Lower bounds for polynomial kernelization

Part 2

Michał Pilipczuk

Institute of Informatics, University of Warsaw, Poland

Recent Advances in Parameterized Complexity

Tel Aviv, December 4th, 2017

PPTs

- **Idea**: Hardness of kernelization can be transferred via reductions, similarly to **NP**-hardness.

PPTs

- **Idea**: Hardness of kernelization can be transferred via reductions, similarly to **NP**-hardness.

Polynomial parameter transformation (PPT)

A **polynomial parameter transformation** from a parameterized problem P to a parameterized problem Q is a polynomial-time algorithm that transforms a given instance (x, k) of P into an equivalent instance (x', k') of Q such that $k' = \mathrm{poly}(k)$.

PPTs

- **Idea**: Hardness of kernelization can be transferred via reductions, similarly to **NP**-hardness.

Polynomial parameter transformation (PPT)

A **polynomial parameter transformation** from a parameterized problem P to a parameterized problem Q is a polynomial-time algorithm that transforms a given instance (x, k) of P into an equivalent instance (x', k') of Q such that $k' = \mathrm{poly}(k)$.

Observation

If problem P PPT-reduces to Q, and P does not admit a polynomial compression algorithm (into any language R), then neither does Q.

PPTs

- **Idea**: Hardness of kernelization can be transferred via reductions, similarly to **NP**-hardness.

Polynomial parameter transformation (PPT)

A **polynomial parameter transformation** from a parameterized problem P to a parameterized problem Q is a polynomial-time algorithm that transforms a given instance (x, k) of P into an equivalent instance (x', k') of Q such that $k' = \mathrm{poly}(k)$.

Observation

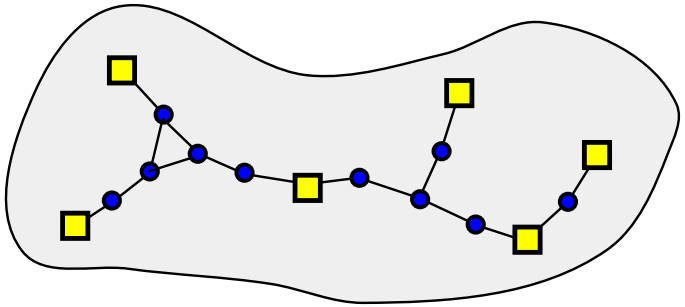
If problem P PPT-reduces to Q, and P does not admit a polynomial compression algorithm (into any language R), then neither does Q.

- **Proof**:
  Compose the PPT with the assumed compression for Q.

Application: Steiner Tree

Steiner Tree

I: Graph G with terminals $T \subseteq V(G)$, $k \in \mathbb{N}$
P: $k + |T|$
Q: Is there a set $X \subseteq V(G) \setminus T$, such that $|X| \leqslant k$ and $G[T \cup X]$ is connected?

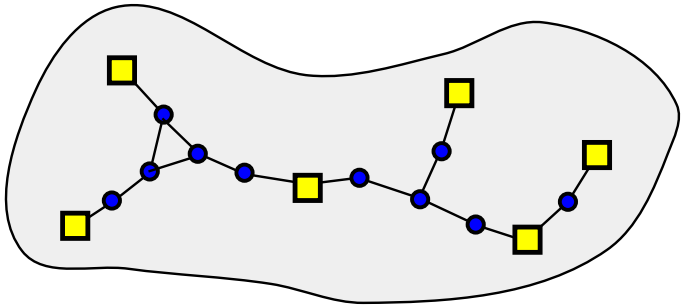


Application: Steiner Tree

Steiner Tree

I: Graph G with terminals $T \subseteq V(G)$, $k \in \mathbb{N}$
P: $k + |T|$
Q: Is there a set $X \subseteq V(G) \setminus T$, such that $|X| \leqslant k$ and $G[T \cup X]$ is connected?

- We show that Steiner Tree has no polykernel (unless...) using a PPT from a auxiliary problem.



The auxiliary problem technique

- Introduce a simpler problem P, which is almost trivially compositional.

The auxiliary problem technique

- Introduce a simpler problem P, which is almost trivially compositional.
- Then design a PPT from P to the target problem.

The auxiliary problem technique

- Introduce a simpler problem P, which is almost trivially compositional.
- Then design a PPT from P to the target problem.
- **Idea**: Move the weight of the proof to the transformation and the actual definition of P.

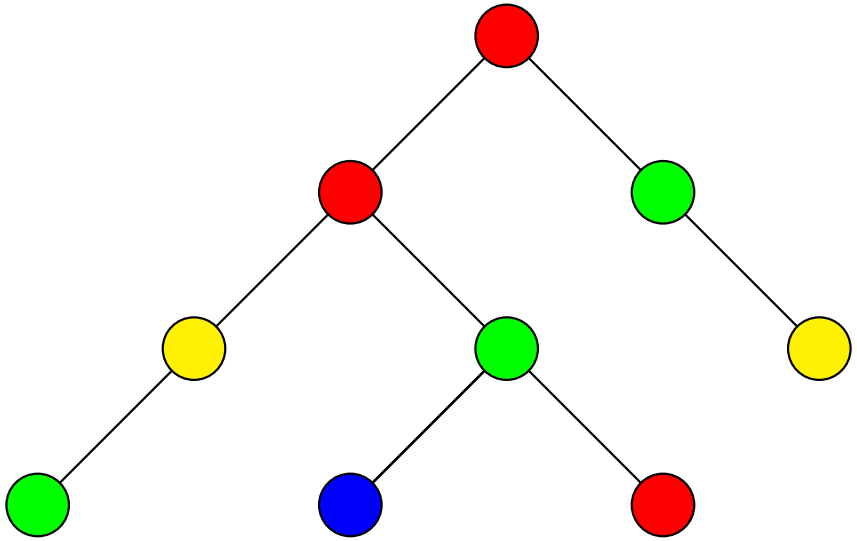
The auxiliary problem technique

- Introduce a simpler problem P, which is almost trivially compositional.
- Then design a PPT from P to the target problem.
- **Idea**: Move the weight of the proof to the transformation and the actual definition of P.
- **High level**: Extract the essence of the original problem into the auxiliary problem.

Colorful Graph Motif

Colorful Graph Motif

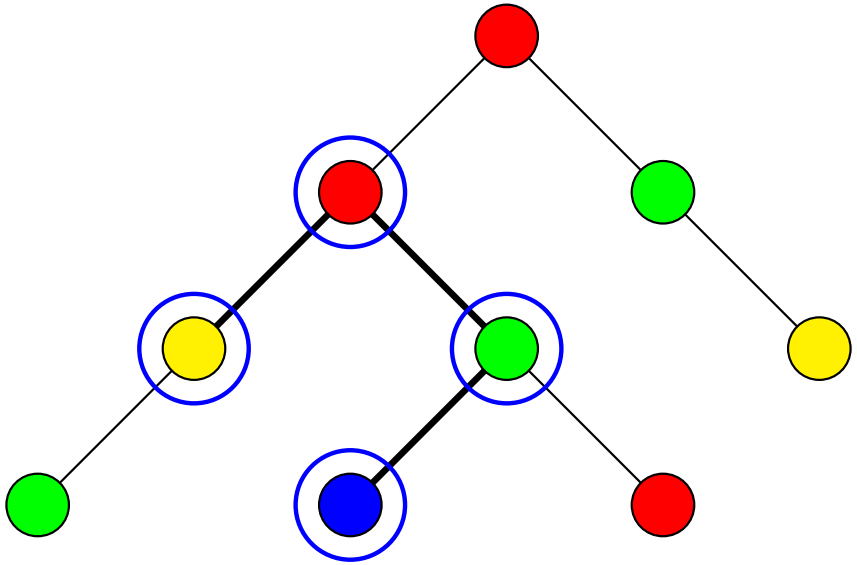
I: Graph G and a coloring function $\phi\colon V(G) \to \{1, 2, \ldots, k\}$
P: k
Q: Does there exists a connected subgraph of G that contains exactly one vertex of each color?



Colorful Graph Motif

Colorful Graph Motif

I: Graph G and a coloring function $\phi\colon V(G) \to \{1, 2, \ldots, k\}$
P: k
Q: Does there exists a connected subgraph of G that contains exactly one vertex of each color?



About CGM

- The problem is **NP**-hard even on trees.

About CGM

- The problem is **NP**-hard even on trees.
- FPT algorithms for various variants using the algebraic approach.

About CGM

- The problem is **NP**-hard even on trees.
- FPT algorithms for various variants using the algebraic approach.
- **Composition**: Take the disjoint union of instances, reuse colors.

About CGM

- The problem is **NP**-hard even on trees.
- FPT algorithms for various variants using the algebraic approach.
- **Composition**: Take the disjoint union of instances, reuse colors.
  - There is a connected colorful motif in the composed instance iff there is one in any of the input instances.

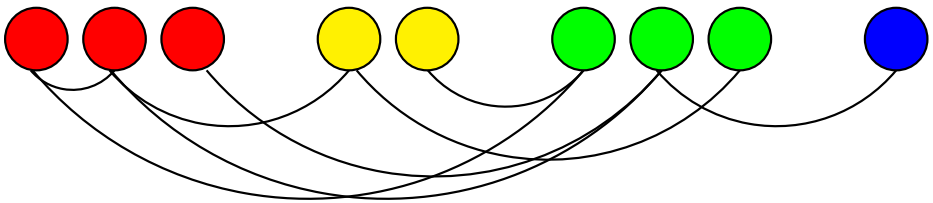
About CGM

- The problem is **NP**-hard even on trees.
- FPT algorithms for various variants using the algebraic approach.
- **Composition**: Take the disjoint union of instances, reuse colors.
  - There is a connected colorful motif in the composed instance iff there is one in any of the input instances.
- **Corollary**: no polykernel for CGM unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

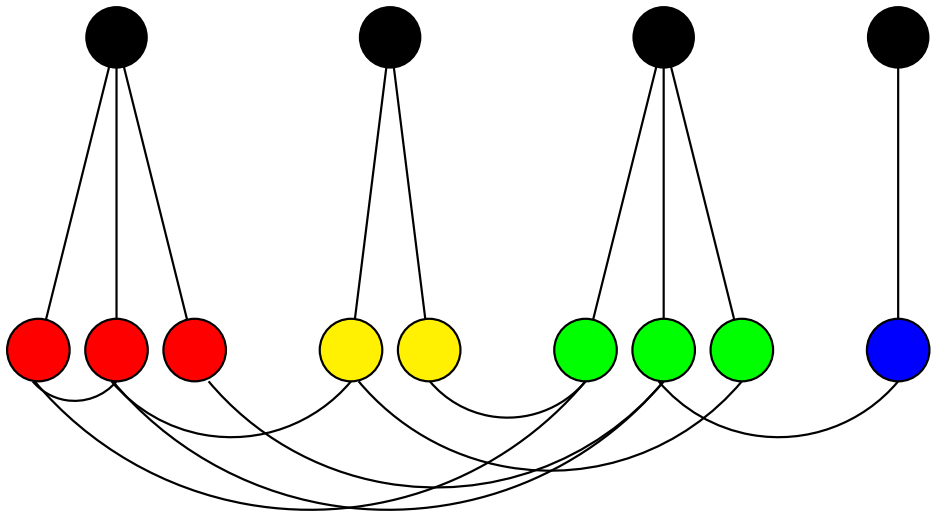
About CGM

- The problem is **NP**-hard even on trees.
- FPT algorithms for various variants using the algebraic approach.
- **Composition**: Take the disjoint union of instances, reuse colors.
  - There is a connected colorful motif in the composed instance iff there is one in any of the input instances.
- **Corollary**: no polykernel for CGM unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
- **Now**: PPT from CGM to ST.

From CGM to ST



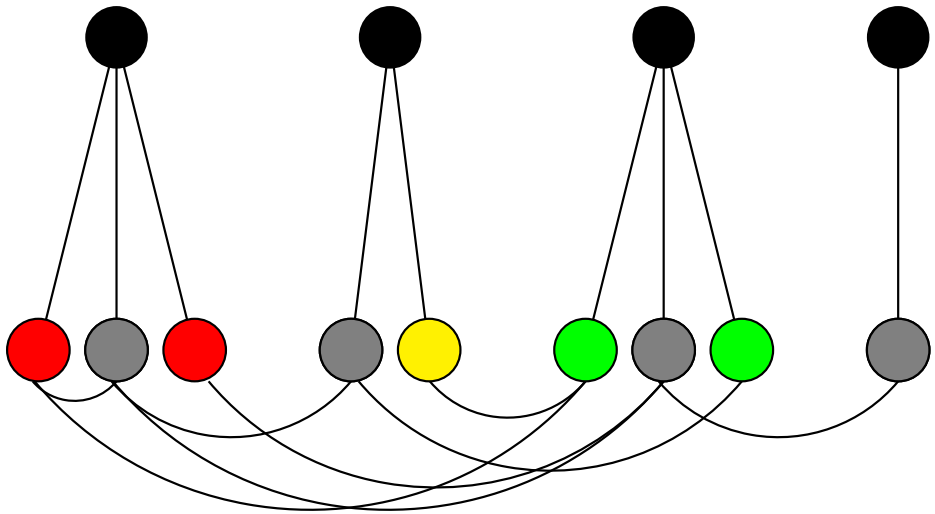
From CGM to ST



Attach a terminal to every color class.

Give budget k for connecting nodes.

From CGM to ST



Attach a terminal to every color class.

Give budget k for connecting nodes.

CGM: wrap up

- CGM has no polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.

CGM: wrap up

- CGM has no polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.
- CGM PPT-reduces to Steiner Tree par. by $k + |T|$.

CGM: wrap up

- CGM has no polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.
- CGM PPT-reduces to Steiner Tree par. by $k + |T|$.
- Hence Steiner Tree par. by $k + |T|$ does not admit a polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.

CGM: wrap up

- CGM has no polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.
- CGM PPT-reduces to Steiner Tree par. by $k + |T|$.
- Hence Steiner Tree par. by $k + |T|$ does not admit a polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.
- **Note**: Composition for CGM is far simpler than trying to do this directly for Steiner Tree.

Another example: Disjoint Paths

Disjoint Paths

I: Graph G with k terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$

P: k

Q: Is there a **linkage** consisting of vertex-disjoint paths $P_1, \ldots, P_k$, where P_i connects s_i with t_i?

Another example: Disjoint Paths

Disjoint Paths

I: Graph G with k terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$
P: k
Q: Is there a **linkage** consisting of vertex-disjoint paths $P_1, \ldots, P_k$, where P_i connects s_i with t_i?

- Disjoint Paths is FPT (Graph Minors).

Another example: Disjoint Paths

Disjoint Paths

I: Graph G with k terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$
P: k
Q: Is there a **linkage** consisting of vertex-disjoint paths $P_1, \ldots, P_k$, where P_i connects s_i with t_i?

- Disjoint Paths is FPT (Graph Minors).
- **Now**: Disjoint Paths do not have a polynomial kernel (unless...).

Another example: Disjoint Paths

Disjoint Paths

I: Graph G with k terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$
P: k
Q: Is there a **linkage** consisting of vertex-disjoint paths $P_1, \ldots, P_k$, where P_i connects s_i with t_i?

- Disjoint Paths is FPT (Graph Minors).
- **Now**: Disjoint Paths do not have a polynomial kernel (unless...).
- Example due to (Bodlaender, Thomassé, Yeo; 2009).

Another example: Disjoint Paths

Disjoint Paths

I: Graph G with k terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$
P: k
Q: Is there a **linkage** consisting of vertex-disjoint paths $P_1, \ldots, P_k$, where P_i connects s_i with t_i?

- Disjoint Paths is FPT (Graph Minors).
- **Now**: Disjoint Paths do not have a polynomial kernel (unless...).
- Example due to (Bodlaender, Thomassé, Yeo; 2009).

Disjoint Factors

I: A word w over an alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$
P: $k = |\Sigma|$
Q: Can one find **disjoint** subwords $u_1, \ldots, u_k$ of w such that each u_i starts and ends with a_i?

$$b\,a\,c\,d\,b\,d\,c\,b\,a\,c\,b\,a\,d\,c\,a\,b\,c\,d\,b\,a\,d\,c$$

Another example: Disjoint Paths

Disjoint Paths

I: Graph G with k terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$
P: k
Q: Is there a **linkage** consisting of vertex-disjoint paths $P_1, \ldots, P_k$, where P_i connects s_i with t_i?

- Disjoint Paths is FPT (Graph Minors).
- **Now**: Disjoint Paths do not have a polynomial kernel (unless...).
- Example due to (Bodlaender, Thomassé, Yeo; 2009).

Disjoint Factors

I: A word w over an alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$
P: $k = |\Sigma|$
Q: Can one find **disjoint** subwords $u_1, \ldots, u_k$ of w such that each u_i starts and ends with a_i?

b a c d b d c b a c b a d c a b c d b a d c

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.
  - W.l.o.g. $t = 2^\ell$ for some integer $\ell = \log_2 t$.

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.
  - W.l.o.g. $t = 2^\ell$ for some integer $\ell = \log_2 t$.
- Define $\Gamma = \Sigma \cup \{d_1, d_2, \ldots, d_\ell\}$, where d_is are fresh symbols.

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.
  - W.l.o.g. $t = 2^\ell$ for some integer $\ell = \log_2 t$.
- Define $\Gamma = \Sigma \cup \{d_1, d_2, \ldots, d_\ell\}$, where d_is are fresh symbols.
- From $w_1, w_2, \ldots, w_t$ construct one word w over Γ as follows:

w_1 w_2 w_3 w_4 w_5 w_6 w_7 w_8

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.
  - W.l.o.g. $t = 2^\ell$ for some integer $\ell = \log_2 t$.
- Define $\Gamma = \Sigma \cup \{d_1, d_2, \ldots, d_\ell\}$, where d_is are fresh symbols.
- From $w_1, w_2, \ldots, w_t$ construct one word w over Γ as follows:

d_1 w_1 w_2 w_3 w_4 d_1 w_5 w_6 w_7 w_8 d_1

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.
  - W.l.o.g. $t = 2^\ell$ for some integer $\ell = \log_2 t$.
- Define $\Gamma = \Sigma \cup \{d_1, d_2, \ldots, d_\ell\}$, where d_is are fresh symbols.
- From $w_1, w_2, \ldots, w_t$ construct one word w over Γ as follows:

d_1 d_2 w_1 w_2 d_2 w_3 w_4 d_2 d_1 d_2 w_5 w_6 d_2 w_7 w_8 d_2 d_1

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.
  - W.l.o.g. $t = 2^\ell$ for some integer $\ell = \log_2 t$.
- Define $\Gamma = \Sigma \cup \{d_1, d_2, \ldots, d_\ell\}$, where d_is are fresh symbols.
- From $w_1, w_2, \ldots, w_t$ construct one word w over Γ as follows:

$$d_1\ d_2\ d_3\ w_1\ d_3\ w_2\ d_3\ d_2\ d_3\ w_3\ d_3\ w_4\ d_3\ d_2\ d_1\ d_2\ d_3\ w_5\ d_3\ w_6\ d_3\ d_2\ d_3\ w_7\ d_3\ w_8\ d_3\ d_2\ d_1$$

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.
  - W.l.o.g. $t = 2^\ell$ for some integer $\ell = \log_2 t$.
- Define $\Gamma = \Sigma \cup \{d_1, d_2, \ldots, d_\ell\}$, where d_is are fresh symbols.
- From $w_1, w_2, \ldots, w_t$ construct one word w over Γ as follows:

$$d_1\ d_2\ d_3\ w_1\ d_3\ w_2\ d_3\ d_2\ d_3\ w_3\ d_3\ w_4\ d_3\ d_2\ d_1\ d_2\ d_3\ w_5\ d_3\ w_6\ d_3\ d_2\ d_3\ w_7\ d_3\ w_8\ d_3\ d_2\ d_1$$

- It is clear that w admits disjoint factors iff any of w_is does, so this constitutes a cross-composition.

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.
  - W.l.o.g. $t = 2^\ell$ for some integer $\ell = \log_2 t$.
- Define $\Gamma = \Sigma \cup \{d_1, d_2, \ldots, d_\ell\}$, where d_is are fresh symbols.
- From $w_1, w_2, \ldots, w_t$ construct one word w over Γ as follows:

$$d_1\ d_2\ d_3\ w_1\ d_3\ w_2\ d_3\ d_2\ d_3\ w_3\ d_3\ w_4\ d_3\ d_2\ d_1\ d_2\ d_3\ w_5\ d_3\ w_6\ d_3\ d_2\ d_3\ w_7\ d_3\ w_8\ d_3\ d_2\ d_1$$

- It is clear that w admits disjoint factors iff any of w_is does, so this constitutes a cross-composition.

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.
  - W.l.o.g. $t = 2^\ell$ for some integer $\ell = \log_2 t$.
- Define $\Gamma = \Sigma \cup \{d_1, d_2, \ldots, d_\ell\}$, where d_is are fresh symbols.
- From $w_1, w_2, \ldots, w_t$ construct one word w over Γ as follows:

$$d_1\ d_2\ d_3\ w_1\ d_3\ w_2\ d_3\ d_2\ d_3\ w_3\ d_3\ w_4\ d_3\ d_2\ d_1\ d_2\ d_3\ w_5\ d_3\ w_6\ d_3\ d_2\ d_3\ w_7\ d_3\ w_8\ d_3\ d_2\ d_1$$

- It is clear that w admits disjoint factors iff any of w_is does, so this constitutes a cross-composition.
- $|\Gamma| = |\Sigma| + \log_2 t$.

Cross-composition for Disjoint Factors

- Disjoint Factors is **NP**-hard.
- We give a cross-composition from the unparameterized variant of DF to the parameterized.
- **Input**: Words $w_1, w_2, \ldots, w_t$ over the same alphabet Σ.
  - W.l.o.g. $t = 2^\ell$ for some integer $\ell = \log_2 t$.
- Define $\Gamma = \Sigma \cup \{d_1, d_2, \ldots, d_\ell\}$, where d_is are fresh symbols.
- From $w_1, w_2, \ldots, w_t$ construct one word w over Γ as follows:

$$d_1\ d_2\ d_3\ w_1\ d_3\ w_2\ d_3\ d_2\ d_3\ w_3\ d_3\ w_4\ d_3\ d_2\ d_1\ d_2\ d_3\ w_5\ d_3\ w_6\ d_3\ d_2\ d_3\ w_7\ d_3\ w_8\ d_3\ d_2\ d_1$$

- It is clear that w admits disjoint factors iff any of w_is does, so this constitutes a cross-composition.
- $|\Gamma| = |\Sigma| + \log_2 t$.
- **Cor**: Disjoint Factors does not admit a polynomial kernel (unless...).

From Disjoint Factors to Disjoint Paths

- We now give a PPT from Disjoint Factors to Disjoint Paths.

From Disjoint Factors to Disjoint Paths

- We now give a PPT from Disjoint Factors to Disjoint Paths.
- We start with the input word w over Σ, say

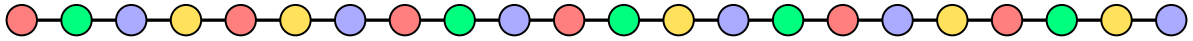
$$b\,a\,c\,d\,b\,d\,c\,b\,a\,c\,b\,a\,d\,c\,a\,b\,c\,d\,b\,a\,d\,c$$

From Disjoint Factors to Disjoint Paths

- We now give a PPT from Disjoint Factors to Disjoint Paths.
- We start with the input word w over Σ, say

$$b\,a\,c\,d\,b\,d\,c\,b\,a\,c\,b\,a\,d\,c\,a\,b\,c\,d\,b\,a\,d\,c$$

- Create a path of length $|w|$.

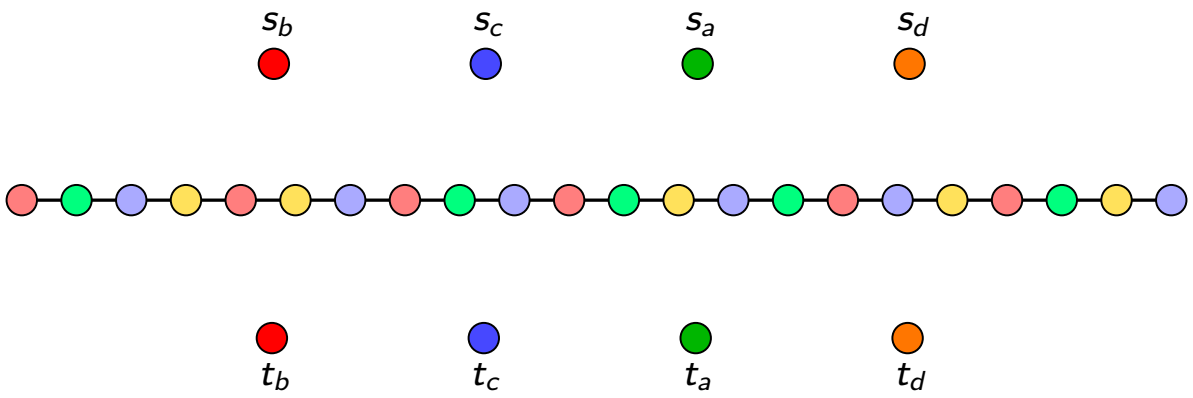


From Disjoint Factors to Disjoint Paths

- We now give a PPT from Disjoint Factors to Disjoint Paths.
- We start with the input word w over Σ, say

$$b\,a\,c\,d\,b\,d\,c\,b\,a\,c\,b\,a\,d\,c\,a\,b\,c\,d\,b\,a\,d\,c$$

- Create a path of length $|w|$.
- For each $\sigma \in \Sigma$, create a terminal pair (s_σ, t_σ).

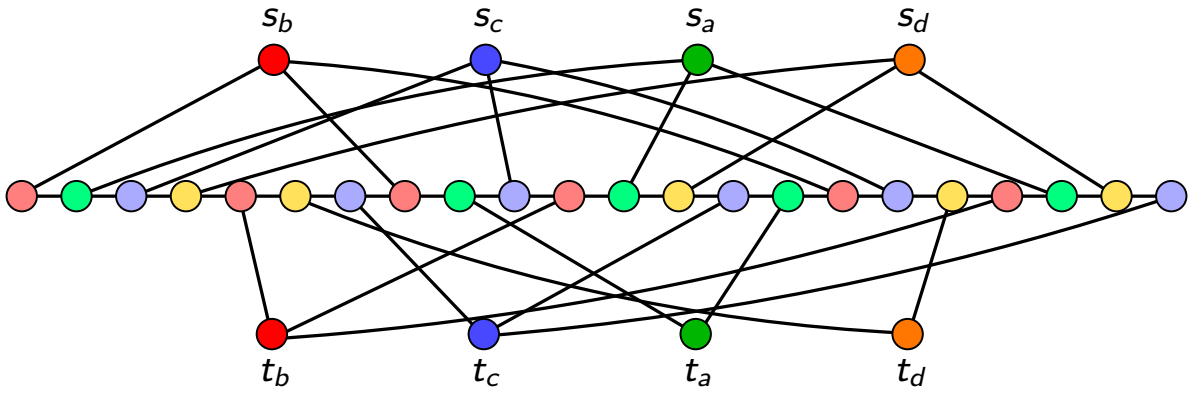


From Disjoint Factors to Disjoint Paths

- We now give a PPT from Disjoint Factors to Disjoint Paths.
- We start with the input word w over Σ, say

$$b\,a\,c\,d\,b\,d\,c\,b\,a\,c\,b\,a\,d\,c\,a\,b\,c\,d\,b\,a\,d\,c$$

- Create a path of length $|w|$.
- For each $\sigma \in \Sigma$, create a terminal pair (s_σ, t_σ).
- Enumerate occurences of σ in w;
  connect s_σ to odd-numbered and t_σ to even-numbered.

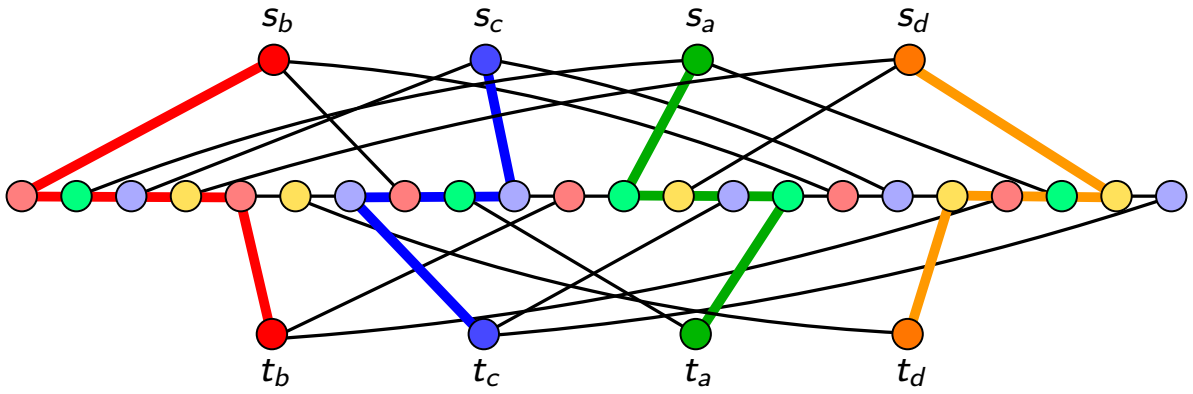


From Disjoint Factors to Disjoint Paths

- We now give a PPT from Disjoint Factors to Disjoint Paths.
- We start with the input word w over Σ, say

$$\underline{b\,a\,c\,d\,b}\;d\;\underline{c\,b\,a\,c}\;b\;\underline{a\,d\,c\,a}\;b\,c\;\underline{d\,b\,a\,d}\;c$$

- Create a path of length $|w|$.
- For each $\sigma \in \Sigma$, create a terminal pair (s_σ, t_σ).
- Enumerate occurences of σ in w;
  connect s_σ to odd-numbered and t_σ to even-numbered.
- There is a linkage iff there are disjoint factors in w.



AND-compositions

- In the compositionality framework, we used the OR function to compose instances.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.
  - Example of problem admitting an AND-composition:
    Is the treewidth of a given graph at most k?

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.
  - Example of problem admitting an AND-composition:
    Is the treewidth of a given graph at most k?
- **AND-conjecture**:
  If $3\mathrm{SAT}$ has an AND-distillation, then **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.
  - Example of problem admitting an AND-composition:
    Is the treewidth of a given graph at most k?
- **AND-conjecture**:
  If $3\mathrm{SAT}$ has an AND-distillation, then **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
  - The proof of Fortnow and Santhanam fails for AND.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.
  - Example of problem admitting an AND-composition:
    Is the treewidth of a given graph at most k?
- **AND-conjecture**:
  If $3\mathrm{SAT}$ has an AND-distillation, then **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
  - The proof of Fortnow and Santhanam fails for AND.
  - The conjecture was proved by Drucker in 2012.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.
  - Example of problem admitting an AND-composition:
    Is the treewidth of a given graph at most k?
- **AND-conjecture**:
  If $3\mathrm{SAT}$ has an AND-distillation, then **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
  - The proof of Fortnow and Santhanam fails for AND.
  - The conjecture was proved by Drucker in 2012.
- **Corollary**: The whole framework works for AND instead of OR.

Weak compositions

- In the morning we have seen a kernel for Vertex Cover with $2k$ vertices, which means bitsize $\mathcal{O}(k^2)$.

Weak compositions

- In the morning we have seen a kernel for Vertex Cover with $2k$ vertices, which means bitsize $\mathcal{O}(k^2)$.
- Can we prove that the quadratic bitsize is optimal?

Weak compositions

- In the morning we have seen a kernel for Vertex Cover with $2k$ vertices, which means bitsize $\mathcal{O}(k^2)$.
- Can we prove that the quadratic bitsize is optimal?
- Turns out that **yes**.

Weak compositions

- In the morning we have seen a kernel for Vertex Cover with $2k$ vertices, which means bitsize $\mathcal{O}(k^2)$.
- Can we prove that the quadratic bitsize is optimal?
- Turns out that **yes**.
- **Weak compositions**: technique for lower bounds on kernelization complexity for problems that do have polynomial kernels.

Weak compositions

- In the morning we have seen a kernel for Vertex Cover with $2k$ vertices, which means bitsize $\mathcal{O}(k^2)$.
- Can we prove that the quadratic bitsize is optimal?
- Turns out that **yes**.
- **Weak compositions**: technique for lower bounds on kernelization complexity for problems that do have polynomial kernels.
- First observed by Dell and van Melkebeek in 2010, then developed by other authors as well.

Back to Fortnow and Santhanam

- OR-distillation of $t = k^{1000}$ instances of size k into one instance with bitsize k^7 is unlikely.

Back to Fortnow and Santhanam

- OR-distillation of $t = k^{1000}$ instances of size k into one instance with bitsize k^7 is unlikely.
- It should be unlikely even if we required any bitsize sublinear in t.

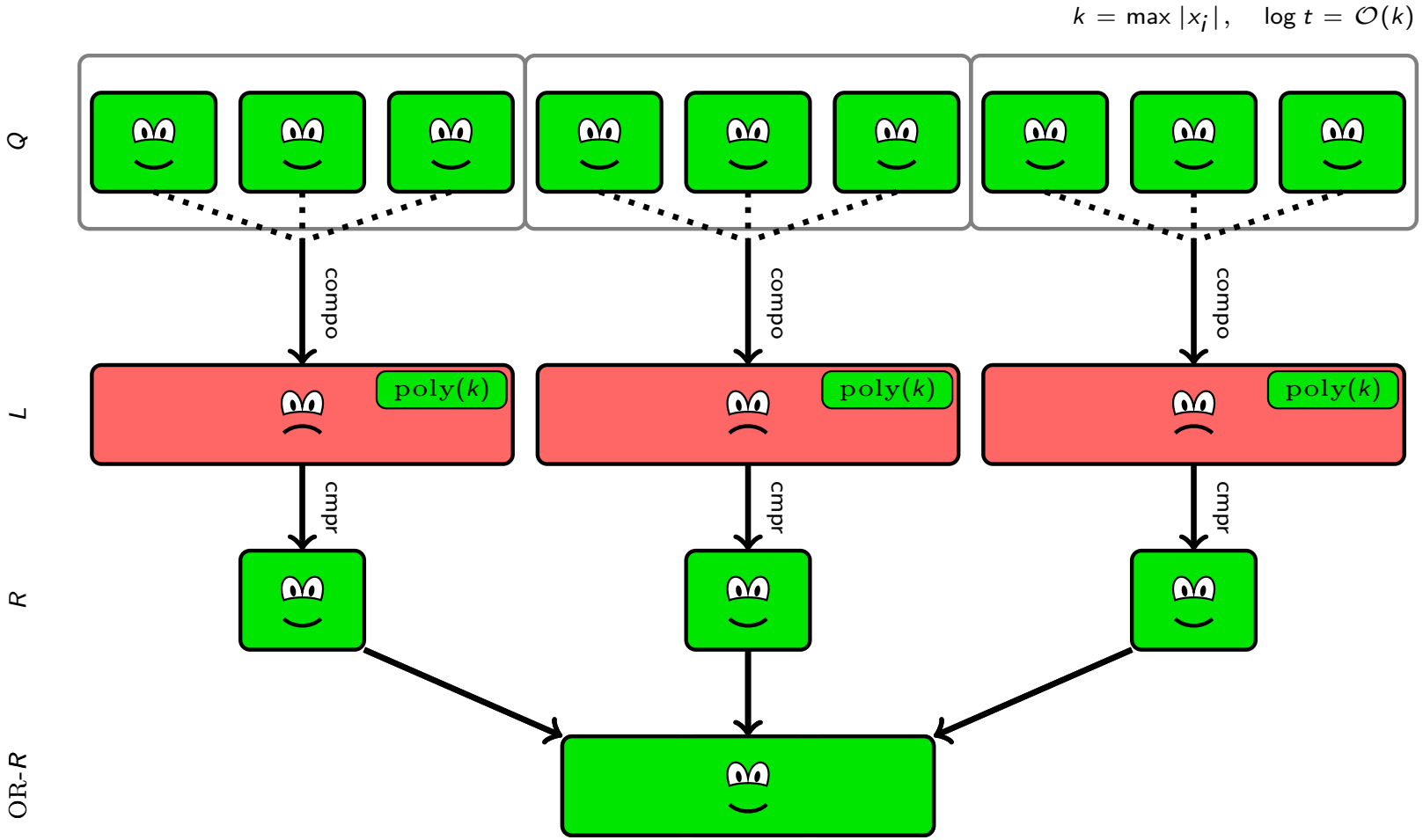
Back to Fortnow and Santhanam

- OR-distillation of $t = k^{1000}$ instances of size k into one instance with bitsize k^7 is unlikely.
- It should be unlikely even if we required any bitsize sublinear in t.
- **Examination of the proof**:
  one can exclude OR-distillation that takes t instances $x_1, \ldots, x_t$ with $|x_i| \leqslant k$, and OR-distills them into bitsize $\mathcal{O}(t^{1-\epsilon} \cdot k^c)$, for any constants $\epsilon > 0$ and c.

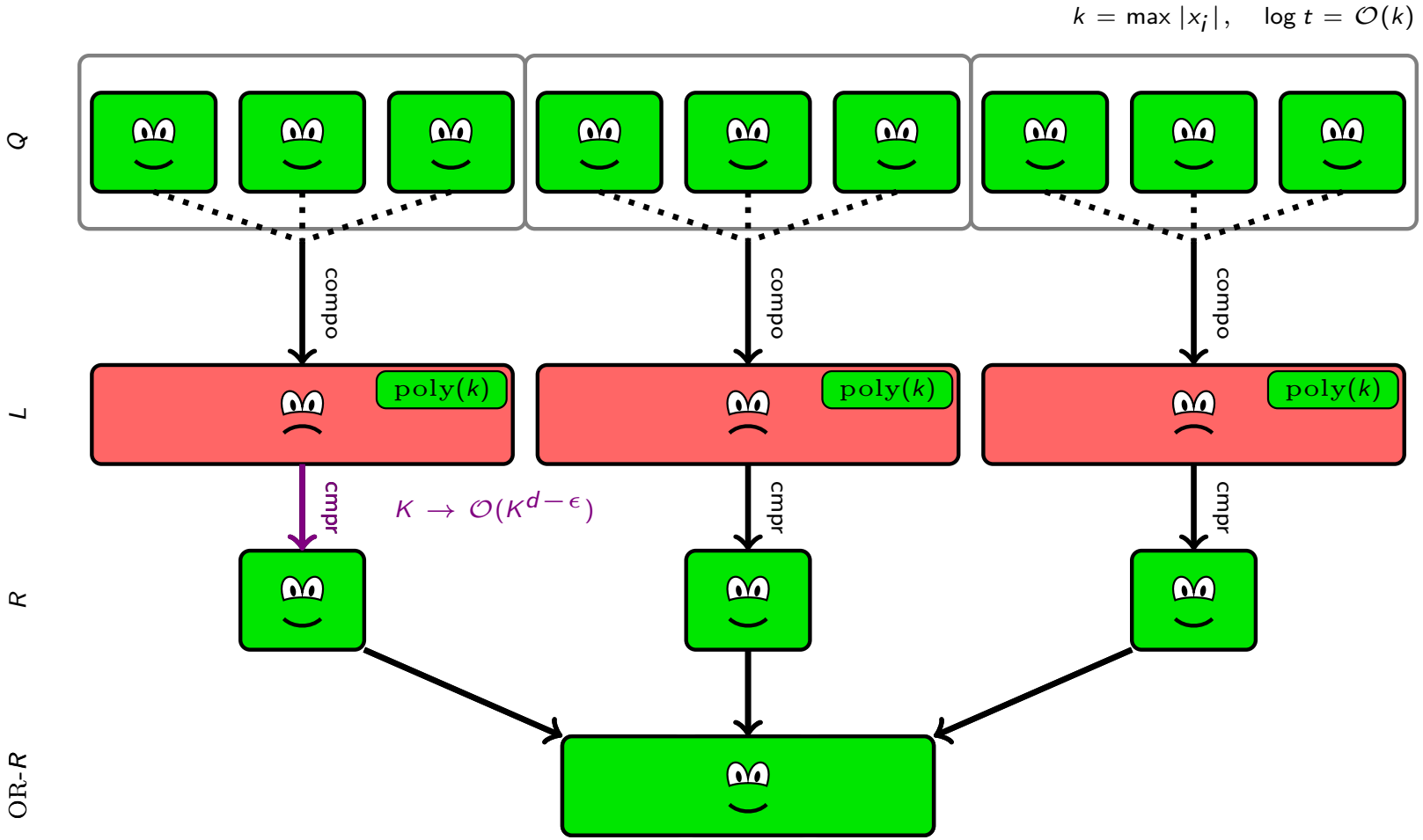
Back to Fortnow and Santhanam

- OR-distillation of $t = k^{1000}$ instances of size k into one instance with bitsize k^7 is unlikely.
- It should be unlikely even if we required any bitsize sublinear in t.
- **Examination of the proof**:
  one can exclude OR-distillation that takes t instances $x_1, \ldots, x_t$ with $|x_i| \leqslant k$, and OR-distills them into bitsize $\mathcal{O}(t^{1-\epsilon} \cdot k^c)$, for any constants $\epsilon > 0$ and c.
- Let's look again at the proof of the cross-composition theorem.

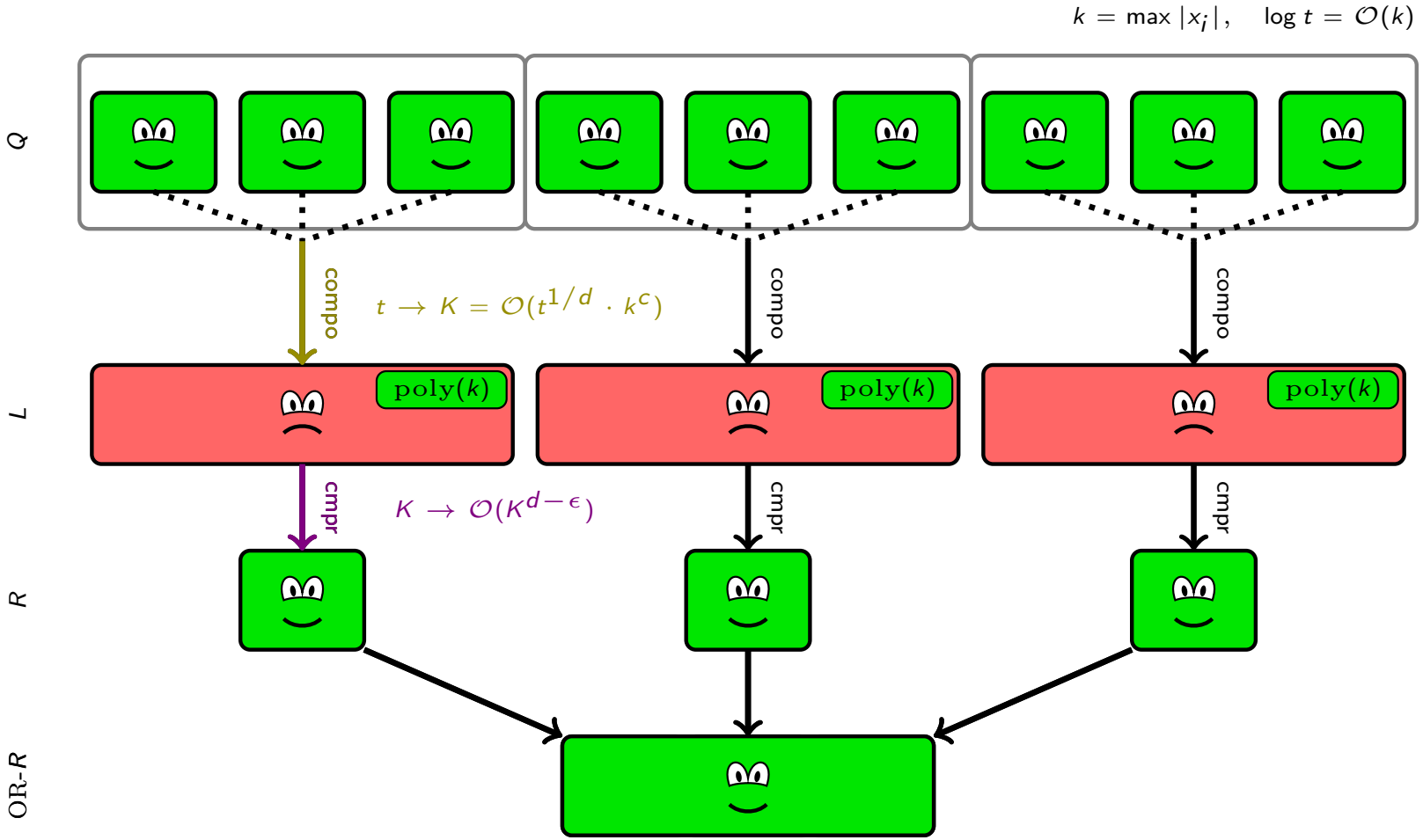
Cross-composition proof, recap



Cross-composition proof, recap



Cross-composition proof, recap



Weak compositions, formally

- **Idea**: A composition with output bitsize dependence on t being $\mathcal{O}(t^{1/d})$ excludes compression into bitsize $\mathcal{O}(k^{d-\epsilon})$.

Weak compositions, formally

- **Idea**: A composition with output bitsize dependence on t being $\mathcal{O}(t^{1/d})$ excludes compression into bitsize $\mathcal{O}(k^{d-\epsilon})$.

Weak cross-composition

An unparameterized problem Q **weakly cross-composes** into a parameterized problem L, if there exists a polynomial equivalence relation $\sim$, a real constant $d \geqslant 1$, and an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$, in time $\mathrm{poly}\left(t + \sum_{i=1}^{t} |x_i|\right)$ produces one instance $(y, k^\star)$ such that

- $(y, k^\star) \in L$ iff $x_i \in Q$ for at least one $i = 1, 2, \ldots, t$,
- $k^\star = t^{1/d+o(1)} \cdot \mathrm{poly}\left(\max_{i=1}^{t} |x_i|\right)$.

Weak compositions, formally

- **Idea**: A composition with output bitsize dependence on t being $\mathcal{O}(t^{1/d})$ excludes compression into bitsize $\mathcal{O}(k^{d-\epsilon})$.

Weak cross-composition

An unparameterized problem Q **weakly cross-composes** into a parameterized problem L, if there exists a polynomial equivalence relation $\sim$, a real constant $d \geqslant 1$, and an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$, in time $\mathrm{poly}\left(t + \sum_{i=1}^{t} |x_i|\right)$ produces one instance $(y, k^\star)$ such that

- $(y, k^\star) \in L$ iff $x_i \in Q$ for at least one $i = 1, 2, \ldots, t$,
- $k^\star = t^{1/d+o(1)} \cdot \mathrm{poly}\left(\max_{i=1}^{t} |x_i|\right)$.

- Constant d will be called the **dimension** of the weak cross-composition.

Weak cross-composition theorem

Weak cross-composition theorem

Suppose an **NP**-hard problem Q admits a weak cross-composition into L with dimension d, and that L admits a polynomial compression with bitsize $\mathcal{O}(k^{d-\epsilon})$, for some $\epsilon > 0$. Then $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

Weak cross-composition theorem

Weak cross-composition theorem

Suppose an **NP**-hard problem Q admits a weak cross-composition into L with dimension d, and that L admits a polynomial compression with bitsize $\mathcal{O}(k^{d-\epsilon})$, for some $\epsilon > 0$. Then $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

- Also called *cross-composition of bounded cost* by Bodlaender et al.

Weak cross-composition theorem

Weak cross-composition theorem

Suppose an **NP**-hard problem Q admits a weak cross-composition into L with dimension d, and that L admits a polynomial compression with bitsize $\mathcal{O}(k^{d-\epsilon})$, for some $\epsilon > 0$. Then $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

- Also called *cross-composition of bounded cost* by Bodlaender et al.
- Using a weak cross-composition we now prove that Vertex Cover does not admit a kernel with bitsize $\mathcal{O}(k^{2-\epsilon})$, for any $\epsilon > 0$ (unless...).

Weak cross-composition theorem

Weak cross-composition theorem

Suppose an **NP**-hard problem Q admits a weak cross-composition into L with dimension d, and that L admits a polynomial compression with bitsize $\mathcal{O}(k^{d-\epsilon})$, for some $\epsilon > 0$. Then **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

- Also called *cross-composition of bounded cost* by Bodlaender et al.
- Using a weak cross-composition we now prove that Vertex Cover does not admit a kernel with bitsize $\mathcal{O}(k^{2-\epsilon})$, for any $\epsilon > 0$ (unless...).
  - **Note**: We do have a kernel with at most $2k$ vertices, but it needs $\mathcal{O}(k^2)$ bits to encode.

Weak cross-composition theorem

Weak cross-composition theorem

Suppose an **NP**-hard problem Q admits a weak cross-composition into L with dimension d, and that L admits a polynomial compression with bitsize $\mathcal{O}(k^{d-\epsilon})$, for some $\epsilon > 0$. Then **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

- Also called *cross-composition of bounded cost* by Bodlaender et al.
- Using a weak cross-composition we now prove that Vertex Cover does not admit a kernel with bitsize $\mathcal{O}(k^{2-\epsilon})$, for any $\epsilon > 0$ (unless...).
  - **Note**: We do have a kernel with at most $2k$ vertices, but it needs $\mathcal{O}(k^2)$ bits to encode.
- **Crux**: choose an appropriate problem Q to start with.

Weak cross-composition theorem

Weak cross-composition theorem

Suppose an **NP**-hard problem Q admits a weak cross-composition into L with dimension d, and that L admits a polynomial compression with bitsize $\mathcal{O}(k^{d-\epsilon})$, for some $\epsilon > 0$. Then **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

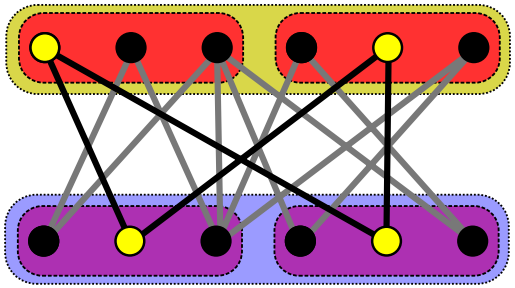
- Also called *cross-composition of bounded cost* by Bodlaender et al.
- Using a weak cross-composition we now prove that Vertex Cover does not admit a kernel with bitsize $\mathcal{O}(k^{2-\epsilon})$, for any $\epsilon > 0$ (unless...).
  - **Note**: We do have a kernel with at most $2k$ vertices, but it needs $\mathcal{O}(k^2)$ bits to encode.
- **Crux**: choose an appropriate problem Q to start with.
- Example due to (Dell, Marx; 2012).

Multicolored Biclique

Multicolored Biclique

Input: Bipartite graph H with bipartition $A \uplus B$, where $A = A_1 \uplus \ldots \uplus A_k$, $B = B_1 \uplus \ldots \uplus B_k$.

Question: Is there a biclique of size $K_{k,k}$ in H that contains one vertex from each A_i and each B_i?



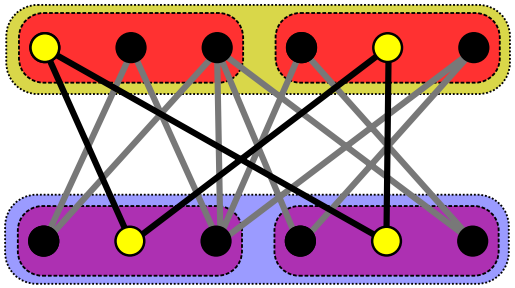
Multicolored Biclique

Multicolored Biclique

Input: Bipartite graph H with bipartition $A \uplus B$, where $A = A_1 \uplus \ldots \uplus A_k$, $B = B_1 \uplus \ldots \uplus B_k$.

Question: Is there a biclique of size $K_{k,k}$ in H that contains one vertex from each A_i and each B_i?

- **NP**-hard even if each A_i and B_i has the same size.



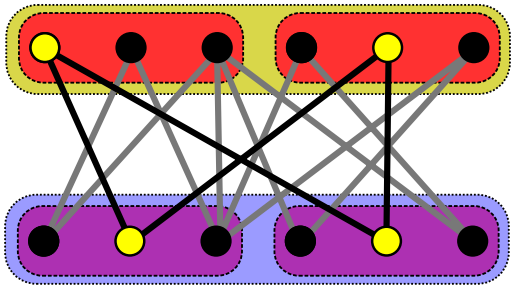
Multicolored Biclique

Multicolored Biclique

Input: Bipartite graph H with bipartition $A \uplus B$, where $A = A_1 \uplus \ldots \uplus A_k$, $B = B_1 \uplus \ldots \uplus B_k$.

Question: Is there a biclique of size $K_{k,k}$ in H that contains one vertex from each A_i and each B_i?

- **NP**-hard even if each A_i and B_i has the same size.
- We give a weak cross composition of dimension 2 from this variant into Vertex Cover.



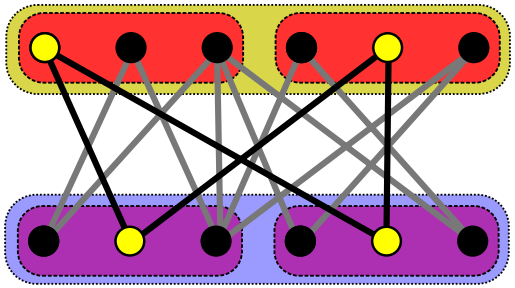
Multicolored Biclique

Multicolored Biclique

Input: Bipartite graph H with bipartition $A \uplus B$, where $A = A_1 \uplus \ldots \uplus A_k$, $B = B_1 \uplus \ldots \uplus B_k$.

Question: Is there a biclique of size $K_{k,k}$ in H that contains one vertex from each A_i and each B_i?

- **NP**-hard even if each A_i and B_i has the same size.
- We give a weak cross composition of dimension 2 from this variant into Vertex Cover.
- **Assumptions**: all the input instances have the same k, each color class has size n in every input instance, $t = s^2$.



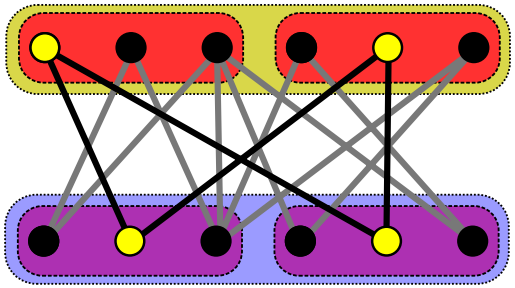
Multicolored Biclique

Multicolored Biclique

Input: Bipartite graph H with bipartition $A \uplus B$, where $A = A_1 \uplus \ldots \uplus A_k$, $B = B_1 \uplus \ldots \uplus B_k$.

Question: Is there a biclique of size $K_{k,k}$ in H that contains one vertex from each A_i and each B_i?

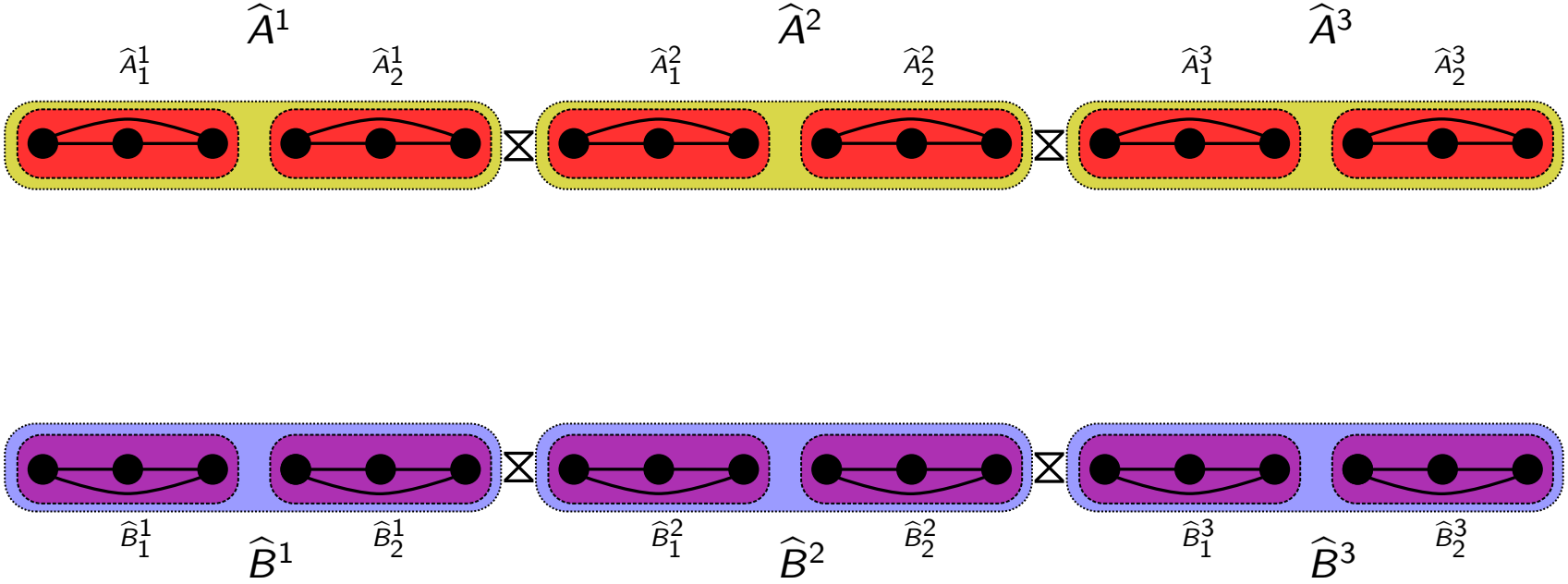
- **NP**-hard even if each A_i and B_i has the same size.
- We give a weak cross composition of dimension 2 from this variant into Vertex Cover.
- **Assumptions**: all the input instances have the same k, each color class has size n in every input instance, $t = s^2$.
  - Thus all instances are on the same vertex set, partitioned in the same way; they are just given as different edge sets $E_1, \ldots, E_t$.



Composition

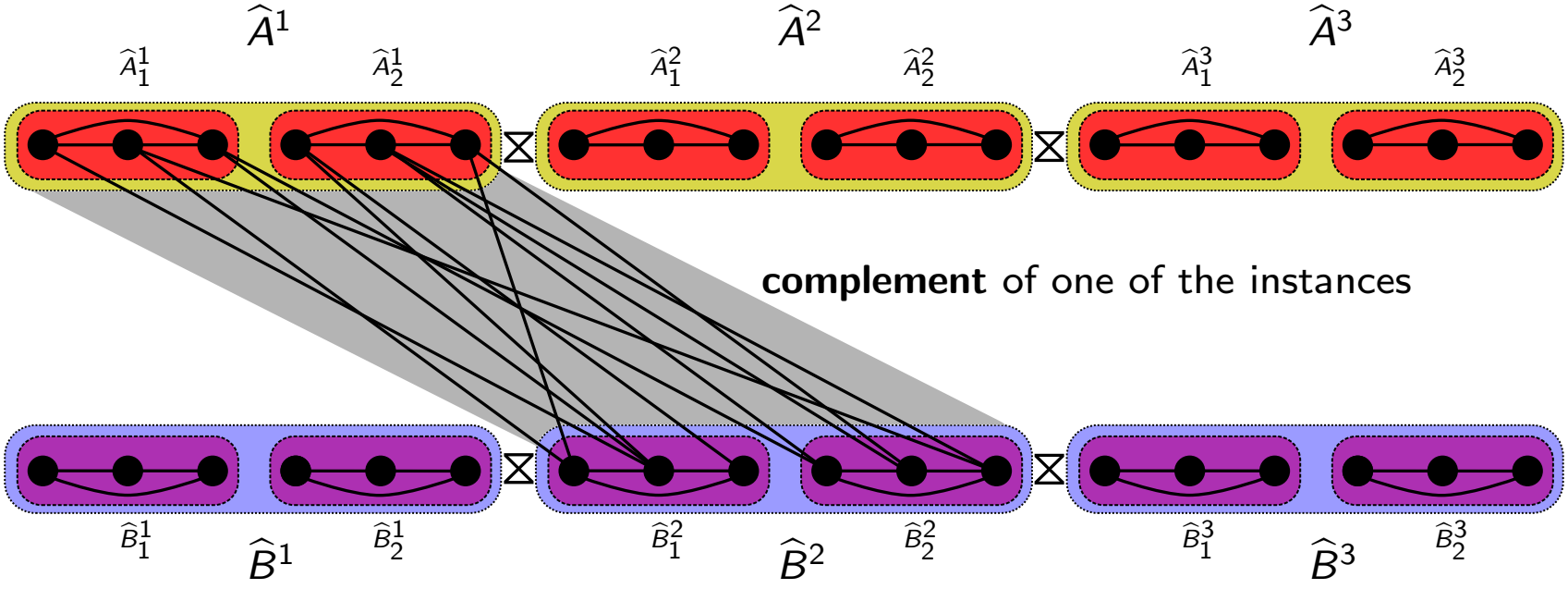
Create s copies of the left side and s copies of the right side.

$N = s \cdot 2kn \leqslant t^{1/2} \cdot \mathrm{poly}(\max_i |G_i|)$



Composition

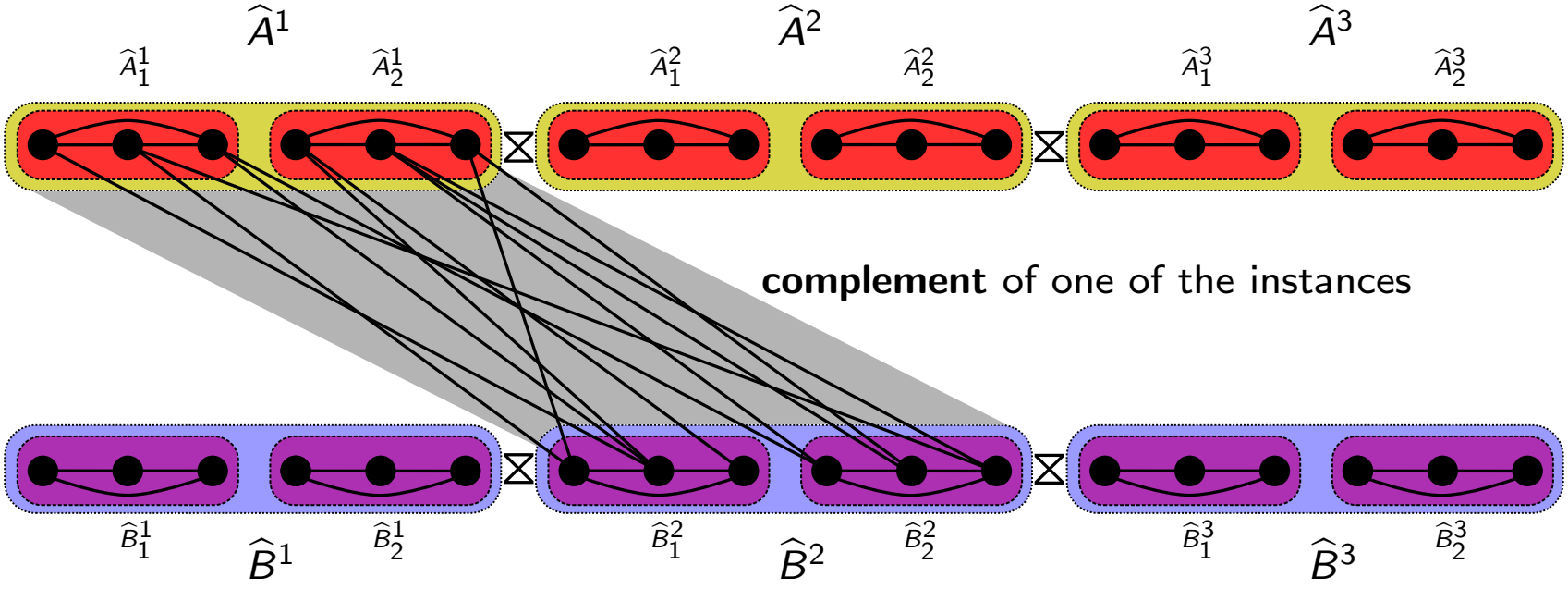
Embed s^2 instances into s^2 pairs of the sides.



Composition

Ask for an independent set of size $2k$;

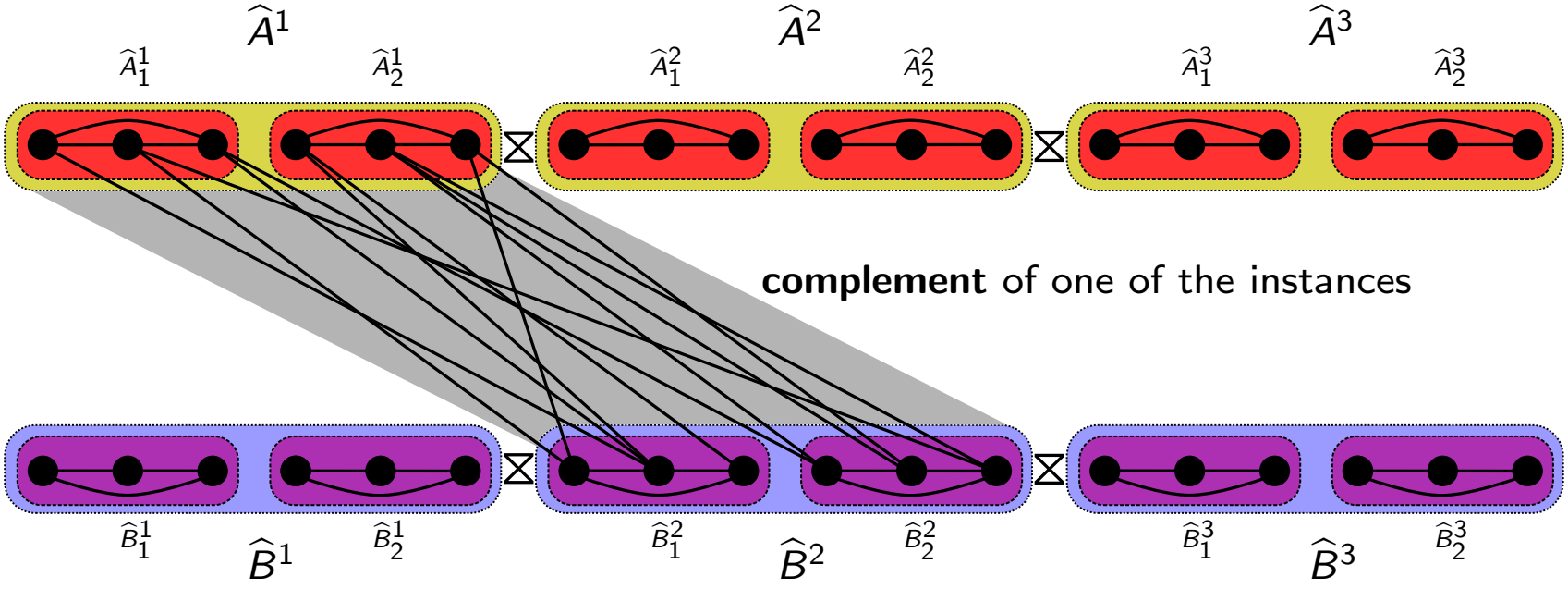
equivalently, a vertex cover of size $N - 2k$.



Composition

Edges on the sides ensure choosing one instance and respecting colors.

Edges originating in this instance ensure that the instance is solved.



Wrapping up

- Parameter in the output instance is $N - 2k = t^{1/2} \cdot 2kn - 2k$, which means that the weak composition has dimension 2.

Wrapping up

- Parameter in the output instance is $N - 2k = t^{1/2} \cdot 2kn - 2k$, which means that the weak composition has dimension 2.
- Hence, there is no kernel with $\mathcal{O}(k^{2-\epsilon})$ bits for VC.

Wrapping up

- Parameter in the output instance is $N - 2k = t^{1/2} \cdot 2kn - 2k$, which means that the weak composition has dimension 2.
- Hence, there is no kernel with $\mathcal{O}(k^{2-\epsilon})$ bits for VC.
- Reduction from VC to FVS: add a degree-2 vertex to every edge, thus creating a triangle.

Wrapping up

- Parameter in the output instance is $N - 2k = t^{1/2} \cdot 2kn - 2k$, which means that the weak composition has dimension 2.
- Hence, there is no kernel with $\mathcal{O}(k^{2-\epsilon})$ bits for VC.
- Reduction from VC to FVS: add a degree-2 vertex to every edge, thus creating a triangle.
- **Cor**: FVS does not admit a $\mathcal{O}(k^{2-\epsilon})$ kernel.

Wrapping up

- Parameter in the output instance is $N - 2k = t^{1/2} \cdot 2kn - 2k$, which means that the weak composition has dimension 2.
- Hence, there is no kernel with $\mathcal{O}(k^{2-\epsilon})$ bits for VC.
- Reduction from VC to FVS: add a degree-2 vertex to every edge, thus creating a triangle.
- **Cor**: FVS does not admit a $\mathcal{O}(k^{2-\epsilon})$ kernel.
- More $\mathcal{O}(k^{d-\epsilon})$ kernelization lower bounds for problems like Set Packing or Hitting Set for families of sets of size at most d.

Turing kernelization

- Standard notion of kernelization is based on many-one reductions:

Turing kernelization

- Standard notion of kernelization is based on many-one reductions:
  - **One** instance (G, k) is mapped to **one** small instance (G', k').

Turing kernelization

- Standard notion of kernelization is based on many-one reductions:
  - **One** instance (G, k) is mapped to **one** small instance (G', k').
- **Motivation**: The idea of kernelization was that small instances may be solved efficiently, so we may allow outputting or solving multiple small instances, instead of just one.

Turing kernelization

- Standard notion of kernelization is based on many-one reductions:
  - **One** instance (G, k) is mapped to **one** small instance (G', k').
- **Motivation**: The idea of kernelization was that small instances may be solved efficiently, so we may allow outputting or solving multiple small instances, instead of just one.

Turing kernelization

A **Turing kernel** of size $p(k)$ for a parameterized problem L is an algorithm that solves any input instance (x, k) in polynomial time given access to an oracle solving instances of L of size at most $p(k)$.

Turing kernelization

- Standard notion of kernelization is based on many-one reductions:
  - **One** instance (G, k) is mapped to **one** small instance (G', k').
- **Motivation**: The idea of kernelization was that small instances may be solved efficiently, so we may allow outputting or solving multiple small instances, instead of just one.

Turing kernelization

A **Turing kernel** of size $p(k)$ for a parameterized problem L is an algorithm that solves any input instance (x, k) in polynomial time given access to an oracle solving instances of L of size at most $p(k)$.

Turing kernelization

- Standard notion of kernelization is based on many-one reductions:
  - **One** instance (G, k) is mapped to **one** small instance (G', k').
- **Motivation**: The idea of kernelization was that small instances may be solved efficiently, so we may allow outputting or solving multiple small instances, instead of just one.

Turing kernelization

A **Turing kernel** of size $p(k)$ for a parameterized problem L is an algorithm that solves any input instance (x, k) in polynomial time given access to an oracle solving instances of L of size at most $p(k)$.

- **Q**: Is Turing kernelization more powerful than the standard one?

Turing kernelization

- Standard notion of kernelization is based on many-one reductions:
  - **One** instance (G, k) is mapped to **one** small instance (G', k').
- **Motivation**: The idea of kernelization was that small instances may be solved efficiently, so we may allow outputting or solving multiple small instances, instead of just one.

Turing kernelization

A **Turing kernel** of size $p(k)$ for a parameterized problem L is an algorithm that solves any input instance (x, k) in polynomial time given access to an oracle solving instances of L of size at most $p(k)$.

- **Q**: Is Turing kernelization more powerful than the standard one?
- **Answer**: Yes, but we do not understand it.

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**/poly.

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
  - The problem is **NP**-hard.

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
  - The problem is **NP**-hard.
  - Disjoint union yields an OR-composition.

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
  - The problem is **NP**-hard.
  - Disjoint union yields an OR-composition.
- **Fact**: Max Leaf on **connected** graphs has a kernel of size $\mathcal{O}(k)$.

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**/poly.
  - The problem is **NP**-hard.
  - Disjoint union yields an OR-composition.
- **Fact**: Max Leaf on **connected** graphs has a kernel of size $\mathcal{O}(k)$.
  - **Thm**: A connected graph with minimum degree $\geqslant 3$ and having at least $4k$ vertices admits a spanning tree with at least k leaves.

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**/poly.
  - The problem is **NP**-hard.
  - Disjoint union yields an OR-composition.
- **Fact**: Max Leaf on **connected** graphs has a kernel of size $\mathcal{O}(k)$.
  - **Thm**: A connected graph with minimum degree $\geqslant 3$ and having at least $4k$ vertices admits a spanning tree with at least k leaves.
  - The number of vertices of degree 1 and 2 can be reduced by easy reduction rules.

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**/poly.
  - The problem is **NP**-hard.
  - Disjoint union yields an OR-composition.
- **Fact**: Max Leaf on **connected** graphs has a kernel of size $\mathcal{O}(k)$.
  - **Thm**: A connected graph with minimum degree $\geqslant 3$ and having at least $4k$ vertices admits a spanning tree with at least k leaves.
  - The number of vertices of degree 1 and 2 can be reduced by easy reduction rules.
- **Corollary**: Max Leaf admits a linear Turing kernel.

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**/poly.
  - The problem is **NP**-hard.
  - Disjoint union yields an OR-composition.
- **Fact**: Max Leaf on **connected** graphs has a kernel of size $\mathcal{O}(k)$.
  - **Thm**: A connected graph with minimum degree $\geqslant 3$ and having at least $4k$ vertices admits a spanning tree with at least k leaves.
  - The number of vertices of degree 1 and 2 can be reduced by easy reduction rules.
- **Corollary**: Max Leaf admits a linear Turing kernel.
  - Kernelize every connected component separately and ask the oracle about each of them.

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**/poly.
  - The problem is **NP**-hard.
  - Disjoint union yields an OR-composition.
- **Fact**: Max Leaf on **connected** graphs has a kernel of size $\mathcal{O}(k)$.
  - **Thm**: A connected graph with minimum degree $\geqslant 3$ and having at least $4k$ vertices admits a spanning tree with at least k leaves.
  - The number of vertices of degree 1 and 2 can be reduced by easy reduction rules.
- **Corollary**: Max Leaf admits a linear Turing kernel.
  - Kernelize every connected component separately and ask the oracle about each of them.
  - **OR-Turing kernel**: Only prepares a polynomial number of instances and asks the oracle about their OR.

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**/poly.
  - The problem is **NP**-hard.
  - Disjoint union yields an OR-composition.
- **Fact**: Max Leaf on **connected** graphs has a kernel of size $\mathcal{O}(k)$.
  - **Thm**: A connected graph with minimum degree $\geqslant 3$ and having at least $4k$ vertices admits a spanning tree with at least k leaves.
  - The number of vertices of degree 1 and 2 can be reduced by easy reduction rules.
- **Corollary**: Max Leaf admits a linear Turing kernel.
  - Kernelize every connected component separately and ask the oracle about each of them.
  - **OR-Turing kernel**: Only prepares a polynomial number of instances and asks the oracle about their OR.
- **Other example**: k-Path on planar graphs (Jansen, 2014).

Example for Turing kernelization

- Max Leaf: Given G and k, does G admits a subtree with k leaves?
- Max Leaf has no polynomial kernel unless **NP** $\subseteq$ **coNP**/poly.
  - The problem is **NP**-hard.
  - Disjoint union yields an OR-composition.
- **Fact**: Max Leaf on **connected** graphs has a kernel of size $\mathcal{O}(k)$.
  - **Thm**: A connected graph with minimum degree $\geqslant 3$ and having at least $4k$ vertices admits a spanning tree with at least k leaves.
  - The number of vertices of degree 1 and 2 can be reduced by easy reduction rules.
- **Corollary**: Max Leaf admits a linear Turing kernel.
  - Kernelize every connected component separately and ask the oracle about each of them.
  - **OR-Turing kernel**: Only prepares a polynomial number of instances and asks the oracle about their OR.
- **Other example**: k-Path on planar graphs (Jansen, 2014).
  - This one is truly adaptative.

Open ends

- **Concrete open problems**:

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set:
    remove k vertices from a given digraph to make it acyclic.

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set:
    remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set:
    remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set: remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set:
    remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.
    - Has a polynomial kernel provided $\mathcal{F}$ contains a planar graph.

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set: remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.
    - Has a polynomial kernel provided $\mathcal{F}$ contains a planar graph.
- **Directions**:

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set: remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.
    - Has a polynomial kernel provided $\mathcal{F}$ contains a planar graph.
- **Directions**:
  - Framework explaining limitations for Turing kernelization.

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set: remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.
    - Has a polynomial kernel provided $\mathcal{F}$ contains a planar graph.
- **Directions**:
  - Framework explaining limitations for Turing kernelization.
  - Complexity theory for kernelization (Hermelin et al.; 2015).

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set: remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.
    - Has a polynomial kernel provided $\mathcal{F}$ contains a planar graph.
- **Directions**:
  - Framework explaining limitations for Turing kernelization.
  - Complexity theory for kernelization (Hermelin et al.; 2015).
  - Lossy kernels.

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set: remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.
    - Has a polynomial kernel provided $\mathcal{F}$ contains a planar graph.
- **Directions**:
  - Framework explaining limitations for Turing kernelization.
  - Complexity theory for kernelization (Hermelin et al.; 2015).
  - Lossy kernels.
  - Bounding resources for kernelization algorithm:

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set: remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.
    - Has a polynomial kernel provided $\mathcal{F}$ contains a planar graph.
- **Directions**:
  - Framework explaining limitations for Turing kernelization.
  - Complexity theory for kernelization (Hermelin et al.; 2015).
  - Lossy kernels.
  - Bounding resources for kernelization algorithm:
    - Linear-time kernelization

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set: remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.
    - Has a polynomial kernel provided $\mathcal{F}$ contains a planar graph.
- **Directions**:
  - Framework explaining limitations for Turing kernelization.
  - Complexity theory for kernelization (Hermelin et al.; 2015).
  - Lossy kernels.
  - Bounding resources for kernelization algorithm:
    - Linear-time kernelization
    - Streaming

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set: remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.
    - Has a polynomial kernel provided $\mathcal{F}$ contains a planar graph.
- **Directions**:
  - Framework explaining limitations for Turing kernelization.
  - Complexity theory for kernelization (Hermelin et al.; 2015).
  - Lossy kernels.
  - Bounding resources for kernelization algorithm:
    - Linear-time kernelization
    - Streaming
- **Thank you for your attention!**

Open ends

- **Concrete open problems**:
  - Directed Feedback Vertex Set: remove k vertices from a given digraph to make it acyclic.
  - Multiway Cut: Given a graph G with terminal set T, disconnect all terminals by removing at most k edges.
    - Has a kernel with $\mathcal{O}(k^{|T|+1})$ vertices; it is open whether the exponent needs to depend on $|T|$.
  - $\mathcal{F}$-Deletion: For a finite family of graphs $\mathcal{F}$, remove k vertices from a given graph G to make it $\mathcal{F}$-minor-free.
    - Has a polynomial kernel provided $\mathcal{F}$ contains a planar graph.
- **Directions**:
  - Framework explaining limitations for Turing kernelization.
  - Complexity theory for kernelization (Hermelin et al.; 2015).
  - Lossy kernels.
  - Bounding resources for kernelization algorithm:
    - Linear-time kernelization
    - Streaming
- **Thank you for your attention!**