122.5	91,874	132,549,663	1.000	1.000
hyperlink2012	10,565	250,477	1,046,929,322	6,496.649	154,410	754,065,464	0.616	0.720

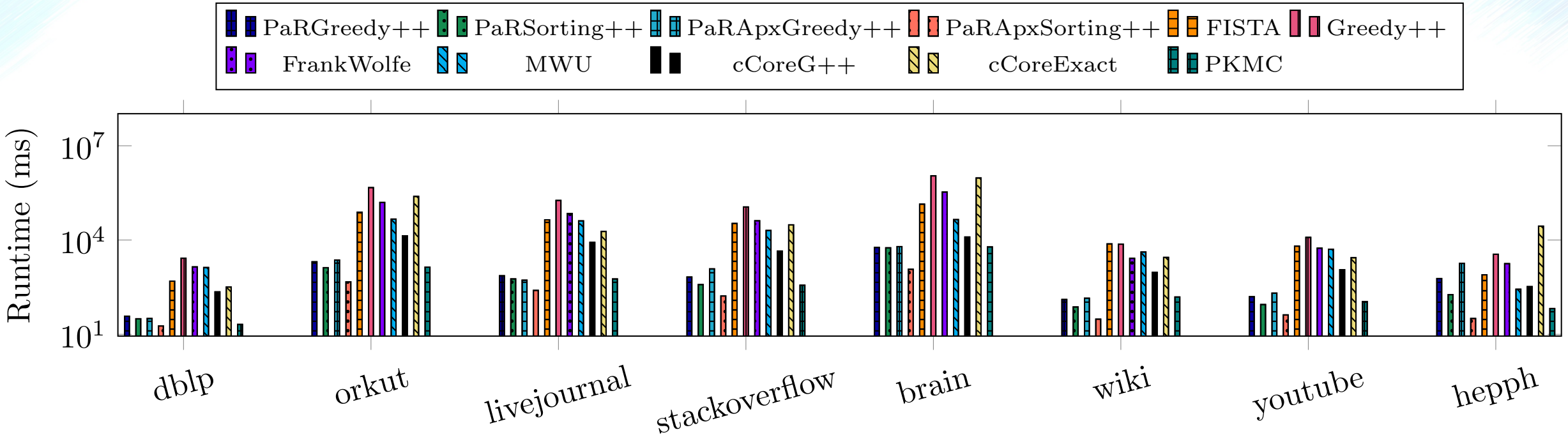
Many graphs have **less than 1/2** of edges in core of max approx. density core than $k_{\max}/2$ core!

Runtime of All Algorithms on Smaller Graphs



Ran sorting-based 100 iterations and peeling-based 20 iterations

Runtime of All Algorithms on Smaller Graphs

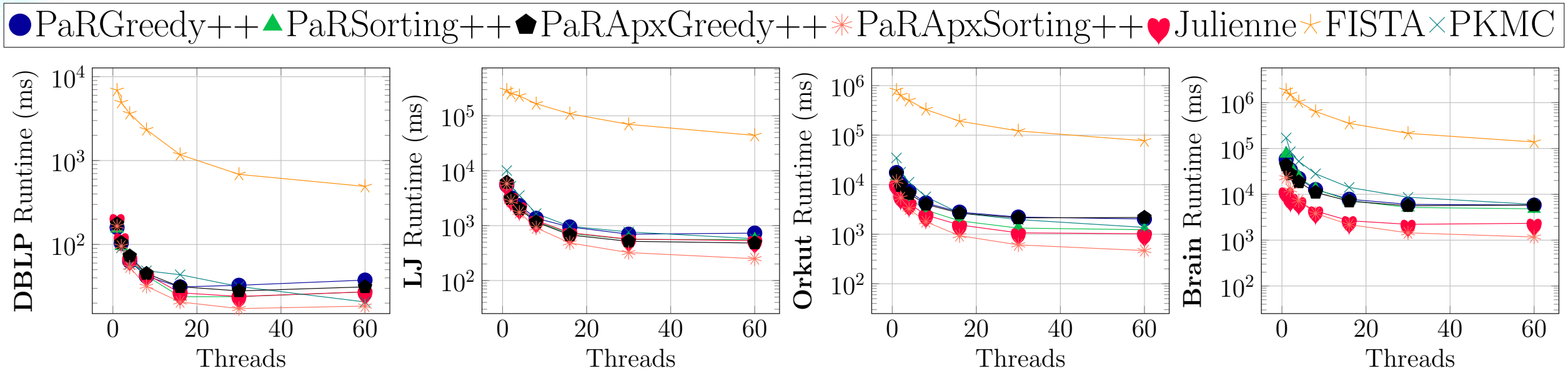


Our algorithms faster on all graphs, **up to 25.9x faster** than second fastest

Obtain the Best Density Approximations on Largest Publicly Available Graphs

- Our algorithms take:
  - **8.41 sec** on twitter
  - **10.54 sec** on friendster
  - **83.91 sec** on clueweb
  - **270.39 sec** on hyperlink2012
- Densities given in our paper:
  - **1643.301** on twitter
  - **273.518** on friendster
  - **2122.5** on clueweb
  - **6496.649** on hyperlink2012

Scalability



We achieve up to a **20.51x self-relative speedup**
and better speedup on **8 of the 12 tested graphs**

Open Questions

- Can we apply our framework for other problems?
- Can any theoretically optimal parallel algorithms be better than our algorithms?
- Can we combine our framework with other densest subgraph algorithms to achieve better runtimes?