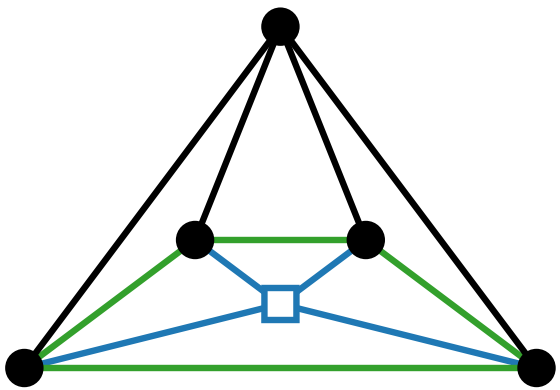
TECHNISCHE UNIVERSITÄT WIEN [1]

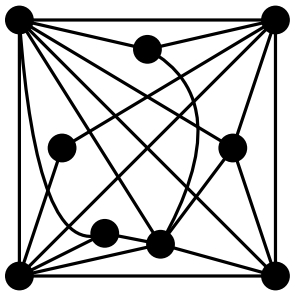
UNIVERSITÄT PASSAU [2]
Fakultät für Informatik und Mathematik



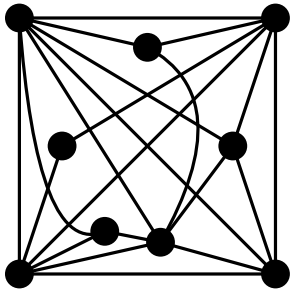
Heuristics for Exact 1-Planarity Testing

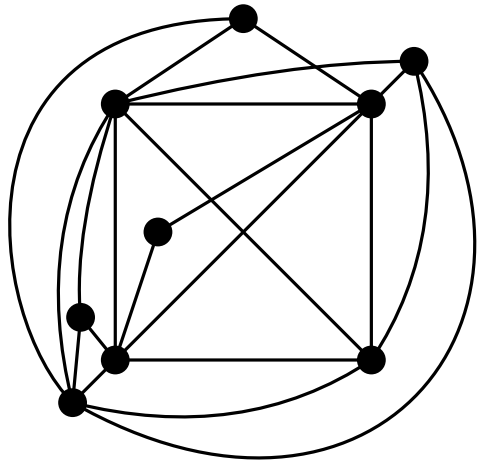
Simon D. Fink[1], Miriam Münch[2],
Matthias Pfretzschner[2], Ignaz Rutter[2]

1-Planarity

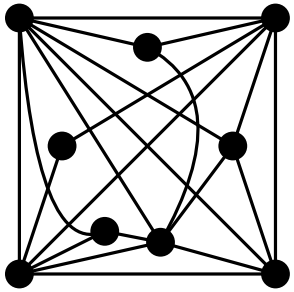


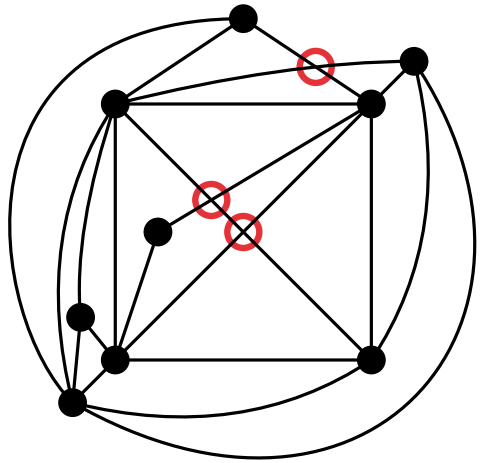
1-Planarity



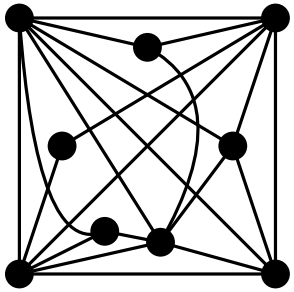


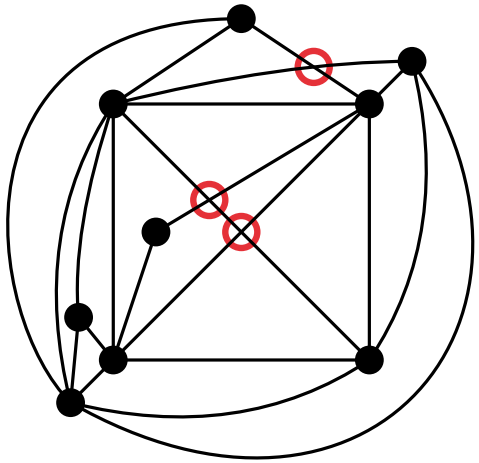
1-Planarity

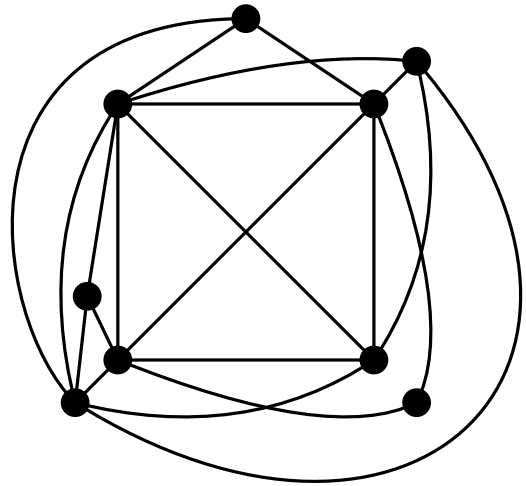




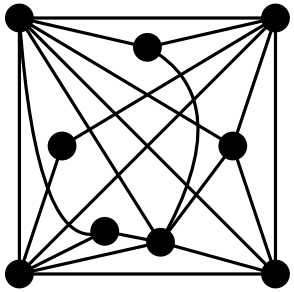
1-Planarity

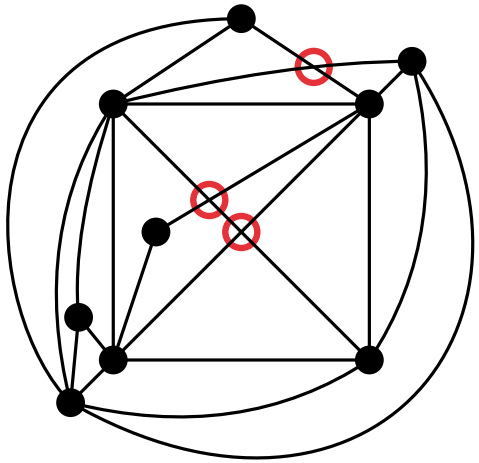


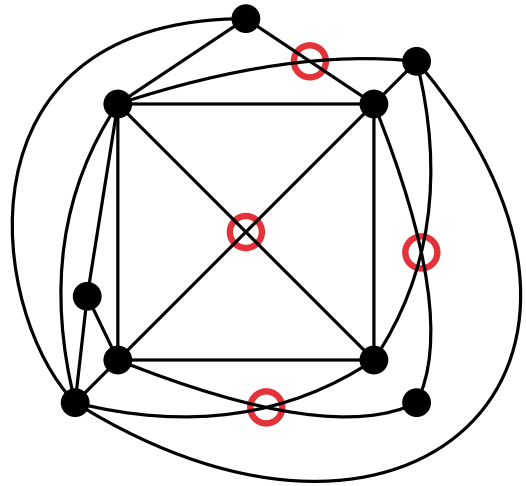




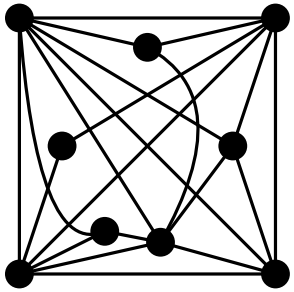
1-Planarity

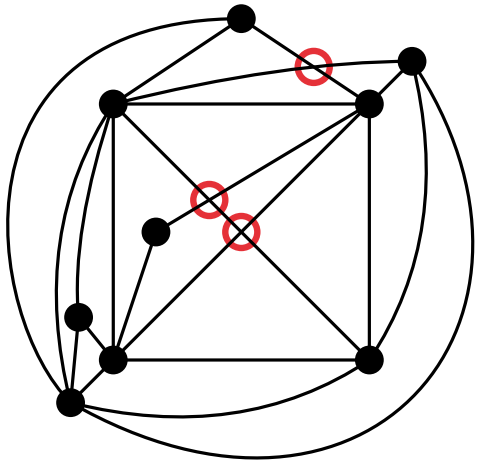




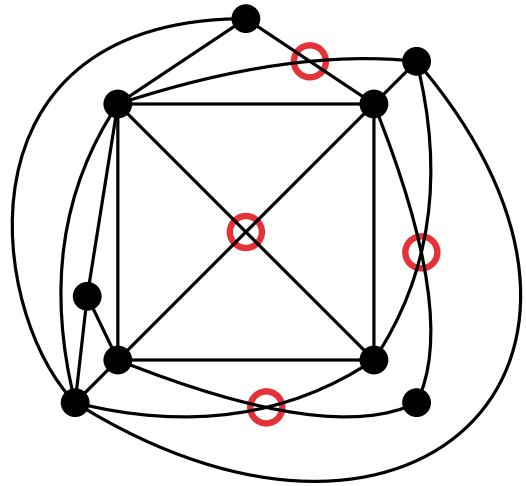


1-Planarity



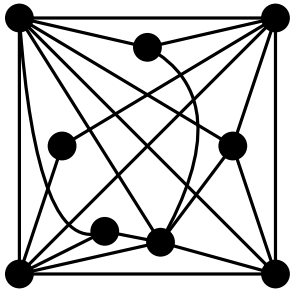


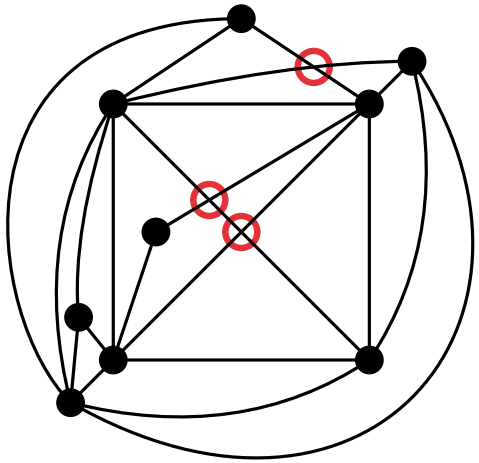
not 1-planar



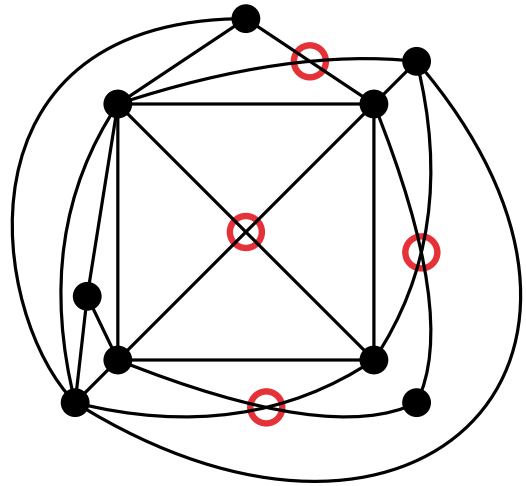
1-planar

1-Planarity





not 1-planar



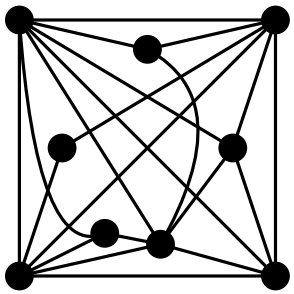
1-planar

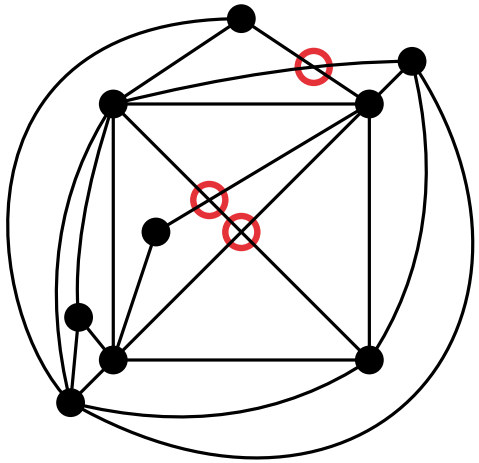
1-Planarity

Input: Graph G

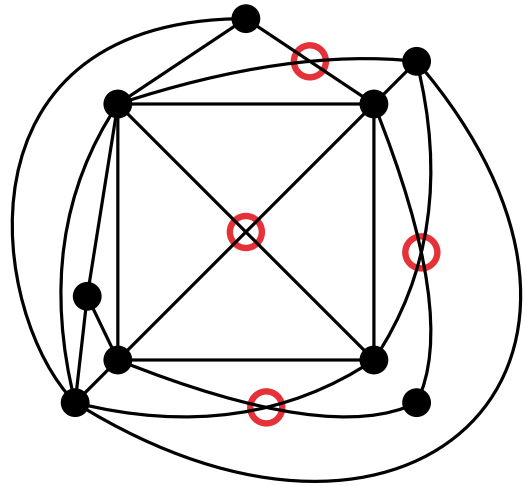
Question: Does G admit a 1-planar drawing?

1-Planarity





not 1-planar



1-planar

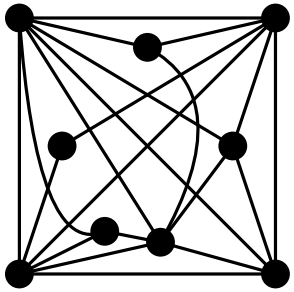
1-Planarity

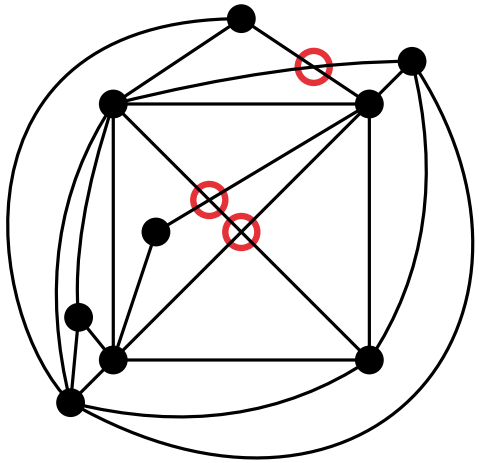
Input: Graph G

Question: Does G admit a 1-planar drawing?

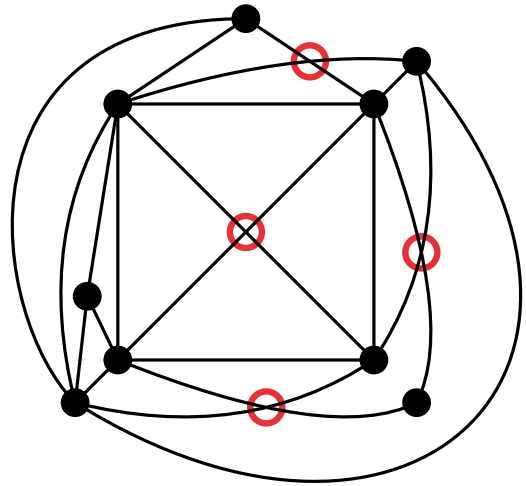
- NP-complete [Grigoriev, Bodlaender ’07; Korzhik, Mohar ’13]

1-Planarity





not 1-planar



1-planar

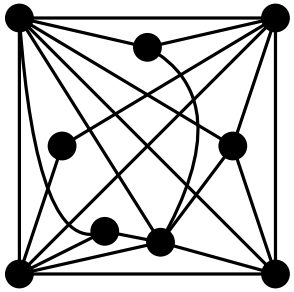
1-Planarity

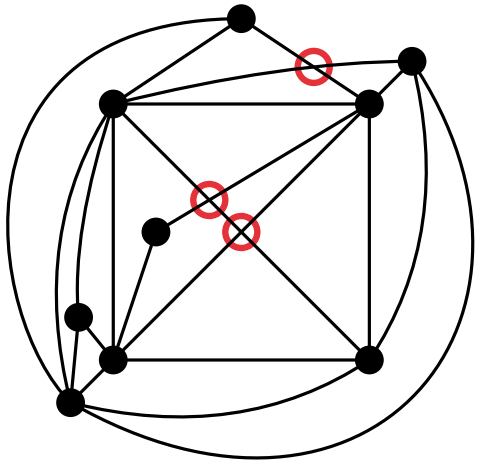
Input: Graph G

Question: Does G admit a 1-planar drawing?

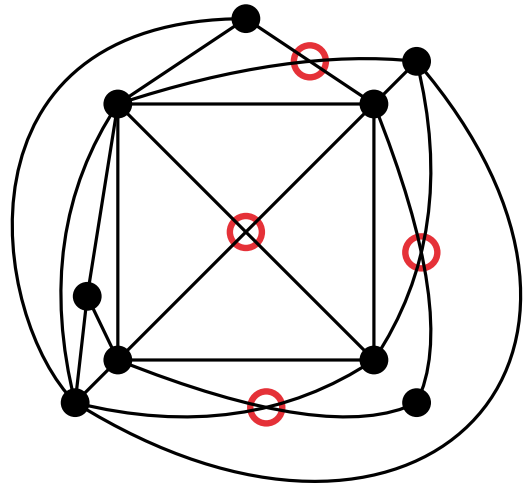
- NP-complete [Grigoriev, Bodlaender ’07; Korzhik, Mohar ’13]
- FPT wrt. vertex cover number, treedepth, cyclomatic number [Bannister, Cabello, Eppstein ’13]

1-Planarity





not 1-planar



1-planar

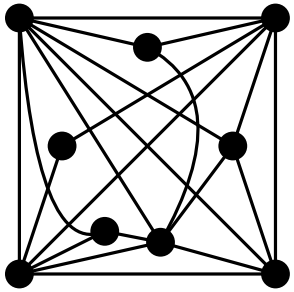
1-Planarity

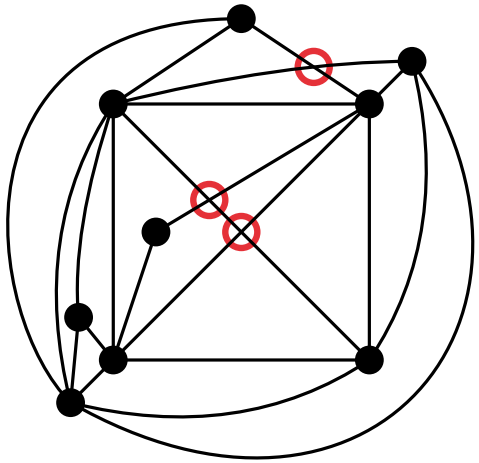
Input: Graph G

Question: Does G admit a 1-planar drawing?

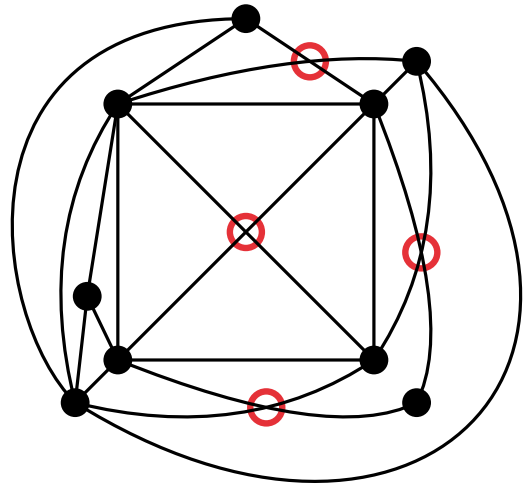
- NP-complete [Grigoriev, Bodlaender ’07; Korzhik, Mohar ’13]
- FPT wrt. vertex cover number, treedepth, cyclomatic number [Bannister, Cabello, Eppstein ’13]
- Exact backtracking procedure [Binucci, Didimo, Montecchiani ’20]

1-Planarity





not 1-planar



1-planar

1-Planarity

Input: Graph G

Question: Does G admit a 1-planar drawing?

- NP-complete [Grigoriev, Bodlaender ’07; Korzhik, Mohar ’13]
- FPT wrt. vertex cover number, treedepth, cyclomatic number [Bannister, Cabello, Eppstein ’13]
- Exact backtracking procedure [Binucci, Didimo, Montecchiani ’20]
- Our contribution: Evaluation of different backtracking strategies

1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

- Decompose graph into biconnected components

1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges

1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

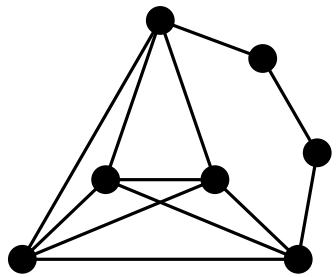
- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges

- Branch over whether to cross edge pairs in this order

1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges

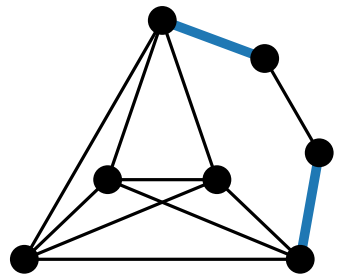
- Branch over whether to cross edge pairs in this order



1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

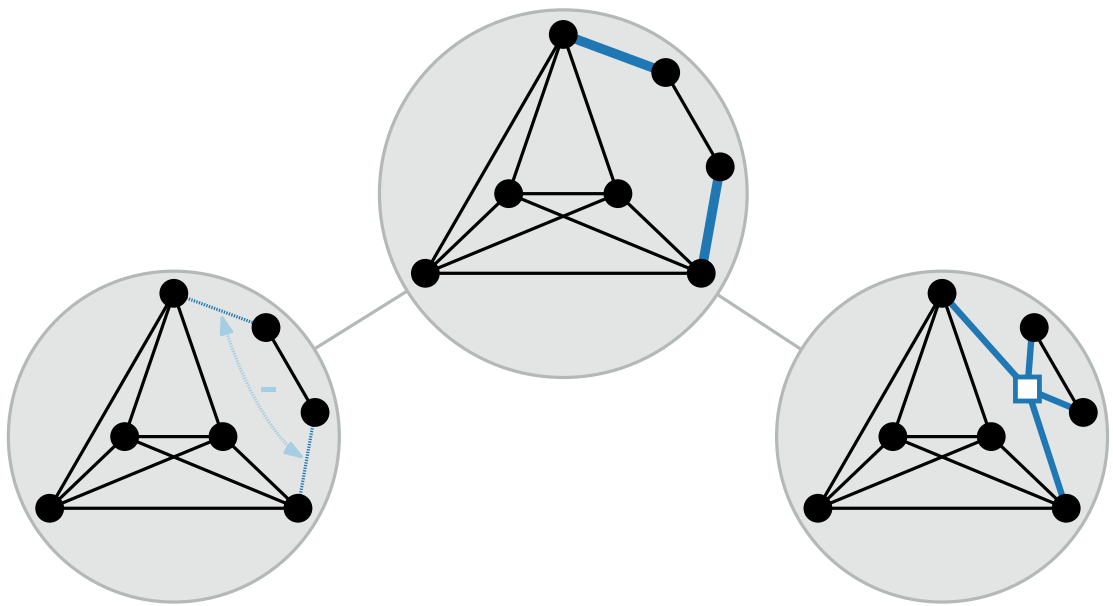
- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges

- Branch over whether to cross edge pairs in this order



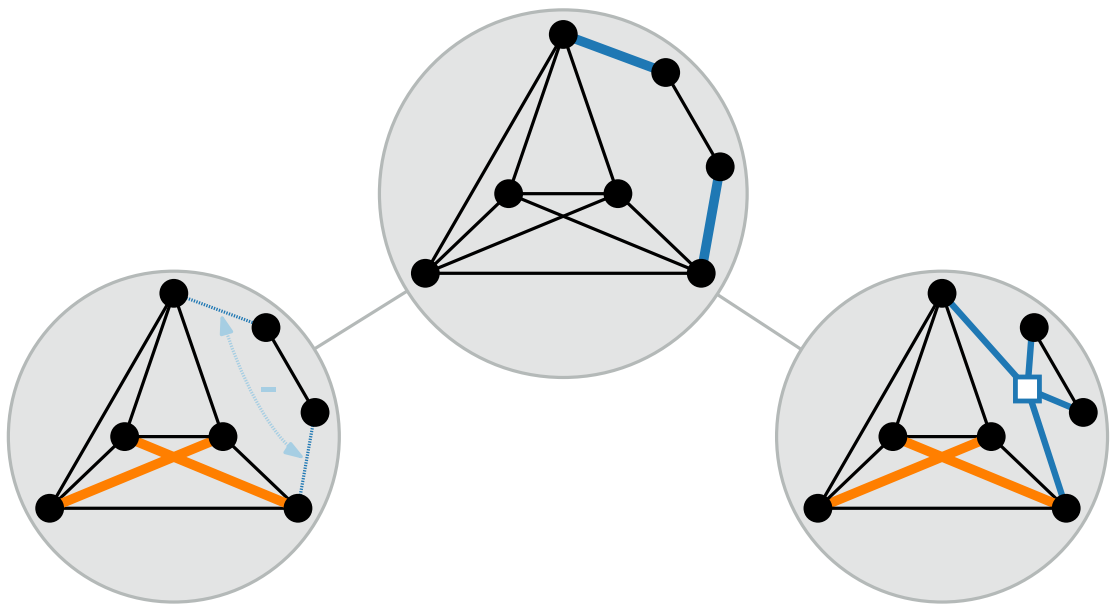
1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges
- Branch over whether to cross edge pairs in this order



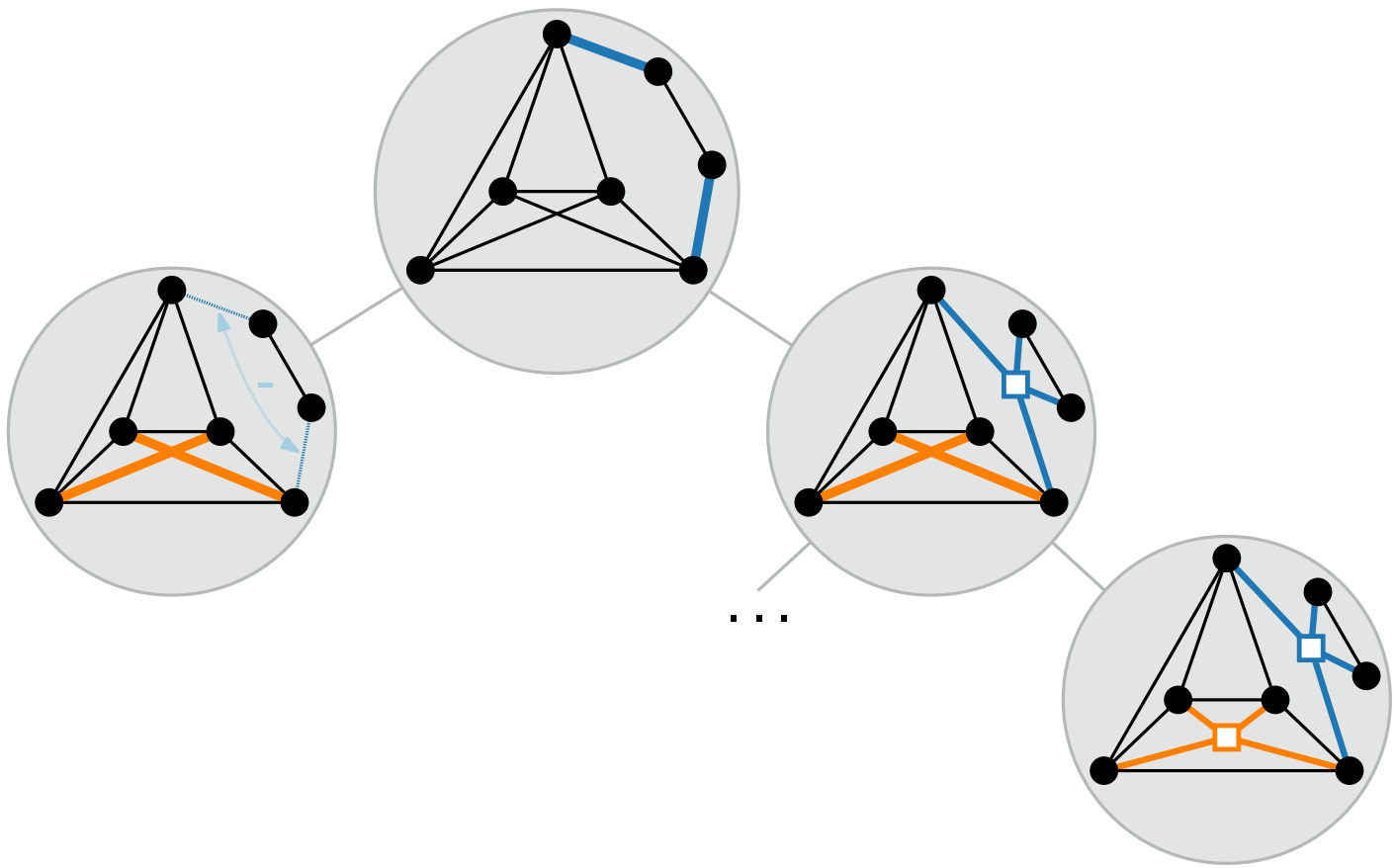
1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges
- Branch over whether to cross edge pairs in this order



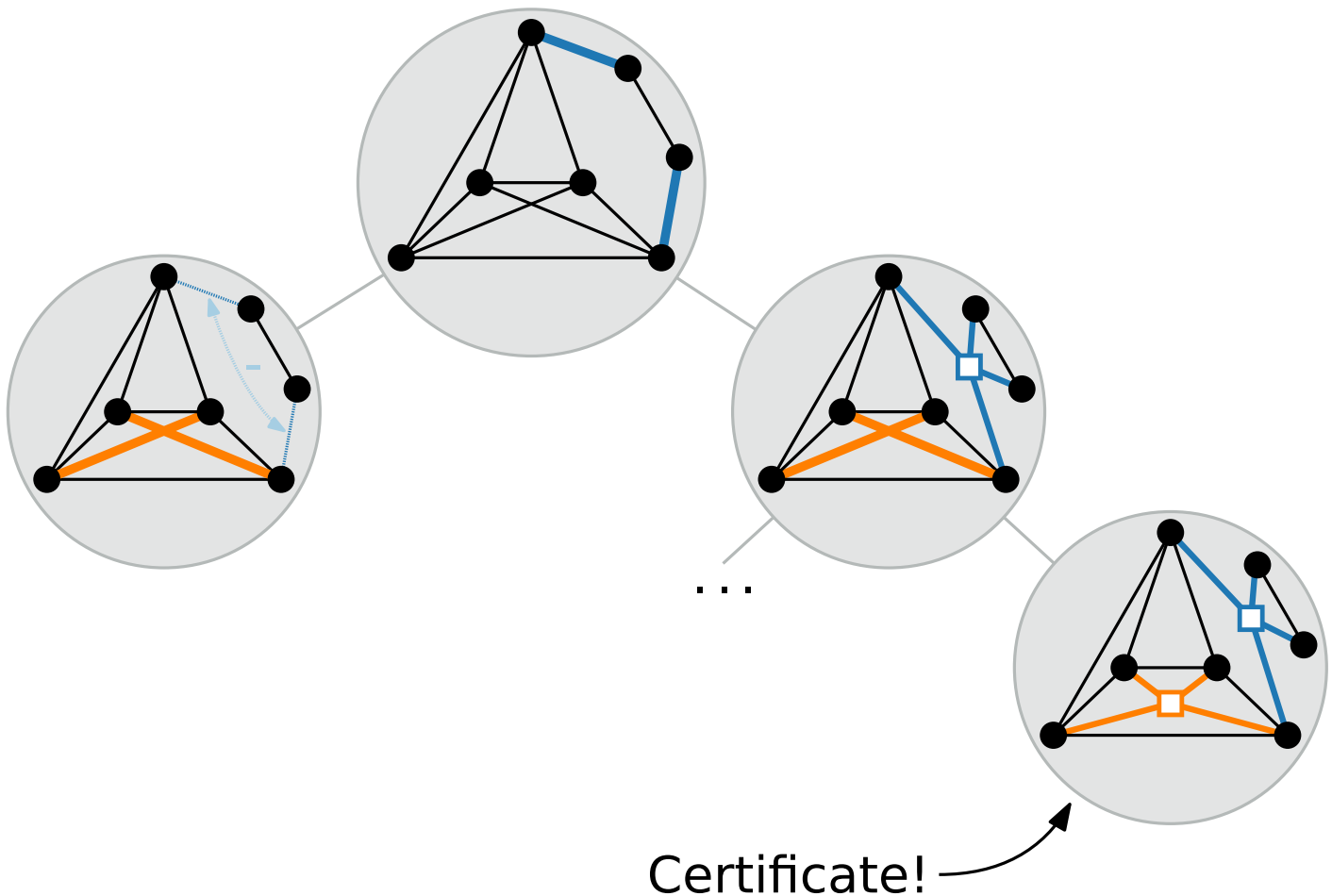
1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges
- Branch over whether to cross edge pairs in this order



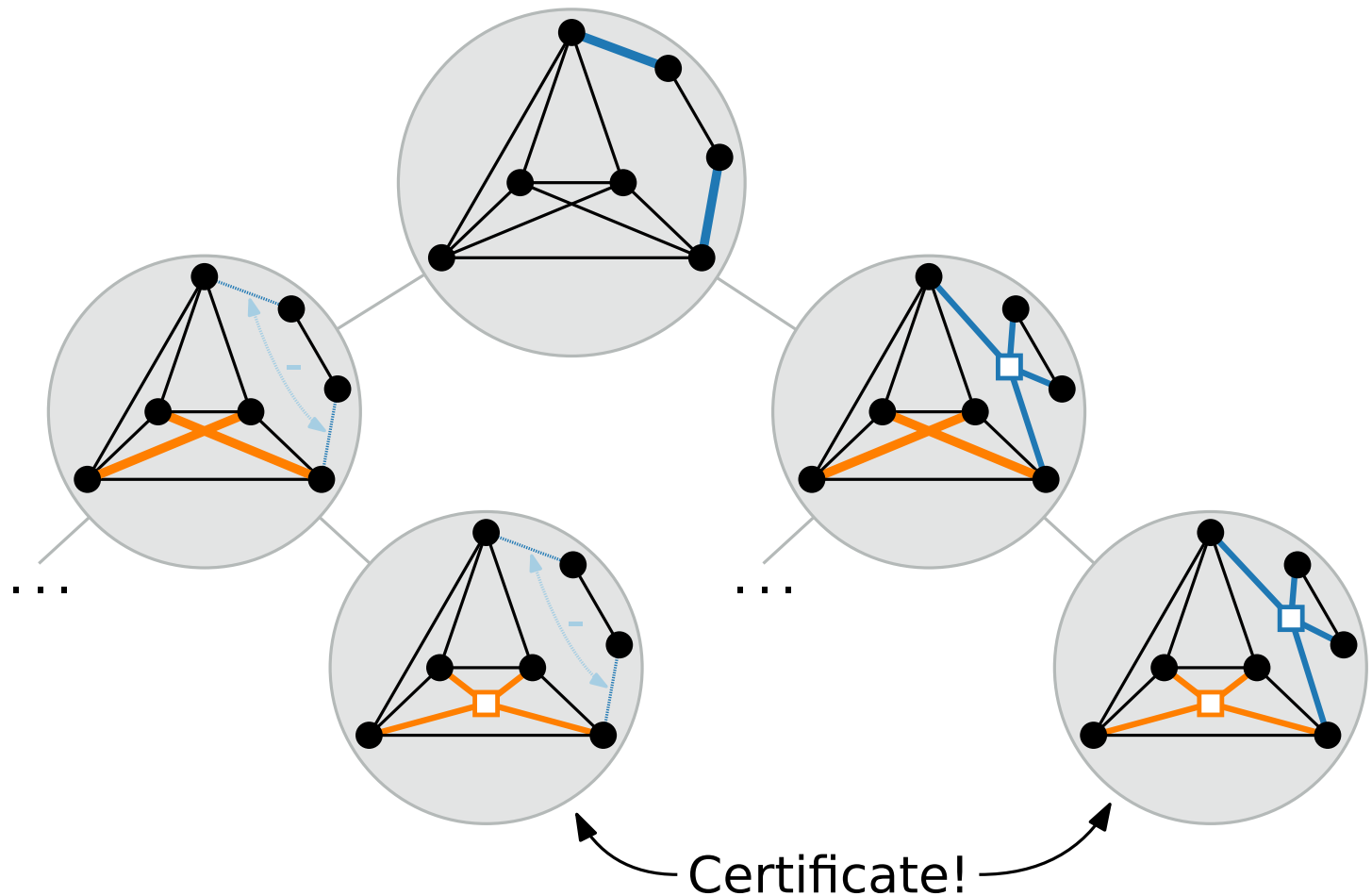
1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges
- Branch over whether to cross edge pairs in this order



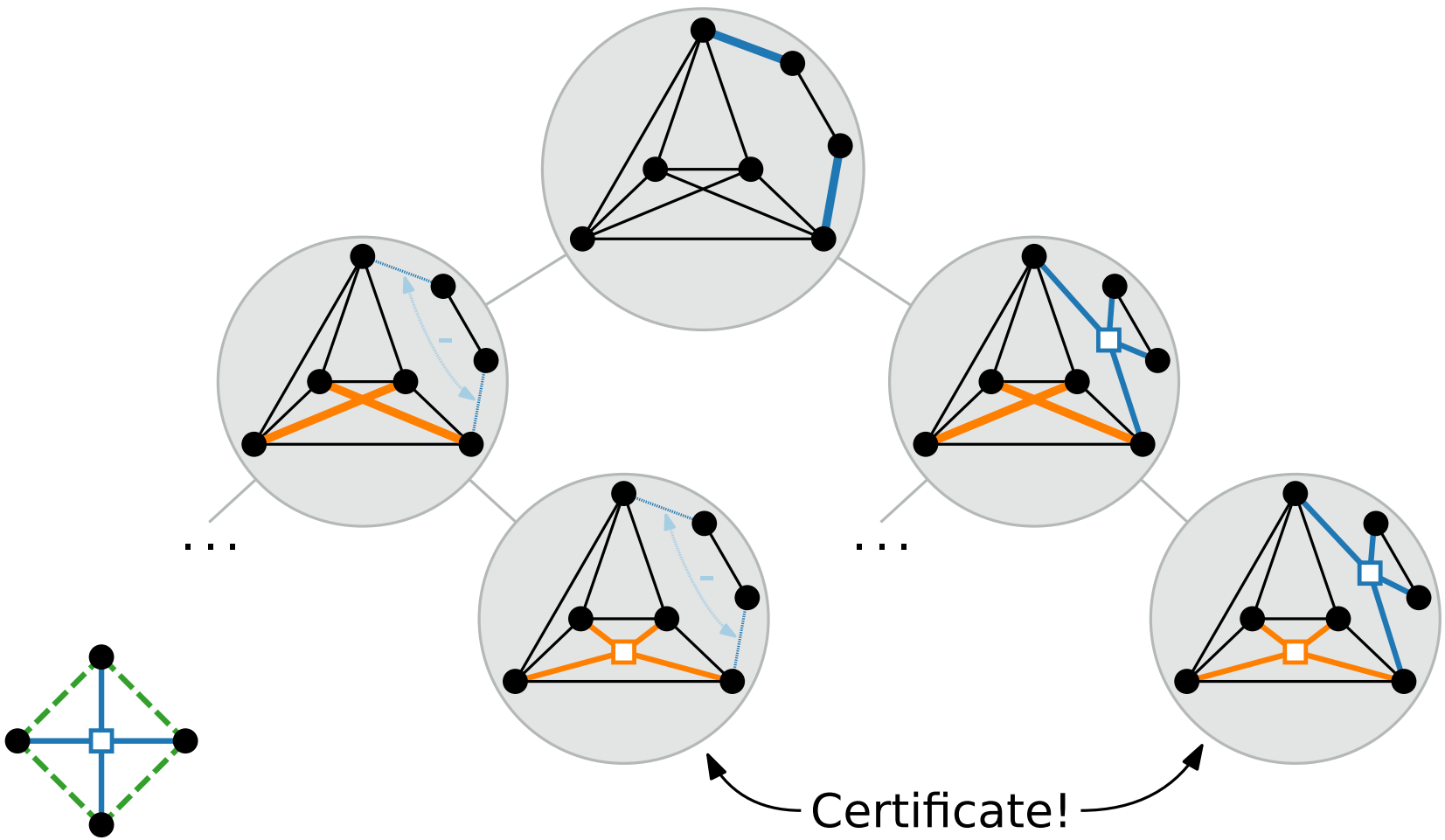
1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges [, , , . . .]
- Branch over whether to cross edge pairs in this order



1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

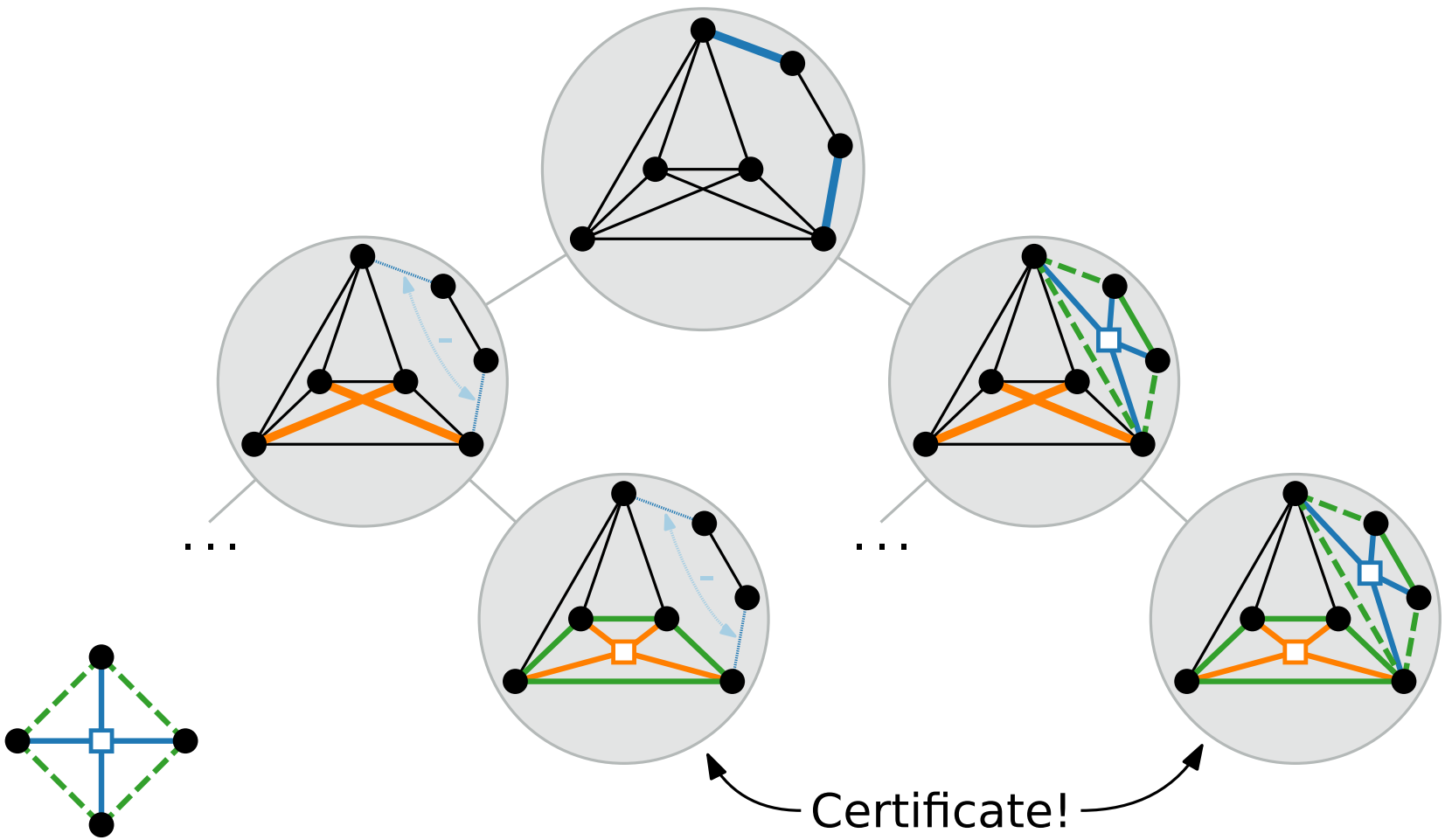
- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges
- Branch over whether to cross edge pairs in this order



- Additional optimization: insert uncrossable **kite edges** around crossings

1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

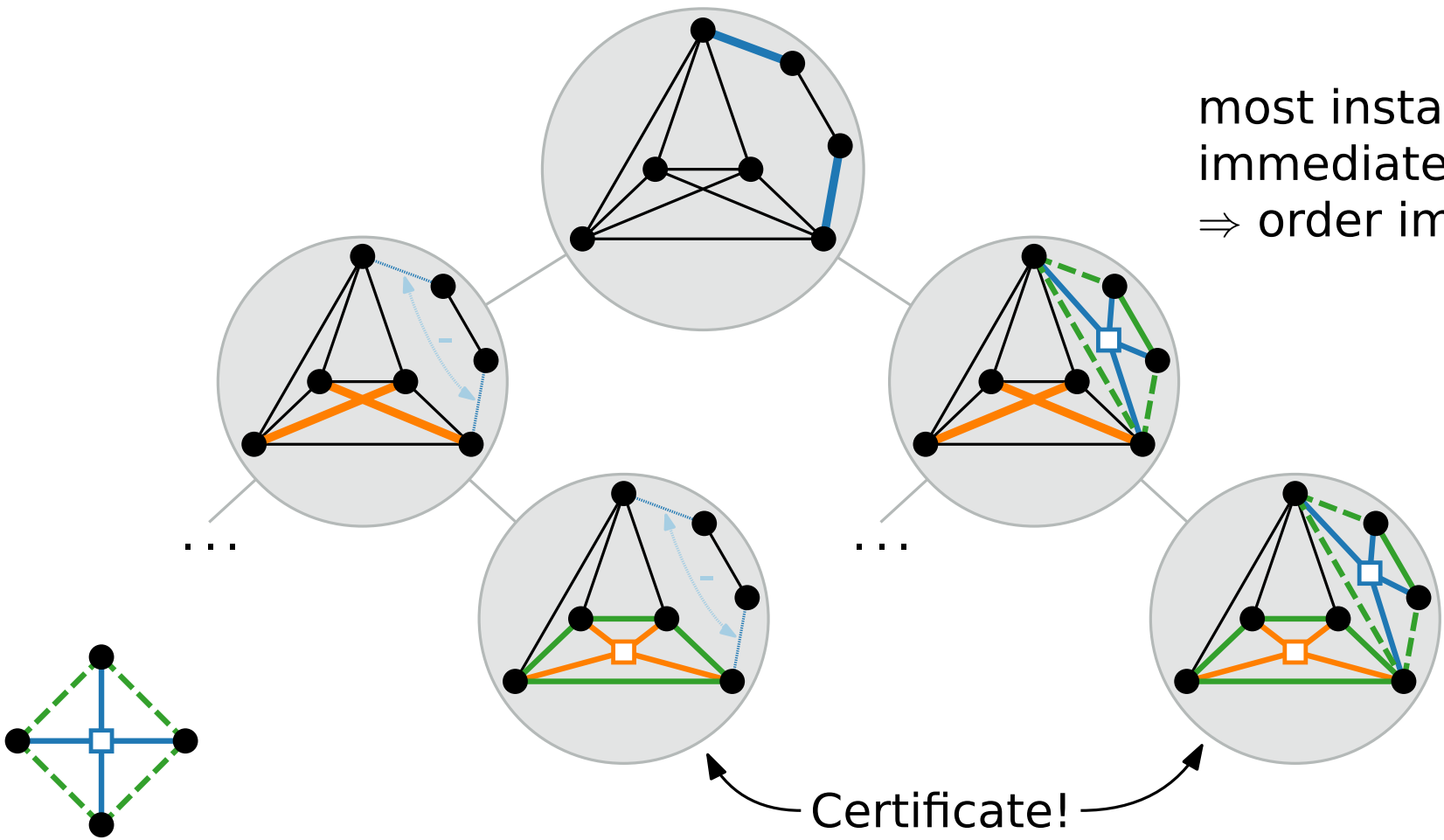
- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges
- Branch over whether to cross edge pairs in this order



- Additional optimization: insert uncrossable **kite edges** around crossings

1-Planarity: Backtracking [Binucci, Didimo, Montecchiani ’20]

- Decompose graph into biconnected components
- Fix arbitrary order for all pairs of non-adjacent edges [, , , . . .]
- Branch over whether to cross edge pairs in this order



- Additional optimization: insert uncrossable **kite edges** around crossings

Backtracking Ingredients

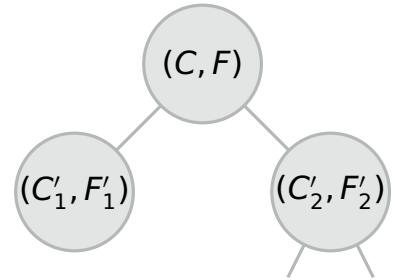
node in the search tree defines partial solution (C, F) with

- $C \subseteq \binom{E}{2}$ crossed edge pairs
- $F \subseteq \binom{E}{2}$ free (crossable) edge pairs

Backtracking Ingredients

node in the search tree defines partial solution (C, F) with

- $C \subseteq \binom{E}{2}$ crossed edge pairs
- $F \subseteq \binom{E}{2}$ free (crossable) edge pairs



1. Branching Strategy

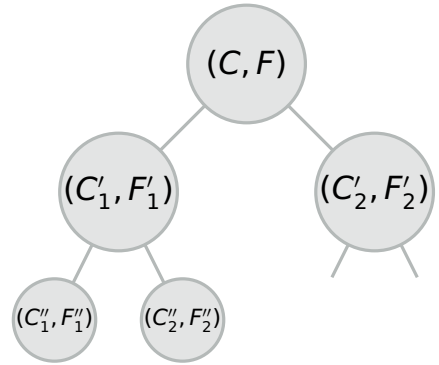
which edge pair(s) of F do we branch on next?

[Binucci et al. ’20]: fixed order

Backtracking Ingredients

node in the search tree defines partial solution (C, F) with

- $C \subseteq \binom{E}{2}$ crossed edge pairs
- $F \subseteq \binom{E}{2}$ free (crossable) edge pairs



1. Branching Strategy

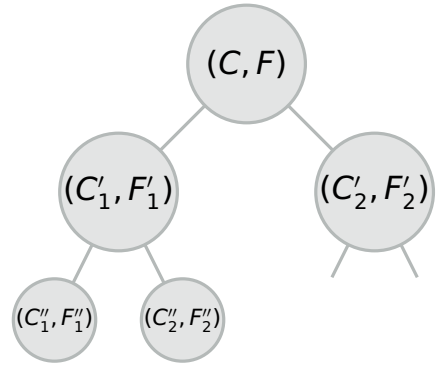
which edge pair(s) of F do we branch on next?

[Binucci et al. ’20]: fixed order

Backtracking Ingredients

node in the search tree defines partial solution (C, F) with

- $C \subseteq \binom{E}{2}$ crossed edge pairs
- $F \subseteq \binom{E}{2}$ free (crossable) edge pairs



1. Branching Strategy

which edge pair(s) of F do we branch on next?

[Binucci et al. ’20]: fixed order

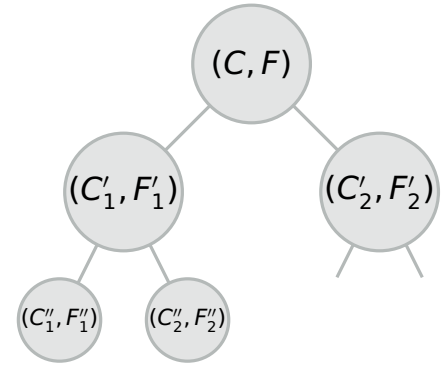
2. Filter Strategies

reject non-realizable partial solutions early

Backtracking Ingredients

node in the search tree defines partial solution (C, F) with

- $C \subseteq \binom{E}{2}$ crossed edge pairs
- $F \subseteq \binom{E}{2}$ free (crossable) edge pairs



1. Branching Strategy

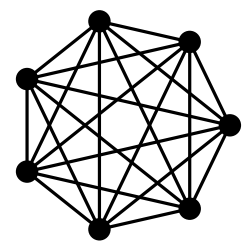
which edge pair(s) of F do we branch on next?

[Binucci et al. ’20]: fixed order

2. Filter Strategies

reject non-realizable partial solutions early

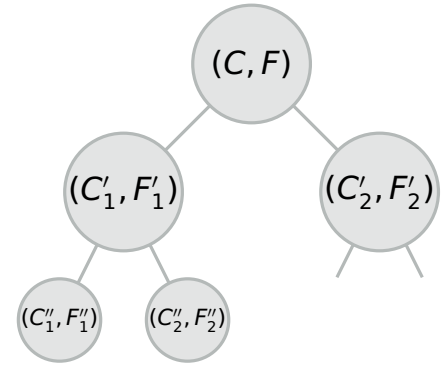
[Binucci et al. ’20]:
- edge density



Backtracking Ingredients

node in the search tree defines partial solution (C, F) with

- $C \subseteq \binom{E}{2}$ crossed edge pairs
- $F \subseteq \binom{E}{2}$ free (crossable) edge pairs



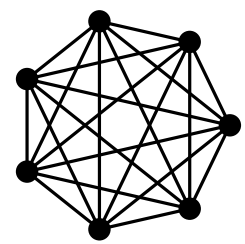
1. Branching Strategy

which edge pair(s) of F do we branch on next?

[Binucci et al. ’20]: fixed order

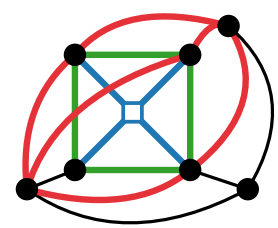
2. Filter Strategies

reject non-realizable partial solutions early



[Binucci et al. ’20]:

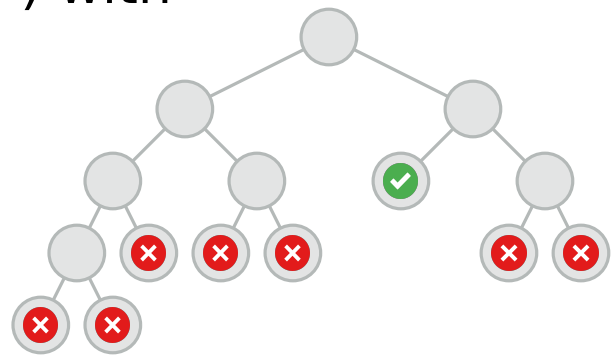
- edge density
- uncrossable subgraph not planar



Backtracking Ingredients

node in the search tree defines partial solution (C, F) with

- $C \subseteq \binom{E}{2}$ crossed edge pairs
- $F \subseteq \binom{E}{2}$ free (crossable) edge pairs

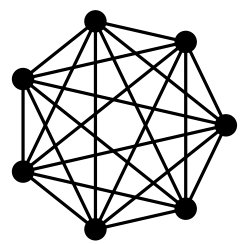


1. Branching Strategy

which edge pair(s) of F do we branch on next?

[Binucci et al. ’20]: fixed order

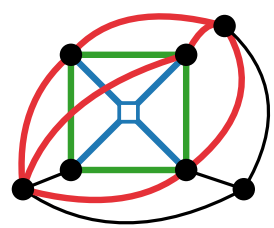
2. Filter Strategies



reject non-realizable partial solutions early

[Binucci et al. ’20]:

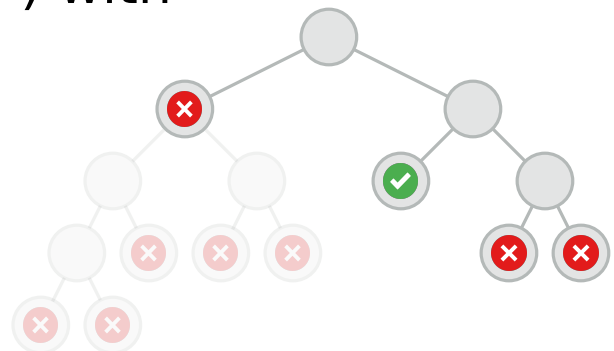
- edge density
- uncrossable subgraph not planar



Backtracking Ingredients

node in the search tree defines partial solution (C, F) with

- $C \subseteq \binom{E}{2}$ crossed edge pairs
- $F \subseteq \binom{E}{2}$ free (crossable) edge pairs



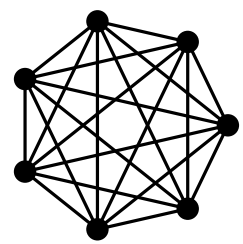
1. Branching Strategy

which edge pair(s) of F do we branch on next?

[Binucci et al. ’20]: fixed order

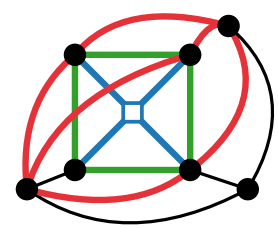
2. Filter Strategies

reject non-realizable partial solutions early



[Binucci et al. ’20]:

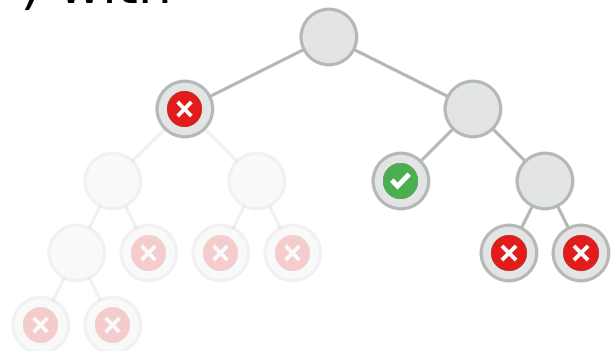
- edge density
- uncrossable subgraph not planar



Backtracking Ingredients

node in the search tree defines partial solution (C, F) with

- $C \subseteq \binom{E}{2}$ crossed edge pairs
- $F \subseteq \binom{E}{2}$ free (crossable) edge pairs

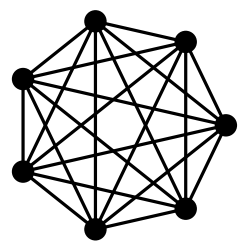


1. Branching Strategy

which edge pair(s) of F do we branch on next?

[Binucci et al. ’20]: fixed order

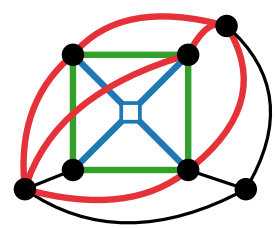
2. Filter Strategies



reject non-realizable partial solutions early

[Binucci et al. ’20]:

- edge density
- uncrossable subgraph not planar

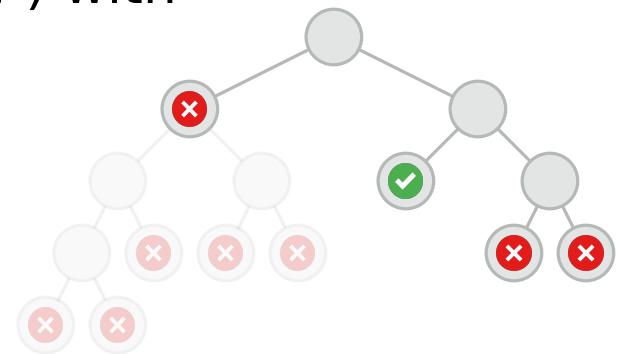


3. Traversal Strategy

Backtracking Ingredients

node in the search tree defines partial solution (C, F) with

- $C \subseteq \binom{E}{2}$ crossed edge pairs
- $F \subseteq \binom{E}{2}$ free (crossable) edge pairs

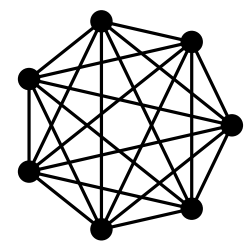


1. Branching Strategy

which edge pair(s) of F do we branch on next?

[Binucci et al. ’20]: fixed order

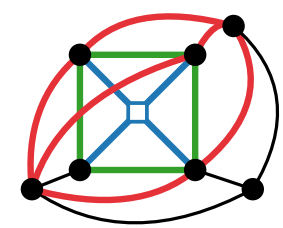
2. Filter Strategies



reject non-realizable partial solutions early

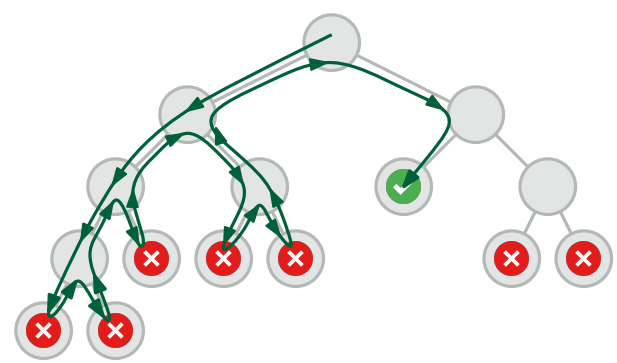
[Binucci et al. ’20]:

- edge density
- uncrossable subgraph not planar

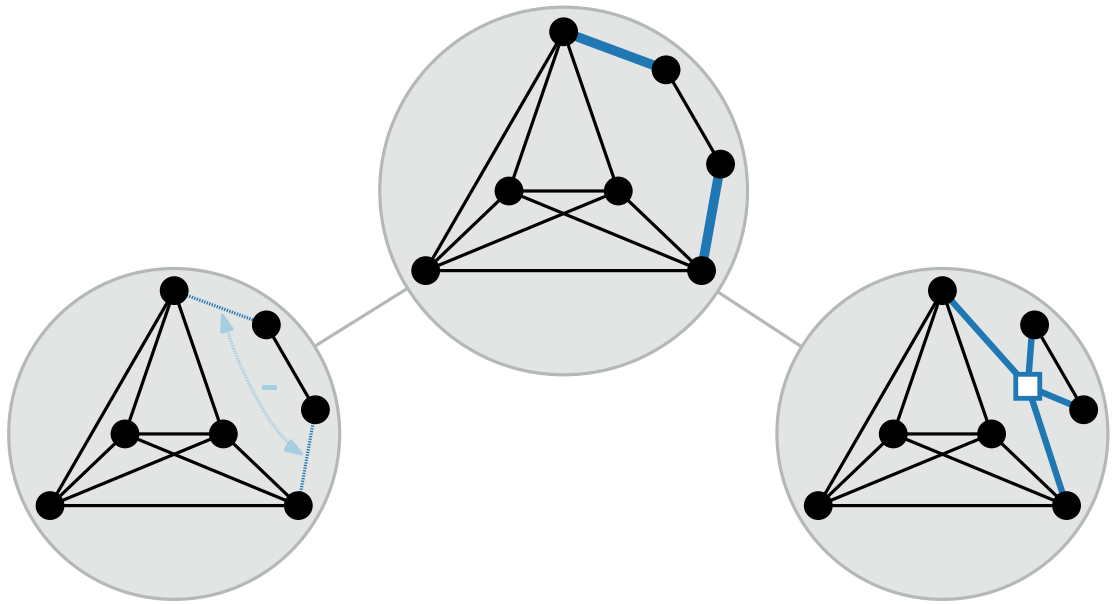


3. Traversal Strategy

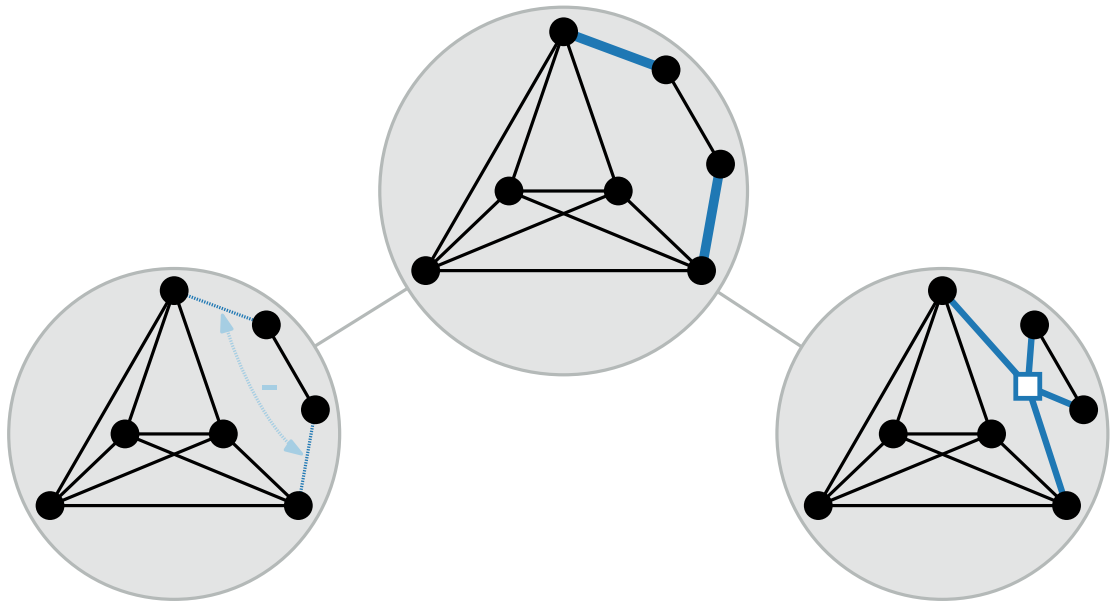
[Binucci et al. ’20]: DFS



Branching Strategies

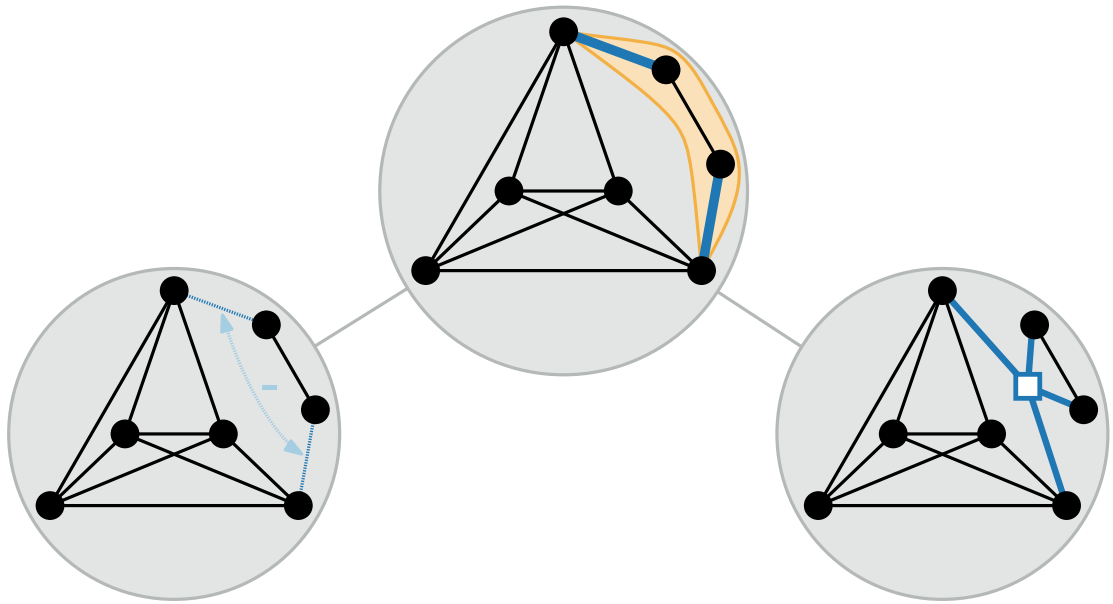


Branching Strategies



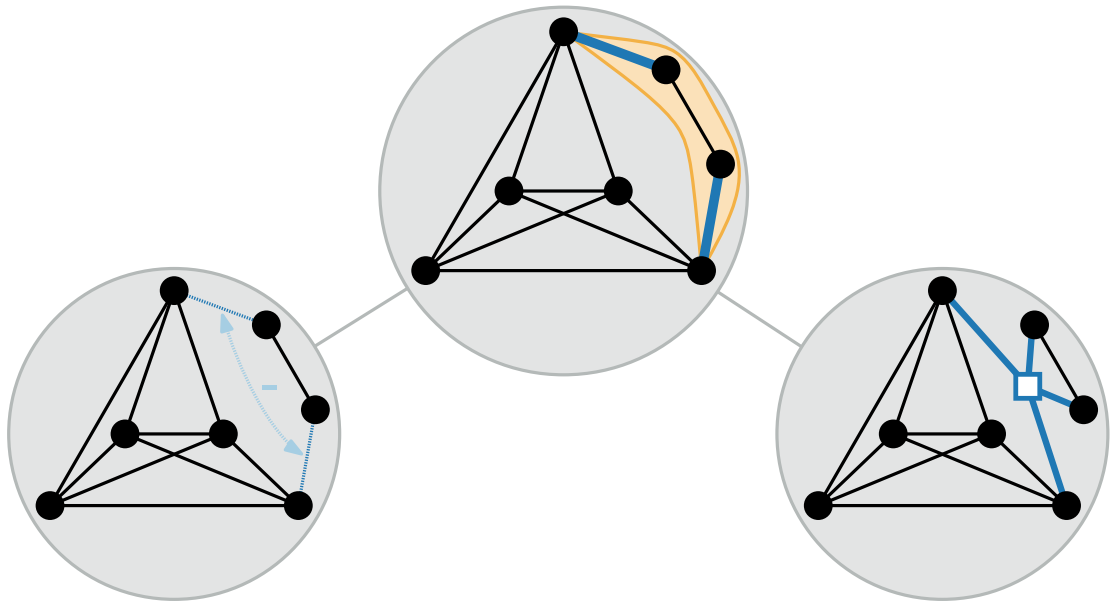
Observation: crossing an edge pair only helps if it removes a $K_5/K_{3,3}$

Branching Strategies



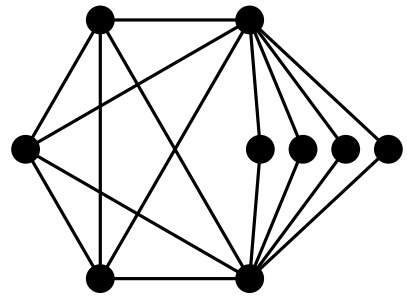
Observation: crossing an edge pair only helps if it removes a $K_5/K_{3,3}$

Branching Strategies

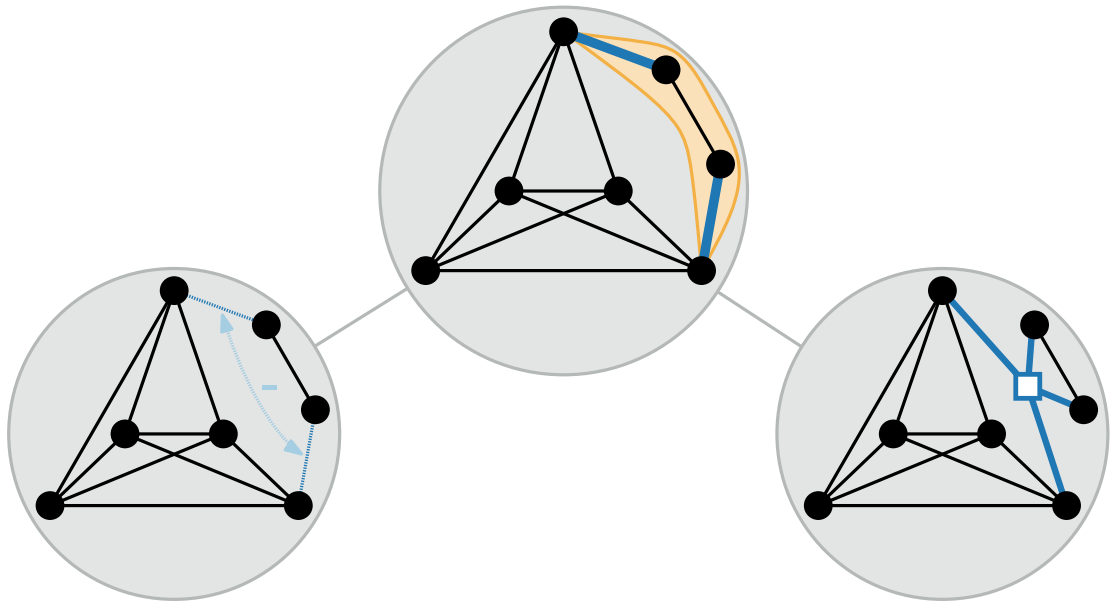


Observation: crossing an edge pair only helps if it removes a $K_5/K_{3,3}$

Strategy 1:
- extract multiple Kuratowski-subdivisions



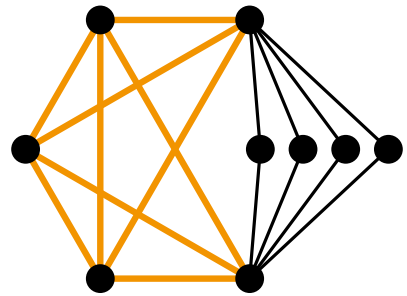
Branching Strategies



Observation: crossing an edge pair only helps if it removes a $K_5/K_{3,3}$

Strategy 1:

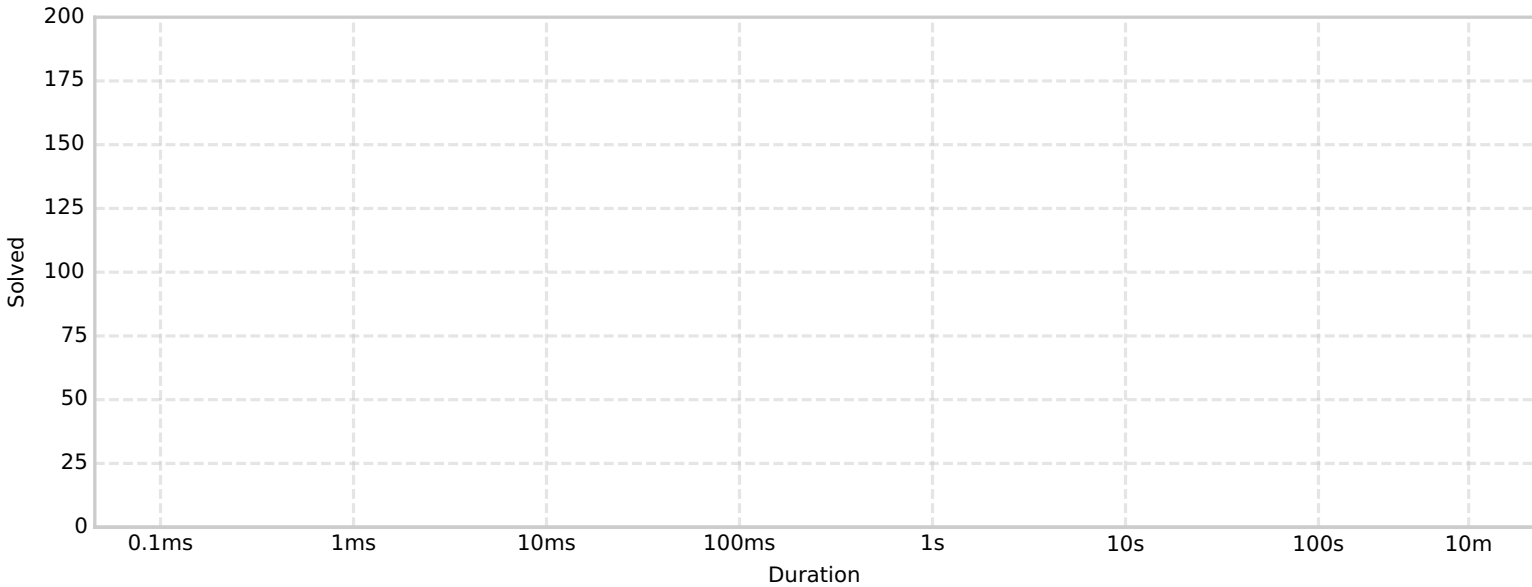
- extract multiple Kuratowski-subdivisions
- prioritize edge pairs contained in many of them



Branching Strategies

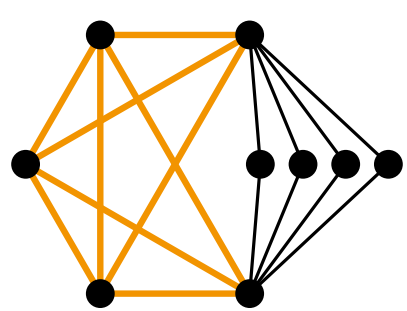
Solved instances over time:

- 200 non-planar bicomps from North/Rome
- 10min time limit



Strategy 1:

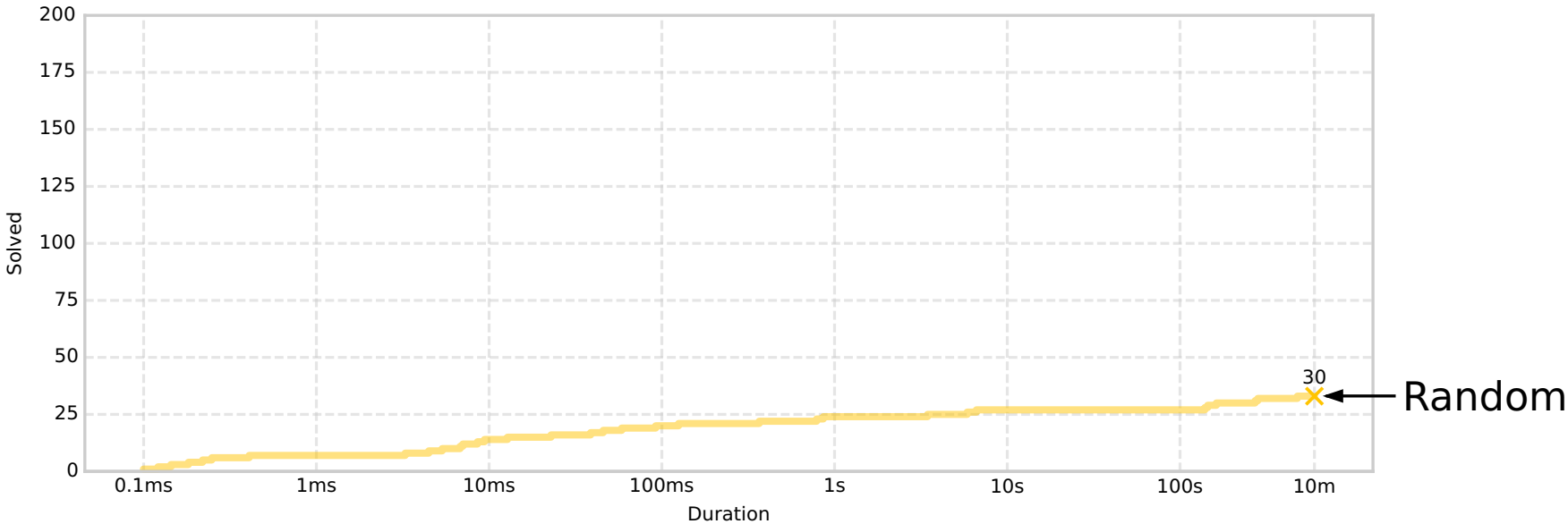
- extract multiple Kuratowski-subdivisions
- prioritize edge pairs contained in many of them



Branching Strategies

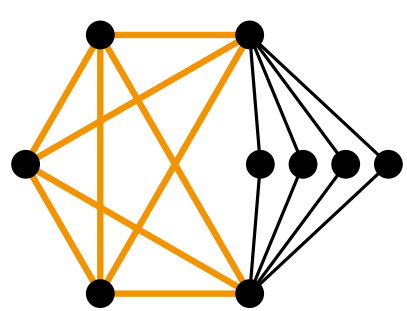
Solved instances over time:

- 200 non-planar bicomps from North/Rome
- 10min time limit



Strategy 1:

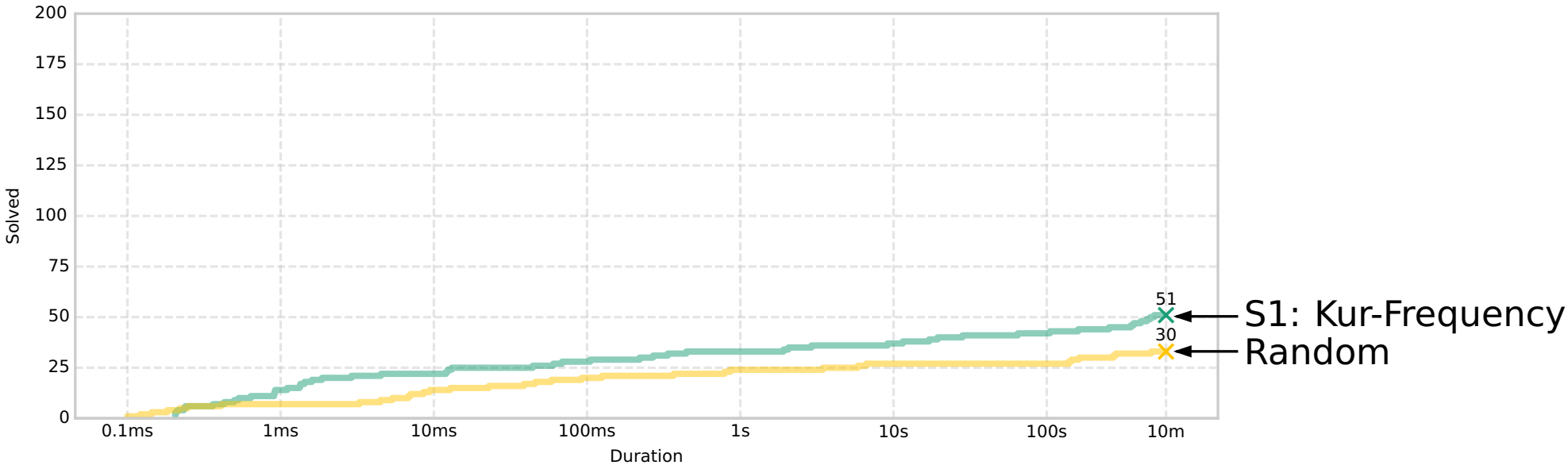
- extract multiple Kuratowski-subdivisions
- prioritize edge pairs contained in many of them



Branching Strategies

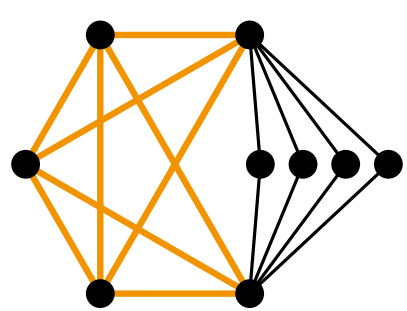
Solved instances over time:

- 200 non-planar bicomps from North/Rome
- 10min time limit

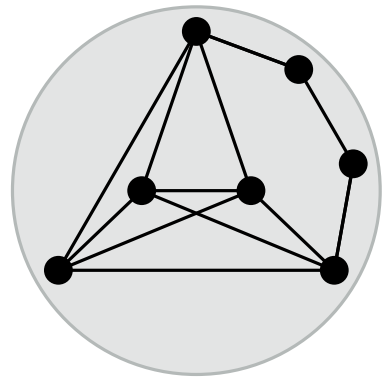


Strategy 1:

- extract multiple Kuratowski-subdivisions
- prioritize edge pairs contained in many of them



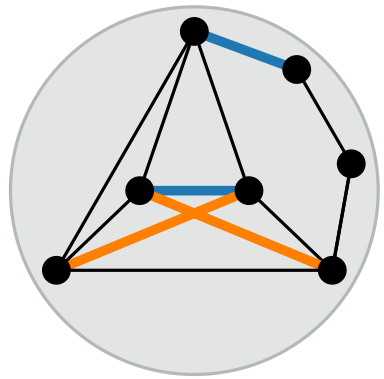
Branching Strategies



Strategy 2:

- pick arbitrary Kuratowski-subdivision $\mathcal{K}$

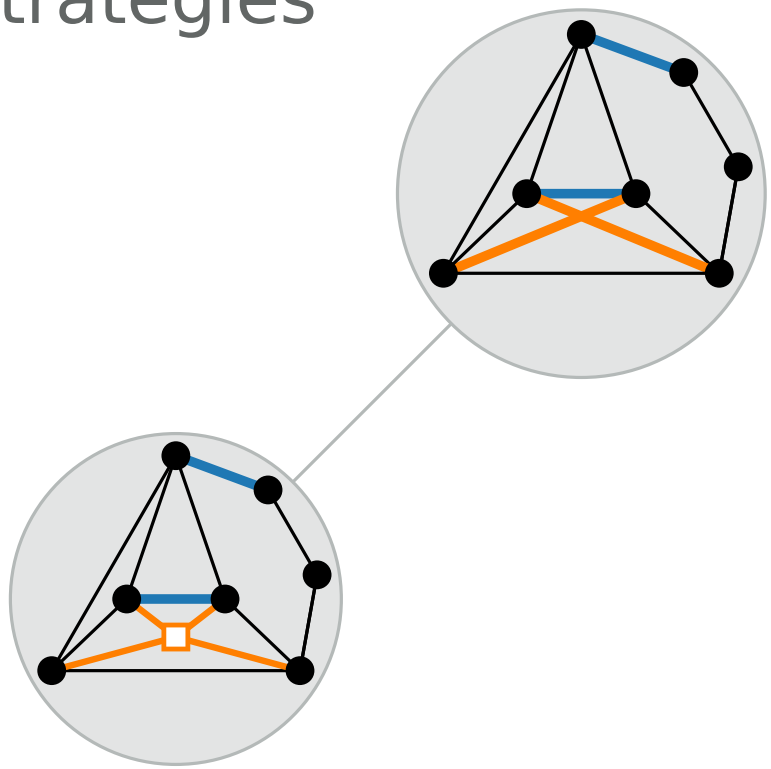
Branching Strategies



Strategy 2:

- pick arbitrary Kuratowski-subdivision $\mathcal{K}$
- branch over crossable independent edge pairs of $\mathcal{K}$

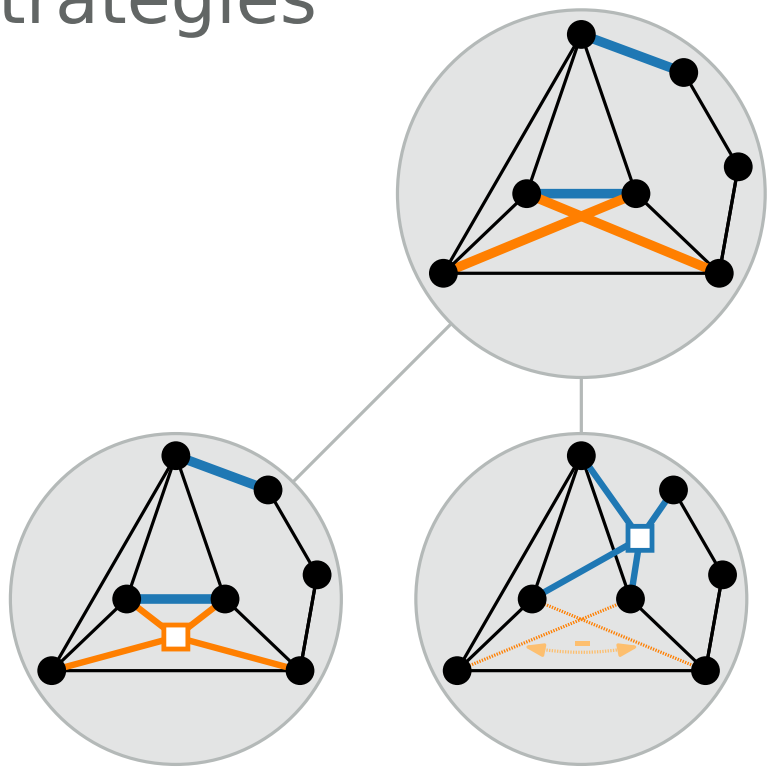
Branching Strategies



Strategy 2:

- pick arbitrary Kuratowski-subdivision $\mathcal{K}$
- branch over crossable independent edge pairs of $\mathcal{K}$

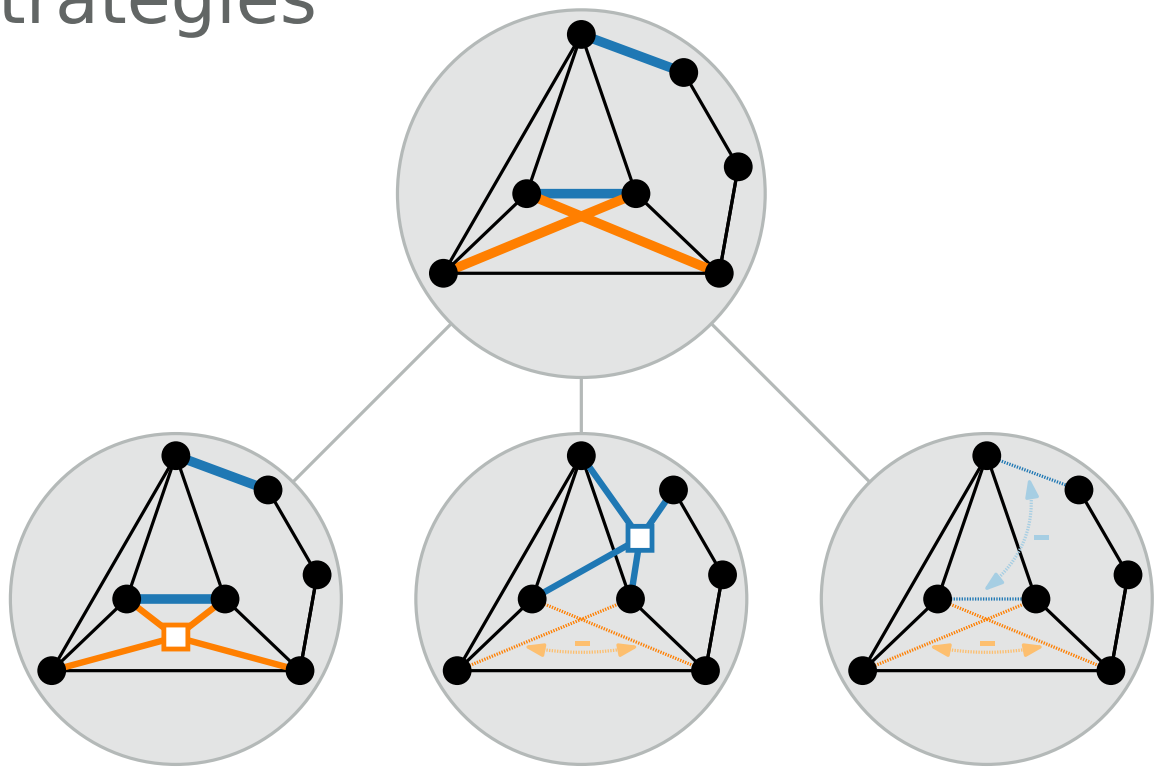
Branching Strategies



Strategy 2:

- pick arbitrary Kuratowski-subdivision $\mathcal{K}$
- branch over crossable independent edge pairs of $\mathcal{K}$

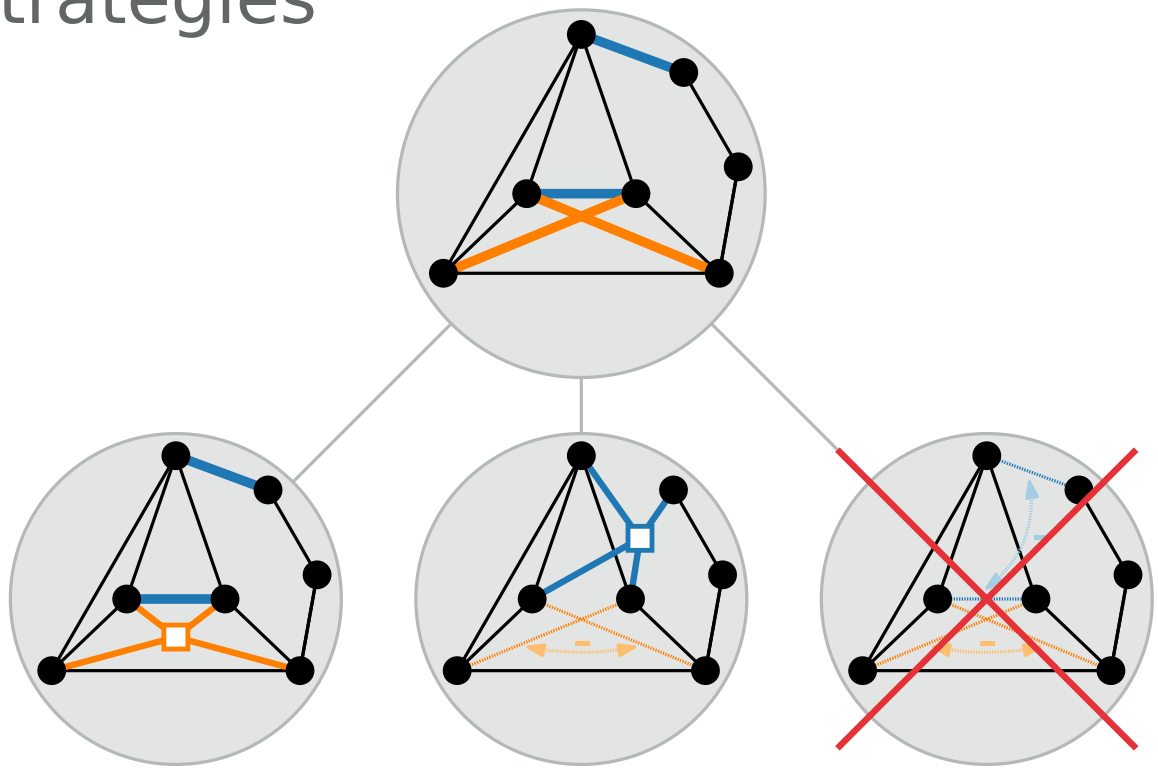
Branching Strategies



Strategy 2:

- pick arbitrary Kuratowski-subdivision $\mathcal{K}$
- branch over crossable independent edge pairs of $\mathcal{K}$

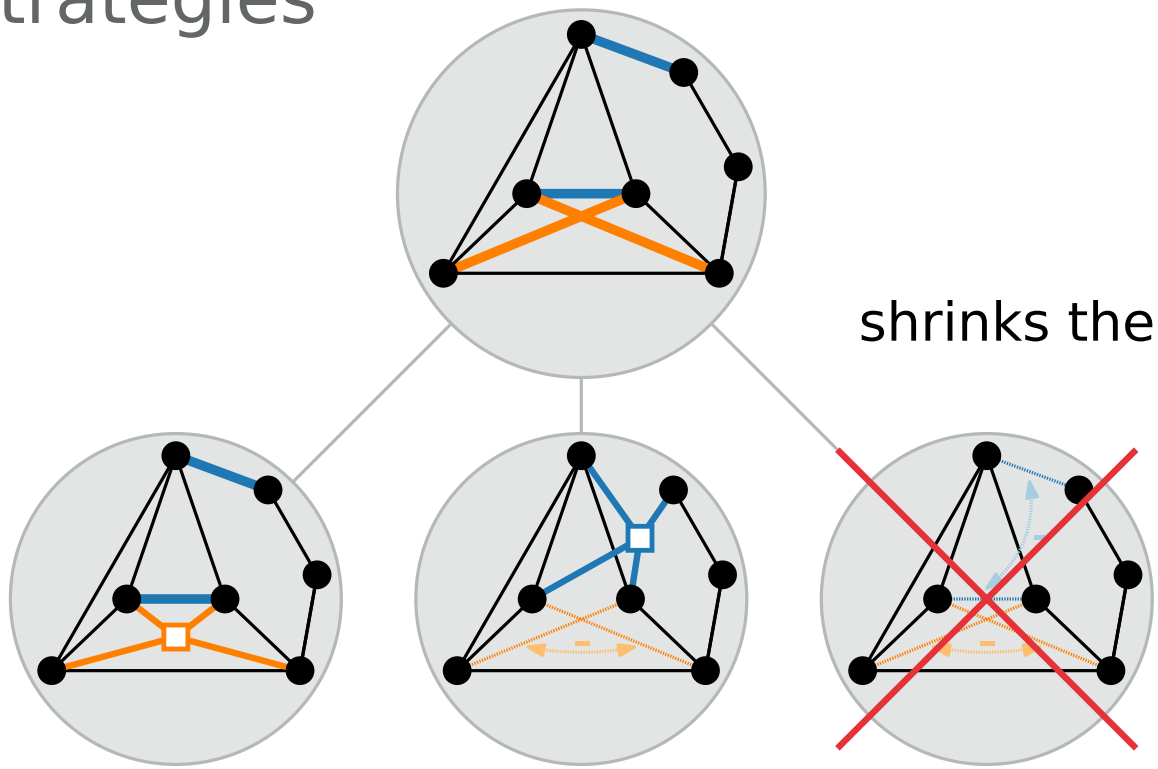
Branching Strategies



Strategy 2:

- pick arbitrary Kuratowski-subdivision $\mathcal{K}$
- branch over crossable independent edge pairs of $\mathcal{K}$
- omit branch where no edge pair of $\mathcal{K}$ is crossed

Branching Strategies



shrinks the search space!

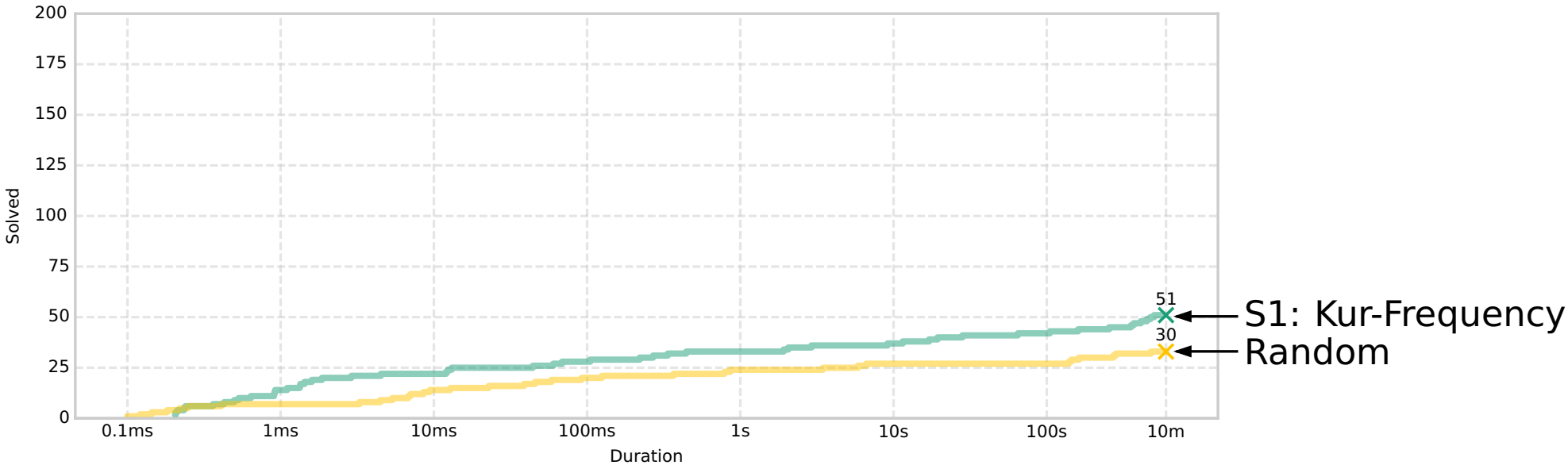
Strategy 2:

- pick arbitrary Kuratowski-subdivision $\mathcal{K}$
- branch over crossable independent edge pairs of $\mathcal{K}$
- omit branch where no edge pair of $\mathcal{K}$ is crossed

Branching Strategies

Solved instances over time:

- 200 non-planar bicomps from North/Rome
- 10min time limit



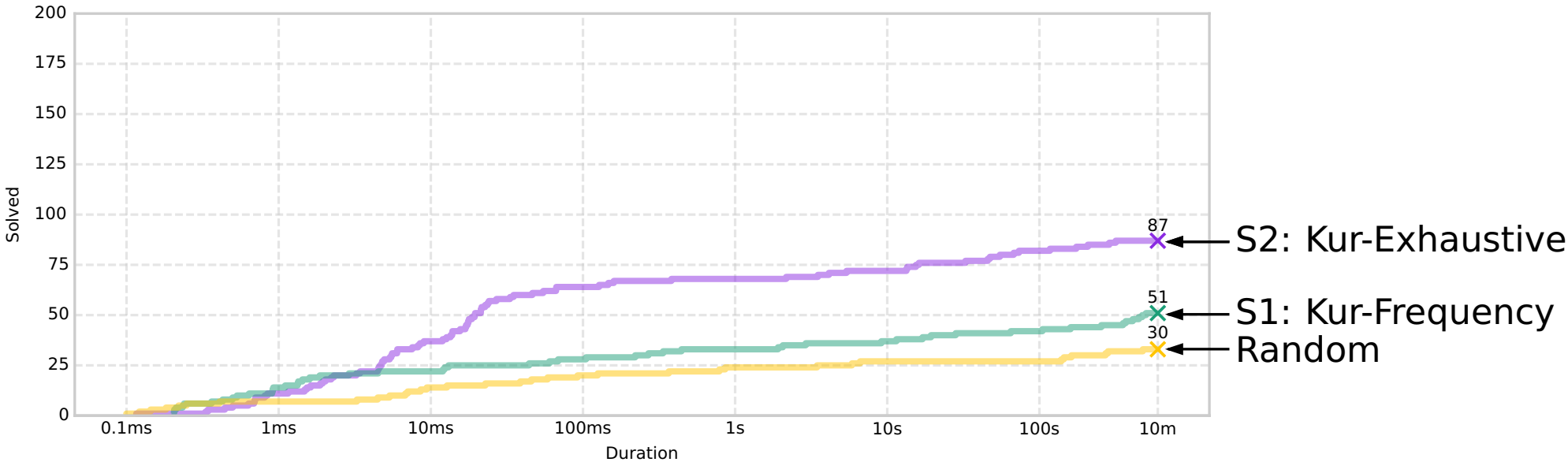
Strategy 2:

- pick arbitrary Kuratowski-subdivision $\mathcal{K}$
- branch over crossable independent edge pairs of $\mathcal{K}$
- omit branch where no edge pair of $\mathcal{K}$ is crossed

Branching Strategies

Solved instances over time:

- 200 non-planar bicomps from North/Rome
- 10min time limit

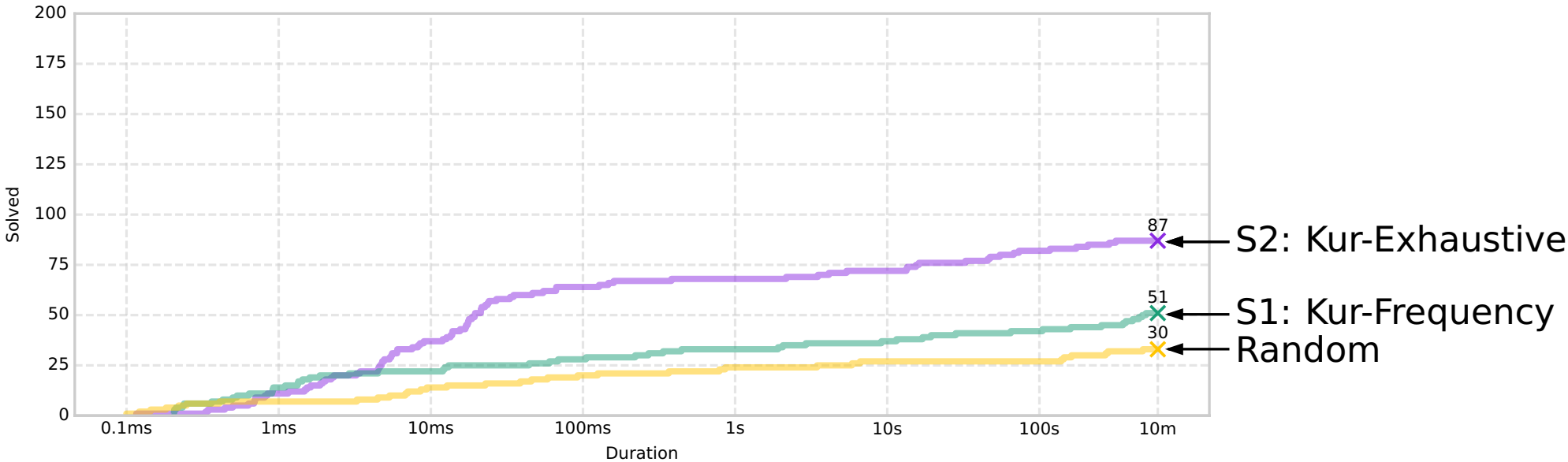


Strategy 2:

- pick arbitrary Kuratowski-subdivision $\mathcal{K}$
- branch over crossable independent edge pairs of $\mathcal{K}$
- omit branch where no edge pair of $\mathcal{K}$ is crossed

Branching Strategies

Solved instances over time:
- 200 non-planar bicomps from North/Rome
- 10min time limit

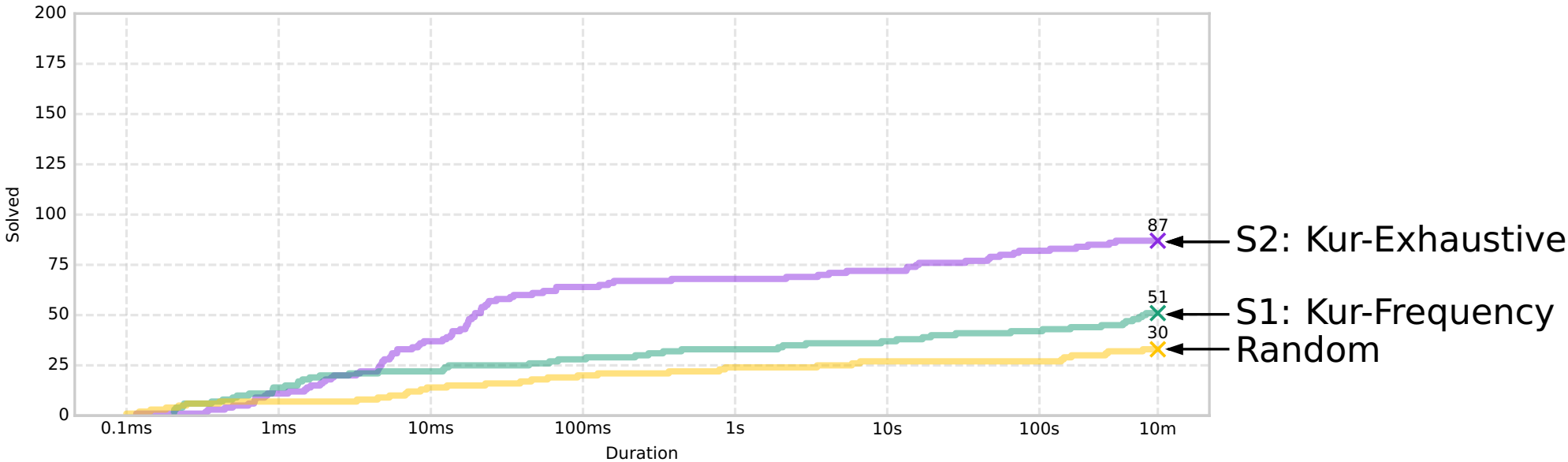


Strategy 3:
- extract multiple Kuratowski-subdivisions

Branching Strategies

Solved instances over time:

- 200 non-planar bicomps from North/Rome
- 10min time limit



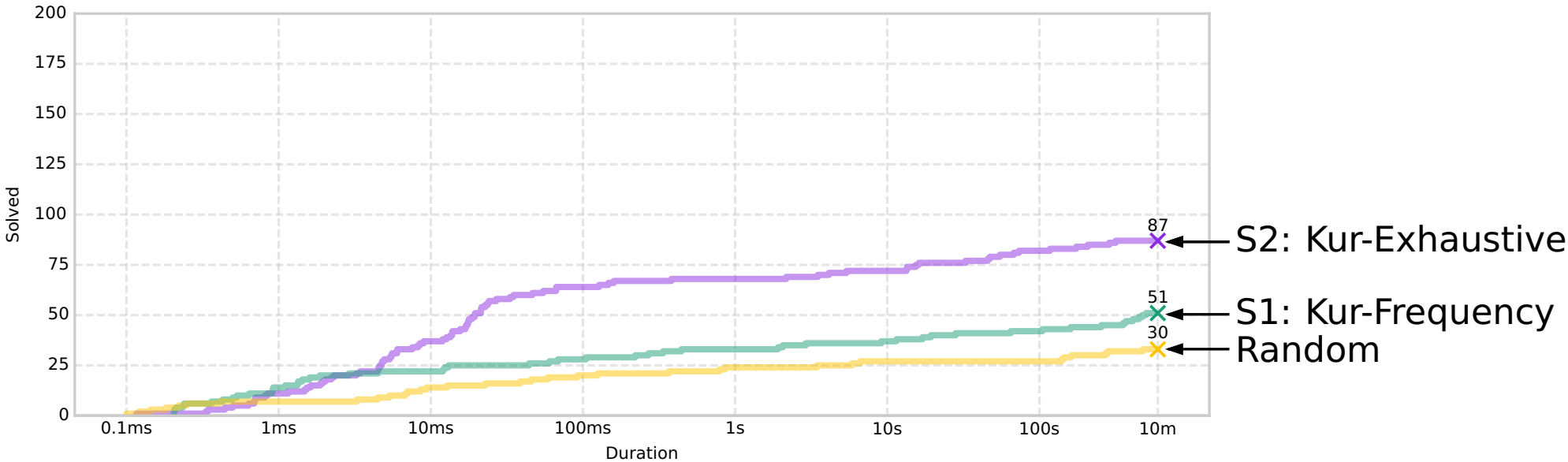
Strategy 3:

- extract multiple Kuratowski-subdivisions
- branch over subdivision with fewest crossable edge pairs ($\rightarrow$ fewer children)

Branching Strategies

Solved instances over time:

- 200 non-planar bicomps from North/Rome
- 10min time limit



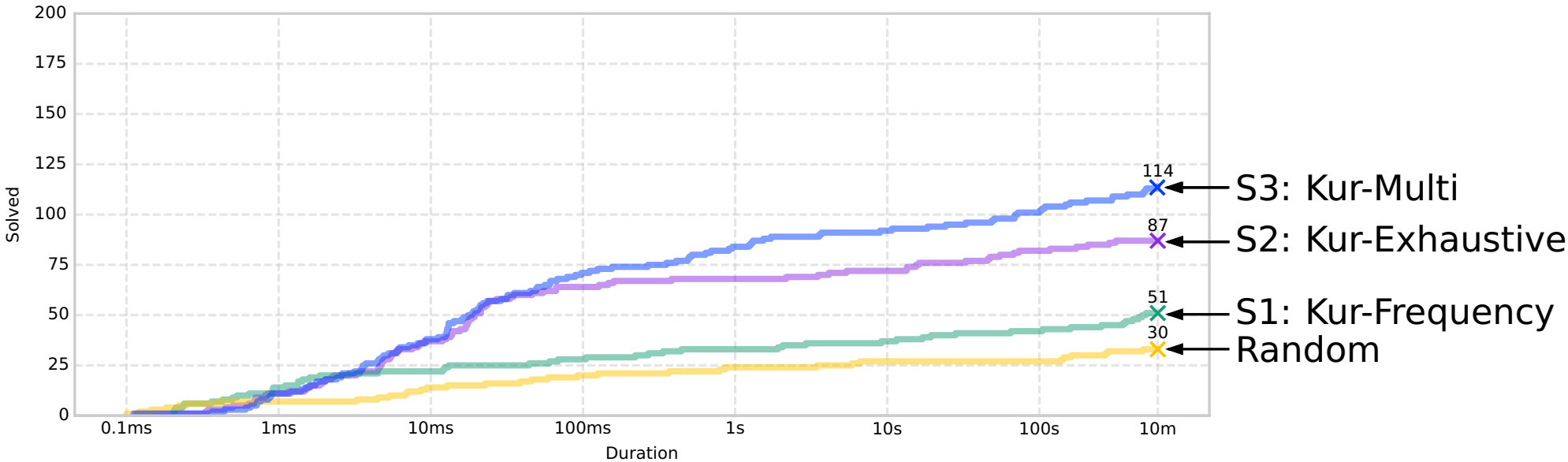
Strategy 3:

- extract multiple Kuratowski-subdivisions
- branch over subdivision with fewest crossable edge pairs ($\rightarrow$ fewer children)
- prioritize edge pairs that appear in many other subdivisions

Branching Strategies

Solved instances over time:

- 200 non-planar bicomps from North/Rome
- 10min time limit



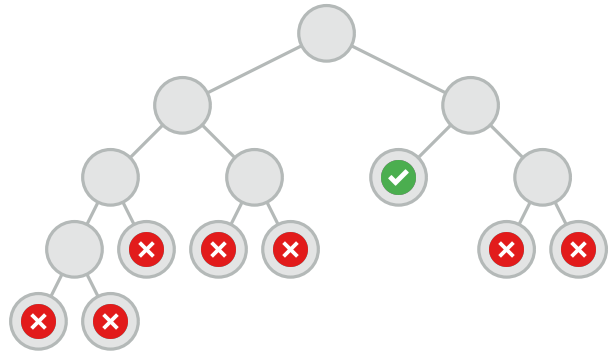
Strategy 3:

- extract multiple Kuratowski-subdivisions
- branch over subdivision with fewest crossable edge pairs ($\rightarrow$ fewer children)
- prioritize edge pairs that appear in many other subdivisions

Filter Strategies

Given: Graph G, partial solution (C, F)

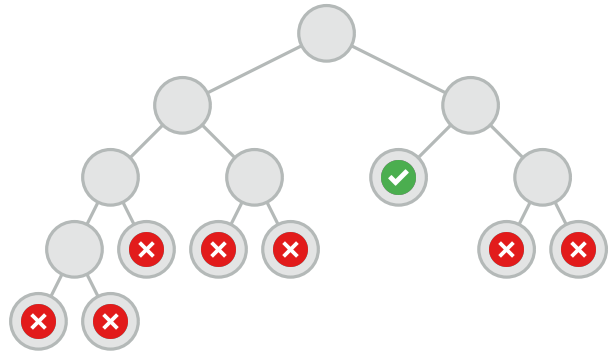
crossed free



Filter Strategies

Given: Graph G, partial solution (C, F) (crossed, free)

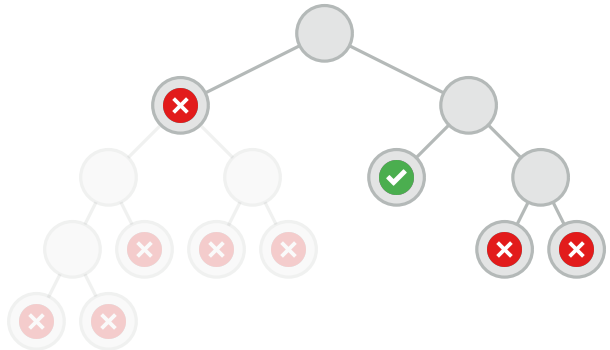
Goal: reject non-realizable partial solutions early



Filter Strategies

Given: Graph G, partial solution (C, F) (crossed, free)

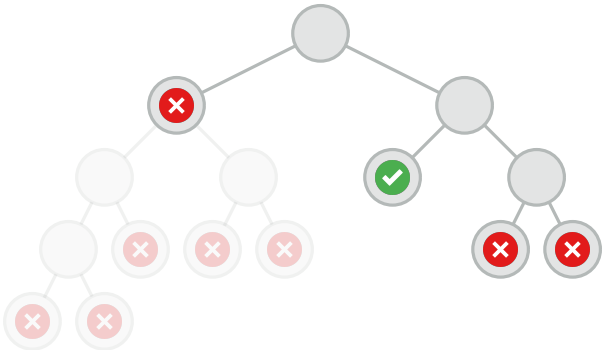
Goal: reject non-realizable partial solutions early

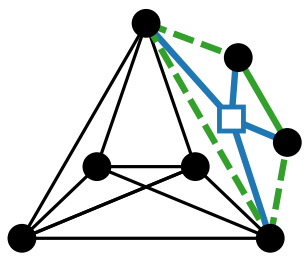


Filter Strategies

Given: Graph G, partial solution (C, F) (crossed, free)

Goal: reject non-realizable partial solutions early



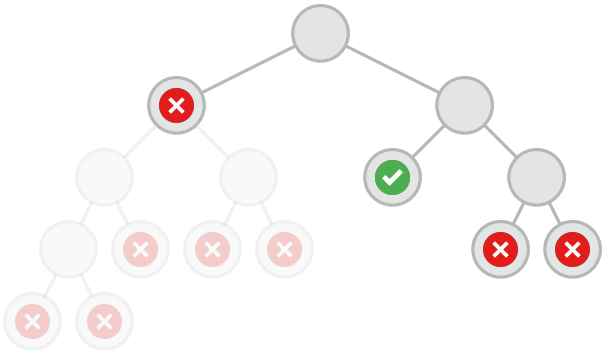


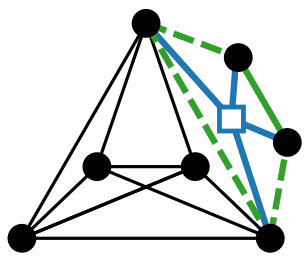
G_C := add crossing vertices + kite edges

Filter Strategies

Given: Graph G, partial solution (C, F) (crossed, free)

Goal: reject non-realizable partial solutions early





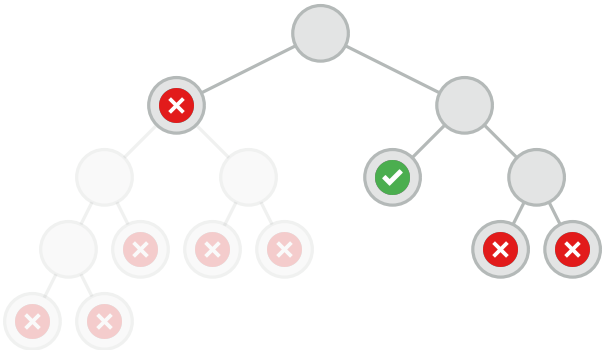
G_C := add crossing vertices + kite edges

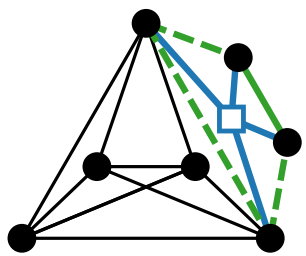
edge of G_C is **free** if contained in some pair of F, otherwise **saturated**

Filter Strategies

Given: Graph G, partial solution (C, F) (crossed, free)

Goal: reject non-realizable partial solutions early

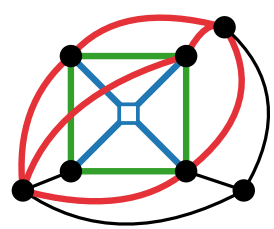




G_C := add crossing vertices + kite edges

edge of G_C is **free** if contained in some pair of F, otherwise **saturated**

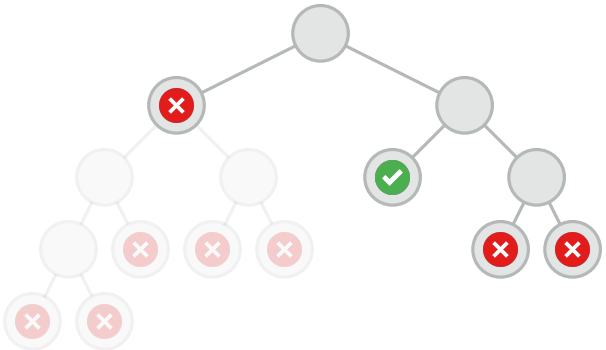
Planarity Filter [Binucci et al. '20]:
Reject if saturated subgraph of G_C not planar

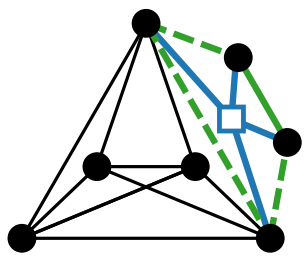


Filter Strategies

Given: Graph G, partial solution (C, F) (crossed, free)

Goal: reject non-realizable partial solutions early



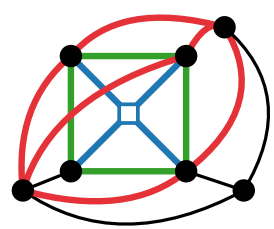


G_C := add crossing vertices + kite edges

edge of G_C is **free** if contained in some pair of F, otherwise **saturated**

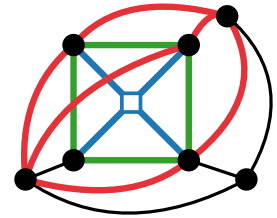
Planarity Filter [Binucci et al. '20]:
Reject if saturated subgraph of G_C not planar

can only happen deep in the search tree 😕



Filter Strategies

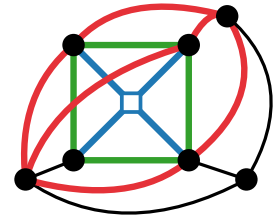
Planarity Filter [Binucci et al. ’20]:
Reject if saturated subgraph of G_C not planar



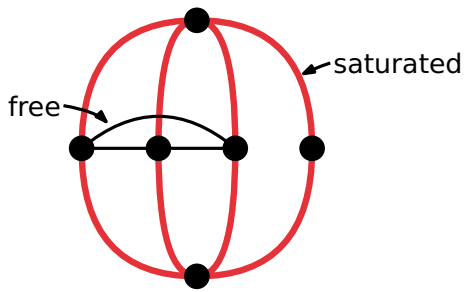
stronger version: also include free edges, allow them to cross arbitrarily

Filter Strategies

Planarity Filter [Binucci et al. ’20]:
Reject if saturated subgraph of G_C not planar

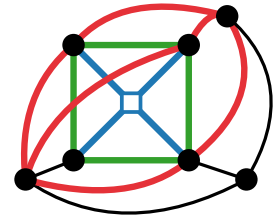


stronger version: also include free edges, allow them to cross arbitrarily

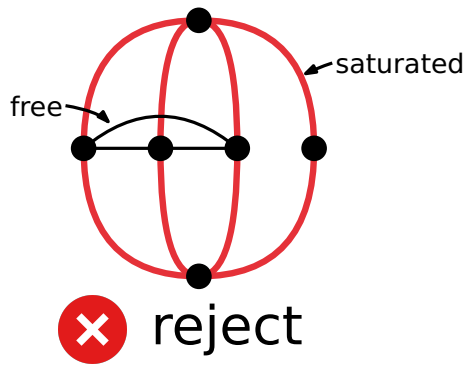


Filter Strategies

Planarity Filter [Binucci et al. ’20]:
Reject if saturated subgraph of G_C not planar

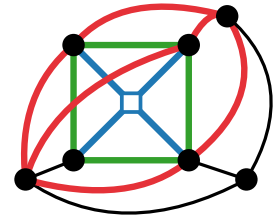


stronger version: also include free edges, allow them to cross arbitrarily

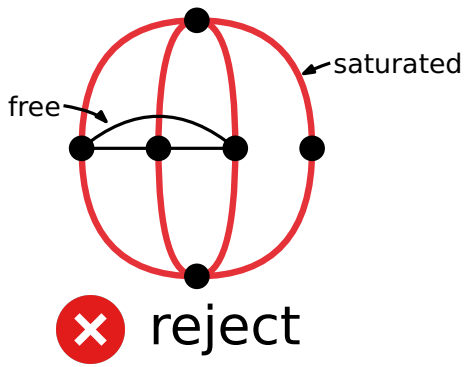


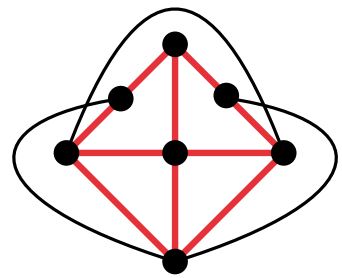
Filter Strategies

Planarity Filter [Binucci et al. ’20]:
Reject if saturated subgraph of G_C not planar



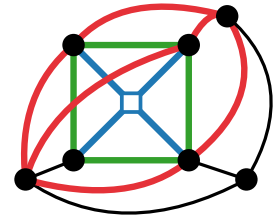
stronger version: also include free edges, allow them to cross arbitrarily



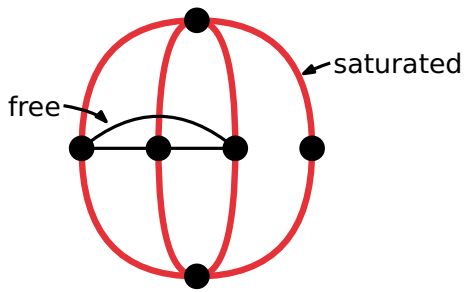


Filter Strategies

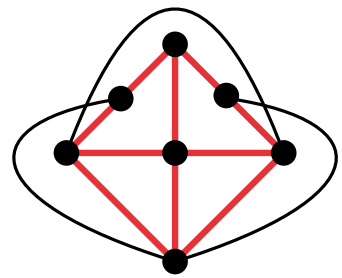
Planarity Filter [Binucci et al. '20]:
Reject if saturated subgraph of G_C not planar



stronger version: also include free edges, allow them to cross arbitrarily



reject

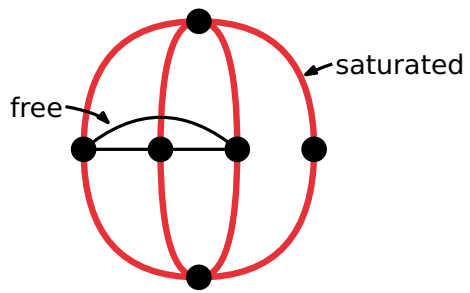


continue

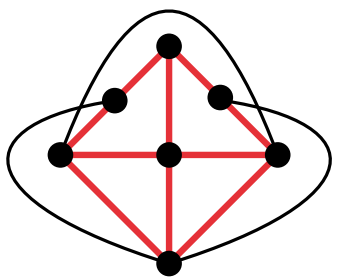
Filter Strategies

Partial Planarity Filter
Reject if saturated subgraph of G_C not partially planar

stronger version: also include free edges, allow them to cross arbitrarily



reject



continue

Partial Planarity

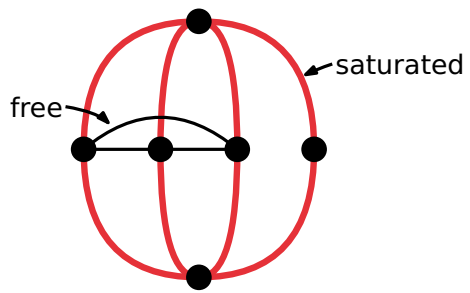
Input: Graph G, $F \subseteq E(G)$

Question: Does G admit a drawing where only edges of F have crossings?

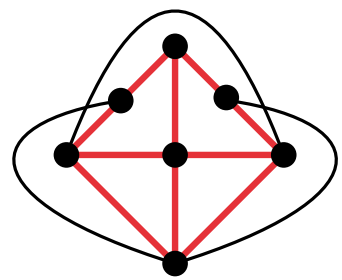
Filter Strategies

Partial Planarity Filter
Reject if saturated subgraph of G_C not partially planar

stronger version: also include free edges, allow them to cross arbitrarily



reject



continue

Partial Planarity

Input: Graph G, $F \subseteq E(G)$

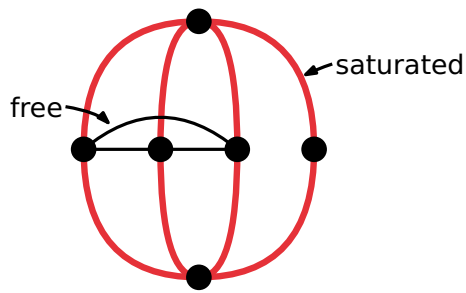
Question: Does G admit a drawing where only edges of F have crossings?

- complicated $O(n)$-algorithm [Da Lozzo, Rutter '19]

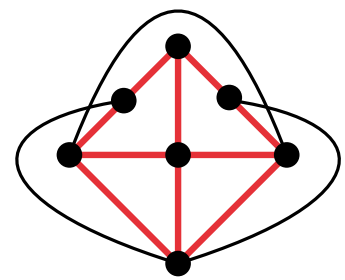
Filter Strategies

Partial Planarity Filter
Reject if saturated subgraph of G_C not partially planar

stronger version: also include free edges, allow them to cross arbitrarily



reject



continue

Partial Planarity

Input: Graph G, $F \subseteq E(G)$

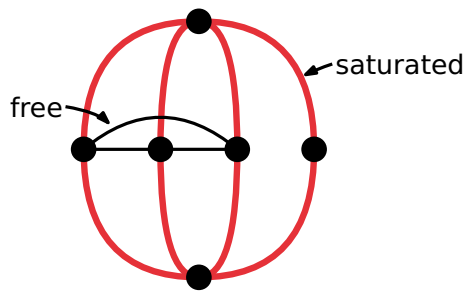
Question: Does G admit a drawing where only edges of F have crossings?

- complicated $O(n)$-algorithm [Da Lozzo, Rutter '19]
- simple $O(n^6)$-algorithm [Schaefer '14]

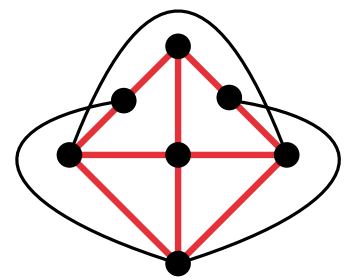
Filter Strategies

Partial Planarity Filter
Reject if saturated subgraph of G_C not partially planar

stronger version: also include free edges, allow them to cross arbitrarily



reject



continue

Partial Planarity

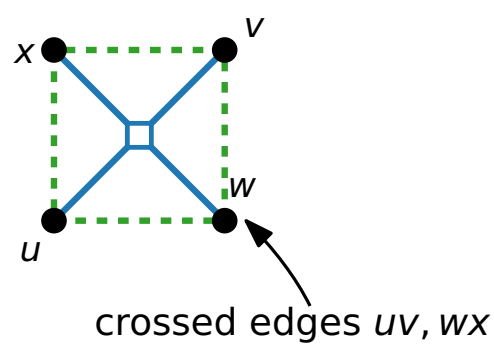
Input: Graph G, $F \subseteq E(G)$

Question: Does G admit a drawing where only edges of F have crossings?

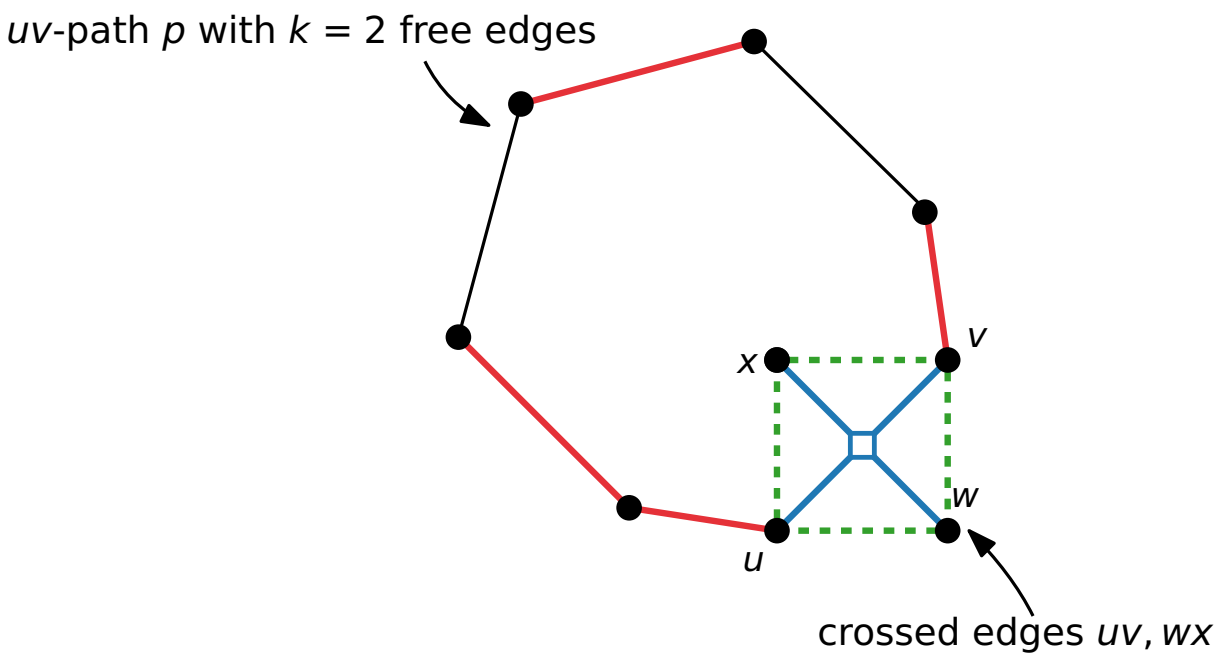
- complicated $O(n)$-algorithm [Da Lozzo, Rutter '19]
- simple $O(n^6)$-algorithm [Schaefer '14]

only minor improvement, even if we count processed nodes instead of running time

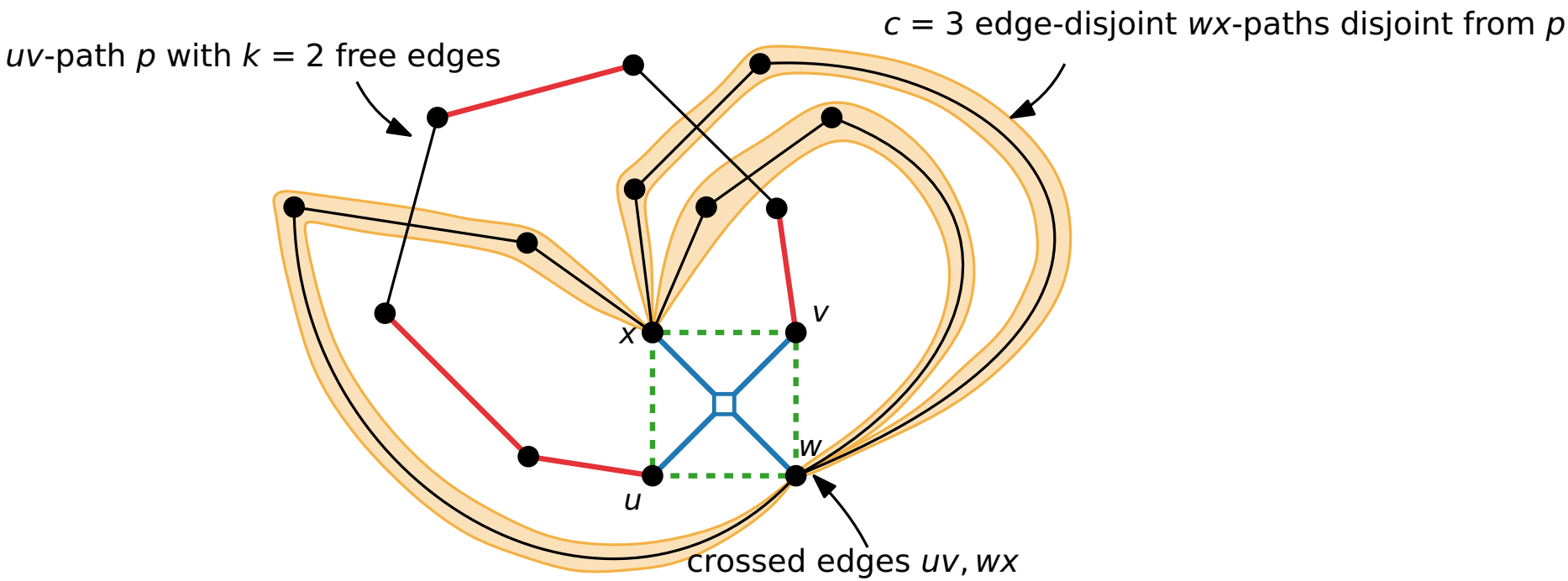
Filter Strategies



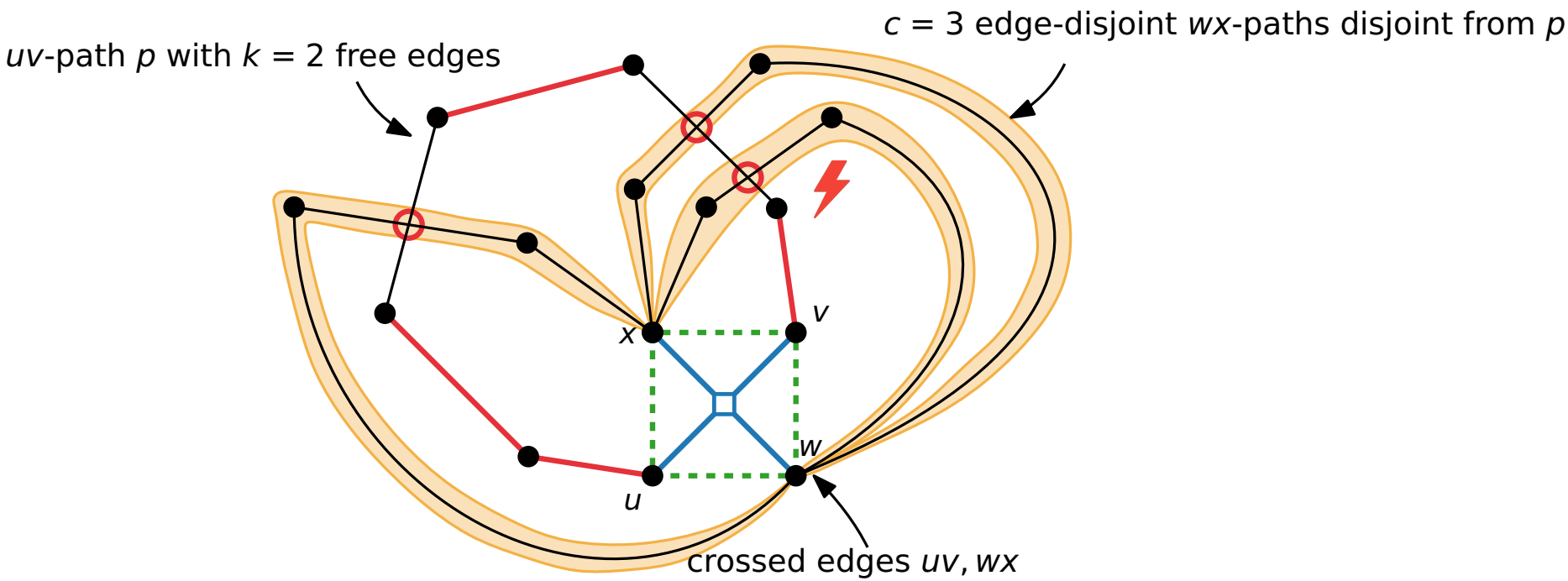
Filter Strategies



Filter Strategies

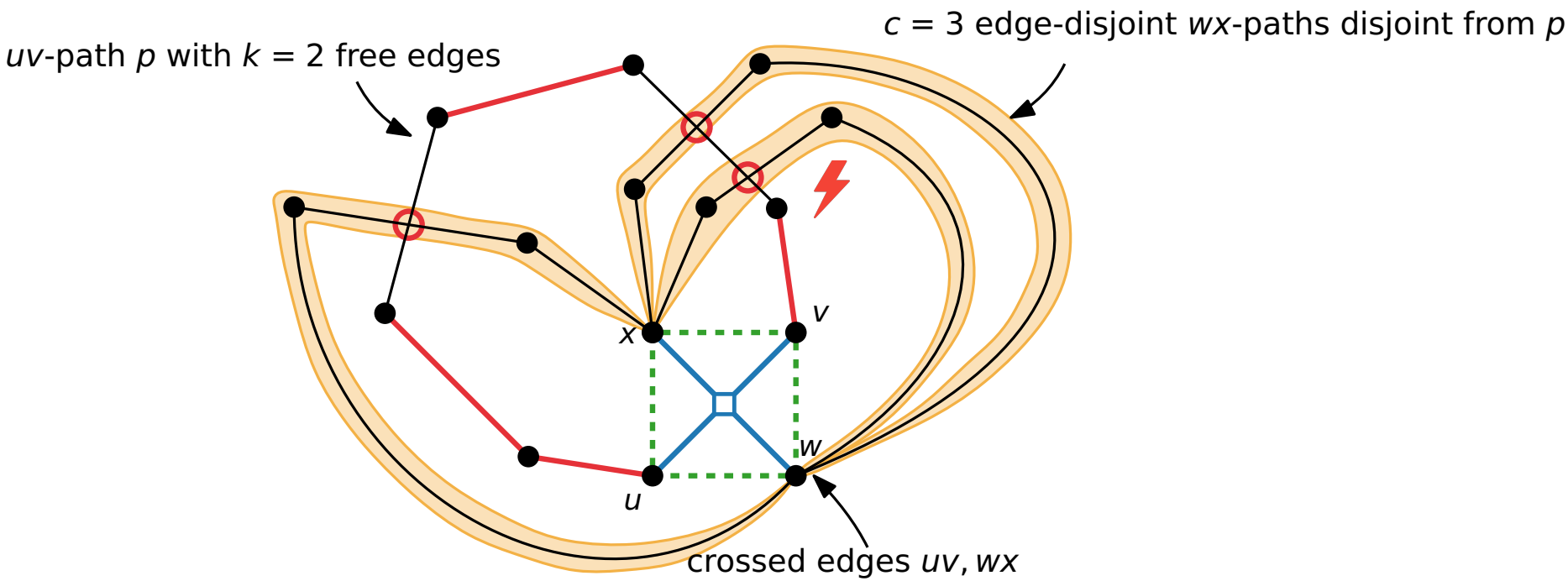


Filter Strategies



Filter Strategies

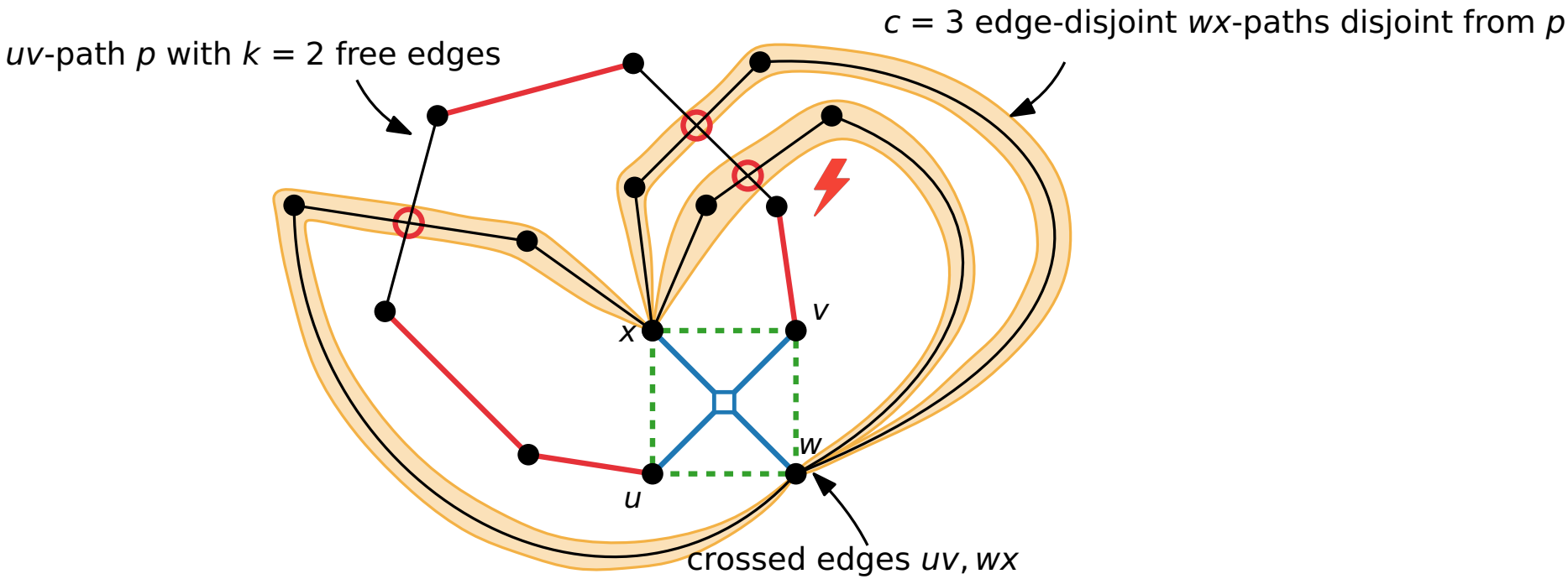
Separating Cycle Filter (SC)
Reject if $k < c$



Filter Strategies

Separating Cycle Filter (SC)
Reject if $k < c$

Rejects 90% of partial solutions! 🙂

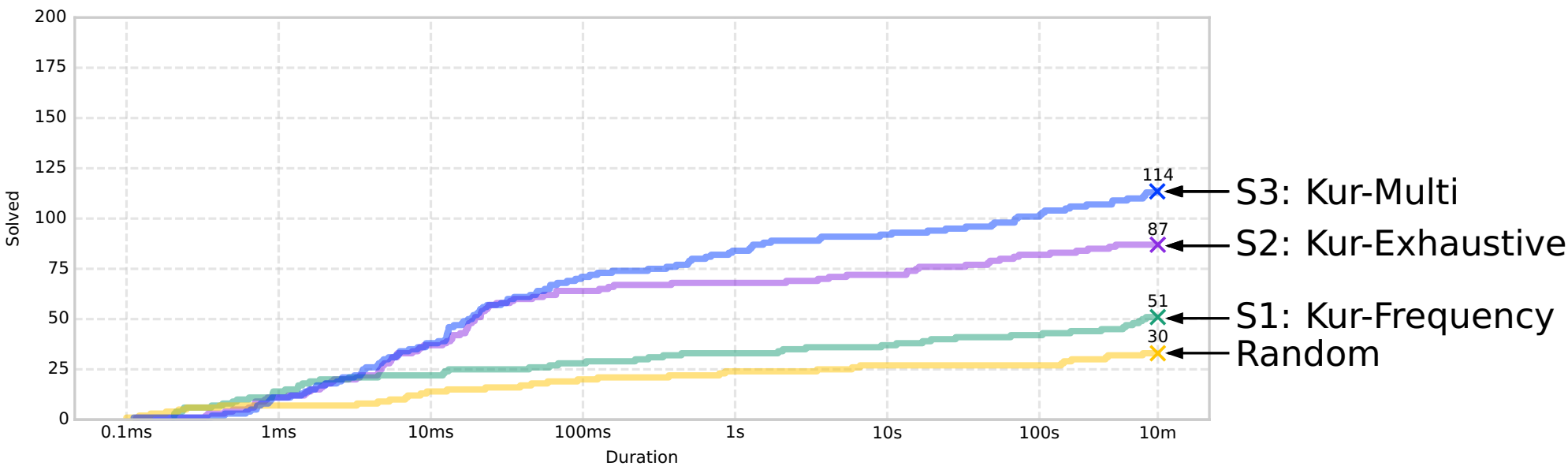


Filter Strategies

Separating Cycle Filter (SC)
Reject if $k < c$

Rejects 90% of partial solutions! 🙂

Solved instances over time:

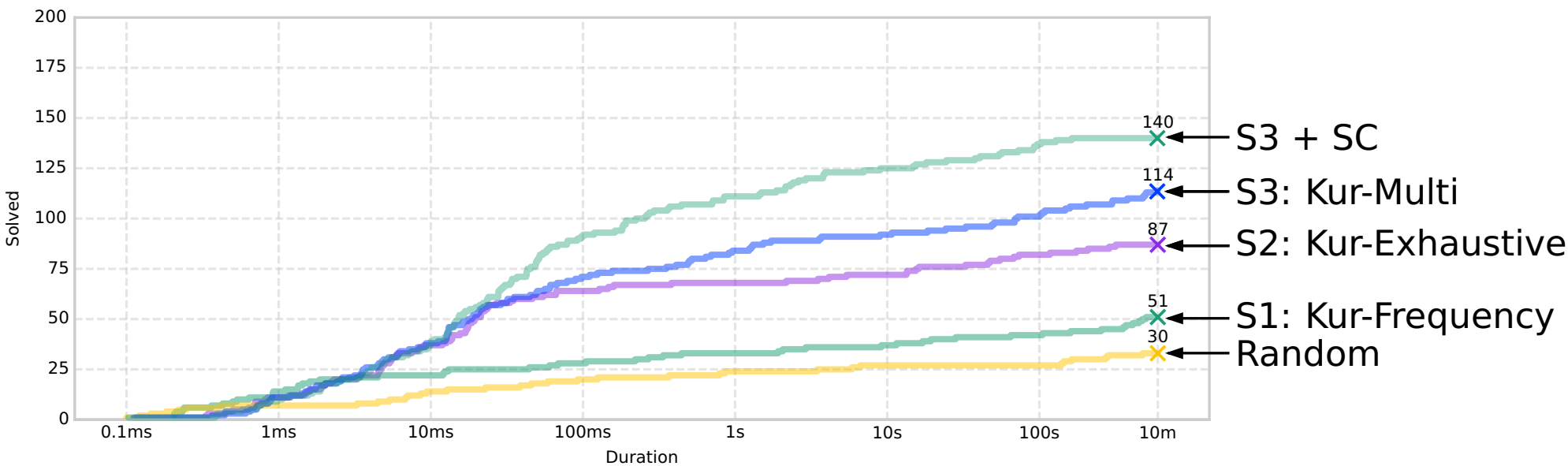


Filter Strategies

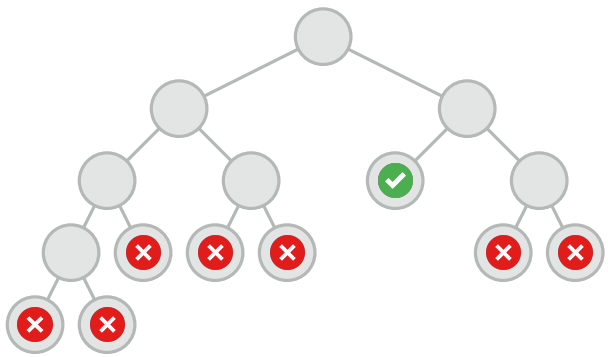
Separating Cycle Filter (SC)
Reject if $k < c$

Rejects 90% of partial solutions! 🙂

Solved instances over time:

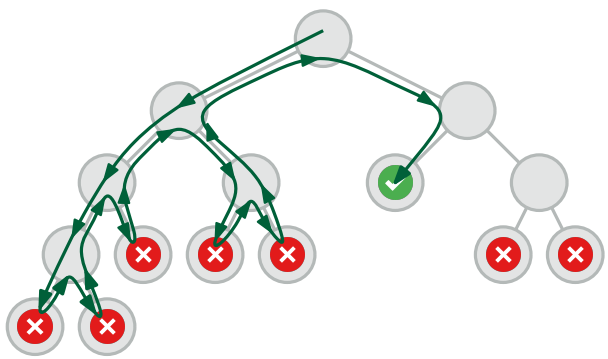


Traversal Strategies



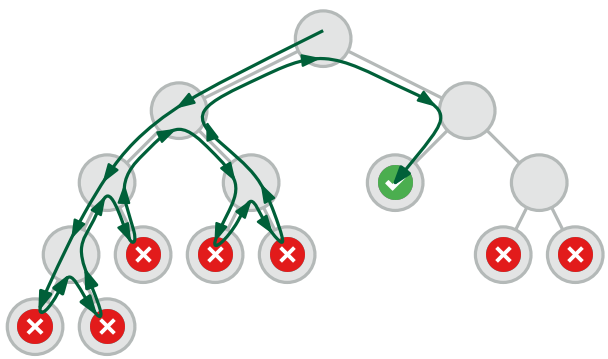
Traversal Strategies

DFS

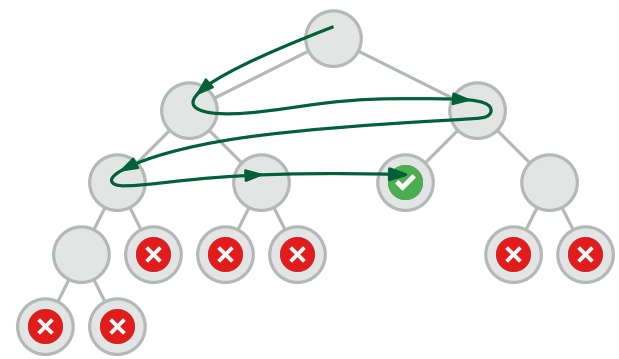


Traversal Strategies

DFS

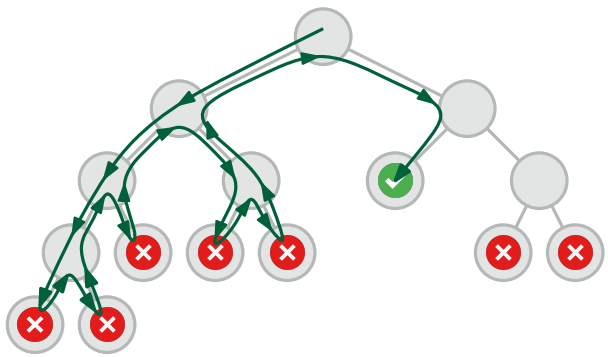


BFS



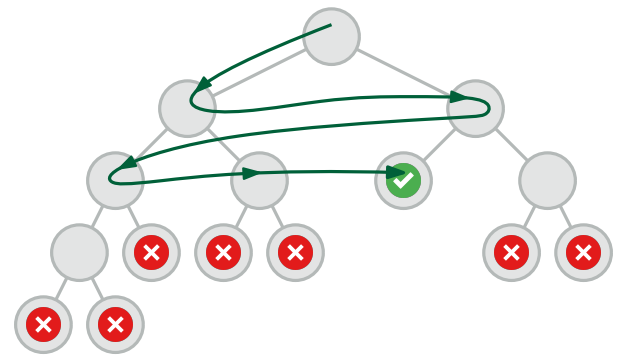
Traversal Strategies

DFS



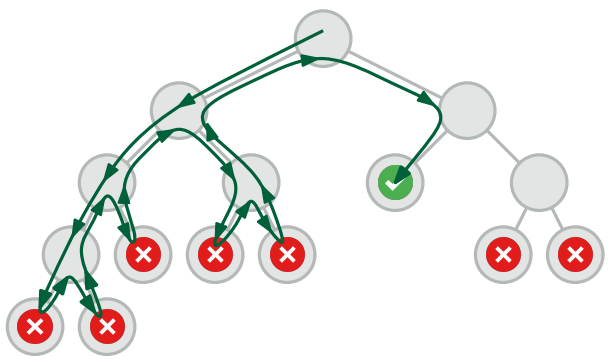
gets stuck in branches ☹️

BFS



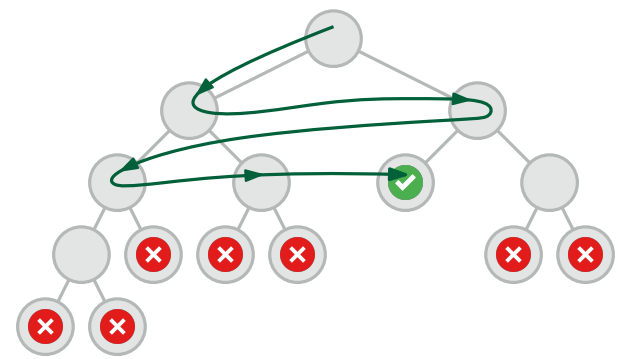
Traversal Strategies

DFS



gets stuck in branches 😕

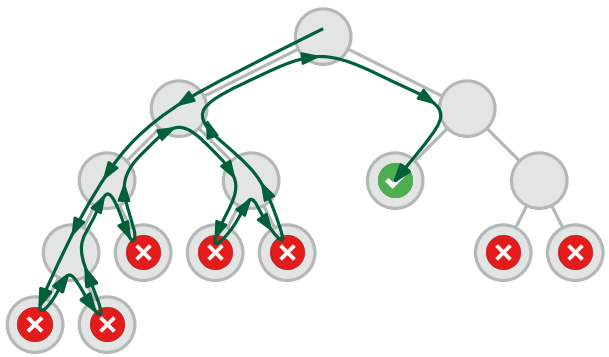
BFS



exponential memory 😕

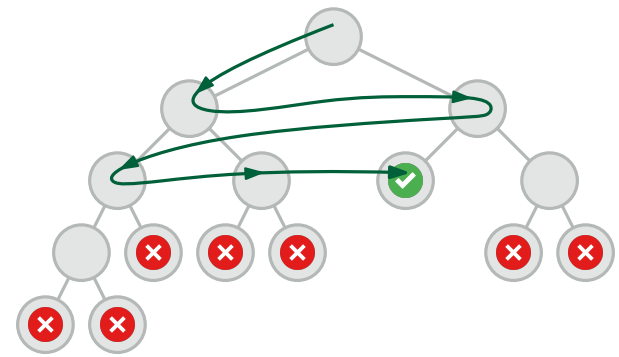
Traversal Strategies

DFS

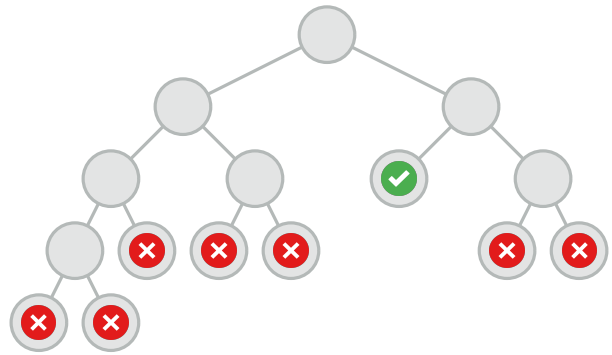


gets stuck in branches ☹

BFS

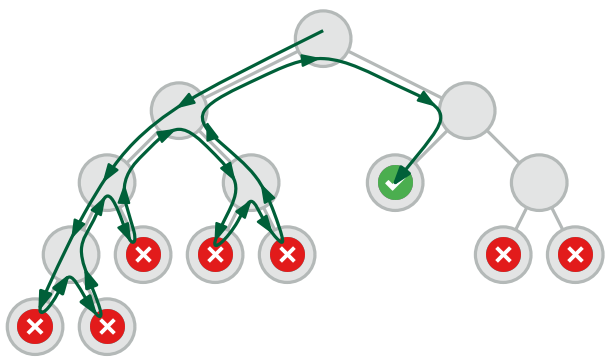


exponential memory ☹



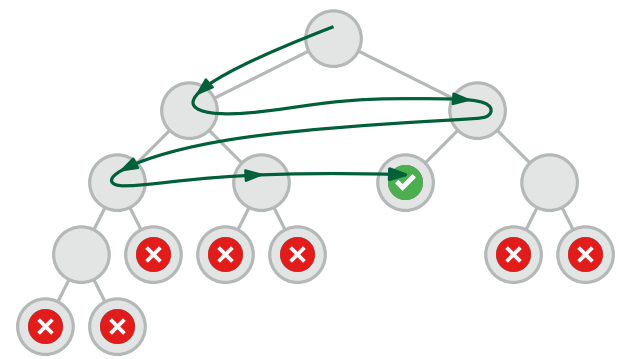
Traversal Strategies

DFS



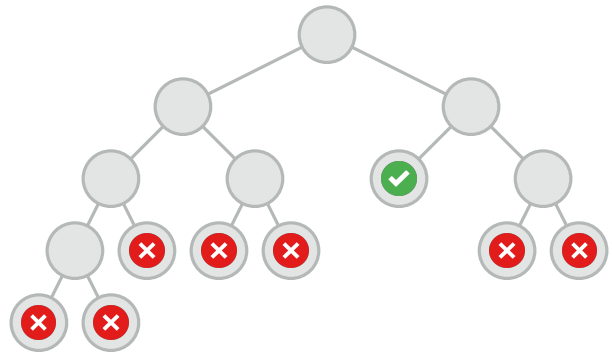
gets stuck in branches ☹

BFS



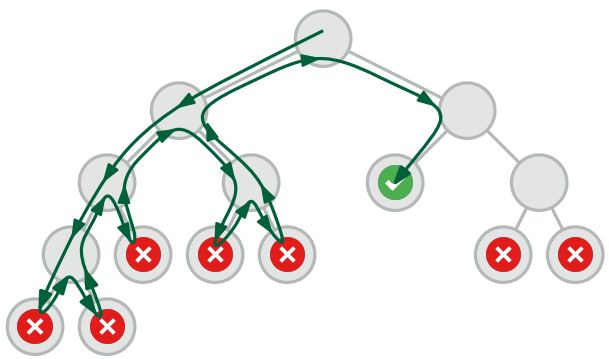
exponential memory ☹

maintain queue of k DFS-threads



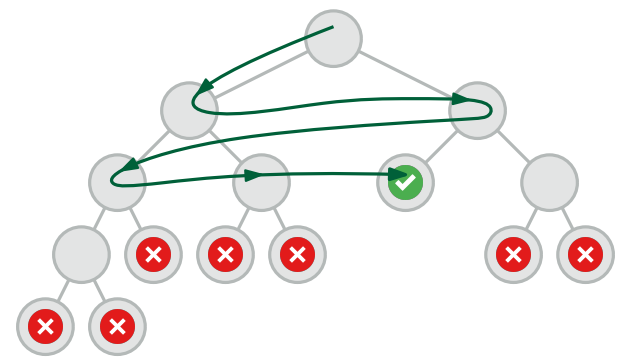
Traversal Strategies

DFS



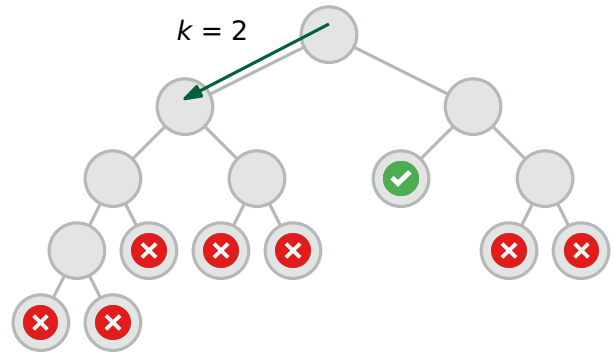
gets stuck in branches ☹

BFS



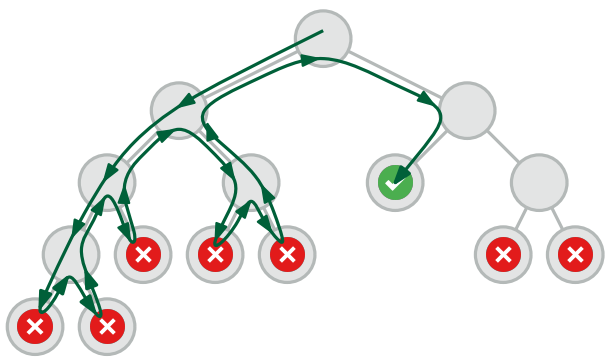
exponential memory ☹

maintain queue of k DFS-threads



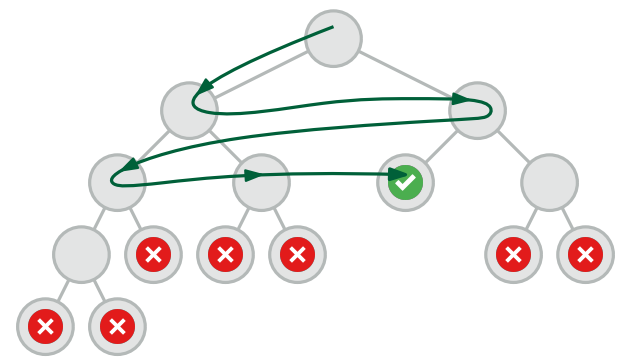
Traversal Strategies

DFS



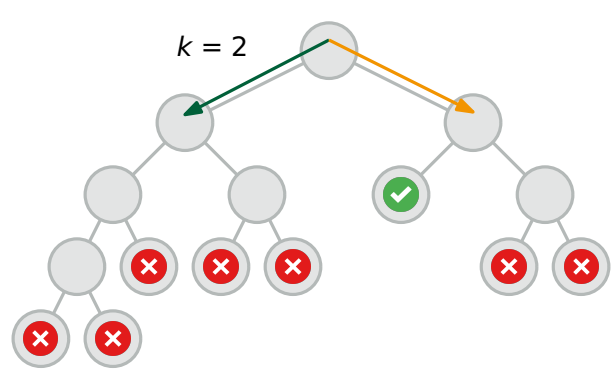
gets stuck in branches ☹

BFS



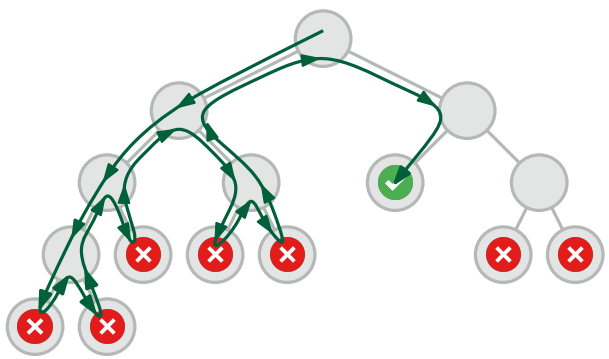
exponential memory ☹

maintain queue of k DFS-threads



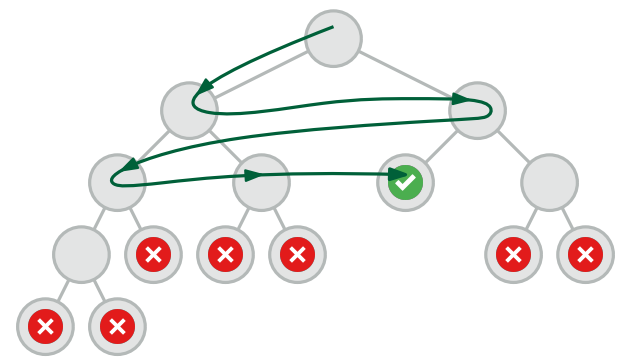
Traversal Strategies

DFS



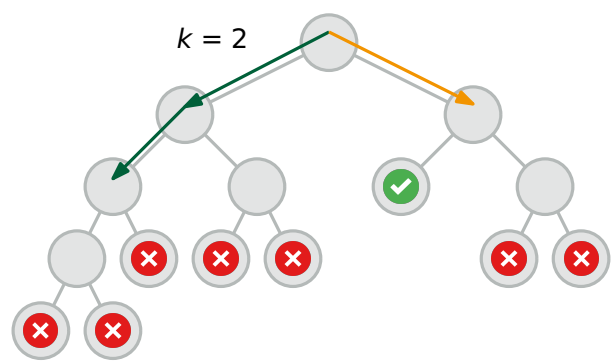
gets stuck in branches ☹

BFS



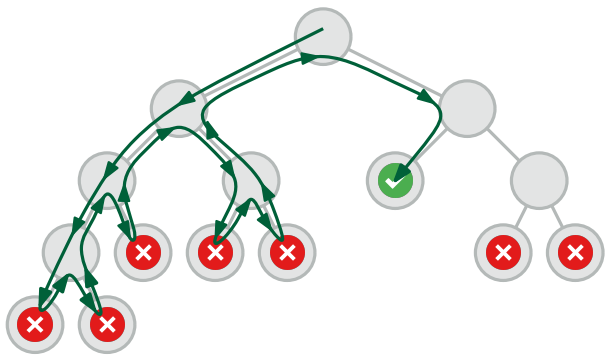
exponential memory ☹

maintain queue of k DFS-threads



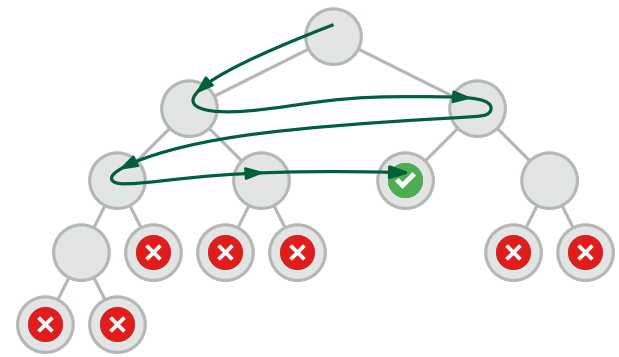
Traversal Strategies

DFS



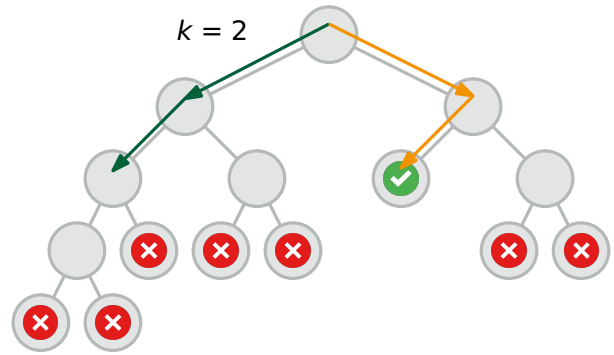
gets stuck in branches ☹

BFS



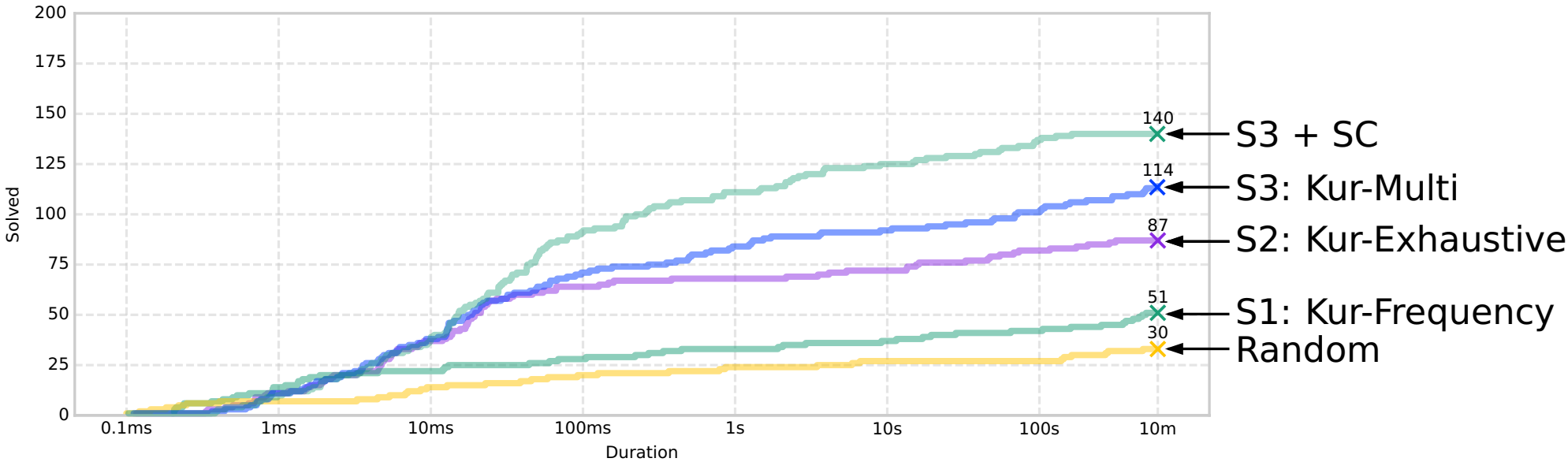
exponential memory ☹

maintain queue of k DFS-threads

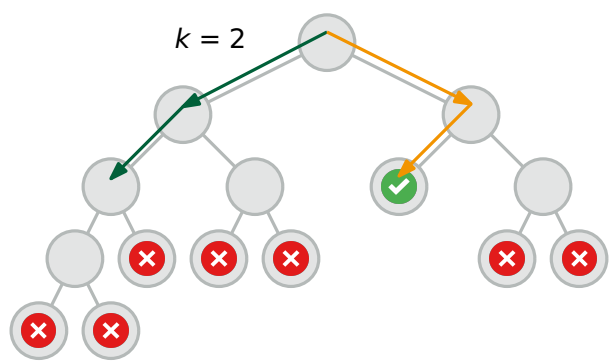


Traversal Strategies

Solved instances over time:

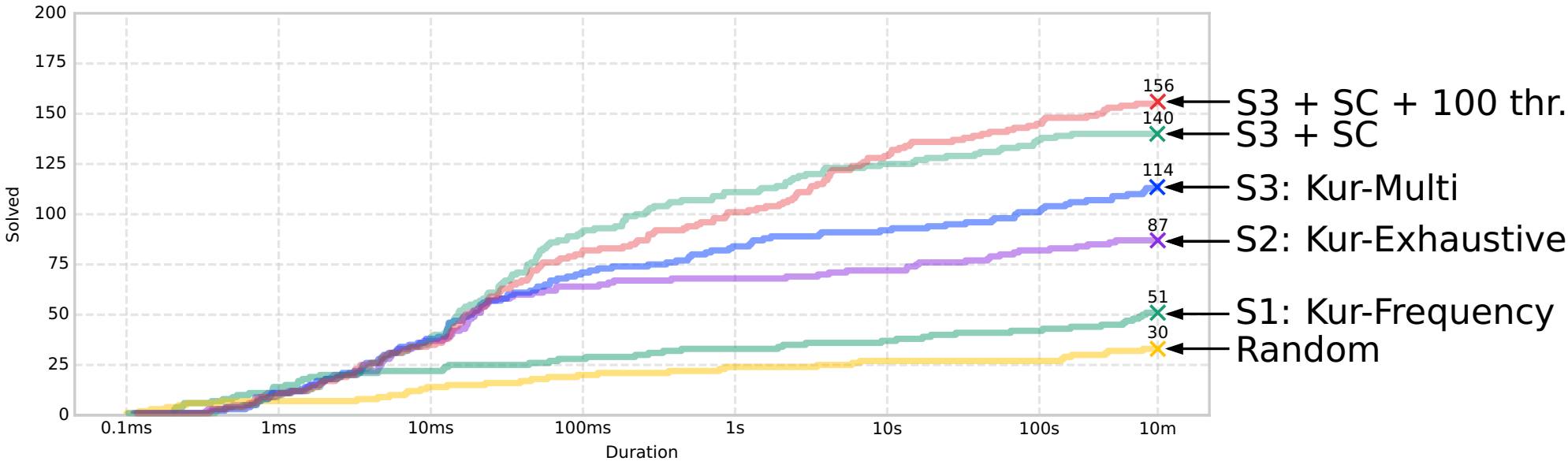


maintain queue of k DFS-threads

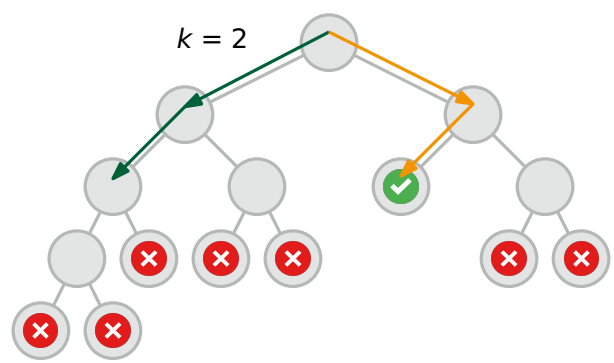


Traversal Strategies

Solved instances over time:



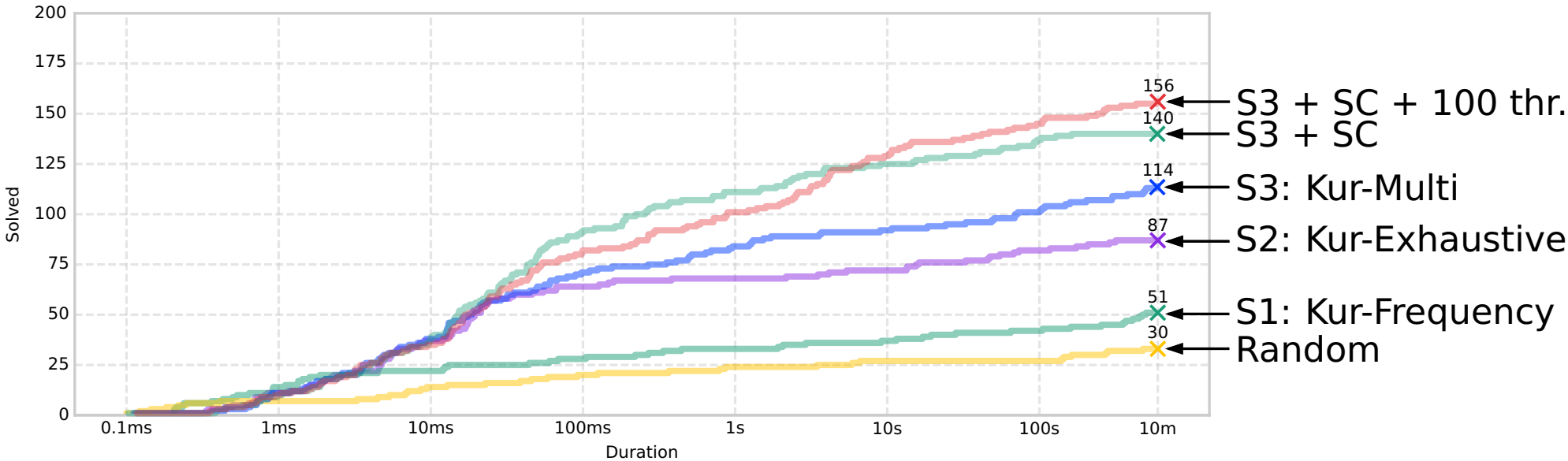
maintain queue of k DFS-threads



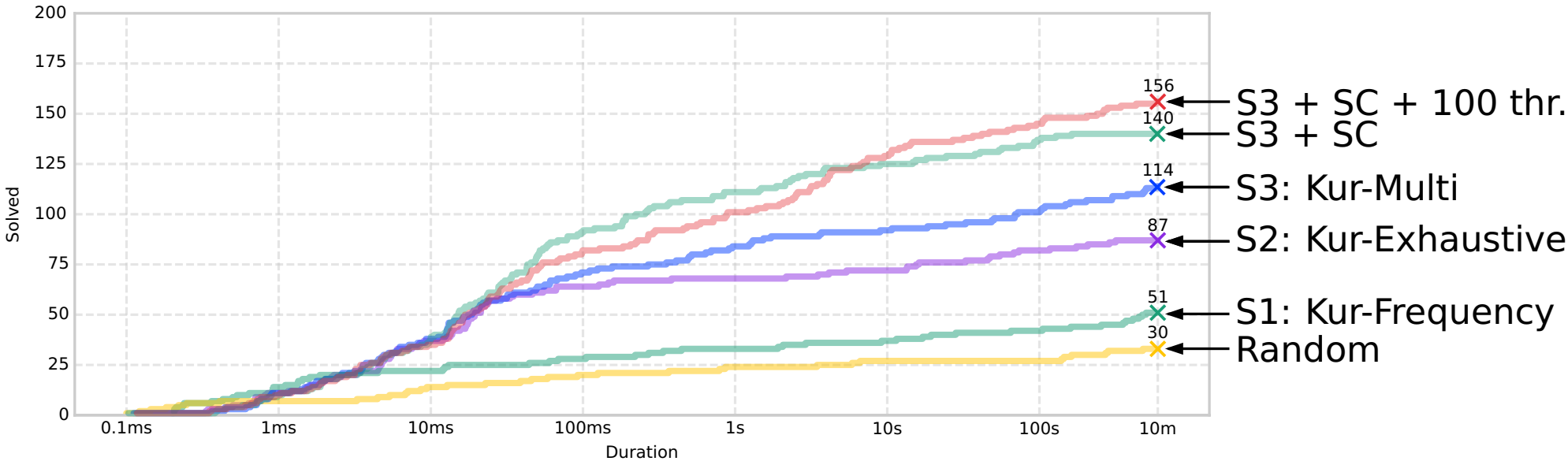
Rome/North biconnected components, 15 min time limit

Group	#	Group Median			Solved		1-Planar %	
		Density	Proc. Nodes	Time (s)	#	%	Yes	No
Bicomps-R 1-10	9	1.57	2	0.0	9	100.0	100.0	0.0
Bicomps-R 11-20	250	1.47	3	0.0	250	100.0	100.0	0.0
Bicomps-R 21-30	1296	1.39	93	0.02	1295	99.9	99.8	0.1
Bicomps-R 31-40	1843	1.43	9404	4.57	1753	95.1	95.1	0.1
Bicomps-R 41-50	1358	1.44	23904	21.35	908	66.9	66.9	0.0
Bicomps-R 51-60	1227	1.44	43785	60.26	401	32.7	32.7	0.0
Bicomps-R 61-70	1126	1.45	46543	80.68	79	7.0	7.0	0.0
Bicomps-R 71-80	929	1.47	48832	116.08	16	1.7	1.7	0.0
Bicomps-R 81-90	214	1.45	44744	145.36	3	1.4	1.4	0.0
Bicomps-R 91-100	1	1.42	–	–	0	0.0	0.0	0.0
Bicomps-R	8253	1.44	7882	3.58	4714	57.1	57.1	0.0
Bicomps-N 1-10	56	2.05	5	0.0	56	100.0	92.9	7.1
Bicomps-N 11-20	124	2.0	52	0.01	124	100.0	60.5	39.5
Bicomps-N 21-30	94	1.9	607	0.26	93	98.9	48.9	50.0
Bicomps-N 31-40	52	1.81	192	0.25	41	78.8	50.0	28.8
Bicomps-N 41-50	30	1.48	1257	3.61	28	93.3	73.3	20.0
Bicomps-N 51-60	44	1.92	440	2.13	19	43.2	11.4	31.8
Bicomps-N 61-70	21	1.92	5216	9.5	2	9.5	4.8	4.8
Bicomps-N 71-80	2	1.58	5690	19.4	2	100.0	100.0	0.0
Bicomps-N 81-90	4	1.67	4088	12.29	2	50.0	50.0	0.0
Bicomps-N 91-100	2	1.52	47117	374.87	2	100.0	100.0	0.0
Bicomps-N	429	1.9	97	0.03	369	86.0	54.3	31.7

Conclusion


200
175
150
125
100
75
50
25
0
Solved
0.1ms
1ms
10ms
100ms
1s
10s
100s
10m
Duration
156
140
114
87
51
30
S3 + SC + 100 thr.
S3 + SC
S3: Kur-Multi
S2: Kur-Exhaustive
S1: Kur-Frequency
Random

Conclusion



Other structural filters that reject early?