Kernelization lower bounds

Showing no polynomial kernels

How do we show that there is no polynomial kernel for a problem?

No answer until 2008.

Showing no polynomial kernels

How do we show that there is no polynomial kernel for a problem?

No answer until 2008.

Decision problems as languages

- Un-parameterized problems can be viewed as languages for a finite alphabet Σ.
- A language L over Σ is a subset of strings in Σ^*.
- A Parameterized problem is a set of pairs (x, k), where $x \in \Sigma^*$ and k is a non-negative integer.
- In the unparameterized version of a parameterized problem k is appended at the end of input in unary.

Kernelization algorithm for a language L

This is a polynomial time algorithm that takes as input (x, k) and outputs (x', k') such that

$$(x, k) \in L \text{ if and only if } (x', k') \in L, \text{ and}$$
$$|x'| + k' \leq f(k)$$

Decision problems as languages

- Un-parameterized problems can be viewed as languages for a finite alphabet Σ.
- A language L over Σ is a subset of strings in Σ^*.
- A Parameterized problem is a set of pairs (x, k), where $x \in \Sigma^*$ and k is a non-negative integer.
- In the unparameterized version of a parameterized problem k is appended at the end of input in unary.

Kernelization algorithm for a language L

This is a polynomial time algorithm that takes as input (x, k) and outputs (x', k') such that

$$(x, k) \in L \text{ if and only if } (x', k') \in L, \text{ and}$$
$$|x'| + k' \leq f(k)$$

Decision problems as languages

- Un-parameterized problems can be viewed as languages for a finite alphabet Σ.
- A language L over Σ is a subset of strings in Σ^*.
- A Parameterized problem is a set of pairs (x, k), where $x \in \Sigma^*$ and k is a non-negative integer.
- In the unparameterized version of a parameterized problem k is appended at the end of input in unary.

Kernelization algorithm for a language L

This is a polynomial time algorithm that takes as input (x, k) and outputs (x', k') such that

$$(x, k) \in L \text{ if and only if } (x', k') \in L, \text{ and}$$
$$|x'| + k' \leq f(k)$$

Decision problems as languages

- Un-parameterized problems can be viewed as languages for a finite alphabet Σ.
- A language L over Σ is a subset of strings in Σ^*.
- A Parameterized problem is a set of pairs (x, k), where $x \in \Sigma^*$ and k is a non-negative integer.
- In the unparameterized version of a parameterized problem k is appended at the end of input in unary.

Kernelization algorithm for a language L

This is a polynomial time algorithm that takes as input (x, k) and outputs (x', k') such that

$$(x, k) \in L \text{ if and only if } (x', k') \in L, \text{ and}$$
$$|x'| + k' \leq f(k)$$

Polynomial Compression

Let L, R be parameterized problems.

Polynomial compression from L to R

This is a polynomial time algorithm that takes as input (x, k) and outputs (x', k') such that

$$(x, k) \in L \text{ if and only if } (x', k') \in R, \text{ and}$$
$$|x'| + k' \leq f(k)$$

- We say L has no polynomial compression if L has no polynomial compression to any problem R.
- We assume nothing about the problem R.
- If $R \in$ NP and L is NP-hard, then a polynomial compression from L to R implies a polynomial kernel for L.

Polynomial Compression

Let L, R be parameterized problems.

Polynomial compression from L to R

This is a polynomial time algorithm that takes as input (x, k) and outputs (x', k') such that

$$(x, k) \in L \text{ if and only if } (x', k') \in R, \text{ and}$$
$$|x'| + k' \leq f(k)$$

- We say L has no polynomial compression if L has no polynomial compression to any problem R.
- We assume nothing about the problem R.
- If $R \in$ NP and L is NP-hard, then a polynomial compression from L to R implies a polynomial kernel for L.

Polynomial Compression

Let L, R be parameterized problems.

Polynomial compression from L to R

This is a polynomial time algorithm that takes as input (x, k) and outputs (x', k') such that

$$(x, k) \in L \text{ if and only if } (x', k') \in R, \text{ and}$$
$$|x'| + k' \leq f(k)$$

- We say L has no polynomial compression if L has no polynomial compression to any problem R.
- We assume nothing about the problem R.
- If $R \in$ NP and L is NP-hard, then a polynomial compression from L to R implies a polynomial kernel for L.

Polynomial Compression

Let L, R be parameterized problems.

Polynomial compression from L to R

This is a polynomial time algorithm that takes as input (x, k) and outputs (x', k') such that

$$(x, k) \in L \text{ if and only if } (x', k') \in R, \text{ and}$$
$$|x'| + k' \leq f(k)$$

- We say L has no polynomial compression if L has no polynomial compression to any problem R.
- We assume nothing about the problem R.
- If $R \in$ NP and L is NP-hard, then a polynomial compression from L to R implies a polynomial kernel for L.

Intuition (with example)

k-Path

Input: A graph G and a positive integer k
Question: Does there exist a path of length k in G?

Suppose (for the sake of contradiction) k-Path admits a polynomial kernel, say with k^3 vertices.

- Let $(G_1, k), (G_2, k), \ldots, (G_t, k)$ be $t = k^7$ instances of k-Path.
- Create another instance (H, k) of k-Path where H is the disjoint union of the graphs $G_1, \ldots, G_t$.
- H has a path of length k if and only if G_i has a path of length k for some $i \in \{1, \ldots, t\}$ ("H is an OR of G_is").
- Kernelize (H, k) and get (H', k') where $|V(H')| \leq k^3$. Bit size of the reduced instance (H', k') is $\approx k^6$.

The bit size of the reduced instance is strictly less than the number of instances (t) that we started with!

Intuition (with example)

k-Path

Input: A graph G and a positive integer k

Question: Does there exist a path of length k in G?

Suppose (for the sake of contradiction) k-Path admits a polynomial kernel, say with k^3 vertices.

- Let $(G_1, k), (G_2, k), \ldots, (G_t, k)$ be $t = k^7$ instances of k-Path.
- Create another instance (H, k) of k-Path where H is the disjoint union of the graphs $G_1, \ldots, G_t$.
- H has a path of length k if and only if G_i has a path of length k for some $i \in \{1, \ldots, t\}$ ("H is an OR of G_is").
- Kernelize (H, k) and get (H', k') where $|V(H')| \leq k^3$. Bit size of the reduced instance (H', k') is $\approx k^6$.

The bit size of the reduced instance is strictly less than the number of instances (t) that we started with!

Intuition (with example)

k-Path

Input: A graph G and a positive integer k

Question: Does there exist a path of length k in G?

Suppose (for the sake of contradiction) k-Path admits a polynomial kernel, say with k^3 vertices.

- Let $(G_1, k), (G_2, k), \ldots, (G_t, k)$ be $t = k^7$ instances of k-Path.
- Create another instance (H, k) of k-Path where H is the disjoint union of the graphs $G_1, \ldots, G_t$.
- H has a path of length k if and only if G_i has a path of length k for some $i \in \{1, \ldots, t\}$ ("H is an OR of G_is").
- Kernelize (H, k) and get (H', k') where $|V(H')| \leq k^3$. Bit size of the reduced instance (H', k') is $\approx k^6$.

The bit size of the reduced instance is strictly less than the number of instances (t) that we started with!

Intuition (with example)

k-Path

Input: A graph G and a positive integer k

Question: Does there exist a path of length k in G?

Suppose (for the sake of contradiction) k-Path admits a polynomial kernel, say with k^3 vertices.

- Let $(G_1, k), (G_2, k), \ldots, (G_t, k)$ be $t = k^7$ instances of k-Path.
- Create another instance (H, k) of k-Path where H is the disjoint union of the graphs $G_1, \ldots, G_t$.
- H has a path of length k if and only if G_i has a path of length k for some $i \in \{1, \ldots, t\}$ ("H is an OR of G_is").
- Kernelize (H, k) and get (H', k') where $|V(H')| \leq k^3$. Bit size of the reduced instance (H', k') is $\approx k^6$.

The bit size of the reduced instance is strictly less than the number of instances (t) that we started with!

Intuition (with example)

k-Path

Input: A graph G and a positive integer k

Question: Does there exist a path of length k in G?

Suppose (for the sake of contradiction) k-Path admits a polynomial kernel, say with k^3 vertices.

- Let $(G_1, k), (G_2, k), \ldots, (G_t, k)$ be $t = k^7$ instances of k-Path.
- Create another instance (H, k) of k-Path where H is the disjoint union of the graphs $G_1, \ldots, G_t$.
- H has a path of length k if and only if G_i has a path of length k for some $i \in \{1, \ldots, t\}$ ("H is an OR of G_is").
- Kernelize (H, k) and get (H', k') where $|V(H')| \leq k^3$. Bit size of the reduced instance (H', k') is $\approx k^6$.

The bit size of the reduced instance is strictly less than the number of instances (t) that we started with!

Intuition (with example)

k-Path

Input: A graph G and a positive integer k
Question: Does there exist a path of length k in G?

Suppose (for the sake of contradiction) k-Path admits a polynomial kernel, say with k^3 vertices.

- Let $(G_1, k), (G_2, k), \ldots, (G_t, k)$ be $t = k^7$ instances of k-Path.
- Create another instance (H, k) of k-Path where H is the disjoint union of the graphs $G_1, \ldots, G_t$.
- H has a path of length k if and only if G_i has a path of length k for some $i \in \{1, \ldots, t\}$ (" H is an OR of G_is").
- Kernelize (H, k) and get (H', k') where $|V(H')| \leq k^3$. Bit size of the reduced instance (H', k') is $\approx k^6$.

The bit size of the reduced instance is strictly less than the number of instances (t) that we started with!

No accountability for the loss of information!

Loss of important information

In polynomial time, we **completely forgot** about some of the (t) input instances.
How did this polynomial time algorithm knew that they could be discarded (that they did not have k-paths)? This is unlikely to have happened for any NP-hard problem!

OR-distillation

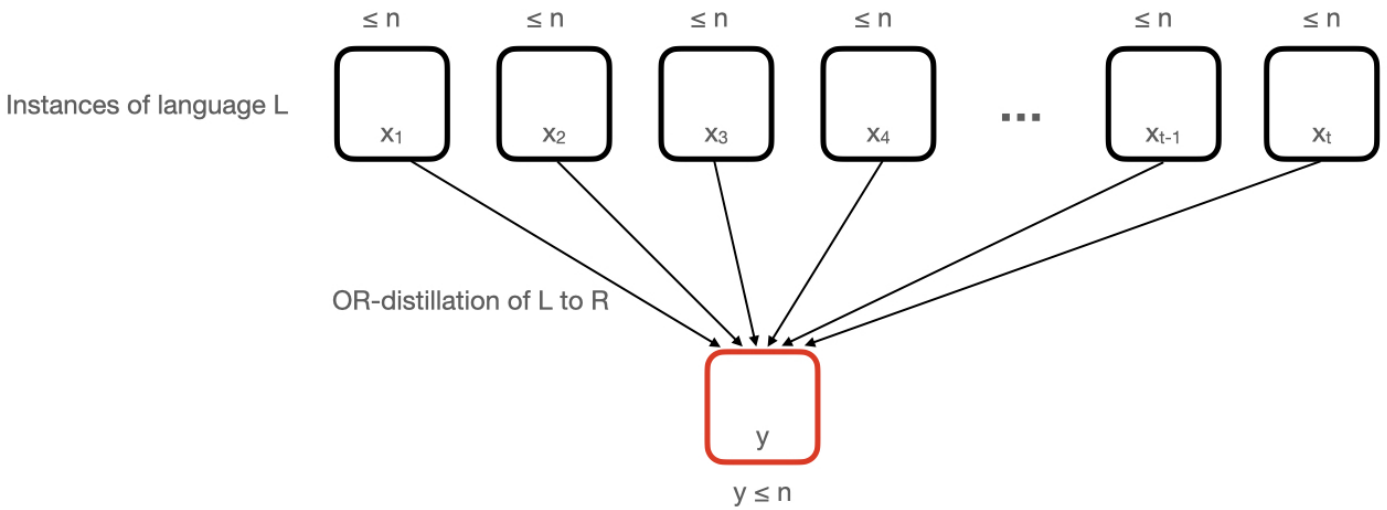
Let L, R be un-parameterized languages.

OR-distillation of L into R

Input: Strings $x_1, \ldots, x_t$ each of length at most n.
Output: A string y such that:

- $y \in R$ if and only if $x_i \in L$ for some $i \in \{1, \ldots, t\}$, and
- $|y| = poly(\max |x_i|)$.

Time: polynomial in the input, that is $poly(\sum_{i=1}^{t} |x_i|)$.



OR-distillation

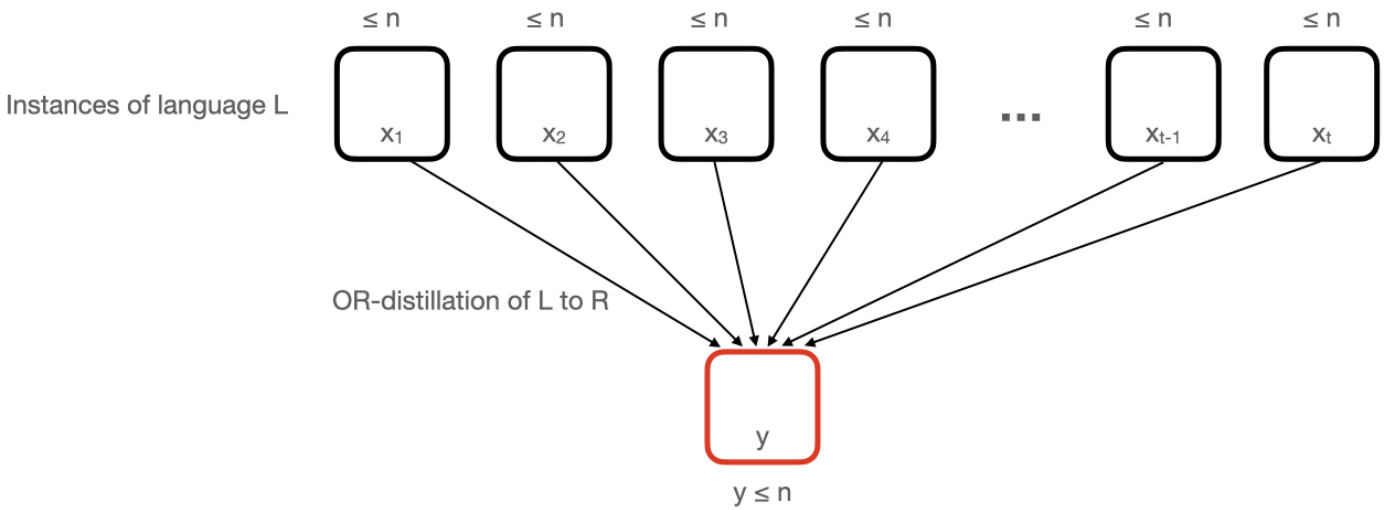
Let L, R be un-parameterized languages.

OR-distillation of L into R

Input: Strings $x_1, \ldots, x_t$ each of length at most n.
Output: A string y such that:

- $y \in R$ if and only if $x_i \in L$ for some $i \in \{1, \ldots, t\}$, and
- $|y| = poly(\max |x_i|)$.

Time: polynomial in the input, that is $poly(\sum_{i=1}^{t} |x_i|)$.



OR-distillation

- Define another language, OR-$L = \{x_1\# \ldots \#x_t : x_i \in L \text{ for some } i\}$.
- OR-distillation from L into R is a polynomial compression from OR-$L/\max|x_i|$ to R.

The main theorem (without proof)

OR-distillation theorem [Fortnow, Santhanam 2008]

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

- Intuitive meaning of NP$\subseteq$coNP/poly: Verifying proofs in polynomial time cannot be turned into verifying counterexamples in polynomial time, even if we allow polynomial advice.
- Implication of NP$\subseteq$coNP/poly: PH$=\Sigma_3^P$, that is the polynomial hierarchy collapses to the third level.
- This is believed to be highly unlikely (not as unlikely as P=NP but still highly unlikely).

The main theorem (without proof)

OR-distillation theorem [Fortnow, Santhanam 2008]

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

- Intuitive meaning of NP$\subseteq$coNP/poly: Verifying proofs in polynomial time cannot be turned into verifying counterexamples in polynomial time, even if we allow polynomial advice.
- Implication of NP$\subseteq$coNP/poly: PH$=\Sigma_3^P$, that is the polynomial hierarchy collapses to the third level.
- This is believed to be highly unlikely (not as unlikely as P=NP but still highly unlikely).

The main theorem (without proof)

OR-distillation theorem [Fortnow, Santhanam 2008]

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

- Intuitive meaning of NP$\subseteq$coNP/poly: Verifying proofs in polynomial time cannot be turned into verifying counterexamples in polynomial time, even if we allow polynomial advice.
- Implication of NP$\subseteq$coNP/poly: PH$=\Sigma_3^P$, that is the polynomial hierarchy collapses to the third level.
- This is believed to be highly unlikely (not as unlikely as P$=$NP but still highly unlikely).

OR-composition

Let L be a parameterized problem.

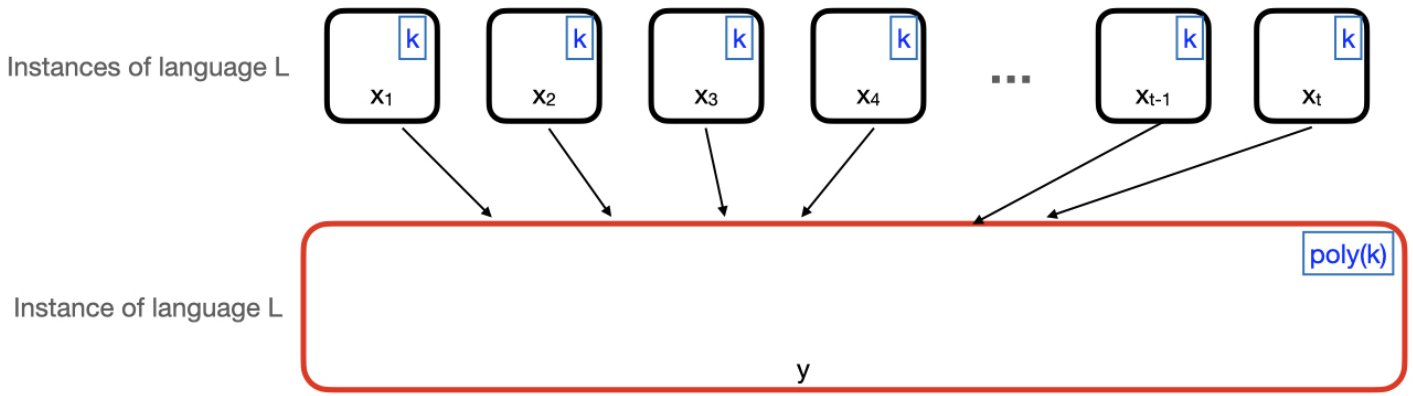
OR-composition for L

Input: $(x_1, k), \ldots, (x_t, k)$ such that $x_i \in \Sigma^*$ and k is a non-negative integer.

Output: (y, k^*) such that

- $(y, k^*) \in L$ if and only if $(x_i, k) \in L$ for some i, and
- $k^* = poly(k)$.

Time: polynomial in the input, that is $poly(\sum_{i=1}^{t} |x_i| + k)$.



OR-composition theorem

OR-composition theorem

If there exists an OR-composition for a parameterized problem L such that the un-parameterized version of L is NP-hard, then L does not admit a polynomial kernel unless $\mathrm{NP} \subseteq \mathrm{coNP/poly}$.

Proof:

OR-composition theorem: Proof

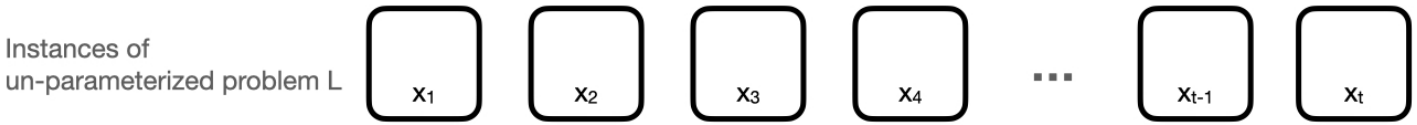


Figure: OR-distillation of L into OR-L

OR-composition theorem: Proof

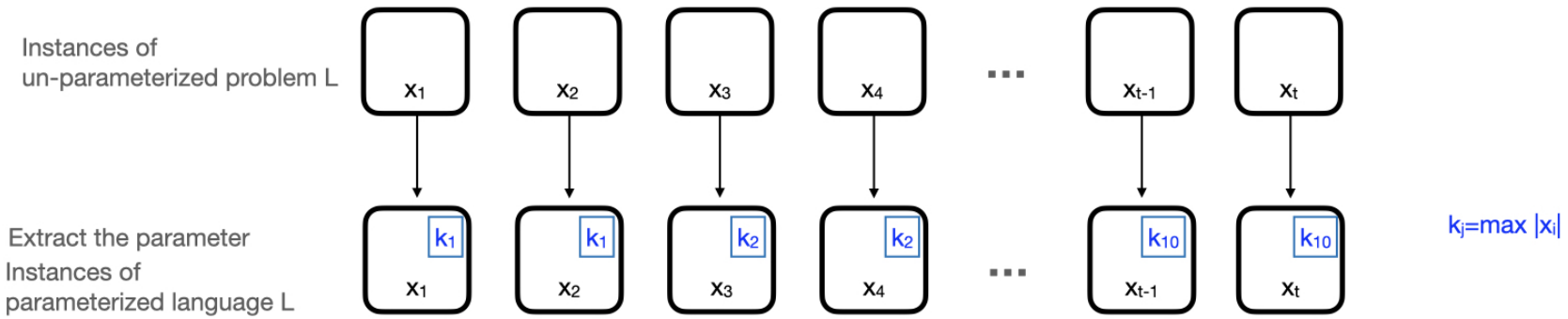


Figure: OR-distillation of L into OR-L

OR-composition theorem: Proof

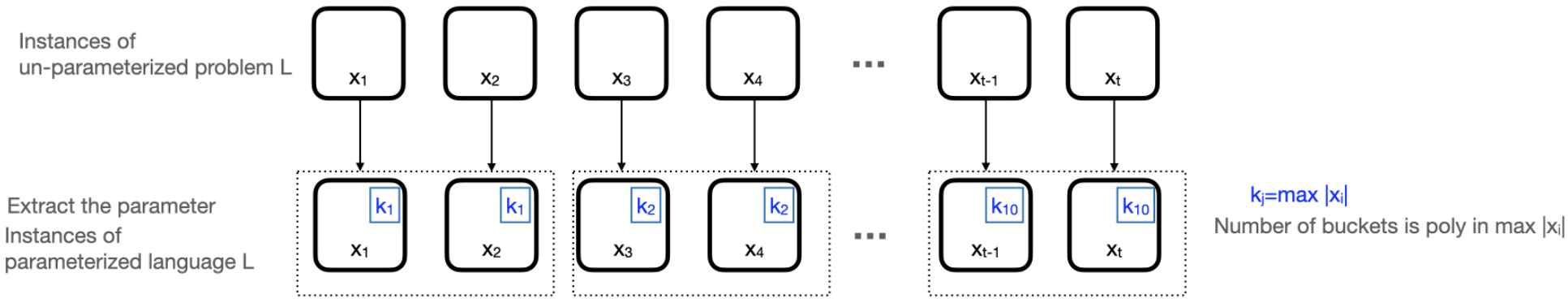


Figure: OR-distillation of L into OR-L

OR-composition theorem: Proof

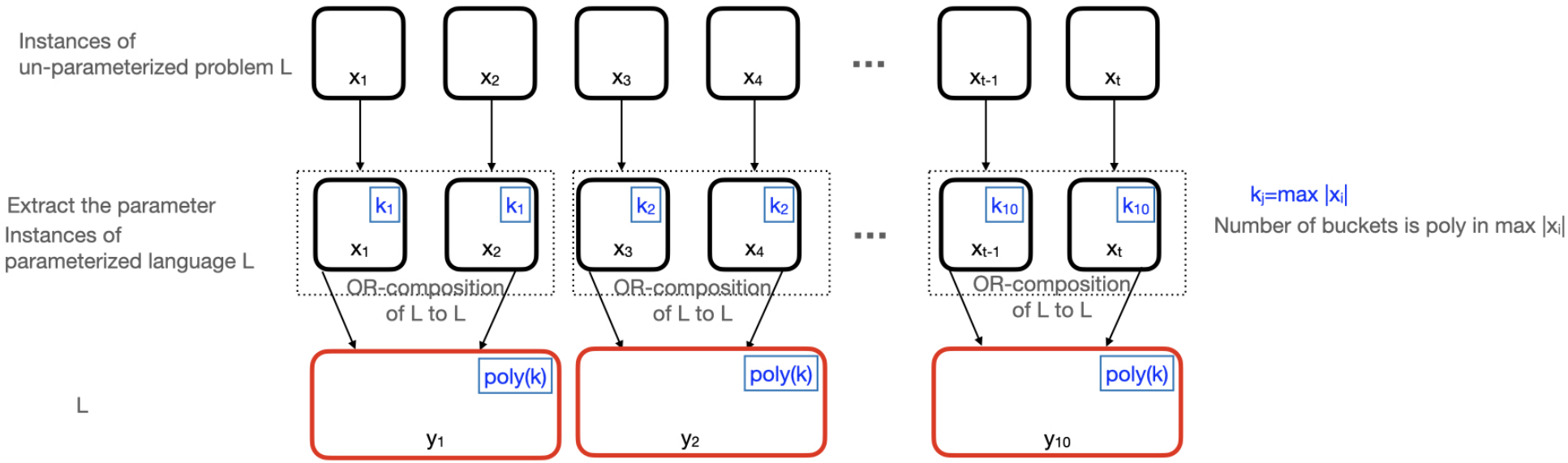


Figure: OR-distillation of L into OR-L

OR-composition theorem: Proof

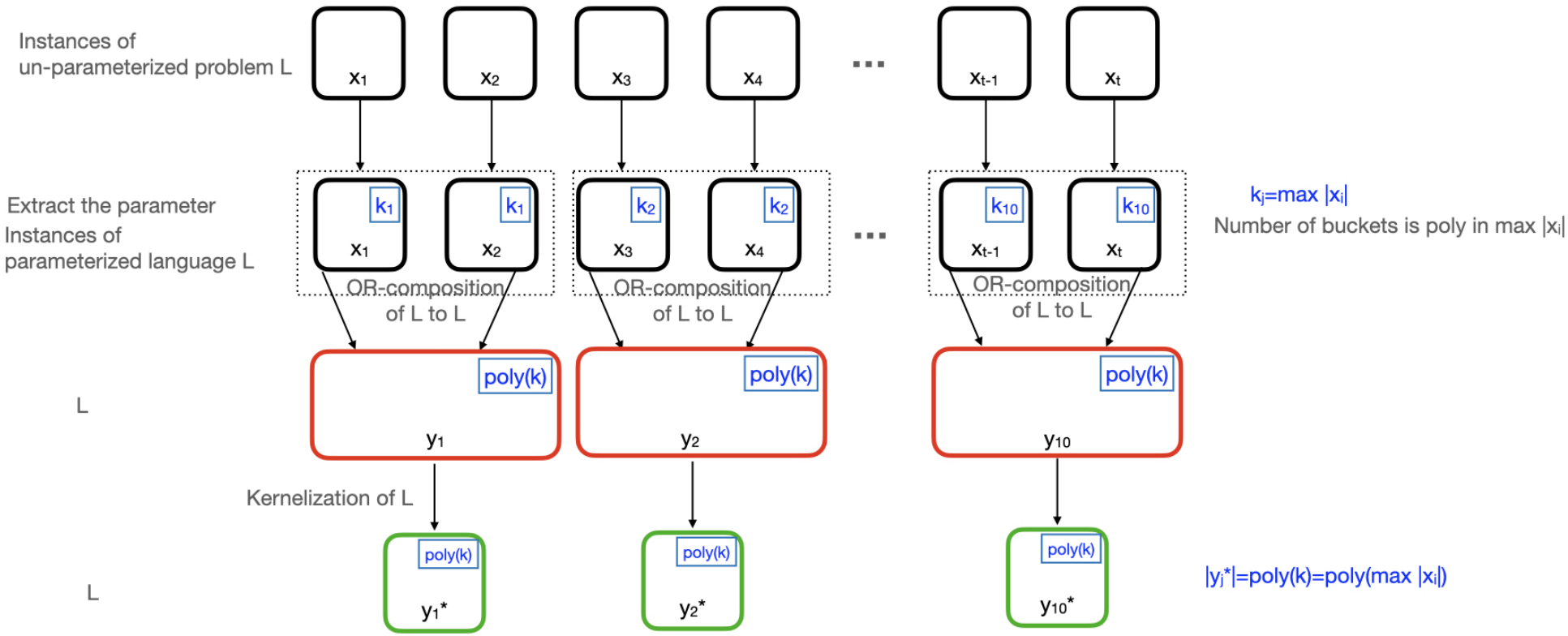


Figure: OR-distillation of L into OR-L

OR-composition theorem: Proof

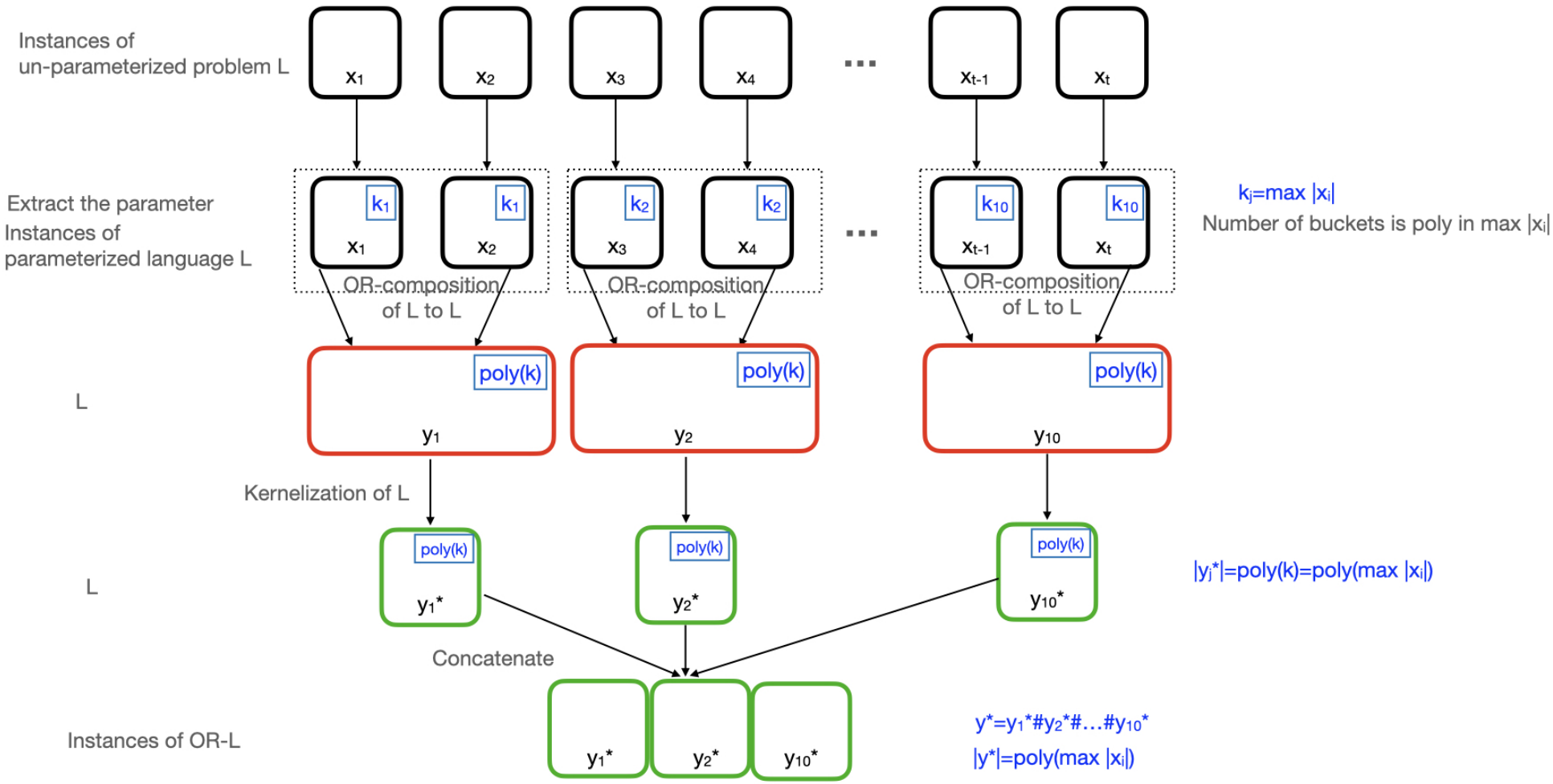


Figure: OR-distillation of L into OR-L

Application of the OR-composition theorem

Theorem

k-Path does not admit a polynomial kernel, unless NP$\subseteq$coNP/poly.

Since k-Path is an NP-hard problem, for the proof of the theorem it is enough to give an OR-composition for k-Path.

OR-composition for k-Path: Given $(G_1, k), \ldots, (G_t, k)$, output $(H = (G_1 \uplus \ldots \uplus G_t), k)$.

Correctness follows because a graph has a path of length k if and only if at least one of its connected components has it.

Show that k-Cycle does not admit a polynomial kernel, unless NP$\subseteq$coNP/poly.

Application of the OR-composition theorem

Theorem

k-Path does not admit a polynomial kernel, unless NP$\subseteq$coNP/poly.

Since k-Path is an NP-hard problem, for the proof of the theorem it is enough to give an OR-composition for k-Path.

OR-composition for k-Path: Given $(G_1, k), \ldots, (G_t, k)$, output $(H = (G_1 \uplus \ldots \uplus G_t), k)$.

Correctness follows because a graph has a path of length k if and only if at least one of its connected components has it.

Show that k-Cycle does not admit a polynomial kernel, unless NP$\subseteq$coNP/poly.

Application of the OR-composition theorem

Theorem

k-Path does not admit a polynomial kernel, unless NP$\subseteq$coNP/poly.

Since k-Path is an NP-hard problem, for the proof of the theorem it is enough to give an OR-composition for k-Path.
OR-composition for k-Path: Given $(G_1, k), \ldots, (G_t, k)$, output $(H = (G_1 \uplus \ldots \uplus G_t), k)$.
Correctness follows because a graph has a path of length k if and only if at least one of its connected components has it.

Show that k-Cycle does not admit a polynomial kernel, unless NP$\subseteq$coNP/poly.

Application of the OR-composition theorem

Theorem

k-Path does not admit a polynomial kernel, unless NP$\subseteq$coNP/poly.

Since k-Path is an NP-hard problem, for the proof of the theorem it is enough to give an OR-composition for k-Path.
OR-composition for k-Path: Given $(G_1, k), \ldots, (G_t, k)$, output $(H = (G_1 \uplus \ldots \uplus G_t), k)$.
Correctness follows because a graph has a path of length k if and only if at least one of its connected components has it.

Show that k-Cycle does not admit a polynomial kernel, unless NP$\subseteq$coNP/poly.

AND-composition

Let L be a parameterized problem.

AND-composition for L

Input: $(x_1, k), \ldots, (x_t, k)$ such that $x_i \in \Sigma^*$ and k is a non-negative integer.

Output: (y, k^*) such that

- $(y, k^*) \in L$ if and only if $(x_i, k) \in L$ for all i, and
- $k^* = poly(k)$.

Time: polynomial in the input, that is $poly(\sum_{i=1}^{t} |x_i| + k)$.

AND-composition theorem

If there exists an AND-composition for a parameterized problem L such that the un-parameterized version of L is NP-hard, then L does not admit a polynomial kernel unless NP$\subseteq$coNP/poly.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

Enhancing the framework for compositionality

We have an OR-composition for an NP-hard problem L (from L to L).

1. Can we show that L has no polynomial compression to any language R?
   - Yes.
2. What happens if we give an OR-composition from Q to L (instead of L to L)? Can we still say no polynomial compression for L?
   - Yes, as long as Q is an NP-hard problem.
3. Can we do more refined bucketting? Eg. Can we ensure that there are same number of vertices in all the graphs of the same bucket?
   - Yes, as long as number of buckets is polynomial.
4. Can we say $k^* = poly(k + \max |x_i|)$?
   - Yes.
5. How large can t be?
   - If $t > \Sigma^{n+1}$, where $n = \max |x_i|$, then we can remove the duplicate input instances.
   - Therefore, $\log t = \mathcal{O}(n)$.
   - Therefore, we can allow $k^* = poly(k + \max |x_i| + \log t)$.

OR-cross composition [Bodlaender, Jansen, Kratsch 2011]

Polynomial equivalence relation

A polynomial equivalence relation is an equivalence relation $\equiv$ defined over strings in Σ^* such that:

- for any two strings $x, y \in \Sigma^*$, one can check whether $x \equiv y$ in $poly(|x| + |y|)$ time, and
- the number of equivalence classes of words of length at most n is $poly(n)$.

Some common examples of polynomial equivalence relations include:

- partitioning the words/graphs such that each equivalence class has words/graphs of same length/number of vertices, or
- ... with same number of vertices, and edges, and size of maximum matching, and budget.

OR-cross composition [Bodlaender, Jansen, Kratsch 2011]

Polynomial equivalence relation

A polynomial equivalence relation is an equivalence relation $\equiv$ defined over strings in Σ^* such that:

- for any two strings $x, y \in \Sigma^*$, one can check whether $x \equiv y$ in $poly(|x| + |y|)$ time, and
- the number of equivalence classes of words of length at most n is $poly(n)$.

Some common examples of polynomial equivalence relations include:

- partitioning the words/graphs such that each equivalence class has words/graphs of same length/number of vertices, or
- ... with same number of vertices, and edges, and size of maximum matching, and budget.

OR-cross composition [Bodlaender, Jansen, Kratsch 2011]

Let Q be an unparameterized problem and L be a parameterized problem

OR-cross composition from Q to L

Input: $x_1, \ldots, x_t$, such that there exists a polynomial equivalence relation $\equiv$ where $x_1, \ldots, x_t$ belong to the same equivalence class of $\equiv$.

Output: (y, k^*) such that:

- $k^* = poly(\log t + \max_{i=1}^{t} |x_i|)$, and
- $(y, k^*) \in L$ if and only if $x_i \in Q$ for some i.

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

OR-cross composition theorem

If there is a OR-cross composition from Q to L and Q is NP-hard, then L has no polynomial compression, unless NP$\subseteq$coNP/poly.

Remark: This notion is particularly useful in refuting polynomial kernels with structural parameterizations, for example, Clique/vertex cover.

OR-cross composition [Bodlaender, Jansen, Kratsch 2011]

Let Q be an unparameterized problem and L be a parameterized problem

OR-cross composition from Q to L

Input: $x_1, \ldots, x_t$, such that there exists a polynomial equivalence relation $\equiv$ where $x_1, \ldots, x_t$ belong to the same equivalence class of $\equiv$.

Output: (y, k^*) such that:

- $k^* = poly(\log t + \max_{i=1}^{t} |x_i|)$, and
- $(y, k^*) \in L$ if and only if $x_i \in Q$ for some i.

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

OR-cross composition theorem

If there is a OR-cross composition from Q to L and Q is NP-hard, then L has no polynomial compression, unless NP$\subseteq$coNP/poly.

Remark: This notion is particularly useful in refuting polynomial kernels with structural parameterizations, for example, Clique/vertex cover.

OR-cross composition [Bodlaender, Jansen, Kratsch 2011]

Let Q be an unparameterized problem and L be a parameterized problem

OR-cross composition from Q to L

Input: $x_1, \ldots, x_t$, such that there exists a polynomial equivalence relation $\equiv$ where $x_1, \ldots, x_t$ belong to the same equivalence class of $\equiv$.

Output: (y, k^*) such that:

- $k^* = poly(\log t + \max_{i=1}^{t} |x_i|)$, and
- $(y, k^*) \in L$ if and only if $x_i \in Q$ for some i.

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

OR-cross composition theorem

If there is a OR-cross composition from Q to L and Q is NP-hard, then L has no polynomial compression, unless NP$\subseteq$coNP/poly.

Remark: This notion is particularly useful in refuting polynomial kernels with structural parameterizations, for example, Clique/vertex cover.

k-Path Packing

k-Path Packing

Input: A graph G and an integer k
Question: Does there exist a collection of k mutually vertex disjoint paths of length k each?
Parameter: k

Does k-Path Packing admit a polynomial kernel/compression?

- Give a reduction from k-Path (G, k) to k-Path Packing (H, k) as follows: H is the disjoint union of G and $k-1$ vertex disjoint paths of length k each.
- G has a path of length k if and only if H has a k-path packing.
- If k-Path Packing admits a polynomial kernel/compression then so does k-path.

k-Path Packing

k-Path Packing

Input: A graph G and an integer k
Question: Does there exist a collection of k mutually vertex disjoint paths of length k each?
Parameter: k

Does k-Path Packing admit a polynomial kernel/compression?

- Give a reduction from k-Path (G, k) to k-Path Packing (H, k) as follows: H is the disjoint union of G and $k-1$ vertex disjoint paths of length k each.
- G has a path of length k if and only if H has a k-path packing.
- If k-Path Packing admits a polynomial kernel/compression then so does k-path.

k-Path Packing

k-Path Packing

Input: A graph G and an integer k
Question: Does there exist a collection of k mutually vertex disjoint paths of length k each?
Parameter: k

Does k-Path Packing admit a polynomial kernel/compression?

- Give a reduction from k-Path (G, k) to k-Path Packing (H, k) as follows: H is the disjoint union of G and $k-1$ vertex disjoint paths of length k each.
- G has a path of length k if and only if H has a k-path packing.
- If k-Path Packing admits a polynomial kernel/compression then so does k-path.

k-Path Packing

k-Path Packing

Input: A graph G and an integer k
Question: Does there exist a collection of k mutually vertex disjoint paths of length k each?
Parameter: k

Does k-Path Packing admit a polynomial kernel/compression?

- Give a reduction from k-Path (G, k) to k-Path Packing (H, k) as follows: H is the disjoint union of G and $k-1$ vertex disjoint paths of length k each.
- G has a path of length k if and only if H has a k-path packing.
- If k-Path Packing admits a polynomial kernel/compression then so does k-path.

Polynomial parameter transformations (PPT)

Let L, R be parameterized problems.

Polynomial parameter transforamtions from L to R

We say there is a polynomial parameter transformation from L to R if there exists a polynomial time algorithm that takes as input (x, k) and outputs (y, k') such that:

- $(x, k) \in L$ if and only if $(y, k') \in R$, and
- $k' = poly(k)$.

PPT Theorem

If there exists a PPT from L to R and L has no polynomial compresssion, then R has no polynomial compression.

Polynomial parameter transformations (PPT)

Let L, R be parameterized problems.

Polynomial parameter transforamtions from L to R

We say there is a polynomial parameter transformation from L to R if there exists a polynomial time algorithm that takes as input (x, k) and outputs (y, k') such that:

- $(x, k) \in L$ if and only if $(y, k') \in R$, and
- $k' = poly(k)$.

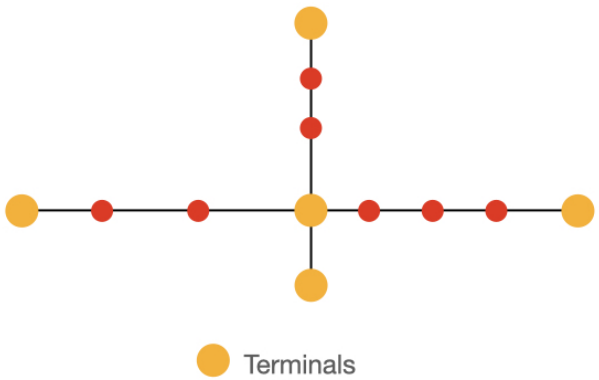
PPT Theorem

If there exists a PPT from L to R and L has no polynomial compresssion, then R has no polynomial compression.

Application: Steiner Tree

Steiner Tree

Input: A graph G, a set of terminals $K \subseteq V(G)$, a positive integer k.
Question: Does there exist a connected subgraph (tree) of G on size at most k vertices that contains all the terminal vertices?
Parameter: k

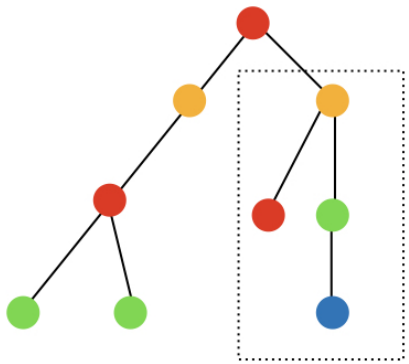


Colorful Graph Motif

Colorful Graph Motif (CGM)

Input: A graph G, a coloring function $c : V(G) \to \{1, \ldots, k\}$.
Question: Does there exist a connected subgraph of G that contains exactly one vertex of each color?
Parameter: k

CGM is NP-hard even on trees.



Colorful Graph Motif

Colorful Graph Motif (CGM)

Input: A graph G, a coloring function $c : V(G) \to \{1, \ldots, k\}$.
Question: Does there exist a connected subgraph of G that contains exactly one vertex of each color?
Parameter: k

OR-composition for Colorful Graph Motif: Given $(G_1, c_1, k), \ldots, (G_t, c_t, k)$, output $(H = G_1 \uplus \ldots \uplus G_t, c = c_1 \cup \ldots \cup c_t, k)$.

Theorem

Colorful Graph Motif does not admit a polynomial kernel/compression unless NP$\subseteq$coNP/poly.

Colorful Graph Motif

Colorful Graph Motif (CGM)

Input: A graph G, a coloring function $c : V(G) \to \{1, \ldots, k\}$.
Question: Does there exist a connected subgraph of G that contains exactly one vertex of each color?
Parameter: k

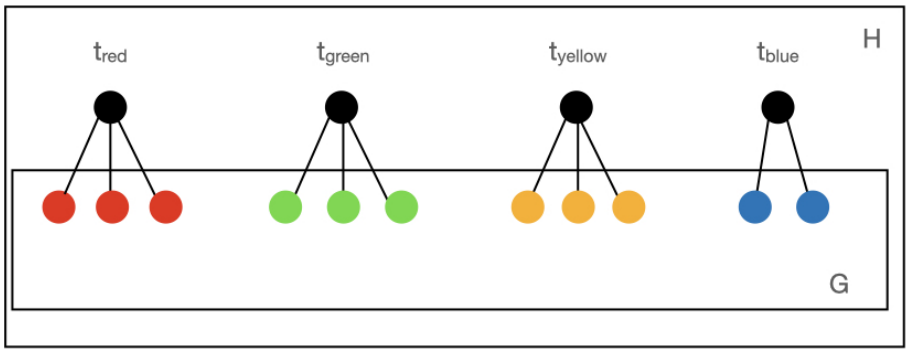
OR-composition for Colorful Graph Motif: Given $(G_1, c_1, k), \ldots, (G_t, c_t, k)$, output $(H = G_1 \uplus \ldots \uplus G_t, c = c_1 \cup \ldots \cup c_t, k)$.

Theorem

Colorful Graph Motif does not admit a polynomial kernel/compression unless NP$\subseteq$coNP/poly.

PPT from CGM to Steiner Tree

Let (G, c, k) be an instance of Colorful Graph Motif.
Construct H from G by adding a new terminal vertex for each color class and making it adjacent to the respective color class.



PPT from CGM to Steiner Tree

Let (G, c, k) be an instance of Colorful Graph Motif.
Construct H from G by adding a new terminal vertex for each color class and making it adjacent to the respective color class.

Claim

(G, c, k) is a Yes-instance of Colorful Graph Motif if and only if $(H, K = \{t_{red}, t_{green}, t_{yellow}, t_{blue}, ..\}, 2k)$ is a Yes-instance of Steiner Tree.

Theorem

Steiner Tree does not admit a polynomial kernel/compression parameterized by k, unless NP$\subseteq$coNP/poly.

PPT from CGM to Steiner Tree

Let (G, c, k) be an instance of Colorful Graph Motif.
Construct H from G by adding a new terminal vertex for each color class and making it adjacent to the respective color class.

Claim

(G, c, k) is a Yes-instance of Colorful Graph Motif if and only if $(H, K = \{t_{red}, t_{green}, t_{yellow}, t_{blue}, ..\}, 2k)$ is a Yes-instance of Steiner Tree.

Theorem

Steiner Tree does not admit a polynomial kernel/compression parameterized by k, unless NP$\subseteq$coNP/poly.

Weak compositions and stronger lower bounds

One can further refine the notion of OR-cross compositions to define weak compositions which give tight polynomial lower bounds for kernel sizes.

For example, they can be used to show the following:

- Vertex Cover does not admit a polynomial compression with bit size $\mathcal{O}(k^{2-\epsilon})$, unless NP$\subseteq$coNP/poly.
- Feedback Vertex Set does not admit a polynomial compression with bit size $\mathcal{O}(k^{2-\epsilon})$, unless NP$\subseteq$coNP/poly.
- d-Hitting Set and d-Set Packing parameterized by $|U|$ does not admit a polynomial compression with bit size $\mathcal{O}(|U|^{d-\epsilon})$, unless NP$\subseteq$coNP/poly.
- ...

Weak compositions and stronger lower bounds

One can further refine the notion of OR-cross compositions to define weak compositions which give tight polynomial lower bounds for kernel sizes. For example, they can be used to show the following:

- Vertex Cover does not admit a polynomial compression with bit size $\mathcal{O}(k^{2-\epsilon})$, unless NP$\subseteq$coNP/poly.
- Feedback Vertex Set does not admit a polynomial compression with bit size $\mathcal{O}(k^{2-\epsilon})$, unless NP$\subseteq$coNP/poly.
- d-Hitting Set and d-Set Packing parameterized by $|U|$ does not admit a polynomial compression with bit size $\mathcal{O}(|U|^{d-\epsilon})$, unless NP$\subseteq$coNP/poly.
- ...