Lower bounds for polynomial kernelization

Michał Pilipczuk

Institute of Informatics, University of Warsaw, Poland

Parameterized Complexity Summer School
Vienna, September 2\textsuperscript{nd}, 2017
Kernelization — recap
Kernelization — recap

instance of $L$
Kernelization — recap

instance of $L$
Kernelization — recap

instance of $L$ $\xrightarrow{P\text{-}time} k$
Kernelization — recap

instance of $L$

$P$-time

instance of $L$

size $\leq f(k)$
Unparameterized problems

⇔

Languages over $\Sigma$, for a finite alphabet $\Sigma$

⇔

Subsets of $\Sigma^*$
Unparameterized problems
\[ \Leftrightarrow \]
Languages over \( \Sigma \), for a finite alphabet \( \Sigma \)
\[ \Leftrightarrow \]
Subsets of \( \Sigma^* \)

Parameterized problems
\[ \Leftrightarrow \]
Sets of pairs \((x, k)\), where \( x \in \Sigma^* \) and \( k \) is a nonnegative integer
Unparameterized problems
\[ \iff \]
Languages over \( \Sigma \), for a finite alphabet \( \Sigma \)
\[ \iff \]
Subsets of \( \Sigma^* \)

Parameterized problems
\[ \iff \]
Sets of pairs \((x, k)\), where \( x \in \Sigma^* \) and \( k \) is a nonnegative integer

*Unparameterized variant*: \( k \) is appended to \( x \) in unary.
Background in complexity theory

Unparameterized problems

\[ \iff \]

Languages over \( \Sigma \), for a finite alphabet \( \Sigma \)

\[ \iff \]

Subsets of \( \Sigma^* \)

Parameterized problems

\[ \iff \]

Sets of pairs \((x, k)\), where \( x \in \Sigma^* \) and \( k \) is a nonnegative integer

- **Unparameterized variant**: \( k \) is appended to \( x \) in unary.
- **Kernelization algorithm** takes on input an instance \((x, k)\), and outputs an instance \((x', k')\) such that

\[(x, k) \in L \iff (x', k') \in L \quad \text{and} \quad |x'| + k' \leq f(k)\]

for some computable function \( f \).
If a decidable problem has a kernelization algorithm, then it is FPT.
• If a decidable problem has a kernelization algorithm, then it is FPT.
• Any FPT problem admits a kernelization algorithm:
If a decidable problem has a kernelization algorithm, then it is FPT.

Any FPT problem admits a kernelization algorithm:

- Let \((x, k)\) be the input instance.

Question of existence of any kernel is equivalent to being FPT.

We are interested in polynomial kernels, where \(f\) is a polynomial.

Before 2008, no tool to classify FPT problems wrt. whether they have polykernels or not.
If a decidable problem has a kernelization algorithm, then it is FPT.

Any FPT problem admits a kernelization algorithm:
- Let \((x, k)\) be the input instance.
- If \(|x| \leq f(k)\), then we already have a kernel.
If a decidable problem has a kernelization algorithm, then it is FPT.

Any FPT problem admits a kernelization algorithm:

- Let \((x, k)\) be the input instance.
- If \(|x| \leq f(k)\), then we already have a kernel.
- Otherwise \(f(k) \cdot |x|^c = O(|x|^{c+1})\).
If a decidable problem has a kernelization algorithm, then it is FPT.

Any FPT problem admits a kernelization algorithm:

- Let \((x, k)\) be the input instance.
- If \(|x| \leq f(k)\), then we already have a kernel.
- Otherwise \(f(k) \cdot |x|^c = O(|x|^{c+1})\).

Question of existence of any kernel is equivalent to being FPT.
If a decidable problem has a kernelization algorithm, then it is FPT.

Any FPT problem admits a kernelization algorithm:
- Let \((x, k)\) be the input instance.
- If \(|x| \leq f(k)\), then we already have a kernel.
- Otherwise \(f(k) \cdot |x|^c = O(|x|^{c+1})\).

Question of existence of any kernel is equivalent to being FPT.

We are interested in polynomial kernels, where \(f\) is a polynomial.
If a decidable problem has a kernelization algorithm, then it is FPT.

Any FPT problem admits a kernelization algorithm:
- Let \((x, k)\) be the input instance.
- If \(|x| \leq f(k)\), then we already have a kernel.
- Otherwise \(f(k) \cdot |x|^c = O(|x|^{c+1})\).

Question of existence of any kernel is equivalent to being FPT.

We are interested in polynomial kernels, where \(f\) is a polynomial.

Before 2008, no tool to classify FPT problems wrt. whether they have polykernels or not.
Motivating intuition

Consider the \textit{k-Path} problem: verify whether the input graph contains a simple path on \( k \) vertices.
Motivating intuition

- Consider the $k$-PATH problem: verify whether the input graph contains a simple path on $k$ vertices.
- Suppose for a moment that $k$-PATH admits a kernelization algorithm that, say, produces kernels with at most $k^3$ vertices.

Intuition

The final number of bits is much less than the number input instances. Most of the instances have to be discarded completely.
Consider the $k$-PATH problem: verify whether the input graph contains a simple path on $k$ vertices.

Suppose for a moment that $k$-PATH admits a kernelization algorithm that, say, produces kernels with at most $k^3$ vertices.

Take $t = k^7$ instances $(G_1, k), (G_2, k), \ldots, (G_t, k)$. 

Let $H$ be a disjoint union of $G_1, G_2, \ldots, G_t$. Then the answer to $(H, k)$ is YES if and only if the answer to any $(G_i, k)$ is YES.

Apply kernelization to $(H, k)$ obtaining an instance with $k^{3/2}$ vertices, encodable in $k^{6/2}$ bits.

Intuition: The final number of bits is much less than the number of input instances. Most of the instances have to be discarded completely.
Consider the $k$-PATH problem: verify whether the input graph contains a simple path on $k$ vertices.

Suppose for a moment that $k$-PATH admits a kernelization algorithm that, say, produces kernels with at most $k^3$ vertices.

Take $t = k^7$ instances $(G_1, k), (G_2, k), \ldots, (G_t, k)$.

Let $H$ be a disjoint union of $G_1, G_2, \ldots, G_t$. Then the answer to $(H, k)$ is YES if and only if the answer to any $(G_i, k)$ is YES.
Motivating intuition

Consider the $k$-\textsc{Path} problem: verify whether the input graph contains a simple path on $k$ vertices.

Suppose for a moment that $k$-\textsc{Path} admits a kernelization algorithm that, say, produces kernels with at most $k^3$ vertices.

Take $t = k^7$ instances $(G_1, k), (G_2, k), \ldots, (G_t, k)$.

Let $H$ be a disjoint union of $G_1, G_2, \ldots, G_t$. Then the answer to $(H, k)$ is YES if and only if the answer to any $(G_i, k)$ is YES.

Apply kernelization to $(H, k)$ obtaining an instance with $k^3$ vertices, encodable in $k^6$ bits.
Motivating intuition

- Consider the \textit{k-PATH} problem: verify whether the input graph contains a simple path on \( k \) vertices.
- Suppose for a moment that \textit{k-PATH} admits a kernelization algorithm that, say, produces kernels with at most \( k^3 \) vertices.
- Take \( t = k^7 \) instances \((G_1, k), (G_2, k), \ldots, (G_t, k)\).
- Let \( H \) be a disjoint union of \( G_1, G_2, \ldots, G_t \). Then the answer to \((H, k)\) is YES if and only if the answer to any \((G_i, k)\) is YES.
- Apply kernelization to \((H, k)\) obtaining an instance with \( k^3 \) vertices, encodable in \( k^6 \) bits.

**Intuition**

The final number of bits is much less than the number input instances. Most of the instances have to be \textbf{discarded completely}.
Kernelization and Compression

**KERNELIZATION**

instance of $L$ \[ \overset{P\text{-time}}{\longrightarrow} \]

instance of $L$

size $\leq p(k)$
**Kernelization and Compression**

**KERNELIZATION**

Instance of $L$ $\xrightarrow{P\text{-time}}$ Instance of $L$ size $\leq p(k)$

**COMPRESSION**

Instance of $L$ $\xrightarrow{P\text{-time}}$ Instance of $R$ (any) size $\leq p(k)$
Kernelization and Compression

KERNELIZATION

Instance of $L$ \quad \rightarrow \quad P$-time \quad \rightarrow \quad Instance of $L$

Instance of $L$ \quad \rightarrow \quad $P$-time

COMPRESSION

Instance of $L$ \quad \rightarrow \quad Instance of $R$ (any)

Instance of $L$ \quad \rightarrow \quad Instance of $R$ (any)
**Intuition:** In compression we only care about shrinking the size of the instance to a small size without mixing YES- and NO-instances.
- **Intuition**: In compression we only care about shrinking the size of the instance to a small size without mixing YES- and NO-instances.
- A polynomial kernelization is always a polynomial compression.
**Intuition**: In compression we only care about shrinking the size of the instance to a small size without mixing YES- and NO-instances.

A polynomial kernelization is always a polynomial compression.

A polynomial compression can be turned into a polynomial kernelization provided that there is a \( P \)-reduction from \( R \) to \( L \).
**Intuition**: In compression we only care about shrinking the size of the instance to a small size without mixing YES- and NO-instances.

- A polynomial kernelization is always a polynomial compression.
- A polynomial compression can be turned into a polynomial kernelization provided that there is a \( \mathsf{P} \)-reduction from \( R \) to \( L \).
  - For instance, when \( R \in \mathsf{NP} \) and \( L \) is \( \mathsf{NP} \)-hard.
**Intuition:** In compression we only care about shrinking the size of the instance to a small size without mixing YES- and NO-instances.

A polynomial kernelization is always a polynomial compression.

A polynomial compression can be turned into a polynomial kernelization provided that there is a $\mathbf{P}$-reduction from $R$ to $L$.

- For instance, when $R \in \mathbf{NP}$ and $L$ is $\mathbf{NP}$-hard.

**Note:** There are examples when a poly-compression is known but a poly-kernel is not known, because it is unclear whether $R$ is in $\mathbf{NP}$. 
Let $L, R$ be unparameterized languages.
Let $L, R$ be \textit{unparameterized} languages.

**OR-distillation of $L$ into $R$**

**Input:** Words $x_1, x_2, \ldots, x_t$, each of length at most $k$.

**Time:** $\text{poly}(t + \sum_{i=1}^{t} |x_i|)$.

**Output:** One word $y$ such that

(a) $|y| = \text{poly}(k)$, and

(b) $y \in R$ if and only if $x_i \in L$ for at least one $i$. 
OR-distillation on picture

\[ \text{instances} \text{ of } \{L\} \leq k \leq \text{poly}\ (k) \]

Intuition: Necessary loss of information \( \Rightarrow \) Contradiction for an \( \text{NP-hard} \)

Define \( \text{OR-L} = \{x_1 \# x_2 \# \ldots \# x_t : x_i \in L \text{ for at least one } i\} \).

\( \text{OR-distillation } L \rightarrow R \) is a polynomial compression \( \text{OR-L} / \max |x_i| \rightarrow R \).
**OR-distillation on picture**

Intuition: Necessary loss of information $\Rightarrow$ Contradiction for an NP-hard

Define $OR-L = \{x_1 \# x_2 \# \ldots \# x_t : x_i \in L \text{ for at least one } i\}$. 

$OR$-distillation $L \rightarrow R$ is a polynomial compression $OR-L / \max |x_i| \rightarrow R$. 

$\leq k$ instances
**OR-distillation on picture**

Intuition: Necessary loss of information $\Rightarrow$ Contradiction for an NP-hard L

Define $\text{OR-L} = \{x_1 \# x_2 \# \ldots \# x_t : x_i \in L \text{ for at least one } i\}$.  

$\text{OR-distillation } L \rightarrow R$ is a polynomial compression $\frac{\text{OR-L}}{\max |x_i|} \rightarrow R$.
Intuition: Necessary loss of information $\Rightarrow$ Contradiction for an $\mathbf{NP}$-hard $L$

Define $\text{OR-}L = \{x_1 \# x_2 \# \ldots \# x_t : x_i \in L$ for at least one $i\}$.

$\text{OR-distillation } L \rightarrow R$ is a polynomial compression $\frac{\text{OR-}L}{\max |x_i|} \rightarrow R$

$P$-time $\leq \text{poly}(k)$
Intuition: Necessary loss of information $\leadsto$ Contradiction for an NP-hard $L$
Intuition: Necessary loss of information $\leadsto$ Contradiction for an \textbf{NP}-hard $L$

Define $\text{OR-}L = \{x_1 \# x_2 \# \ldots \# x_t : x_i \in L \text{ for at least one } i \}$. OR-distillation $L \rightarrow R$ is a polynomial compression $\text{OR-}L/\max |x_i| \rightarrow R$
Backbone theorem

**OR-distillation theorem** [Fortnow, Santhanam; 2008]

SAT does not admit an OR-distillation algorithm into any language \( R \), unless \( \text{NP} \subseteq \text{coNP}/\text{poly} \).

**Corollary**

No \( \text{NP} \)-hard problem admits an OR-distillation algorithm into any language \( R \), unless \( \text{NP} \subseteq \text{coNP}/\text{poly} \).

**Assumption** \( \text{NP} \subseteq \text{coNP}/\text{poly} \) may seem mysterious.

**Intuition**:
Verifying proofs in \( \text{P} \)-time cannot be turned into verifying counterexamples in \( \text{P} \)-time, even if we allow polynomial advice.

\( \text{NP} \subseteq \text{coNP}/\text{poly} \) implies \( \text{PH} = \Sigma_3^P \).

Not as bad as \( \text{P} = \text{NP} \), but still considered very unlikely.

The proof is very short, but very tricky.
Backbone theorem

**OR-distillation theorem** [Fortnow, Santhanam; 2008]

SAT does not admit an OR-distillation algorithm into any language $R$, unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$.

**Corollary**

No $\textbf{NP}$-hard problem admits an OR-distillation algorithm into any language $R$, unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$.
**Backbone theorem**

**OR-distillation theorem**  
[Fortnow, Santhanam; 2008]  
SAT does not admit an OR-distillation algorithm into any language $R$, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

**Corollary**

No $\text{NP}$-hard problem admits an OR-distillation algorithm into any language $R$, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

- Assumption $\text{NP} \subseteq \text{coNP}/\text{poly}$ may seem mysterious.
OR-distillation theorem

SAT does not admit an OR-distillation algorithm into any language $R$, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

Corollary

No $\text{NP}$-hard problem admits an OR-distillation algorithm into any language $R$, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

- Assumption $\text{NP} \subseteq \text{coNP}/\text{poly}$ may seem mysterious.
- **Intuition:** Verifying proofs in $\text{P}$-time cannot be turned into verifying counterexamples in $\text{P}$-time, even if we allow *polynomial advice*. 
**Backbone theorem**

**OR-distillation theorem**

SAT does not admit an OR-distillation algorithm into any language $R$, unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$.

**Corollary**

No $\textbf{NP}$-hard problem admits an OR-distillation algorithm into any language $R$, unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$.

- Assumption $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$ may seem mysterious.
- **Intuition**: Verifying proofs in $\textbf{P}$-time cannot be turned into verifying counterexamples in $\textbf{P}$-time, even if we allow *polynomial advice*.
- $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$ implies $\text{PH} = \Sigma_3^\textbf{P}$. 

Not as bad as $\textbf{P} = \textbf{NP}$, but still considered very unlikely. The proof is very short, but very tricky.
**OR-distillation theorem**

SAT does not admit an OR-distillation algorithm into any language $R$, unless $NP \subseteq coNP/poly$.

**Corollary**

No $NP$-hard problem admits an OR-distillation algorithm into any language $R$, unless $NP \subseteq coNP/poly$.

- Assumption $NP \subseteq coNP/poly$ may seem mysterious.
  - **Intuition**: Verifying proofs in $P$-time cannot be turned into verifying counterexamples in $P$-time, even if we allow *polynomial advice*.
  - $NP \subseteq coNP/poly$ implies $PH = \Sigma_3^P$.
  - Not as bad as $P = NP$, but still considered very unlikely.
**Backbone theorem**

**OR-distillation theorem** [Fortnow, Santhanam; 2008]

SAT does not admit an OR-distillation algorithm into any language $R$, unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$.

**Corollary**

No $\textbf{NP}$-hard problem admits an OR-distillation algorithm into any language $R$, unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$.

- Assumption $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$ may seem mysterious.
  - **Intuition**: Verifying proofs in $\textbf{P}$-time cannot be turned into verifying counterexamples in $\textbf{P}$-time, even if we allow *polynomial advice*.
  - $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$ implies $\textbf{PH} = \Sigma_3^\textbf{P}$.
  - Not as bad as $\textbf{P} = \textbf{NP}$, but still considered very unlikely.

- The proof is very short, but very tricky.
Let $L$ be a parameterized language.
Let $L$ be a parameterized language.

### OR-composition algorithm for $L$

- **Input**: Instances $(x_1, k), (x_2, k), \ldots, (x_t, k)$.
- **Time**: $\text{poly}(t + \sum_{i=1}^{t} |x_i| + k)$. 
- **Output**: One instance $(y, k^*)$ such that
  
  (a) $k^* = \text{poly}(k)$, and 
  
  (b) $(y, k^*) \in L$ iff $(x_i, k) \in L$ for at least one $i$. 

Suppose a parameterized problem $L$ admits an OR-composition algorithm, and the unparameterized version of $L$ is $\text{NP}$-hard. Then $L$ does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. 

[OR-composition on picture]
Suppose a parameterized problem $L$ admits an OR-composition algorithm, and the unparameterized version of $L$ is $\text{NP}$-hard. Then $L$ does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.
Suppose a parameterized problem $L$ admits an OR-composition algorithm, and the unparameterized version of $L$ is $\text{NP}$-hard. Then $L$ does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. 

Michael Pilipczuk

Kernelization lower bounds
Suppose a parameterized problem $L$ admits an OR-composition algorithm, and the unparameterized version of $L$ is $\text{NP}$-hard. Then $L$ does not admit a polynomial kernel unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. 
Suppose a parameterized problem $L$ admits an OR-composition algorithm, and the unparameterized version of $L$ is $\textbf{NP}$-hard.

Then $L$ does not admit a polynomial kernel unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$. 

OR-composition theorem [BDFH; 2008]
Proof
Proof

OR-SAT

\[ \widehat{L} \]

NP-hrd

\[ \widehat{L} \]

NP-hrd

\[ \widehat{L} \]

NP-hrd

\[ \widehat{L} \]

NP-hrd

\[ \widehat{L} \]

NP-hrd

\[ \widehat{L} \]

NP-hrd

\[ \widehat{L} \]

NP-hrd

\[ \widehat{L} \]

NP-hrd

\[ L \]

cmp

poly

\( (k) \)

\[ L \]

cmp

poly

\( (k) \)

\[ L \]

cmp

poly

\( (k) \)

\[ L \]

cmp

poly

\( (k) \)

OR-

\[ L \]

\[ L \]

\[ L \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]

\[ \widehat{L} \]
Proof

\[ \text{Proof:} \]

\[ \text{OR-SAT} \]

\[ \tilde{L} \]

\[ \text{MICHAŁ PILIPCZUK} \]

Kernelization lower bounds
Proof

\[ \sim L_{NP-\text{hrd}} \]

\[ L \]

OR-SAT

Michał Pilipczuk
Kernelization lower bounds
Proof
Proof

OR-SAT

NP-hard

NP-hard

NP-hard

NP-hard

NP-hard

NP-hard

NP-hard

NP-hard

L

poly(k)

poly(k)

poly(k)

L

kern

kern

kern
Proof

Michał Pilipczuk  Kernelization lower bounds  13/33
$k$-Path does not admit a polykernel, unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$. 
Corollaries

- **$k$-Path** does not admit a polykernel, unless $\text{NP} \subseteq \text{coNP/poly}$.

- **Composition:**
  Take the disjoint union of the input graphs and the same parameter.
Corollaries

- **k-Path** does not admit a polykernel, unless \( \text{NP} \subseteq \text{coNP}/\text{poly} \).

- **Composition:**
  Take the disjoint union of the input graphs and the same parameter.
  - A graph admits a \( k \)-path iff any of its connected components does.
Corollaries

- **$k$-Path** does not admit a polykernel, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.
- **Composition:**
  - Take the disjoint union of the input graphs and the same parameter.
  - A graph admits a $k$-path iff any of its connected components does.
- **Same for $k$-Cycle** and many other problems.
Corollaries

- **$k$-Path** does not admit a polykernel, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

- **Composition**: Take the disjoint union of the input graphs and the same parameter.
  - A graph admits a $k$-path iff any of its connected components does.

- Same for $k$-**Cycle** and many other problems.

- Today, investigating the existence of a polynomial kernel is often a secondary goal after showing that a problem is FPT.
Does the proof actually exclude even polynomial compression into any $R$, not just kernelization?
Adding features

- Does the proof actually exclude even polynomial compression into any $R$, not just kernelization?
  - Sure, we will just end up with an instance of $OR-R$. 

(Addendum: 

- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
  - Yes, as long as we have polynomial number of buckets.
  - How large can $t$ be?
    - Well, not larger than $|\Sigma|^k + 1$, as we may remove duplicates of the input instances.
    - Hence, we may assume that $\log t = O(k)$.
    - Ergo, the parameter of the composed instance may depend polynomially on both $k$ and $\log t$.
    - Observed also earlier via different arguments. (Dom, Lokshtanov, and Saurabh; 2009)
Adding features

- Does the proof actually exclude even polynomial compression into any $R$, not just kernelization?
  - Sure, we will just end up with an instance of OR-$R$.
- Do we need to start the composition with the same language $L$ as we apply the compression to?
Does the proof actually exclude even polynomial compression into any \( R \), not just kernelization?

- Sure, we will just end up with an instance of OR-\( R \).

Do we need to start the composition with the same language \( L \) as we apply the compression to?

- No, the composition algorithm can compose instances of any \( \text{NP} \)-hard language \( Q \) into one instance of \( L \).
Does the proof actually exclude even polynomial compression into any $R$, not just kernelization?
- Sure, we will just end up with an instance of OR-$R$.

Do we need to start the composition with the same language $L$ as we apply the compression to?
- No, the composition algorithm can compose instances of any \textbf{NP}-hard language $Q$ into one instance of $L$.

Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
Does the proof actually exclude even polynomial compression into any $R$, not just kernelization?
  - Sure, we will just end up with an instance of OR-$R$.

Do we need to start the composition with the same language $L$ as we apply the compression to?
  - No, the composition algorithm can compose instances of any $\text{NP}$-hard language $Q$ into one instance of $L$.

Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
  - Yes, as long as we have polynomial number of buckets.
Adding features

- Does the proof actually exclude even polynomial compression into any $R$, not just kernelization?
  - Sure, we will just end up with an instance of OR-$R$.

- Do we need to start the composition with the same language $L$ as we apply the compression to?
  - No, the composition algorithm can compose instances of any $\text{NP}$-hard language $Q$ into one instance of $L$.

- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
  - Yes, as long as we have polynomial number of buckets.

- How large can $t$ be?

  Well, not larger than $|\Sigma|^{k+1}$, as we may remove duplicates of the input instances. Hence, we may assume that $\log t = O(k)$. Ergo, the parameter of the composed instance may depend polynomially on both $k$ and $\log t$. Observed also earlier via different arguments. (Dom, Lokshtanov, and Saurabh; 2009)
Does the proof actually exclude even polynomial compression into any $R$, not just kernelization?
- Sure, we will just end up with an instance of OR-$R$.

Do we need to start the composition with the same language $L$ as we apply the compression to?
- No, the composition algorithm can compose instances of any NP-hard language $Q$ into one instance of $L$.

Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
- Yes, as long as we have polynomial number of buckets.

How large can $t$ be?
- Well, not larger than $|\Sigma|^{k+1}$, as we may remove duplicates of the input instances.
Does the proof actually exclude even polynomial compression into any $R$, not just kernelization?
- Sure, we will just end up with an instance of OR-$R$.

Do we need to start the composition with the same language $L$ as we apply the compression to?
- No, the composition algorithm can compose instances of any \textbf{NP}-hard language $Q$ into one instance of $L$.

Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
- Yes, as long as we have polynomial number of buckets.

How large can $t$ be?
- Well, not larger than $|\Sigma|^{k+1}$, as we may remove duplicates of the input instances.
- Hence, we may assume that $\log t = O(k)$. 
Adding features

- Does the proof actually exclude even polynomial compression into any $R$, not just kernelization?
  - Sure, we will just end up with an instance of OR-$R$.

- Do we need to start the composition with the same language $L$ as we apply the compression to?
  - No, the composition algorithm can compose instances of any \textbf{NP}-hard language $Q$ into one instance of $L$.

- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
  - Yes, as long as we have polynomial number of buckets.

- How large can $t$ be?
  - Well, not larger than $|\Sigma|^{k+1}$, as we may remove duplicates of the input instances.
  - Hence, we may assume that $\log t = \mathcal{O}(k)$.
  - \textbf{Ergo, the parameter of the composed instance may depend polynomially on both $k$ and $\log t$.}
Does the proof actually exclude even polynomial compression into any \( R \), not just kernelization?
- Sure, we will just end up with an instance of OR-\( R \).

Do we need to start the composition with the same language \( L \) as we apply the compression to?
- No, the composition algorithm can compose instances of any \( \text{NP} \)-hard language \( Q \) into one instance of \( L \).

Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
- Yes, as long as we have polynomial number of buckets.

How large can \( t \) be?
- Well, not larger than \(|\Sigma|^{k+1}\), as we may remove duplicates of the input instances.
- Hence, we may assume that \( \log t = O(k) \).
- Ergo, the parameter of the composed instance may depend polynomially on both \( k \) and \( \log t \).
- Observed also earlier via different arguments. (Dom, Lokshtanov, and Saurabh; 2009)
After the invention of the technique of OR-compositions, there was a huge number of no-polykernel results.
After the invention of the technique of OR-compositions, there was a huge number of no-polykernel results.

- As we’ll see later, there can be much more intricate compositions than just “disjoint union”.

Examples:
- Max Leaf Subtree
- Set Cover / m
- Set Cover / n
- Steiner Tree
- Connected Vertex Cover
- Disjoint Paths
- Directed Multiway Cut with 2 terminals...
After the invention of the technique of OR-compositions, there was a huge number of no-polykernel results.

- As we’ll see later, there can be much more intricate compositions than just “disjoint union”.
- **Examples:** Max Leaf Subtree, Set Cover/$m$, Set Cover/$n$, Steiner Tree, Connected Vertex Cover, Disjoint Paths, Directed Multiway Cut with 2 terminals, ...
After the invention of the technique of OR-compositions, there was a huge number of no-polykernel results.

As we’ll see later, there can be much more intricate compositions than just “disjoint union”.

**Examples:** Max Leaf Subtree, Set Cover/$m$, Set Cover/$n$, Steiner Tree, Connected Vertex Cover, Disjoint Paths, Directed Multiway Cut with 2 terminals, ...

Most of the works use a subset of mentioned features.
After the invention of the technique of OR-compositions, there was a huge number of no-polykernel results.

- As we’ll see later, there can be much more intricate compositions than just “disjoint union”.
- **Examples:** Max Leaf Subtree, Set Cover/$m$, Set Cover/$n$, Steiner Tree, Connected Vertex Cover, Disjoint Paths, Directed Multiway Cut with 2 terminals, ...

Most of the works use a subset of mentioned features.

**Later:** a new formalism **cross-composition** gathers all the features. (Bodlaender, Jansen, and Kratsch; 2011)
Equivalence relation $\sim$ on $\Sigma^*$ is a **polynomial equivalence relation** if:
- checking whether two words $x, y \in \Sigma^*$ are $\sim$-equivalent can be done in $\text{poly}(|x| + |y|)$ time; and
- $\sim$ partitions words of length $\leq n$ into $\text{poly}(n)$ equivalence classes.
Polynomial equivalence relation

Equivalence relation $\sim$ on $\Sigma^*$ is a **polynomial equivalence relation** if:

- checking whether two words $x, y \in \Sigma^*$ are $\sim$-equivalent can be done in $\text{poly}(|x| + |y|)$ time; and
- $\sim$ partitions words of length $\leq n$ into $\text{poly}(n)$ equivalence classes.
Equivalence relation $\sim$ on $\Sigma^*$ is a polynomial equivalence relation if:
- checking whether two words $x, y \in \Sigma^*$ are $\sim$-equivalent can be done in $\text{poly}(|x| + |y|)$ time; and
- $\sim$ partitions words of length $\leq n$ into $\text{poly}(n)$ equivalence classes.

Examples, supposing some reasonable graph encoding:
Polynomial equivalence relation

Equivalence relation $\sim$ on $\Sigma^*$ is a **polynomial equivalence relation** if:
- checking whether two words $x, y \in \Sigma^*$ are $\sim$-equivalent can be done in $\text{poly}(|x| + |y|)$ time; and
- $\sim$ partitions words of length $\leq n$ into $\text{poly}(n)$ equivalence classes.

**Examples**, supposing some reasonable graph encoding:
- partitioning with respect to the number of vertices of the graph;
Polynomial equivalence relation

Equivalence relation $\sim$ on $\Sigma^*$ is a **polynomial equivalence relation** if:
- checking whether two words $x, y \in \Sigma^*$ are $\sim$-equivalent can be done in $\text{poly}(|x| + |y|)$ time; and
- $\sim$ partitions words of length $\leq n$ into $\text{poly}(n)$ equivalence classes.

**Examples**, supposing some reasonable graph encoding:
- partitioning with respect to the number of vertices of the graph;
- or with respect to (i) the number of vertices, (ii) the number of edges, (iii) size of the maximum matching, (iv) budget.
An unparameterized problem \( Q \) **cross-composes** into a parameterized problem \( L \), if there exists a polynomial equivalence relation \( \sim \) and an algorithm that, given \( \sim \)-equivalent strings \( x_1, x_2, \ldots, x_t \), in time \( \text{poly} \left( t + \sum_{i=1}^{t} |x_i| \right) \) produces one instance \((y, k^*)\) such that

- \((y, k^*) \in L\) iff \( x_i \in Q \) for at least one \( i = 1, 2, \ldots, t \),
- \( k^* = \text{poly} \left( \log t + \max_{i=1}^{t} |x_i| \right) \).
An unparameterized problem $Q$ **cross-composes** into a parameterized problem $L$, if there exists a polynomial equivalence relation $\sim$ and an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$, in time $\text{poly} \left( t + \sum_{i=1}^{t} |x_i| \right)$ produces one instance $(y, k^*)$ such that
- $(y, k^*) \in L$ iff $x_i \in Q$ for at least one $i = 1, 2, \ldots, t$,
- $k^* = \text{poly} \left( \log t + \max_{i=1}^{t} |x_i| \right)$.

**Cross-composition theorem**  
[Bodlaender, Jansen, Kratsch]

If some $\textbf{NP}$-hard problem $Q$ cross-composes into $L$, then $L$ has no polynomial compression into any language $R$, unless $\textbf{NP} \subseteq \text{coNP}/\text{poly}$.
Proof

\[ q_k = \max |x_i|, \log t = O(k) \]

\[
L_{\text{compo}}(k) = \underbrace{L_{\text{compo}}(k) + L_{\text{compo}}(k) + \ldots + L_{\text{compo}}(k)}_{\text{OR-}}
\]
Proof

\[ k = \max |x_i|, \quad \log t = O(k) \]
Proof

\[ k = \max |x_i|, \quad \log t = \mathcal{O}(k) \]
Proof

$k = \max |x_i|, \quad \log t = O(k)$
Proof

\[ k = \max |x_i|, \quad \log t = O(k) \]

\[ Q = \text{max} |x_i|, \quad \log t = O(k) \]
Proof

\[ k = \max |x_i|, \quad \log t = O(k) \]
Proof

\[ k = \max |x_i|, \quad \log t = O(k) \]
Original application of Bodlaender, Jansen and Kratsch was that of structural parameters.
Original application of Bodlaender, Jansen and Kratsch was that of **structural parameters**.

In fact, cross-composition is a good framework to express also all the previous results.
Original application of Bodlaender, Jansen and Kratsch was that of **structural parameters**.

In fact, cross-composition is a good framework to express also all the previous results.

**Plan for now**: show some non-trivial cross-composition to give an intuition about basic tricks.
**Set Splitting**

**I:** Universe $U$ and family of subsets $\mathcal{F} \subseteq 2^U$

**P:** $|U|$

**Q:** Is there a coloring $C : U \rightarrow \{\mathbf{B}, \mathbf{W}\}$ such that every set $X \in \mathcal{F}$ is split, i.e., contains a black and a white element?
Application 1: Set Splitting

**Set Splitting**

**I:** Universe $U$ and family of subsets $\mathcal{F} \subseteq 2^U$

**P:** $|U|$

**Q:** Is there a coloring $C : U \rightarrow \{B, W\}$ such that every set $X \in \mathcal{F}$ is split, i.e., contains a black and a white element?

- We show a cross-composition of Set Splitting into itself.
Application 1: **Set Splitting**

We show a cross-composition of *Set Splitting* into itself.

We may assume that the universes are of the same size, hence we think of them as of one, common universe.

---

**Set Splitting**

I: Universe $U$ and family of subsets $\mathcal{F} \subseteq 2^U$

P: $|U|$

Q: Is there a coloring $\mathcal{C}: U \rightarrow \{\text{B, W}\}$ such that every set $X \in \mathcal{F}$ is split, i.e., contains a black and a white element?
Application 1: Set Splitting

**Set Splitting**

**I:** Universe $U$ and family of subsets $\mathcal{F} \subseteq 2^U$

**P:** $|U|$

**Q:** Is there a coloring $C : U \rightarrow \{B, W\}$ such that every set $X \in \mathcal{F}$ is split, i.e., contains a black and a white element?

- We show a cross-composition of Set Splitting into itself.
- We may assume that the universes are of the same size, hence we think of them as of one, common universe.
- Assume that $t$ is a power of 2 (by copying the instances).
Cross-composing into Set Splitting

**Input:** Instances $(U, \mathcal{F}^i)$

**Output:** Instance $(U^*, \mathcal{F}^*)$
Cross-composing into \textbf{Set Splitting}

**Input:** Instances \((U, F^i)\)

**Output:** Instance \((U^*, F^*)\)

**PLAYGROUND**

Joint universe \(U\)

*Kernelization lower bounds*
**Cross-composing into Set Splitting**

**INSTANCE SELECTOR**

Input: Instances $(U, F_i)$

Output: Instance $(U^*, F^*)$

1 + log $t$ pairs of vertices

---

**PLAYGROUND**

joint universe $U$
Cross-composing into \textsc{Set Splitting}

\textbf{INSTANCE SELECTOR}

\begin{itemize}
  \item \textbf{Input:} Instances \((U, F^i)\)
  \item \textbf{Output:} Instance \((U^*, F^*)\)
\end{itemize}

\[ |U^*| = |U| + 2 \log t + 2 \]
Cross-composing into Set Splitting

**INSTANCE SELECTOR**

Input: Instances \((U, \mathcal{F}^i)\)

Output: Instance \((U^*, \mathcal{F}^*)\)

\[ |U^*| = |U| + 2 \log t + 2 \]

\(\mathcal{F}^*\) consists of:

- 1 + \(\log t\) pairs of vertices

**PLAYGROUND**

joint universe \(U\)
Cross-composing into **Set Splitting**

**INSTANCE SELECTOR**

1 + \log t pairs of vertices

Input: Instances \((U, F^i)\)

Output: Instance \((U^*, F^*)\)

\(|U^*| = |U| + 2 \log t + 2\)

\(F^*\) consists of:
1 + \log t 2-element sets for pairs,

**PLAYGROUND**

Joint universe \(U\)
Cross-composing into **Set Splitting**

**Input**: Instances \((U, F^i)\)

**Output**: Instance \((U^*, F^*)\)

\[|U^*| = |U| + 2 \log t + 2\]

\(F^*\) consists of:
- \(1 + \log t\) 2-element sets for pairs,
- \(\forall X \in F^i\), two sets \(X^*_0, X^*_1\)

**INSTANCE SELECTOR**

\(1 + \log t\) pairs of vertices

**PLAYGROUND**

joint universe \(U\)

There is exactly one index \(i\) with monochromatic parts from IS.

\((\Rightarrow)\): \(C\) on IS defines, which instance must be solved in PL.

\((\Leftarrow)\): If \((U, F^i)\) is solvable, we set IS accordingly, and solve this instance in PL. Remaining sets are split for free.
Cross-composing into \textbf{Set Splitting}

**Input**: Instances \((U, F_i)\)

**Output**: Instance \((U^*, F^*)\)

\[|U^*| = |U| + 2 \log t + 2\]

\(F^*\) consists of:
1. \(1 + \log t\) 2-element sets for pairs,
2. \(\forall X \in F^i\), two sets \(X_0^*, X_1^*\)

\(X_0^*\): \(X\), left special vertex, and binary encoding of \(i\) in IS

\textbf{INSTANACE SELECTOR}

1 + log t pairs of vertices

\textbf{PLAYGROUND}

joint universe \(U\)
Cross-composing into Set Splitting

**Input:** Instances $(U, \mathcal{F}^i)$

**Output:** Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2 \log t + 2$

$\mathcal{F}^*$ consists of:
- $1 + \log t$ 2-element sets for pairs,
- $\forall X \in \mathcal{F}^i$, two sets $X_0^*$, $X_1^*$

$X_0^*$: $X$, left special vertex, and binary encoding of $i$ in IS

$X_1^*$: reverse $X_0^*$ on IS

**INSTANCE SELECTOR**

1 + log t pairs of vertices

**PLAYGROUND**

joint universe $U$
Cross-composing into Set Splitting

**INSTANCE SELECTOR**

*Input:* Instances \((U, \mathcal{F}^i)\)

*Output:* Instance \((U^*, \mathcal{F}^*)\)

\[|U^*| = |U| + 2 \log t + 2\]

**\(\mathcal{F}^*\) consists of:**

1. \(1 + \log t\) 2-element sets for pairs,
2. \(\forall X \in \mathcal{F}^i\), two sets \(X^*_0, X^*_1\)

**PLAYGROUND**

joint universe \(U\)

Take any solution \(C\)
Cross-composing into Set Splitting

**Input**: Instances \((U, F^i)\)

**Output**: Instance \((U^*, F^*)\)

\(|U^*| = |U| + 2 \log t + 2\)

\(F^*\) consists of:
- \(1 + \log t\) 2-element sets for pairs,
- \(\forall X \in F^i\), two sets \(X^*_0, X^*_1\)

**Instance Selector**

1 + log \(t\) pairs of vertices

Take any solution \(C\)

There is exactly one index \(i\) with monochromatic parts from IS.

**Playground**

Joint universe \(U\)
Cross-composing into **Set Splitting**

**INSTANCE SELECTOR**

1 + log \( t \) pairs of vertices

**Input:** Instances \((U, F^i)\)

**Output:** Instance \((U^*, F^*)\)

\[ |U^*| = |U| + 2 \log t + 2 \]

\(F^*\) consists of:

1 + log \( t \) 2-element sets for pairs,

\(\forall X \in F^i\), two sets \(X^*_0, X^*_1\)

Take any solution \(C\)

There is exactly one index \(i\) with monochromatic parts from \(IS\).

**PLAYGROUND**

joint universe \(U\)
Cross-composing into Set Splitting

**INSTANCE SELECTOR**

1 + log t pairs of vertices

Input: Instances \((U, \mathcal{F}^i)\)

Output: Instance \((U^*, \mathcal{F}^*)\)

\[|U^*| = |U| + 2 \log t + 2\]

\(\mathcal{F}^*\) consists of:
- 1 + log t 2-element sets for pairs,
- \(\forall X \in \mathcal{F}^i,\) two sets \(X_0^*, X_1^*\)

Take any solution \(C\)

There is exactly one index \(i\) with monochromatic parts from IS.

PLAYGROUND

joint universe \(U\)
Cross-composing into Set Splitting

**Input**: Instances \((U, \mathcal{F}^i)\)

**Output**: Instance \((U^*, \mathcal{F}^*)\)

\[ |U^*| = |U| + 2 \log t + 2 \]

\(\mathcal{F}^*\) consists of:
- \(1 + \log t\) 2-element sets for pairs,
- \(\forall X \in \mathcal{F}^i\), two sets \(X_0^*, X_1^*\)

Take any solution \(C\)

There is exactly one index \(i\) with monochromatic parts from IS.

(⇒): \(C\) on IS defines, which instance must be solved in PL

⇒: C on IS defines, which instance must be solved in PL

PLAYGROUND

joint universe \(U\)
Cross-composing into Set Splitting

**INSTANCE SELECTOR**

1 + log \( t \) pairs of vertices

- **Input:** Instances \((U, \mathcal{F}^i)\)
- **Output:** Instance \((U^*, \mathcal{F}^*)\)

\[ |U^*| = |U| + 2 \log t + 2 \]

\(\mathcal{F}^*\) consists of:
- 1 + log \( t \) 2-element sets for pairs,
- \(\forall X \in \mathcal{F}^i\), two sets \(X_0^*, X_1^*\)

Take any solution \(C\)

- There is exactly one index \(i\) with monochromatic parts from \(IS\).

\((\Rightarrow):\) \(C\) on \(IS\) defines, which instance must be solved in \(PL\)

\((\Leftarrow):\) If \((U, \mathcal{F}^i)\) is solvable, we set \(IS\) accordingly, and solve this instance in \(PL\). Remaining sets are split for free.
Unparameterized Set Splitting cross-composes into Set Splitting parameterized by $|U|$.
Unparameterized \textsc{Set Splitting} cross-composes into \textsc{Set Splitting} parameterized by $|U|$.

Unparameterized \textsc{Set Splitting} is \textbf{NP}-hard.
Set Splitting: wrap up

- Unparameterized Set Splitting cross-composes into Set Splitting parameterized by $|U|$.
- Unparameterized Set Splitting is NP-hard.
- Hence, Set Splitting parameterized by $|U|$ does not admit a polynomial kernel, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

Main lesson: Model the choice of the instance to be solved. Idea: choose $\log t$ bits of its index on an appropriate gadget. Choice of the index makes the instance active, while the other instances are "switched off".
**Unparameterized Set Splitting** cross-composes into **Set Splitting** parameterized by $|U|$.

- **Unparameterized Set Splitting** is **NP-hard**.
- Hence, **Set Splitting** parameterized by $|U|$ does not admit a polynomial kernel, unless **NP $\subseteq$ coNP/poly**.

**Main lesson:**
Unparameterized \textsc{Set Splitting} cross-composes into \textsc{Set Splitting} parameterized by $|U|$.

Unparameterized \textsc{Set Splitting} is \textbf{NP}-hard.

Hence, \textsc{Set Splitting} parameterized by $|U|$ does not admit a polynomial kernel, unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$.

\textbf{Main lesson:}
- Model the \textbf{choice} of the instance to be solved.
Unparameterized Set Splitting cross-composes into Set Splitting parameterized by $|U|$.

Unparameterized Set Splitting is $\text{NP}$-hard.

Hence, Set Splitting parameterized by $|U|$ does not admit a polynomial kernel, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

**Main lesson:**
- Model the choice of the instance to be solved.
- **Idea:** choose $\log t$ bits of its index on an appropriate gadget.
Unparameterized Set Splitting cross-composes into Set Splitting parameterized by $|U|$.

Unparameterized Set Splitting is NP-hard.

Hence, Set Splitting parameterized by $|U|$ does not admit a polynomial kernel, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

**Main lesson:**
- Model the choice of the instance to be solved.
- **Idea:** choose $\log t$ bits of its index on an appropriate gadget.
- **Choice of the index makes the instance active, while the other instances are “switched off”**.
- **Idea**: Hardness of kernelization can be transferred via reductions, similarly to \textbf{NP}-hardness.
**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 $k' = \text{poly}(k)$. 
• **Idea:** Hardness of kernelization can be transferred via reductions, similarly to \textbf{NP}-hardness.

**Polynomial parameter transformation (PPT)**

A \textit{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 \( k' = \text{poly}(k) \).

**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 \( k' = \text{poly}(k) \).

**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 \( Q \).
**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 $|X| \leq k$ and $G[T \cup X]$ is connected?

We show that Steiner Tree has no polykernel (unless...) using a \textit{PPT} from an auxiliary problem.
Application 2: **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 $|X| \leq k$ 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**

**I:** Graph $G$ and a coloring function $\phi: V(G) \rightarrow \{1, 2, \ldots, k\}$

**P:** $k$

**Q:** Does there exist a connected subgraph of $G$ that contains exactly one vertex of each color?
**Colorful Graph Motif**

**I:** Graph $G$ and a coloring function $\phi : V(G) \rightarrow \{1, 2, \ldots, k\}$

**P:** $k$

**Q:** Does there exists a connected subgraph of $G$ that contains exactly one vertex of each color?
The problem is \textbf{NP}-hard even on trees.
About CGM

- The problem is \textbf{NP}-hard even on trees.
- \textbf{FPT} algorithms for various variants using the algebraic approach.
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 \( \text{NP} \subseteq \text{coNP} / \text{poly} \).

Now: PPT from CGM to ST.
The problem is \textbf{NP}-hard even on trees.

FPT algorithms for various variants using the algebraic approach.

\textbf{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.

\textbf{Corollary}: no polykernel for \textbf{CGM} unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. 

Now: PPT from \textbf{CGM} to \textbf{ST}. 

The problem is \textbf{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 \textbf{NP} \subseteq \textbf{coNP}/poly.
The problem is \textbf{NP}-hard even on trees.

- FPT algorithms for various variants using the algebraic approach.
- \textbf{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.

\textbf{Corollary}: no polykernel for CGM unless \textbf{NP} \subseteq \textbf{coNP}/poly.

\textbf{Now}: PPT from CGM to ST.
From CGM to ST

Attach a terminal to every color class.

Give budget $k$ for connecting nodes.

Michał Pilipczuk

Kernelization lower bounds

29/33
Attach a terminal to every color class.

Give budget $k$ for connecting nodes.
Attach a terminal to every color class.

Give budget $k$ for connecting nodes.
\textbf{CGM: wrap up}

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

\textit{Note}: Composition for CGM is far simpler than trying to do this directly for Steiner Tree. 
CGM: wrap up

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

CGM PPT-reduces to \textsc{Steiner Tree} par. by $k + |T|$.

Hence \textsc{Steiner Tree} par. by $k + |T|$ does not admit a polynomial kernel, unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$. 
CGM has no polynomial kernel, unless \( \textsf{NP} \subseteq \textsf{coNP}/\text{poly} \).

CGM PPT-reduces to \textsc{Steiner Tree par.} by \( k + |T| \).

Hence \textsc{Steiner Tree par.} by \( k + |T| \) does not admit a polynomial kernel, unless \( \textsf{NP} \subseteq \textsf{coNP}/\text{poly} \).

\textbf{Note:} Composition for CGM is far simpler than trying to do this directly for \textsc{Steiner Tree}. 

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

What about replacing it with, say, AND?
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: Treewidth.

**AND-conjecture:**
If 3SAT has an AND-distillation, then NP ⊆ coNP/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.
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: **TREEWIDTH.**

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.
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: TREETREED.

**AND-conjecture:**
If 3SAT has an AND-distillation, then $\text{NP} \subseteq \text{coNP} / \text{poly}$. 
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: Treewidth.

**AND-conjecture:**
If 3SAT has an AND-distillation, then $\text{NP} \subseteq \text{coNP}/\text{poly}$.
- 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.
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: Treewidth.

**AND-conjecture:**
If 3SAT has an AND-distillation, then \( \text{NP} \subseteq \text{coNP}/\text{poly} \).

- 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: Treewidth.
- **AND-conjecture:**
  - If 3SAT has an AND-distillation, then $\mathbf{NP} \subseteq \mathbf{coNP}/\mathbf{poly}$.
    - 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

- **Idea**: Inspect the proof of FS to get precise estimates.
Weak compositions

- **Idea**: Inspect the proof of FS to get precise estimates.
- **Cor**: A framework for lower bounds on kernel sizes.
Weak compositions

- **Idea**: Inspect the proof of FS to get precise estimates.
- **Cor**: A framework for lower bounds on kernel sizes.

### Weak cross-composition

A **weak cross-composition of dimension** $d$ from an unpar. problem $Q$ to a par. problem $L$, is an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$ for some polynomial equivalence relation $\sim$, in time $\text{poly} \left( t + \sum_{i=1}^{t} |x_i| \right)$ produces one instance $(y, k^*)$ such that

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

- **Idea**: Inspect the proof of FS to get precise estimates.
- **Cor**: A framework for lower bounds on kernel sizes.

**Weak cross-composition**

A weak cross-composition of dimension $d$ from an unpar. problem $Q$ to a par. problem $L$, is an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$ for some polynomial equivalence relation $\sim$, in time $\text{poly}(t + \sum_{i=1}^{t} |x_i|)$ produces one instance $(y, k^*)$ such that

- $(y, k^*) \in L$ iff $x_i \in Q$ for at least one $i = 1, 2, \ldots, t$,
- $k^* = \frac{t^{1/d}}{d} \cdot \text{poly}(\max_{i=1}^{t} |x_i|)$.

**Weak cross-composition theorem**

Suppose $\textbf{NP} \not\subseteq \textbf{coNP}/\text{poly}$. If some $\textbf{NP}$-hard problem $Q$ has a cross-composition of dimension $d$ into $L$, then $L$ does not admit a compression into any language $R$ with bitsize $\mathcal{O}(k^{d-\varepsilon})$ for any $\varepsilon > 0$. 

---

Ex: Vertex Cover has no compression into bitsize $\mathcal{O}(k^{2-\varepsilon})$.

Note: The $2^k$-kernel for VC needs $\mathcal{O}(k^2)$ bits for the encoding.
Weak compositions

- **Idea**: Inspect the proof of FS to get precise estimates.
- **Cor**: A framework for lower bounds on kernel sizes.

### Weak cross-composition

A **weak cross-composition of dimension** $d$ from an unpar. problem $Q$ to a par. problem $L$, is an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$ for some polynomial equivalence relation $\sim$, in time $\text{poly} \left( t + \sum_{i=1}^{t} |x_i| \right)$ produces one instance $(y, k^*)$ such that

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

### Weak cross-composition theorem

Suppose $\textbf{NP} \not\subset \textbf{coNP}/\text{poly}$. If some $\textbf{NP}$-hard problem $Q$ has a cross-composition of dimension $d$ into $L$, then $L$ does not admit a compression into any language $R$ with bitsize $\mathcal{O}(k^{d-\varepsilon})$ for any $\varepsilon > 0$.

- **Ex**: **Vertex Cover** has no compression into bitsize $\mathcal{O}(k^{2-\varepsilon})$. 

---

Michał Pilipczuk  
Kernelization lower bounds  
32/33
Weak compositions

- **Idea**: Inspect the proof of FS to get precise estimates.
- **Cor**: A framework for lower bounds on kernel sizes.

**Weak cross-composition**

A **weak cross-composition of dimension** $d$ from an unpar. problem $Q$ to a par. problem $L$, is an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$ for some polynomial equivalence relation $\sim$, in time \( \text{poly} \left( t + \sum_{i=1}^{t} |x_i| \right) \) produces one instance \((y, k^*)\) such that
- \((y, k^*) \in L\) iff \(x_i \in Q\) for at least one \(i = 1, 2, \ldots, t\),
- \(k^* = t^{1/d} \cdot \text{poly} \left( \max_{i=1}^{t} |x_i| \right) \).

**Weak cross-composition theorem**

Suppose \(\text{NP} \not\subseteq \text{coNP}/\text{poly} \). If some \(\text{NP}\)-hard problem $Q$ has a cross-composition of dimension $d$ into $L$, then $L$ does not admit a compression into any language $R$ with bitsize \(O(k^{d-\varepsilon})\) for any \(\varepsilon > 0\).

- **Ex**: **Vertex Cover** has no compression into bitsize \(O(k^{2-\varepsilon})\).
- **Note**: The 2$k$-kernel for **VC** needs \(O(k^2)\) bits for the encoding.
**Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is *unbounded choice*.
Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.
Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is *unbounded choice*.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.

- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size \( \text{poly}(k) \).
**Conclusions**

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is *unbounded choice*.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.

- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\text{poly}(k)$.
  - Composition framework does not apply.
Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.

- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\text{poly}(k)$.
  - Composition framework does not apply.
  - There are problems that have polynomial Turing kernels, but no polynomial kernel under $\text{NP} \not\subseteq \text{coNP}/\text{poly}$.

Thank you for your attention!

Tikz faces based on a code by Raoul Kessels, http://www.texample.net/tikz/examples/emoticons/, under Creative Commons Attribution 2.5 license (CC BY 2.5)
**Composition**: a versatile framework for proving lower bounds for polynomial kernelization.

- **Message 1**: What is hard for kernelization is **unbounded choice**.
- **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.

**Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\text{poly}(k)$.

- Composition framework does not apply.
- There are problems that have polynomial Turing kernels, but no polynomial kernel under $\text{NP} \not\subset \text{coNP}/\text{poly}$.
- **Open**: A technique for ruling out Turing kernels.
**Conclusions**

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.

- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\text{poly}(k)$.
  - Composition framework does not apply.
  - There are problems that have polynomial Turing kernels, but no polynomial kernel under $\text{NP} \not\subset \text{coNP}/\text{poly}$.
  - **Open**: A technique for ruling out Turing kernels.

- **Complexity theory for kernelization**: Using PPT as reductions, one can build a hierarchy of complexity classes [HKSWW].

Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.

- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\text{poly}(k)$.
  
  - Composition framework does not apply.
  
  - There are problems that have polynomial Turing kernels, but no polynomial kernel under $\text{NP} \not\subseteq \text{coNP}/\text{poly}$.

  - **Open**: A technique for ruling out Turing kernels.

- **Complexity theory for kernelization**: Using PPT as reductions, one can build a hierarchy of complexity classes [HKSWW].

- **Thank you for your attention!**

Tikz faces based on a code by Raoul Kessels, http://www.texample.net/tikz/examples/emoticons/, under Creative Commons Attribution 2.5 license (CC BY 2.5)