Exact Combinatorial Algorithms for Graph Bisection

Daniel Delling Andrew V. Goldberg
Ilya Razenshteyn Renato Werneck

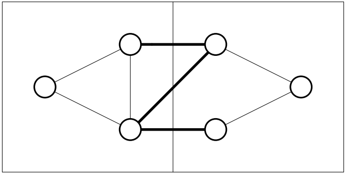
Microsoft Research Silicon Valley

DIMACS Challenge

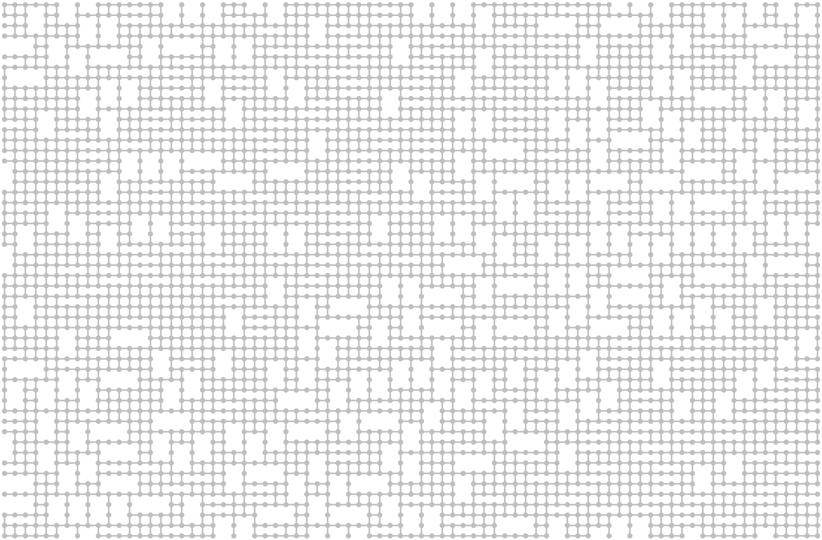
Minimum Bisection

- Input: undirected, unweighted graph $G = (V, E)$
- Output: partition V into sets A and B such that
  1. $|A|, |B| \leq \lceil |V|/2 \rceil$
  2. the number of edges between A and B is minimized

- Variants: ϵ-unbalanced, weights, more parts...

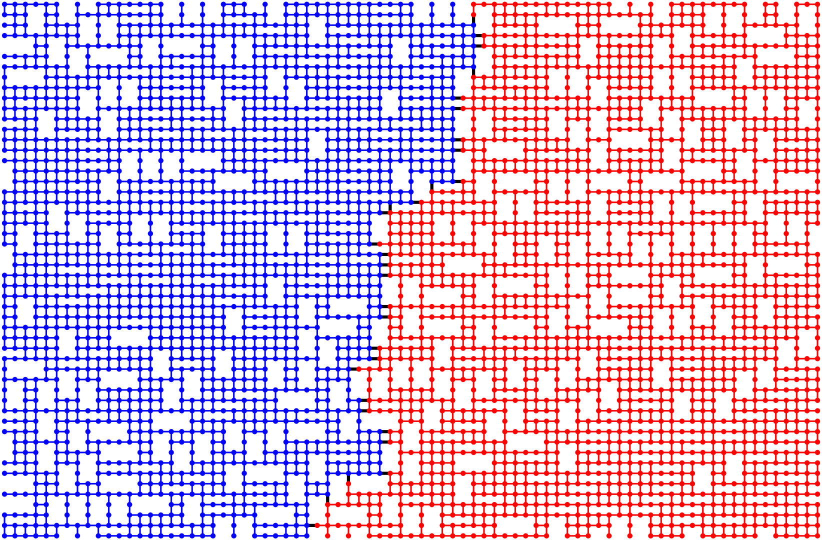


Example Instance



[3524 nodes, 5560, solution 30]

Example Instance



[3524 nodes, 5560, solution 30]

Motivation

Applications:

- load balancing for parallel computing
- preprocessing step for some road network algorithms
- divide-and-conquer (e.g. VLSI design)

Solutions:

- NP-hard, $O(\log n)$ best approximation [Räcke '08].
- Heuristics:
  - numerous fast and good partitioners
  - often tailored to specific graph classes (e.g. road networks)
  - no approximation/optimality guarantees

Motivation

Applications:

- load balancing for parallel computing
- preprocessing step for some road network algorithms
- divide-and-conquer (e.g. VLSI design)

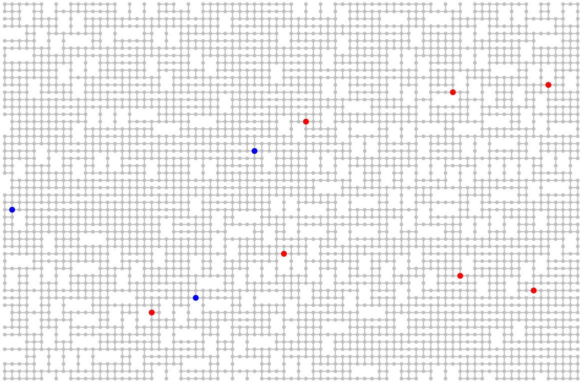
Solutions:

- NP-hard, $O(\log n)$ best approximation [Räcke '08].
- Heuristics:
  - numerous fast and good partitioners
  - often tailored to specific graph classes (e.g. road networks)
  - no approximation/optimality guarantees

We want **exact** algorithms!

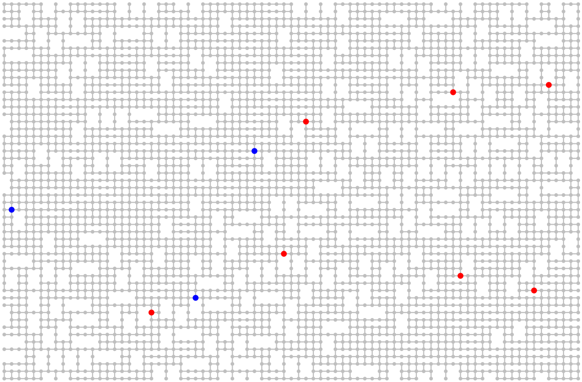
Branch-and-Bound

- Branch-and-bound: implicit enumeration.
- Subproblem = partial assignment (A, B), with $A, B \subseteq V$
  - represents bisections (A^+, B^+) with $A \subseteq A^+$ and $B \subseteq B^+$



Branch-and-Bound

- Branch-and-bound: implicit enumeration.
- Subproblem = partial assignment (A, B), with $A, B \subseteq V$
  - represents bisections (A^+, B^+) with $A \subseteq A^+$ and $B \subseteq B^+$
- **Branch**:
  - pick $v \notin A \cup B$, create subproblems $(A \cup \{v\}, B)$ and $(A, B \cup \{v\})$.



Branch-and-Bound

- Branch-and-bound: implicit enumeration.
- Subproblem = partial assignment (A, B), with $A, B \subseteq V$
  - represents bisections (A^+, B^+) with $A \subseteq A^+$ and $B \subseteq B^+$
- **Branch**:
  - pick $v \notin A \cup B$, create subproblems $(A \cup \{v\}, B)$ and $(A, B \cup \{v\})$.
- **Bound**:
  - L: lower bound on all bisections consistent with (A, B)
  - U: best known bisection (updated on-line)
  - if $L \geq U$, done with (A, B); otherwise **branch**

Branch-and-Bound

- Branch-and-bound: implicit enumeration.
- Subproblem = partial assignment (A, B), with $A, B \subseteq V$
  - represents bisections (A^+, B^+) with $A \subseteq A^+$ and $B \subseteq B^+$
- **Branch**:
  - pick $v \notin A \cup B$, create subproblems $(A \cup \{v\}, B)$ and $(A, B \cup \{v\})$.
- **Bound**:
  - L: lower bound on all bisections consistent with (A, B)
  - U: best known bisection (updated on-line)
  - if $L \geq U$, done with (A, B); otherwise **branch**

Crucial ingredient: computing lower bounds.

Lower Bounds

Known bounds:

- linear programming [FMdSWW98, Sen01]
  - hundreds of nodes
- quadratic programming [HPZ11]
  - up to 3000 nodes
- semidefinite programming [AFHM08, Arm07]
  - up to 6000 nodes
- multicommodity flows [SST03]
  - hundreds of nodes
- degree-based combinatorial [Fel05]
  - tens of nodes (random)

Lower Bounds

Known bounds:

- linear programming [FMdSWW98, Sen01]
  - hundreds of nodes
- quadratic programming [HPZ11]
  - up to 3000 nodes
- semidefinite programming [AFHM08, Arm07]
  - up to 6000 nodes
- multicommodity flows [SST03]
  - hundreds of nodes
- degree-based combinatorial [Fel05]
  - tens of nodes (random)

We want to do better.

Summary

Our result:

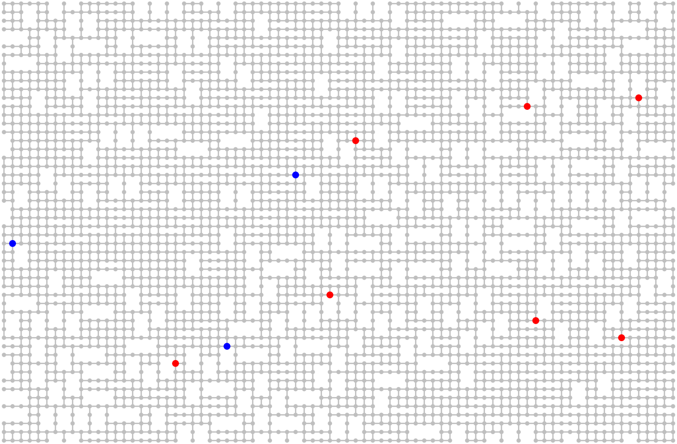
- exact combinatorial algorithm for graph bisection
- works well for graphs **with a small minimum bisection**
  - road networks, VLSI instances, meshes...
- solves much larger instances than previous approaches

Main contributions:

- new lower bounds
- branching rules
- novel decomposition technique

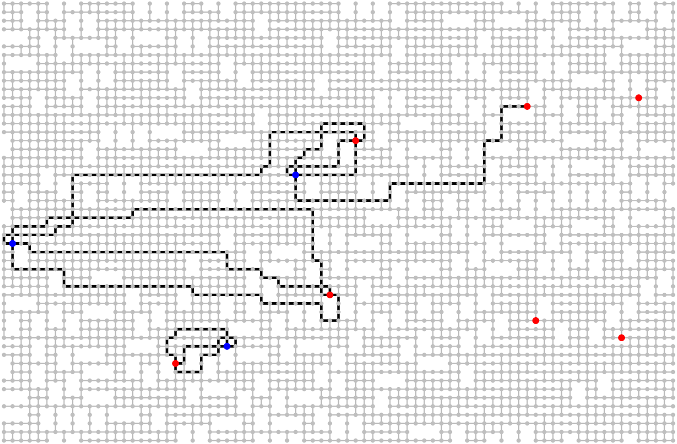
Flow Lower Bound

- Goal: lower-bound all bisections consistent with (A, B).



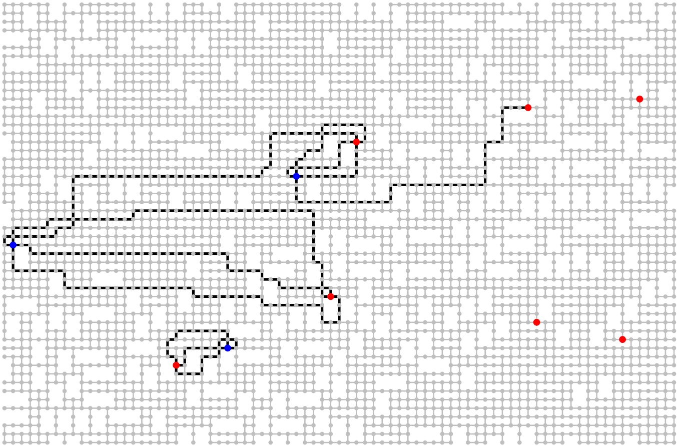
Flow Lower Bound

- Goal: lower-bound all bisections consistent with (A, B).
- Known bound: min-cut (max-flow) between A and B.



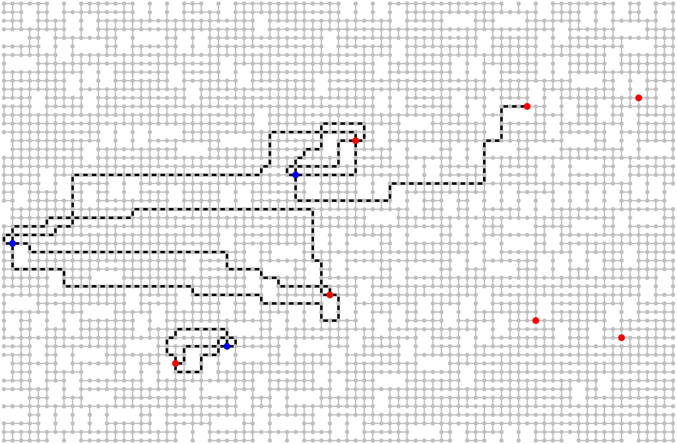
Flow Lower Bound

- Goal: lower-bound all bisections consistent with (A, B).
- Known bound: min-cut (max-flow) between A and B.
  - pros: simple, upper bound when balanced;



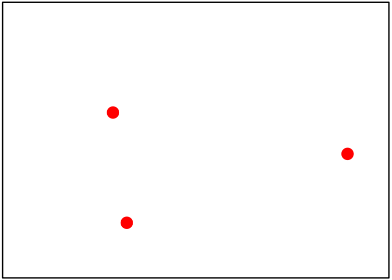
Flow Lower Bound

- Goal: lower-bound all bisections consistent with (A, B).
- Known bound: min-cut (max-flow) between A and B.
  - pros: simple, upper bound when balanced;
  - cons: weak if $|A| \ll |B|$, typically very unbalanced.



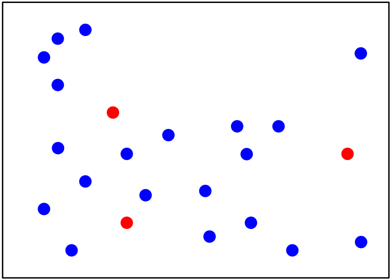
Packing Lower Bound

- $A, B \subseteq V$: current partial assignment



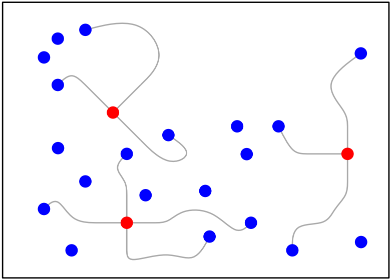
Packing Lower Bound

- $A, B \subseteq V$: current partial assignment
- A^{+}: an extension of A of size $|V|/2$



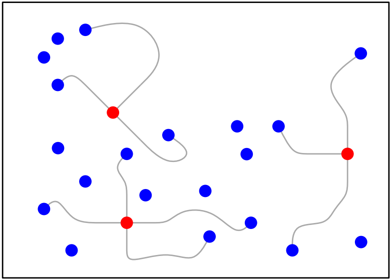
Packing Lower Bound

- $A, B \subseteq V$: current partial assignment
- A^+: an extension of A of size $|V|/2$



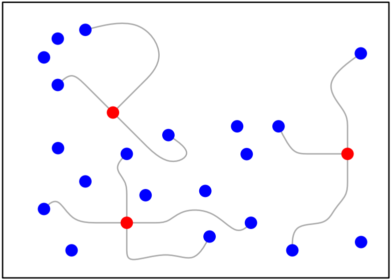
Packing Lower Bound

- A, $B \subseteq V$: current partial assignment
- A^+: an extension of A of size $|V|/2$
- **Fact:** $\min_{A^+} \mathrm{cut}(A^+, B)$ is a valid lower bound for (A, B)
  - $\mathrm{cut}(A^+, B)$: min-cut between A^+ and B.



Packing Lower Bound

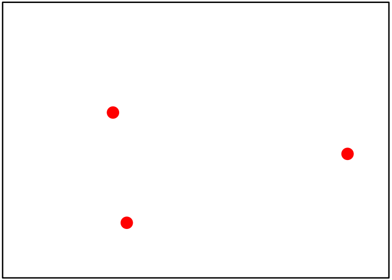
- A, $B \subseteq V$: current partial assignment
- A^+: an extension of A of size $|V|/2$
- **Fact:** $\min_{A^+} \mathrm{cut}(A^+, B)$ is a valid lower bound for (A, B)
  - $\mathrm{cut}(A^+, B)$: min-cut between A^+ and B.



We must reason about the **worst possible extension** A^+.

Packing Lower Bound

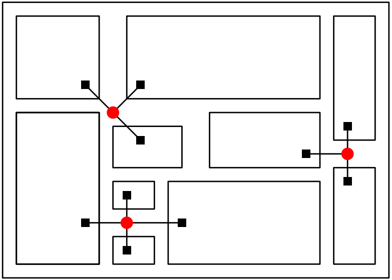
Basic algorithm:



Packing Lower Bound

Basic algorithm:

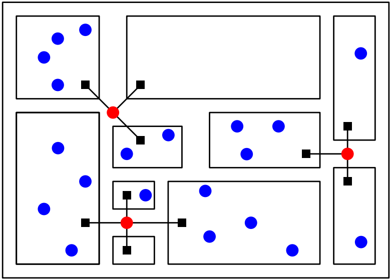
- partition free nodes into connected cells adjacent to B



Packing Lower Bound

Basic algorithm:

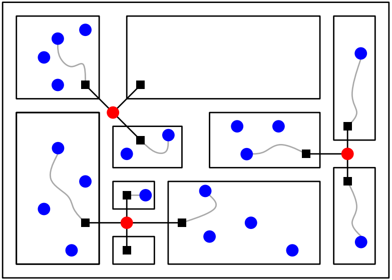
- partition free nodes into connected cells adjacent to B
- if a subset A^+ of V of size $|V|/2$ hits k cells...



Packing Lower Bound

Basic algorithm:

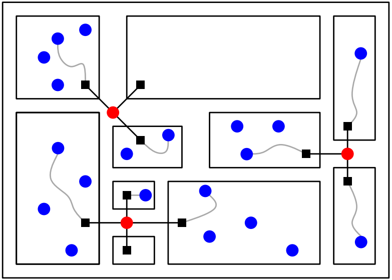
- partition free nodes into connected cells adjacent to B
- if a subset A^+ of V of size $|V|/2$ hits k cells...
- ...there is a flow of at least k units between A^+ and B.



Packing Lower Bound

Basic algorithm:

- partition free nodes into connected cells adjacent to B
- if a subset A^+ of V of size $|V|/2$ hits k cells...
- ...there is a flow of at least k units between A^+ and B.

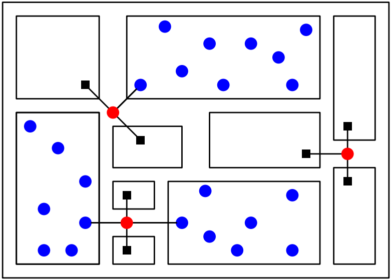


- Lower bound for (A, B) = worst-case A^+:
  - hits the fewest possible cells

Packing Lower Bound

Basic algorithm:

- partition free nodes into connected cells adjacent to B
- if a subset A^+ of V of size $|V|/2$ hits k cells...
- ...there is a flow of at least k units between A^+ and B.

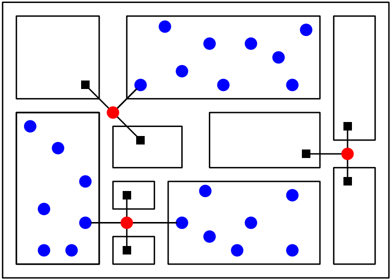


- Lower bound for (A, B) = worst-case A^+:
  - hits the fewest possible cells
  - "adversary" picks entire cells, from biggest to smallest

Packing Lower Bound

Basic algorithm:

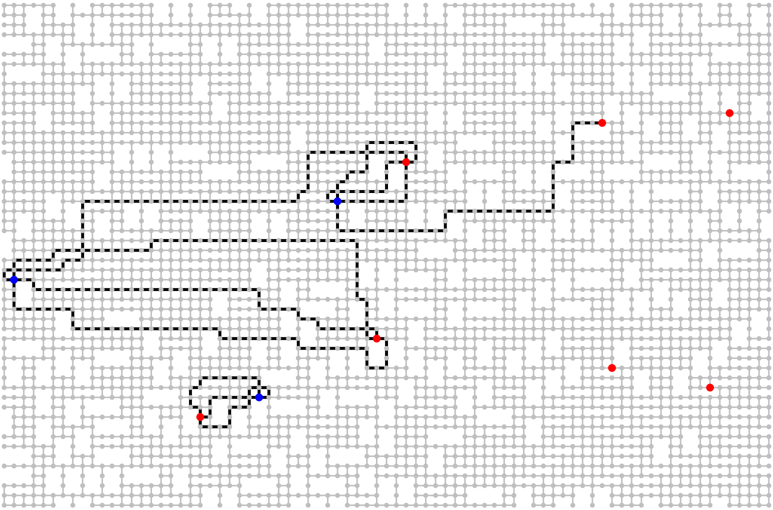
- partition free nodes into connected cells adjacent to B
- if a subset A^+ of V of size $|V|/2$ hits k cells...
- ...there is a flow of at least k units between A^+ and B.



- Lower bound for (A, B) = worst-case A^+:
  - hits the fewest possible cells
  - "adversary" picks entire cells, from biggest to smallest
  - $\Rightarrow$ partition should have balanced cells (greedy + local search)

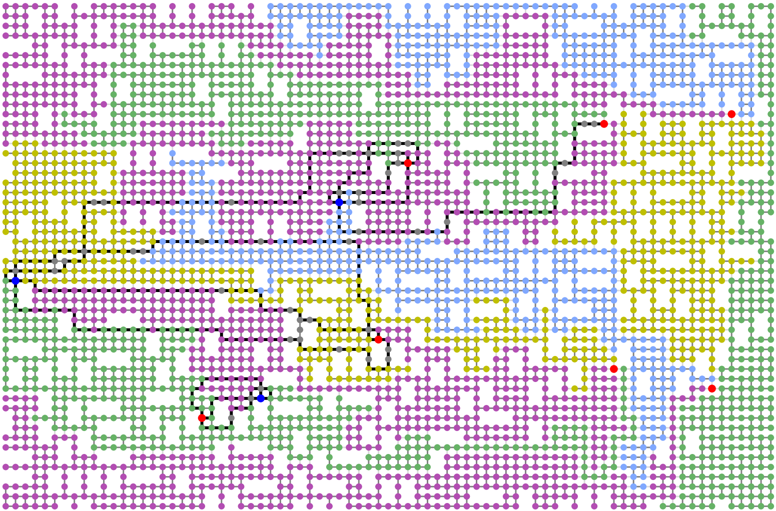
Flow + Packing

To combine flow and packing, remove flow edges before computing cells.



Flow + Packing

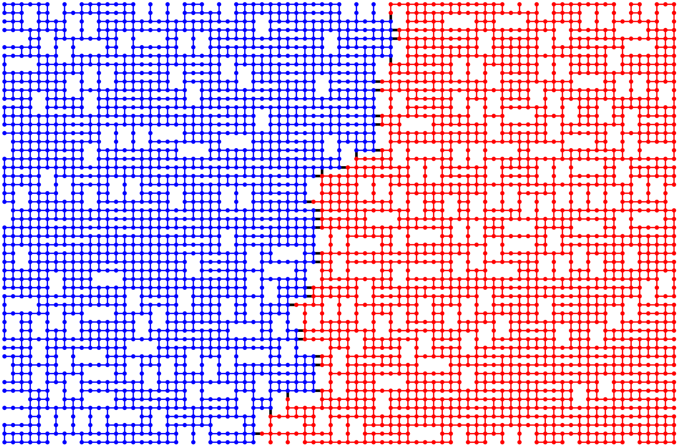
To combine flow and packing, remove flow edges before computing cells.



Performance

- Proving 30 is a lower bound:
  - search tree: 1.3M nodes
  - time: 50 minutes

1



[3524 vertices, 5560 edges, answer 30]

Branching Rule and Forced Assignments

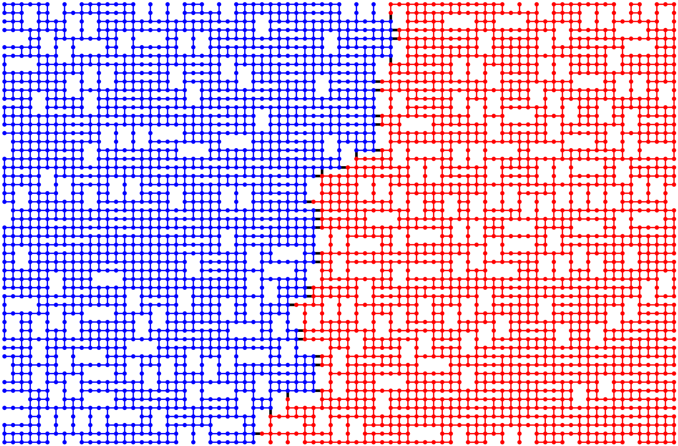
- Branch on vertices likely to increase L the most:
  1. far from A and B (to produce better cells)
  2. well connected to other vertices (to increase flow)
  3. contained in large packing cells

- Forced assignments:
  - use logical implications to fix some vertices to A or B
  - works if upper and lower bounds are close
  - discards many potential branching nodes

Branching Rule and Forced Assignments

- Proving 30 is a lower bound:
  - search tree: 1.3M nodes
  - time: 50 minutes

1

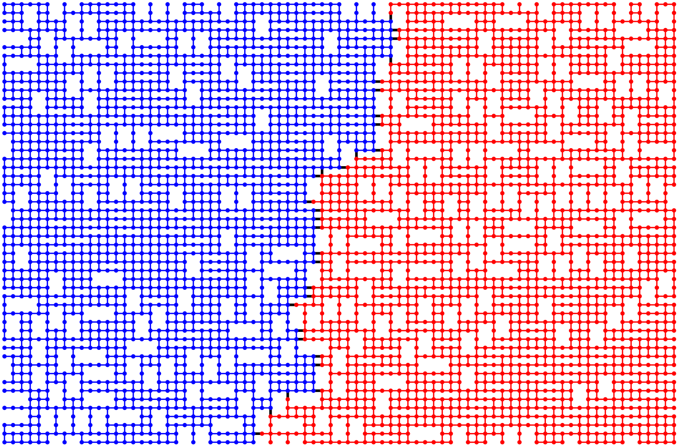


[3524 vertices, 5560 edges, answer 30]

Branching Rule and Forced Assignments

- Proving 30 is a lower bound:
  - search tree: 1.3M nodes → 90K nodes
  - time: 50 minutes → 3.5 minutes

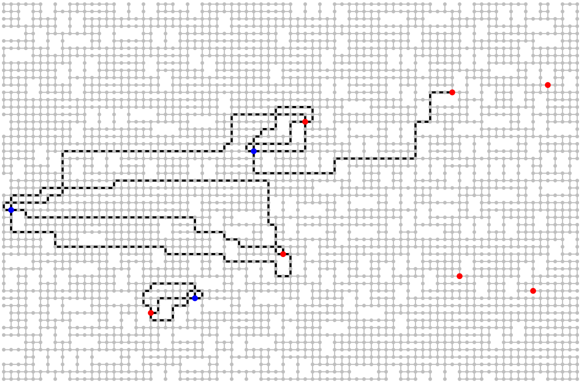
1



[3524 vertices, 5560 edges, answer 30]

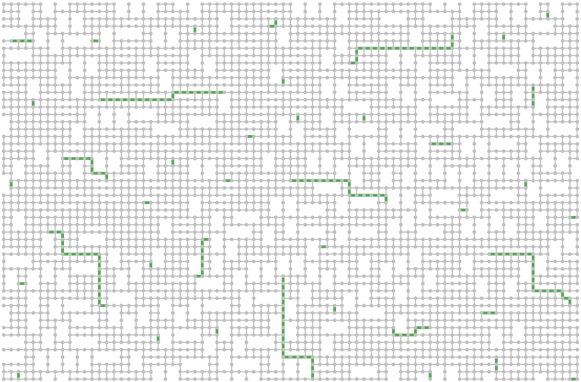
Increasing Degrees

- Algorithm is very sensitive to the degrees of assigned nodes.



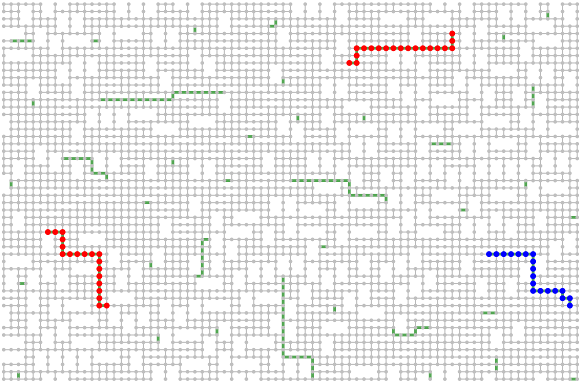
Increasing Degrees

- Algorithm is very sensitive to the degrees of assigned nodes.
- Idea: branch on entire regions.



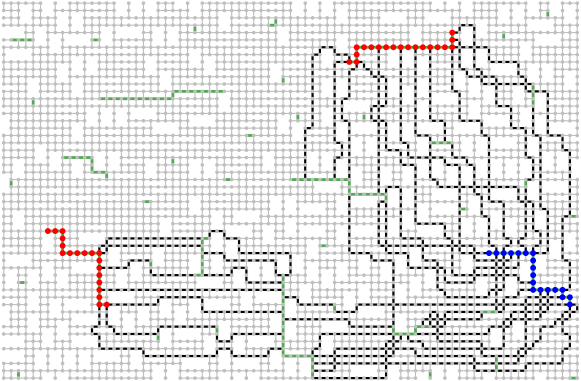
Increasing Degrees

- Algorithm is very sensitive to the degrees of assigned nodes.
- Idea: branch on entire regions.



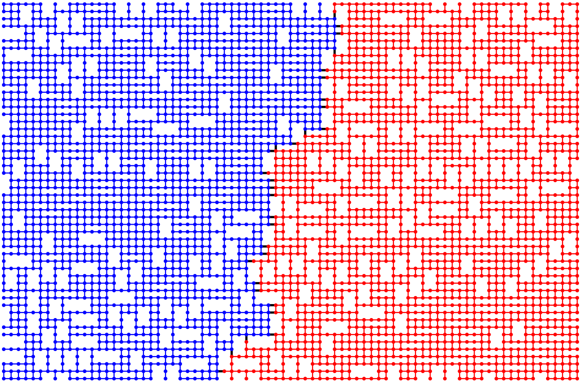
Increasing Degrees

- Algorithm is very sensitive to the degrees of assigned nodes.
- Idea: branch on entire regions.



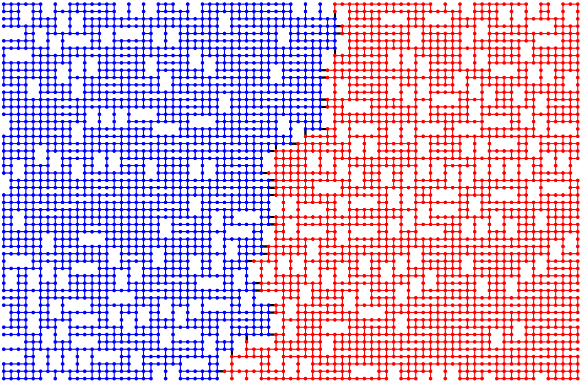
Increasing Degrees

- Algorithm is very sensitive to the degrees of assigned nodes.
- Idea: branch on entire regions.
- Problem: a region can cross the minimum bisection.



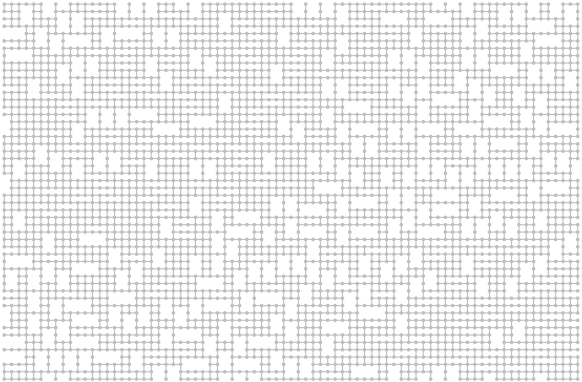
Increasing Degrees

- Algorithm is very sensitive to the degrees of assigned nodes.
- Idea: branch on entire regions.
- Problem: a region can cross the minimum bisection.
- Solution: decomposition!



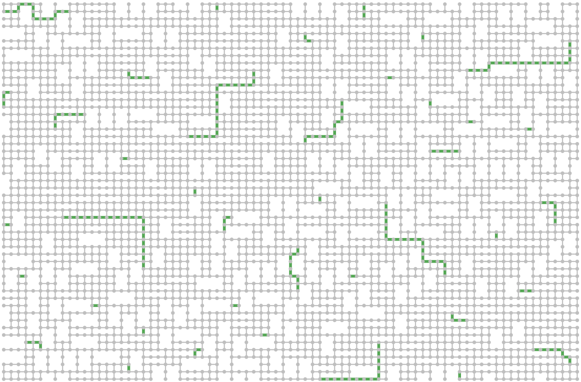
Decomposition

- Given an upper bound U:
  1. split edges into $U + 1$ disjoint sets E_1, E_2, ..., E_{U+1}



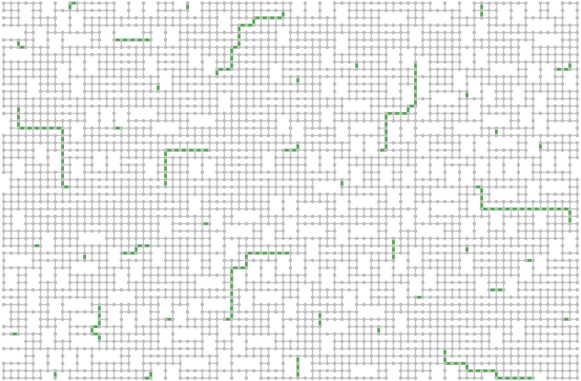
Decomposition

- Given an upper bound U:
  1. split edges into $U + 1$ disjoint sets E_1, E_2, ..., E_{U+1}



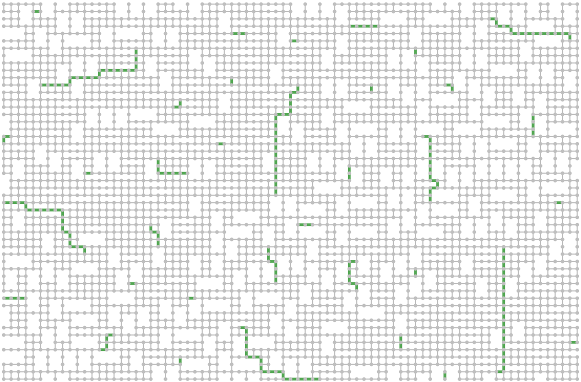
Decomposition

- Given an upper bound U:
  1. split edges into $U + 1$ disjoint sets E_1, E_2, ..., E_{U+1}



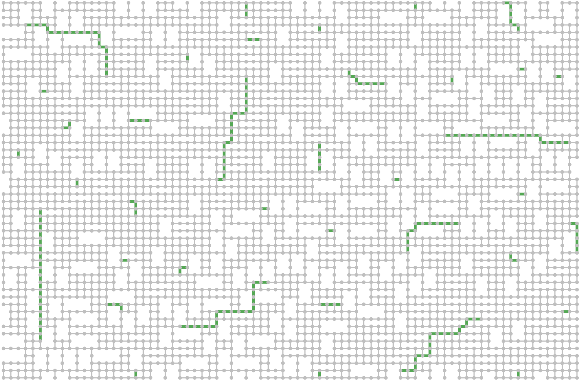
Decomposition

- Given an upper bound U:
  1. split edges into $U + 1$ disjoint sets E_1, E_2, ..., E_{U+1}



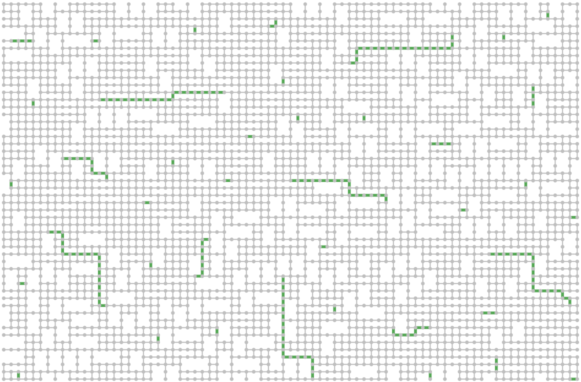
Decomposition

- Given an upper bound U:
  1. split edges into $U + 1$ disjoint sets E_1, E_2, ..., E_{U+1}



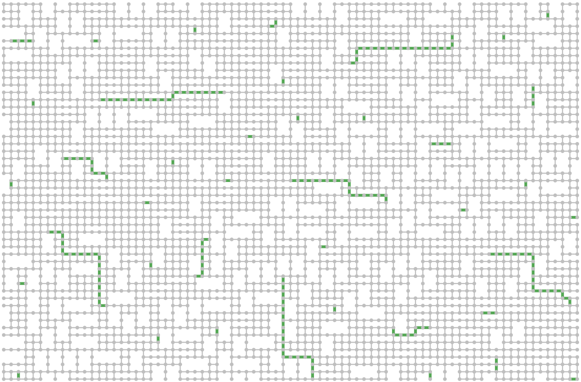
Decomposition

- Given an upper bound U:
  1. split edges into $U + 1$ disjoint sets E_1, E_2, ..., E_{U+1}



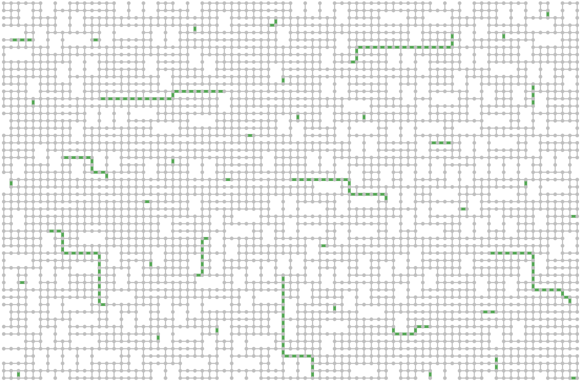
Decomposition

- Given an upper bound U:
  1. split edges into $U + 1$ disjoint sets E_1, E_2, ..., E_{U+1}
  2. for every i, contract E_i and run branch-and-bound



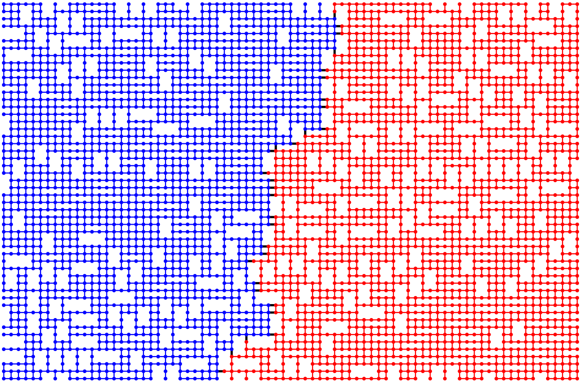
Decomposition

- Given an upper bound U:
  1. split edges into $U + 1$ disjoint sets $E_1, E_2, \ldots, E_{U+1}$
  2. for every i, contract E_i and run branch-and-bound
  3. return best solution found



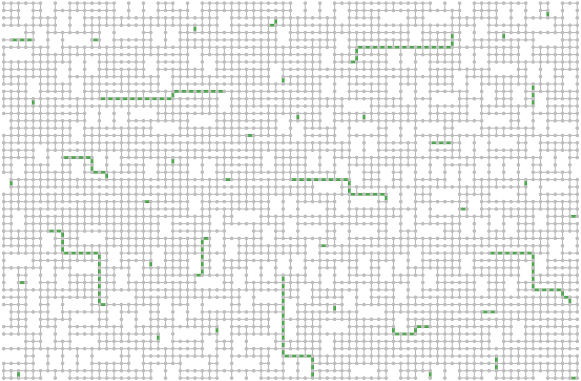
Decomposition

- Given an upper bound U:
  1. split edges into $U+1$ disjoint sets E_1, E_2, ..., E_{U+1}
  2. for every i, contract E_i and run branch-and-bound
  3. return best solution found
- Some E_i will cross the optimum bisection; **at least one will not!**



Decomposition

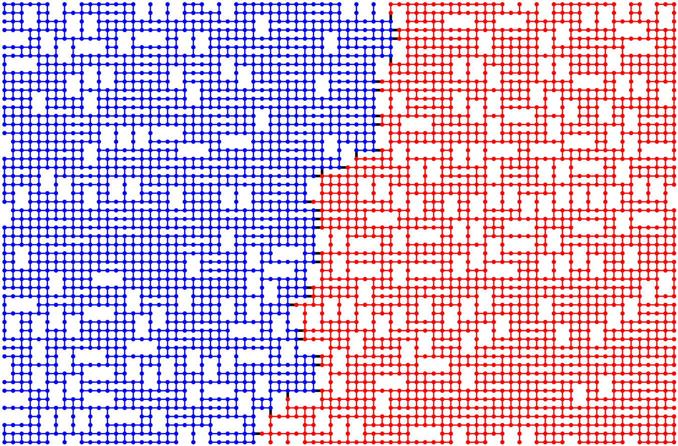
- Given an upper bound U:
  1. split edges into $U+1$ disjoint sets $E_1, E_2, \ldots, E_{U+1}$
  2. for every i, contract E_i and run branch-and-bound
  3. return best solution found
- Some E_i will cross the optimum bisection; **at least one will not!**
- E_i should be a set of clumps (high degree, well spread).



Decomposition

- Proving 30 is a lower bound:
  - search tree: 90K nodes
  - time: 3.5 minutes

1

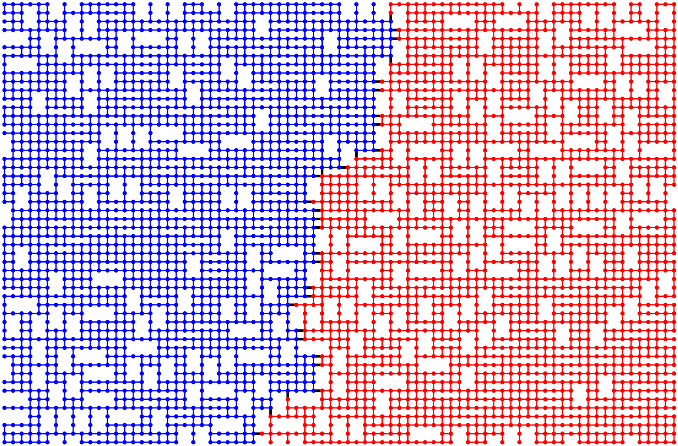


[3524 vertices, 5560 edges, answer 30]

Decomposition

- Proving 30 is a lower bound:
  - search tree: 90K nodes → 1369 nodes
  - time: 3.5 minutes → 2 seconds

1

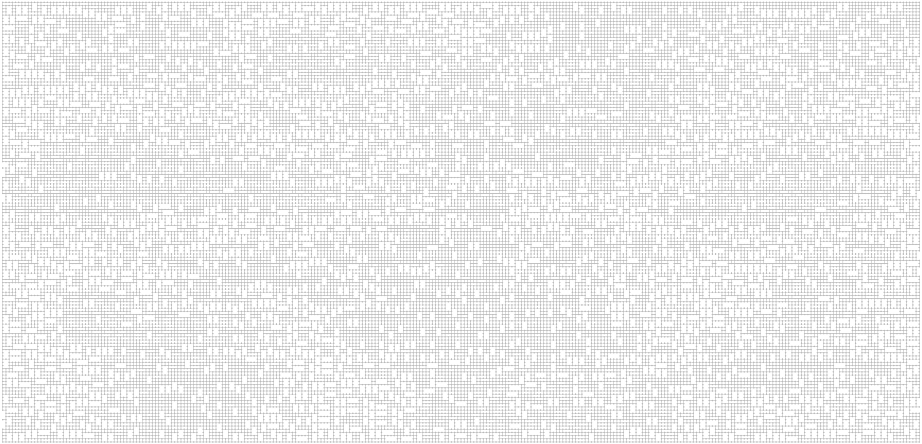


[3524 vertices, 5560 edges, answer 30]

Experiments

Examples

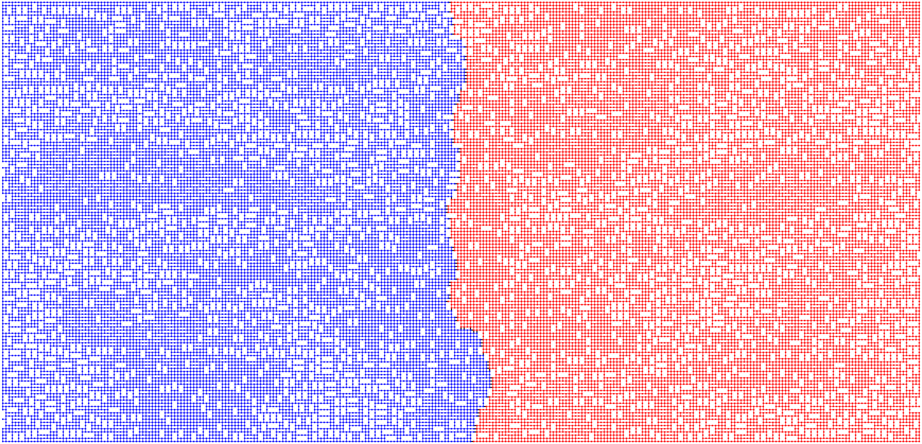
- VLSI instance:
  - search tree: 20K nodes
  - time: 9 minutes



[34046 vertices, 54841 edges, answer 80]

Examples

- VLSI instance:
  - search tree: 20K nodes
  - time: 9 minutes



[34046 vertices, 54841 edges, answer 80]

Walshaw Instances

Standard benchmark for graph partitioning ($\epsilon = 0$).

instance	n	m	opt	BB nodes	time [s]
add32	4 960	9 462	11	225	3
uk	4 824	6 837	19	1 624	4
3elt	4 720	13 722	90	12 707	82
whitaker3	9 800	28 989	127	7 044	133
fe_4elt2	11 143	32 818	130	10 391	224
4elt	15 606	45 878	139	25 912	769
data*	2 851	15 093	189	495 569 759	5 750 388

[*: distributed execution using DryadOpt]

Walshaw Instances

Standard benchmark for graph partitioning ($\epsilon = 0$).

instance	n	m	opt	BB nodes	time [s]
add32	4 960	9 462	11	225	3
uk	4 824	6 837	19	1 624	4
3elt	4 720	13 722	90	12 707	82
whitaker3	9 800	28 989	127	7 044	133
fe_4elt2	11 143	32 818	130	10 391	224
4elt	15 606	45 878	139	25 912	769
data*	2 851	15 093	189	495 569 759	5 750 388

[*: distributed execution using DryadOpt]

Optimum bisections were known before, but without proofs.

Other Challenge Instances

series	instance	n	m	opt	BB nodes	time [s]
clustering	karate	34	78	10	4	0.00
	chesapeake	39	170	46	110 138	3.08
	dolphins	62	159	15	110	0.01
	lesmis	77	820	61	3 905 756	230.30
	polbooks	105	441	19	8	0.00
	football	115	613	61	7 301	1.08
	power	4 941	6 594	12	94	0.21
delaunay	delaunay_n10	1 024	3 056	63	14 361	18.25
	delaunay_n11	2 048	6 127	86	65 080	175.73
	delaunay_n12	4 096	12 264	118	474 844	2 711.73
	delaunay_n13	8 192	24 547	156	3 122 845	37 615.97
streets	luxembourg	114 599	119 666	17	786	91.17

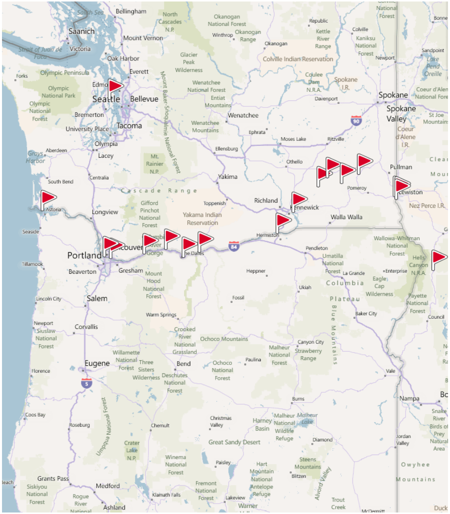
Other Challenge Instances

series	instance	n	m	opt	BB nodes	time [s]
clustering	karate	34	78	10	4	0.00
	chesapeake	39	170	46	110 138	3.08
	dolphins	62	159	15	110	0.01
	lesmis	77	820	61	3 905 756	230.30
	polbooks	105	441	19	8	0.00
	football	115	613	61	7 301	1.08
	power	4 941	6 594	12	94	0.21
delaunay	delaunay_n10	1 024	3 056	63	14 361	18.25
	delaunay_n11	2 048	6 127	86	65 080	175.73
	delaunay_n12	4 096	12 264	118	474 844	2 711.73
	delaunay_n13	8 192	24 547	156	3 122 845	37 615.97
streets	luxembourg	114 599	119 666	17	786	91.17

We can solve very large instances with small bisections.

Examples

- NW (9th Challenge): 264 branch-and-bound nodes, 25 minutes



[1.2M vertices, 1.4M edges, answer 18]

Instances from "exact" literature

- State-of-the-art approaches:
  - [Arm07]: semidefinite programming
  - [HPZ11]: quadratic programming

instance	n	m	opt	time [s]	[Arm07]	[HPZ11]
KKT_putt01_m2	115	433	28	0.81	1.67	1.51
mesh.274.469	274	469	37	0.03	8.52	24.62
gap2669.24859	2669	29037	55	0.15	348.95	—
taq170.424	170	4317	55	3.00	28.68	—
gap2669.6182	2669	12280	74	34.90	651.03	—
taq1021.2253	1021	4510	118	134.61	169.65	—

Instances from "exact" literature

- State-of-the-art approaches:
  - [Arm07]: semidefinite programming
  - [HPZ11]: quadratic programming

instance	n	m	opt	time [s]	[Arm07]	[HPZ11]
KKT_putt01_m2	115	433	28	0.81	1.67	1.51
mesh.274.469	274	469	37	0.03	8.52	24.62
gap2669.24859	2669	29037	55	0.15	348.95	—
taq170.424	170	4317	55	3.00	28.68	—
gap2669.6182	2669	12280	74	34.90	651.03	—
taq1021.2253	1021	4510	118	134.61	169.65	—

Excellent performance for small bisections.

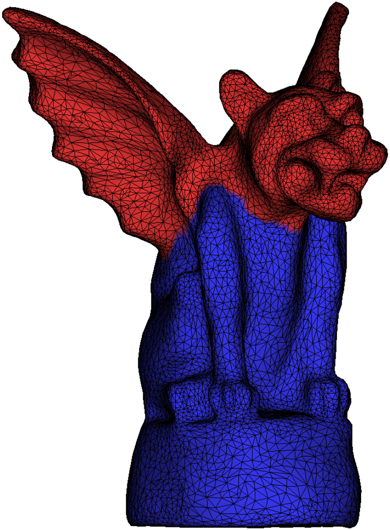
Examples

- Gargoyle: 2.6M branch-and-bound nodes, 13 hours

[10002 vertices, 30000 edges, answer 175]

Examples

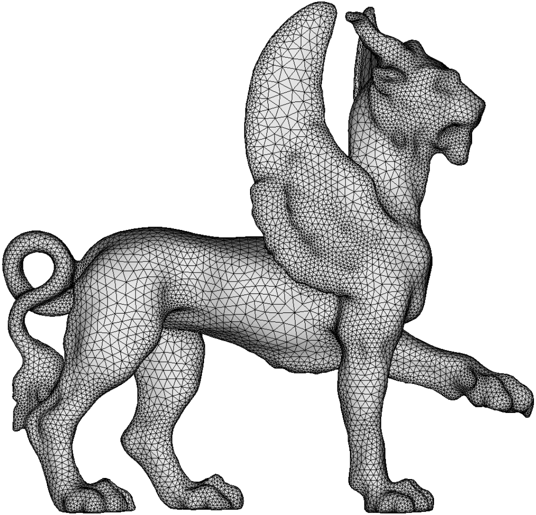
- Gargoyle: 2.6M branch-and-bound nodes, 13 hours



[10002 vertices, 30000 edges, answer 175]

Examples

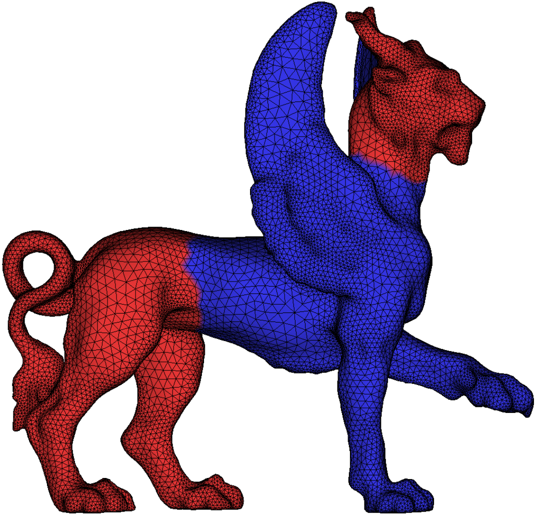
- Feline (mesh): 150K branch-and-bound nodes, 75 minutes



[20629 vertices, 61893 edges, answer 148]

Examples

- Feline (mesh): 150K branch-and-bound nodes, 75 minutes



[20629 vertices, 61893 edges, answer 148]

Conclusion

- New algorithm for minimum graph bisection
  - packing bound
  - decomposition
- Cut size matters
- Potential applications: evaluate/improve heuristics

Conclusion

- New algorithm for minimum graph bisection
  - packing bound
  - decomposition
- Cut size matters
- Potential applications: evaluate/improve heuristics

Thank you!