Karlsruher Institut für Technologie

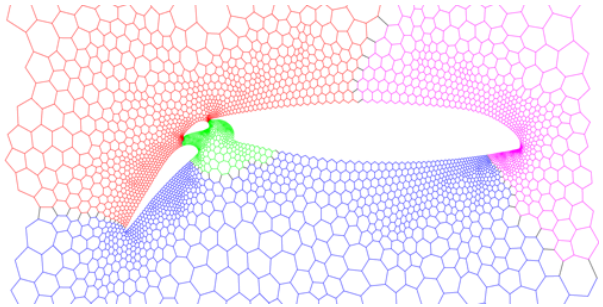
High Quality Graph Partitioning

Peter Sanders, *Christian Schulz*

ISMP 2012

Institute for Theoretical Computer Science, Algorithmics II

HPC



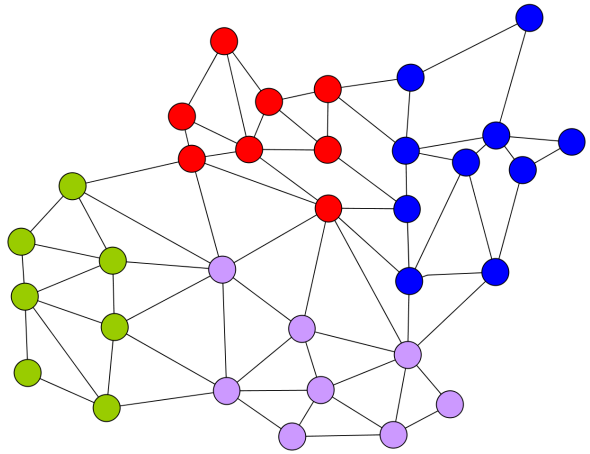
- Mesh partitioned
  1. nodes $\leftrightarrow$ data, computation
  2. edges $\leftrightarrow$ interdependencies
- All PE's get same amount of work
- Communication is expensive

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

ϵ-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.

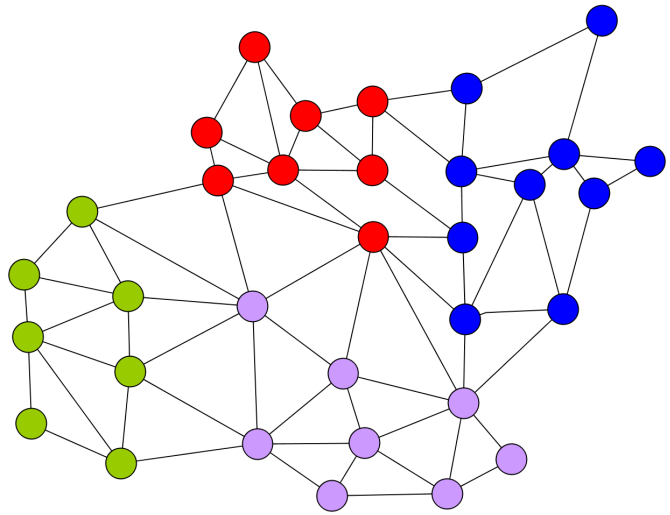
- total node weight of each block $\leq \frac{1+\epsilon}{k}$ total node weight
- total weight of cut edges as small as possible



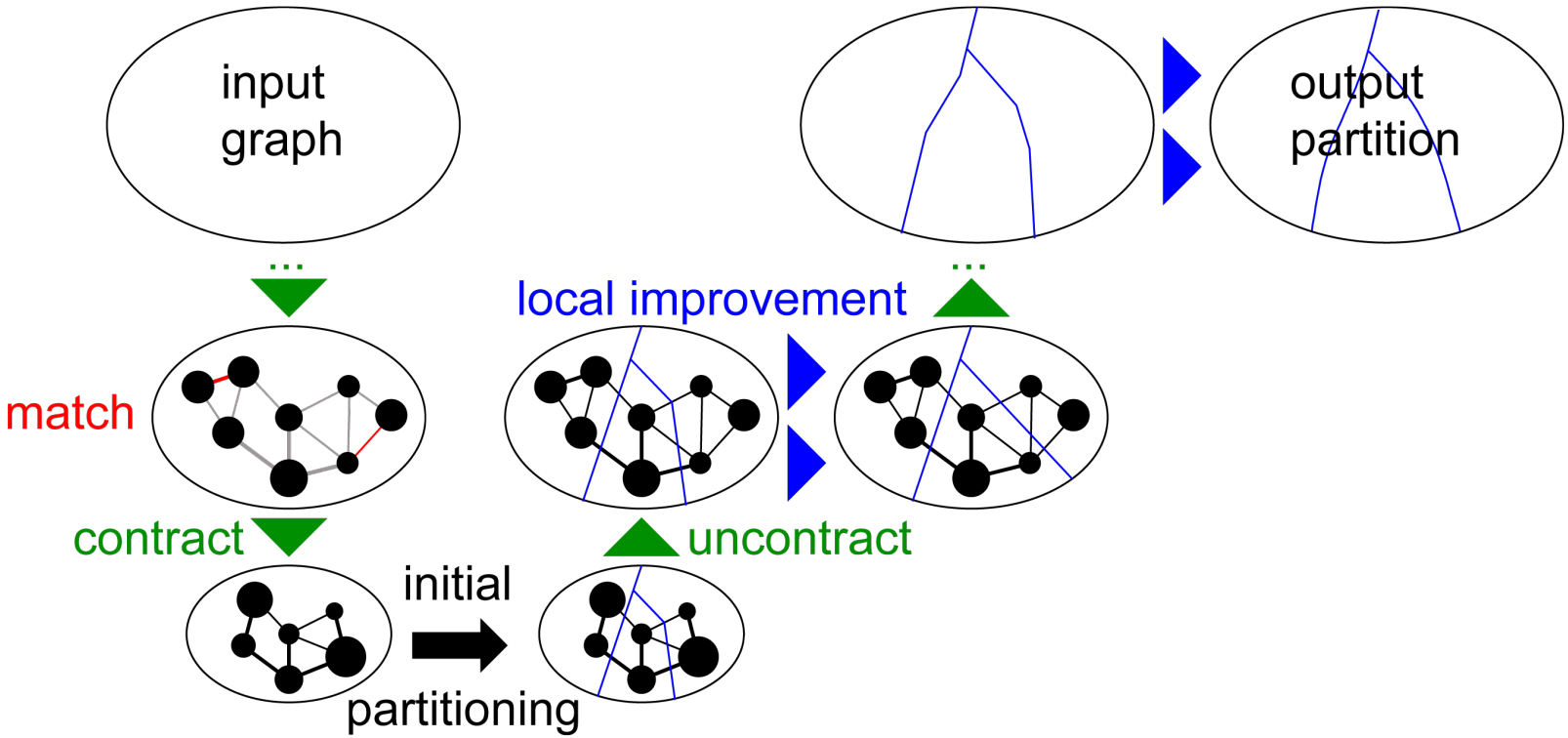
Applications:
linear equation systems, VLSI design, route planning, ...

Overview

- Introduction
- Multilevel Algorithms
- Advanced Techniques
- Evolutionary Techniques
- Experiments
- Summary



Multi-Level Graph Partitioning

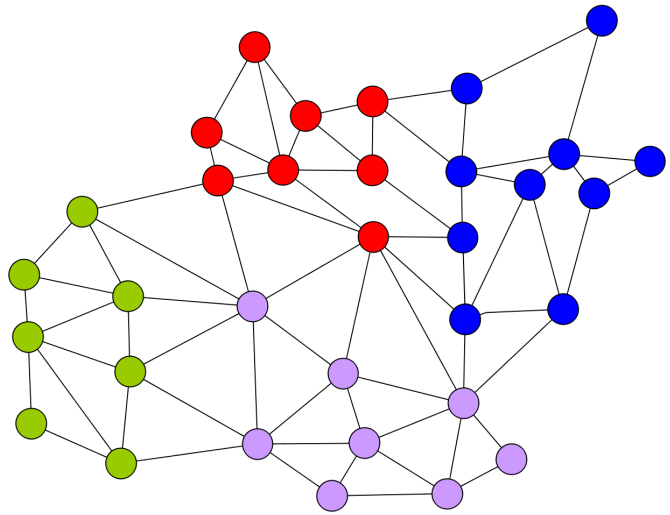


Successful in existing algorithms:
Metis, Scotch, DiBaP,. . . , **KaPPa**, **KaSPar**, **KaFFPa**, **KaFFPaE**

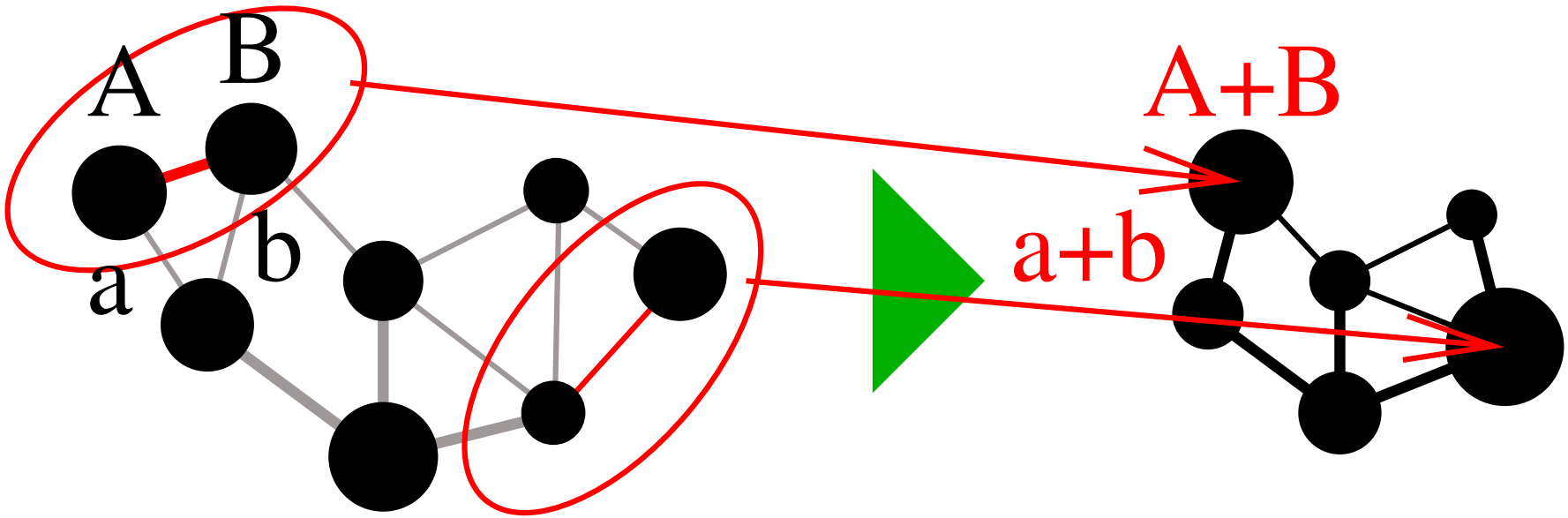
Advanced Techniques

Talk Today

- GP Folklore
- Edge Ratings
- Flow Based Refinements
- More Localized Local Search
- F-cycles for Graph Partitioning



Contract Matchings



But how are the edges selected?

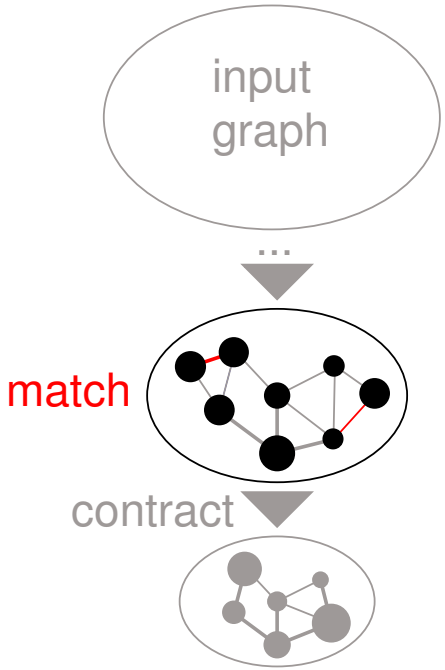
Graph Partitioning

Matching Selection

Goals:

1. large edge weights $\rightsquigarrow$ sparsify
2. large #edges $\rightsquigarrow$ few levels
3. uniform node weights $\rightsquigarrow$ "represent" input
4. small node degrees $\rightsquigarrow$ "represent" input

$\rightsquigarrow$ unclear objective
$\rightsquigarrow$ gap to approx. weighted matching
which only considers 1.,2.



Our Solution:
Apply approx. weighted matching to general edge rating function

Graph Partitioning

Edge Ratings

$$\omega(\{u,v\})$$

$$\text{expansion}^*(\{u,v\}) := \frac{\omega(\{u,v\})}{c(u)c(v)}$$

$$\text{expansion}^{*2}(\{u,v\}) := \frac{\omega(\{u,v\})^2}{c(u)c(v)}$$

$$\text{innerOuter}(\{u,v\}) := \frac{\omega(\{u,v\})}{\text{Out}(v) + \text{Out}(u) - 2\omega(u,v)}$$

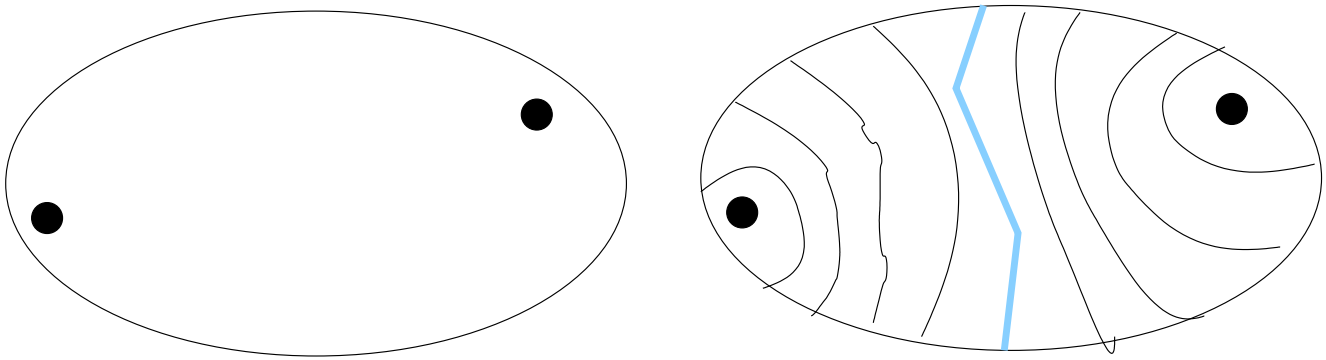
where c = node weight, ω =edge weight,
$\text{Out}(u) := \sum_{\{u,v\} \in E} \omega(\{u,v\})$

Initial Partitioning

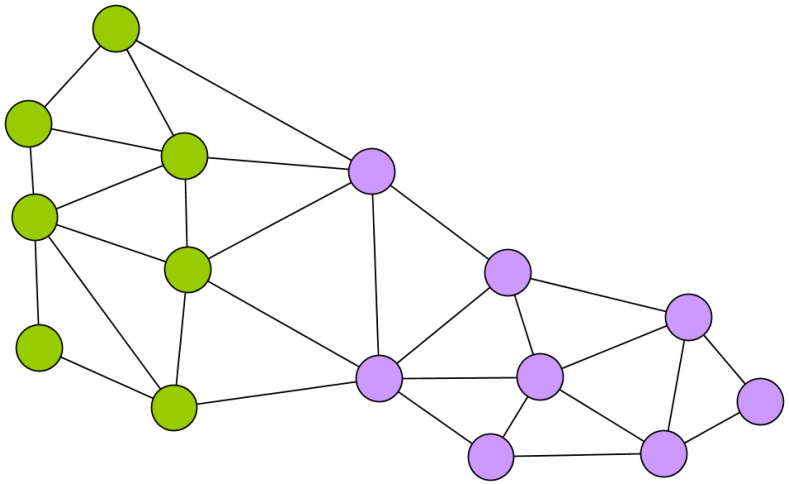
Usually done by recursive bipartitioning, e.g. using BFS

- we currently use Scotch [Pellegrini]
- multiple tries pay off

Open Problem:
Direct k-partitioner that achieves better quality or speed.

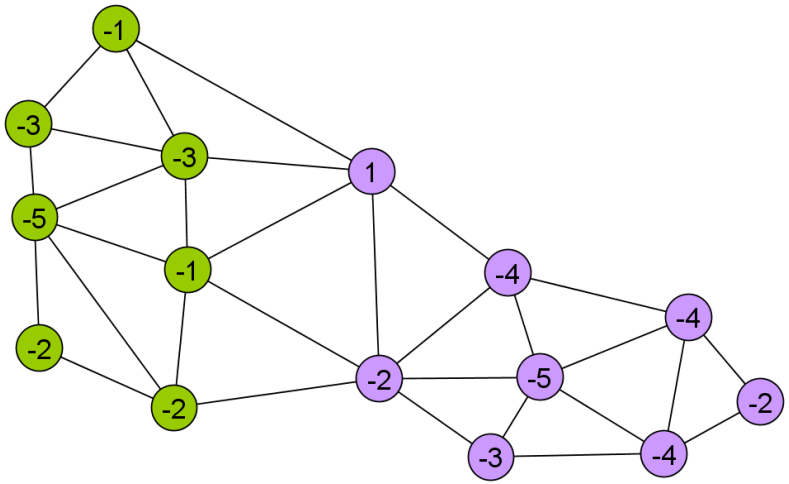


FM Local Search



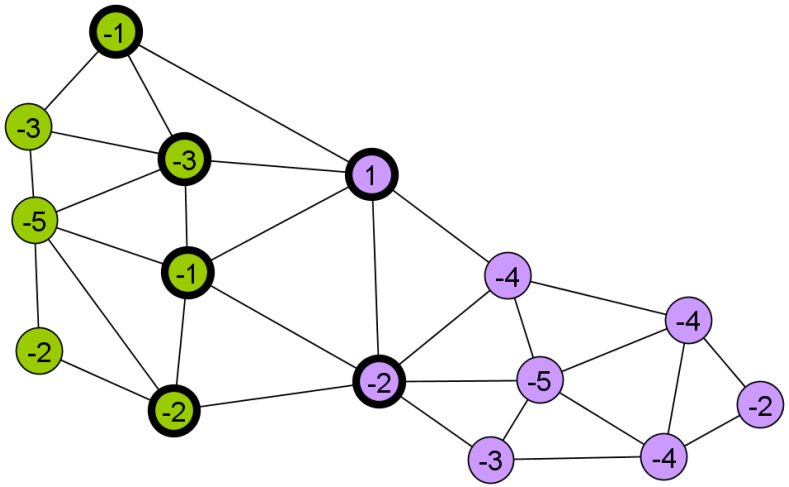
- compute gain $\forall\ v \in V$

FM Local Search



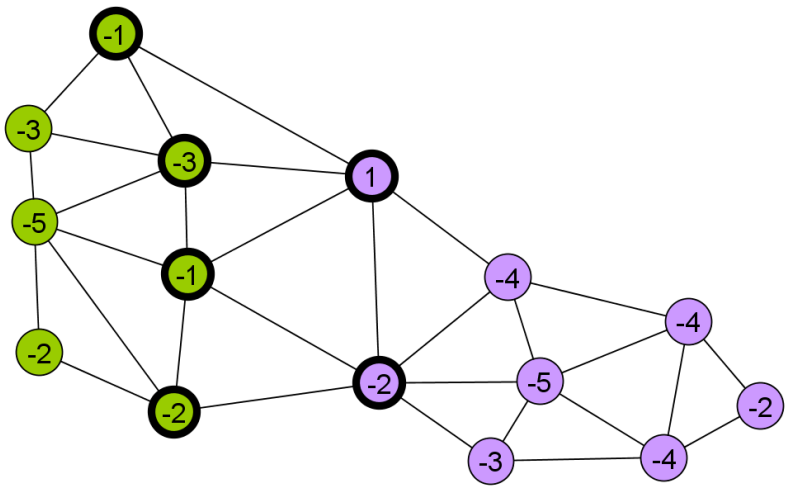
- compute gain $\forall\, v \in V$
- $g(v) = d_{ext}(v) - d_{int}(v)$

FM Local Search



- compute gain $\forall\, v \in V$
- $g(v) = d_{ext}(v) - d_{int}(v)$
- store gain of boundary nodes (e.g. in a heap)

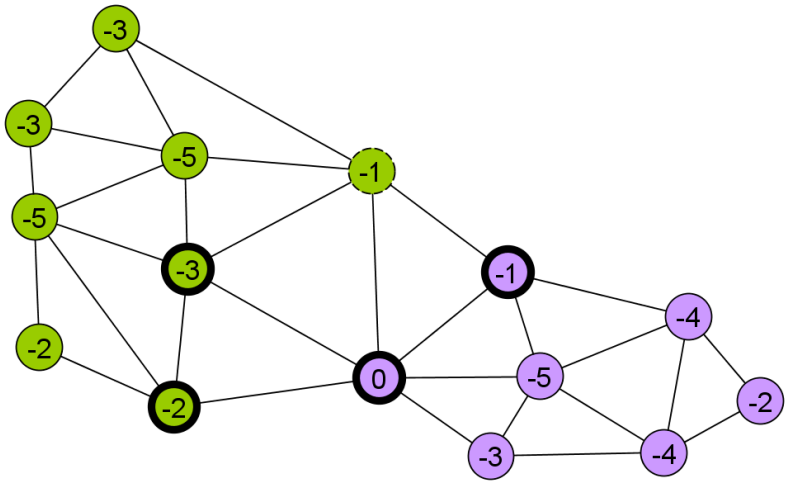
FM Local Search



Step:	0
Edge Cut:	5

- move highest gain vertices to opposite block
- each node at most once
- update gain of neighbors

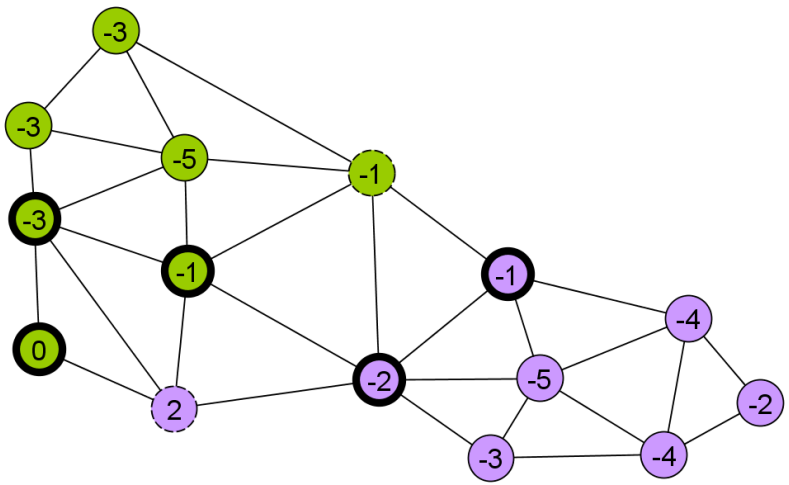
FM Local Search



Step:	0	1
Edge Cut:	5	4

- move highest gain vertices to opposite block
- each node at most once
- update gain of neighbors

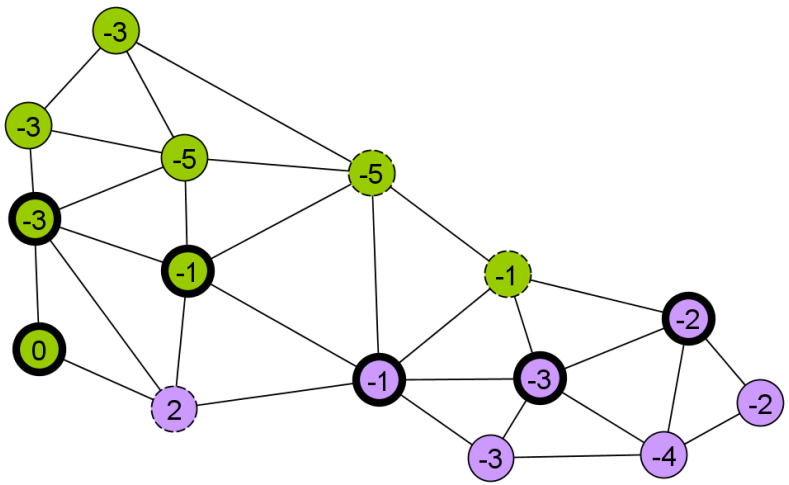
FM Local Search



Step:	0	1	2
Edge Cut:	5	4	6

- move highest gain vertices to opposite block
- each node at most once
- update gain of neighbors

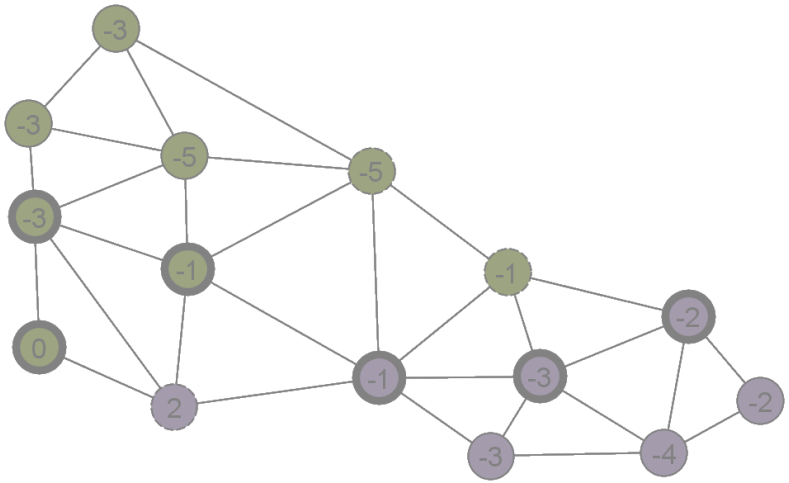
FM Local Search



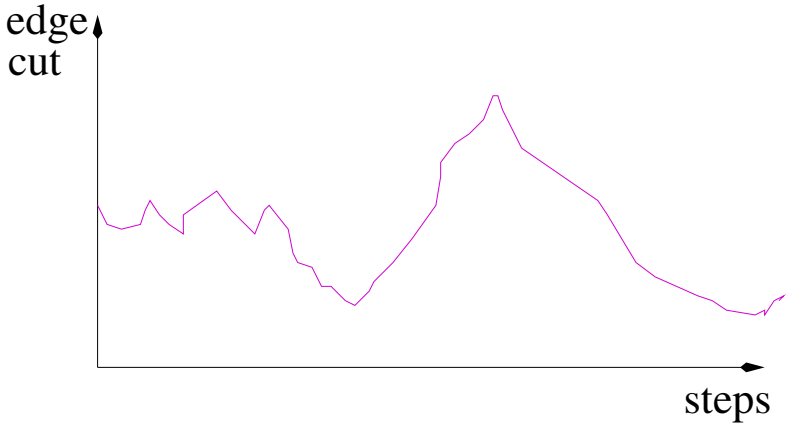
- move highest gain vertices to opposite block
- each node at most once
- update gain of neighbors

Step:	0	1	2	3
Edge Cut:	5	4	6	8

FM Local Search

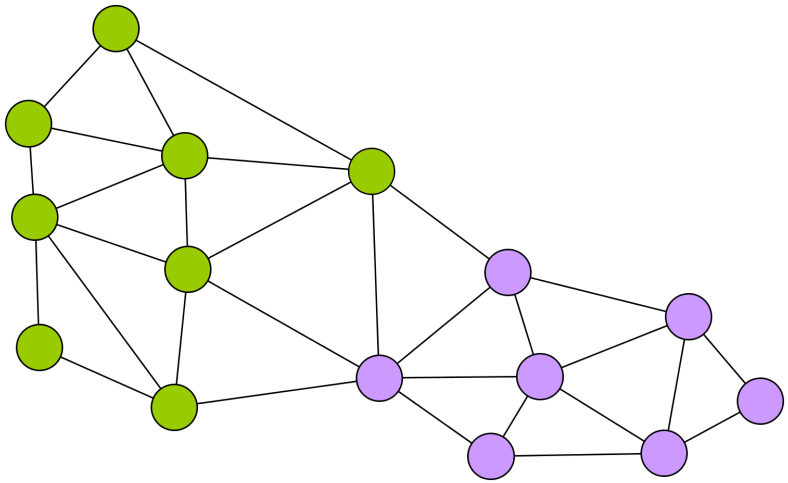


- stop after limit
- take best edge cut
- within balance constraint

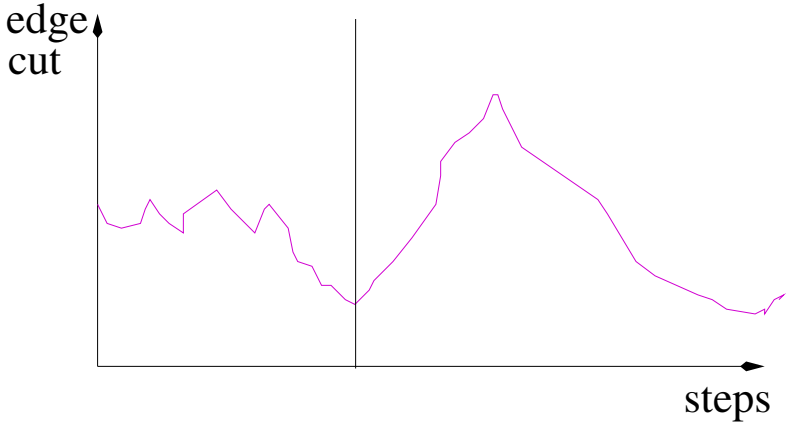


Step:	0	1	2	3
Edge Cut:	5	4	6	8

FM Local Search



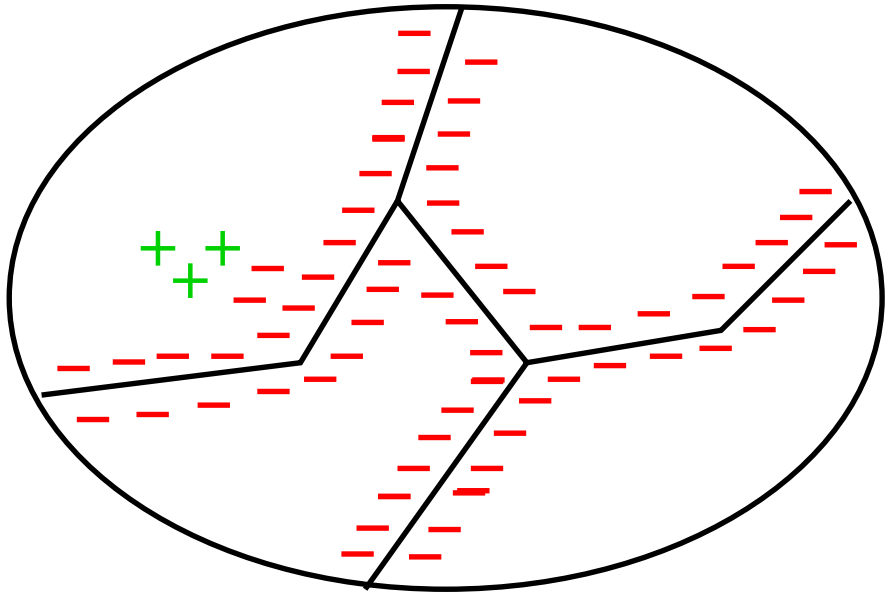
- stop after limit
- take best edge cut
- within balance constraint



Step:	0	**1**	2	3
Edge Cut:	5	**4**	6	8

FM Local Search – Discussion

+ Generalizable for multiple blocks
+ Linear time
– Unlikely to find improvements requiring $\geq$ 2 negative gain moves

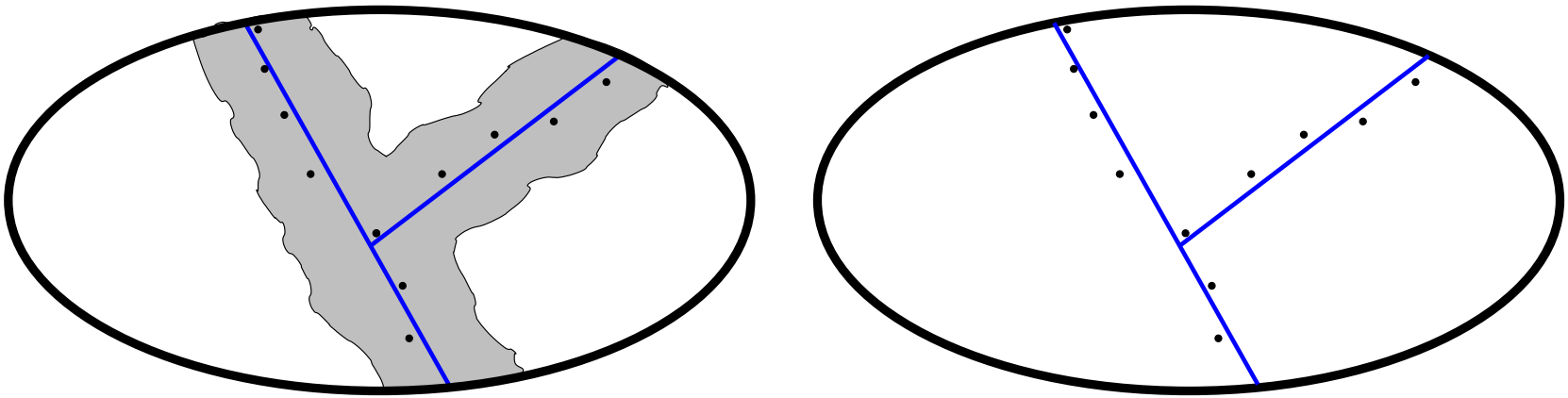


More Localized Local Search

More Localized Local Search

inspired by n-level search

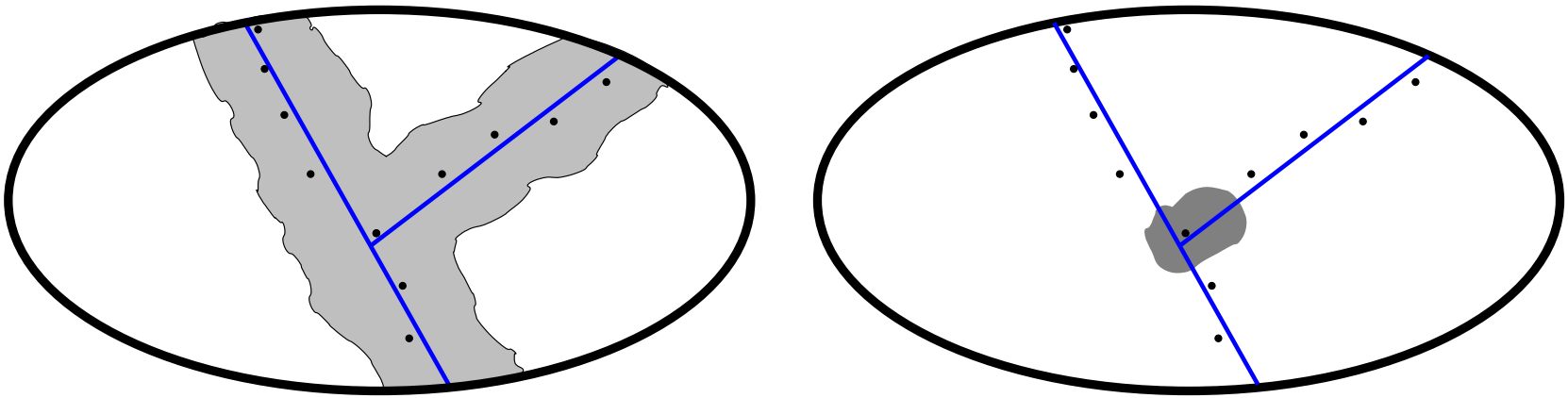
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

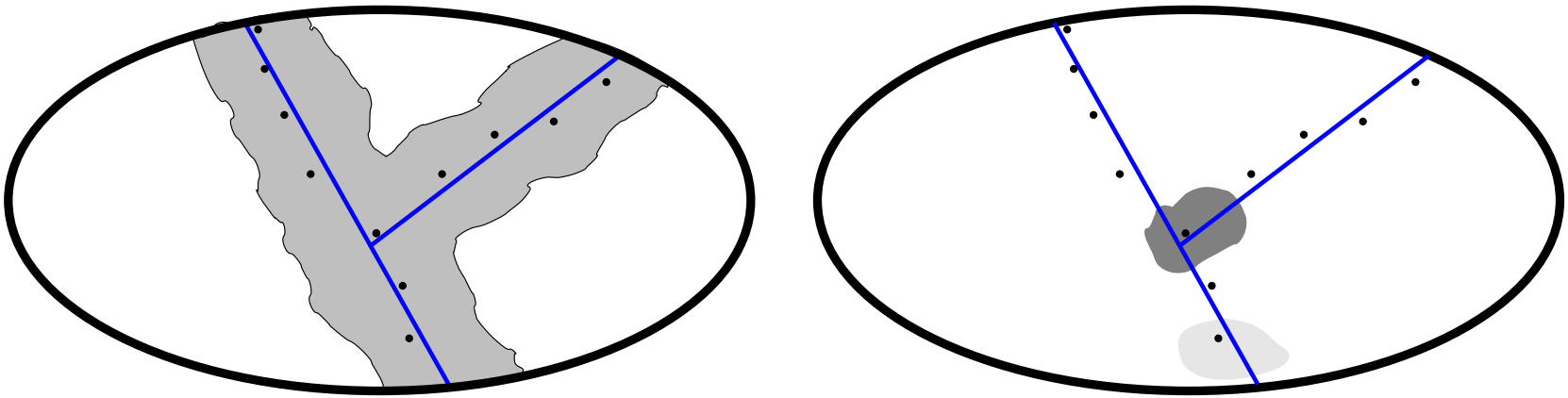
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

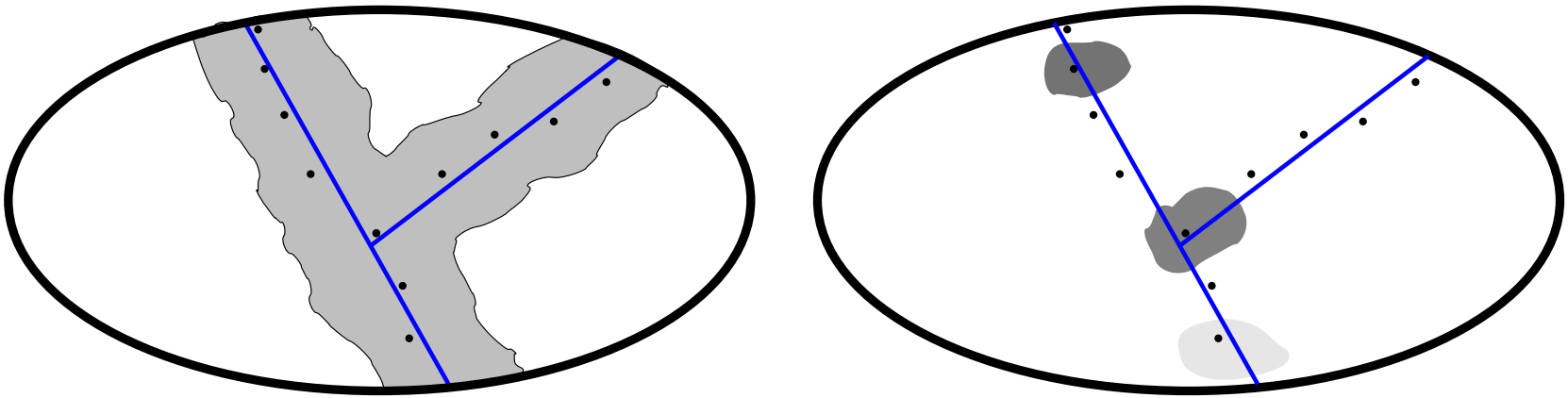
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

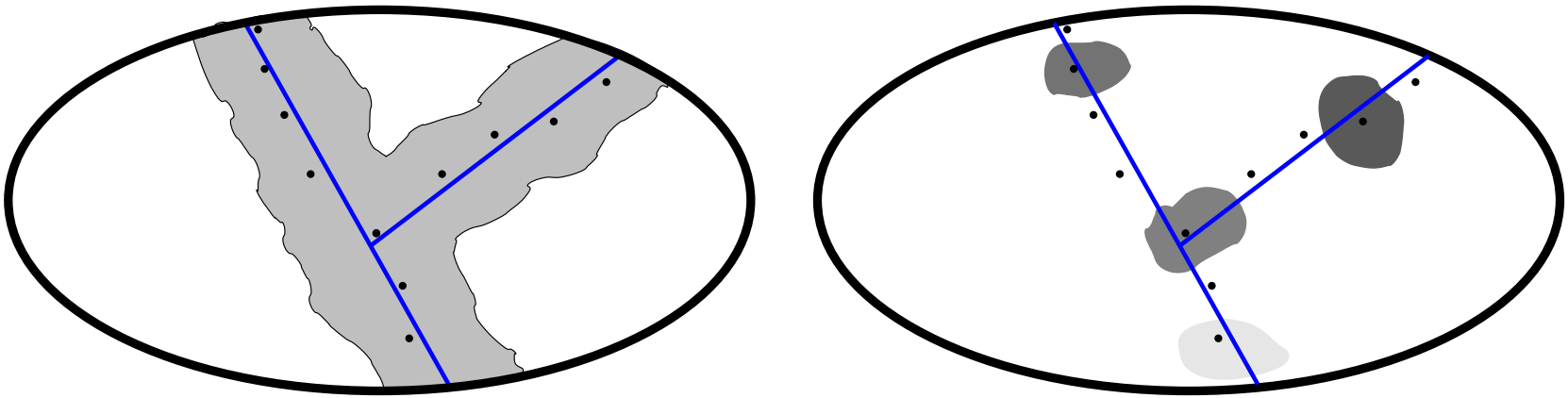
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

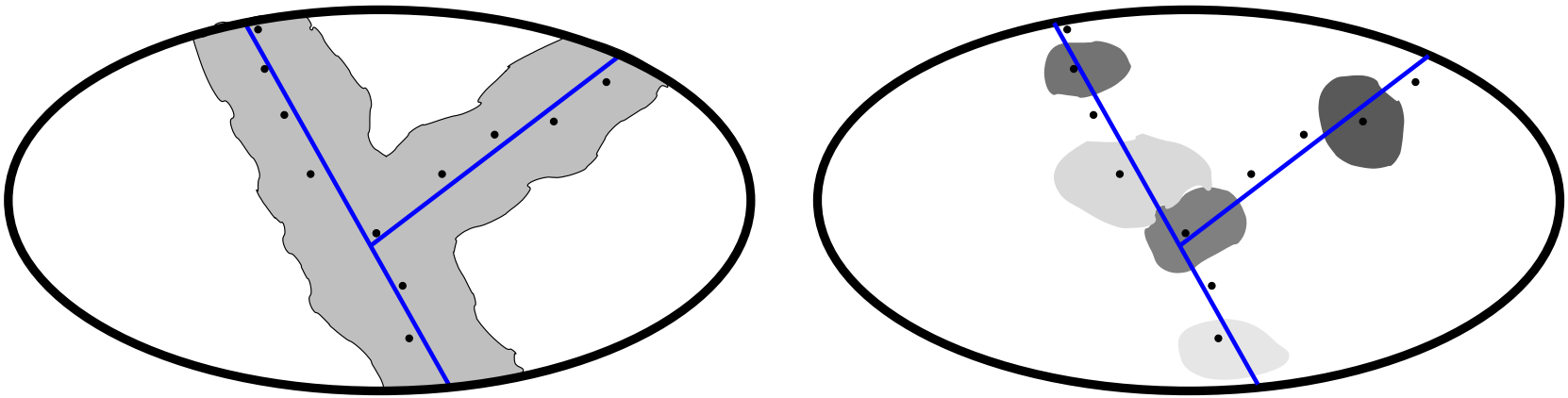
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

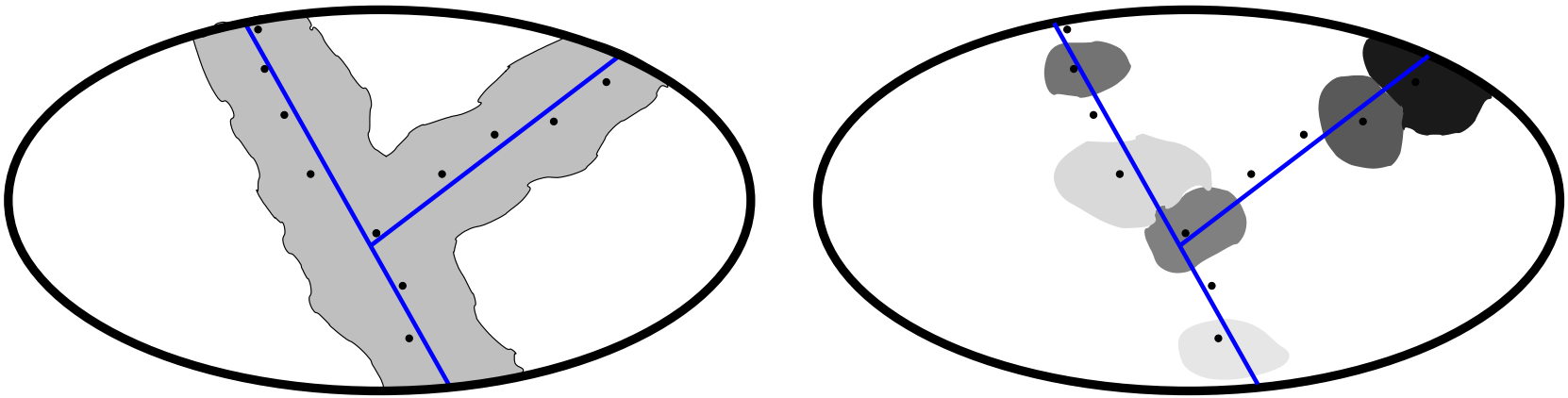
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

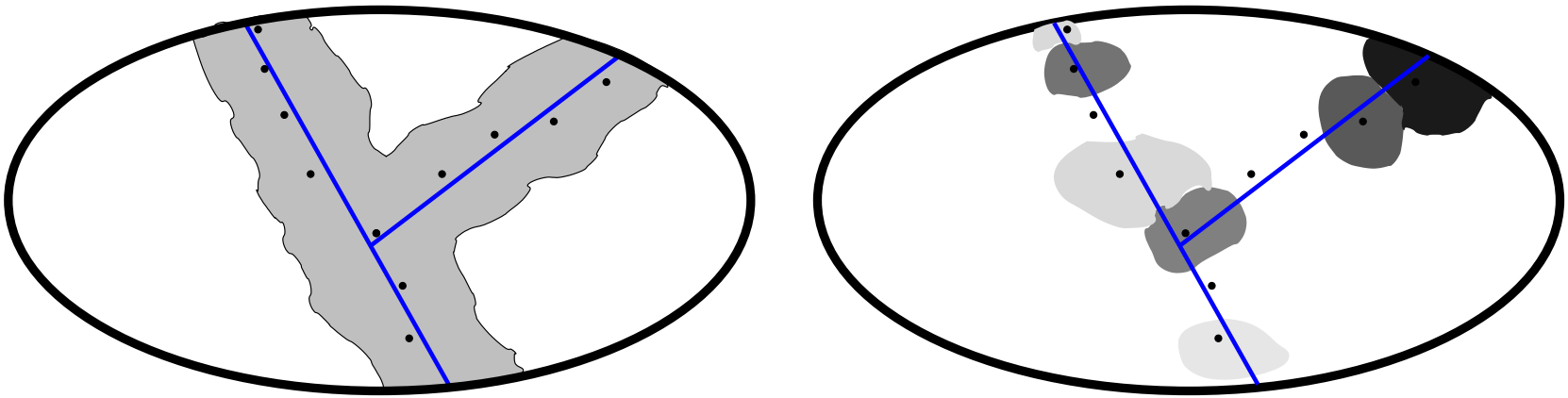
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

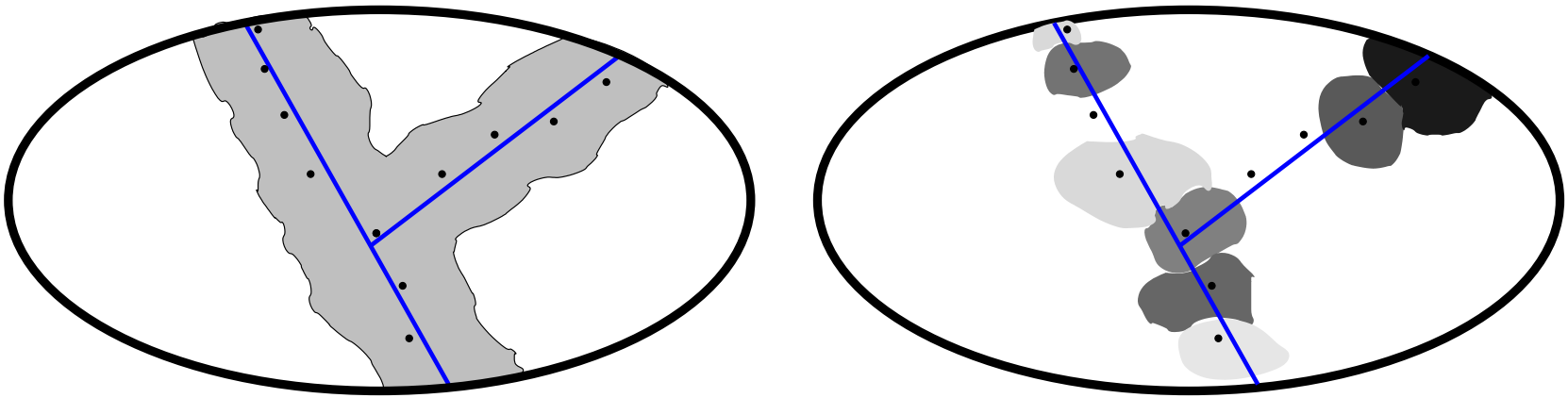
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

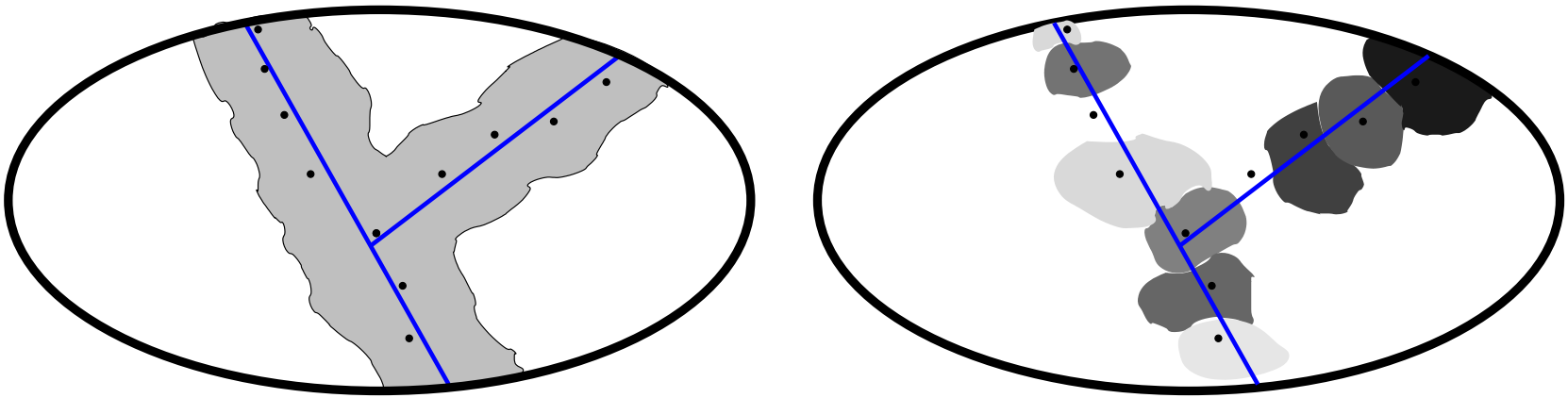
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

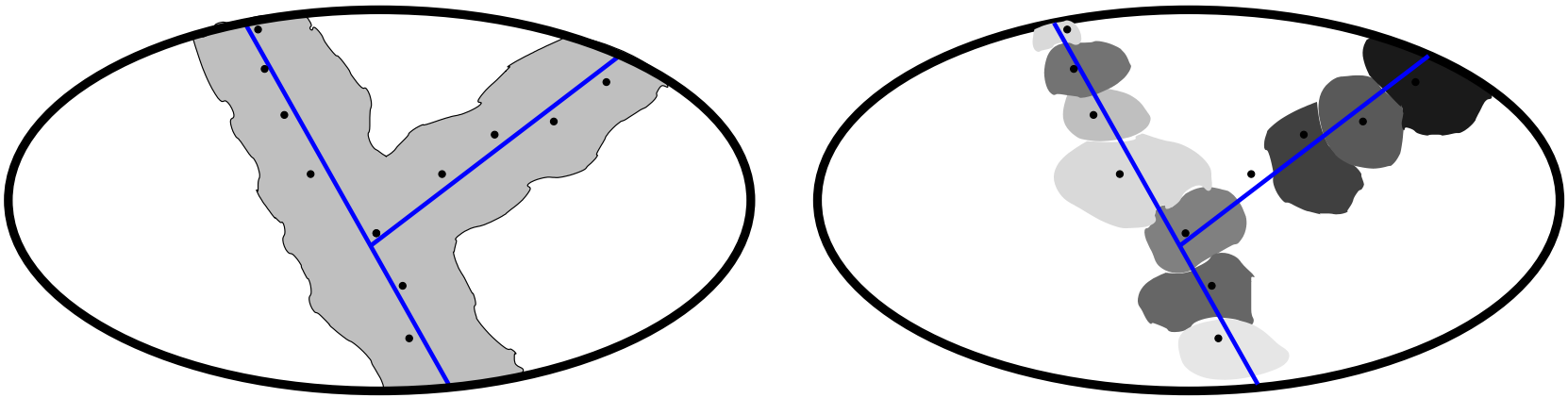
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

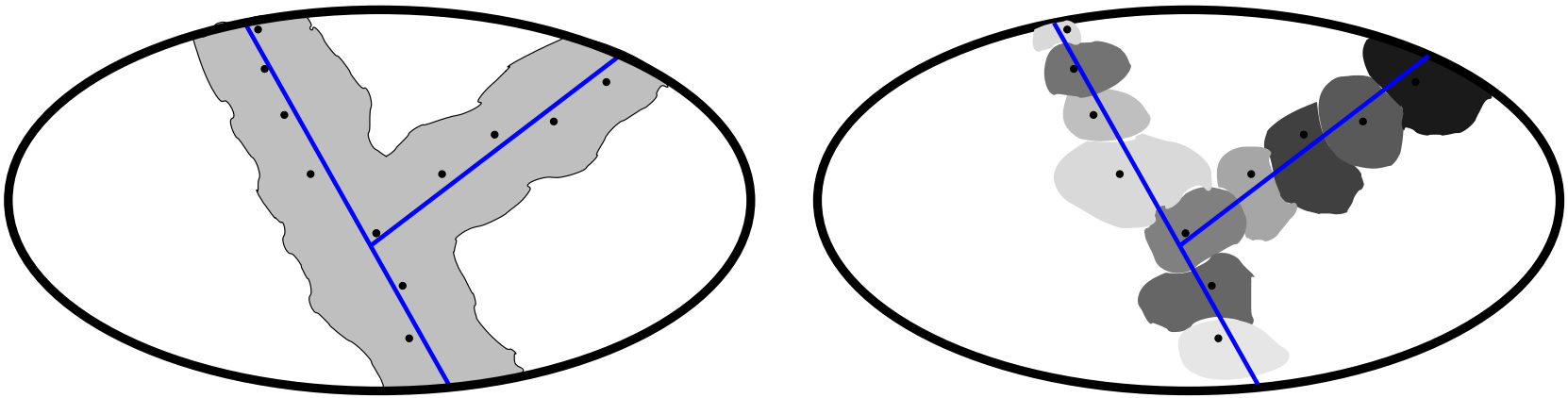
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



More Localized Local Search

inspired by n-level search

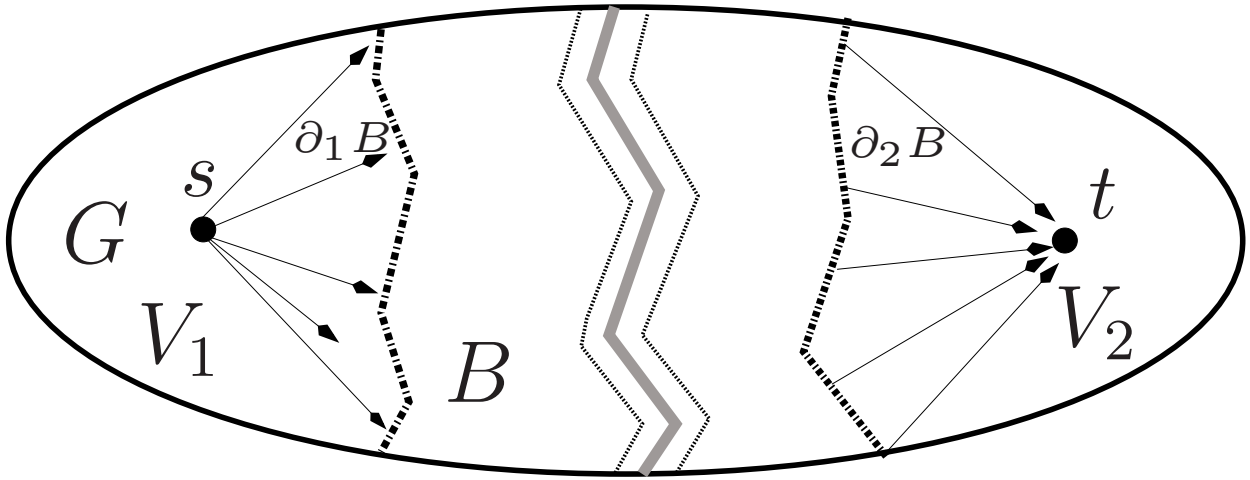
- Typical: k-way local search initialized with complete boundary
- Localization:
  1. complete boundary $\Rightarrow$ maintained todo list T
  2. initialize search with single node $v \in_{\text{rnd}} T$
  3. iterate until $T = \emptyset$
- each node moved at most once



Flows as Local Improvement

Flows as Local Improvement

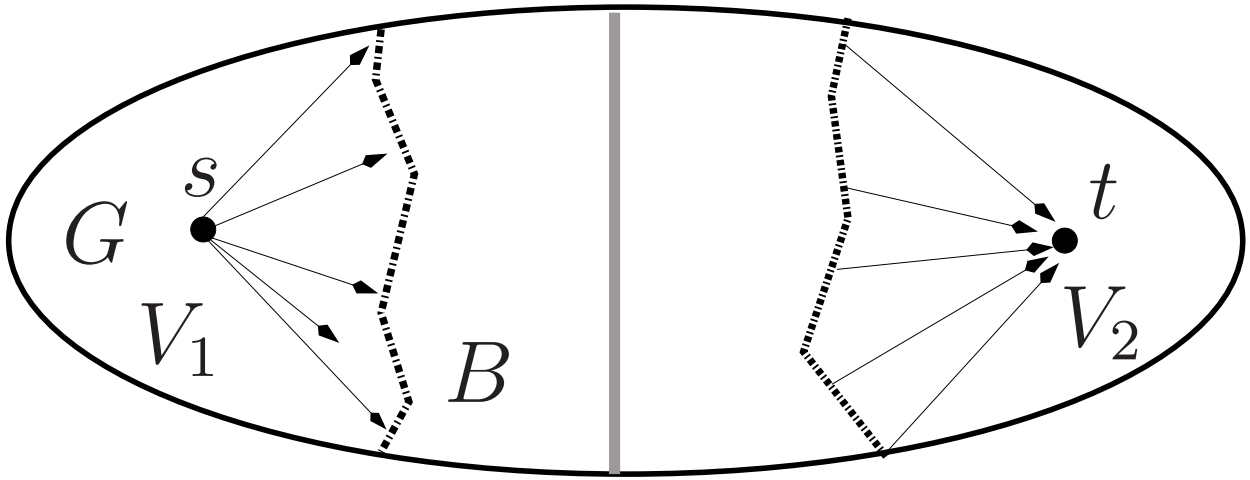
Two Blocks



- area B, such that **each** (s, t)-min cut is ϵ-balanced cut in G
- e.g. 2 times BFS (left, right)

Flows as Local Improvement

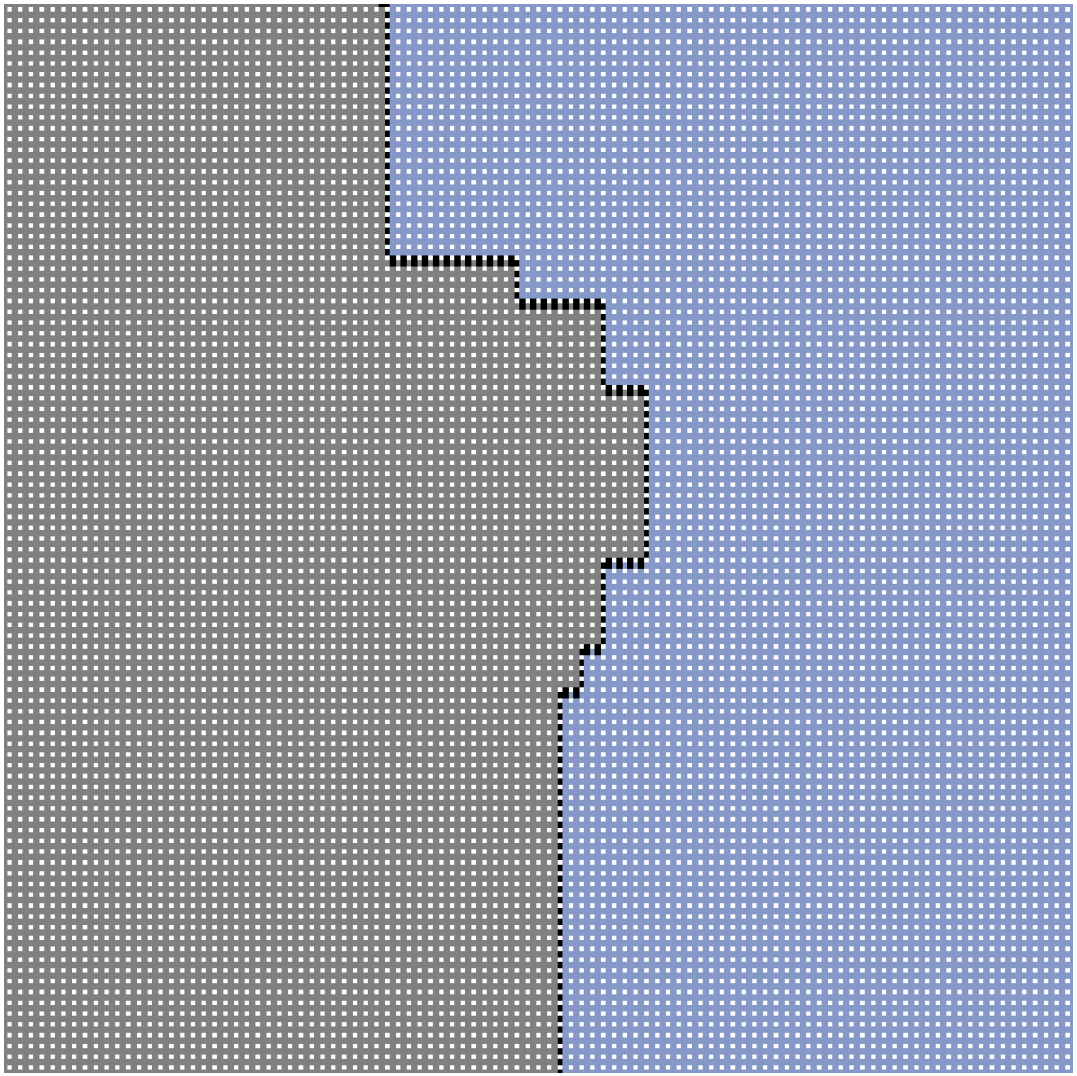
Two Blocks



- obtain optimal cut in B
- since each cut in B yields a feasible partition
  $\rightarrow$ improved two-partition
- advanced techniques possible and necessary
- combination with local search works best

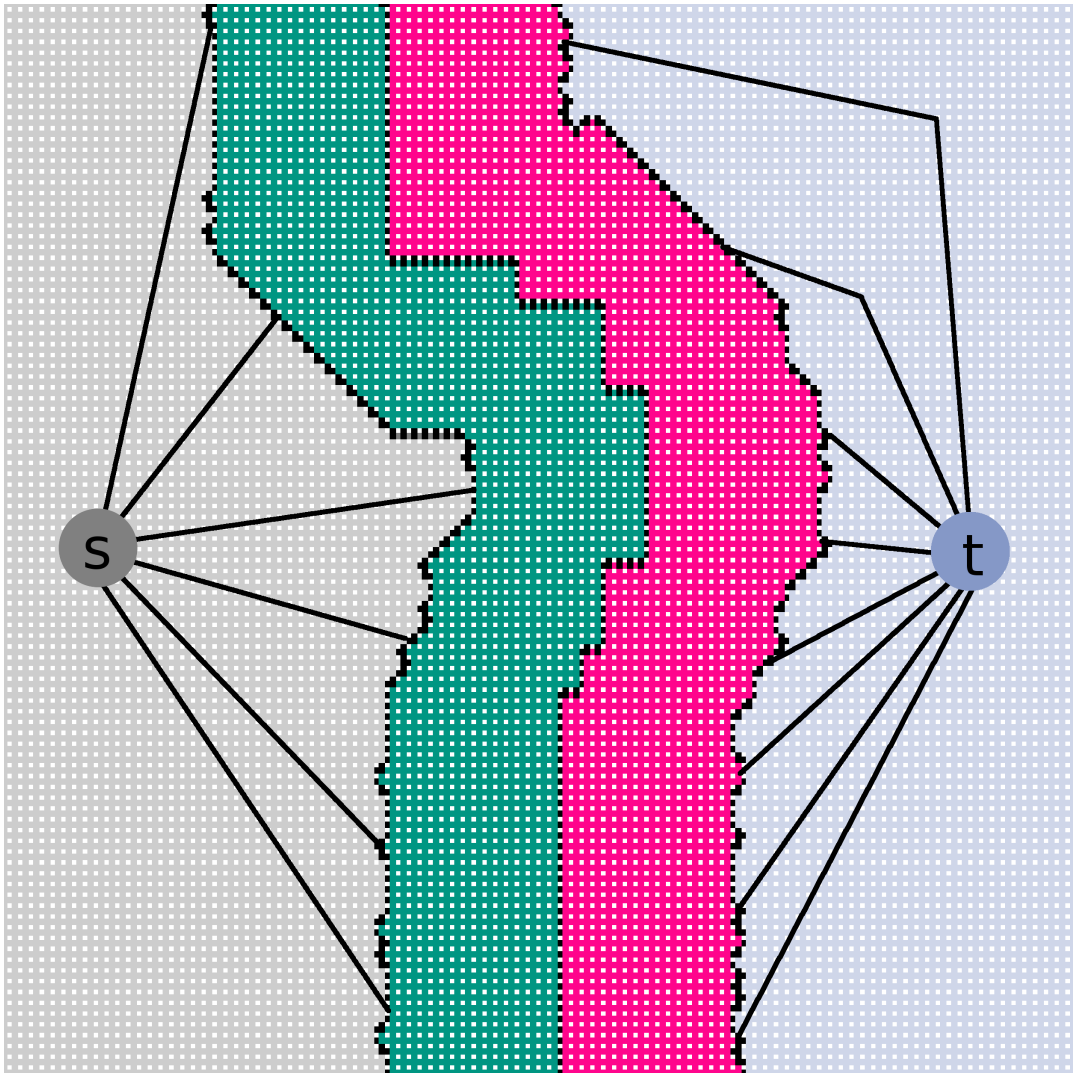
Example

100x100 Grid



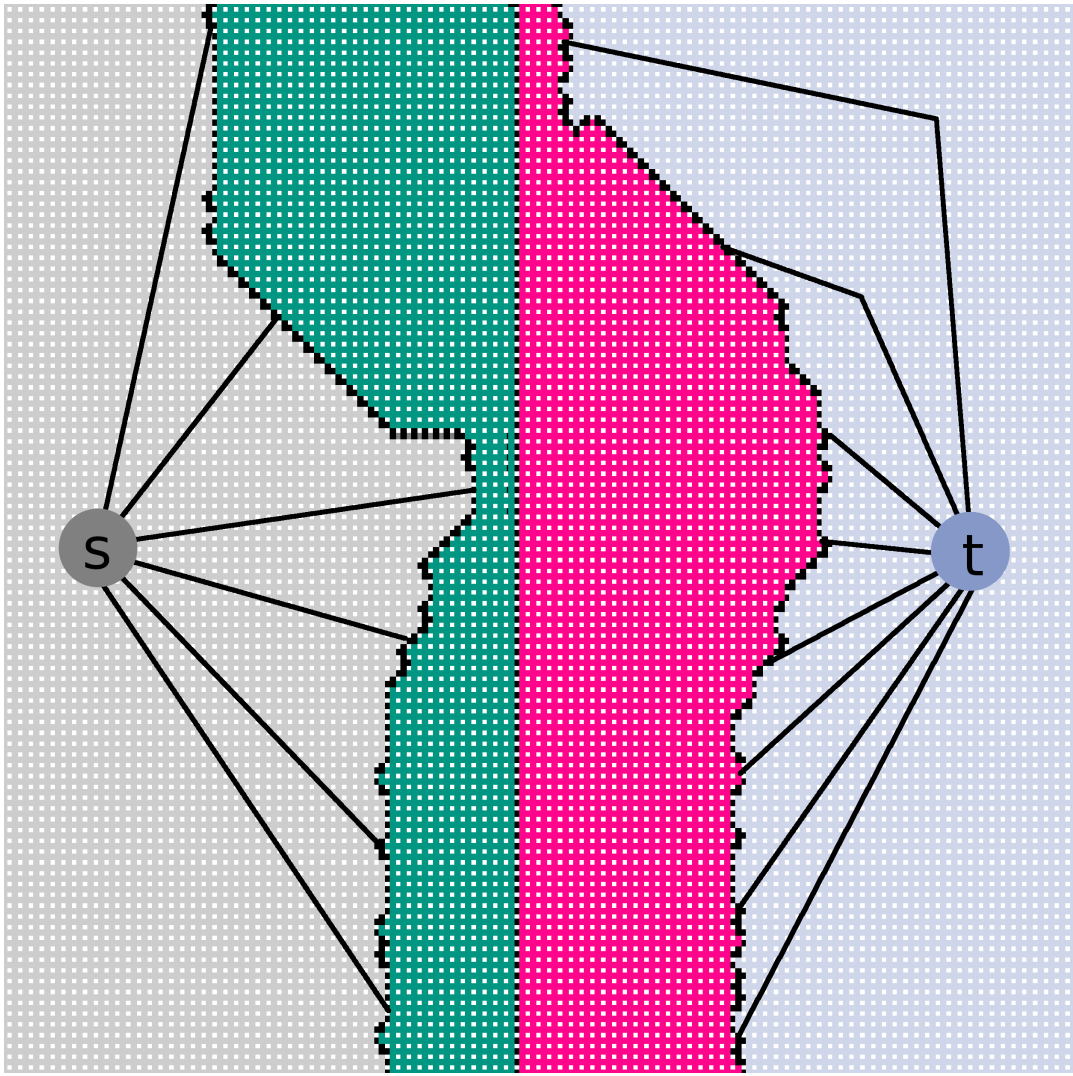
Example

Constructed Flow Problem (using BFS)



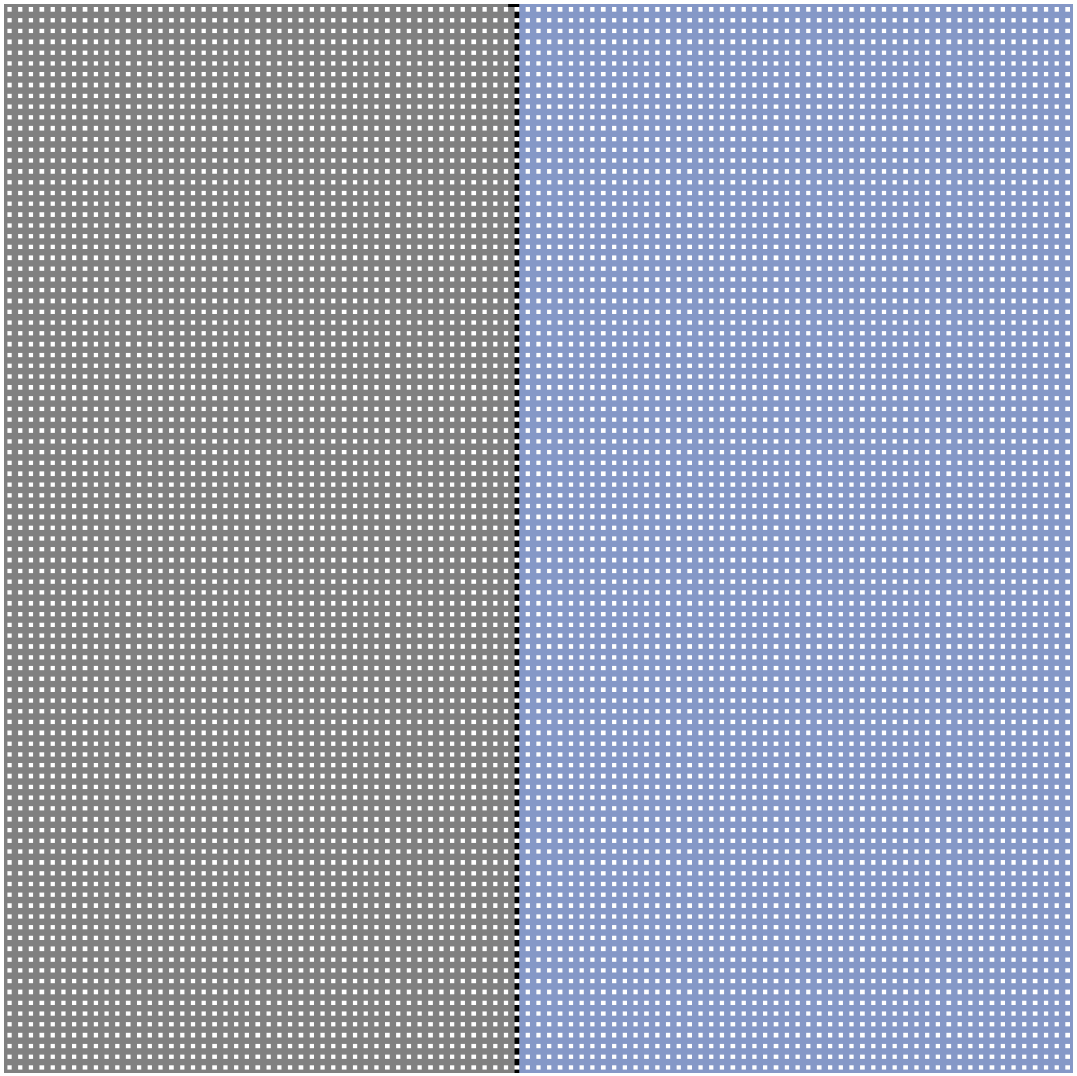
Example

Apply Max-Flow Min-Cut



Example

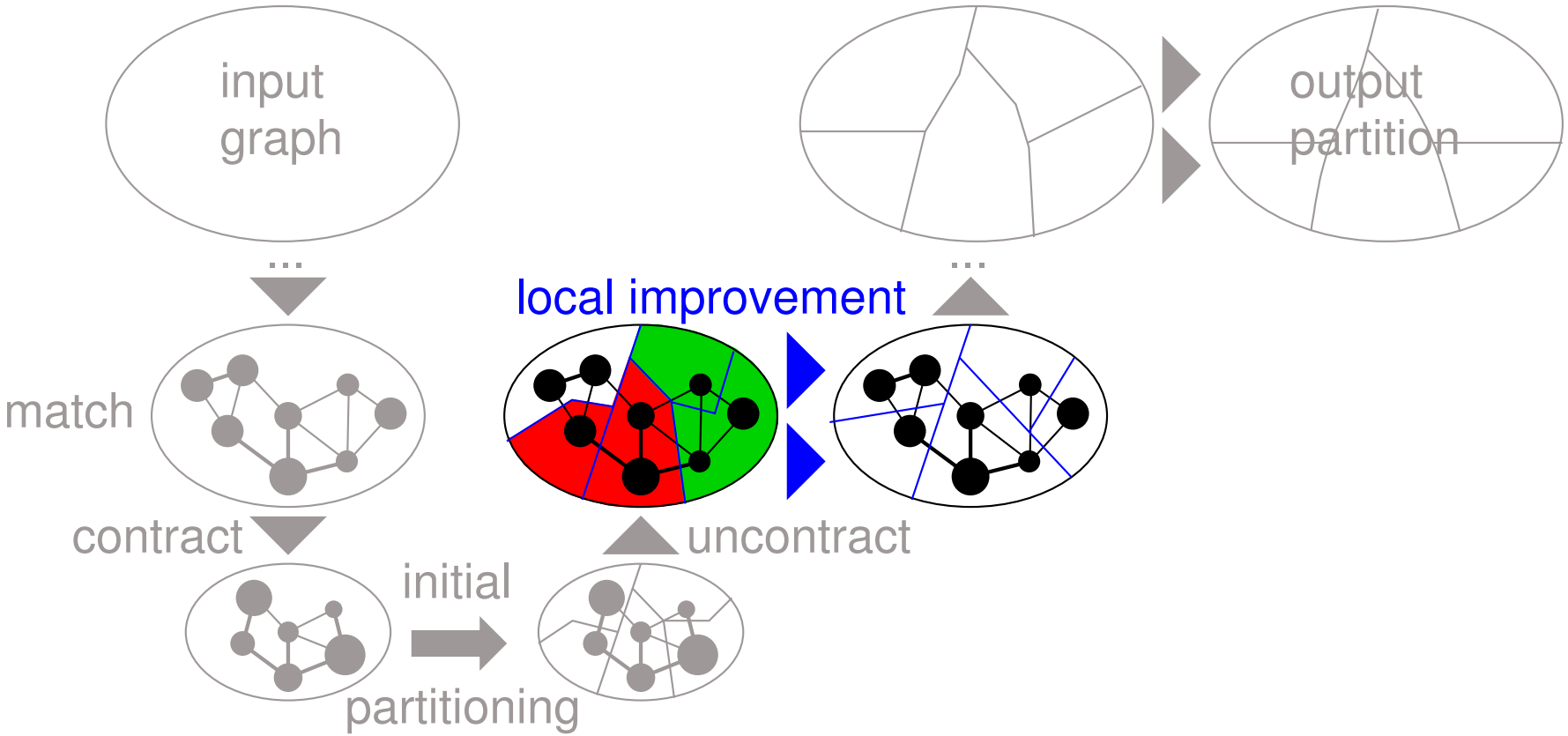
Output Improved Partition



Local Improvement for k-partitions

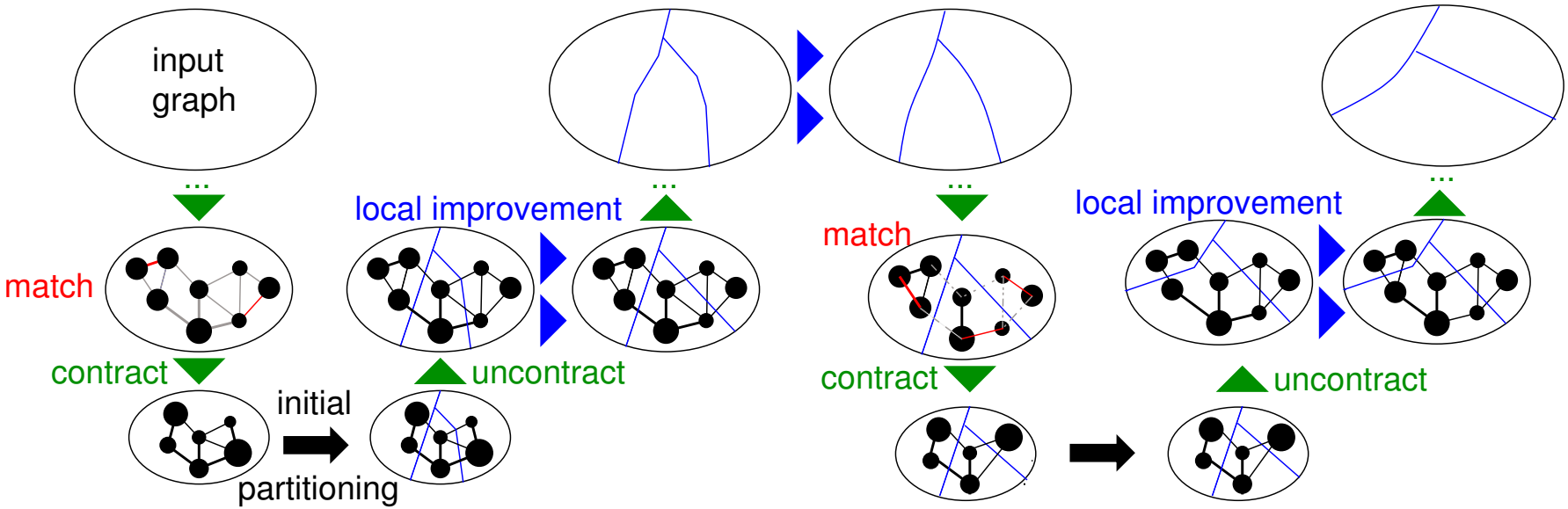
Using Flows?

on each pair of blocks



Iterated Multilevel

Iterated Multilevel [Walshaw 2004]



- don't contract cut edges
- adapt previous solution as initial partitioning
- cuts can only improve
- V-cycles / F-cycles

Evolutionary Techniques

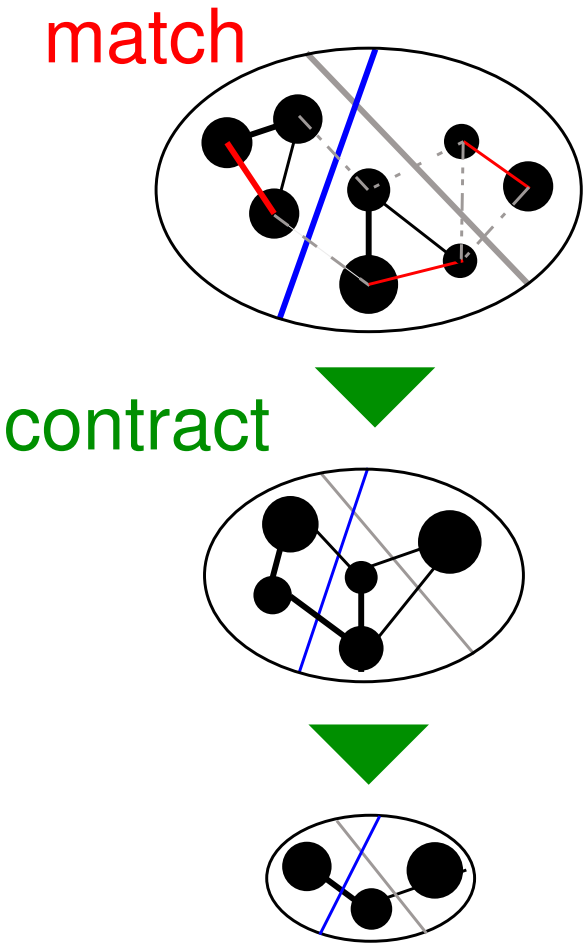
Distributed Evolutionary Graph Partitioning

- Evolutionary Algorithms:
  - population of individuals
  - selection (based on fitness), mutation, recombination, ...
- Evolutionary Graph Partitioning:

  individuals $\leftrightarrow$ partitions

  fitness $\leftrightarrow$ edge cut

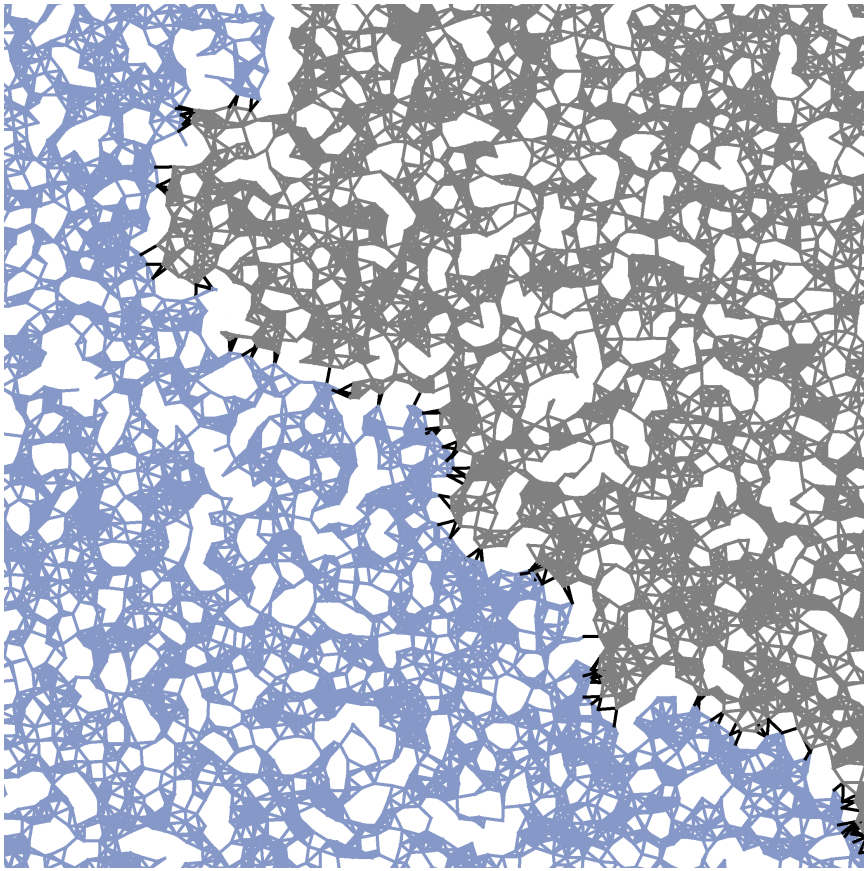
Combine

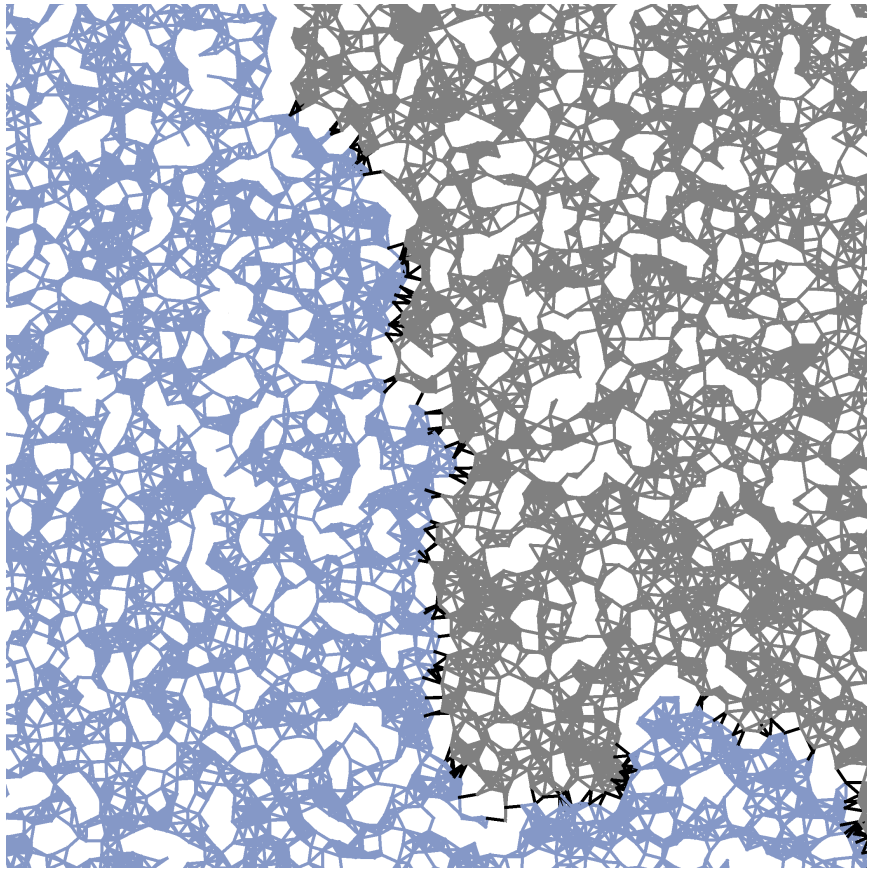


- two individuals $\mathcal{P}_1$, $\mathcal{P}_2$:
  don't contract cut edges of $\mathcal{P}_1$ or $\mathcal{P}_2$
- until no matchable edge is left
- coarsest graph $\leftrightarrow$ Q-graph of overlay
- $\rightarrow$ exchanging good parts is easy
- inital solution: use better of both parents

Example

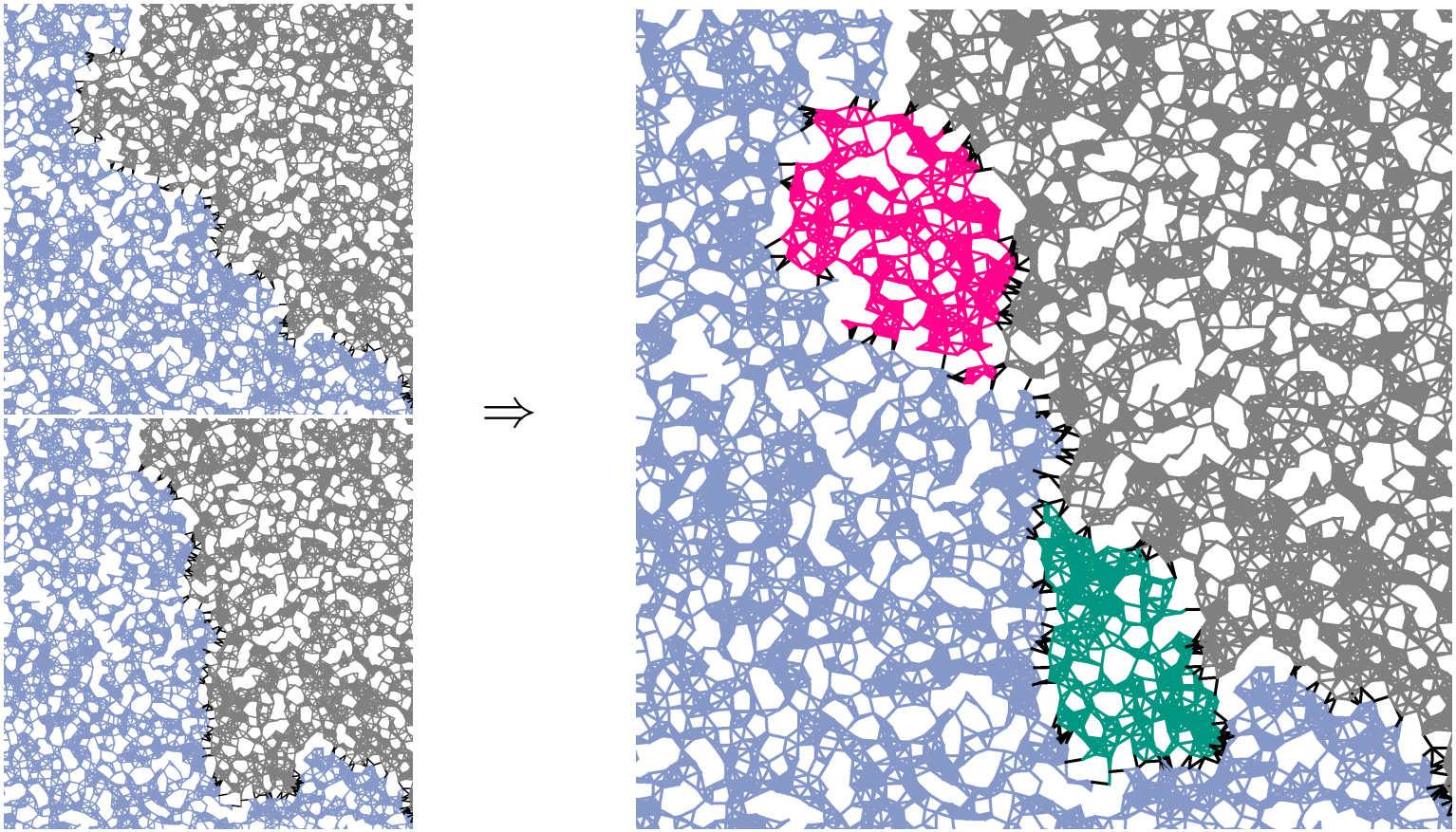
Two Individuals $\mathcal{P}_1$, $\mathcal{P}_2$





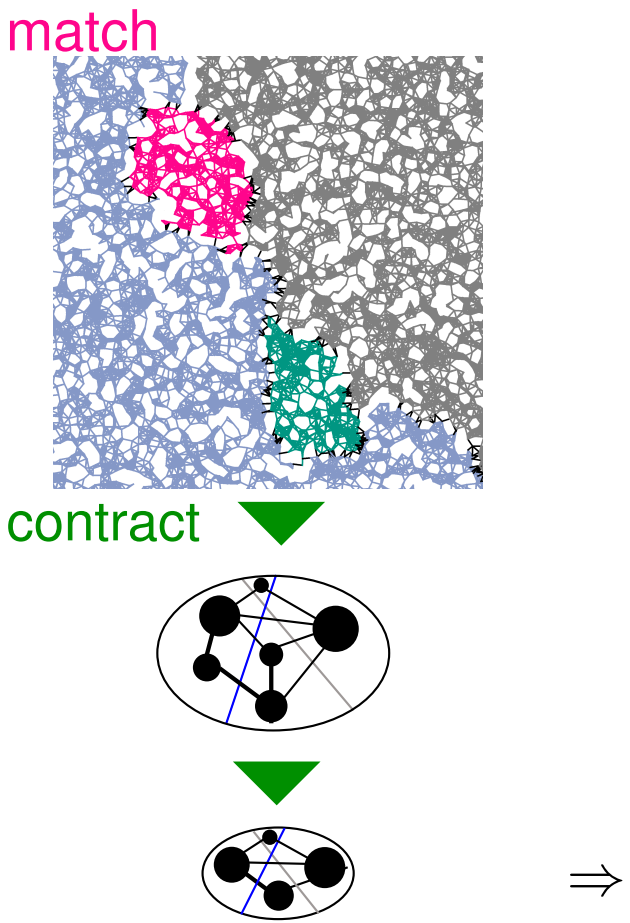
Example

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

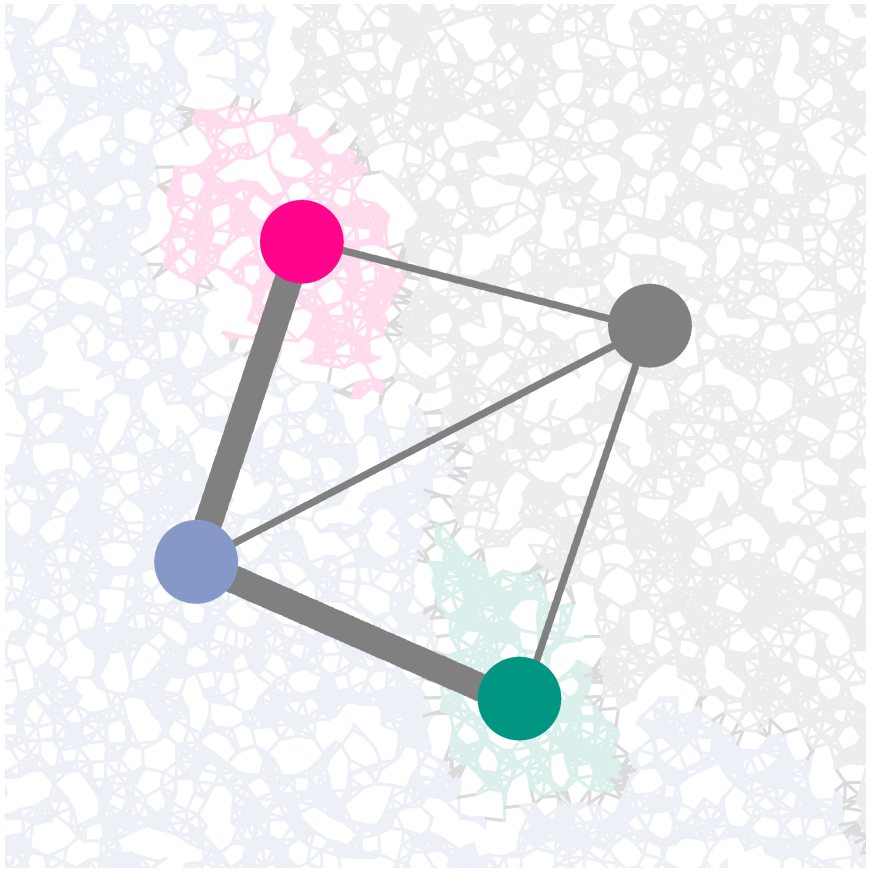


Example

Multilevel Combine of $\mathcal{P}_1$, $\mathcal{P}_2$

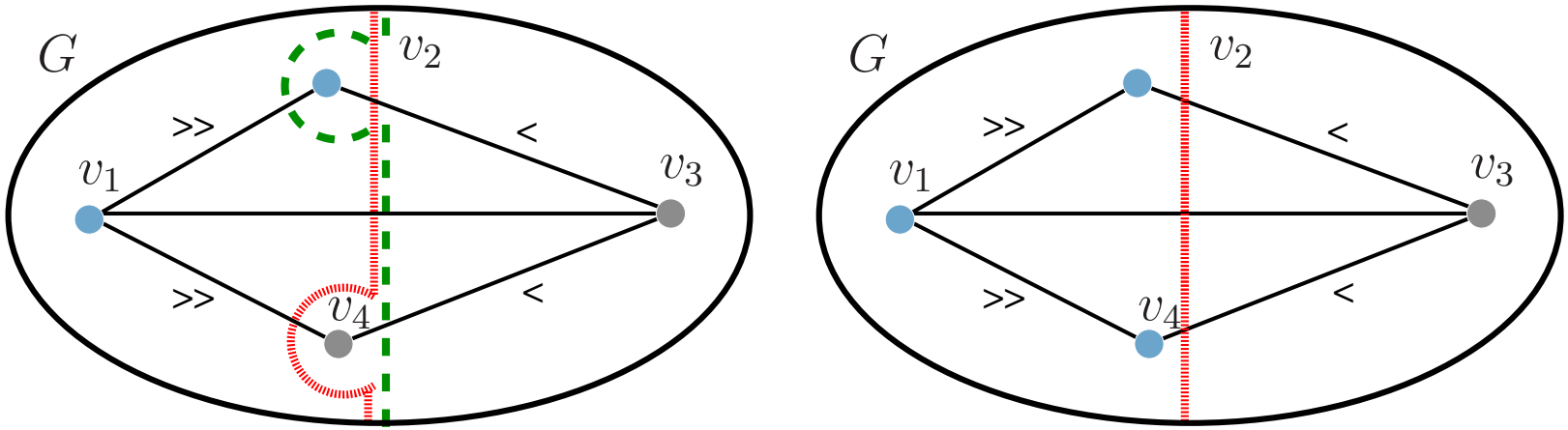


$\Rightarrow$



Exchanging good parts is easy

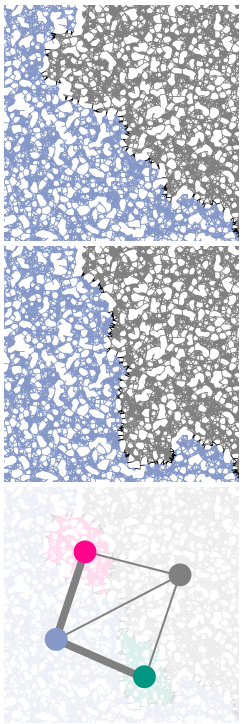
Coarsest Level



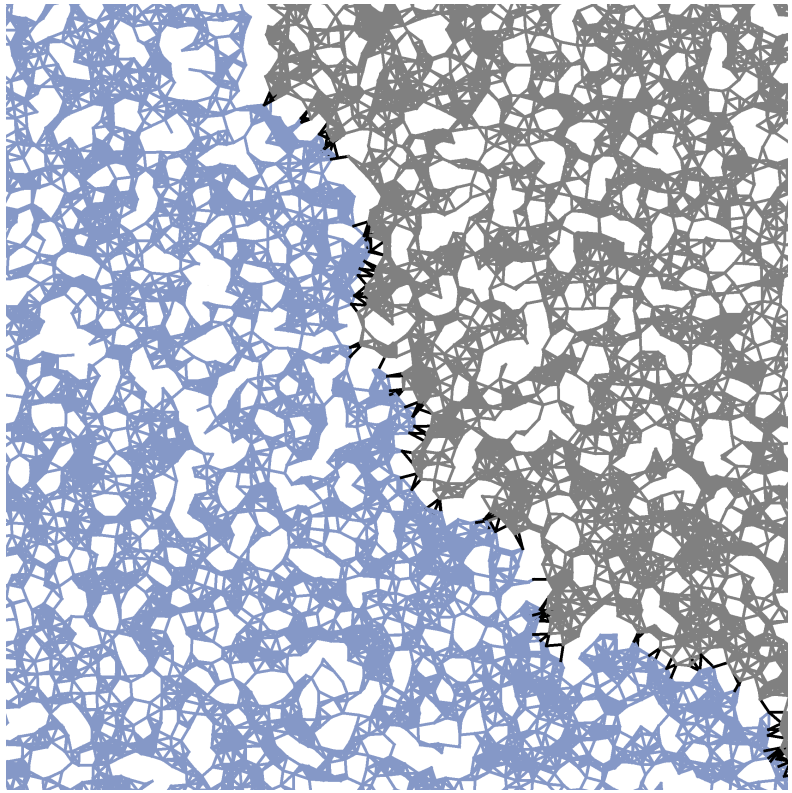
- $>>$ large weight, $<$ small weight
- start with the better partition (red, $\mathcal{P}_2$)
- move v_4 to the opposite block
- integrated into multilevel scheme (+local search on each level)

Example

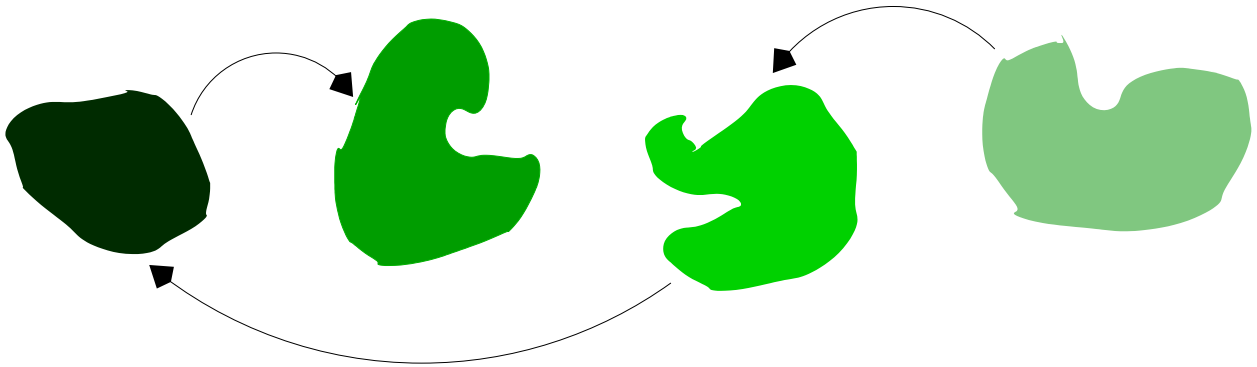
Result of $\mathcal{P}_1$, $\mathcal{P}_2$



$\Rightarrow$



Parallelization



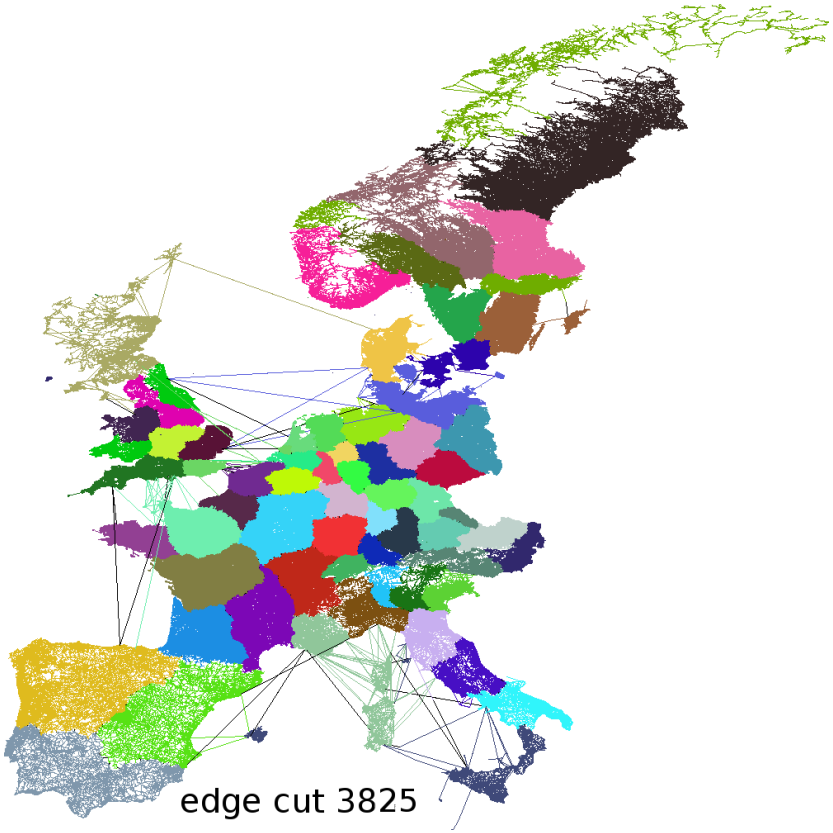
- each PE has its own island (a local population)
- locally: perform combine and mutation operations
- communicate analog to *randomized rumor spreading*
  1. rumor $\leftrightarrow$ currently best local partition
  2. local best partition *changed* $\rightarrow$ send it to $\mathcal{O}(\log P)$ random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

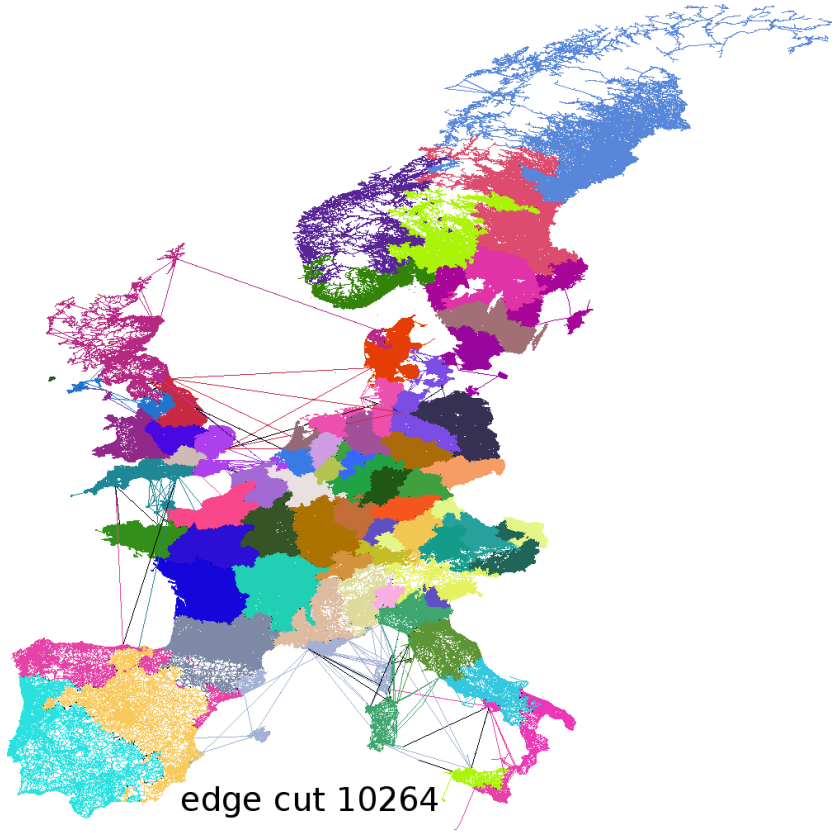
  $\rightarrow$ quality records in a few minutes for small graphs

Experiments

Example

Street network Europe $|V| = 18M$, $|E| = 44M$, $k = 64$
Buffoon $\leftrightarrow$ kMetis





Experimental Results

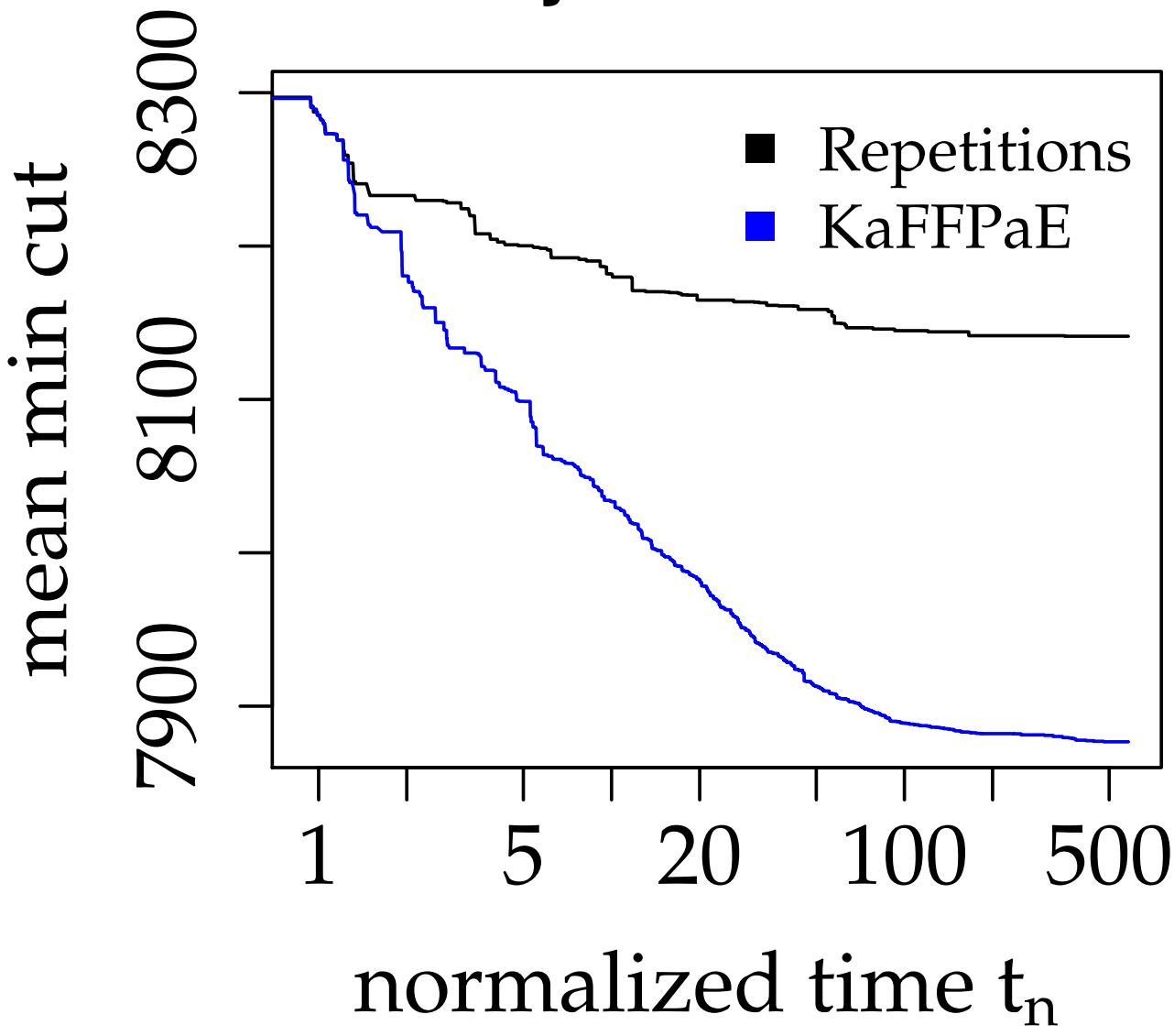
Comparison with Other Systems

Geometric mean, imbalance $\epsilon = 0.03$:
11 graphs (78K–18M nodes) $\times k \in \{2, 4, 8, 16, 32, 64\}$

Algorithm	large graphs		
	Best	Avg.	t[s]
KaFFPa strong	12 053	12 182	121.22
KaSPar strong	12 450	+3%	87.12
KaFFPa eco	12 763	+6%	3.82
Scotch	14 218	+20%	3.55
KaFFa fast	15 124	+24%	0.98
kMetis	15 167	+33%	0.83

- Walshaw instances, road networks, Florida Sparse Matrix Collection, random Delaunay triangulations, random geometric graphs

KaFFPaEvolutionary k=64



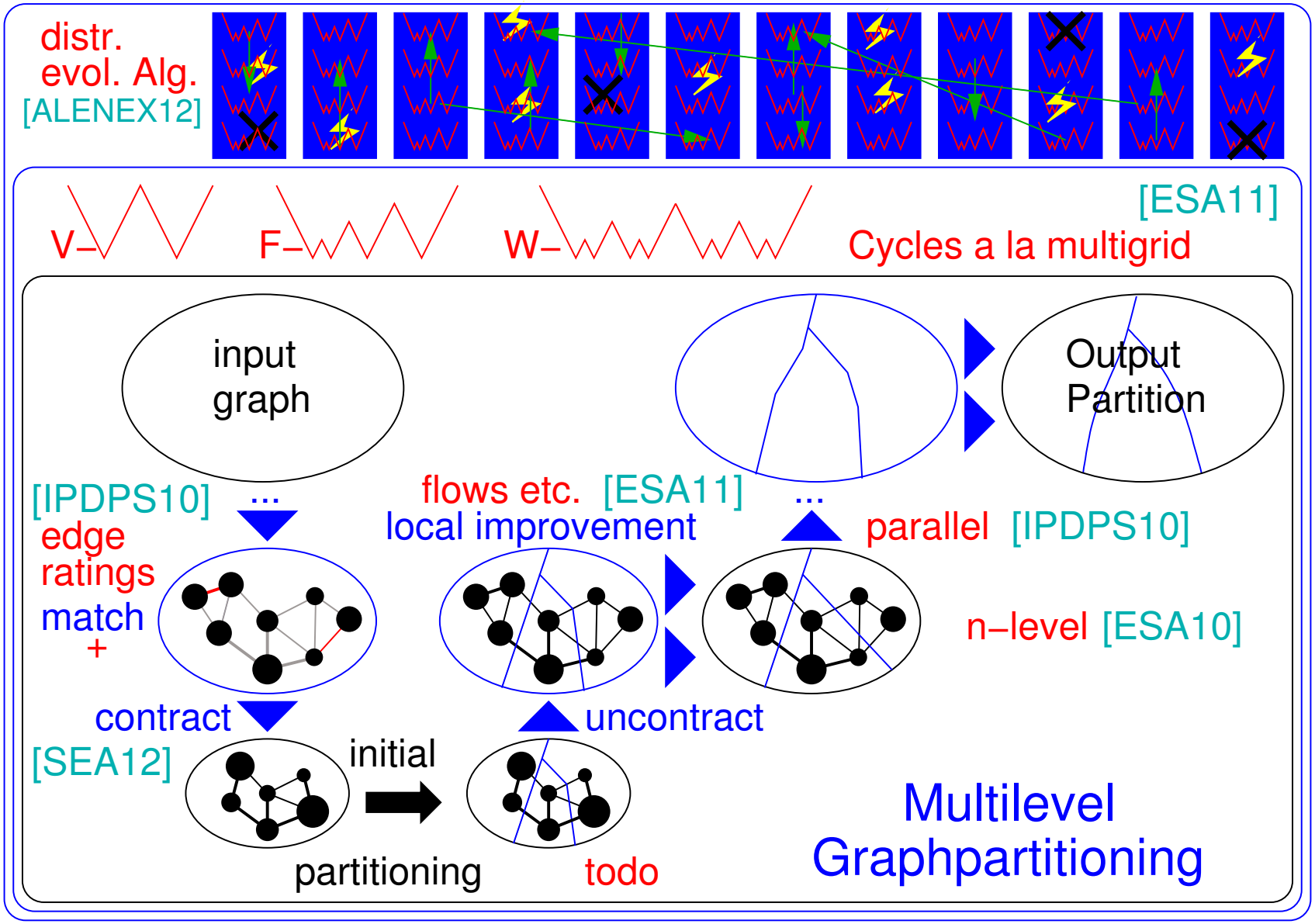
Walshaw Benchmark

- 816 instances ($\epsilon \in \{0, 1\%, 3\%, 5\%\}$)
- focus on partition quality

- overall quality records $\leq$:

ϵ	$\leq$	
0%	78%	**new**
1%	78%	
3%	92%	
5%	94%	

Summary


distr.
evol. Alg.
[ALENEX12]
V–
F–
W–
[ESA11]
Cycles a la multigrid
input
graph
Output
Partition
[IPDPS10]
edge
ratings
match
+
...
flows etc. [ESA11]
local improvement
...
parallel [IPDPS10]
n–level [ESA10]
contract
uncontract
[SEA12]
initial
partitioning
todo
Multilevel
Graphpartitioning

Current and Future Work

- $\epsilon = / \approx 0$
- open source release
- back to parallelization ($+$ external?)
- reconsider n-level? (flows?, . . .)
- other objective functions ((max.) communication volume, separators,. . .)
- hypergraph partitioning
- clustering
- mapping onto processors
- close gap to theory?
- etc.

Karlsruher Institut für Technologie

Thank you!