Minor Containment and Disjoint Paths in almost-linear time

Tuukka Korhonen

UNIVERSITY OF BERGEN

based on joint work with Michał Pilipczuk and Giannos Stamoulis
from the University of Warsaw

LIRMM, Montpellier

9 April 2024

The Graph Minors series

The Graph Minors series

- A series of 23 papers in 1983–2012 by Robertson & Seymour


The Graph Minors series

- A series of 23 papers in 1983–2012 by Robertson & Seymour

Theorem (Robertson & Seymour, Graph Minors 20)

Any infinite set of graphs contains graphs $G_1 \neq G_2$ s.t. G_1 is a minor of G_2.

The Graph Minors series

- A series of 23 papers in 1983–2012 by Robertson & Seymour

Theorem (Robertson & Seymour, Graph Minors 20)

Any infinite set of graphs contains graphs $G_1 \neq G_2$ s.t. G_1 is a minor of G_2.

$\Rightarrow$ Every minor-closed graph class is characterized by a finite set of obstructions

The Graph Minors series

- A series of 23 papers in 1983–2012 by Robertson & Seymour

Theorem (Robertson & Seymour, Graph Minors 20)

Any infinite set of graphs contains graphs $G_1 \neq G_2$ s.t. G_1 is a minor of G_2.

$\Rightarrow$ Every minor-closed graph class is characterized by a finite set of obstructions

Theorem (Robertson & Seymour, Graphs Minors 13)

There is an $\mathcal{O}_H(n^3)$ time algorithm to test if H is a minor of G.

The Graph Minors series

- A series of 23 papers in 1983–2012 by Robertson & Seymour


Theorem (Robertson & Seymour, Graph Minors 20)

Any infinite set of graphs contains graphs $G_1 \neq G_2$ s.t. G_1 is a minor of G_2.

$\Rightarrow$ Every minor-closed graph class is characterized by a finite set of obstructions

Theorem (Robertson & Seymour, Graphs Minors 13)

There is an $\mathcal{O}_H(n^3)$ time algorithm to test if H is a minor of G.

$\Rightarrow$ Every minor-closed graph class has a $\mathcal{O}(n^3)$ recognition algorithm

The Graph Minors series

- A series of 23 papers in 1983–2012 by Robertson & Seymour

Theorem (Robertson & Seymour, Graph Minors 20)

Any infinite set of graphs contains graphs $G_1 \neq G_2$ s.t. G_1 is a minor of G_2.

$\Rightarrow$ Every minor-closed graph class is characterized by a finite set of obstructions

Theorem (Robertson & Seymour, Graphs Minors 13)

There is an $\mathcal{O}_H(n^3)$ time algorithm to test if H is a minor of G.

$\Rightarrow$ Every minor-closed graph class has a $\mathcal{O}(n^3)$ recognition algorithm

- Improved to $\mathcal{O}_H(n^2)$ by [Kawarabayashi, Kobayashi & Reed, 2012].

The Graph Minors series

- A series of 23 papers in 1983–2012 by Robertson & Seymour

Theorem (Robertson & Seymour, Graph Minors 20)

Any infinite set of graphs contains graphs $G_1 \neq G_2$ s.t. G_1 is a minor of G_2.

$\Rightarrow$ Every minor-closed graph class is characterized by a finite set of obstructions

Theorem (Robertson & Seymour, Graphs Minors 13)

There is an $\mathcal{O}_H(n^3)$ time algorithm to test if H is a minor of G.

$\Rightarrow$ Every minor-closed graph class has a $\mathcal{O}(n^3)$ recognition algorithm

- Improved to $\mathcal{O}_H(n^2)$ by [Kawarabayashi, Kobayashi & Reed, 2012].
- These algorithms work for ROOTED MINOR CONTAINMENT, which captures also the k-DISJOINT PATHS problem

The Graph Minors series

- A series of 23 papers in 1983–2012 by Robertson & Seymour

Theorem (Robertson & Seymour, Graph Minors 20)

Any infinite set of graphs contains graphs $G_1 \neq G_2$ s.t. G_1 is a minor of G_2.

$\Rightarrow$ Every minor-closed graph class is characterized by a finite set of obstructions

Theorem (Robertson & Seymour, Graphs Minors 13)

There is an $\mathcal{O}_H(n^3)$ time algorithm to test if H is a minor of G.

$\Rightarrow$ Every minor-closed graph class has a $\mathcal{O}(n^3)$ recognition algorithm

- Improved to $\mathcal{O}_H(n^2)$ by [Kawarabayashi, Kobayashi & Reed, 2012].
- These algorithms work for ROOTED MINOR CONTAINMENT, which captures also the k-DISJOINT PATHS problem
- Linear-time algorithms known for some special cases, like planar graphs [Bodlaender 1993], [Reed, Robertson, Schrijver & Seymour 1993]

Our result

Theorem (This work)

There is an $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for ROOTED MINOR CONTAINMENT

Our result

Theorem (This work)

There is an $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for ROOTED MINOR CONTAINMENT

- $m = |V(G)| + |E(G)|$ the number of vertices + edges of G
- $X \subseteq V(G)$ the set of roots

Our result

Theorem (This work)

There is an $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for ROOTED MINOR CONTAINMENT

- $m = |V(G)| + |E(G)|$ the number of vertices + edges of G
- $X \subseteq V(G)$ the set of roots
- Dependence on H and $|X|$ huge but computable.

Our result

Theorem (This work)

There is an $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for Rooted Minor Containment

- $m = |V(G)| + |E(G)|$ the number of vertices + edges of G
- $X \subseteq V(G)$ the set of roots
- Dependence on H and $|X|$ huge but computable.

Corollary

Every minor-closed graph class has a $\mathcal{O}(n^{1+o(1)})$ recognition algorithm

Our result

Theorem (This work)

There is an $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for Rooted Minor Containment

- $m = |V(G)| + |E(G)|$ the number of vertices + edges of G
- $X \subseteq V(G)$ the set of roots
- Dependence on H and $|X|$ huge but computable.

Corollary

Every minor-closed graph class has a $\mathcal{O}(n^{1+o(1)})$ recognition algorithm

Corollary

There is a $\mathcal{O}_k(m^{1+o(1)})$ time algorithm for k-Disjoint Paths

Outline of the algorithm

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

Outline of the 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 the 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]
   - In the generality where $G - X$ is apex-minor-free

Outline of the 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]
   - In the generality where $G - X$ is apex-minor-free

2. Reducing compact clique-minor-free graphs to apex-minor-free graphs

Outline of the 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]
   - In the generality where $G - X$ is apex-minor-free

2. Reducing compact clique-minor-free graphs to apex-minor-free graphs
   - Standard fact that near-embeddable graphs are almost apex-minor-free

Outline of the 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]
   - In the generality where $G - X$ is apex-minor-free
2. Reducing compact clique-minor-free graphs to apex-minor-free graphs
   - Standard fact that near-embeddable graphs are almost apex-minor-free
3. Reducing clique-minor-free graphs to compact clique-minor-free graphs

Outline of the 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]
   - In the generality where $G - X$ is apex-minor-free
2. Reducing compact clique-minor-free graphs to apex-minor-free graphs
   - Standard fact that near-embeddable graphs are almost apex-minor-free
3. Reducing clique-minor-free graphs to compact clique-minor-free graphs
   - Fast implementation of the recursive understanding technique

Outline of the 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]
   - In the generality where $G - X$ is apex-minor-free
2. Reducing compact clique-minor-free graphs to apex-minor-free graphs
   - Standard fact that near-embeddable graphs are almost apex-minor-free
3. Reducing clique-minor-free graphs to compact 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 (deterministic) max-flow [van den Brand, Chen, Kyng, Liu, Peng, Probst Gutenberg, Sachdeva & Sidford, 2023], and mimicking networks of [Saranurak & Yingchareonthawornchai, 2022]

Outline of the 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]
   - In the generality where $G - X$ is apex-minor-free
2. Reducing compact clique-minor-free graphs to apex-minor-free graphs
   - Standard fact that near-embeddable graphs are almost apex-minor-free
3. Reducing clique-minor-free graphs to compact 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 (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

Outline of the 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]
   - In the generality where $G - X$ is apex-minor-free
2. Reducing compact clique-minor-free graphs to apex-minor-free graphs
   - Standard fact that near-embeddable graphs are almost apex-minor-free
3. Reducing clique-minor-free graphs to compact 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 (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
   - Using a lemma of Robertson & Seymour about generic folios when containing a clique minor

Outline of the 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]
   - In the generality where $G - X$ is apex-minor-free
2. Reducing compact clique-minor-free graphs to apex-minor-free graphs
   - Standard fact that near-embeddable graphs are almost apex-minor-free
3. Reducing clique-minor-free graphs to compact 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 (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
   - Using a lemma of Robertson & Seymour about generic folios when containing a clique minor
   - Need again the graph to be compact

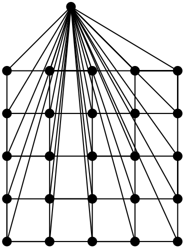
The algorithm for apex-minor-free graphs

The algorithm for apex-minor-free graphs

Goal: Given

- a graph G (with m vertices+edges),
- a set $X \subseteq V(G)$,
- an X-rooted graph H, and
- an integer p,

The algorithm for apex-minor-free graphs



Goal: Given

- a graph G (with m vertices+edges),
- a set $X \subseteq V(G)$,
- an X-rooted graph H, and
- an integer p,

return in time $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ either

1. whether H is an X-rooted minor of G (and a minor model of H), or
2. a minor model of an apex-grid of order p in $G - X$.

The irrelevant vertex technique

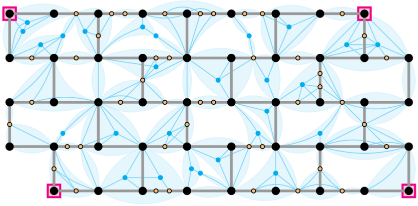
- Robertson & Seymour: If G has large treewidth, we can remove a vertex from G without changing the solution

The irrelevant vertex technique

- Robertson & Seymour: If G has large treewidth, we can remove a vertex from G without changing the solution
- On minor-free graphs, this works via flat walls:

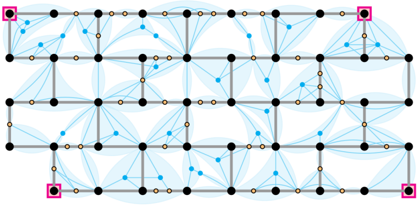
The irrelevant vertex technique

- Robertson & Seymour: If G has large treewidth, we can remove a vertex from G without changing the solution
- On minor-free graphs, this works via flat walls:
  - If G excludes K_t-minor and has high treewidth, then there is a set $Z \subseteq V(G)$ with $|Z| \leq t^2$ so that $G - Z$ contains a large flat wall



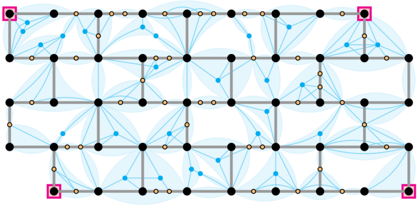
The irrelevant vertex technique

- Robertson & Seymour: If G has large treewidth, we can remove a vertex from G without changing the solution
- On minor-free graphs, this works via flat walls:
  - If G excludes K_t-minor and has high treewidth, then there is a set $Z \subseteq V(G)$ with $|Z| \leq t^2$ so that $G - Z$ contains a large flat wall
  - If v is a central vertex of a homogeneous flat wall, then v is irrelevant



The irrelevant vertex technique

- Robertson & Seymour: If G has large treewidth, we can remove a vertex from G without changing the solution
- On minor-free graphs, this works via flat walls:
  - If G excludes K_t-minor and has high treewidth, then there is a set $Z \subseteq V(G)$ with $|Z| \leq t^2$ so that $G - Z$ contains a large flat wall
  - If v is a central vertex of a homogeneous flat wall, then v is irrelevant
- If $G - X$ is apex-minor-free, then we can assume $Z = X$



Dynamic treewidth

Dynamic treewidth

Theorem [K., Majewski, Nadara, Pilipczuk & Sokołowski, FOCS 2023]:

There is data structure that

- is initialized with integer k and empty n-vertex graph G
- supports edge insertions and deletions in amortized time $f(k) \cdot 2^{\sqrt{\log n} \log \log n} = f(k) \cdot n^{o(1)}$ under the promise that the treewidth of G never exceeds k
- maintains a tree decomposition of G of width at most $6k + 5$

Dynamic treewidth

Theorem [K., Majewski, Nadara, Pilipczuk & Sokołowski, FOCS 2023]:

There is data structure that

- is initialized with integer k and empty n-vertex graph G
- supports edge insertions and deletions in amortized time $f(k) \cdot 2^{\sqrt{\log n}\log\log n} = f(k) \cdot n^{o(1)}$ under the promise that the treewidth of G never exceeds k
- maintains a tree decomposition of G of width at most $6k + 5$
- can also maintain any dynamic programming scheme on the decomposition within similar running time (formalized by $CMSO_2$)

Dynamic treewidth

Theorem [K., Majewski, Nadara, Pilipczuk & Sokołowski, FOCS 2023]:

There is data structure that

- is initialized with integer k and empty n-vertex graph G
- supports edge insertions and deletions in amortized time $f(k) \cdot 2^{\sqrt{\log n} \log \log n} = f(k) \cdot n^{o(1)}$ under the promise that the treewidth of G never exceeds k
- maintains a tree decomposition of G of width at most $6k + 5$
- can also maintain any dynamic programming scheme on the decomposition within similar running time (formalized by CMSO_2)
- ...can easily be extended to also support vertex-colors and to output a vertex v that satisfies a CMSO_2 formula $\varphi(v)$

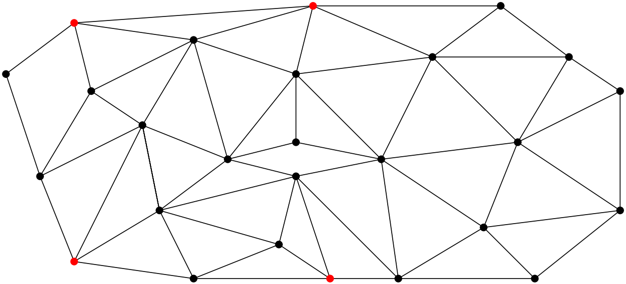
Dynamic treewidth

Theorem [K., Majewski, Nadara, Pilipczuk & Sokołowski, FOCS 2023]:

There is data structure that

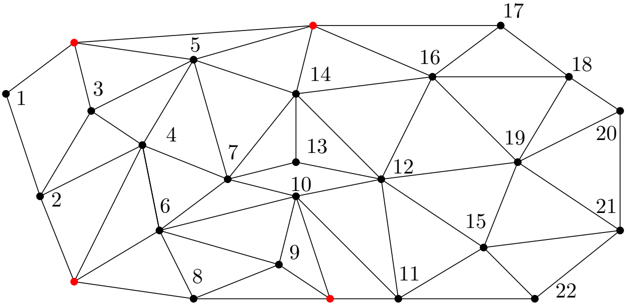
- is initialized with integer k and empty n-vertex graph G
- supports edge insertions and deletions in amortized time $f(k) \cdot 2^{\sqrt{\log n} \log \log n} = f(k) \cdot n^{o(1)}$ under the promise that the treewidth of G never exceeds k
- maintains a tree decomposition of G of width at most $6k + 5$
- can also maintain any dynamic programming scheme on the decomposition within similar running time (formalized by CMSO_2)
- ...can easily be extended to also support vertex-colors and to output a vertex v that satisfies a CMSO_2 formula $\varphi(v)$
- ...and to deletions of vertices and insertions of isolated vertices

Algorithm for apex-minor-free graphs



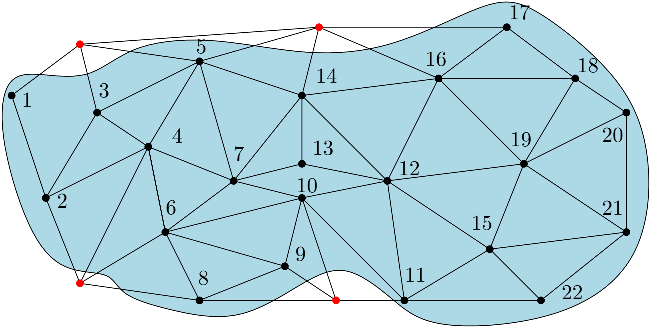
Algorithm for apex-minor-free graphs

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



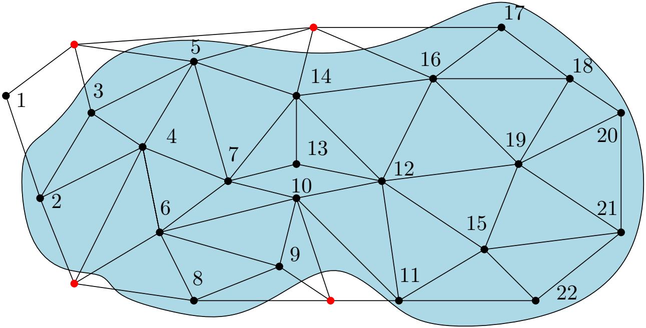
Algorithm for apex-minor-free graphs

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



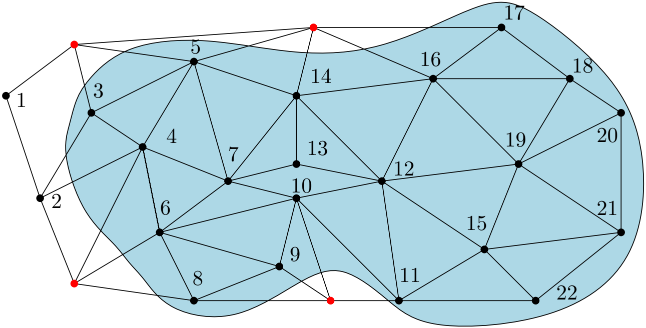
Algorithm for apex-minor-free graphs

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 $v_1, \ldots, v_\ell$



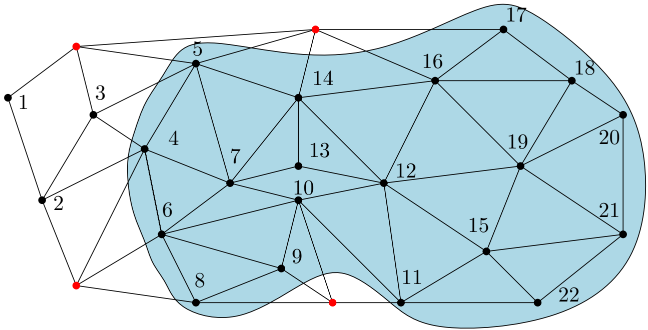
Algorithm for apex-minor-free graphs

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 $v_1, \ldots, v_\ell$



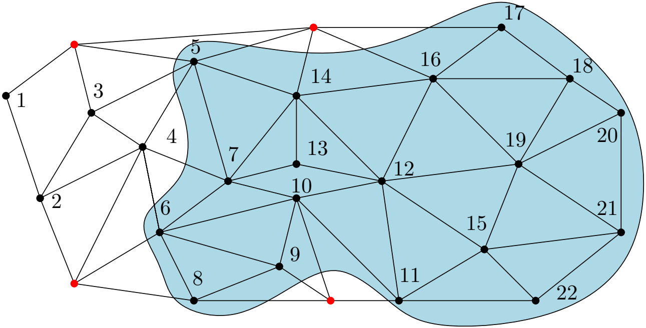
Algorithm for apex-minor-free graphs

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 $v_1, \ldots, v_\ell$



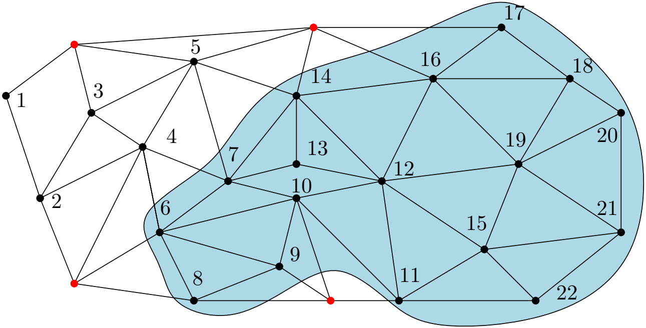
Algorithm for apex-minor-free graphs

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 $v_1, \ldots, v_\ell$



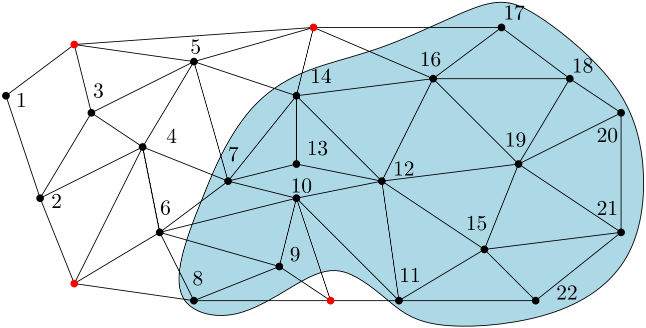
Algorithm for apex-minor-free graphs

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 $v_1, \ldots, v_\ell$



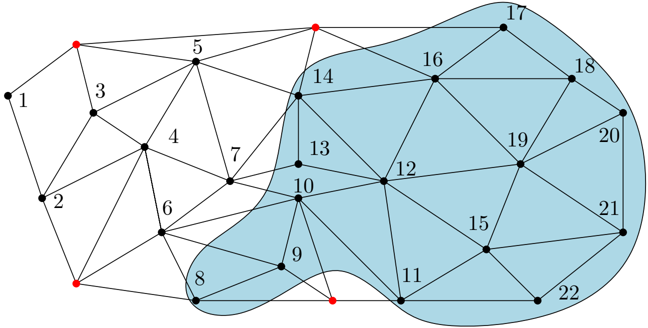
Algorithm for apex-minor-free graphs

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 $v_1, \ldots, v_\ell$



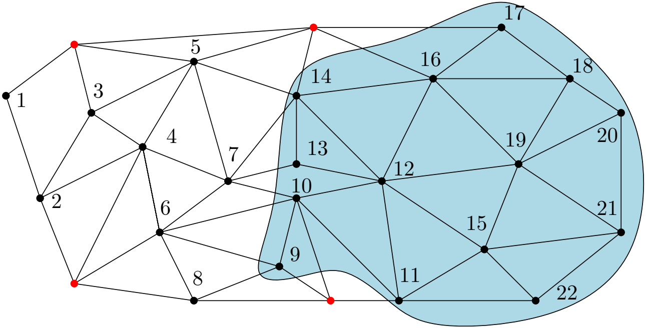
Algorithm for apex-minor-free graphs

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 $v_1, \ldots, v_\ell$



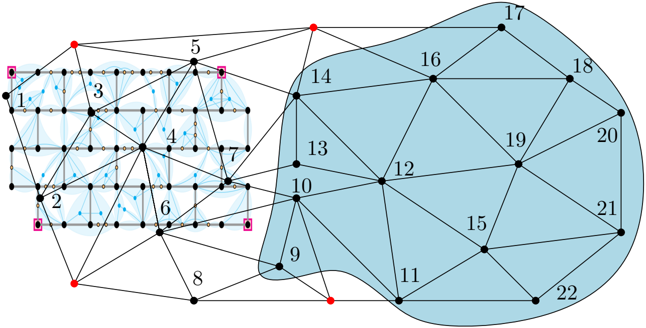
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ 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 $v_1, \ldots, v_\ell$



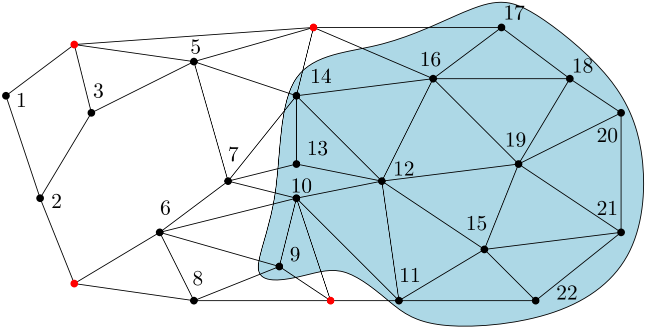
Algorithm for apex-minor-free graphs

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 $v_1, \ldots, v_\ell$
4. When treewidth becomes large, find a flat wall whose compass does not contain the mega-vertex, and delete an irrelevant vertex from it



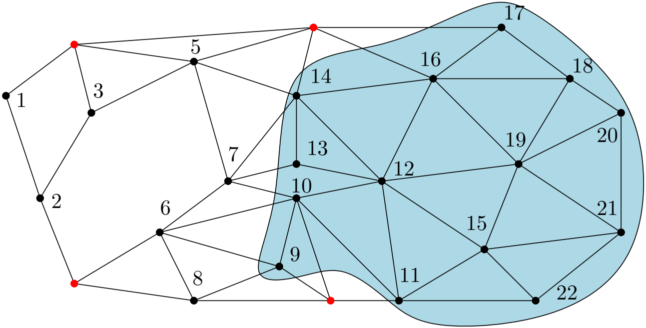
Algorithm for apex-minor-free graphs

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



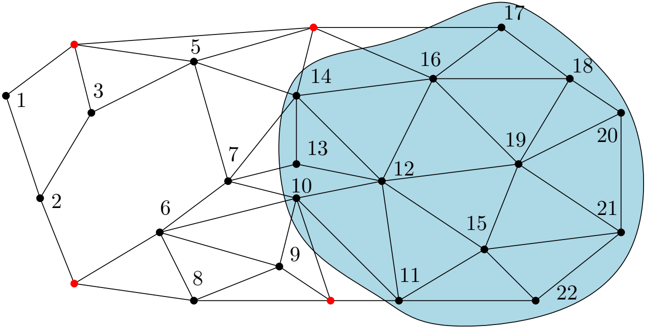
Algorithm for apex-minor-free graphs

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



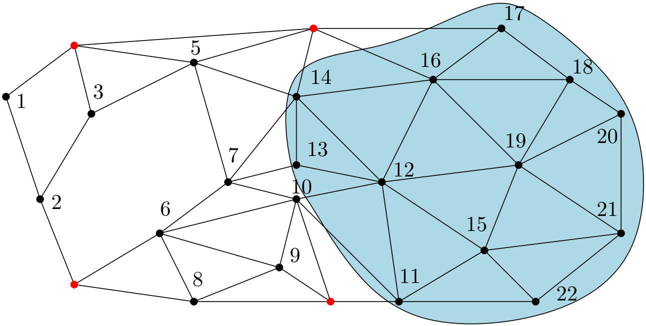
Algorithm for apex-minor-free graphs

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



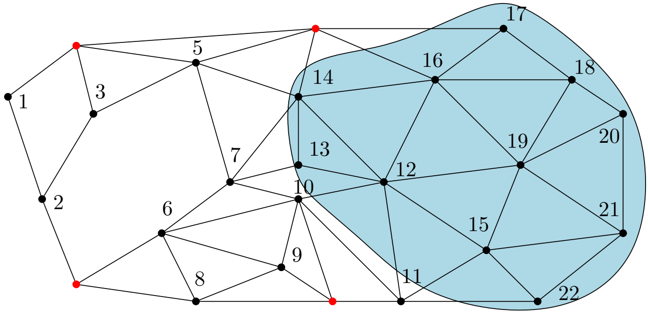
Algorithm for apex-minor-free graphs

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



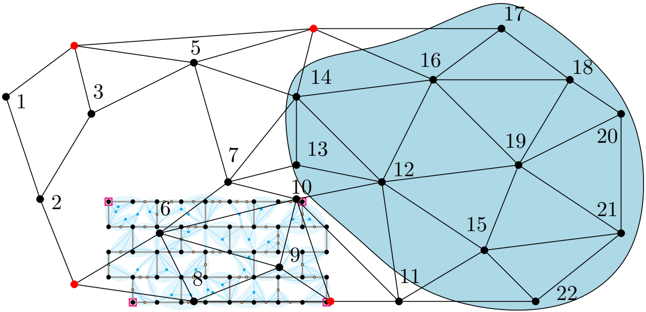
Algorithm for apex-minor-free graphs

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



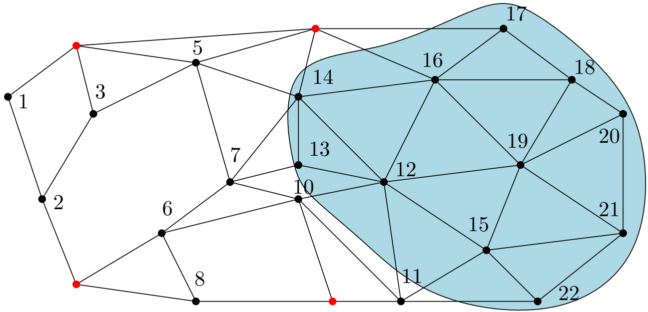
Algorithm for apex-minor-free graphs

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



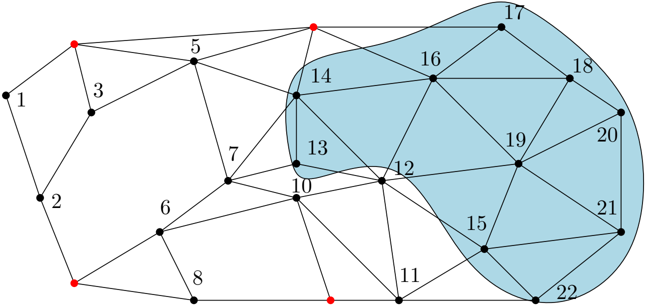
Algorithm for apex-minor-free graphs

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



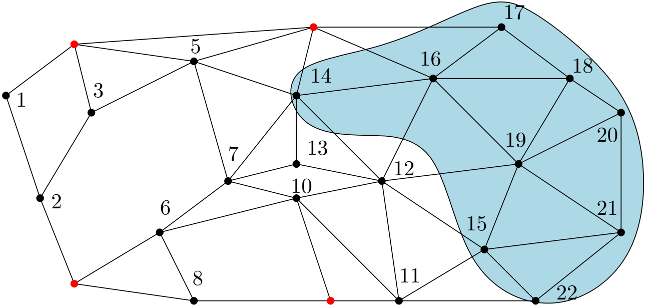
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ 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



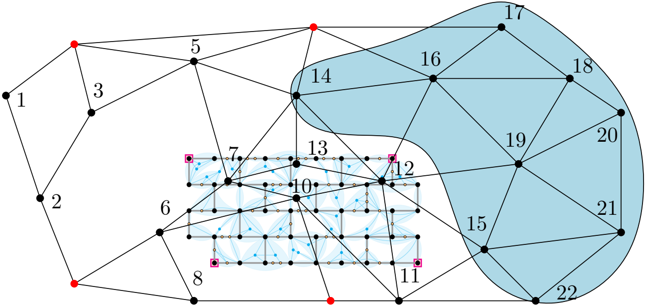
Algorithm for apex-minor-free graphs

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



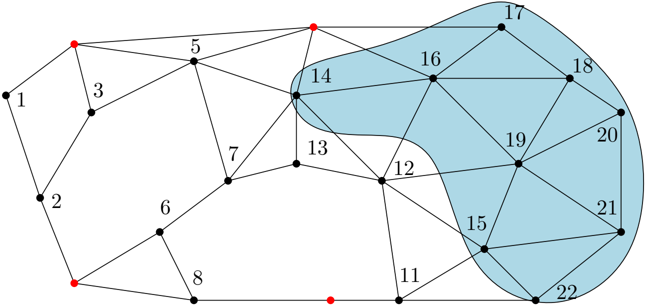
Algorithm for apex-minor-free graphs

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



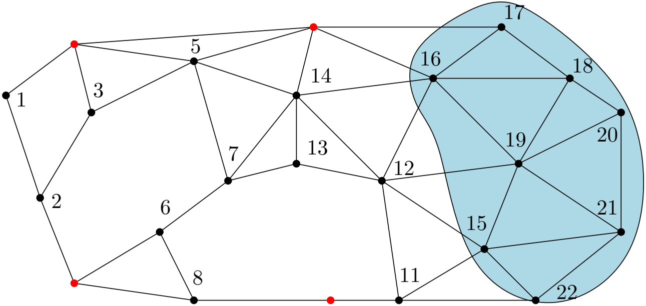
Algorithm for apex-minor-free graphs

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



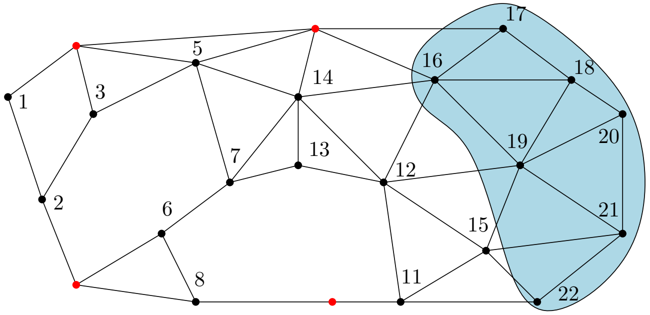
Algorithm for apex-minor-free graphs

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



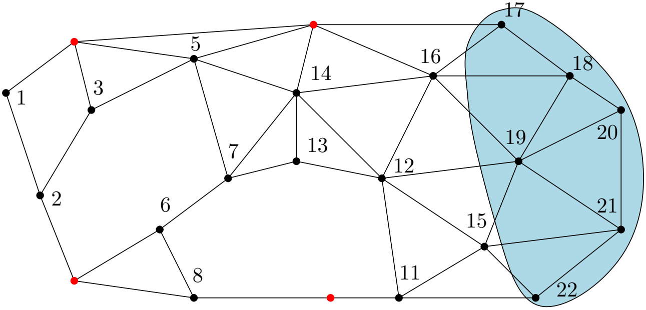
Algorithm for apex-minor-free graphs

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



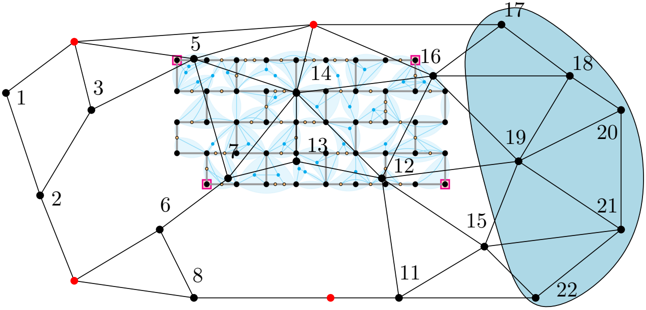
Algorithm for apex-minor-free graphs

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



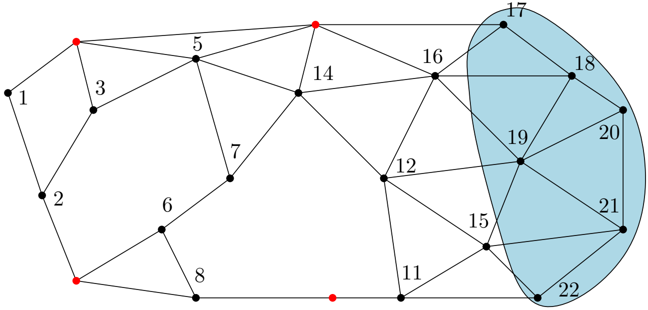
Algorithm for apex-minor-free graphs

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



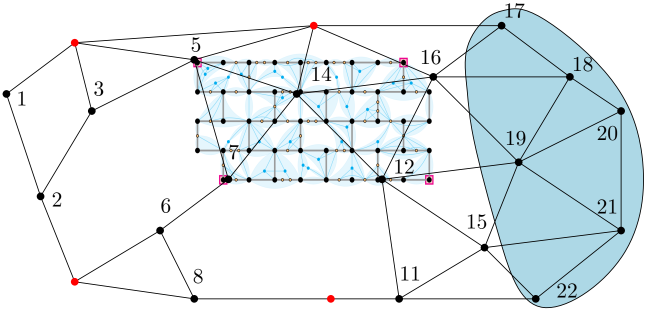
Algorithm for apex-minor-free graphs

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



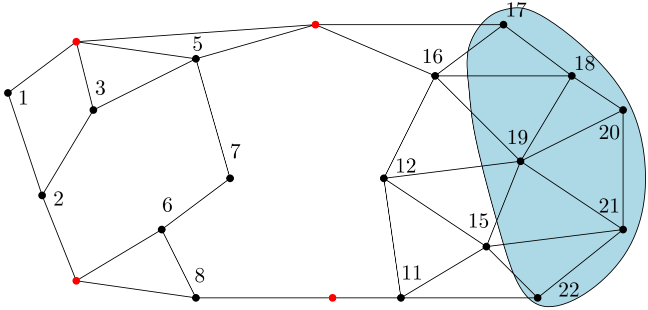
Algorithm for apex-minor-free graphs

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



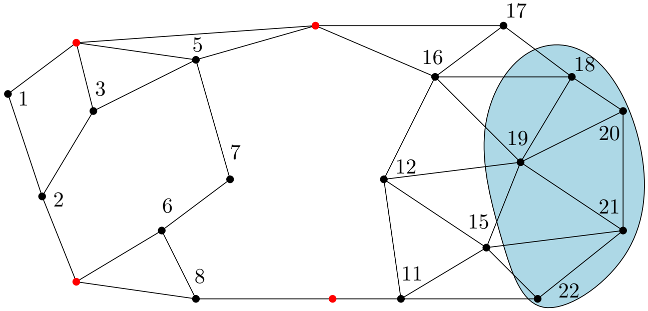
Algorithm for apex-minor-free graphs

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



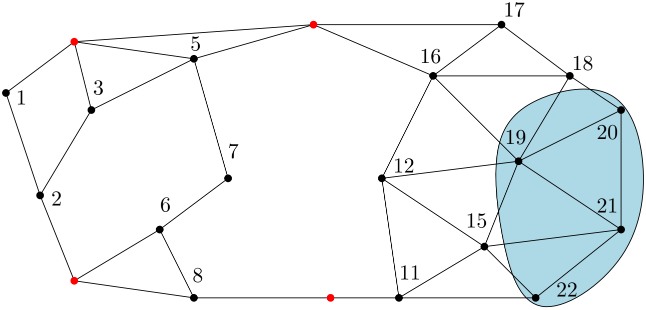
Algorithm for apex-minor-free graphs

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



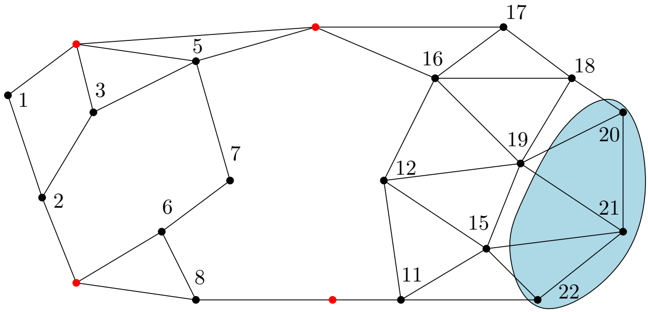
Algorithm for apex-minor-free graphs

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



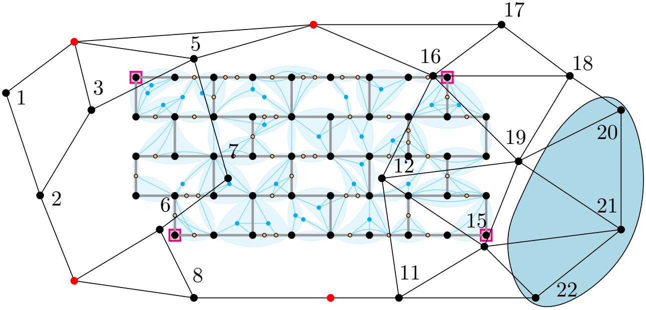
Algorithm for apex-minor-free graphs

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



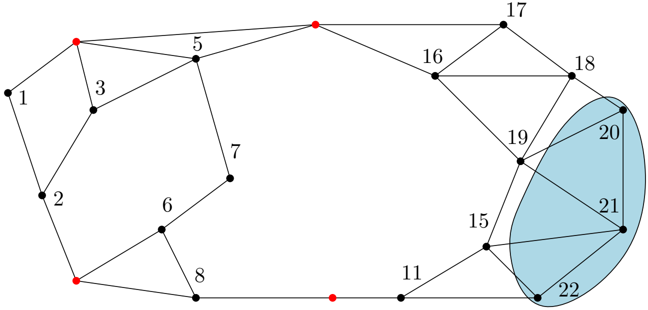
Algorithm for apex-minor-free graphs

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



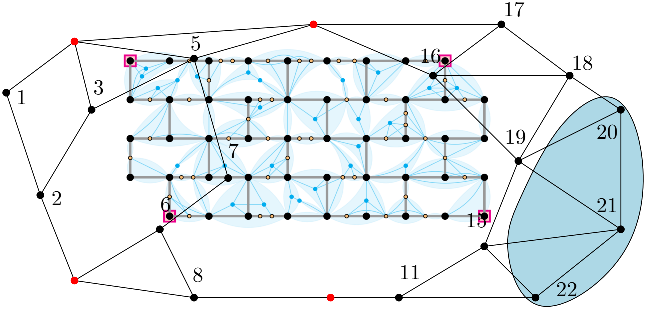
Algorithm for apex-minor-free graphs

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



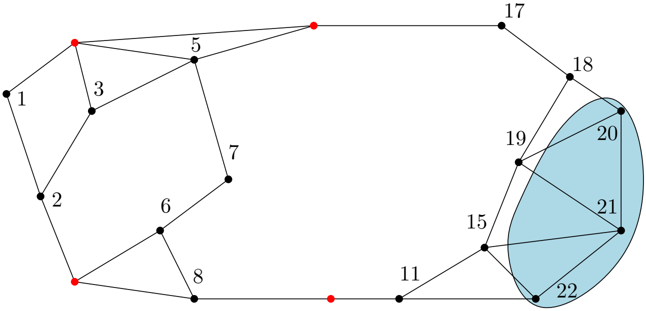
Algorithm for apex-minor-free graphs

1. Find an ordering $v_1, \ldots, v_\ell$ 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



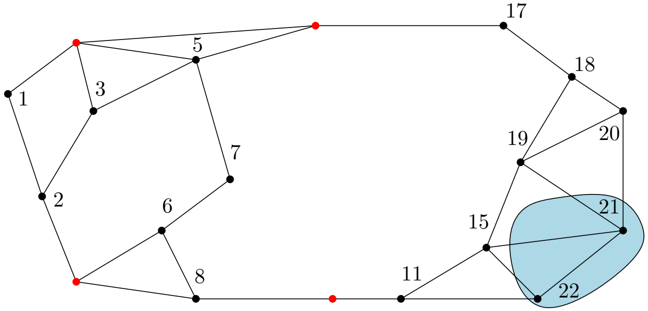
Algorithm for apex-minor-free graphs

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



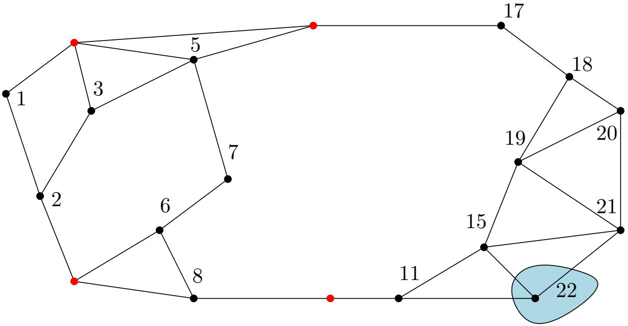
Algorithm for apex-minor-free graphs

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



Some details

Details:

Some details

Details:

1. Apex-minor-free graphs admit flat walls without deleting vertices

Some details

Details:

1. Apex-minor-free graphs admit flat walls without deleting vertices
2. We can always select a flat wall whose compass does not contain the megavertex

Some details

Details:

1. Apex-minor-free graphs admit flat walls without deleting vertices
2. We can always select a flat wall whose compass does not contain the megavertex
3. We can define "central vertex of a homogeneous flat wall" in MSO_2

Some details

Details:

1. Apex-minor-free graphs admit flat walls without deleting vertices
2. We can always select a flat wall whose compass does not contain the megavertex
3. We can define “central vertex of a homogeneous flat wall” in MSO_2 (16 pages of writing)

Compact graphs

Let $X \subseteq V(G)$ be the terminals.

G, X is (k, α)-compact if

Compact graphs

Let $X \subseteq V(G)$ be the terminals.

G, X is (k, α)-compact if

- X is well-linked

Compact graphs

Let $X \subseteq V(G)$ be the terminals.

G, X is (k, α)-compact if

- X is well-linked
- There is no connected set $C \subseteq V(G)$ with $|C| \geq \alpha$ and $|N(C)| < k$.

Compact graphs

Let $X \subseteq V(G)$ be the terminals.

G, X is (k, α)-compact if

- X is well-linked
- There is no connected set $C \subseteq V(G)$ with $|C| \geq \alpha$ and $|N(C)| < k$.

Lemma (Follows from RS-decomposition)

If G excludes K_t, there is $k = f_1(t)$ so that if $|X| \geq k$ and G is (k, α)-compact, then G excludes apex-grid of order $p = f_2(t, \alpha)$ after deletion of $f_3(t)$ vertices.

Compact graphs

Let $X \subseteq V(G)$ be the terminals.

G, X is (k, α)-compact if

- X is well-linked
- There is no connected set $C \subseteq V(G)$ with $|C| \geq \alpha$ and $|N(C)| < k$.

Lemma (Follows from RS-decomposition)

If G excludes K_t, there is $k = f_1(t)$ so that if $|X| \geq k$ and G is (k, α)-compact, then G excludes apex-grid of order $p = f_2(t, \alpha)$ after deletion of $f_3(t)$ vertices.

Compact clique-minor free $\Leftrightarrow$ almost apex-minor-free

Reducing to compact graphs

Recursive scheme:

Reducing to compact graphs

Recursive scheme:

1. X not well-linked
   $\Rightarrow$ recurse by a separator

Reducing to compact graphs

Recursive scheme:

1. X not well-linked
   $\Rightarrow$ recurse by a separator
2. If X well-linked but $|X| < k$

Reducing to compact graphs

Recursive scheme:

1. X not well-linked
   $\Rightarrow$ recurse by a separator
2. If X well-linked but $|X| < k$
   $\Rightarrow$ Either recurse by a small balanced separator, or add to X a well-linked set controlling the global tangle

Reducing to compact graphs

Recursive scheme:

1. X not well-linked
   $\Rightarrow$ recurse by a separator
2. If X well-linked but $|X| < k$
   $\Rightarrow$ Either recurse by a small balanced separator, or add to X a well-linked set controlling the global tangle
   - (this step key to keep recursion-depth $\mathcal{O}(\log n)$)

Reducing to compact graphs

Recursive scheme:

1. X not well-linked
   $\Rightarrow$ recurse by a separator
2. If X well-linked but $|X| < k$
   $\Rightarrow$ Either recurse by a small balanced separator, or add to X a well-linked set controlling the global tangle
   - (this step key to keep recursion-depth $\mathcal{O}(\log n)$)
3. X well-linked and $|X| \geq k$

Reducing to compact graphs

Recursive scheme:

1. X not well-linked
   $\Rightarrow$ recurse by a separator
2. If X well-linked but $|X| < k$
   $\Rightarrow$ Either recurse by a small balanced separator, or add to X a well-linked set controlling the global tangle
   - (this step key to keep recursion-depth $\mathcal{O}(\log n)$)
3. X well-linked and $|X| \geq k$
   $\Rightarrow$ Recursive understanding to make G, X into (k, α)-compact

Recursive understanding

Now: X is well-linked and $|X| \geq k$

Want also: There is no connected set $C \subseteq V(G)$ with $|C| \geq \alpha$ and $|N(C)| < k$

,

Recursive understanding

Now: X is well-linked and $|X| \geq k$

Want also: There is no connected set $C \subseteq V(G)$ with $|C| \geq \alpha$ and $|N(C)| < k$

- While exists such set C, solve Folio recursively on $(G[N[C]], N(C))$ and replace C by a gadget of size $< \alpha/2$

,

Recursive understanding

Now: X is well-linked and $|X| \geq k$

Want also: There is no connected set $C \subseteq V(G)$ with $|C| \geq \alpha$ and $|N(C)| < k$

- While exists such set C, solve Folio recursively on $(G[N[C]], N(C))$ and replace C by a gadget of size $< \alpha/2$
- Need to do it fast! Two ingredients:

,

Recursive understanding

Now: X is well-linked and $|X| \geq k$

Want also: There is no connected set $C \subseteq V(G)$ with $|C| \geq \alpha$ and $|N(C)| < k$

- While exists such set C, solve Folio recursively on $(G[N[C]], N(C))$ and replace C by a gadget of size $< \alpha/2$
- Need to do it fast! Two ingredients:

1. Algorithm for finding a collection of non-touching such sets C containing a fraction of $1/\mathcal{O}_{k,\alpha}(\log n)$ of all vertices in such sets

,

Recursive understanding

Now: X is well-linked and $|X| \geq k$

Want also: There is no connected set $C \subseteq V(G)$ with $|C| \geq \alpha$ and $|N(C)| < k$

- While exists such set C, solve Folio recursively on $(G[N[C]], N(C))$ and replace C by a gadget of size $< \alpha/2$
- Need to do it fast! Two ingredients:

1. Algorithm for finding a collection of non-touching such sets C containing a fraction of $1/\mathcal{O}_{k,\alpha}(\log n)$ of all vertices in such sets
    - After color-coding, about computing a (simpler) variant of Gomory-Hu tree, randomized version almost-directly from [Pettie, Saranurak, and Yin, 2022],

Recursive understanding

Now: X is well-linked and $|X| \geq k$

Want also: There is no connected set $C \subseteq V(G)$ with $|C| \geq \alpha$ and $|N(C)| < k$

- While exists such set C, solve Folio recursively on $(G[N[C]], N(C))$ and replace C by a gadget of size $< \alpha/2$
- Need to do it fast! Two ingredients:

1. Algorithm for finding a collection of non-touching such sets C containing a fraction of $1/\mathcal{O}_{k,\alpha}(\log n)$ of all vertices in such sets
   - After color-coding, about computing a (simpler) variant of Gomory-Hu tree, randomized version almost-directly from [Pettie, Saranurak, and Yin, 2022],deterministic by combining: Isolating cuts [Li & Panigrahi, 2020], almost-linear (deterministic) max-flow [van den Brand, Chen, Kyng, Liu, Peng, Probst Gutenberg, Sachdeva & Sidford, 2023], and mimicking networks of [Saranurak & Yingchareonthawornchai, 2022]
2. Construction of gadgets that not only preserve folio, but also don't create "new" such sets C

Recursive understanding

Now: X is well-linked and $|X| \geq k$

Want also: There is no connected set $C \subseteq V(G)$ with $|C| \geq \alpha$ and $|N(C)| < k$

- While exists such set C, solve Folio recursively on $(G[N[C]], N(C))$ and replace C by a gadget of size $< \alpha/2$
- Need to do it fast! Two ingredients:

1. Algorithm for finding a collection of non-touching such sets C containing a fraction of $1/\mathcal{O}_{k,\alpha}(\log n)$ of all vertices in such sets
   - After color-coding, about computing a (simpler) variant of Gomory-Hu tree, randomized version almost-directly from [Pettie, Saranurak, and Yin, 2022],deterministic by combining: Isolating cuts [Li & Panigrahi, 2020], almost-linear (deterministic) max-flow [van den Brand, Chen, Kyng, Liu, Peng, Probst Gutenberg, Sachdeva & Sidford, 2023], and mimicking networks of [Saranurak & Yingchareonthawornchai, 2022]
2. Construction of gadgets that not only preserve folio, but also don't create "new" such sets C
   - Challenge: Size of gadget should depend on k, not α

From general graphs to minor-free

From general graphs to minor-free

Lemma (Robertson & Seymour, Graph Minors 13)

If G, X is (k, α)-compact and G contains a $K_{3 \cdot (k+\delta) \cdot \alpha}$-minor, then the (X, δ)-folio of G is generic.

From general graphs to minor-free

Lemma (Robertson & Seymour, Graph Minors 13)

If G, X is (k, α)-compact and G contains a $K_{3 \cdot (k+\delta) \cdot \alpha}$-minor, then the (X, δ)-folio of G is generic.

Same recursive scheme to make G, X into (k, α)-compact...

From general graphs to minor-free

Lemma (Robertson & Seymour, Graph Minors 13)

If G, X is (k, α)-compact and G contains a $K_{3\cdot(k+\delta)\cdot\alpha}$-minor, then the (X, δ)-folio of G is generic.

Same recursive scheme to make G, X into (k, α)-compact...

Need actually a new proof of the lemma to make it more algorithmic

Conclusion

- $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for rooted minor containment

Conclusion

- $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for rooted minor containment
- "Simple" algorithm for apex-minor-free graphs with dynamic treewidth

Conclusion

- $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for rooted minor containment
- “Simple” algorithm for apex-minor-free graphs with dynamic treewidth
- Reduction from general graphs to apex-minor-free with fast recursive understanding

Conclusion

- $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for rooted minor containment
- “Simple” algorithm for apex-minor-free graphs with dynamic treewidth
- Reduction from general graphs to apex-minor-free with fast recursive understanding

Future work:

Conclusion

- $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for rooted minor containment
- “Simple” algorithm for apex-minor-free graphs with dynamic treewidth
- Reduction from general graphs to apex-minor-free with fast recursive understanding

Future work:

- Extensions to compute the Robertson-Seymour decomposition, topological minor containment

Conclusion

- $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for rooted minor containment
- "Simple" algorithm for apex-minor-free graphs with dynamic treewidth
- Reduction from general graphs to apex-minor-free with fast recursive understanding

Future work:

- Extensions to compute the Robertson-Seymour decomposition, topological minor containment
- Optimization to $\mathcal{O}_{H,|X|}(m \text{ polylog } n)$?

Conclusion

- $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for rooted minor containment
- "Simple" algorithm for apex-minor-free graphs with dynamic treewidth
- Reduction from general graphs to apex-minor-free with fast recursive understanding

Future work:

- Extensions to compute the Robertson-Seymour decomposition, topological minor containment
- Optimization to $\mathcal{O}_{H,|X|}(m \text{ polylog } n)$?
  - Important problem: Optimization of dynamic treewidth to $\mathcal{O}_k(\text{polylog } n)$?

Conclusion

- $\mathcal{O}_{H,|X|}(m^{1+o(1)})$ time algorithm for rooted minor containment
- "Simple" algorithm for apex-minor-free graphs with dynamic treewidth
- Reduction from general graphs to apex-minor-free with fast recursive understanding

Future work:

- Extensions to compute the Robertson-Seymour decomposition, topological minor containment
- Optimization to $\mathcal{O}_{H,|X|}(m \text{ polylog } n)$?
  - Important problem: Optimization of dynamic treewidth to $\mathcal{O}_k(\text{polylog } n)$?

Thank you!