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: \[ \rho(G) = \frac{|E|}{|V|} = \frac{9}{7} \]

\[ V = \text{set of vertices, } E = \text{set of edges} \]
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 Densest Subgraph $H^*$: $\rho(H^*) = \frac{7}{5}$

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

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

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

$\textbf{Core Number}$ of Node $\nu$: $\textbf{Maximum } k \text{ where } k\text{-core contains } \nu$

$k$-Core: Maximal induced subgraph where each vertex has degree at least $k$
**Core Number** of Node \( \nu \):
**Maximum** \( k \) where \( k \)-core contains \( \nu \)

**\( k \)-Core**: Maximal induced subgraph where each vertex has degree at least \( k \)
Densest Subgraph and $k$-Core Decomposition

- Let $G^*$ be the densest subgraph, $\rho^*$ be its density, $k_{\text{max}}$ be the maximum core number, and $G_{k'}$ be the $k'$-core
Densest Subgraph and $k$-Core Decomposition

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

• Folklore:

\[
\frac{k_{\text{max}}}{2} \leq \rho^* \leq k_{\text{max}}
\]

and

\[
G^* \in G_{k_{\text{max}}/2}
\]
Densest Subgraph and $k$-Core Decomposition

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

• Folklore:

\[
\frac{k_{\text{max}}}{2} \leq \rho^* \leq k_{\text{max}}
\]

and

\[
G^* \in G_{k_{\text{max}}/2}
\]

for any $k \leq \lfloor \rho^* \rfloor$
Parallel Framework

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

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

Refine the estimate of the density of the densest subgraph.
Parallel Framework

Find the $k_{\text{max}}/2$-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 $k_{\text{max}}/2$-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 $k_{\text{max}}/2$-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 $k_{\text{max}}/2$-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 $k_{\text{max}}/2$-core and **prune** all vertices not in the core
- **Refine** the estimate of the density of the densest subgraph
- **Prune again**
- Output **best density**

Existing **Fast and parallel** approximate $k$-core decomposition alg
Parallel Framework

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

Refine the estimate of the density of the densest subgraph

Prune again

Output best density

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

Fast and almost optimally parallel approximate MWU densest subgraph alg
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

Output best density

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

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

Prune again

Fast and almost optimally parallel approximate MWU densest subgraph alg

[Dhulipala-Blelloch-Shun '17]
[Liu et al. '22]
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 \textit{lowest degree}
- Return \textit{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

\[ \rho(G_0) \approx 1.6 \]
2-Approx Peeling Algorithm [Charikar ‘00]

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

\[ \rho(G_0) \approx 1.6 \quad \rho(G_1) = 1.5 \]
2-Approx Peeling Algorithm [Charikar ‘00]

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

\[
\rho(G_0) \approx 1.6 \quad \rho(G_1) = 1.5 \quad \rho(G_2) \approx 1.3
\]
2-Approx Peeling Algorithm [Charikar ‘00]

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

\[
\rho(G_0) \approx 1.6 \quad \rho(G_1) = 1.5 \quad \rho(G_2) \approx 1.3
\]

\[
\rho(G_3) \approx 1.25
\]

\[G_3\]
2-Approx Peeling Algorithm [Charikar ‘00]

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

\[ \rho(G_0) \approx 1.6 \quad \rho(G_1) = 1.5 \quad \rho(G_2) \approx 1.3 \quad \rho(G_3) \approx 1.25 \quad \ldots \]
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

3

3

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

![Diagram of a graph with labeled nodes and edges]
(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

\[3\]

\[2\] \[3\]

\[2\]

\[3\]
(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

\[ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 2 & 3 & 2 & 3 \\ 3 & 3 & 2 & 2 \\ 2 & 3 & 3 & 3 \end{bmatrix} \]
(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++ \cite{Boob et al. '20}

- Run peeling for multiple rounds
  - Each round: peel node with smallest \textit{degree} + \textit{load}
  - \textit{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 + \(\epsilon\))-Approx Greedy++ \[\text{Boob et al. ‘20}\]

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

\[(1 + \epsilon)\text{-approx. in } \tilde{O}\left(\frac{\Delta}{\rho^*\epsilon^2}\right) \text{ rounds}\]

[Chekuri-Quanrud-Torres ‘22]

Max Density Round 1: 1.6

Max Density Round 2: 1.7
(1 + \varepsilon)-Approx GreedySorting++

For \( t = 1 \ldots T \)

while \( G \) is not empty
  \( v_{\text{min}} \leftarrow \arg \min_v \deg(v) + \ell(v) \)
  \( \ell(v_{\text{min}}) \leftarrow \ell(v_{\text{min}}) + \deg(v_{\text{min}}) \)
  \( G \leftarrow G \setminus v_{\text{min}} \)
  \( \rho^* \leftarrow \max(\rho^*, \rho(G)) \)
reset \( G \)
return \( \rho^* \)

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

For \( t = 1 \ldots T \)

while \( G \) is not empty

\( v_{\text{min}} \leftarrow \arg\min_v \deg(v) + \ell(v) \)

\( \ell(v_{\text{min}}) \leftarrow \ell(v_{\text{min}}) + \deg(v_{\text{min}}) \)

\( G \leftarrow G \setminus v_{\text{min}} \)

\( \rho^* \leftarrow \max(\rho^*, \rho(G)) \)

reset \( G \)

return \( \rho^* \)

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

For \(t = 1 \ldots T\)

while \(G\) is not empty

\[v_{\text{min}} \leftarrow \arg \min_v \deg(v) + \ell(v)\]
\[\ell(v_{\text{min}}) \leftarrow \ell(v_{\text{min}}) + \deg(v_{\text{min}})\]
\[G \leftarrow G \setminus v_{\text{min}}\]
\[\rho^* \leftarrow \max(\rho^*, \rho(G))\]
reset \(G\)
return \(\rho^*\)

Greedy++

Highly sequential

For \(t = 1 \ldots T\)

while \(G\) is not empty

\[v_{\text{min}} \leftarrow \arg \min_v \ell(v)\]
\[\ell(v_{\text{min}}) \leftarrow \ell(v_{\text{min}}) + \deg(v_{\text{min}})\]
\[G \leftarrow G \setminus v_{\text{min}}\]
\[\rho^* \leftarrow \max(\rho^*, \rho(G))\]
reset \(G\)
return \(\rho^*\)

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

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

GreedySorting++

GreedySorting++
very parallelizable
(1 + \epsilon)-Approx GreedySorting++

GreedySorting++
very parallelizable

(1 + \epsilon)-approx.
in \tilde{O}\left(\frac{\Delta}{\rho^*\epsilon^2}\right) rounds

[Chekuri-Quanrud-Torres ‘22]

For \(t = 1 \ldots T\)
while \(G\) is not empty
\[
\nu_{\text{min}} \leftarrow \text{argmin}_v \ell(v)
\]
\[
\ell(\nu_{\text{min}}) \leftarrow \ell(\nu_{\text{min}}) + \deg(\nu_{\text{min}})
\]
\[
G \leftarrow G \setminus \nu_{\text{min}}
\]
\[
\rho^* \leftarrow \max(\rho^*, \rho(G))
\]
reset \(G\)
return \(\rho^*\)

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 $O(\log n)$ depth
Experimental Setup

• 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

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

• (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

<table>
<thead>
<tr>
<th>Graph Dataset</th>
<th>Original Graph</th>
<th>$k_{max}$</th>
<th>$core(G, \left\lfloor \frac{k_{max}}{2} \right\rfloor)$</th>
<th>Vertex Ratio</th>
<th>Edge Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Num. Vertices</td>
<td>Num. Edges</td>
<td>Num. Vertices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>closecliques</td>
<td>3,230</td>
<td>95,400</td>
<td>59</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>cahepth</td>
<td>9,877</td>
<td>25,973</td>
<td>31</td>
<td>0.0097</td>
<td>0.088</td>
</tr>
<tr>
<td>ascedia</td>
<td>26,475</td>
<td>106,762</td>
<td>22</td>
<td>0.007</td>
<td>0.048</td>
</tr>
<tr>
<td>hepgh</td>
<td>28,094</td>
<td>3,148,447</td>
<td>410</td>
<td>0.224</td>
<td>0.494</td>
</tr>
<tr>
<td>dbp</td>
<td>317,080</td>
<td>1,049,866</td>
<td>101</td>
<td>0.001</td>
<td>0.010</td>
</tr>
<tr>
<td>brain</td>
<td>784,262</td>
<td>267,844,669</td>
<td>1,200</td>
<td>0.239</td>
<td>0.512</td>
</tr>
<tr>
<td>wiki</td>
<td>1,094,018</td>
<td>2,787,967</td>
<td>124</td>
<td>0.034</td>
<td>0.090</td>
</tr>
<tr>
<td>youtube</td>
<td>1,138,499</td>
<td>2,990,443</td>
<td>51</td>
<td>0.011</td>
<td>0.110</td>
</tr>
<tr>
<td>stackoverflow</td>
<td>2,584,164</td>
<td>28,183,518</td>
<td>163</td>
<td>0.016</td>
<td>0.187</td>
</tr>
<tr>
<td>livejournal</td>
<td>4,846,609</td>
<td>42,851,237</td>
<td>329</td>
<td>0.001</td>
<td>0.022</td>
</tr>
<tr>
<td>orkut</td>
<td>3,072,441</td>
<td>117,185,083</td>
<td>253</td>
<td>0.023</td>
<td>0.113</td>
</tr>
<tr>
<td>twitter</td>
<td>41,652,230</td>
<td>1,202,513,046</td>
<td>2,484</td>
<td>0.001</td>
<td>0.029</td>
</tr>
<tr>
<td>friendster</td>
<td>65,608,366</td>
<td>1,806,067,135</td>
<td>304</td>
<td>0.022</td>
<td>0.146</td>
</tr>
<tr>
<td>cluemweb</td>
<td>978,408,098</td>
<td>37,372,179,311</td>
<td>4,244</td>
<td>9.39e-05</td>
<td>0.003</td>
</tr>
<tr>
<td>hyperlink2012</td>
<td>3,563,602,789</td>
<td>112,920,331,616</td>
<td>10,565</td>
<td>7.02e-05</td>
<td>0.009</td>
</tr>
</tbody>
</table>
## Dataset Information

<table>
<thead>
<tr>
<th>Graph Dataset</th>
<th>Original Graph</th>
<th>$k_{max}$</th>
<th>$\text{core}(G, \frac{k_{max}}{2})$</th>
<th>Vertex Ratio</th>
<th>Edge Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>closecliques</td>
<td>3,230</td>
<td>95,400</td>
<td>59</td>
<td>3,230</td>
<td>95,400</td>
</tr>
<tr>
<td>caehepht</td>
<td>9,877</td>
<td>25,973</td>
<td>31</td>
<td>96</td>
<td>2,306</td>
</tr>
<tr>
<td>ascaida</td>
<td>26,475</td>
<td>106,762</td>
<td>22</td>
<td>208</td>
<td>6,244</td>
</tr>
<tr>
<td>heph</td>
<td>28,094</td>
<td>3,148,447</td>
<td>410</td>
<td>6,304</td>
<td>1,562,818</td>
</tr>
<tr>
<td>dblp</td>
<td>317,080</td>
<td>1,049,866</td>
<td>101</td>
<td>280</td>
<td>13,609</td>
</tr>
<tr>
<td>brain</td>
<td>784,262</td>
<td>267,844,669</td>
<td>1,200</td>
<td>187,494</td>
<td>137,354,946</td>
</tr>
<tr>
<td>wiki</td>
<td>1,094,018</td>
<td>2,787,967</td>
<td>124</td>
<td>3,807</td>
<td>344,553</td>
</tr>
<tr>
<td>youtube</td>
<td>1,138,499</td>
<td>2,990,443</td>
<td>51</td>
<td>12,836</td>
<td>439,678</td>
</tr>
<tr>
<td>stackoverflow</td>
<td>2,584,164</td>
<td>28,183,518</td>
<td>163</td>
<td>41,651</td>
<td>5,709,796</td>
</tr>
<tr>
<td>livejournal</td>
<td>4,846,609</td>
<td>42,851,237</td>
<td>329</td>
<td>6,090</td>
<td>1,054,941</td>
</tr>
<tr>
<td>orkut</td>
<td>3,072,441</td>
<td>117,185,083</td>
<td>253</td>
<td>71,507</td>
<td>13,469,722</td>
</tr>
<tr>
<td>twitter</td>
<td>41,652,230</td>
<td>1,202,513,046</td>
<td>2,484</td>
<td>24,480</td>
<td>36,136,023</td>
</tr>
<tr>
<td>friendster</td>
<td>65,608,366</td>
<td>1,806,067,135</td>
<td>304</td>
<td>1,474,236</td>
<td>271,902,207</td>
</tr>
<tr>
<td>cluweb</td>
<td>978,408,098</td>
<td>37,372,179,311</td>
<td>4,244</td>
<td>91,874</td>
<td>132,549,663</td>
</tr>
<tr>
<td>hyperlink2012</td>
<td>3,563,602,789</td>
<td>112,920,331,616</td>
<td>10,565</td>
<td>250,477</td>
<td>1,046,929,322</td>
</tr>
</tbody>
</table>
## Dataset Information

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

Most graphs have less than 15% of edges in $k_{max}/2$ core!
Many graphs have **less than 1/2** of edges in core of max approx. density core than $k_{\text{max}}/2$ core!
Runtime of All Algorithms on Smaller Graphs

- PaRGreedy++
- PaRSorting++
- PaRApxGreedy++
- PaRApxSorting++
- FISTA
- Greedy++
- FrankWolfe
- MWU
- cCoreG++
- cCoreExact
- PKMC

Figure 2: Densities on different iterations for various algorithms. Only our algorithms can successfully process all of the large graphs (bottom row) within the 1 hour limit.

Figure 3: Runtimes (ms) of PaRGreedy++, PaRSorting++, PaRApxGreedy++, PaRApxSorting++, Julienne, FISTA, and PKMC versus the number of threads when running for 5 iterations.

Figure 4: Runtimes of different densest subgraph algorithms on our small graph inputs. The algorithms are run for 20 iterations. Parallel algorithms use 60 hyper-threads.

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?