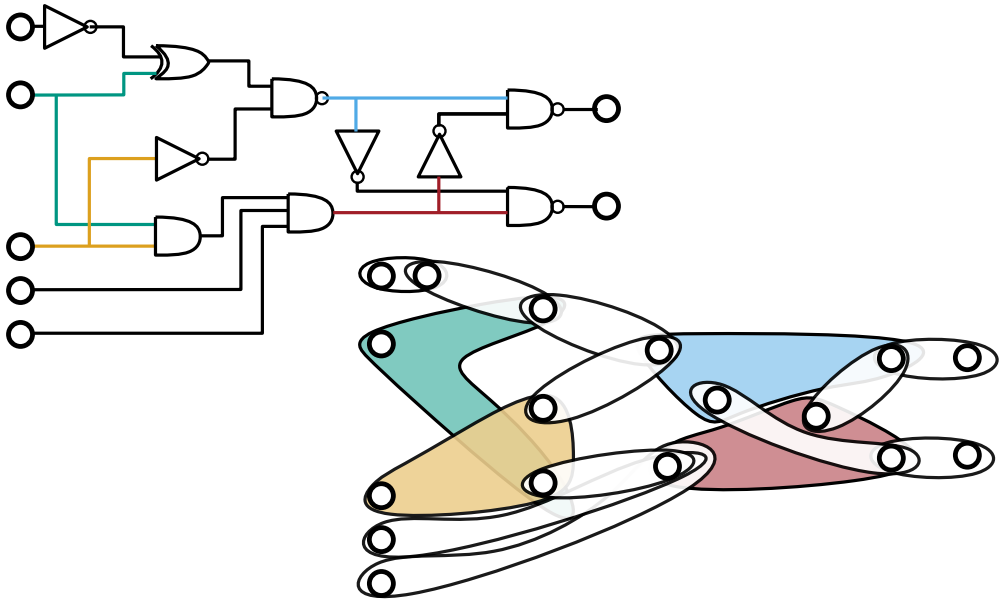
Karlsruhe Institute of Technology

High Quality Hypergraph Partitioning via Max-Flow-Min-Cut Computations

Master Thesis · February 16, 2018

Tobias Heuer

INSTITUTE OF THEORETICAL INFORMATICS · ALGORITHMICS GROUP



KIT – University of the State of Baden-Wuerttemberg and
National Laboratory of the Helmholtz Association

Outline

Task

Developing a **refinement** algorithm based on **Max-Flow-Min-Cut** computations for the *n*-level hypergraph partitioner **KaHyPar**.

Institute of Theoretical Informatics
Algorithmics Group

Outline

Task

Developing a **refinement** algorithm based on **Max-Flow-Min-Cut** computations for the *n*-level hypergraph partitioner **KaHyPar**.

Contributions

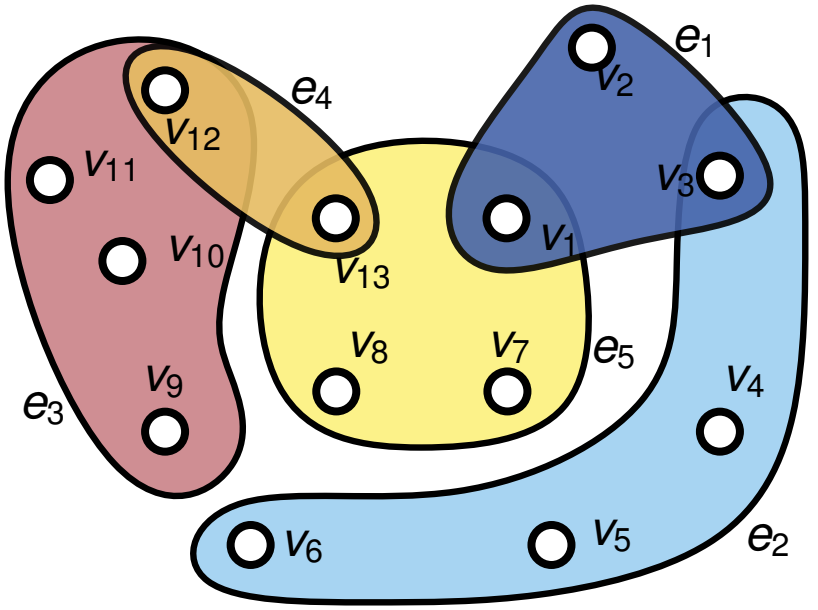
- Outperforms 5 different systems on 73% of 3216 benchmark instances
- Improve quality of *KaHyPar* by 2.5%, while only incurring a slowdon by a factor of 2
- Comparable running time to *hMetis* and outperforms it on 84% of the instances

Hypergraphs
[from SEA'17]

- Generalization of graphs
  $\Rightarrow$ hyperedges connect ≥ 2 nodes

- Graphs $\Rightarrow$ dyadic (**2-ary**) relationships
- Hypergraphs $\Rightarrow$ (**d-ary**) relationships

- Hypergraph $H = (V, E, c, \omega)$
  - Vertex set $V = \{1, ..., n\}$
  - Edge set $E \subseteq \mathcal{P}(V) \setminus \emptyset$
  - Node weights $c : V \to \mathbb{R}_{\geq 1}$
  - Edge weights $\omega : E \to \mathbb{R}_{\geq 1}$



Institute of Theoretical Informatics
Algorithmics Group

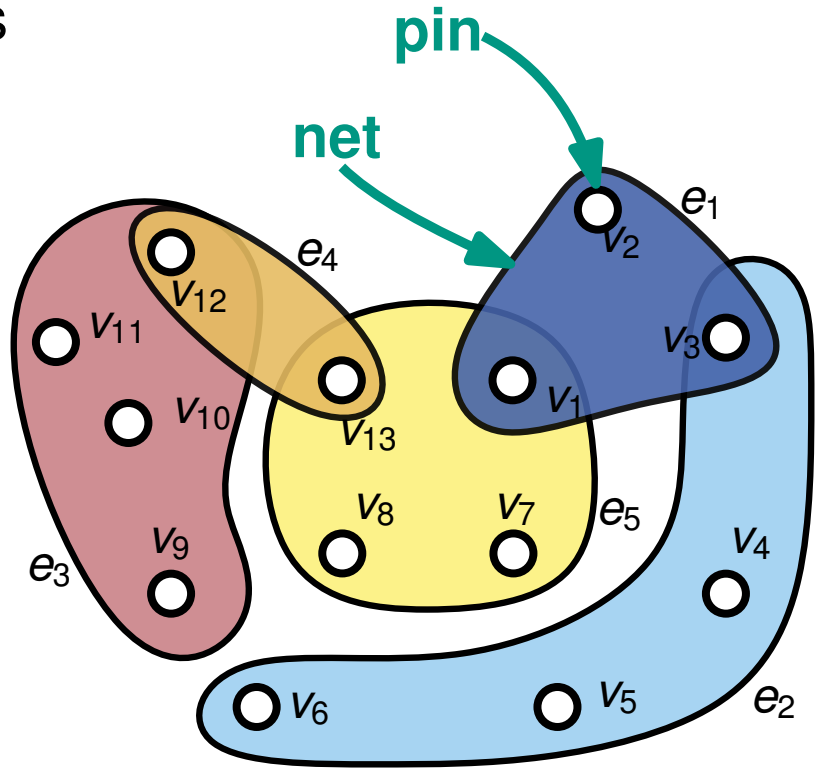
Hypergraphs
[from SEA'17]

- Generalization of graphs
  $\Rightarrow$ hyperedges connect ≥ 2 nodes

- Graphs $\Rightarrow$ dyadic (**2-ary**) relationships
- Hypergraphs $\Rightarrow$ (**d-ary**) relationships

- Hypergraph $H = (V, E, c, \omega)$
  - Vertex set $V = \{1, ..., n\}$
  - Edge set $E \subseteq \mathcal{P}(V) \setminus \emptyset$
  - Node weights $c : V \to \mathbb{R}_{\geq 1}$
  - Edge weights $\omega : E \to \mathbb{R}_{\geq 1}$

- $|P| = \sum_{e \in E} |e| = \sum_{v \in V} d(v)$



Institute of Theoretical Informatics
Algorithmics Group

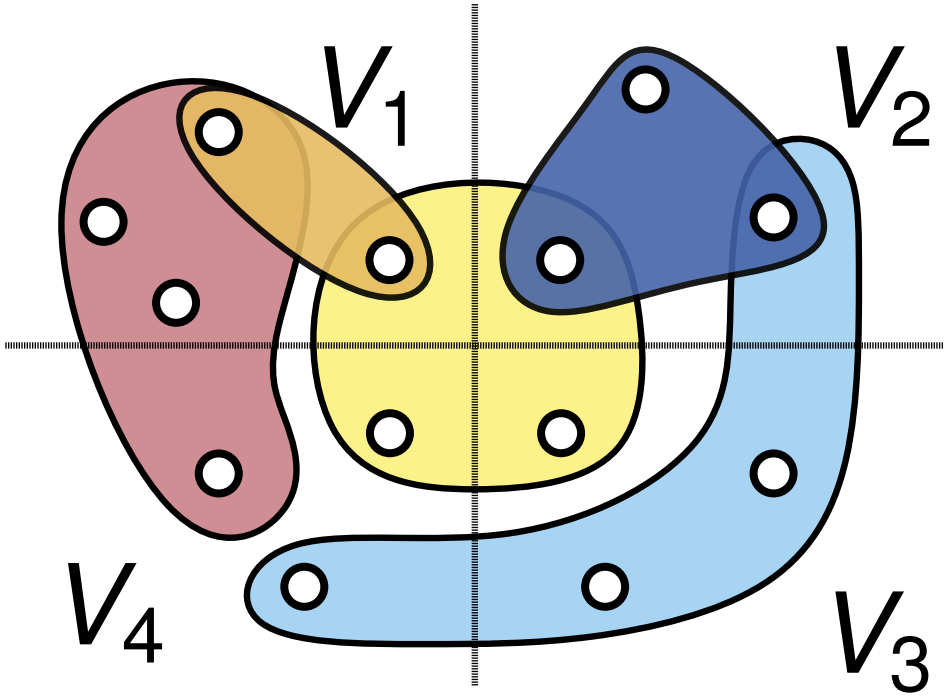
Hypergraph Partitioning Problem

[from SEA'17]

Partition hypergraph $H = (V, E, c, \omega)$ into $\mathbf{k}$ non-empty disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$ such that:

- blocks V_i are **roughly equal-sized**:

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$



Hypergraph Partitioning Problem

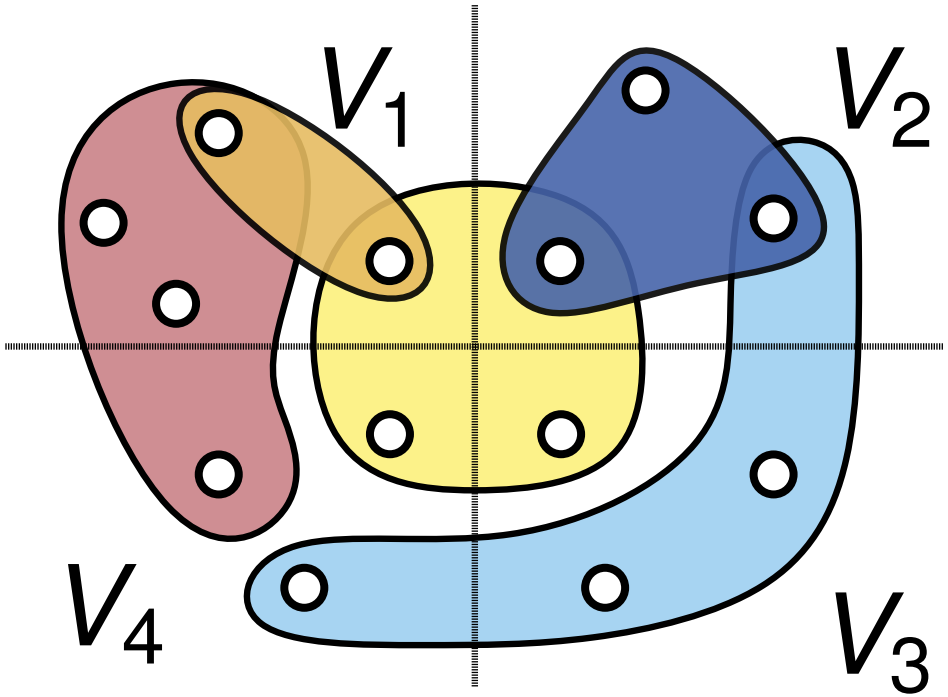
[from SEA'17]

Partition hypergraph $H = (V, E, c, \omega)$ into $\mathbf{k}$ non-empty disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$ such that:

- blocks V_i are **roughly equal-sized**:

imbalance parameter

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$



Institute of Theoretical Informatics
Algorithmics Group

Hypergraph Partitioning Problem

[from SEA'17]

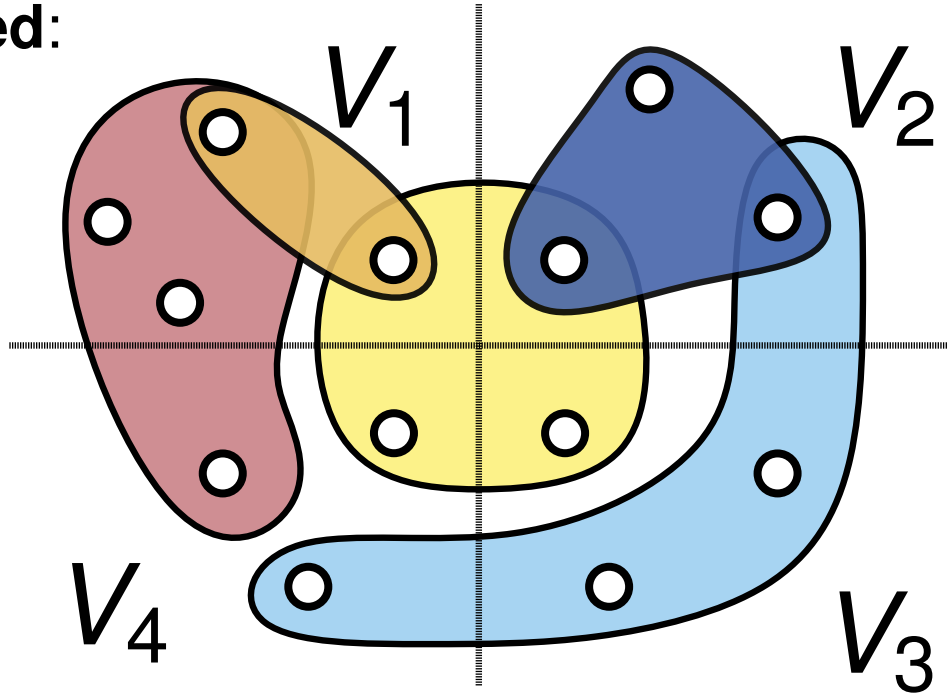
Partition hypergraph $H = (V, E, c, \omega)$ into $\mathbf{k}$ non-empty disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$ such that:

- blocks V_i are **roughly equal-sized**:

imbalance parameter

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

- **connectivity** objective is **minimized**:



Hypergraph Partitioning Problem

[from SEA'17]

Partition hypergraph $H = (V, E, c, \omega)$ into $\mathbf{k}$ non-empty disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$ such that:

- blocks V_i are **roughly equal-sized**:

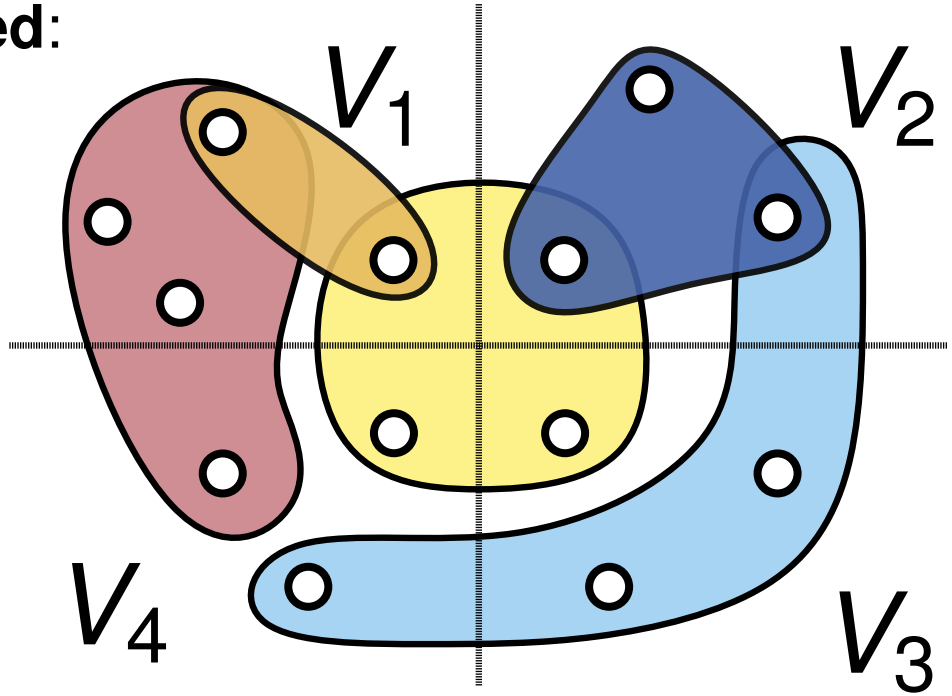
imbalance parameter

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

- **connectivity** objective is **minimized**:

$$\sum_{e \in \text{cut}} (\lambda - 1)\, \omega(e)$$

connectivity:
 # blocks connected by net e



Hypergraph Partitioning Problem

[from SEA'17]

Partition hypergraph $H = (V, E, c, \omega)$ into $\mathbf{k}$ non-empty disjoint blocks $\Pi = \{V_1, \ldots, V_k\}$ such that:

- blocks V_i are **roughly equal-sized**:

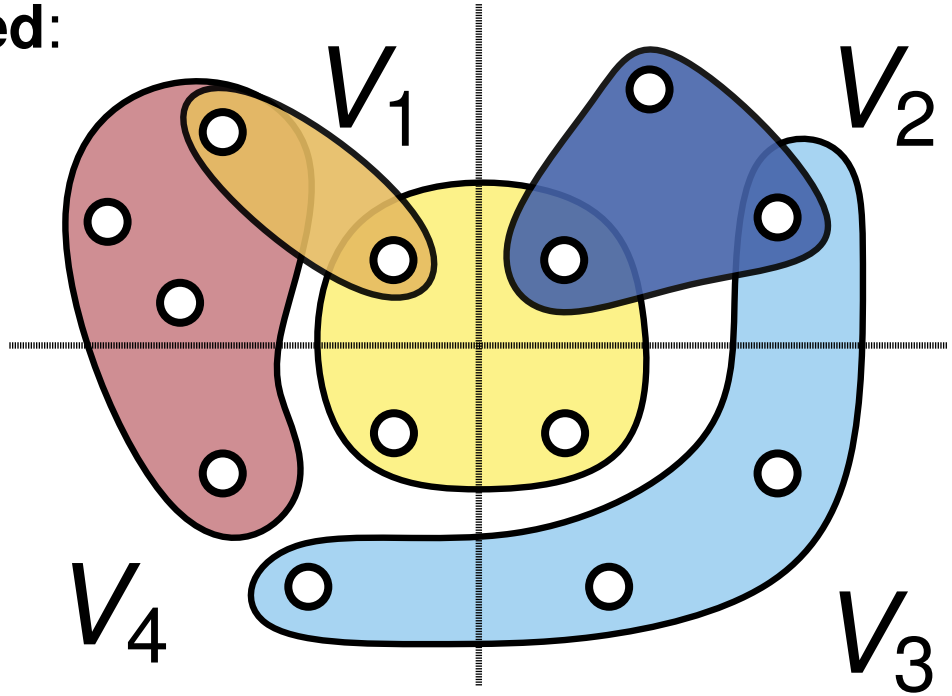
imbalance parameter

$$c(V_i) \leq (1 + \varepsilon) \left\lceil \frac{c(V)}{k} \right\rceil$$

- **connectivity** objective is **minimized**:

$$\sum_{e \in \text{cut}} (\lambda - 1)\, \omega(e) = 6$$

connectivity:
 # blocks connected by net e

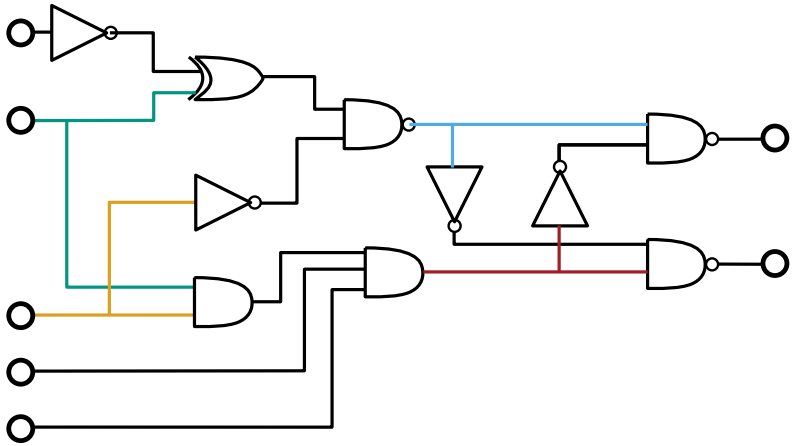


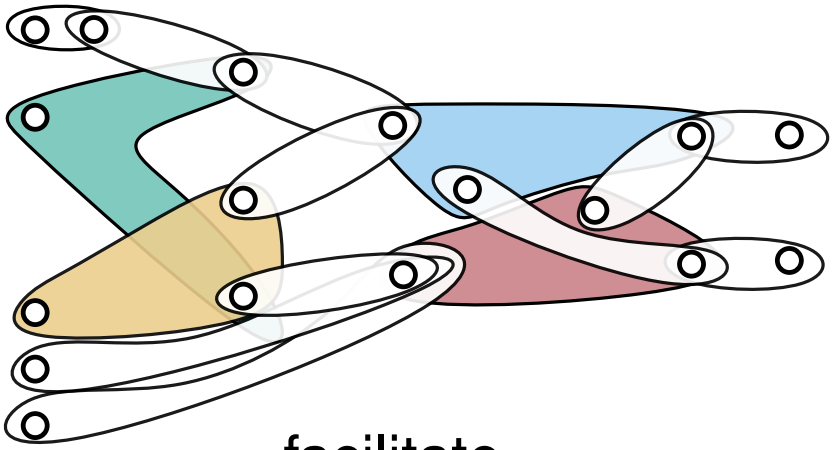
Applications

[from SEA'17]

Karlsruhe Institute of Technology

VLSI Design





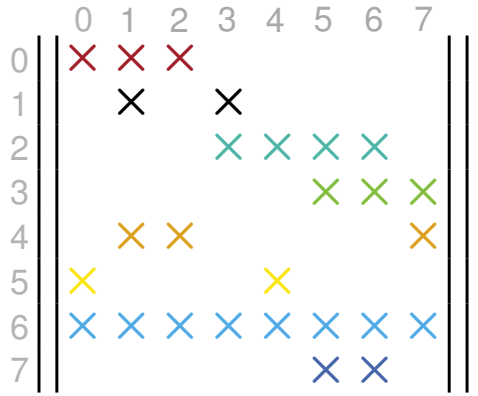
facilitate
floorplanning & placement

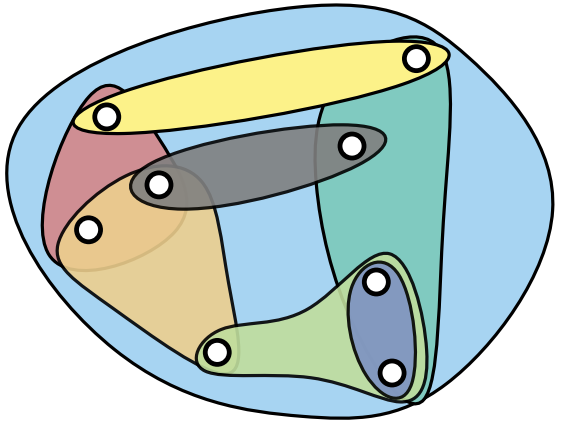
Application Domain

Hypergraph Model

Goal

Scientific Computing



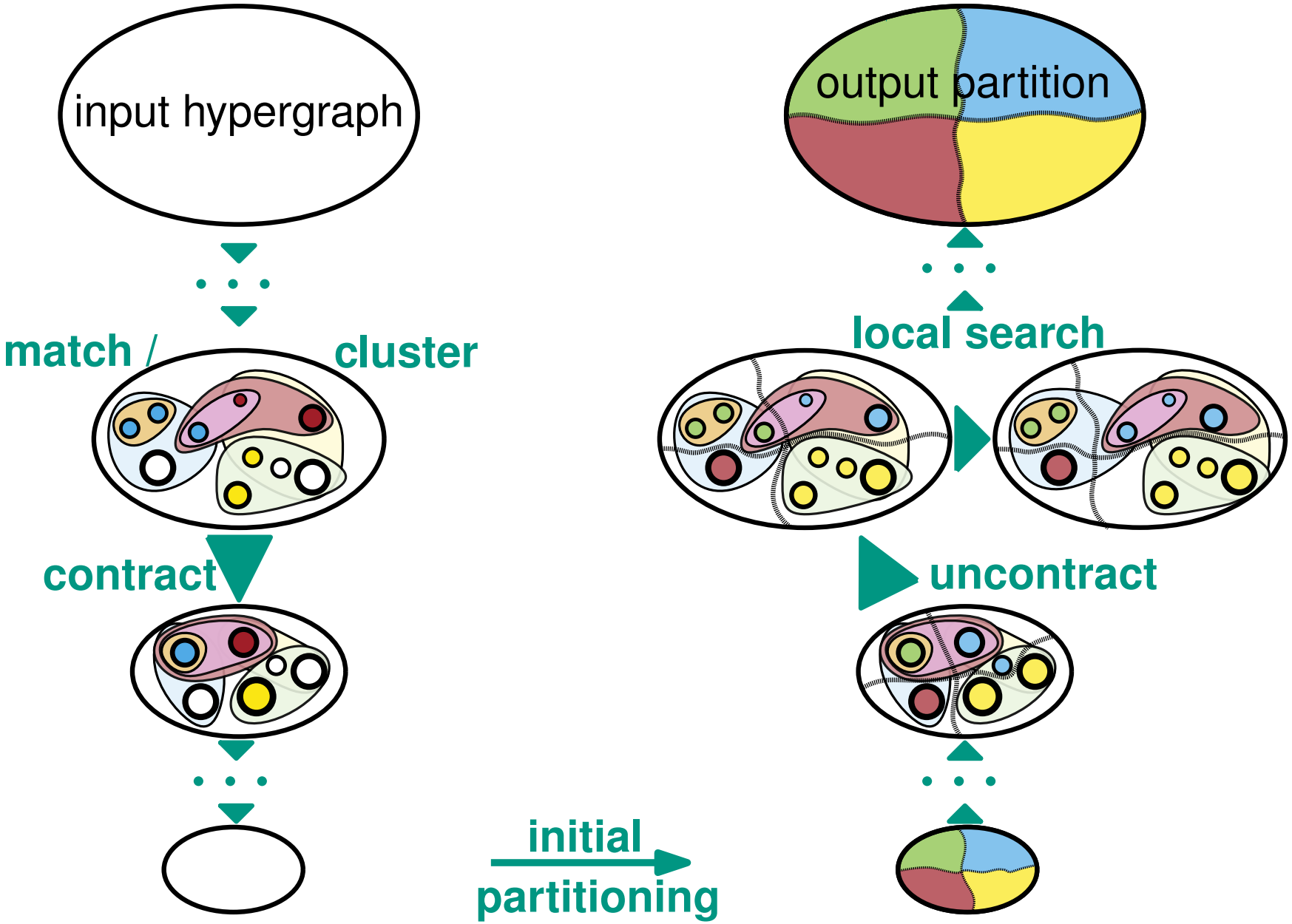


minimize
communication

Institute of Theoretical Informatics
Algorithmics Group

The Multilevel Framework

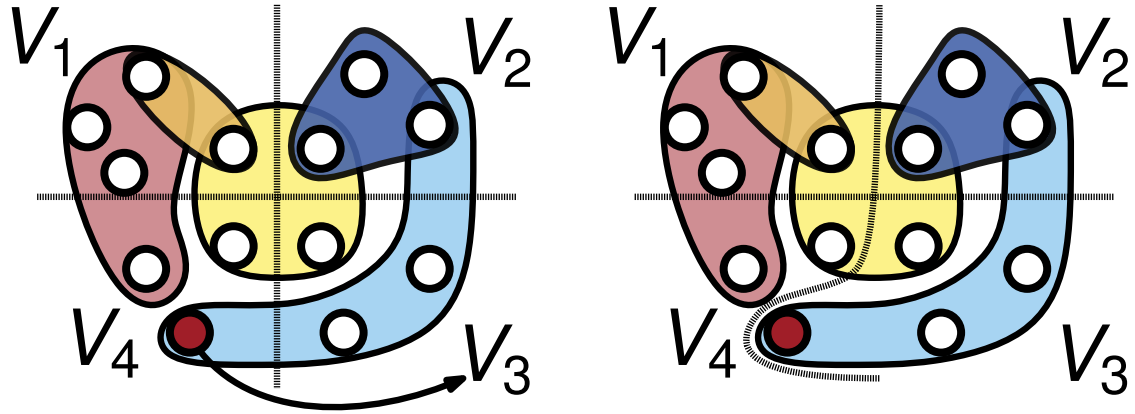
[from SEA'17]



Karlsruhe Institute of Technology
Institute of Theoretical Informatics
Algorithmics Group

FM Algorithm

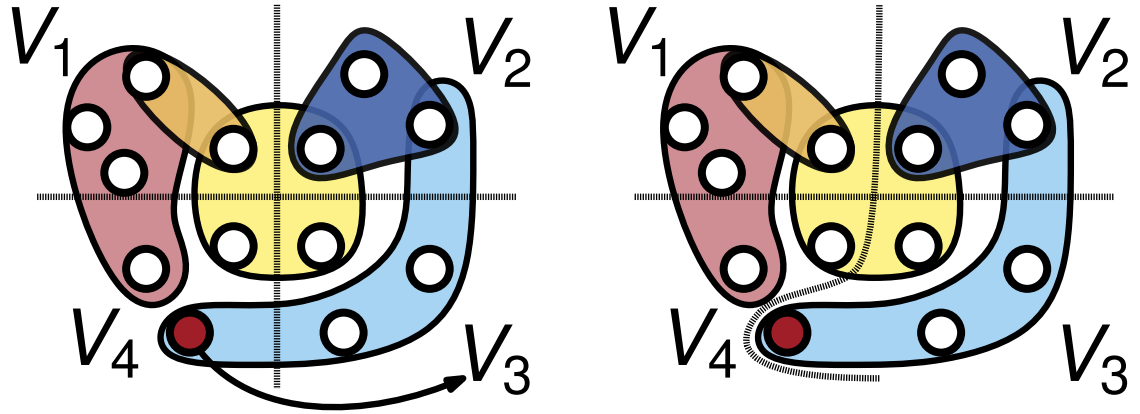
- **Move**-based heuristic that **greedily** move vertices between blocks based on **local** informations of incident nets



Moving ● from V_4 to V_3 reduces cut by 1

FM Algorithm

- **Move**-based heuristic that **greedily** move vertices between blocks based on **local** informations of incident nets

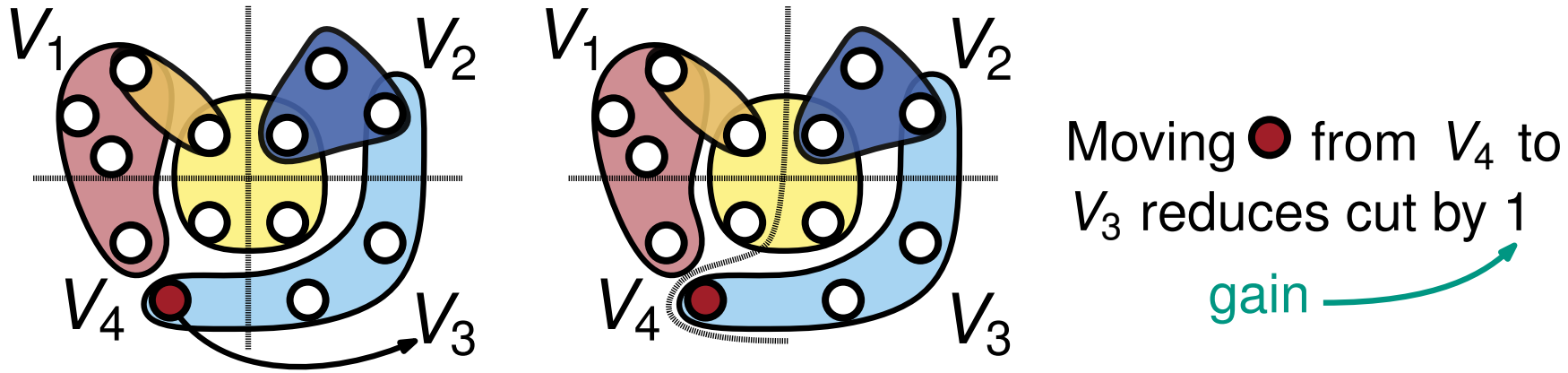


Moving ● from V_4 to V_3 reduces cut by 1

gain

FM Algorithm

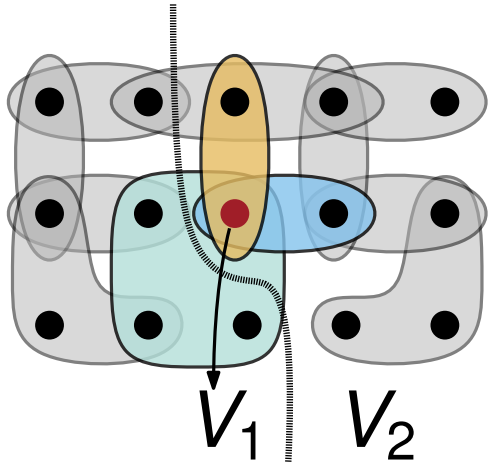
- **Move**-based heuristic that **greedily** move vertices between blocks based on **local** informations of incident nets



- Performs moves of vertices with **maximum gain** in each step
- All modern hypergraph partitioners implements variations of the *FM* algorithm

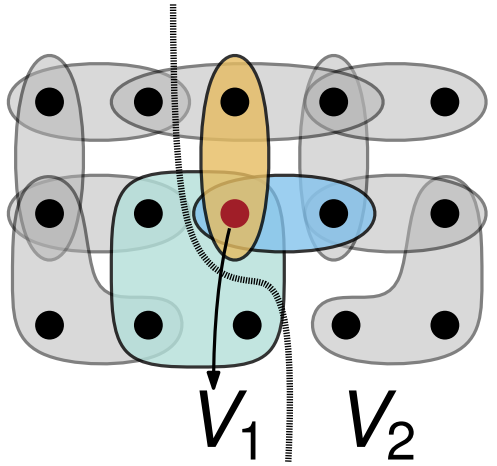
FM Algorithm - Disadvantages

- Only incorparates **local** informations about the problem structure
  - Heavily depends on *initial partition*
  - In multilevel context: Depends on quality of *coarsening*

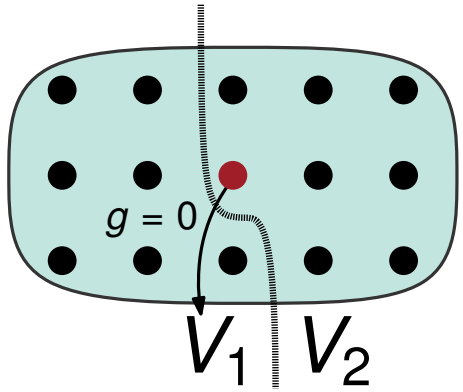


FM Algorithm - Disadvantages

- Only incorparates **local** informations about the problem structure
  - Heavily depends on *initial partition*
  - In multilevel context: Depends on quality of *coarsening*



- Large hyperedges induce **Zero-Gain** moves
  - Quality mainly depends on random decisions made within the algorithm



Flow-based Approaches

Institute of Theoretical Informatics
Algorithmics Group

Flows

Given a graph $G = (V, E, u)$ and two nodes $s, t \in V$

- $u : E \to \mathbb{R}_+$ is the **capacity** function
- s and t are called **source** and **sink**

Institute of Theoretical Informatics
Algorithmics Group

Flows

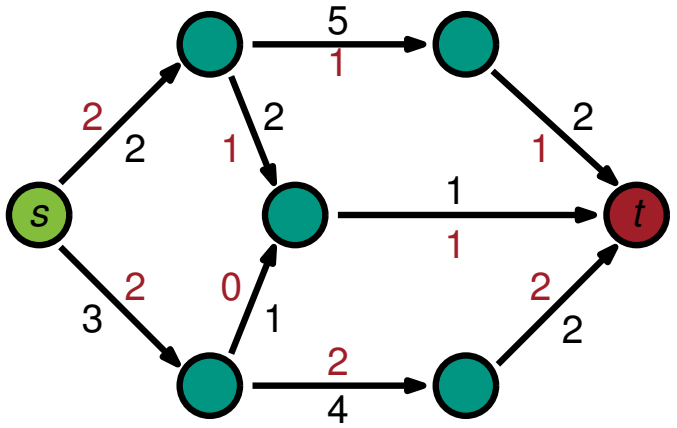
Given a graph $G = (V, E, u)$ and two nodes $s, t \in V$

- $u : E \to \mathbb{R}_+$ is the **capacity** function
- s and t are called **source** and **sink**

A valid **flow** is a function $f : E \to \mathbb{R}_+$ with the constraints:

- $\forall (v, w) \in E : f(v, w) \leq u(v, w)$
- $\forall v \in V \setminus \{s, t\} : \sum_{(w,v)\in E} f(w, v) = \sum_{(v,w)\in E} f(v, w)$

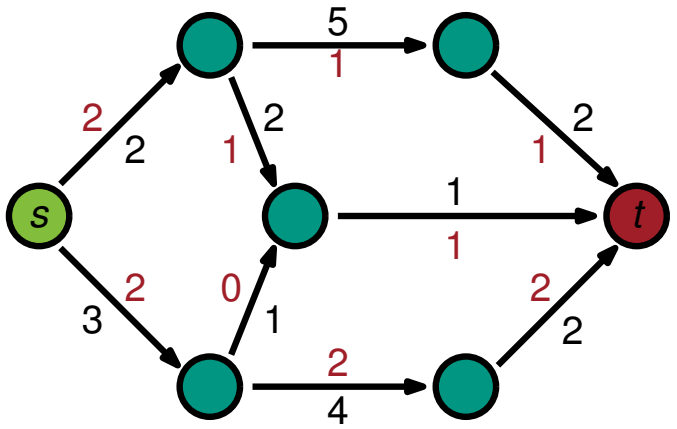
The value of the flow is $|f| = \sum_{(s,v)\in E} f(s, v)$



Flows

The **residual capacity** $r_f : V \times V \to \mathbb{R}_+$ is defined as follows:

- $\forall (v, w) \in E : r_f(v, w) = u(v, w) - f(v, w)$
- $\forall (v, w) \in E$: If $f(v, w) > 0$ and $u(w, v) = 0$, then $r_f(w, v) = f(v, w)$

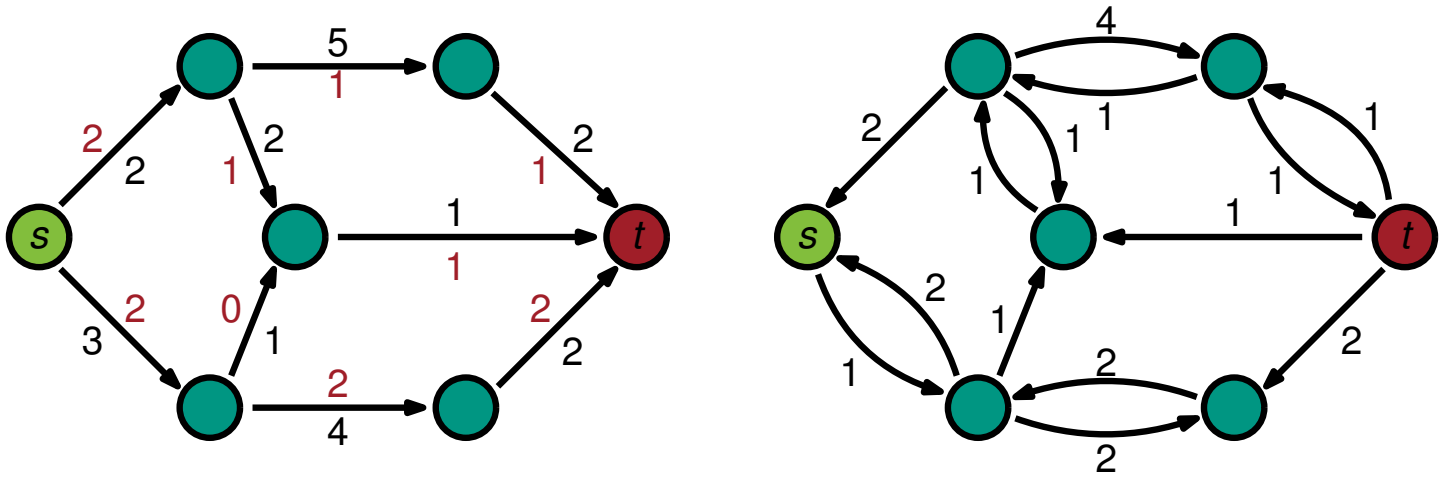


Flows

The **residual capacity** $r_f : V \times V \to \mathbb{R}_+$ is defined as follows:

- $\forall (v, w) \in E : r_f(v, w) = u(v, w) - f(v, w)$
- $\forall (v, w) \in E$: If $f(v, w) > 0$ and $u(w, v) = 0$, then $r_f(w, v) = f(v, w)$

The **residual graph** $G_f = (V, E_f, r_f)$ contains all edges $(v, w) \in V \times V$ with $r_f(v, w) > 0$



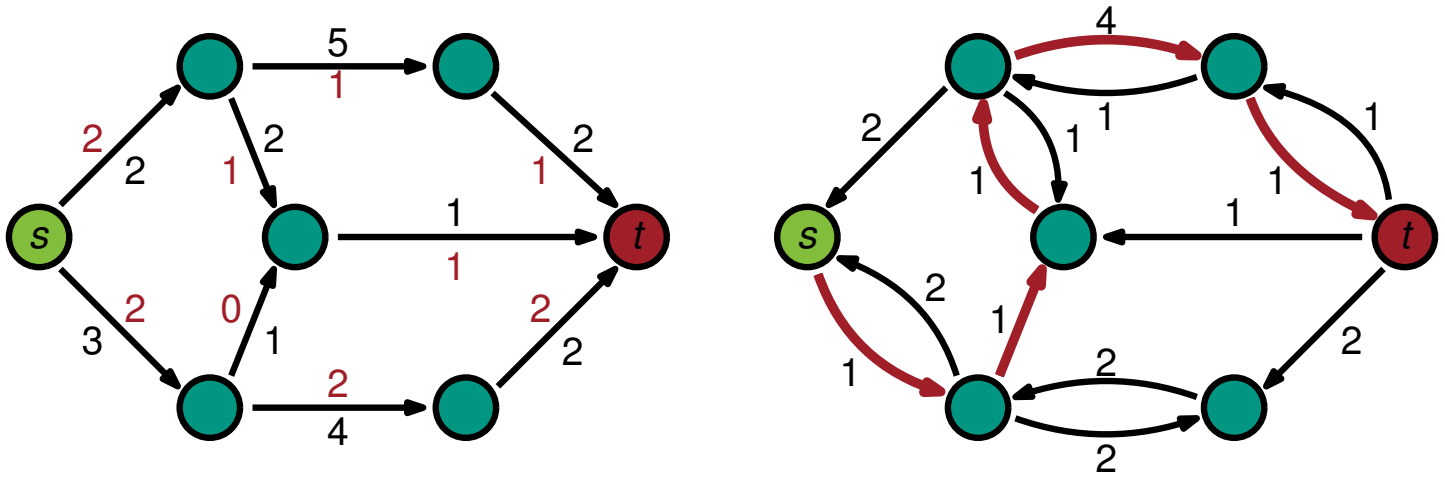
Flows

The **residual capacity** $r_f : V \times V \to \mathbb{R}_+$ is defined as follows:

- $\forall (v, w) \in E : r_f(v, w) = u(v, w) - f(v, w)$
- $\forall (v, w) \in E$: If $f(v, w) > 0$ and $u(w, v) = 0$, then $r_f(w, v) = f(v, w)$

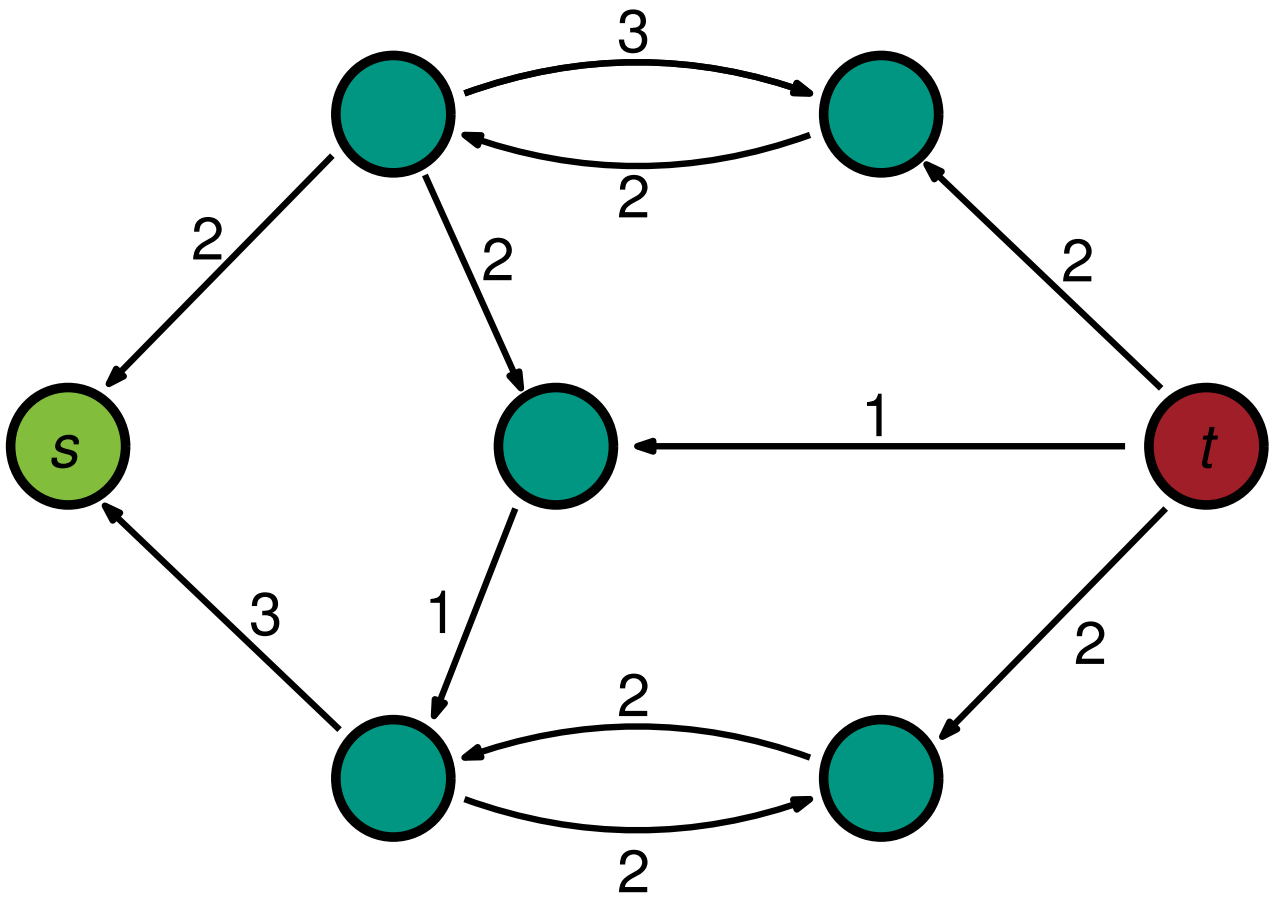
The **residual graph** $G_f = (V, E_f, r_f)$ contains all edges $(v, w) \in V \times V$ with $r_f(v, w) > 0$

- An **augmenting path** is a path in G_f from s to t
- f is a **maximum flow**, if there is no augmenting path from s to t in G_f



Minimum (s, t)-Bipartition

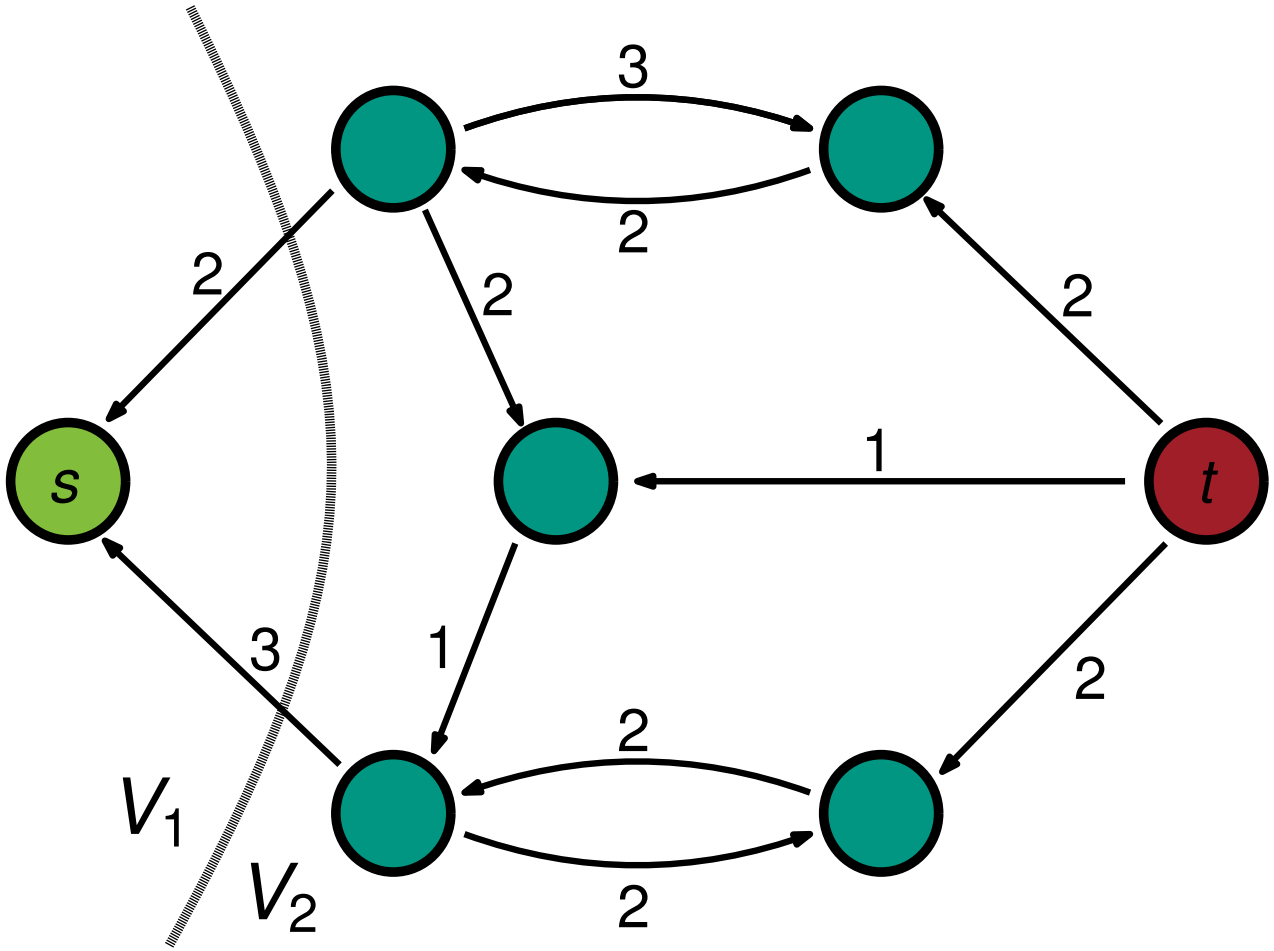
All nodes *reachable* from s are part of V_1 and $V_2 = V \setminus V_1$



Residual Graph G_f of a maximum flow f

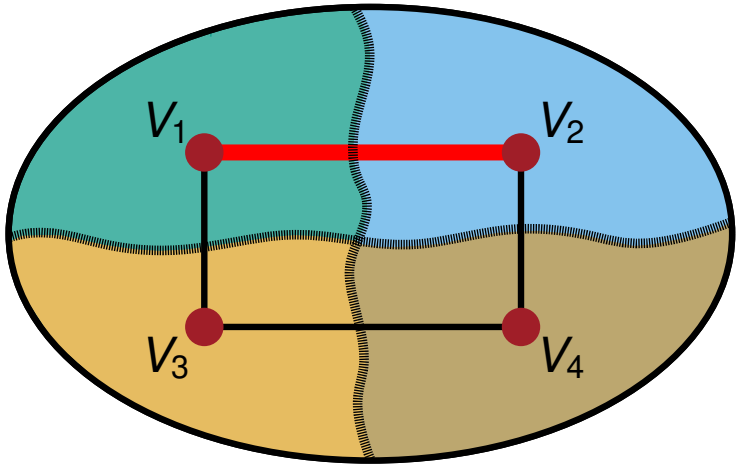
Minimum (s, t)-Bipartition

All nodes *reachable* from s are part of V_1 and $V_2 = V \setminus V_1$

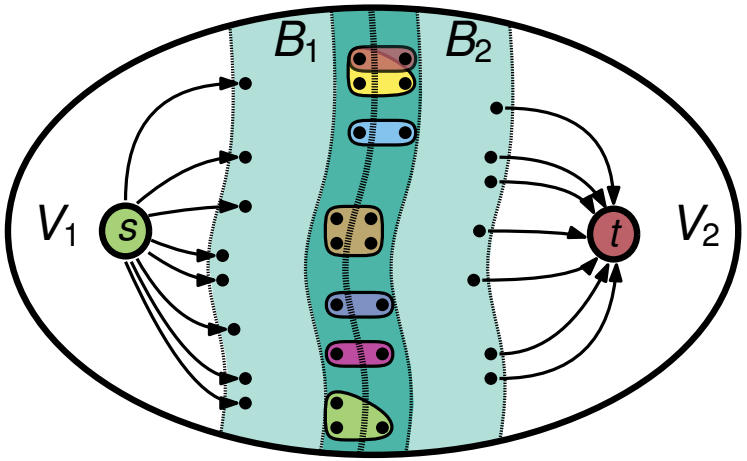


Residual Graph G_f of a maximum flow f

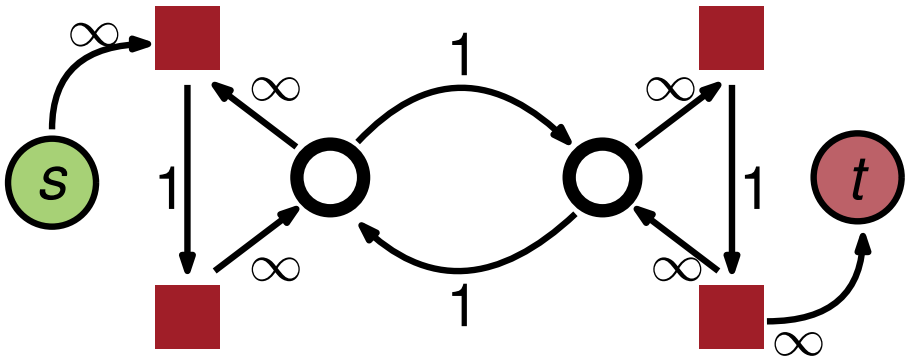
Our Flow-Based Refinement Framework



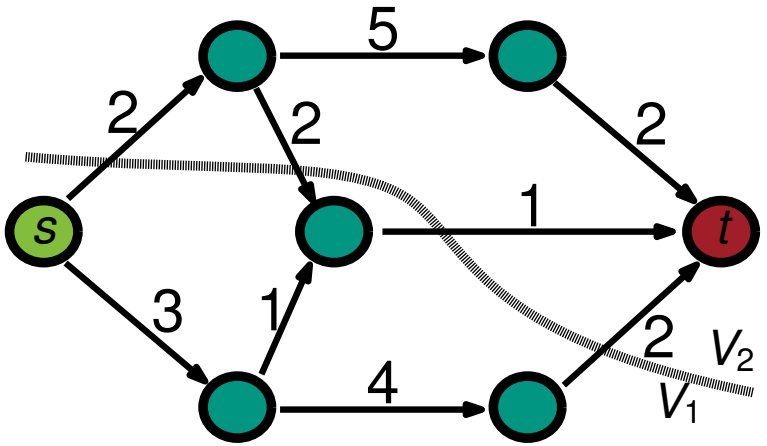
Select two adjacent blocks for refinement



Build Flow Problem



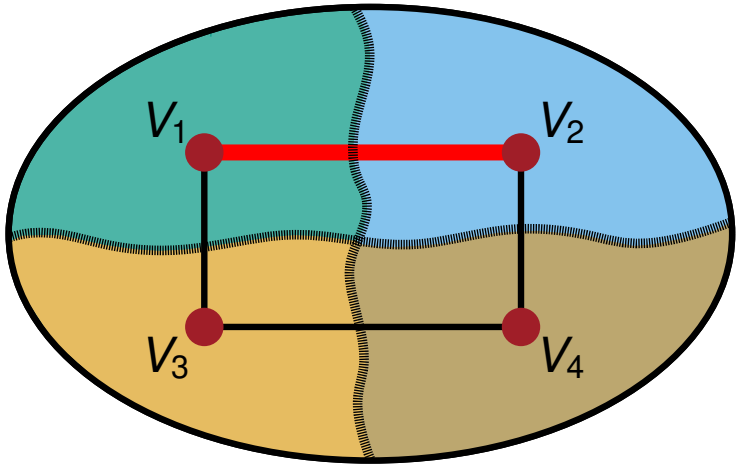
Solve Flow Problem



Find feasible minimum cut

Institute of Theoretical Informatics
Algorithmics Group

Our Flow-Based Refinement Framework



Select two adjacent blocks for refinement

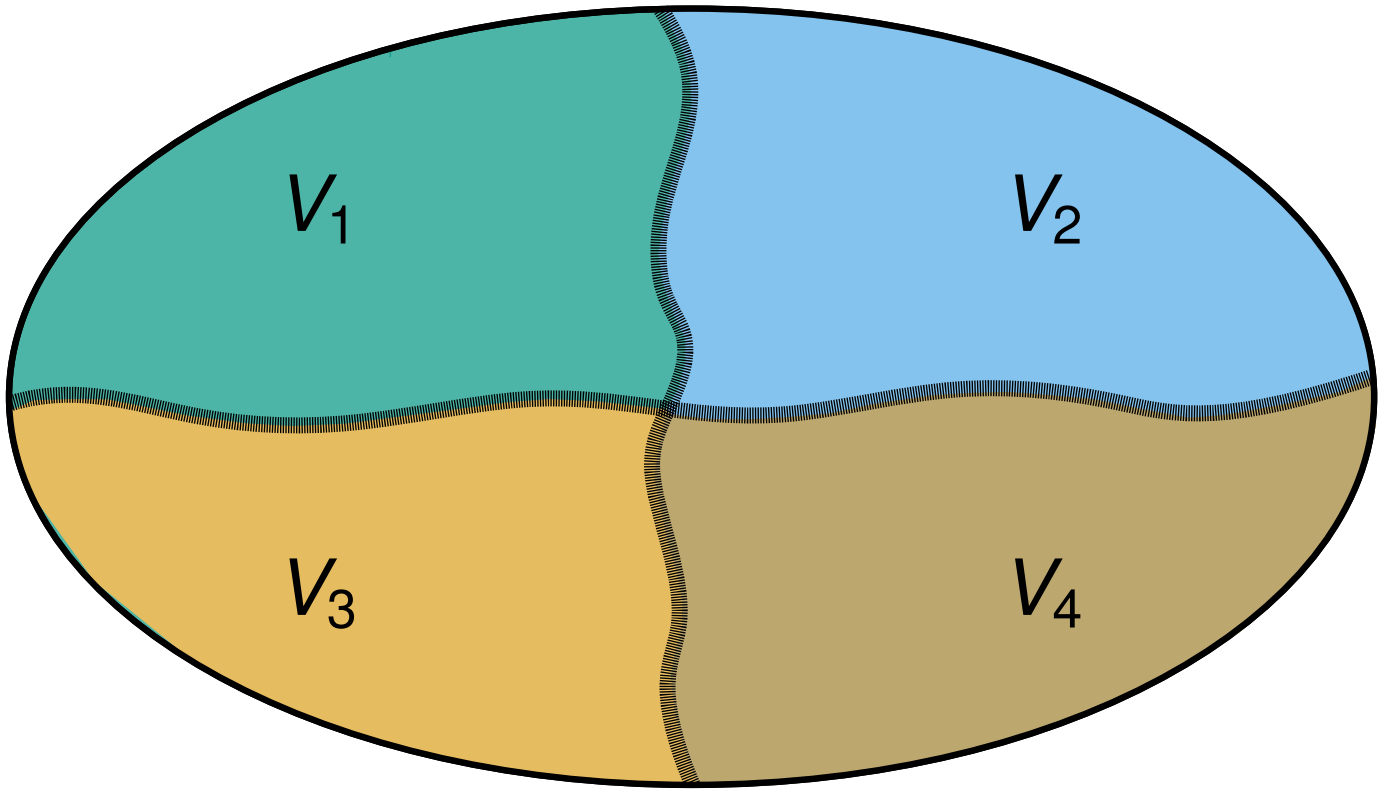
Build Flow Problem

Solve Flow Problem

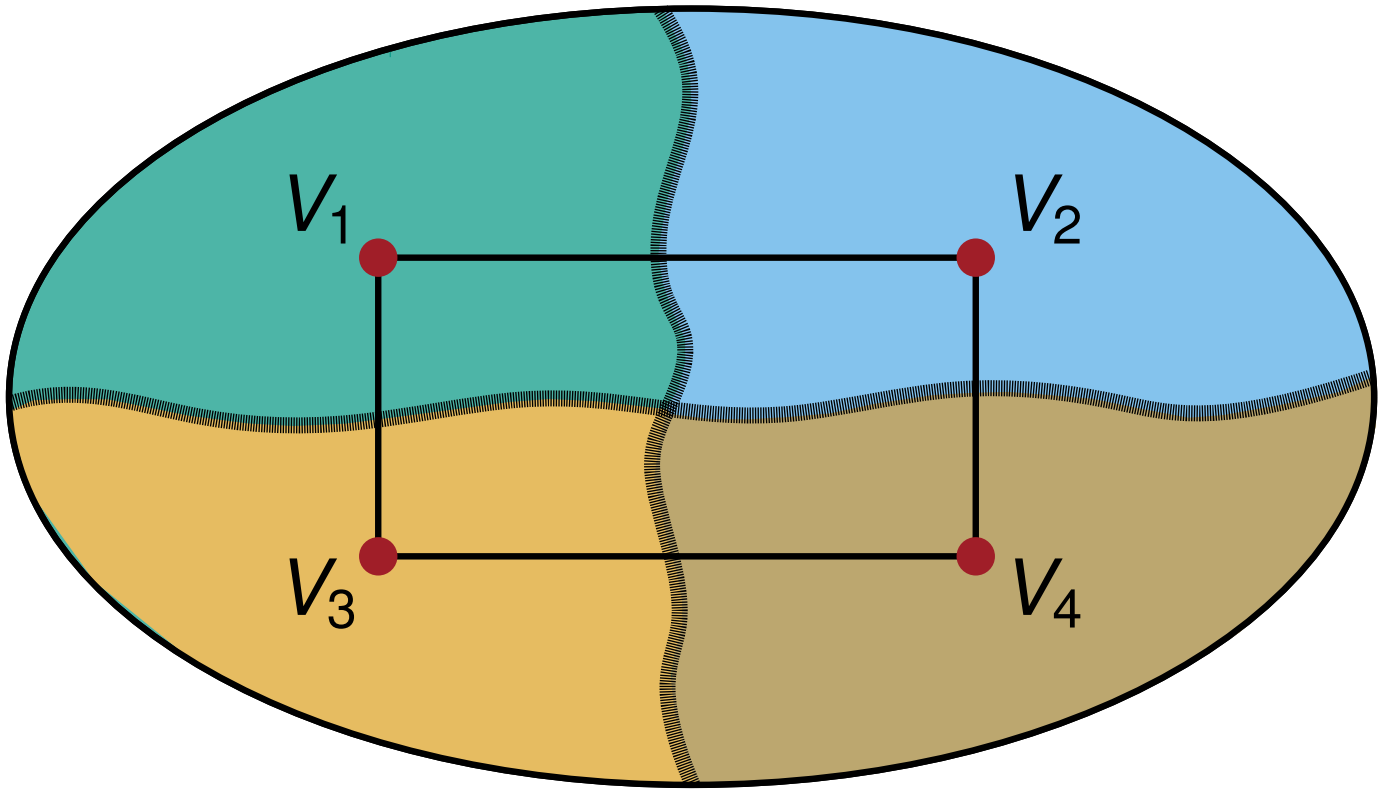
Find feasible minimum cut

Institute of Theoretical Informatics
Algorithmics Group

Active Block Scheduling



Active Block Scheduling

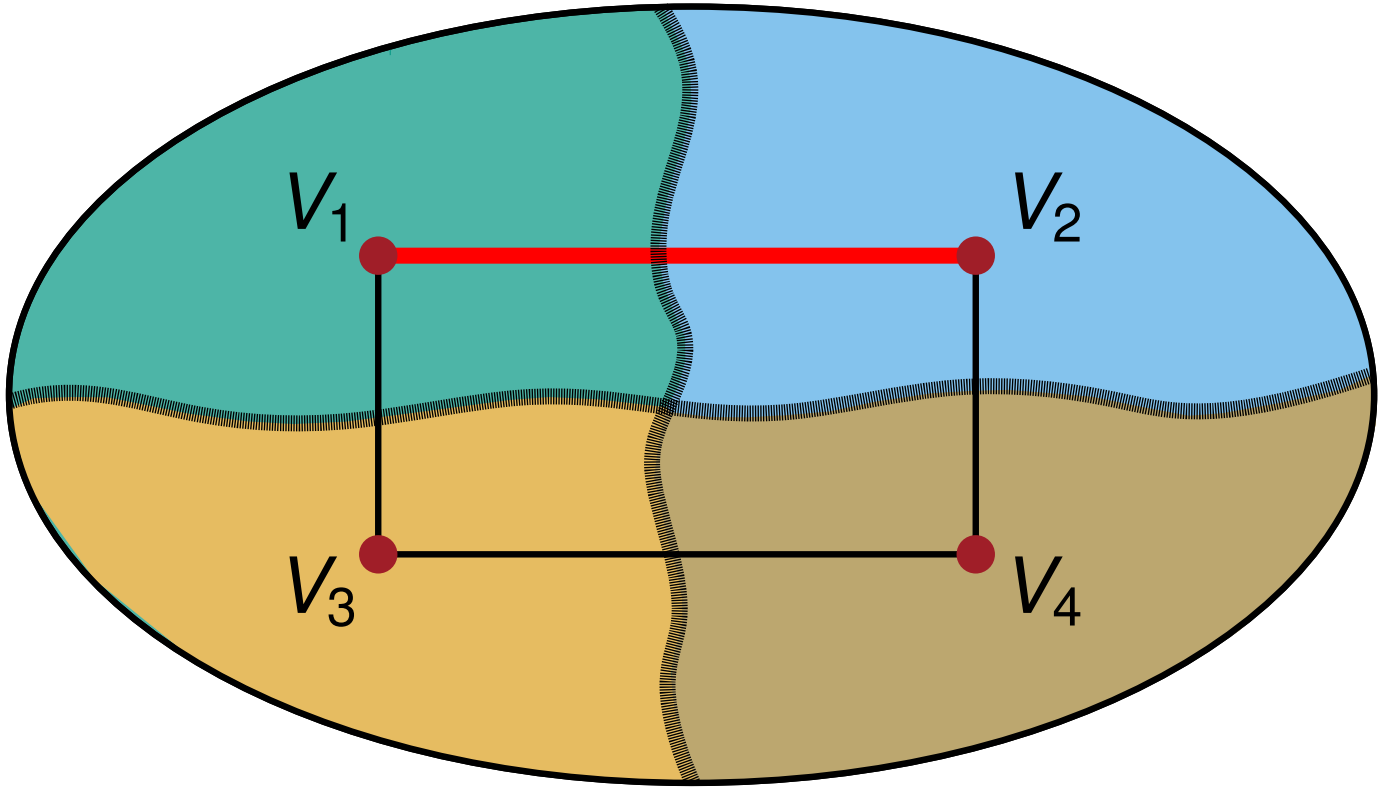


Build Quotient Graph

Active Block Scheduling

Round 1

refine(V_1, V_2) = **Improvement!**

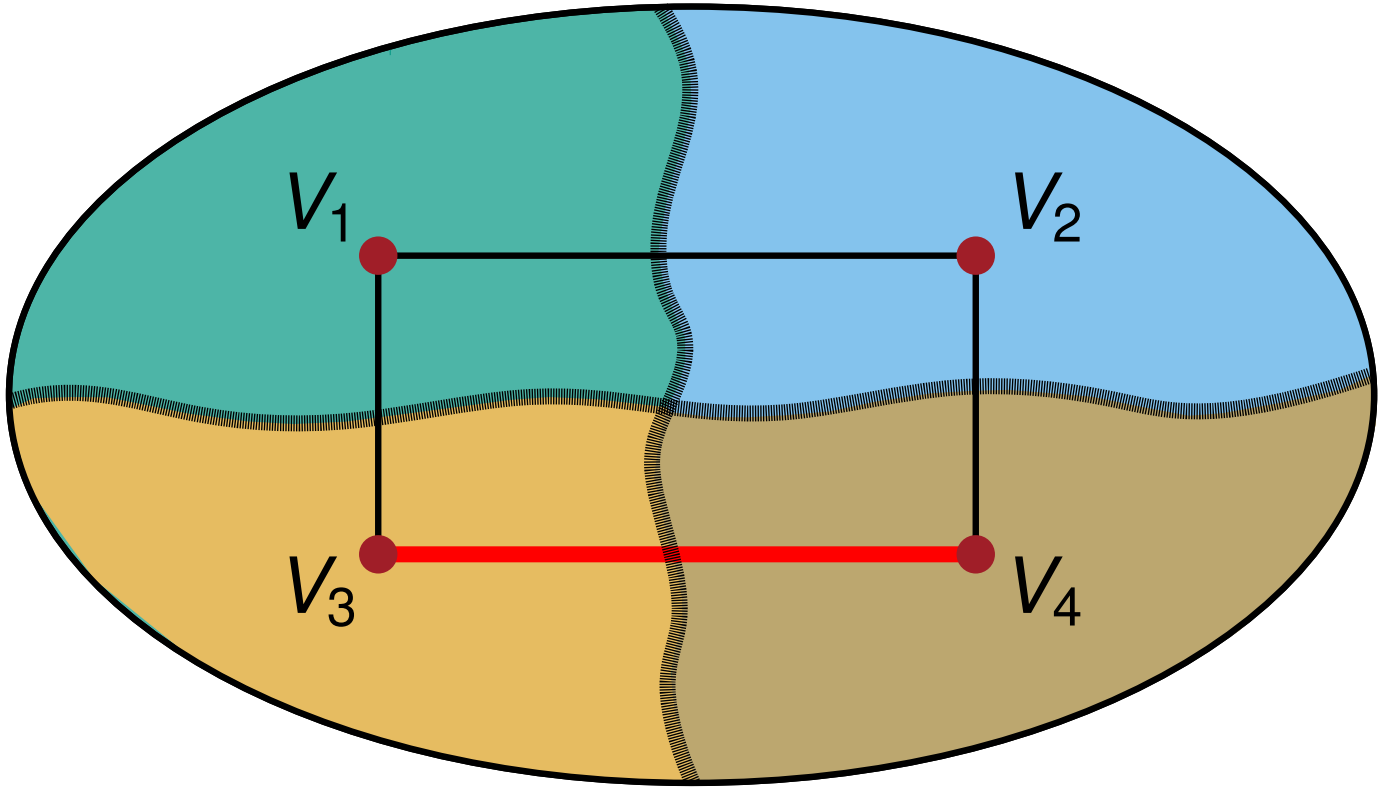


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 1

refine(V_3, V_4) = **No Improvement!**

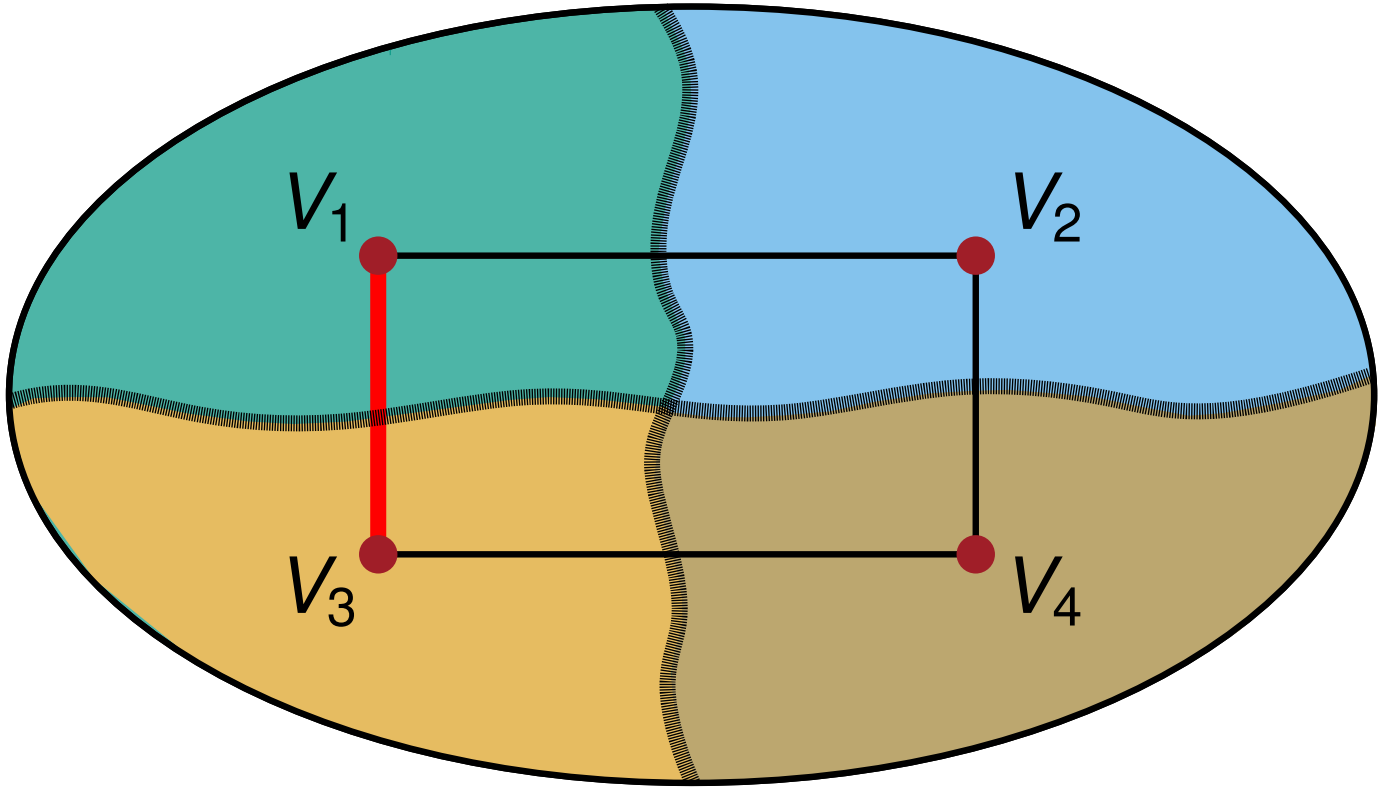


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 1

refine(V_1, V_3) = **No Improvement!**

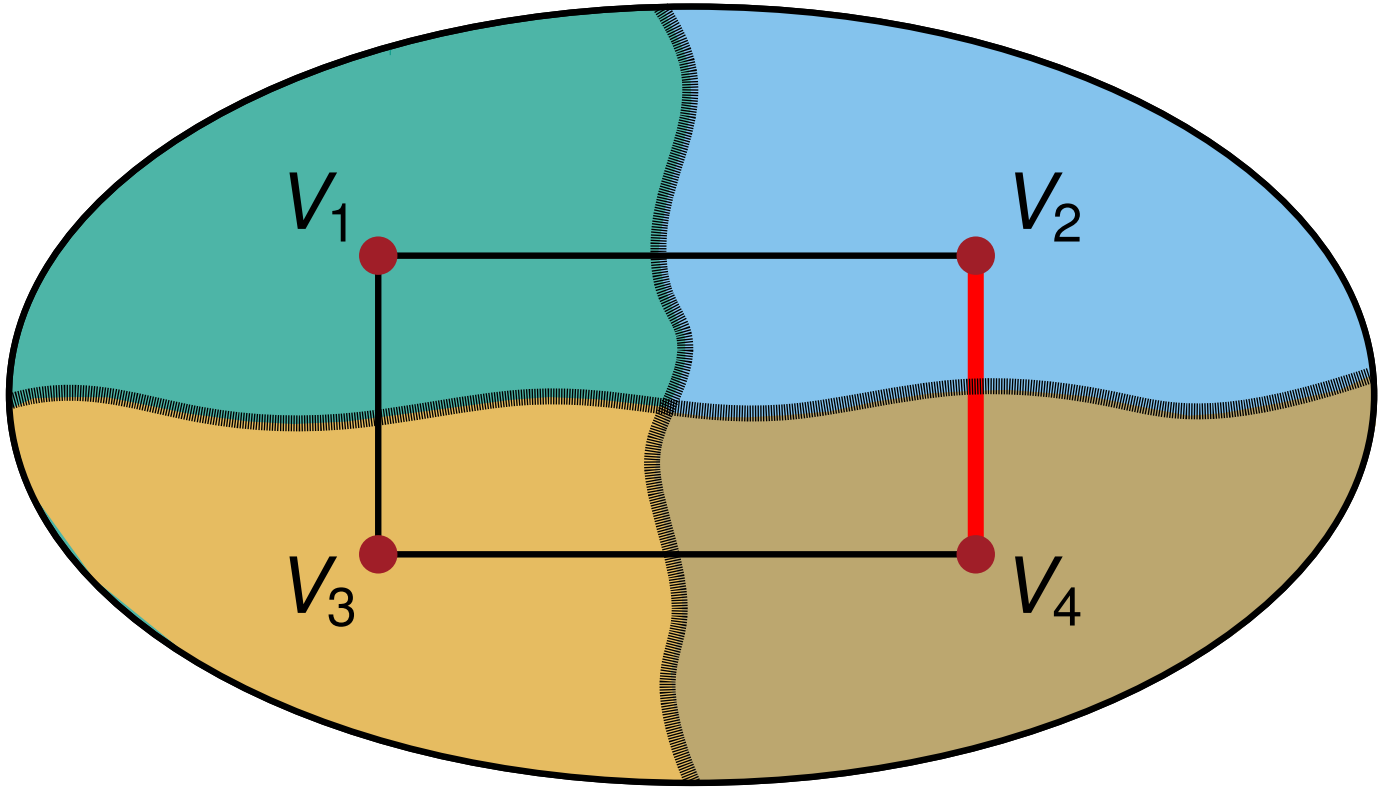


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 1

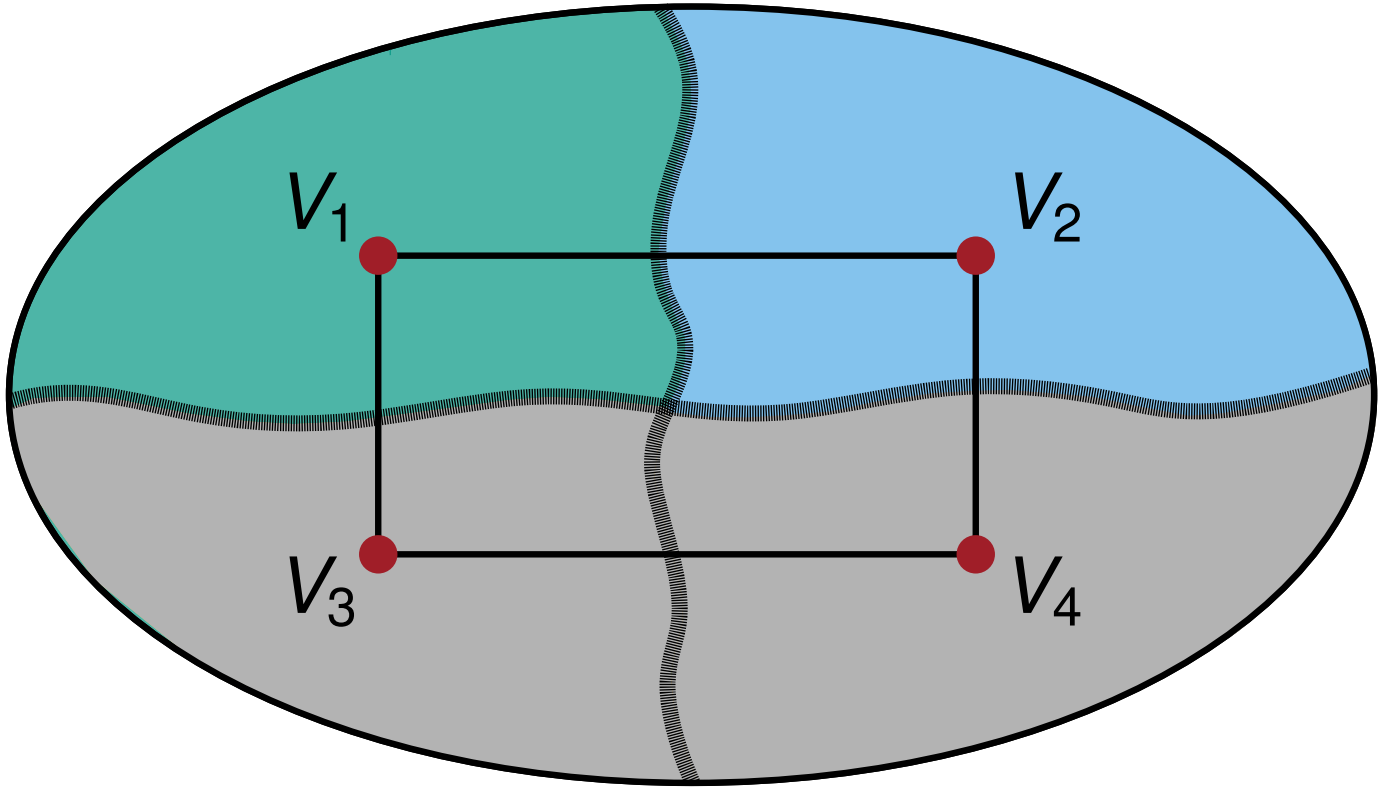
refine(V_2, V_4) = **No Improvement!**



Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 1 Boundary did not change $\Rightarrow$ Mark block as **inactive**

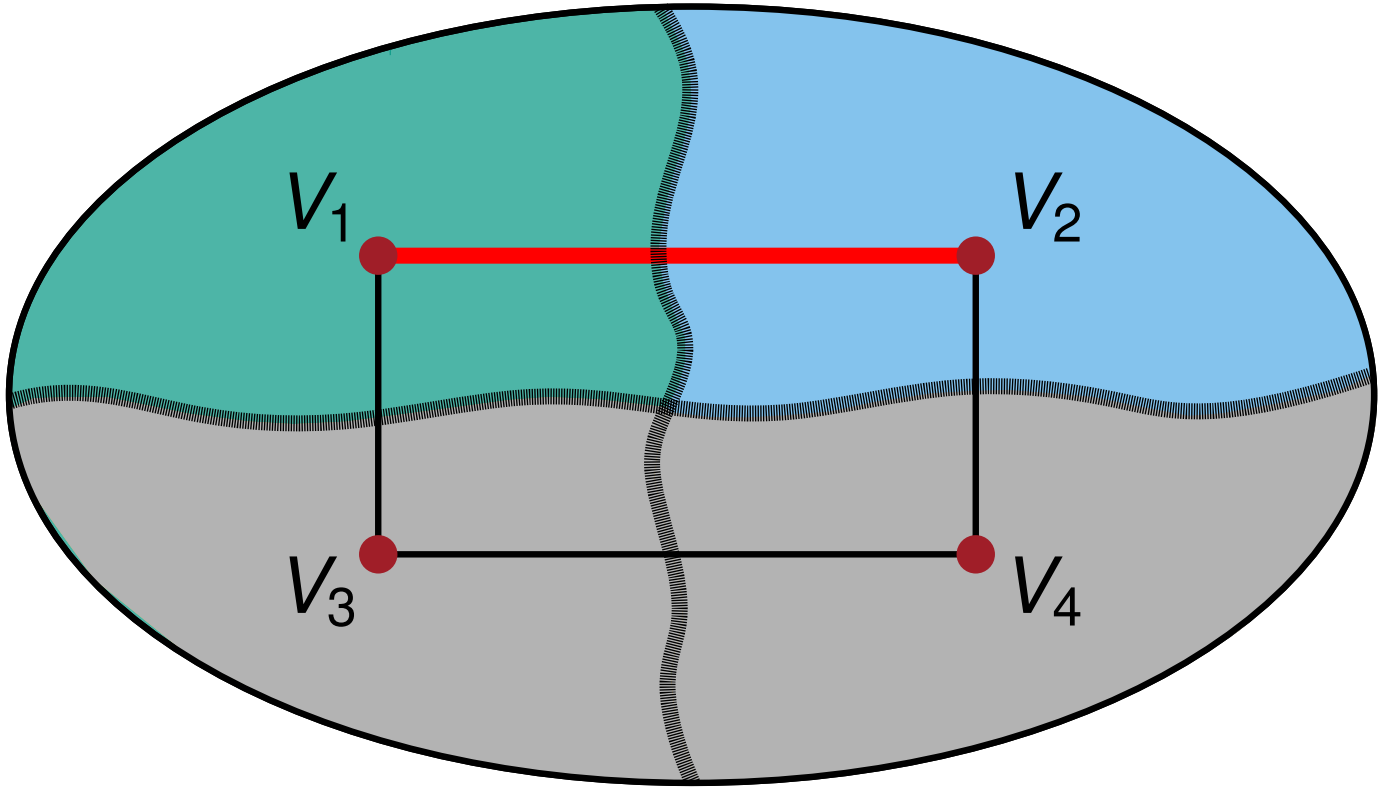


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 2

refine(V_1, V_2) = **No Improvement!**

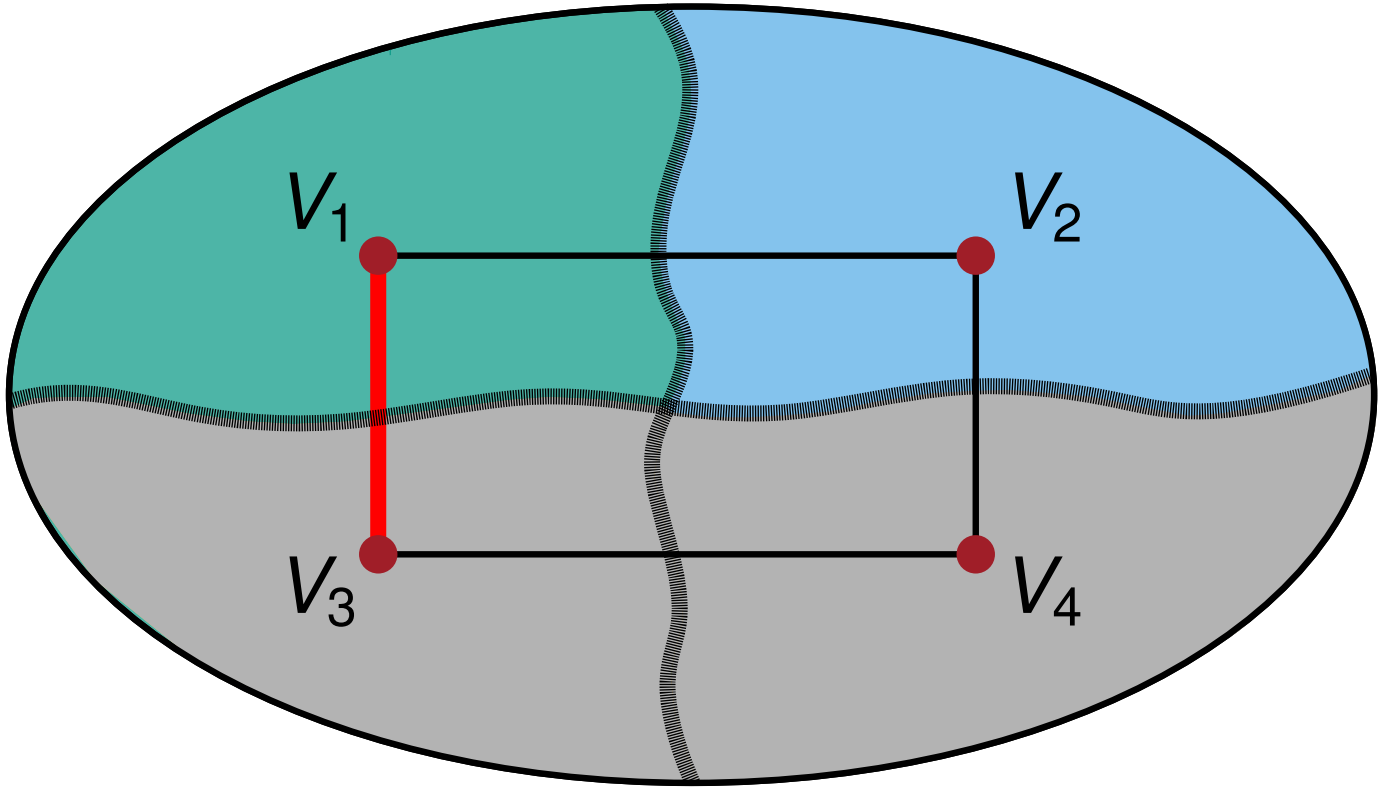


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 2

refine(V_1, V_3) = **No Improvement!**

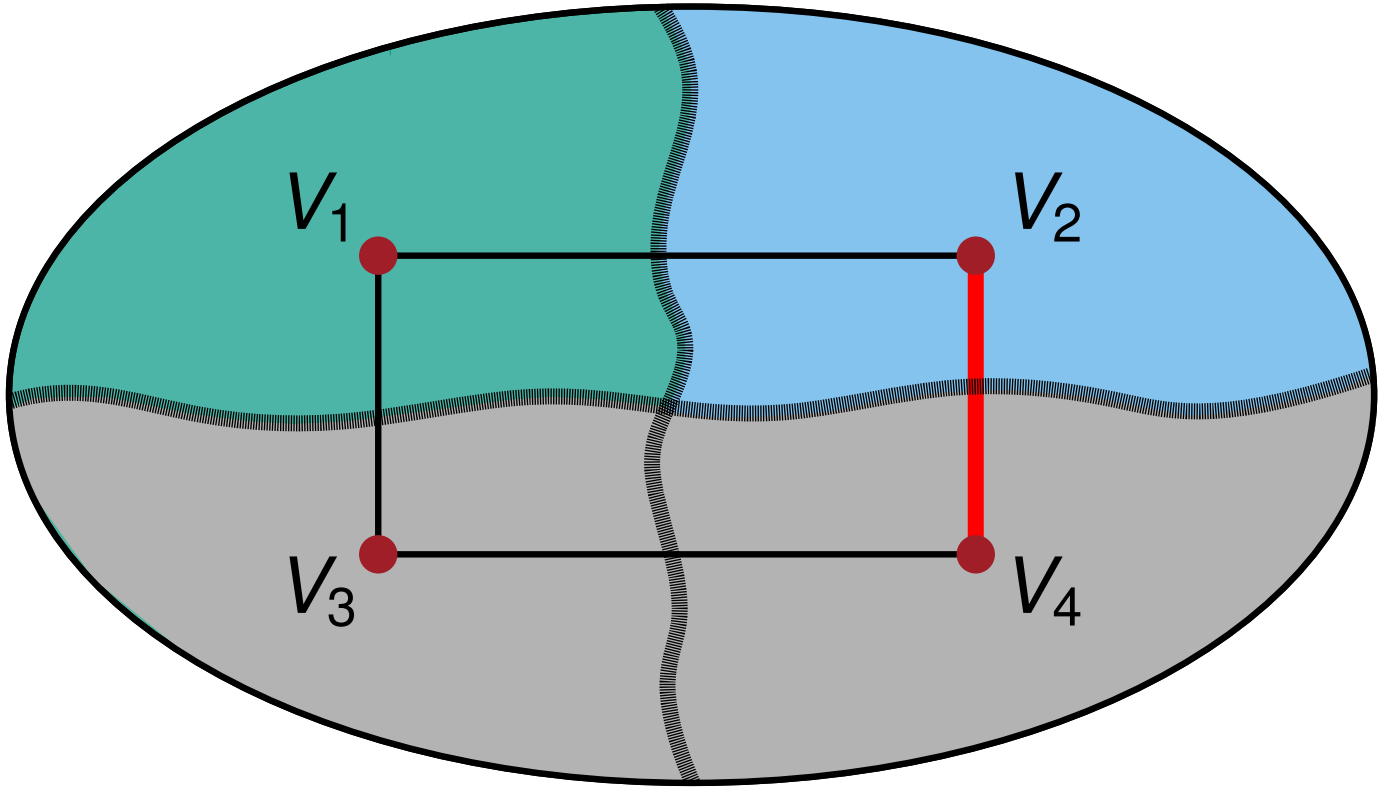


Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

Round 2

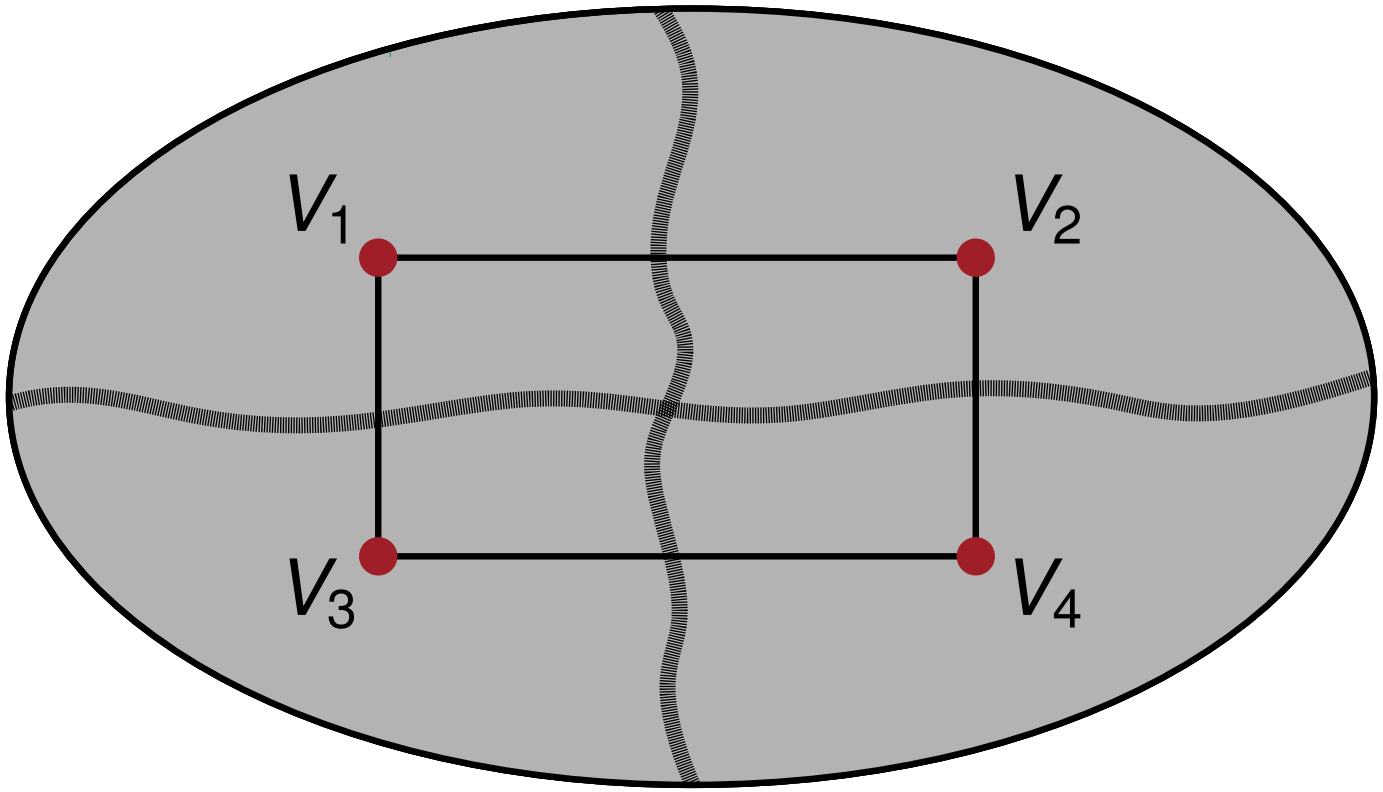
refine(V_2, V_4) = **No Improvement!**



Using 2-way refinement algorithm for **active** blocks of the quotient graph

Active Block Scheduling

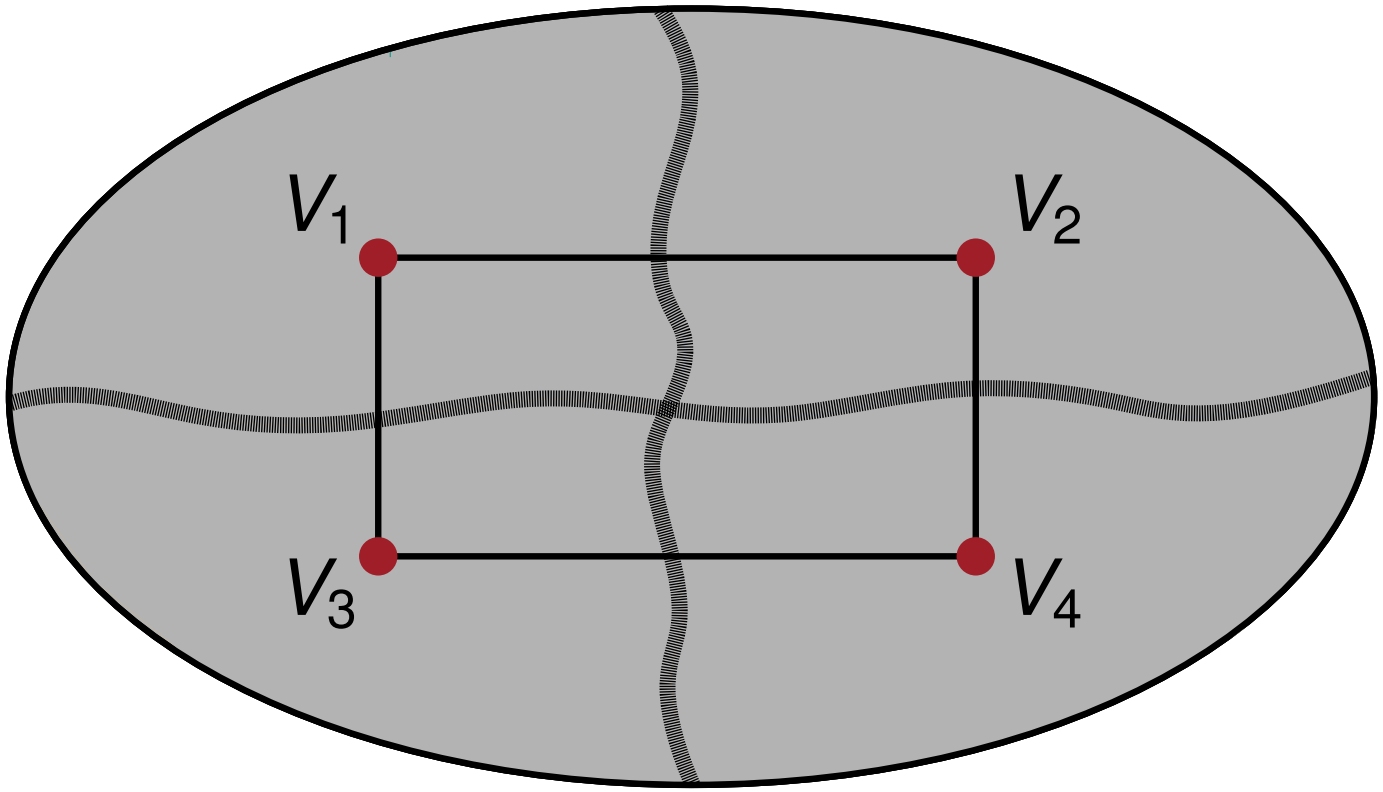
Round 2 Boundary did not change $\Rightarrow$ Mark block as **inactive**



Using 2-way refinement algorithm for **active** blocks of the quotient graph

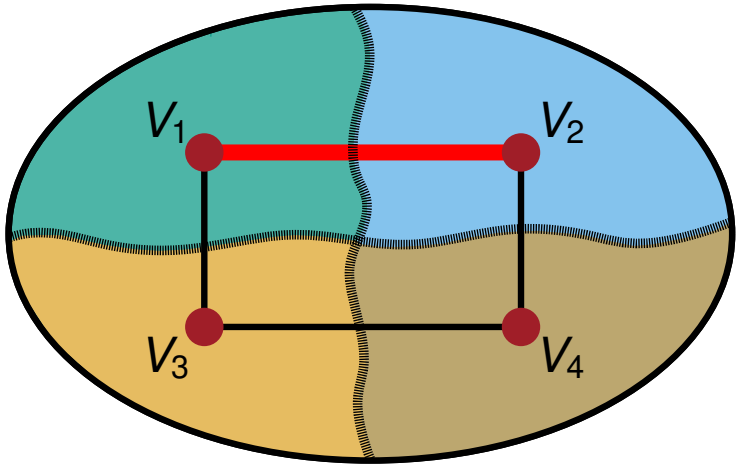
Active Block Scheduling

Round 2 All blocks are **inactive** $\Rightarrow$ Algorithm terminates

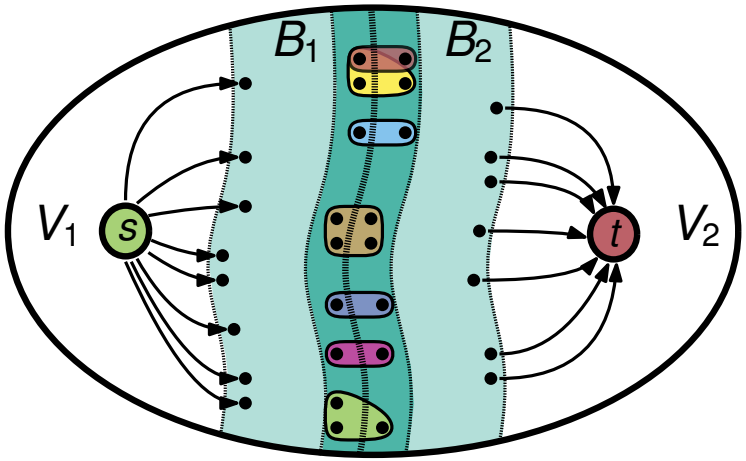


Using 2-way refinement algorithm for **active** blocks of the quotient graph

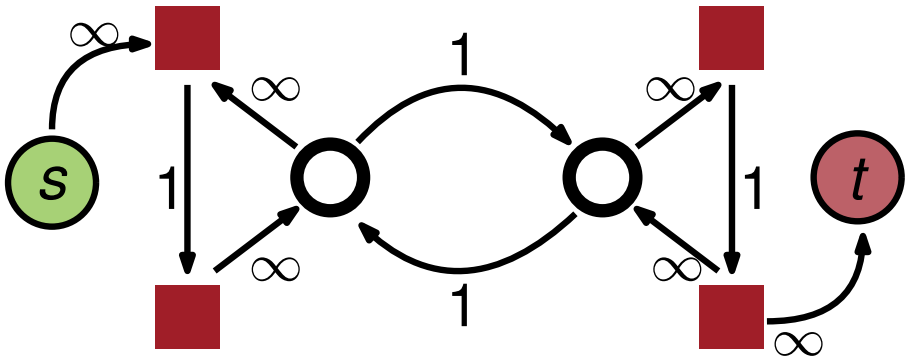
Our Flow-Based Refinement Framework



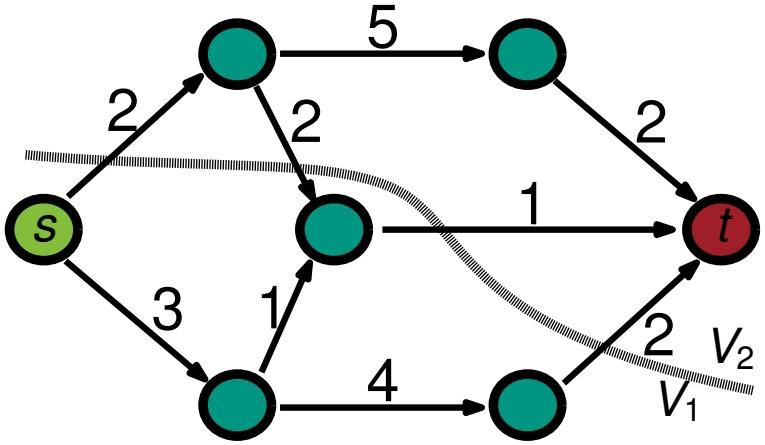
Select two adjacent blocks for refinement



Build Flow Problem



Solve Flow Problem

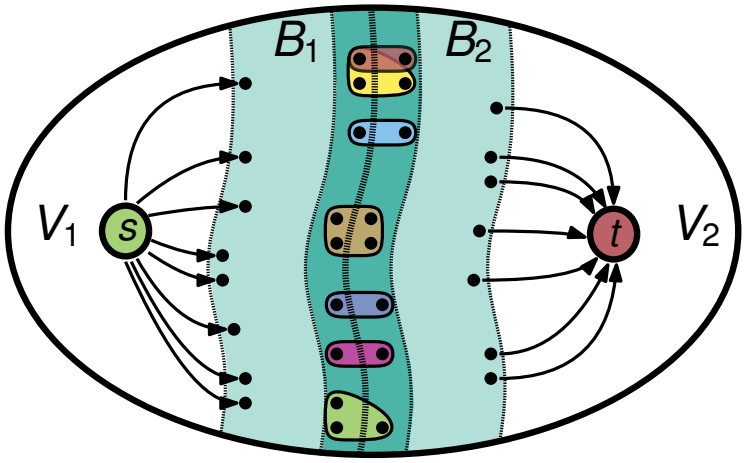


Find feasible minimum cut

Institute of Theoretical Informatics
Algorithmics Group

Our Flow-Based Refinement Framework

Select two adjacent blocks for refinement



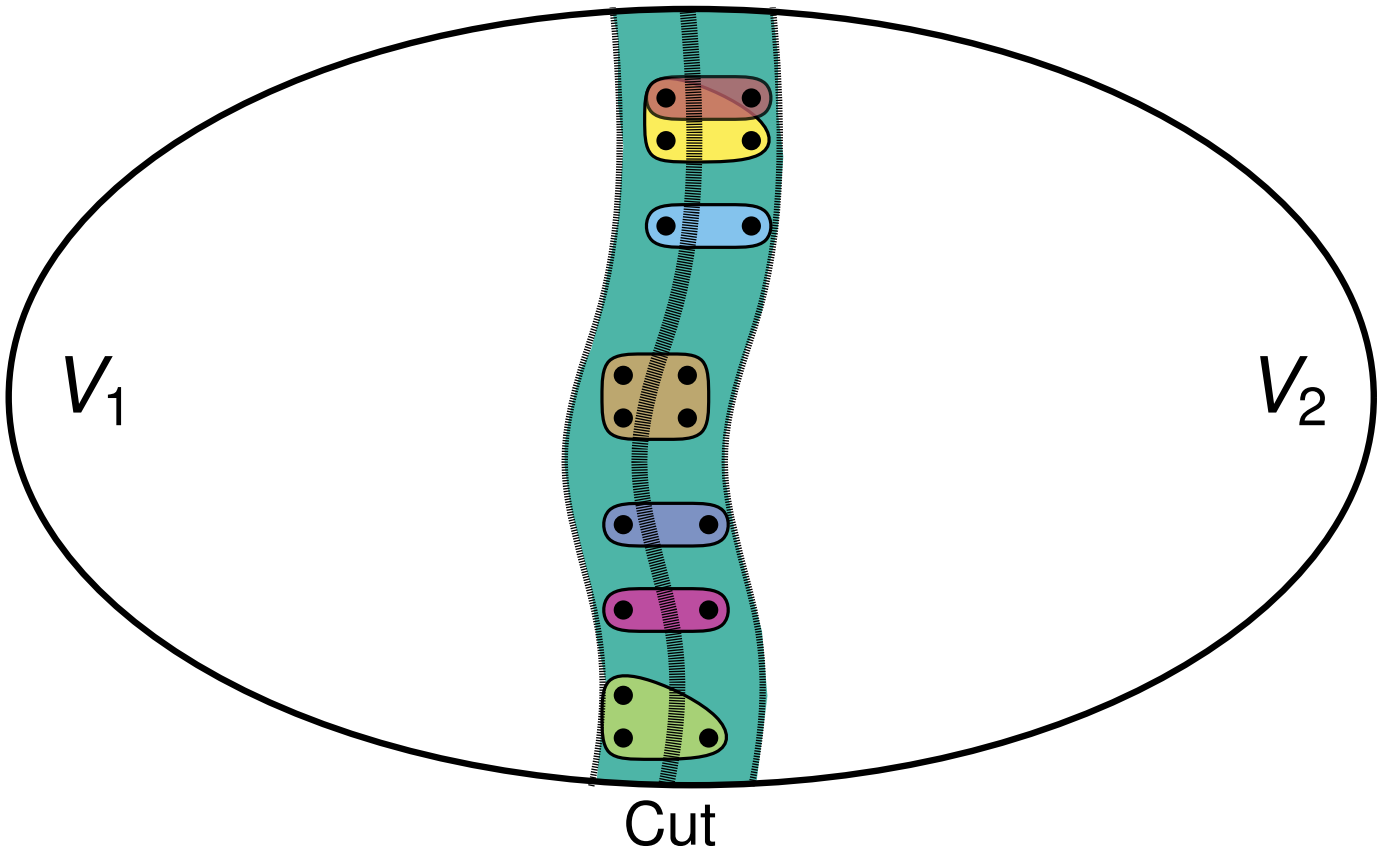
Build Flow Problem

Solve Flow Problem

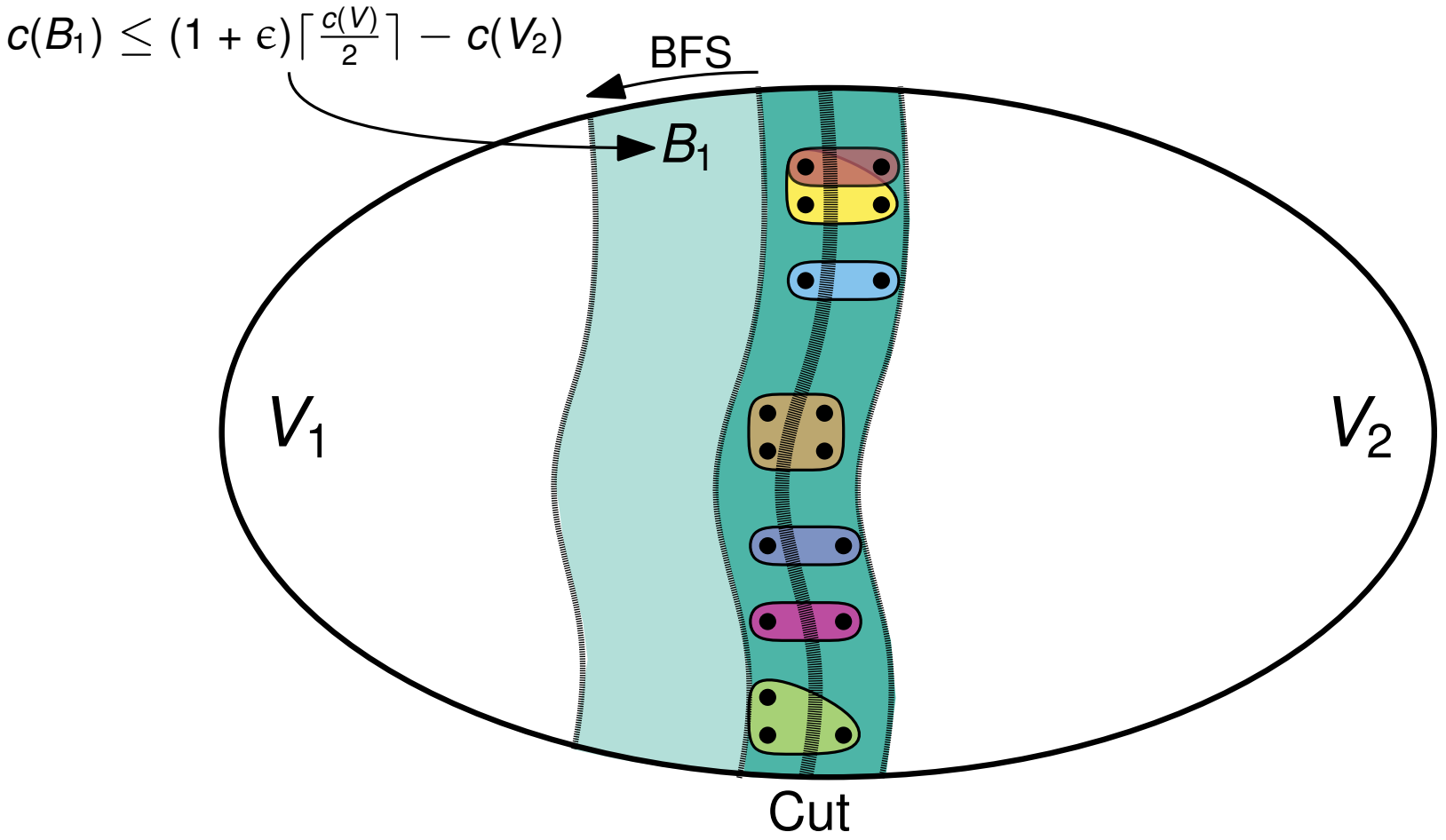
Find feasible minimum cut

Institute of Theoretical Informatics
Algorithmics Group

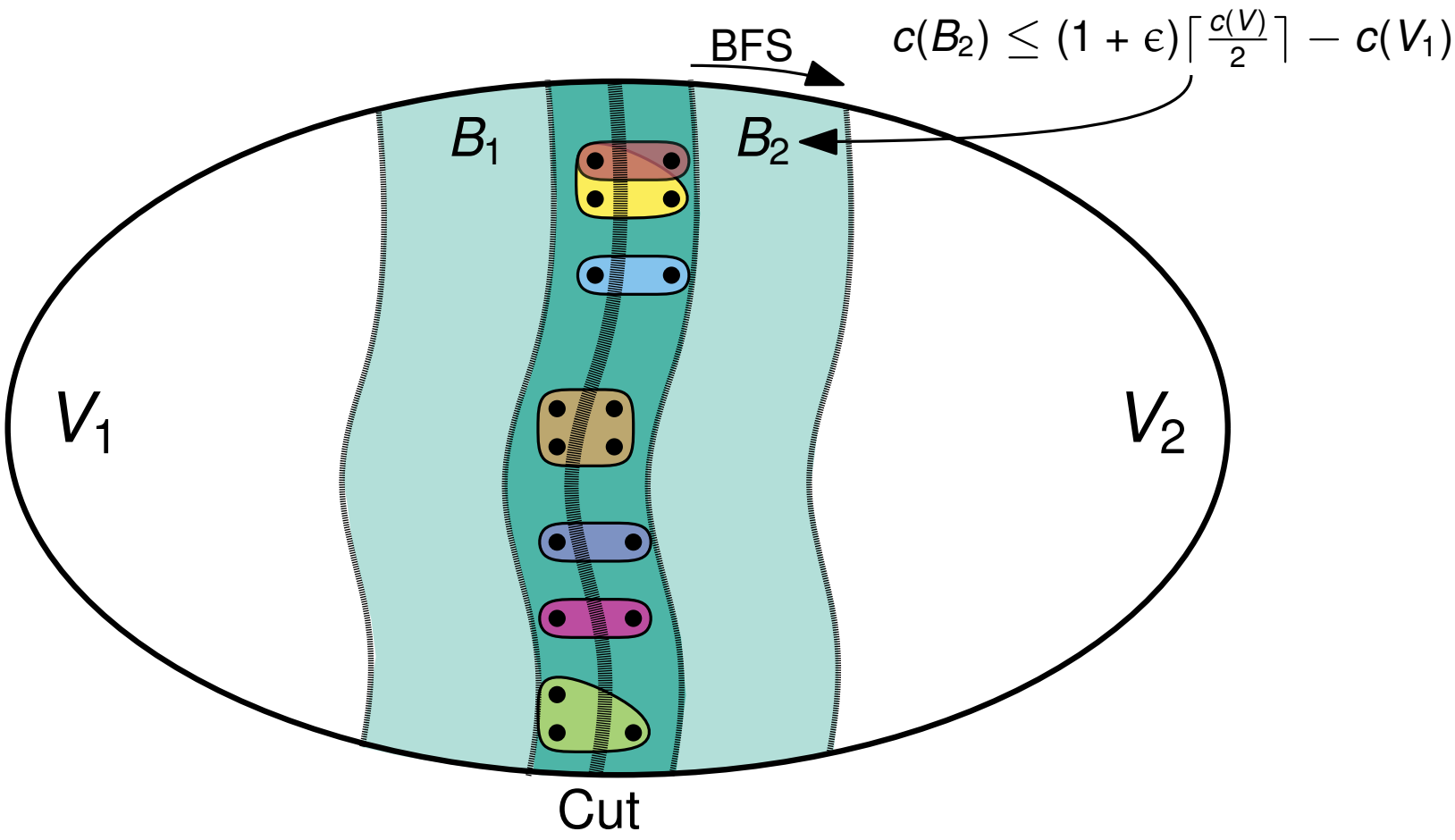
Adaptive Flow Iterations



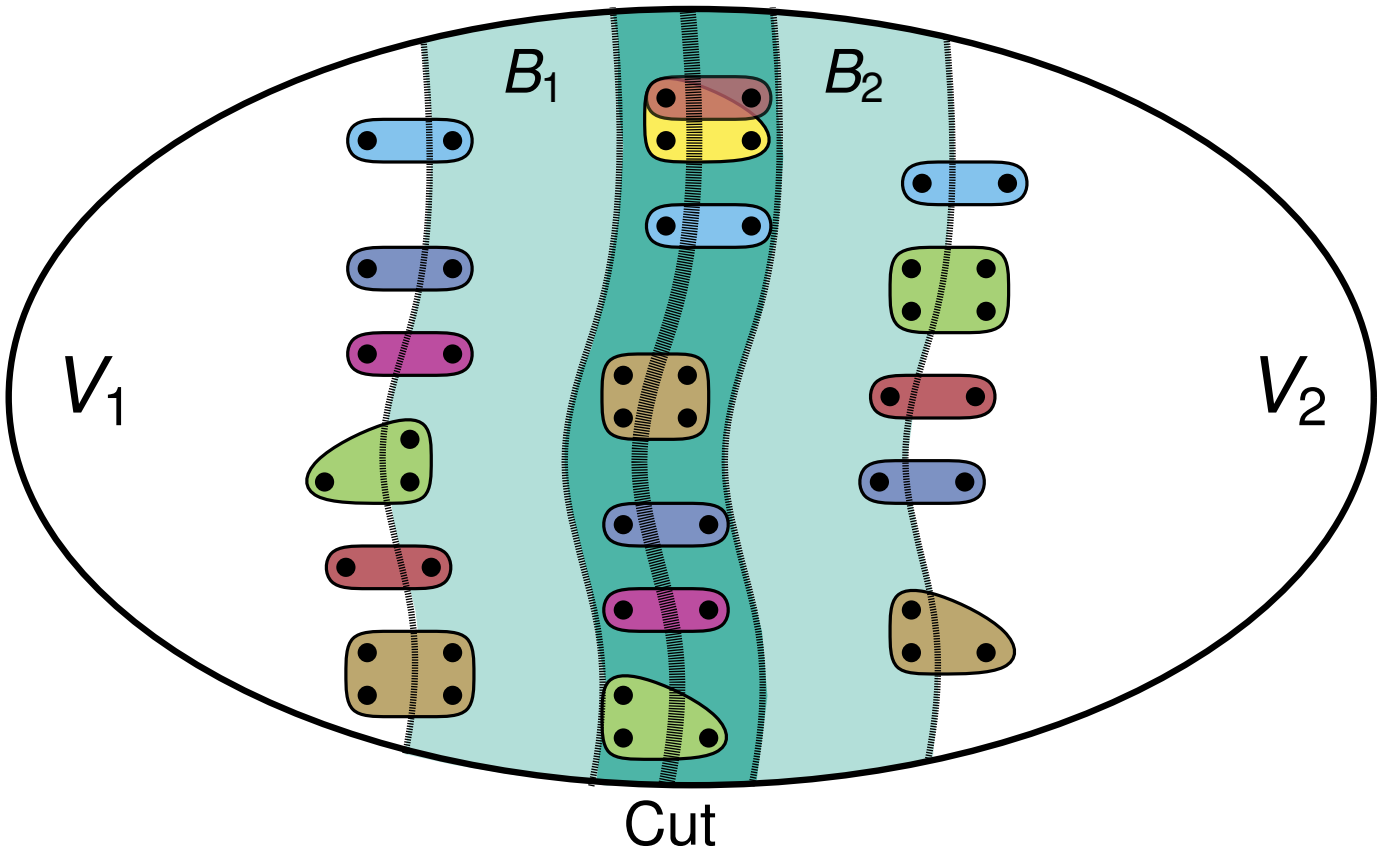
Adaptive Flow Iterations



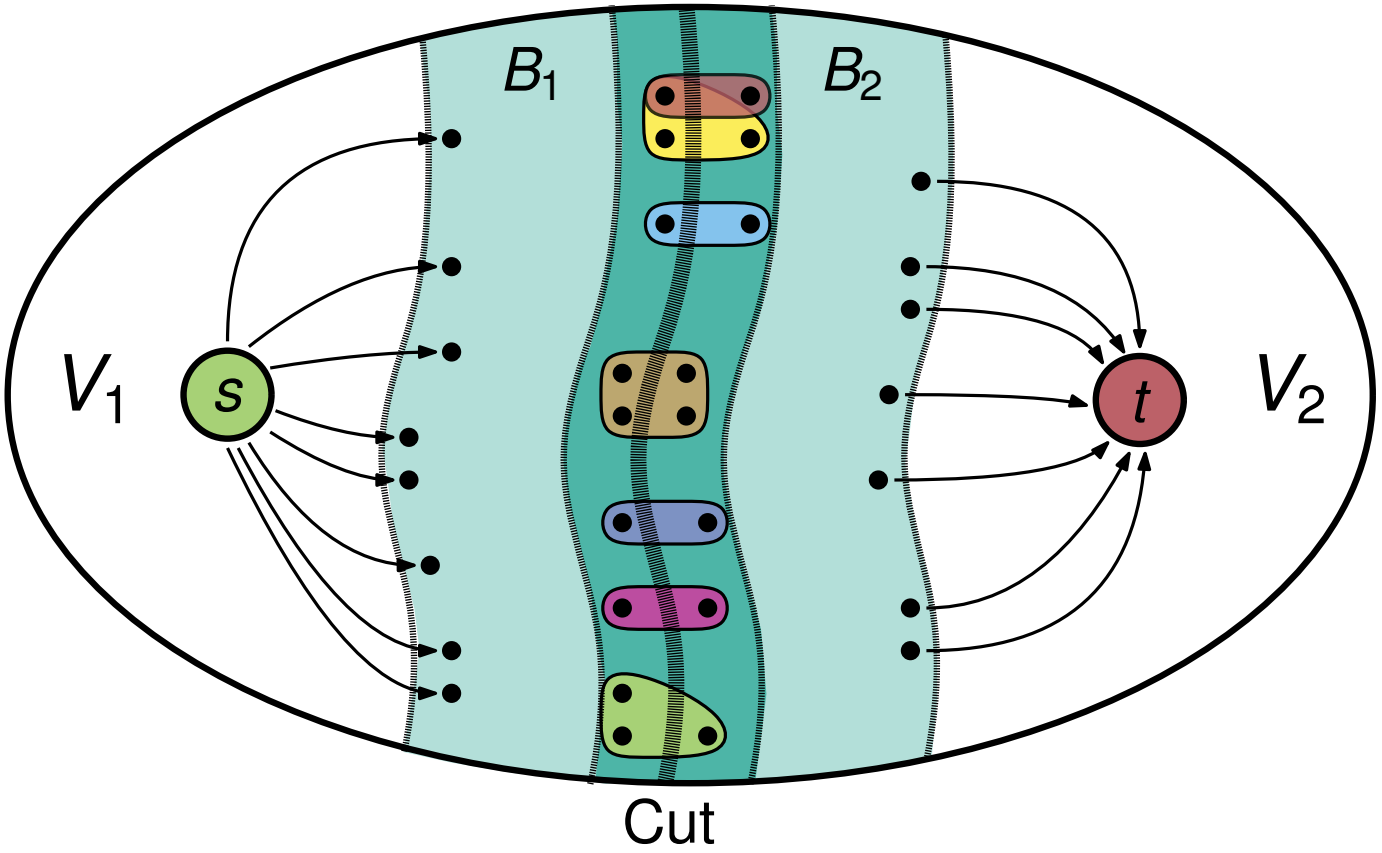
Adaptive Flow Iterations



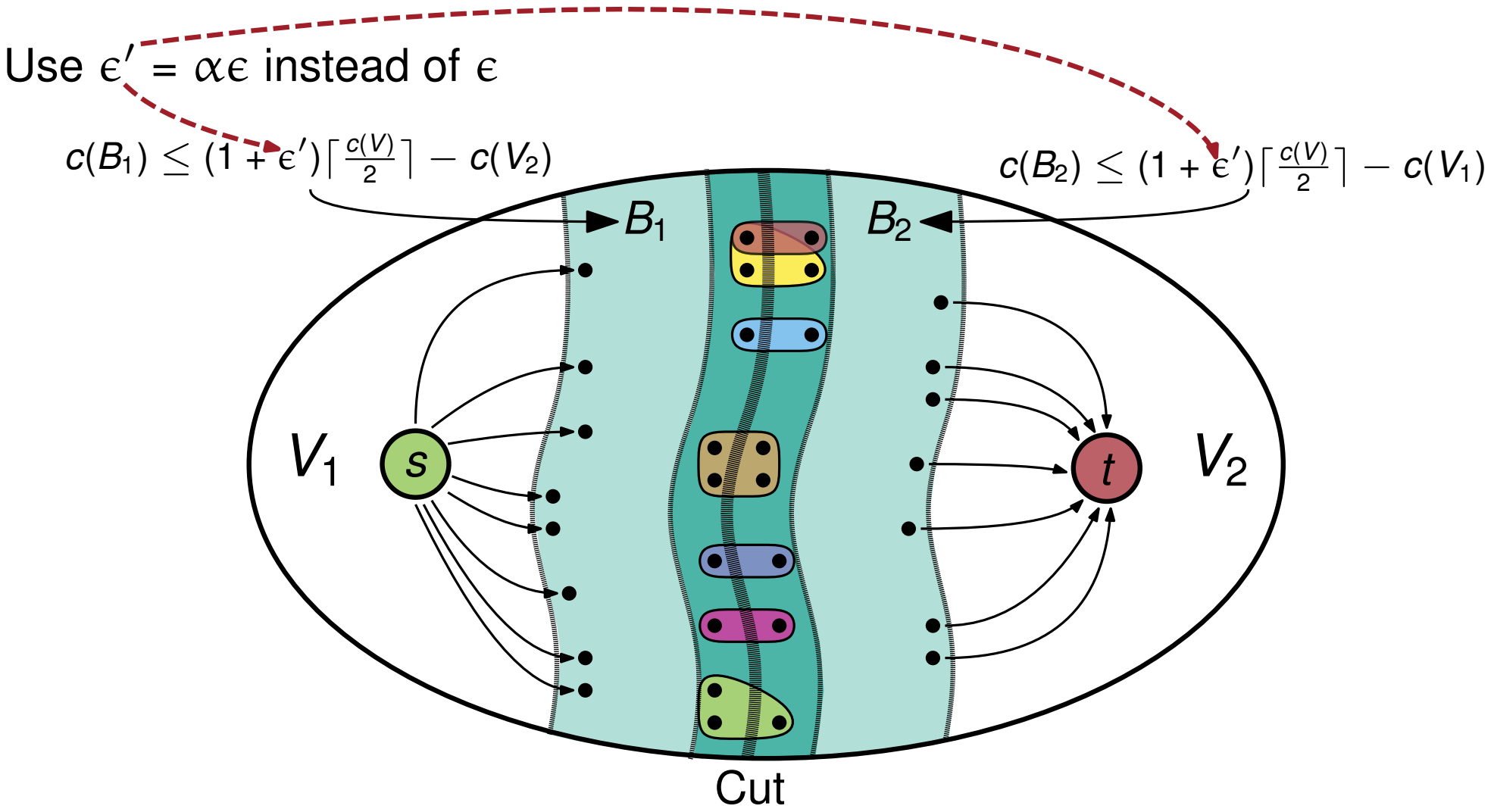
Adaptive Flow Iterations



Adaptive Flow Iterations



Adaptive Flow Iterations

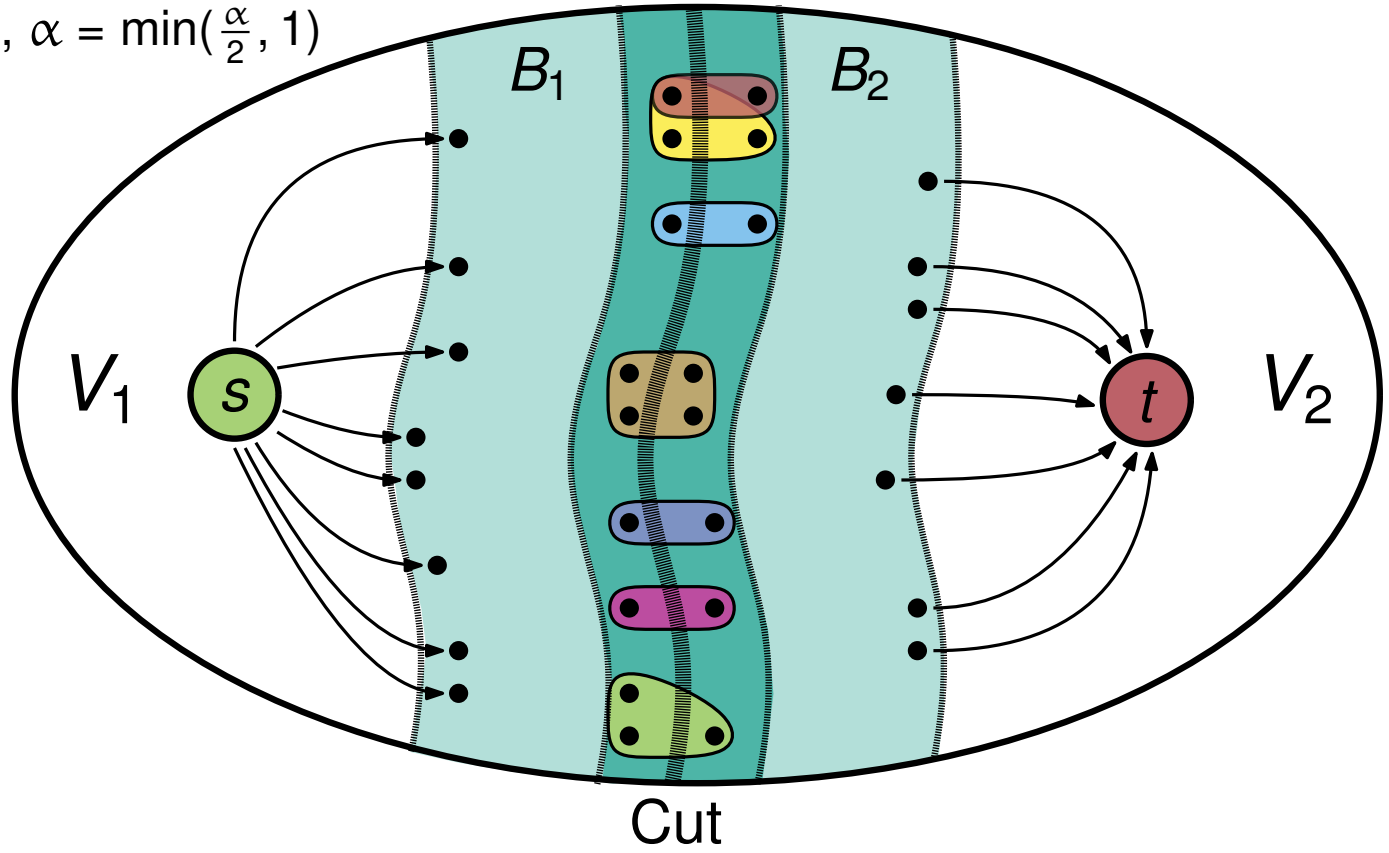


Institute of Theoretical Informatics
Algorithmics Group

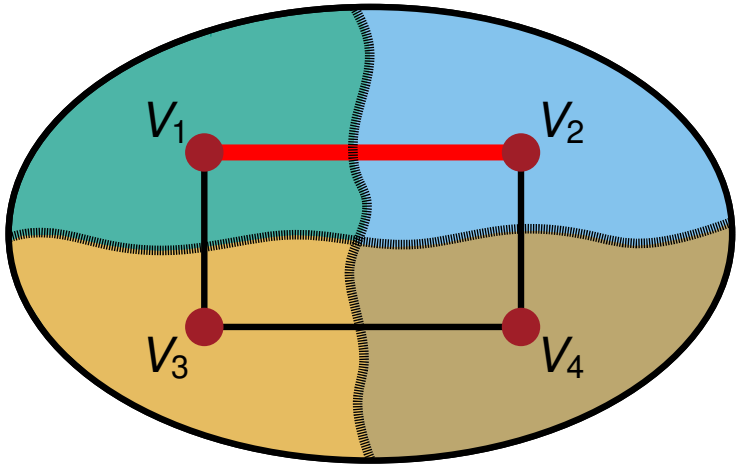
Adaptive Flow Iterations

Use $\epsilon' = \alpha\epsilon$ instead of ϵ

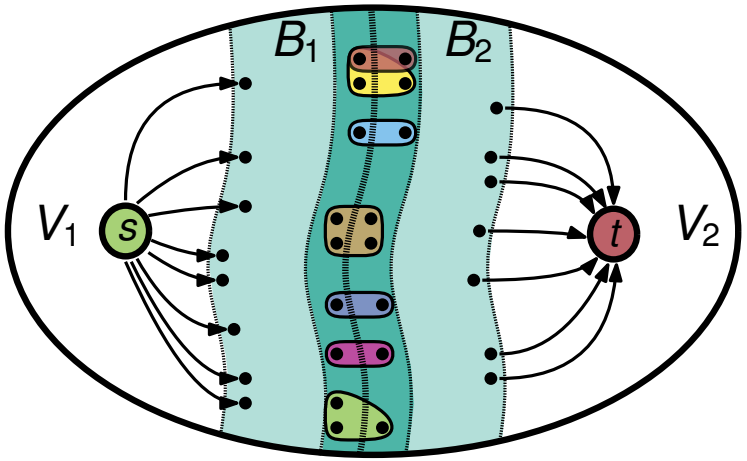
- If improvement, $\alpha = \max(2\alpha, \alpha')$ where α' is a predefined upper bound
- Otherwise, $\alpha = \min(\frac{\alpha}{2}, 1)$



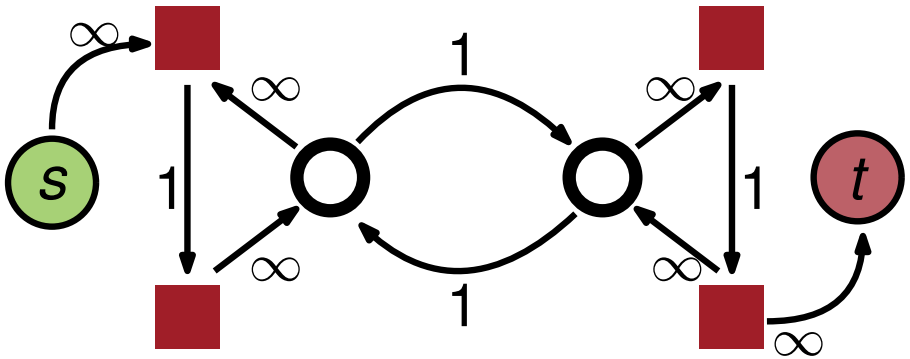
Our Flow-Based Refinement Framework



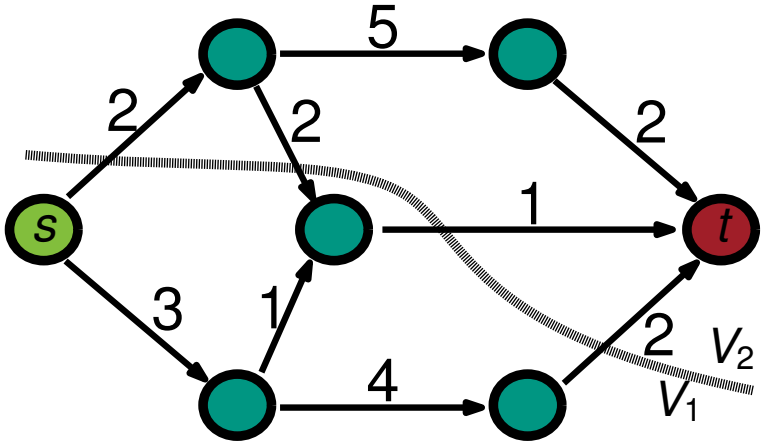
Select two adjacent blocks for refinement



Build Flow Problem



Solve Flow Problem



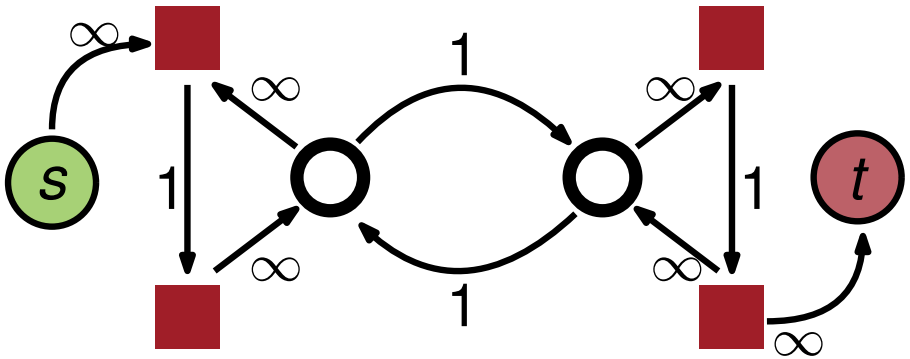
Find feasible minimum cut

Institute of Theoretical Informatics
Algorithmics Group

Our Flow-Based Refinement Framework

Select two adjacent blocks for refinement

Build Flow Problem



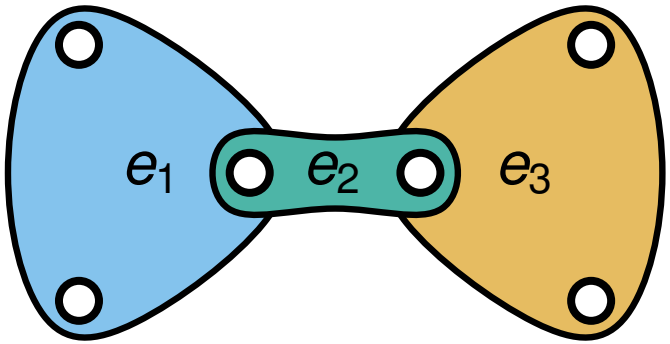
Solve Flow Problem

Find feasible minimum cut

Institute of Theoretical Informatics
Algorithmics Group

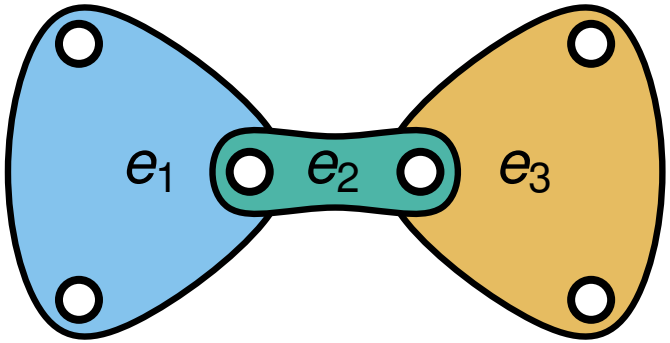
Hypergraph Flow Network

Hypergraph H

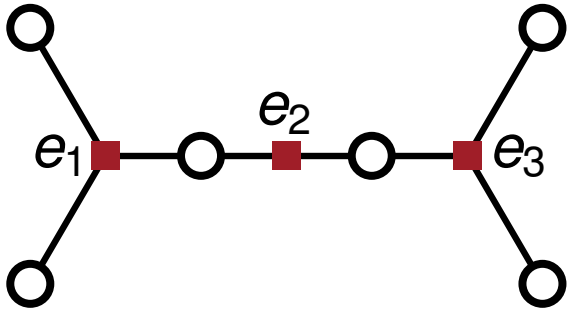


Hypergraph Flow Network

Hypergraph H

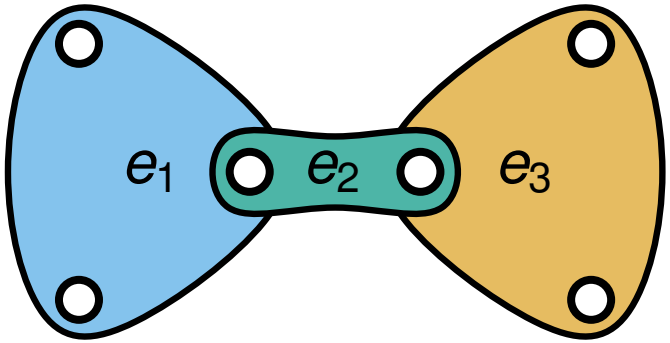


Bipartite Graph $G_*(H)$

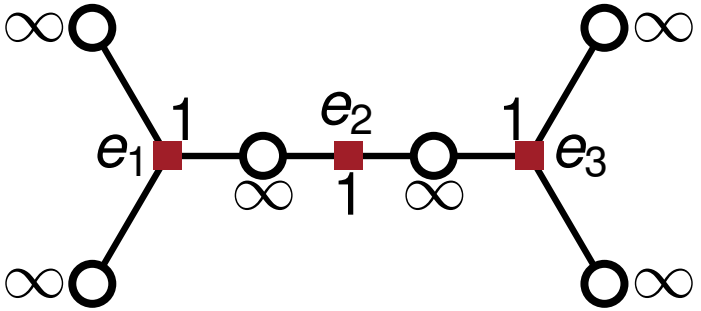


Hypergraph Flow Network

Hypergraph H



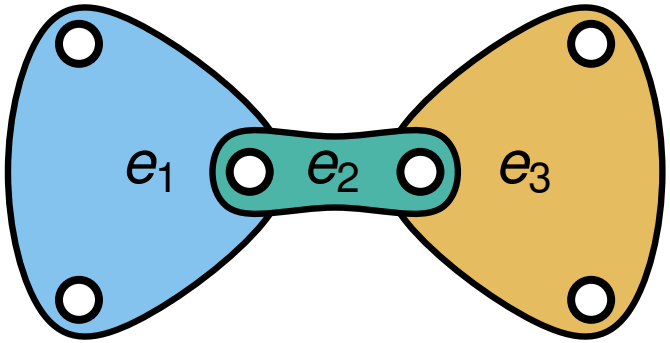
Bipartite Graph $G_*(H)$



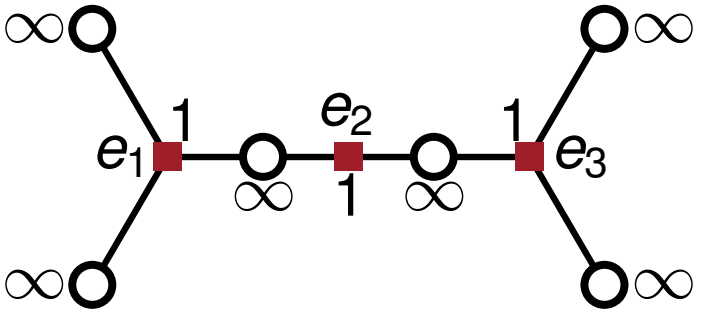
Vertex Separator Problem

Hypergraph Flow Network

Hypergraph H

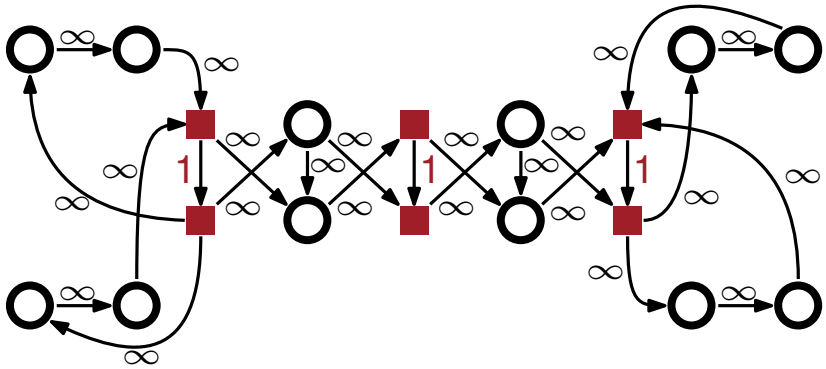


Bipartite Graph $G_*(H)$



Vertex Separator Problem

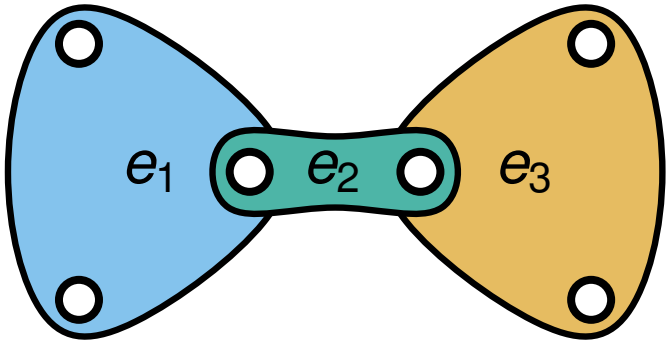
Vertex Separator Transformation



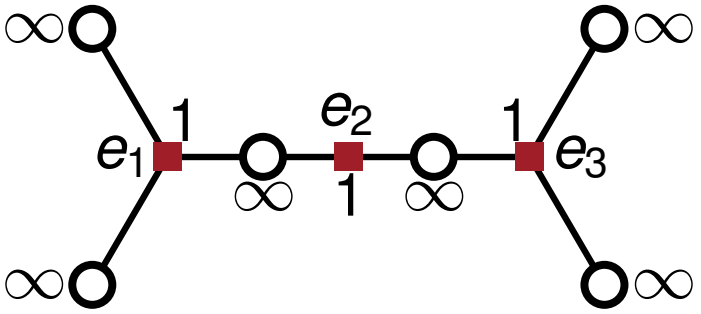
Institute of Theoretical Informatics
Algorithmics Group

Hypergraph Flow Network

Hypergraph H

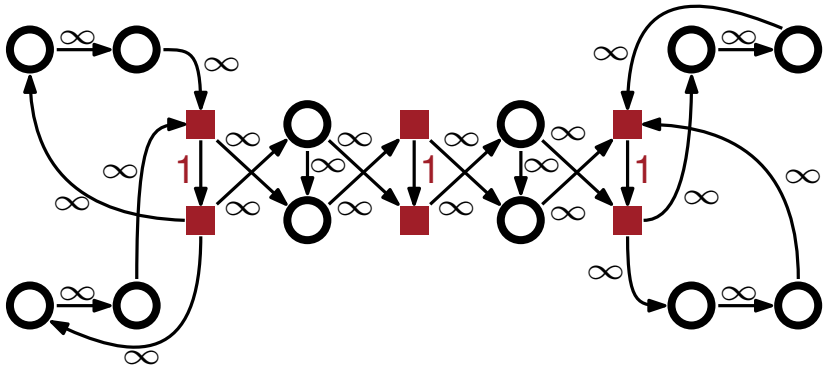


Bipartite Graph $G_*(H)$

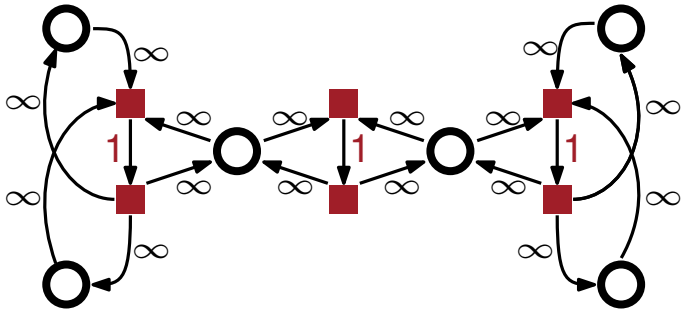


Vertex Separator Problem

Vertex Separator Transformation



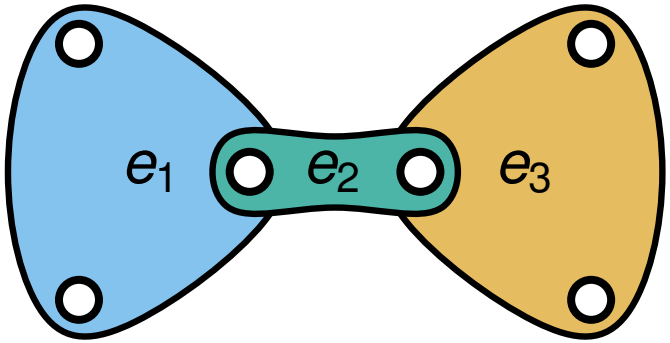
Lawler Network



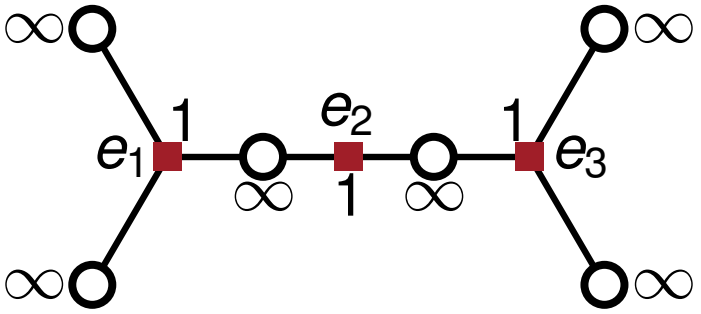
Institute of Theoretical Informatics
Algorithmics Group

Hypergraph Flow Network

Hypergraph H

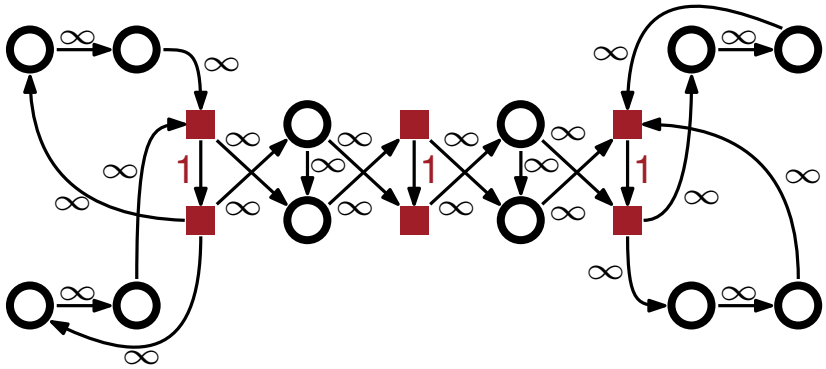


Bipartite Graph $G_*(H)$

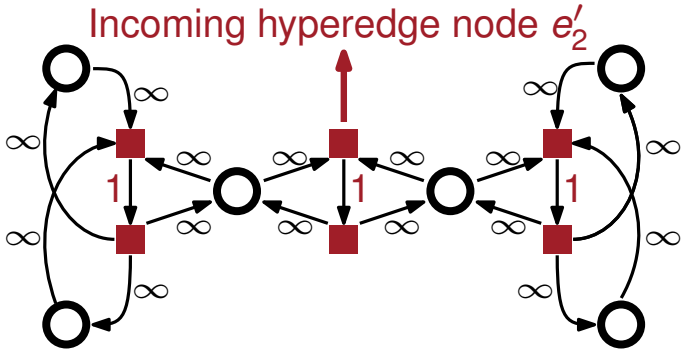


Vertex Separator Problem

Vertex Separator Transformation



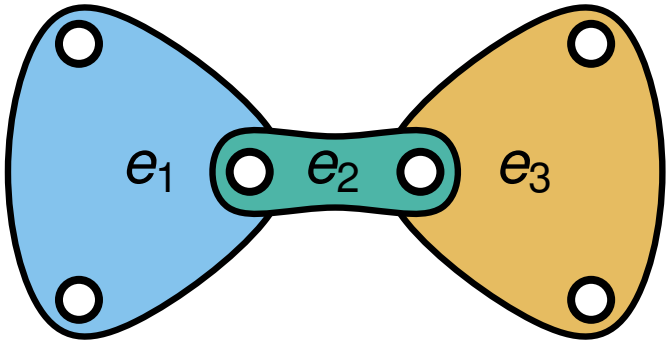
Lawler Network



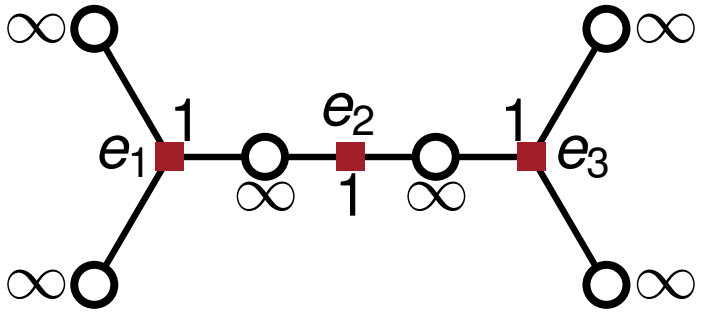
Institute of Theoretical Informatics
Algorithmics Group

Hypergraph Flow Network

Hypergraph H

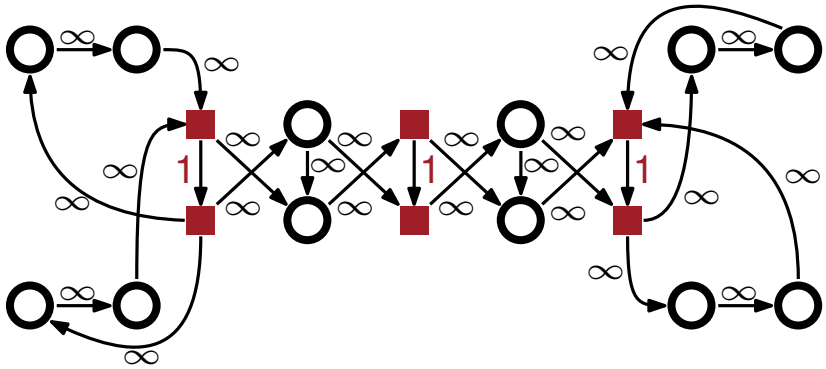


Bipartite Graph $G_*(H)$

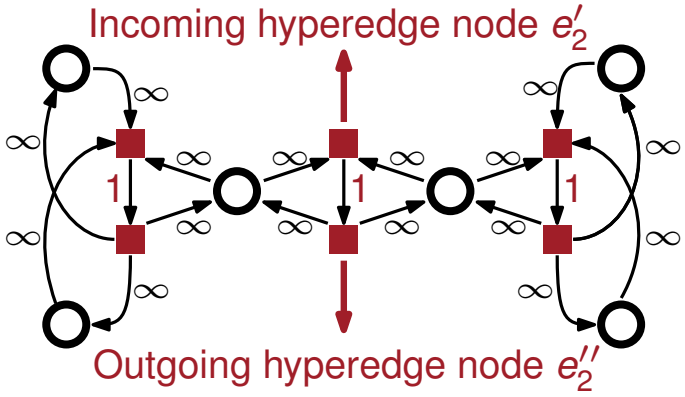


Vertex Separator Problem

Vertex Separator Transformation



Lawler Network

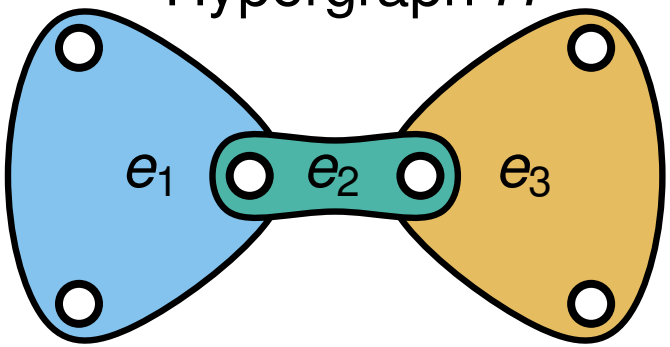


Institute of Theoretical Informatics
Algorithmics Group

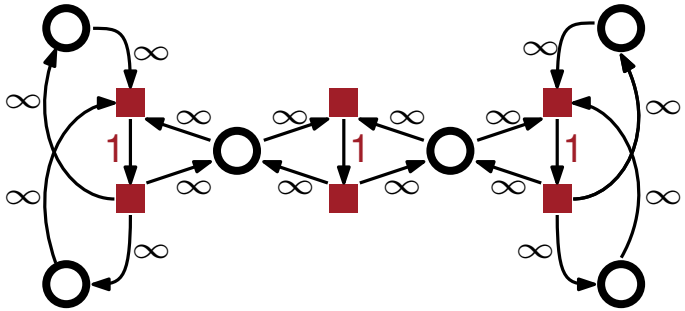
Hypergraph Flow Network - Graph Edges

Karlsruhe Institute of Technology

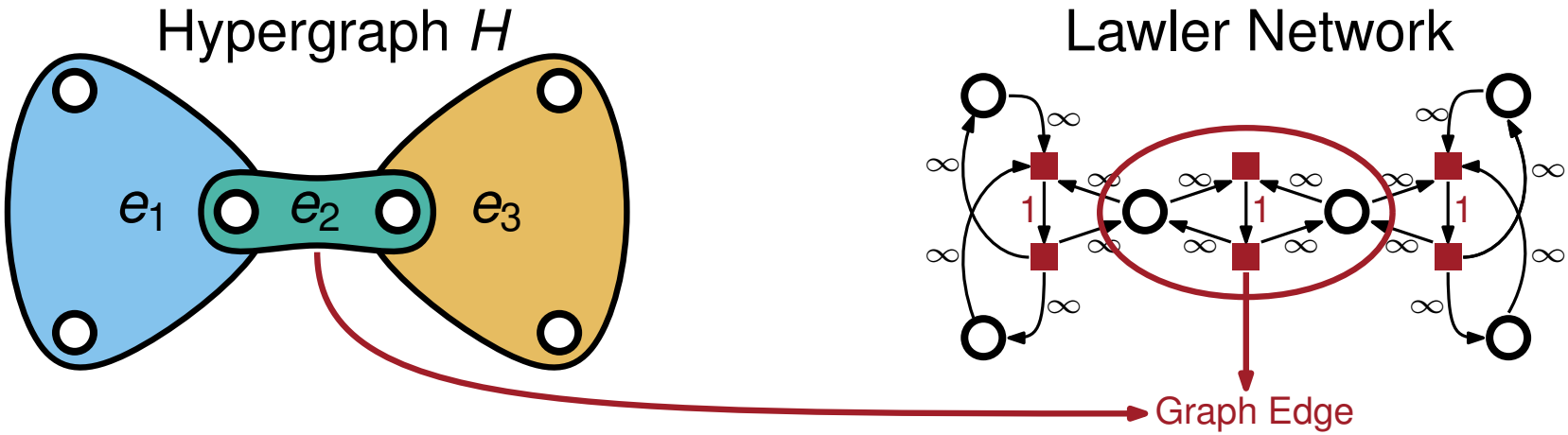
Hypergraph H



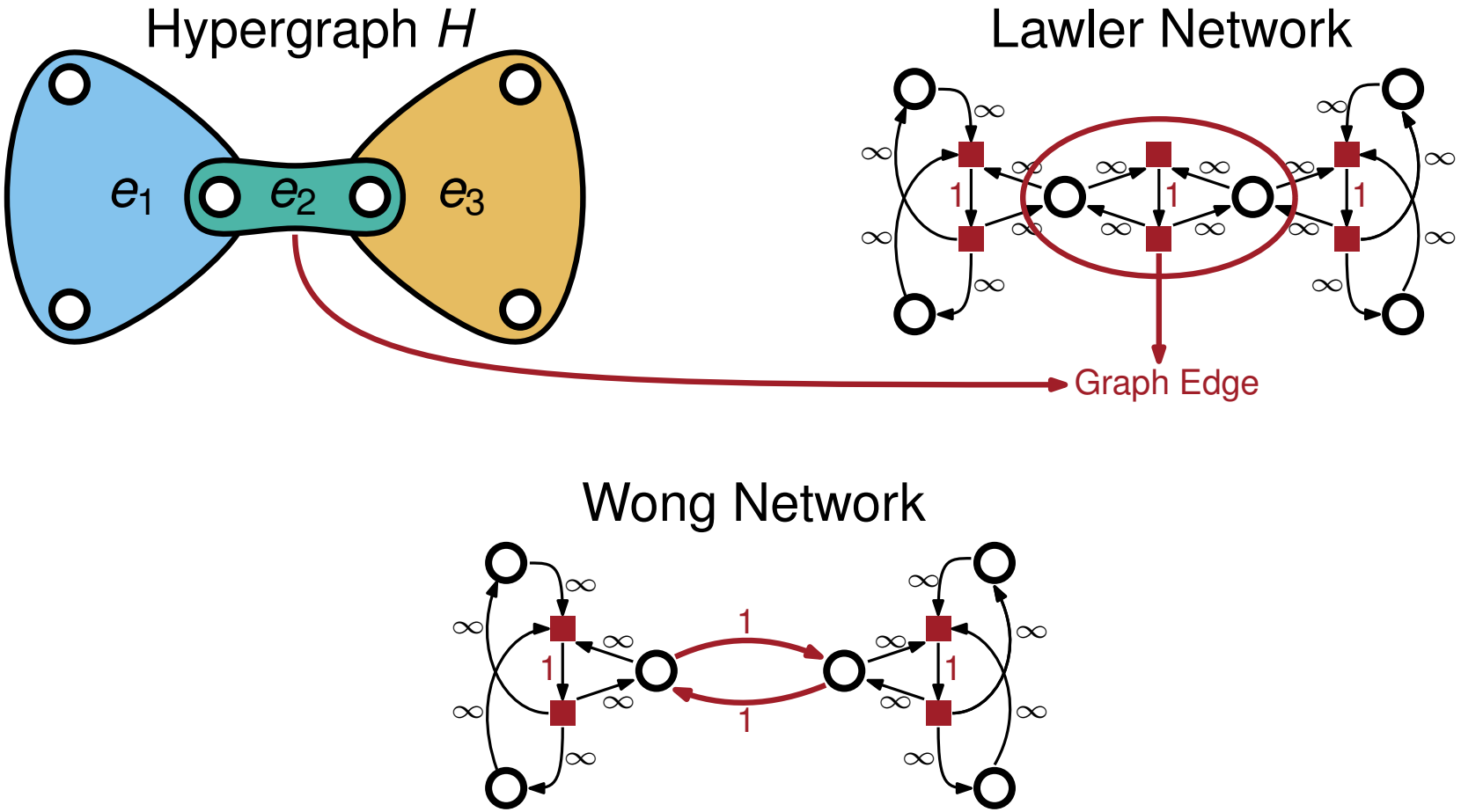
Lawler Network



Hypergraph Flow Network - Graph Edges

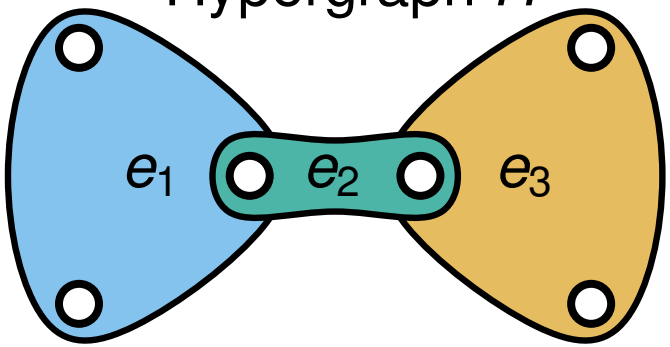


Hypergraph Flow Network - Graph Edges

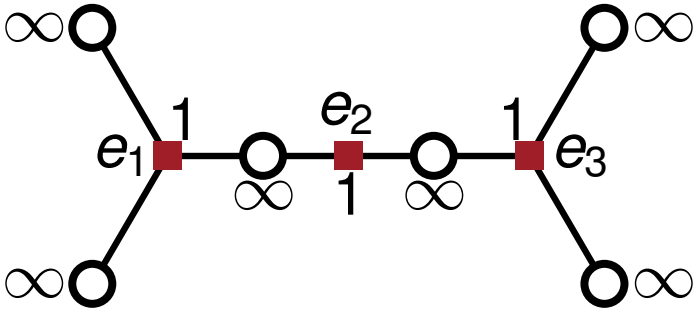


Hypergraph Flow Network - Low Degree Vertices

Hypergraph H

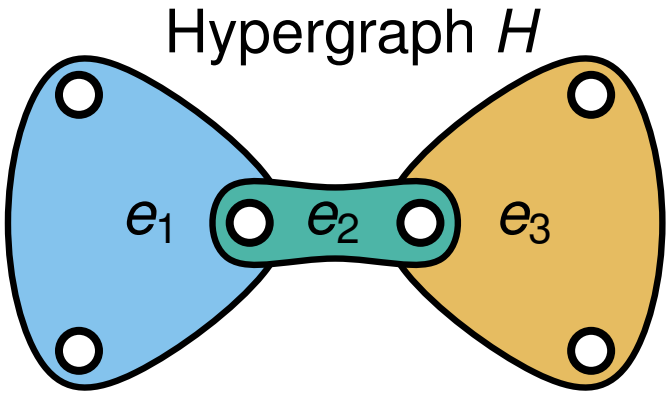


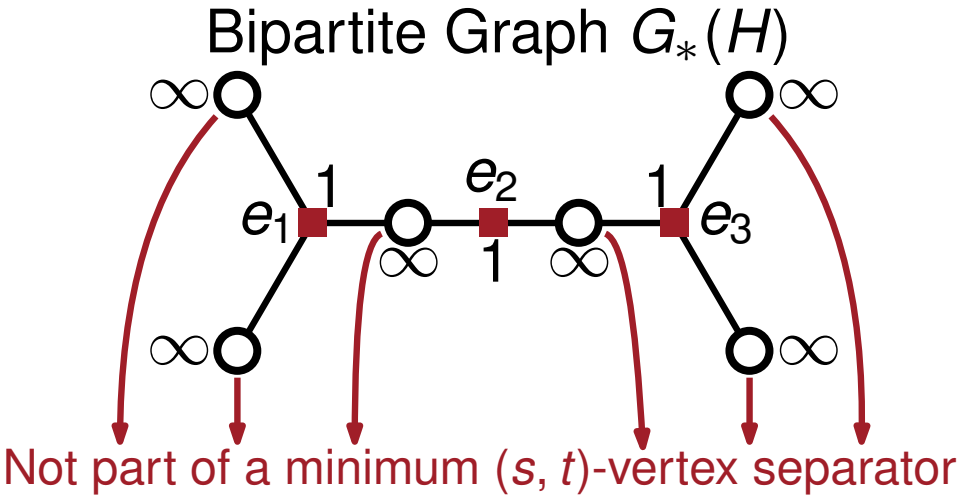
Bipartite Graph $G_*(H)$



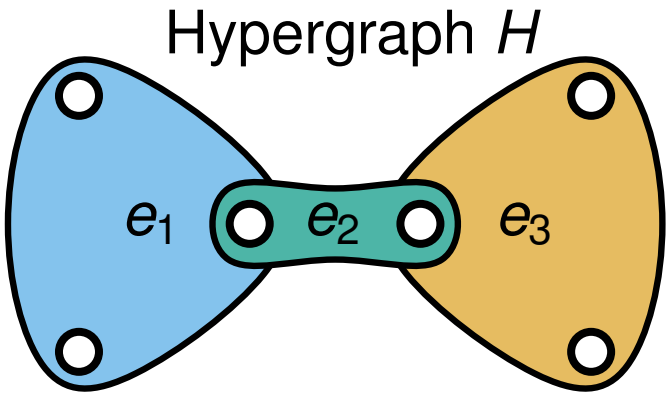
Institute of Theoretical Informatics
Algorithmics Group

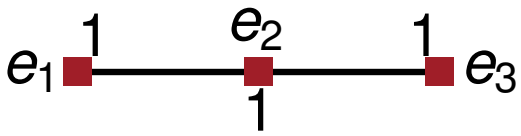
Hypergraph Flow Network - Low Degree Vertices





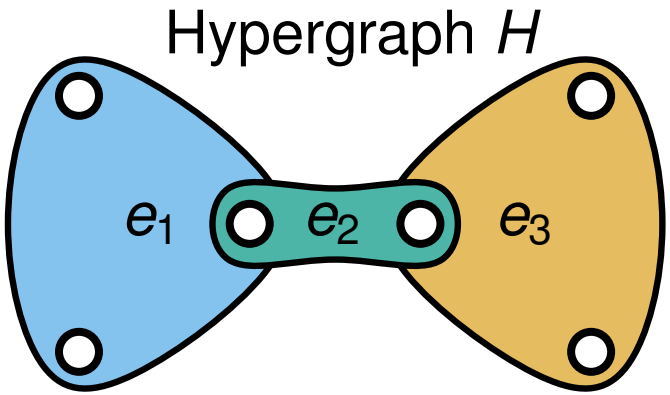
Hypergraph Flow Network - Low Degree Vertices

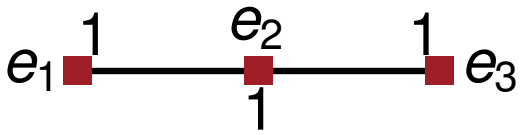




Remove all vertices by adding a clique

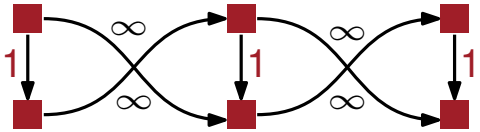
Hypergraph Flow Network - Low Degree Vertices



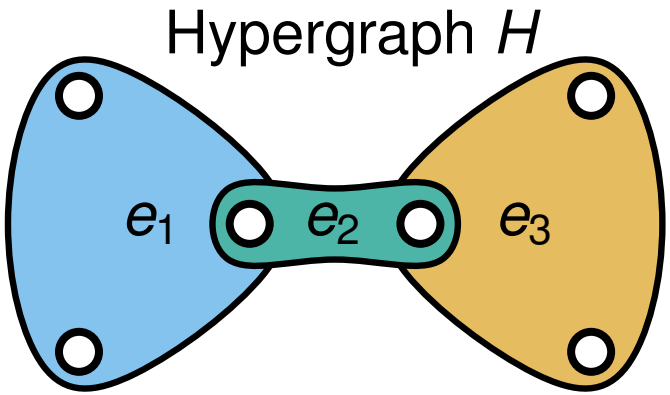


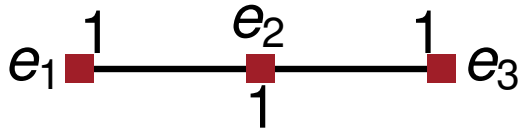
Remove all vertices by adding a clique

Our Network



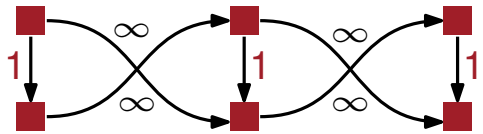
Hypergraph Flow Network - Low Degree Vertices





Remove all vertices by adding a clique

Our Network

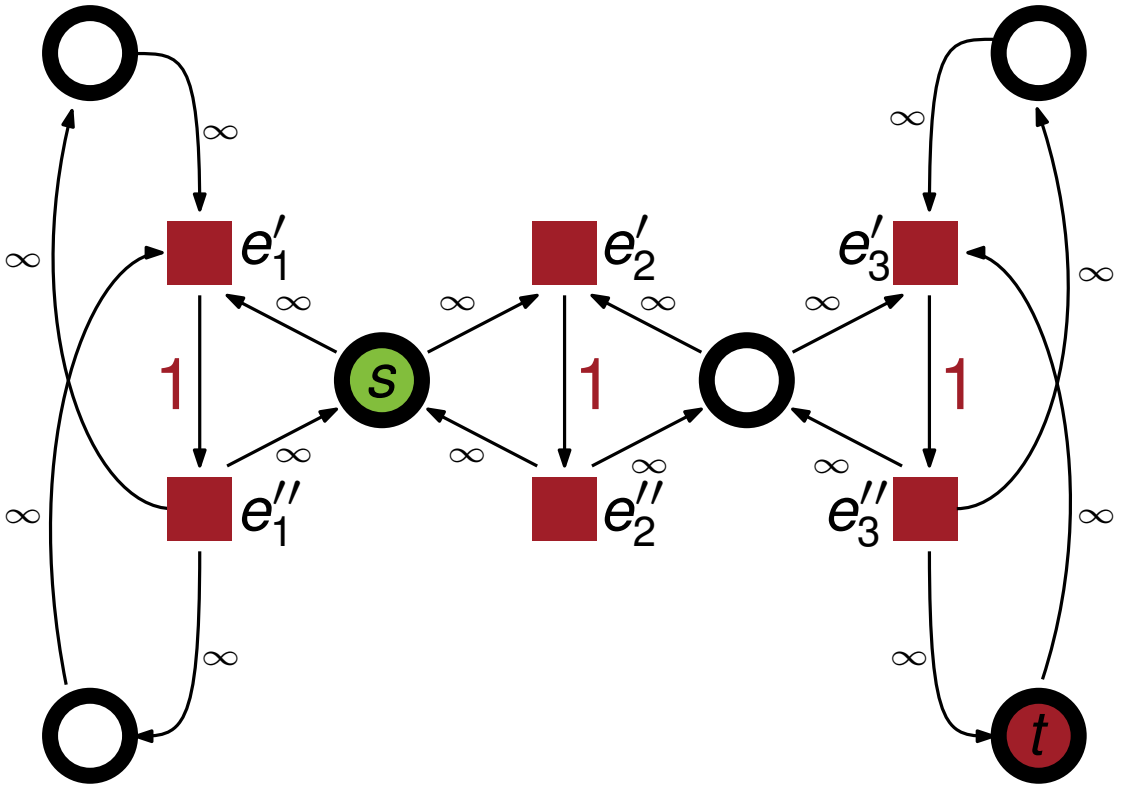


A hypernode v induces . . .

- . . . $2d(v)$ edges in the Lawler Network
- . . . $d(v)(d(v)-1)$ edges in our network

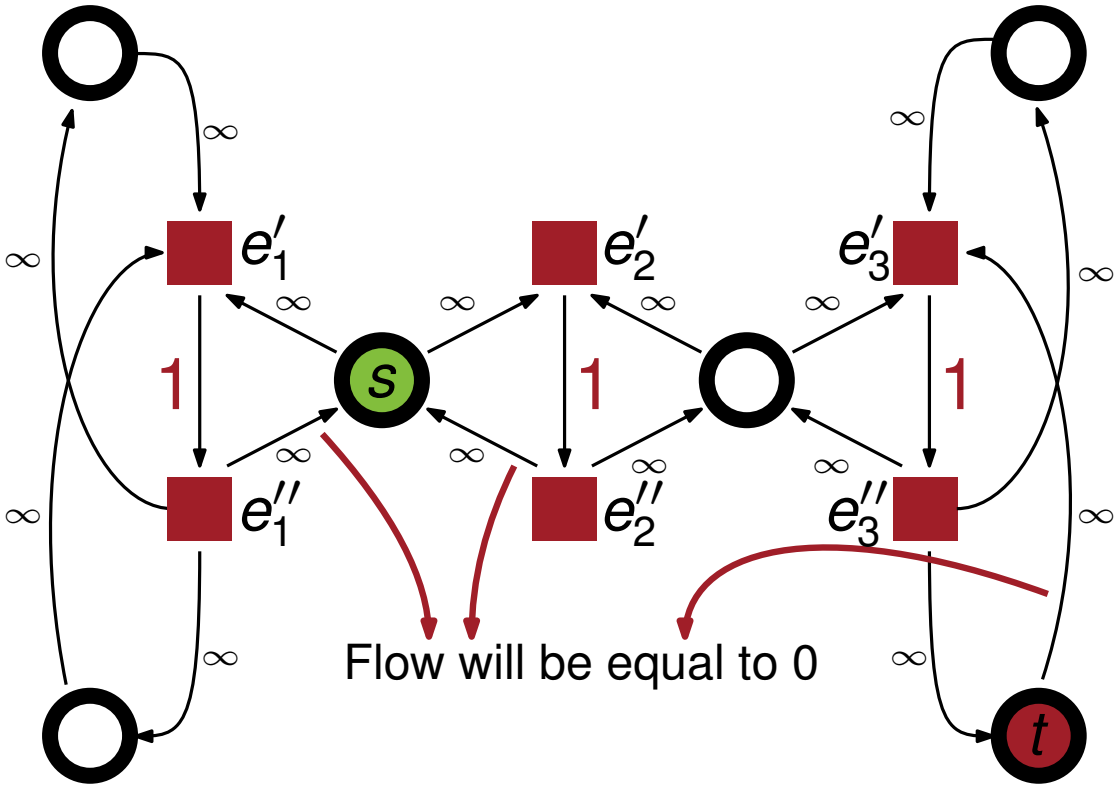
If $d(v) \leq 3$, then $d(v)(d(v)-1) \leq 2d(v)$

Removing Source and Sink Vertices



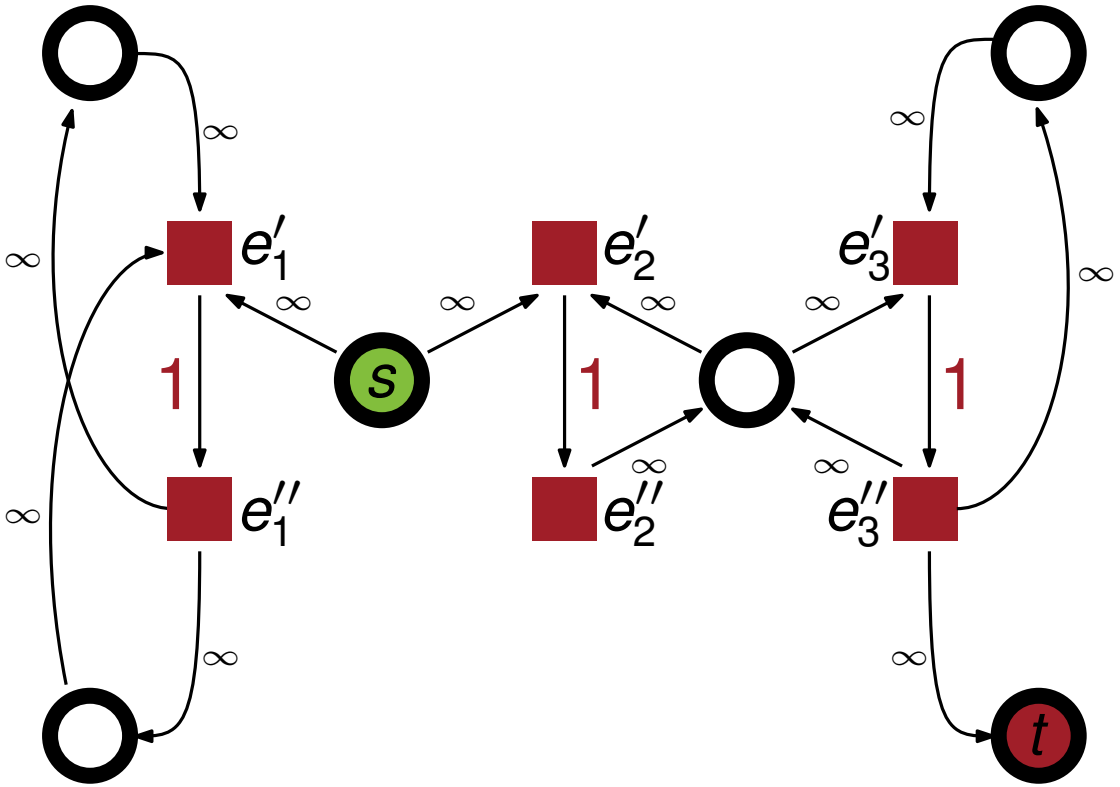
Lawler Network

Removing Source and Sink Vertices



Lawler Network

Removing Source and Sink Vertices

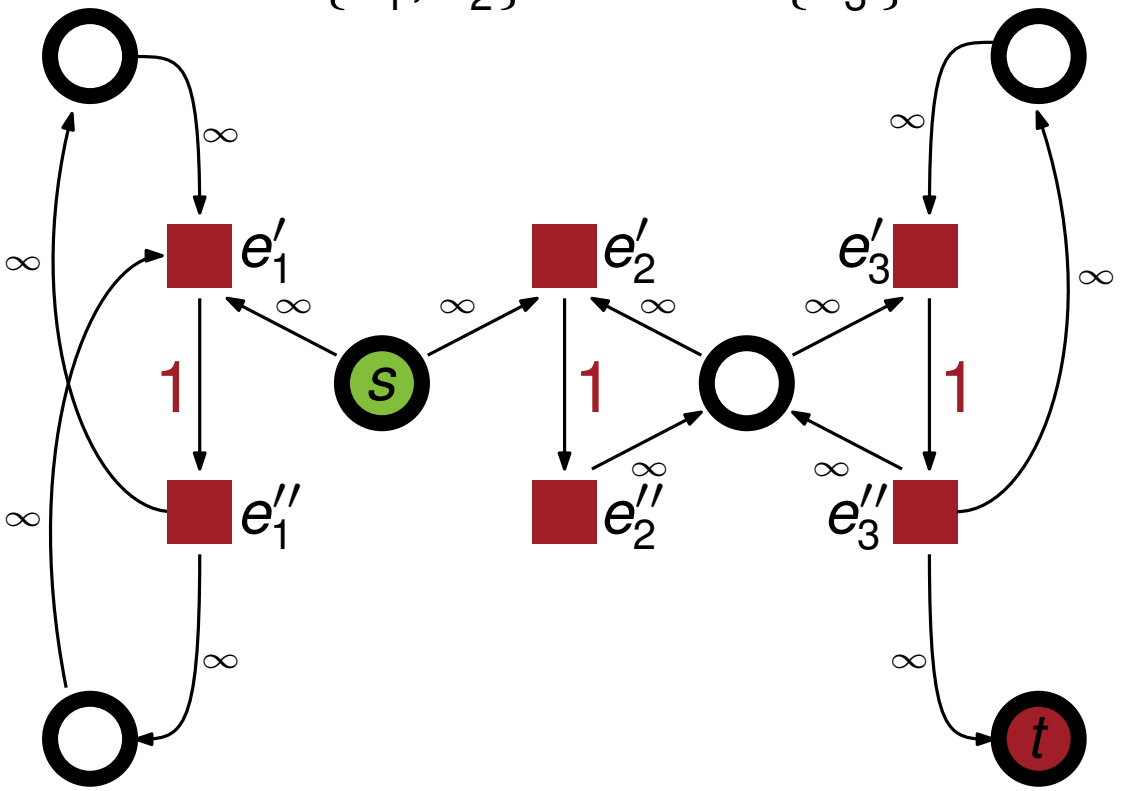


Lawler Network

Institute of Theoretical Informatics
Algorithmics Group

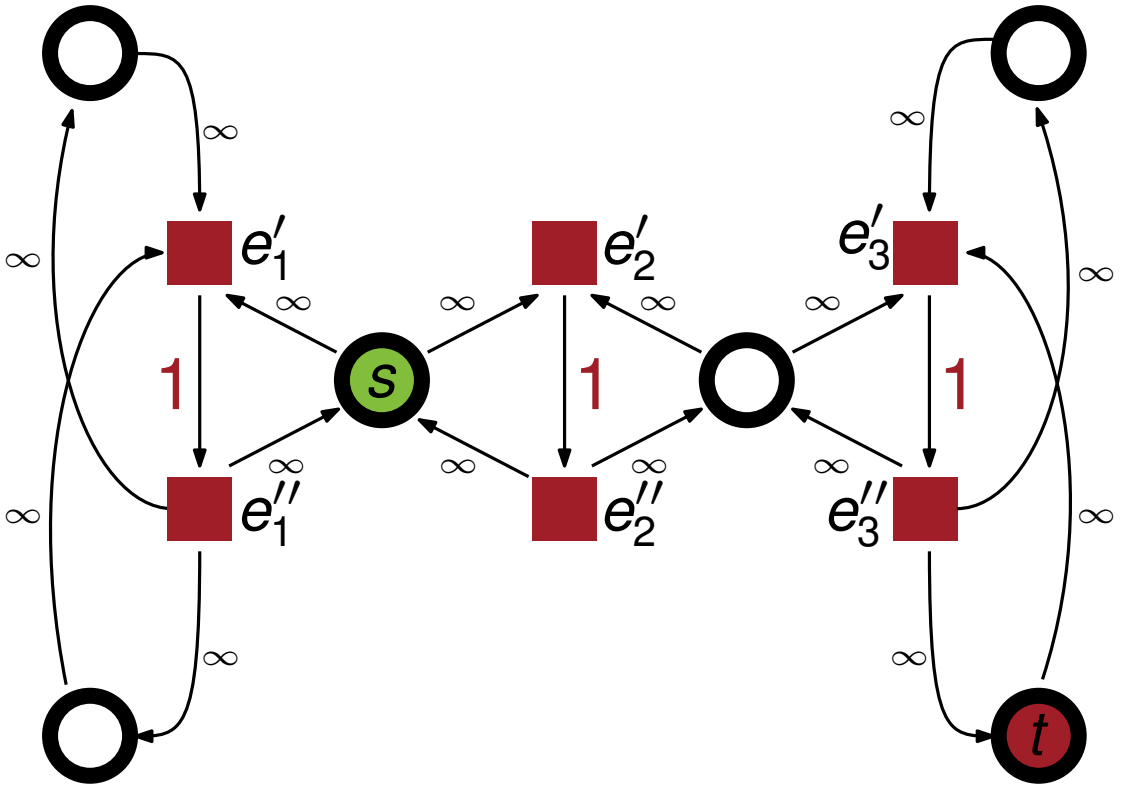
Removing Source and Sink Vertices

Corresponds to *Multi-Source Multi-Sink* problem with
$S = \{e'_1, e'_2\}$ and $T = \{e''_3\}$

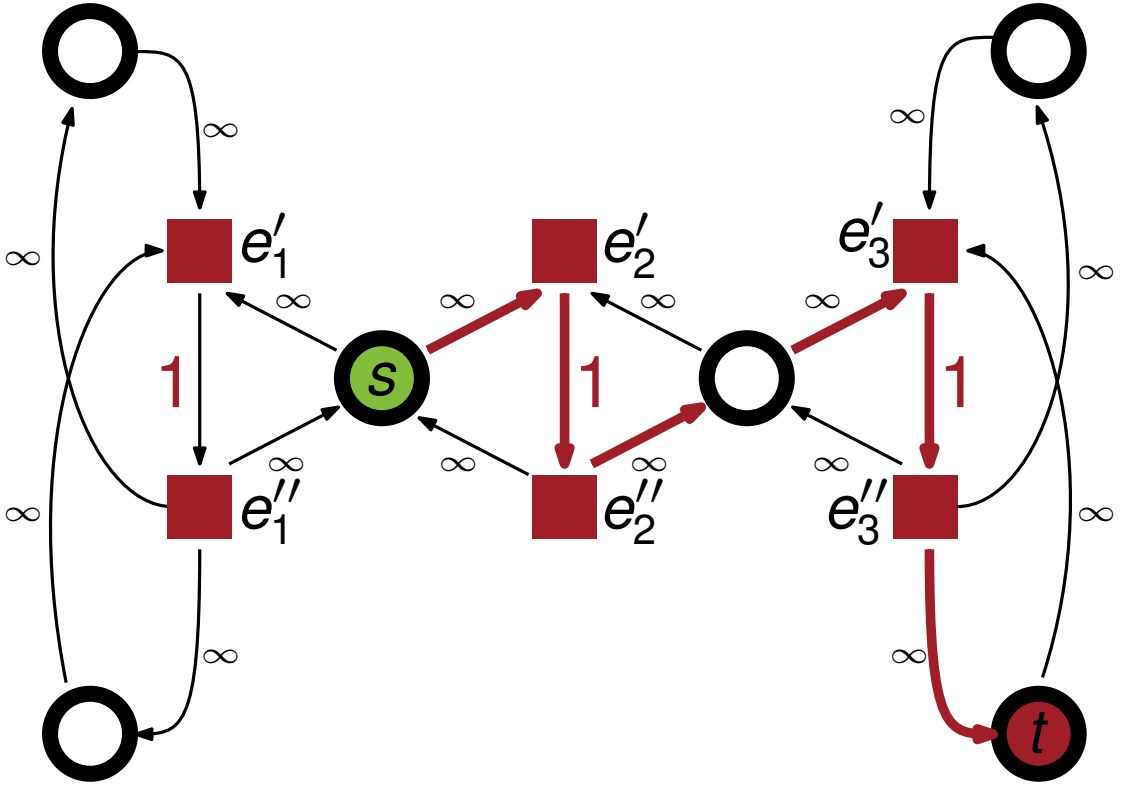


Lawler Network

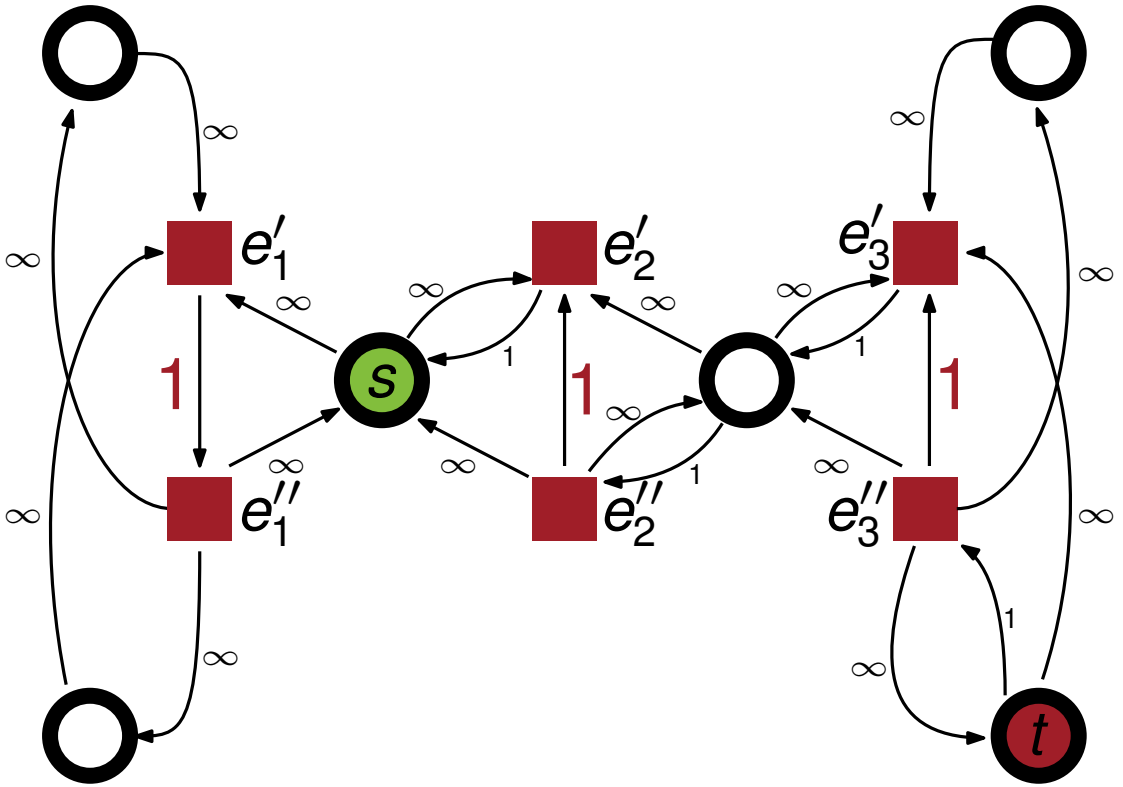
Minimum (s, t)-Bipartition



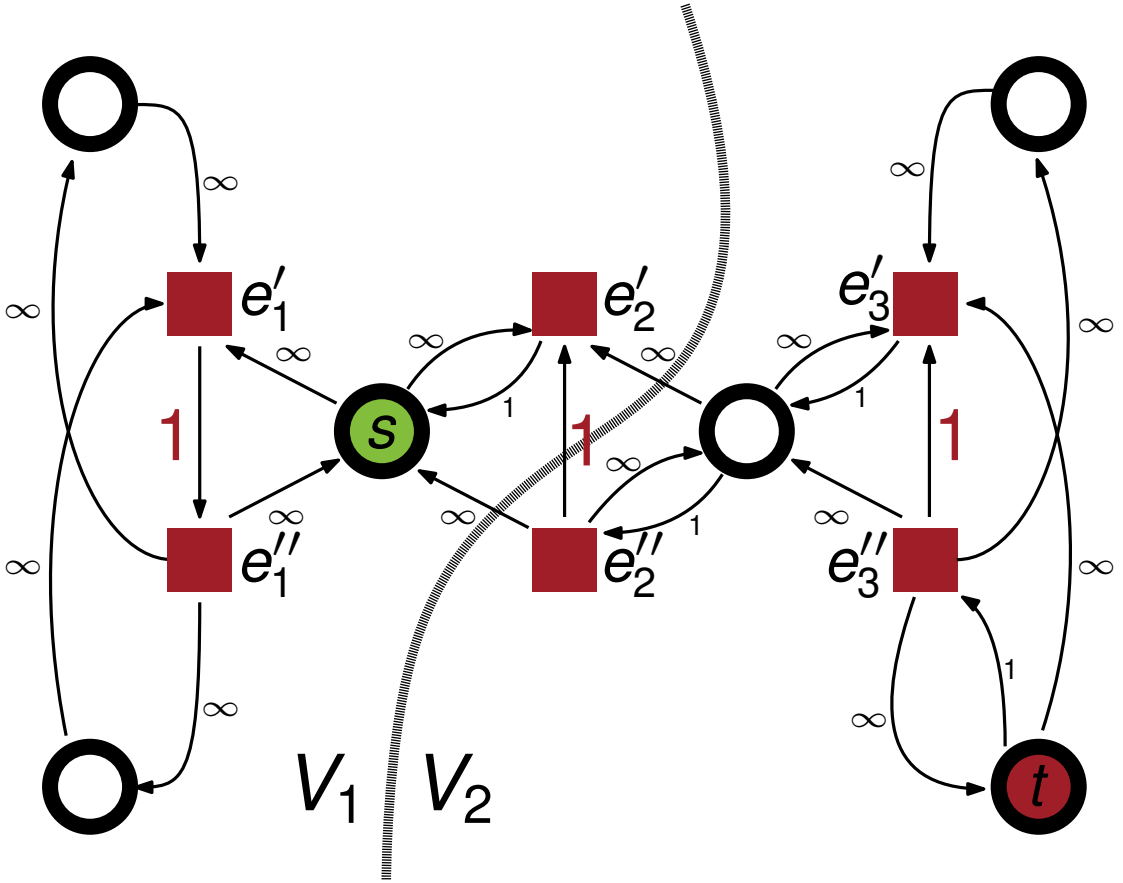
Minimum (s, t)-Bipartition



Minimum (s, t)-Bipartition



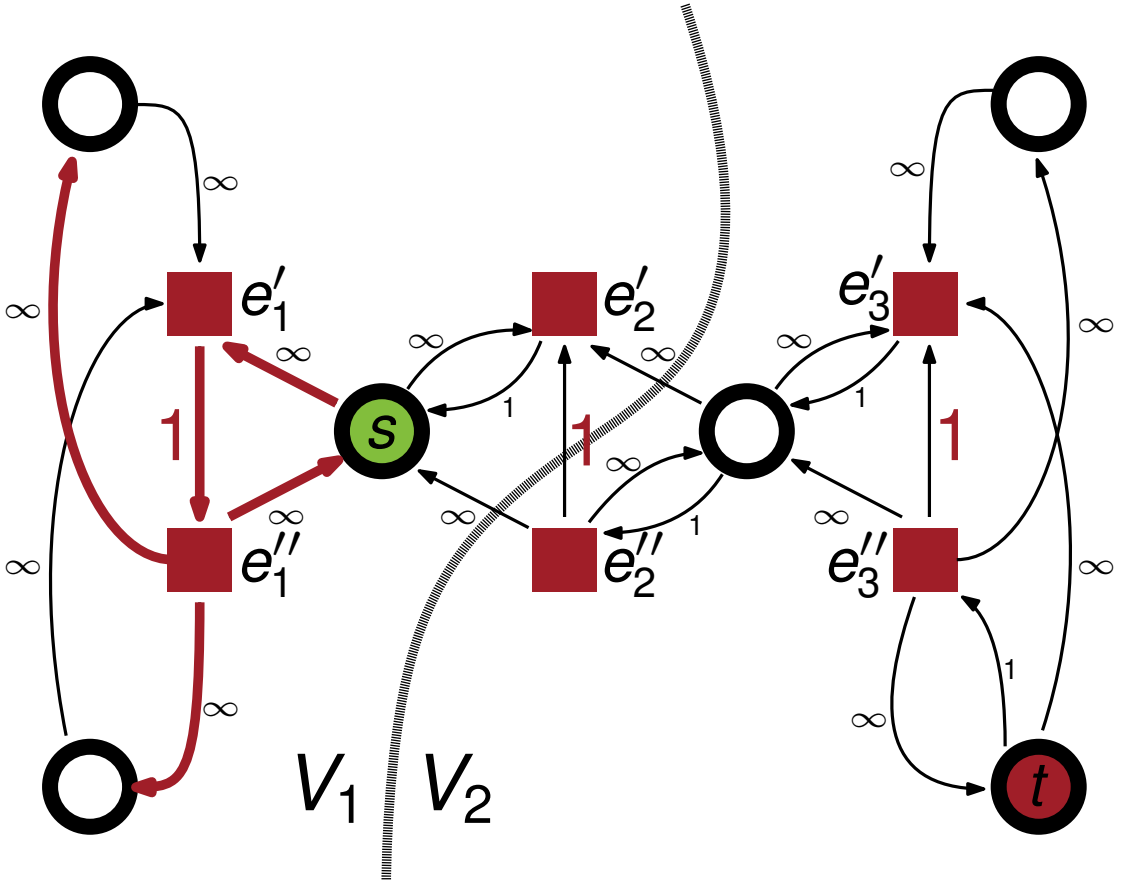
Minimum (s, t)-Bipartition



All nodes *reachable* from s are part of V_1 and $V_2 = V \setminus V_1$

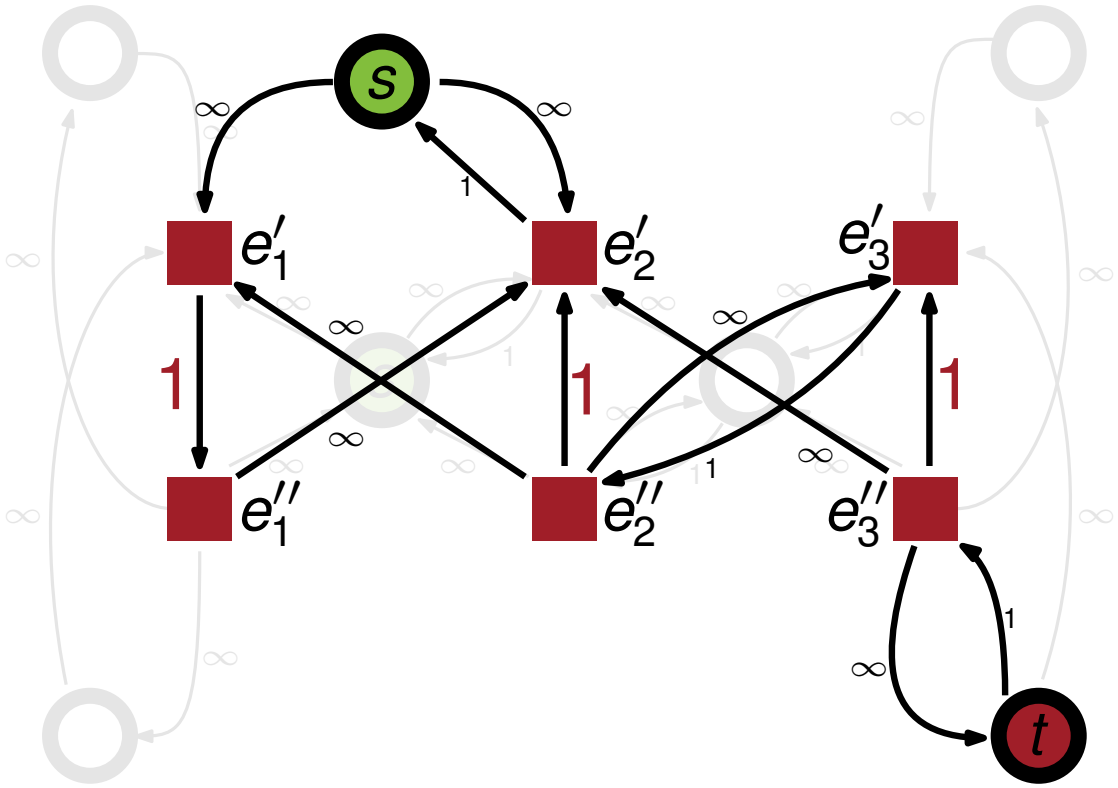
Minimum (s, t)-Bipartition

For each hypernode $v \in V_1$, there exists at least one $e \in I(v)$ with $e'' \in V_1$



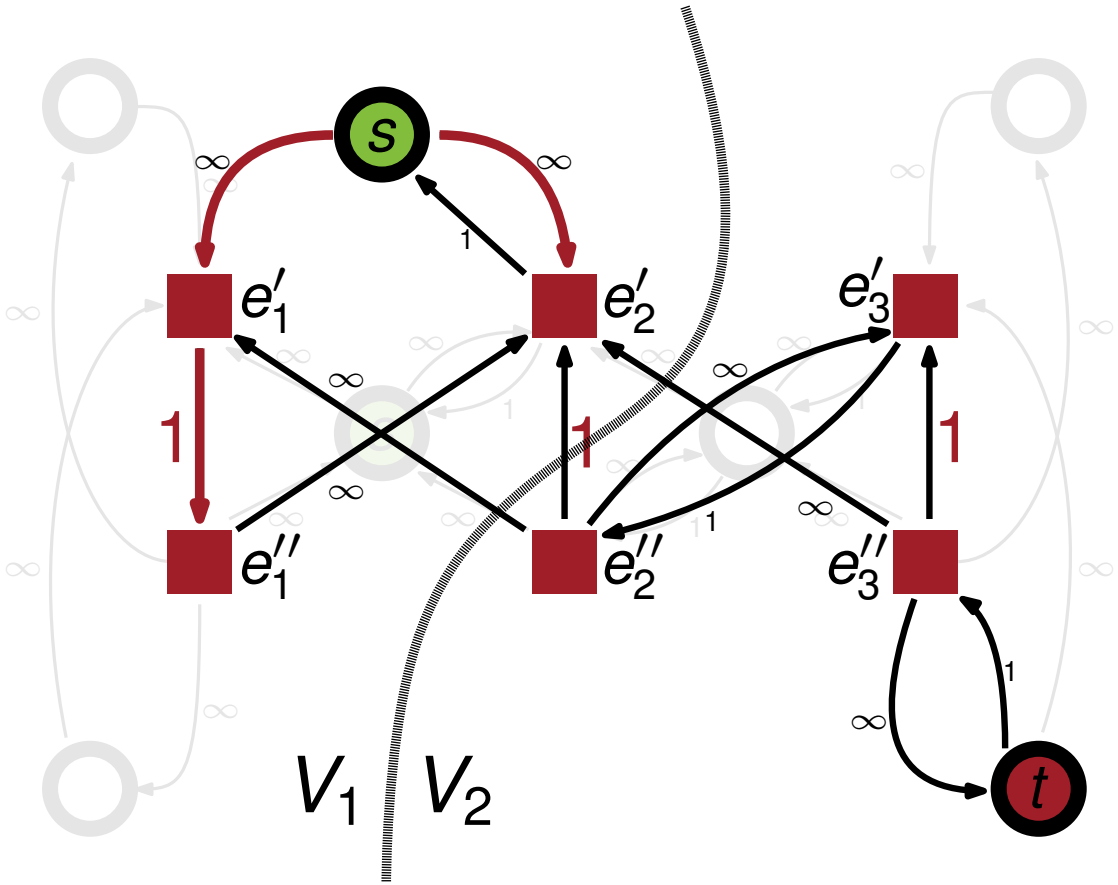
All nodes *reachable* from s are part of V_1 and $V_2 = V \setminus V_1$

Minimum (s, t)-Bipartition



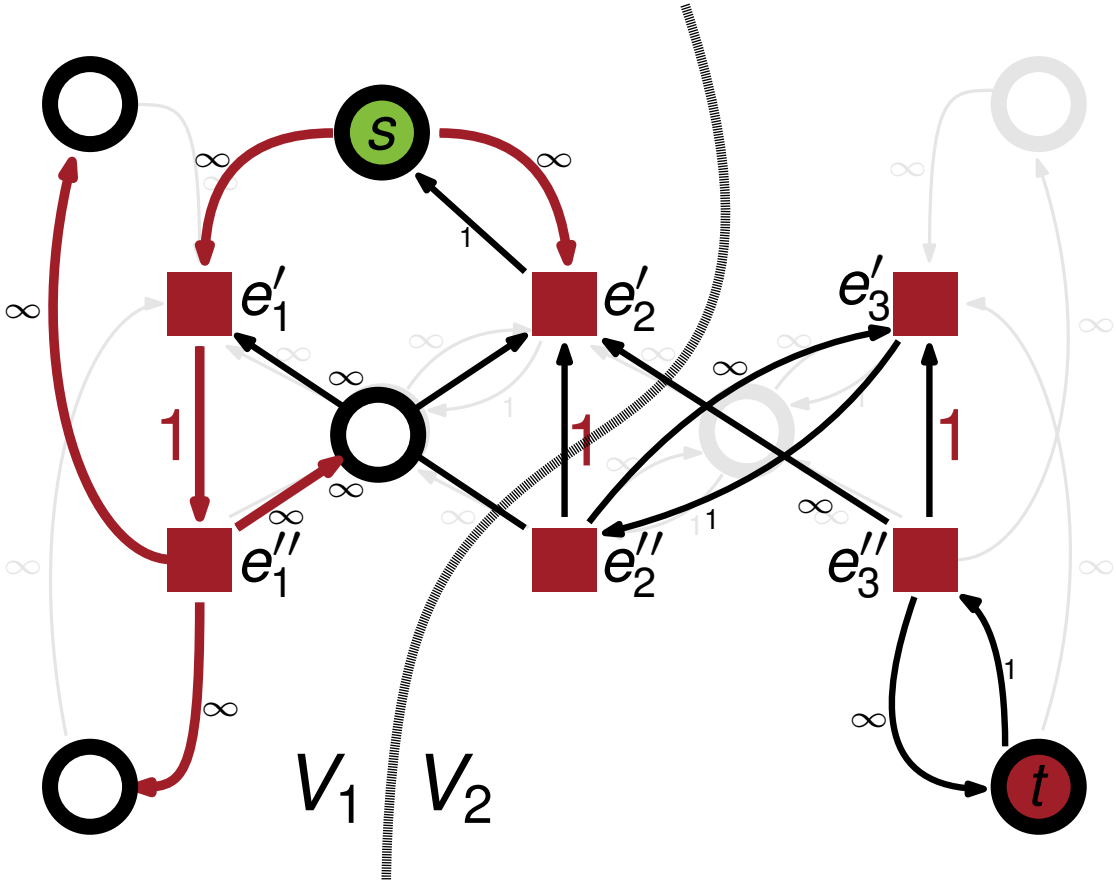
Institute of Theoretical Informatics
Algorithmics Group

Minimum (s, t)-Bipartition



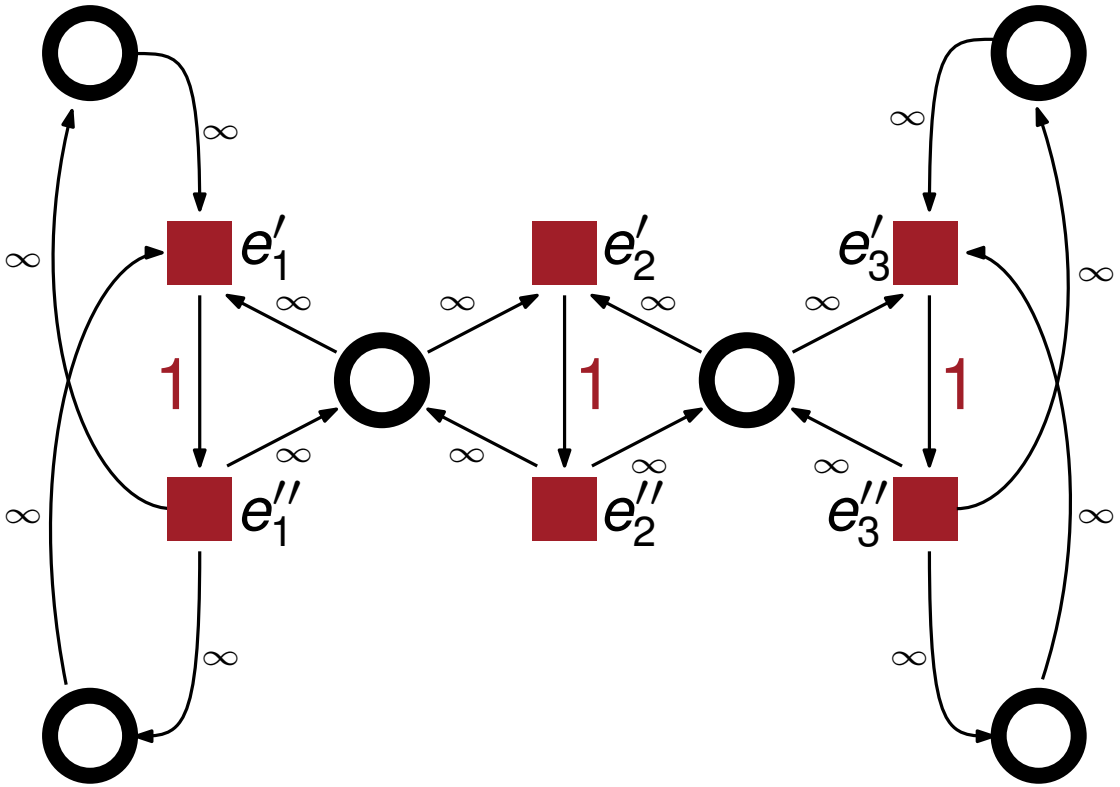
Institute of Theoretical Informatics
Algorithmics Group

Minimum (s, t)-Bipartition



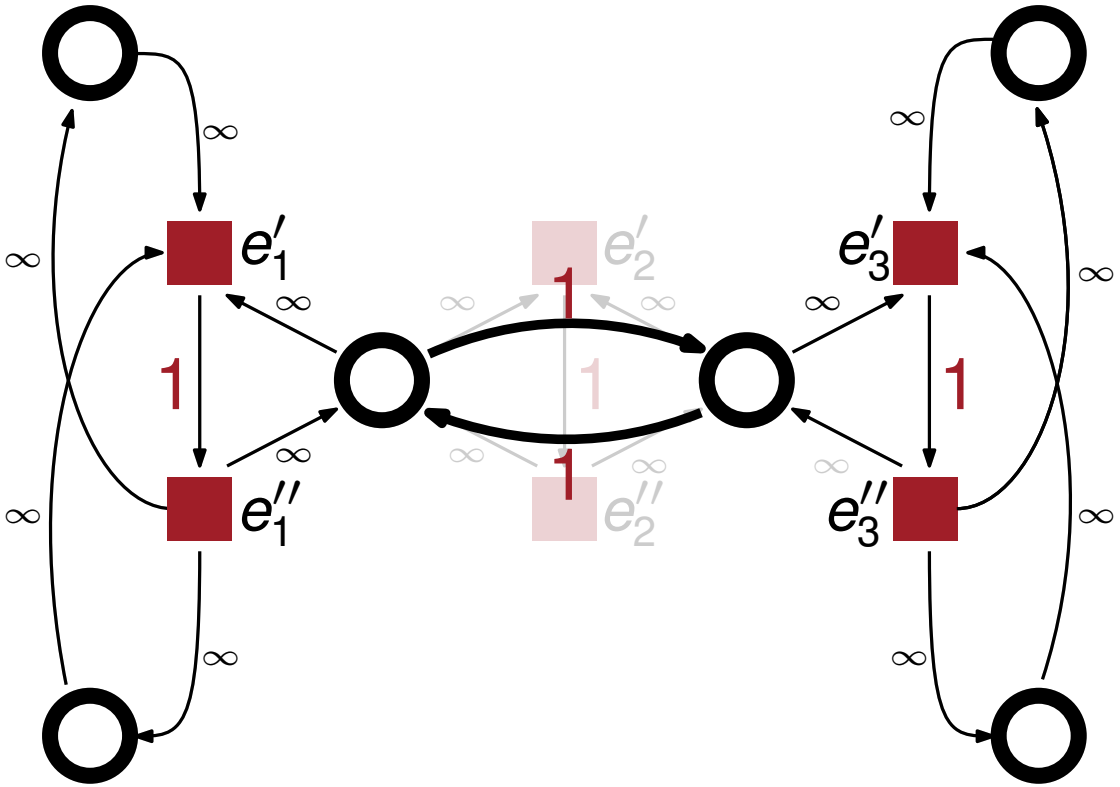
Institute of Theoretical Informatics
Algorithmics Group

Hypergraph Flow Network - Summary



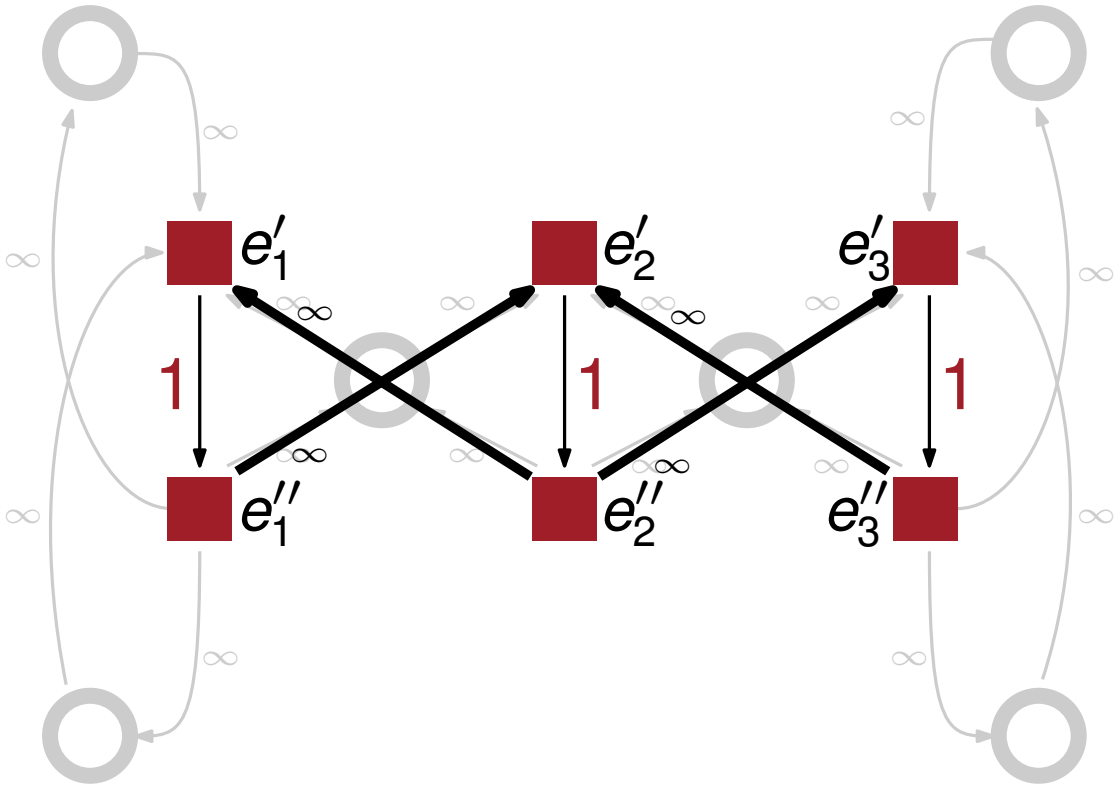
Lawler Network

Hypergraph Flow Network - Summary



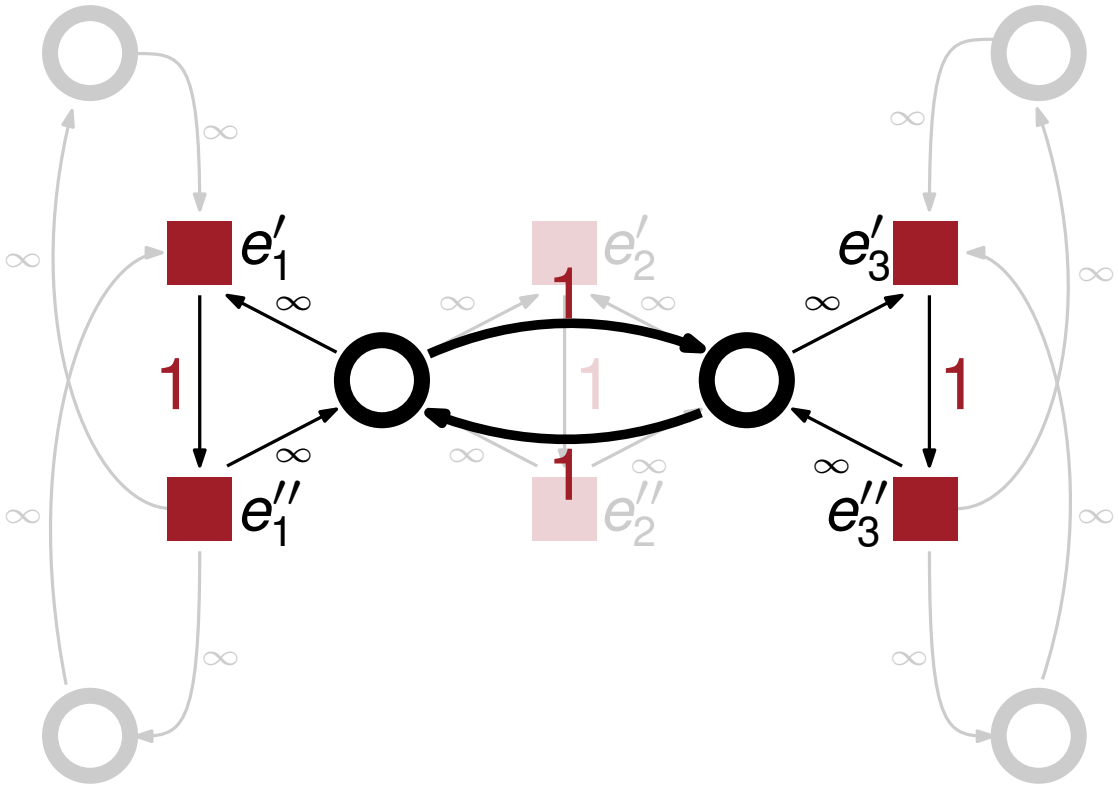
Wong Network

Hypergraph Flow Network - Summary



Our Network

Hypergraph Flow Network - Summary

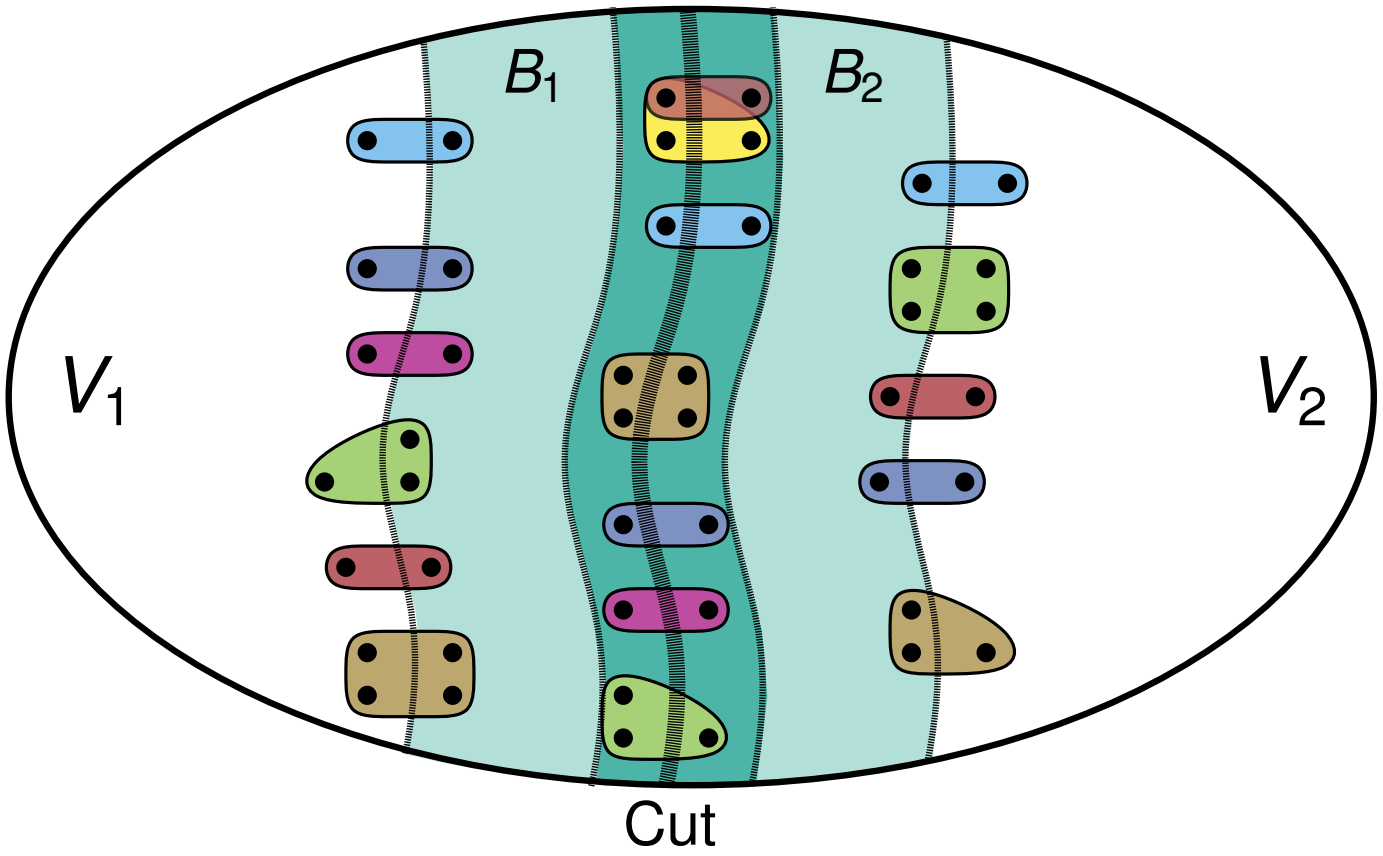


Hybrid Network

Institute of Theoretical Informatics
Algorithmics Group

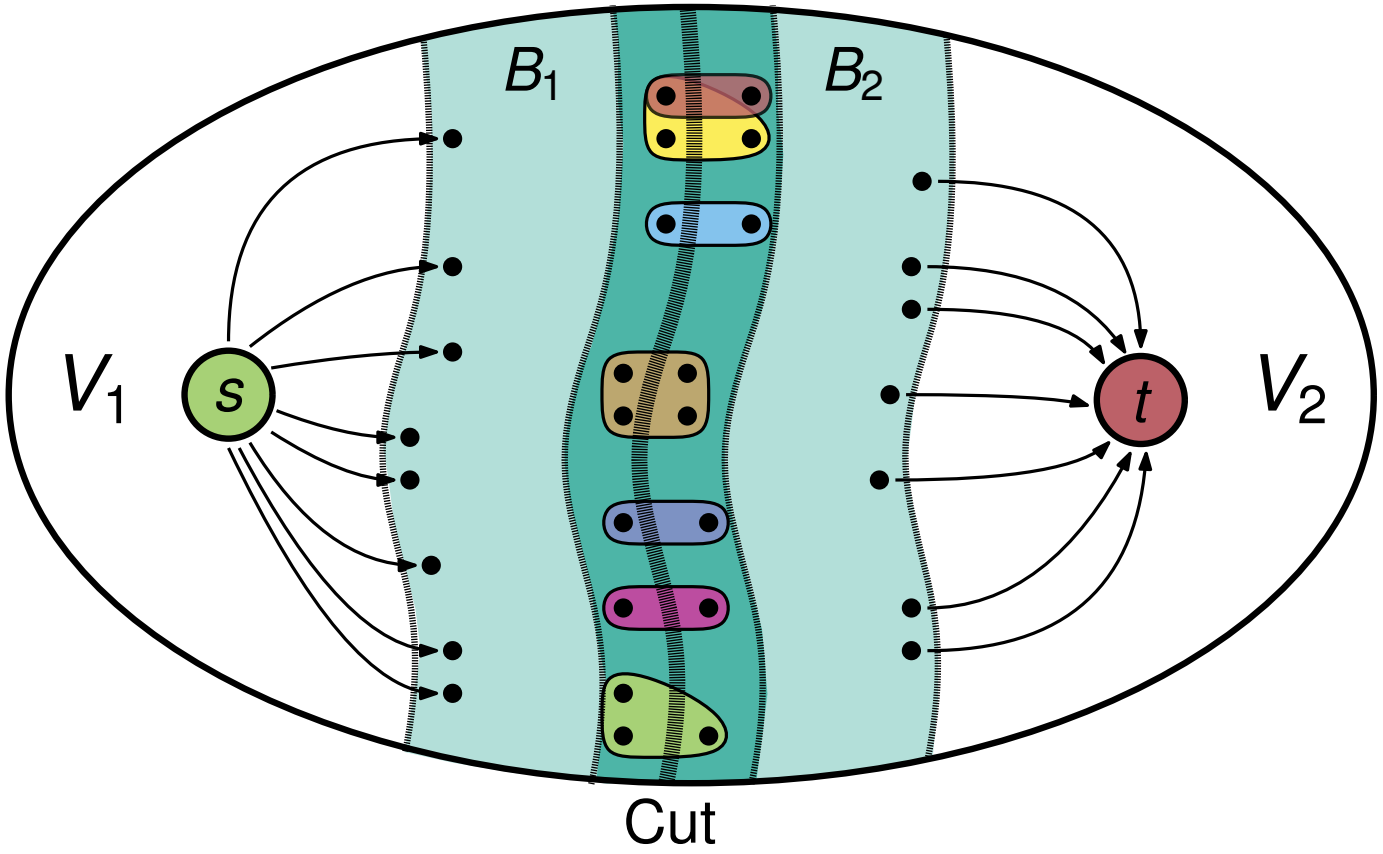
Optimized Flow Problem Modeling Approach

Modeling Approach in *KaFFPa*



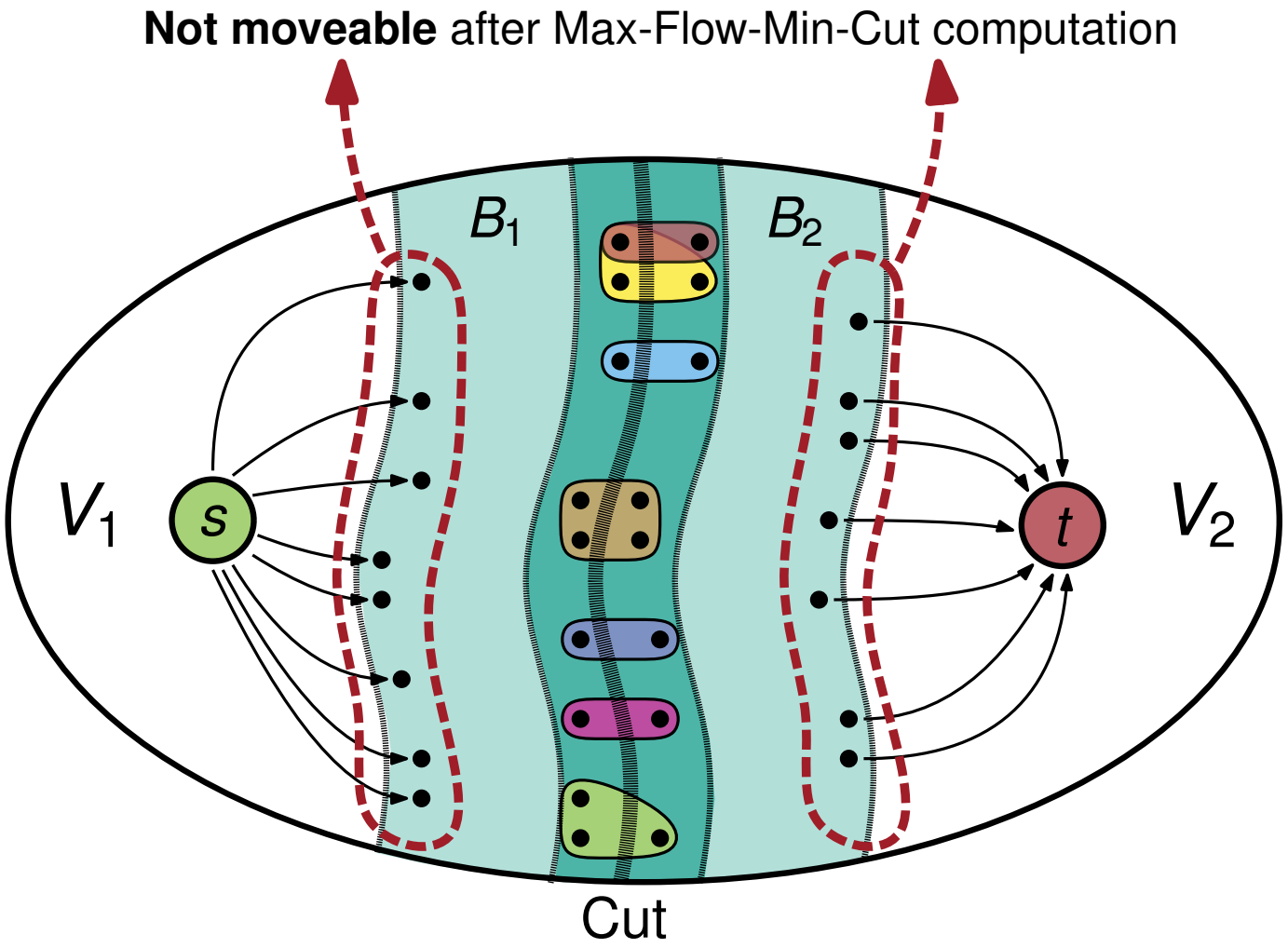
Optimized Flow Problem Modeling Approach

Modeling Approach in *KaFFPa*



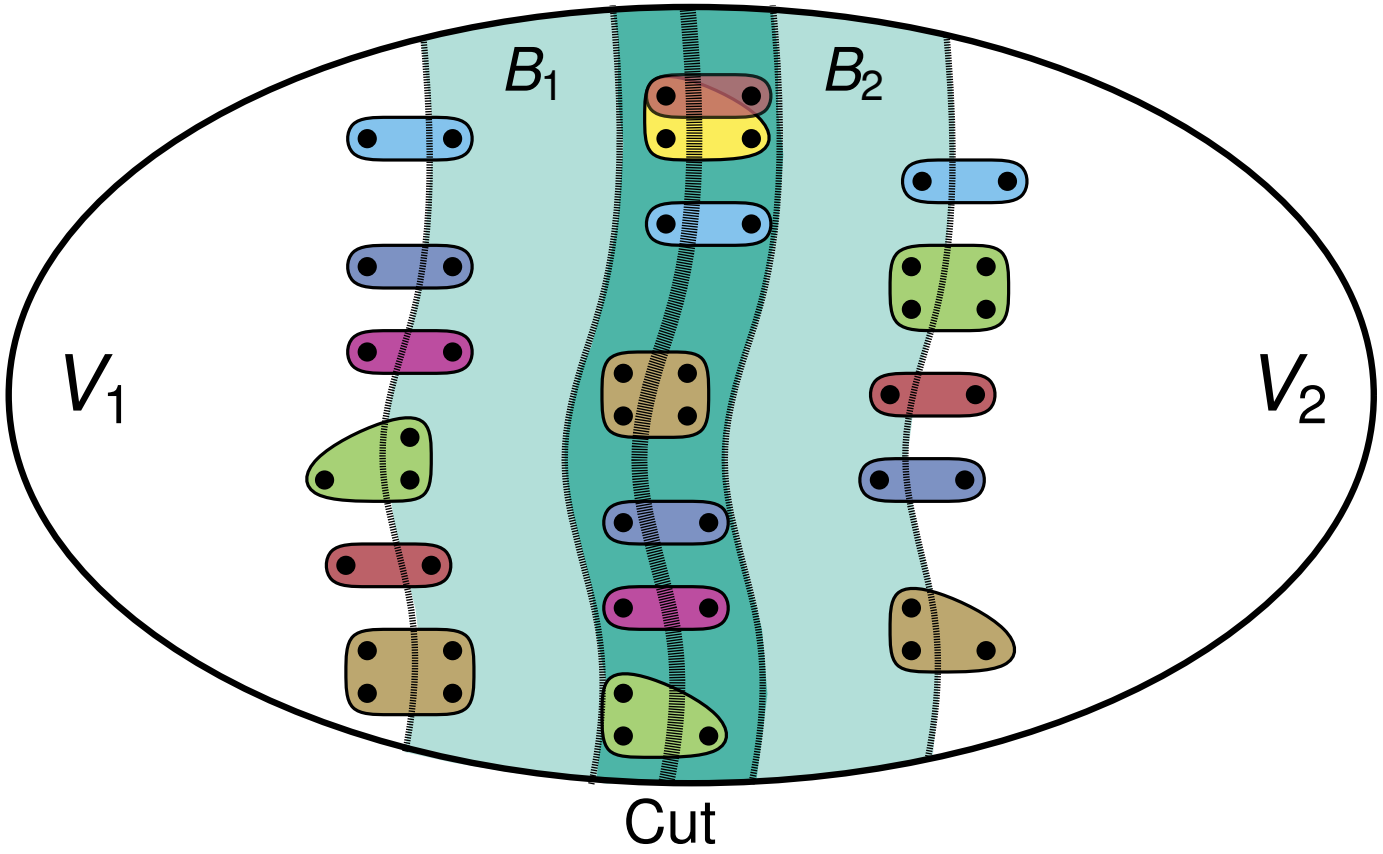
Optimized Flow Problem Modeling Approach

Modeling Approach in *KaFFPa*



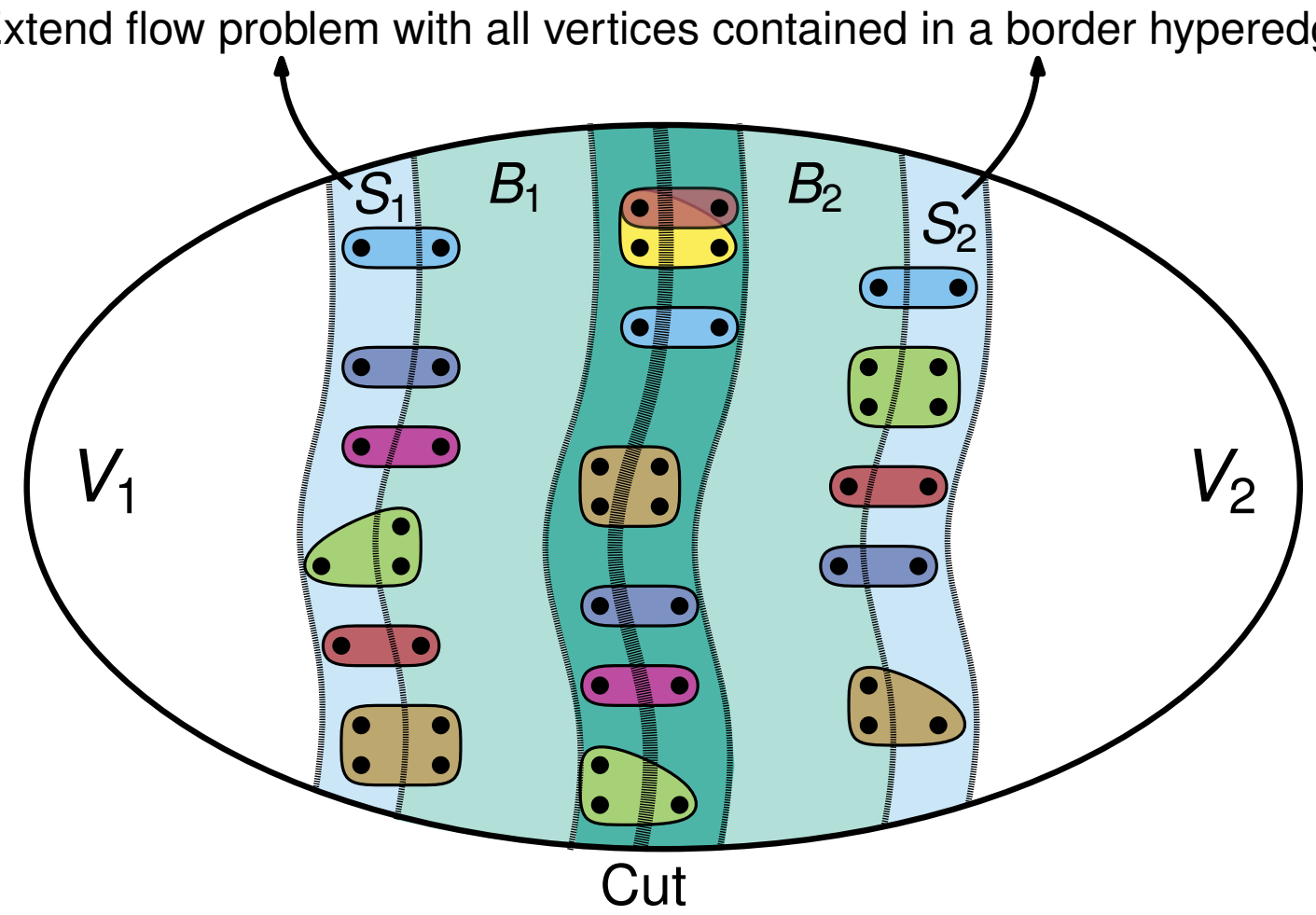
Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*



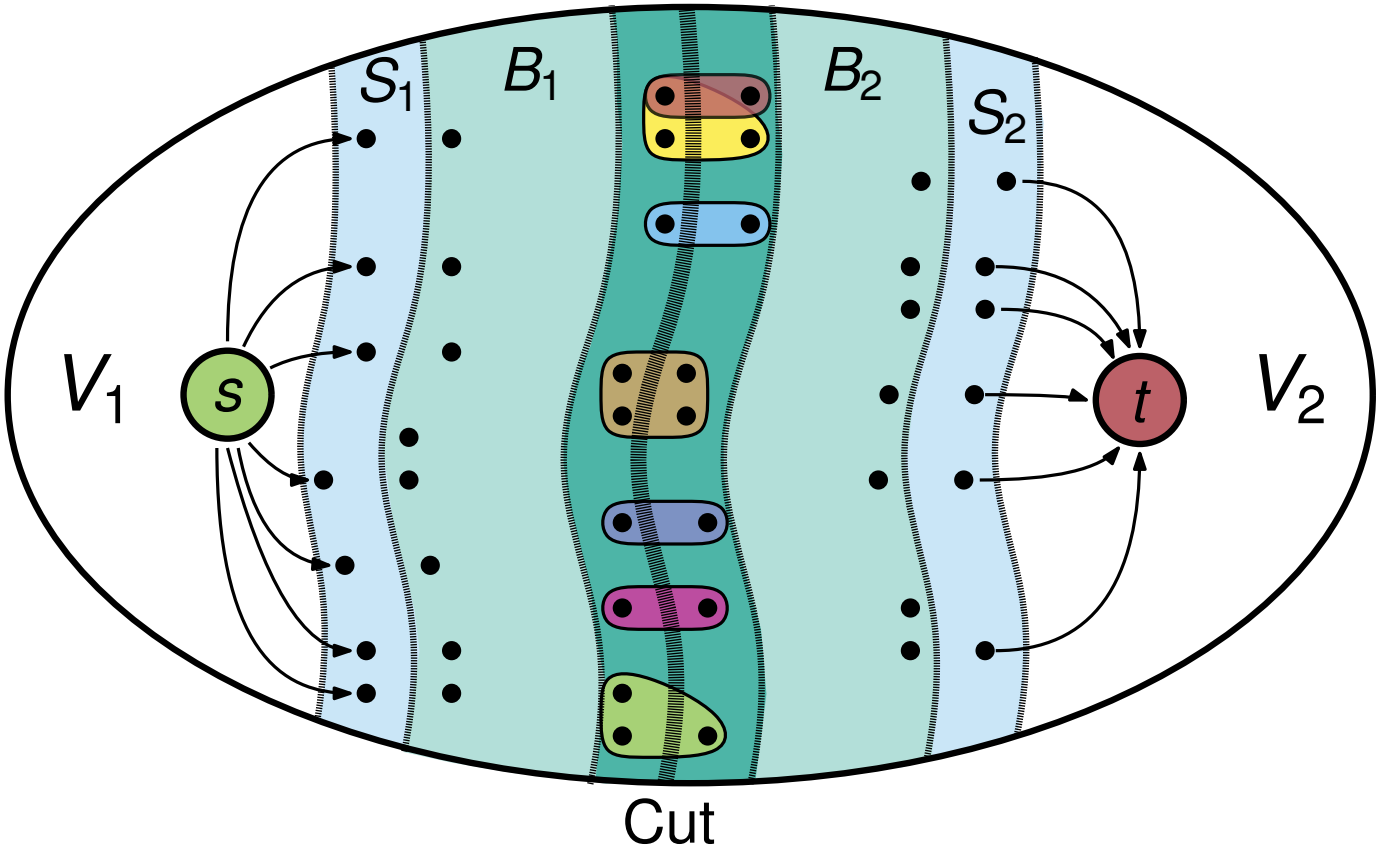
Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*



Optimized Flow Problem Modeling Approach

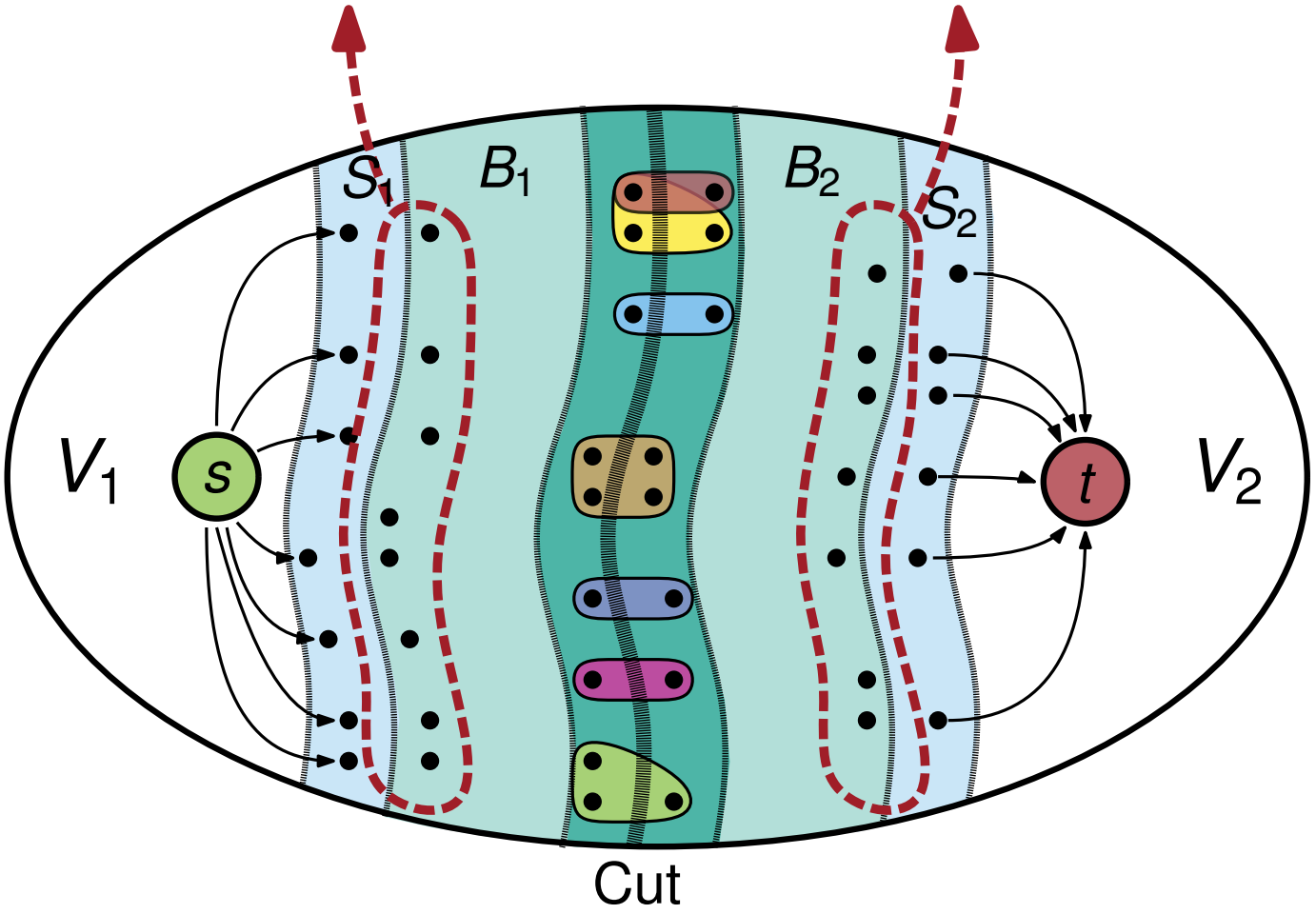
Modeling Approach in *KaHyPar*



Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*

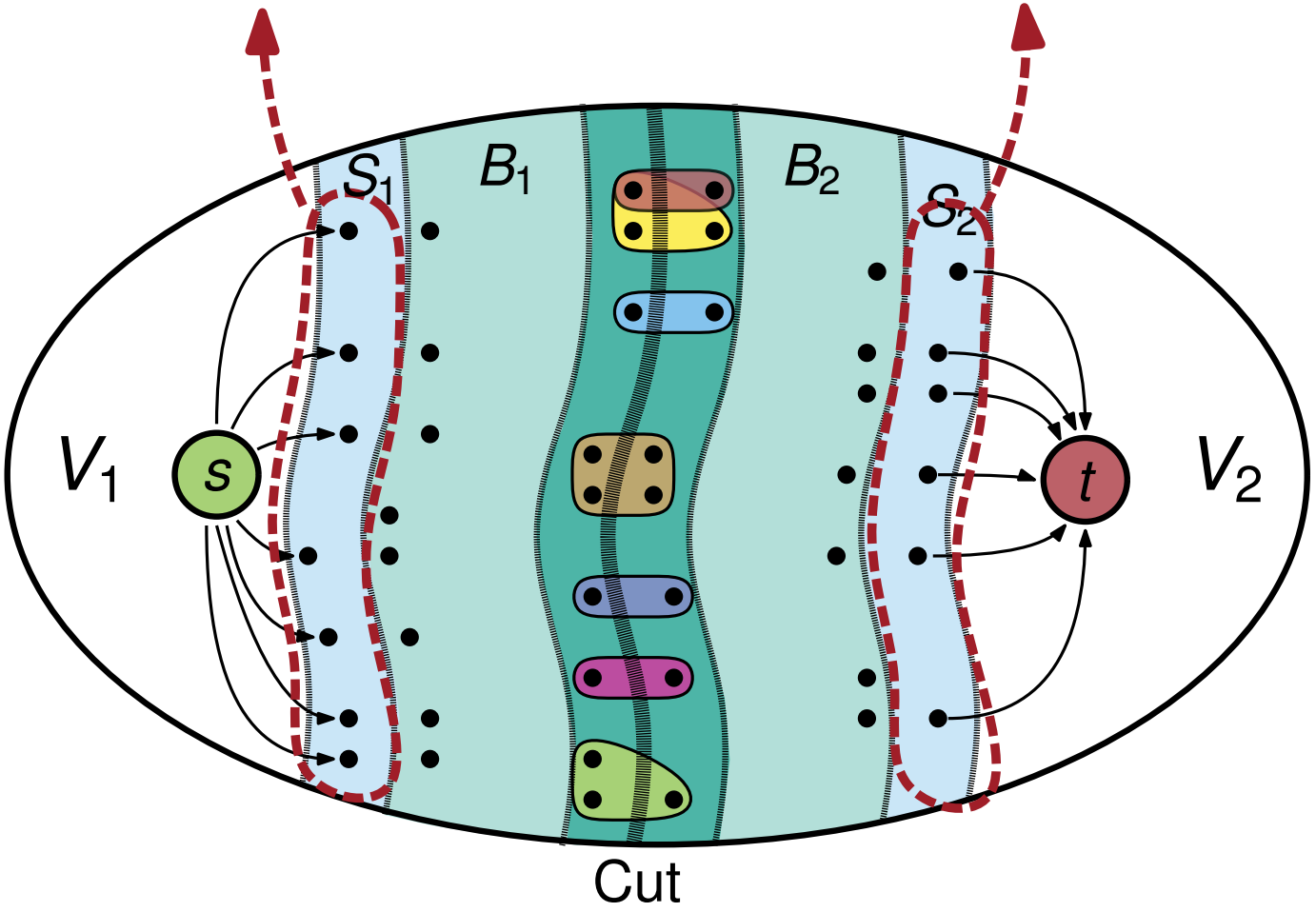
Moveable after Max-Flow-Min-Cut computation, but . . .



Optimized Flow Problem Modeling Approach

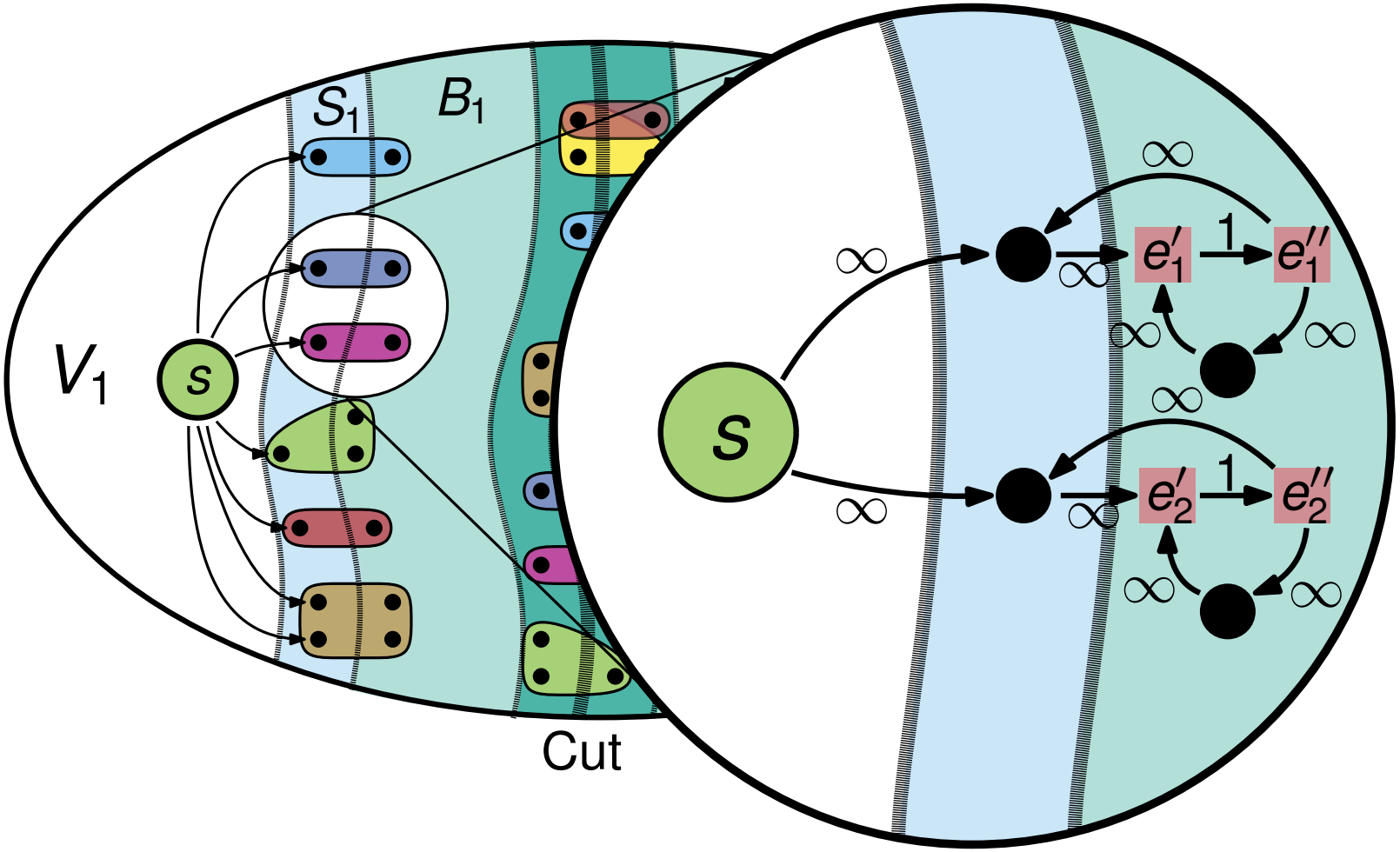
Modeling Approach in *KaHyPar*

. . . flow problem has significantly more nodes and edges.



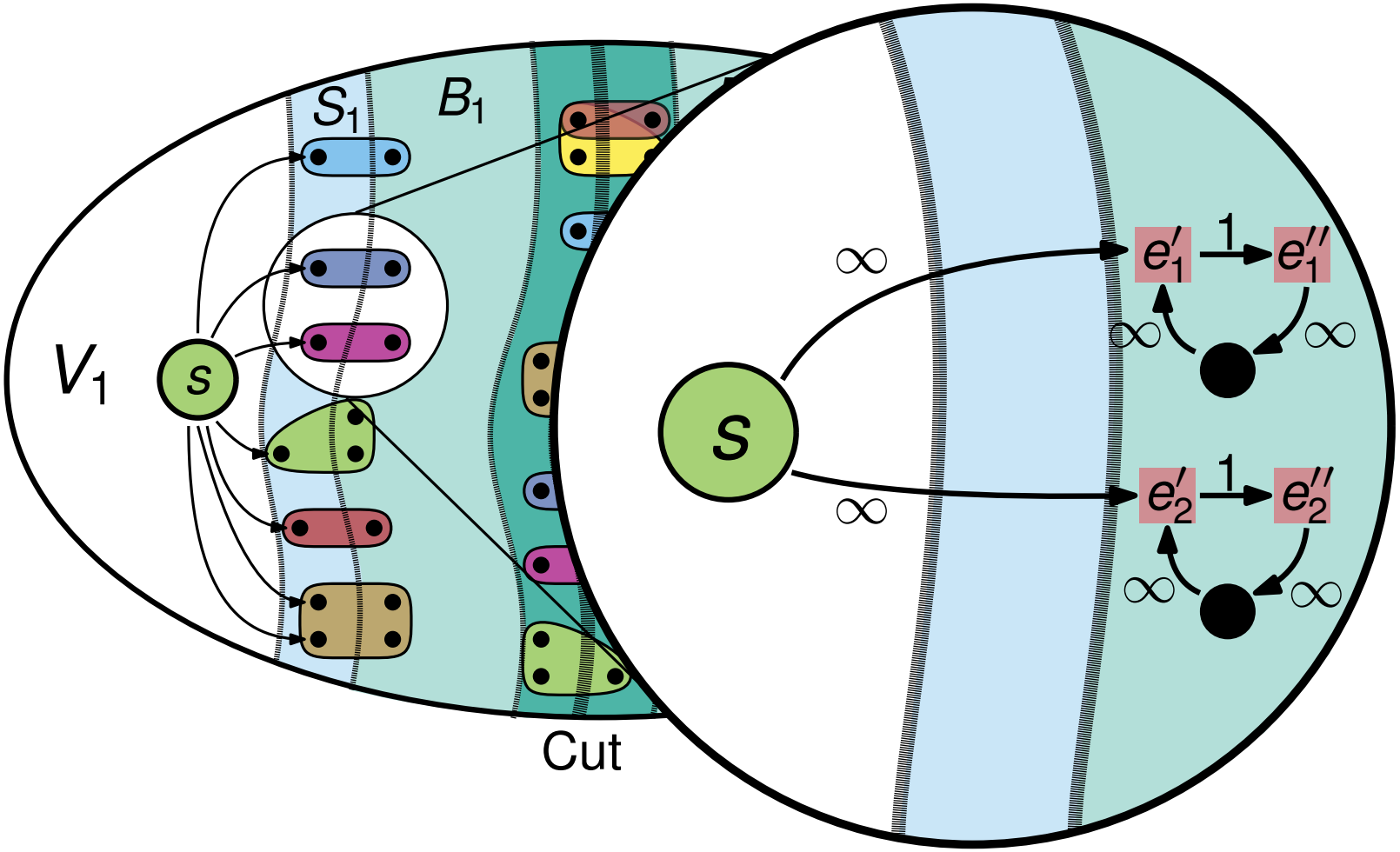
Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*



Optimized Flow Problem Modeling Approach

Modeling Approach in *KaHyPar*

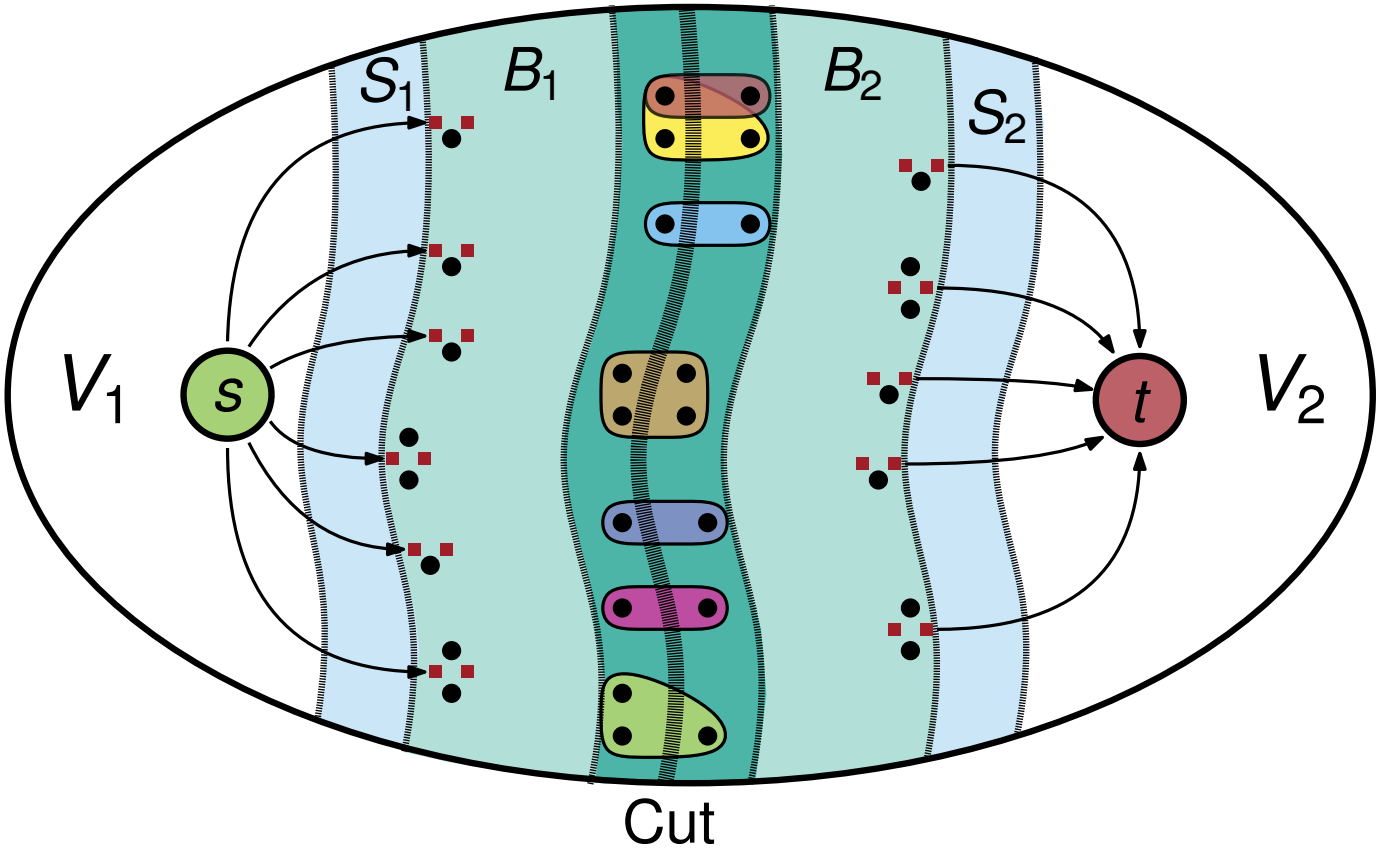


Optimized Flow Problem Modeling Approach

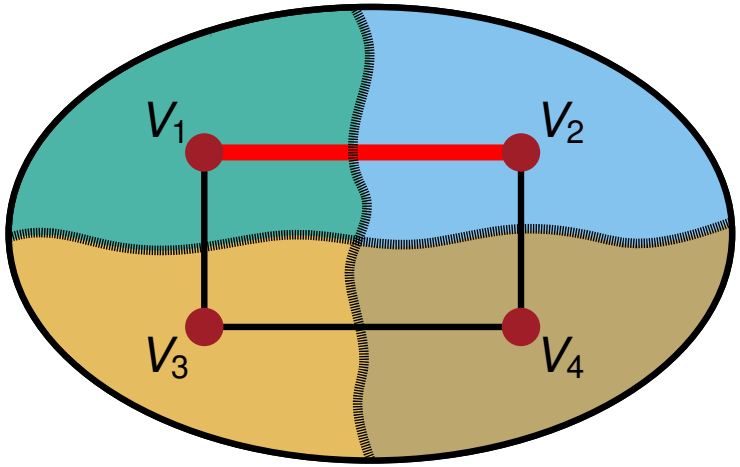
Modeling Approach in *KaHyPar*

$$S = \{e' \mid e \in I(S_1)\}$$

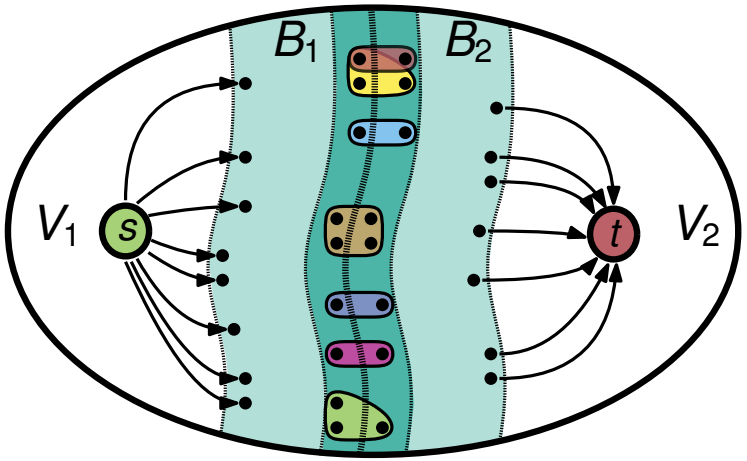
$$T = \{e'' \mid e \in I(S_2)\}$$



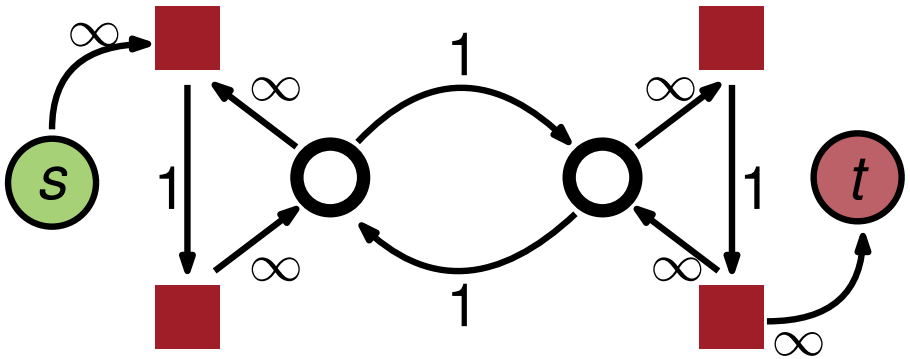
Our Flow-Based Refinement Framework



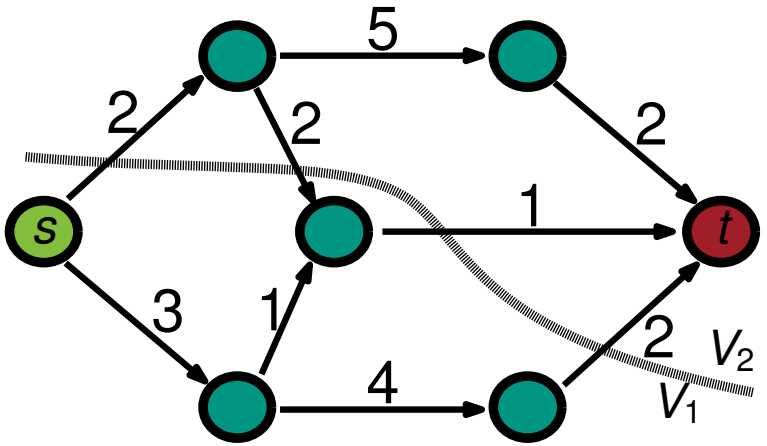
Select two adjacent blocks for refinement



Build Flow Problem



Solve Flow Problem



Find feasible minimum cut

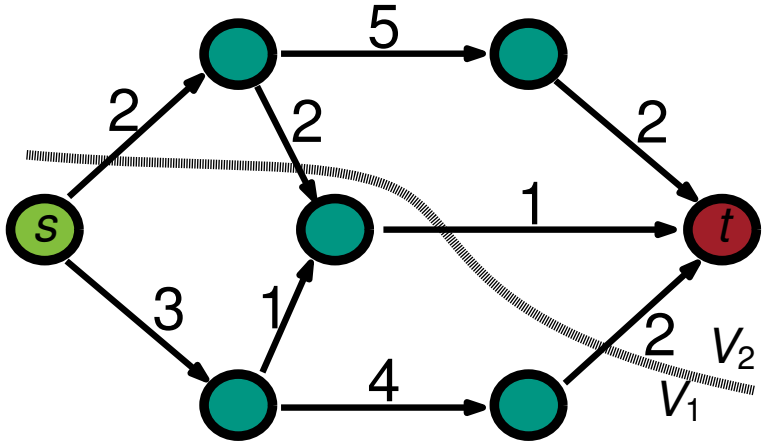
Institute of Theoretical Informatics
Algorithmics Group

Our Flow-Based Refinement Framework

Select two adjacent blocks for refinement

Build Flow Problem

Solve Flow Problem



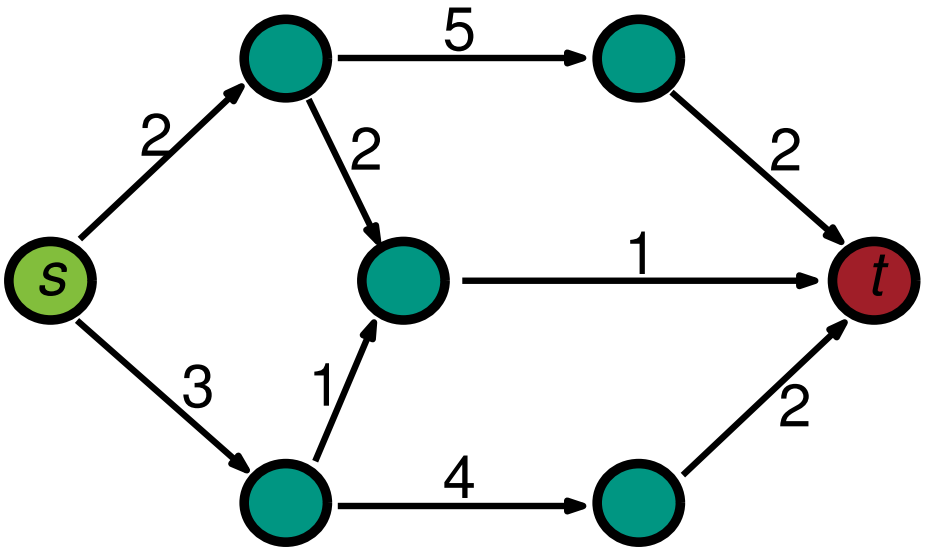
Find feasible minimum cut

Institute of Theoretical Informatics
Algorithmics Group

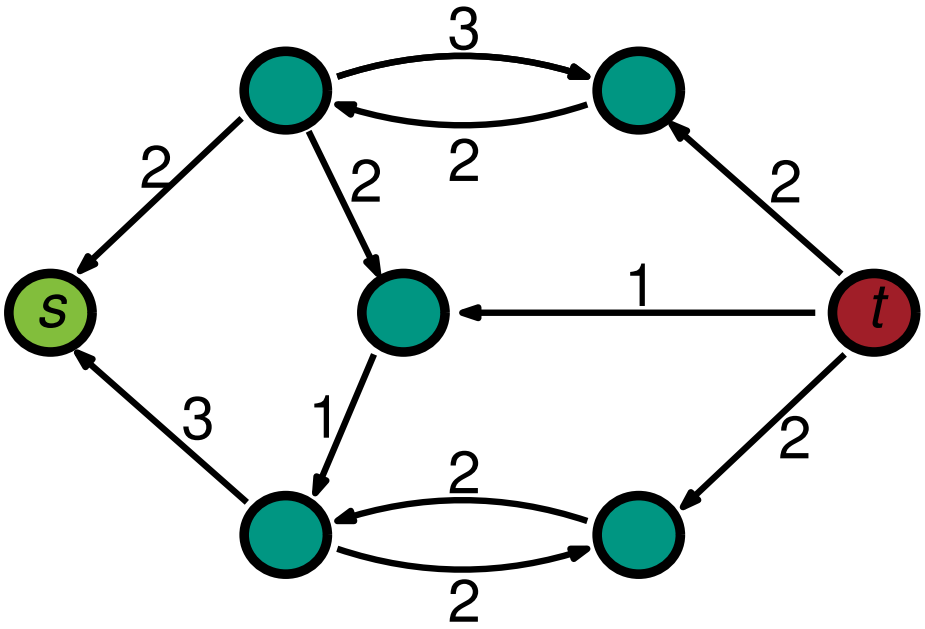
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t)-cuts

Flow Graph



Residual Graph

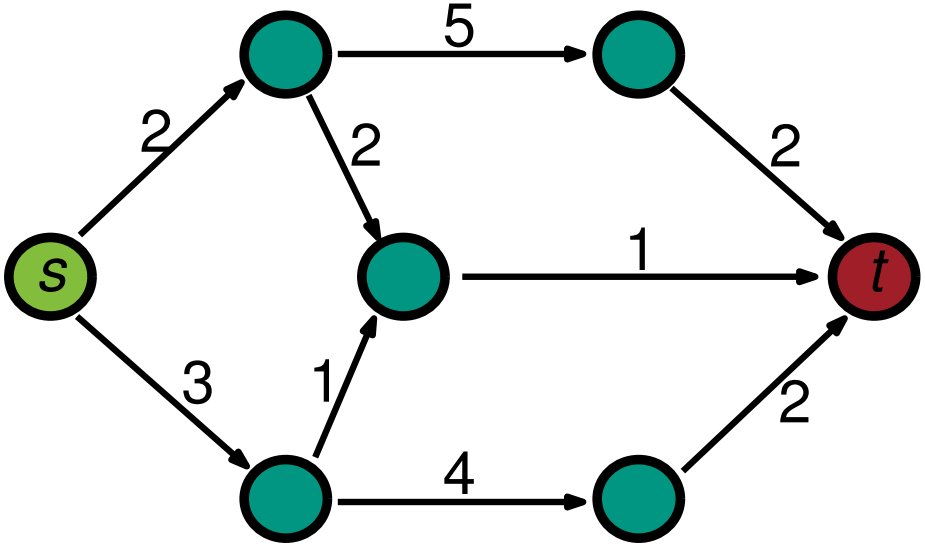


Institute of Theoretical Informatics
Algorithmics Group

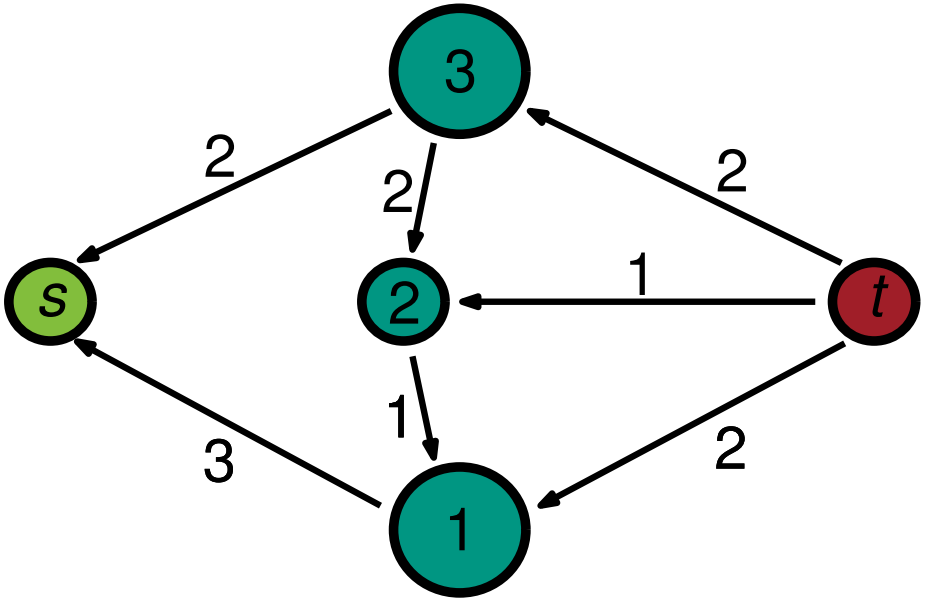
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t)-cuts

Flow Graph



Picard-Queryanne DAC



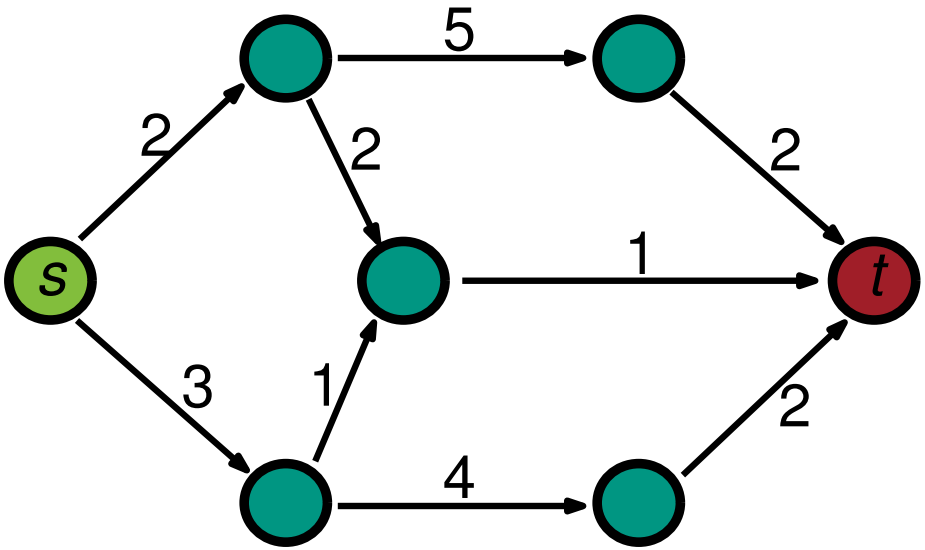
Contract all *strongly connected components* in the residual graph

Institute of Theoretical Informatics
Algorithmics Group

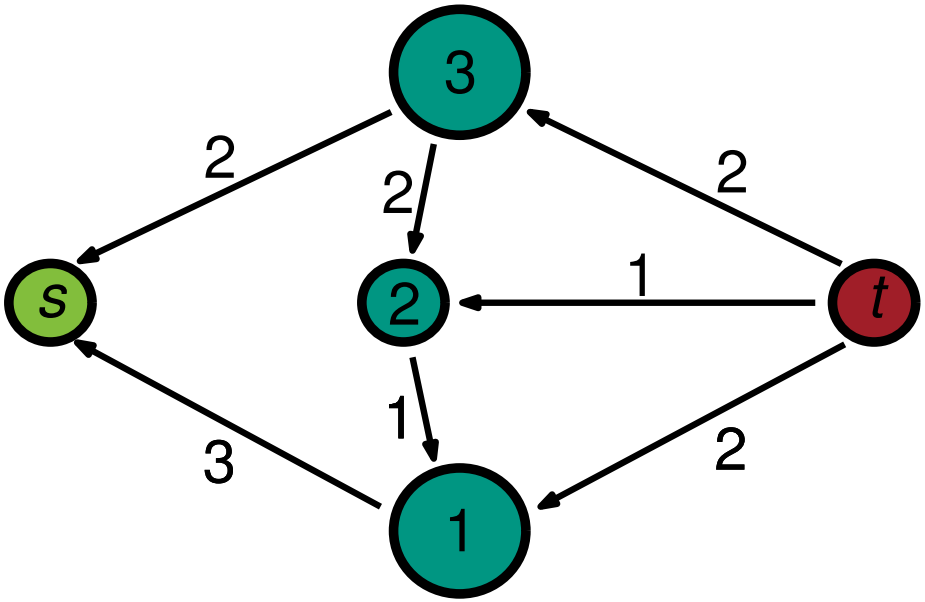
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t)-cuts

Flow Graph



Picard-Queryanne DAC



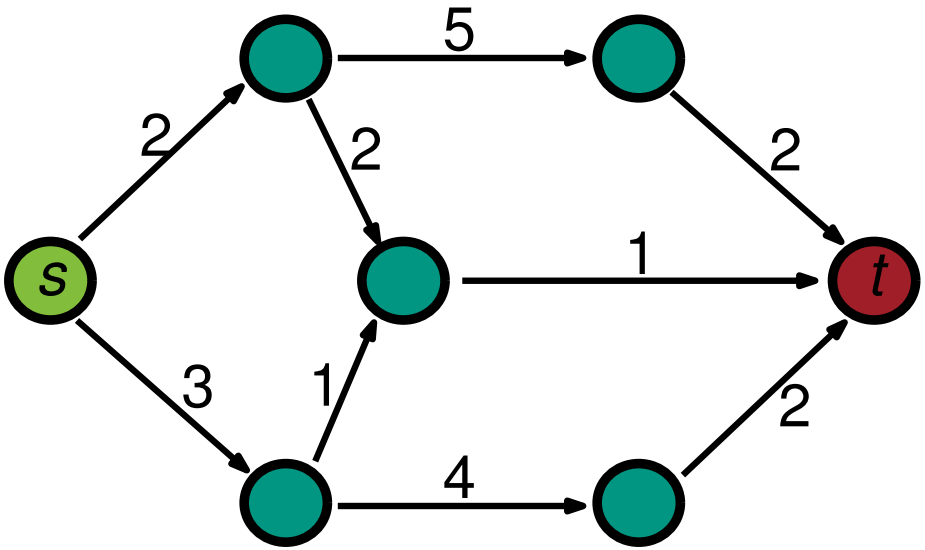
$\langle t, \quad 3, \quad 2, \quad 1, \quad s \rangle$

Find *topological order*

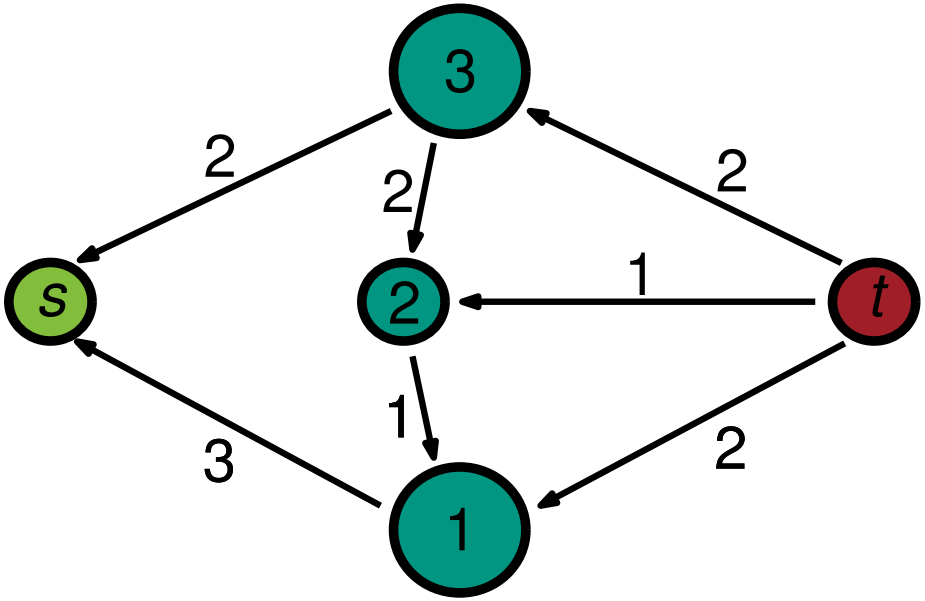
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t)-cuts

Flow Graph



Picard-Queryanne DAC



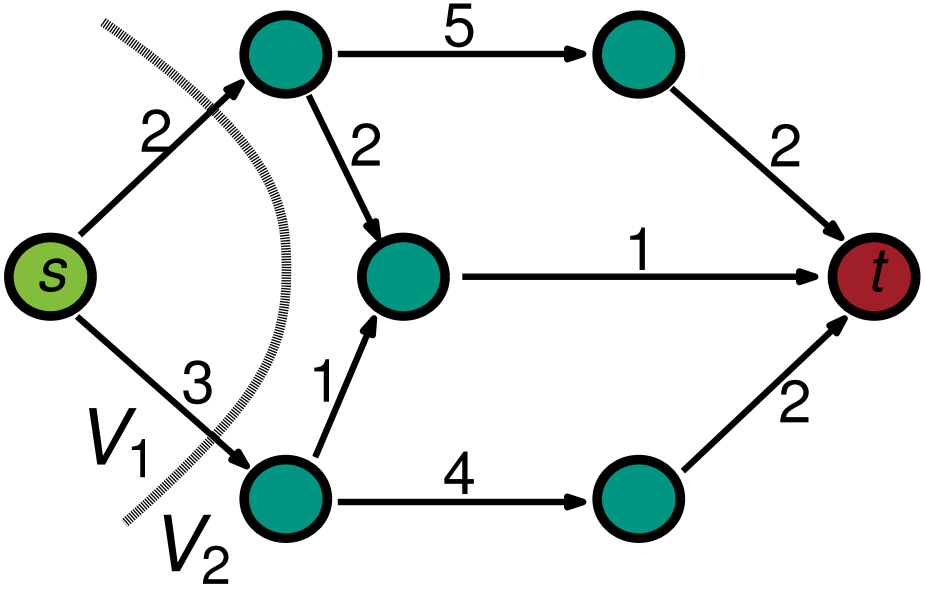
$\langle t, \quad 3, \quad 2, \quad 1, \quad s \rangle$

Sweep through **reverse** topological order

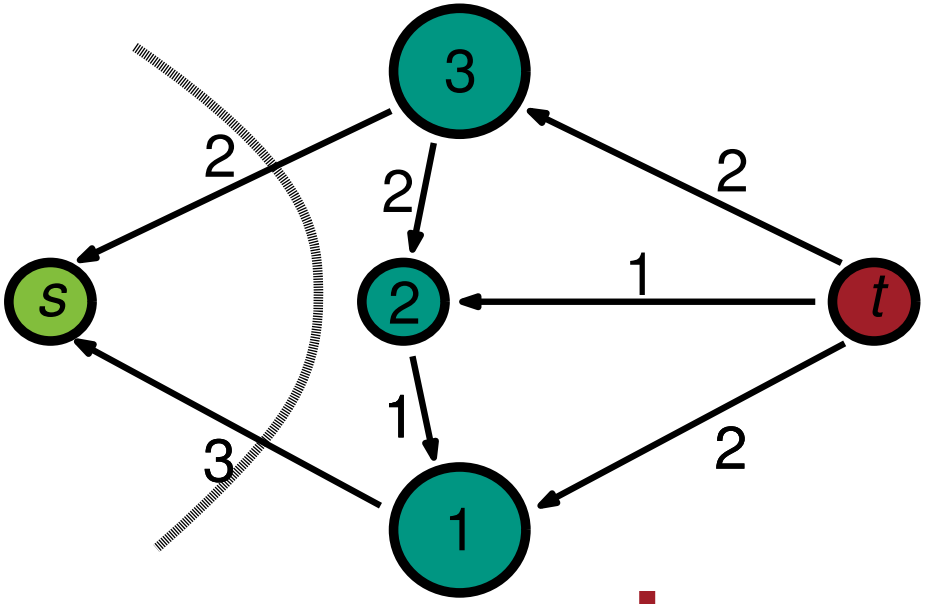
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t)-cuts

Flow Graph



Picard-Queryanne DAC



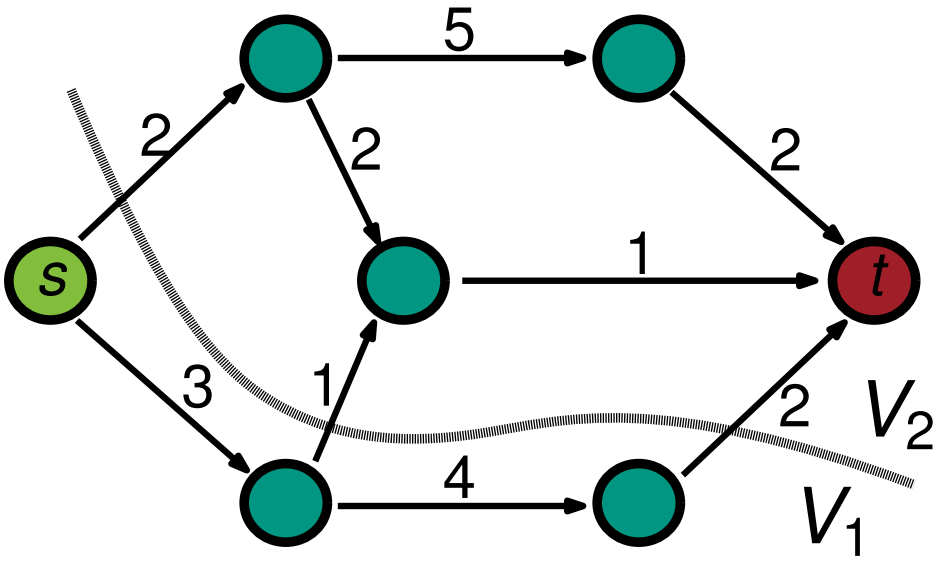
$\langle t, \quad 3, \quad 2, \quad 1, \quad | \quad s \rangle$

Sweep through **reverse** topological order

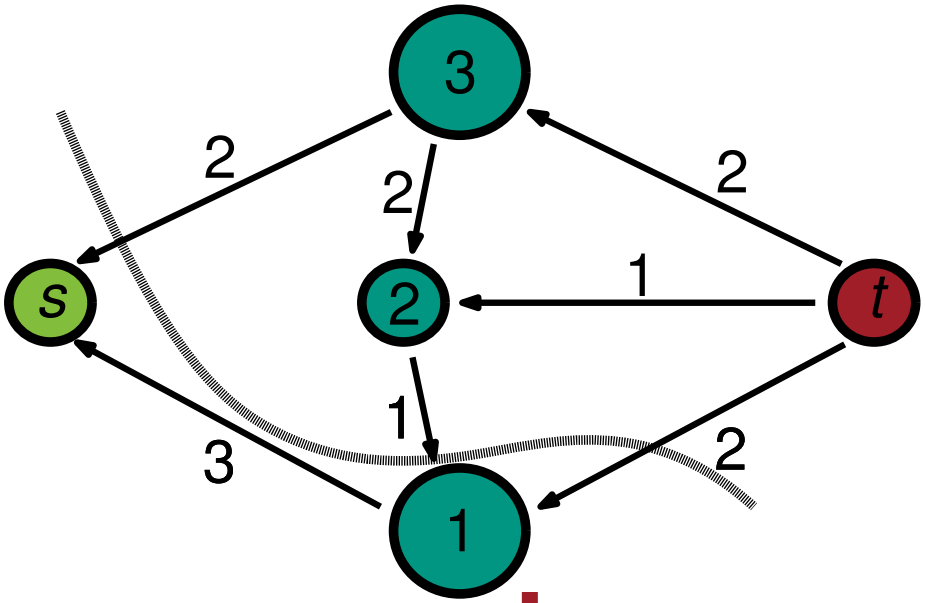
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t)-cuts

Flow Graph



Picard-Queryanne DAC



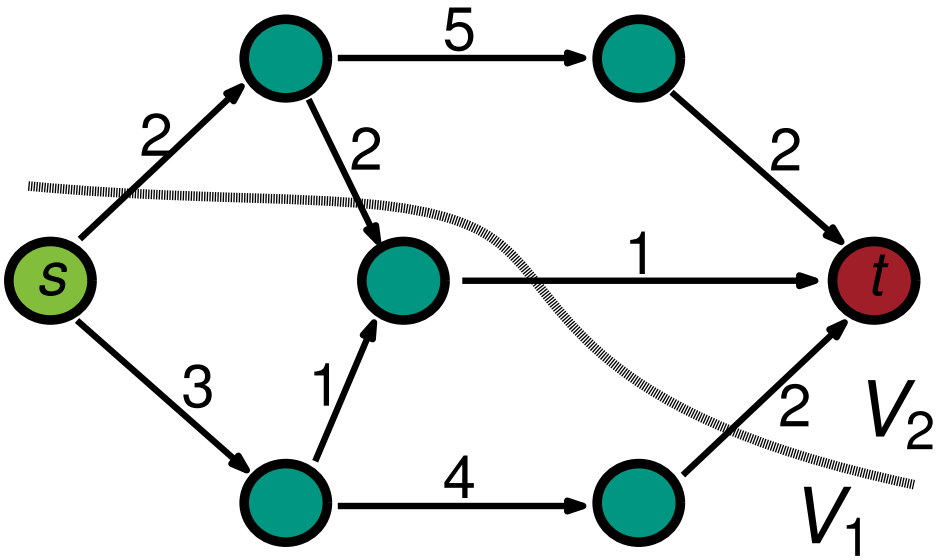
Sweep through **reverse** topological order

Institute of Theoretical Informatics
Algorithmics Group

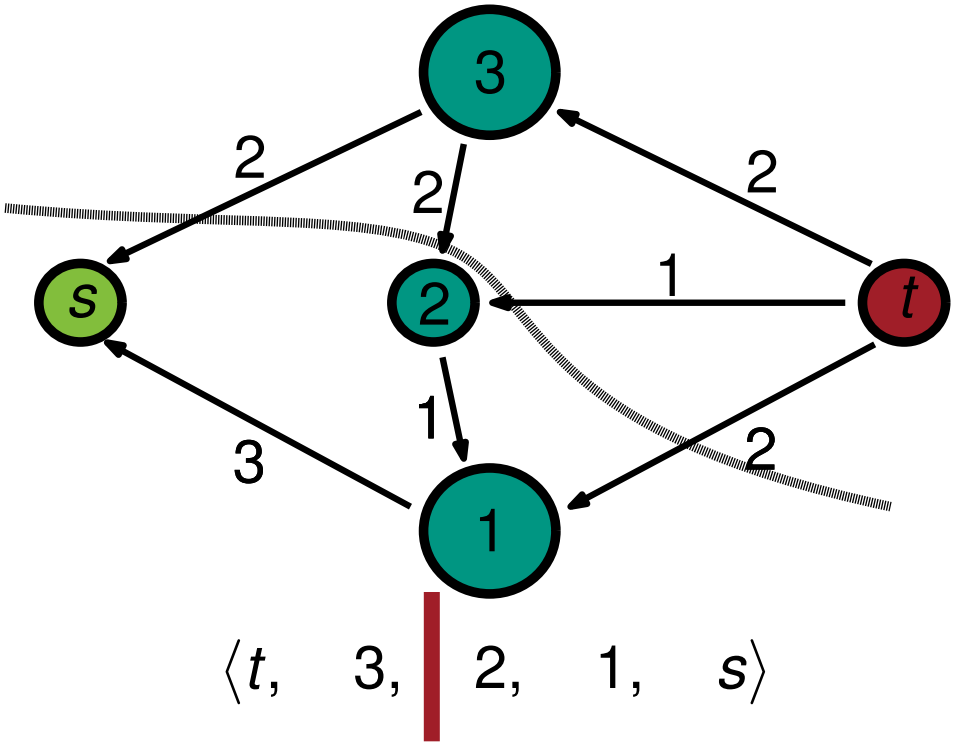
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t)-cuts

Flow Graph



Picard-Queryanne DAC

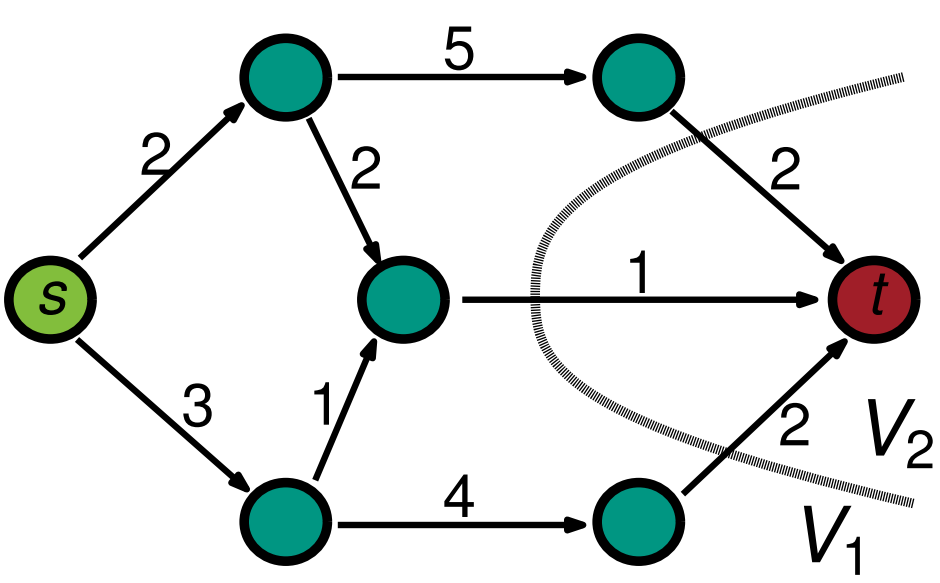


Sweep through **reverse** topological order

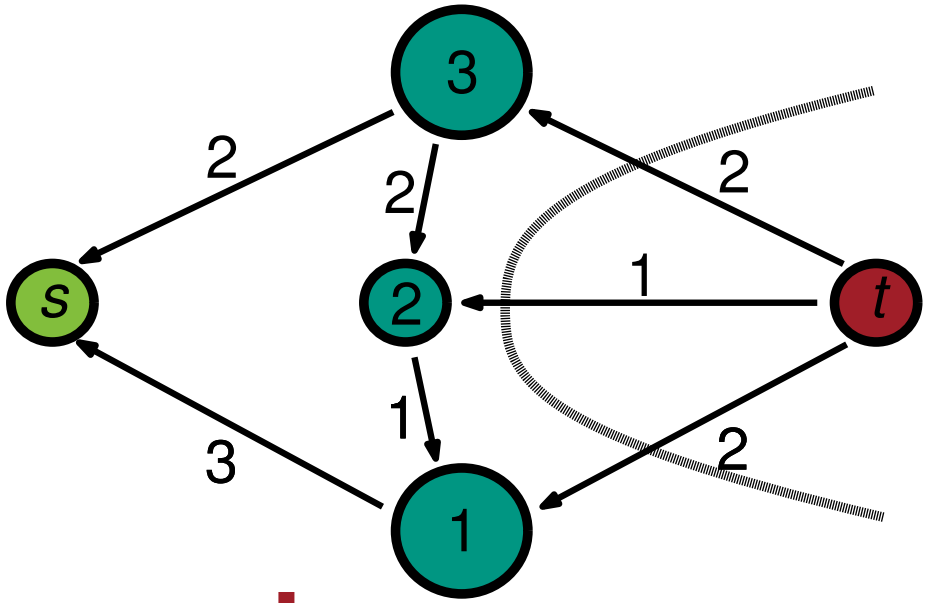
Most Balanced Minimum Cut

One maximum flow f has enough information to enumerate all minimum (s, t)-cuts

Flow Graph



Picard-Queryanne DAC



$\langle t, \mid 3, \quad 2, \quad 1, \quad s \rangle$

Sweep through **reverse** topological order

Integration into KaHyPar

- KaHyPar is a *n*-level hypergraph partitioner

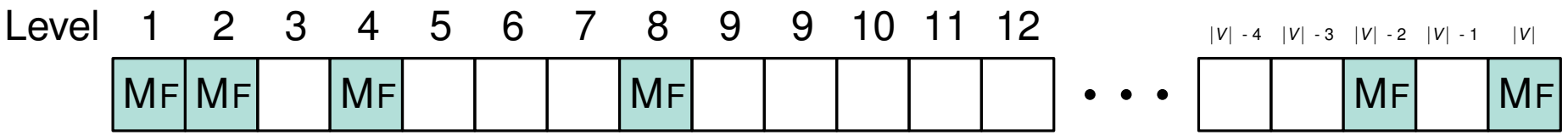
Institute of Theoretical Informatics
Algorithmics Group

Integration into KaHyPar

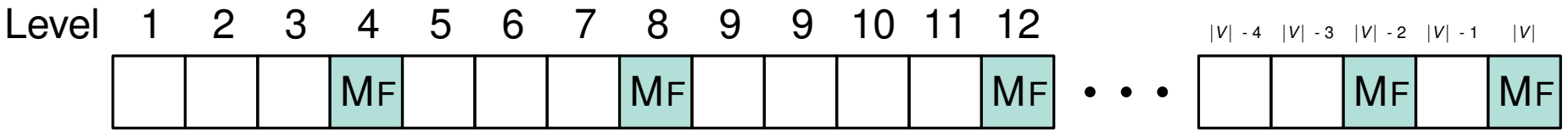
- KaHyPar is a *n*-level hypergraph partitioner

Flow Execution Policies

Multilevel: Execute *Max-Flow-Min-Cut* computations (MF) on each level i with $i = 2^j$



Constant: Execute *Max-Flow-Min-Cut* computations (MF) on each level i with $i = \beta \cdot j$

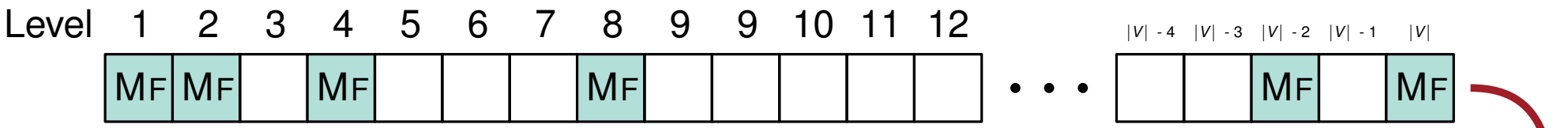


Integration into KaHyPar

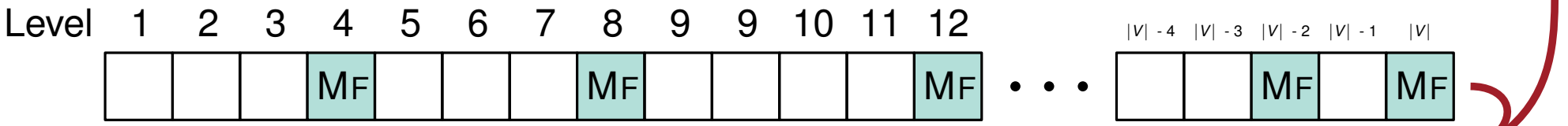
- KaHyPar is a *n*-level hypergraph partitioner

Flow Execution Policies

Multilevel: Execute *Max-Flow-Min-Cut* computations (MF) on each level i with $i = 2^j$



Constant: Execute *Max-Flow-Min-Cut* computations (MF) on each level i with $i = \beta \cdot j$



Note, each policy uses *flow*-based refinement on the **last level**

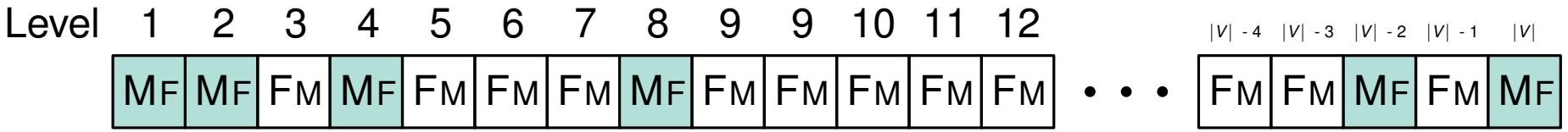
Institute of Theoretical Informatics
Algorithmics Group

Integration into KaHyPar

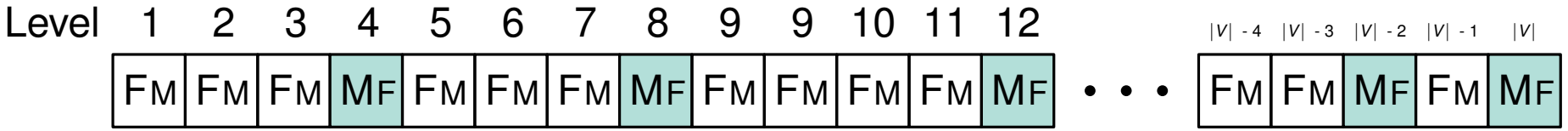
- KaHyPar is a *n*-level hypergraph partitioner

Flow Execution Policies

Multilevel: Execute *Max-Flow-Min-Cut* computations (MF) on each level i with $i = 2^j$

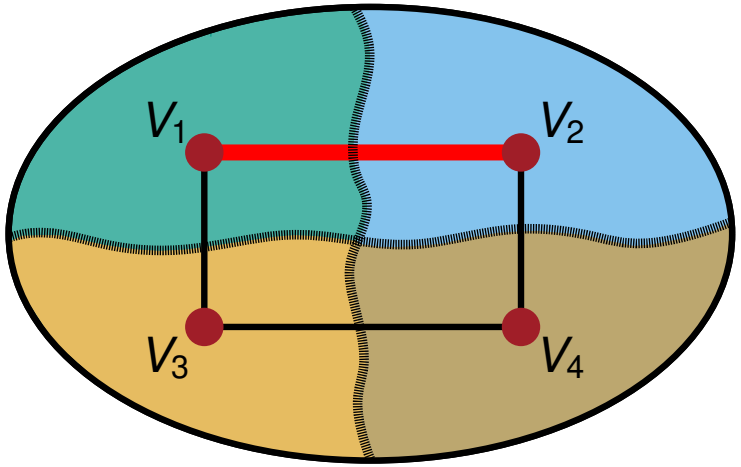


Constant: Execute *Max-Flow-Min-Cut* computations (MF) on each level i with $i = \beta \cdot j$



Institute of Theoretical Informatics
Algorithmics Group

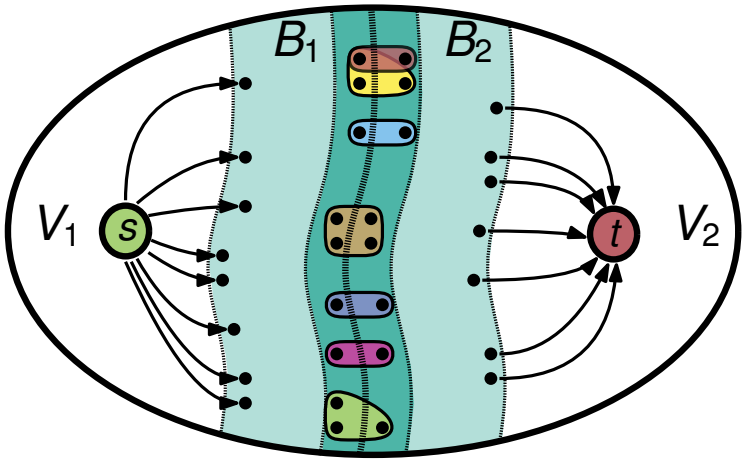
Speed-Up Heuristics



Active Block Scheduling

(R1) If cut between two blocks is small (e.g. $\leq$ 10) skip flow-based refinement, except on the last level

(R2) Only execute flow-based refinement if previous computations lead to an improvement (except in first round)



Adaptive Flow Iterations

(R3) If no hypernode change its block after *Max-Flow-Min-Cut* computation, then break

Flow Configuration

- $+/-$ F = Enabled/Disabled Flow-based refinement
- $+/-$ M = Enabled/Disabled Most Balanced Minimum Cut
- $+/-$ FM = Enabled/Disabled FM Heuristic
- CONSTANT128 = (+F,+M,+FM) with **constant** flow execution policy and $\beta = 128$

Config.	(+F,-M,-FM)		(+F,+M,-FM)		(+F,+M,+FM)		CONSTANT128	
α'	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$
1	−6.09	12.91	−5.60	13.40	0.23	15.37	0.53	55.75
2	−3.19	15.75	−2.07	16.74	0.74	18.06	1.09	87.93
4	−1.82	20.37	−0.19	21.88	1.21	22.49	1.61	144.42
8	−0.85	28.49	0.98	30.67	1.71	30.23	2.16	257.41
16	−0.19	43.32	1.75	46.66	2.21	43.53	2.69	498.29
Ref.	(-F,-M,+FM)		6373.88	13.73				

Flow Configuration

- $+/-$ F = Enabled/Disabled Flow-based refinement
- $+/-$ M = Enabled/Disabled Most Balanced Minimum Cut
- $+/-$ FM = Enabled/Disabled FM Heuristic
- CONSTANT128 = (+F,+M,+FM) with **constant** flow execution policy and $\beta = 128$

Config.	(+F,-M,-FM)		(+F,+M,-FM)		(+F,+M,+FM)		CONSTANT128	
α'	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$	Avg [%]	$t[s]$
1	−6.09	12.91	−5.60	13.40	0.23	15.37	0.53	55.75
2	−3.19	15.75	−2.07	16.74	0.74	18.06	1.09	87.93
4	−1.82	20.37	−0.19	21.88	1.21	22.49	1.61	144.42
8	−0.85	28.49	0.98	30.67	1.71	30.23	2.16	257.41
16	−0.19	43.32	1.75	46.66	2.21	43.53	2.69	498.29
Ref.	(-F,-M,+FM)		6373.88	13.73				

Institute of Theoretical Informatics
Algorithmics Group

Flow Configuration

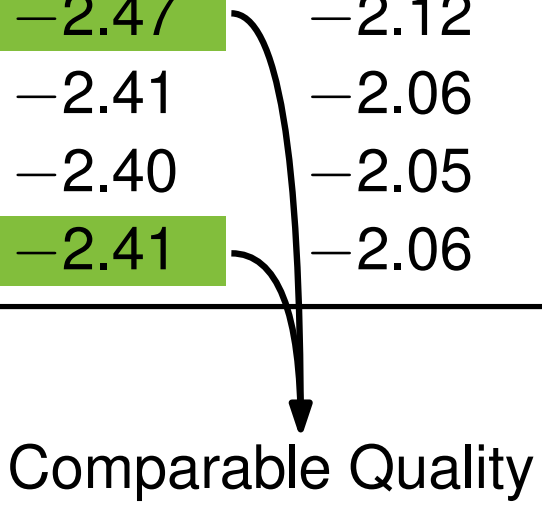
Config. / α'	(+F,-M,-FM) Avg [%] KaFFPa	(+F,-M,-FM) Avg [%] Our	(+F,+M,-FM) Avg [%] KaFFPa	(+F,+M,-FM) Avg [%] Our	(+F,+M,+FM) Avg [%] KaFFPa	(+F,+M,+FM) Avg [%] Our
1	−15.48	−6.10	−15.26	−5.62	0.14	0.23
2	−10.50	−3.20	−10.12	−2.08	0.36	0.74
4	−5.98	−1.82	−5.08	−0.20	0.67	1.21
8	−3.22	−0.85	−1.64	0.98	1.25	1.71
16	−1.52	−0.20	0.51	1.75	1.87	2.21
Ref.	(-F,-M,+FM)		6373.88			

Speed-Up Heuristics

Algorithm	Avg [%]	Min [%]	$t_{\text{flow}}[s]$	$t[s]$
KaHyPar-CA	7077.20	6820.17	-	29.26
KaHyPar-MF	−2.47	−2.12	43.04	72.30
KaHyPar-MF$_{(R1)}$	−2.41	−2.06	33.89	63.15
KaHyPar-MF$_{(R1,R2)}$	−2.40	−2.05	28.52	57.78
KaHyPar-MF$_{(R1,R2,R3)}$	−2.41	−2.06	21.23	50.49

Speed-Up Heuristics

Algorithm	Avg [%]	Min [%]	$t_{\text{flow}}[s]$	$t[s]$
KaHyPar-CA	7077.20	6820.17	-	29.26
KaHyPar-MF	−2.47	−2.12	43.04	72.30
KaHyPar-MF$_{(R1)}$	−2.41	−2.06	33.89	63.15
KaHyPar-MF$_{(R1,R2)}$	−2.40	−2.05	28.52	57.78
KaHyPar-MF$_{(R1,R2,R3)}$	−2.41	−2.06	21.23	50.49



Speed-Up Heuristics

Algorithm	Avg [%]	Min [%]	$t_{\text{flow}}[s]$	$t[s]$
KaHyPar-CA	7077.20	6820.17	-	29.26
KaHyPar-MF	−2.47	−2.12	43.04	72.30
KaHyPar-MF$_{(R1)}$	−2.41	−2.06	33.89	63.15
KaHyPar-MF$_{(R1,R2)}$	−2.40	−2.05	28.52	57.78
KaHyPar-MF$_{(R1,R2,R3)}$	−2.41	−2.06	21.23	50.49

Speed-up by a factor of 2

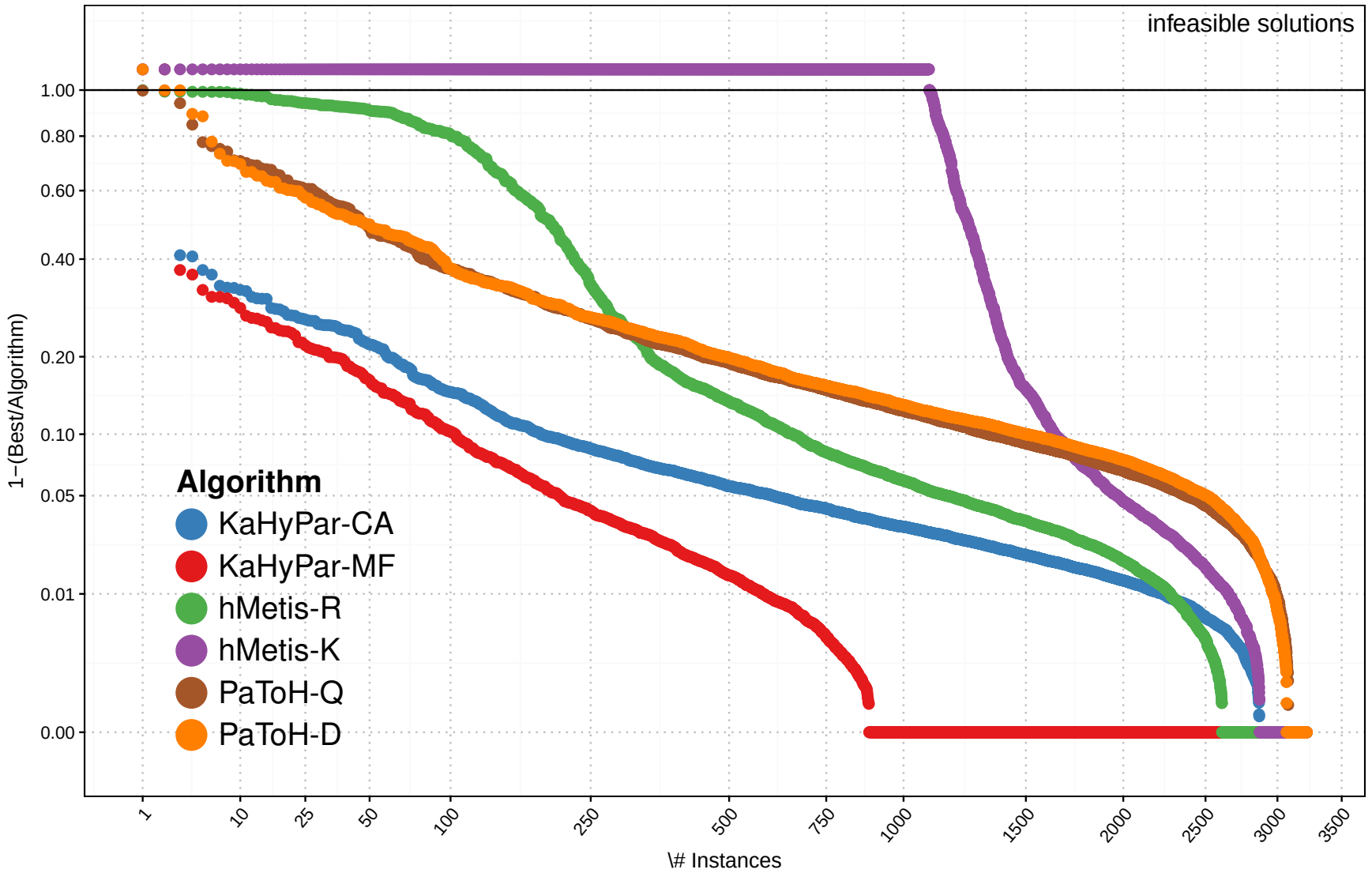
Institute of Theoretical Informatics
Algorithmics Group

Speed-Up Heuristics

Algorithm	Avg [%]	Min [%]	$t_{flow}[s]$	$t[s]$
KaHyPar-CA	7077.20	6820.17	-	29.26
KaHyPar-MF	−2.47	−2.12	43.04	72.30
KaHyPar-MF$_{(R1)}$	−2.41	−2.06	33.89	63.15
KaHyPar-MF$_{(R1,R2)}$	−2.40	−2.05	28.52	57.78
KaHyPar-MF$_{(R1,R2,R3)}$	−2.41	−2.06	21.23	50.49

Slow-down compared to KaHyPar-CA by factor of 1.72

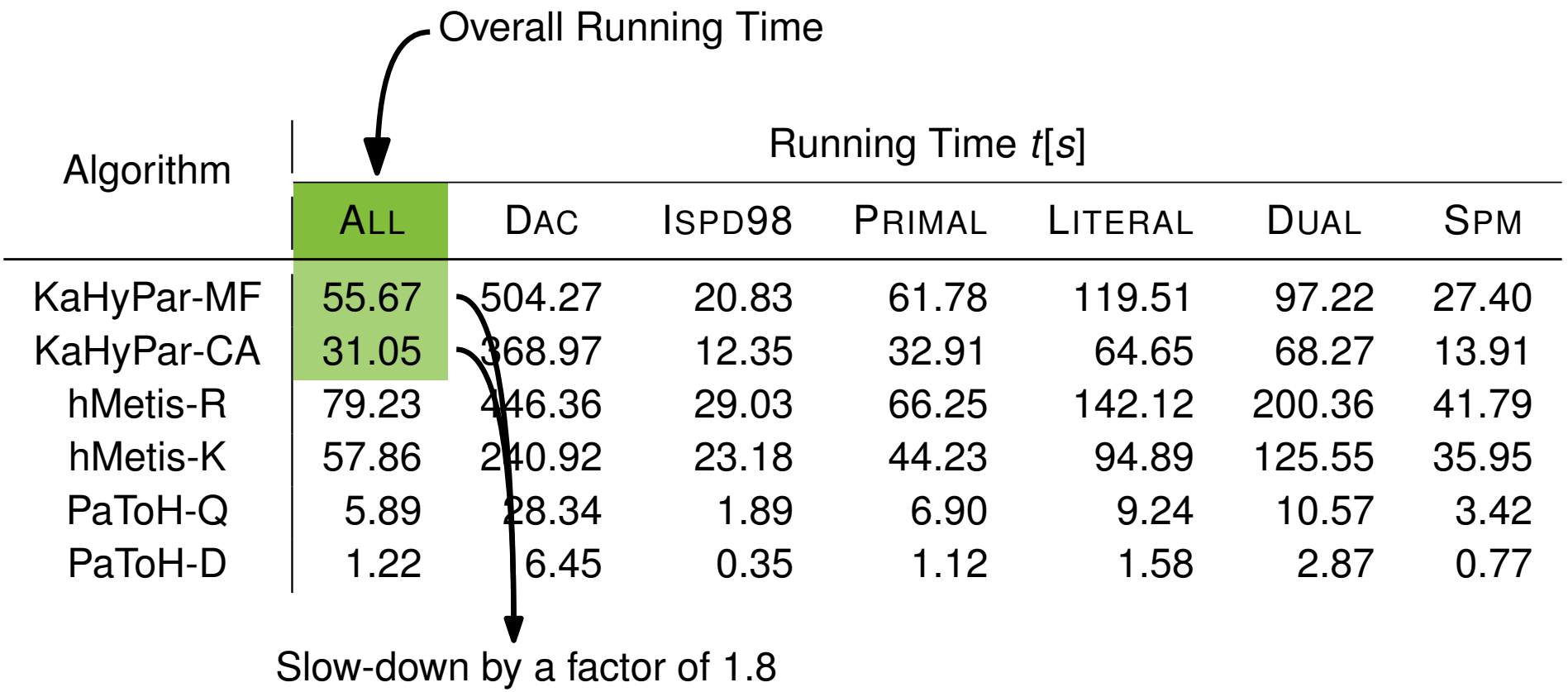
Quality - Full Benchmark Set



Running Time - Full Benchmark Set

Algorithm	Running Time $t[s]$						
	ALL	DAC	ISPD98	PRIMAL	LITERAL	DUAL	SPM
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95
PaToH-Q	5.89	28.34	1.89	6.90	9.24	10.57	3.42
PaToH-D	1.22	6.45	0.35	1.12	1.58	2.87	0.77

Running Time - Full Benchmark Set



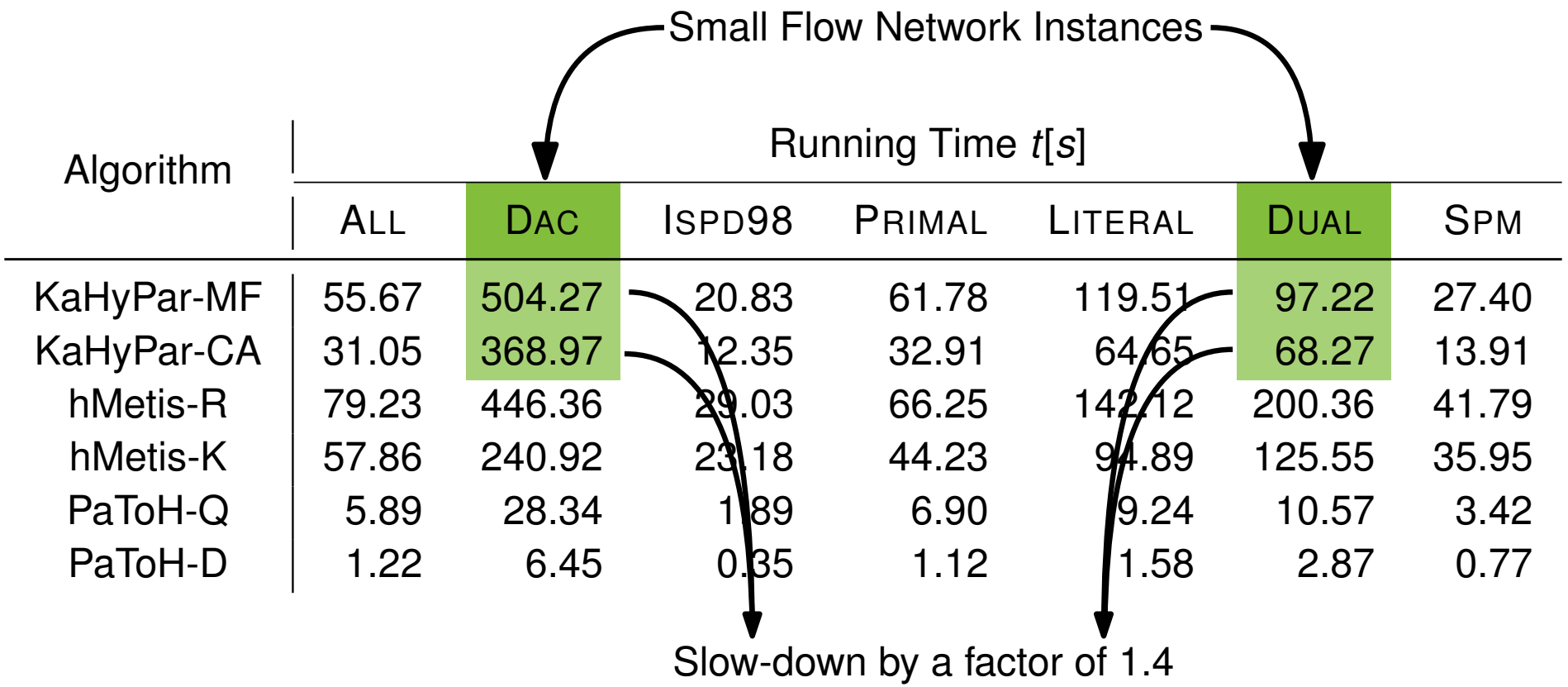
Running Time - Full Benchmark Set

Overall Running Time

Algorithm	Running Time $t[s]$						
	ALL	DAC	ISPD98	PRIMAL	LITERAL	DUAL	SPM
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95
PaToH-Q	5.89	28.34	1.89	6.90	9.24	10.57	3.42
PaToH-D	1.22	6.45	0.35	1.12	1.58	2.87	0.77

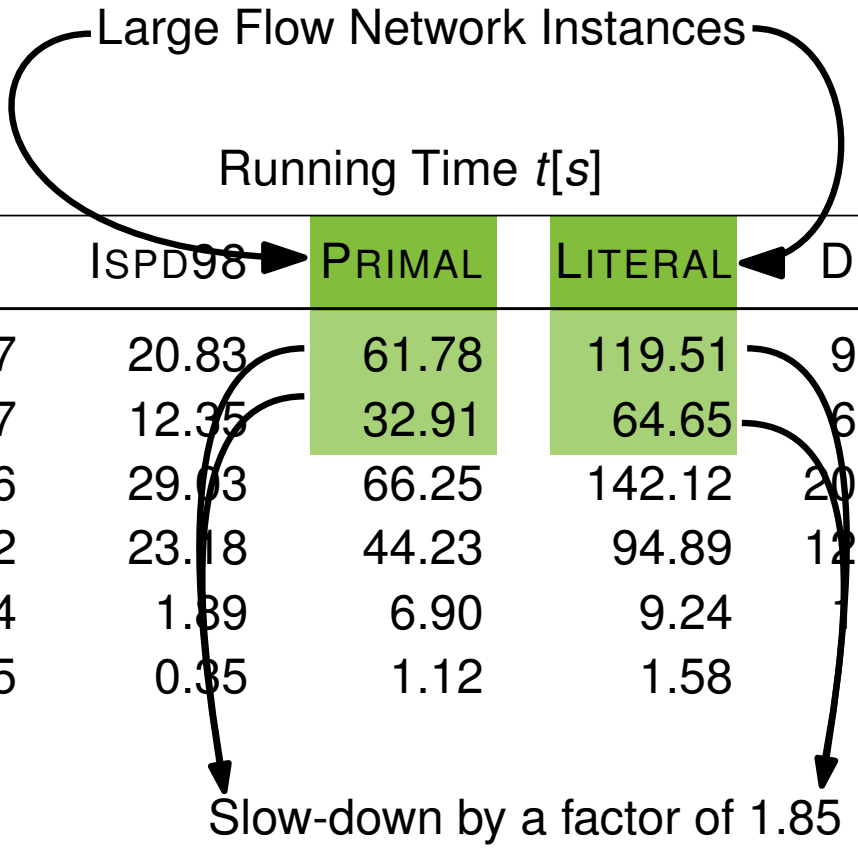
Comparable running time to hMetis-K

Running Time - Full Benchmark Set



Algorithm	Running Time $t[s]$						
	ALL	DAC	ISPD98	PRIMAL	LITERAL	DUAL	SPM
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95
PaToH-Q	5.89	28.34	1.89	6.90	9.24	10.57	3.42
PaToH-D	1.22	6.45	0.35	1.12	1.58	2.87	0.77

Running Time - Full Benchmark Set



Algorithm	Running Time $t[s]$						
	ALL	DAC	ISPD98	PRIMAL	LITERAL	DUAL	SPM
KaHyPar-MF	55.67	504.27	20.83	61.78	119.51	97.22	27.40
KaHyPar-CA	31.05	368.97	12.35	32.91	64.65	68.27	13.91
hMetis-R	79.23	446.36	29.03	66.25	142.12	200.36	41.79
hMetis-K	57.86	240.92	23.18	44.23	94.89	125.55	35.95
PaToH-Q	5.89	28.34	1.89	6.90	9.24	10.57	3.42
PaToH-D	1.22	6.45	0.35	1.12	1.58	2.87	0.77

Conclusion

Karlsruhe Institute of Technology
Institute of Theoretical Informatics
Algorithmics Group