Shared-Memory Exact Minimum Cuts
M. Henzinger, A. Noe, C. Schulz, D. Strash
Kernelization
General Idea

Reductions:
rules to decrease graph size, while maintaining optimality

solve problem on problem kernel
→ obtain solution on input graph
Kernelization

General Idea

Reductions:
rules to decrease graph size, while maintaining optimality

solve problem on problem kernel (using a heuristic)
→ obtain solution on input graph quickly
Kernelization

General Idea

Reductions:
rules to decrease graph size, while maintaining optimality

solve problem on problem kernel
→ obtain solution on input graph

Independent Sets
- evolutionary [SEA’15]
- reduction + evolutionary [ALX’16]
- online reductions + LS [SEA’16]
- shared-mem parallel [ALX’18]
- weighted exact [ALX’19]
Kernelization

General Idea

Reductions:
rules to decrease graph size, while maintaining optimality

solve problem on problem kernel
→ obtain solution on input graph

Independent Sets
- ...
- shared-mem parallel [ALX'18]
- weighted exact [ALX'19]

“Graph Partitioning” […]

Minimum Cuts
- shared-mem parallel [ALX'18]
- exact minimum cut [ALX'19]
Concrete Example
"(In)exact Reductions" in Minimum Cuts

joint work with
M. Henzinger,
A. Noe,
D. Strash
Minimum Cuts

Cut: A **cut** in a multigraph is a partition of $V = C \cup \overline{C}$

$\rightarrow$ size of the cut is weight of edges between $C$ and $\overline{C}$

Minimum Cut Problem:

what is the size of the minimum cut in $G$?
If the size of the minimum cut is $\lambda$, then it follows

- $\forall v \in V : \deg(v) \geq \lambda$
- number of edges $m \geq n\lambda / 2$

**Proof:** Assume $\exists v \in V : \deg(v) < \lambda$, then $C = \{v\}$ is a cut whose size is $< \lambda$. Contradiction. The second claim follows from the first one.
**Contraction**

In a multigraph $G$, let $u$ and $v$ be connected by an edge $e = \{x, y\}$

Create $G/e = (V', E')$ by **contracting** $e$:
- set $V'$ to $V \backslash \{x, y\} \cup \{z\}$ ($z$ is new)
- build $E'$ from $E$ by
  - remove all edges between $u$ and $v$
  - replace every edge between $v \in V \backslash \{x, y\}$ and $x$ or $y$ by an edge between $v$ and $z$
  - keep all other edges from $E$

→ multi-edges can be created (⇝ practice use weights)!
Minimum Cut $\leftrightarrow$ Contraction

A minimum cut in $G/e$ is at least as a minimum cut in $G$.

Proof:
Let $(K, \overline{K})$ be a minimum cut in $G/e$.
Let the size of the cut be $\lambda$.
Wlog let $x$ and $y$ be the vertices of $e$, and $z \in K$
Unpack $z$ and leave $x$ and $y$ in $K \rightarrow$ cut in $G$ of size $\lambda$
Algorithm

Exibit A

\[ H \leftarrow G \]

\textbf{while} \( H \) has more than 2 nodes \textbf{do}

\quad \textbf{e} \leftarrow \text{edge of } H \text{ picked uniformly at random}

\quad \textbf{H} \leftarrow \text{contract}(H, e)

\textbf{done}

\((C, \overline{C}) \leftarrow \text{vertex set in } G \text{ that correspond to the vertices in } H\)
Algorithm

Exibit A

\( H \leftarrow G \)

**while** \( H \) has more than 2 nodes **do**

\( e \leftarrow \text{edge of } H \text{ picked uniformly at random} \)

\( H \leftarrow \text{contract}(H, e) \)

**done**

\( (C, \overline{C}) \leftarrow \text{vertex set in } G \text{ that correspond to the vertices in } H \)

---

\[ H \leftarrow G \]

**while** \( H \) has more than 2 nodes **do**

\( e \leftarrow \text{edge of } H \text{ picked uniformly at random} \)

\( H \leftarrow \text{contract}(H, e) \)

**done**

\( (C, \overline{C}) \leftarrow \text{vertex set in } G \text{ that correspond to the vertices in } H \)
Algorithm

Exibit A

\[ H \leftarrow G \]

while \( H \) has more than 2 nodes do

\[ e \leftarrow \text{edge of } H \text{ picked uniformly at random} \]

\[ H \leftarrow \text{contract}(H, e) \]

done

\[ (C, \overline{C}) \leftarrow \text{vertex set in } G \text{ that correspond to the vertices in } H \]
Algorithm
Exhibit A

\[
H \leftarrow G
\]

while \( H \) has more than 2 nodes do
\[
e \leftarrow \text{edge of } H \text{ picked uniformly at random}
H \leftarrow \text{contract}(H, e)
\]
done

\((C, \overline{C}) \leftarrow \text{vertex set in } G \text{ that correspond to the vertices in } H\)
Algorithm
Exhibit A

$H \leftarrow G$

while $H$ has more than 2 nodes do

\[ e \leftarrow \text{edge of } H \text{ picked uniformly at random} \]

\[ H \leftarrow \text{contract}(H, e) \]

done

$(C, \overline{C}) \leftarrow \text{vertex set in } G \text{ that correspond to the vertices in } H$
Algorithm
Exibit A

$H ← G$

\textbf{while} $H$ has more than 2 nodes \textbf{do}

\hspace{1em} $e ←$ edge of $H$ picked uniformly at random

\hspace{1em} $H ← \text{contract}(H, e)$

\textbf{done}

$(C, \overline{C}) ←$ vertex set in $G$ that correspond to the vertices in $H$
Algorithm
Exhibit A

\[
H \leftarrow G
\]

while \( H \) has more than 2 nodes do

\[
e \leftarrow \text{edge of } H \text{ picked uniformly at random}
\]

\[
H \leftarrow \text{contract}(H, e)
\]

done

\[
(C, \overline{C}) \leftarrow \text{vertex set in } G \text{ that correspond to the vertices in } H
\]
Algorithm
Exibit A

\[ H \leftarrow G \]

while \( H \) has more than 2 nodes do
    \( e \leftarrow \) edge of \( H \) picked uniformly at random
    \( H \leftarrow \) contract\((H, e)\)

done

\((C, \overline{C})\leftarrow \) vertex set in \( G \) that correspond to the vertices in \( H \)
Algorithm
Exibit A

\[ H \leftarrow G \]

while \( H \) has more than 2 nodes do
  \( e \leftarrow \text{edge of } H \text{ picked uniformly at random} \)
  \( H \leftarrow \text{contract}(H, e) \)
done

\((C, \overline{C}) \leftarrow \text{vertex set in } G \text{ that correspond to the vertices in } H \)

The runtime of the simple minimum cut algorithm is \( O(n^2) \)

Proof:

- every call \( \text{contract}(H, e) \) is done in \( O(n) \)
- every loop iteration reduces \( n \) by 1 \( \rightarrow n - 2 \) iterations
Algorithm

Exhibit A

\[ H \leftarrow G \]

while \( H \) has more than 2 nodes do
  \[ e \leftarrow \text{edge of } H \text{ picked uniformly at random} \]
  \[ H \leftarrow \text{contract}(H, e) \]
done

\[ (C, \overline{C}) \leftarrow \text{vertex set in } G \text{ that correspond to the vertices in } H \]

The algorithm finds a minimum cut with probability \( \Omega(n^{-2}) \)

Proof (sketch):
- let minimum cut size be \( \lambda \)
- probability to select a cut edge \( \frac{\lambda}{|E|} \leq \frac{\lambda}{n\lambda/2} = 2/n \)
- \( p_n \) probability that \( n \)-vertex graph avoids cut edges

\[ p_n \geq (1 - 2/n)p_{n-1} \geq \ldots = \left(\frac{n}{2}\right)^{-1} \]
Algorithm

Exhibit A

\[ H \leftarrow G \]

while \( H \) has more than 2 nodes do

\[ e \leftarrow \text{edge of } H \text{ picked uniformly at random} \]

\[ H \leftarrow \text{contract}(H, e) \]

done

\[ (C, \overline{C}) \leftarrow \text{vertex set in } G \text{ that correspond to the vertices in } H \in \text{runtime } O(n^4) \]

---

Standard Trick: Multiple Repetitions

- non-error probability \( 1/n^2 \) very low
- smallest out of \( n^2 / 2 \) is minimum with probability \( 1 - 1/e \):

\[ (1 - 2/n^2)^2/n^2 < 1/e \]

\[ \Rightarrow \text{runtime } O(n^4) \]
Better Algorithm
IterContract
\[ H \leftarrow G \]
while \( H \) has more than \( t \) nodes do
\[ e \leftarrow \text{edge of } H \text{ picked uniformly at random} \]
\[ H \leftarrow \text{contract}(H, e) \]
done
return \( H \)

\[ H \text{ still contains minimum cut with probability at least } \frac{t^2}{n^2} \]
Karger-Stein

\[
\text{if } |V| \leq 6 \text{ then } C \leftarrow \text{optimal cut by deterministic algorithm}
\]

\[
\text{else}
\]

\[
t \leftarrow \lceil 1 + n / \sqrt{2} \rceil
\]

\[
H_1 \leftarrow \text{IterContract}(G, t)
\]

\[
H_2 \leftarrow \text{IterContract}(G, t)
\]

\[
C_1 \leftarrow \text{CallRecursive}(H_1)
\]

\[
C_2 \leftarrow \text{CallRecursive}(H_2)
\]

\[
C \leftarrow \min(C_1, C_2)
\]

\text{done}

\text{return } C

\Rightarrow \text{running time } O(n^2 \log n)

\Rightarrow \text{minimum cut with probability } \Omega(1/ \log n)

\Rightarrow \text{repeat } \log^2 n \text{ to achieve probability } \Omega(1/n)
Main Questions

how can kernelization help?
Main Questions

something better than contracting random edges?
Main Questions

can we still obtain good cuts in practice?
can we then use this to obtain better kernels?
Kernelization
Padberg-Rinaldi Tests

\[ \deg(x) \leq 2\omega(x, y) \]
Kernelization
Padberg-Rinaldi Tests

$$\omega(x, y) \geq \hat{\lambda}$$

$4$

x

y
\[ \exists z : \deg(x) \leq 2\{\omega(x, y) + \omega(x, z)\} \text{ and } \deg(y) \leq 2\{\omega(x, y) + \omega(y, z)\} \]
Kernelization
Padberg-Rinaldi Tests

\[ \omega(x, y) + \sum_z \min\{\omega(x, z), \omega(y, z)\} \geq \hat{\lambda} \]
Kernelization
Nagamochi, Ono, and Ibaraki

**Key Idea:** a spanning tree contains at least one edge from any cut

- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)
  - $\leadsto$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$
  - $\leadsto$ contract all of them

Example: $\hat{\lambda} = 4 \leadsto$ compute 3 edge-disjoint spanning forests
Kernelization
Nagamochi, Ono, and Ibaraki

Key Idea: a spanning tree contains at least one edge from any cut
- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)
  $\Rightarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$
  $\Rightarrow$ contract all of them

Example: $\hat{\lambda} = 4 \Rightarrow$ compute 3 edge-disjoint spanning forests
Kernelization
Nagamochi, Ono, and Ibaraki

Key Idea: a spanning tree contains at least one edge from any cut

- Let $\lambda$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\lambda - 1$ maximal spanning forests (iteratively)
  $\Rightarrow$ edges not in forests connect vertices with connectivity $\geq \lambda$
  $\Rightarrow$ contract all of them

Example: $\lambda = 4 \Rightarrow$ compute 3 edge-disjoint spanning forests
Kernelization
Nagamochi, Ono, and Ibaraki

Key Idea: a spanning tree contains at least one edge from any cut

- Let \( \hat{\lambda} \) be your current bound for minimum cut
- Want: smaller minimum cut
- Compute \( \lambda - 1 \) maximal spanning forests (iteratively)
  - \( \rightsquigarrow \) edges not in forests connect vertices with connectivity \( \geq \hat{\lambda} \)
  - \( \rightsquigarrow \) contract all of them

NOI define modified BFS to detect contractable edges
(more later)
Kernelization
Nagamochi, Ono, and Ibaraki

Key Idea: a spanning tree contains at least one edge from any cut

- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)
  $\Rightarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$
  $\Rightarrow$ contract all of them

Note: initial $\hat{\lambda}$ comes from minimum degree
Some of the reductions depend heavily on $\hat{\lambda}$
Label Propagation
Cut-based, Linear Time Clustering Algorithm [Raghavan et. al]

- **cut-based** clustering using label propagation
  - start with **singletons**
  - traverse nodes in random order or **smallest degree first**
  - move node to cluster having **strongest** connection

![Diagram of Label Propagation](image)
Label Propagation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Cut [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>8.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>5.66</td>
</tr>
<tr>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
</tr>
<tr>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>5.21</td>
</tr>
<tr>
<td>8</td>
<td>5.18</td>
</tr>
<tr>
<td>...</td>
<td>5.09</td>
</tr>
</tbody>
</table>
Label Propagation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Cut [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>8.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>5.66</td>
</tr>
<tr>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
</tr>
<tr>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>5.21</td>
</tr>
<tr>
<td>8</td>
<td>5.18</td>
</tr>
<tr>
<td>...</td>
<td>5.09</td>
</tr>
</tbody>
</table>
## Label Propagation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Cut [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>8.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>5.66</td>
</tr>
<tr>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
</tr>
<tr>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>5.21</td>
</tr>
<tr>
<td>8</td>
<td>5.18</td>
</tr>
<tr>
<td>...</td>
<td>5.09</td>
</tr>
</tbody>
</table>
Label Propagation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Cut [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>8.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>5.66</td>
</tr>
<tr>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
</tr>
<tr>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>5.21</td>
</tr>
<tr>
<td>8</td>
<td>5.18</td>
</tr>
<tr>
<td>...</td>
<td>5.09</td>
</tr>
</tbody>
</table>
Label Propagation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Cut [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>8.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>5.66</td>
</tr>
<tr>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
</tr>
<tr>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>5.21</td>
</tr>
<tr>
<td>8</td>
<td>5.18</td>
</tr>
<tr>
<td>...</td>
<td>5.09</td>
</tr>
</tbody>
</table>
## Label Propagation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Cut [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>8.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>5.66</td>
</tr>
<tr>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
</tr>
<tr>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>5.21</td>
</tr>
<tr>
<td>8</td>
<td>5.18</td>
</tr>
<tr>
<td>...</td>
<td>5.09</td>
</tr>
</tbody>
</table>
Label Propagation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Cut [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>8.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>5.66</td>
</tr>
<tr>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
</tr>
<tr>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>5.21</td>
</tr>
<tr>
<td>8</td>
<td>5.18</td>
</tr>
<tr>
<td>...</td>
<td>5.09</td>
</tr>
</tbody>
</table>
Label Propagation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Cut [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>8.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>5.66</td>
</tr>
<tr>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
</tr>
<tr>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>5.21</td>
</tr>
<tr>
<td>8</td>
<td>5.18</td>
</tr>
<tr>
<td>...</td>
<td>5.09</td>
</tr>
</tbody>
</table>
## Label Propagation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Cut [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>8.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>5.66</td>
</tr>
<tr>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
</tr>
<tr>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>5.21</td>
</tr>
<tr>
<td>8</td>
<td>5.18</td>
</tr>
<tr>
<td>...</td>
<td>5.09</td>
</tr>
</tbody>
</table>
#### Label Propagation

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Cut [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td>8.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
</tr>
<tr>
<td>3</td>
<td>5.66</td>
</tr>
<tr>
<td>4</td>
<td>5.44</td>
</tr>
<tr>
<td>5</td>
<td>5.28</td>
</tr>
<tr>
<td>6</td>
<td>5.25</td>
</tr>
<tr>
<td>7</td>
<td>5.21</td>
</tr>
<tr>
<td>8</td>
<td>5.18</td>
</tr>
<tr>
<td>...</td>
<td>5.09</td>
</tr>
</tbody>
</table>
Basic Idea
Contraction of Clusterings

- cluster paradigm: internally dense, externally sparse
- “unlikely” to contract minimum cut edges
- clustering not main goal: only perform a couple of iterations
Fast Inexact Minimum Cuts

- (Inexact) Cluster reduction + Exact reductions
- Solve kernel to optimality using Nagamochi, Ono and Ibaraki’s algorithm
  → overall linear running time, but potentially suboptimal cuts
Parallelization
Shared-memory with OpenMP

- Parallel label propagation

- as brutal as: pragma openmp for and ignore conflicts on labels

- Parallel Padberg-Rinaldi:
  - check edges independently $\leadsto$ embarassingly parallel
  - collect edges then contract
    $\rightarrow$ essentially linear time

- Parallel contraction (not here)

- run Nagamochi, Ono and Ibarakis algorithm sequentially
Random Hyperbolic Networks

$n=12.5K - 200K \ d=10\% \ k=2$

- **seqVieCut** optimal in 99% of runs, **Matualna** optimal in 69% of runs
Real-world Networks

No incorrect results (expect Karger-Stein in 36% of the cases)
Parallelization

- Average speedup using 12 cores: 6.3 (24: 7.9)
- Average speedup to next fastest (Matula): 13.2 (24: 15.8)
Stating the Obvious

- now $\approx$ 16 times faster than Matualala
- NO guarantee for minimum cut, but experiments say very likely
- reductions depend on bound $\hat{\lambda}$

$\rightsquigarrow$ PLUG IN our result into exact NOI algorithm + parallelization
$\rightsquigarrow$ currently fastest exact minimum cut algorithm
Kernelization
Nagamochi, Ono, and Ibaraki

Key Idea: a spanning tree contains at least one edge from any cut

- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)
  $\Rightarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$
  $\Rightarrow$ contract all of them

Example: $\hat{\lambda} = 4 \Rightarrow$ compute 3 edge-disjoint spanning forests

![Diagram of a graph with three vertices connected by edges, illustrating the concept of spanning forests.](image-url)
Kernelization
Nagamochi, Ono, and Ibaraki

Key Idea: a spanning tree contains at least one edge from any cut
- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)
  $\Rightarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$
  $\Rightarrow$ contract all of them

Example: $\hat{\lambda} = 4 \Rightarrow$ compute 3 edge-disjoint spanning forests
Kernelization
Nagamochi, Ono, and Ibaraki

Key Idea: a spanning tree contains at least one edge from any cut

- Let \( \hat{\lambda} \) be your current bound for minimum cut
- Want: smaller minimum cut
- Compute \( \hat{\lambda} - 1 \) maximal spanning forests (iteratively)
  - \( \Rightarrow \) edges not in forests connect vertices with connectivity \( \geq \hat{\lambda} \)
  - \( \Rightarrow \) contract all of them

Example: \( \hat{\lambda} = 4 \Rightarrow \) compute 3 edge-disjoint spanning forests
Nagamochi, Ono, Ibaraki
Details

- $\lambda(x, y)$ capacity of minimum cut separating $x$ and $y$
- $\lambda(x, y) \geq \hat{\lambda} \iff \exists$ no cut separating $x$ and $y$ with capacity $\leq \hat{\lambda}$
  - $\iff$ we can contract $(x, y)$
- but computing $\lambda(x, y)$ expensive (max-flow algorithm)
- NOI: compute lower bound $q(e)$ on $\lambda(x, y)$, i.e.

\[ \lambda(x, y) \geq q(e) \geq \hat{\lambda} \]
  - $\iff$ can contract edge $e$

\[ q(e) = \# \text{ edge disjoint paths that connect } x, y \]
\[ q(e) \text{ via } k\text{-edge-connected subgraph } \iff \text{ following algorithm} \]
\(k\)-edge-connected subgraph

**Invariant** \( r[v] = i \) smallest \( i \) s.t. \( E_{i+1} \cup \{e\} \) does not contain a cycle

initialize \( r[v] = 0 \)
all nodes and edges are non-scanned
\( E_1 = E_2 = \ldots = E_{|E|} = \emptyset \)

**while** \( \exists \) non-scanned node
  \( u := \) non-scanned node \( v \) with maximal \( r[v] \)
  **foreach** non-scanned edge \( e = (u, v) \in E \) **do**
    insert \( e \) into \( E_{r(v)+1} \)
    \( q(e) = r(v) + 1, \quad r(v) = r(v) + 1 \)

\( \leadsto H_i = (V, E_i) \) is a maximal spanning forest in \( G \setminus E_1 \cup \ldots \cup E_{i-1} \)

Long story short:

Everything in \( E_{\hat{\lambda}} \cup \ldots \cup E_{|E|} \) can be contracted.

\( \leadsto \) contract \( e \) if \( q(e) \geq \hat{\lambda} \)
**k-edge-connected subgraph**

**Invariant** $r[v] = i$ smallest $i$ s.t. $E_{i+1} \cup \{e\}$ does not contain a cycle

**Invariant** $r[v] = i$ incident to first $i$ trees

Initialize $r[v] = 0$

All nodes and edges are non-scanned

$E_1 = E_2 = \ldots = E_{|E|} = \emptyset$

While $\exists$ non-scanned node

$u :=$ non-scanned node $v$ with maximal $r[v]$

**foreach** non-scanned edge $e = (u, v) \in E$ do

Insert $e$ into $E_{r(v)+1}$

$q(e) = r(v) + 1, r(v) = r(v) + 1, r(u) = r(u) + 1$

$H_i = (V, E_i)$ is a maximal spanning forest in $G \setminus E_1 \cup \ldots \cup E_{i-1}$

Long story short:

Everything in $E_{\hat{\lambda}} \cup \ldots \cup E_{|E|}$ can be contracted.

$\Rightarrow$ contract $e$ if $q(e) \geq \hat{\lambda}$
**k-edge-connected subgraph**

**Invariant** $r[v] = i$ smallest $i$ s.t. $E_{i+1} \cup \{e\}$ does not contain a cycle

initialize $r[v] = 0$

all nodes and edges are non-scanned

$E_1 = E_2 = \ldots = E_{|E|} = \emptyset$

while $\exists$ non-scanned node

$u :=$ non-scanned node $v$ with maximal $r[v]$

foreach non-scanned edge $e = (u, v) \in E$

insert $e$ into $E_{r(v)+1}, \ldots, E_{r(v)+c(e)}$

$q(e) = r(v) + c(e)$, $r(v) = r(v) + c(e)$

$\Rightarrow H_i = (V, E_i)$ is a maximal spanning forest in $G \setminus E_1 \cup \ldots \cup E_{i-1}$

Long story short:

Everything in $E_{\hat{\lambda}} \cup \ldots \cup E_{|E|}$ can be contracted.

$\Rightarrow$ contract $e$ if $q(e) \geq \hat{\lambda}$

$c(e)$ replaces one edge by $c(e)$ edges
Example
Overall Parallel Algorithm

- each thread selects random start vertex
- make sure each vertex scanned by exactly one worker
- mark contractible edge in parallel union-find data structure

1: $\hat{\lambda} \leftarrow \text{VieCut}(G), G_C \leftarrow G$
2: while $G_C$ has more than 2 vertices
3: $\hat{\lambda} \leftarrow \text{Parallel CAPFOREST}(G_C, \hat{\lambda})$
4: if no edges marked contractible
5: $\hat{\lambda} \leftarrow \text{CAPFOREST}(G_C, \hat{\lambda})$
6: $G_C, \hat{\lambda} \leftarrow \text{Parallel Graph Contract}(G_C)$
7: return $\hat{\lambda}$
More Optimizations

- Observation: values in PQ often higher than bound $\hat{\lambda}$
- Algorithm still correct when limiting values to $\hat{\lambda}$
- Use BucketPQ in weighted case also!

\[ \Rightarrow O(1) \text{ for push, pop, and increaseKey} \]
\[ \Rightarrow \text{Bucket implementations make a difference} \]

stack vs queue
breadth vs depth
All Graphs

HO – original Hao, Orlin algorithm implementation
NOI-CGKLS – original NOI implementation
NOI-HNSS – our own NOI implementation
NOI:
  BStack, BQueue, Heap
  $\hat{\lambda}$ – bounding PQ
  *-VieCut – initialize $\hat{\lambda}$ with VieCut
Random Hyperbolic Graphs

Average Node Degree: $2^8$

HO – original Hao, Orlin algorithm implementation
NOI-CGKLS – original NOI implementation
NOI-HNSS – our own NOI implementation
NOI:
  BStack, BQueue, Heap
  $\hat{\lambda}$ – bounding PQ
  *-VieCut – initialize $\hat{\lambda}$ with VieCut
Social and Web Graphs

HO – original Hao, Orlin algorithm implementation
NOI-CGKLS – original NOI implementation
NOI-HNSS – our own NOI implementation
NOI: BStack, BQueue, Heap
\(\hat{\lambda}\) – bounding PQ
*VieCut – initialize \(\hat{\lambda}\) with VieCut

**Figure:**
- Y-axis: Slow Down
- X-axis: Number of Edges
- Legend:
  - HO
  - NOI-CGKLS
  - NOI-HNSS
  - NOI\(\hat{\lambda}\)-Heap
  - NOI\(\hat{\lambda}\)-BStack
  - NOI\(\hat{\lambda}\)-BQueue
  - NOI\(\hat{\lambda}\)-HNSS-VieCut
  - NOI\(\hat{\lambda}\)-Heap-VieCut

33 Christian Schulz:
Shared-Memory Exact Minimum Cuts
Scalability

\[ \text{twitter-2010} \ (\lambda = 3) \quad \text{rhg}_{25\_8\_2} \ (\lambda = 73) \]

Number of Processes

34 Christian Schulz: Shared-Memory Exact Minimum Cuts
Open Things & Software

- apply heuristics on kernel
- use inexact results to get better bounds for reductions
- heuristic reduction to break up reduction space

Open Questions:
- what about the order or reductions in practice?
- MORE problems? (minimum fill, ...)
- the other way around: exact reductions for multi-level schemes
- integrating reductions in currently used algorithms

Software:
- https://viecut.taa.univie.ac.at