# Recent Advances on the Complexity of Parameterized Counting Problems

Dániel Marx

Institute for Computer Science and Control,
Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary

Joint work with Radu Curticapean and Holger Dell

SODA 2019
San Diego, CA
January 7, 2019

# Counting problems

Instead of **finding** one solution, we need to **count** the number of solutions.

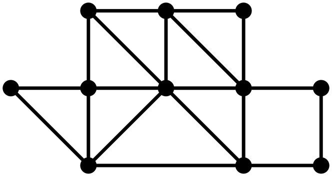
**Applications:** probability, statistical physics, pattern frequency...

# Counting problems

Instead of **finding** one solution, we need to **count** the number of solutions.

**Applications:** probability, statistical physics, pattern frequency...

RELIABILITY: If each edge fails with probability $\frac{1}{2}$ independently, what is the probability that the graph remains connected?



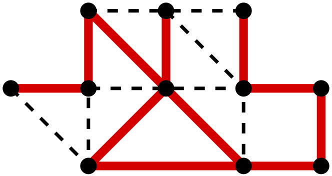
= counting the number of connected subgraphs

# Counting problems

Instead of **finding** one solution, we need to **count** the number of solutions.

**Applications:** probability, statistical physics, pattern frequency...

RELIABILITY: If each edge fails with probability $\frac{1}{2}$ independently, what is the probability that the graph remains connected?



= counting the number of connected subgraphs

# Finding vs. counting

**Finding** a perfect matching in a bipartite graph is polynomial-time solvable.

[Ford and Fulkerson 1956]

vs.

**Counting** the number of perfect matchings in a bipartite graph is #P-hard.

[Valiant 1979]

# This talk

Counting problems in the area of parameterized algorithms.

- Quick intro to parameterized algorithms.
- Connection between counting homomorphisms and subgraphs.
- Algorithmic applications.
- Complexity applications.

# This talk

Counting problems in the area of parameterized algorithms.

- Quick intro to parameterized algorithms.
- Connection between counting homomorphisms and subgraphs.
- Algorithmic applications.
- Complexity applications.

**Main message**

Parameterized subgraph counting problems can be understood via homomorphism counting problems.

...and this connection gives both algorithmic and complexity results!

# Parameterized problems

## Main idea

Instead of expressing the running time as a function $T(n)$ of $n$, we express it as a function $T(n,k)$ of the input size $n$ and some parameter $k$ of the input.

In other words: we do not want to be efficient on all inputs of size $n$, only for those where $k$ is small.

# Parameterized problems

**Main idea**

Instead of expressing the running time as a function $T(n)$ of $n$, we express it as a function $T(n, k)$ of the input size $n$ and some parameter $k$ of the input.

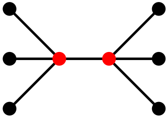
In other words: we do not want to be efficient on all inputs of size $n$, only for those where $k$ is small.

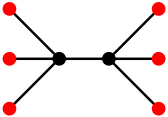
What can be the parameter $k$?

- The size $k$ of the solution we are looking for.
- The maximum degree of the input graph.
- The dimension of the point set in the input.
- The length of the strings in the input.
- The length of clauses in the input Boolean formula.
- ...

# Parameterized complexity

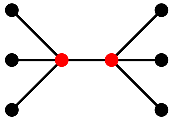
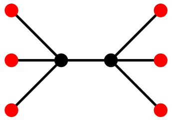
| | | |
|---|---|---|
| **Problem:** | VERTEX COVER | INDEPENDENT SET |
| **Input:** | Graph $G$, integer $k$ | Graph $G$, integer $k$ |
| **Question:** | Is it possible to cover the edges with $k$ vertices? | Is it possible to find $k$ independent vertices? |





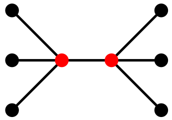
| | | |
|---|---|---|
| **Complexity:** | NP-complete | NP-complete |

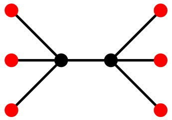
# Parameterized complexity

| | | |
|---|---|---|
| **Problem:** | VERTEX COVER | INDEPENDENT SET |
| **Input:** | Graph $G$, integer $k$ | Graph $G$, integer $k$ |
| **Question:** | Is it possible to cover the edges with $k$ vertices? | Is it possible to find $k$ independent vertices? |
| |  |  |
| **Complexity:** | NP-complete | NP-complete |
| **Brute force:** | $O(n^k)$ possibilities | $O(n^k)$ possibilities |

# Parameterized complexity

| | | |
|---|---|---|
| **Problem:** | VERTEX COVER | INDEPENDENT SET |
| **Input:** | Graph $G$, integer $k$ | Graph $G$, integer $k$ |
| **Question:** | Is it possible to cover the edges with $k$ vertices? | Is it possible to find $k$ independent vertices? |





| | | |
|---|---|---|
| **Complexity:** | NP-complete | NP-complete |
| **Brute force:** | $O(n^k)$ possibilities | $O(n^k)$ possibilities |

$O(2^k n^2)$ **algorithm exists** ☺

**No** $n^{o(k)}$ **algorithm known** ☹

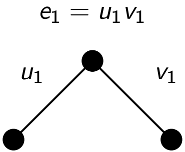
# Example: Bounded search tree method

Algorithm for VERTEX COVER:

$e_1 = u_1 v_1$

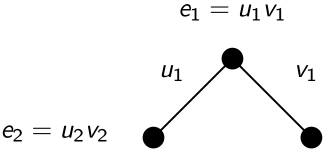
# Example: Bounded search tree method

Algorithm for VERTEX COVER:



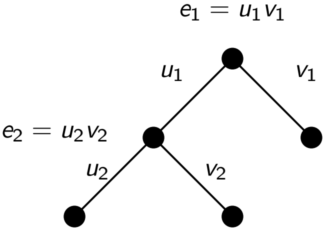
# Example: Bounded search tree method

Algorithm for Vertex Cover:



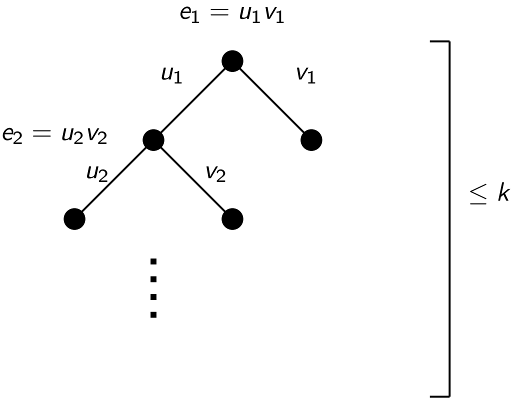
# Example: Bounded search tree method

Algorithm for Vertex Cover:



# Example: Bounded search tree method

Algorithm for Vertex Cover:




Height of the search tree $\leq k$ $\Rightarrow$ at most $2^k$ leaves $\Rightarrow$ $2^k \cdot n^{O(1)}$ time algorithm.

# Fixed-parameter tractability

**Main definition**

A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant $c$.

# Fixed-parameter tractability

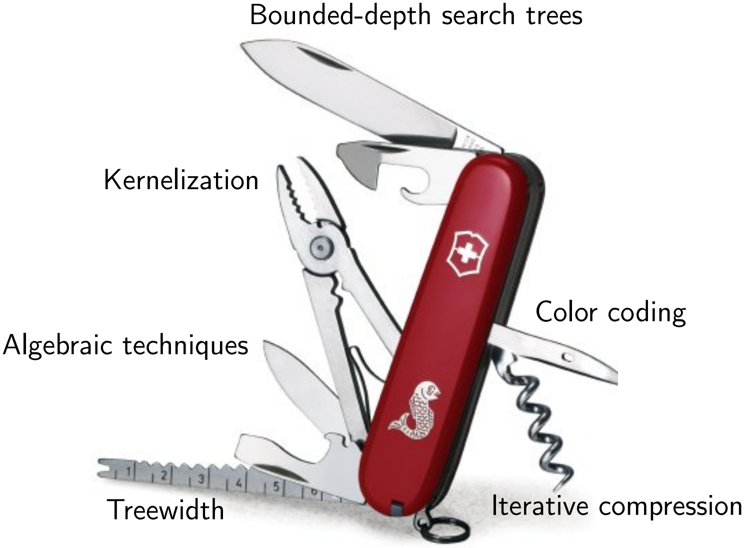
## Main definition

A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant $c$.

Examples of NP-hard problems that are FPT:

- Finding a vertex cover of size $k$.
- Finding a path of length $k$.
- Finding $k$ disjoint triangles.
- Drawing the graph in the plane with $k$ edge crossings.
- Finding disjoint paths that connect $k$ pairs of points.
- ...

# FPT techniques



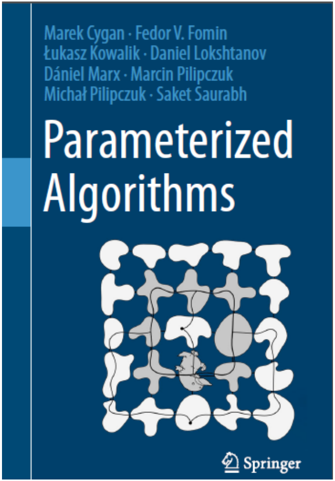
# W[1]-hardness

Negative evidence similar to NP-completeness. If a problem is **W[1]-hard,** then the problem is not FPT unless FPT=W[1].

Some W[1]-hard problems:

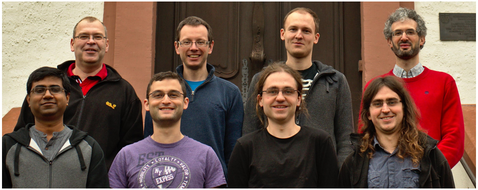
- Finding a clique/independent set of size $k$.
- Finding a dominating set of size $k$.
- Finding $k$ pairwise disjoint sets.
- ...

# Parameterized Algorithms




Marek Cygan, Fedor V. Fomin,
Lukasz Kowalik, Daniel Lokshtanov,
Dániel Marx, Marcin Pilipczuk,
Michał Pilipczuk, Saket Saurabh

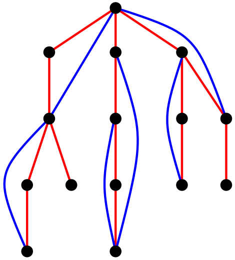
Springer 2015



# Example: Win/Win for $k$-Path

Simple $2^{O(k)} \cdot n^{O(1)}$ time algorithm for finding a path of length $k$.

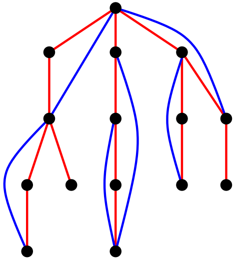
1. Compute a DFS tree.



# Example: Win/Win for $k$-Path

Simple $2^{O(k)} \cdot n^{O(1)}$ time algorithm for finding a path of length $k$.

1. Compute a DFS tree.

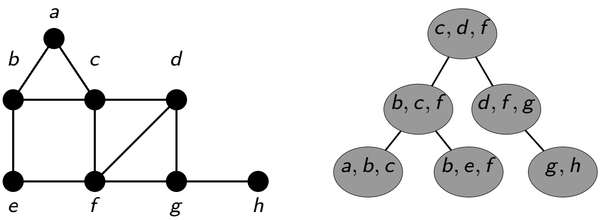


2. If DFS tree has height $> k$:
   we can find a $k$-path.
3. If DFS tree has height $\leq k$:
   treewidth is $\leq k \Rightarrow$ Use an algorithm with running time $2^{O(\text{tw})} \cdot n^{O(1)}$ for finding the longest path.

# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

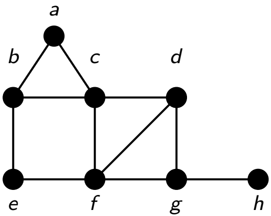
1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.

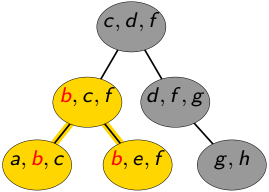


# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.

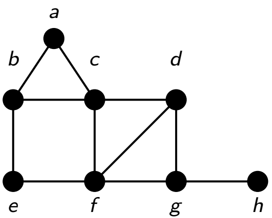


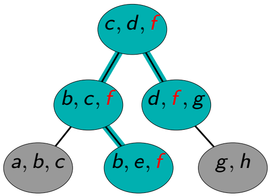



# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.



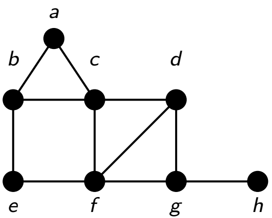
# Treewidth — a measure of "tree-likeness"

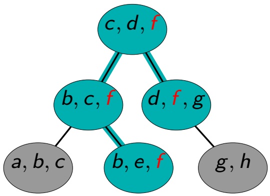
**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.

**Width of the decomposition:** largest bag size $-1$.

**treewidth:** width of the best decomposition.



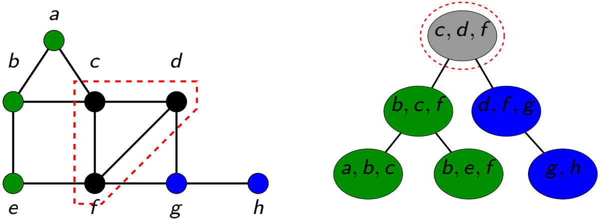
# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.

**Width of the decomposition:** largest bag size $-1$.

**treewidth:** width of the best decomposition.




Each bag is a separator.

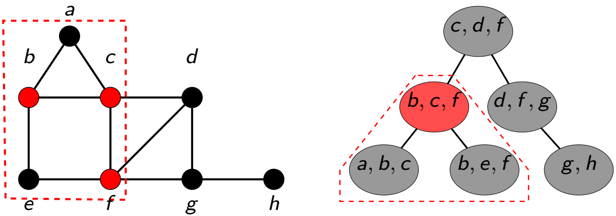
# Treewidth — a measure of "tree-likeness"

**Tree decomposition:** Vertices are arranged in a tree structure satisfying the following properties:

1. If $u$ and $v$ are neighbors, then there is a bag containing both of them.
2. For every $v$, the bags containing $v$ form a connected subtree.

**Width of the decomposition:** largest bag size $-1$.

**treewidth:** width of the best decomposition.




A subtree communicates with the outside world only via the root of the subtree.

# Counting complexity

- W[1]-hardness: "as hard as finding a $k$-clique"
- #W[1]-hardness: "as hard as counting $k$-cliques"

What can happen to the counting version of an FPT problem?

1. **(easy)** The same algorithmic technique shows that the counting problem is FPT.
2. **(easy, but different)** New algorithmic techniques are needed to show that the counting version is FPT.
3. **(hard)** New lower bound techniques are needed to show that the counting version is #W[1]-hard.

# Finding vs. counting

Generalization to counting:

- **WORKS** for the VERTEX COVER branching algorithm

  $\Rightarrow$ #VERTEX COVER is FPT.

- **DOES NOT WORK** for the #$k$-PATH win/win algorithm

  What is the parameterized complexity of #$k$-PATH?

# Finding vs. counting

Generalization to counting:

- **WORKS** for the VERTEX COVER branching algorithm

  $\Rightarrow$ #VERTEX COVER is FPT.

- **DOES NOT WORK** for the #$k$-PATH win/win algorithm

  What is the parameterized complexity of #$k$-PATH?

Even more troubling question:

What is the parameterized complexity of
the (even simpler) #$k$-MATCHING?

#$k$-MATCHING $\Rightarrow$ #$k$-PATH

# Counting $k$-paths and $k$-matchings

Colorful history:

- $\#k$-Path is #W[1]-hard
  [Flum and Grohe, FOCS 2002]

# Counting $k$-paths and $k$-matchings

Colorful history:

- #$k$-Path is #W[1]-hard
  [Flum and Grohe, FOCS 2002]
- Weighted #$k$-Matching is #W[1]-hard
  [Bläser and Curticapean, IPEC 2012]

# Counting $k$-paths and $k$-matchings

Colorful history:

- $\#k$-Path is #W[1]-hard
  [Flum and Grohe, FOCS 2002]
- Weighted $\#k$-Matching is #W[1]-hard
  [Bläser and Curticapean, IPEC 2012]
- Unweighted $\#k$-Matching is #W[1]-hard
  [Curticapean, ICALP 2013] — complicated proof.

# Counting $k$-paths and $k$-matchings

Colorful history:

- $\#k$-Path is #W[1]-hard
  [Flum and Grohe, FOCS 2002]
- Weighted $\#k$-Matching is #W[1]-hard
  [Bläser and Curticapean, IPEC 2012]
- Unweighted $\#k$-Matching is #W[1]-hard
  [Curticapean, ICALP 2013] — complicated proof.
- Unweighted $\#k$-Matching is #W[1]-hard
  [Curticapean and M, FOCS 2014] — simpler proof.

# Counting $k$-paths and $k$-matchings

Colorful history:

- $\#k$-PATH is #W[1]-hard
  [Flum and Grohe, FOCS 2002]
- Weighted $\#k$-MATCHING is #W[1]-hard
  [Bläser and Curticapean, IPEC 2012]
- Unweighted $\#k$-MATCHING is #W[1]-hard
  [Curticapean, ICALP 2013] — complicated proof.
- Unweighted $\#k$-MATCHING is #W[1]-hard
  [Curticapean and M, FOCS 2014] — simpler proof.
- Unweighted $\#k$-MATCHING is #W[1]-hard
  [Curticapean and M, unpublished] — even simpler proof.

# Counting $k$-paths and $k$-matchings

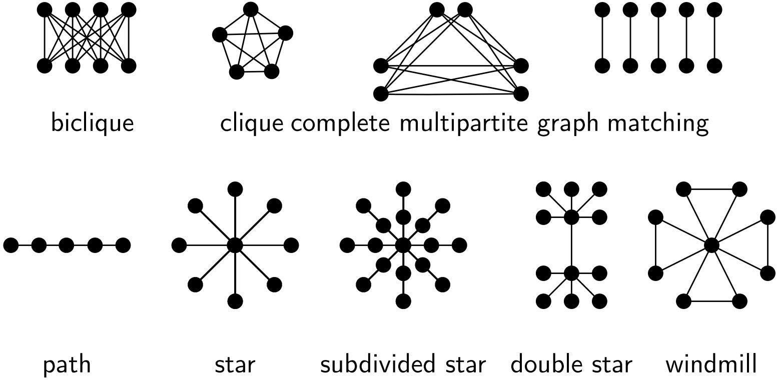
Colorful history:

- $\#k$-PATH is #W[1]-hard
  [Flum and Grohe, FOCS 2002]
- Weighted $\#k$-MATCHING is #W[1]-hard
  [Bläser and Curticapean, IPEC 2012]
- Unweighted $\#k$-MATCHING is #W[1]-hard
  [Curticapean, ICALP 2013] — complicated proof.
- Unweighted $\#k$-MATCHING is #W[1]-hard
  [Curticapean and M, FOCS 2014] — simpler proof.
- Unweighted $\#k$-MATCHING is #W[1]-hard
  [Curticapean and M, unpublished] — even simpler proof.
- Unweighted $\#k$-MATCHING is #W[1]-hard
  [Curticapean, Dell, and M, STOC 2017] — *tells the real story.*

# Counting patterns

## Main question

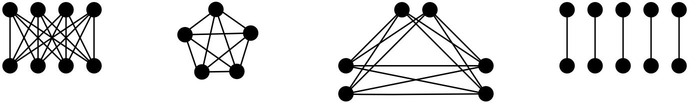
Which type of subgraph patterns are easy to count?



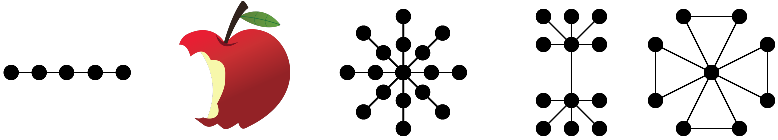
# Counting patterns

## Main question

Which type of subgraph patterns are easy to count?



biclique clique complete multipartite graph matching

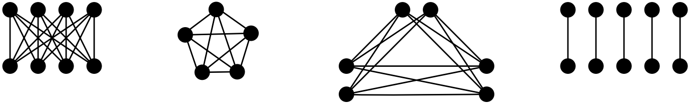


path star subdivided star double star windmill

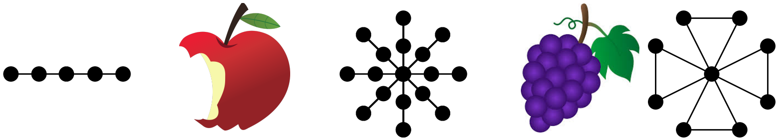
# Counting patterns

## Main question

Which type of subgraph patterns are easy to count?



biclique clique complete multipartite graph matching

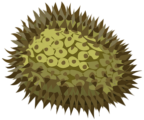
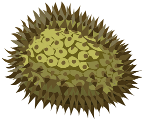
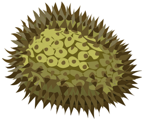
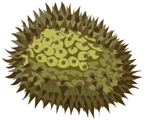


path star subdivided star double star windmill

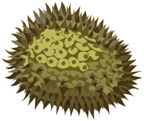
# Counting patterns

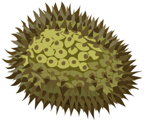
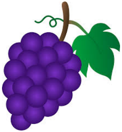
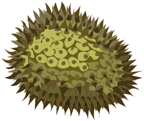
## Main question

Which type of subgraph patterns are easy to count?

biclique clique complete multipartite graph matching



path star subdivided star double star windmill

# Counting patterns

Patterns with small vertex cover number are is easy to count:

**Theorem** [multiple references]

$\#\textsc{Sub}(H)$ can be solved in time $n^{\mathsf{vc}(H)+O(1)}$.

But what about patterns with large vertex cover number?

# Counting patterns

Patterns with small vertex cover number are is easy to count:

**Theorem** [multiple references]

$\#\textsc{Sub}(H)$ can be solved in time $n^{\mathrm{vc}(H)+O(1)}$.

But what about patterns with large vertex cover number?

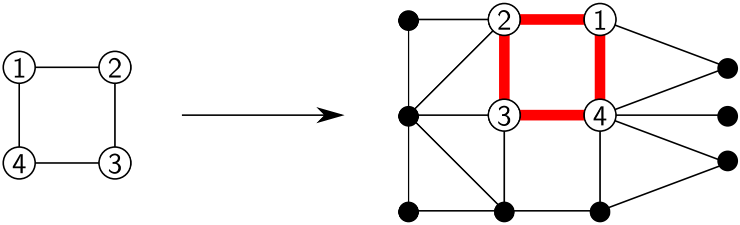
We will understand the complexity of counting any class of patterns, not just paths or matchings!

**Main message**

Parameterized subgraph counting problems can be understood via homomorphism counting problems.

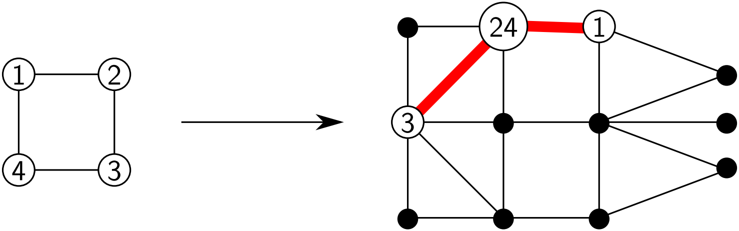
# Homomorphisms

A **homomorphism** from $H$ to $G$ is a mapping $f\colon V(H) \to V(G)$ such that if $ab$ is an edge of $H$, then $f(a)f(b)$ is an edge of $G$.



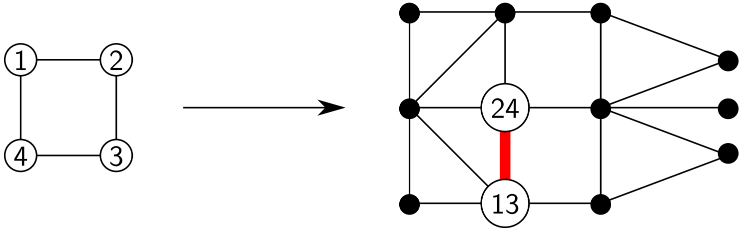
# Homomorphisms

A **homomorphism** from $H$ to $G$ is a mapping $f\colon V(H) \to V(G)$ such that if $ab$ is an edge of $H$, then $f(a)f(b)$ is an edge of $G$.



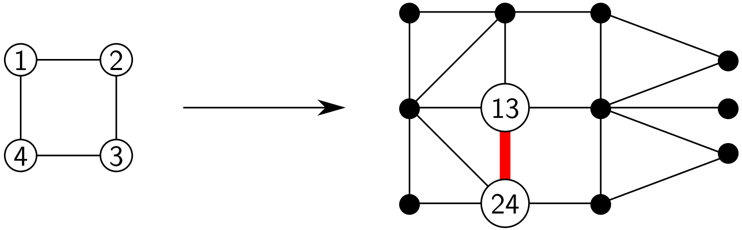
# Homomorphisms

A **homomorphism** from $H$ to $G$ is a mapping $f\colon V(H) \to V(G)$ such that if $ab$ is an edge of $H$, then $f(a)f(b)$ is an edge of $G$.



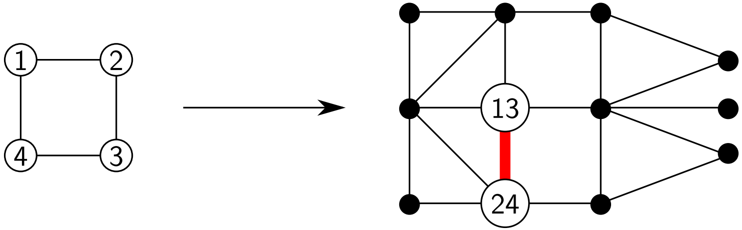
# Homomorphisms

A **homomorphism** from $H$ to $G$ is a mapping $f\colon V(H) \to V(G)$ such that if $ab$ is an edge of $H$, then $f(a)f(b)$ is an edge of $G$.



# Homomorphisms

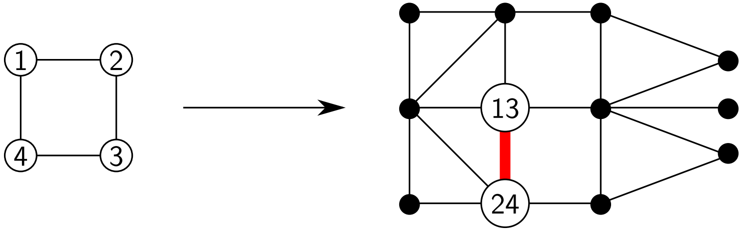
A **homomorphism** from $H$ to $G$ is a mapping $f\colon V(H) \to V(G)$ such that if $ab$ is an edge of $H$, then $f(a)f(b)$ is an edge of $G$.




Which pattern graphs $H$ are easy for counting homomorphisms?

# Homomorphisms

A **homomorphism** from $H$ to $G$ is a mapping $f\colon V(H) \to V(G)$ such that if $ab$ is an edge of $H$, then $f(a)f(b)$ is an edge of $G$.




Which pattern graphs $H$ are easy for counting homomorphisms?

**Theorem (trivial)**

For every fixed $H$, the problem $\#\textsc{Hom}(H)$ (count homomorphisms from $H$ to the given graph $G$) is polynomial-time solvable.

... because we can try all $|V(G)|^{|V(H)|}$ possible mappings $f\colon V(H) \to V(G)$.

# Counting homomorphisms

**Better questions:**

- Which classes $\mathcal{H}$ (e.g., paths, stars, matchings) of patterns can be counted in polynomial time?
- What is the best exponent for $\textsc{Hom}(H)$?

**Fact**

$\#\textsc{Hom}(H)$ can be solved in time $O(n^{\mathrm{tw}(H)+1})$.

# Counting homomorphisms

**Better questions:**

- Which classes $\mathcal{H}$ (e.g., paths, stars, matchings) of patterns can be counted in polynomial time?
- What is the best exponent for $\textsc{Hom}(H)$?

**Fact**

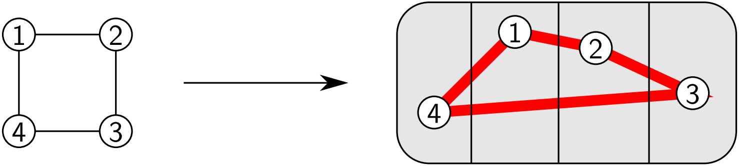
$\#\textsc{Hom}(H)$ can be solved in time $O(n^{\mathrm{tw}(H)+1})$.

Difference between finding and counting:

| | | |
|---|---|---|
| **Finding:** $\textsc{Hom}(K_{k,k})$ is trivial | vs. | **Counting:** $\#\textsc{Hom}(K_{k,k})$ is W[1]-hard |

# Partitioned homomorphism

**PARTITIONED HOMOMORPHISM**
**Input:** $H$, $G$, and partition $\Pi$ of $V(G)$ into $|V(H)|$ classes.
**Task:** Find a homomorphism $\phi$ that maps the vertices of $H$ to different classes.




**Theorem** [M 2010]

There is a universal constant $\gamma > 0$ such that if for some $H$ there is an $O(n^{\gamma \cdot \mathrm{tw}(H)/\log \mathrm{tw}(H)})$ time algorithm for PARTHOM($H$), then ETH fails.

# Counting partitioned homomorphisms

$\#\textsc{PartHom}(H)$ $\Rightarrow$ $\#\textsc{Hom}(H)$

$G_P$ for $P \subseteq \mathrm{class}(\Pi)$: subgraph of $G$ induced by the classes $P$.

Simple application of the inclusion-exclusion principle:

$$\#\text{part-hom}(H, G) = \sum_{P \subseteq \mathrm{class}(\Pi)} (-1)^{|\mathrm{class}(P)| - |P|} \cdot \#\text{hom}(H, G_P)$$

# Counting partitioned homomorphisms

$\#\textsc{PartHom}(H)$ $\Rightarrow$ $\#\textsc{Hom}(H)$

$G_P$ for $P \subseteq \mathrm{class}(\Pi)$: subgraph of $G$ induced by the classes $P$.

Simple application of the inclusion-exclusion principle:

$$\#\text{part-hom}(H,G) = \sum_{P \subseteq \mathrm{class}(\Pi)} (-1)^{|\mathrm{class}(P)|-|P|} \cdot \#\mathrm{hom}(H, G_P)$$

**Theorem** [Dalmau and Jonsson 2004][M 2010]

There is a universal constant $\gamma > 0$ such that if for some $H$ there is an $O(n^{\gamma \cdot \mathrm{tw}(H)/\log \mathrm{tw}(H)})$ time algorithm for $\#\textsc{Hom}(H)$, then ETH fails.

# Counting homomorphisms — summary

Treewidth of $H$ determines the complexity of the problem:

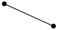
- $O(n^{\mathrm{tw}(H)+1})$ upper bound.
- $\Omega(n^{\gamma \cdot \mathrm{tw}(H)/\log \mathrm{tw}(H)})$ lower bound (assuming ETH).

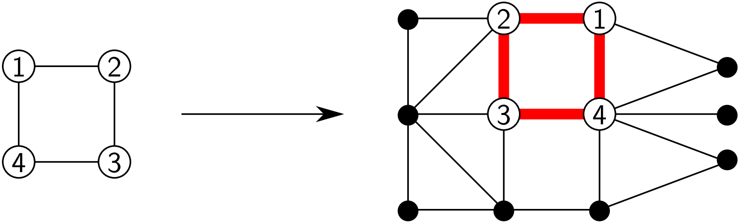
If we restrict the problem to a class $\mathcal{H}$ of patterns:

- If $\mathcal{H}$ has bounded treewidth (e.g, stars, paths, ...), then the problem is polynomial-time solvable.
- If $\mathcal{H}$ has unbounded treewidth (e.g, cliques, bicliques, grids, ...), then the problem is **not** polynomial-time solvable (assuming ETH).

# Subgraphs $\Leftrightarrow$ homomorphisms

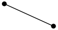
**Easy to check:**

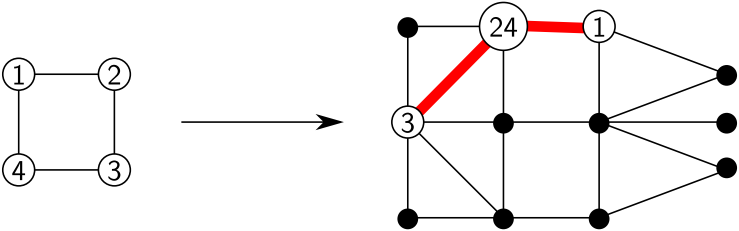
hom( , $G$) $=$ 8sub( , $G$) $+$ 4sub( , $G$) $+$ 2sub( , $G$)



# Subgraphs $\Leftrightarrow$ homomorphisms

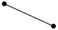
**Easy to check:**

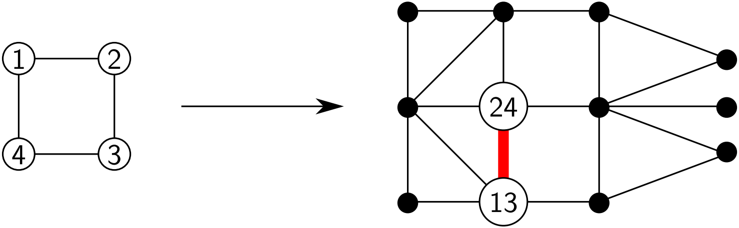
hom(  , $G$) $=$ 8sub(  , $G$) $+$ 4sub(  , $G$) $+$ 2sub(  , $G$)



# Subgraphs $\Leftrightarrow$ homomorphisms

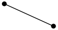
**Easy to check:**

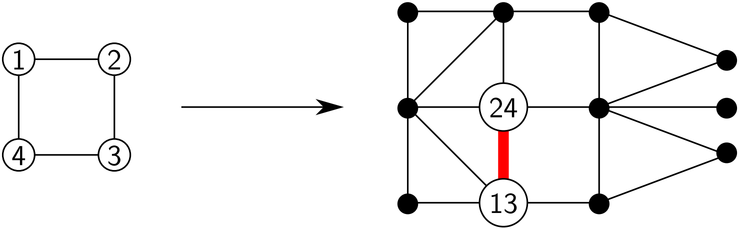
hom(  , $G$) $=$ 8sub(  , $G$) $+$ 4sub(  , $G$) $+$ 2sub(  , $G$)



# Subgraphs ⇔ homomorphisms

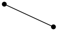
**Easy to check:**

hom(  , $G$) $=$ 8sub(  , $G$) $+$ 4sub(  , $G$) $+$ 2sub(  , $G$)




**Not completely obvious:**

The formula can be reversed by inclusion-exclusion.

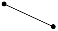
sub(  , $G$) $= \frac{1}{8}$hom(  , $G$) $- \frac{1}{4}$hom(  , $G$) $+ \frac{1}{8}$hom(  , $G$)

# Subgraphs ⇔ homomorphisms

**Definition**

surj($H$, $G$): number of surjective homomorphisms from $H$ to $G$ (every vertex and edge of $G$ appears in the image).

Homomorphisms can be counted by classifying according to the image $F$:

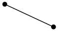
hom(  , $G$) $=$ 8sub(  , $G$) $+$ 4sub(  , $G$) $+$ 2sub(  , $G$)

# Subgraphs ⇔ homomorphisms

**Definition**

surj$(H, G)$: number of surjective homomorphisms from $H$ to $G$ (every vertex and edge of $G$ appears in the image).

Homomorphisms can be counted by classifying according to the image $F$:

hom(  , $G$) = 8sub(  , $G$) + 4sub(  , $G$) + 2sub(  , $G$)

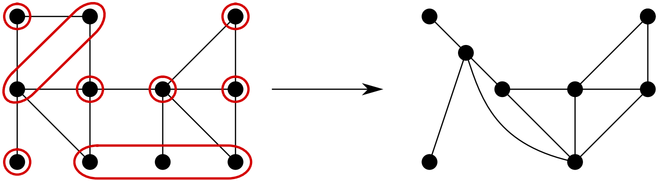
$$\Downarrow$$

$$\hom(H, G) = \sum_F \mathrm{surj}(H, F)\mathrm{sub}(F, G)$$

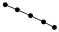
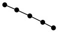
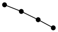
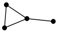
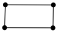
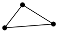
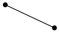
Which of the terms can be nonzero?

# Spasm

- $\text{Part}_0(H)$: set of partitions of $V(H)$ where each class is an independent set.
- For $\Pi \in \text{Part}_0(H)$, $H_{|\Pi}$ is obtained by contracting each class of $\Pi$ to a single vertex.



- $\text{Spasm} = \{H_{|\Pi} \mid \Pi \in \text{Part}_0(H)\}$

Spasm(  ) = { , , , , , , ,  }

# Subgraphs $\Leftrightarrow$ homomorphisms

**From subgraphs to homomorphisms:**

$$\mathrm{hom}(H, G) = \sum_F \mathrm{surj}(H, F)\mathrm{sub}(F, G)$$

where $\mathrm{surj}(H, F) \neq 0$ if and only if $F \in \mathrm{Spasm}(H)$.

**From homomorphisms to subgraphs:** [Lovász 1967]

$$\mathrm{sub}(H, G) = \sum_F \beta_F \cdot \mathrm{hom}(F, G)$$

where $\beta_F \neq 0$ if and only if $F \in \mathrm{Spasm}(H)$.

# Subgraphs $\Leftrightarrow$ homomorphisms

**From subgraphs to homomorphisms:**

$$\mathrm{hom}(H,G) = \sum_F \mathrm{surj}(H,F)\mathrm{sub}(F,G)$$

where $\mathrm{surj}(H,F) \neq 0$ if and only if $F \in \mathrm{Spasm}(H)$.

. . . useless.

**From homomorphisms to subgraphs:** [Lovász 1967]

$$\mathrm{sub}(H,G) = \sum_F \beta_F \cdot \mathrm{hom}(F,G)$$

where $\beta_F \neq 0$ if and only if $F \in \mathrm{Spasm}(H)$.

# Subgraphs $\Leftrightarrow$ homomorphisms

**From subgraphs to homomorphisms:**

$$\mathrm{hom}(H, G) = \sum_F \mathrm{surj}(H, F)\mathrm{sub}(F, G)$$

where $\mathrm{surj}(H, F) \neq 0$ if and only if $F \in \mathrm{Spasm}(H)$.

. . . useless.

**From homomorphisms to subgraphs:** [Lovász 1967]

$$\mathrm{sub}(H, G) = \sum_F \beta_F \cdot \mathrm{hom}(F, G)$$

where $\beta_F \neq 0$ if and only if $F \in \mathrm{Spasm}(H)$.

Extremely useful for applications in algorithms and complexity!

# Algorithmic applications

$$\mathrm{sub}(H, G) = \sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, G)$$

$$\Downarrow$$

Max. treewidth in $\mathrm{Spasm}(H)$ gives an upper bound on complexity:

**Corollary**

If every graph in $\mathrm{Spasm}(H)$ has treewidth at most $c$, then $\mathrm{sub}(H, G)$ can be computed in time $O(n^{c+1})$.

# Algorithmic applications

**Corollary**

If every graph in $\mathrm{Spasm}(H)$ has treewidth at most $c$, then $\mathrm{sub}(H, G)$ can be computed in time $O(n^{c+1})$.

**Observe:** If $H$ has $k$ edges, then every graph in $\mathrm{Spasm}(H)$ has at most $k$ edges.

# Algorithmic applications

## Corollary

If every graph in $\mathrm{Spasm}(H)$ has treewidth at most $c$, then $\mathrm{sub}(H, G)$ can be computed in time $O(n^{c+1})$.

**Observe:** If $H$ has $k$ edges, then every graph in $\mathrm{Spasm}(H)$ has at most $k$ edges.

## Theorem [Scott and Sorkin 2007]

Every graph with $\leq k$ edges has treewidth at most $0.174k + O(1)$.

## Corollary

If $H$ has $k$ edges, then $\mathrm{sub}(H, G)$ can be computed in time $n^{0.174k+O(1)}$.

# Counting $k$-paths

**Corollary**

If $H$ has $k$ edges, then $\mathrm{sub}(H, G)$ can be computed in time $n^{0.174k+O(1)}$.

**Example:** Counting $k$-paths

- Brute force: $O(n^k)$.
- Meet in the middle: $O(n^{0.5k})$
  [Björklund et al., ESA 2009],[Koutis and Williams, ICALP 2009]
- [Björklund et al., SODA 2014]: $n^{0.455k+O(1)}$.
- New! by counting homomorphisms in the spasm: $n^{0.174k+O(1)}$.

# Counting small cycles

Counting triangles using matrix multiplication:

$$\mathrm{sub}(C_3, G) = \frac{1}{6}\mathrm{tr}\ \mathrm{Adj}^3(G)$$

# Counting small cycles

Counting triangles using matrix multiplication:

$$\text{sub}(C_3, G) = \frac{1}{6}\text{tr Adj}^3(G)$$

**Theorem** [Alon, Yuster, and Zwick, ESA 1994]

For $k \leq 7$, we can compute $\text{sub}(C_k, G)$ in time $n^\omega$ (where $\omega < 2.373$ is the matrix-multiplication exponent).

# Counting small cycles

Counting triangles using matrix multiplication:

$$\mathrm{sub}(C_3, G) = \frac{1}{6}\mathrm{tr}\ \mathrm{Adj}^3(G)$$

**Theorem** [Alon, Yuster, and Zwick, ESA 1994]

For $k \leq 7$, we can compute $\mathrm{sub}(C_k, G)$ in time $n^\omega$ (where $\omega < 2.373$ is the matrix-multiplication exponent).

We can recover this result:

- Check: if $k \leq 7$, then every graph in $\mathrm{Spasm}(C_k, G)$ has treewidth at most $2$.
- For treewidth 2, the $O(n^{2+1})$ homomorphism algorithm can be improved to $O(n^\omega)$ with fast matrix multiplication.
- $\Rightarrow O(n^\omega)$ algorithm for $\mathrm{sub}(C_k, G)$ if $k \leq 7$.

# Complexity applications

$$\mathrm{sub}(H, G) = \sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, G)$$

**Note:** Every $\beta_F$ is nonzero.

# Complexity applications

$$\mathrm{sub}(H, G) = \sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, G)$$

**Note:** Every $\beta_F$ is nonzero.

Reductions:

- **Obvious:**
  if we can compute $\mathrm{hom}(F, G)$ for any $F \in \mathrm{Spasm}(H)$
  $\Rightarrow$ we can compute $\mathrm{sub}(H, G)$.

# Complexity applications

$$\mathrm{sub}(H, G) = \sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, G)$$

**Note:** Every $\beta_F$ is nonzero.

Reductions:

- **Obvious:**
  if we can compute $\mathrm{hom}(F, G)$ for any $F \in \mathrm{Spasm}(H)$
  $\Rightarrow$ we can compute $\mathrm{sub}(H, G)$.
- **Highly nontrivial:**
  if we can compute $\mathrm{sub}(H, G)$
  $\Rightarrow$ we can compute $\mathrm{hom}(F, G)$ for any $F \in \mathrm{Spasm}(H)$.

# Complexity applications

$$\mathrm{sub}(H, G) = \sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, G)$$

**Note:** Every $\beta_F$ is nonzero.

Reductions:

- **Obvious:**
  if we can compute $\mathrm{hom}(F, G)$ for any $F \in \mathrm{Spasm}(H)$
  $\Rightarrow$ we can compute $\mathrm{sub}(H, G)$.
- **Highly nontrivial:**
  if we can compute $\mathrm{sub}(H, G)$
  $\Rightarrow$ we can compute $\mathrm{hom}(F, G)$ for any $F \in \mathrm{Spasm}(H)$.

Complexity of $\mathrm{hom}(F, G)$ for any $F \in \mathrm{Spasm}(H)$ is a **lower bound** on the complexity of $\mathrm{sub}(H, G)$.

# Matrices

Fix an enumeration of graphs with $\leq k$ edges with nondecreasing number of edges.

- Hom matrix: row $i$, column $j$ is $\hom(H_i, H_j)$.
- Sub matrix: row $i$, column $j$ is $\mathrm{sub}(H_i, H_j)$.
- Surj matrix: row $i$, column $j$ is $\mathrm{surj}(H_i, H_j)$.

# Matrices

Fix an enumeration of graphs with $\leq k$ edges with nondecreasing number of edges.

- Hom matrix: row $i$, column $j$ is $\mathrm{hom}(H_i, H_j)$.
- Sub matrix: row $i$, column $j$ is $\mathrm{sub}(H_i, H_j)$.
- Surj matrix: row $i$, column $j$ is $\mathrm{surj}(H_i, H_j)$.

$$\mathrm{hom}(H, G) = \sum_F \mathrm{surj}(H, F)\mathrm{sub}(F, G)$$
$$\Downarrow$$
$$\mathrm{Hom} = \mathrm{Surj} \cdot \mathrm{Sub}$$

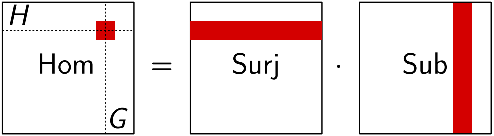
$$\boxed{\mathrm{Hom}} = \boxed{\mathrm{Surj}} \cdot \boxed{\mathrm{Sub}}$$

# Matrices

Fix an enumeration of graphs with $\leq k$ edges with nondecreasing number of edges.

- Hom matrix: row $i$, column $j$ is $\hom(H_i, H_j)$.
- Sub matrix: row $i$, column $j$ is $\mathrm{sub}(H_i, H_j)$.
- Surj matrix: row $i$, column $j$ is $\mathrm{surj}(H_i, H_j)$.

$$\hom(H, G) = \sum_F \mathrm{surj}(H, F)\mathrm{sub}(F, G)$$
$$\Downarrow$$
$$\mathrm{Hom} = \mathrm{Surj} \cdot \mathrm{Sub}$$

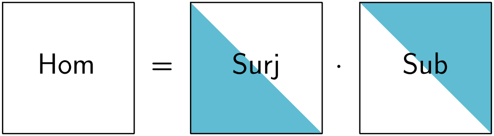


# Matrices

Fix an enumeration of graphs with $\leq k$ edges with nondecreasing number of edges.

- Hom matrix: row $i$, column $j$ is $\mathrm{hom}(H_i, H_j)$.
- Sub matrix: row $i$, column $j$ is $\mathrm{sub}(H_i, H_j)$.
- Surj matrix: row $i$, column $j$ is $\mathrm{surj}(H_i, H_j)$.

$$\mathrm{hom}(H, G) = \sum_F \mathrm{surj}(H, F)\mathrm{sub}(F, G)$$
$$\Downarrow$$
$$\mathrm{Hom} = \mathrm{Surj} \cdot \mathrm{Sub}$$




The Hom matrix is invertible!

# Categorical product

One of the standard graph products:

**Definition**

$G_1 \times G_2$ has vertex set $V(G_1) \times V(G_2)$ and $(v_1, v_2)$ and $(v_1', v_2')$ adjacent in $G_1 \times G_2 \iff v_1 v_1' \in E(G_1)$ and $v_2 v_2' \in E(G_2)$.

[missing figure]

Exercise:

$$\hom(H, G_1 \times G_2) = \hom(H, G_1) \cdot \hom(H, G_2)$$

# Extracting a term

**Lemma**

Given an algorithm for $\mathrm{sub}(H, G) = \sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, G)$ (with $\beta_F \neq 0$), we can compute $\mathrm{hom}(F, G)$ for any $F \in \mathrm{Spasm}(H)$.

Use the algorithm on $Z \times G$ for every $Z$ with $\leq k = |E(H)|$ edges.

$$\mathrm{sub}(H, Z \times G) = b_Z$$

# Extracting a term

**Lemma**

Given an algorithm for $\mathrm{sub}(H, G) = \sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, G)$ (with $\beta_F \neq 0$), we can compute $\mathrm{hom}(F, G)$ for any $F \in \mathrm{Spasm}(H)$.

Use the algorithm on $Z \times G$ for every $Z$ with $\leq k = |E(H)|$ edges.

$$\sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, Z \times G) = b_Z$$

# Extracting a term

**Lemma**

Given an algorithm for $\mathrm{sub}(H, G) = \sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, G)$ (with $\beta_F \neq 0$), we can compute $\mathrm{hom}(F, G)$ for any $F \in \mathrm{Spasm}(H)$.

Use the algorithm on $Z \times G$ for every $Z$ with $\leq k = |E(H)|$ edges.

$$\sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, Z) \cdot \mathrm{hom}(F, G) = b_Z$$

# Extracting a term

**Lemma**

Given an algorithm for $\mathrm{sub}(H, G) = \sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, G)$ (with $\beta_F \neq 0$), we can compute $\mathrm{hom}(F, G)$ for any $F \in \mathrm{Spasm}(H)$.

Use the algorithm on $Z \times G$ for every $Z$ with $\leq k = |E(H)|$ edges.

$$\sum_{F \in \mathrm{Spasm}(H)} \mathrm{hom}(F, Z) \cdot \beta_F \cdot \mathrm{hom}(F, G) = b_Z$$

# Extracting a term

**Lemma**

Given an algorithm for $\mathrm{sub}(H,G)=\sum_{F\in\mathrm{Spasm}(H)}\beta_F\cdot\mathrm{hom}(F,G)$ (with $\beta_F\neq 0$), we can compute $\mathrm{hom}(F,G)$ for any $F\in\mathrm{Spasm}(H)$.

Use the algorithm on $Z\times G$ for every $Z$ with $\leq k=|E(H)|$ edges.

$$\sum_{F\in\mathrm{Spasm}(H)}\mathrm{hom}(F,Z)\cdot\beta_F\cdot\mathrm{hom}(F,G)=b_Z$$

$$\mathrm{Hom}^T\cdot\begin{pmatrix}\beta_{F_1}\cdot\mathrm{hom}(F_1,G)\\ \vdots\\ \beta_{F_t}\cdot\mathrm{hom}(F_t,G)\end{pmatrix}=\begin{pmatrix}b_{Z_1}\\ \vdots\\ b_{Z_t}\end{pmatrix}$$

# Extracting a term

**Lemma**

Given an algorithm for $\mathrm{sub}(H,G) = \sum_{F \in \mathrm{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F,G)$ (with $\beta_F \neq 0$), we can compute $\mathrm{hom}(F,G)$ for any $F \in \mathrm{Spasm}(H)$.

Use the algorithm on $Z \times G$ for every $Z$ with $\leq k = |E(H)|$ edges.

$$\sum_{F \in \mathrm{Spasm}(H)} \mathrm{hom}(F,Z) \cdot \beta_F \cdot \mathrm{hom}(F,G) = b_Z$$

$$\mathrm{Hom}^T \cdot \begin{pmatrix} \beta_{F_1} \cdot \mathrm{hom}(F_1, G) \\ \vdots \\ \beta_{F_t} \cdot \mathrm{hom}(F_t, G) \end{pmatrix} = \begin{pmatrix} b_{Z_1} \\ \vdots \\ b_{Z_t} \end{pmatrix}$$

The Hom matrix is invertible, so we can solve this system of equations!

# Extracting a term

**Lemma**

Given an algorithm for $\mathrm{sub}(H, G) = \sum_{F \in \mathsf{Spasm}(H)} \beta_F \cdot \mathrm{hom}(F, G)$ (with $\beta_F \neq 0$), we can compute $\mathrm{hom}(F, G)$ for any $F \in \mathsf{Spasm}(H)$.

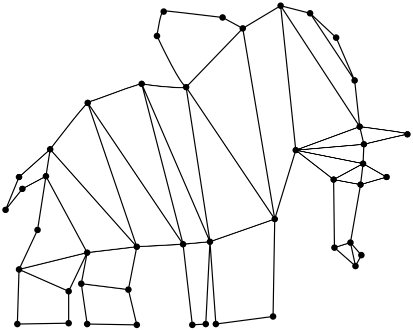
**Bottom line:**

| complexity of $\#\textsc{Sub}(H)$ | $=$ | hardest $\#\textsc{Hom}(F)$ for $F \in \mathsf{Spasm}(F)$ |
|---|---|---|

Complexity depends on the maximum treewidth in $\mathsf{Spasm}(H)$!

# Complexity of counting patterns

What is the best exponent for counting occurrences of this 46-vertex graph $H$?



**Answer:** Compute Spasm($H$) and find the best exponent for each of the resulting homomorphism problems!

# Hardness results for $\#k$-Matching

**Not FPT:**

**Theorem**

Counting $k$-matchings is #W[1]-hard.

## Hardness results for $\#k$-Matching

**Not FPT:**

**Theorem**

Counting $k$-matchings is #W[1]-hard.

**Proof:** As $K_k \in \mathrm{Spasm}(M_{\binom{k}{2}})$, counting $k$-cliques can be reduced to counting $\binom{k}{2}$-matchings. $\square$

# Hardness results for $\#k$-Matching

**Not FPT:**

**Theorem**

Counting $k$-matchings is #W[1]-hard.

**Proof:** As $K_k \in \text{Spasm}(M_{\binom{k}{2}})$, counting $k$-cliques can be reduced to counting $\binom{k}{2}$-matchings. □

**More precise bound:**

$\text{Spasm}(M_k)$ contains every graph with $k$ edges

$\Downarrow$

$\text{Spasm}(M_k)$ contains graphs with treewidth $\Omega(k)$

$\Downarrow$

no $f(k)n^{o(k/\log k)}$ time algorithm for $\#k$-Matching, assuming ETH.

# Role of vertex cover number

What property of $H$ determines the max. treewidth in $\mathrm{Spasm}(H)$?

# Role of vertex cover number

What property of $H$ determines the max. treewidth in $\mathrm{Spasm}(H)$?

The vertex cover number of $H$:

- **Upper bound:**
  For every $F \in \mathrm{Spasm}(H)$, we have $\mathrm{tw}(F) \leq \mathrm{vc}(F) \leq \mathrm{vc}(H)$.
- **Lower bound:**
  $H$ contains a matching of size $\mathrm{vc}(H)/2$. We can show that for any $F$ with at most $\mathrm{vc}(H)/2$ edges, there is a graph in $\mathrm{Spasm}(H)$ that contains $F$ as a minor $\Rightarrow$ there is a graph in $\mathrm{Spasm}(H)$ with treewidth $\Omega(\mathrm{vc}(H))$.

## Counting subgraphs — summary

Vertex cover number of $H$ determines the complexity of $\textsc{Sub}(H)$:

- $n^{\mathrm{vc}(H)+O(1)}$ upper bound.
- $\Omega(n^{\gamma\cdot \mathrm{vc}(H)/\log \mathrm{vc}(H)})$ lower bound.

If we restrict the problem to a class $\mathcal{H}$ of patterns:

- If $\mathcal{H}$ has bounded vertex cover number (e.g, stars, double stars, ...), then the problem is polynomial-time solvable.
- If $\mathcal{H}$ has unbounded vertex cover number (e.g, cliques, paths, matchings, disjoint triangles, ...), then the problem is **not** polynomial-time solvable (assuming ETH).

# Conclusions

**Main message**

Parameterized subgraph counting problems can be understood via homomorphism counting problems.

. . .and this connection gives both algorithmic and complexity results!

Working on counting problems is fun:

- You can revisit fundamental, “well-understood” problems.
- Requires a new set of lower bound techniques.
- Requires new algorithmic techniques.