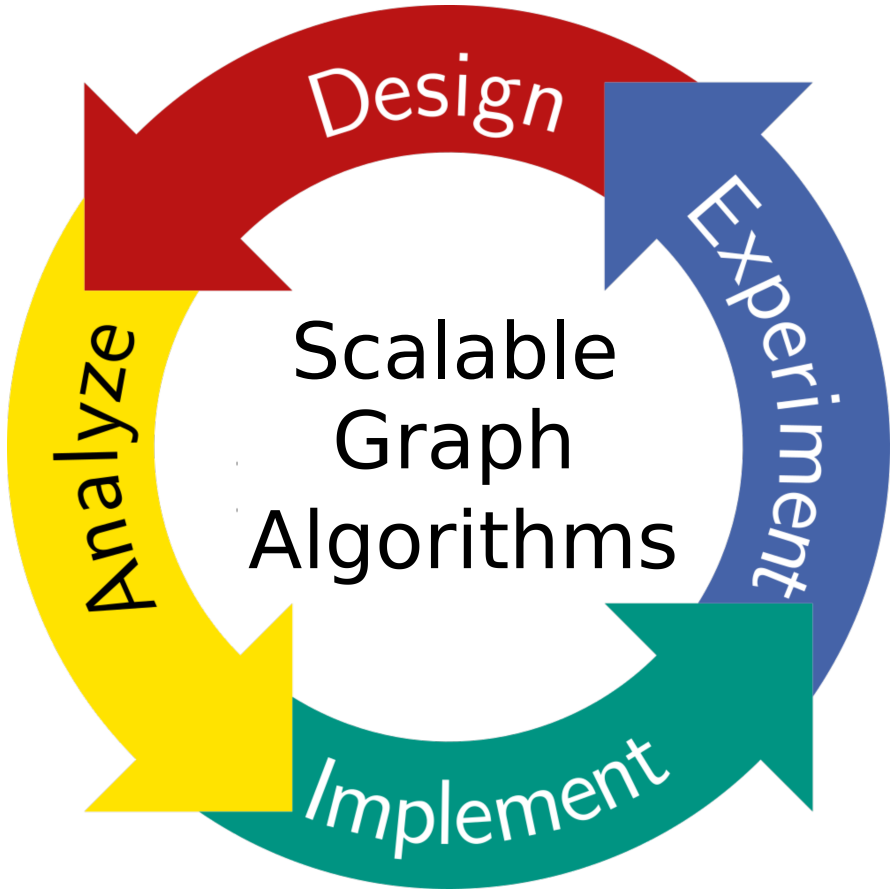

Design
Experiment
Implement
Analyze
Scalable
Graph
Algorithms

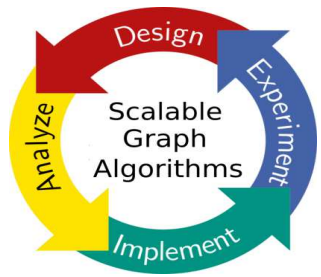
# Research Overview

Multilevel Algorithms Evolutionary Computation

Parallel Programming Kernelization

Dynamic Algorithms

shared–, distributed–, external–, internal memory




Algorithms for

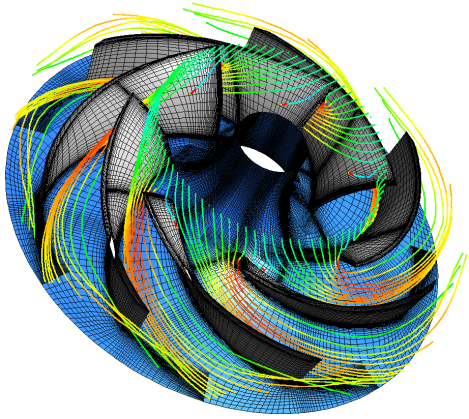
| | | | |
|---|---|---|---|
| graph partitioning | process mapping | longest paths | node separators |
| graph clustering | minimum cuts | graph drawing | dyn. reachability |
| graph generation | independent sets | (dyn.) matching | .... |

Open Source

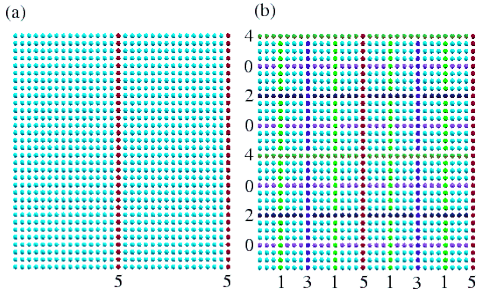
Applications

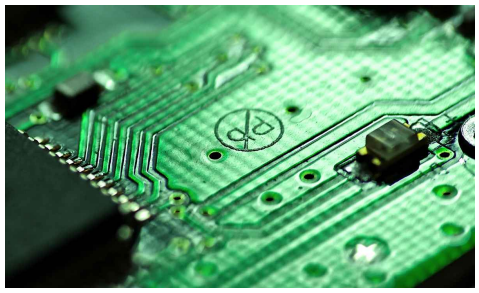
| | | |
|---|---|---|
| territory design | route planning | multiprocessor scheduling |
| large scale simulations | distributed system design | high–throughput DNA sequencing |
| quantum annealing | nuclei segmentation | .... |

# Applications



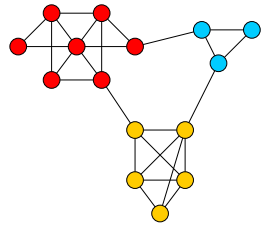
$$\mathbf{R}^{n \times n} \ni Ax = b \in \mathbf{R}^n$$



# Highlights

# Graph Clustering

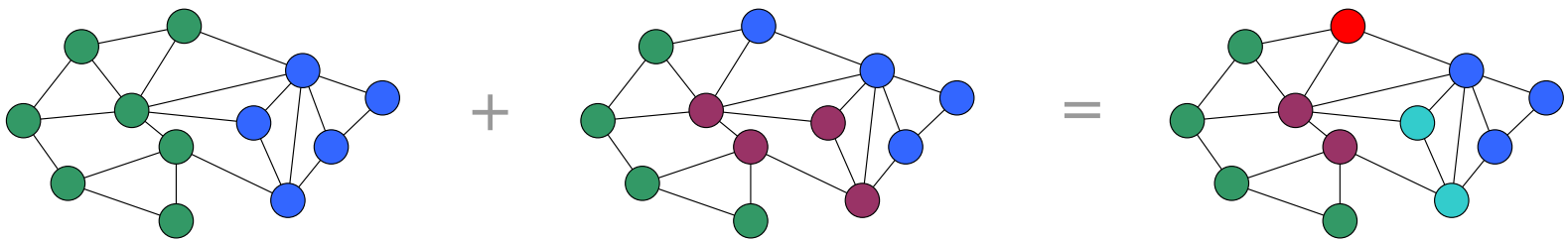


**Problem:**

- partition graph into tightly connected groups
- clustering paradigm: internally dense and externally sparse
- quality measure: modularity

**Recombination Mechanism**

- borrowed from ML: ensemble learning → maximum overlap



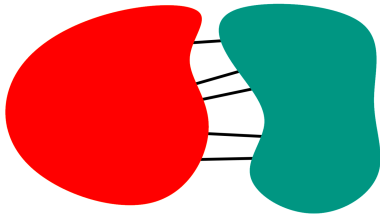
→ results in overlap clustering, contract + recluster
→ + coarse-grained parallelization

Outperform all DIMACS Challenge results
with ONE solver and in less time

# Minimum Cuts

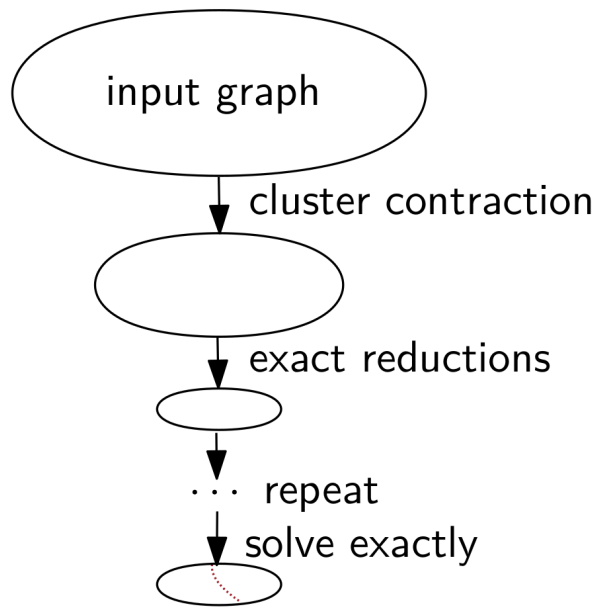
## ... and Multiterminal Cuts

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



**Contributions**:

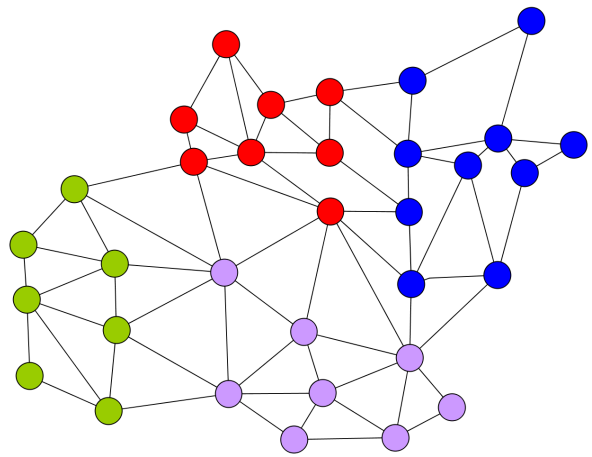
- (in)exact reductions + solve kernel to optimality
  $\rightarrow$ linear running time, but potentially suboptimal cuts
- NO guarantee, but experiments say likely opt
- reductions depend on bound $\hat{\lambda}$
  $\rightsquigarrow$ use $\hat{\lambda}$ in exact (parallel) NOI algorithm
  $\rightsquigarrow$ fastest exact minimum cut algorithm
  $\approx$ order of magnitude
- multiterminal cut:
  first practical solver for large inputs



# $\epsilon$-Balanced Graph Partitioning

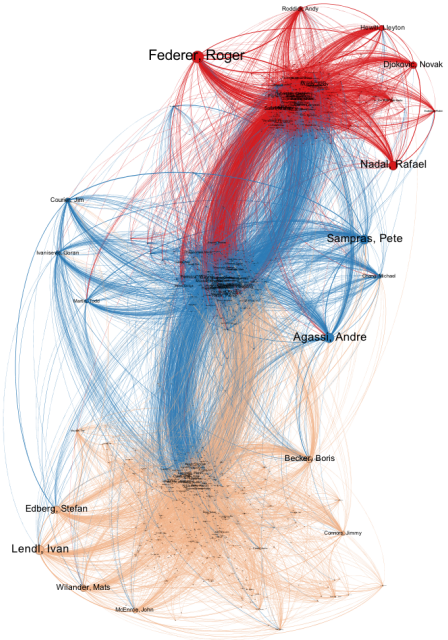
Partition graph $G = (V, E, c : V \to \mathbf{R}_{>0}, \omega : E \to \mathbf{R}_{>0})$
into $k$ disjoint blocks s.t.

- total node weight of each block $\leq \frac{1+\epsilon}{k}$ total node weight
- total weight of cut edges as small as possible



**Applications:**
graph processing frameworks, parallel sparse matrix vector mult., ...

# The Common Parallel Approach

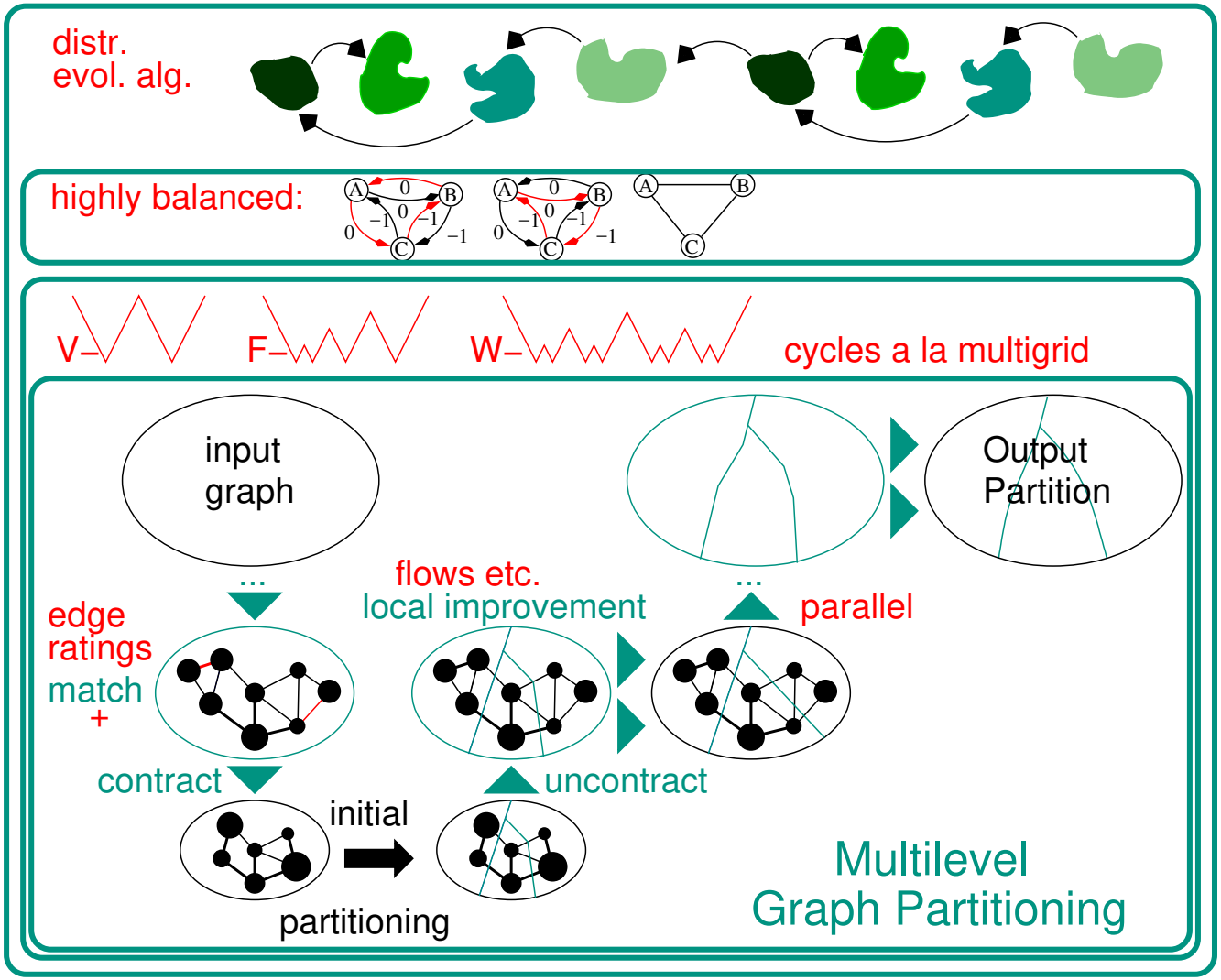



- Mesh partitioned via dual graph
  1. Each volume (data, calculation) represented by a vertex (+edges)
  2. Interdependencies represented by edges
- All PE's get same amount of work
- Communication is expensive

**Graph Partitioning Problem:**
Partition a graph into (almost) equally sized blocks, such that the number of edges connecting vertices from different blocks is minimal.

# Karlsruhe High Quality Partitioning

https://github.com/KaHIP/KaHIP

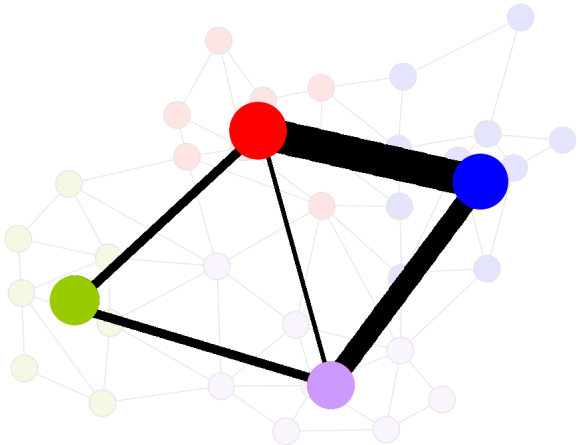



THE solver, when it comes to high solution quality.

# Karlsruhe High Quality Partitioning

- pre 2014:
  - multi-level, localized and flow-based local search
  - perfectly balanced,
  - distributed evolutionary algorithms
  - specialized algorithms for road networks
- social networks
- distributed memory
- semi-external memory
- hypergraph partitioning
- node separators
- (hyper)DAG partitioning
- shared-memory parallel
- ILP-based
- node ordering

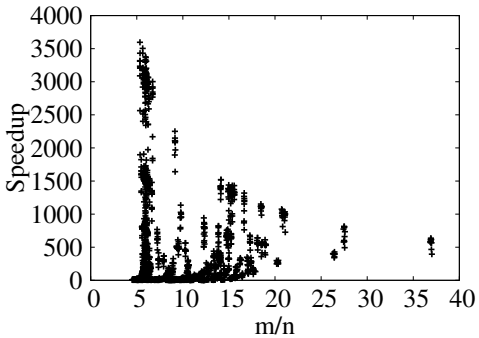
# Process Mapping



$\rightarrow ?$


**Exploiting Assumptions:**

- communication matrix $\mathcal{C}$ is sparse $\rightarrow$ graph $G_{\mathcal{C}}$
- compute system is hierarchically structured




**Yields:**

- three orders of magnitude faster local search
- 50% improved mapping quality over state-of-the-art

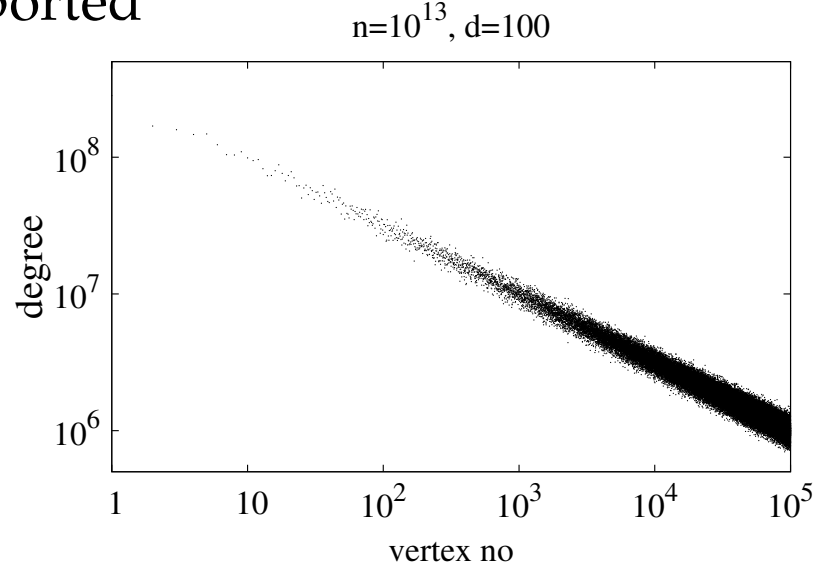
# Graph Generation

**Ideas:**

- replace randomness by pseudo-randomness, e.g. hashing
- replace array accesses by recomputation
  $\rightarrow$ embarrassingly parallel algorithm without communication

**Results:**

- generate Peta-edge BA graph in $<$ one hour ($10^{15}$ edges, 16K cores)
- 20 000 times larger than previously reported
- 16x faster than RMAT ($m = 50 \cdot 10^9$)
- communication efficient algorithms




**more work:**

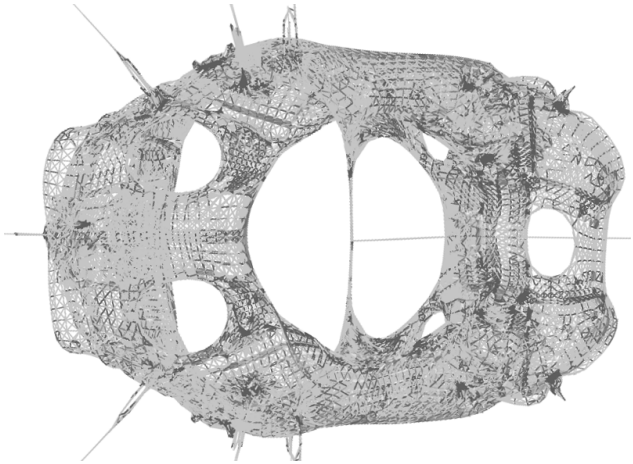
- Barabasi-Albert
- more models:
  ER, G($n$,$m$), G($n$,$p$),
  RHG, RGG, DEL

# Graph Drawing

$$G = (V, E, d)$$
$$d : E \to \mathbf{R}$$

$\Rightarrow$

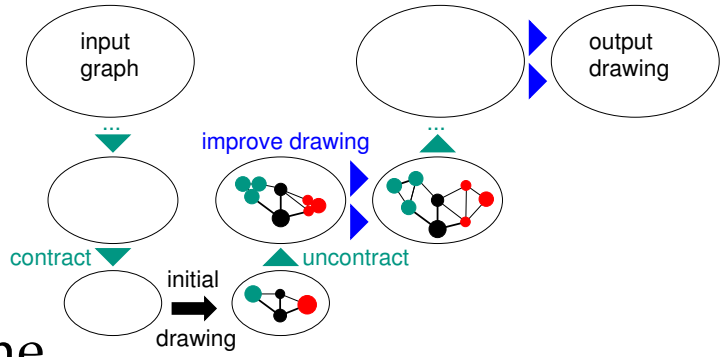
$$x : V \to \mathbf{R}^2$$



**Maximal Entropy Stress Model:**

$$\max H(x) := \sum_{\{u,v\} \notin E} \ln ||x_u - x_v||$$

$$\text{subject to } ||x_u - x_v|| = d_{uv}, \{u, v\} \in E$$




**Contributions:**

- multilevel integration of iterative scheme
- approximate long-range forces, by cluster contraction
- employ parallelism

Draw graphs with millions of vertices in seconds.

# Support Vector Machines

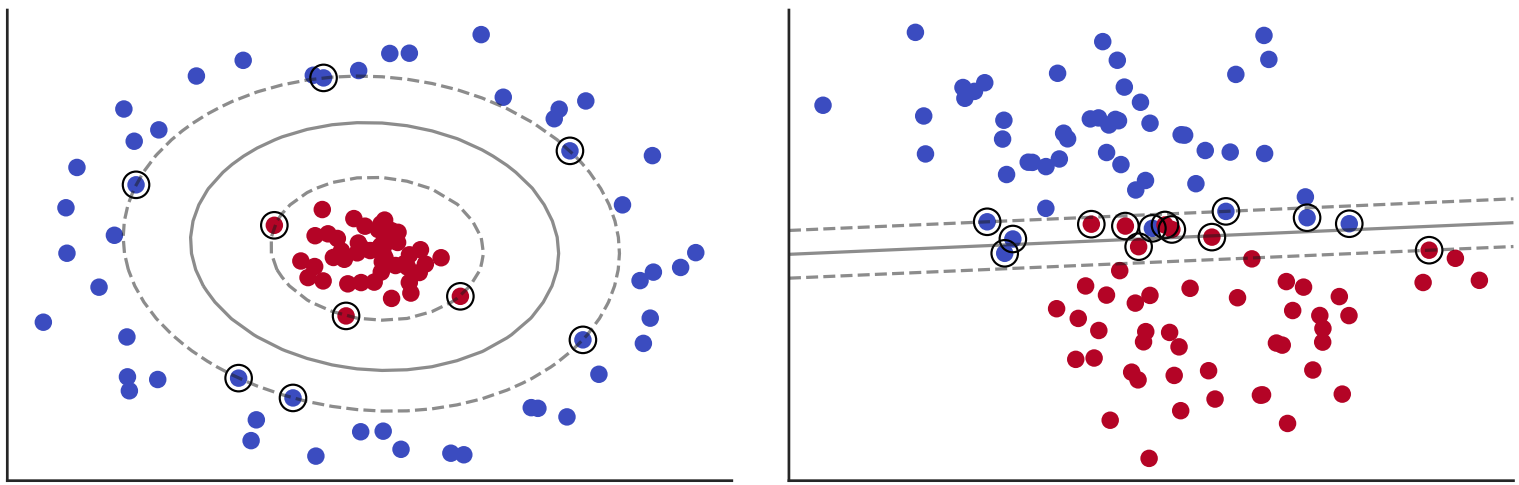
- **Binary Classification Problem**:

  Train classifier on $n$ labeled data points $(x_i \in \mathbf{R}^d, y_i \in \{-1, 1\})$
  **Goal:** Assign label $y_{n+1}$ to *new* data points $x_{n+1}$

- **SVM Optimization Problem:**

$$\text{min.} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(w \cdot \phi(x_i) - b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$$



# Support Vector Machines

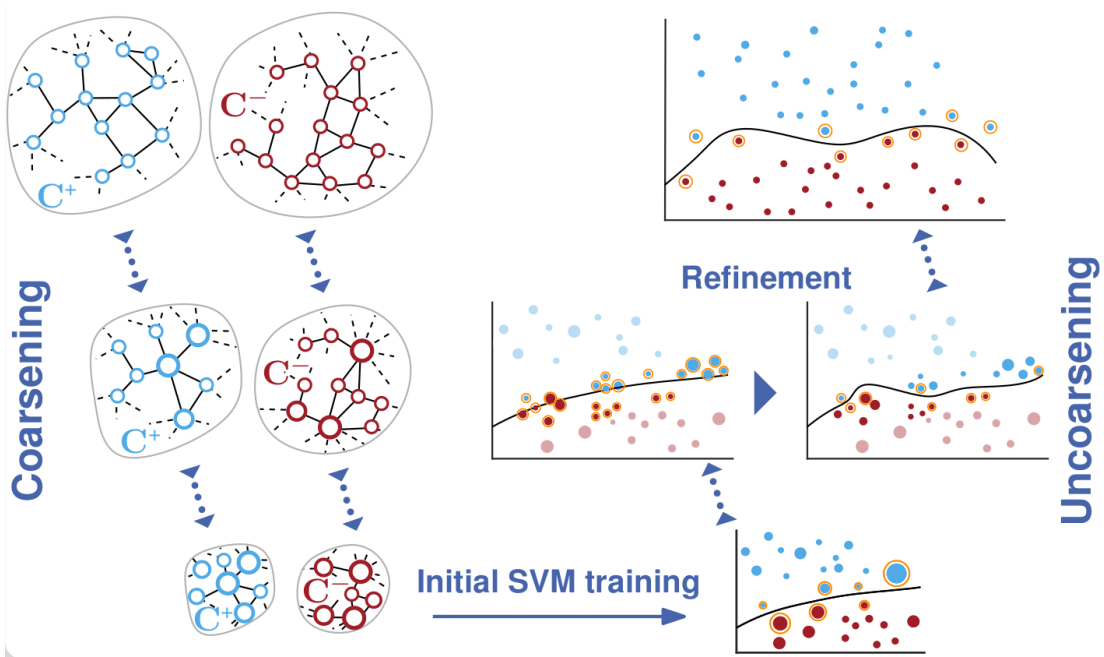
- **Binary Classification Problem**:

  Train classifier on $n$ labeled data points $(x_i \in \mathbf{R}^d, y_i \in \{-1, 1\})$
  **Goal:** Assign label $y_{n+1}$ to *new* data points $x_{n+1}$

- **SVM Optimization Problem:**

$$\text{min.} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t.} \quad y_i(w \cdot \phi(x_i) - b) \geq 1 - \xi_i, \quad \xi_i \geq 0.$$



# Support Vector Machines

- **Binary Classification Problem**:

  Train classifier on $n$ labeled data points $(x_i \in \mathbf{R}^d, y_i \in \{-1, 1\})$
  **Goal:** Assign label $y_{n+1}$ to *new* data points $x_{n+1}$

- **SVM Optimization Problem:**

$$\begin{aligned} \text{min.} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i(w \cdot \phi(x_i) - b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \end{aligned}$$

multilevel SVM using label propagation
$\rightarrow$ comparable classification quality
$\rightarrow$ up to **two** orders of magnitude faster training

currently working on parallelization

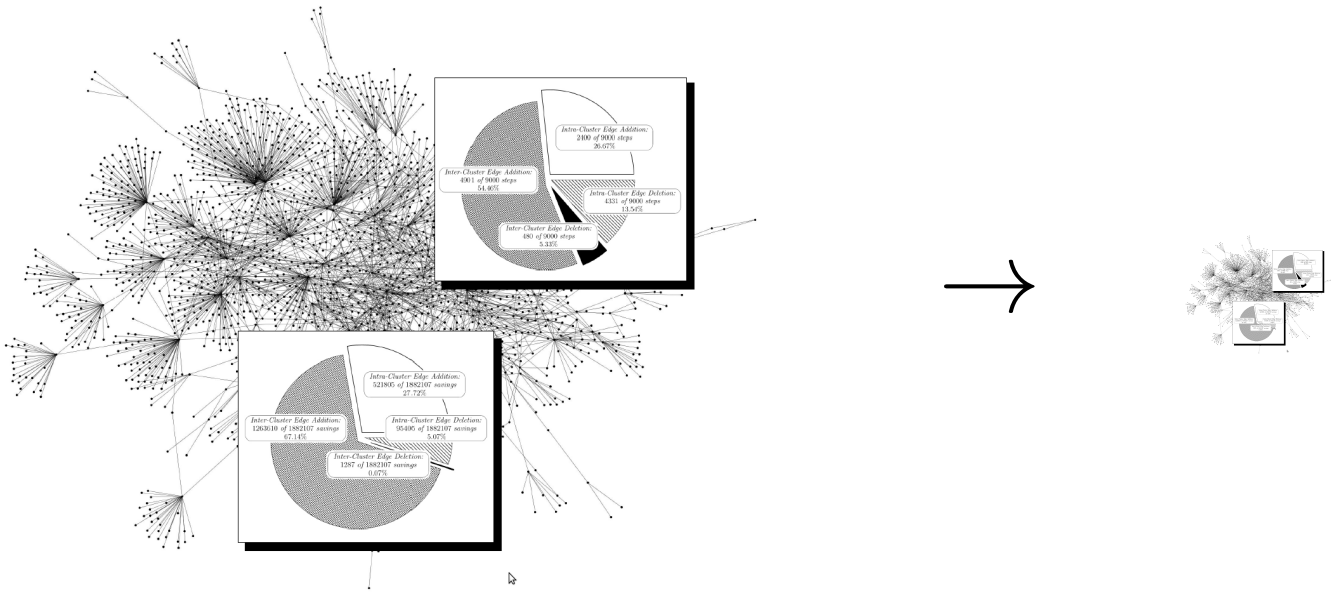
# Kernelization

## Example Independent Sets

find subset $S \subseteq V$ such that there are no adjacent nodes in S

**Reductions:**

rules to decrease graph size, while maintaining optimality




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

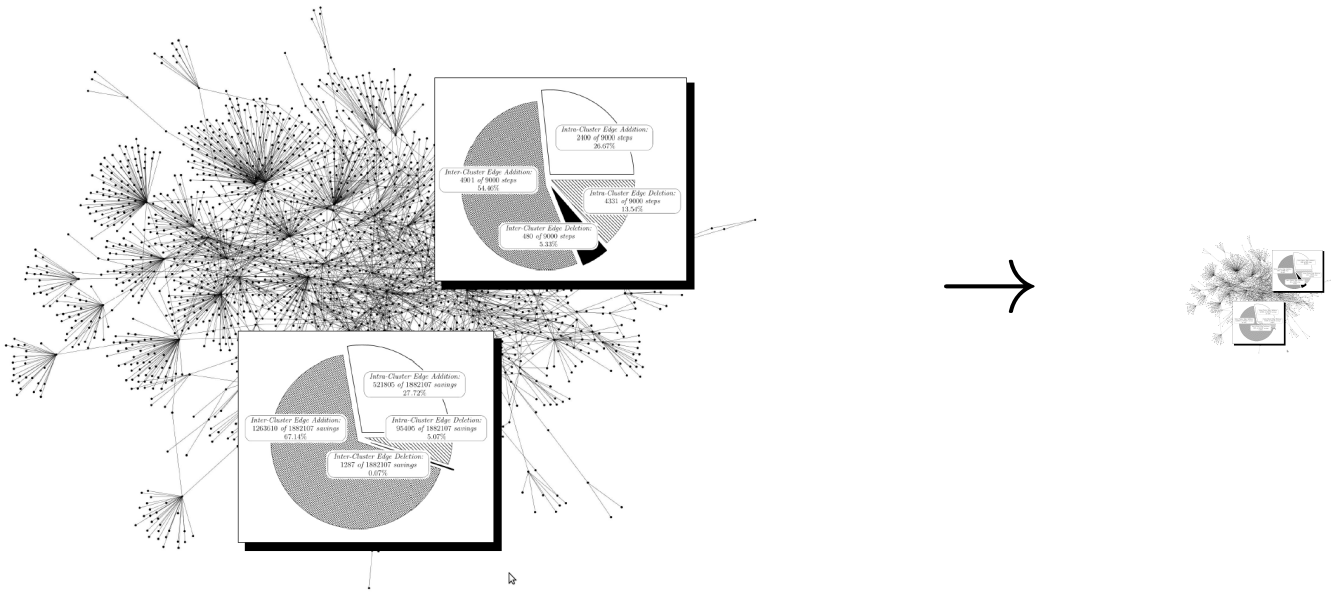
# Kernelization

## Example Independent Sets

find subset $S \subseteq V$ such that there are no adjacent nodes in S

**Reductions:**

rules to decrease graph size, while maintaining optimality



solve problem on problem kernel (using a heuristic)

$\rightarrow$ obtain solution on input graph quickly

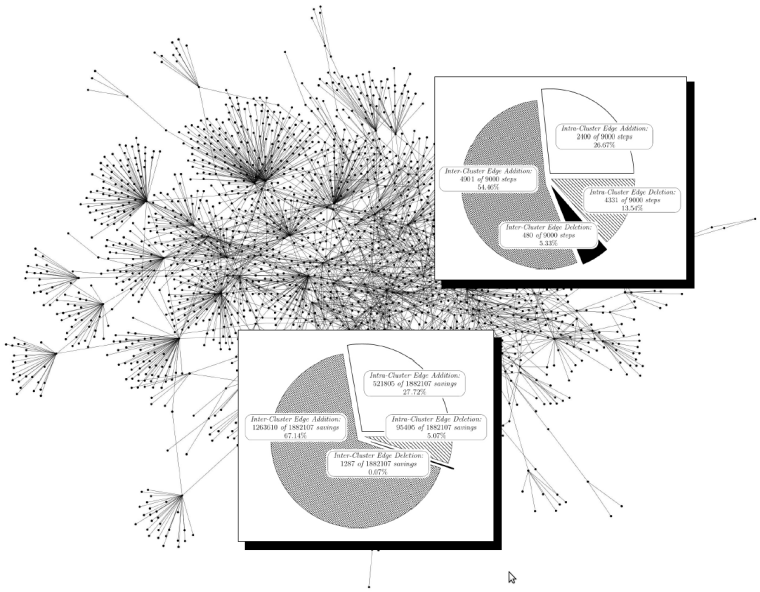
# Kernelization

## Example Independent Sets

find subset $S \subseteq V$ such that there are no adjacent nodes in S

**Reductions:**

rules to decrease graph size, while maintaining optimality



solve problem on problem kernel
→ obtain solution on input graph

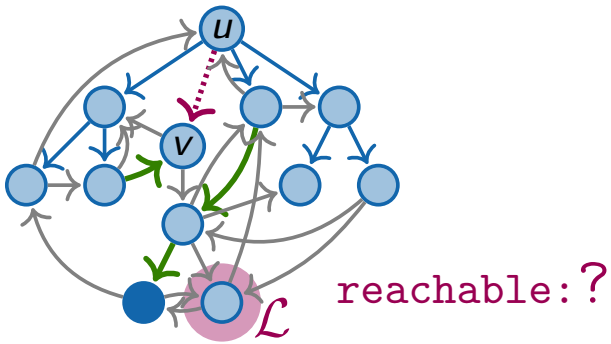
**Independent Sets**

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

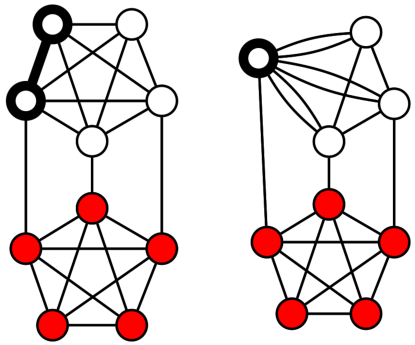
State-of-the-art for (weighted) independent sets

# Dynamic Graph Algorithms

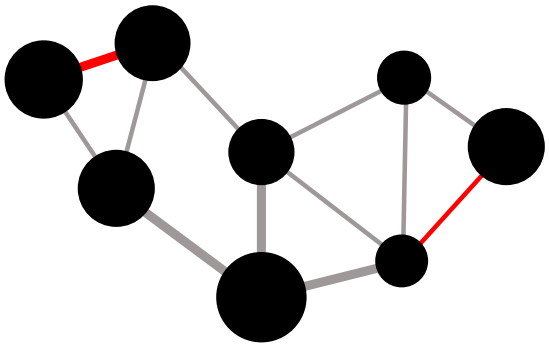
dynamic reachability



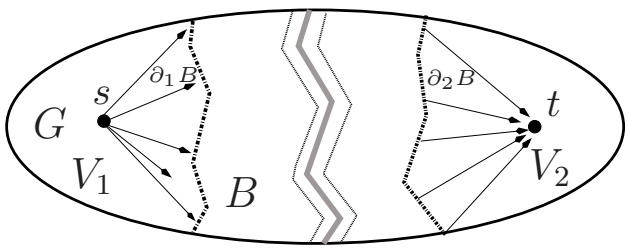

dynamic mincut



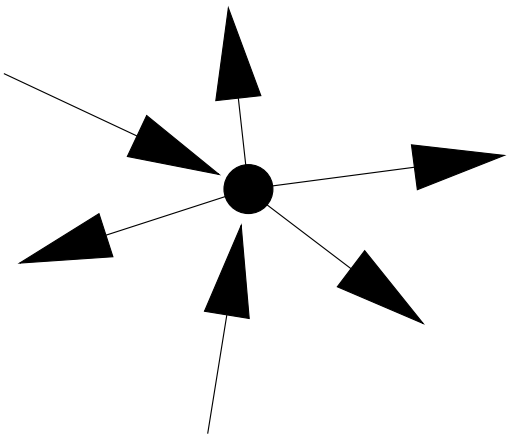
dynamic matching



dynamic maxflow




dynamic $\delta$-orientation

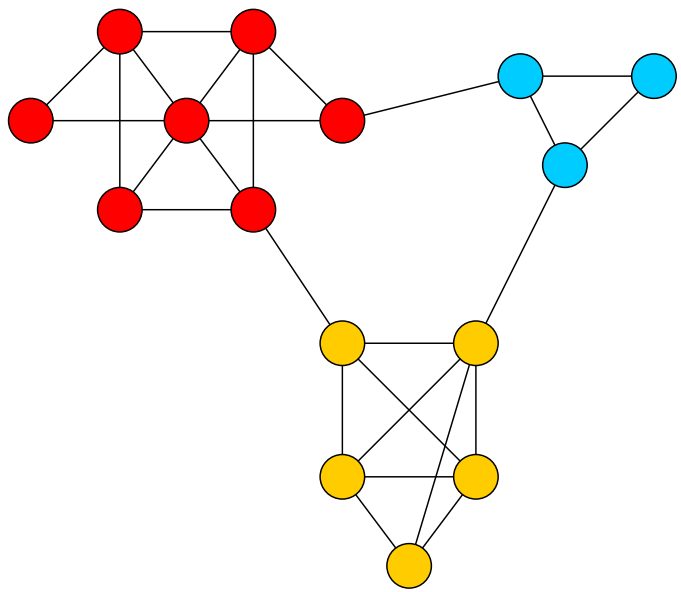


# In Depth Example
## Graph Clustering

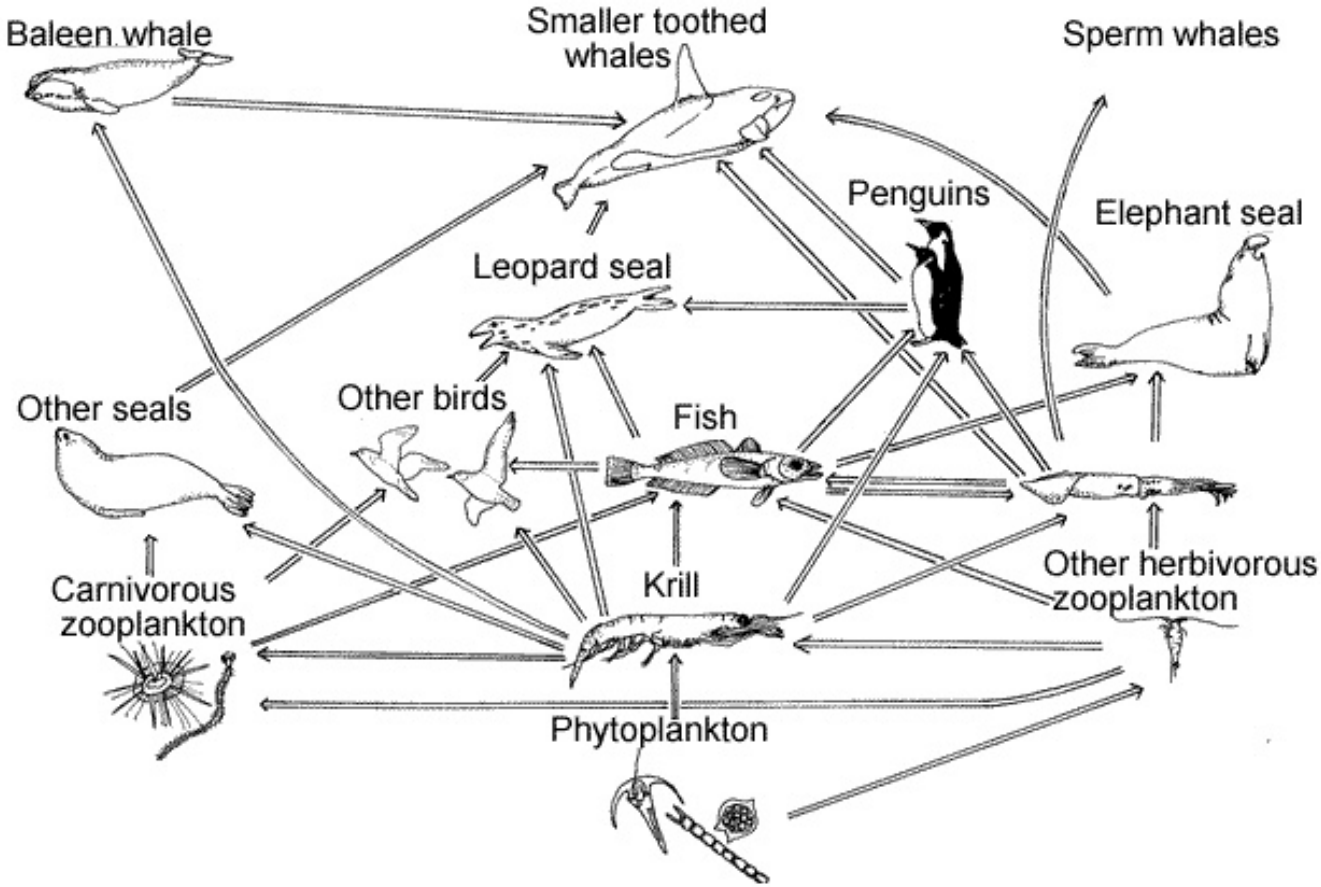
# Graph Clustering

## Basics

- partitioning of a graph into tightly connected groups
- clustering paradigm: internally dense and externally sparse
- quality measures: many, most popular: modularity
- modularity maximization is NP-hard $\Rightarrow$ use heuristics



# Clustering Applications

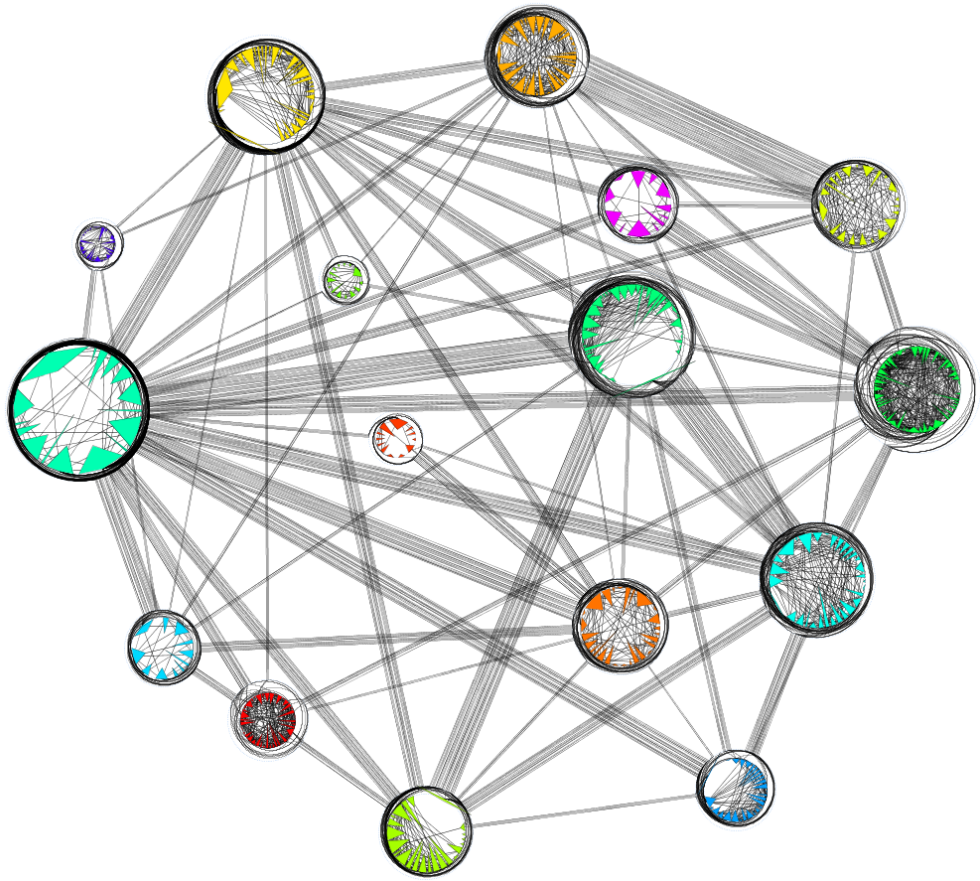



A simplified ecosystem: the antarctic food web

(source: Antarctica)

cluster $\approx$ self-sustaining / indivisible subsystem

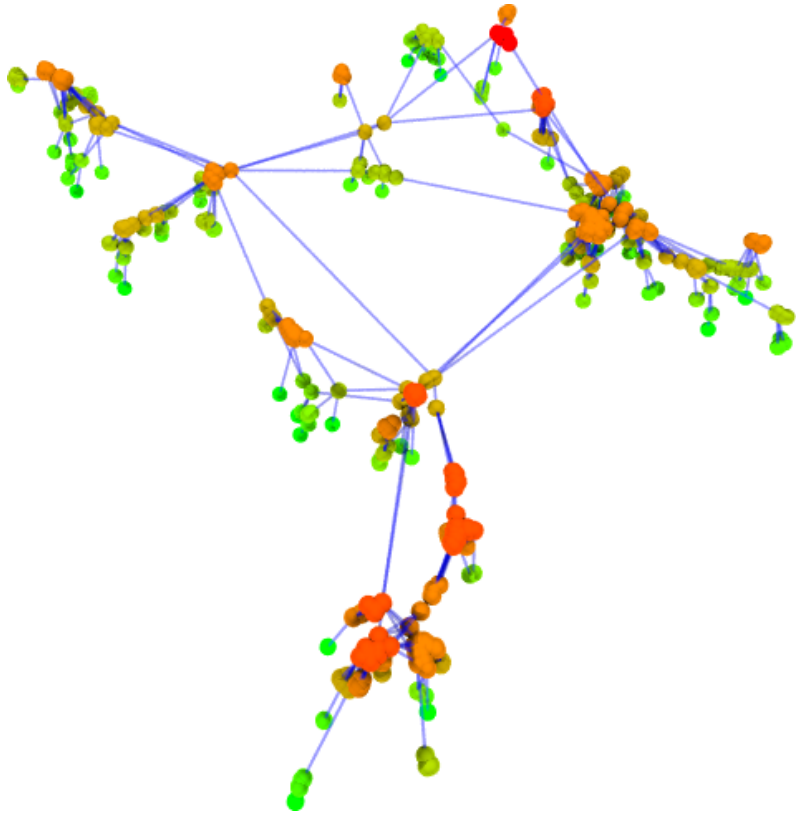
# Clustering Applications



Company-internal email traffic, groups are departments

(source: VISONE)

# Clustering Applications

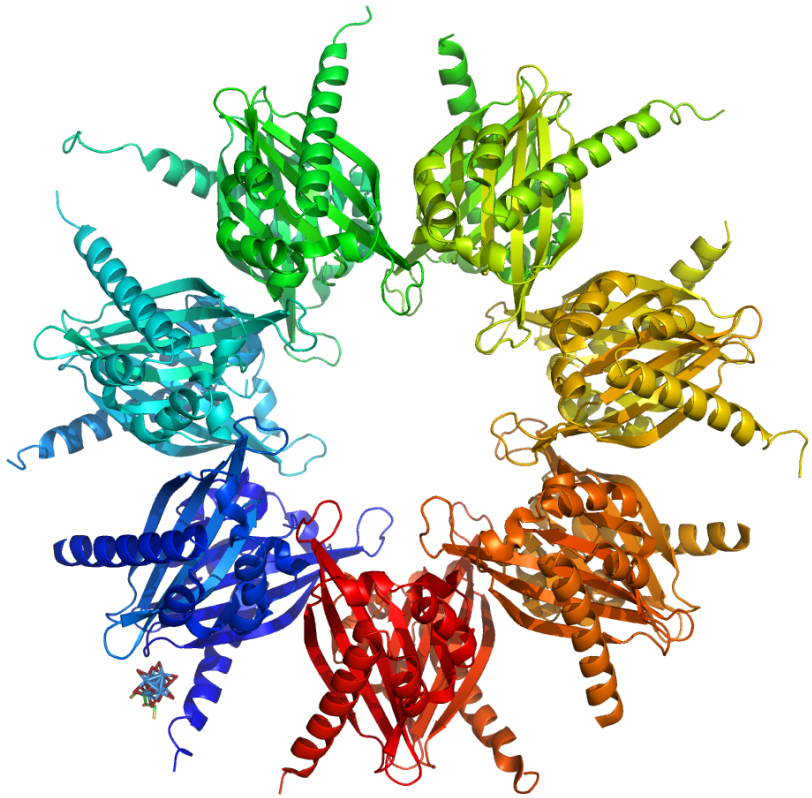


Excerpt of the network of Amazon recommendations, around *"VW Beetle Repairs"*

(source: [Gaertler'07])

cluster $\approx$ customer profile

# Clustering Applications

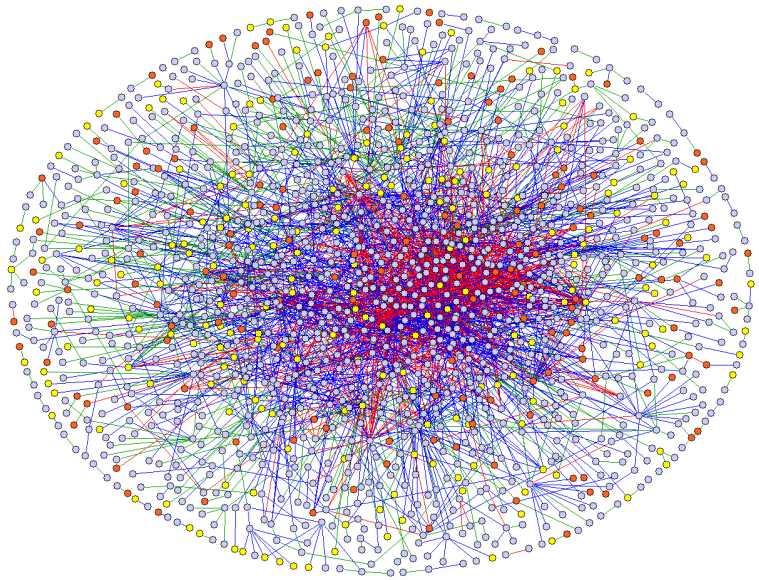


molecular structure of a protein

($Ca^{2+}$ /Calmodulin-dependent kinase II (CaMKII)
source: protein database www.rcsb.org)

cluster $\approx$ functional unit (*domain*) of a protein

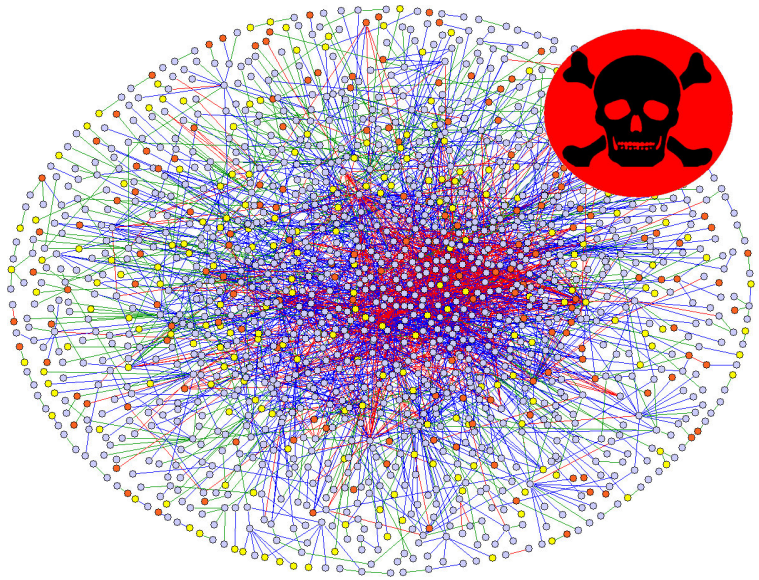
# Clustering Applications



protein interactions

(source: Max-Delbrück-Centre for molecular medicine, www.mdc-berlin.de)

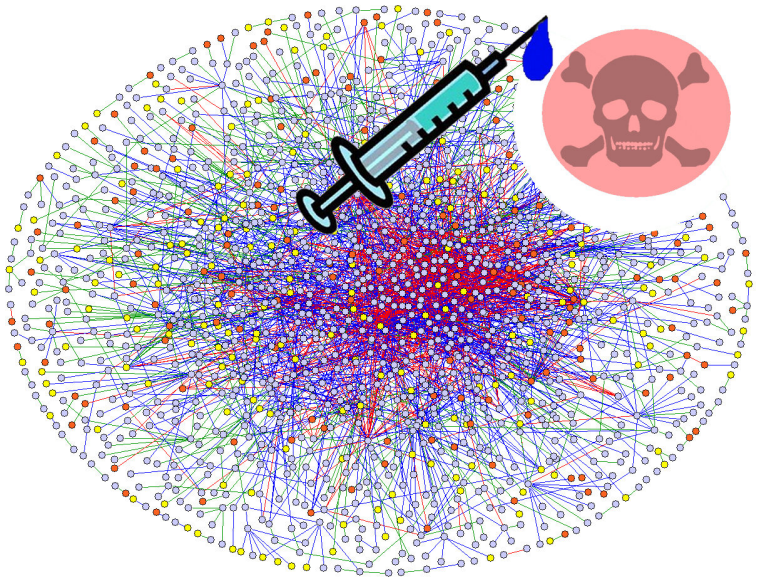
# Clustering Applications



protein interactions

(source: Max-Delbrück-Centre for molecular medicine, www.mdc-berlin.de)

# Clustering Applications

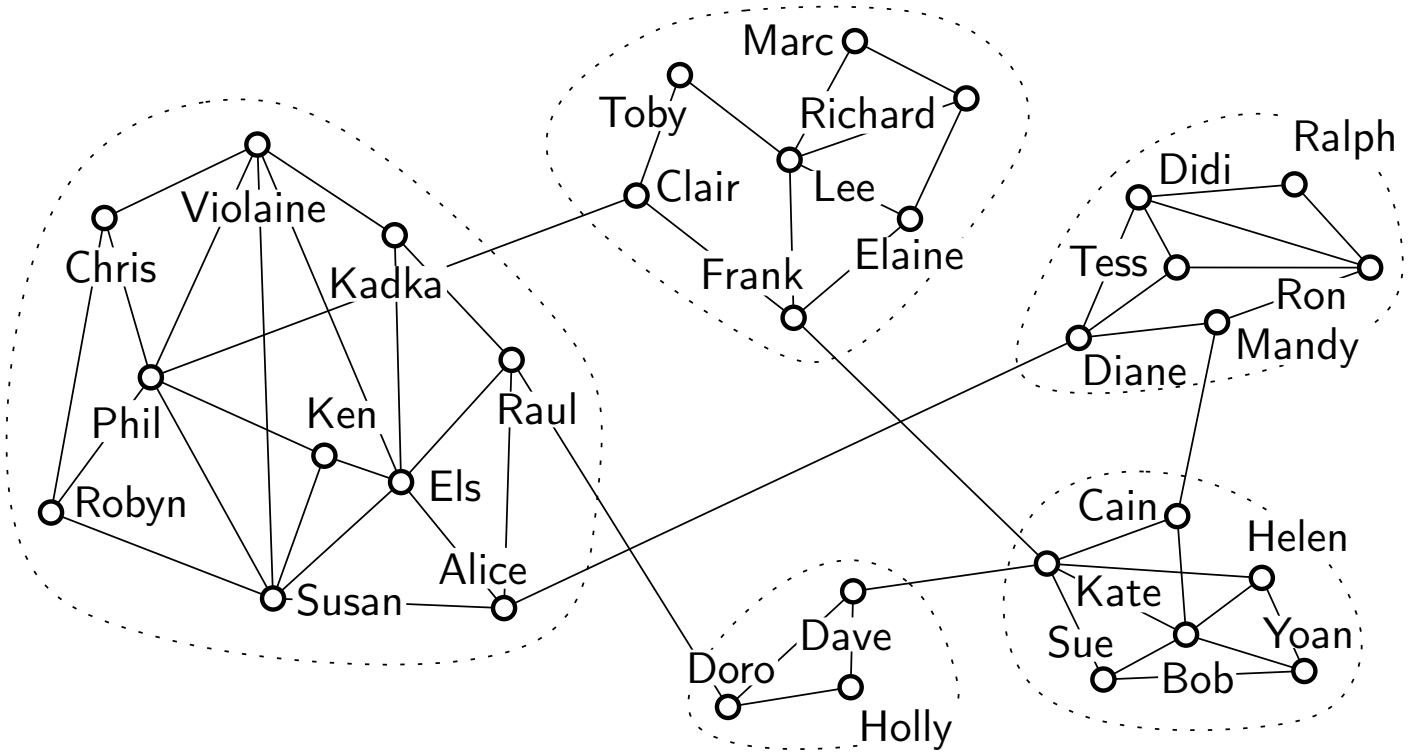


protein interactions

(source: Max-Delbrück-Centre for molecular medicine, www.mdc-berlin.de)

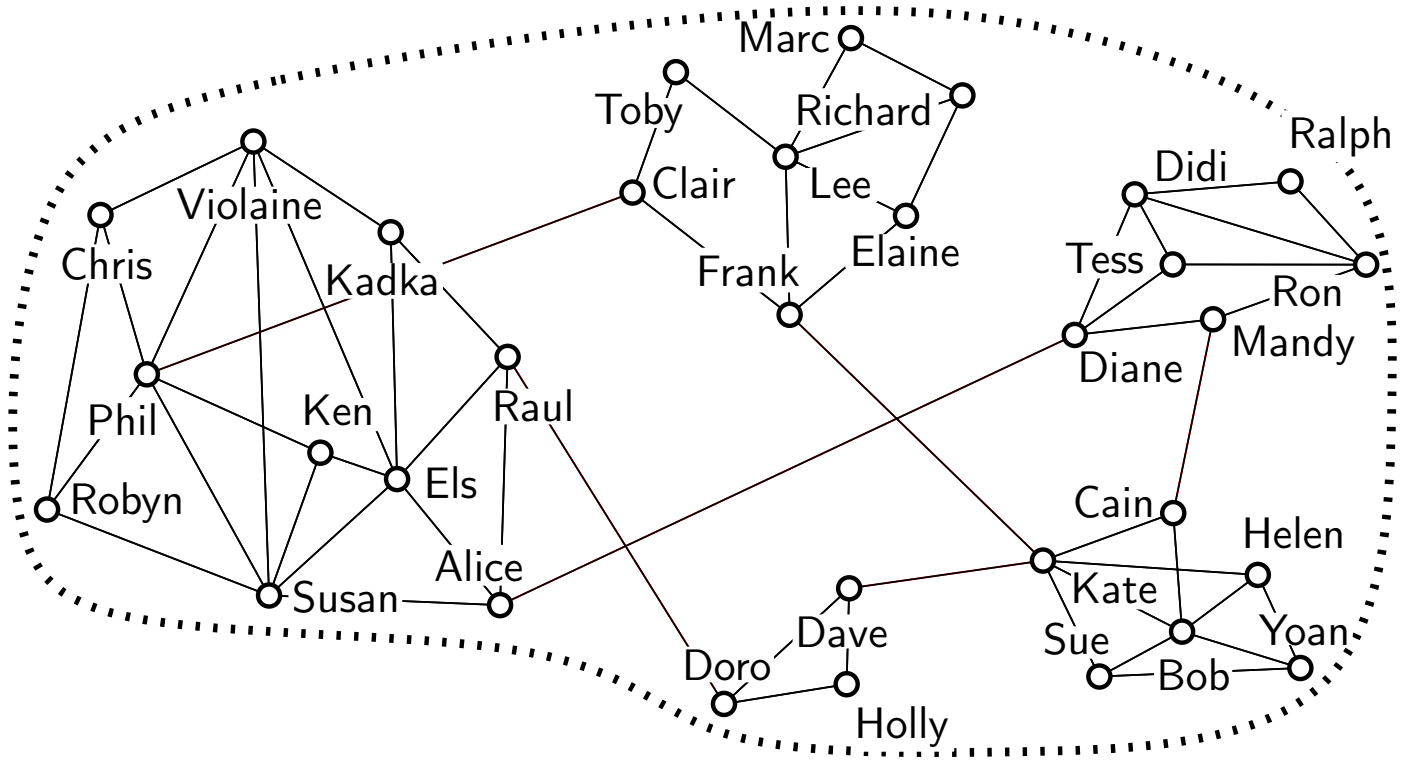
cluster $\approx$ isolatable seat of disease

# Motivation for Modularity




- $cov(\mathcal{C}) = \frac{\text{\# intra-cluster edges}}{\text{\# edges}} \approx 0.9$

# Motivation for Modularity




- $cov(\mathcal{C}) = \frac{\text{\# intra-cluster edges}}{\text{\# edges}} \approx 0.9$
- only one cluster $\Rightarrow cov(\mathcal{C}') = 1.0$

# A Promising Remedy

**Modularity [Girvan and Newman'04]**

- subtract from coverage the **expected** value $\rightarrow$ useful measure

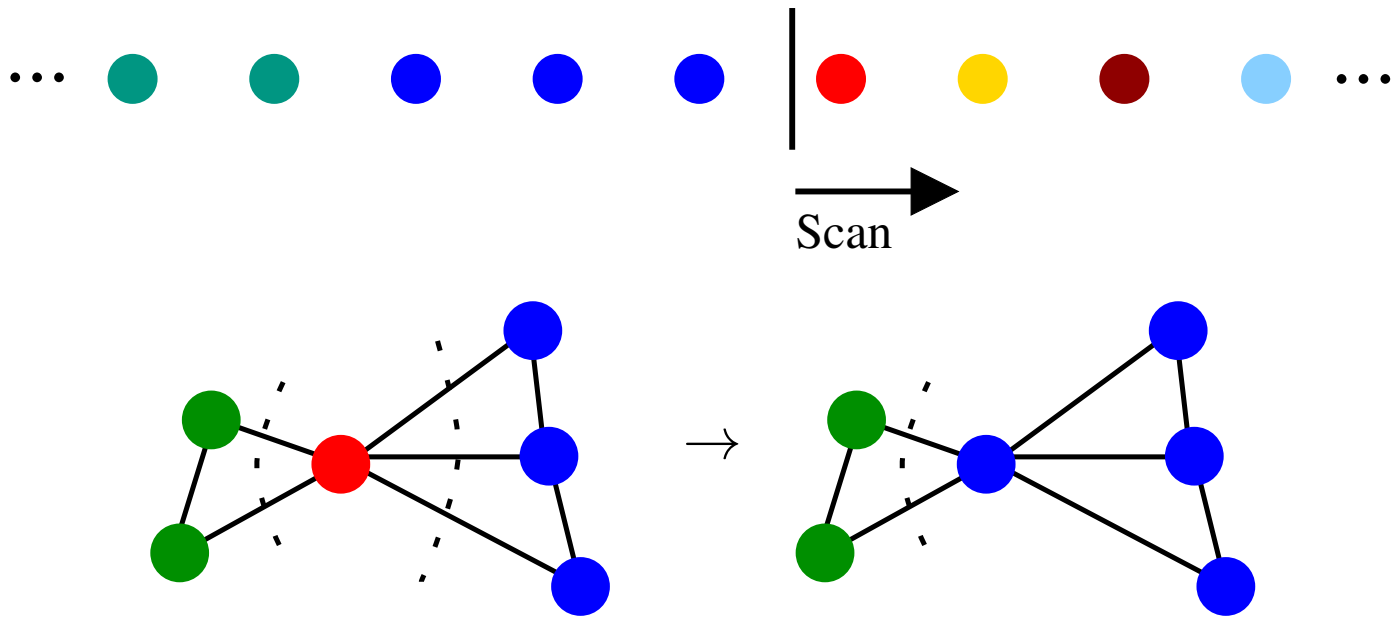
$$\begin{aligned} \text{mod}(\mathcal{C}) := & \quad cov(\mathcal{C}) & - \mathbb{E}[cov(\mathcal{C})] \\ = & \quad \frac{m(\mathcal{C})}{m} & - \frac{1}{4m^2} \sum_{C \in \mathcal{C}} \left( \sum_{v \in C} \deg(v) \right)^2 \end{aligned}$$

- probability model:
  1. random graphs, keep the same clustering
  2. goal: keep *expected* node degrees, randomly throw in edges
  3. edge set multi-set

# Louvain Method

[Blondel et al.'08]

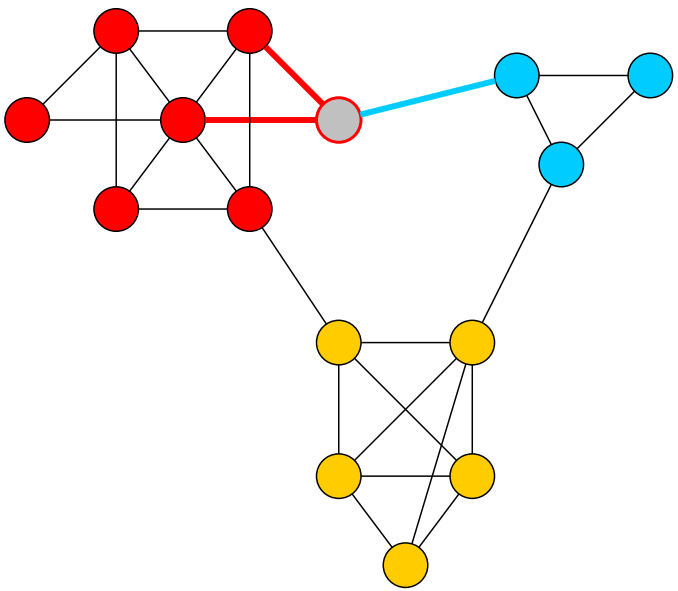
- modularity-based local movement + multi-level clustering
  - start with singletons
  - traverse nodes in random order
  - move node to neighboring cluster yielding highest *mod* increase

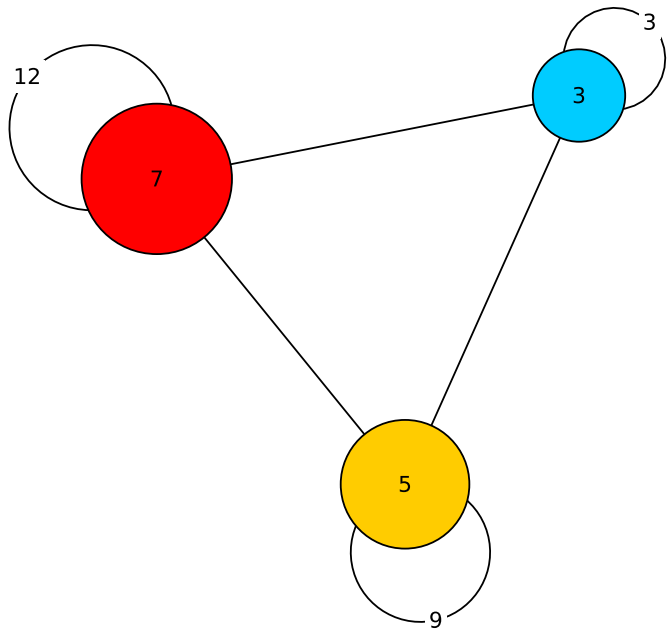


# Graph Clustering

## Selected Methods: Louvain Method

- multilevel modularity maximizing approach
- works in alternating phases: local search followed by contraction
- fast and non-deterministic

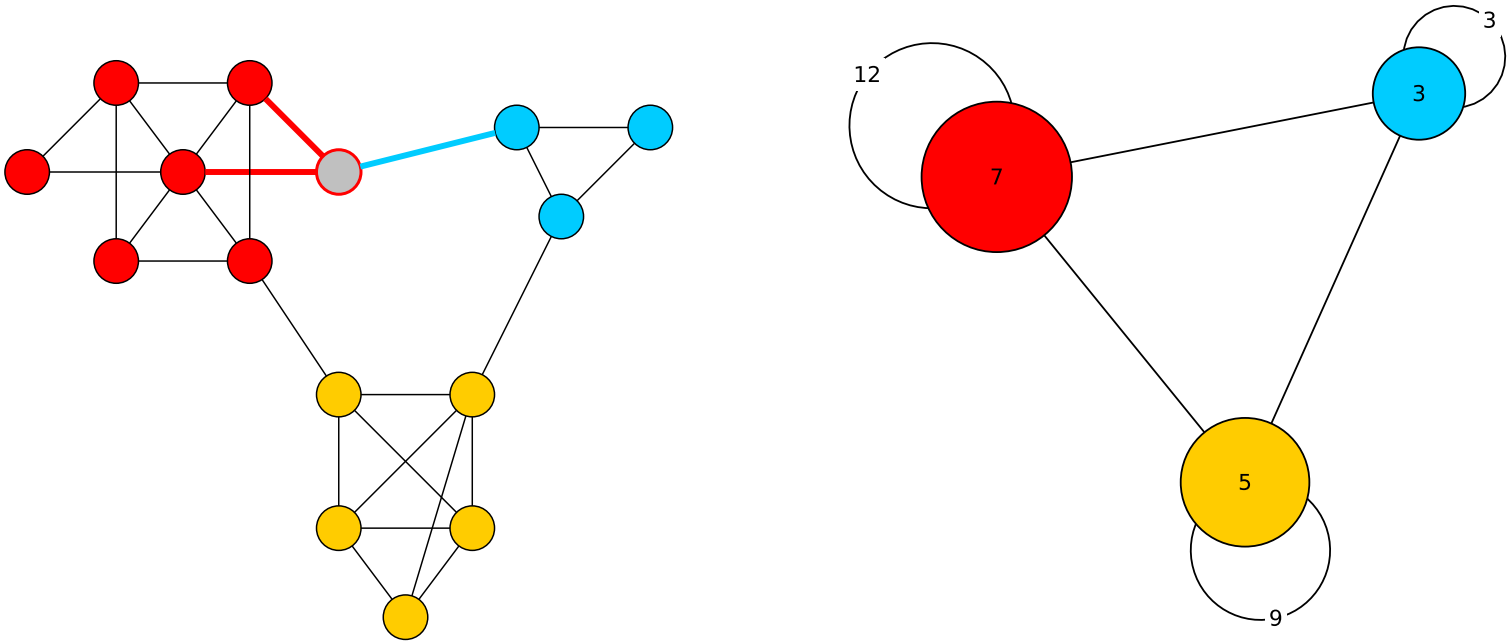




# Graph Clustering

## Selected Methods: Louvain Method

- multilevel modularity maximizing approach
- works in alternating phases: local search followed by contraction
- fast and non-deterministic

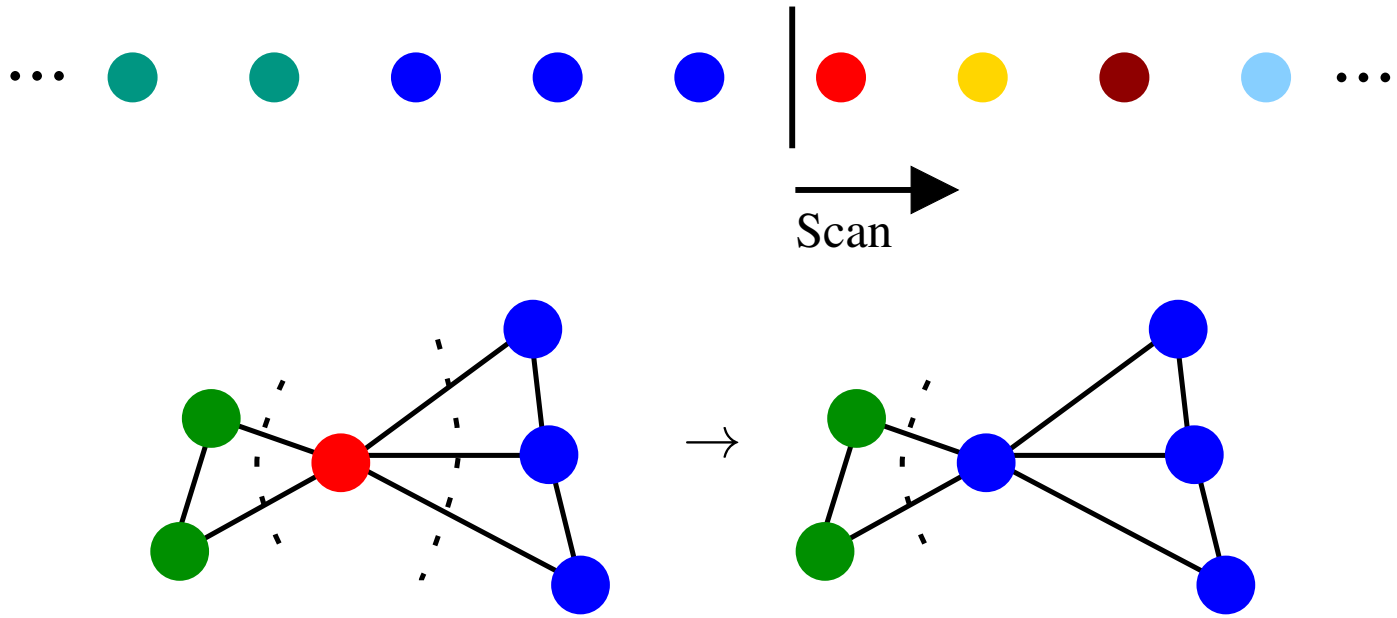



- uncoarsening: transfer clustering + local movement every level

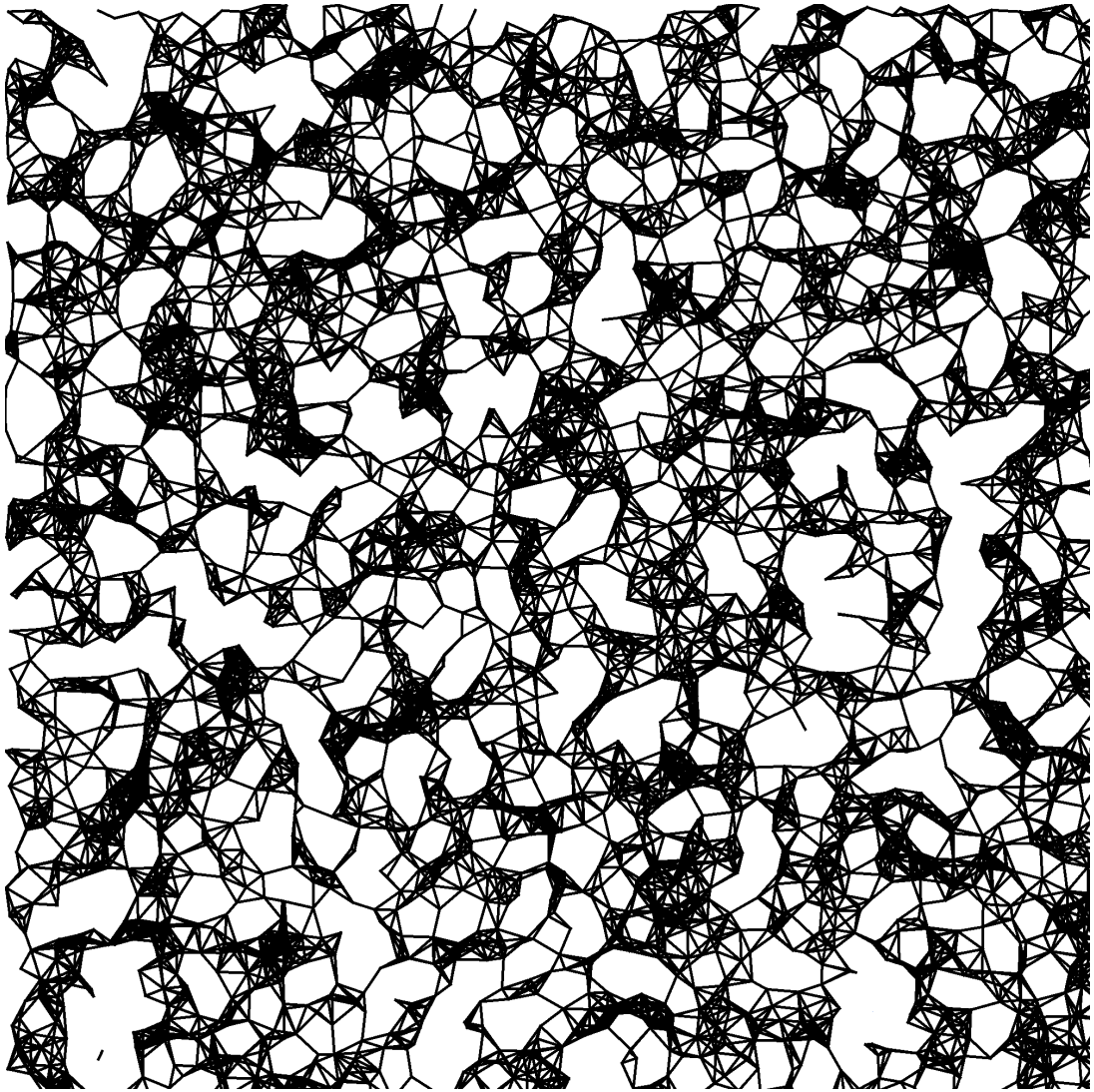
# Label Propagation

## Cut-based, Linear Time Clustering Algorithm

- cut-based clustering using size-constraint label propagation
  - start with singletons
  - traverse nodes in random order or smallest degree first
  - move node to cluster having strongest *eligible* connection
  - *eligible*: w.r.t size constraint $U$

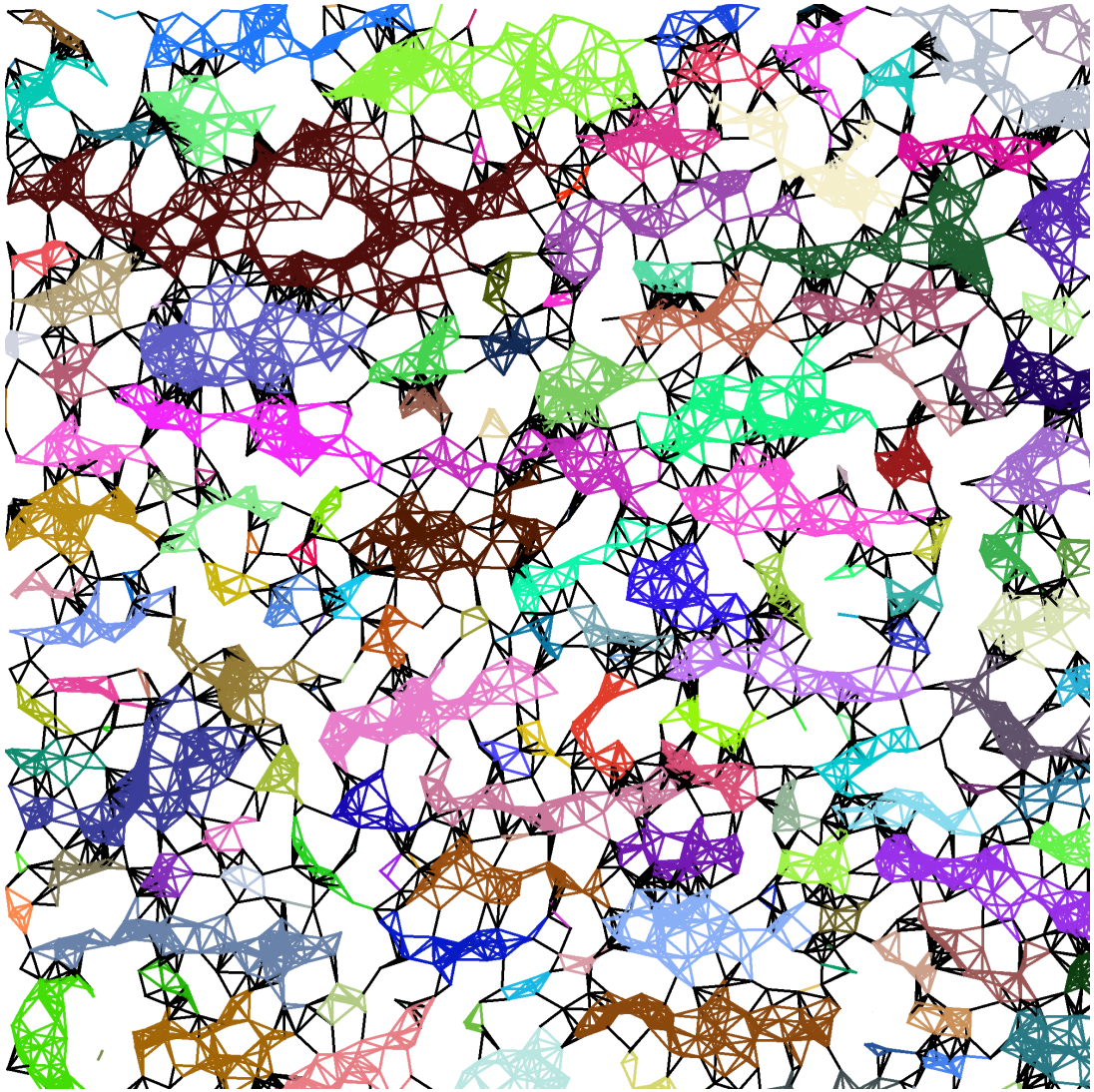


# Label Propagation



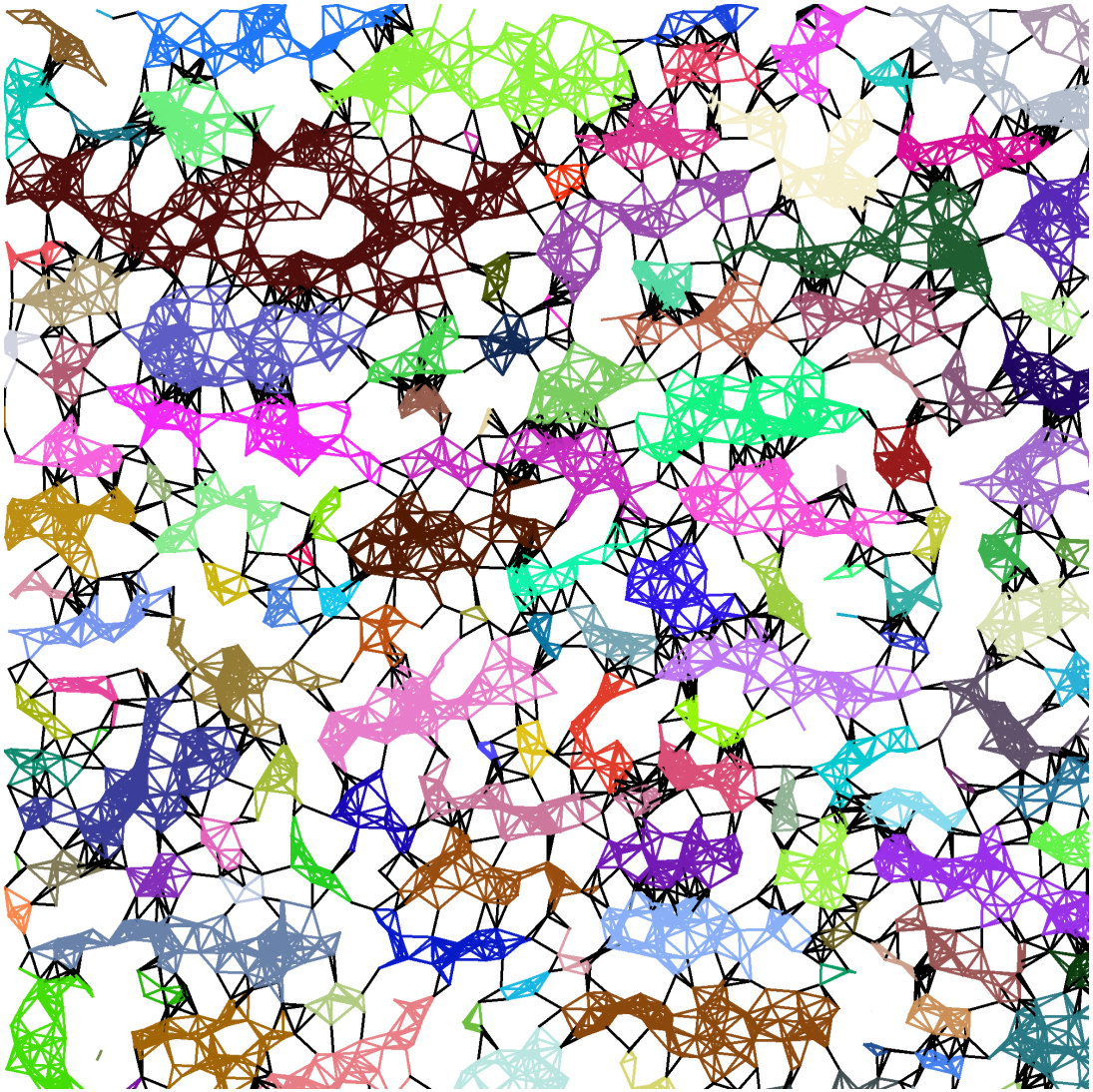
| Iteration | Cut [%] |
|---|---|
| **0** | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



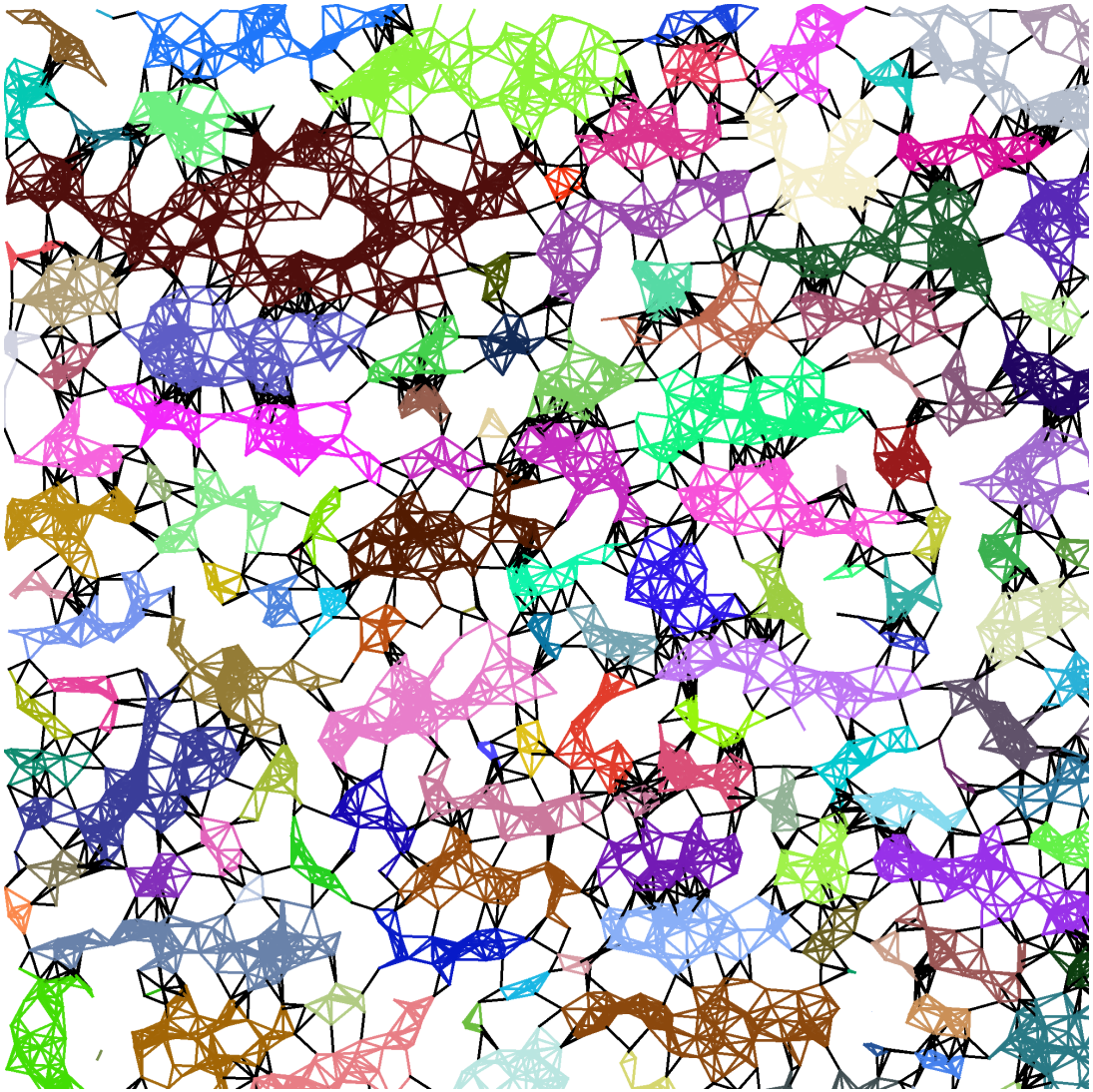
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| **1** | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



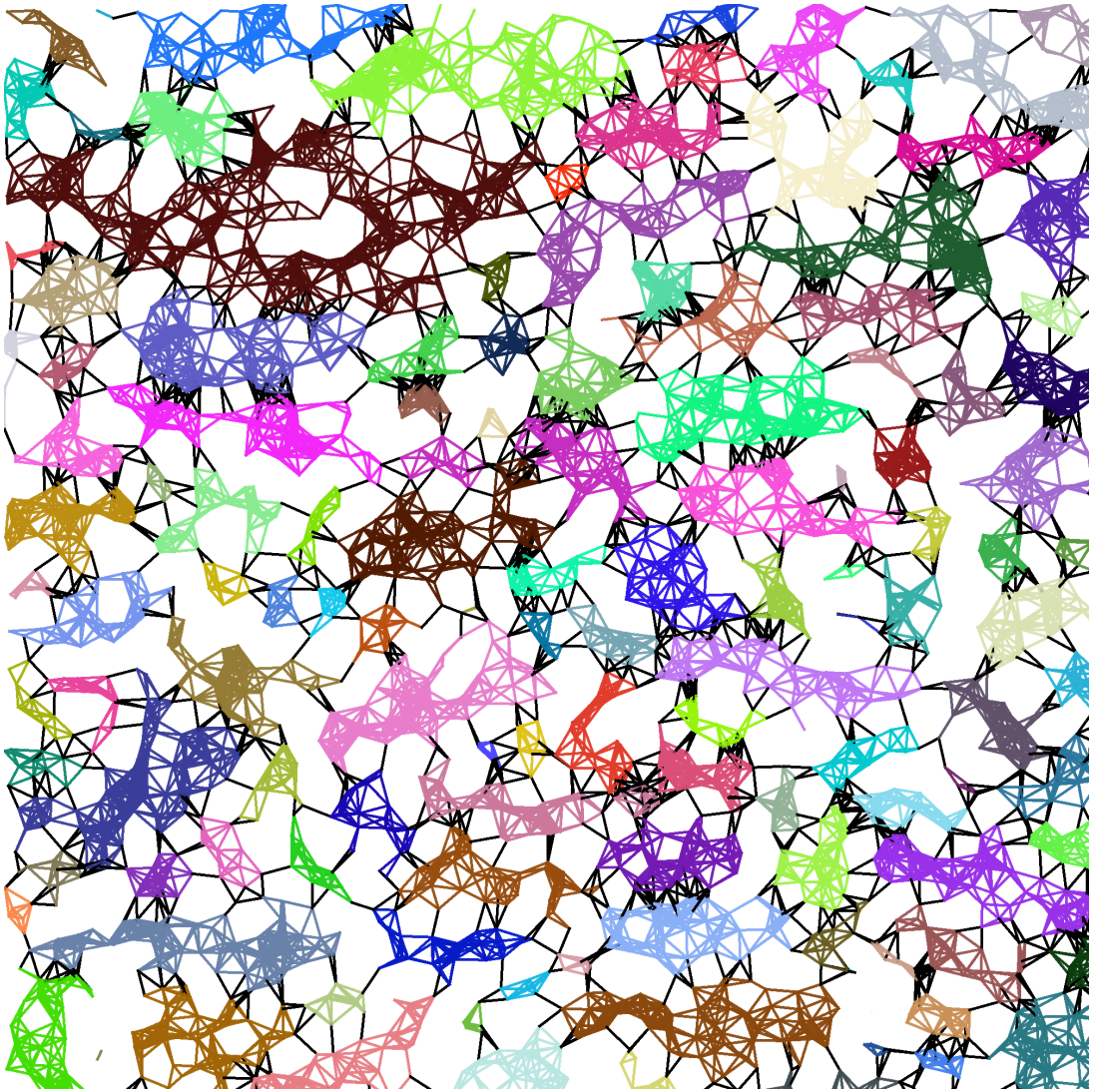
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| **2** | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



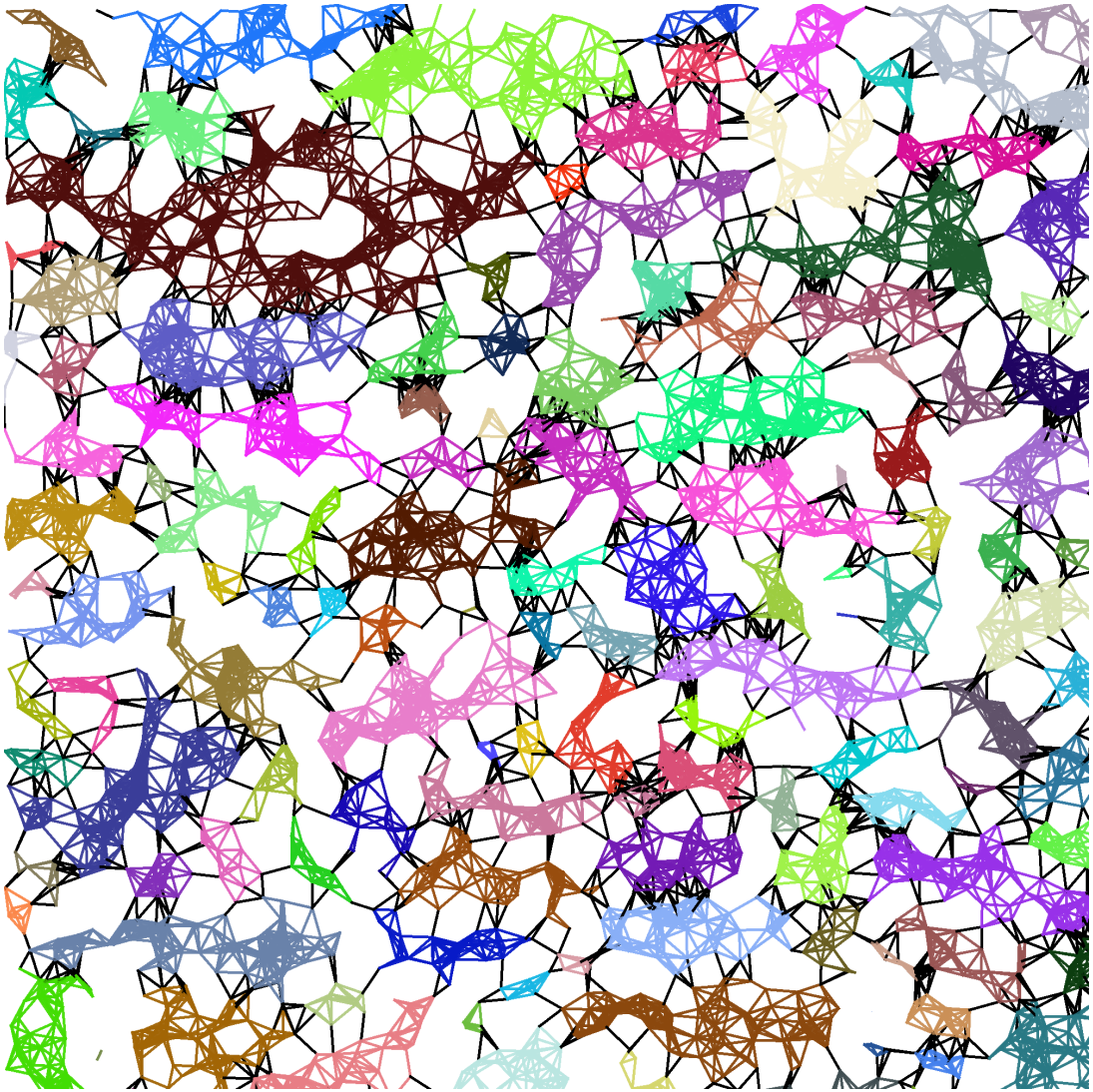
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| **3** | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



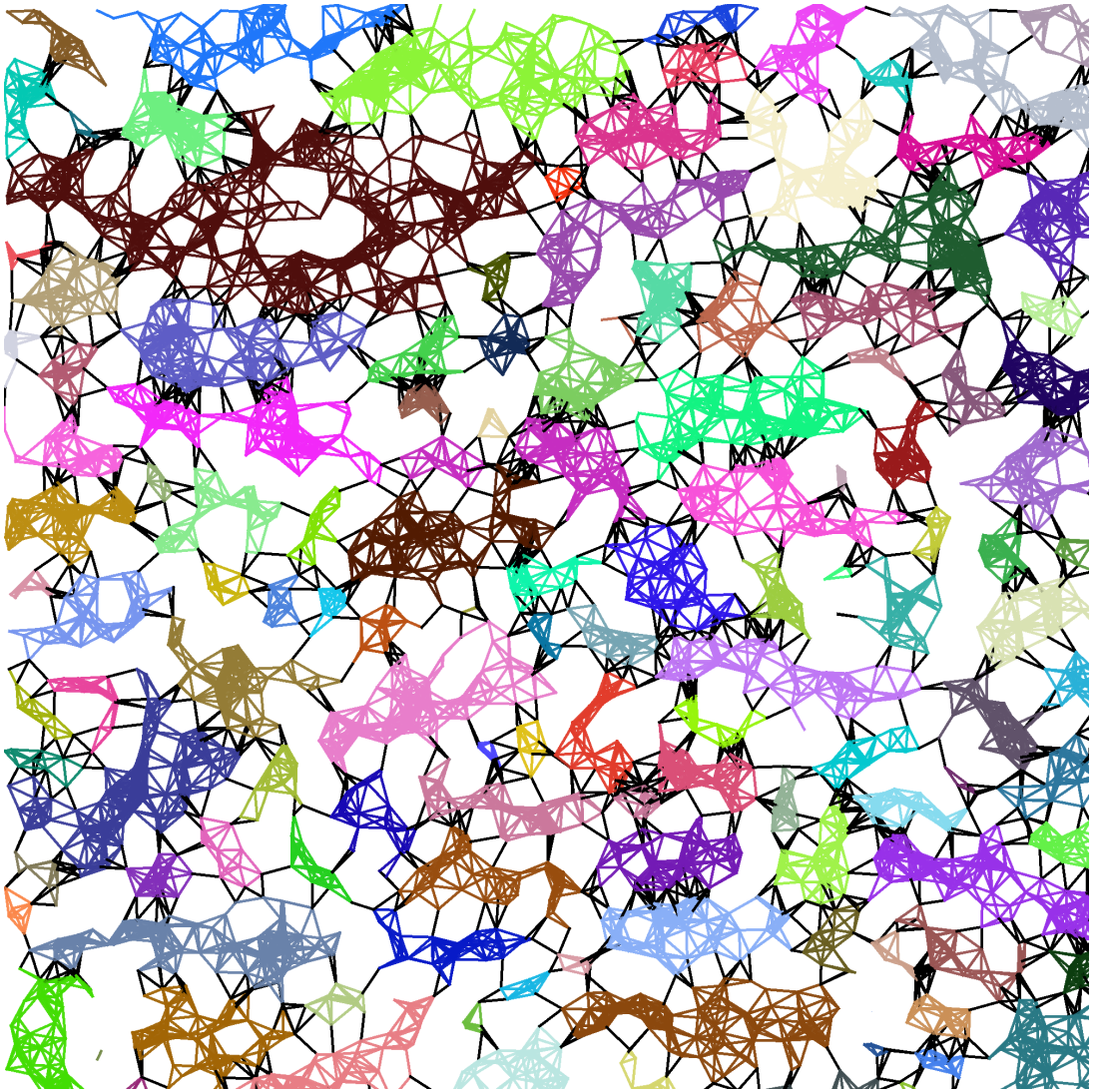
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| **4** | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



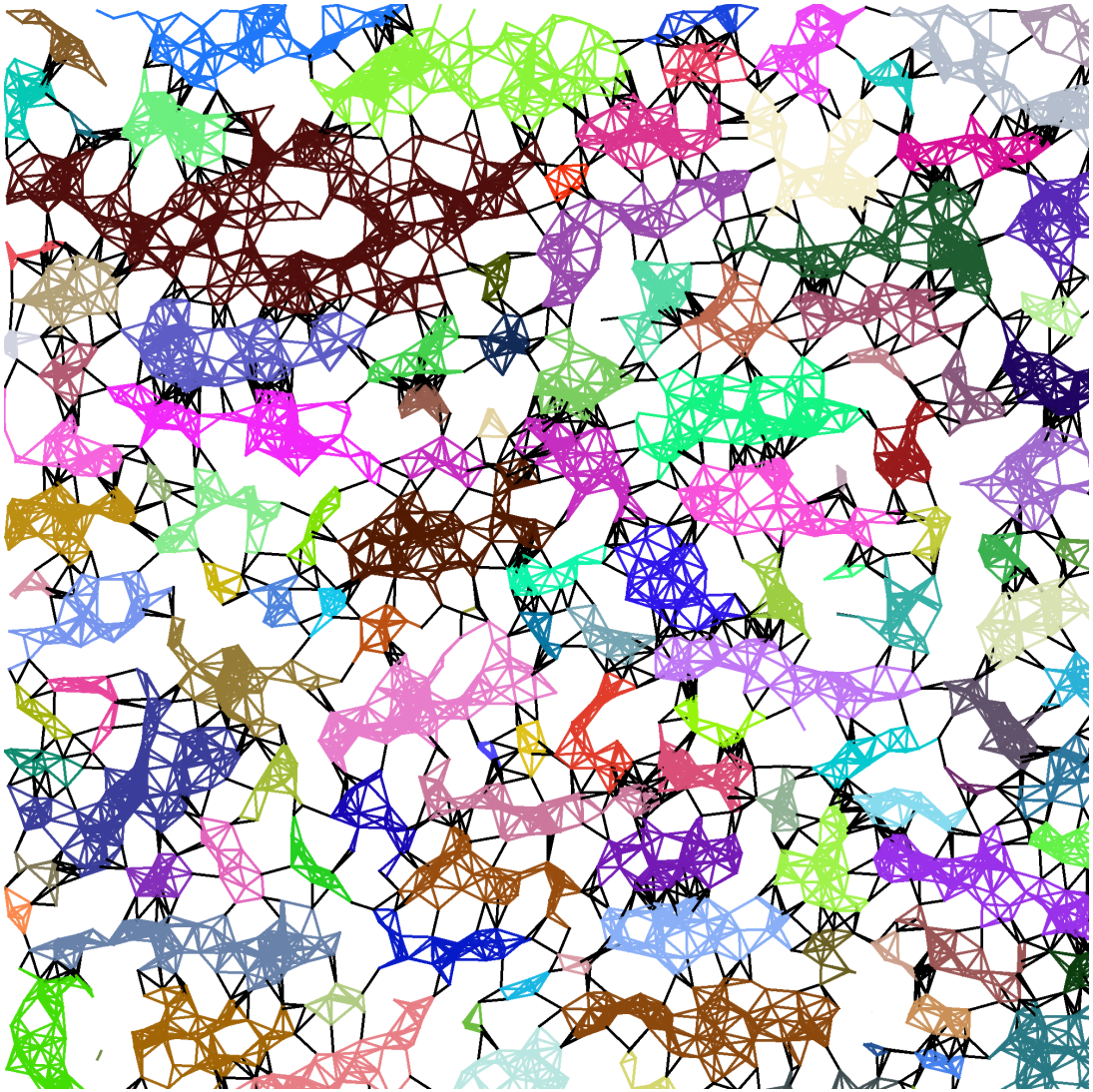
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| **5** | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



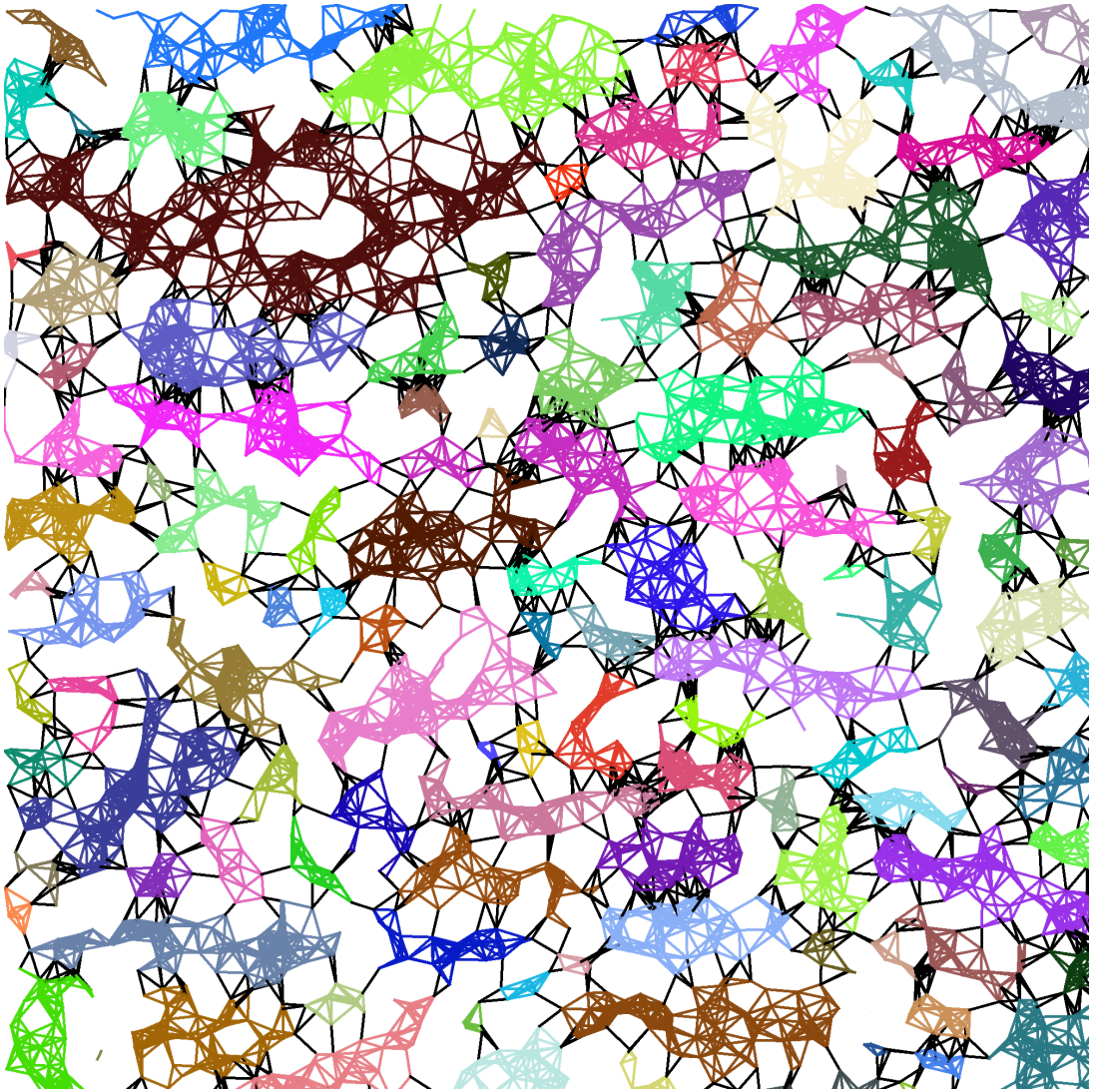
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| **6** | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Label Propagation



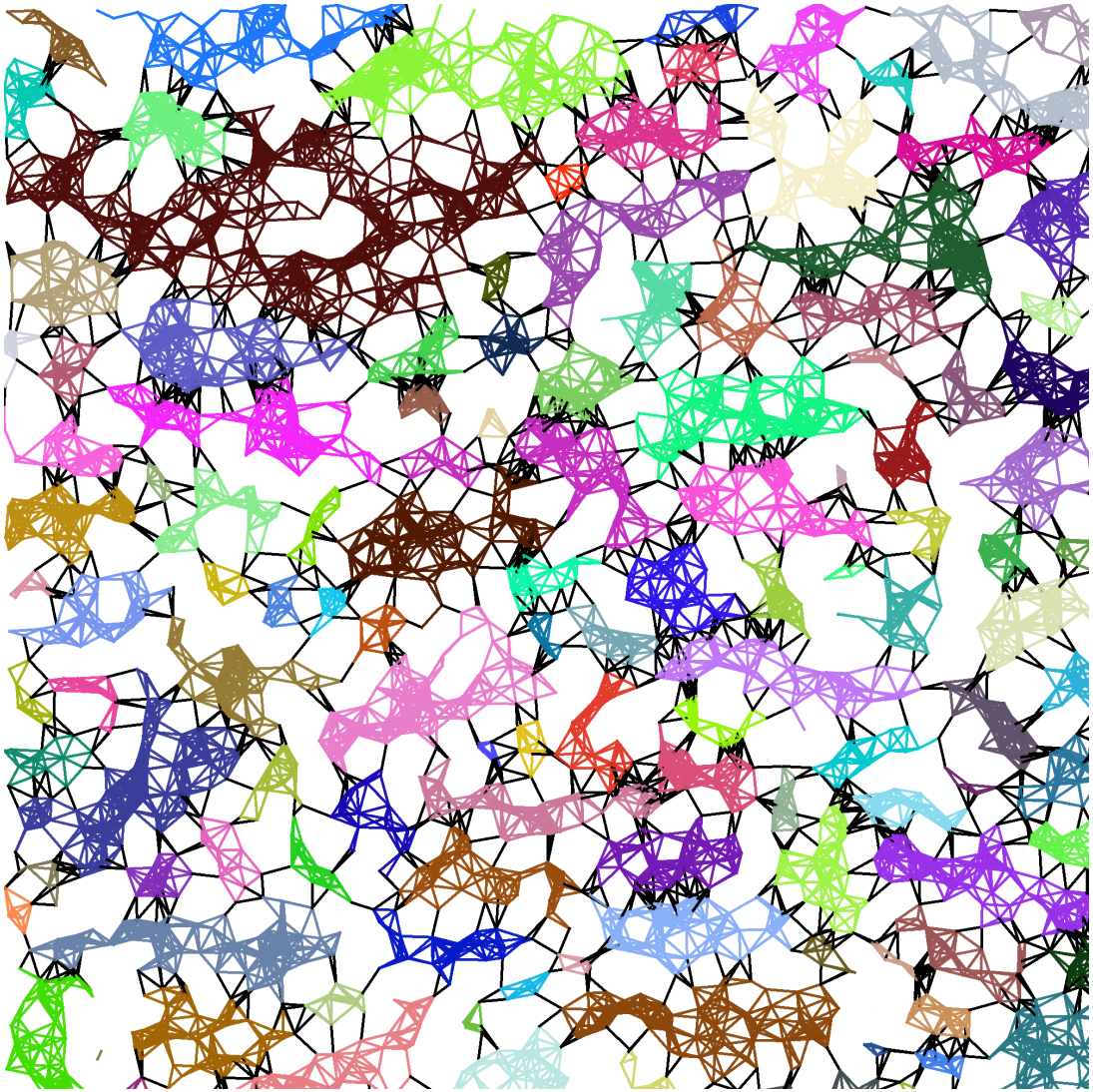
| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# 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

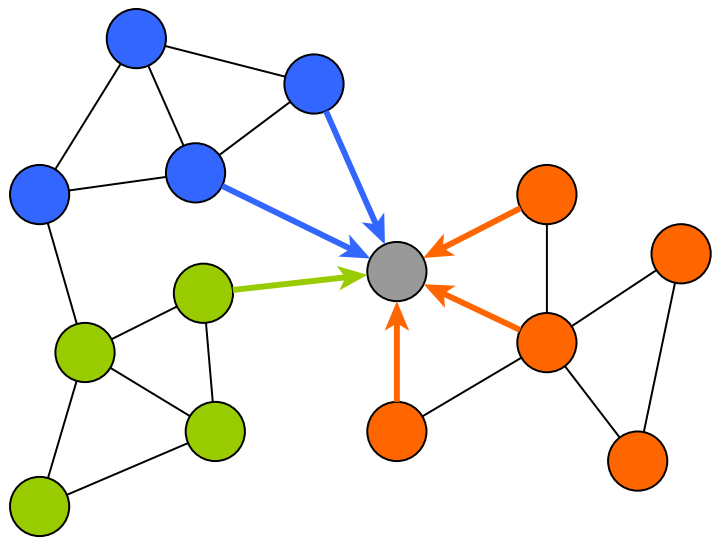


| Iteration | Cut [%] |
|---|---|
| 0 | 100 |
| 1 | 8.96 |
| 2 | 6.15 |
| 3 | 5.66 |
| 4 | 5.44 |
| 5 | 5.28 |
| 6 | 5.25 |
| 7 | 5.21 |
| 8 | 5.18 |
| ... | 5.09 |

# Graph Clustering

## Selected Methods: Louvain Method with Preprocessing

- fast preprocessing step:
  run label propagation to obtain a preliminary clustering
- speeds up Louvain considerably on large instances
- results in more diverse clusterings between runs



# Memetic Graph Clustering

## Introduction: VieClus

- memetic algorithms (MA): genetic algorithms (GA) + local search
- quality measure agnostic: focus modularity
- GAs do like nature does:
  - solutions selected from population by fitness (quality)
  - solutions are combined $\rightarrow$ offspring
  - offspring may be mutated
  - offspring inserted into population


*Natural selection does not grant organisms what they "need".*

# Memetic Graph Clustering

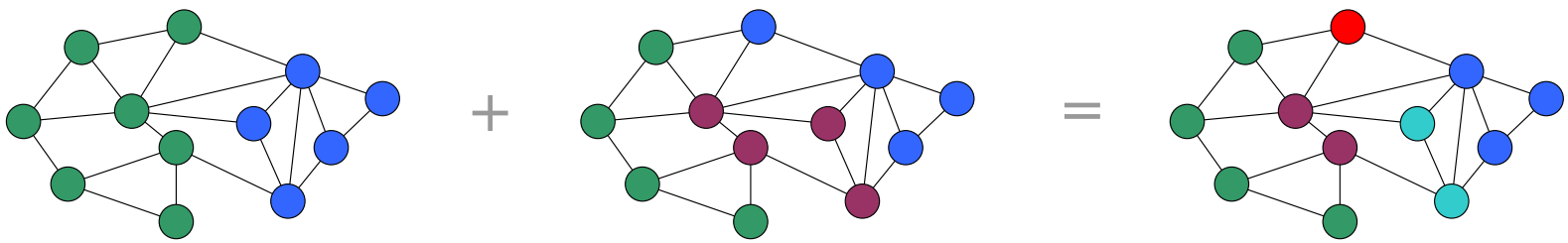
## Initial Population and Recombination Mechanism

### Initial Population

- initial population generated by Louvain method (random seeds)
- both with and without Label Propagation preprocessing
- results in a reasonably diverse base population

### Recombination Mechanism

- borrowed from ML: ensemble learning → maximum overlap



→ results in overlap clustering

# Memetic Graph Clustering

## Recombination Operators

- two kinds: flat and multilevel
- fundamental aim: high diversification
- diversity is good: prevents premature convergence

### Flat Recombination

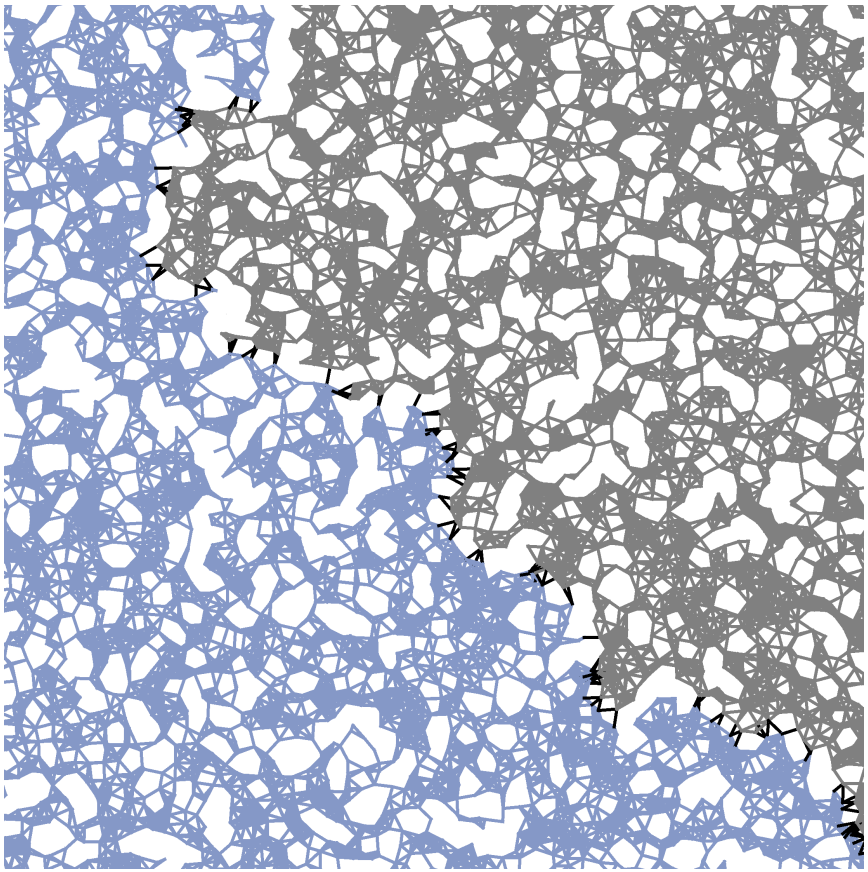
1. calculate overlap of two clusterings
2. contract overlap
3. run Louvain method on the contracted overlap

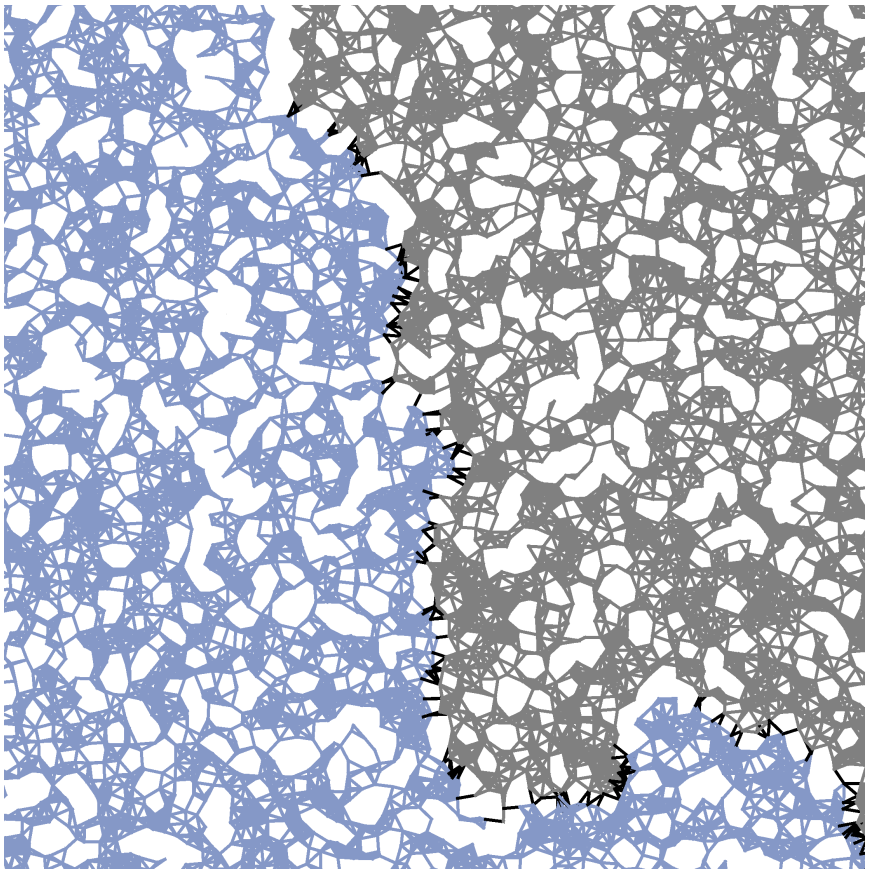
### Variations on Flat Recombination

- start Louvain with the better parent clustering
- generate second parent using graph partitioning or LP
- this introduces a lot of diversity

# Example

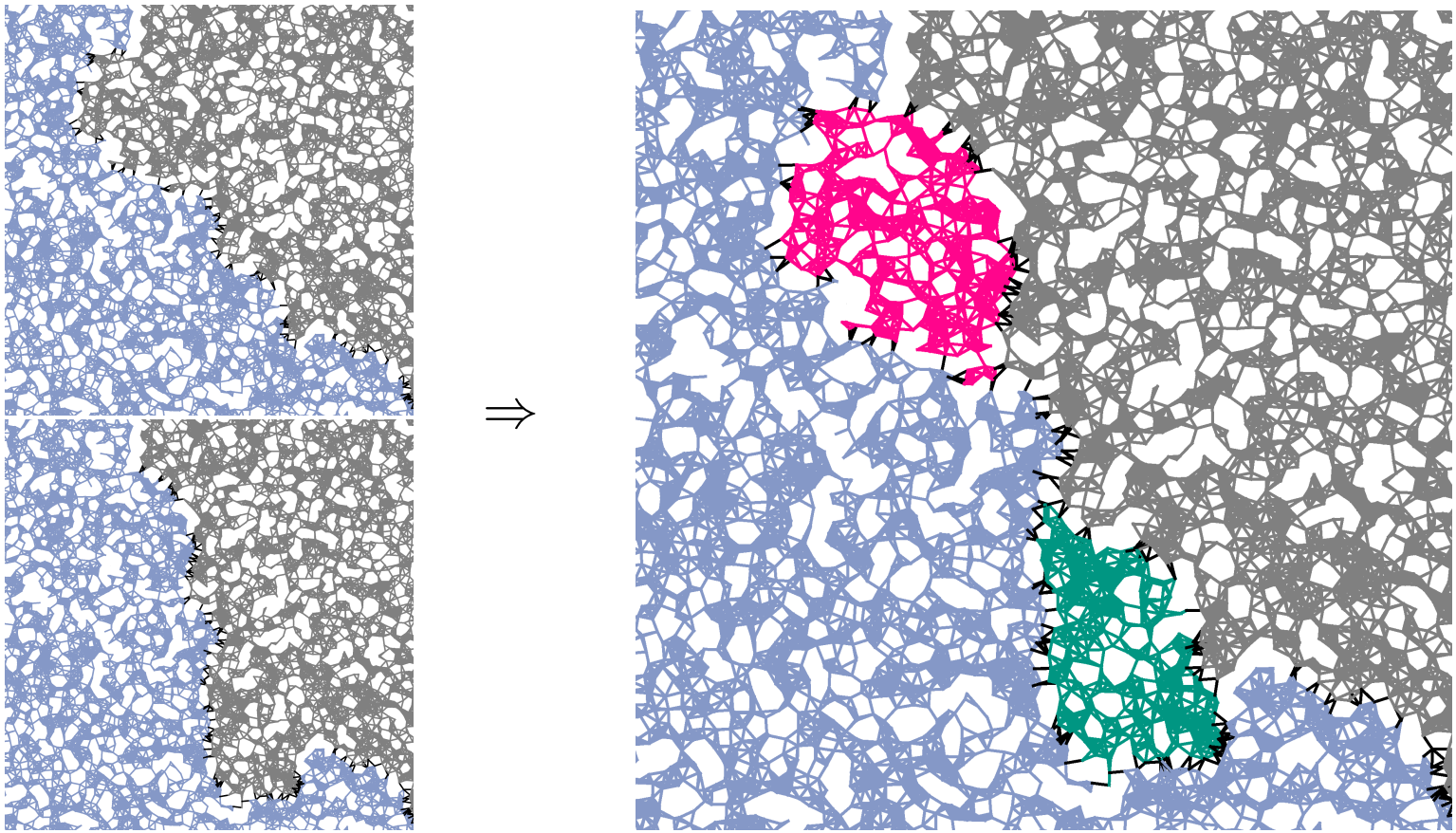
## Two Individuals $\mathcal{P}_1$, $\mathcal{P}_2$





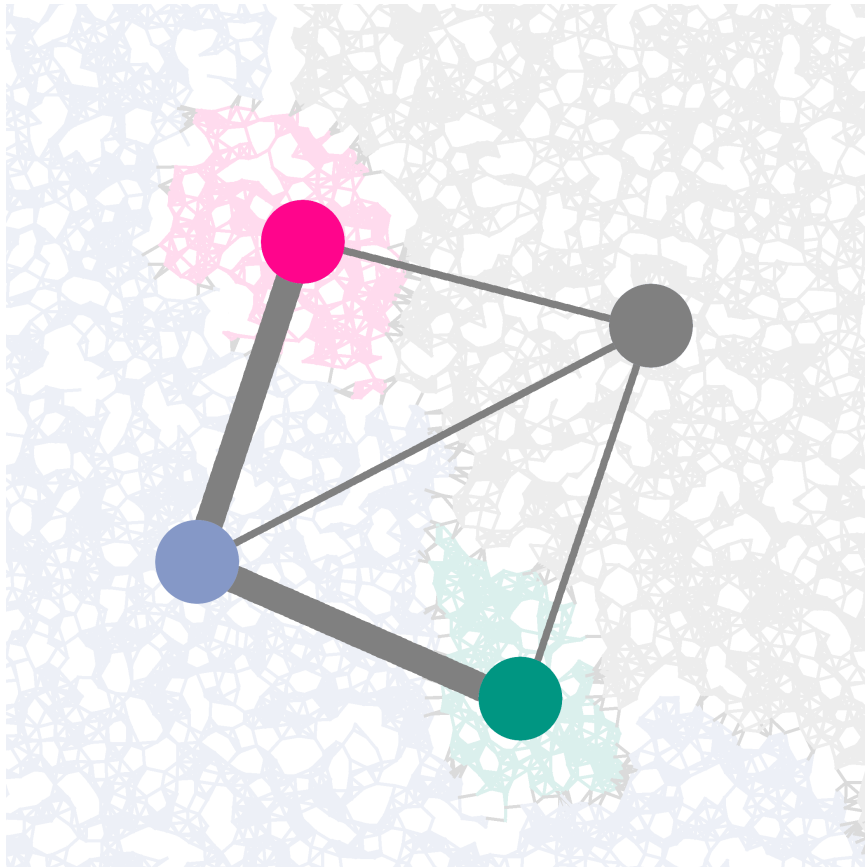
# Example

## Overlay of $\mathcal{P}_1, \mathcal{P}_2$



# Example

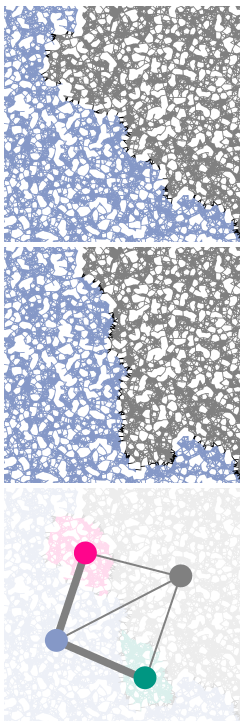
## Flat Recombination of $\mathcal{P}_1, \mathcal{P}_2$



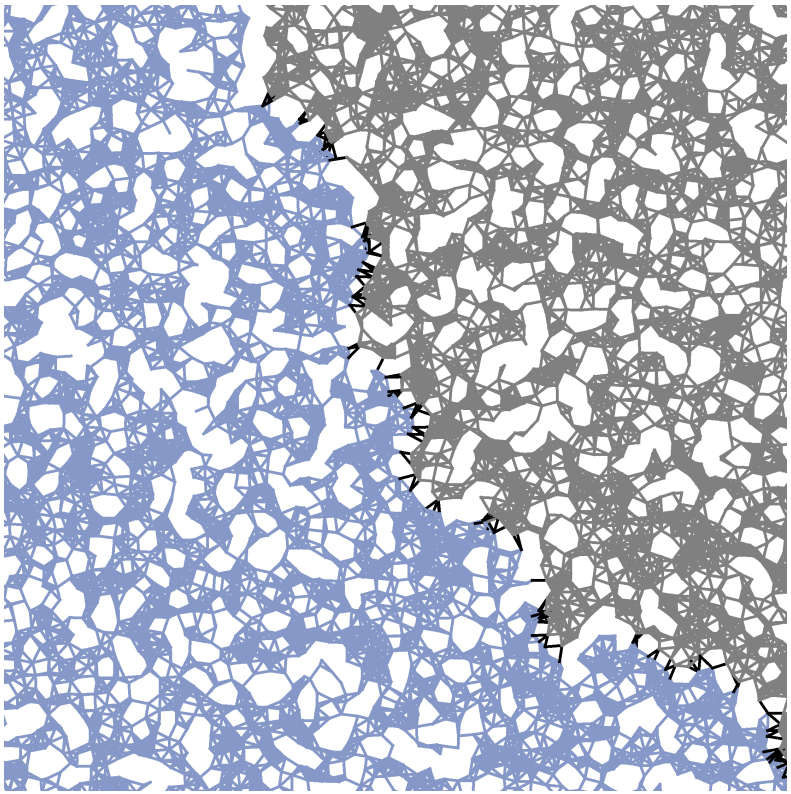
- run Louvain on this graph
- variation: start with better parent clustering

# Example

## Result of $\mathcal{P}_1$, $\mathcal{P}_2$



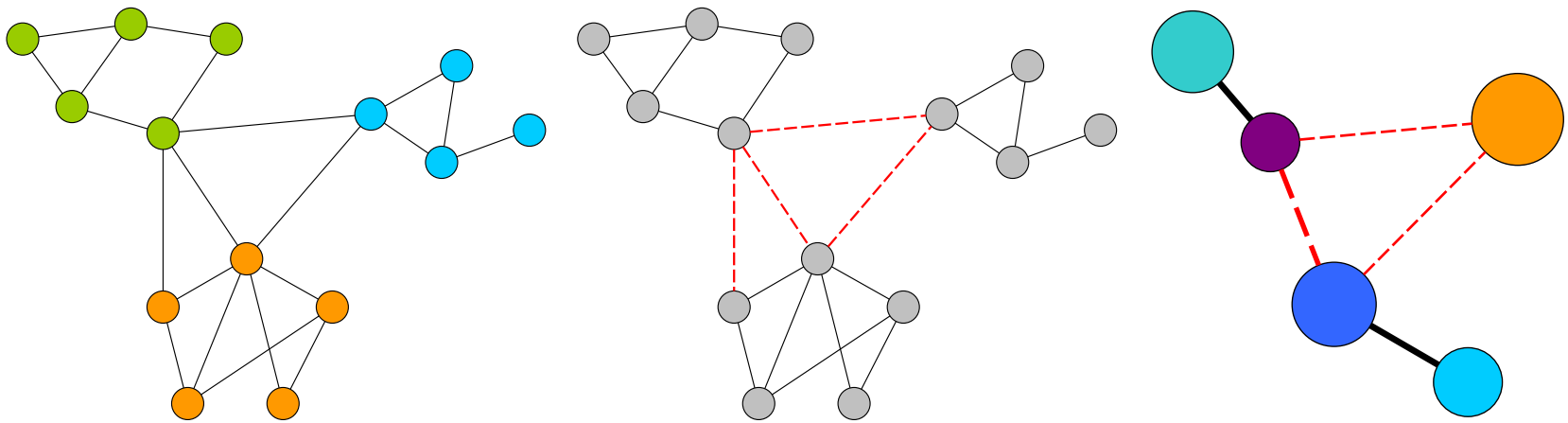
$\Rightarrow$



# Memetic Graph Clustering

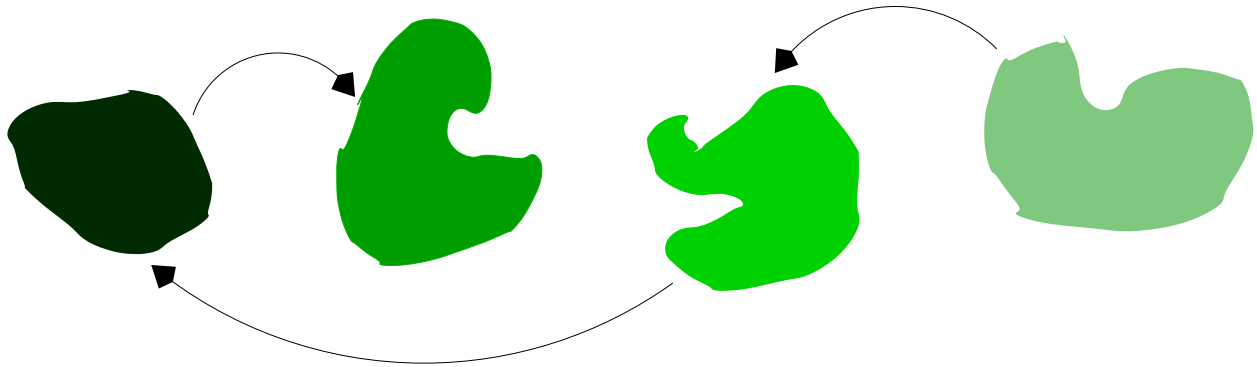
## Recombination Operator: Multilevel

- run the Louvain method, but forbid LS to consider cut edges
  $\rightarrow$ cut edges are not contracted
- allows for finer, iterative changes compared to running Louvain on the contracted overlap



- coarsest graph is the contracted overlap, contraction is usually stopped earlier

# Parallelization



- each PE has its own island (a local population)
- locally: perform combine and mutation operations
- communicate analog to *randomized rumor spreading*
  1. rumor $\leftrightarrow$ currently best local clustering
  2. local best partition *changed* $\rightarrow$ send it to $\mathcal{O}(\log P)$ random PEs
  3. asynchronous communication (MPI Isend)

  $\rightarrow$ quality records in a few minutes for small graphs

# Benchmark Instances

| Graph | $n$ | $m$ |
|---|---|---|
| Small Instances | | |
| celegans_metabolic | 453 | 2 025 |
| email | 1 133 | 5 451 |
| polblogs | 1 490 | 16 715 |
| power | 4 941 | 6 594 |
| PGPgiantcompo | 10 680 | 24 316 |
| astro-ph | 16 706 | 121 251 |
| memplus | 17 758 | 54 196 |
| as-22july06 | 22 963 | 48 436 |
| cond-mat-2005 | 40 421 | 175 691 |
| G_pin_pout | 100 000 | 501 198 |
| smallworld | 100 000 | 499 998 |
| luxembourg.osm | 114 599 | 119 666 |
| rgg_n_2_17_s0 | 131 072 | 728 753 |
| caidaRouterLevel | 192 244 | 609 066 |
| Large Instances | | |
| prefAttachment | 100 000 | 499 985 |
| coAuthorsCiteseer | 227 320 | 814 134 |
| citationCiteseer | 268 495 | 1 156 647 |
| coPapersDBLP | 540 486 | 15 245 729 |
| eu-2005 | 862 664 | 16 138 468 |
| ldoor | 952 203 | 22 785 136 |
| in-2004 | 1 382 908 | 13 591 473 |
| belgium.osm | 1 441 295 | 1 549 970 |
| 333SP | 3 712 815 | 11 108 633 |

- 10th DIMACS implementation challenge competition instances

# Results

## Small instances

| Graph | $\bar{t}$ [m] | Max. $\mathcal{Q}$ | DIMACS $\mathcal{Q}$ | $t_{\text{sol}}$[m] | Solver |
|---|---|---|---|---|---|
| as-22july06 | $< 1$ | **0.679 396** | 0.678 267 | 6.6 | CGGC |
| astro-ph | $< 1$ | **0.746 292** | 0.744 621 | 11.9 | VNS |
| celegans_metabol | $< 1$ | *0.453 248* | 0.453 248 | $< 1$ | VNS |
| cond-mat-2005 | $< 1$ | **0.750 171** | 0.746 254 | 40.9 | CGGC |
| email | $< 1$ | *0.582 829* | 0.582 829 | $< 1$ | VNS |
| PGPgiantcompo | $< 1$ | **0.886 853** | 0.886 564 | 1.9 | CGGC |
| polblogs | $< 1$ | *0.427 105* | 0.427 105 | $< 1$ | VNS |
| power | $< 1$ | **0.940 977** | 0.940 851 | $< 1$ | VNS |
| smallworld | $< 1$ | **0.793 187** | 0.793 042 | 16.8 | VNS |
| memplus | 2.6 | **0.701 275** | 0.700 473 | 3.2 | CGGC |
| G_n_pin_pout | 3.9 | **0.500 466** | 0.500 098 | 64.8 | CGGC |
| caidaRouterLevel | 5.0 | **0.872 828** | 0.872 042 | 81.0 | CGGC |
| rgg_n17 | 5.0 | **0.978 454** | 0.978 324 | 37.5 | VNS |
| luxembourg.osm | 7.3 | **0.989 672** | 0.989 621 | 40.9 | VNS |

- $\bar{t}$ [m]: time in minutes to reach DIMACS result
- $t_{\text{sol}}$ [m]: time in minutes to reach DIMACS result of prev. solver

# Results

## Large Instances

| Graph | $\bar{t}$ [m] | Max. $\mathcal{Q}$ | DIMACS $\mathcal{Q}$ | $t_{\text{sol}}$ [m] | Solver |
|---|---|---|---|---|---|
| coAuthorsCiteseer | 3.9 | **0.906 830** | 0.905 297 | 91.3 | CGGC |
| citationCiteseer | 12.9 | **0.825 545** | 0.823 930 | 77.6 | CGGC |
| coPapersDBLP | 20.5 | **0.868 058** | 0.866 794 | 603.3 | CGGC |
| belgium.osm | 29.5 | **0.995 064** | 0.994 940 | 102.9 | CGGC |
| ldoor | 35.1 | **0.970 555** | 0.969 370 | 485.6 | ParMod |
| eu-2005 | 65.8 | **0.941 575** | 0.941 554 | 341.5 | CGGC |
| in-2004 | 237.4 | **0.980 690** | 0.980 622 | 244.0 | CGGC |
| 333SP | 297.1 | **0.989 356** | 0.989 095 | 976.9 | ParMod |
| prefAttachment | * | **0.316 089** | 0.315 994 | 1 353.1 | VNS |

- $\bar{t}$ [m]: time in minutes to reach DIMACS result
- $t_{\text{sol}}$ [m]: time in minutes to reach DIMACS result of prev. solver

# More Results

- better than all solvers reported in the recent literature
  - Karypis and LaSalle
  - Lu et al.
  - Ryu and Kim
  - Džamić et al. (ascent-decent VNS)
  - ...

  → paper

Vienna Graph Clustering:
`https://github.com/VieClus/VieClus`

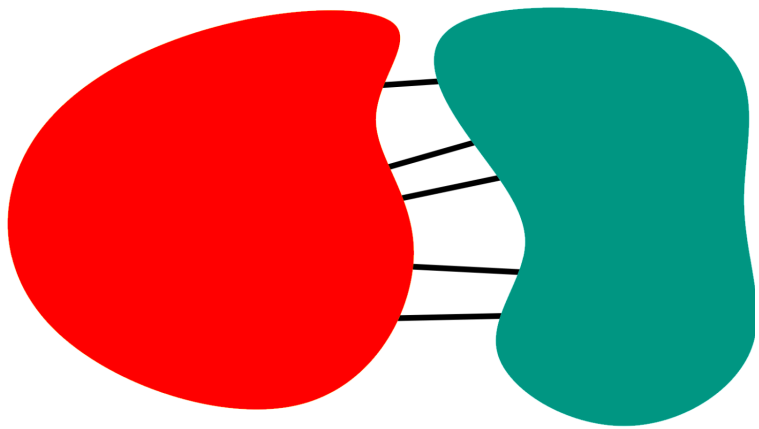
# In Depth Example
## "(In)exact Reductions" in Minimum Cuts

# 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 $m \geq n\lambda/2$

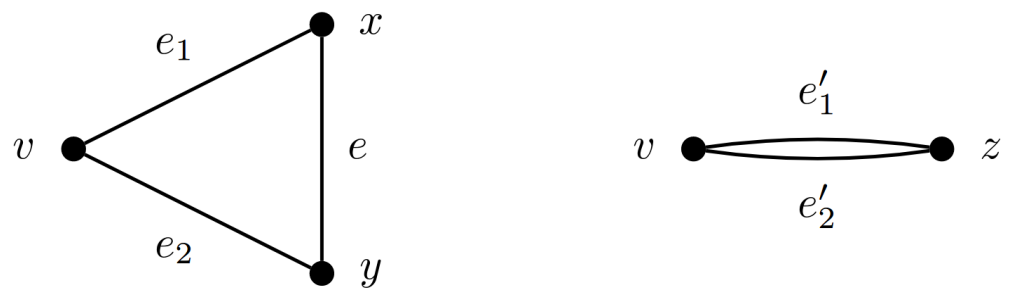
**Proof:** Assume $\exists v \in V : deg(v) < \lambda$,
then $C = \{v\}$ is a cut whose size is $< \lambda$. Contradiction.
The second claim follows from the first one.

# 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 $E$

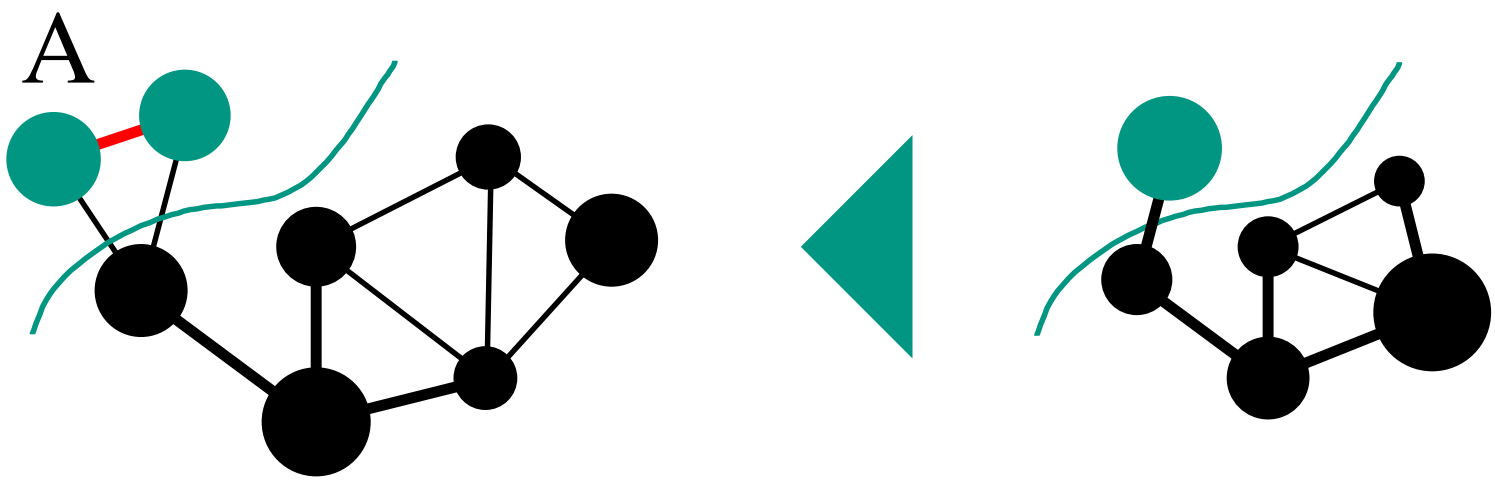



$\rightarrow$ multi-edges can be created ($\rightsquigarrow$ practice use weights)!

# Minimum Cut $\leftrightarrow$ Contraction

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

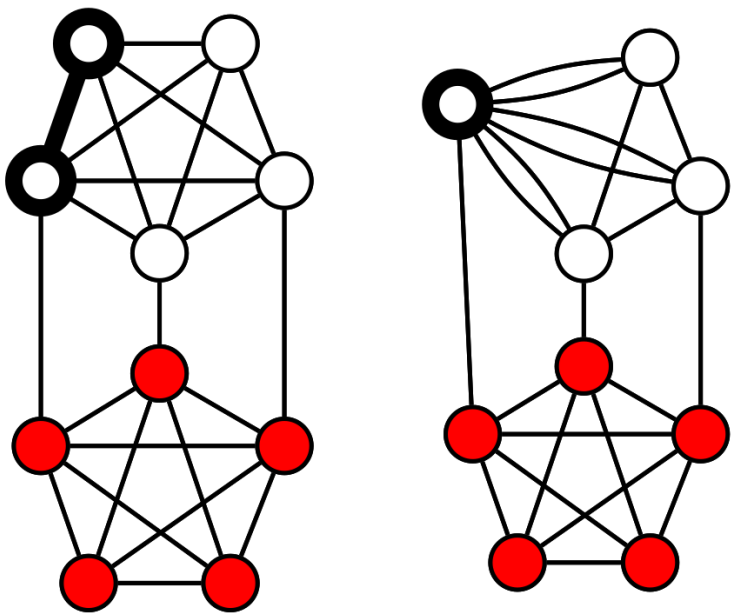
**Proof:**
Let $(K, \overline{K})$ be a minimum cut in $G/e$.
Let the size of the cut be $\lambda$.
Wlog let $x$ and $y$ be the vertices of $e$, and $z \in K$
Unpack $z$ and leave $x$ and $y$ in $K \rightarrow$ cut in $G$ of size $\lambda$



# Algorithm

**Exibit A**

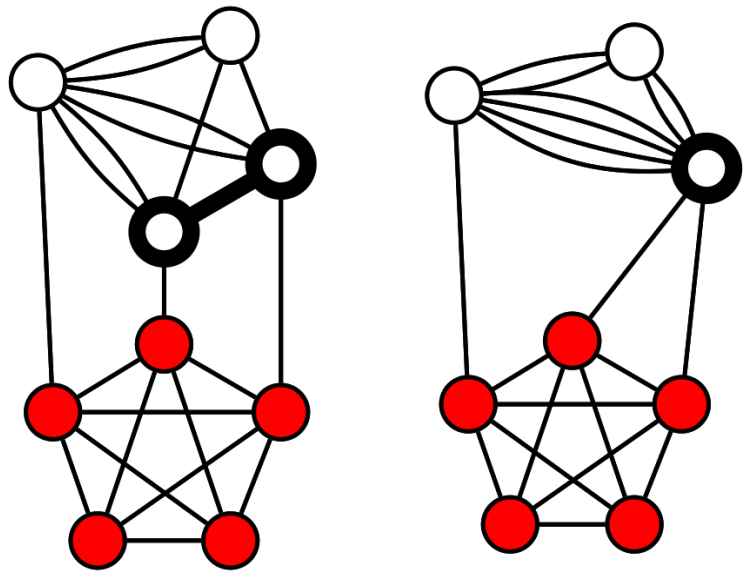
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

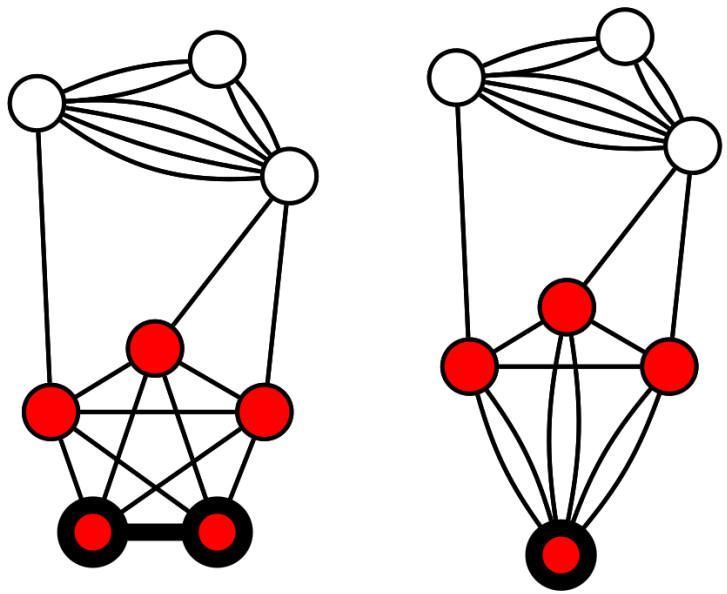
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

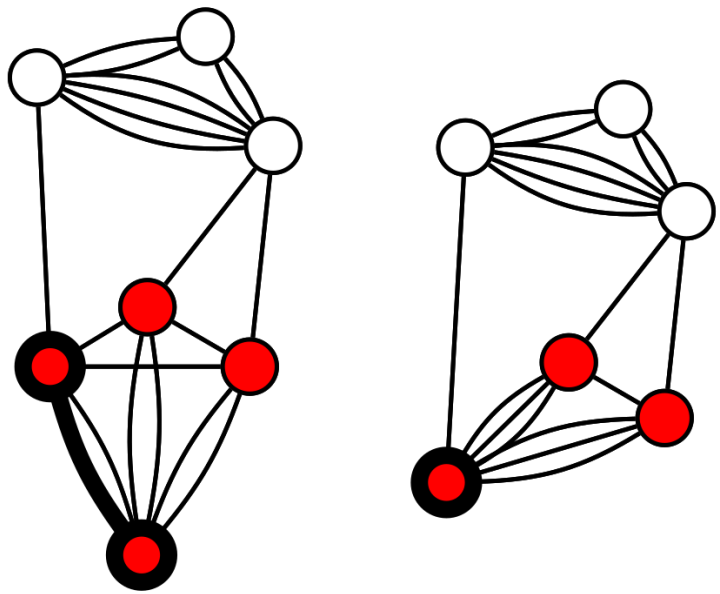
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

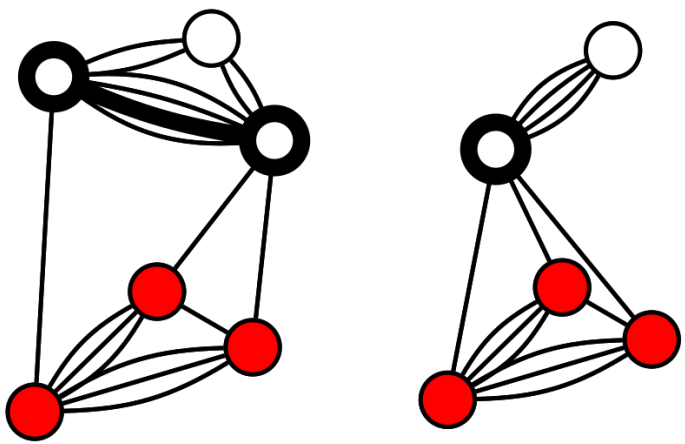
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# 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 $H$

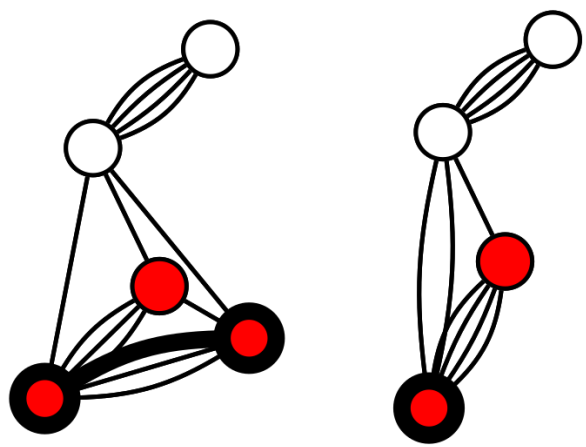


# Algorithm

**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 $H$

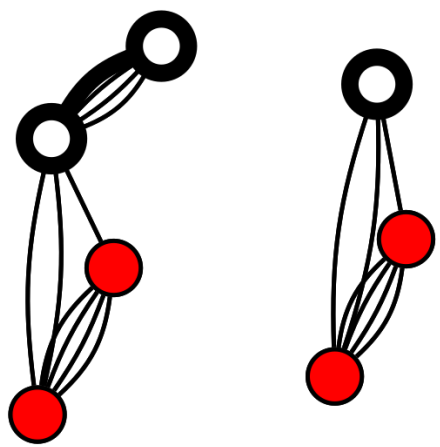
---



# Algorithm

**Exibit A**

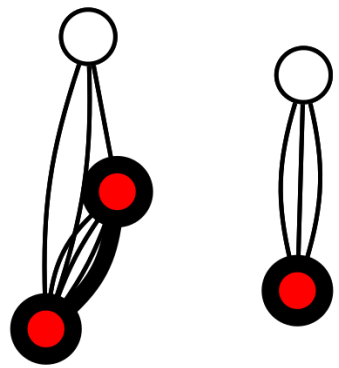
$H \leftarrow G$
**while** $H$ has more than 2 nodes **do**
  $e \leftarrow$ edge of $H$ picked uniformly at random
  $H \leftarrow \text{contract}(H, e)$
**done**
$(C, \overline{C}) \leftarrow$ vertex set in $G$ that correspond to the vertices in $H$



# Algorithm

**Exibit A**

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

---

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
  $H \leftarrow \text{contract}(H, e)$
**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 $1 - 1/e$:

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

$\rightsquigarrow$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 error probability $O(1/n)$

# Main Questions

how can kernelization help?

# Main Questions

something better than contracting random edges?

# Main Questions

can we still obtain good cuts in practice?

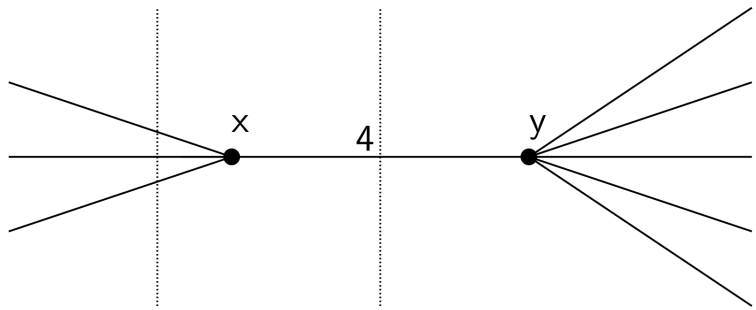
# Main Questions

can we then use this to obtain better kernels?

# Kernelization

## Padperg-Rinaldi Tests

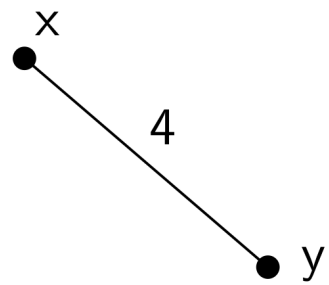
$$\deg(x) \leq 2\omega(x, y)$$



# Kernelization

## Padperg-Rinaldi Tests

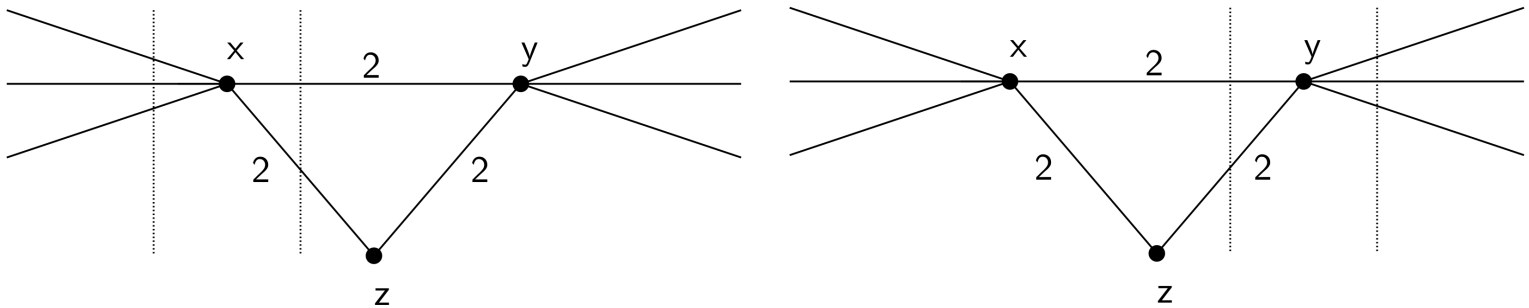
$$\omega(x, y) \geq \hat{\lambda}$$



# 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

**Nagamochi, Ono, and Ibaraki**

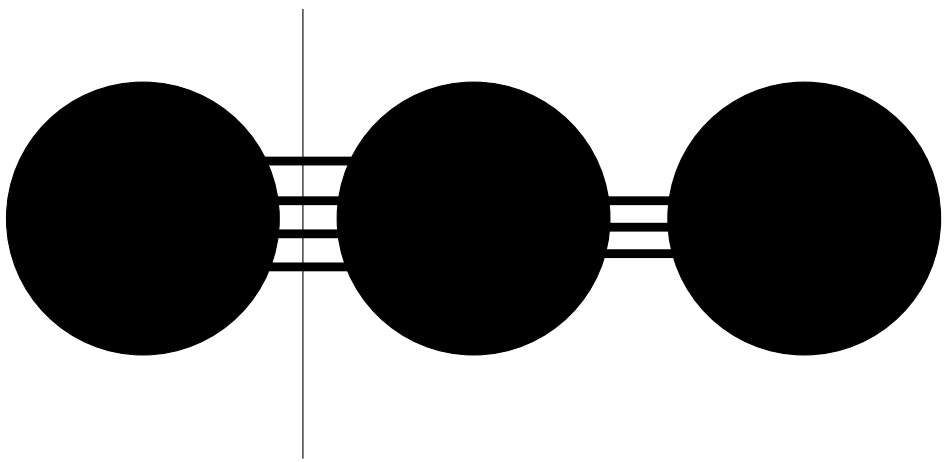
Key Idea: a spanning tree contains at least one edge from any cut

- 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**

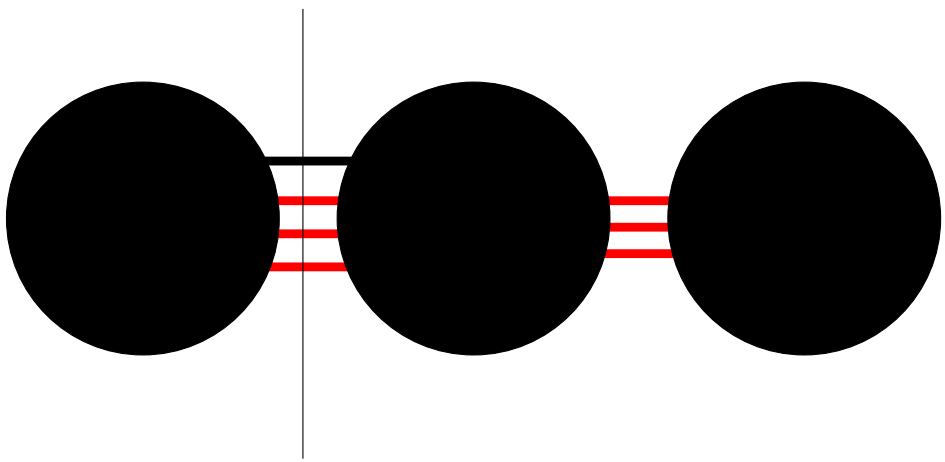
Key Idea: a spanning tree contains at least one edge from any cut

- 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**

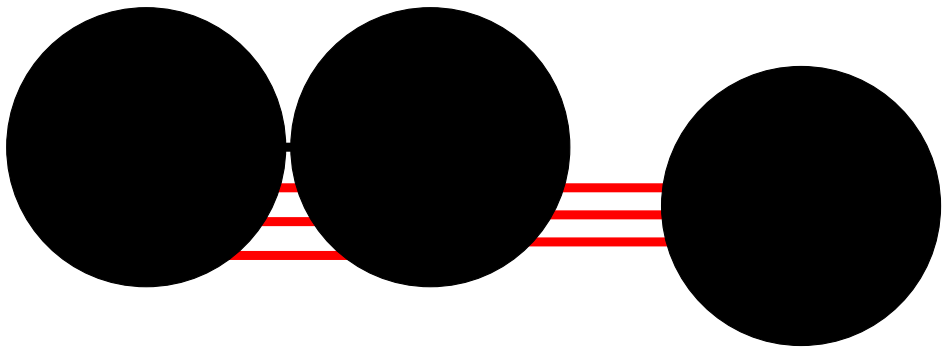
Key Idea: a spanning tree contains at least one edge from any cut

- 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 $\geq \hat{\lambda}$

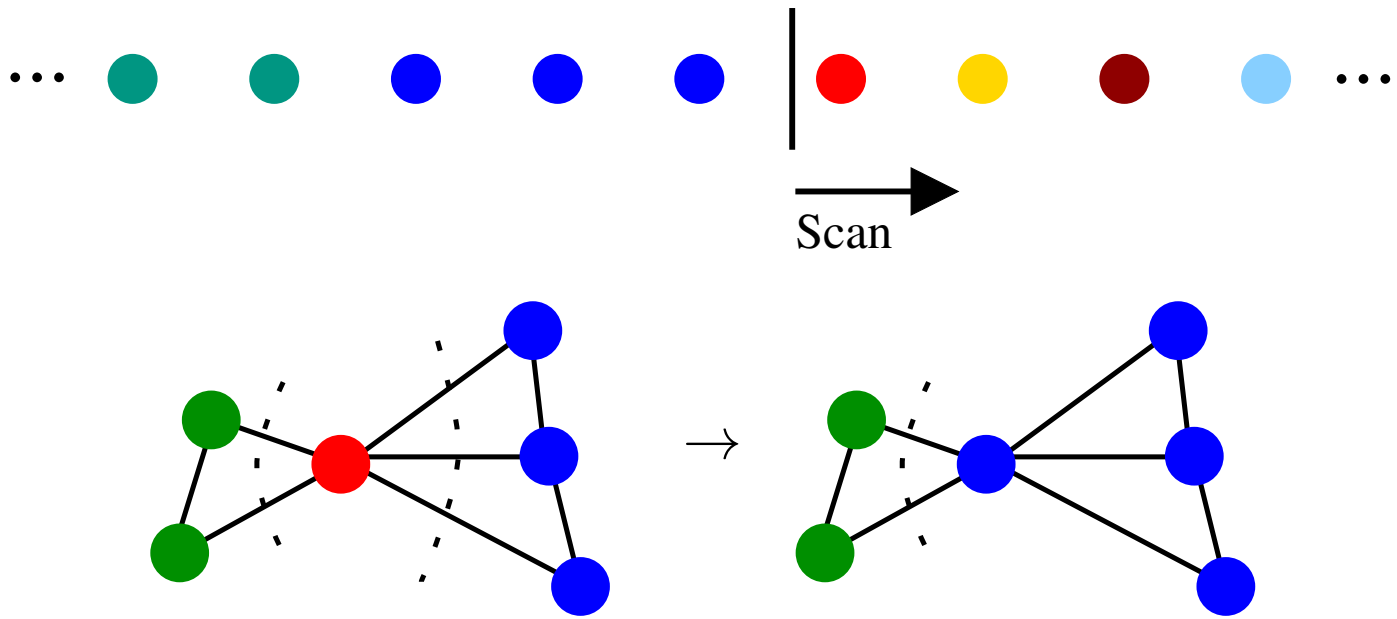
  $\rightsquigarrow$ contract all of them

Note: initial $\hat{\lambda}$ comes from minimum degree
Some of the reductions depend heavily on $\hat{\lambda}$

# Label Propagation

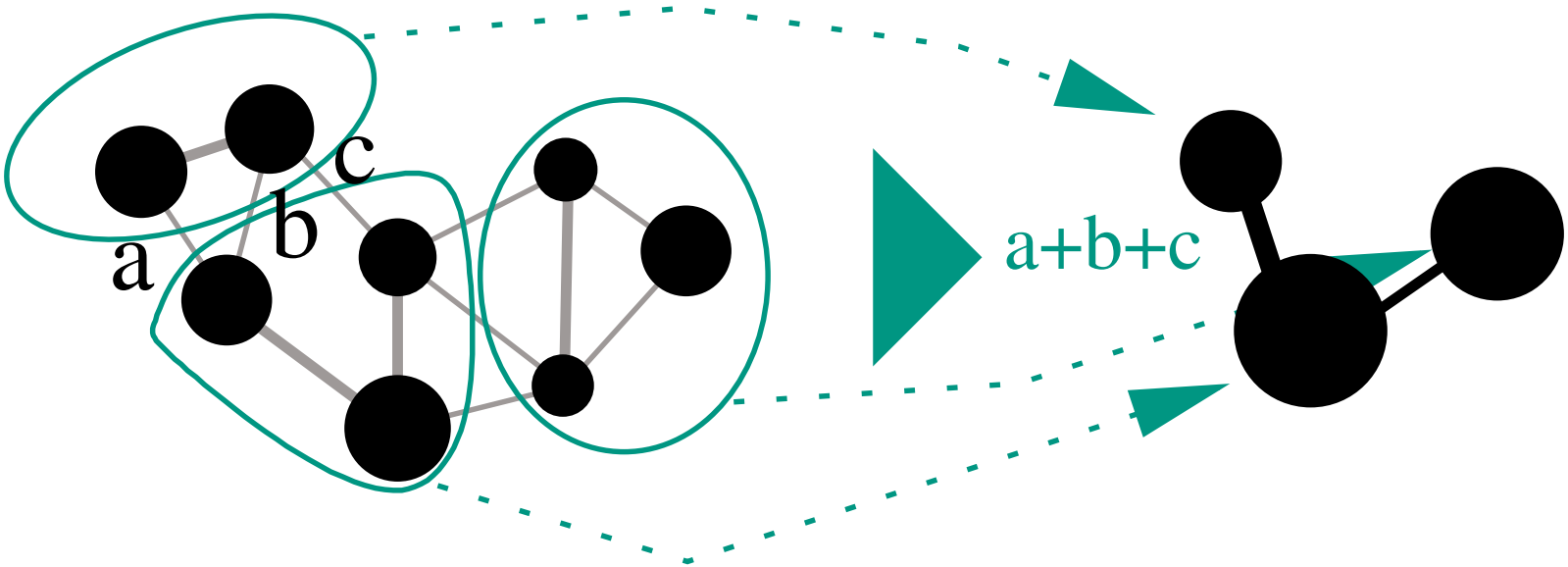
**Cut-based, Linear Time Clustering Algorithm [Raghavan et. al]**

- 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



# Basic Idea

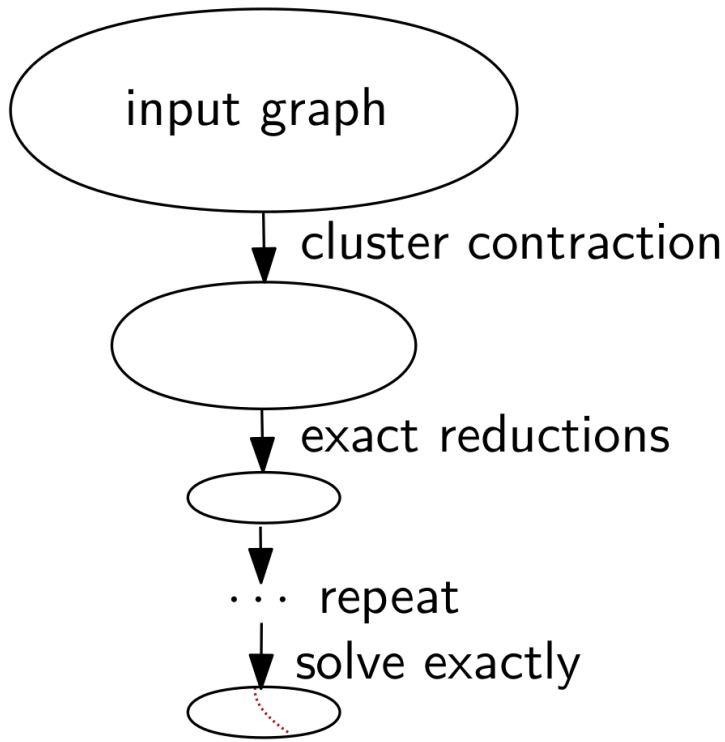
## Contraction of Clusterings




- 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

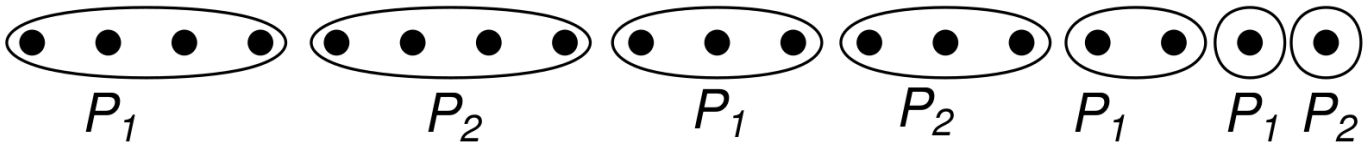
- (Inexact) Cluster reduction + Exact reductions
- Solve kernel to optimality
  using Nagamochi, Ono and Ibaraki's algorithm
  → overall linear running time, but potentially suboptimal cuts



# Parallelization

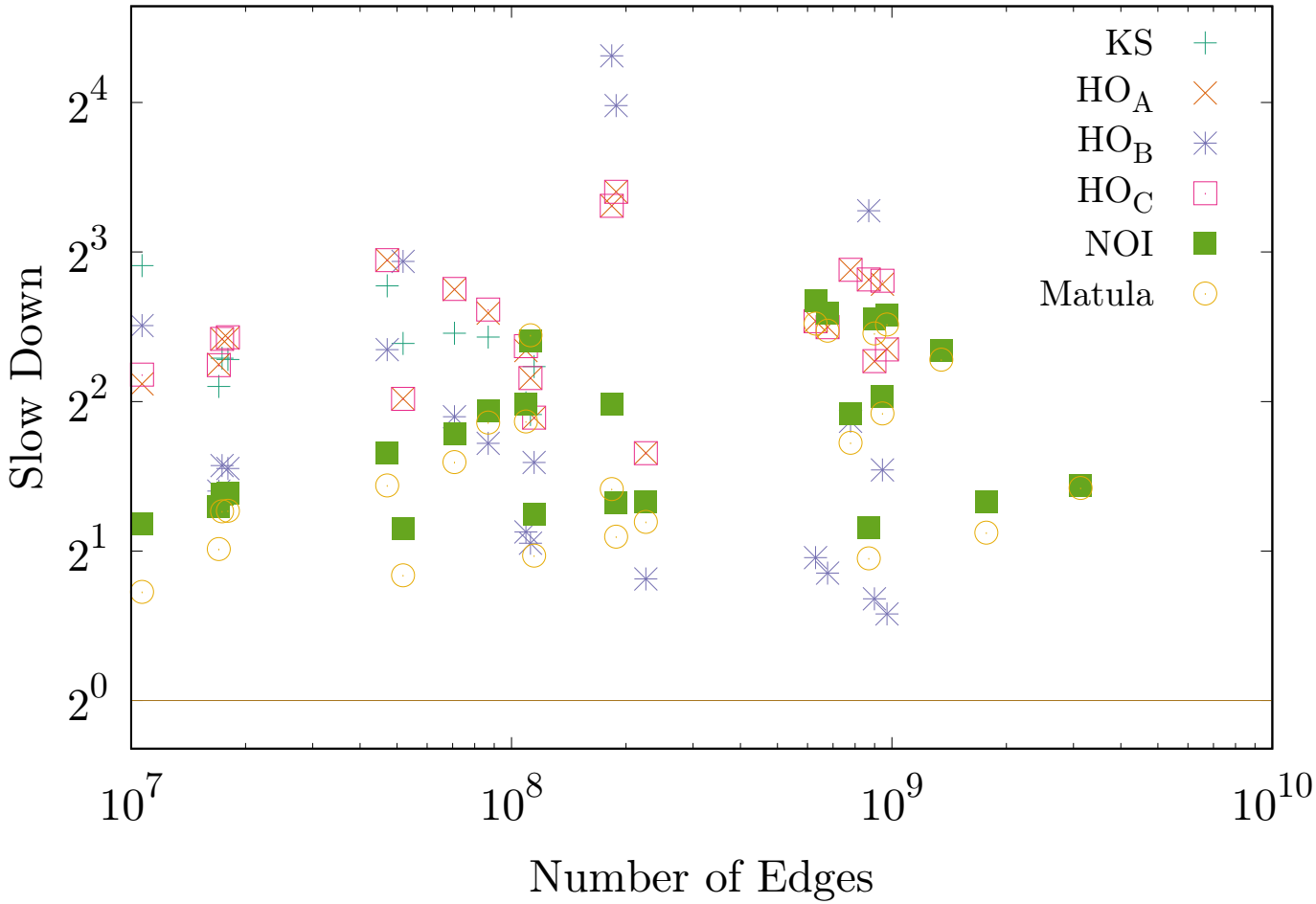
## Shared-memory with OpenMP

- Parallel label propagation



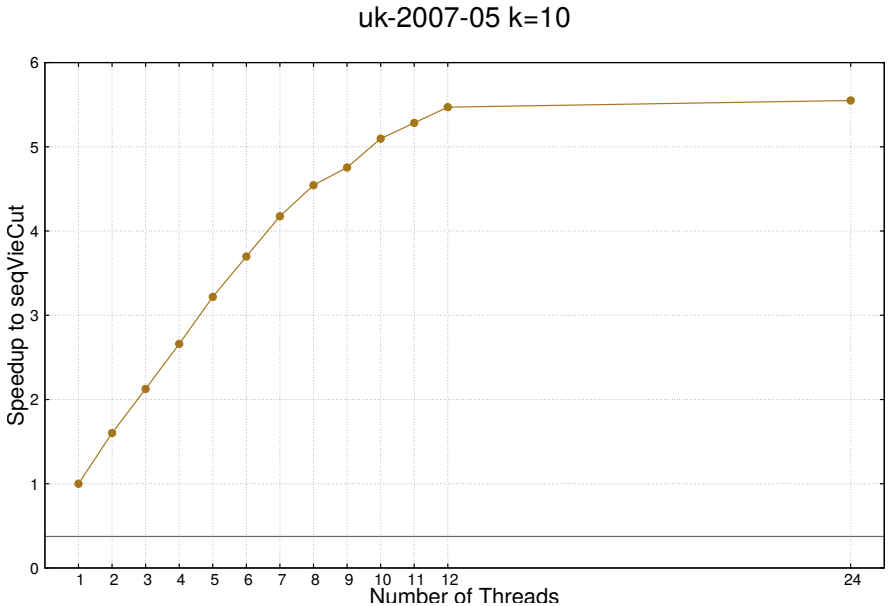

  - as brutal as: pragma openmp for and ignore conflicts on labels
- Parallel Padberg-Rinaldi:
  - check edges independently $\rightsquigarrow$ embarassingly parallel
  - collect edges then contract
    $\rightarrow$ essentially linear time
- Parallel contraction (not here)
- run Nagamochi, Ono and Ibarakis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

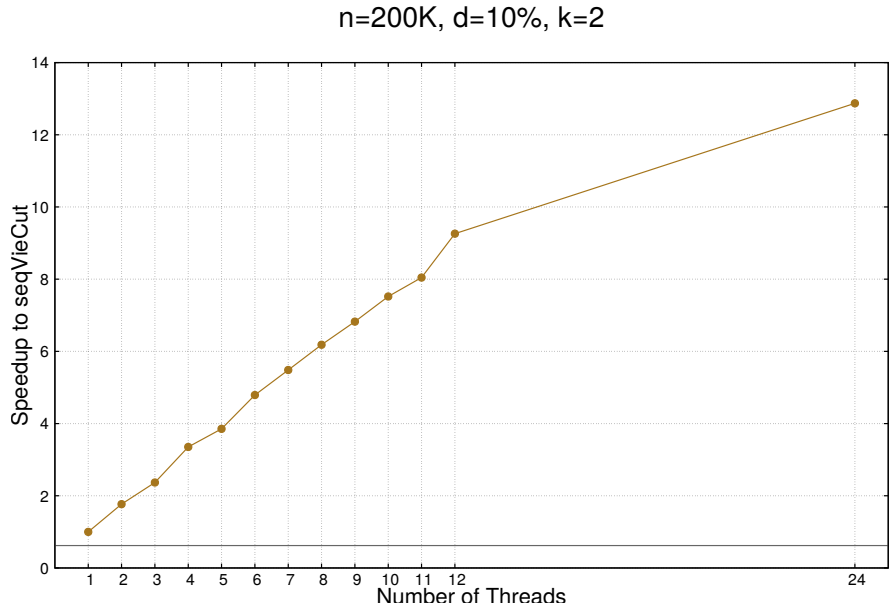
# Real-world Networks




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

# Parallelization



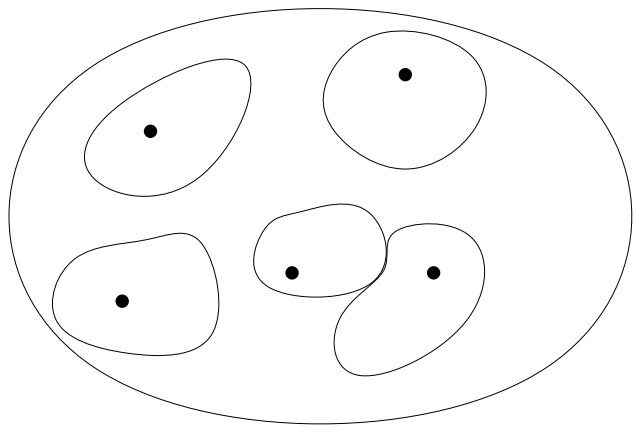

- Average speedup using 12 cores: 6.3 (24: 7.9)
- Average speedup to next fastest (Matula): 13.2 (24: 15.8)

# Stating the Obvious

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

$\rightsquigarrow$ PLUG IN our result into exact NOI algorithm + parallelization

$\rightsquigarrow$ currently fastest exact minimum cut algorithm
(by an order of magnitude)



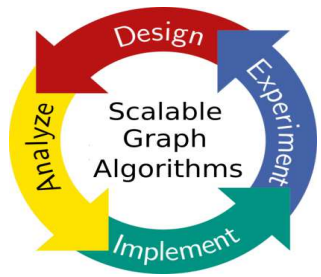
- also: extension to ALL minimum cuts, and dynamic minimum cuts

Multilevel Algorithms  Evolutionary Computation

Parallel Programming  Kernelization

Dynamic Algorithms

shared–, distributed–, external–, internal memory



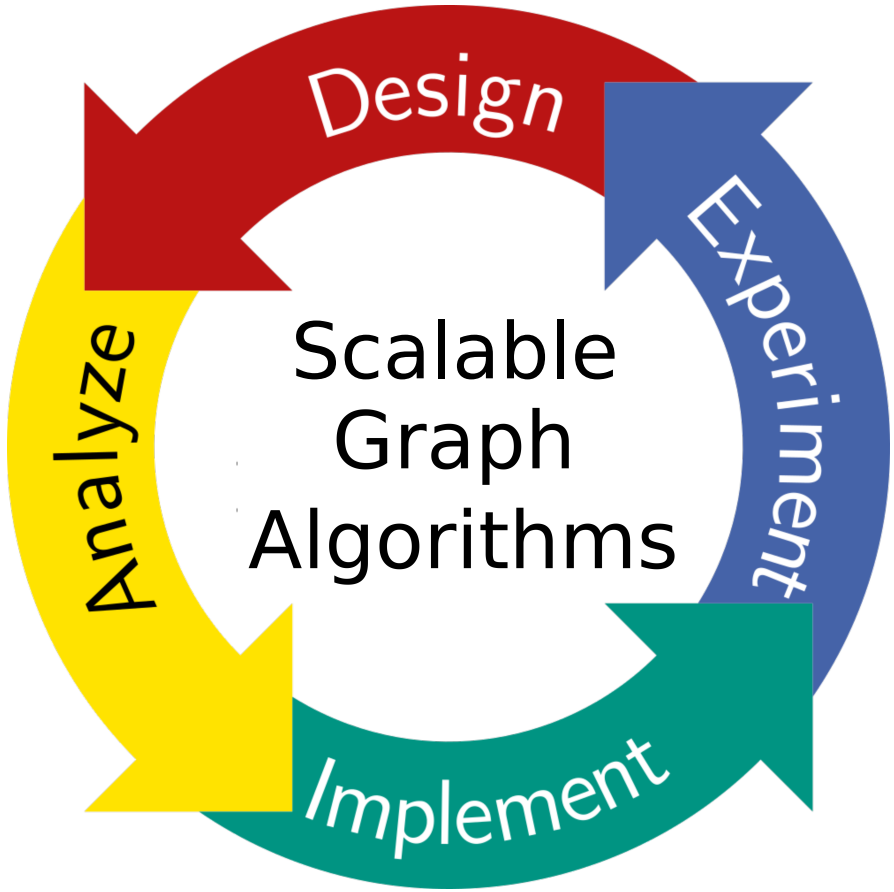

Algorithms for

| | | | |
|---|---|---|---|
| graph partitioning | process mapping | longest paths | node separators |
| graph clustering | minimum cuts | graph drawing | dyn. reachability |
| graph generation | independent sets | (dyn.) matching | .... |

Open Source

Applications

| | | |
|---|---|---|
| territory design | route planning | multiprocessor scheduling |
| large scale simulations | distributed system design | high–throughput DNA sequencing |
| quantum annealing | nuclei segmentation | .... |


Design
Experiment
Implement
Analyze
Scalable
Graph
Algorithms

backup slides for other problems

# Kernelization

**Nagamochi, Ono, and Ibaraki**

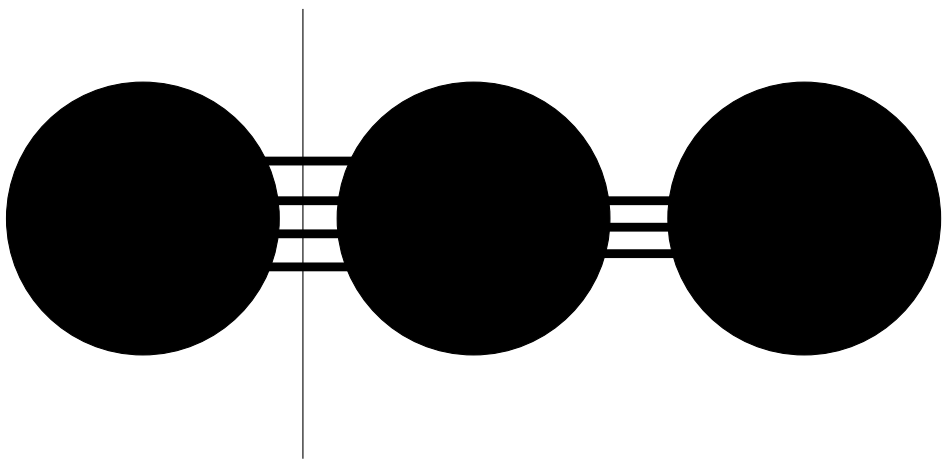
Key Idea: a spanning tree contains at least one edge from any cut

- 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**

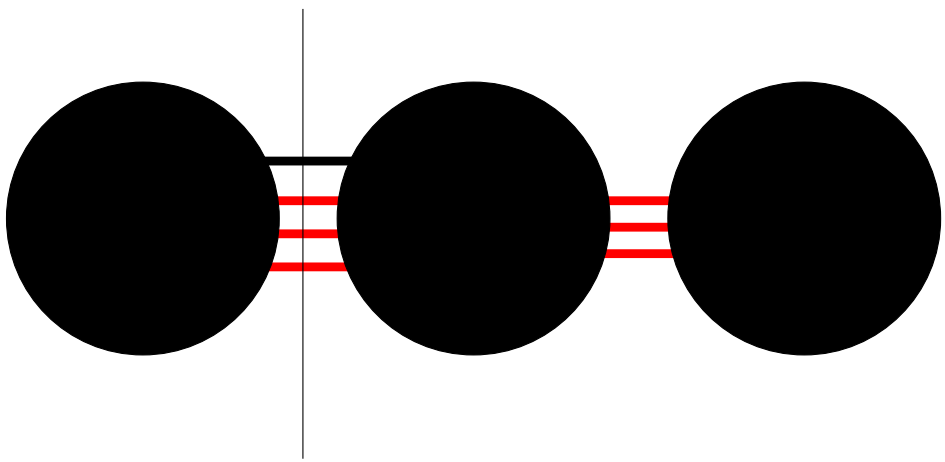
Key Idea: a spanning tree contains at least one edge from any cut

- 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**

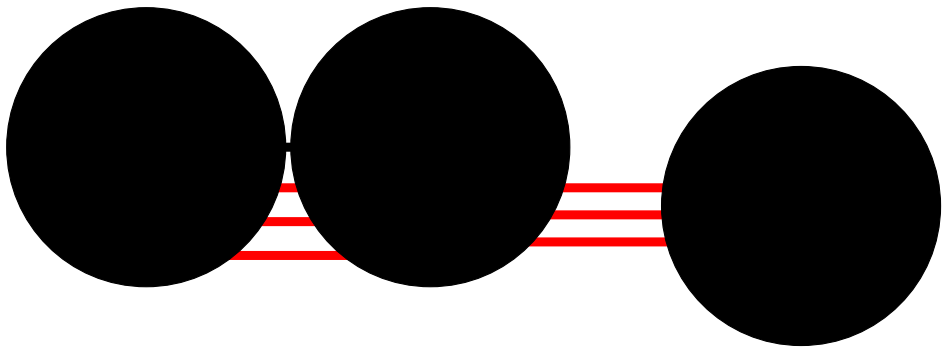
Key Idea: a spanning tree contains at least one edge from any cut

- 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 $r[v]$
    **for each** 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 $r[v]$
    **for each** 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

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 $r[v]$
    **for each** 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:

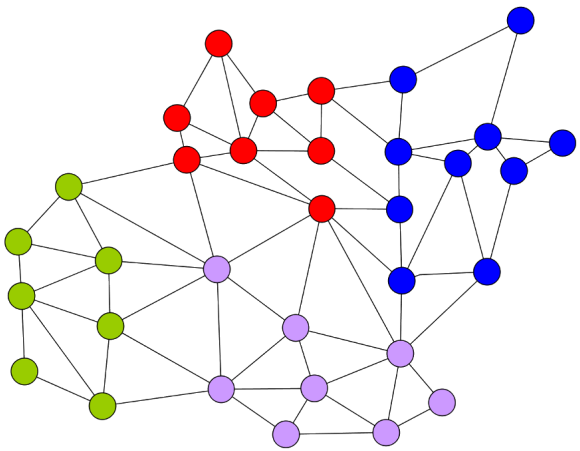
Everything in $E_{\hat{\lambda}} \cup \ldots \cup E_{|E|}$ can be contracted.
$\rightsquigarrow$ contract $e$ if $q(e) \geq \hat{\lambda}$

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

# Partitioning

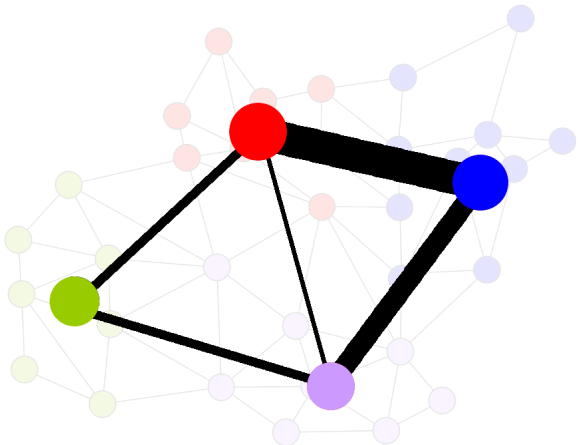
see lecture slides

# Process Mapping and ...




Single blocks represents *process* to be executed by a PE
$\mathcal{C}_{i,j}$ amount of communication between process $i$, process $j$

# Process Mapping and ...



$\rightarrow ?$


Single blocks represents *process* to be executed by a PE
$\mathcal{C}_{i,j}$ amount of communication between process $i$, process $j$

# ... and Quadratic Assignment

**[Brandfass et al. and others]**

$\mathcal{C} \in \mathbf{R}^{n \times n}$ communication matrix
$\mathcal{D} \in \mathbf{R}^{n \times n}$ distance matrix

**Quadratic assignment problem (QAP):**
Find one-to-one mapping $\Pi$ (processes to PEs) which minimizes

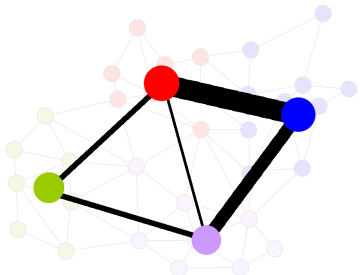
$$J(\mathcal{C}, \mathcal{D}, \Pi) := \sum_{i,j} \mathcal{C}_{\Pi(i),\Pi(j)} \mathcal{D}_{i,j}$$

$\mathcal{C}_{\Pi(i),\Pi(j)} \mathcal{D}_{i,j}$: cost for communication between PE $i$ and $j$

$\rightarrow J(\mathcal{C}, \mathcal{D}, \Pi)$ overall cost of communication

# Basic Assumptions

## ... Sparse Quadratic Assignment



$\rightarrow ?$


- communication matrix $\mathcal{C}$ is sparse $\rightarrow$ graph $G_{\mathcal{C}}$
- compute system is hierarchically structured

$\mathcal{S} = a_1, \ldots, a_k$ system hierarchy
e.g. $a_1$ cores per processor, $a_2$ processors per node, $a_3$ nodes per rack, ...

$D = d_1, \ldots, d_k$ distances in hierarchy
$d_i$ distance of cores in same subsystems for $i' < i$, different subsystems for $i' \geq i$

# Initial Solutions

**Intuition:**

- place well connected subgraphs in $G_{\mathcal{C}}$ closely

**Algorithms:**

- Identity
- Müller-Merbach (Greedy)
  - iteratively build mapping
  - assign process with largest comm. vol to core with smallest total distance
- Dual-Recursive-Bisection
  - **bi**partition $G_{\mathcal{C}}$ and $\mathcal{D}$ recursively
- Bottom up approach
- Top down approach

# Initial Solutions

**Intuition:**

- place well connected subgraphs in $G_C$ closely

**Top down approach:**

*Idea:* partition along the system hierarchy
$\mathcal{S} = a_1, a_2, ..., a_k$ hierarchy (with $n = \Pi_i a_i$)

compute *perfectly balanced* partition of $G_C$ into $a_k$ blocks
each block has $n/a_k$ vertices (processes)
each system entity provides $n/a_k$ PEs
$\rightarrow$assign blocks to system entities
recurse on each block with $a_{k-1}, \ldots$ until $a_1$ vertices left

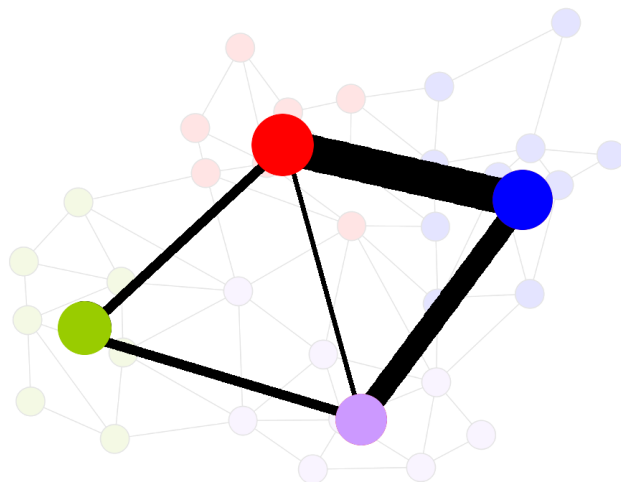
# Initial Solutions

## Top Down Illustrated

**System configuration $\mathcal{S}$**

| | |
|---|---|
| racks | 4 |
| nodes per rack | 4 |
| PEs per node | 4 |
| cores per PE | 4 |

**Model $G_{\mathcal{C}}$:**



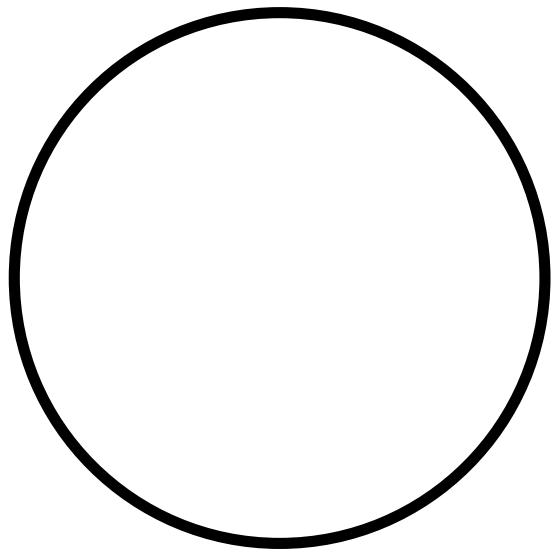
# Initial Solutions

## Top Down Illustrated

**System configuration $\mathcal{S}$**

| | |
|---|---|
| racks | 4 |
| nodes per rack | 4 |
| PEs per node | 4 |
| cores per PE | 4 |

**Model $G_{\mathcal{C}}$:**



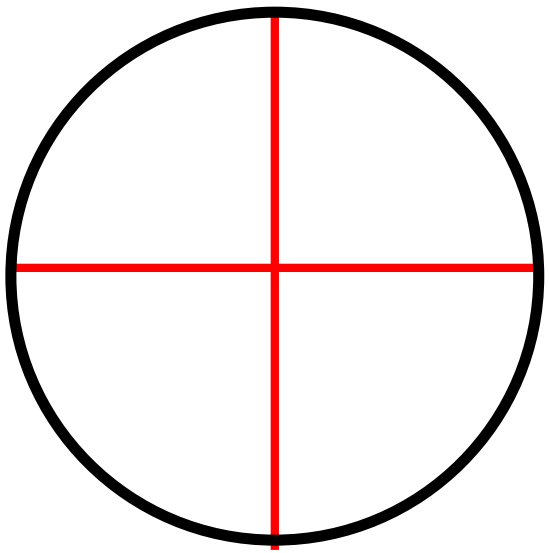
# Initial Solutions

## Top Down Illustrated

**System configuration $\mathcal{S}$**

| | |
|---|---|
| racks | 4 |
| nodes per rack | 4 |
| PEs per node | 4 |
| cores per PE | 4 |

**Model $G_{\mathcal{C}}$:**



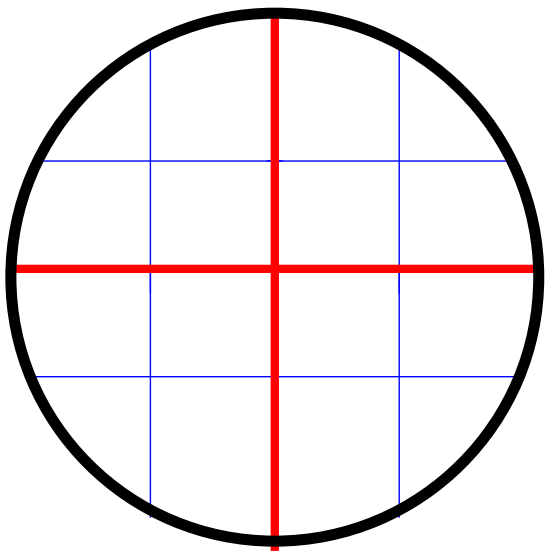
# Initial Solutions

## Top Down Illustrated

**System configuration $\mathcal{S}$**

| | |
|---|---|
| racks | 4 |
| nodes per rack | 4 |
| PEs per node | 4 |
| cores per PE | 4 |

**Model $G_{\mathcal{C}}$:**



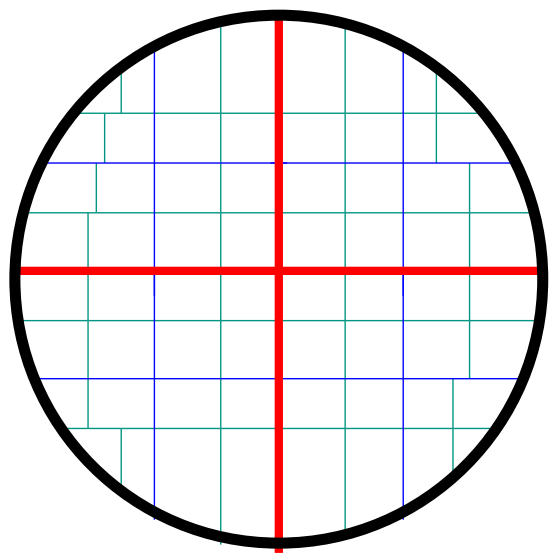
# Initial Solutions

## Top Down Illustrated

**System configuration $\mathcal{S}$**

| | |
|---|---|
| racks | 4 |
| nodes per rack | 4 |
| PEs per node | 4 |
| cores per PE | 4 |


**Model $G_{\mathcal{C}}$:**



# Local Search

[Heider'72]

**Goal:** perform swaps to improve mapping

**Swaps** in pair-exchange neighborhood $N(\Pi)$:

all permutations reachable by swapping two elements in $\Pi$

- swap $i, j$: $\Pi(i)$ assigned to PE $j$, $\Pi(j)$ is assigned to PE $i$
- check $N(\Pi)$ in cyclic manner
- keep swap if objective reduced (else undo)

**Overall runtime** $O(n^3)$

- quadratic amount of pairs
- each swap takes $O(n)$ time (update objective)

# Local Search

**Modifications by [Brandfass et al.'13]**

$\mathcal{N}_p$ ***(pruned neighborhood)***

- remove pairs $(i, j)$ if objective cannot change
- partition search space into $s$ consecutive index blocks
  $\rightarrow$ only perform swaps inside those blocks
- number of pairs $O(ns) \rightarrow$ overall runtime $O(n^2 s)$

$\Pi$ :


**Considered initial solutions:**

- Identity
- Müller-Merbach

  $\rightarrow$ improves runtime of local search and objective

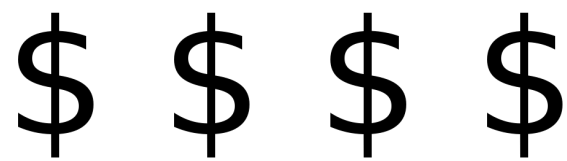
# Local Search

## Runtime Problem

Computing and updating objective is expensive!

$$J(\mathcal{C}, \mathcal{D}, \Pi) := \sum_{i,k} \mathcal{C}_{\Pi(i),\Pi(k)} \mathcal{D}_{i,k}$$

- matrices have quadratic number of elements
  $\rightarrow$ initial computation of objective $O(n^2)$
- update objective using old value:
  swap $(i, j)$ needs access to all elements
  in two columns of **communication and distance matrix**
  $\rightarrow$update step takes $O(n)$ time
  $\rightarrow$bottleneck for sparse $G_{\mathcal{C}}$



# Faster Local Search

**Observation**: initial objective can be computed in $O(n+m)$

rewrite objective to work with inverse of permutation:

$$J(\mathcal{C},\mathcal{D},\Pi) = \sum_{i,k} \mathcal{C}_{\Pi(i),\Pi(k)} \mathcal{D}_{i,k}$$
$$= \sum_{u,v} \mathcal{C}_{u,v} \mathcal{D}_{\Pi^{-1}(u),\Pi^{-1}(v)}$$

with the interpretation that task $u$ is assigned to PE $\Pi^{-1}(u)$

rewrite objective to work with graph representation:

$$J(\mathcal{C},\mathcal{D},\Pi) := \sum_{(u,v)\in E[\mathcal{C}]} \mathcal{C}_{u,v} \mathcal{D}_{\Pi^{-1}(u),\Pi^{-1}(v)}$$

# Faster Local Search

**Next goal:** *make update of objective fast*
Vertex contribution to objective:

$$\Gamma_{\Pi^{-1}}(u) := \sum_{v \in N(u)} \mathcal{C}_{u,v} \mathcal{D}_{\Pi^{-1}(u),\Pi^{-1}(v)}$$

Rewrite objective using vertex contribution

$$J(\mathcal{C}, \mathcal{D}, \Pi) := \sum_{u \in V} \Gamma_{\Pi^{-1}}(u)$$

**Throughout the algorithm:**

- keep vertex contributions $\Gamma$ up to date
- swap affects nodes $u$, $v$ and their neighborhood
  $\rightarrow$ update node contributions of those nodes
  $\rightarrow$ update objective accordingly
- $O(d_u + d_v)$ time

# Alternative Local Search Spaces

**Simple version $N_{\mathcal{C}}$:**
swap only if there is an edge in $G_{\mathcal{C}}$

- $|N_{\mathcal{C}}| = m$ (contains exactly $m$ pairs)
- swaps are performed in random order
- terminate after $m$ unsuccessful swaps
- assumes that swaps with positive gain are close in $G_{\mathcal{C}}$

**Augmented version $N_{\mathcal{C}}^d$:**
swaps only if processes have distance less than $d$ in $G_{\mathcal{C}}$

# Benchmark Instances

**Two different configurations:**

$\mathcal{S} = a_1, \ldots, a_k$ system hierarchy
$D = d_1, \ldots, d_k$ distances in hierarchy

$d_i$ distance of two cores in same subsystems for $i' < i$, and different subsystems for $i' \geq i$

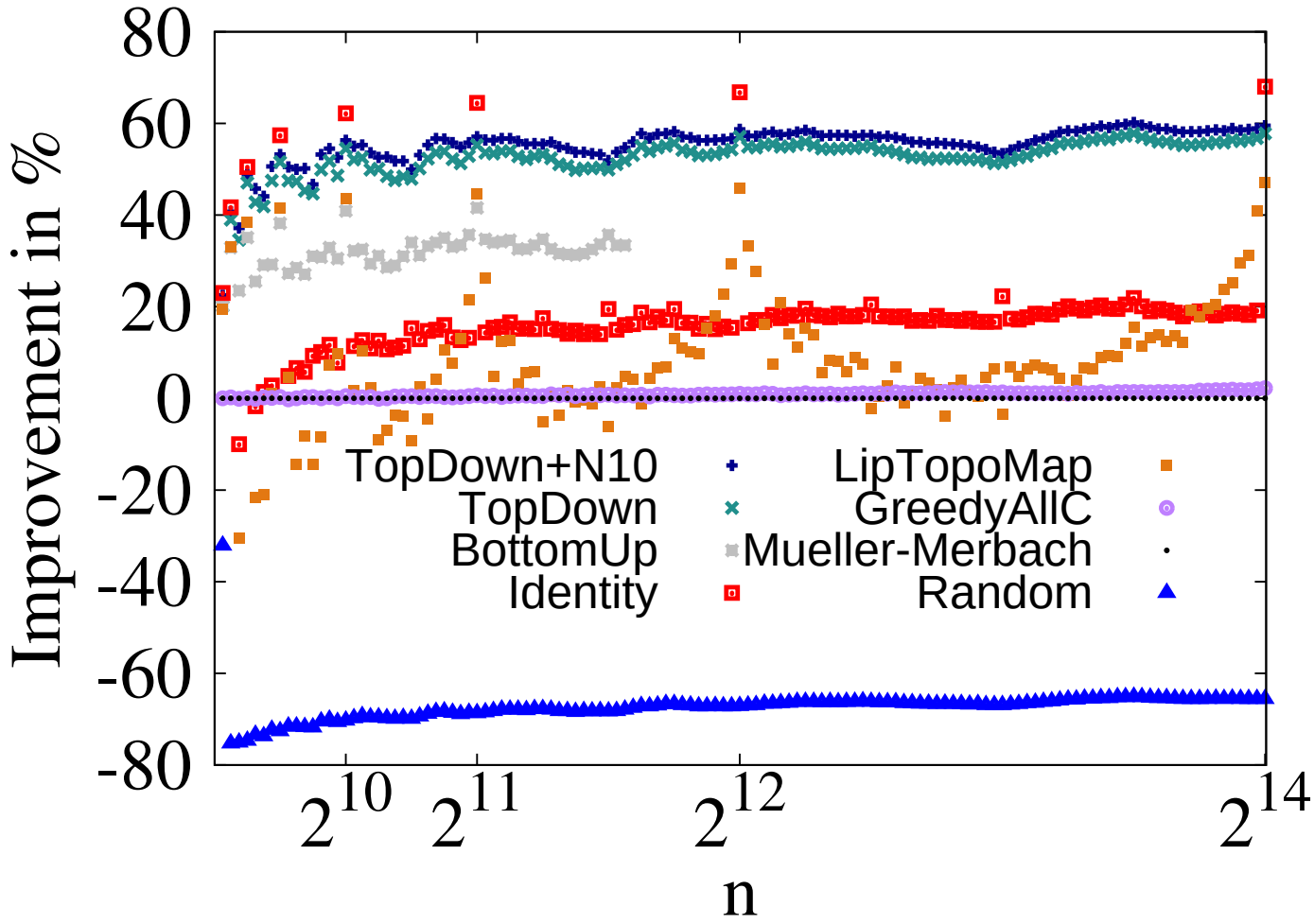
$n = \prod_i a_i$ total number of cores

**Process to create instances:**

- partition graph into $n$ blocks
- compute the communication graph
- use edge cut as communication volume
- compute mapping to specified system

| Graph | $n$ | $m$ |
|---|---|---|
| UF Graphs | | |
| cop20k_A | 99 843 | 1 262 244 |
| 2cubes_sphere | 101 492 | 772 886 |
| thermomech_TC | 102 158 | 304 700 |
| cfd2 | 123 440 | 1 482 229 |
| boneS01 | 127 224 | 3 293 964 |
| Dubcova3 | 146 689 | 1 744 980 |
| bmwcra_1 | 148 770 | 5 247 616 |
| G2_circuit | 150 102 | 288 286 |
| shipsec5 | 179 860 | 4 966 618 |
| cont-300 | 180 895 | 448 799 |
| Large Walshaw Graphs | | |
| 598a | 110 971 | 741 934 |
| fe_ocean | 143 437 | 409 593 |
| 144 | 144 649 | 1 074 393 |
| wave | 156 317 | 1 059 331 |
| m14b | 214 765 | 1 679 018 |
| auto | 448 695 | 3 314 611 |
| Large Other Graphs | | |
| del23 | ≈8.4M | ≈25.2M |
| del24 | ≈16.7M | ≈50.3M |
| rgg23 | ≈8.4M | ≈63.5M |
| rgg24 | ≈16.7M | ≈132.6M |
| deu | ≈4.4M | ≈5.5M |
| eur | ≈18.0M | ≈22.2M |
| af_shell9 | ≈504K | ≈8.5M |
| thermal2 | ≈1.2M | ≈3.7M |
| nlr | ≈4.2M | ≈12.5M |

# Initial Heuristics




baseline: Müller-Merbach

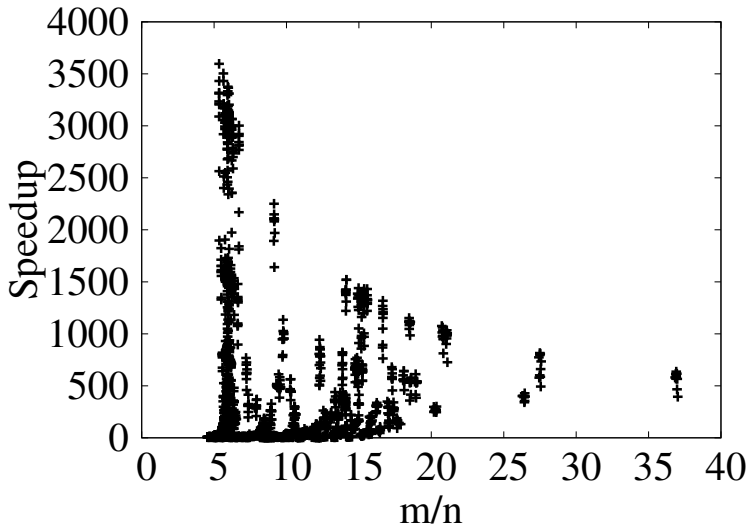
# Speedup of Local Search

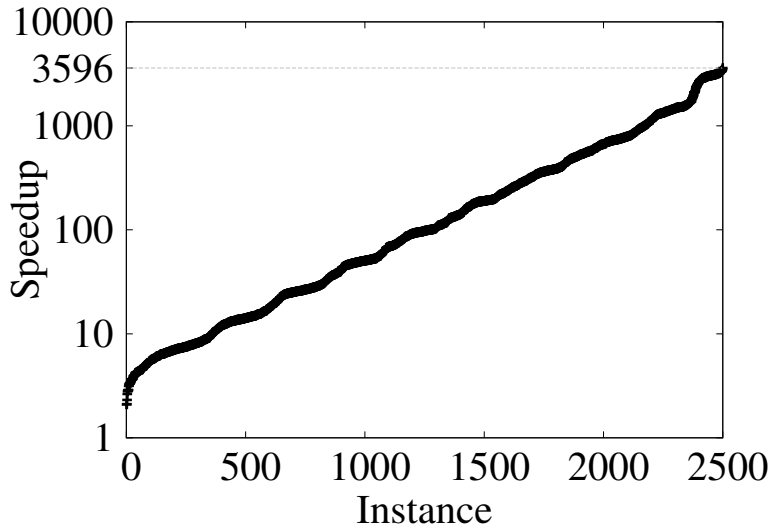
**Running Time and Speedup of Local Search for $N_p$**

| $n$ | $\overline{m/n}$ | $t_{\text{LS}}[s]$ | $t_{\text{fastLS}}$[s] | speedup |
|---|---|---|---|---|
| 64 | 6.7 | 0.016 | 0.003 | 5.3 |
| 128 | 7.3 | 0.064 | 0.006 | 10.7 |
| 256 | 7.9 | 0.268 | 0.014 | 19.1 |
| 512 | 8.3 | 1.073 | 0.029 | 37.0 |
| 1K | 8.8 | 4.263 | 0.059 | 72.3 |
| 2K | 9.2 | 17.083 | 0.124 | 137.8 |
| 4K | 9.7 | 68.360 | 0.260 | 262.9 |
| 8K | 10.3 | 268.907 | 0.540 | 498.0 |
| 16K | 11.2 | 1 075.107 | 1.158 | 928.4 |
| 32K | 12.5 | 4 348.374 | 2.472 | 1 759.1 |

- $t_{\text{LS}}$ running time using slow gain computations
- $t_{\text{fastLS}}$ running time using fast gain computations

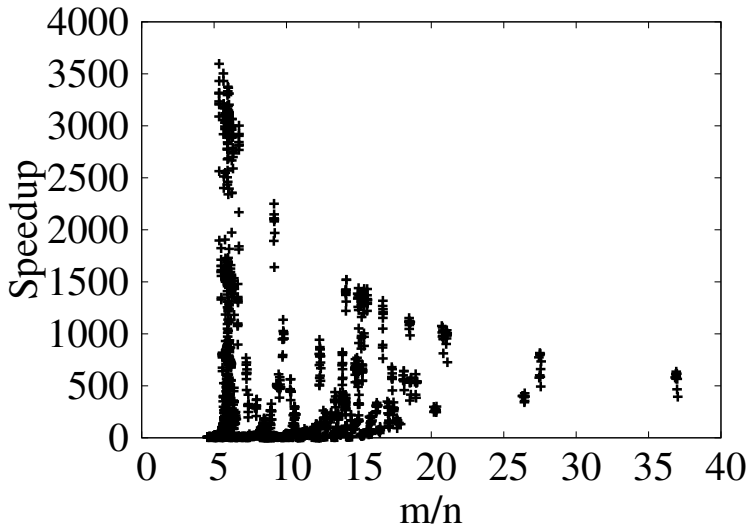
# Faster Local Search

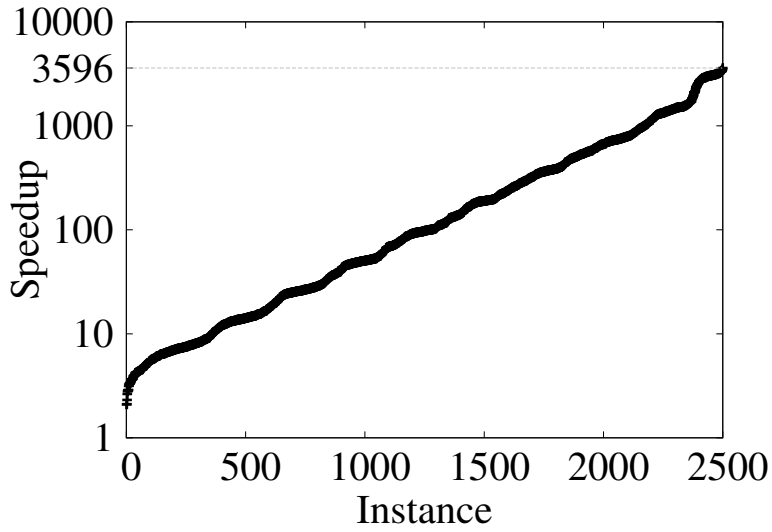



- algorithmic speedup as a function of graph density
- algorithmic speedup for the different instances

# Faster Local Search




- algorithmic speedup as a function of graph density
- algorithmic speedup for the different instances

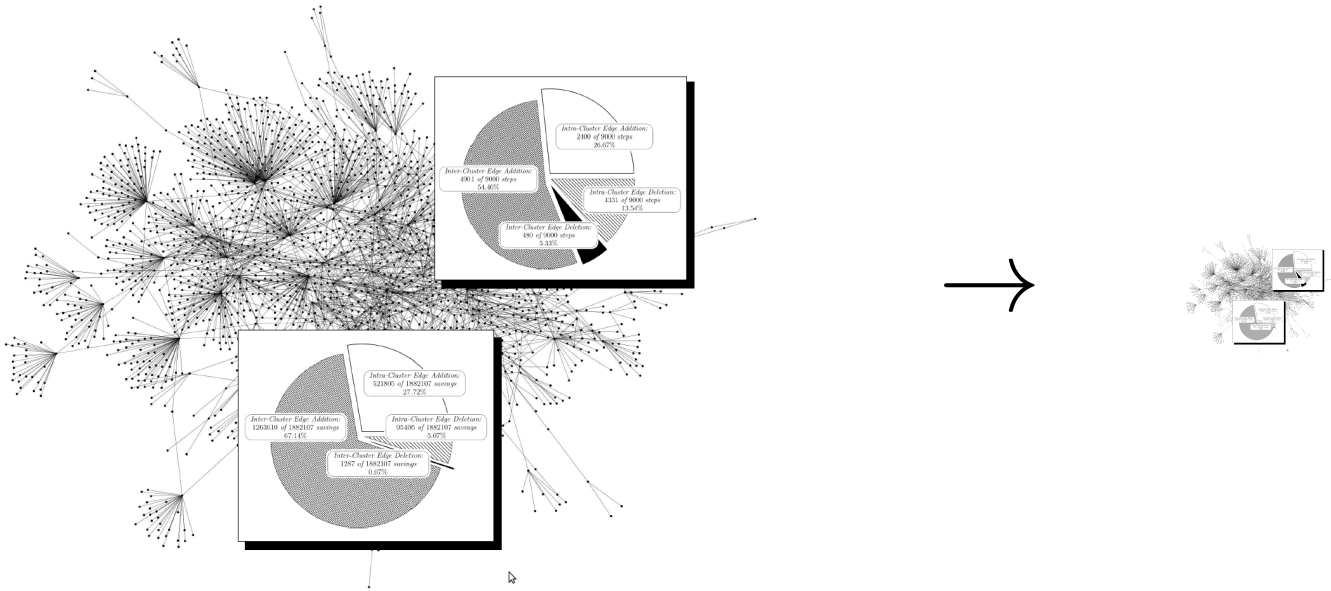
Vienna Mapping and Sparse Quadratic Assignment:
viem.taa.univie.ac.at

# Concrete Example
## Applied Kernelization for Independent Sets

# Kernelization

## General Idea

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

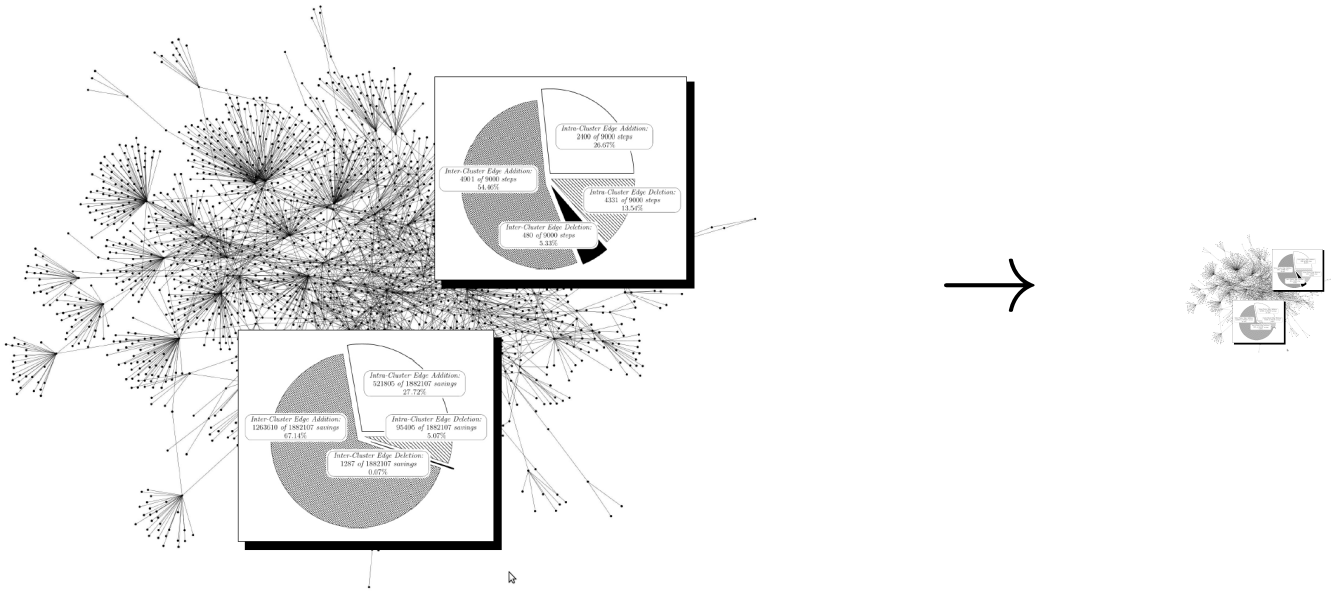



solve problem on problem kernel
$\rightarrow$ obtain solution on input graph

# Kernelization

## General Idea

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

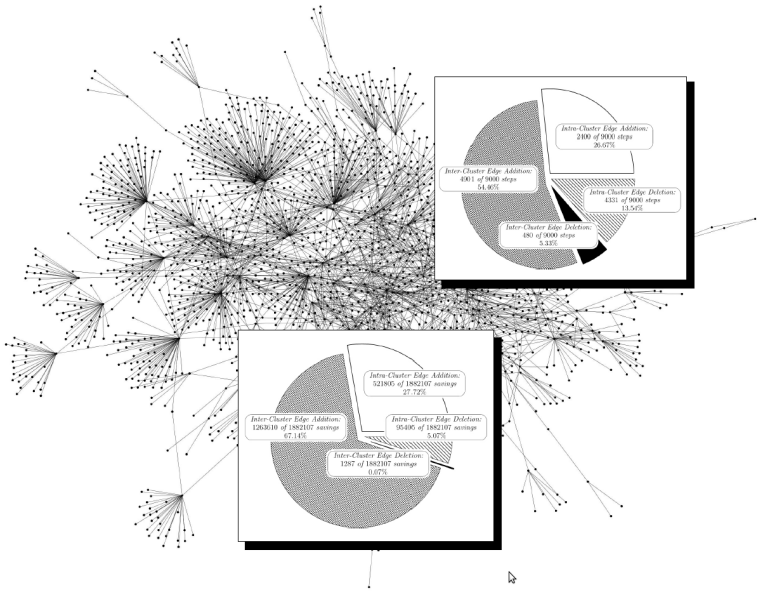



solve problem on problem kernel (using a heuristic)
$\rightarrow$ obtain solution on input graph quickly

# 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]

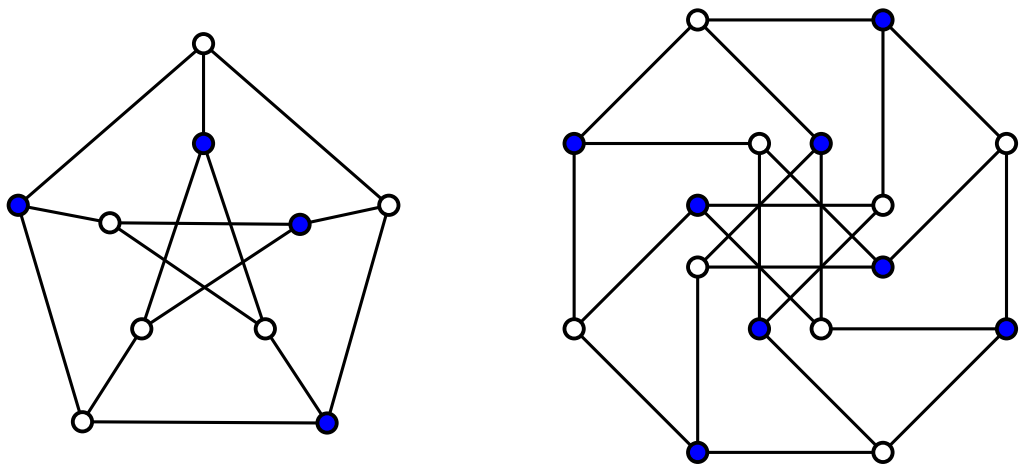
# Definitions

**Independent Set**:

- subset $S \subseteq V$ such that there are no adjacent nodes in S

**Maximum Independent Set (MIS):**

- maximum cardinality set $S$



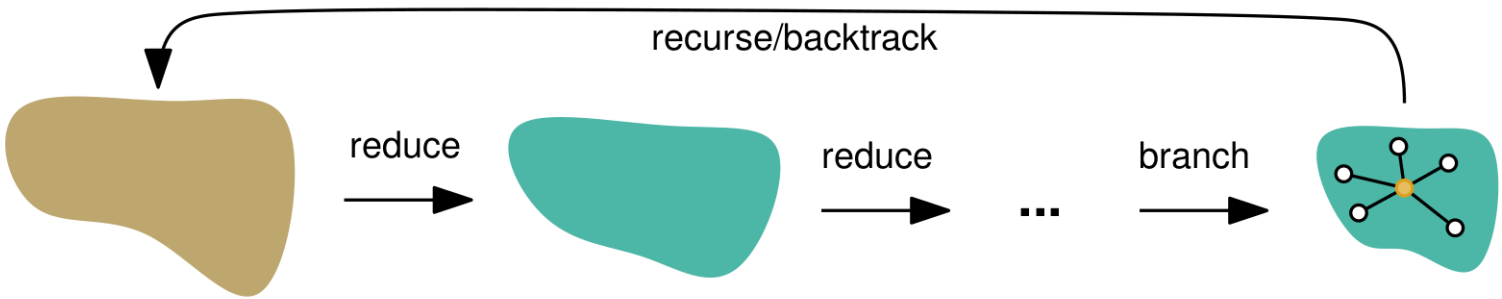
- related to maximum clique and minimum vertex cover
- finding a MIS is **NP-hard** and hard to approximate

# Exact Algorithms

## Independent Sets

Exact solution using **branch and bound**:

- Modify and remove subgraphs during recursions $\rightarrow$ **reductions**
- Branch when the graph can no longer be reduced




$\rightarrow$ Running time $O(1.211^n)$ [Akiba, Iwata'16]

Works well, except when it doesn't:

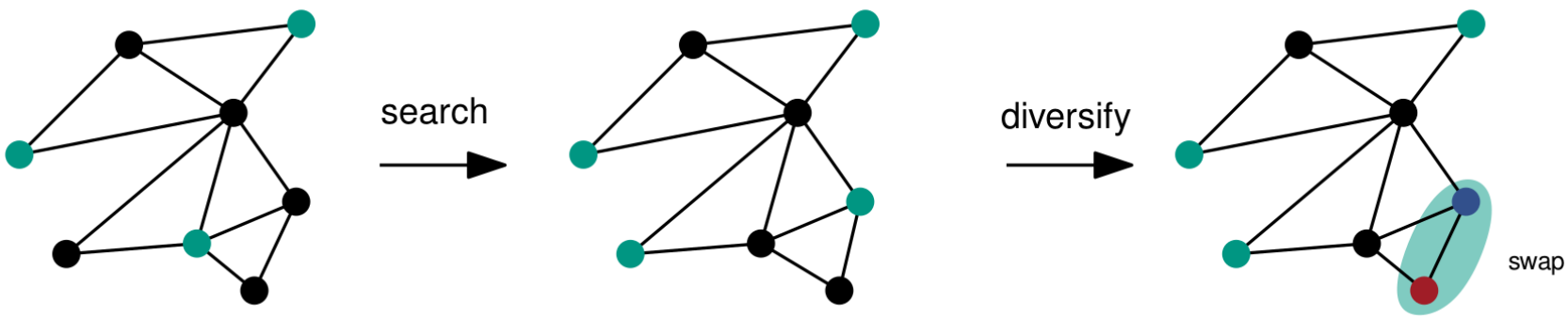
- Many networks with **small kernels** are easy to solve
- Similar instances with **large kernels** remain unsolved

# Heuristic Algorithms

Huge real-world networks infeasible for exact algorithms
$\rightarrow$ finding **high-quality** approximations in short time

**Recent Approaches:**

- Local search based on swaps [Andrade et al. 2012]

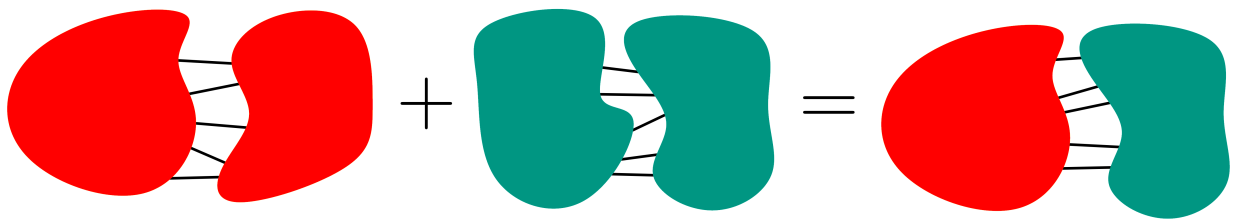


# Heuristic Algorithms

Huge real-world networks infeasible for exact algorithms
$\rightarrow$ finding **high-quality** approximations in short time

**Recent Approaches:**

- Local search based on swaps [Andrade et al. 2012]
- Evolutionary algorithms (EvoMIS) [SEA'15]

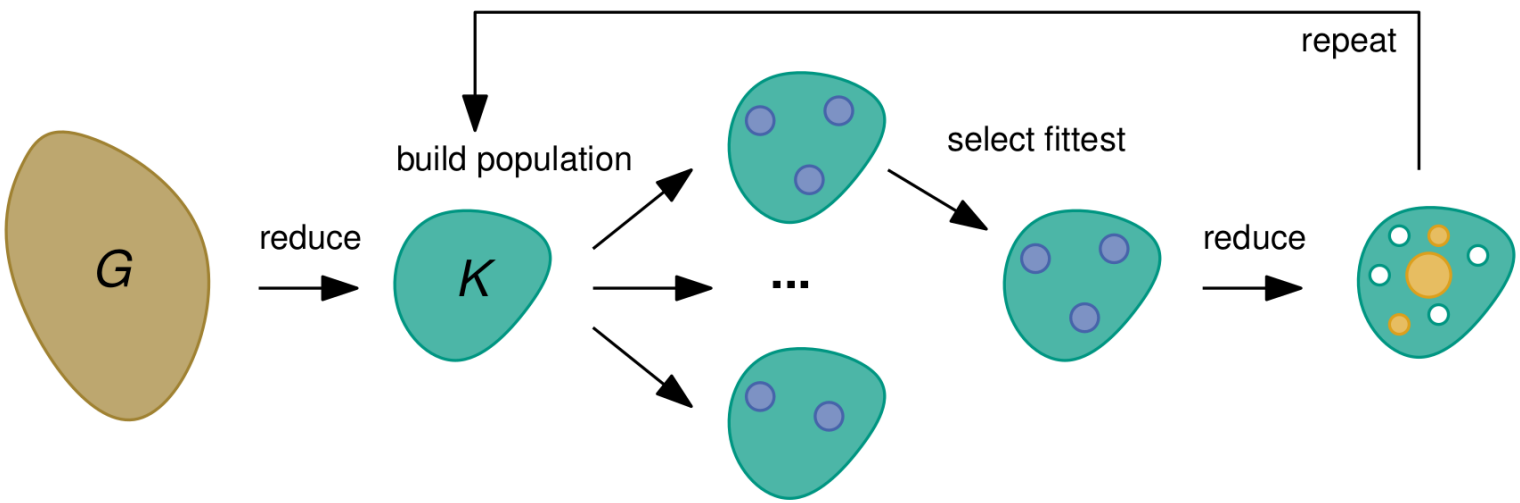


# Heuristic Algorithms

Huge real-world networks infeasible for exact algorithms
$\rightarrow$ finding **high-quality** approximations in short time

**Recent Approaches:**

- Local search based on swaps [Andrade et al. 2012]
- Evolutionary algorithms (EvoMIS) [SEA'15]
- Combine with reductions (ReduMIS) [ALENEX'16]

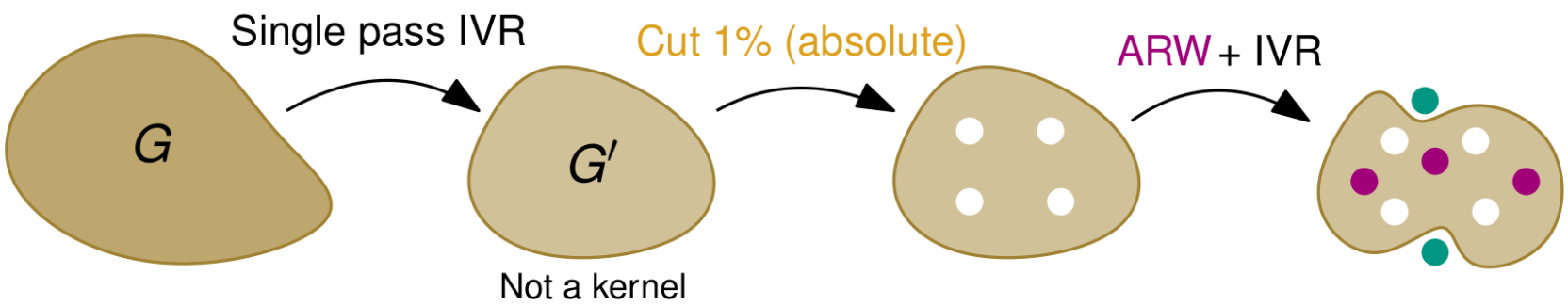


# Heuristic Algorithms

Huge real-world networks infeasible for exact algorithms
$\rightarrow$ finding **high-quality** approximations in short time

**Recent Approaches:**

- Local search based on swaps [Andrade et al. 2012]
- Evolutionary algorithms (EvoMIS) [SEA'15]
- Combine with reductions (ReduMIS) [ALENEX'16]
- Online reductions (OnlineMIS) [SEA'16]

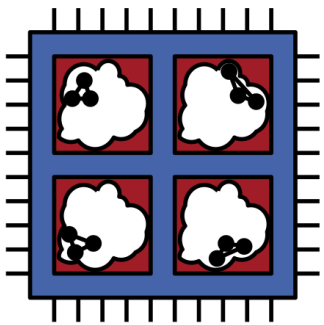


# Heuristic Algorithms

Huge real-world networks infeasible for exact algorithms
$\rightarrow$ finding **high-quality** approximations in short time

**Recent Approaches:**

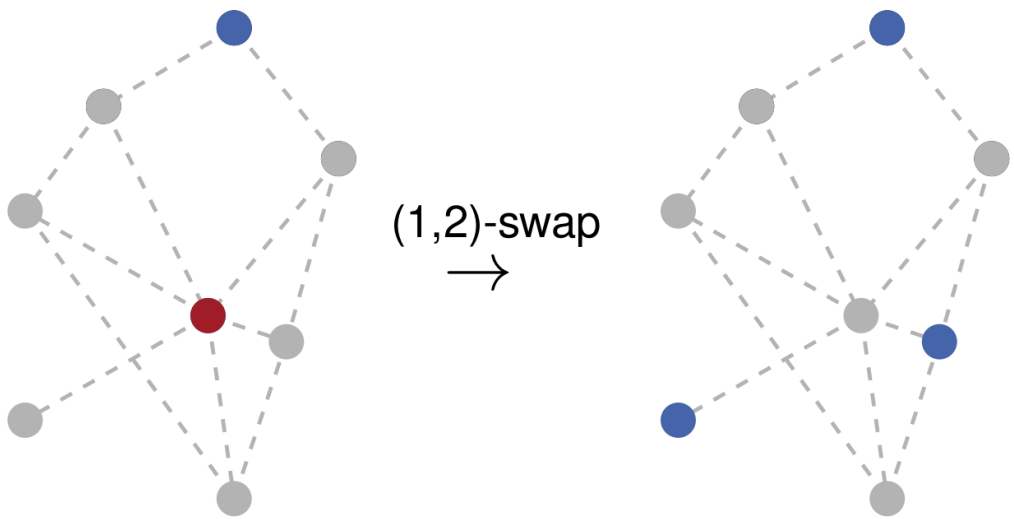
- Local search based on swaps [Andrade et al. 2012]
- Evolutionary algorithms (EvoMIS) [SEA'15]
- Combine with reductions (ReduMIS) [ALENEX'16]
- Online reductions (OnlineMIS) [SEA'16]
- Shared-Memory parallel (ParMIS) [ALENEX'17]



# Iterated Local Search

[Andrade et al.'12]

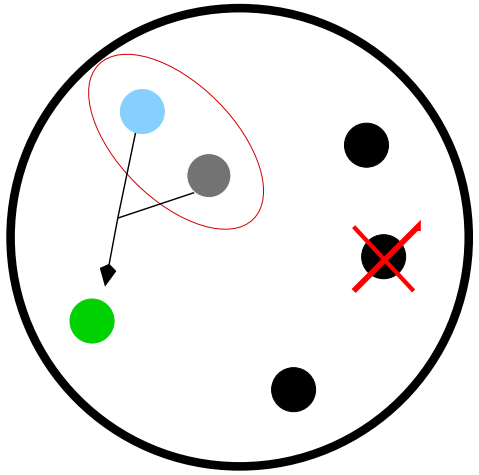
- Compute initial maximal independent set
- Improve using $(1,2)$-swaps:
  - remove single solution node and insert two new ones
  - search for $(1,2)$-swaps in time $O(m)$
- **Tabu mechanism** for "recently" swapped vertices
- No (1,2)-swap $\rightarrow$ **perturbation step**



# Evolutionary Algorithm

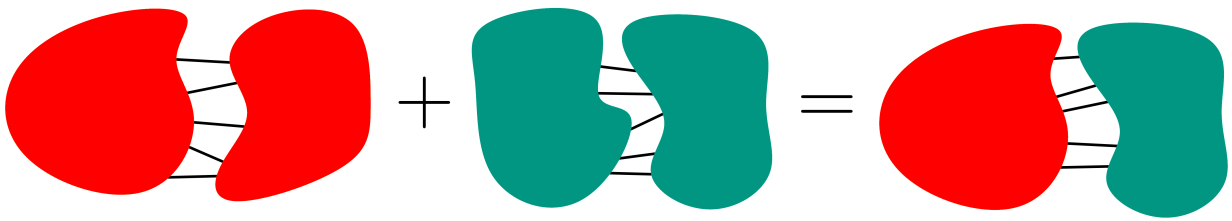
[SEA'15]

create initial population $\mathcal{P}$
**while** stopping criterion not fulfilled
  *select* parents $\mathcal{P}_1$, $\mathcal{P}_2$ from $\mathcal{P}$
  *recombine* $\mathcal{P}_1$ with $\mathcal{P}_2$ to create offspring $o$
  *mutate* offspring $o$
  *evict* individual in population using $o$
**return** *fittest* individual



**Recombine Using Graph Partitioning:**

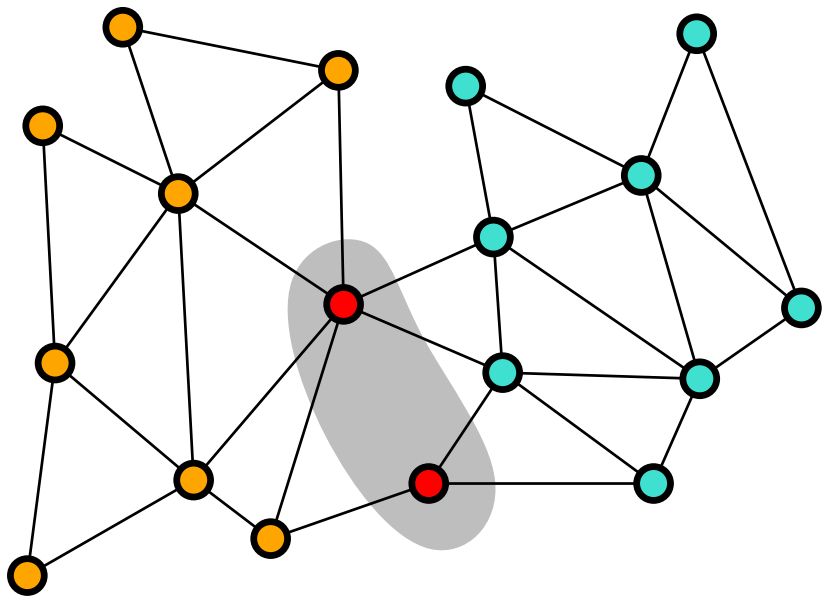
- exchange whole blocks of solutions
- small objective vital for efficiency



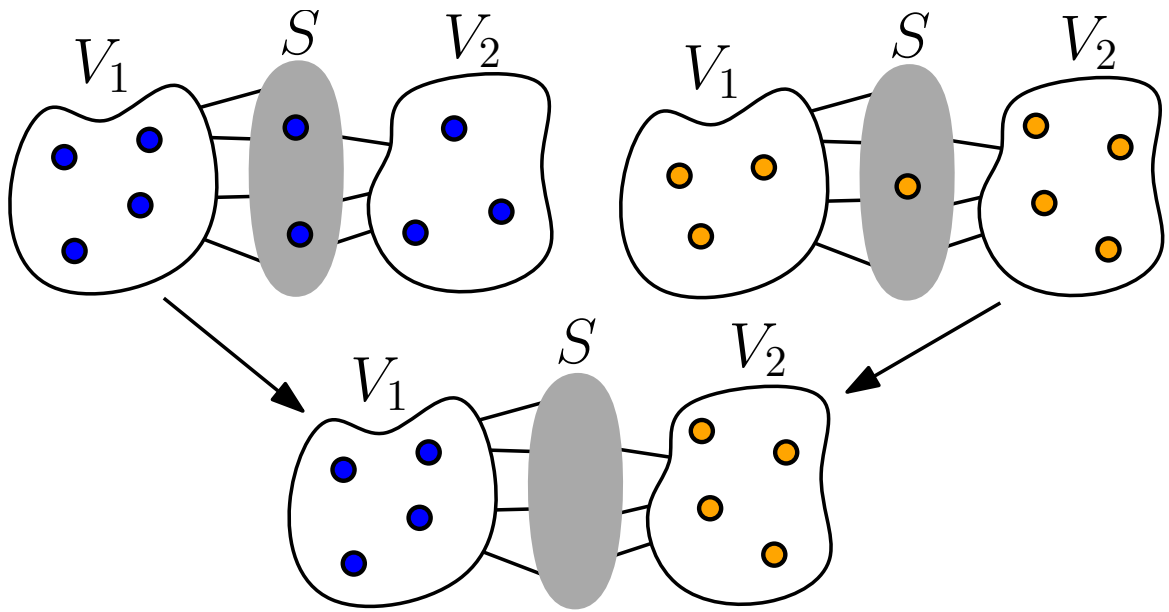
# Separators

Partition graph $G = (V, E, c : V \to \mathbf{R}_{>0}, \omega : E \to \mathbf{R}_{>0})$
into $k$ disjoint blocks + node separator s.t.

- total node weight of each block $\leq \frac{1+\epsilon}{k}$ total node weight
- total size of node separator as small as possible

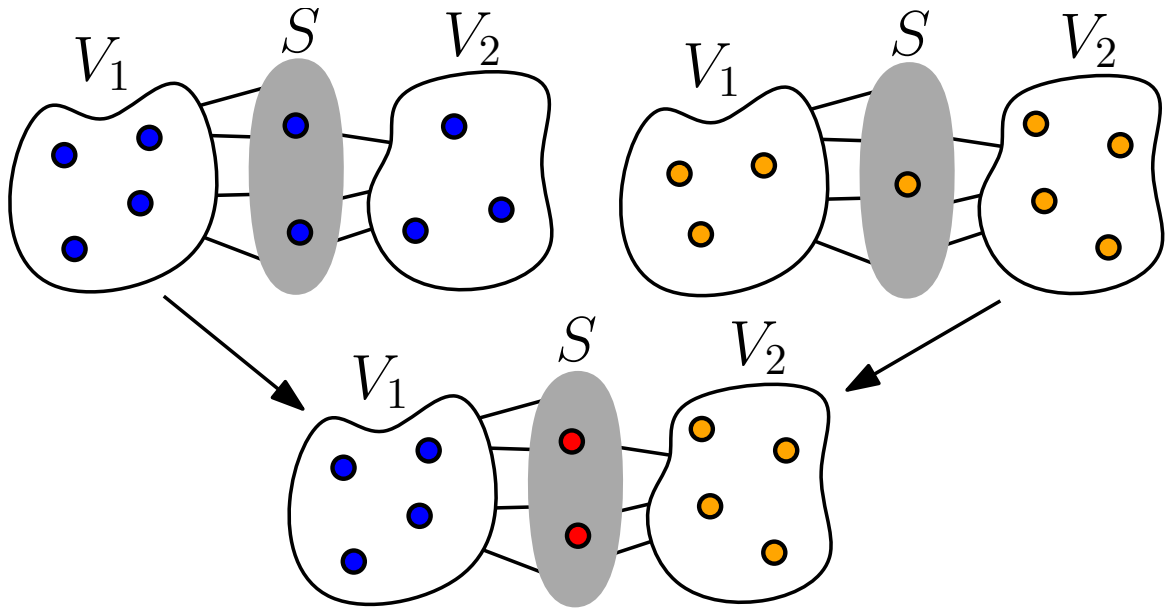


# Separator Combine



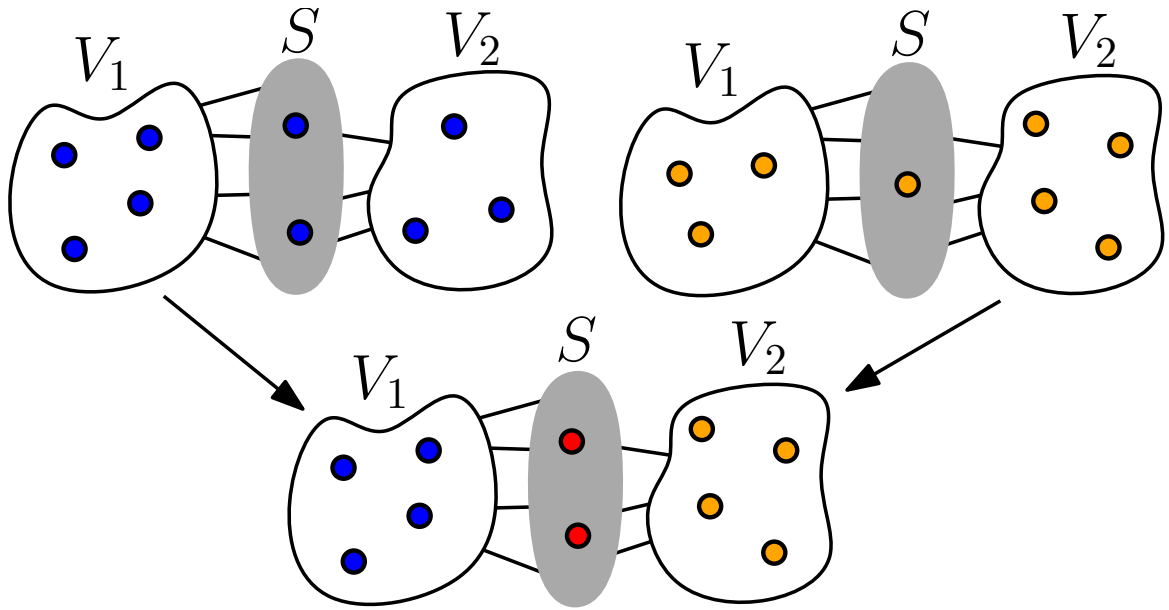

- build node separator $V = V_1 \cup V_2 \cup S$
- use node separator as crossover point
- combination takes linear time $O(n)$
- maximize with local search

# Separator Combine



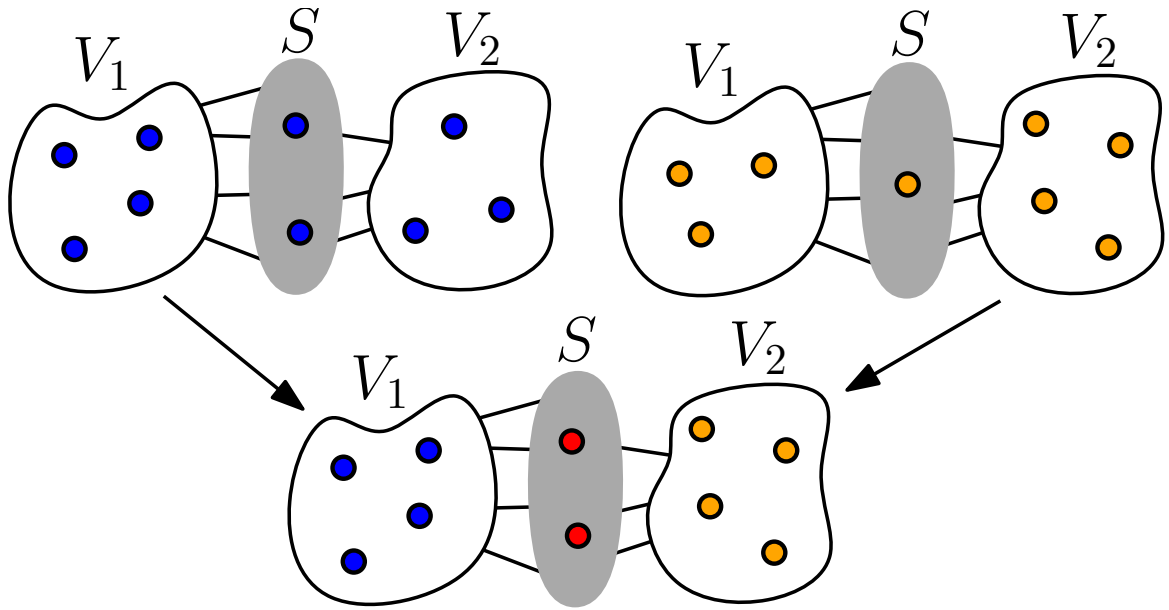

- build node separator $V = V_1 \cup V_2 \cup S$
- use node separator as crossover point
- combination takes linear time $O(n)$
- maximize with local search

# Separator Combine




Good Solutions, but Slow

# Separator Combine



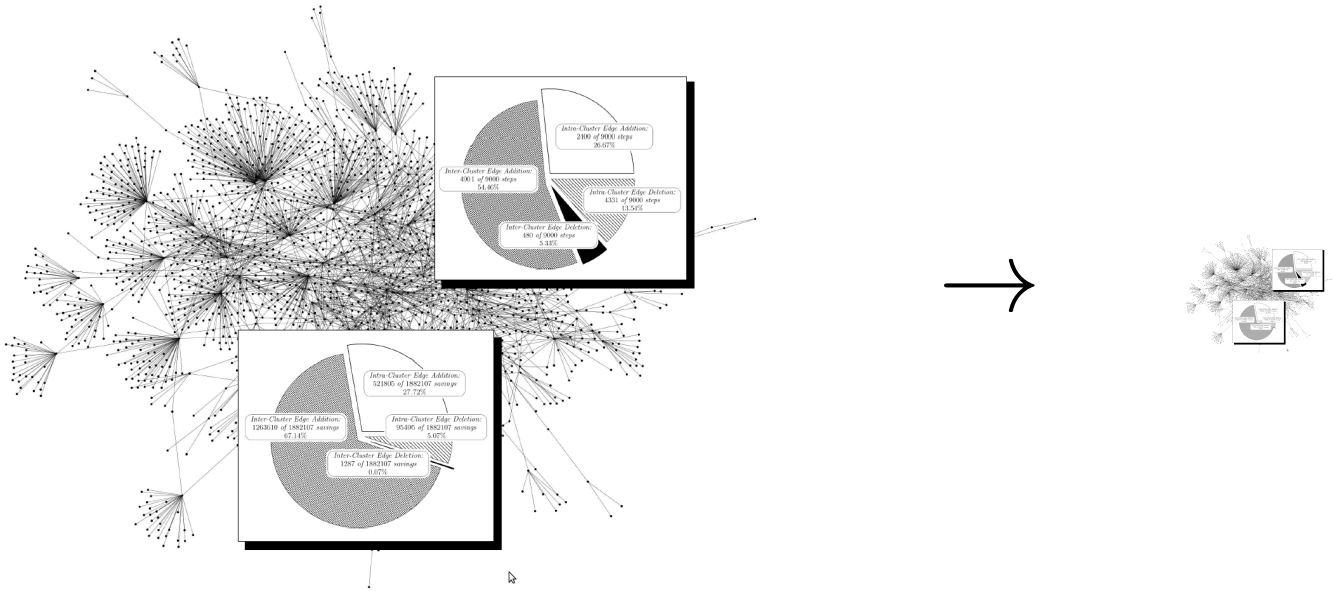

Good Solutions, but Slow

Apply EA on Kernel

# Kernelization

[Akiba, Iwata'15]

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



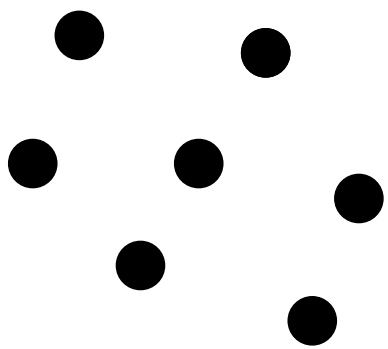

solve problem on problem kernel (using EA)
$\rightarrow$ obtain solution on input graph

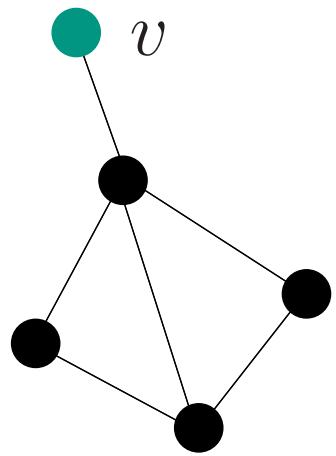
# Kernelization

[Akiba, Iwata'15]

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

**Example:**
remove degree 0 or 1 vertices recursively





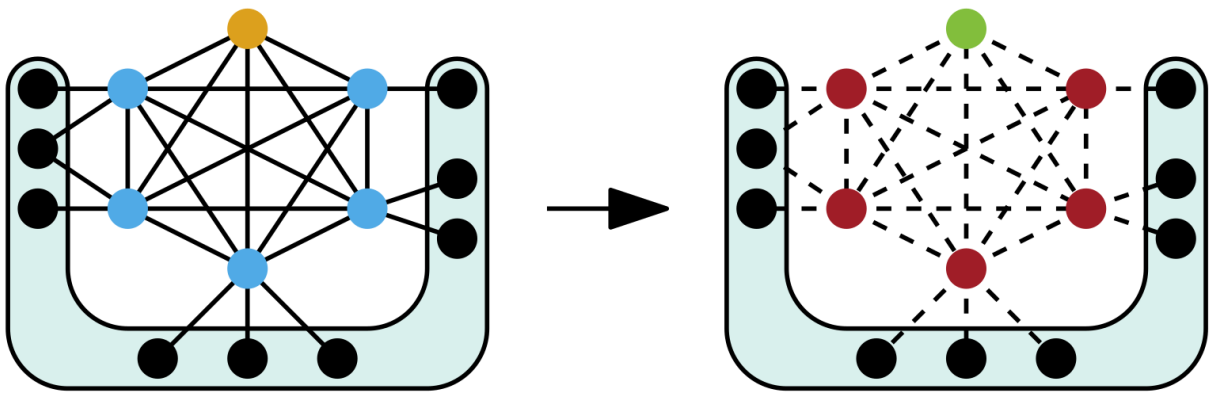

there is always a MIS that contains $v$
if neighbor of $v$ in MIS choose $v$ instead; otherwise add $v$

# Kernelization

[Akiba, Iwata'15]

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

**Example:**
isolated clique reduction

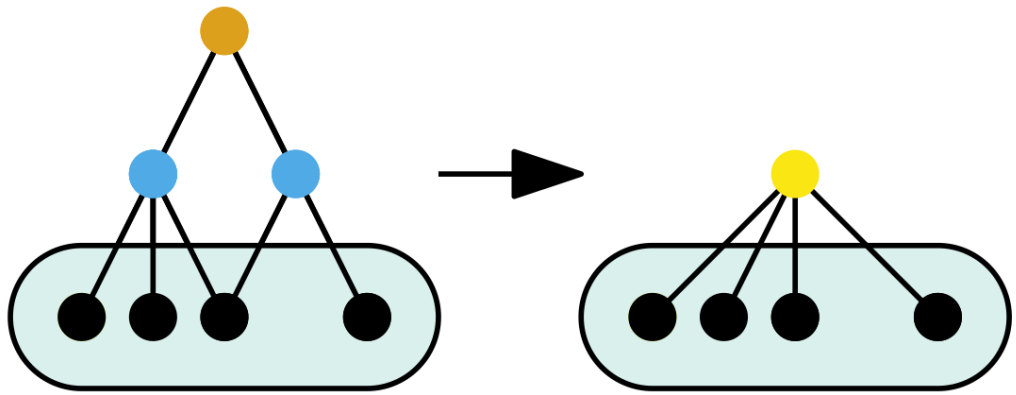


# Kernelization

[Akiba, Iwata'15]

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

**Example:**
vertex folding



# Kernelization

[Akiba, Iwata'15]

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

**Example:**
linear programming relaxation for MIS

$$\max \sum_{v \in V} x_v \text{ s.t.}$$
$$\forall (u,v) \in E : x_u + x_v \leq 1$$
$$\forall v \in V : x_v \geq 0$$

- $\exists$ half integral solution (i.e., using only values 0, 1/2, and 1)
- solve via maximum bipartite matching (also parallel, stay tuned)
- vertices with value 1, MUST be in the MIS

# Kernelization

[Akiba, Iwata'15]

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

**Example:**
linear programming relaxation for MIS

$$\max \sum_{v \in V} x_v \text{ s.t.}$$
$$\forall (u,v) \in E : x_u + x_v \leq 1$$
$$\forall v \in V : x_v \geq 0$$

- $\exists$ half integral solution (i.e., using only values 0, 1/2, and 1)
- solve via maximum bipartite matching (also parallel, stay tuned)
- vertices with value 1, MUST be in the MIS

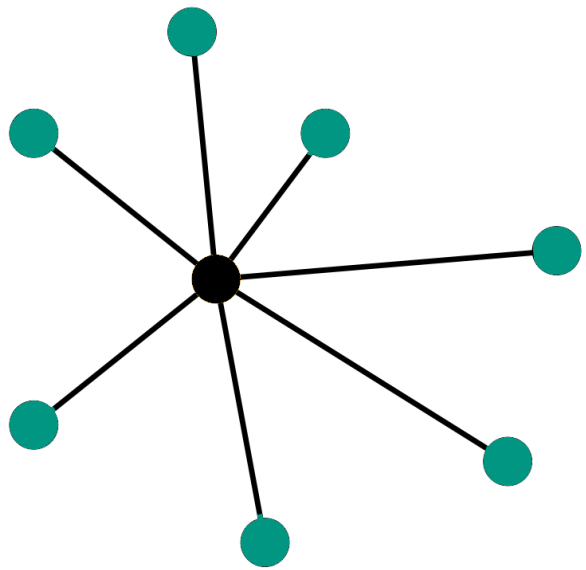
more reductions used in practice $\rightarrow$ [ALENEX'16]

# Guess “likely candidates”

can we guess vertices that are in MIS?

idea: select small-degree vertices from “fittest” independent set

apply more reductions and recurse!

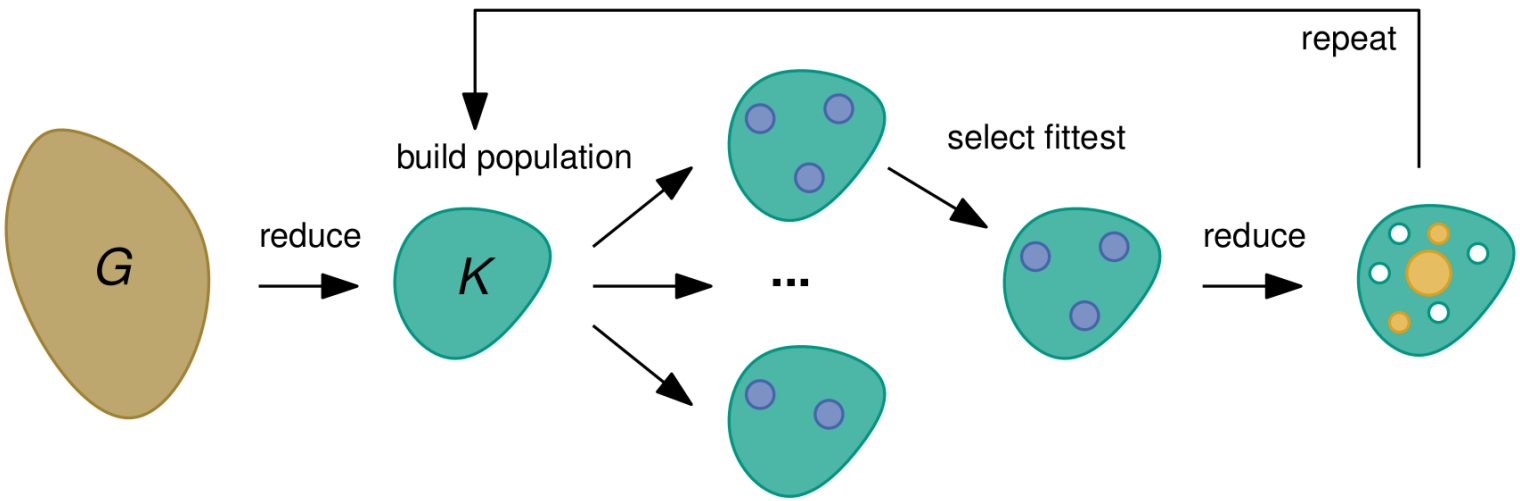


# Guess “likely candidates”

can we guess vertices that are in MIS?

idea: select small-degree vertices from “fittest” independent set

apply more reductions and recurse!

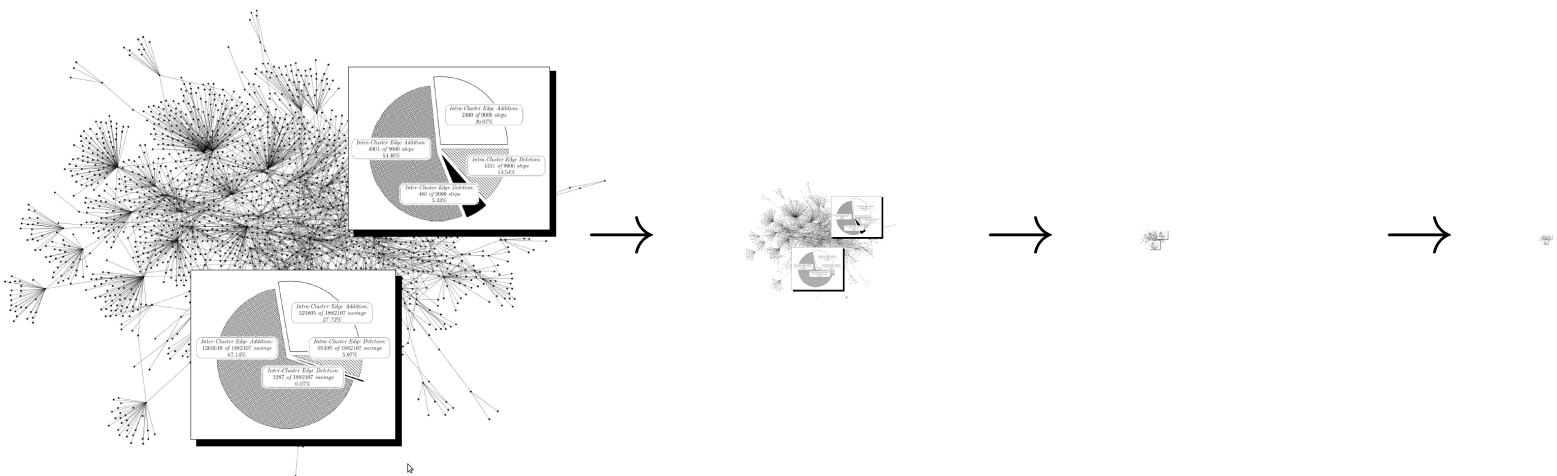


# Guess “likely candidates”

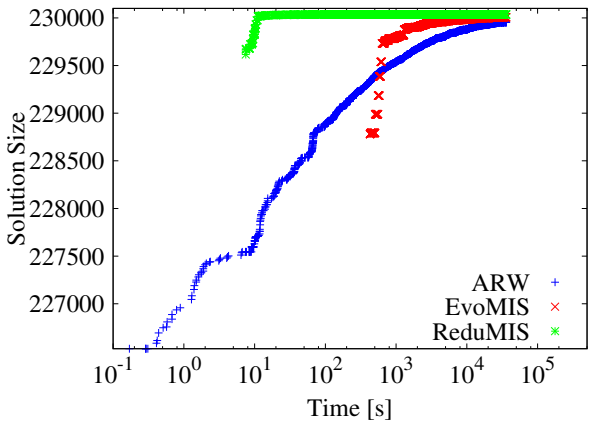
can we guess vertices that are in MIS?

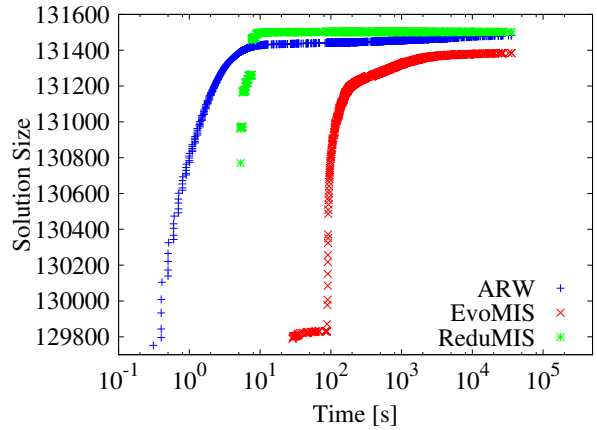
idea: select small-degree vertices from “fittest” independent set

apply more reductions and recurse!



# Near-optimal on “difficult networks”

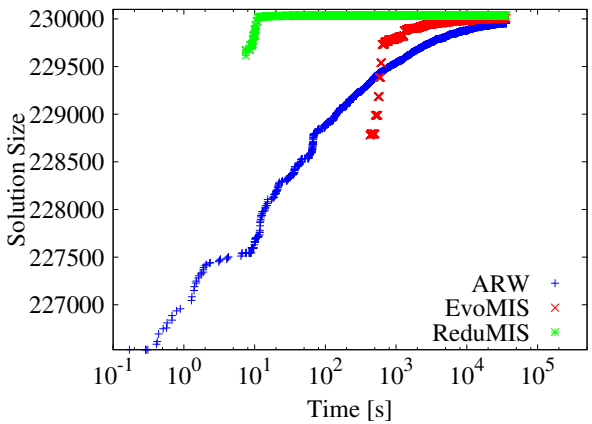


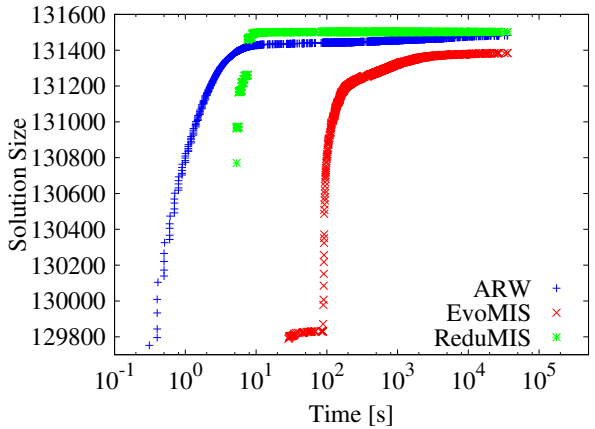




**Results:**

- finds exact MIS faster, when exact algorithm is slow:
  - Skitter **48 min** $\rightarrow$ **21 min**
  - Stanford **13 hours** $\rightarrow$ **5 min**
  - bcsstk30 **8.6 hours** $\rightarrow$ **2.4 sec**
  - Skitter **2 hours** $\rightarrow$ **28 sec**, ...
- finds exact MIS, for large networks with known MIS size
- consistently finds larger solutions on social and road networks
- even as we scale to graphs to **10M** to **100M** nodes

# Near-optimal on “difficult networks”




**Problems:**

- Kernelization gives **high-quality** at cost of preprocessing time
- ARW is **fast** but struggles with complex scale-free networks

$\rightarrow$ accelerate local search? parallelization?

# ARW on Complex Networks

## Issues

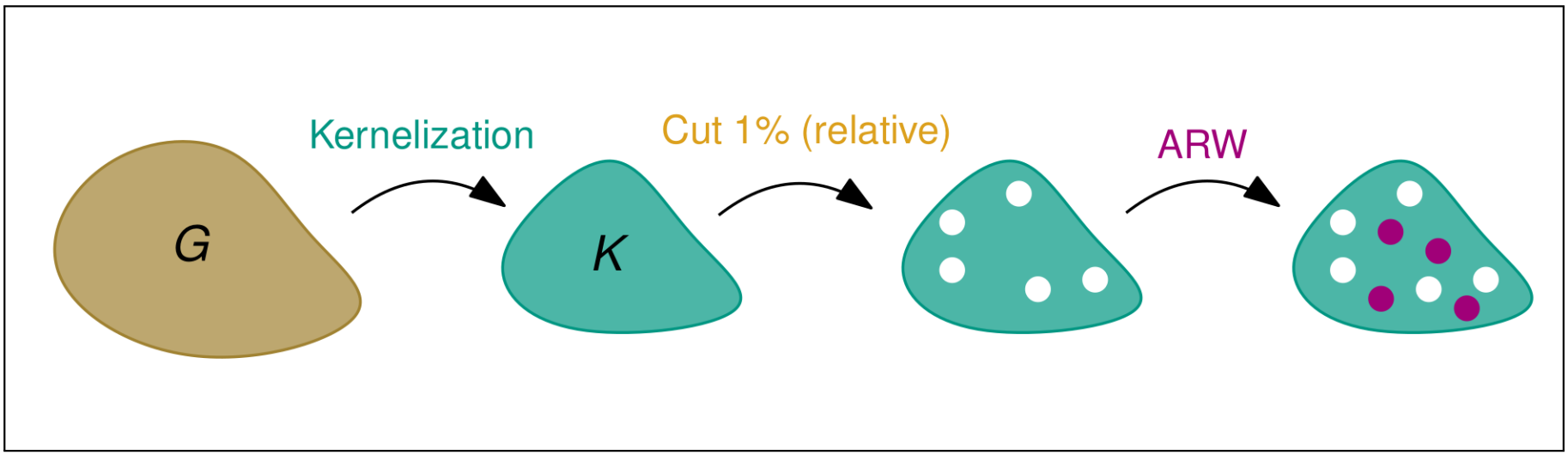
- High degree vertices take long to process ...
  - ... and are unlikely to be in MIS → High-degree cutting
- Many vertices are always in some solution →Kernelization

# Strategy I

## Full Kernelization

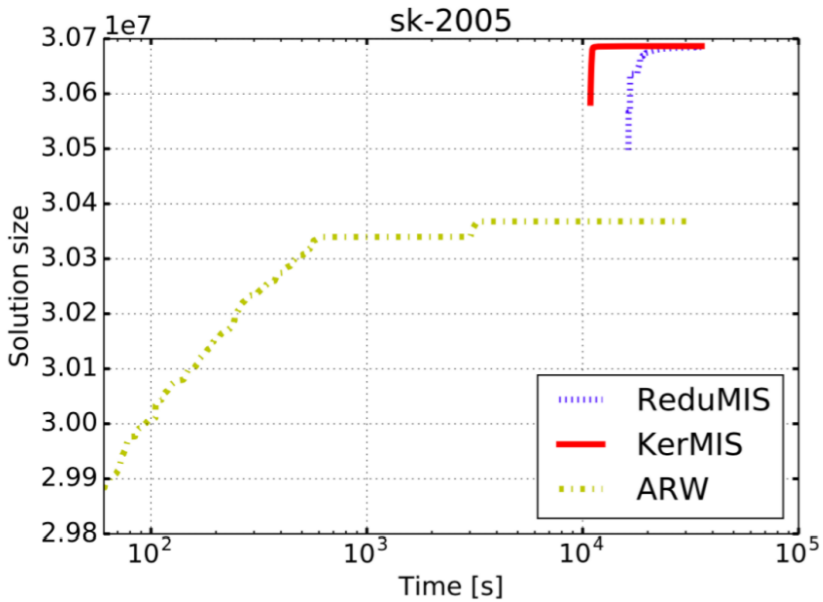
- Repeatedly apply reductions → kernel graph

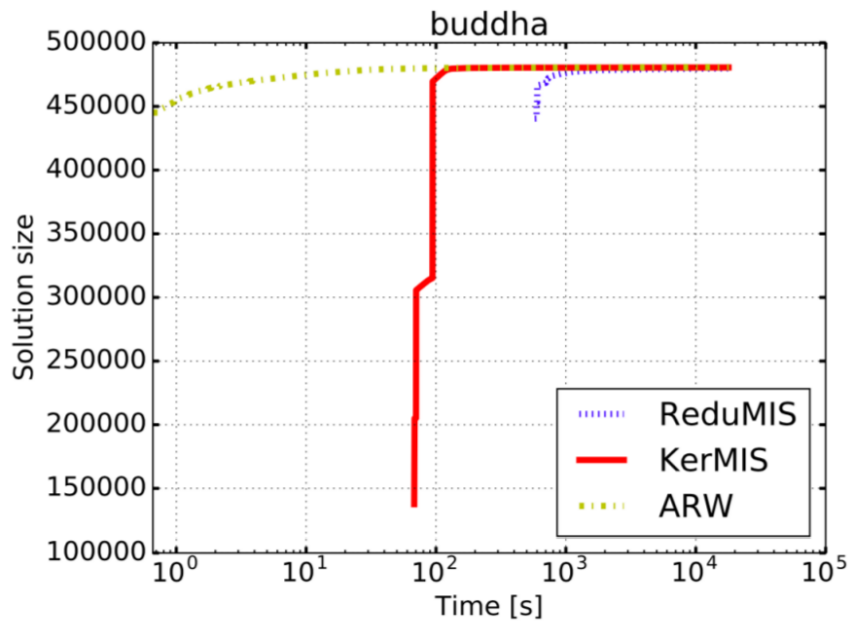

**KerMIS:**



# KerMIS evaluation

- Still long **preprocessing time** to compute kernel
- Inexact reductions **improve performance** of local search
- Quality comparable to **ReduMIS**
  - → Inexact reductions **don't worsen solution quality**

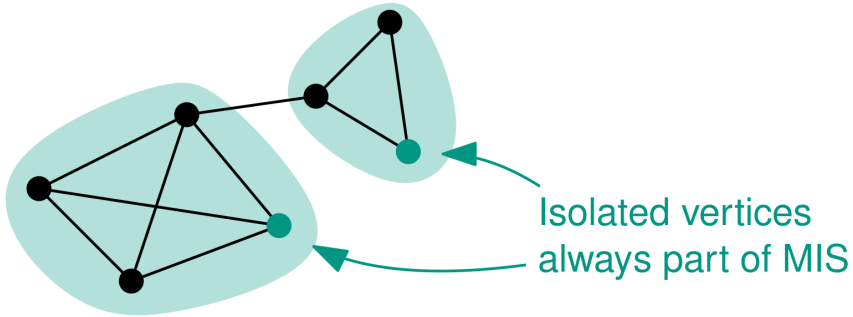


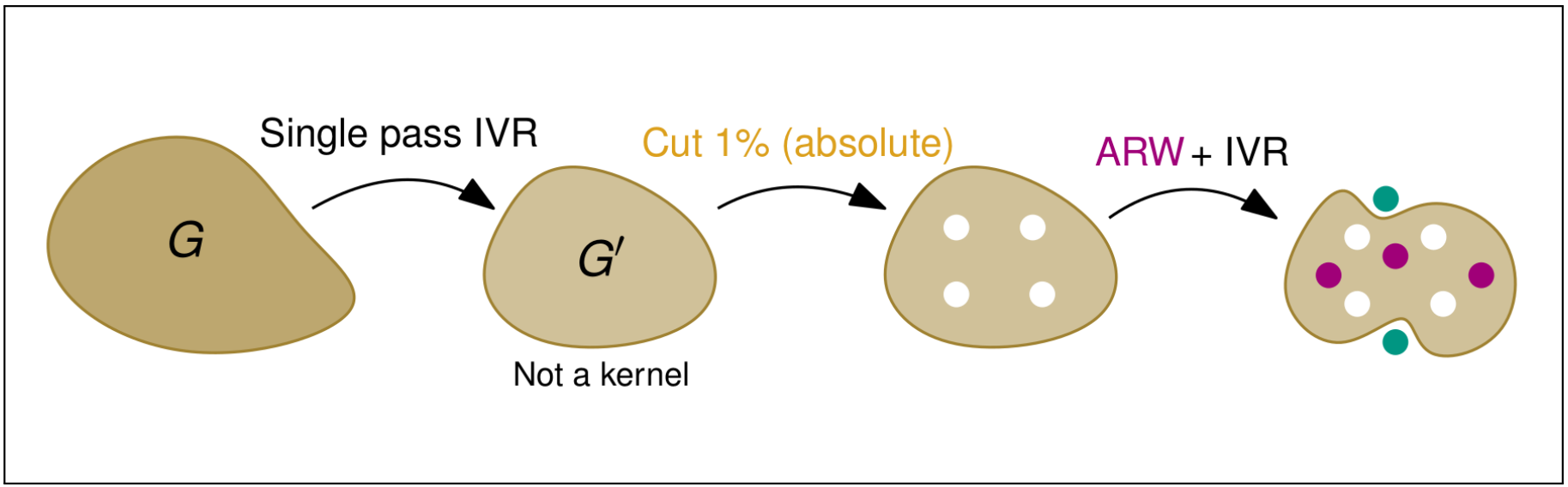
# Strategy II

## Online "Removal"

- Isolated vertex removal (IVR)

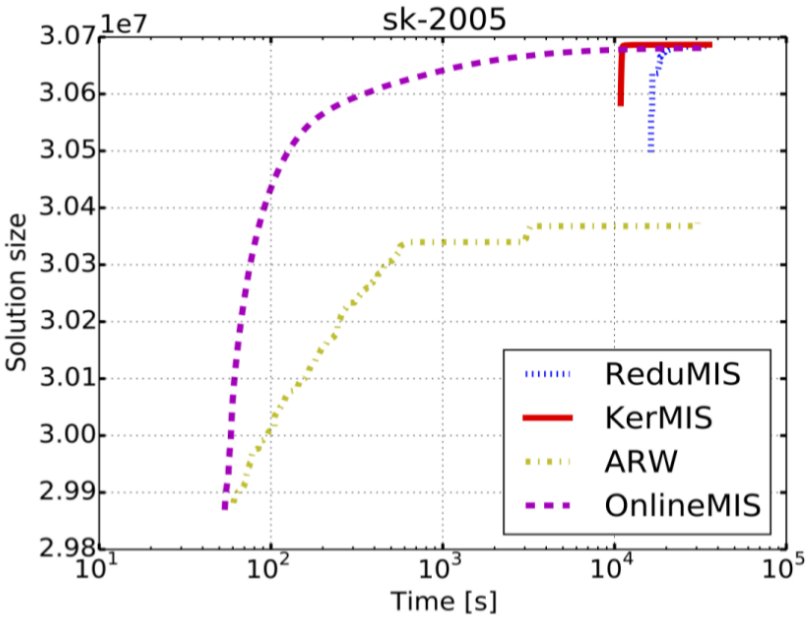


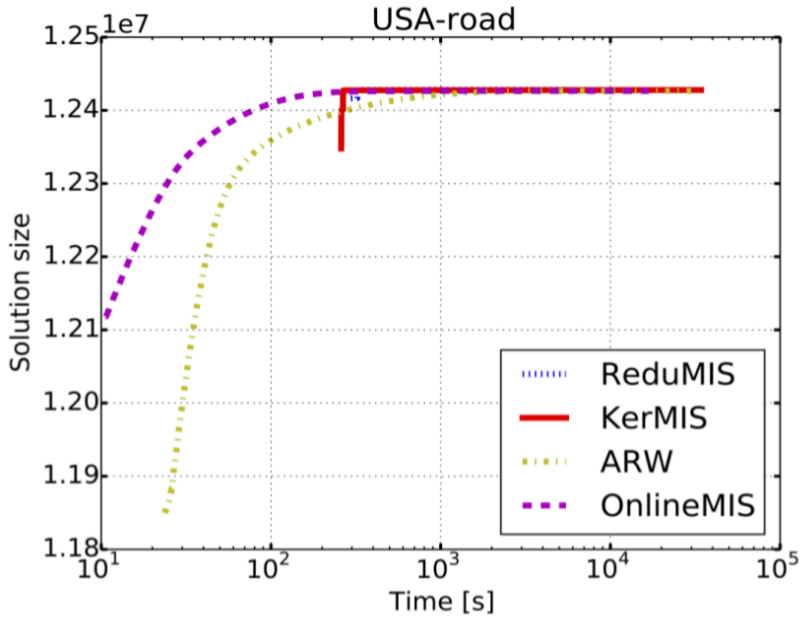

**OnlineMIS:**



# OnlineMIS evaluation

- Significant improvements in **speed and quality**
- **Huge scale-free networks** can be processed efficiently



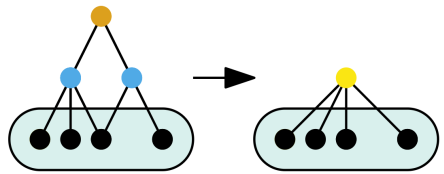




- Maximum speedup $\geq$ 300 over ReduMIS for 14/24 instances

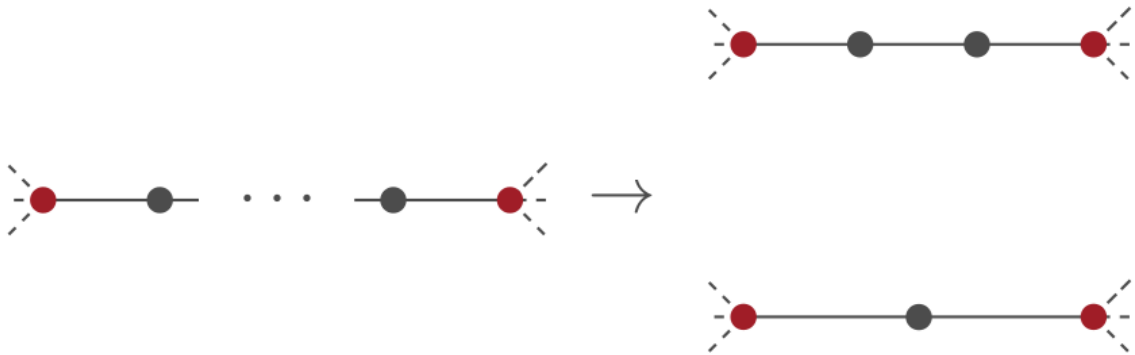
# Linear Time Reductions

[Chang et al.'17]

⇝ vertex folding is slow with high-degree neighbors



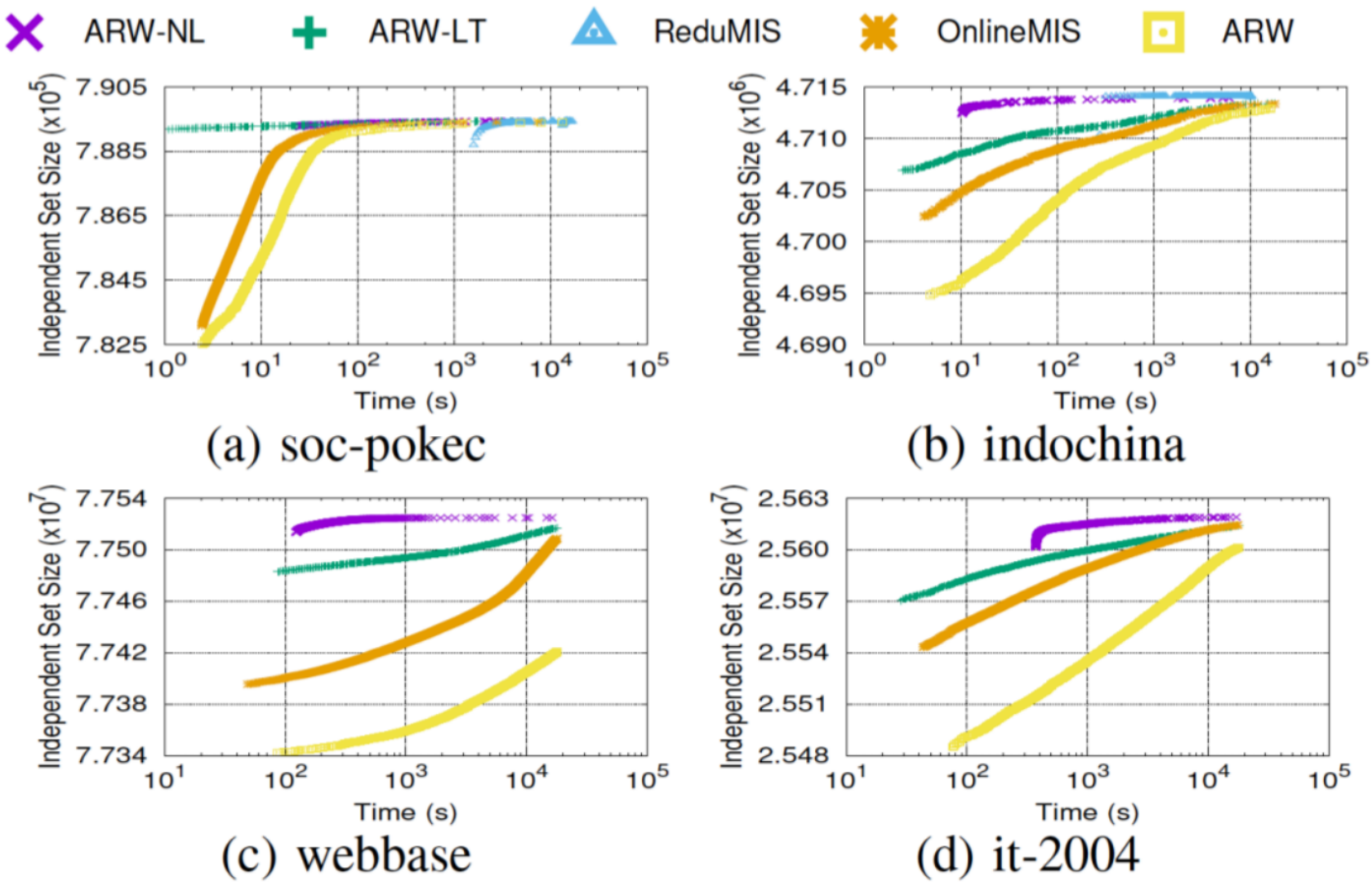
⇝ avoid it



**Repeat:** add small degree vertices to solution + reduce

# Linear Time Reductions

[Chang et al.'17]



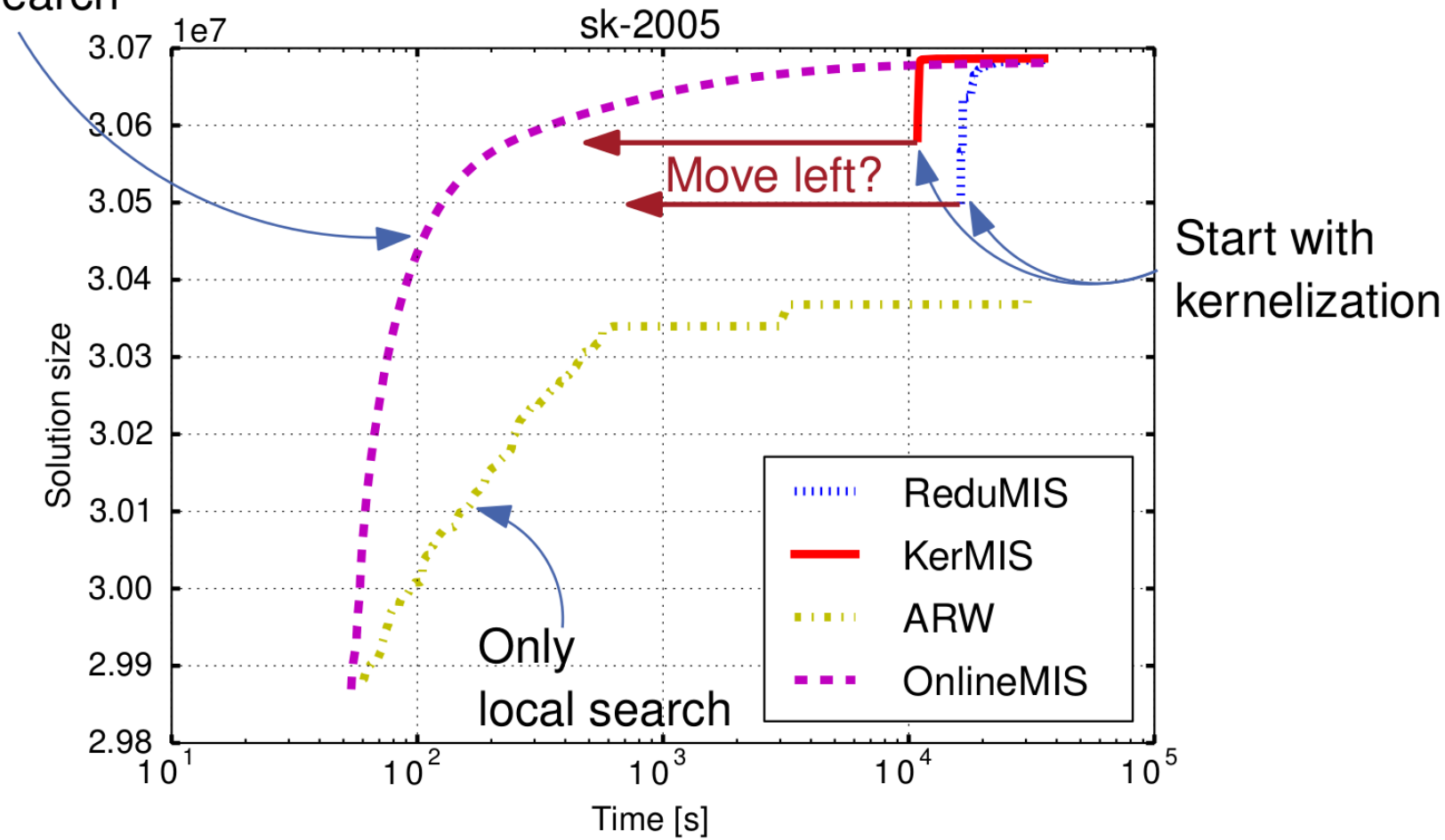
# Scalable Reductions

## Effective Reductions Are Slow!

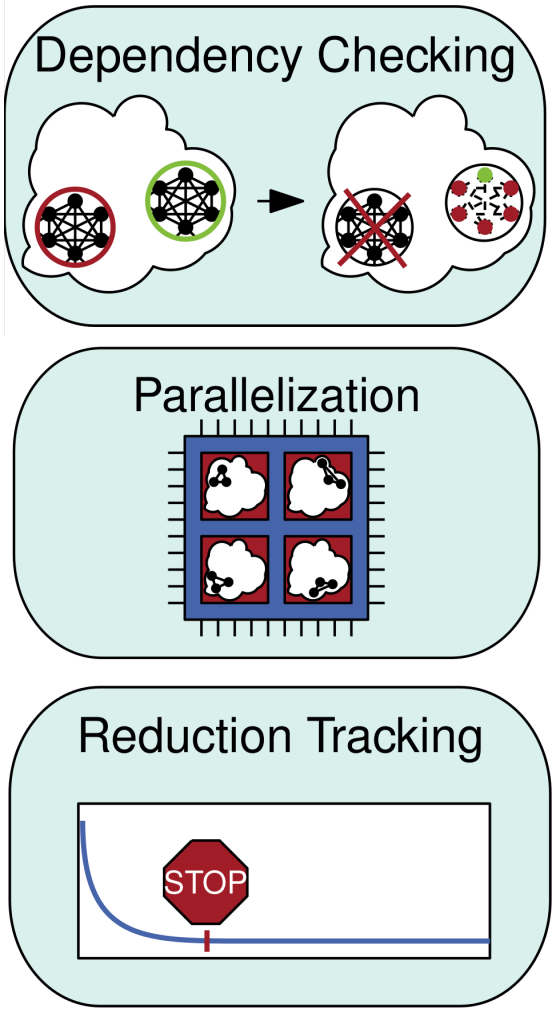
| Graph | | LinearTime | | NearLinear | | VCSolver | |
|---|---|---|---|---|---|---|---|
| name | $n$ | $\|\mathcal{K}\|$ | time | $\|\mathcal{K}\|$ | time | $\|\mathcal{K}\|$ | time |
| uk-2002 | 19M | 11.7M | 1.5 | 4.0M | 28.0 | 0.2M | 336.9 |
| arabic-2005 | 23M | 15.6M | 2.6 | 6.7M | 246.1 | 0.6M | 1 033.2 |
| gsh-2015-tpd | 31M | 2.0M | 11.6 | 1.2M | 97.4 | 0.4M | 372.3 |
| uk-2005 | 39M | 28.2M | 2.5 | 5.9M | 60.5 | 0.8M | 541.4 |
| it-2004 | 41M | 27.1M | 3.3 | 11.3M | 1 544.6 | 1.6M | 6 749.0 |
| sk-2005 | 51M | * | * | * | * | 3.2M | 10 010.5 |
| uk-2007-05 | 106M | * | * | * | * | 3.5M | 18 829.4 |
| webbase-2001 | 118M | 51.7M | 13.0 | 17.3M | 121.1 | 0.7M | 4 207.8 |
| asia.osm | 12M | 626.7K | 0.8 | 594.4K | 1.4 | 15.2K | 204.7 |
| road_usa | 24M | 2.5M | 2.5 | 2.4M | 4.1 | 0.2M | 310.0 |
| europe.osm | 51M | 1 500.0K | 4.1 | 1 329.9K | 6.1 | 8.4K | 302.4 |
| rgg26 | 67M | 67.1M | 1.0 | 51.3M | 172.6 | 49.6M | 9 887.7 |
| rhg | 100M | * | * | * | * | 0 | 124.0 |
| del24 | 17M | 16.8M | 0.2 | 15.6M | 12.7 | 12.4M | 4 789.5 |
| del26 | 67M | 67.1M | 0.7 | 62.5M | 53.3 | 49.9M | 20 728.7 |

# Even Faster??

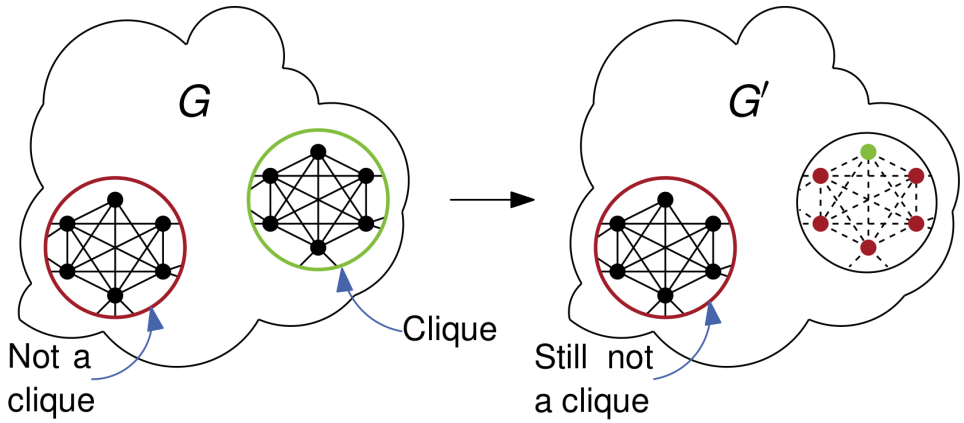
Reductions during local search



# Remaining Ingredients




## Dependency Checking

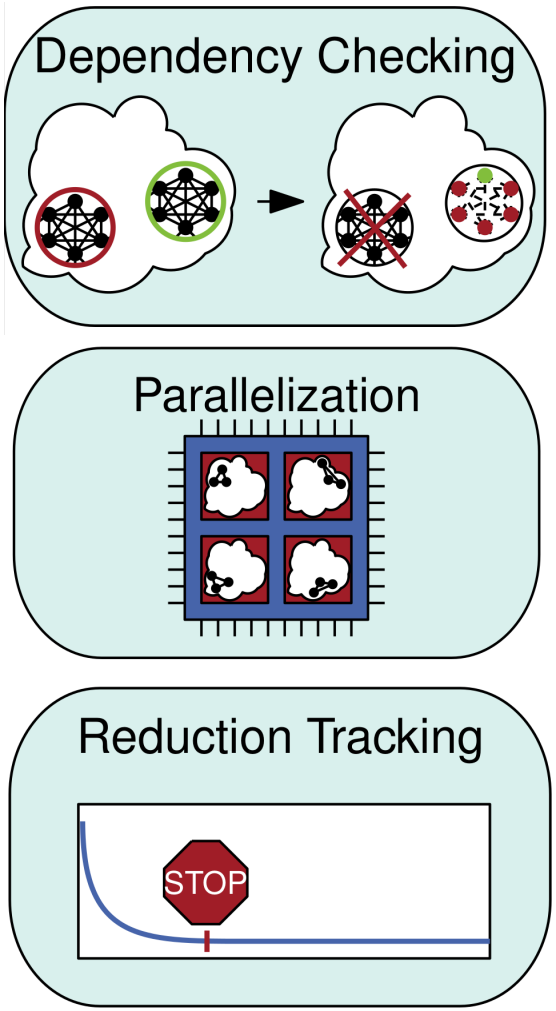



No reduction in $G$ and $N_G(v) = N_{G'}(v) \Rightarrow$ No reduction in $G'$

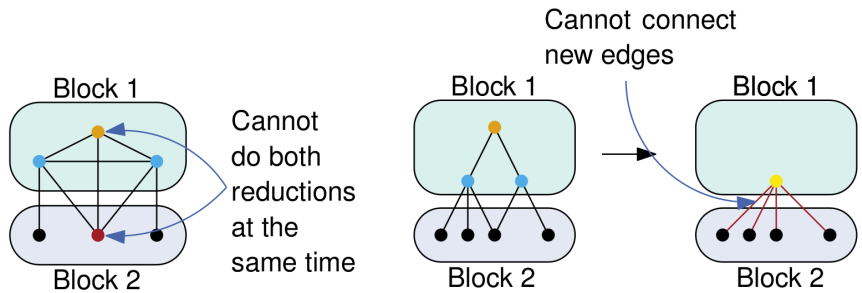
**Used for**

- Isolated Clique Reduction
- Degree 2 Fold
- Twin Reduction

# Remaining Ingredients




**Parallelization**
partition graph in blocks, reduce separately

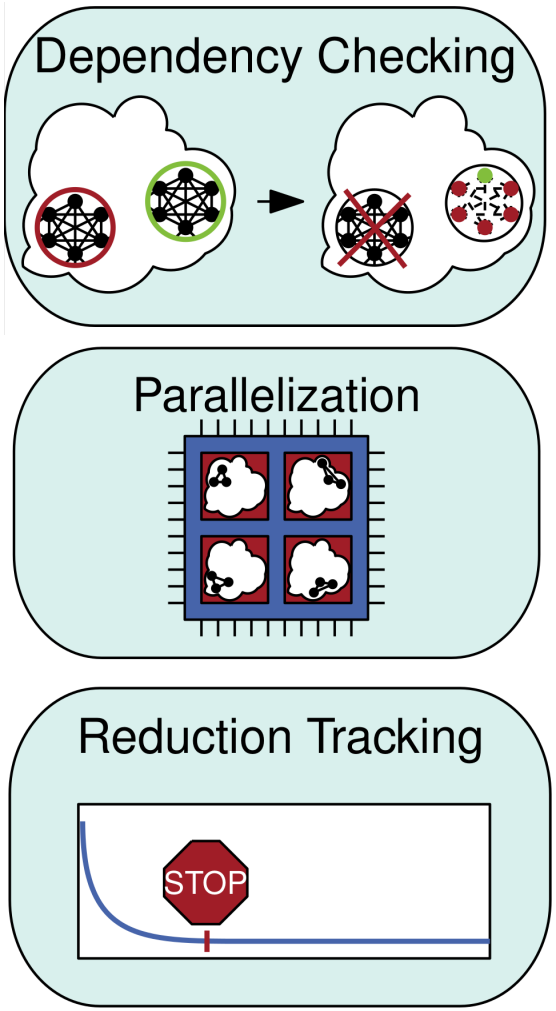



$\rightarrow$ deal with conflicts

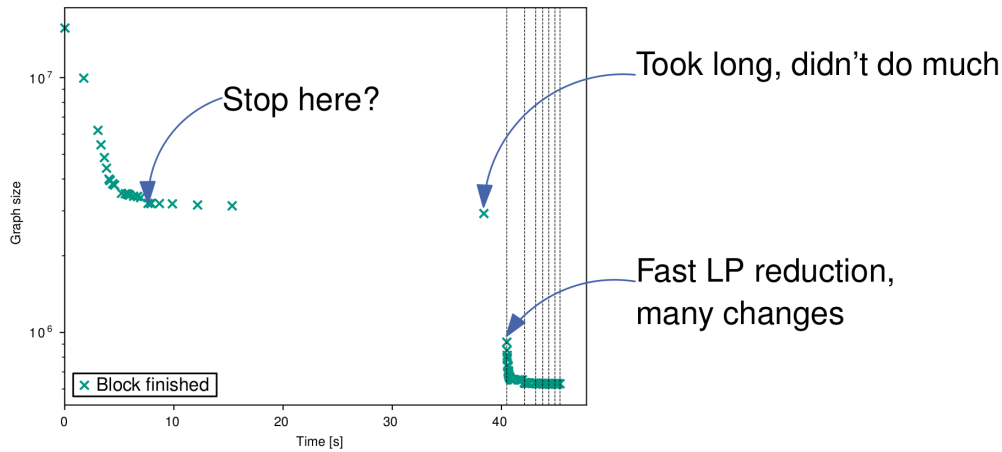
**Ingredients:**

- ParHIP for partitioning
- Parallelize LP reduction with parallel maximum bipartite matching [Azad et al.'17]

# Remaining Ingredients




**Reduction Tracking**
stop kernelization early



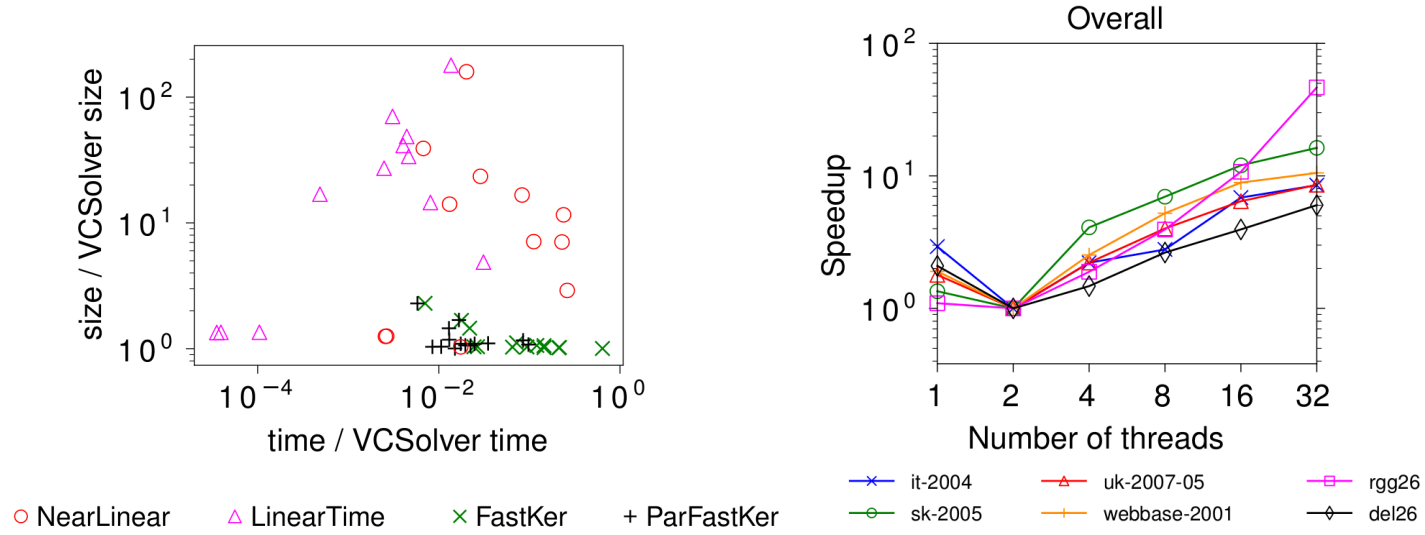

- some blocks take longer than others
- few changes after a while

→ stop "local" reductions early
→ run LP reduction

# Scalable Reductions

| NearLinear | | VCSolver | | ParFastKer | | |
|---|---|---|---|---|---|---|
| $\|\mathcal{K}\|$ | time | $\|\mathcal{K}\|$ | time | $\|\mathcal{K}\|$ | time | su |
| 4.0M | 28.0 | 0.2M | 336.9 | **0.3M** | 11.8 | 28.4 |
| 6.7M | 246.1 | 0.6M | 1 033.2 | **0.6M** | 25.7 | 40.2 |
| 1.2M | 97.4 | 0.4M | 372.3 | **0.5M** | 32.0 | 11.7 |
| 5.9M | 60.5 | 0.8M | 541.4 | **0.9M** | 53.3 | 10.1 |
| 11.3M | 1 544.6 | 1.6M | 6 749.0 | **1.7M** | 151.8 | 44.4 |
| * | * | 3.2M | 10 010.5 | 3.5M | 178.3 | 56.1 |
| * | * | 3.5M | 18 829.4 | **3.7M** | 372.4 | 50.6 |
| 17.3M | 121.1 | 0.7M | 4 207.8 | **0.9M** | 54.9 | 76.6 |
| 594.4K | 1.4 | 15.2K | 204.7 | **34.9K** | 1.2 | 169.8 |
| 2.4M | 4.1 | 0.2M | 310.0 | **0.2M** | 4.1 | 76.0 |
| 1 329.9K | 6.1 | 8.4K | 302.4 | **14.2K** | 4.9 | 61.3 |
| 51.3M | 172.6 | 49.6M | 9 887.7 | **9.8M** | 150.3 | 65.8 |
| * | * | 0 | 124.0 | **16** | 64.6 | 1.9 |
| 15.6M | 12.7 | 12.4M | 4 789.5 | 12.9M | 51.5 | 93.1 |
| 62.5M | 53.3 | 49.9M | 20 728.7 | 51.7M | 179.0 | 115.8 |

# Experimental Results




- VCSolver [Akiba and Iwata'16]: slow but small kernels
- LinearTime and NearLinear [Chang et al.'17]: fast but large kernels
  → integrate as preprocessing

**Summary:**

- Orders of magnitude smaller than fast methods
- Orders of magnitude faster than algorithm with similar-sized kernels
- Integration into local search (small kernels matter):
  - larger independent sets faster

# PACE Challenge 2019

## Results

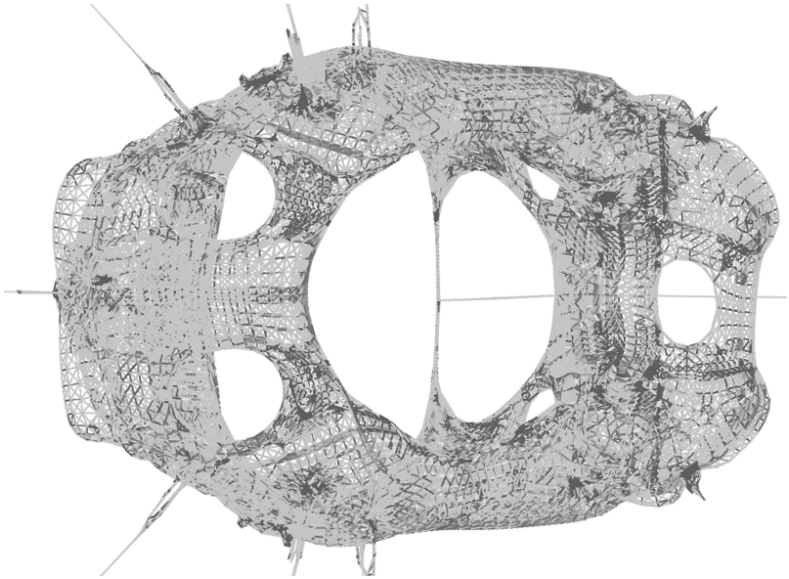
- Exact vertex cover solver
  $\rightarrow$based on reductions + local search + portfolio of exact solvers
- **Exact Solvers:**
  - Branch-and-Reduce for vertex cover
  - Clique Solver by Li et al.
  - Get the easy instances fast, then get the hard instances.
  - 24 competing algorithms
  - **Results:** FIRST place

# Problem

$$G = (V, E, d)$$
$$d : E \to \mathbf{R}$$

$\Rightarrow$

$$x : V \to \mathbf{R}^2$$

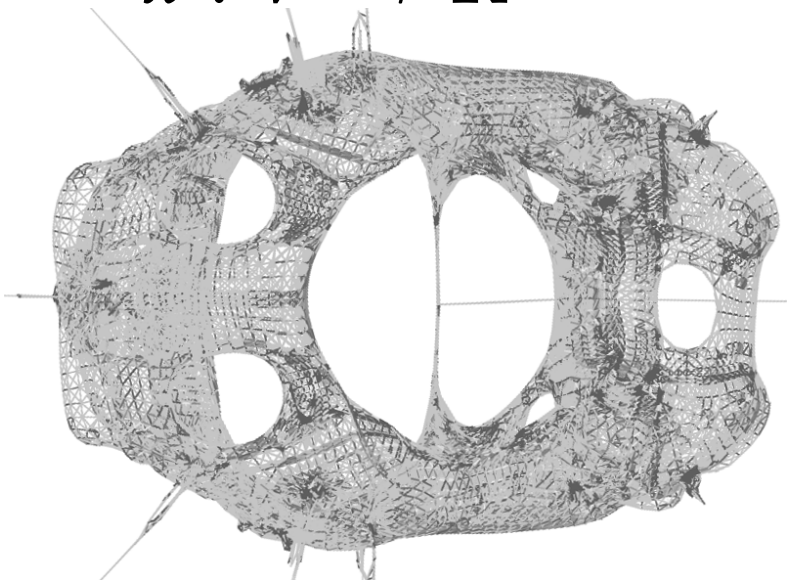


# Problem

$G = (V, E, d)$
$d : E \to \mathbf{R}$
$d \equiv 1$

$\Rightarrow$

$x : V \to \mathbf{R}^2$



# Maximal Entropy Stress Model

[Gansner et al.'13]

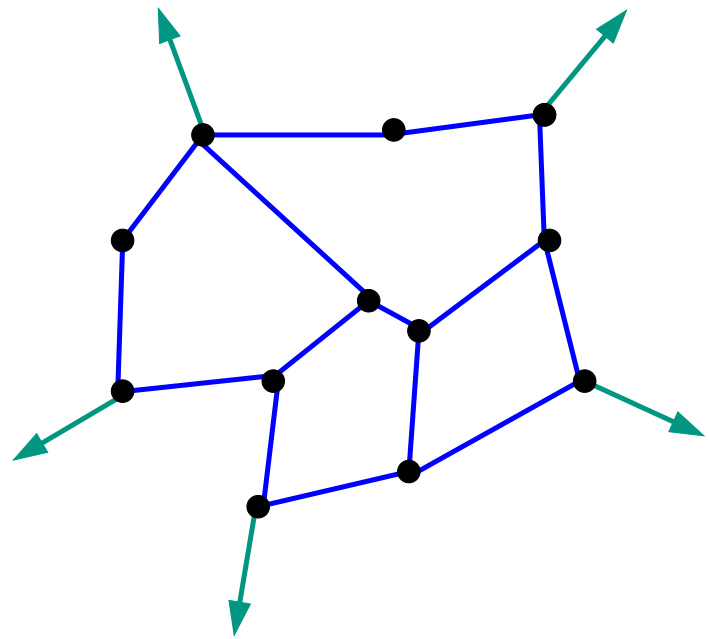
Entropy $H(x)$:

- physics: nodes evenly dispersed
  $\rightarrow$ nodes as far away as possible
- some nodes have predefined distance!

Maximal Entropy Stress Model:

$$\max H(x) := \sum_{\{u,v\} \notin E} \ln ||x_u - x_v||$$

$$\text{subject to } ||x_u - x_v|| = d_{uv}, \{u,v\} \in E$$



# Maximal Entropy Stress Model

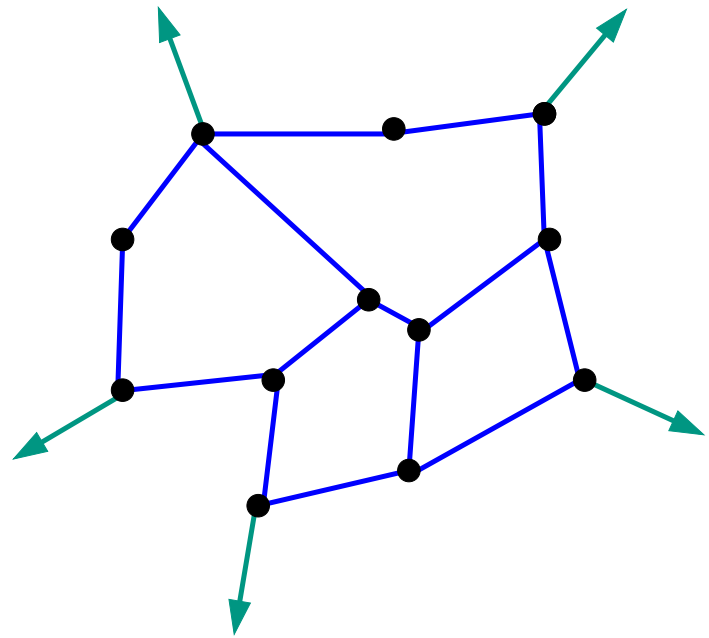
[Gansner et al.'13]

Entropy $H(x)$:

- physics: nodes evenly dispersed
  $\rightarrow$ nodes as far away as possible
- some nodes have predefined distance!

Maximal Entropy Stress Model:

$$\max H(x) := \sum_{\{u,v\} \notin E} \ln ||x_u - x_v||$$
$$\text{subject to } ||x_u - x_v|| = d_{uv}, \{u,v\} \in E$$



- not possible to satisfy all constraints!
  $\rightarrow$ model may be infeasible

# Maximal Entropy Stress Model

[Gansner et al.'13]

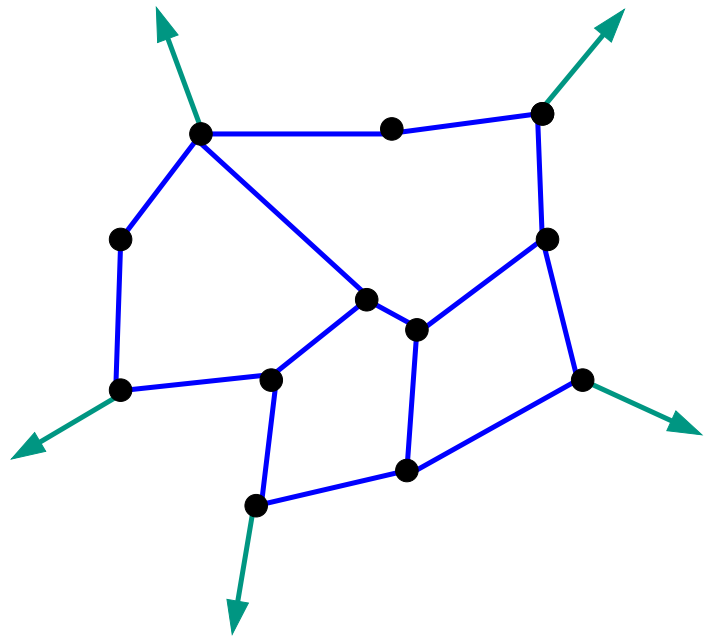
Compromise: min error, max entropy

$$\min \sum_{u,v \in E} w_{uv}(||x_u - x_v|| - d_{uv})^2 - \alpha H(x)$$

$\alpha$ trade-off parameter

Solve optimization problem by

- repeatedly solving Laplacian systems
- or iterative scheme ...



# Maximal Entropy Stress Model

[Gansner et al.'13]

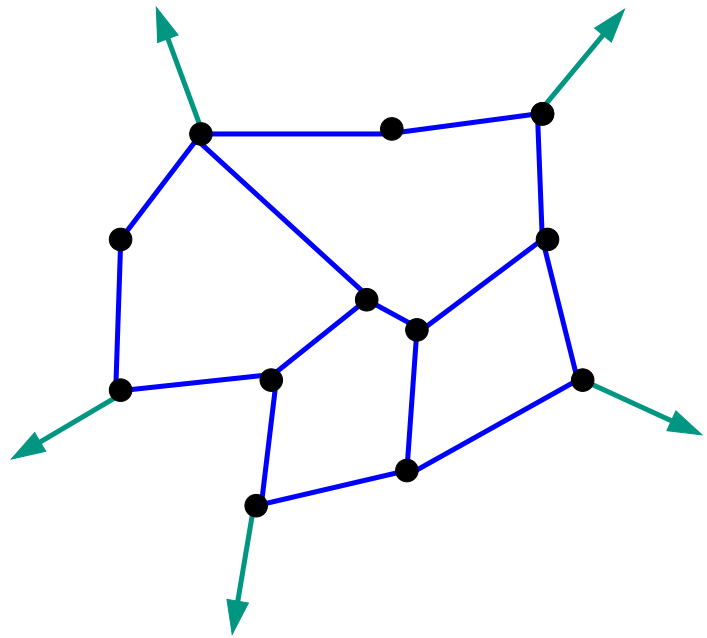
Compromise: min error, max entropy

$$\min \sum_{u,v \in E} w_{uv}(||x_u - x_v|| - d_{uv})^2 - \alpha H(x)$$

$\alpha$ trade-off parameter

Solve optimization problem by

- repeatedly solving Laplacian systems
- or iterative scheme ...



# Maximal Entropy Stress Model

[Gansner et al.'13]

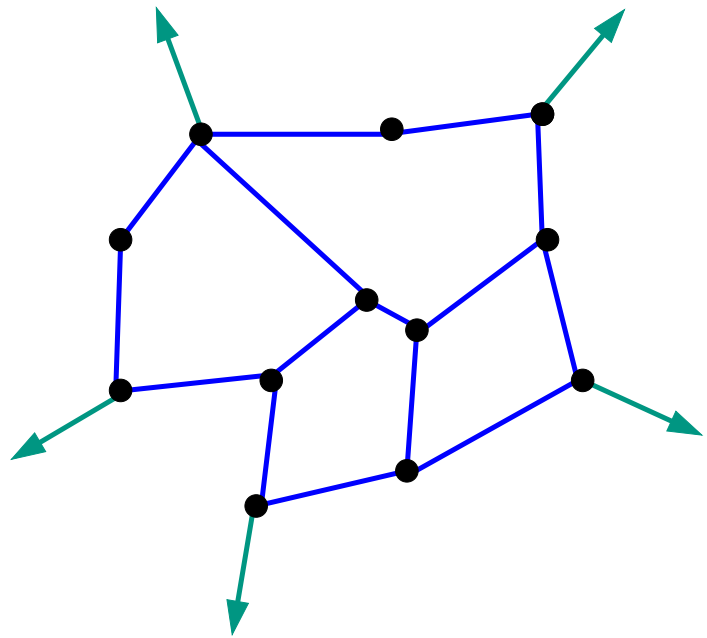
... or iterative scheme:

$$x_u \leftarrow \frac{1}{\rho_u} \sum_{\{u,v\} \in E} w_{uv} \left( x_v + d_{uv} \frac{x_u - x_v}{\|x_u - x_v\|} \right) + \frac{\alpha}{\rho_u} \sum_{\{u,v\} \notin E} \frac{x_u - x_v}{\|x_u - x_v\|^2}$$

$\rightarrow$ overall update costs $O(n \log n)$ per iteration

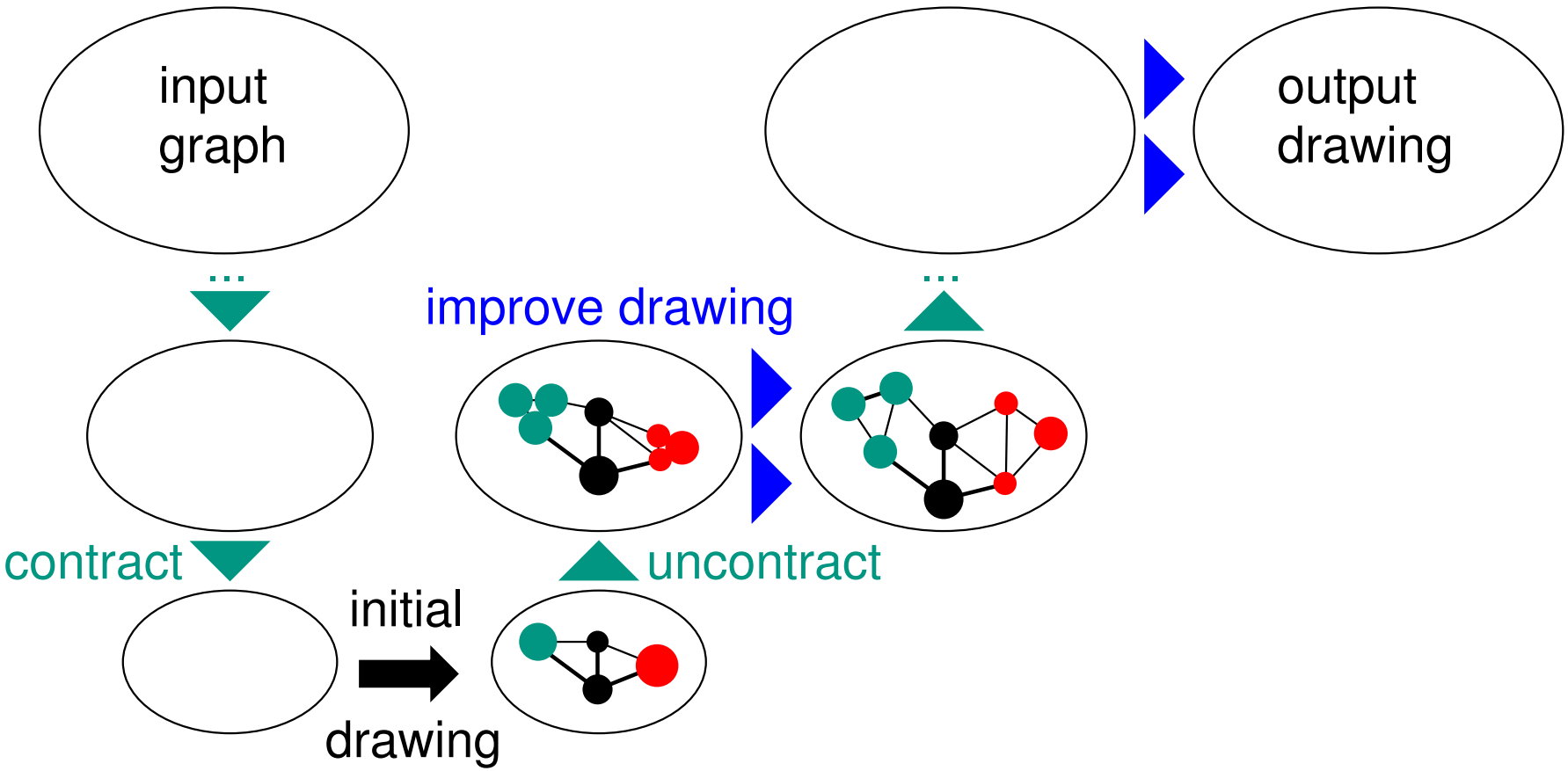
Our contributions:

- make this usable and fast in practice
- multilevel integration
- approximate long-range forces
- employ parallelism



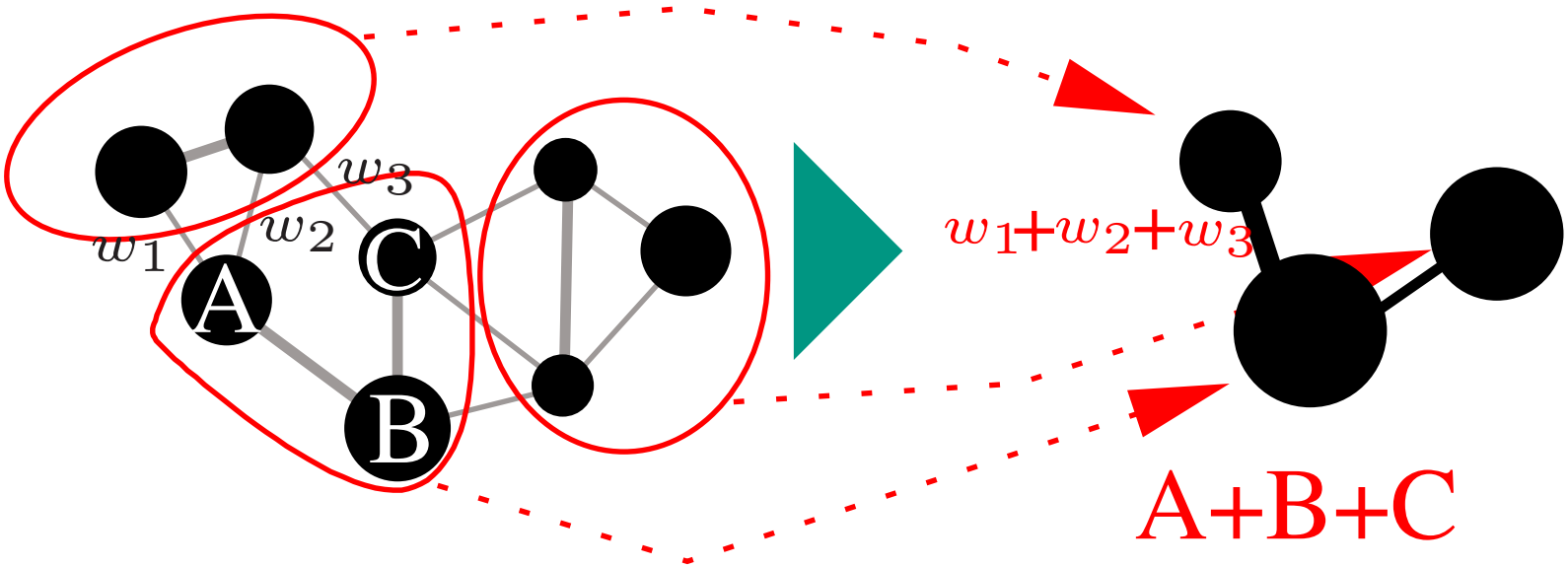
# Multilevel Graph Drawing

[Hadany, Harel'99]



# Basic Idea

## Contraction of Clusterings

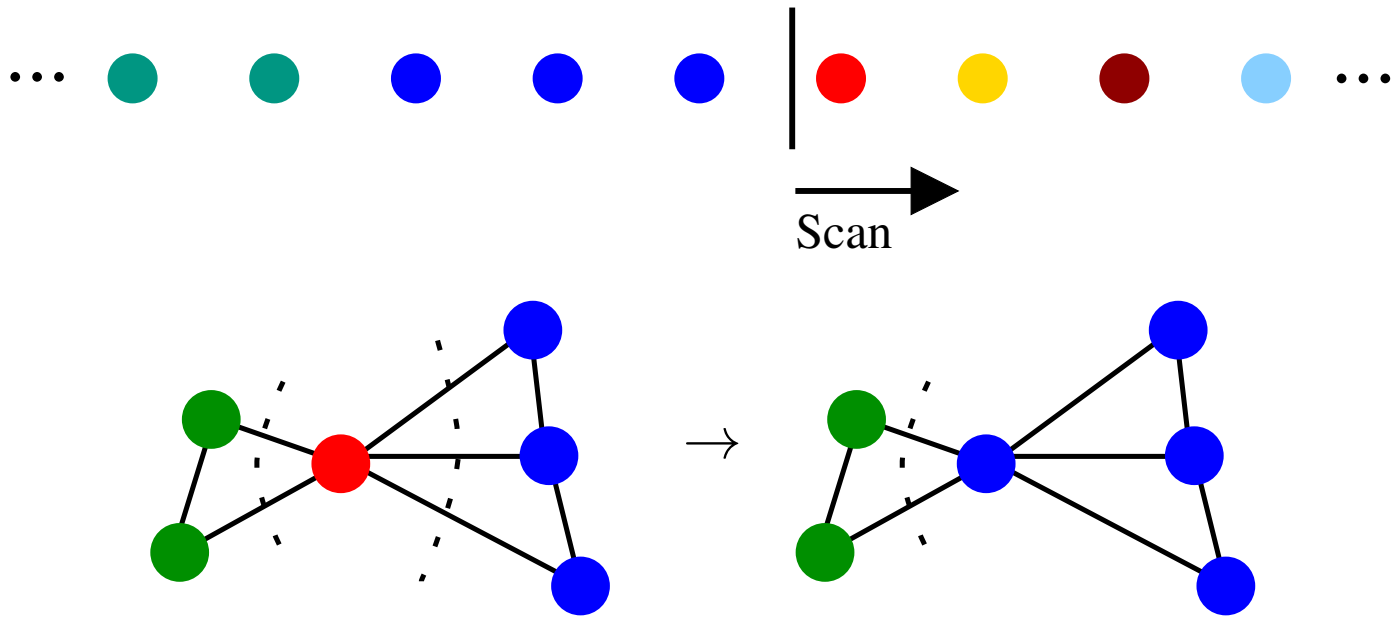



- control block size: size-constraint $U$
- recurse until graph is small
- $c : V_i \rightarrow \mathbf{R}$ number of nodes on finest level

# Label Propagation

**Cut-based, Linear Time Clustering Algorithm [Raghavan et. al]**

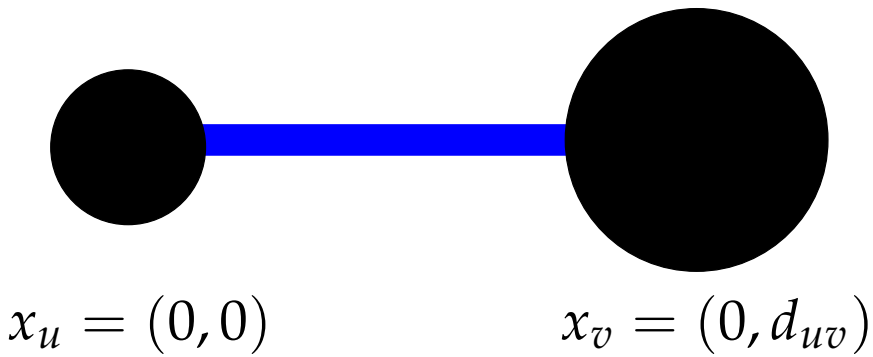
- cut-based clustering using size-constraint label propagation
  - start with singletons
  - traverse nodes in random order or smallest degree first
  - move node to cluster having strongest *eligible* connection
  - modification *eligible*: w.r.t size constraint U



# Multilevel Graph Drawing

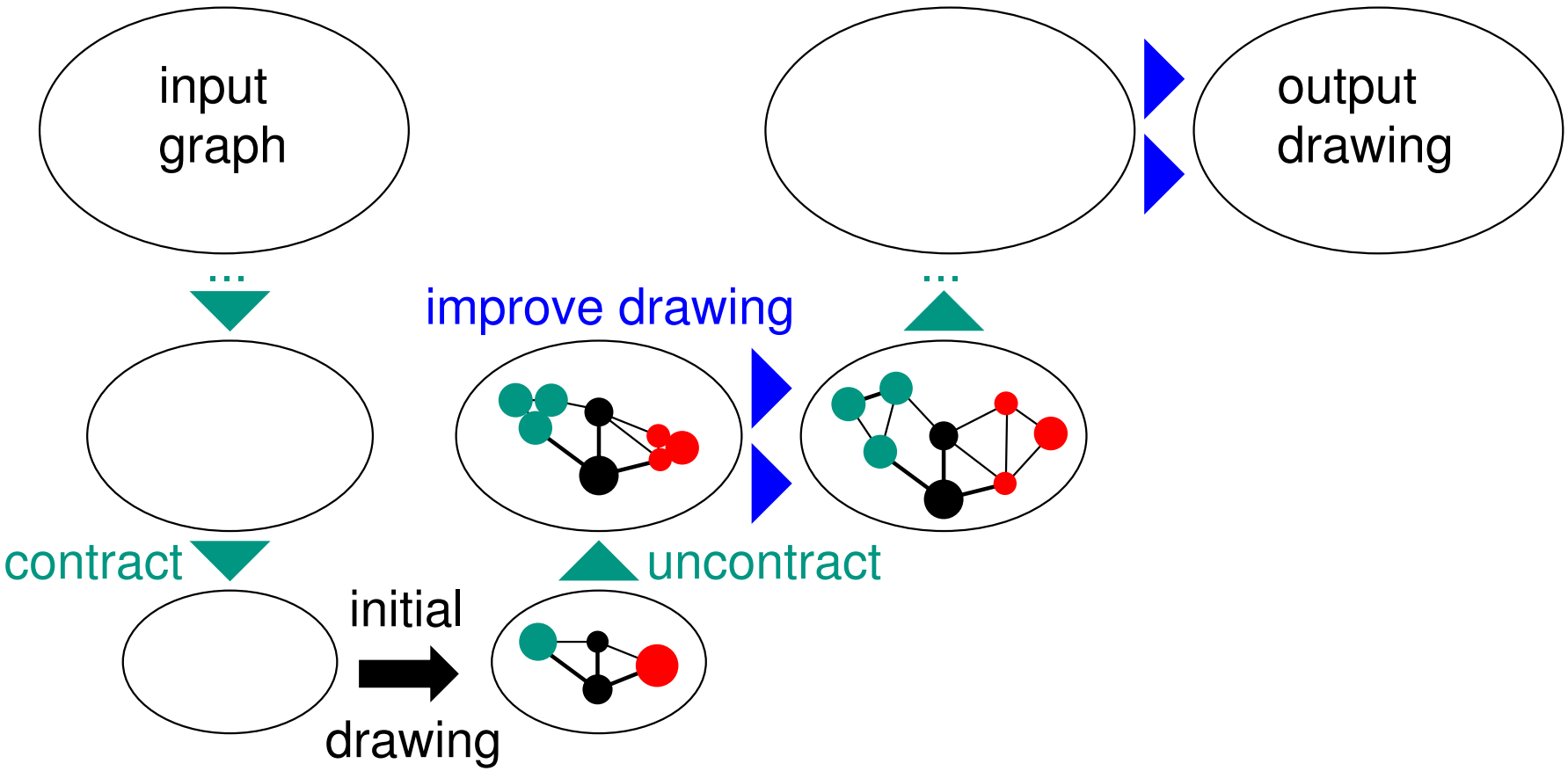
## Initial Drawing

- coarsen until only two nodes left
- place them at optimal distance
- define distances on coarse graphs, stay tuned



# Multilevel Graph Drawing

[Hadany, Harel'99]



# Uncoarsening

**Uncontraction/Transfer**

- $v$ fine vertex of coarse representative $v'$ at $P = (x, y)$
- put $v$ at random position in disk around $P$ with radius $r = \sqrt{c(v')}$

**Local Improvement**

- minimize maxent-stress on each level of hierarchy
- assume disk with radius $\sqrt{c(u)}$ to draw $c(u)$ vertices

  $\rightarrow$ define distance $d_{uv} := \frac{\sqrt{c(u)}+\sqrt{c(v)}}{2}$ on current level

**Iterative scheme**

$$x_u \leftarrow \frac{1}{\rho_u} \sum_{\{u,v\} \in E} w_{uv} \left( x_v + d_{uv} \frac{x_u - x_v}{\|x_u - x_v\|} \right) + \frac{\alpha}{\rho_u} \sum_{\{u,v\} \notin E} \frac{x_u - x_v}{\|x_u - x_v\|^2}$$

# Uncoarsening

**Uncontraction/Transfer**

- $v$ fine vertex of coarse representative $v'$ at $P = (x, y)$
- put $v$ at random position in disk around $P$ with radius $r = \sqrt{c(v')}$

**Local Improvement**

- minimize maxent-stress on each level of hierarchy
- assume disk with radius $\sqrt{c(u)}$ to draw $c(u)$ vertices

  $\rightarrow$ define distance $d_{uv} := \frac{\sqrt{c(u)}+\sqrt{c(v)}}{2}$ on current level

**Iterative scheme**

$$x_u \leftarrow \frac{1}{\rho_u} \sum_{\{u,v\} \in E} w_{uv} \left( x_v + d_{uv} \frac{x_u - x_v}{\|x_u - x_v\|} \right) + \frac{\alpha}{\rho_u} \sum_{\{u,v\} \notin E} \frac{x_u - x_v}{\|x_u - x_v\|^2}$$

# Local Improvement

## Iterative scheme

$$x_u \leftarrow \dots \sum_{\{u,v\} \notin E} \underbrace{\frac{x_u - x_v}{\|x_u - x_v\|^2}}_{=:r(u,v)}$$

## Approximation

$$x_u \leftarrow \dots \sum_{\substack{u \neq v \\ M(u)=M(v)}} r(u,v) + \sum_{\substack{v' \in V' \\ v' \neq M(u)}} \nu(v') \frac{x_u - x'_{v'}}{\|x_u - x'_{v'}\|^2} - \sum_{\{u,v\} \in E} r(u,v)$$

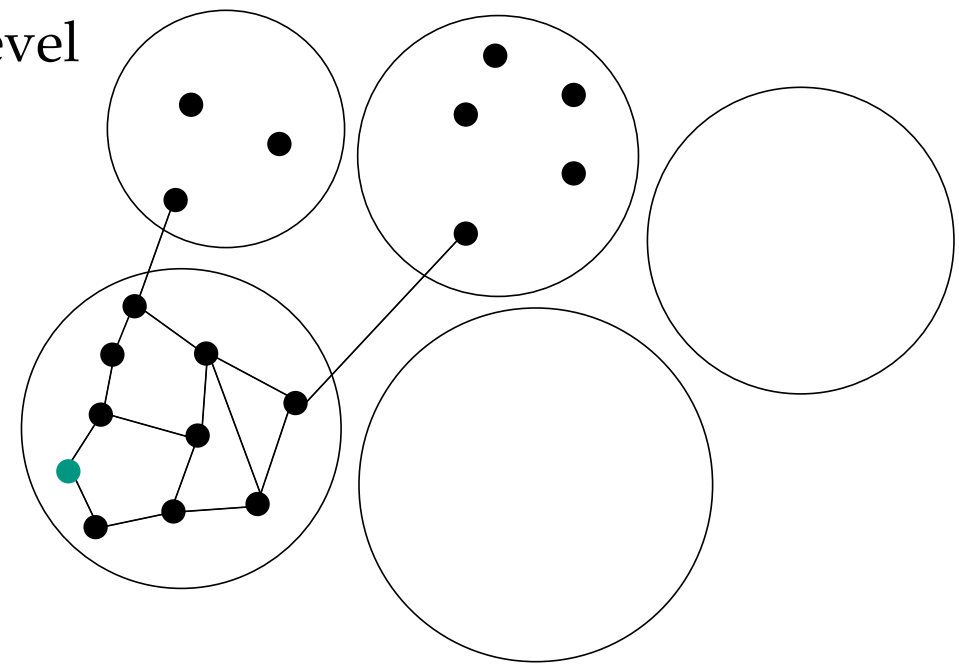
- $M(u)$ cluster of $u$
- $\nu(v')$ number of finer vertices of $v'$ on current level
- $V'$ vertex set of next coarser level
- $x'_{v'}$ coordinate of $v'$

# Local Improvement

## Approximation

$$x_u \leftarrow \cdots \sum_{\substack{u \neq v \\ M(u)=M(v)}} r(u,v) + \sum_{\substack{v' \in V' \\ v' \neq M(u)}} \nu(v') \frac{x_u - x'_{v'}}{\|x_u - x'_{v'}\|^2} - \sum_{\{u,v\} \in E} r(u,v)$$

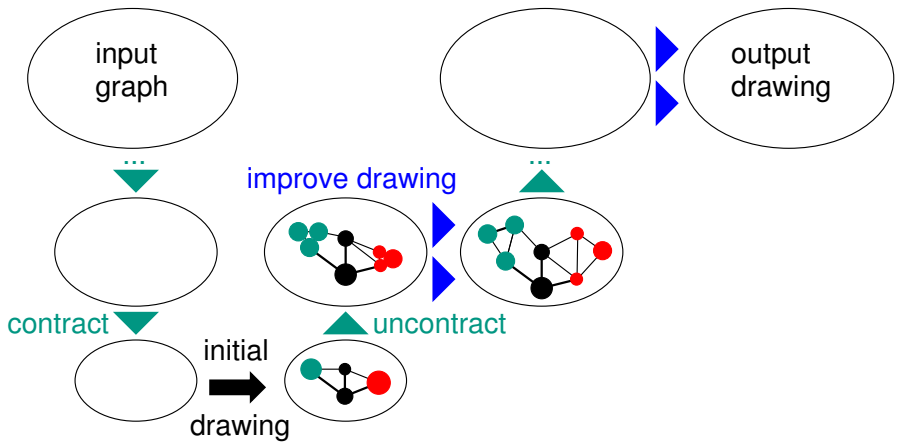
- $M(u)$ cluster of $u$
- $\nu(v')$ number of finer vertices of $v'$ on current level
- $V'$ vertex set of next coarser level
- $x'_{v'}$ coordinate of $v'$



# Local Improvement

## Additional Enhancements

- after each iteration $\rightarrow$ update barycenter of coarse nodes
- vertex computations independent $\rightarrow$ add parallelism
- use approximation multiple – $h$ – levels beneath in hierarchy

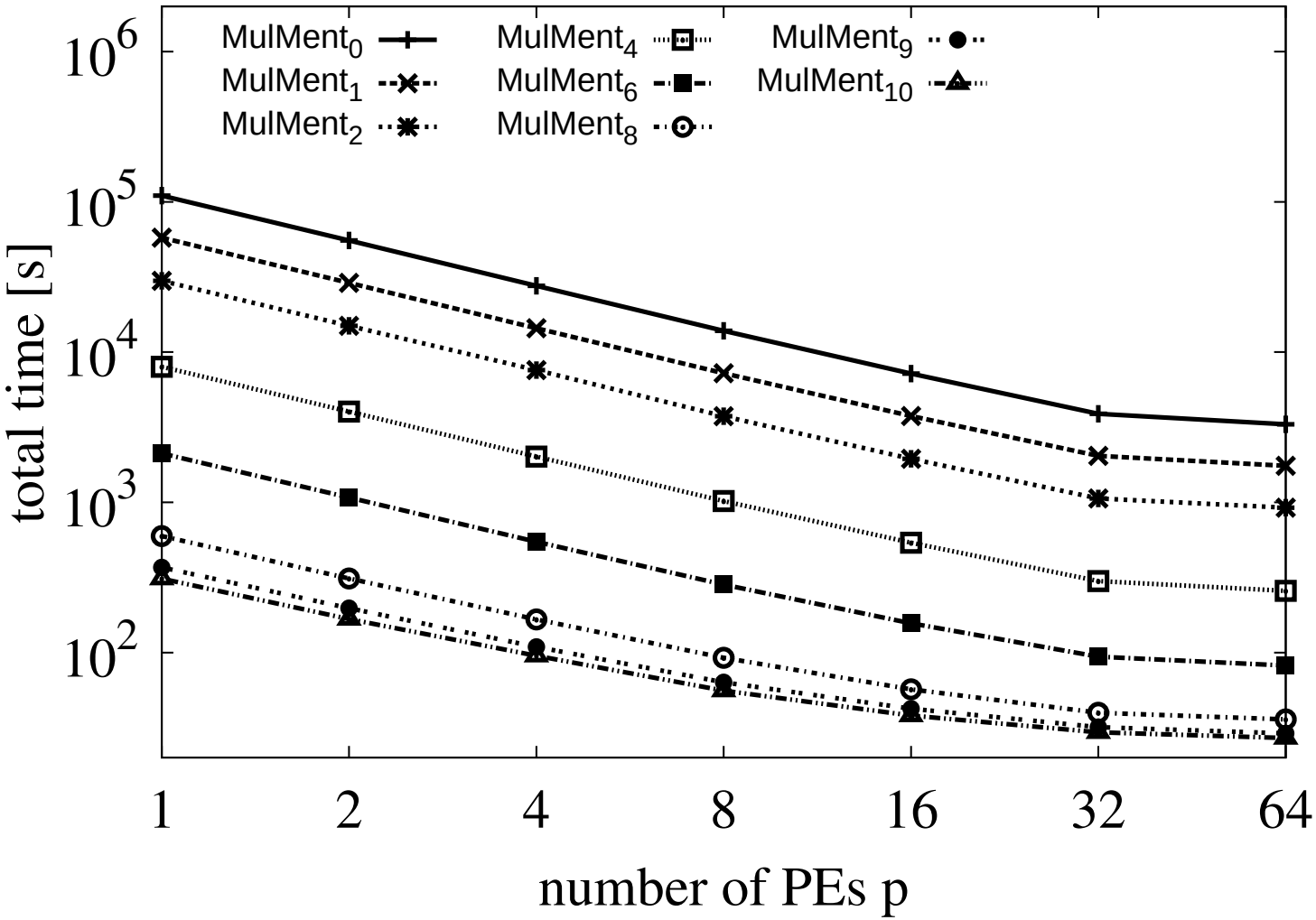



**Proposition:** Assume equal cluster sizes. The running time of one iteration of MulMent$_h$, $h \geq 0$, is $\mathcal{O}(m + n^{\frac{h+2}{h+1}})$

# Experimental Results

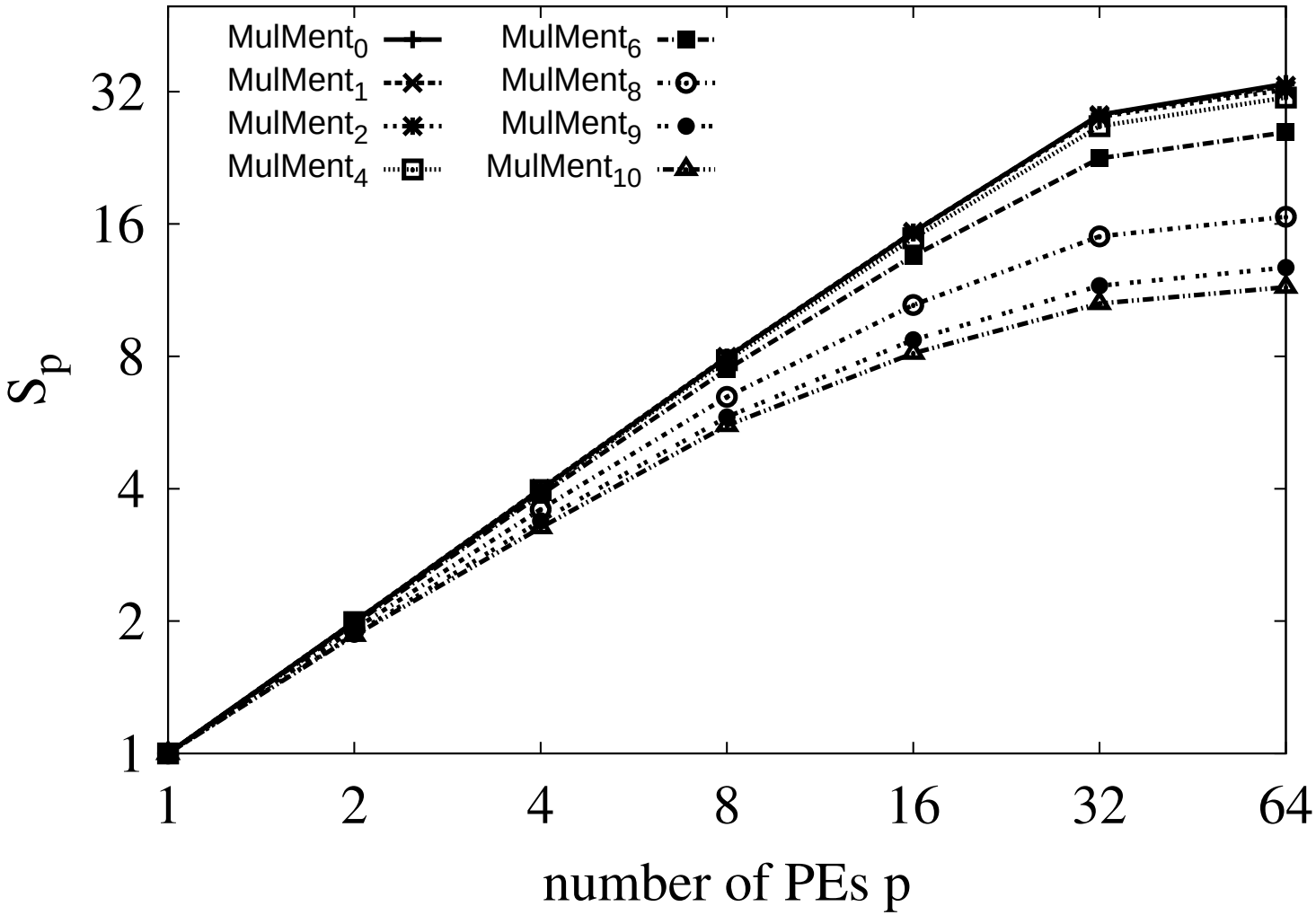
# Scalability

## Running Time [Delaunay $n = 2^{20}$]



# Scalability

## Speedup [Delaunay $n = 2^{20}$]



# Running Times

| graph | PMDS | MaxEnt | $\text{MulMent}_0$ | $\text{MulMent}_1$ | $\text{MulMent}_{10}$ |
|---|---|---|---|---|---|
| btree | 0.02 | 1.14 | 0.10 | 0.17 | 0.11 |
| 1138_bus | 0.04 | 1.41 | 0.15 | 0.13 | 0.11 |
| USpowerG | 0.14 | 3.82 | 0.29 | 0.23 | 0.18 |
| 3elt | 0.16 | 3.45 | 0.21 | 0.19 | 0.17 |
| commanche | 0.24 | 5.42 | 0.32 | 0.27 | 0.18 |
| bcsstk31 | 3.44 | 48.48 | 5.63 | 3.97 | 1.82 |
| fe_pwt | 1.49 | 31.60 | 4.46 | 2.67 | 0.66 |
| del16 | 2.86 | 61.42 | 13.63 | 7.75 | 1.10 |
| luxembourg | 3.10 | 96.10 | 40.94 | 22.87 | 1.39 |
| nyc | 9.03 | 233.94 | 216.27 | 119.33 | 3.70 |
| auto | 41.80 | 665.67 | 613.51 | 329.08 | 20.70 |
| del20 | 53.80 | 1125.03 | 3303.82 | 1749.77 | 27.01 |

Table: Running times in seconds per graph. Smaller is better. PivotMDS and MaxEnt use one thread (sequential codes), the $\text{MulMent}_*$ algorithms use 32 cores (64 threads). Running times of MaxEnt are without the time of PMDS (which yields input coordinates to MaxEnt)

# Experimental Results

## Summary

**Influence of $h$**

- increasing $h$ not a large impact on solution quality
- maxent-stress remains comparable
- slight increase in FSM on 2 instances

**Comparison**

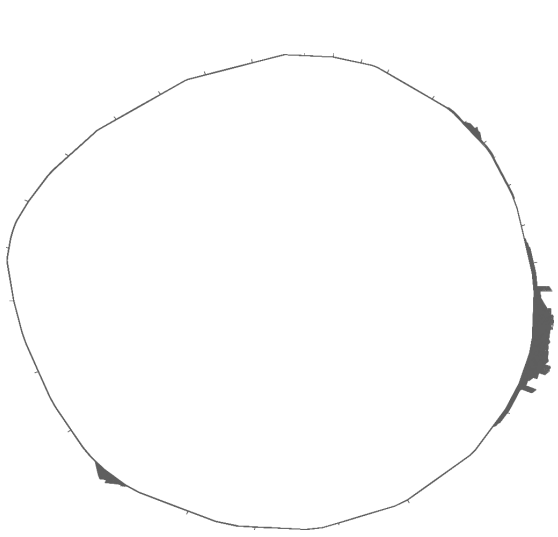
- maxent-stress of MaxEnt and MulMent more or less similar
- four out of nine full stress compareable to MaxEnt
- largest three instances worse full stress (not astonishing!)

**Dynamic Networks**

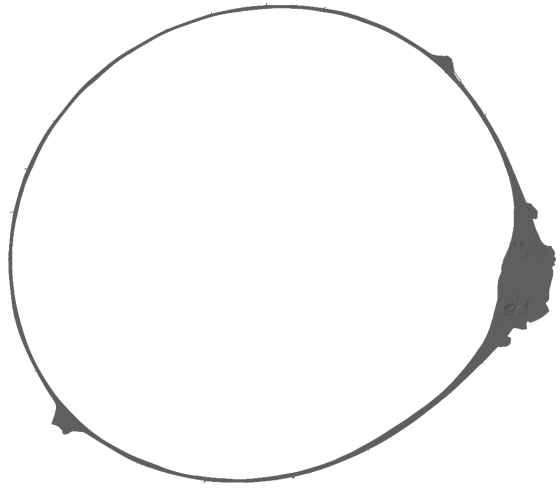
- $\approx$model: remove $x$% random edges, insert $x$% edges (distance $\leq \mathcal{D}$)
- $4\times$ faster ($h = 0$), save 50% time ($h = 7$)
- 9% worse full-stress, 1% better maxent-stress

# Example Drawings

**fe_pwt**



**PMDS**

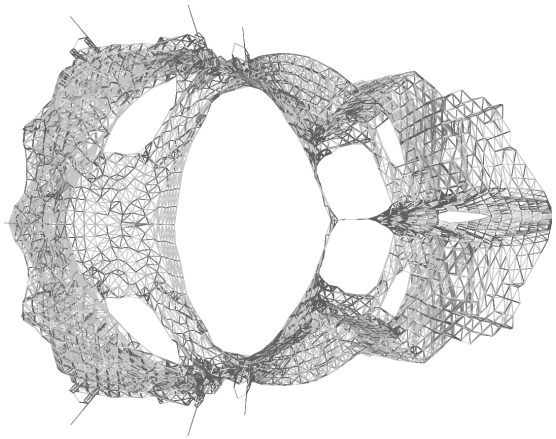


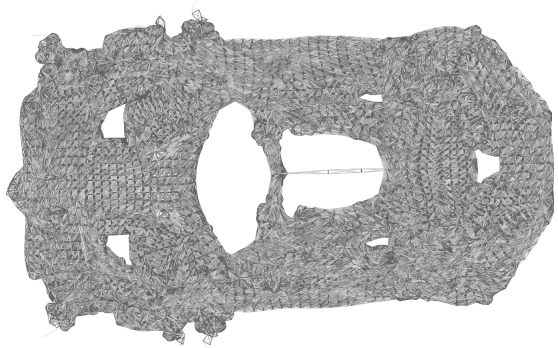
**MaxEnt**

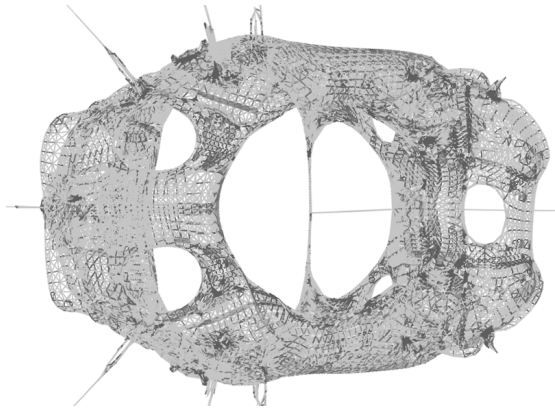
**MulMent**

# Example Drawings

**bcsstk31**



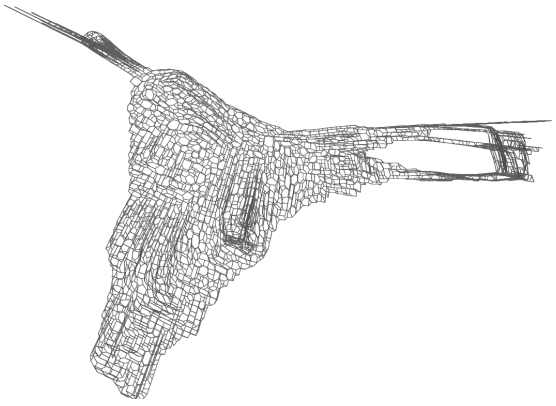




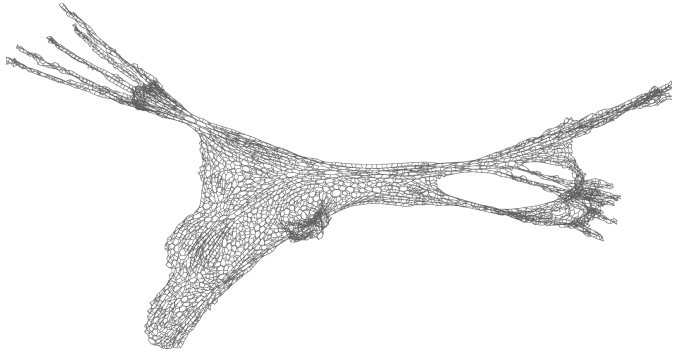
**PMDS** **MaxEnt** **MulMent**

# Example Drawings

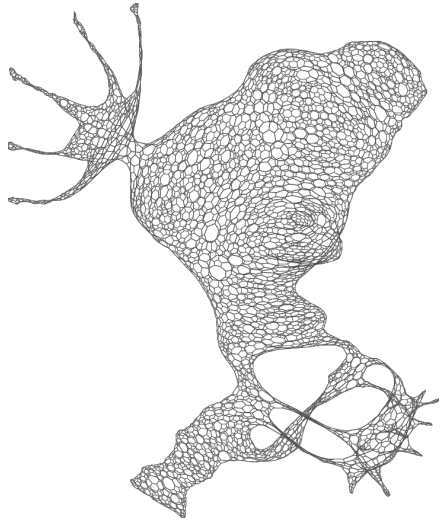
**commanche**



**PMDS**



**MaxEnt**



**MulMent**