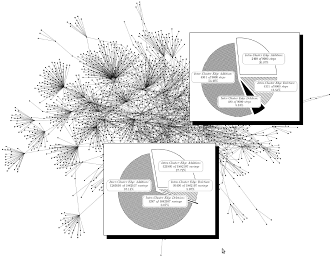
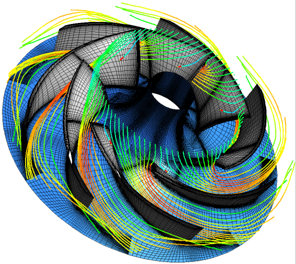
# Shared-Memory Exact Minimum Cuts

M. Henzinger, A. Noe, C. Schulz, D. Strash

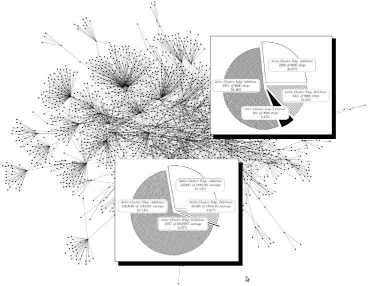




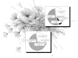
# Kernelization

## General Idea

**Reductions:**
rules to decrease graph size, while maintaining optimality




$\rightarrow$

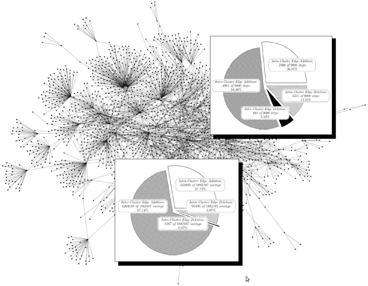


solve problem on problem kernel
$\rightarrow$ obtain solution on input graph

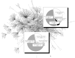
# Kernelization

## General Idea

**Reductions:**
rules to decrease graph size, while maintaining optimality




→

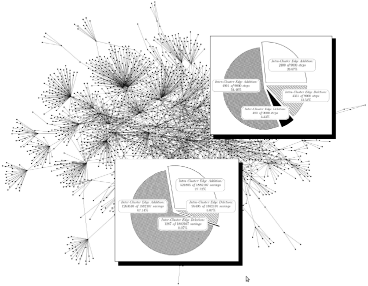


solve problem on problem kernel (using a heuristic)
→ obtain solution on input graph quickly

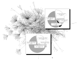
# Kernelization

## General Idea

**Reductions:**
rules to decrease graph size, while maintaining optimality



→



solve problem on problem kernel
→ obtain solution on input graph

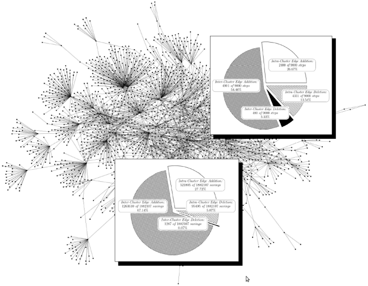
**Independent Sets**

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

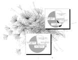
# Kernelization

## 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

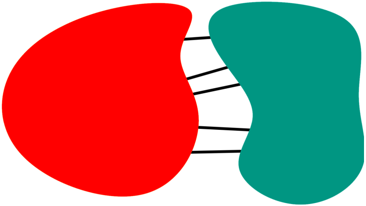
*joint work with*
*M. Henzinger,*
*A. Noe,*
*D. Strash*

# 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 $G$?



# Basics

If the size of the minimum cut is $\lambda$, then it follows

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

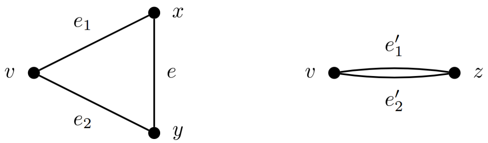
**Proof:** Assume $\exists v \in V : deg(v) < \lambda$,
then $C = \{v\}$ is a cut whose size is $< \lambda$. Contradiction.
The second claim follows from the first one.

# 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 $E$

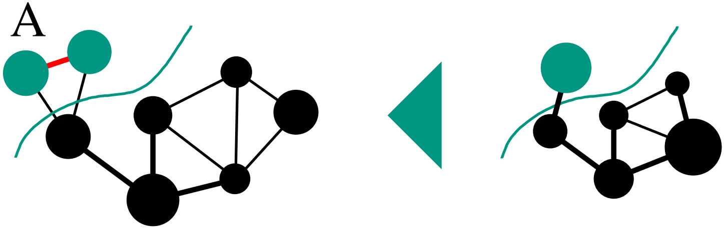



$\rightarrow$ multi-edges can be created ($\rightsquigarrow$ practice use weights)!

# Minimum Cut $\leftrightarrow$ Contraction

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

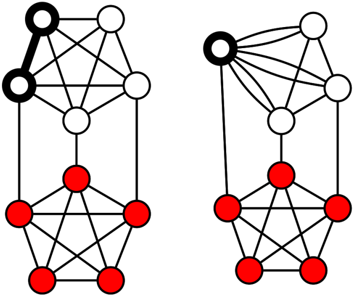
**Proof:**
Let $(K, \overline{K})$ be a minimum cut in $G/e$.
Let the size of the cut be $\lambda$.
Wlog let $x$ and $y$ be the vertices of $e$, and $z \in K$
Unpack $z$ and leave $x$ and $y$ in $K \rightarrow$ cut in $G$ of size $\lambda$



# Algorithm

**Exibit A**

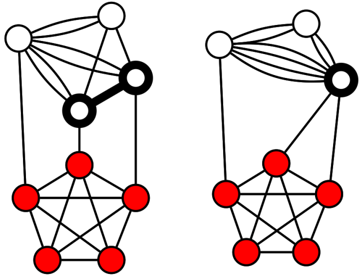
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

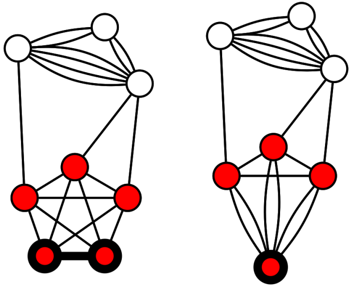
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

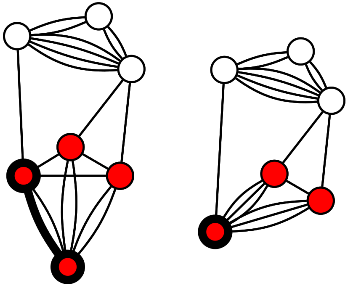
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

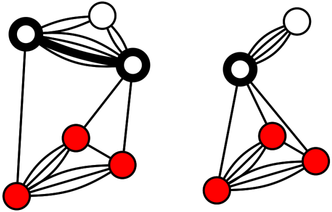
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

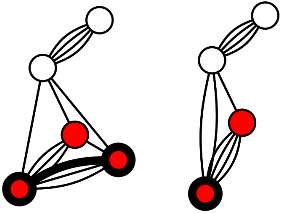
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

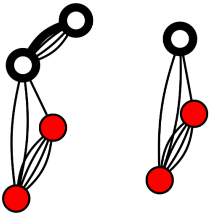
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

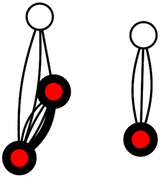
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ 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**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$

---

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

**Proof:**

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

# 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 $H$

---

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

$$p_n \geq (1 - 2/n)p_{n-1} \geq \ldots = \binom{n}{2}^{-1}$$

# Algorithm

**Exibit A**

$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$

---

Standard Trick: Multiple Repetitions

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

$$(1 - 2/n^2)^{2/n^2} < 1/e$$

$\rightsquigarrow$runtime $O(n^4)$

# Better Algorithm

**IterContract**

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

$H$ still contains minimum cut with probability at least

$$\binom{t}{2} / \binom{n}{2}$$

# Karger-Stein

**if** $|V| \leq 6$ **then** $C \leftarrow$ optimial cut by deterministic algorithm
**else**
$t \leftarrow \lceil 1 + n/\sqrt{2} \rceil$
$H_1 \leftarrow \text{IterContract}(G, t)$
$H_2 \leftarrow \text{IterContract}(G, t)$
$C_1 \leftarrow \text{CallRecursive}(H_1)$
$C_2 \leftarrow \text{CallRecursive}(H_2)$
$C \leftarrow \min(C_1, C_2)$
**done**
**return** $C$

$\rightsquigarrow$ running time $O(n^2 \log n)$

$\rightsquigarrow$ minimum cut with probability $\Omega(1/\log n)$

$\rightsquigarrow$ repeat $\log^2 n$ 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?

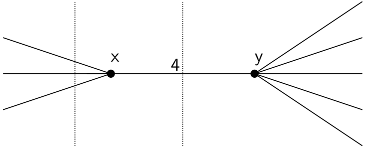
# Main Questions

can we then use this to obtain better kernels?

# Kernelization

## Padperg-Rinaldi Tests

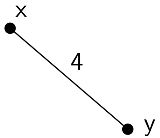
$$\deg(x) \leq 2\omega(x,y)$$



# Kernelization

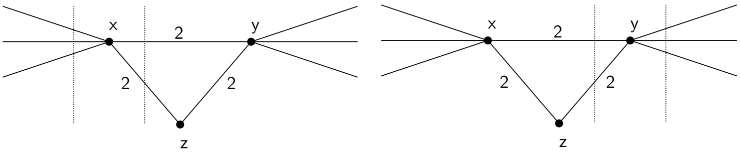
## Padperg-Rinaldi Tests

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



# Kernelization
## Padperg-Rinaldi Tests

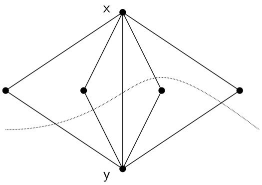
$$\exists z : \deg(x) \leq 2\{\omega(x,y) + \omega(x,z)\} \text{ and } \deg(y) \leq 2\{\omega(x,y) + \omega(y,z)\}$$



# Kernelization

## Padperg-Rinaldi Tests

$$\omega(x,y) + \sum_z \min\{\omega(x,z), \omega(y,z)\} \geq \hat{\lambda}$$



# Kernelization

**Nagamochi, Ono, and Ibaraki**

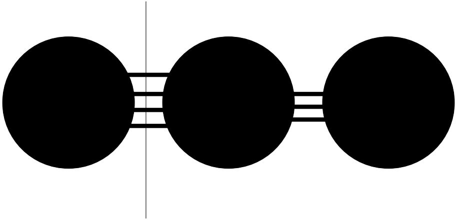
Key Idea: a spanning tree contains at least one edge from any cut

- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)

  $\rightsquigarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$

  $\rightsquigarrow$ contract all of them

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



# Kernelization

**Nagamochi, Ono, and Ibaraki**

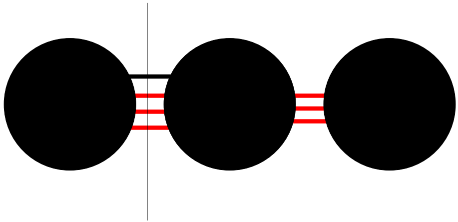
Key Idea: a spanning tree contains at least one edge from any cut

- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)

  $\rightsquigarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$

  $\rightsquigarrow$ contract all of them

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



# Kernelization

**Nagamochi, Ono, and Ibaraki**

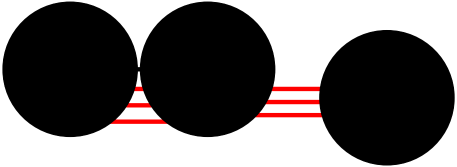
Key Idea: a spanning tree contains at least one edge from any cut

- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)

  $\rightsquigarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$

  $\rightsquigarrow$ contract all of them

Example: $\hat{\lambda} = 4 \rightsquigarrow$ 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)

$\rightsquigarrow$ 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)

  $\rightsquigarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$

  $\rightsquigarrow$ contract all of them

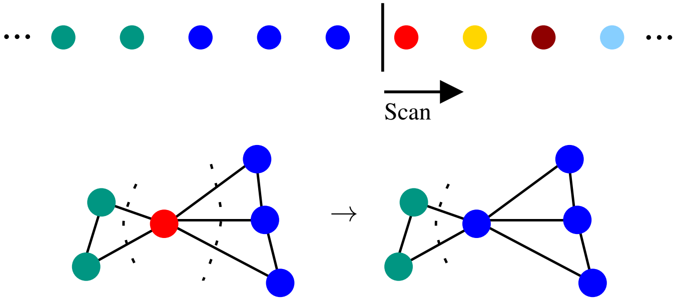
Note: initial $\hat{\lambda}$ comes from minimum degree

Some of the reductions depend heavily on $\hat{\lambda}$

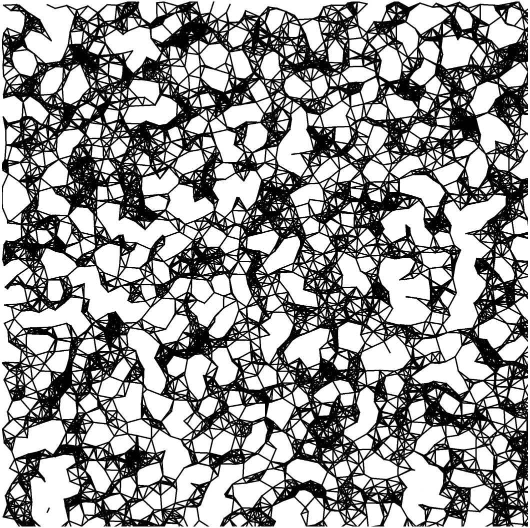
# Label Propagation

**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

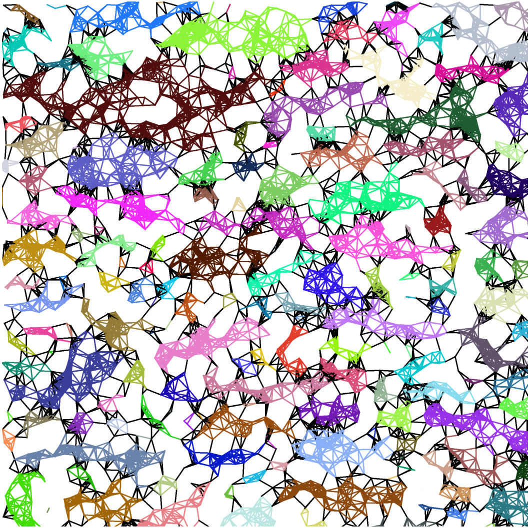


# Label Propagation



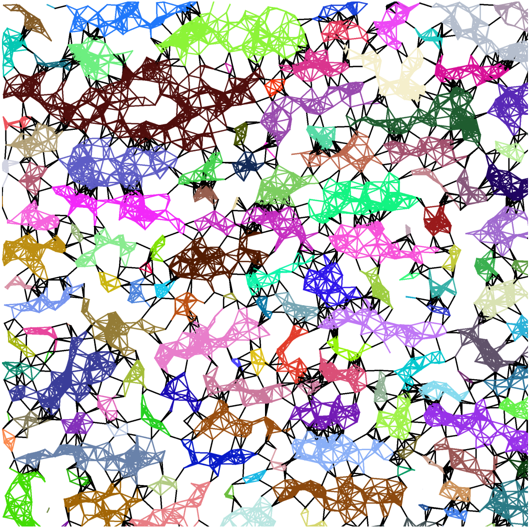
| Iteration | Cut [%] |
|---|---|
| **0** | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



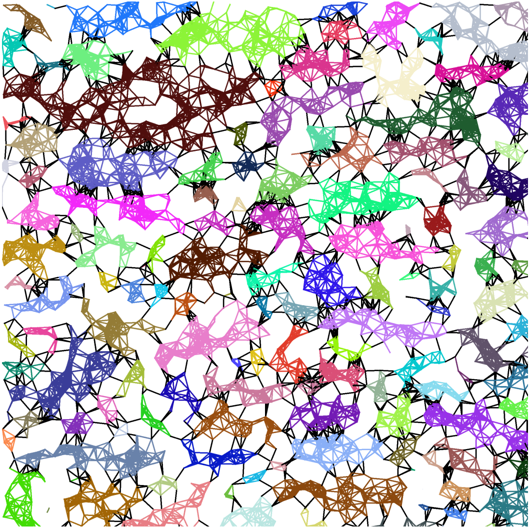
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| **1** | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



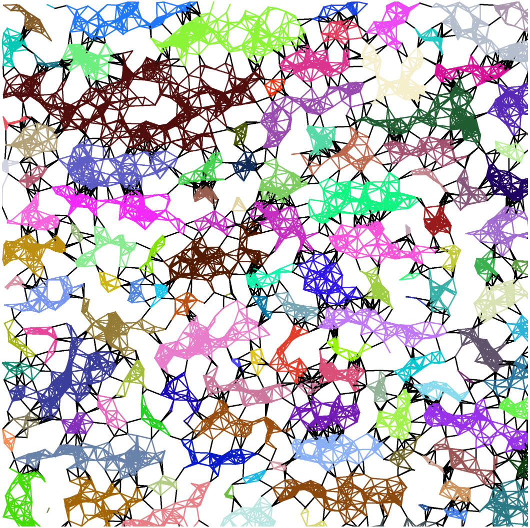
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| **2** | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



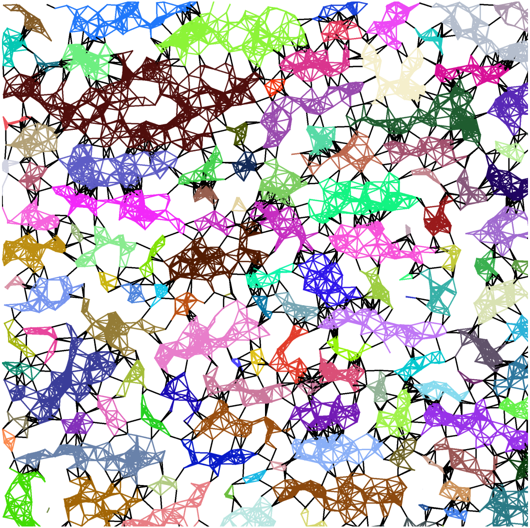
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| **3** | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



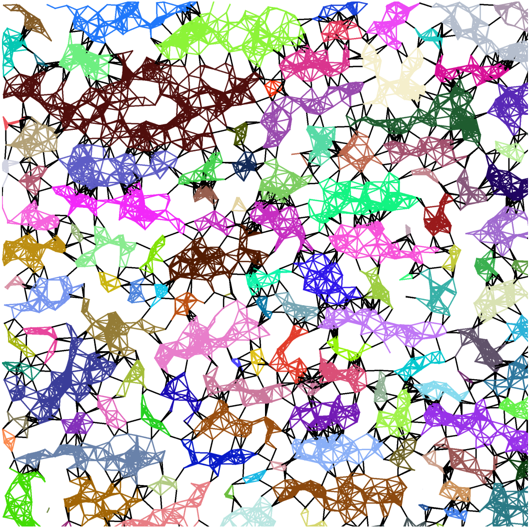
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| **4** | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



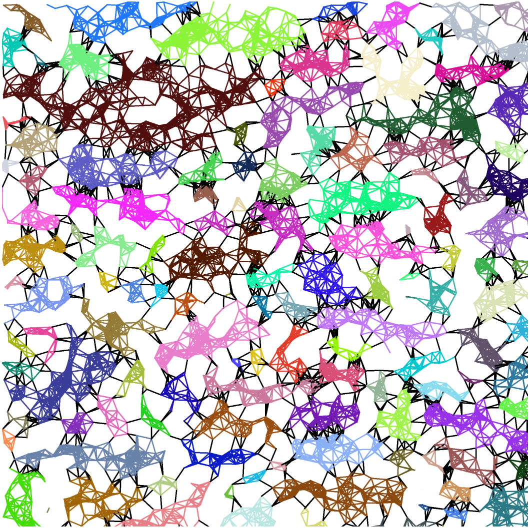
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| **5** | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



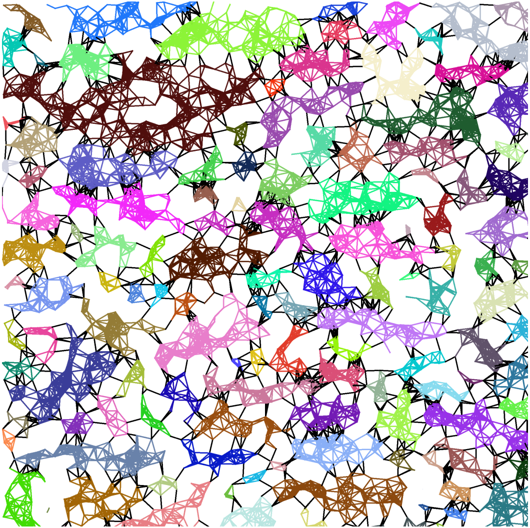
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| **6** | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



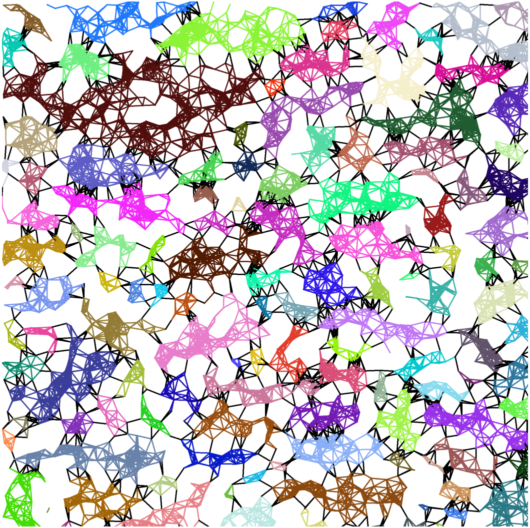
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| **7** | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

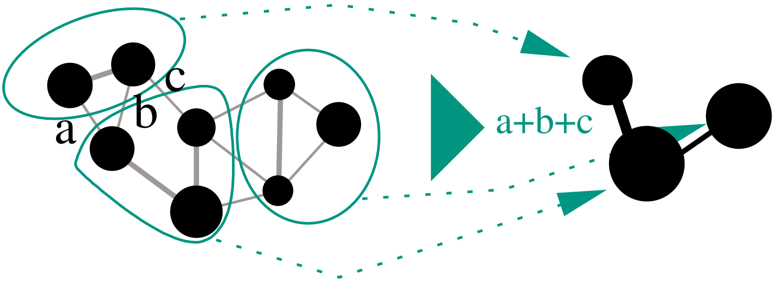
# Label Propagation



| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Basic Idea

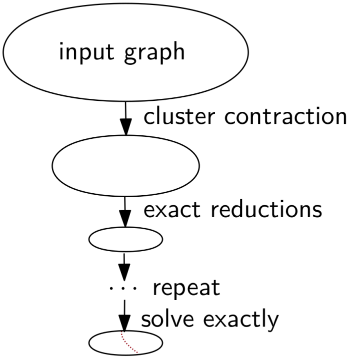
## Contraction of Clusterings




- 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

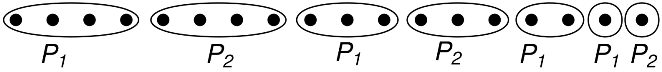
- (Inexact) Cluster reduction + Exact reductions
- Solve kernel to optimality
  using Nagamochi, Ono and Ibaraki's algorithm
  → overall linear running time, but potentially suboptimal cuts



# Parallelization

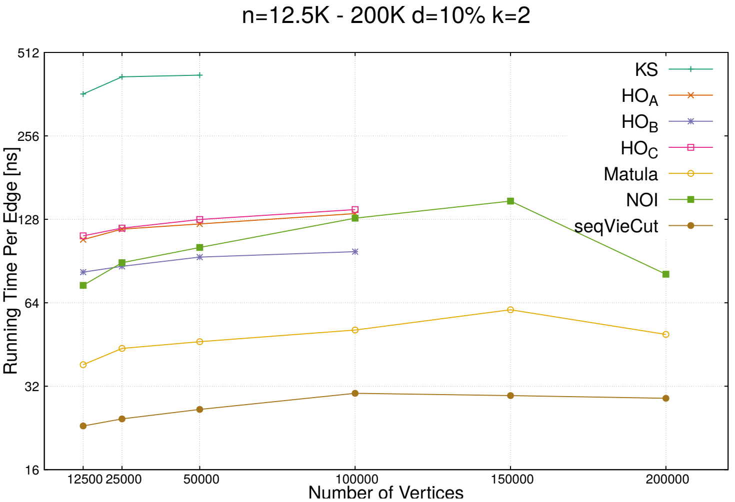
## Shared-memory with OpenMP

- Parallel label propagation



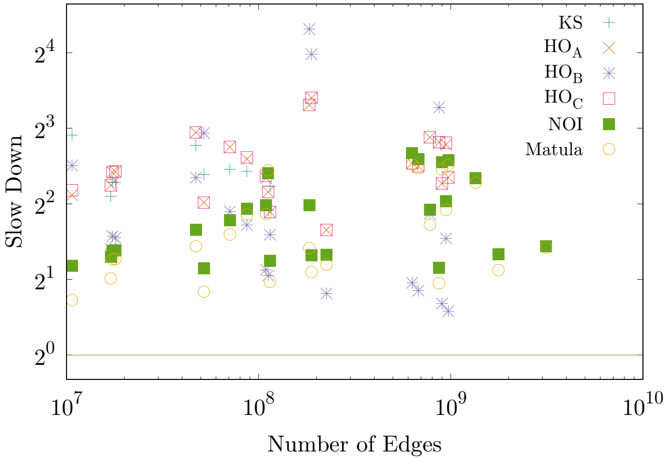

  - as brutal as: pragma openmp for and ignore conflicts on labels
- Parallel Padberg-Rinaldi:
  - check edges independently ⇝ embarassingly parallel
  - collect edges then contract
    → essentially linear time
- Parallel contraction (not here)
- run Nagamochi, Ono and Ibarakis algorithm sequentially

# Random Hyperbolic Networks



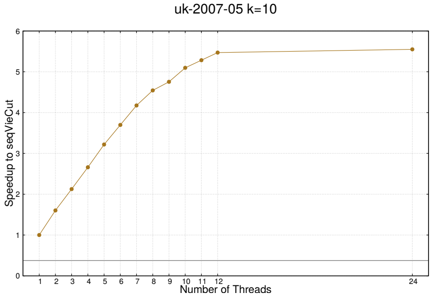

- seqVieCut optimal in 99% of runs, Matuala optimal in 69% of runs

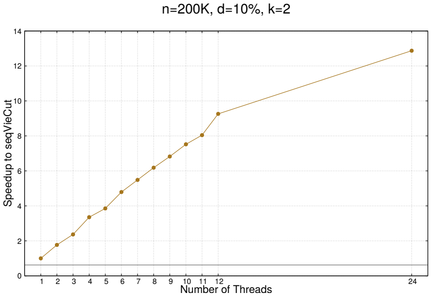
# Real-world Networks




- 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 Matuala
- NO guarantee for minimum cut, but experiments say very likely
- reductions depend on bound $\hat{\lambda}$

$\leadsto$ PLUG IN our result into exact NOI algorithm + parallelization
$\leadsto$ currently fastest exact minimum cut algorithm

# Kernelization

**Nagamochi, Ono, and Ibaraki**

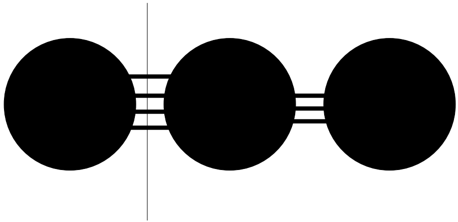
Key Idea: a spanning tree contains at least one edge from any cut

- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)

  $\rightsquigarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$

  $\rightsquigarrow$ contract all of them

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



# Kernelization

**Nagamochi, Ono, and Ibaraki**

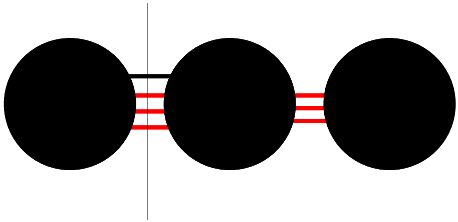
Key Idea: a spanning tree contains at least one edge from any cut

- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)

  $\rightsquigarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$

  $\rightsquigarrow$ contract all of them

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



# Kernelization

**Nagamochi, Ono, and Ibaraki**

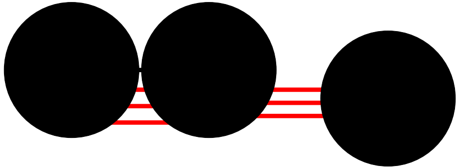
Key Idea: a spanning tree contains at least one edge from any cut

- Let $\hat{\lambda}$ be your current bound for minimum cut
- Want: smaller minimum cut
- Compute $\hat{\lambda} - 1$ maximal spanning forests (iteratively)

  $\rightsquigarrow$ edges not in forests connect vertices with connectivity $\geq \hat{\lambda}$

  $\rightsquigarrow$ contract all of them

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



# Nagamochi, Ono, Ibaraki

## Details

- $\lambda(x, y)$ capacity of minimum cut separating $x$ and $y$
- $\lambda(x, y) \geq \hat{\lambda} \rightsquigarrow \exists$ no cut separating $x$ and $y$ with capacity $\leq \hat{\lambda}$
  $\rightsquigarrow$ 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.

$$\lambda(x, y) \geq q(e) \geq \hat{\lambda}$$
$$\rightsquigarrow \text{can contract edge } e$$

$q(e)$ = # edge disjoint paths that connect $x, y$
$q(e)$ via $k$-edge-connected subgraph $\rightsquigarrow$ 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 $E_{r(v)+1}$
        $q(e) = r(v) + 1, r(v) = r(v) + 1$

$\rightsquigarrow H_i = (V, E_i)$ 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.

$\rightsquigarrow$ 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** $r[v] = i$ incidient 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 $E_{r(v)+1}$
    $q(e) = r(v) + 1, r(v) = r(v) + 1, r(u) = r(u) + 1$

$\rightsquigarrow H_i = (V, E_i)$ 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.

$\rightsquigarrow$ 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 $e = (u, v) \in E$ **do**
        insert $e$ into $E_{r(v)+1}, \ldots, E_{r(v)+c(e)}$
        $q(e) = r(v) + c(e), r(v) = r(v) + c(e)$

$\rightsquigarrow H_i = (V, E_i)$ 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.

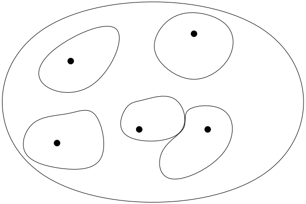
$\rightsquigarrow$ contract $e$ if $q(e) \geq \hat{\lambda}$

$c(e)$ replaces one edge by $c(e)$ edges

# Example

# Overall Parallel Algorithm

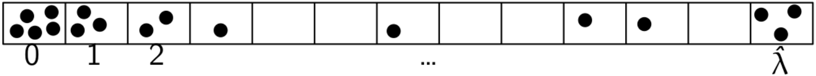
- each thread selects random start vertex
- make sure each vertex scanned by exactly one worker
- mark contractible edge in parallel union-find data structure



$\hat{\lambda} \leftarrow$ `VieCut`$(G)$, $G_C \leftarrow G$
**while** $G_C$ has more than 2 vertices
    $\hat{\lambda} \leftarrow$ Parallel CAPFOREST$(G_C, \hat{\lambda})$
    **if** no edges marked contractible
        $\hat{\lambda} \leftarrow$ CAPFOREST$(G_C, \hat{\lambda})$
    $G_C, \hat{\lambda} \leftarrow$ Parallel Graph Contract$(G_C)$
**return** $\hat{\lambda}$

# More Optimizations

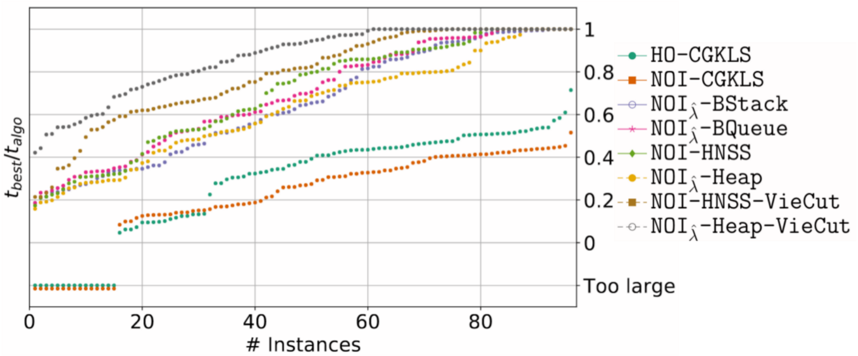
- Observation: values in PQ often higher than bound $\hat{\lambda}$
- Algorithm still correct when limiting values to $\hat{\lambda}$
- Use BucketPQ in weighted case also!




$\rightsquigarrow$ $O(1)$ for push, pop, and increaseKey
$\rightsquigarrow$ Bucket implementations make a difference

stack vs queue
breadth vs depth

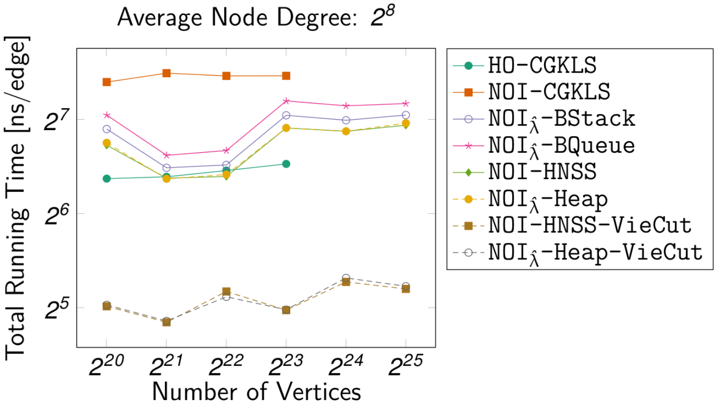
# All Graphs




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

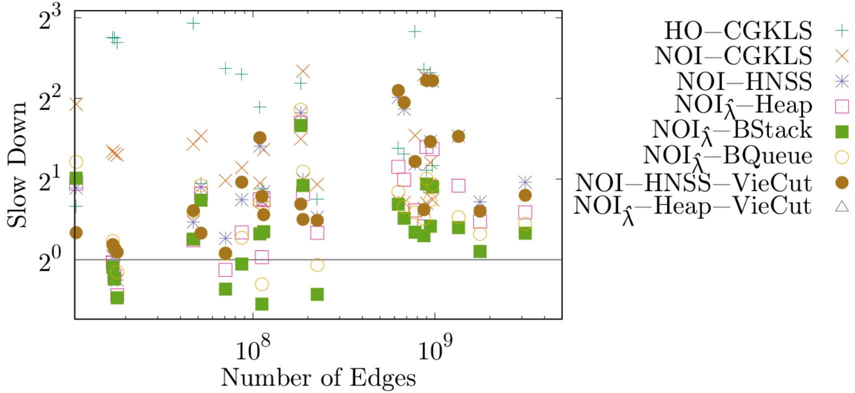
# Random Hyperbolic Graphs




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

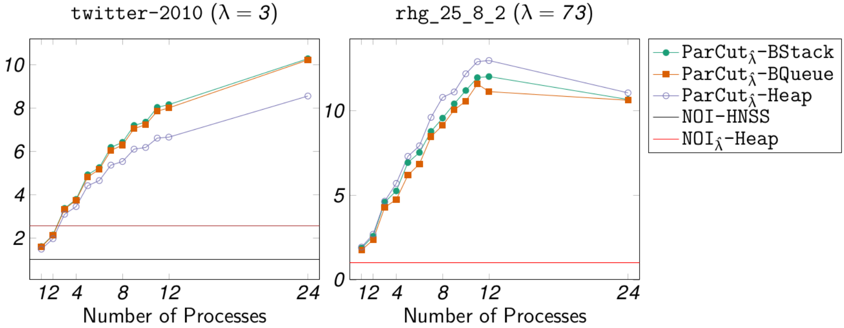
# Social and Web Graphs




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

# Scalability



# 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