CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Hitting Meets Packing: How Hard Can It Be?

Jacob Focke[1] Fabian Frei[1] Shaohua Li[1] Dániel Marx[1]

Philipp Schepper[1] Roohani Sharma[2] Karol Węgrzycki[3,4]

[1] CISPA | [2] University of Bergen

[3] MPI Informatics, SIC | [4] Saarland University

ESA 2024 – September 2, 2024

A Set of Unrelated Problems

Triangle Partition

Vertex Cover

Minimum s-t-cut

Maximum Matching

Feedback Vertex Set

Cycle Cover

Odd Cycle Transversal

A Set of Unrelated Problems

Partition the graph
into triangles
Triangle Partition

Select vertices
to cover all edges
Vertex Cover

Minimum s-t-cut *Separate two vertices by the optimal vertex removals*

Select the maximum number of disjoint edges **Maximum Matching**

Feedback Vertex Set *Remove vertices to make the graph a forest*

Cycle Cover
Cover (all) vertices
using only cycles

Odd Cycle Transversal
Delete vertices to make
the graph bipartite

A Set of Unrelated Problems

Partition the graph into triangles
Triangle Partition

Select vertices to cover all edges
Vertex Cover

Odd Cycle Packing

Minimum s-t-cut *Separate two vertices by the optimal vertex removals*

Select the maximum number of disjoint edges **Maximum Matching**

Chordal Deletion

Feedback Vertex Set *Remove vertices to make the graph a forest*

Tree Cover

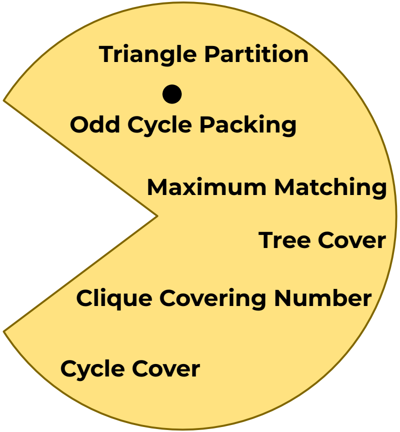
H-Hitting

Clique Covering Number

Odd Cycle Transversal
Delete vertices to make the graph bipartite

Cycle Cover
Cover (all) vertices using only cycles

A Set of Unrelated Problems



Vertex Cover

Minimum s-t-cut

Chordal Deletion

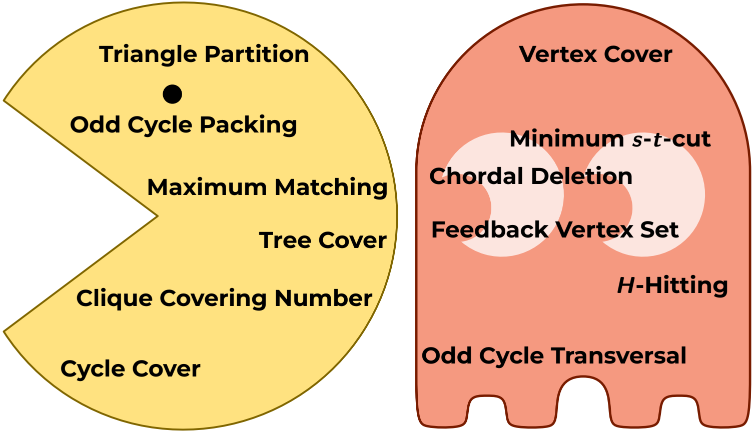
Feedback Vertex Set

H-Hitting

Odd Cycle Transversal

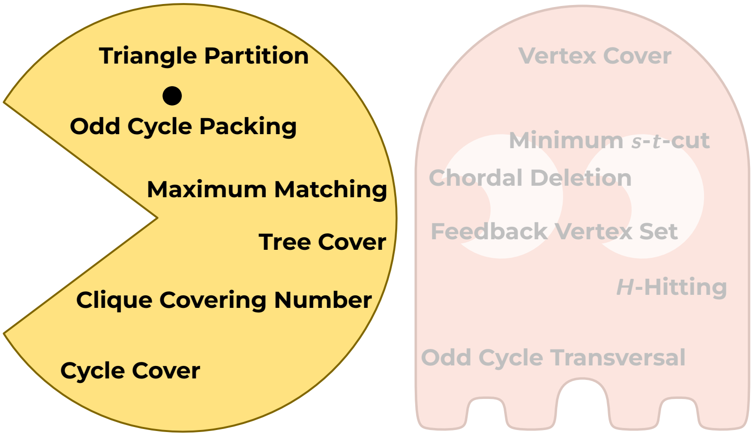
Packing Problems

A Set of Unrelated Problems



Packing Problems **Hitting Problems**

Packing Problems

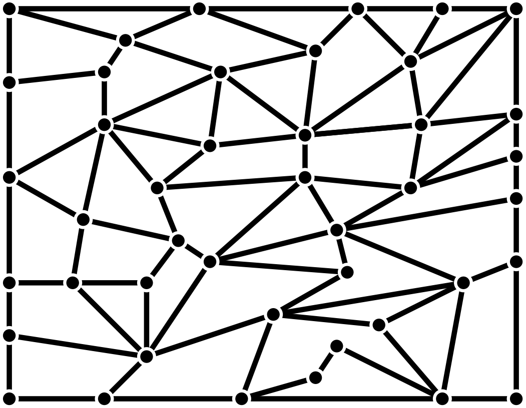


Packing Problems

H*-Packing for a fixed graph *H

Input: A graph G, an integer ℓ.

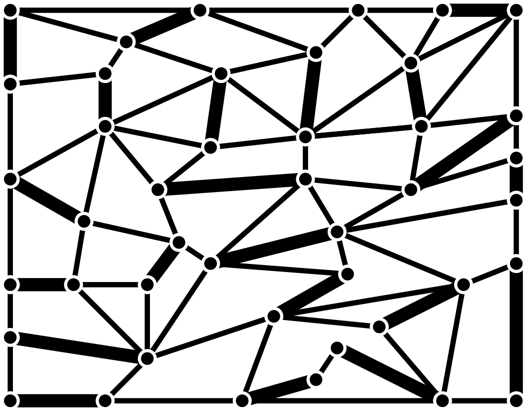
Task: Pack ℓ copies of H in a vertex-disjoint way in G.



Packing Problems

H-PACKING for a fixed graph *H*
Input: A graph G, an integer ℓ.
Task: Pack ℓ copies of H in a vertex-disjoint way in G.

- Maximum Matching:
  $H = K_2 \rightsquigarrow$ "Edge Packing"



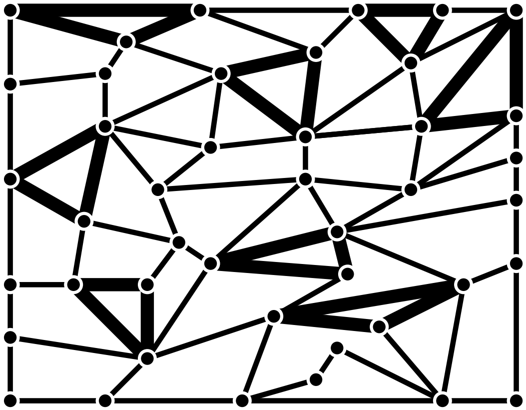
Packing Problems

H-PACKING for a fixed graph H

Input: A graph G, an integer ℓ.

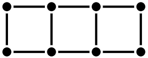
Task: Pack ℓ copies of H in a vertex-disjoint way in G.

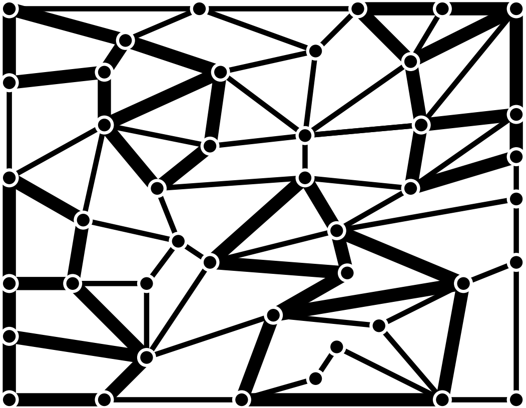
- Maximum Matching:
  $H = K_2 \rightsquigarrow$ "Edge Packing"
- Triangle Covering:
  $H = C_3 \rightsquigarrow$ "Triangle Packing"



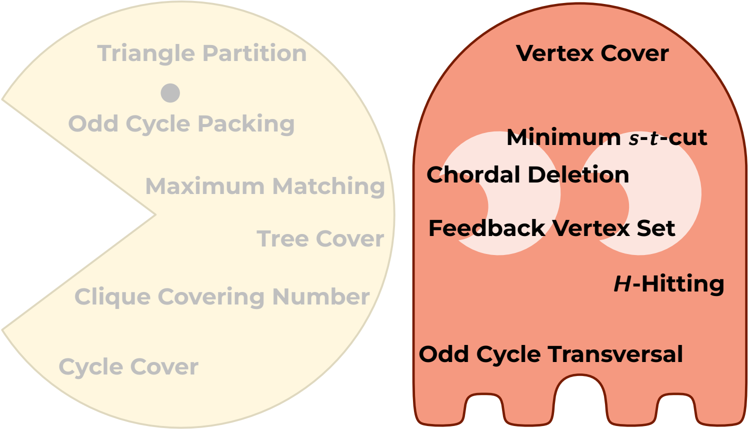
Packing Problems

H-Packing for a fixed graph *H*
Input: A graph G, an integer ℓ.
Task: Pack ℓ copies of H in a vertex-disjoint way in G.

- Maximum Matching:
  $H = K_2 \rightsquigarrow$ "Edge Packing"
- Triangle Covering:
  $H = C_3 \rightsquigarrow$ "Triangle Packing"
- H-Packing where
  $H =$ 



Hitting Problems

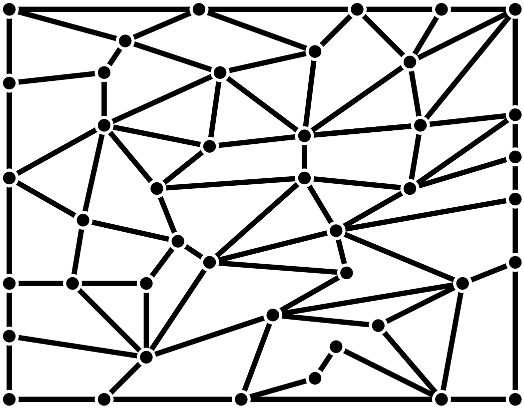


Hitting Problems

H-HITTING for a fixed graph H

Input: A graph G, an integer k.

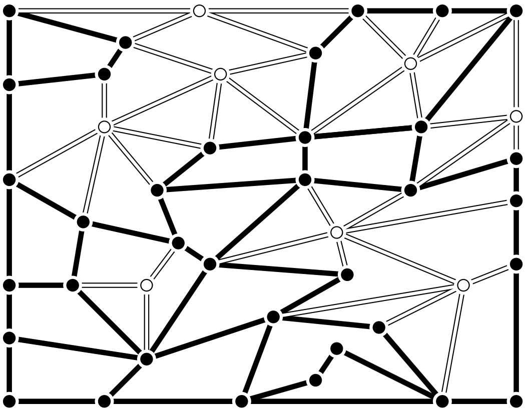
Task: Delete k vertices such that no copy of H remains in G.



Hitting Problems

H*-Hitting for a fixed graph *H
Input: A graph G, an integer k.
Task: Delete k vertices such that no copy of H remains in G.

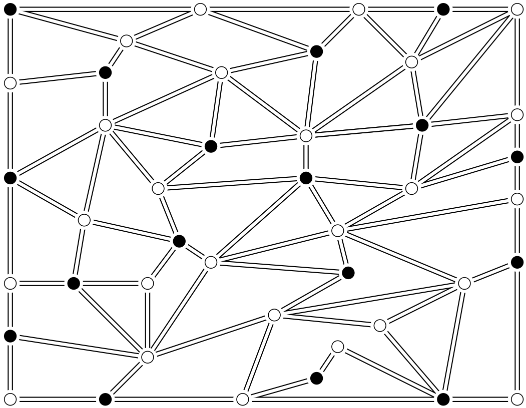
- Triangle Hitting:
  $H = C_3 =$ a triangle



Hitting Problems

H-HITTING for a fixed graph H
Input: A graph G, an integer k.
Task: Delete k vertices such that no copy of H remains in G.

- Triangle Hitting:
  $H = C_3 =$ a triangle
- Vertex Cover:
  $H = K_2 \rightsquigarrow$ “Edge Hitting”



Deleted Vertices = Vertex Cover!

Hitting Problems

H*-HITTING for a fixed graph *H

Input: A graph G, an integer k.

Task: Delete k vertices such that no copy of H remains in G.

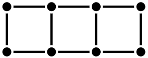
- Triangle Hitting:

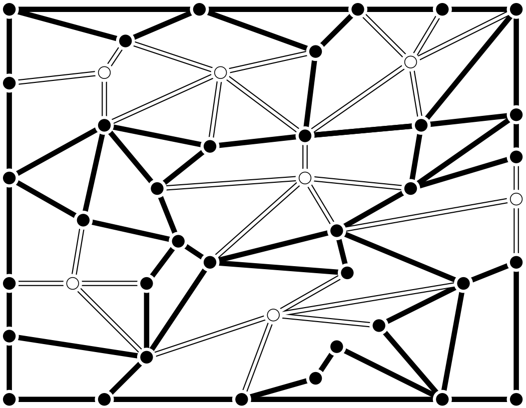
  $H = C_3 =$ a triangle

- Vertex Cover:

  $H = K_2 \rightsquigarrow$ “Edge Hitting”

- H-HITTING where

  $H =$ 

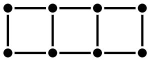


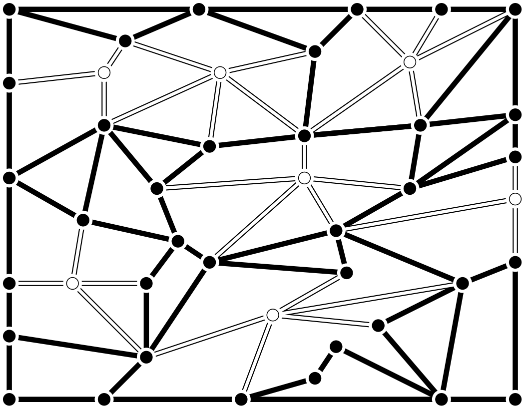
Hitting Problems

H*-HITTING for a fixed graph *H

Input: A graph G, an integer k.

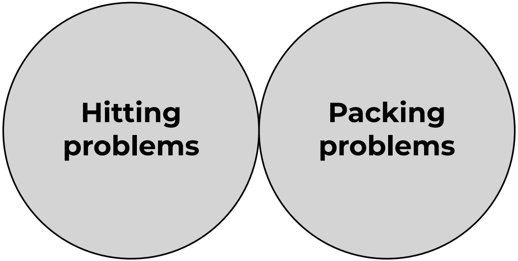
Task: Delete k vertices such that no copy of H remains in G.

- Triangle Hitting:
  $H = C_3 =$ a triangle
- Vertex Cover:
  $H = K_2 \rightsquigarrow$ "Edge Hitting"
- H-HITTING where
  $H =$ 

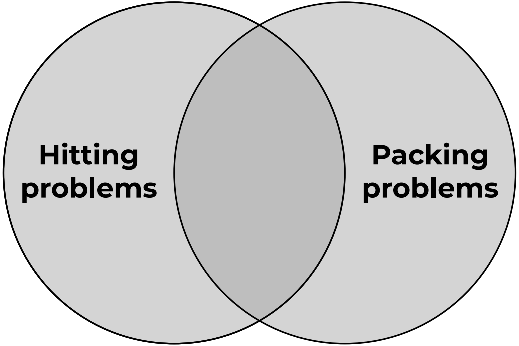


also known as covering or transversal

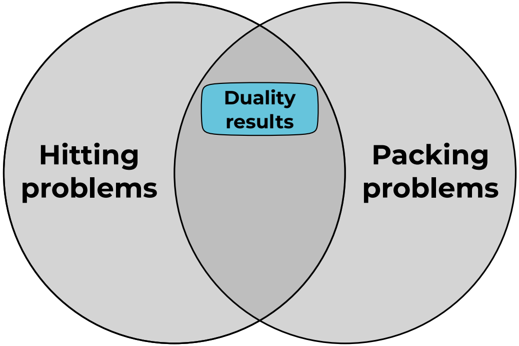
Hitting Meets Packing



Hitting Meets Packing



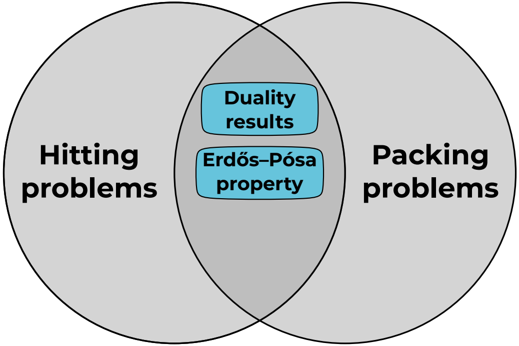
Hitting Meets Packing



Duality Results:

If we destroy all copies of H by deleting k vertices, then we can pack at most k copies of H.

Hitting Meets Packing

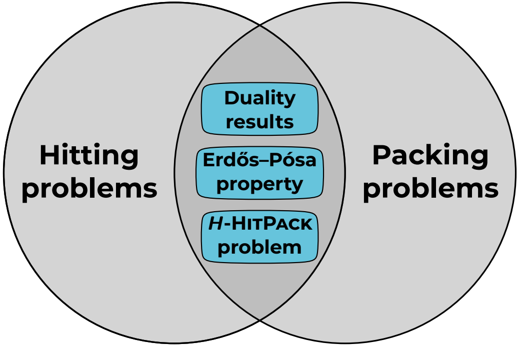


Erdős-Pósa property:

If we can pack ℓ cycles,

then we can hit all cycles by removing $k = \mathcal{O}(\ell \log \ell)$ vertices.

Hitting Meets Packing



***H*-HitPack:**

A generalization of Hitting and Packing

that makes both problems “more robust” ($\rightsquigarrow$ notion of stability).

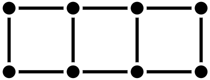
H-HitPack

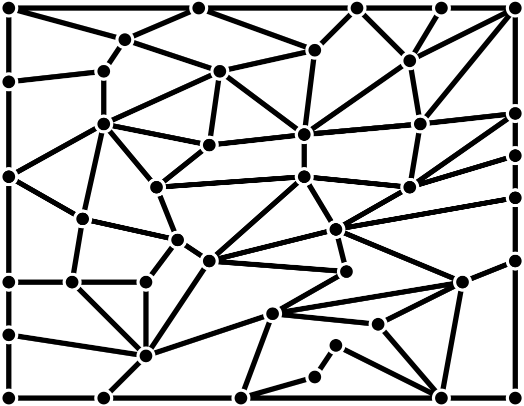
H-HitPack for a fixed graph H

Input: A graph G, integers k and ℓ.

Task:

Delete k vertices such that we cannot pack $\ell + 1$ copies of H.

Graph H = 



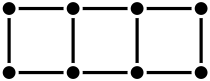
H-HitPack

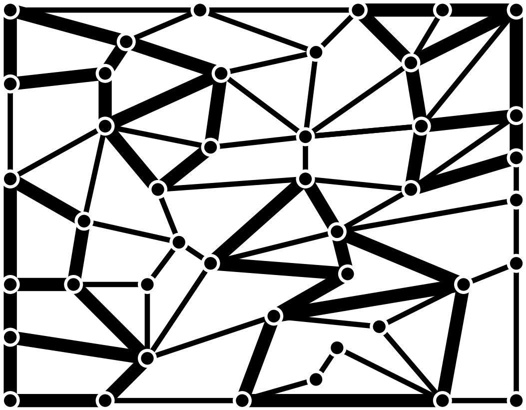
H-HitPack for a fixed graph H

Input: A graph G, integers k and ℓ.

Task:

Delete k vertices such that we cannot pack $\ell + 1$ copies of H.

Graph $H =$ 



H-Packing

$(k = 0, \ell = 4)$

H-HitPack

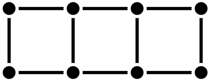
H-HitPack for a fixed graph H

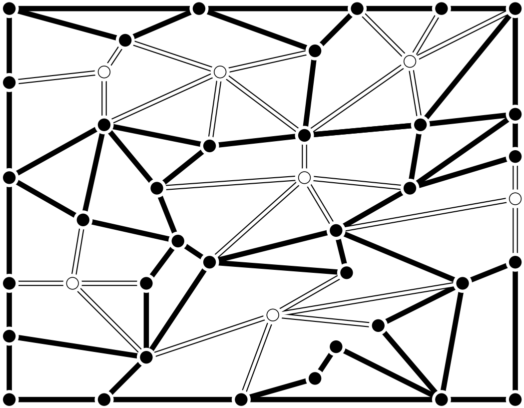
Input: A graph G, integers k and ℓ.

Task:

Delete k vertices such that we cannot pack $\ell + 1$ copies of H.

Graph H =





H-Hitting

$(k = 7, \ell = 0)$

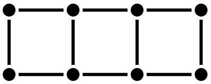
H-HITPACK

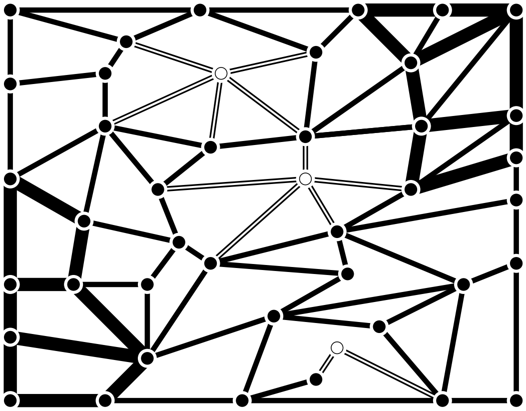
H-HITPACK for a fixed graph H

Input: A graph G, integers k and ℓ.

Task:

Delete k vertices such that we cannot pack $\ell + 1$ copies of H.

Graph $H =$ 



H-HITPACK

$(k = 3, \ell = 2)$

Our Contribution

Fix a connected graph H:

Theorem (Algorithmic Results)

H-HitPack can be solved in time $2^{2^{\mathcal{O}(\text{tw} \log \text{tw})}} \cdot |G|^{\mathcal{O}(1)}$.

Our Contribution

Fix a connected graph H:

Theorem (Algorithmic Results)

H-HITPACK can be solved in time $2^{2^{\mathcal{O}(\mathrm{tw}\log \mathrm{tw})}} \cdot |G|^{\mathcal{O}(1)}$.

Theorem (Hardness Results)

H-HITPACK is complete for $\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}} = \mathrm{NP}^{\mathrm{coNP}}$.

No $2^{2^{o(\mathrm{tw})}} \cdot |G|^{\mathcal{O}(1)}$-time algorithm unless ETH fails.

Our Contribution

Fix a connected graph H:

Theorem (Algorithmic Results)

H-HitPack can be solved in time $2^{2^{\mathcal{O}(\mathrm{tw}\log\mathrm{tw})}} \cdot |G|^{\mathcal{O}(1)}$.

Theorem (Hardness Results)

H-HitPack is complete for $\Sigma_2^P = \mathrm{NP}^{\mathrm{NP}} = \mathrm{NP}^{\mathrm{coNP}}$.

No $2^{2^{o(\mathrm{tw})}} \cdot |G|^{\mathcal{O}(1)}$-time algorithm unless ETH fails.

Summary: The problem is *double*-exponential in treewidth!

Although the base problems are much simpler.

The Source of Hardness

H-HITPACK for a fixed graph H

Input: A graph G, integers k and ℓ.

Task:

Delete k vertices such that we cannot pack $\ell + 1$ copies of H.

The Source of Hardness

H-HitPack for a fixed graph H

Input: A graph G, integers k and ℓ.

Task:

Is there a set S of at most k vertices such that every H-packing in $G - S$ contains at most ℓ copies of H.

The Source of Hardness

H-HITPACK for a fixed graph H

Input: A graph G, integers k and ℓ.

Task:

Is there a set S of at most k vertices such that every H-packing in $G - S$ contains at most ℓ copies of H.

Once we "guessed" the set S, we need to argue about *all possible* H-packings for the remaining graph!

The Source of Hardness

H*-HitPack for a fixed graph *H		
Input: A graph G, integers k and ℓ.		
Task:		
Is there a set S of at most k vertices	$\rightsquigarrow$	$\exists$
such that *every* H-packing in $G - S$	$\rightsquigarrow$	$\forall$
contains at most ℓ copies of H.		

Once we "guessed" the set S, we need to argue about *all possible* H-packings for the remaining graph!

Hints at the Σ_2^P-hardness ←

The Source of Hardness

H-HITPACK for a fixed graph H

Input: A graph G, integers k and ℓ.

Task:

Is there a set S of at most k vertices $\rightsquigarrow$ $\exists$

such that every H-packing in $G - S$ $\rightsquigarrow$ $\forall$

contains at most ℓ copies of H.

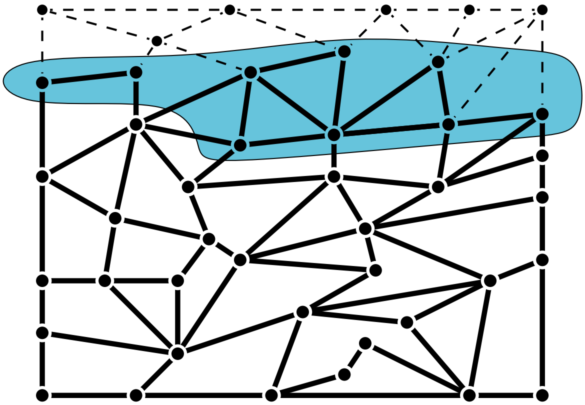
Once we "guessed" the set S, we need to argue about *all possible* H-packings for the remaining graph!

Hints at the Σ_2^P-hardness

Main Question: How does this affect the algorithm?

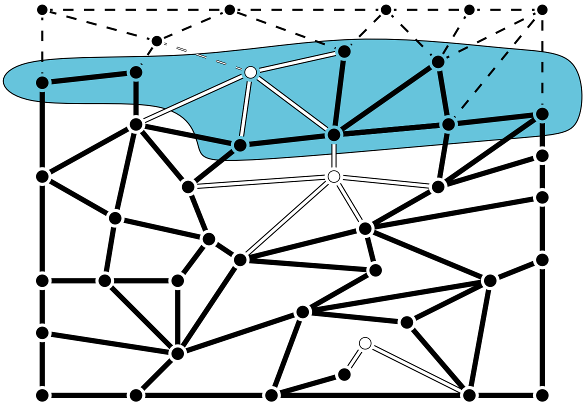
Algorithmic Idea

Fix a bag of the decomposition (i.e., a separator):



Algorithmic Idea

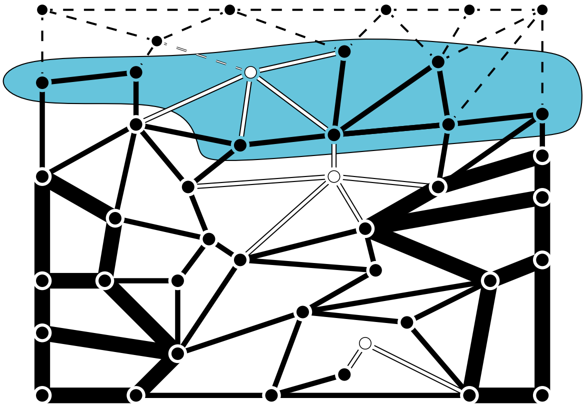
Fix a bag of the decomposition (i.e., a separator):



1 Guess the deleted vertices.

Algorithmic Idea

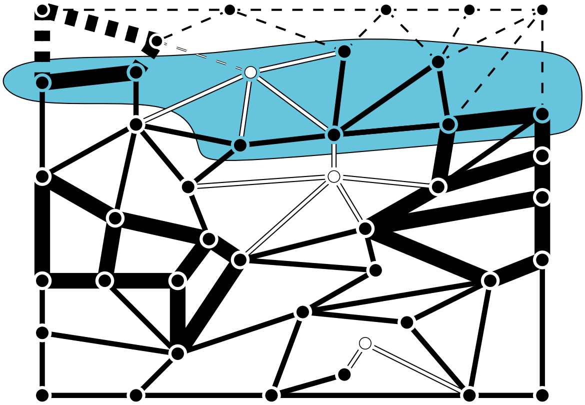
Fix a bag of the decomposition (i.e., a separator):



1. Guess the deleted vertices.
2. For each partial H-packing, store the size of the packing.

Algorithmic Idea

Fix a bag of the decomposition (i.e., a separator):

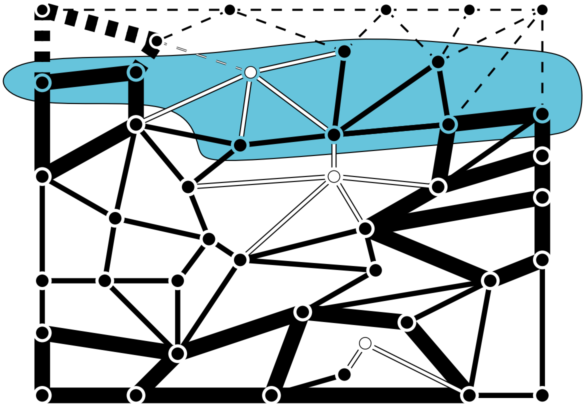


1 Guess the deleted vertices.

2 For each partial H-packing, store the size of the packing.

Algorithmic Idea

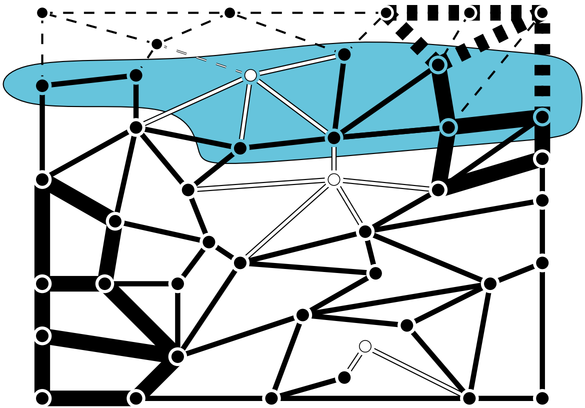
Fix a bag of the decomposition (i.e., a separator):



1. Guess the deleted vertices.
2. For each partial H-packing, store the size of the packing.

Algorithmic Idea

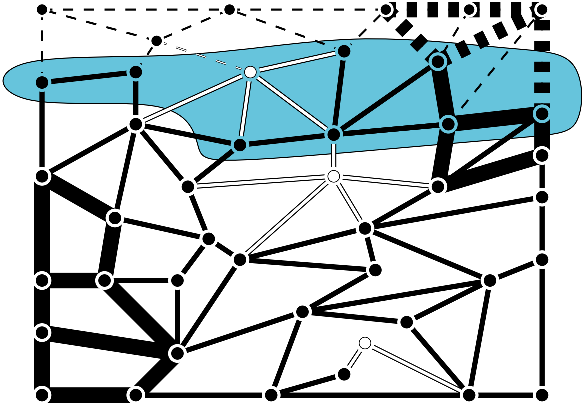
Fix a bag of the decomposition (i.e., a separator):



1. Guess the deleted vertices.
2. For each partial H-packing, store the size of the packing.

Algorithmic Idea

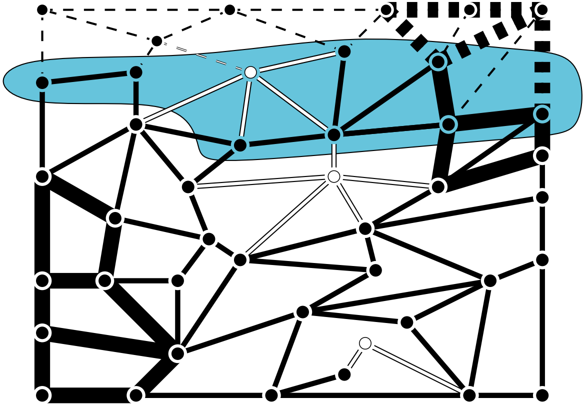
Fix a bag of the decomposition (i.e., a separator):



1. Guess the deleted vertices.
2. For each partial H-packing, store the size of the packing.
3. If there is a packing with $> \ell$ copies of H, then discard the current guess.

Algorithmic Idea

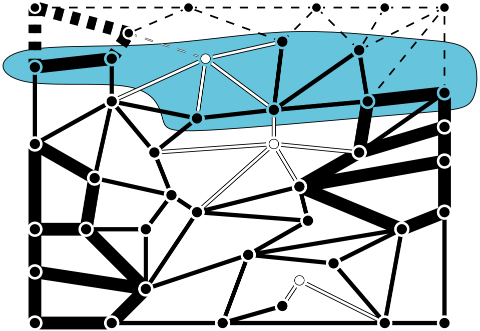
Fix a bag of the decomposition (i.e., a separator):

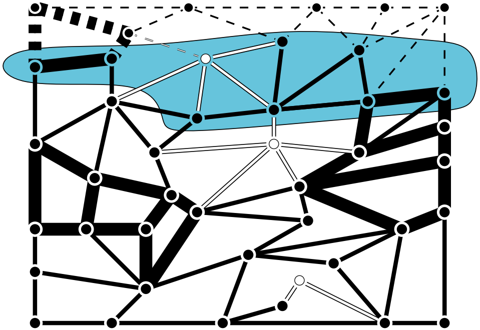


1. Guess the deleted vertices.
2. For each partial H-packing, store the size of the packing.
3. If there is a packing with $> \ell$ copies of H, then discard the current guess.

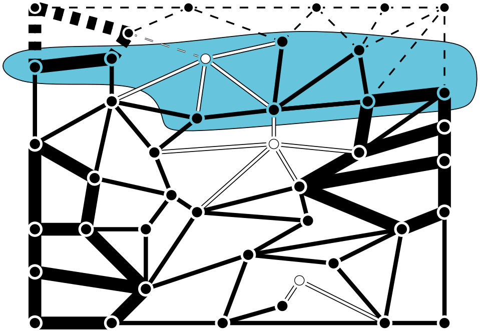
$\rightsquigarrow$ $|H|^{|G|}$ packings to consider!

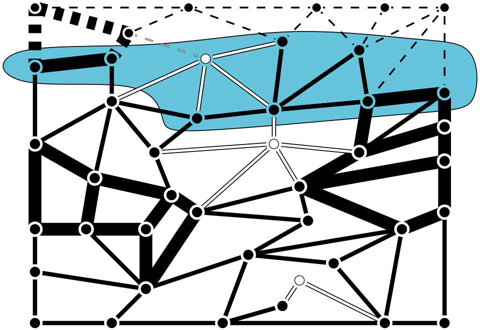
Algorithmic Idea: Improving the Running Time





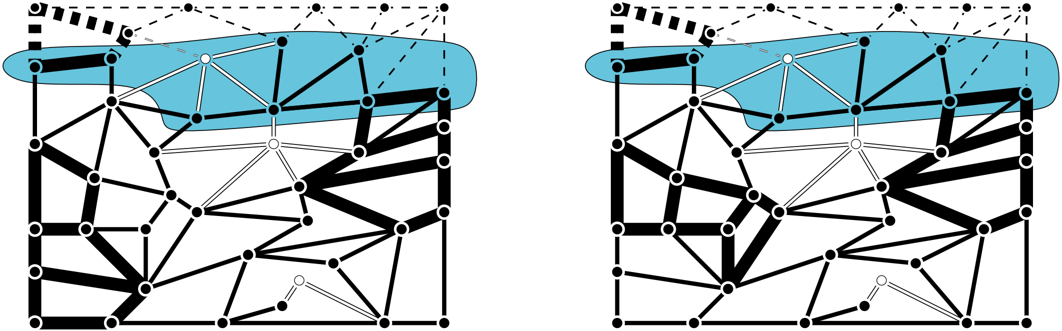
Algorithmic Idea: Improving the Running Time





1. Only the intersection with the bag matters!

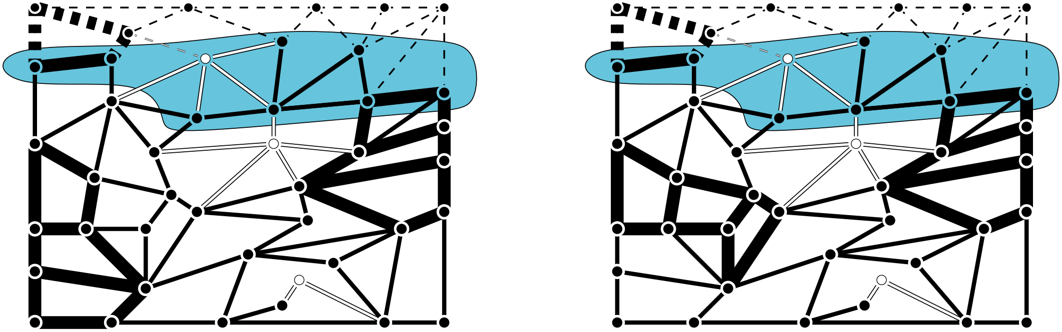
Algorithmic Idea: Improving the Running Time



1. Only the intersection with the bag matters!

   ⇝ Need a partition of the bag and mappings from there to *H*.

Algorithmic Idea: Improving the Running Time



- **1** Only the intersection with the bag matters!
  - $\rightsquigarrow$ Need a partition of the bag and mappings from there to H.
  - $\rightsquigarrow$ $\mathrm{tw}^{\mathrm{tw}} \cdot |H|^{\mathrm{tw}}$ possible types of packings

Algorithmic Idea: Improving the Running Time

1. Only the intersection with the bag matters!

   $\rightsquigarrow$ Need a partition of the bag and mappings from there to H.

   $\rightsquigarrow$ $\mathrm{tw}^{\mathrm{tw}} \cdot |H|^{\mathrm{tw}}$ possible types of packings

Algorithmic Idea: Improving the Running Time

1. Only the intersection with the bag matters!

   $\rightsquigarrow$ Need a partition of the bag and mappings from there to H.

   $\rightsquigarrow$ $\text{tw}^{\text{tw}} \cdot |H|^{\text{tw}}$ possible types of packings

2. For each type, the packing can have size $\in \{0, 1, 2, \ldots, \ell\}$

   (if it is larger, we removed the "wrong" vertices)

Algorithmic Idea: Improving the Running Time

1. Only the intersection with the bag matters!

   $\rightsquigarrow$ Need a partition of the bag and mappings from there to H.

   $\rightsquigarrow$ $\text{tw}^{\text{tw}} \cdot |H|^{\text{tw}}$ possible types of packings

2. For each type, the packing can have size $\in \{0, 1, 2, \ldots, \ell\}$

   (if it is larger, we removed the "wrong" vertices)

$\rightsquigarrow$ $\ell^{\text{tw}^{\mathcal{O}(\text{tw})}}$ states for each bag

Algorithmic Idea: Improving the Running Time

1. Only the intersection with the bag matters!

   $\rightsquigarrow$ Need a partition of the bag and mappings from there to H.

   $\rightsquigarrow$ $\mathrm{tw}^{\mathrm{tw}} \cdot |H|^{\mathrm{tw}}$ possible types of packings

2. For each type, the packing can have size $\in \{0, 1, 2, \ldots, \ell\}$
   (if it is larger, we removed the “wrong” vertices)

$\rightsquigarrow$ $\ell^{\mathrm{tw}^{\mathcal{O}(\mathrm{tw})}}$ states for each bag

3. Discard very small packings

$\rightsquigarrow$ only $\mathcal{O}(\mathrm{tw} \cdot |H|)^{\mathrm{tw}^{\mathcal{O}(\mathrm{tw})}}$ states for each bag

Algorithmic Idea: Improving the Running Time

1. Only the intersection with the bag matters!

   $\rightsquigarrow$ Need a partition of the bag and mappings from there to H.

   $\rightsquigarrow$ $\text{tw}^{\text{tw}} \cdot |H|^{\text{tw}}$ possible types of packings

2. For each type, the packing can have size $\in \{0, 1, 2, \ldots, \ell\}$ (if it is larger, we removed the "wrong" vertices)

$\rightsquigarrow$ $\ell^{\text{tw}^{\mathcal{O}(\text{tw})}}$ states for each bag

3. Discard very small packings

$\rightsquigarrow$ only $\mathcal{O}(\text{tw} \cdot |H|)^{\text{tw}^{\mathcal{O}(\text{tw})}}$ states for each bag

Theorem

For every connected graph H, H-HitPack can be solved in time $2^{2^{\mathcal{O}(\text{tw} \log \text{tw})}}$.

Our Contribution

Graph H	Upper Bound	Lower Bound under ETH	Completeness
Connected	$2^{2^{\mathcal{O}(\mathrm{tw}\log\mathrm{tw})}}$	no $2^{2^{o(\mathrm{tw})}}$	$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$

Our Contribution

Graph H	Upper Bound	Lower Bound under ETH	Completeness
Square	$2^{2^{\mathcal{O}(\mathrm{tw}\log \mathrm{tw})}}$	no $2^{2^{o(\mathrm{tw}\log \mathrm{tw})}}$	$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$
Connected		no $2^{2^{o(\mathrm{tw})}}$	

Our Contribution

Graph H	Upper Bound	Lower Bound under ETH	Completeness
Square		no $2^{2^{o(\text{tw} \log \text{tw})}}$	
Connected	$2^{2^{\mathcal{O}(\text{tw} \log \text{tw})}}$	no $2^{2^{o(\text{tw})}}$	$\Sigma_2^{\text{P}} = \text{NP}^{\text{NP}}$
q-Clique	$2^{2^{\mathcal{O}(\text{tw})}}$		

Our Contribution

<table>
<tr><th>Graph H</th><th>Upper Bound</th><th>Lower Bound under ETH</th><th>Completeness</th></tr>
<tr><td>Square</td><td rowspan="2">$2^{2^{\mathcal{O}(\text{tw} \log \text{tw})}}$</td><td>no $2^{2^{o(\text{tw} \log \text{tw})}}$</td><td rowspan="3">$\Sigma_2^{\text{P}} = \text{NP}^{\text{NP}}$</td></tr>
<tr><td>Connected ≥ 3 vtcs.</td><td rowspan="2">no $2^{2^{o(\text{tw})}}$</td></tr>
<tr><td>q-Clique</td><td>$2^{2^{\mathcal{O}(\text{tw})}}$</td></tr>
<tr><td>Edge</td><td></td><td></td><td></td></tr>
</table>

Our Contribution

Graph H	Upper Bound	Lower Bound under ETH	Completeness
Square	$2^{2^{\mathcal{O}(\mathrm{tw}\log \mathrm{tw})}}$	no $2^{2^{o(\mathrm{tw}\log \mathrm{tw})}}$	$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$
Connected ≥ 3 vtcs.	$2^{2^{\mathcal{O}(\mathrm{tw}\log \mathrm{tw})}}$	no $2^{2^{o(\mathrm{tw})}}$	$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$
q-Clique	$2^{2^{\mathcal{O}(\mathrm{tw})}}$	no $2^{2^{o(\mathrm{tw})}}$	$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$
Edge	$2^{\mathrm{poly}(\mathrm{tw})}$	no $2^{o(\mathrm{tw})}$ †	NP †

† Previously known

Our Contribution

Graph H	Upper Bound	Lower Bound under ETH	Completeness
Square	$2^{2^{\mathcal{O}(\text{tw} \log \text{tw})}}$	no $2^{2^{o(\text{tw} \log \text{tw})}}$	$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$
Connected ≥ 3 vtcs.	$2^{2^{\mathcal{O}(\text{tw} \log \text{tw})}}$	no $2^{2^{o(\text{tw})}}$	$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$
q-Clique	$\mathbf{2^{2^{\mathcal{O}(\text{tw})}}}$	no $2^{2^{o(\text{tw})}}$	$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$
Edge	$\mathbf{2^{\text{poly}(\text{tw})}}$	no $2^{o(\text{tw})}$ †	NP †

No changes in the treewidth-DP needed!

† Previously known

The Special Case of EDGE-HITPACK

Covers Maximum Matching (Packing) and Vertex Cover (Hitting)

Standard DP: "top-down approach" for the computation

$\leadsto$ DP considers all theoretically possible states

The Special Case of Edge-HitPack

Covers Maximum Matching (Packing) and Vertex Cover (Hitting)

Standard DP: “top-down approach” for the computation

$\rightsquigarrow$ DP considers all theoretically possible states

Our DP: “bottom-up approach”

$\rightsquigarrow$ DP considers only those states that actually exist

The Special Case of EDGE-HITPACK

Covers Maximum Matching (Packing) and Vertex Cover (Hitting)

Standard DP: “top-down approach” for the computation

$\rightsquigarrow$ DP considers all theoretically possible states

Our DP: “bottom-up approach”

$\rightsquigarrow$ DP considers only those states that actually exist

We give a *non-constructive* proof using (delta-)matroids to bound the number of states.

The Special Case of EDGE-HITPACK

Covers Maximum Matching (Packing) and Vertex Cover (Hitting)

Standard DP: “top-down approach” for the computation

$\rightsquigarrow$ DP considers all theoretically possible states

Our DP: “bottom-up approach”

$\rightsquigarrow$ DP considers only those states that actually exist

We give a *non-constructive* proof using (delta-)matroids to bound the number of states.

$\rightsquigarrow$ Significantly faster algorithm *without any changes*!

Generalizing H-HitPack Further

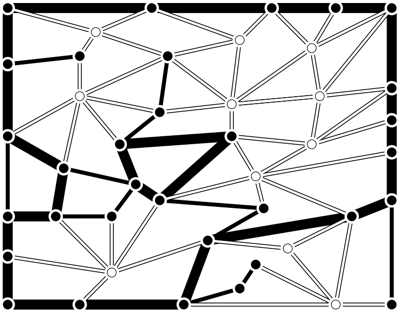
Instead of a fixed graph H, we allow arbitrary graph objects $\mathcal{X}$

Generalizing H-HitPack Further

Instead of a fixed graph H, we allow arbitrary graph objects $\mathcal{X}$

$\mathcal{X}$ = "class of all cycles":

generalizes Feedback Vertex Set and Cycle Packing



Generalizing H-HitPack Further

Instead of a fixed graph H, we allow arbitrary graph objects $\mathcal{X}$

$\mathcal{X} =$ "class of all cycles":
generalizes Feedback Vertex Set and Cycle Packing

Why treewidth might not always be the best parameter:

- Assume we deleted k vertices

Generalizing H-HitPack Further

Instead of a fixed graph H, we allow arbitrary graph objects $\mathcal{X}$

$\mathcal{X} =$ "class of all cycles":

generalizes Feedback Vertex Set and Cycle Packing

Why treewidth might not always be the best parameter:

- Assume we deleted k vertices
- Erdős–Pósa: Can destroy ℓ cycles by deleting $\mathcal{O}(\ell \log \ell)$ vertices

Generalizing H-HitPack Further

Instead of a fixed graph H, we allow arbitrary graph objects $\mathcal{X}$

$\mathcal{X} =$ “class of all cycles”:
generalizes Feedback Vertex Set and Cycle Packing

Why treewidth might not always be the best parameter:

- Assume we deleted k vertices
- Erdős–Pósa: Can destroy ℓ cycles by deleting $\mathcal{O}(\ell \log \ell)$ vertices

$\rightsquigarrow$ Treewidth is $\mathcal{O}(k + \ell \log \ell)$

$\rightsquigarrow$ Algorithm with running time $2^{2^{(k+\ell)\,\mathrm{poly}\log(k+\ell)}}$

Generalizing H-HitPack Further

Instead of a fixed graph H, we allow arbitrary graph objects $\mathcal{X}$

$\mathcal{X} =$ "class of all cycles":

generalizes Feedback Vertex Set and Cycle Packing

Why treewidth might not always be the best parameter:

- Assume we deleted k vertices
- Erdős–Pósa: Can destroy ℓ cycles by deleting $\mathcal{O}(\ell \log \ell)$ vertices

$\rightsquigarrow$ Treewidth is $\mathcal{O}(k + \ell \log \ell)$

$\rightsquigarrow$ Algorithm with running time $2^{2^{(k+\ell)\text{ poly}\log(k+\ell)}}$

Theorem

Cycle-HitPack can be solved in time $2^{\text{poly}(k+\ell)}$.

Our Contribution

Graph H	Upper Bounds		LB under ETH	Completeness
Square			no $2^{2^{o(\mathrm{tw}\log\mathrm{tw})}}$	
Conn 3+ vtx	$2^{\mathcal{O}((k+\ell)\log(k+\ell))}$	$2^{2^{\mathcal{O}(\mathrm{tw}\log\mathrm{tw})}}$	no $2^{2^{o(\mathrm{tw})}}$	$\Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}}$
q-Clique		$2^{2^{\mathcal{O}(\mathrm{tw})}}$		
Edge	$3^{k+\ell}$	$2^{\mathrm{poly}(\mathrm{tw})}$	no $2^{o(\mathrm{tw})}$	NP
Class of all cycles	$2^{\mathrm{poly}(k+\ell)}$	$2^{2^{\mathcal{O}(\mathrm{tw}\log\mathrm{tw})}}$	no $2^{2^{o(\mathrm{tw}\log\mathrm{tw})}}$	Σ_2^{P}

Conclusion

$\mathcal{X}$-HitPack generalizes $\mathcal{X}$-Hitting and $\mathcal{X}$-Packing

- Significantly harder than base problems
- New algorithmic ideas are needed
- Some cases still allow for faster algorithms

Conclusion

$\mathcal{X}$-HitPack generalizes $\mathcal{X}$-Hitting and $\mathcal{X}$-Packing

- Significantly harder than base problems
- New algorithmic ideas are needed
- Some cases still allow for faster algorithms

Open questions:

- What about induced subgraphs or directed versions?
- Approximation results by relaxing the condition on k and ℓ?
- Is there a non-trivial relation to Erdős–Pósa for other graph classes?

Conclusion

$\mathcal{X}$-HitPack generalizes $\mathcal{X}$-Hitting and $\mathcal{X}$-Packing

- Significantly harder than base problems
- New algorithmic ideas are needed
- Some cases still allow for faster algorithms

Open questions:

- What about induced subgraphs or directed versions?
- Approximation results by relaxing the condition on k and ℓ?
- Is there a non-trivial relation to Erdős–Pósa for other graph classes?

Take away: Design the DPs to not waste time on impossible states.

Full version: `arXiv:2402.14927`

Appendix

The Vanilla Treewidth-DP

How does a typical treewidth-DP work?

The Vanilla Treewidth-DP

How does a typical treewidth-DP work?

- Identify the states of a vertex in a (partial) solution; usually a constant number of states, say c.

The Vanilla Treewidth-DP

How does a typical treewidth-DP work?

- Identify the states of a vertex in a (partial) solution;
  usually a constant number of states, say c.
- At each node of the tree decomposition:
  Adjust the states based on the changes in the graph.

The Vanilla Treewidth-DP

How does a typical treewidth-DP work?

- Identify the states of a vertex in a (partial) solution;
  usually a constant number of states, say c.
- At each node of the tree decomposition:
  Adjust the states based on the changes in the graph.

$\rightsquigarrow$ Running time of $c^{\mathrm{tw}^2} \cdot n^{\mathcal{O}(1)}$
(better convolution techniques frequently give $c^{\mathrm{tw}} \cdot n^{\mathcal{O}(1)}$).

The Vanilla Treewidth-DP

How does a typical treewidth-DP work?

- Identify the states of a vertex in a (partial) solution;
  usually a constant number of states, say c.
- At each node of the tree decomposition:
  Adjust the states based on the changes in the graph.

$\rightsquigarrow$ Running time of $c^{\mathrm{tw}^2} \cdot n^{\mathcal{O}(1)}$
(better convolution techniques frequently give $c^{\mathrm{tw}} \cdot n^{\mathcal{O}(1)}$).

For H-HitPack the states are “deleted” and “not deleted”, so $c = 2$?!

Then why is the problem double-exponential in treewidth?

Double-Exponential Lower Bound

Reduction from SAT to H-HitPack

SAT
Input: CNF-formula φ
Task: Decide if φ can be satisfied.

Double-Exponential Lower Bound

Reduction from SAT to H-HitPack

SAT
Input: CNF-formula φ
Task: Does there exist an assignment to the variables such that for all clauses at least one literal is true.

Double-Exponential Lower Bound

Reduction from SAT to H-HITPACK

SAT
Input: CNF-formula φ
Task:
Does there exist an assignment to the variables such that for all clauses not all literals are false.

Double-Exponential Lower Bound

Reduction from SAT to H-HitPack

SAT
Input: CNF-formula φ
Task: Does there exist an assignment to the variables such that for all clauses not all literals are false.

SAT		***H*-HitPack**
Select a variable assignment	$\rightsquigarrow$	Delete vertices
Select every clause (for verification)	$\rightsquigarrow$	Consider every packing
All literals are false	$\rightsquigarrow$	Packing is too large

Double-Exponential Lower Bound

Create an instance with treewidth $\mathcal{O}(\log m)$.