Lower bounds for polynomial kernelization
A very quick introduction

Michał Pilipczuk

Institutt for Informatikk, Universitetet i Bergen

13th September 2012
Outline

- **Compositionality**: why polynomial kernels can be implausible?
  - The OR-compression theorem of Fortnow and Santhanam
  - OR-composition (Bodlaender, Downey, Fellows, Hermelin)

- **The toolbox**: the art of (composition) gadgeteering
  - Cross-composition (Bodlaender, Jansen, Kratsch)
  - Case study: **Set Splitting, Set Cover**
  - PPT reductions: **Steiner Tree**

- **Structural parameters**: **Clique** parameterized by **VC**

- **Miscellaneous**:
  - Weak compositions (Dell, Hermelin, van Melkebeek, Marx, Wu)
  - AND-Compositions
Kernelization — recap
instance of $L$
Kernelization — recap

instance of $L$
Kernelization — recap

instance of $L$

$P$-time

Michał Pilipczuk

No-poly-kernels tutorial
Kernelization — recap

instance of $L$ \hspace{1cm} P-time \hspace{1cm} instance of $L$

\begin{align*}
\text{size} \leq f(k)
\end{align*}
If a decidable problem admits a kernelization algorithm, then it is FPT.
If a decidable problem admits a kernelization algorithm, then it is FPT.

Any FPT problem admits a kernelization algorithm:
Kernelization and FPT

- If a decidable problem admits a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - If $|I| \leq f(k)$, then we already have a kernel.
If a decidable problem admits a kernelization algorithm, then it is FPT.

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

If a decidable problem admits a kernelization algorithm, then it is FPT.

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

We are interested in **polynomial kernels**, where $f$ is a polynomial.
 Kernelization and FPT

- If a decidable problem admits a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - If $|I| \leq f(k)$, then we already have a kernel.
  - Otherwise $f(k) \cdot |I|^c = O(|I|^{c+1})$.
- We are interested in **polynomial kernels**, where $f$ is a polynomial.
- Since 90. many kernels constructed for various problems;
Kernelization and FPT

- If a decidable problem admits a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - If $|I| \leq f(k)$, then we already have a kernel.
  - Otherwise $f(k) \cdot |I|^c = O(|I|^{c+1})$.
- We are interested in **polynomial kernels**, where $f$ is a polynomial.
- Since 90. many kernels constructed for various problems;
- however, before 2008 virtually no tool to show that polynomial kernelization is impossible.
Kernelization vs. Compression

KERNELIZATION

k instance of \( L \)-time instance of \( L \) size \( \not\in \mathbb{P} \( (k) \)

COMPRESSION

k instance of \( L \)-time \( R \) (any) size \( \not\in \mathbb{P} \( (k) \)

Michał Pilipczuk
No-poly-kernels tutorial
Kernelization vs. Compression

**KERNELIZATION**

\[ k \]\text{-}instance of \( L \) \hspace{2cm} P\text{-}time \hspace{2cm} \text{instance of } L \]

\[\text{size} \leq p(k)\]
Kernelization vs. 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 vs. Compression

**KERNELIZATION**

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

\[\text{size } \leq p(k)\]

**COMPRESSION**

instance of $L$ \[\xrightarrow{P\text{-time}}\] instance of $R$ (any)

\[\text{size } \leq p(k)\]
The OR-compression theorem

**OR-SAT**

- **Input:** formulas $\phi_1, \phi_2, \ldots, \phi_t$, each of size at most $k$.
- **Parameter:** $k$
- **Question:** Is at least one of $\phi_1, \phi_2, \ldots, \phi_t$ satisfiable?
The OR-compression theorem

**OR-SAT**

**Input:** formulas $\phi_1, \phi_2, \ldots, \phi_t$, each of size at most $k$.

**Parameter:** $k$

**Question:** Is at least one of $\phi_1, \phi_2, \ldots, \phi_t$ satisfiable?

**OR-compression theorem**

OR-SAT does not admit a polynomial compression algorithm, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.  

Fortnow, Santhanam; STOC 2008, JCSS 2011
The **OR**-compression theorem

**OR-SAT**

| Input:  | formulas $\phi_1, \phi_2, \ldots, \phi_t$, each of size at most $k$. |
| Parameter: | $k$ |
| Question: | Is at least one of $\phi_1, \phi_2, \ldots, \phi_t$ satisfiable? |

**OR**-compression theorem

**OR-SAT** does not admit a polynomial compression algorithm, unless $\text{NP} \subseteq \text{coNP/poly}$.

**Corollary**

For any **NP**-hard problem $L$, **OR**-$L$ does not admit a polynomial compression algorithm unless $\text{coNP} \subseteq \text{NP/poly}$.
A glimpse into the proof

- The proof is purely information-theoretical.
A glimpse into the proof

- The proof is purely information-theoretical.
- Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.
A glimpse into the proof

- The proof is purely information-theoretical.
- Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.
- An algorithm in \( \mathcal{P} \) cannot guess, which instance is more prone to have a positive answer, so we need to store information about all of them.
A glimpse into the proof

- The proof is purely information-theoretical.
- Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.
- An algorithm in $\mathbf{P}$ cannot guess, which instance is more prone to have a positive answer, so we need to store information about all of them.
- **Main trick:**
A glimpse into the proof

- The proof is purely information-theoretical.
- Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.
- An algorithm in P cannot guess, which instance is more prone to have a positive answer, so we need to store information about all of them.
- **Main trick:**
  - show that the space for kernels is so small that one can find a linear number of representative kernels;
The proof is purely information-theoretical.

Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.

An algorithm in P cannot guess, which instance is more prone to have a positive answer, so we need to store information about all of them.

**Main trick:**
- show that the space for kernels is so small that one can find a linear number of *representative* kernels;
- plug these kernels as the advice to a coNP-algorithm for SAT.
The proof is purely information-theoretical.

Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.

An algorithm in $\mathsf{P}$ cannot guess, which instance is more prone to have a positive answer, so we need to store information about all of them.

**Main trick:**
- show that the space for kernels is so small that one can find a linear number of *representative* kernels;
- plug these kernels as the advice to a $\mathsf{coNP}$-algorithm for SAT.

**It is literally a half-a-page proof.**
A composition algorithm for a parameterized language $L$ is an algorithm that takes $t$ instances $(x_1, k), (x_2, k), \ldots, (x_t, k)$, and in $\text{poly}(\sum_{i=1}^{t} |x_i| + k)$ time returns one instance $(y, k')$ such that

- $k' = \text{poly}(k)$,
- $(y, k') \in L$ if and only if at least one of $(x_i, k) \in L$.
OR-composition on picture
OR-composition on picture

$t$ instances

\[ \text{instances} \]
OR-composition on picture
OR-composition on picture

$t$ instances

$P$-time

$k'$
**OR-composition theorem**

If a parameterized problem $L$ admits a composition algorithm, and the unparameterized version of $L$ is $\mathsf{NP}$-complete, then $L$ does not admit a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{coNP}/\mathsf{poly}$. 

Bodlaender, Downey, Fellows, Hermelin; ICALP 2008, JCSS 2009
Proof
Proof

OR-SAT

L

NP

hrd

1

2

k

k'

L

cmp
poly
(k)

cmp
poly
(k)

cmp
poly
(k)

OR-
L

Michał Pilipczuk  No-poly-kernels tutorial  11/31
Proof
Proof
Proof

\[
\begin{align*}
\text{OR-SAT} & \rightarrow \text{NP-hrd} \\
\text{NP-hrd}^1 & \rightarrow \text{NP-hrd}^1 \\
\text{NP-hrd}^2 & \rightarrow \text{NP-hrd}^2 \\
\text{NP-hrd}^{k'} & \rightarrow \text{NP-hrd}^{k'} \\
\text{L} & \rightarrow \text{cmp} \\
\text{cmp} & \rightarrow \text{poly}(k) \\
\end{align*}
\]
Proof

\[ \text{OR-SAT} \]

\[ L \]

\[ \text{cmp} \]

\[ \text{poly}(k) \]

\[ \text{kern} \]

\[ \text{Michał Pilipczuk} \]

No-poly-kernels tutorial
Proof

OR-SAT

L

k

k'

poly(k)

kern

L

OR-L

Michał Pilipczuk

No-poly-kernels tutorial

11/31
Corollaries

- \texttt{k-Path} does not admit poly-kernel, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$. 
Corollaries

- \textit{k-Path} does not admit poly-kernel, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.
- \textbf{Composition:} Take disjoint union of graphs and the same parameter.
**k-Path** does not admit poly-kernel, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.

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

This opens a bag of results.
Corollaries

- **k-Path** does not admit poly-kernel, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.
- **Composition**: Take disjoint union of graphs and the same parameter.
- This opens a bag of results.
- Now, investigating possibility of existence of a polynomial kernel is an immediate second goal after showing that a problem is FPT.
The theorem of Fortnow and Santhanam actually excludes compression algorithms working in $\text{coNP}/\text{poly}$.
The theorem of Fortnow and Santhanam actually excludes compression algorithms working in $\text{coNP}/\text{poly}$.

Hence, the composition algorithm may in fact work in $\text{coNP}$. 

Sometimes you can see this in papers.
A few comments

- The theorem of Fortnow and Santhanam actually excludes compression algorithms working in $\text{coNP}/\text{poly}$.
- Hence, the composition algorithm may in fact work in $\text{coNP}$.
- Can be useful (Kratsch; SODA 2012).
The theorem of Fortnow and Santhanam actually excludes compression algorithms working in $\text{coNP}/\text{poly}$.

Hence, the composition algorithm may in fact work in $\text{coNP}$.

Can be useful (Kratsch; SODA 2012).

Composition excludes also $\text{coNP}$-kernelization, in particular $\text{coRP}$.
A few comments

- The theorem of Fortnow and Santhanam actually excludes compression algorithms working in $\text{coNP}/\text{poly}$.
- Hence, the composition algorithm may in fact work in $\text{coNP}$.
- Can be useful (Kratsch; SODA 2012).
- Composition excludes also $\text{coNP}$-kernelization, in particular $\text{coRP}$.
- $\text{NP} \subseteq \text{coNP}/\text{poly}$ implies that $\text{PH} = \Sigma_3^p$. 
A few comments

- The theorem of Fortnow and Santhanam actually excludes compression algorithms working in \( \text{coNP/poly} \).
- Hence, the composition algorithm may in fact work in \( \text{coNP} \).
- Can be useful (Kratsch; SODA 2012).
- Composition excludes also \( \text{coNP} \)-kernelization, in particular \( \text{coRP} \).
- \( \text{NP} \subseteq \text{coNP/poly} \) implies that \( \text{PH} = \Sigma_3^p \).
- Sometimes you can see this in papers.
Do we need $\text{NP}$-completeness of $\tilde{L}$?
Adding features

- Do we need \( \mathbf{NP} \)-completeness of \( \tilde{L} \)?
  - No, just \( \mathbf{NP} \)-hardness.
Adding features

- Do we need \textbf{NP}-completeness of $\tilde{L}$?
  - No, just \textbf{NP}-hardness.

- Do we need to start with the same language as $L$?
Adding features

- Do we need \( \text{NP} \)-completeness of \( \tilde{L} \)?
  - No, just \( \text{NP} \)-hardness.

- Do we need to start with the same language as \( L \)?
  - No, we can start from any \( \text{NP} \)-hard language; we even do not need a parameter.
**Adding features**

- Do we need $\textbf{NP}$-completeness of $\tilde{L}$?
  - No, just $\textbf{NP}$-hardness.

- Do we need to start with the same language as $L$?
  - No, we can start from any $\textbf{NP}$-hard language; we even do not need a parameter.

- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
Adding features

- Do we need $\text{NP}$-completeness of $\tilde{L}$?
  - No, just $\text{NP}$-hardness.

- Do we need to start with the same language as $L$?
  - No, we can start from any $\text{NP}$-hard language; we even do not need a parameter.

- 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 more features

- How large can $t$ be?
Adding more features

- How large can $t$ be?
- Well, not larger than $(|\Sigma| + 1)^k$, as we may remove duplicates of formulas.
Adding more features

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

- How large can $t$ be?
- Well, not larger than $(|\Sigma| + 1)^k$, as we may remove duplicates of formulas.
- Hence, we may assume that $\log t = O(k)$,
- which means that 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; ICALP 2009): We apply single-exponential FPT algorithm.
Adding more features

- How large can $t$ be?
- Well, not larger than $(|\Sigma| + 1)^k$, as we may remove duplicates of formulas.
- Hence, we may assume that $\log t = O(k)$,
- which means that 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; ICALP 2009):
Adding more features

- How large can $t$ be?
- Well, not larger than $(|\Sigma| + 1)^k$, as we may remove duplicates of formulas.
- Hence, we may assume that $\log t = O(k)$,
- which means that 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; ICALP 2009):
  - We apply single-exponential FPT algorithm.
A huge amount of no-poly-kernel results.
After invention of the composition framework

- A huge amount of no-poly-kernel results.
- Among them, the work of Dom, Lokshtanov, and Saurabh:
After invention of the composition framework

- A huge amount of no-poly-kernel results.
- Among them, the work of Dom, Lokshtanov, and Saurabh:
  - Basic covering problems: Hitting Set/universe size, Set Cover/universe size, Steiner Tree/budget
A huge amount of no-poly-kernel results.

Among them, the work of Dom, Lokshtanov, and Saurabh:
- Basic covering problems: **Hitting Set**/universe size, **Set Cover**/universe size, **Steiner Tree**/budget
- In fact introduces an **instance selector** (calls it IDs).
After invention of the composition framework

- A huge amount of no-poly-kernel results.
- Among them, the work of Dom, Lokshtanov, and Saurabh:
  - Basic covering problems: \textsc{Hitting Set}/universe size, \textsc{Set Cover}/universe size, \textsc{Steiner Tree}/budget
  - In fact introduces an \textit{instance selector} (calls it IDs).
- Most of the works use a subset of mentioned features.
A huge amount of no-poly-kernel results.

Among them, the work of Dom, Lokshtanov, and Saurabh:
- Basic covering problems: **Hitting Set**/universe size, **Set Cover**/universe size, **Steiner Tree**/budget
- In fact introduces an **instance selector** (calls it IDs).

Most of the works use a subset of mentioned features.

**STACS 2011**: Bodlaender, Jansen, and Kratsch propose a new formalism, dubbed **cross-composition**, that gathers all these features.
An equivalence relation $\mathcal{R}$ on $\Sigma^*$ is called a polynomial equivalence relation if the following two conditions hold:

- Checking whether two strings $x, y \in \Sigma^*$ are $\mathcal{R}$-equivalent can be done in $\text{poly}(|x| + |y|)$ time.
- $\mathcal{R}$ partitions strings of length at most $n$ into $\text{poly}(n)$ equivalence classes.
An equivalence relation $\mathcal{R}$ on $\Sigma^*$ is called a polynomial equivalence relation if the following two conditions hold:

- Checking whether two strings $x, y \in \Sigma^*$ are $\mathcal{R}$-equivalent can be done in $\text{poly}(|x| + |y|)$ time.
- $\mathcal{R}$ partitions strings of length at most $n$ into $\text{poly}(n)$ equivalence classes.

**Examples:**
An equivalence relation $\mathcal{R}$ on $\Sigma^*$ is called a polynomial equivalence relation if the following two conditions hold:

- Checking whether two strings $x, y \in \Sigma^*$ are $\mathcal{R}$-equivalent can be done in $\text{poly}(|x| + |y|)$ time.
- $\mathcal{R}$ partitions strings of length at most $n$ into $\text{poly}(n)$ equivalence classes.

**Examples:**
- partitioning with respect to the number of vertices of the graph;
An equivalence relation $R$ on $\Sigma^*$ is called a polynomial equivalence relation if the following two conditions hold:

- Checking whether two strings $x, y \in \Sigma^*$ are $R$-equivalent can be done in $\text{poly}(|x| + |y|)$ time.
- $R$ partitions strings of length at most $n$ into $\text{poly}(n)$ equivalence classes.

Examples:

- 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.
We say that an unparameterized problem $R$ \emph{cross-composes} into a parameterized problem $L$, if there exists a polynomial equivalence relation $\mathcal{R}$ and an algorithm, that given $t$ $\mathcal{R}$-equivalent strings $x_1, x_2, \ldots, x_t$, in time $\text{poly} \left( \sum_{i=1}^{t} |x_i| \right)$ produces one instance $(y, k^*)$ such that

- $(y, k^*) \in L$ if and only if $x_i \in L$ for at least one $i = 1, 2, \ldots, t$,
- $k^* = \text{poly} \left( \log t + \max_{i=1}^{t} |x_i| \right)$.
Cross-composition

We say that an unparameterized problem \( R \) \textit{cross-composes} into a parameterized problem \( L \), if there exists a polynomial equivalence relation \( \mathcal{R} \) and an algorithm, that given \( t \) \( \mathcal{R} \)-equivalent strings \( x_1, x_2, \ldots, x_t \), in time \( \text{poly} \left( \sum_{i=1}^{t} |x_i| \right) \) produces one instance \((y, k^*)\) such that

- \((y, k^*) \in L\) if and only if \( x_i \in L \) 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

If some \( \textbf{NP} \)-hard problem \( R \) cross-composes to \( L \), then \( L \) does not admit a polynomial compression unless \( \textbf{NP} \subseteq \textbf{coNP}/\text{poly} \).
Applications

- 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.
Applications

- 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 a few cross-compositions and give intuition about basic tricks.
Application 1: Set Splitting

Set Splitting

Input: Universe \( U \) and family of subsets \( F \subseteq 2^U \)

Parameter: \( |U| \)

Question: Does there exist a colouring \( C : U \rightarrow \{B, W\} \) such that every set \( X \in F \) is split, i.e., contains a black and a white element?
Application 1: **Set Splitting**

**Set Splitting**

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

**Parameter:** $|U|$  

**Question:** Does there exist a colouring $\mathcal{C} : U \rightarrow \{B, W\}$ 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**

**Set Splitting**

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

**Parameter:** $|U|$

**Question:** Does there exist a colouring $C : U \rightarrow \{B, W\}$ 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.
**Application 1: Set Splitting**

<table>
<thead>
<tr>
<th><strong>Set Splitting</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Universe $U$ and family of subsets $\mathcal{F} \subseteq 2^U$</td>
</tr>
<tr>
<td><strong>Parameter:</strong> $</td>
</tr>
<tr>
<td><strong>Question:</strong> Does there exist a colouring $\mathcal{C} : U \rightarrow {B, W}$ such that every set $X \in \mathcal{F}$ is split, i.e., contains a black and a white element?</td>
</tr>
</tbody>
</table>

- 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 **Set Splitting**

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

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

**INSTANCE SELECTOR**

1 + log \(t\) pairs of vertices

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

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

**PLAYGROUND**

joint universe \(U\)

Michał Pilipczuk  
No-poly-kernels tutorial  
21/31
Cross-composing into \textit{Set Splitting}

**INSTANCE SELECTOR**

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

\[\mathcal{F}^* \text{ consists of:}\]

\[1 + \log t \text{ pairs of vertices}\]

**PLAYGROUND**

\[\text{joint universe } U\]
Cross-composing into **Set Splitting**

**INSTANCE SELECTOR**

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

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

\(F^*\) consists of:

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

---

**PLAYGROUND**

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

**INSTANCE SELECTION**

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

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

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

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

**PLAYGROUND**

joint universe \(U\)

Michał Pilipczuk  No-poly-kernels tutorial
Cross-composing into \textbf{Set Splitting}

\textbf{INSTANCE SELECTOR}

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

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

$\mathcal{F}^*$ 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 guy,

and binary encoding of $i$ in $IS$

\begin{itemize}
  \item PLAYGROUND
  \item joint universe $U$
\end{itemize}
Cross-composing into **Set Splitting**

**INSTANCE SELECTOR**

1 + log \( t \) pairs of vertices

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

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

\(\mathcal{F}^*\) 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 guy, and binary encoding of \(i\) in IS

\(X_1^*\): reverse \(X_0^*\) on IS

**PLAYGROUND**

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

**INSTANCE SELECTION**

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

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

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

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

\(\forall X \in \mathcal{F}^i\), two sets \(X_0^*, X_1^*\)

Playground

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

**INSTANCE SELECTOR**

1 + log \( t \) pairs of vertices

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

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

\(F^*\) 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 \((U^*, \mathcal{F}^*)\)

\(\mathcal{F}^*\) 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

INSTANCE SELECTOR

1 + log t pairs of vertices

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

\(\mathcal{F}^*\) 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

**INSTANCE SELECTOR**

1 + log \( t \) pairs of vertices

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

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

\(\mathcal{F}^*\) 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 \(PL\); remaining sets are split for free.

**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 \((U^*, \mathcal{F}^*)\)

\(\mathcal{F}^*\) 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 \(PL\); remaining sets are split for free.

\((\Leftarrow)\): If \((U, \mathcal{F}^i)\) is solvable, we set \(IS\) accordingly, and solve this instance in \(PL\).
## Application 2: Set Cover

### Set Cover

**Input:** Universe $U$, a family of subsets $\mathcal{F} \subseteq 2^U$, integer $k$

**Parameter:** $|U|$

**Question:** Can you find a subfamily $\mathcal{G} \subseteq \mathcal{F}$, $|\mathcal{G}| \leq k$, such that $\bigcup \mathcal{G} = U$?

---

We need a few more tricks.

Convention: We view it as a bipartite graph with one side (blue) trying to dominate the other one (red). W.l.o.g. $k \leq |U|$.
Application 2: Set Cover

**Set Cover**

**Input:** Universe $U$, a family of subsets $\mathcal{F} \subseteq 2^U$, integer $k$

**Parameter:** $|U|$

**Question:** Can you find a subfamily $\mathcal{G} \subseteq \mathcal{F}$, $|\mathcal{G}| \leq k$, such that $\bigcup \mathcal{G} = U$?

- **Source:** Dom, Lokshtanov, and Saurabh.
Application 2: **Set Cover**

**Set Cover**

**Input:** Universe $U$, a family of subsets $\mathcal{F} \subseteq 2^U$, integer $k$

**Parameter:** $|U|$

**Question:** Can you find a subfamily $\mathcal{G} \subseteq \mathcal{F}$, $|\mathcal{G}| \leq k$, such that $\bigcup \mathcal{G} = U$?

- **Source:** Dom, Lokshtanov, and Saurabh.
- **We need a few more tricks.**
**Application 2: Set Cover**

**Set Cover**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Universe $U$, a family of subsets $\mathcal{F} \subseteq 2^U$, integer $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter:</td>
<td>$</td>
</tr>
<tr>
<td>Question:</td>
<td>Can you find a subfamily $\mathcal{G} \subseteq \mathcal{F}$, $</td>
</tr>
</tbody>
</table>

- **Source:** Dom, Lokshtanov, and Saurabh.
- We need a few more tricks.
- **Convention:** We view it as a bipartite graph with one side (blue) trying to dominate the other one (red).
## Application 2: Set Cover

**Set Cover**

**Input:** Universe $U$, a family of subsets $\mathcal{F} \subseteq 2^U$, integer $k$

**Parameter:** $|U|$

**Question:** Can you find a subfamily $\mathcal{G} \subseteq \mathcal{F}$, $|\mathcal{G}| \leq k$, such that $\bigcup \mathcal{G} = U$?

- **Source:** Dom, Lokshtanov, and Saurabh.
- We need a few more tricks.
- **Convention:** We view it as a bipartite graph with one side (blue) trying to dominate the other one (red).
- **W.l.o.g.** $k \leq |U|$. 
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 $(y, k')$ of $Q$ such that $k' = \text{poly}(k)$.
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 $(y, k')$ of $Q$ such that $k' = \text{poly}(k)$.

Observation

If $P$ PPT-reduces to $Q$ and $P$ does not admit a polynomial compression algorithm, then neither does $Q$. 
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 $(y, k')$ of $Q$ such that $k' = \text{poly}(k)$.

Observation

If $P$ PPT-reduces to $Q$ and $P$ does not admit a polynomial compression algorithm, then neither does $Q$.

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

Michał Pilipczuk 
No-poly-kernels tutorial
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 $(y, k')$ of $Q$ such that $k' = \text{poly}(k)$.

**Observation**

If $P$ PPT-reduces to $Q$ and $P$ does not admit a polynomial compression algorithm, then neither does $Q$.

- **Proof**: Compose the PPT-reduction with the assumed compression for $Q$.
- **Idea**: One can start with an easier problem, for which it is easier to design a composition.
Input: Universe $U$ and families $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_k \subseteq 2^U$

Parameter: $|U| + k$

Question: Can you find a family $\mathcal{G}$ containing exactly one set from each family $\mathcal{F}_i$, such that $\bigcup \mathcal{G} = U$?
Equivalence of the problems

\[ SC \leq_{PPT} CSC:\]

- For every \( i \), set \( F_i = F \).
- Add \( k \) elements \( e_1, e_2, \ldots, e_k \); include \( e_i \) in every set from \( F_i \).
- Take \( F = \bigcup F_i \).

We need just the second reduction.

We will cross-compose Colourful Set Cover into itself.

Assume: the same universe \( U \), the same \( k \), and \( t \) being a power of 2.

Michał Pilipczuk

No-poly-kernels tutorial

25/31
Equivalence of the problems

\[ \text{SC} \leq_{PPT} \text{CSC}: \]
- Put \( \mathcal{F}_i = \mathcal{F} \) for every \( i \).
Equivalence of the problems

\[ SC \leq_{PPT} CSC: \]
- Put \( F_i = \mathcal{F} \) for every \( i \).

\[ CSC \leq_{PPT} SC: \]
Equivalence of the problems

- SC $\leq_{PPT} CSC$:
  - Put $F_i = F$ for every $i$.

- CSC $\leq_{PPT} SC$:
  - Add $k$ elements $e_1, e_2, \ldots, e_k$; include $e_i$ in every set from $F_i$. 

We need just the second reduction.

We will cross-compose Colourful Set Cover into itself.

Assume:
- the same universe $U$,
- the same $k$,
- and $t$ being a power of 2.
Equivalence of the problems

- $\text{SC} \leq_{\text{PPT}} \text{CSC}$:
  - Put $\mathcal{F}_i = \mathcal{F}$ for every $i$.

- $\text{CSC} \leq_{\text{PPT}} \text{SC}$:
  - Add $k$ elements $e_1, e_2, \ldots, e_k$; include $e_i$ in every set from $\mathcal{F}_i$.
  - Then take $\mathcal{F} = \bigcup \mathcal{F}_i$. 

We need just the second reduction. We will cross-compose Colourful Set Cover into itself. Assume: the same universe $U$, the same $k$, and $t$ being a power of 2.
Equivalence of the problems

- $SC \leq_{PPT} CSC$: 
  - Put $F_i = F$ for every $i$.

- $CSC \leq_{PPT} SC$: 
  - Add $k$ elements $e_1, e_2, \ldots, e_k$; include $e_i$ in every set from $F_i$.
  - Then take $F = \bigcup F_i$.

- We need just the second reduction.
Equivalence of the problems

- **$SC \leq_{PPT} CSC$:**
  - Put $F_i = F$ for every $i$.

- **$CSC \leq_{PPT} SC$:**
  - Add $k$ elements $e_1, e_2, \ldots, e_k$; include $e_i$ in every set from $F_i$.
  - Then take $F = \bigcup F_i$.

- We need just the second reduction.

- We will cross-compose **Colourful Set Cover** into itself.
Equivalence of the problems

- $SC \leq_{PPT} CSC$:
  - Put $\mathcal{F}_i = \mathcal{F}$ for every $i$.

- $CSC \leq_{PPT} SC$:
  - Add $k$ elements $e_1, e_2, \ldots, e_k$; include $e_i$ in every set from $\mathcal{F}_i$.
  - Then take $\mathcal{F} = \bigcup \mathcal{F}_i$.

We need just the second reduction.

We will cross-compose Colourful Set Cover into itself.

Assume: the same universe $U$, the same $k$, and $t$ being a power of 2.
Cross-composing into **Colourful Set Cover**

**Input:** Instances \((U, (\mathcal{F}_j^i)_{1 \leq j \leq k})\)

**Output:** Instance \((U^*, (\mathcal{F}_j^*)_{1 \leq j \leq k})\)
Cross-composing into Colourful Set Cover

Input: Instances \((U, (F_j^i)_{1 \leq j \leq k})\)

Output: Instance \((U^*, (F_j^*)_{1 \leq j \leq k})\)
Cross-composing into Colourful Set Cover

Input: Instances \((U, (F_j^1)_{1 \leq j \leq k})\)

Output: Instance \((U^*, (F_j^*)_{1 \leq j \leq k})\)
Cross-composing into **Colourful Set Cover**

**Input:** Instances \( (U, (F^i_j)_{1 \leq j \leq k}) \)

**Output:** Instance \( (U^*, (F^*_j)_{1 \leq j \leq k}) \)

Problem: Ensure consistent instance choice

Equality gadgets for \( i < j \) make a gadget \( U \log t \) pairs \((3, 4)\)

Add \( bin(i) \) to sets from \( F_i \)

Add \( bin(i) \) to sets from \( F_i \)

Eq. gadgets are covered \( \iff \) Instance choices are equal

New parameter: \( |U| + O(k^2 \log t) \)
Cross-composing into **Colourful Set Cover**

**Input:** Instances $\left( U, (F_j^i)_{1 \leq j \leq k} \right)$

**Output:** Instance $\left( U^*, (F_j^*)_{1 \leq j \leq k} \right)$

**Problem:** Ensure consistent instance choice

**Solution:** Equality gadgets

- $\forall i < j$ make a gadget $U \log t$ pairs $(3, 4)$
- $\forall i$ add $bin(i)$ to sets from $F_i$

Eq. gadgets are covered $\iff$ Instance choices are equal

New parameter: $|U| + O(k^2 \log t)$

Michał Pilipczuk

No-poly-kernels tutorial
Cross-composing into Colourful Set Cover

Input: Instances \((U, (\mathcal{F}_j^i)_{1 \leq j \leq k})\)

Output: Instance \((U^*, (\mathcal{F}_j^*)_{1 \leq j \leq k})\)
Cross-composing into Colourful Set Cover

Input: Instances \( U, (\mathcal{F}_j^i)_{1 \leq j \leq k} \)

Output: Instance \( (U^*, (\mathcal{F}_j^*)_{1 \leq j \leq k}) \)

Problem: Ensure consistent instance choice
Solution: Equality gadgets

\[ \forall i < j \text{ make a gadget } \]

\[ \forall i \text{ add bin } (i) \text{ to sets from } \mathcal{F}_i \]

Equality gadgets are covered \( \iff \) Instance choices are equal

New parameter: \( |U| + O(k^2 \log t) \)
Cross-composing into **Colourful Set Cover**

**Input**: Instances $(U, (F_i^j)_{1 \leq j \leq k})$

**Output**: Instance $(U^*, (F_i^* j)_{1 \leq j \leq k})$

∀₁ add bin(i) to sets from $F_i^j$

∀₁ add bin(i) to sets from $F_i^j$

Problem: Ensure consistent instance choice

Solution: Equality gadgets

∀₁ make a gadget

∀₁ add bin(i) to sets from $F_i^j$

∀₁ add bin(i) to sets from $F_i^j$

Eq. gadgets are covered ⇔ Instance choices are equal

New parameter: $|U| + O(k^2 \log t)$

Michał Pilipczuk

No-poly-kernels tutorial

26/31
Cross-composing into **Colourful Set Cover**

**Input**: Instances \((U, (F_i^j)_{1 \leq j \leq k})\)

**Output**: Instance \((U^*, (F_i^j)_{1 \leq j \leq k})\)

- \(\forall i\) add \(bin(i)\) to sets from \(F_i^3\)
- \(\forall i\) add \(\overline{bin(i)}\) to sets from \(F_i^4\)

**Problem**: Ensure consistent instance choice

**Solution**: Equality gadgets

\[\forall i < j \text{ make a gadget} U \log t \text{ pairs} (3, 4)\]

\[\forall i \text{ add } bin(i) \text{ to sets from } F_i^3\]

\[\forall i \text{ add } \overline{bin(i)} \text{ to sets from } F_i^4\]

Eq. gadgets are covered \(\iff\) Instance choices are equal

New parameter: \(|U| + O(k^2 \log t)\)

Michał Pilipczuk

No-poly-kernels tutorial
Cross-composing into **Colourful Set Cover**

**Input**: Instances $\{U, (F^i_j)_{1 \leq j \leq k}\}$

**Output**: Instance $\{U^*, (F^*_j)_{1 \leq j \leq k}\}$

**Problem**: Ensure consistent instance choice

**Solution**: Equality gadgets

- $\forall i < j$ make a gadget $U \log t$ pairs $(3, 4)$
- $\forall i$ add $bin(i)$ to sets from $F^i_3$
- $\forall i$ add $bin(i)$ to sets from $F^i_4$

Eq. gadgets are covered $\iff$ Instance choices are equal

New parameter: $|U| + O(k^2 \log t)$

Michał Pilipczuk
No-poly-kernels tutorial
26/31
Cross-composing into **Colourful Set Cover**

**Input:** Instances \((U, (F_i^j)_{1 \leq j \leq k})\)

**Output:** Instance \((U^*, (F_i^j)^*)_{1 \leq j \leq k}\)

**Problem:** Ensure consistent instance choice

**Solution:** Equality gadgets

\[ \forall i < j \text{ make a gadget } U \log t \text{ pairs } (3, 4) \]

\[ \forall i \text{ add bin}(i) \text{ to sets from } F_i^j \]

\[ \forall i \text{ add bin}(i) \text{ to sets from } F_i^j \]

New parameter: \(|U| + O(k^2 \log t)|
Application 3: **Steiner Tree**

**Steiner Tree**

**Input:** Graph $G$ with designated terminals $T \subseteq V(G)$, and an integer $k$

**Parameter:** $k + |T|$

**Question:** Is there a set $X \subseteq V(G) \setminus T$, such that $|X| \leq k$ and $G[T \cup X]$ is connected?

Follows from an easy reduction from Set Cover (Dom, Lokshtanov, Saurabh). But we will present an alternative approach.

Source: Cygan, Pilipczuk, P, Wojtaszczyk; WG 2010.
**Application 3: Steiner Tree**

<table>
<thead>
<tr>
<th>Steiner Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
</tr>
<tr>
<td><strong>Parameter:</strong></td>
</tr>
<tr>
<td><strong>Question:</strong></td>
</tr>
</tbody>
</table>

- Follows from an easy reduction from **Set Cover** (Dom, Lokshtanov, Saurabh).
Application 3: **Steiner Tree**

**Steiner Tree**

**Input:** Graph $G$ with designated terminals $T \subseteq V(G)$, and an integer $k$

**Parameter:** $k + |T|$

**Question:** Is there a set $X \subseteq V(G) \setminus T$, such that $|X| \leq k$ and $G[T \cup X]$ is connected?

- Follows from an easy reduction from **Set Cover** (Dom, Lokshtanov, Saurabh).
- But we will present an alternative approach.
Application 3: **Steiner Tree**

**Steiner Tree**

**Input:** Graph $G$ with designated terminals $T \subseteq V(G)$, and an integer $k$

**Parameter:** $k + |T|$

**Question:** Is there a set $X \subseteq V(G) \setminus T$, such that $|X| \leq k$ and $G[T \cup X]$ is connected?

- Follows from an easy reduction from **Set Cover** (Dom, Lokshtanov, Saurabh).
- But we will present an alternative approach.
- **Source:** Cygan, Pilipczuk, P, Wojtaszczyk; WG 2010.
The pivot problem technique

- Introduce an easier problem $P$, which is almost trivially compositional.
The pivot problem technique

- Introduce an easier problem $P$, which is almost trivially compositional.
- Move the weight of the proof to the PPT-reduction and actual definition of $P$. 
The pivot problem technique

- Introduce an easier problem $P$, which is almost trivially compositional.
- Move the weight of the proof to the PPT-reduction and actual definition of $P$.
- **Idea**: Extract the hardness of the problem.
**Input:** Graph $G$ and a colouring function $C : V(G) \to \{1, 2, \ldots, k\}$

**Parameter:** $k$

**Question:** Does there exist a connected subgraph $H$ of $G$ containing exactly one vertex of each colour?
Colourful Graph Motif — example
Colourful Graph Motif — example
About CGM

About CGM

- \textbf{NP}-hard even on trees.

\textbf{NP}-hard even on trees.

Trivial composition algorithm: disjoint union of instances.
About CGM

- \textbf{NP}-hard even on trees.
- Trivial composition algorithm: disjoint union of instances.
- \textbf{Now}: PPT-reduction from CGM to ST.
Attach a terminal to every colour class
Give budget for Steiner nodes
From CGM to ST

Attach a terminal to every colour class
Give budget $k$ for Steiner nodes
Attach a terminal to every colour class
Give budget $k$ for Steiner nodes
**Idea**: Parameterize the problem by the quantitative measure of structure of the graph, rather than intended solution size.
Idea: Parameterize the problem by the quantitative measure of structure of the graph, rather than intended solution size.

- Example: Courcelle’s theorem — treewidth parameterization
Idea: Parameterize the problem by the quantitative measure of structure of the graph, rather than intended solution size.

Example: Courcelle’s theorem — treewidth parameterization

From kernelization point of view: work of Bodlaender, Jansen, and Kratsch.
**Idea:** Parameterize the problem by the quantitative measure of structure of the graph, rather than intended solution size.

- **Example:** Courcelle’s theorem — treewidth parameterization

From kernelization point of view: work of Bodlaender, Jansen, and Kratsch.

- **Advantage:** Enables to pin-point and understand, where lies the difficulty of the problem.
Application 4: \textsc{Clique par. by vertex cover}

\textbf{Clique/VC}

\begin{itemize}
    \item \textbf{Input:} Graph $G$, a vertex cover $X$ of $G$, integer $k$
    \item \textbf{Parameter:} $|X|$
    \item \textbf{Question:} Is there a clique of size $k$ in $G$?
\end{itemize}

W.l.o.g. $k / \leq |X| + 1$.

We make a cross-composition from classical Clique problem.

Assume the same number of vertices $n$ and the same asked size of the clique $k$.
Clique/VC

Input: Graph $G$, a vertex cover $X$ of $G$, integer $k$
Parameter: $|X|$
Question: Is there a clique of size $k$ in $G$?

- W.l.o.g. $k \leq |X| + 1$. 
Application 4: CLIQUE par. by vertex cover

**CLIQUE/VC**

**Input:** Graph $G$, a vertex cover $X$ of $G$, integer $k$

**Parameter:** $|X|$

**Question:** Is there a clique of size $k$ in $G$?

- W.l.o.g. $k \leq |X| + 1$.
- We make a cross-composition from classical CLIQUE problem.
Application 4: **Clique** par. by vertex cover

**Clique/VC**

**Input:** Graph $G$, a vertex cover $X$ of $G$, integer $k$

**Parameter:** $|X|$

**Question:** Is there a clique of size $k$ in $G$?

- W.l.o.g. $k \leq |X| + 1$.
- We make a cross-composition from classical **Clique** problem.
- Assume the same number of vertices $n$ and the same asked size of the clique $k$. 
Cross-composing into \textit{Clique}/\textit{VC}

\textbf{Input:} Instances \((G_i, k)\)

\textbf{Output:} Instance \((G, X, k^*)\)
Cross-composing into \textbf{CLIQUE}/\textbf{VC}

\begin{itemize}
  \item \textbf{Input:} Instances \((G_i, k)\)
  \item \textbf{Output:} Instance \((G, X, k^*)\)
\end{itemize}

Size constraints force us to take one vertex from \(I\).
Cross-composing into \textit{Clique}/VC

Input: Instances \((G_i, k)\)

Output: Instance \((G, X, k^*)\)

Size constraints force us to take one vertex from \(I\).

Neighbourhood of \(i\)-th vertex from \(I\) acts as instance \(G_i\).
Cross-composing into \textit{Clique/VC}

**Input:** Instances \((G_i, k)\)

**Output:** Instance \((G, X, k^*)\)

Size constraints force us to take one vertex from \(I\).

Neighbourhood of \(i\)-th vertex from \(I\) acts as instance \(G_i\).

**Problem:**
Design a 'universal' modulator \(X\).
Cross-composing into \textbf{Clique/VC}

\textbf{Input:} Instances \((G_i, k)\)

\textbf{Output:} Instance \((G, X, k^*)\)

All connections are present except ones in the same row/column.

\begin{itemize}
  \item \textbf{Input:} Instances \((G_i, k)\)
  \item \textbf{Output:} Instance \((G, X, k^*)\)
\end{itemize}
Cross-composing into **Clique/VC**

**Input:** Instances \((G_i, k)\)

**Output:** Instance \((G, X, k^*)\)

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(k)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(n\) triples

\(\left(\begin{array}{c} n \\ 2 \end{array}\right)\)
Cross-composing into \textbf{Clique/VC}

\textbf{Input}: Instances \((G_i, k)\)

\textbf{Output}: Instance \((G, X, k^*)\)

\(n\) \(\times\) \(n\) matrix with \(k\) columns.

1. Choose \(i\)-th vertex from \(I\).
2. Neighbourhood of \(i\)-th vertex \(G_i\) acts as instance \(G_i\).
3. All connections are present except ones in the same row/column.
4. \((a, b)\) \(\notin E(G_i)\), columns \(a\) and \(b\) cannot be chosen simultaneously.
5. New parameter: \(|X| = kn + 3\binom{n}{2}\)

\(\binom{n}{2}\) triples

\(\text{(a, b)}\)
Cross-composing into \textit{Clique}/VC

Input: Instances \((G_i, k)\)

Output: Instance \((G, X, k^*)\)

\begin{itemize}
  \item All connections are present except ones in the same row/column.
  \item \((\binom{n}{2})\) triples
  \item \((a, b)\)
  \item \(-a\)
\end{itemize}

Requested size of the clique:
\[ k^* = k + \left(\binom{n}{2}\right) + 1 \]

\(i\)-th vertex is chosen from \(I\) \(\Rightarrow \forall (a, b) \notin E(G_i), \) columns \(a\) and \(b\) cannot be chosen simultaneously

New parameter:
\[ |X| = kn + 3 \left(\binom{n}{2}\right) \]
Cross-composing into \textbf{Clique/VC}

\textbf{Input}: Instances \((G_i, k)\)

\textbf{Output}: Instance \((G, X, k^*)\)

\begin{itemize}
  \item \text{all} \quad \neg a \quad \neg b
  \end{itemize}

\( (a, b) \)

\( \binom{n}{2} \) triples
Cross-composing into \textit{Clique}/VC

\textbf{Input:} Instances \((G_i, k)\)

\textbf{Output:} Instance \((G, X, k^*)\)

\textit{Problem:} Design a ‘universal’ modulator \(X\).

\begin{itemize}
  \item All connections are present except ones in the same row/column.
  \item \((n^2)\) triples \((a, b)\) \(\not\in E(G_i)\), columns \(a\) and \(b\) cannot be chosen simultaneously
  \item \(\forall (a, b) \in E(G_i)\), columns \(a\) and \(b\) cannot be chosen simultaneously
  \item \(\text{Requested size of the clique: } k^* = k + (n^2) + 1\)
  \item \(i\)-th vertex is chosen from \(I\)
  \item \(\forall (a, b) \not\in E(G_i)\), columns \(a\) and \(b\) cannot be chosen simultaneously
  \item New parameter: \(|X| = kn + 3(n^2)\)
\end{itemize}
Cross-composing into \textsc{Clique}/VC

**Input**: Instances \((G_i, k)\)

**Output**: Instance \((G, X, k^*)\)

- \(\text{all } \neg a \neg b\)
- \((a, b)\)

\(\binom{n}{2}\) triples

requested size of the clique: \(k^* = k + \frac{n^2}{2} + 1\)

- \(i\)-th vertex is chosen from \(I\) \(\Rightarrow\) \(\forall (a, b) \not\in E(G_i), \) columns \(a\) and \(b\) cannot be chosen simultaneously

New parameter: \(|X| = kn + 3\frac{n^2}{2}\)
Cross-composing into Clique/VC

Input: Instances \((G_i, k)\)

Output: Instance \((G, X, k^*)\)

1. All connections are present except ones in the same row/column. \((\frac{n^2}{2})\) triples
2. Requested size of the clique: \(k^* = k + \frac{n^2}{2} + 1\)
3. \(i\)-th vertex is chosen from \(I\) \(\Rightarrow\) \(\forall (a, b) \notin E(G_i), \) columns \(a\) and \(b\) cannot be chosen simultaneously
4. New parameter: \(|X| = kn + 3\) \(\frac{n^2}{2}\)
Cross-composing into **CLIQUE/VC**

**Input:** Instances \((G_i, k)\)

**Output:** Instance \((G, X, k^*)\)

\[(a, b) \in E(G_i)\]

\(1 \quad 2 \quad 3 \quad 4 \quad \ldots \quad n\)

\(1 \quad 2 \quad 3 \quad 4 \quad \ldots \quad n\)

\(n\) \choose 2 \text{ triples}

\(\forall (a, b) /\in E(G_i),\) columns \(a\) and \(b\) cannot be chosen simultaneously

Requested size of the clique:

\[k^* = k + (n^2) + 1\]

\(i\)-th vertex is chosen from \(I\)

\(∀ (a, b) /\in E(G_i)\), new parameter

\[|X| = kn + 3(n^2)\]
Cross-composing into \textbf{Clique}/VC

Input: Instances \((G_i, k)\)

Output: Instance \((G, X, k^*)\)

\((a, b) \notin E(G_i)\)

\((a, b)\) triples

\(\binom{n}{2}\) triples

\(\text{all} \quad \neg a \quad \neg b\)
Cross-composing into **Clique/VC**

**Input**: Instances \((G_i, k)\)

**Output**: Instance \((G, X, k^*)\)

In order to take one vertex from \(I\), the neighborhood of \(i\)-th vertex acts as instance \(G_i\).

**Problem**: Design a 'universal' modulator \(X\).

Requested size of the clique:

\[
k^* = k + \binom{n}{2} + 1
\]

All \(\neg a \neg b\) triples \((a, b)\) \(\not\in E(G_i)\), columns \(a\) and \(b\) cannot be chosen simultaneously.

New parameter:

\[
|X| = kn + 3 \binom{n}{2}
\]

\[\binom{n}{2}\] triples
Cross-composing into \textsc{Clique}/\textsc{VC}

\begin{itemize}
\item \textbf{Input}: Instances \((G_i, k)\)
\item \textbf{Output}: Instance \((G, X, k^*)\)
\end{itemize}

\textit{Problem}: Design a 'universal' modulator \(X\).

\[
\begin{array}{c}
1 \\
2 \\
3 \\
k \\
1 \quad 2 \quad 3 \quad 4 \quad n
\end{array}
\]

\textit{i-th vertex is chosen from } \(I) \Rightarrow \forall (a, b) \notin E(G_i), \text{ columns } a \text{ and } b \text{ cannot be chosen simultaneously}

\textit{New parameter}: \(|X| = kn + 3 \binom{n}{2}\)

\(\binom{n}{2}\) triples
Cross-composing into \textbf{Clique/VC}

\textbf{Input}: Instances \((G_i, k)\)

\textbf{Output}: Instance \((G, X, k^*)\)

1. \(X\) contains vertices
2. \(X\) fulfills the size constraints
   - Force us to take one vertex from \(I\)
3. \(X\) acts as instance \(G_i\)
   - Neighbourhood of \(i\)-th vertex
4. \(X\) is a 'universal' modulator

\textbf{Problem}:
Design a 'universal' modulator \(X\).

\textbf{Constraint}:
All connections between vertices except those in the same row/column.

\(n^2\) triples \((a, b)\) where \(a \neq b\) and \(a, b \in V(G_i)\)

\(k^* = k + \left(\frac{n^2}{2}\right)\)

\(k\)-th vertex is chosen from \(I\) ⇒ \(∀(a, b)\) \(\notin E(G_i)\), columns \(a\) and \(b\) cannot be chosen simultaneously

New parameter:
\(|X| = kn + 3\left(\frac{n^2}{2}\right)\)

\(\frac{n^2}{2}\) triples

\(\binom{n}{2}\) triples
Conclusions on the case study

- Making compositions is highly non-trivial.
Conclusions on the case study

- Making compositions is highly non-trivial.
- Requires good understanding of the problem.
Conclusions on the case study

- Making compositions is highly non-trivial.
- Requires good understanding of the problem.
- Intuitively, what we compose is 'choice'. We need to identify it, and build a technical construction on top of it.
Conclusions on the case study

- Making compositions is highly non-trivial.
- Requires good understanding of the problem.
- Intuitively, what we compose is 'choice'. We need to identify it, and build a technical construction on top of it.
- Many times, requires a lot of gadgeteering...
Conclusions on the case study

- Making compositions is highly non-trivial.
- Requires good understanding of the problem.
- Intuitively, what we compose is ‘choice’. We need to identify it, and build a technical construction on top of it.
- Many times, requires a lot of gadgeteering...
- … tricks that I did not mention …
Making compositions is highly non-trivial.
Requires good understanding of the problem.
Intuitively, what we compose is 'choice'. We need to identify it, and build a technical construction on top of it.
Many times, requires a lot of gadgeteering...
... tricks that I did not mention ...
or clever ideas.
Lower bounds on the exponent

**Goal**: Establish lower bounds on the size of kernels for problems that do admit polynomial kernelization.
Goal: Establish lower bounds on the size of kernels for problems that do admit polynomial kernelization.

The first breakthrough by Dell and van Melkebeek (STOC 2010).
Goal: Establish lower bounds on the size of kernels for problems that do admit polynomial kernelization.

The first breakthrough by Dell and van Melkebeek (STOC 2010).

Idea: a very non-trivial refinement of the approach of Fortnow and Santhanam.
Results of Dell and van Melkebeek

- **Vertex Cover** does not admit compression into bit-size $O(k^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\text{NP} \subseteq \text{coNP/poly}$.}

In particular, the kernel cannot have $O(k^{2-\varepsilon})$ edges. In fact, works for vertex deletion to any subgraph-hereditary class of graphs, which is infinite but not equal to all the graphs. E.g. **Feedback Vertex Set**.

$d$-**Hitting Set** does not admit compression into bit-size $O(k^{d-\varepsilon})$.

Original theorem: $d$-**CNF-SAT** par. by the number of variables $n$ does not admit compression into bit-size $O(n^{d-\varepsilon})$.

Gives an alternative way to show no-poly-kernel: Make a reduction from $d$-**CNF-SAT** that constructs an instance with parameter $f(d) \cdot n^c$ for constant $c$. 

Michał Pilipczuk
No-poly-kernels tutorial
38/31


**Results of Dell and van Melkebeek**

- **Vertex Cover** does not admit compression into bit-size $O(k^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.
  - In particular, the kernel cannot have $O(k^{2-\varepsilon})$ edges.
**Results of Dell and van Melkebeek**

- **Vertex Cover** does not admit compression into bit-size $O(k^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.
  
  - In particular, the kernel cannot have $O(k^{2-\varepsilon})$ edges.
  
  - In fact, works for vertex deletion to any subgraph-hereditary class of graphs, which is infinite but not equal to all the graphs.
**Results of Dell and van Melkebeek**

- **Vertex Cover** does not admit compression into bit-size $O(k^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$.
  - In particular, the kernel cannot have $O(k^{2-\varepsilon})$ edges.
  - In fact, works for vertex deletion to any subgraph-hereditary class of graphs, which is infinite but not equal to all the graphs.
- **E.g. Feedback Vertex Set.**
Vertex Cover does not admit compression into bit-size $O(k^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\textbf{NP} \subseteq \textbf{coNP}/\textbf{poly}$.

- In particular, the kernel cannot have $O(k^{2-\varepsilon})$ edges.
- In fact, works for vertex deletion to any subgraph-hereditary class of graphs, which is infinite but not equal to all the graphs.
- E.g. Feedback Vertex Set.
- $d$-Hitting Set does not admit compression into bit-size $O(k^{d-\varepsilon})$. 
Results of Dell and van Melkebeek

- **Vertex Cover** does not admit compression into bit-size $O(k^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{coNP}/\mathsf{poly}$.
  
  - In particular, the kernel cannot have $O(k^{2-\varepsilon})$ edges.
  - In fact, works for vertex deletion to any subgraph-hereditary class of graphs, which is infinite but not equal to all the graphs.
  - E.g. **Feedback Vertex Set**.
  - **d-Hitting Set** does not admit compression into bit-size $O(k^{d-\varepsilon})$.

- **Original theorem**: **d-CNF-SAT** par. by the number of variables $n$ does not admit compression into bit-size $O(n^{d-\varepsilon})$. 
Results of Dell and van Melkebeek

- **Vertex Cover** does not admit compression into bit-size $O(k^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\text{NP} \subseteq \text{coNP/poly}$.
  - In particular, the kernel cannot have $O(k^{2-\varepsilon})$ edges.
  - In fact, works for vertex deletion to any subgraph-hereditary class of graphs, which is infinite but not equal to all the graphs.
  - E.g. **Feedback Vertex Set**.
  - **d-Hitting Set** does not admit compression into bit-size $O(k^{d-\varepsilon})$.

- **Original theorem**: $d$-CNF-SAT par. by the number of variables $n$ does not admit compression into bit-size $O(n^{d-\varepsilon})$.

- Gives an alternative way to show no-poly-kernel:
**Vertex Cover** does not admit compression into bit-size $O(k^{2-\varepsilon})$ for any $\varepsilon > 0$, unless $\text{NP} \subseteq \text{coNP}/\text{poly}$.  

- In particular, the kernel cannot have $O(k^{2-\varepsilon})$ edges.
- In fact, works for vertex deletion to any subgraph-hereditary class of graphs, which is infinite but not equal to all the graphs.
- E.g. **Feedback Vertex Set**.
- **d-Hitting Set** does not admit compression into bit-size $O(k^{d-\varepsilon})$.

**Original theorem**: **d-CNFSAT** par. by the number of variables $n$ does not admit compression into bit-size $O(n^{d-\varepsilon})$.

Gives an alternative way to show no-poly-kernel:

- Make a reduction from **d-CNFSAT** that constructs an instance with parameter $f(d) \cdot n^c$ for constant $c$. 
Weak compositions, extracted

Weak composition theorem

Assume that an **NP**-hard language $L$ composes into $Q$ in the following sense: the composition, given $t$ instances $x_1, x_2, \ldots, x_t$ of size at most $k$, returns instance $(y, k^*)$ such that

- $k^* = t^{1/d} \cdot \text{poly}(k)$
- $(y, k^*) \in Q$ if and only if $x_i \in L$ for at least one $i$.

Then $Q$ does not admit compression into bit-size $O(k^{d-\varepsilon})$ unless $\textbf{NP} \subseteq \textbf{coNP}/\text{poly}$. 
Weak compositions, extracted

Weak composition theorem

Assume that an \textbf{NP}-hard language \( L \) composes into \( Q \) in the following sense: the composition, given \( t \) instances \( x_1, x_2, \ldots, x_t \) of size at most \( k \), returns instance \((y, k^*)\) such that

\begin{itemize}
  \item \( k^* = t^{1/d} \cdot \text{poly}(k) \)
  \item \((y, k^*) \in Q \) if and only if \( x_i \in L \) for at least one \( i \).
\end{itemize}

Then \( Q \) does not admit compression into bit-size \( O(k^{d-\varepsilon}) \) unless \( \textbf{NP} \subseteq \textbf{coNP}/\text{poly} \).

- We can assume that \( x_1, x_2, \ldots, x_t \) are equivalent w.r.t. some polynomial equivalence relation.
A number of tight or almost tight bounds developed independently by Dell and Marx, and by Hermelin and Wu (SODA 2012).
A number of tight or almost tight bounds developed independently by Dell and Marx, and by Hermelin and Wu (SODA 2012).

By choosing the language $L$ (Multicoloured Biclique) in a smart manner, Dell and Marx reprove the previous results circumventing the technicalities.
Results

- A number of tight or almost tight bounds developed independently by Dell and Marx, and by Hermelin and Wu (SODA 2012).
- By choosing the language $L$ (Multicoloured Biclique) in a smart manner, Dell and Marx reprove the previous results circumventing the technicalities.
- Some tight bounds: $O(k^d)$ kernel (Fellows at el.) vs. $O(k^{d-\varepsilon})$ lower bound (Dell and Marx) for $d$-Set Packing.
A number of tight or almost tight bounds developed independently by Dell and Marx, and by Hermelin and Wu (SODA 2012).

By choosing the language $L$ (MULTICOLOURED BICLIQUE) in a smart manner, Dell and Marx reprove the previous results circumventing the technicalities.

Some tight bounds: $O(k^d)$ kernel (Fellows at el.) vs. $O(k^{d-\varepsilon})$ lower bound (Dell and Marx) for $d$-SET PACKING.

Some gaps: $O(k^d)$ kernel vs. $O(k^{d-1-\varepsilon})$ lower bound (Dell and Marx) for packing $K_d$-s into a graph.
Results

- A number of tight or almost tight bounds developed independently by Dell and Marx, and by Hermelin and Wu (SODA 2012).
- By choosing the language $L$ (Multicoloured Biclique) in a smart manner, Dell and Marx reprove the previous results circumventing the technicalities.
- Some tight bounds: $O(k^d)$ kernel (Fellows et al.) vs. $O(k^{d-\varepsilon})$ lower bound (Dell and Marx) for $d$-Set Packing.
- Some gaps: $O(k^d)$ kernel vs. $O(k^{d-1-\varepsilon})$ lower bound (Dell and Marx) for packing $K_d$-s into a graph.
- Still a lot of work to do.
Already Bodlaender et al. conjectured that everything should work the same if OR was replaced by AND.
The AND-Conjecture

- Already Bodlaender et al. conjectured that everything should work the same if OR was replaced by AND.
- The only missing part was an analogue of the backbone theorem of Fortnow and Santhanam.

The AND-Conjecture has been proven by Andrew Drucker (FOCS 2012) under the same complexity assumption. The whole framework, including cross-composition, works in the same manner.
The AND-Conjecture

- Already Bodlaender et al. conjectured that everything should work the same if OR was replaced by AND.
- The only missing part was an analogue of the backbone theorem of Fortnow and Santhanam.
- That would imply that several structural problems, like \textsc{TreeWidth}, do not admit polynomial kernels.
The AND-Conjecture

- Already Bodlaender et al. conjectured that everything should work the same if OR was replaced by AND.
- The only missing part was an analogue of the backbone theorem of Fortnow and Santhanam.
- That would imply that several structural problems, like Tree-width, do not admit polynomial kernels.
- So-called AND-Conjecture.

The AND-Conjecture has been proven by Andrew Drucker (FOCS 2012) under the same complexity assumption. The whole framework, including cross-composition, works in the same manner.
The AND-Composition Theorem

- Already Bodlaender et al. conjectured that everything should work the same if OR was replaced by AND.
- The only missing part was an analogue of the backbone theorem of Fortnow and Santhanam.
- That would imply that several structural problems, like tree width, do not admit polynomial kernels.
- So-called AND-Conjecture.
- The AND-Conjecture has been proven by Andrew Drucker (FOCS 2012) under the same complexity assumption.
The AND-Composition Theorem

- Already Bodlaender et al. conjectured that everything should work the same if OR was replaced by AND.
- The only missing part was an analogue of the backbone theorem of Fortnow and Santhanam.
- That would imply that several structural problems, like TreeWidth, do not admit polynomial kernels.
- So-called AND-Conjecture.
- The AND-Conjecture has been proven by Andrew Drucker (FOCS 2012) under the same complexity assumption.
- The whole framework, including cross-composition, works in the same manner.
Challenges

- Compression vs. Kernelization
Challenges

- **Compression vs. Kernelization**
  - VC has kernel with $O(k)$ vertices and $O(k^2)$ edges.
  - What about FVS?

---

**Is compositionality the only reason why polynomial kernelization is infeasible?**

**Completeness theory for kernelization.**

**On-going work of Hermelin, Kratsch, Sołtys, Wahlström, Wu.**

**Turing kernelization**

A Turing kernel is a polynomial-time algorithm with an access to an oracle that resolves kernels.

**No clue how to show infeasibility of Turing kernelization.**

**Take your favourite problem and compose!**
Challenges

- Compression vs. Kernelization
  - VC has kernel with $O(k)$ vertices and $O(k^2)$ edges.
  - What about FVS?
  - Is compositionality the only reason why polynomial kernelization is infeasible?
Challenges

Compression vs. Kernelization
- VC has kernel with $O(k)$ vertices and $O(k^2)$ edges.
  What about FVS?
- Is compositionality the only reason why polynomial kernelization is infeasible?

Completeness theory for kernelization.
Challenges

- Compression vs. Kernelization
  - VC has kernel with $O(k)$ vertices and $O(k^2)$ edges.
  - What about FVS?
  - Is compositionality the only reason why polynomial kernelization is infeasible?

- Completeness theory for kernelization.
  - On-going work of Hermelin, Kratsch, Sołtys, Wahlström, Wu.
Challenges

- **Compression vs. Kernelization**
  - VC has kernel with $O(k)$ vertices and $O(k^2)$ edges.
  - What about FVS?
  - Is compositionality the only reason why polynomial kernelization is infeasible?

- **Completeness theory for kernelization.**
  - On-going work of Hermelin, Kratsch, Sołtys, Wahlström, Wu.

- **Turing kernelization**
Challenges

- Compression vs. Kernelization
  - VC has kernel with $O(k)$ vertices and $O(k^2)$ edges.
  - What about FVS?
  - Is compositionality the only reason why polynomial kernelization is infeasible?

- Completeness theory for kernelization.
  - On-going work of Hermelin, Kratsch, Sołtys, Wahlström, Wu.

- Turing kernelization
  - Turing kernel is a polynomial-time algorithm with an access to an oracle that resolves kernels.
Challenges

- **Compression vs. Kernelization**
  - VC has kernel with $O(k)$ vertices and $O(k^2)$ edges.
  - What about FVS?
  - Is compositionality the only reason why polynomial kernelization is infeasible?

- **Completeness theory for kernelization.**
  - On-going work of Hermelin, Kratsch, Sołtys, Wahlström, Wu.

- **Turing kernelization**
  - Turing kernel is a polynomial-time algorithm with an access to an oracle that resolves kernels.
  - No clue how to show infeasibility of Turing kernelization.
Challenges

- Compression vs. Kernelization
  - VC has kernel with $O(k)$ vertices and $O(k^2)$ edges.
  - What about FVS?
  - Is compositionality the only reason why polynomial kernelization is infeasible?

- Completeness theory for kernelization.
  - On-going work of Hermelin, Kratsch, Sołtys, Wahlström, Wu.

- Turing kernelization
  - Turing kernel is a polynomial-time algorithm with an access to an oracle that resolves kernels.
  - No clue how to show infeasibility of Turing kernelization.

- Take your favourite problem and compose!
Questions?

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)