Minor Containment and Disjoint Paths in almost-linear time

Tuukka Korhonen

UNIVERSITY OF COPENHAGEN

based on joint work with Michał Pilipczuk and Giannos Stamoulis
from the University of Warsaw (accepted to FOCS 2024)

19 September 2024

Minors of graphs

Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by

Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions

Minors of graphs

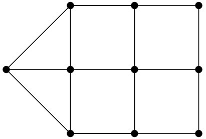
- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions

Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions

Minors of graphs

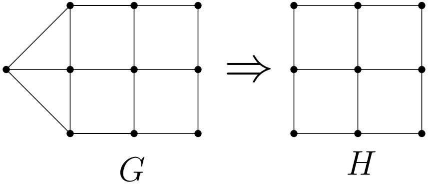
- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



G $\Rightarrow$

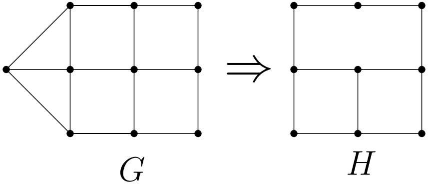
Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



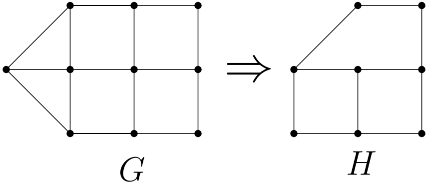
Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



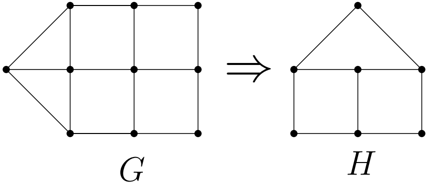
Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



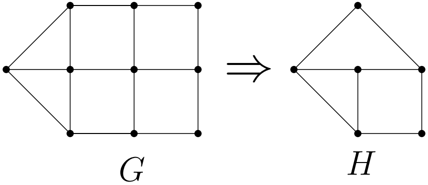
Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



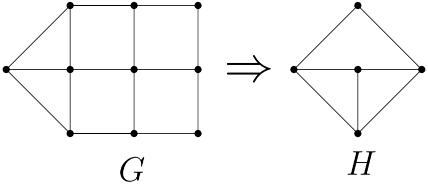
Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



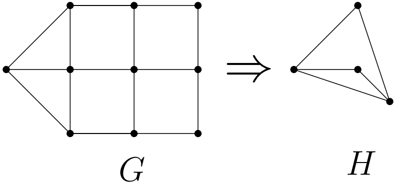
Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



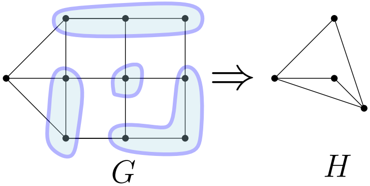
Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



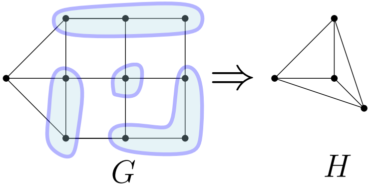
Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



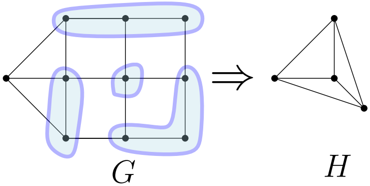
Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions

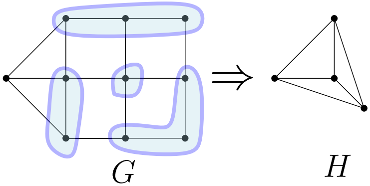


Theorem (Kuratowski-Wagner, 1930, 1937)

A graph is planar if and only if it does not contain K_5 or $K_{3,3}$ as a minor.

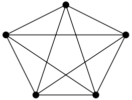
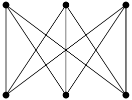
Minors of graphs

- A graph H is a minor of a graph G if H can be obtained from G by
  - Vertex deletions
  - Edge deletions
  - Edge contractions



Theorem (Kuratowski-Wagner, 1930, 1937)

A graph is planar if and only if it does not contain K_5 or $K_{3,3}$ as a minor.

Planar $\Leftrightarrow$ no  or 

The Graph Minor Theorem

The Graph Minor Theorem

Theorem (Robertson & Seymour, 1984-2004)

Let $\mathcal{C}$ be a minor-closed graph class. There exists a finite set of graphs $\mathcal{H}$, s.t. a graph G is in $\mathcal{C}$ if and only if G does not contain a graph from $\mathcal{H}$ as a minor.

The Graph Minor Theorem

Theorem (Robertson & Seymour, 1984-2004)

Let $\mathcal{C}$ be a minor-closed graph class. There exists a finite set of graphs $\mathcal{H}$, s.t. a graph G is in $\mathcal{C}$ if and only if G does not contain a graph from $\mathcal{H}$ as a minor.

- Minor-closed: Every minor of a graph in $\mathcal{C}$ is in $\mathcal{C}$

The Graph Minor Theorem

Theorem (Robertson & Seymour, 1984-2004)

Let $\mathcal{C}$ be a minor-closed graph class. There exists a finite set of graphs $\mathcal{H}$, s.t. a graph G is in $\mathcal{C}$ if and only if G does not contain a graph from $\mathcal{H}$ as a minor.

- Minor-closed: Every minor of a graph in $\mathcal{C}$ is in $\mathcal{C}$
- Examples:

The Graph Minor Theorem

Theorem (Robertson & Seymour, 1984-2004)

Let $\mathcal{C}$ be a minor-closed graph class. There exists a finite set of graphs $\mathcal{H}$, s.t. a graph G is in $\mathcal{C}$ if and only if G does not contain a graph from $\mathcal{H}$ as a minor.

- Minor-closed: Every minor of a graph in $\mathcal{C}$ is in $\mathcal{C}$
- Examples:
  - $\mathcal{C} = \{$the planar graphs$\}$, $\mathcal{H} = \{K_5, K_{3,3}\}$

The Graph Minor Theorem

Theorem (Robertson & Seymour, 1984-2004)

Let $\mathcal{C}$ be a minor-closed graph class. There exists a finite set of graphs $\mathcal{H}$, s.t. a graph G is in $\mathcal{C}$ if and only if G does not contain a graph from $\mathcal{H}$ as a minor.

- Minor-closed: Every minor of a graph in $\mathcal{C}$ is in $\mathcal{C}$
- Examples:
  - $\mathcal{C} = \{\text{the planar graphs}\}$, $\mathcal{H} = \{K_5, K_{3,3}\}$
  - $\mathcal{C} = \{\text{graphs admitting a linkless embedding in 3D}\}$, $\mathcal{H} = \{\text{Petersen family}\}$

The Graph Minor Theorem

Theorem (Robertson & Seymour, 1984-2004)

Let $\mathcal{C}$ be a minor-closed graph class. There exists a finite set of graphs $\mathcal{H}$, s.t. a graph G is in $\mathcal{C}$ if and only if G does not contain a graph from $\mathcal{H}$ as a minor.

- Minor-closed: Every minor of a graph in $\mathcal{C}$ is in $\mathcal{C}$
- Examples:
  - $\mathcal{C} = \{\text{the planar graphs}\}$, $\mathcal{H} = \{K_5, K_{3,3}\}$
  - $\mathcal{C} = \{\text{graphs admitting a linkless embedding in 3D}\}$, $\mathcal{H} = \{\text{Petersen family}\}$
  - $\mathcal{C} = \{\text{graphs that can be made forests by deleting at most 10 vertices}\}$

The Graph Minor Theorem

Theorem (Robertson & Seymour, 1984-2004)

Let $\mathcal{C}$ be a minor-closed graph class. There exists a finite set of graphs $\mathcal{H}$, s.t. a graph G is in $\mathcal{C}$ if and only if G does not contain a graph from $\mathcal{H}$ as a minor.

- Minor-closed: Every minor of a graph in $\mathcal{C}$ is in $\mathcal{C}$
- Examples:
  - $\mathcal{C} = \{$the planar graphs$\}$, $\mathcal{H} = \{K_5, K_{3,3}\}$
  - $\mathcal{C} = \{$graphs admitting a linkless embedding in 3D$\}$, $\mathcal{H} = \{$Petersen family$\}$
  - $\mathcal{C} = \{$graphs that can be made forests by deleting at most 10 vertices$\}$
  - $\mathcal{C} = \{$graphs that can be embedded on a torus after deleting at most 5 edges$\}$

The Graph Minor Theorem

Theorem (Robertson & Seymour, 1984-2004)

Let $\mathcal{C}$ be a minor-closed graph class. There exists a finite set of graphs $\mathcal{H}$, s.t. a graph G is in $\mathcal{C}$ if and only if G does not contain a graph from $\mathcal{H}$ as a minor.


- Minor-closed: Every minor of a graph in $\mathcal{C}$ is in $\mathcal{C}$
- Examples:
  - $\mathcal{C} = \{\text{the planar graphs}\}$, $\mathcal{H} = \{K_5, K_{3,3}\}$
  - $\mathcal{C} = \{\text{graphs admitting a linkless embedding in 3D}\}$, $\mathcal{H} = \{\text{Petersen family}\}$
  - $\mathcal{C} = \{\text{graphs that can be made forests by deleting at most 10 vertices}\}$
  - $\mathcal{C} = \{\text{graphs that can be embedded on a torus after deleting at most 5 edges}\}$
  - $\mathcal{C} = \{\text{graphs of treewidth at most 20}\}$

The Graph Minor Theorem

Theorem (Robertson & Seymour, 1984-2004)

Let $\mathcal{C}$ be a minor-closed graph class. There exists a finite set of graphs $\mathcal{H}$, s.t. a graph G is in $\mathcal{C}$ if and only if G does not contain a graph from $\mathcal{H}$ as a minor.

- Minor-closed: Every minor of a graph in $\mathcal{C}$ is in $\mathcal{C}$
- Examples:
  - $\mathcal{C} = \{\text{the planar graphs}\}$, $\mathcal{H} = \{K_5, K_{3,3}\}$
  - $\mathcal{C} = \{\text{graphs admitting a linkless embedding in 3D}\}$, $\mathcal{H} = \{\text{Petersen family}\}$
  - $\mathcal{C} = \{\text{graphs that can be made forests by deleting at most 10 vertices}\}$
  - $\mathcal{C} = \{\text{graphs that can be embedded on a torus after deleting at most 5 edges}\}$
  - $\mathcal{C} = \{\text{graphs of treewidth at most 20}\}$
- Proved in the Graph Minors Series of Robertson & Seymour, spanning 23 papers in 1983–2012.

Algorithms?

Algorithms?

Theorem (Robertson & Seymour, 1984-2012)

There exists an $f(H) \cdot n^3$ time algorithm to test if a given graph H is a minor of a given n-vertex graph G.

Algorithms?

Theorem (Robertson & Seymour, 1984-2012)

There exists an $f(H) \cdot n^3$ time algorithm to test if a given graph H is a minor of a given n-vertex graph G.


Combined with the Graph Minor Theorem, we get:

Corollary

For every minor-closed graph class $\mathcal{C}$, there exists an $\mathcal{O}(n^3)$ time algorithm to test if a given n-vertex graph is in $\mathcal{C}$.

Algorithms?

Theorem (Robertson & Seymour, 1984-2012)

There exists an $f(H) \cdot n^3$ time algorithm to test if a given graph H is a minor of a given n-vertex graph G.

Combined with the Graph Minor Theorem, we get:

Corollary

For every minor-closed graph class $\mathcal{C}$, there exists an $\mathcal{O}(n^3)$ time algorithm to test if a given n-vertex graph is in $\mathcal{C}$.

$\Rightarrow$ $\mathcal{O}(n^3)$ time algorithms for many graph problems, some of which were not even known to be decidable before the Graph Minors Series

Algorithms?

Theorem (Robertson & Seymour, 1984-2012)

There exists an $f(H) \cdot n^3$ time algorithm to test if a given graph H is a minor of a given n-vertex graph G.

Combined with the Graph Minor Theorem, we get:

Corollary

For every minor-closed graph class $\mathcal{C}$, there exists an $\mathcal{O}(n^3)$ time algorithm to test if a given n-vertex graph is in $\mathcal{C}$.

$\Rightarrow$ $\mathcal{O}(n^3)$ time algorithms for many graph problems, some of which were not even known to be decidable before the Graph Minors Series

- (non-constructive) $f(k) \cdot n^3$ time algorithms for parameterized problems

Algorithms?

Theorem (Robertson & Seymour, 1984-2012)

There exists an $f(H) \cdot n^3$ time algorithm to test if a given graph H is a minor of a given n-vertex graph G.

Combined with the Graph Minor Theorem, we get:

Corollary

For every minor-closed graph class $\mathcal{C}$, there exists an $\mathcal{O}(n^3)$ time algorithm to test if a given n-vertex graph is in $\mathcal{C}$.

$\Rightarrow$ $\mathcal{O}(n^3)$ time algorithms for many graph problems, some of which were not even known to be decidable before the Graph Minors Series

- (non-constructive) $f(k) \cdot n^3$ time algorithms for parameterized problems
  - Inspired the birth of Parameterized complexity in the late 80s

Algorithms?

Theorem (Robertson & Seymour, 1984-2012)

There exists an $f(H) \cdot n^3$ time algorithm to test if a given graph H is a minor of a given n-vertex graph G.

Combined with the Graph Minor Theorem, we get:

Corollary

For every minor-closed graph class $\mathcal{C}$, there exists an $\mathcal{O}(n^3)$ time algorithm to test if a given n-vertex graph is in $\mathcal{C}$.

$\Rightarrow$ $\mathcal{O}(n^3)$ time algorithms for many graph problems, some of which were not even known to be decidable before the Graph Minors Series

- (non-constructive) $f(k) \cdot n^3$ time algorithms for parameterized problems
  - Inspired the birth of Parameterized complexity in the late 80s
- More generally, an $f(H) \cdot n^3$ time algorithm for Rooted Minor Containment

Algorithms?


Theorem (Robertson & Seymour, 1984-2012)

There exists an $f(H) \cdot n^3$ time algorithm to test if a given graph H is a minor of a given n-vertex graph G.

Combined with the Graph Minor Theorem, we get:

Corollary

For every minor-closed graph class $\mathcal{C}$, there exists an $\mathcal{O}(n^3)$ time algorithm to test if a given n-vertex graph is in $\mathcal{C}$.

$\Rightarrow$ $\mathcal{O}(n^3)$ time algorithms for many graph problems, some of which were not even known to be decidable before the Graph Minors Series

- (non-constructive) $f(k) \cdot n^3$ time algorithms for parameterized problems
  - Inspired the birth of Parameterized complexity in the late 80s
- More generally, an $f(H) \cdot n^3$ time algorithm for Rooted Minor Containment

  $\Rightarrow$ $f(k) \cdot n^3$ time algorithm for the k-Disjoint Paths problem

Algorithms

Algorithms

- The algorithm of Robertson & Seymour was improved to $f(H) \cdot n^2$ by [Kawarabayashi, Kobayashi & Reed, 2012]

Algorithms

- The algorithm of Robertson & Seymour was improved to $f(H) \cdot n^2$ by [Kawarabayashi, Kobayashi & Reed, 2012]
- Linear-time algorithms for planar graphs by [Bodlaender, 1993] and [Reed, Robertson, Schrijver & Seymour, 1993]

Algorithms

- The algorithm of Robertson & Seymour was improved to $f(H) \cdot n^2$ by [Kawarabayashi, Kobayashi & Reed, 2012]
- Linear-time algorithms for planar graphs by [Bodlaender, 1993] and [Reed, Robertson, Schrijver & Seymour, 1993]

Theorem (K., Pilipczuk, Stamoulis, FOCS 2024)

There is an $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment

Algorithms

- The algorithm of Robertson & Seymour was improved to $f(H) \cdot n^2$ by [Kawarabayashi, Kobayashi & Reed, 2012]
- Linear-time algorithms for planar graphs by [Bodlaender, 1993] and [Reed, Robertson, Schrijver & Seymour, 1993]

Theorem (K., Pilipczuk, Stamoulis, FOCS 2024)

There is an $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment

- $m = |V(G)| + |E(G)|$ the number of vertices + edges of G

Algorithms

- The algorithm of Robertson & Seymour was improved to $f(H) \cdot n^2$ by [Kawarabayashi, Kobayashi & Reed, 2012]
- Linear-time algorithms for planar graphs by [Bodlaender, 1993] and [Reed, Robertson, Schrijver & Seymour, 1993]

Theorem (K., Pilipczuk, Stamoulis, FOCS 2024)

There is an $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment

- $m = |V(G)| + |E(G)|$ the number of vertices + edges of G
- Dependence on H huge but computable.

Algorithms

- The algorithm of Robertson & Seymour was improved to $f(H) \cdot n^2$ by [Kawarabayashi, Kobayashi & Reed, 2012]
- Linear-time algorithms for planar graphs by [Bodlaender, 1993] and [Reed, Robertson, Schrijver & Seymour, 1993]

Theorem (K., Pilipczuk, Stamoulis, FOCS 2024)

There is an $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment

- $m = |V(G)| + |E(G)|$ the number of vertices + edges of G
- Dependence on H huge but computable.

Corollary

Every minor-closed graph class has an $n^{1+o(1)}$ time recognition algorithm

Algorithms

- The algorithm of Robertson & Seymour was improved to $f(H) \cdot n^2$ by [Kawarabayashi, Kobayashi & Reed, 2012]
- Linear-time algorithms for planar graphs by [Bodlaender, 1993] and [Reed, Robertson, Schrijver & Seymour, 1993]

Theorem (K., Pilipczuk, Stamoulis, FOCS 2024)

There is an $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment

- $m = |V(G)| + |E(G)|$ the number of vertices + edges of G
- Dependence on H huge but computable.

Corollary

Every minor-closed graph class has an $n^{1+o(1)}$ time recognition algorithm

Corollary

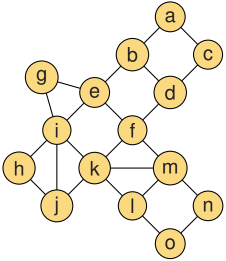
There is an $f(k) \cdot m^{1+o(1)}$ time algorithm for the k-Disjoint Paths problem

The algorithm

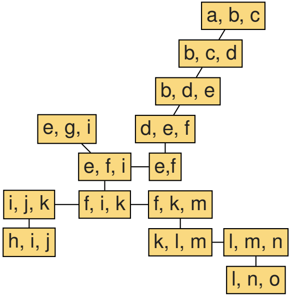
The Irrelevant Vertex technique

The Irrelevant Vertex technique

- Treewidth of a graph: Parameter between 0 and $n-1$ measuring how tree-like the graph is



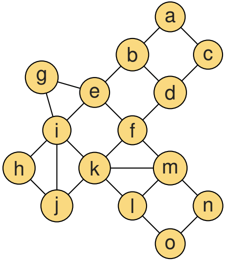
Graph G
Treewidth 2



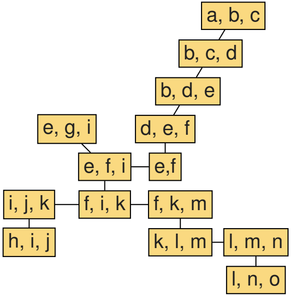
A tree decomposition of G
Width = 2

The Irrelevant Vertex technique

- Treewidth of a graph: Parameter between 0 and $n-1$ measuring how tree-like the graph is
- If treewidth of G is $\leq f(H)$, solve the problem by dynamic programming in $f(H) \cdot n$ time



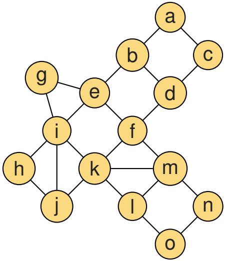
Graph G
Treewidth 2



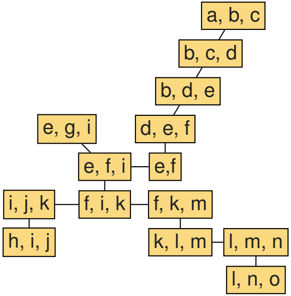
A tree decomposition of G
Width = 2

The Irrelevant Vertex technique

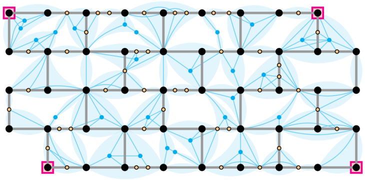
- Treewidth of a graph: Parameter between 0 and $n-1$ measuring how tree-like the graph is
- If treewidth of G is $\leq f(H)$, solve the problem by dynamic programming in $f(H) \cdot n$ time
- If treewidth is $> f(H)$, detect and remove an Irrelevant Vertex from G



Graph G
Treewidth 2

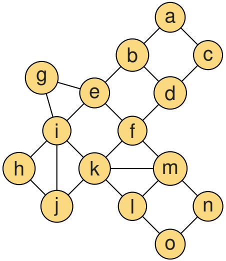


A tree decomposition of G
Width = 2

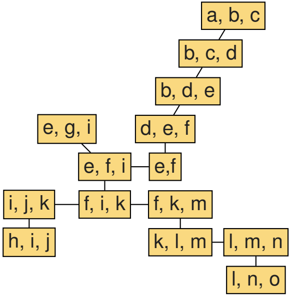


The Irrelevant Vertex technique

- Treewidth of a graph: Parameter between 0 and $n-1$ measuring how tree-like the graph is
- If treewidth of G is $\leq f(H)$, solve the problem by dynamic programming in $f(H) \cdot n$ time
- If treewidth is $> f(H)$, detect and remove an Irrelevant Vertex from G

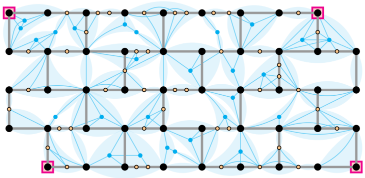


Graph G
Treewidth 2



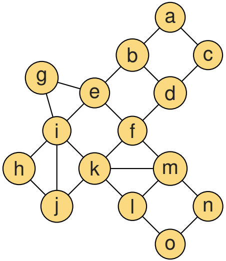
A tree decomposition of G
Width = 2

- Robertson & Seymour: Detect irrelevant vertex in $f(H) \cdot n^2$ time $\Rightarrow f(H) \cdot n^3$ time algorithm

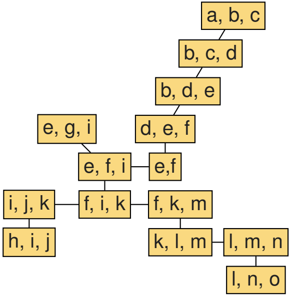


The Irrelevant Vertex technique

- Treewidth of a graph: Parameter between 0 and $n-1$ measuring how tree-like the graph is
- If treewidth of G is $\leq f(H)$, solve the problem by dynamic programming in $f(H) \cdot n$ time
- If treewidth is $> f(H)$, detect and remove an Irrelevant Vertex from G

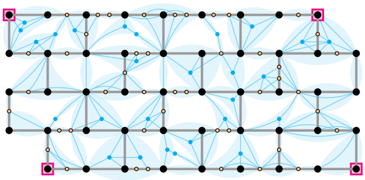


Graph G
Treewidth 2



A tree decomposition of G
Width = 2

- Robertson & Seymour: Detect irrelevant vertex in $f(H) \cdot n^2$ time $\Rightarrow$ $f(H) \cdot n^3$ time algorithm
- Kawarabayashi, Kobayashi & Reed: Detect irrelevant vertex in $f(H) \cdot n$ time $\Rightarrow$ $f(H) \cdot n^2$ time algorithm



Outline of our algorithm

Outline of our algorithm

1. Fast implementation of the irrelevant vertex technique on apex-minor-free graphs

Outline of our algorithm

1. Fast implementation of the irrelevant vertex technique on apex-minor-free graphs
   - Using dynamic treewidth data structure of [K., Majewski, Nadara, Pilipczuk & Sokołowski, 2023]

Outline of our algorithm

1. Fast implementation of the irrelevant vertex technique on apex-minor-free graphs
   - Using dynamic treewidth data structure of [K., Majewski, Nadara, Pilipczuk & Sokołowski, 2023]
2. Reducing unbreakable clique-minor-free graphs to apex-minor-free graphs

Outline of our algorithm

1. Fast implementation of the irrelevant vertex technique on apex-minor-free graphs
   - Using dynamic treewidth data structure of [K., Majewski, Nadara, Pilipczuk & Sokołowski, 2023]
2. Reducing unbreakable clique-minor-free graphs to apex-minor-free graphs
3. Reducing clique-minor-free graphs to unbreakable clique-minor-free graphs

Outline of our algorithm

1. Fast implementation of the irrelevant vertex technique on apex-minor-free graphs
   - Using dynamic treewidth data structure of [K., Majewski, Nadara, Pilipczuk & Sokołowski, 2023]
2. Reducing unbreakable clique-minor-free graphs to apex-minor-free graphs
3. Reducing clique-minor-free graphs to unbreakable clique-minor-free graphs
   - Fast implementation of the recursive understanding technique

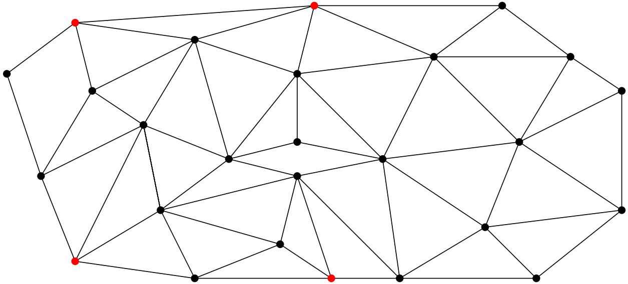
Outline of our algorithm

1. Fast implementation of the irrelevant vertex technique on apex-minor-free graphs
   - Using dynamic treewidth data structure of [K., Majewski, Nadara, Pilipczuk & Sokołowski, 2023]
2. Reducing unbreakable clique-minor-free graphs to apex-minor-free graphs
3. Reducing clique-minor-free graphs to unbreakable clique-minor-free graphs
   - Fast implementation of the recursive understanding technique
   - Using recent breakthroughs in almost-linear time graph algorithms: Isolating cuts [Li & Panigrahi, 2020], almost-linear time (deterministic) max-flow [van den Brand, Chen, Kyng, Liu, Peng, Probst Gutenberg, Sachdeva & Sidford, 2023], and mimicking networks of [Saranurak & Yingchareonthawornchai, 2022]

Outline of our algorithm

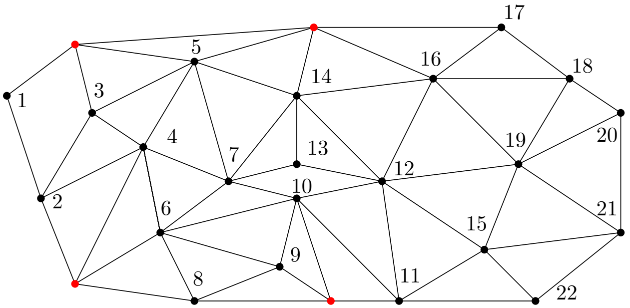
1. Fast implementation of the irrelevant vertex technique on apex-minor-free graphs
   - Using dynamic treewidth data structure of [K., Majewski, Nadara, Pilipczuk & Sokołowski, 2023]
2. Reducing unbreakable clique-minor-free graphs to apex-minor-free graphs
3. Reducing clique-minor-free graphs to unbreakable clique-minor-free graphs
   - Fast implementation of the recursive understanding technique
   - Using recent breakthroughs in almost-linear time graph algorithms: Isolating cuts [Li & Panigrahi, 2020], almost-linear time (deterministic) max-flow [van den Brand, Chen, Kyng, Liu, Peng, Probst Gutenberg, Sachdeva & Sidford, 2023], and mimicking networks of [Saranurak & Yingchareonthawornchai, 2022]
4. Reducing general graphs to clique-minor-free graphs

Algorithm for apex-minor-free graphs



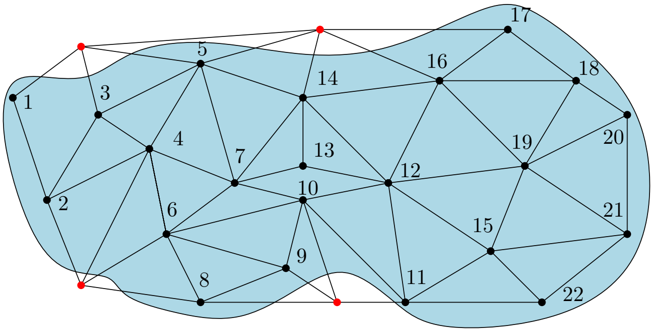
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected



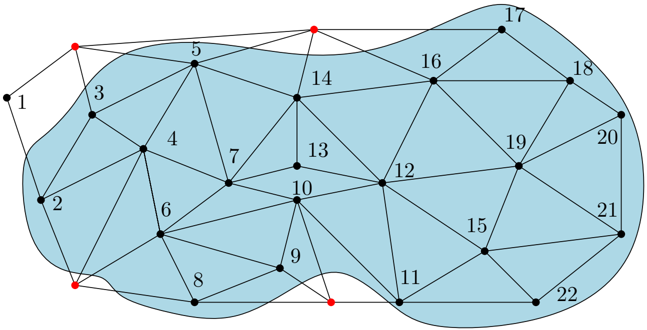
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex



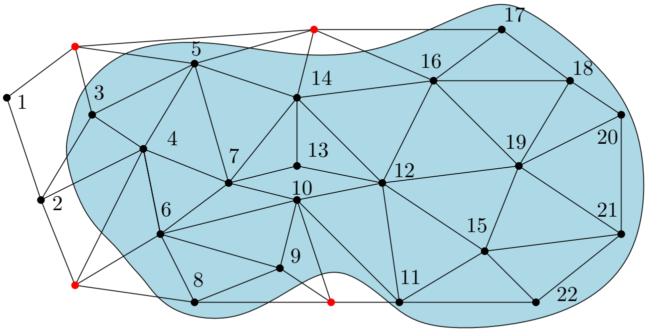
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$



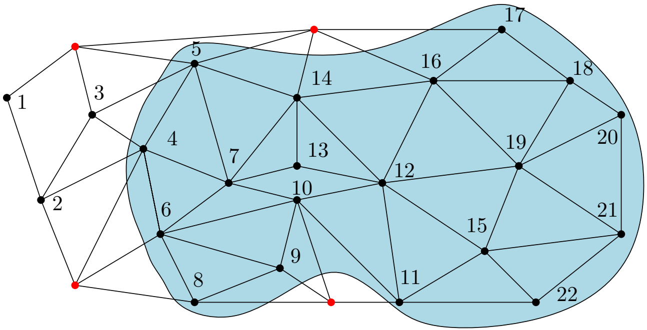
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$



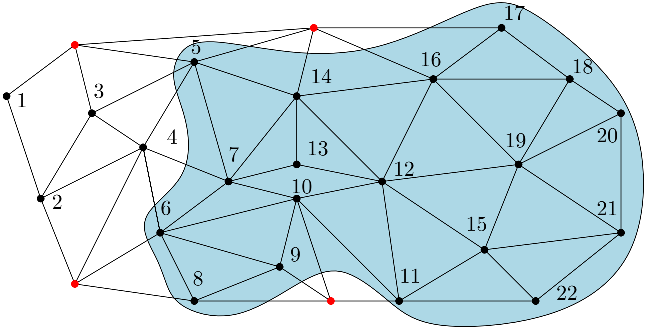
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$



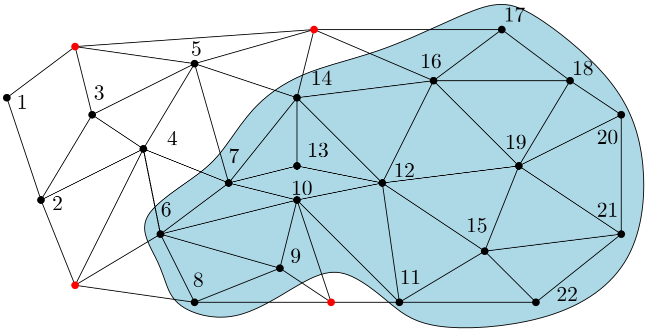
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$



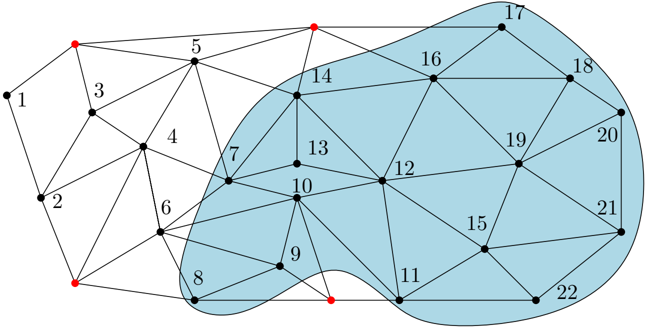
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$



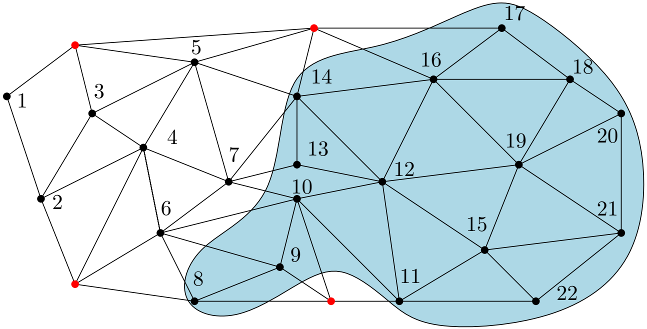
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$



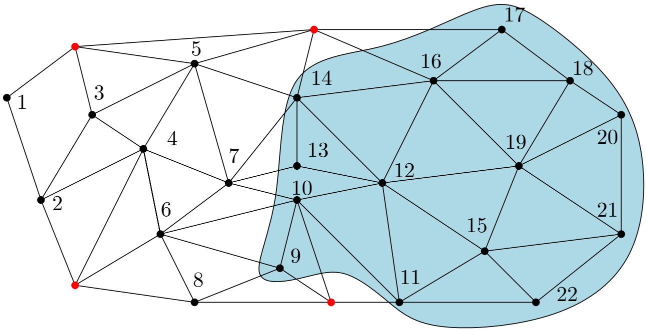
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$



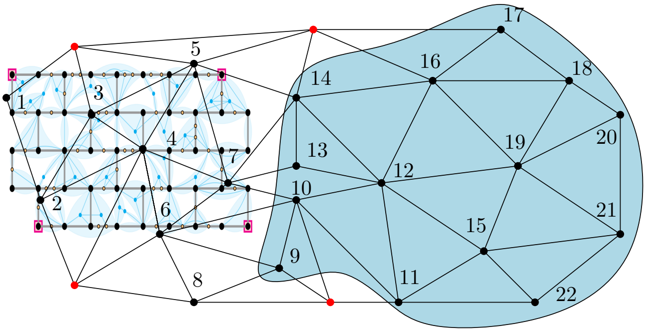
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \dots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \dots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \dots, v_\ell$



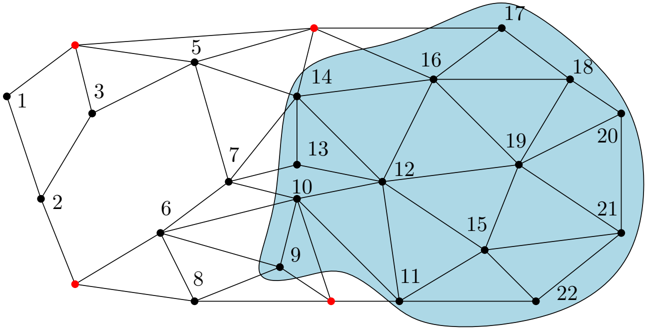
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it



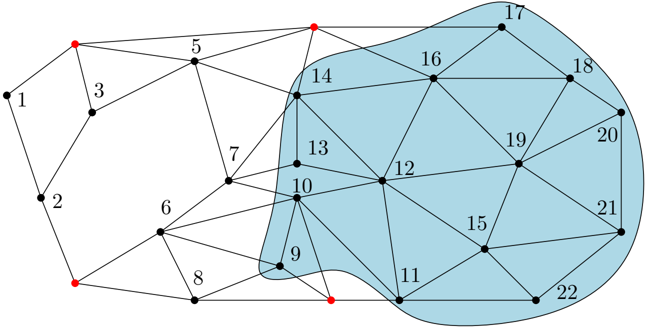
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it



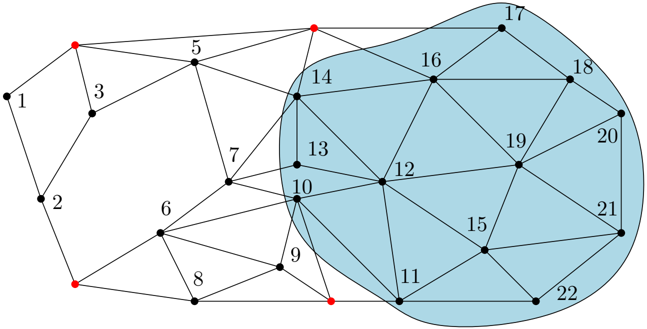
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



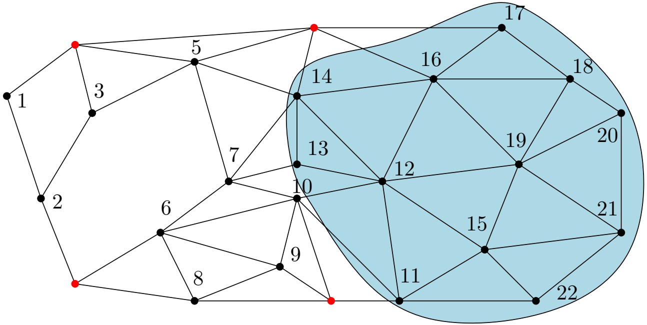
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



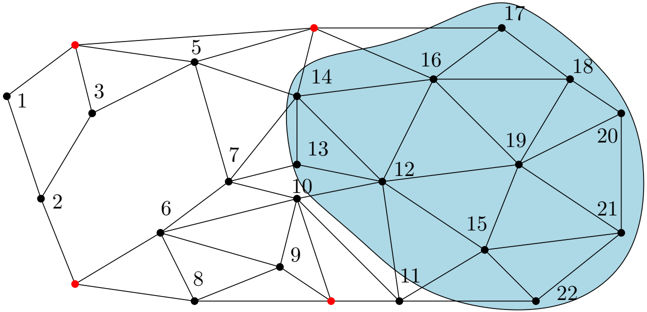
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



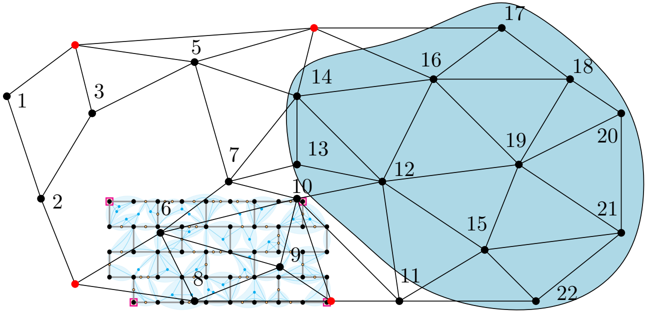
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



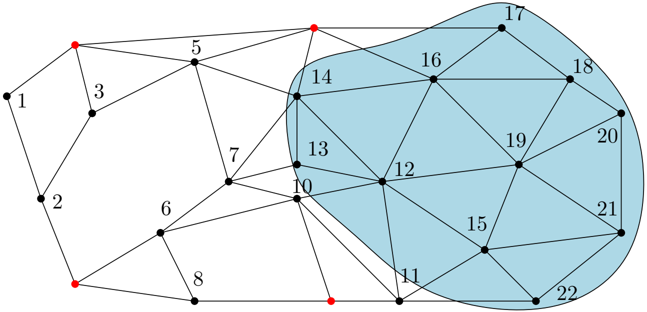
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



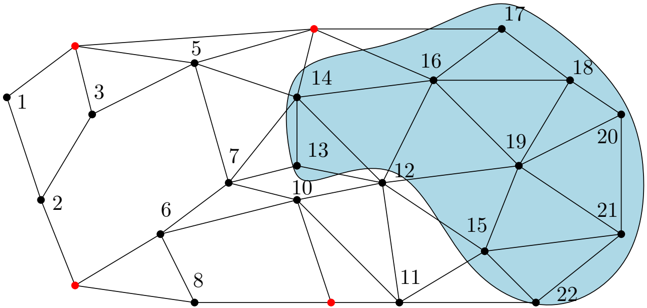
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



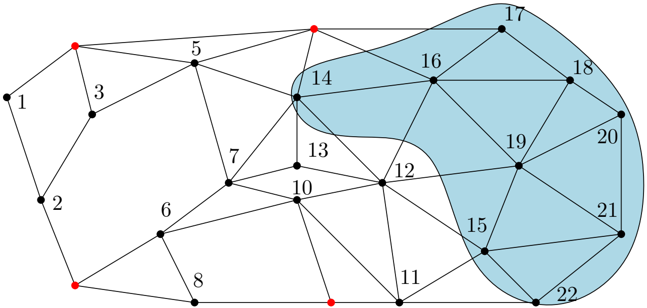
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \dots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \dots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \dots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



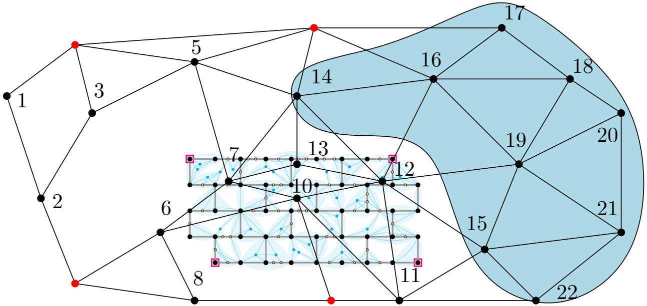
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



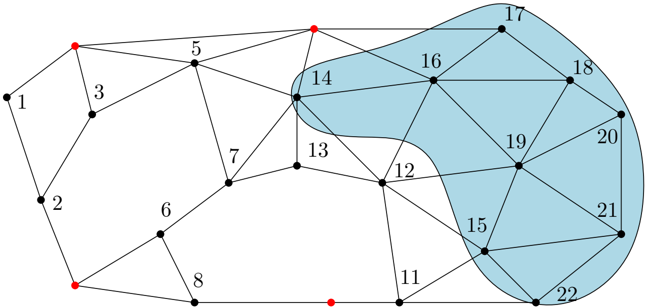
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



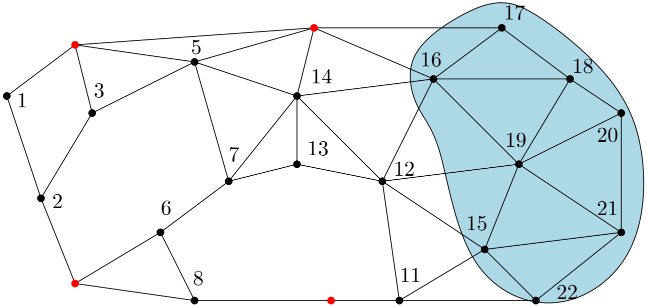
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



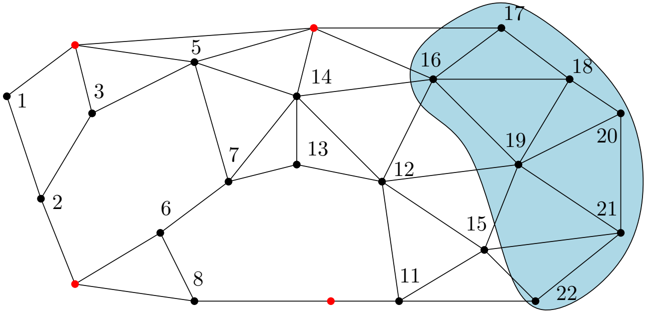
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



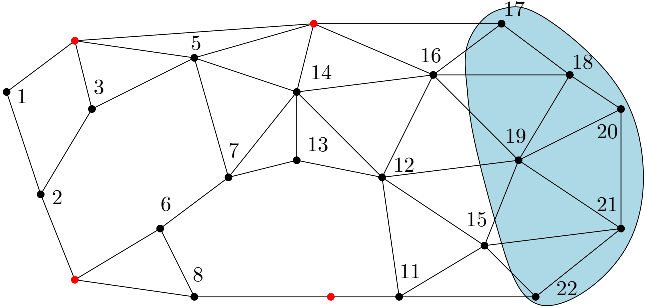
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



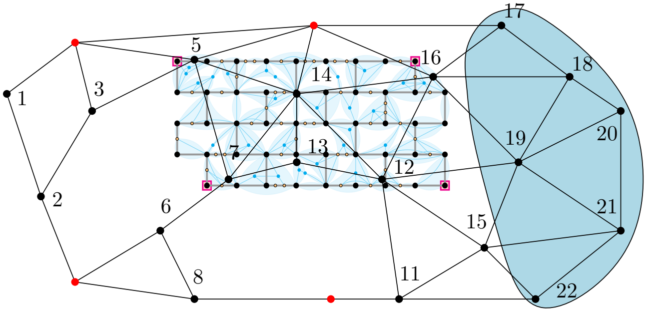
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



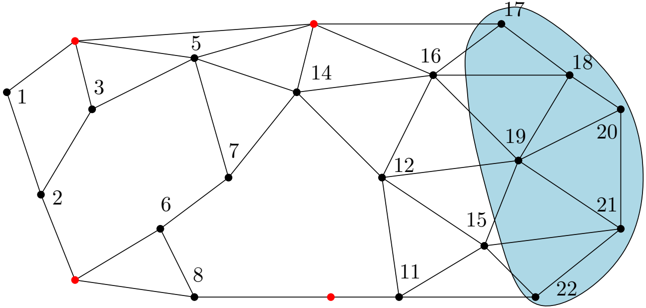
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



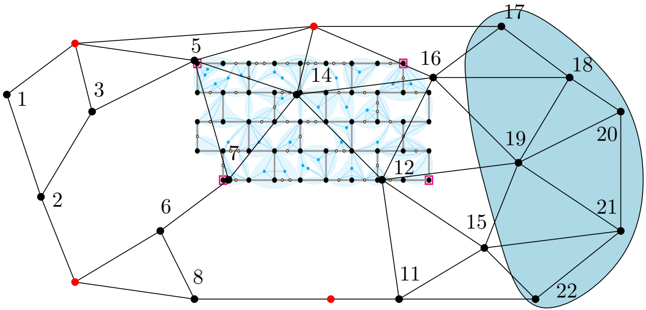
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



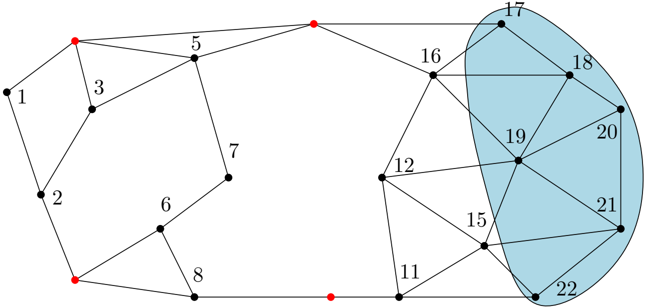
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



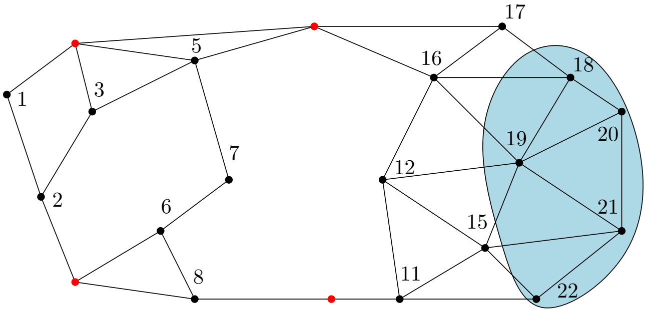
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



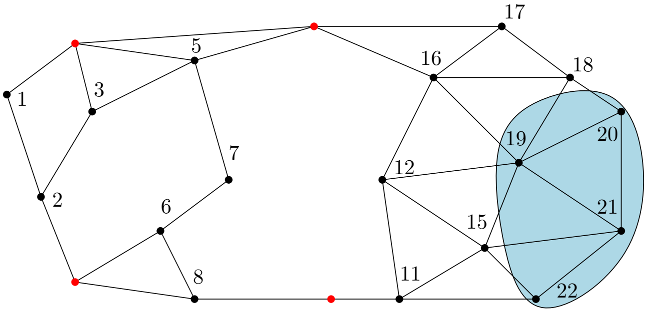
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



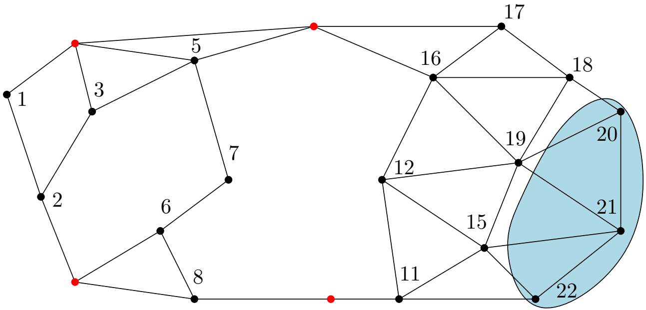
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



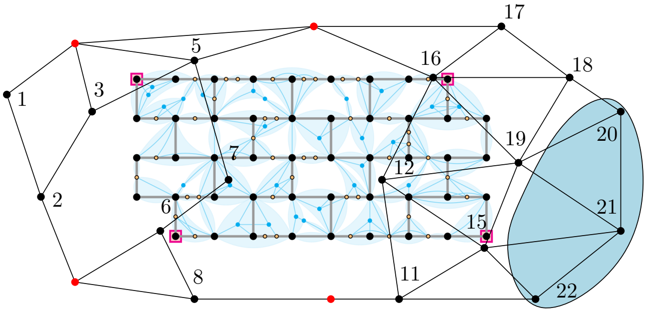
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



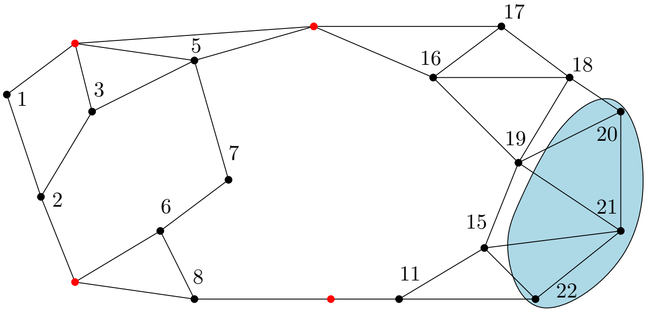
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



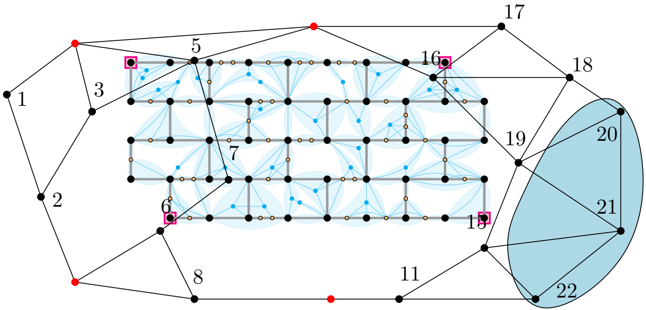
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \dots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \dots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \dots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



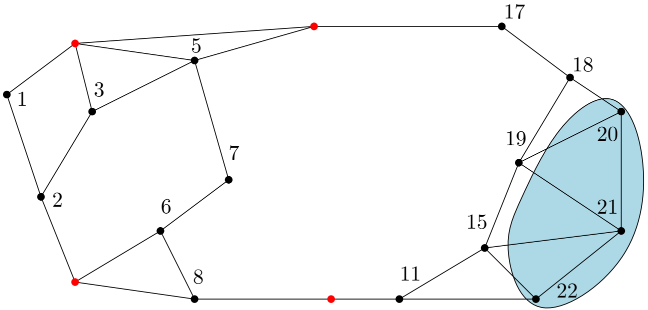
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



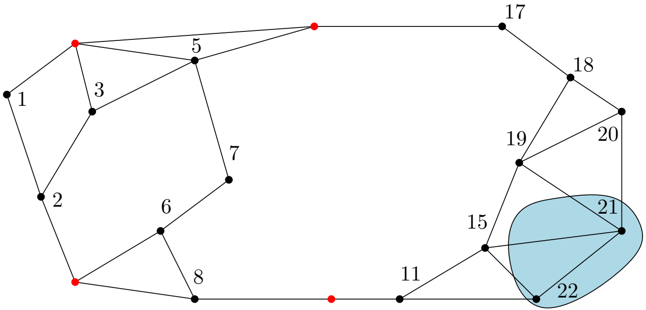
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



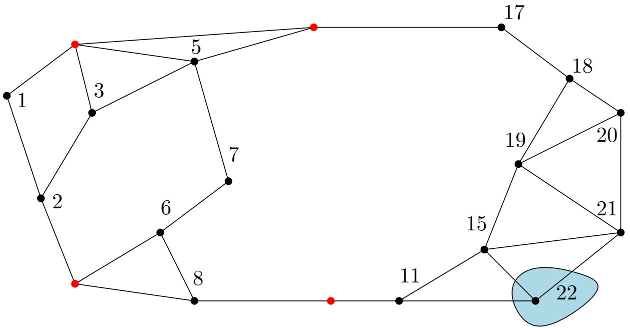
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \ldots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \dots, v_\ell$ of $G - X$ so that every suffix is connected
2. Contract $v_1, \dots, v_\ell$ into a mega-vertex
3. Start uncontracting in the order $v_1, \dots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it
5. Main idea: If the compass of the flat wall does not contain the mega-vertex, then it is the same in the contracted and the original graph



Conclusion

- $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment

Conclusion

- $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment
- Fast irrelevant vertex technique for apex-minor-free graphs using dynamic treewidth

Conclusion

- $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment
- Fast irrelevant vertex technique for apex-minor-free graphs using dynamic treewidth
- Reduction to apex-minor-free using fast recursive understanding

Conclusion

- $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment
- Fast irrelevant vertex technique for apex-minor-free graphs using dynamic treewidth
- Reduction to apex-minor-free using fast recursive understanding

Future work:

Conclusion

- $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment
- Fast irrelevant vertex technique for apex-minor-free graphs using dynamic treewidth
- Reduction to apex-minor-free using fast recursive understanding

Future work:

- Computing the Robertson-Seymour decomposition, topological minor containment

Conclusion

- $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment
- Fast irrelevant vertex technique for apex-minor-free graphs using dynamic treewidth
- Reduction to apex-minor-free using fast recursive understanding

Future work:

- Computing the Robertson-Seymour decomposition, topological minor containment
- Replacing recursive understanding by recent almost-linear time algorithm for unbreakable decomposition by [Anand, Lee, Li, Long & Saranurak, 2024]

Conclusion

- $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment
- Fast irrelevant vertex technique for apex-minor-free graphs using dynamic treewidth
- Reduction to apex-minor-free using fast recursive understanding

Future work:

- Computing the Robertson-Seymour decomposition, topological minor containment
- Replacing recursive understanding by recent almost-linear time algorithm for unbreakable decomposition by [Anand, Lee, Li, Long & Saranurak, 2024]
- Optimization to $f(H) \cdot m \operatorname{polylog} n$?

Conclusion

- $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment
- Fast irrelevant vertex technique for apex-minor-free graphs using dynamic treewidth
- Reduction to apex-minor-free using fast recursive understanding

Future work:

- Computing the Robertson-Seymour decomposition, topological minor containment
- Replacing recursive understanding by recent almost-linear time algorithm for unbreakable decomposition by [Anand, Lee, Li, Long & Saranurak, 2024]
- Optimization to $f(H) \cdot m \text{ polylog } n$?
  - Important problem: Optimization of dynamic treewidth to $f(k) \cdot \text{polylog } n$?

Conclusion

- $f(H) \cdot m^{1+o(1)}$ time algorithm for Rooted Minor Containment
- Fast irrelevant vertex technique for apex-minor-free graphs using dynamic treewidth
- Reduction to apex-minor-free using fast recursive understanding

Future work:

- Computing the Robertson-Seymour decomposition, topological minor containment
- Replacing recursive understanding by recent almost-linear time algorithm for unbreakable decomposition by [Anand, Lee, Li, Long & Saranurak, 2024]
- Optimization to $f(H) \cdot m$ polylog n?
  - Important problem: Optimization of dynamic treewidth to $f(k) \cdot$ polylog n?

Thank you!