Parameterized Algorithms

Lower Bounds for Kernelization

Thomas Bläsius

Kernelization – formal definition

Definition
For a parameterized problem L, a **kernelization** is a polynomial-time algorithm that transforms any instance (x, k) into an equivalent instance (x', k') (the **kernel**) such that $|x'| + k' \leq g(k)$.

Kernelization – formal definition

Definition
For a parameterized problem L, a **kernelization** is a polynomial-time algorithm that transforms any instance (x, k) into an equivalent instance (x', k') (the **kernel**) such that $|x'| + k' \leq g(k)$.

$(x, k) \in L \Leftrightarrow (x', k') \in L$

independent of $|x|$

Kernelization – formal definition

poly-time required; FPT-time not allowed

Definition
For a parameterized problem L, a **kernelization** is a polynomial-time algorithm that transforms any instance (x, k) into an equivalent instance (x', k') (the **kernel**) such that $|x'| + k' \leq g(k)$.

$(x, k) \in L \Leftrightarrow (x', k') \in L$

independent of $|x|$

Kernelization – formal definition

poly-time required; FPT-time not allowed

Definition
For a parameterized problem L, a **kernelization** is a polynomial-time algorithm that transforms any instance (x, k) into an equivalent instance (x', k') (the **kernel**) such that $|x'| + k' \leq g(k)$.

$(x, k) \in L \Leftrightarrow (x', k') \in L$

independent of $|x|$

Theorem
A decidable problem is in FPT if and only if it has a kernelization.

Which of the two directions is easy to prove?

Kernelization – formal definition

poly-time required; FPT-time not allowed

Definition
For a parameterized problem L, a **kernelization** is a polynomial-time algorithm that transforms any instance (x, k) into an equivalent instance (x', k') (the **kernel**) such that $|x'| + k' \leq g(k)$.

$(x, k) \in L \Leftrightarrow (x', k') \in L$

independent of $|x|$

Theorem
A decidable problem is in FPT if and only if it has a kernelization.

Which of the two directions is easy to prove?

Proof (of the difficult direction)

- let $\mathcal{A}$ be an algorithm that solves (x, k) in $f(k)|x|^c$ time

Kernelization – formal definition

poly-time required; FPT-time not allowed

Definition
For a parameterized problem L, a **kernelization** is a polynomial-time algorithm that transforms any instance (x, k) into an equivalent instance (x', k') (the **kernel**) such that $|x'| + k' \leq g(k)$.

$(x, k) \in L \Leftrightarrow (x', k') \in L$

independent of $|x|$

Theorem
A decidable problem is in FPT if and only if it has a kernelization.

Which of the two directions is easy to prove?

Proof (of the difficult direction)

- let $\mathcal{A}$ be an algorithm that solves (x, k) in $f(k)|x|^c$ time
- **case 1:** $f(k) \leq |x|$
- **case 2:** $f(k) > |x|$

Kernelization – formal definition

poly-time required; FPT-time not allowed

Definition
For a parameterized problem L, a **kernelization** is a polynomial-time algorithm that transforms any instance (x, k) into an equivalent instance (x', k') (the **kernel**) such that $|x'| + k' \leq g(k)$.

$(x, k) \in L \Leftrightarrow (x', k') \in L$

independent of $|x|$

Theorem
A decidable problem is in FPT if and only if it has a kernelization.

Which of the two directions is easy to prove?

Proof (of the difficult direction)

- let $\mathcal{A}$ be an algorithm that solves (x, k) in $f(k)|x|^c$ time
- **case 1:** $f(k) \leq |x|$
  - solve (x, k) with $\mathcal{A}$ in polynomial time
- **case 2:** $f(k) > |x|$

Kernelization – formal definition

poly-time required; FPT-time not allowed

Definition
For a parameterized problem L, a **kernelization** is a polynomial-time algorithm that transforms any instance (x, k) into an equivalent instance (x', k') (the **kernel**) such that $|x'| + k' \leq g(k)$.

$(x, k) \in L \Leftrightarrow (x', k') \in L$

independent of $|x|$

Theorem
A decidable problem is in FPT if and only if it has a kernelization.

Which of the two directions is easy to prove?

Proof (of the difficult direction)

- let $\mathcal{A}$ be an algorithm that solves (x, k) in $f(k)|x|^c$ time
- **case 1:** $f(k) \leq |x|$
  - solve (x, k) with $\mathcal{A}$ in polynomial time
  - kernelization: return trivial kernel
- **case 2:** $f(k) > |x|$

Kernelization – formal definition

poly-time required; FPT-time not allowed

Definition
For a parameterized problem L, a **kernelization** is a polynomial-time algorithm that transforms any instance (x, k) into an equivalent instance (x', k') (the **kernel**) such that $|x'| + k' \leq g(k)$.

$(x, k) \in L \Leftrightarrow (x', k') \in L$

independent of $|x|$

Theorem
A decidable problem is in FPT if and only if it has a kernelization.

Which of the two directions is easy to prove?

Proof (of the difficult direction)

- let $\mathcal{A}$ be an algorithm that solves (x, k) in $f(k)|x|^c$ time
- **case 1:** $f(k) \leq |x|$
  - solve (x, k) with $\mathcal{A}$ in polynomial time
  - kernelization: return trivial kernel
- **case 2:** $f(k) > |x|$
  - (x, k) itself is already a suitable kernel
  - kernelization: just return (x, k)

The size of a kernel

VERTEX COVER kernelization using the theorem

- FPT-algorithm for VERTEX COVER: $f(k) \in O(1.2529^k)$

The size of a kernel

VERTEX COVER kernelization using the theorem

- FPT-algorithm for VERTEX COVER: $f(k) \in O(1.2529^k)$
- case-2 situation: $f(x) > |x| \rightarrow$ kernel of size $O(1.2529^k)$

The size of a kernel

VERTEX COVER kernelization using the theorem

- FPT-algorithm for VERTEX COVER: $f(k) \in O(1.2529^k)$
- case-2 situation: $f(x) > |x| \rightarrow$ kernel of size $O(1.2529^k)$
- we know: there is a kernel of size k^2 and one with $2k$ vertices

The size of a kernel

VERTEX COVER kernelization using the theorem

- FPT-algorithm for VERTEX COVER: $f(k) \in O(1.2529^k)$
- case-2 situation: $f(x) > |x| \rightarrow$ kernel of size $O(1.2529^k)$
- we know: there is a kernel of size k^2 and one with $2k$ vertices

Is there a polynomial kernel?

- the theorem can only give us super-polynomial kernels (unless the problem is already in P)

The size of a kernel

VERTEX COVER kernelization using the theorem

- FPT-algorithm for VERTEX COVER: $f(k) \in O(1.2529^k)$
- case-2 situation: $f(x) > |x| \rightarrow$ kernel of size $O(1.2529^k)$
- we know: there is a kernel of size k^2 and one with $2k$ vertices

Is there a polynomial kernel?

- the theorem can only give us super-polynomial kernels (unless the problem is already in P)
- having a polynomial kernel seems stronger than being in FPT

The size of a kernel

VERTEX COVER kernelization using the theorem

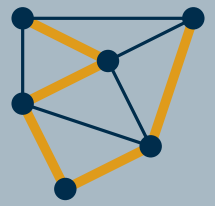
- FPT-algorithm for VERTEX COVER: $f(k) \in O(1.2529^k)$
- case-2 situation: $f(x) > |x| \rightarrow$ kernel of size $O(1.2529^k)$
- we know: there is a kernel of size k^2 and one with $2k$ vertices

Is there a polynomial kernel?

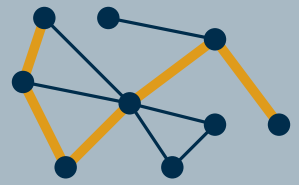
- the theorem can only give us super-polynomial kernels (unless the problem is already in P)
- having a polynomial kernel seems stronger than being in FPT
- today: probably no polynomial kernel for longest path

Cross-composition: Hamiltonian $\rightarrow$ longest path

Problem: HAMILTONIAN PATH
Given a graph G. Does G have a path that visits all vertices?

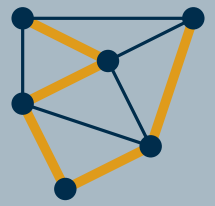


Problem: LONGEST PATH
Given a graph G and a parameter k. Does G have a path of size k?

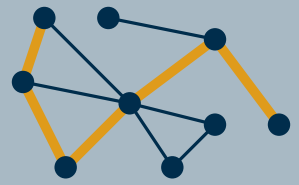


Cross-composition: Hamiltonian $\rightarrow$ longest path

Problem: HAMILTONIAN PATH
Given a graph G. Does G have a path that visits all vertices?

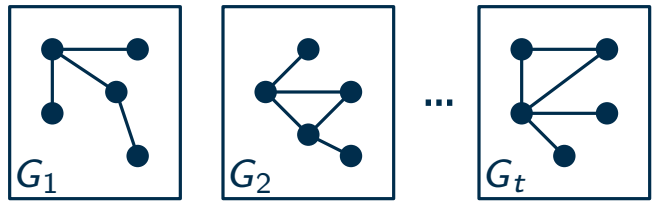


Problem: LONGEST PATH
Given a graph G and a parameter k. Does G have a path of size k?



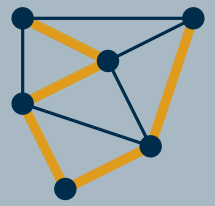
Many instances of HAMILTONIAN PATH (HAMP)

- given: tuple $(G_1, \ldots, G_t)$ of t n-vertex instances of HAMP
- goal: decide whether at least one G_i is a yes-instances

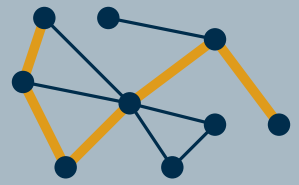


Cross-composition: Hamiltonian $\rightarrow$ longest path

Problem: HAMILTONIAN PATH
Given a graph G. Does G have a path that visits all vertices?



Problem: LONGEST PATH
Given a graph G and a parameter k. Does G have a path of size k?

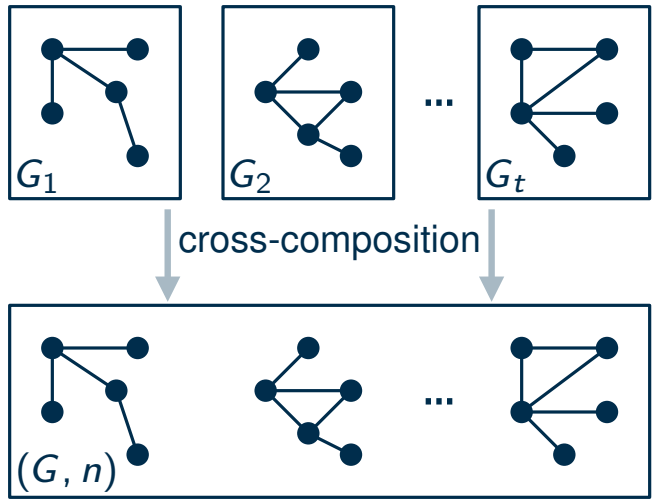


Many instances of HAMILTONIAN PATH (HAMP)

- given: tuple $(G_1, \ldots, G_t)$ of t n-vertex instances of HAMP
- goal: decide whether at least one G_i is a yes-instances

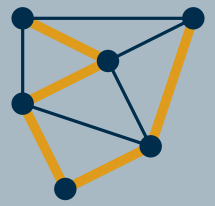
Cross-composition into LONGEST PATH (LONGP)

- form disjoint union $G = \bigcup G_i$ and set parameter $k = n$
- (G, n) is yes-instance of LONGP $\Leftrightarrow$
  one of the G_i is yes-instance of HAMP

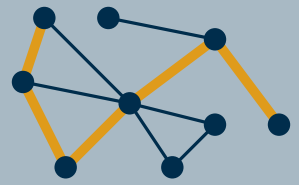


Cross-composition: Hamiltonian $\rightarrow$ longest path

Problem: HAMILTONIAN PATH
Given a graph G. Does G have a path that visits all vertices?



Problem: LONGEST PATH
Given a graph G and a parameter k. Does G have a path of size k?

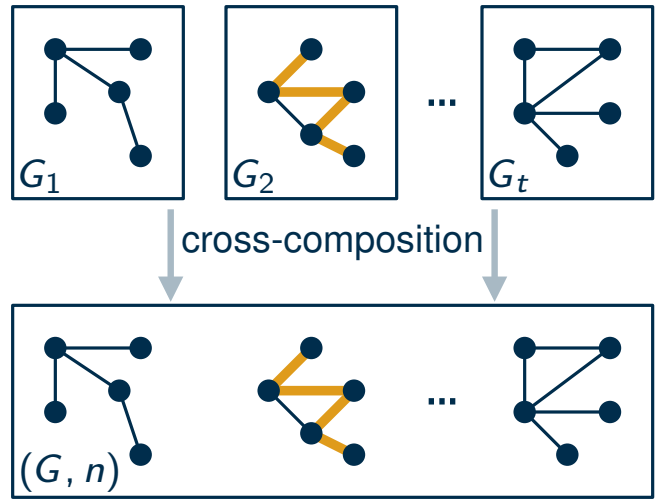


Many instances of HAMILTONIAN PATH (HAMP)

- given: tuple $(G_1, \ldots, G_t)$ of t n-vertex instances of HAMP
- goal: decide whether at least one G_i is a yes-instances

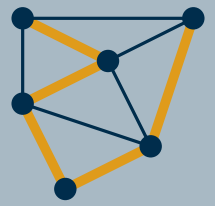
Cross-composition into LONGEST PATH (LONGP)

- form disjoint union $G = \bigcup G_i$ and set parameter $k = n$
- (G, n) is yes-instance of LONGP $\Leftrightarrow$
  one of the G_i is yes-instance of HAMP

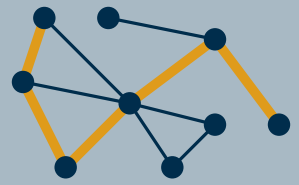


Cross-composition: Hamiltonian → longest path

Problem: HAMILTONIAN PATH
Given a graph G. Does G have a path that visits all vertices?



Problem: LONGEST PATH
Given a graph G and a parameter k. Does G have a path of size k?



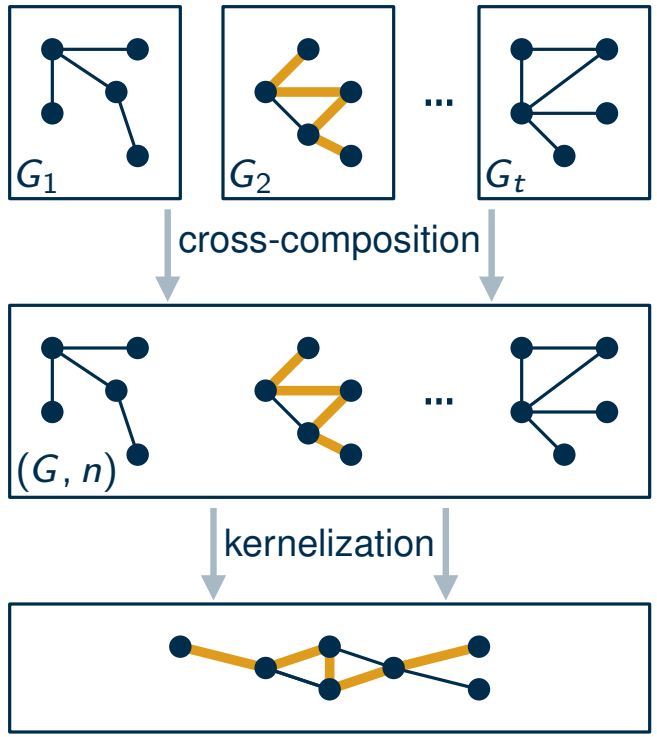
Many instances of HAMILTONIAN PATH (HAMP)

- given: tuple $(G_1, \ldots, G_t)$ of t n-vertex instances of HAMP
- goal: decide whether at least one G_i is a yes-instances

Cross-composition into LONGEST PATH (LONGP)

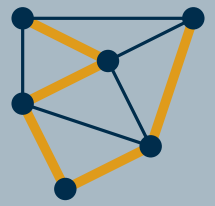
- form disjoint union $G = \bigcup G_i$ and set parameter $k = n$
- (G, n) is yes-instance of LONGP $\Leftrightarrow$ one of the G_i is yes-instance of HAMP

Does LONGP admit a polynomial kernel?

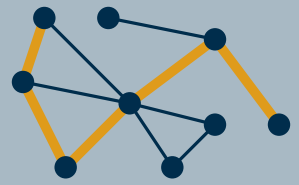


Cross-composition: Hamiltonian → longest path

Problem: HAMILTONIAN PATH
Given a graph G. Does G have a path that visits all vertices?



Problem: LONGEST PATH
Given a graph G and a parameter k. Does G have a path of size k?



Many instances of HAMILTONIAN PATH (HAMP)

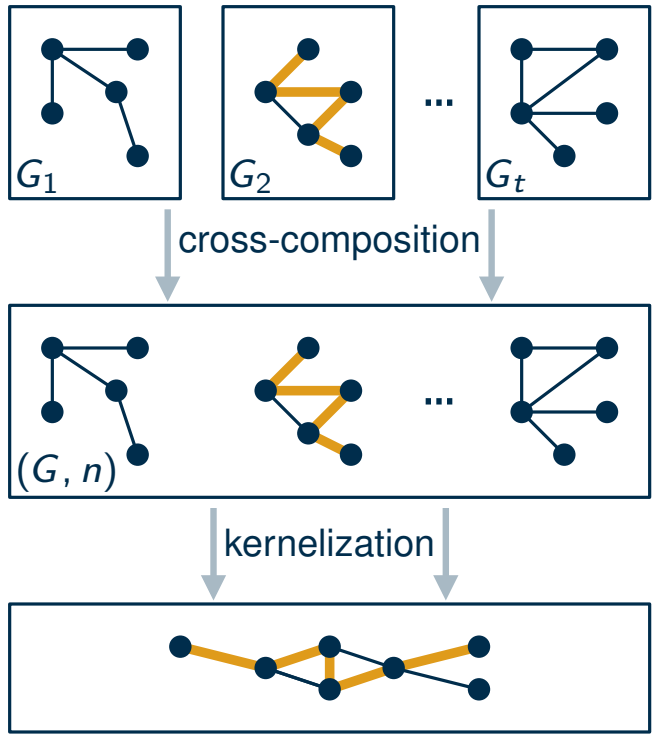
- given: tuple $(G_1, \ldots, G_t)$ of t n-vertex instances of HAMP
- goal: decide whether at least one G_i is a yes-instances

Cross-composition into LONGEST PATH (LONGP)

- form disjoint union $G = \bigcup G_i$ and set parameter $k = n$
- (G, n) is yes-instance of LONGP $\Leftrightarrow$ one of the G_i is yes-instance of HAMP

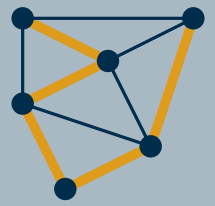
Does LONGP admit a polynomial kernel?

- kernel of size n^c seems implausible if $t \gg n^c$

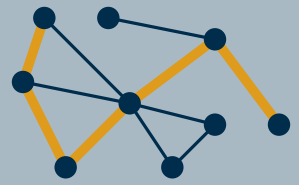


Cross-composition: Hamiltonian → longest path

Problem: HAMILTONIAN PATH
Given a graph G. Does G have a path that visits all vertices?



Problem: LONGEST PATH
Given a graph G and a parameter k. Does G have a path of size k?



Many instances of HAMILTONIAN PATH (HAMP)

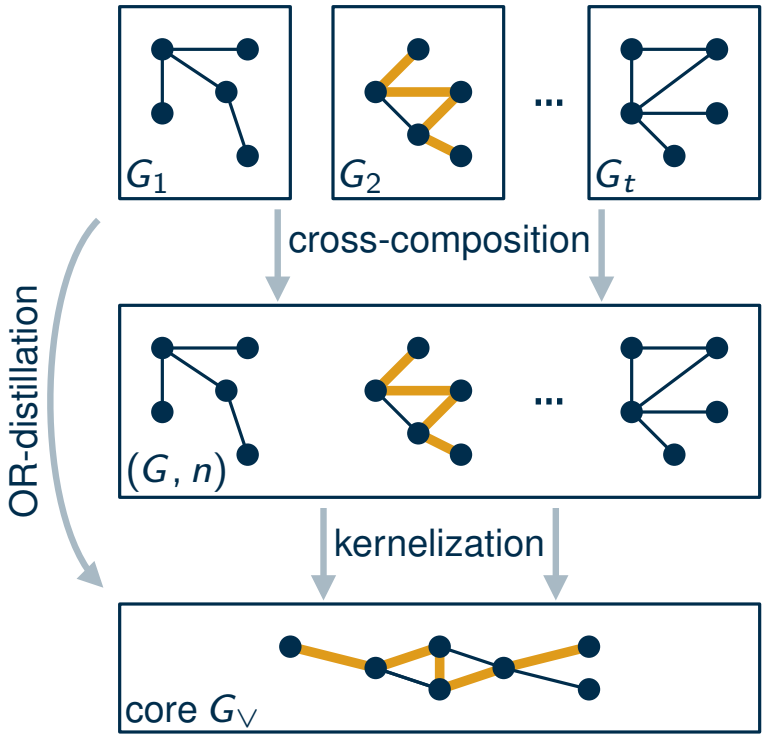
- given: tuple $(G_1, \ldots, G_t)$ of t n-vertex instances of HAMP
- goal: decide whether at least one G_i is a yes-instances

Cross-composition into LONGEST PATH (LONGP)

- form disjoint union $G = \bigcup G_i$ and set parameter $k = n$
- (G, n) is yes-instance of LONGP $\Leftrightarrow$ one of the G_i is yes-instance of HAMP

Does LONGP admit a polynomial kernel?

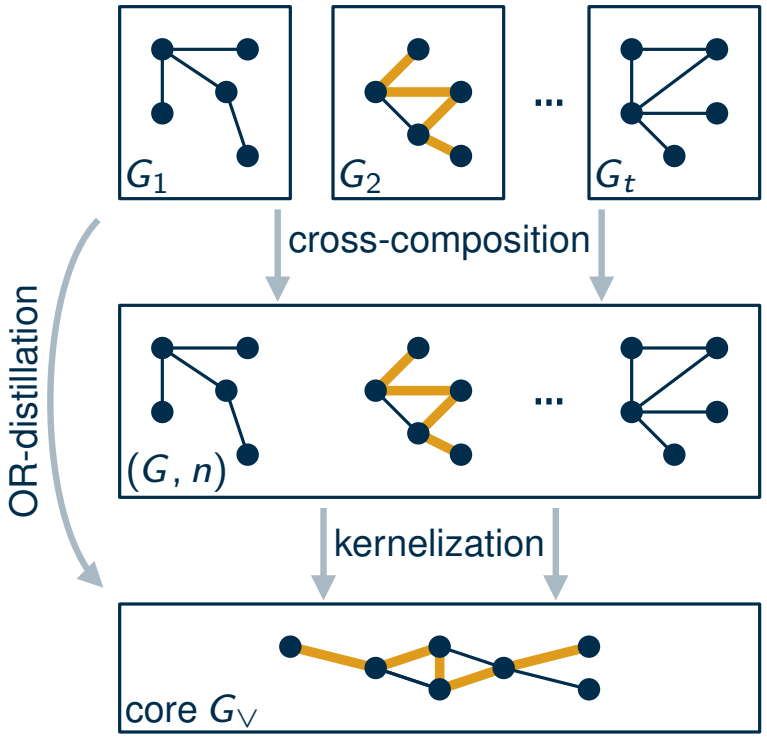
- kernel of size n^c seems implausible if $t \gg n^c$



OR-Distillation

OR-Distillation algorithm $\mathcal{D}$ from HAMP into LONGP

- turns a t-tuple of n-vertex graphs $(G_1, \ldots, G_t)$ into an $n^{O(1)}$-vertex graph $G_\vee$ in time $(tn)^{O(1)}$
- $\bigvee_{i \in [t]}$ (G_i is yes-instance of HAMP) $\Leftrightarrow$ $G_\vee$ is yes-instance of LONGP (for some parameter)

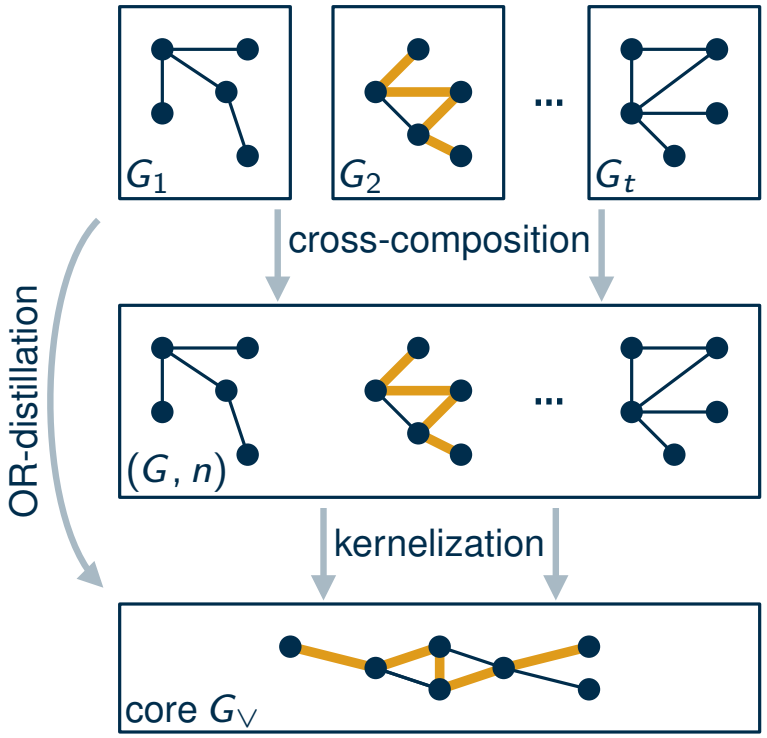


OR-Distillation

OR-Distillation algorithm $\mathcal{D}$ from HAMP into LONGP

- turns a t-tuple of n-vertex graphs $(G_1, \ldots, G_t)$ into an $n^{O(1)}$-vertex graph $G_\vee$ in time $(tn)^{O(1)}$
- $\bigvee_{i \in [t]}$ (G_i is yes-instance of HAMP) $\Leftrightarrow$ $G_\vee$ is yes-instance of LONGP (for some parameter)

Observation
If LONGP admits a polynomial kernel, then there is an OR-distillation from HAMP into LONGP.



OR-Distillation

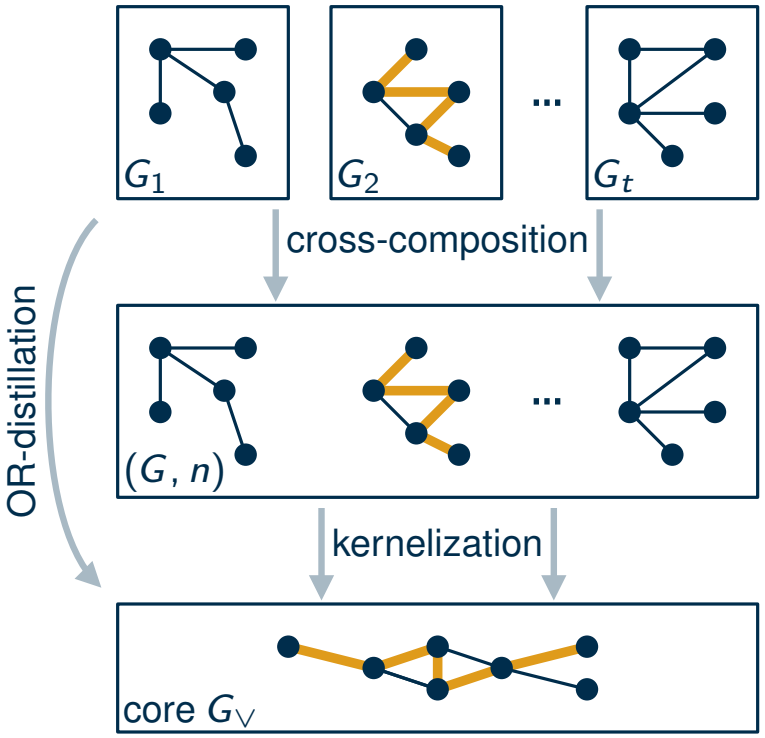
OR-Distillation algorithm $\mathcal{D}$ from HAMP into LONGP

- turns a t-tuple of n-vertex graphs $(G_1, \ldots, G_t)$ into an $n^{O(1)}$-vertex graph $G_\vee$ in time $(tn)^{O(1)}$
- $\bigvee_{i \in [t]}$ (G_i is yes-instance of HAMP) $\Leftrightarrow$ $G_\vee$ is yes-instance of LONGP (for some parameter)

Observation
If LONGP admits a polynomial kernel, then there is an OR-distillation from HAMP into LONGP.

Plan for the following

- show that the OR-distillation implies HAMP $\in$ coNP/poly



OR-Distillation

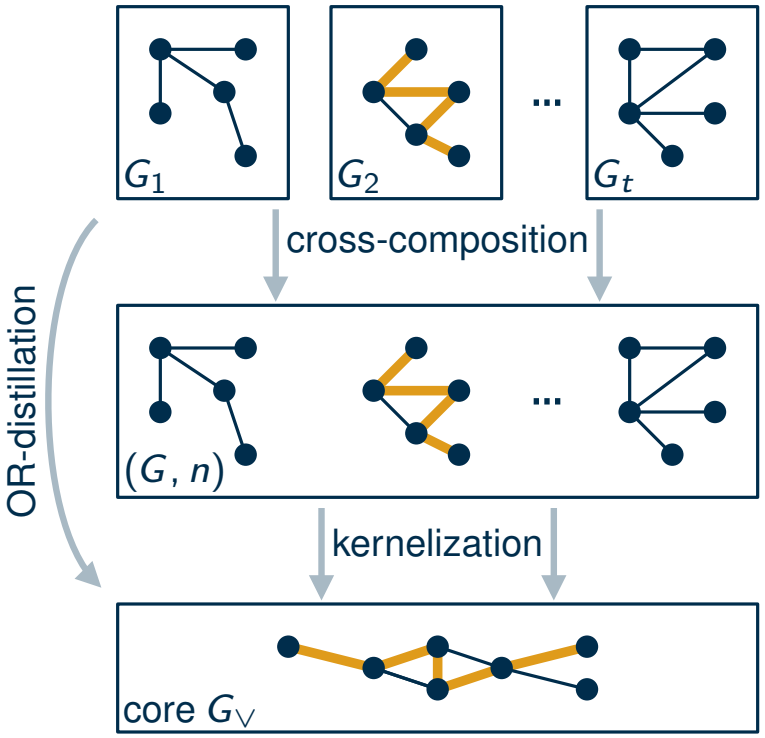
OR-Distillation algorithm $\mathcal{D}$ from HAMP into LONGP

- turns a t-tuple of n-vertex graphs $(G_1, \ldots, G_t)$ into an $n^{O(1)}$-vertex graph $G_\vee$ in time $(tn)^{O(1)}$
- $\bigvee_{i \in [t]}$ (G_i is yes-instance of HAMP) $\Leftrightarrow$ $G_\vee$ is yes-instance of LONGP (for some parameter)

Observation
If LONGP admits a polynomial kernel, then there is an OR-distillation from HAMP into LONGP.

Plan for the following

- show that the OR-distillation implies HAMP $\in$ coNP/poly
- as HAMP is NP-complete, this implies NP $\subseteq$ coNP/poly

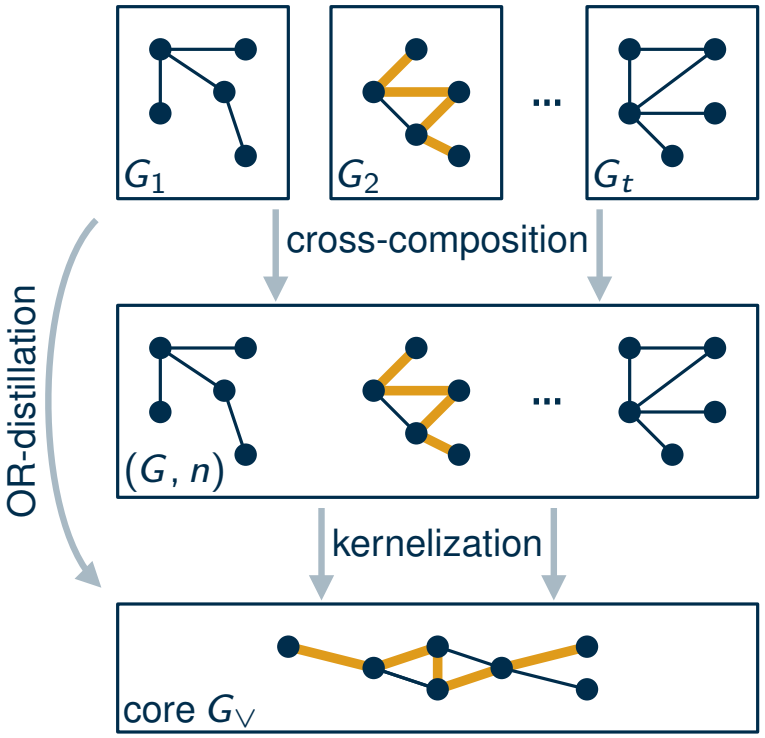


OR-Distillation

OR-Distillation algorithm $\mathcal{D}$ from HAMP into LONGP

- turns a t-tuple of n-vertex graphs $(G_1, \ldots, G_t)$ into an $n^{O(1)}$-vertex graph $G_\vee$ in time $(tn)^{O(1)}$
- $\bigvee_{i \in [t]}$ (G_i is yes-instance of HAMP) $\Leftrightarrow$ $G_\vee$ is yes-instance of LONGP (for some parameter)

Observation
If LONGP admits a polynomial kernel, then there is an OR-distillation from HAMP into LONGP.



Plan for the following

- show that the OR-distillation implies HAMP $\in$ coNP/poly
- as HAMP is NP-complete, this implies NP $\subseteq$ coNP/poly
- as we believe NP $\not\subseteq$ coNP/poly, we should believe that LONGP does not admit a polynomial kernel

coNP/poly

coNP/poly

Nondeterministic polynomial

- guess certificate for input being a yes-instance
- verify certificate in polynomial time

coNP/poly

Nondeterministic polynomial

- guess certificate for input being a yes-instance
- verify certificate in polynomial time

Complement

- verify certificate for no-instance instead

coNP/poly

Nondeterministic polynomial

- guess certificate for input being a yes-instance
- verify certificate in polynomial time

Complement

- verify certificate for no-instance instead

Polynomial advice

- there are advice strings $\alpha_1, \alpha_2, \ldots$
- size-n input $\rightarrow$ algo has access to α_n
- size of α_n is polynomial in n

coNP/poly

Nondeterministic polynomial

- guess certificate for input being a yes-instance
- verify certificate in polynomial time

Complement

- verify certificate for no-instance instead

Polynomial advice

- there are advice strings $\alpha_1, \alpha_2, \ldots$
- size-n input $\rightarrow$ algo has access to α_n
- size of α_n is polynomial in n

To prove HAMILTONIAN PATH $\in$ coNP/poly

- for every n, define an advice string α_n of length $n^{O(1)}$

coNP/poly

Nondeterministic polynomial

- guess certificate for input being a yes-instance
- verify certificate in polynomial time

Complement

- verify certificate for no-instance instead

Polynomial advice

- there are advice strings $\alpha_1, \alpha_2, \ldots$
- size-n input $\rightarrow$ algo has access to α_n
- size of α_n is polynomial in n

To prove HAMILTONIAN PATH $\in$ coNP/poly

- for every n, define an advice string α_n of length $n^{O(1)}$
- define no-certificates such that every no-instance has one

coNP/poly

Nondeterministic polynomial

- guess certificate for input being a yes-instance
- verify certificate in polynomial time

Complement

- verify certificate for no-instance instead

Polynomial advice

- there are advice strings $\alpha_1, \alpha_2, \ldots$
- size-n input $\rightarrow$ algo has access to α_n
- size of α_n is polynomial in n

To prove HAMILTONIAN PATH $\in$ coNP/poly

- for every n, define an advice string α_n of length $n^{O(1)}$
- define no-certificates such that every no-instance has one
- give a poly-time algorithm that:
  - has access to α_n for graphs on n vertices
  - checks for a guess whether it is a no-certificate of the given graph

coNP/poly

Nondeterministic polynomial

- guess certificate for input being a yes-instance
- verify certificate in polynomial time

Complement

- verify certificate for no-instance instead

Polynomial advice

- there are advice strings $\alpha_1, \alpha_2, \ldots$
- size-n input $\rightarrow$ algo has access to α_n
- size of α_n is polynomial in n

To prove HAMILTONIAN PATH $\in$ coNP/poly

- for every n, define an advice string α_n of length $n^{O(1)}$
- define no-certificates such that every no-instance has one

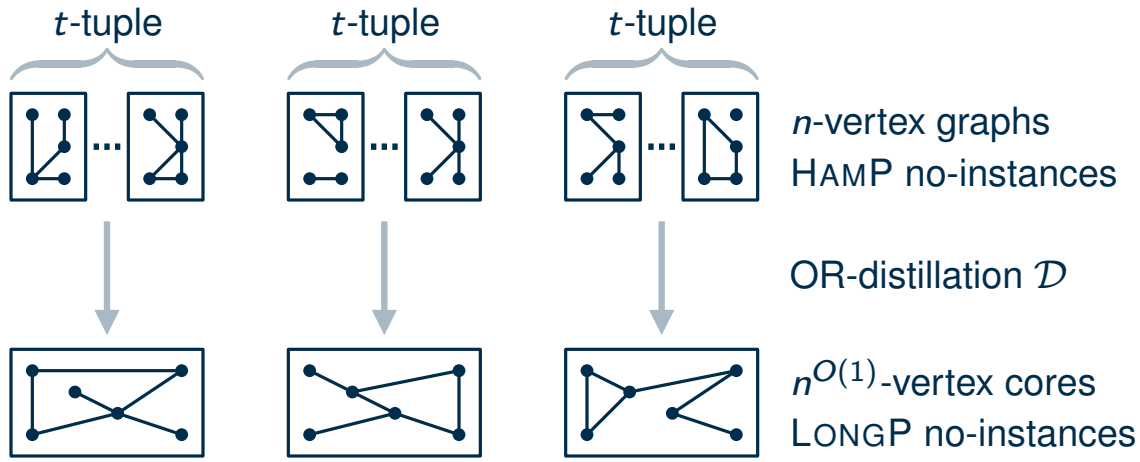
} existence is enough, no need to construct them efficiently

- give a poly-time algorithm that:
  - has access to α_n for graphs on n vertices
  - checks for a guess whether it is a no-certificate of the given graph

A coNP/poly algorithm for HAMP – overview

Advice string α_n for graphs with n vertices

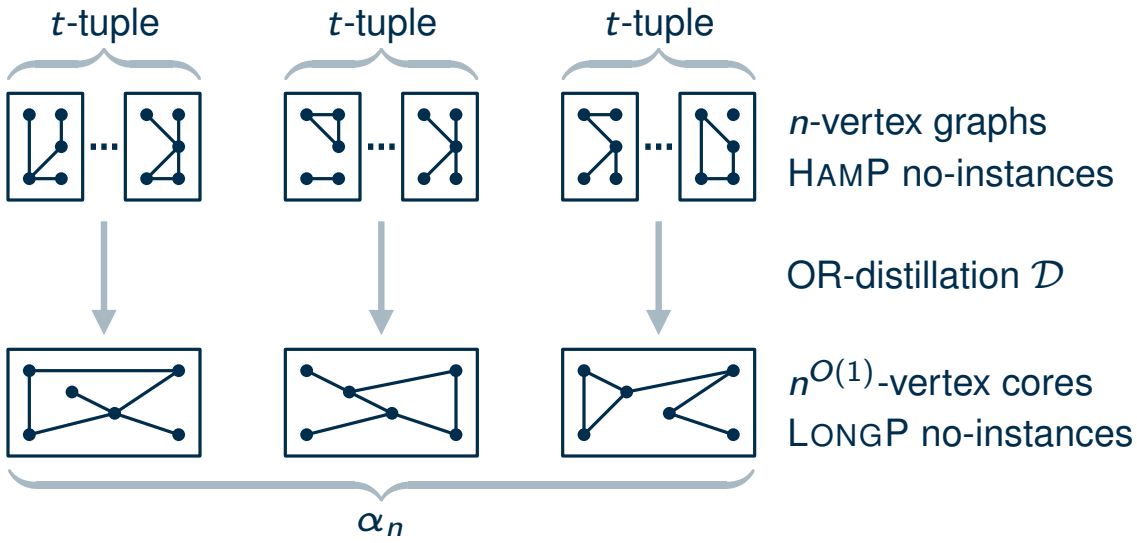
- run $\mathcal{D}$ on multiple t-tuples of n-vertex graphs that are no-instances of HAMP



A coNP/poly algorithm for HAMP – overview

Advice string α_n for graphs with n vertices

- run $\mathcal{D}$ on multiple t-tuples of n-vertex graphs that are no-instances of HAMP
- α_n union of the resulting cores



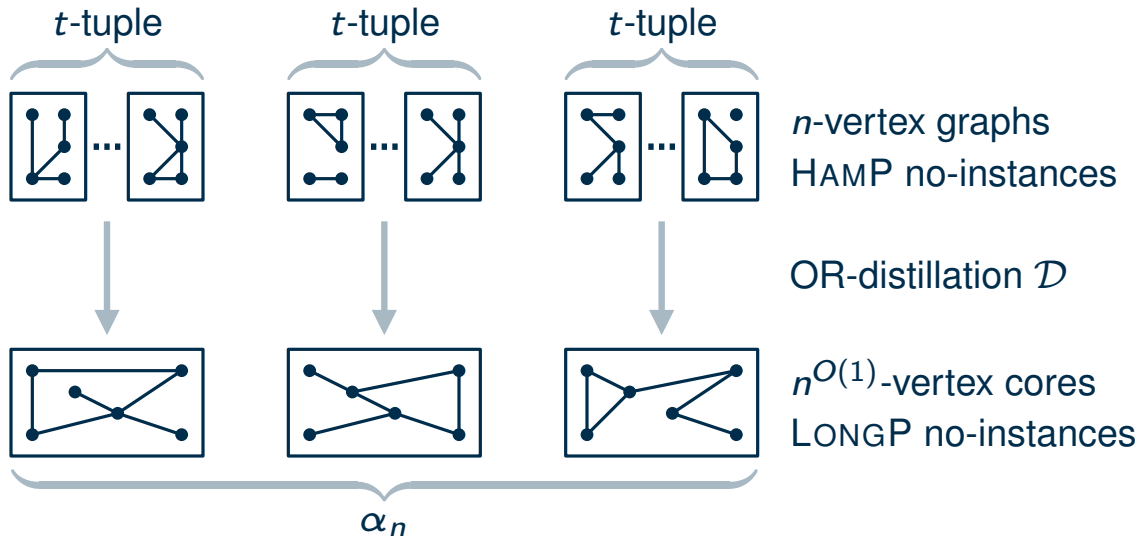
A coNP/poly algorithm for HAMP – overview

Advice string α_n for graphs with n vertices

- run $\mathcal{D}$ on multiple t-tuples of n-vertex graphs that are no-instances of HAMP
- α_n union of the resulting cores

No-certificates

- $(G_1, \ldots, G_t)$ is a no-certificate for each G_i if α_n contains its OR-distillation $\mathcal{D}(G_1, \ldots, G_t)$



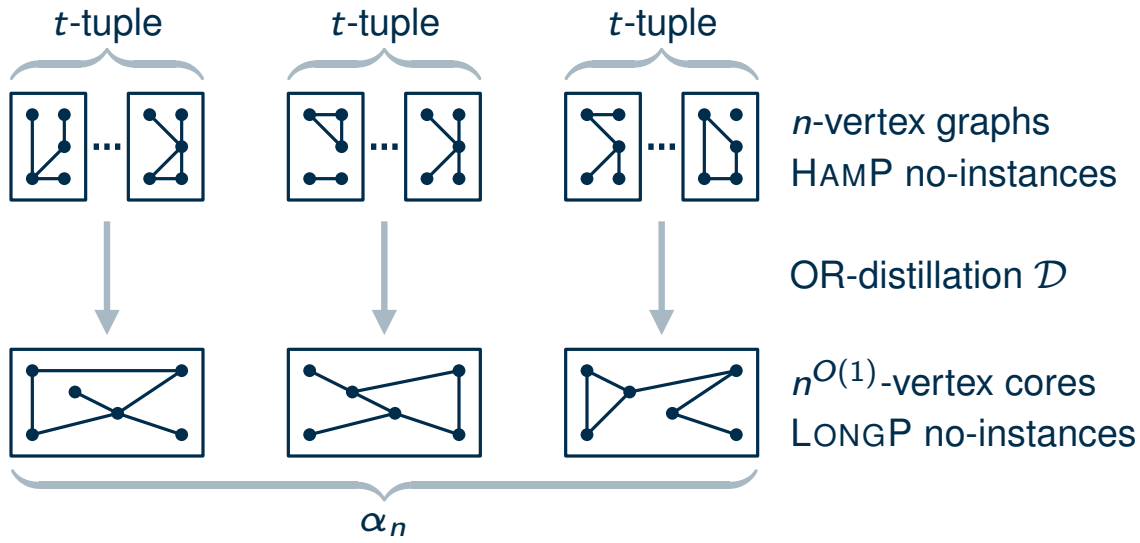
A coNP/poly algorithm for HAMP – overview

Advice string α_n for graphs with n vertices

- run $\mathcal{D}$ on multiple t-tuples of n-vertex graphs that are no-instances of HAMP
- α_n union of the resulting cores

No-certificates

- $(G_1, \ldots, G_t)$ is a no-certificate for each G_i if α_n contains its OR-distillation $\mathcal{D}(G_1, \ldots, G_t)$
- todo: every no-instance needs a certificate



A coNP/poly algorithm for HAMP – overview

Advice string α_n for graphs with n vertices

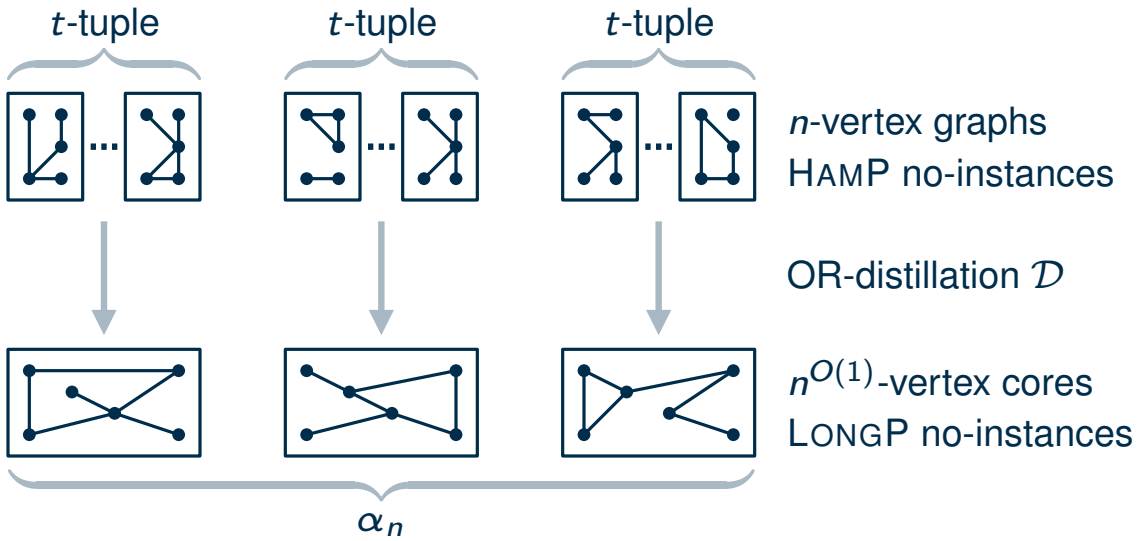
- run $\mathcal{D}$ on multiple t-tuples of n-vertex graphs that are no-instances of HAMP
- α_n union of the resulting cores

No-certificates

- $(G_1, \ldots, G_t)$ is a no-certificate for each G_i if α_n contains its OR-distillation $\mathcal{D}(G_1, \ldots, G_t)$
- todo: every no-instance needs a certificate

A coNP/poly algorithm

- input: graph G with n vertices
- guess a t-tuple $(G_1, \ldots, G_t)$ containing G
- use $\mathcal{D}$ and α_n to check if it is a no-certificate



A coNP/poly algorithm for HAMP – overview

Advice string α_n for graphs with n vertices

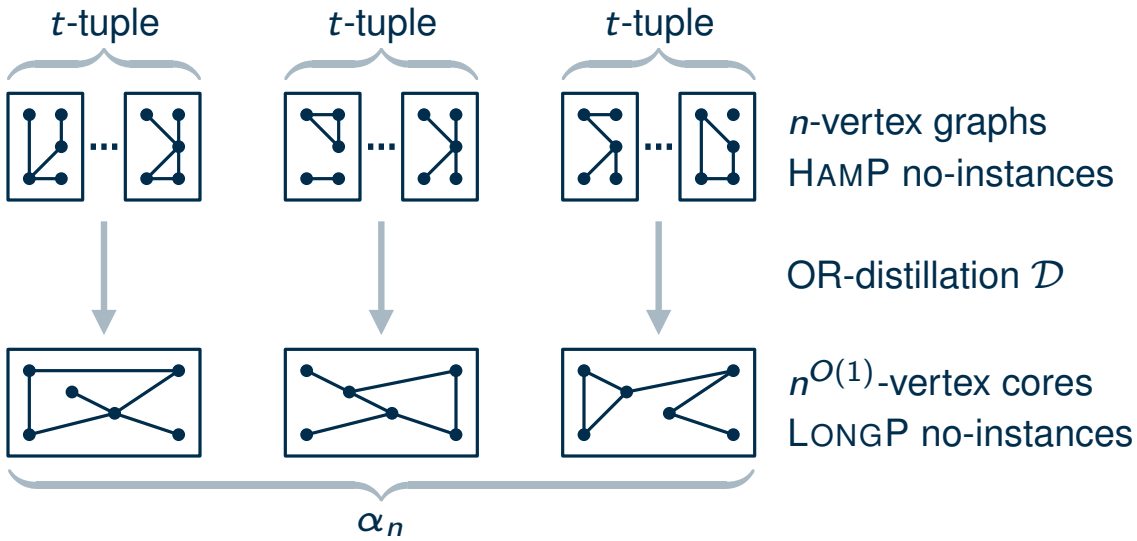
- run $\mathcal{D}$ on multiple t-tuples of n-vertex graphs that are no-instances of HAMP
- α_n union of the resulting cores

No-certificates

- $(G_1, \ldots, G_t)$ is a no-certificate for each G_i if α_n contains its OR-distillation $\mathcal{D}(G_1, \ldots, G_t)$
- todo: every no-instance needs a certificate

A coNP/poly algorithm

- input: graph G with n vertices
- guess a t-tuple $(G_1, \ldots, G_t)$ containing G
- use $\mathcal{D}$ and α_n to check if it is a no-certificate



Todo

- choose clever set of cores:
  - every no-instance needs certificate tuple
  - advice α_n should be poly $\rightarrow$ $n^{O(1)}$ cores
  - $\mathcal{D}$ should run in $n^{O(1)}$ time $\rightarrow$ $t \in n^{O(1)}$

A coNP/poly algorithm for HAMP – overview

Advice string α_n for graphs with n vertices

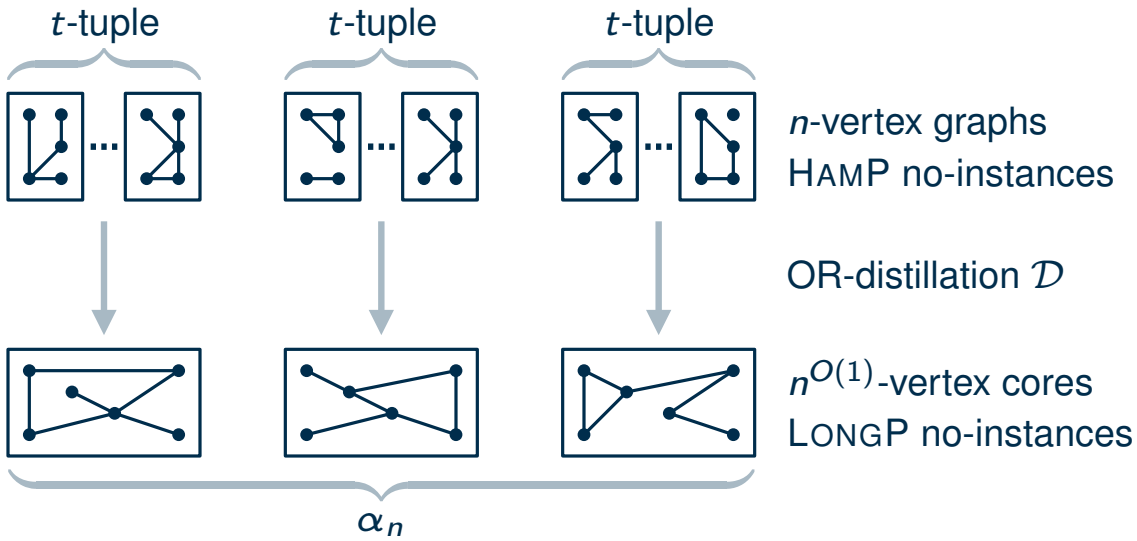
- run $\mathcal{D}$ on multiple t-tuples of n-vertex graphs that are no-instances of HAMP
- α_n union of the resulting cores

No-certificates

- $(G_1, \ldots, G_t)$ is a no-certificate for each G_i if α_n contains its OR-distillation $\mathcal{D}(G_1, \ldots, G_t)$
- todo: every no-instance needs a certificate

A coNP/poly algorithm

- input: graph G with n vertices
- guess a t-tuple $(G_1, \ldots, G_t)$ containing G
- use $\mathcal{D}$ and α_n to check if it is a no-certificate

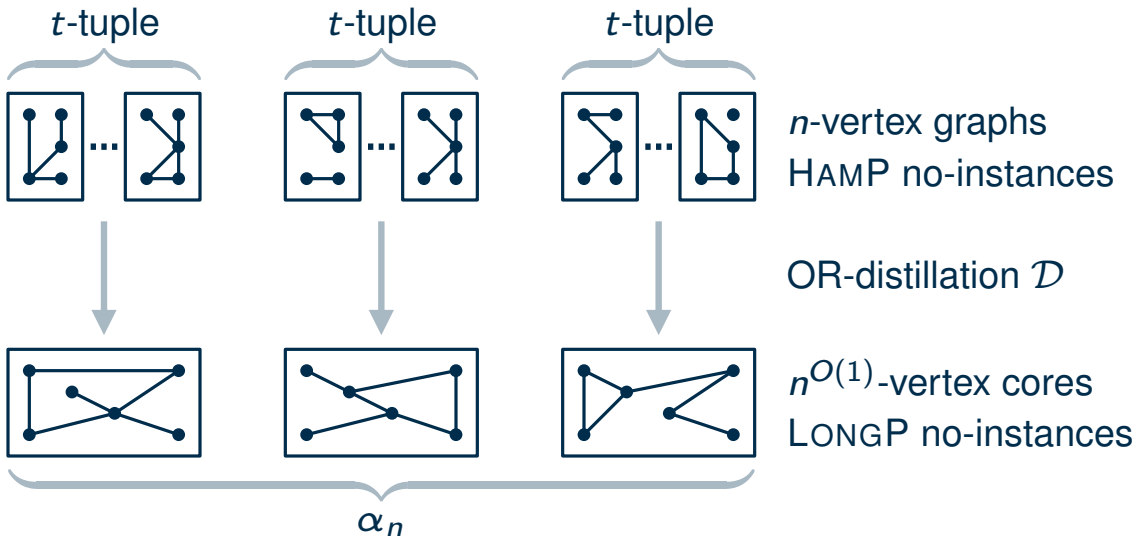


Todo

- choose clever set of cores:
  - every no-instance needs certificate tuple
  - advice α_n should be poly $\rightarrow$ $n^{O(1)}$ cores
  - $\mathcal{D}$ should run in $n^{O(1)}$ time $\rightarrow$ $t \in n^{O(1)}$
- use that disjoint tuples can share a core

Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.



Todo

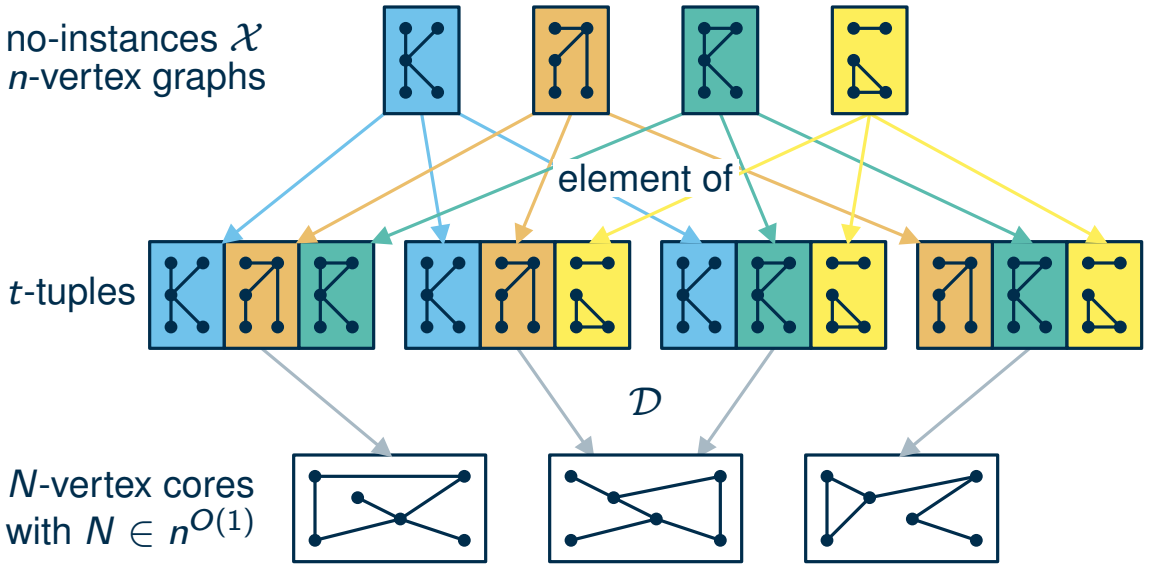
- choose clever set of cores:
  - every no-instance needs certificate tuple
  - advice α_n should be poly $\rightarrow$ $n^{O(1)}$ cores
  - $\mathcal{D}$ should run in $n^{O(1)}$ time $\rightarrow$ $t \in n^{O(1)}$
- use that disjoint tuples can share a core

Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

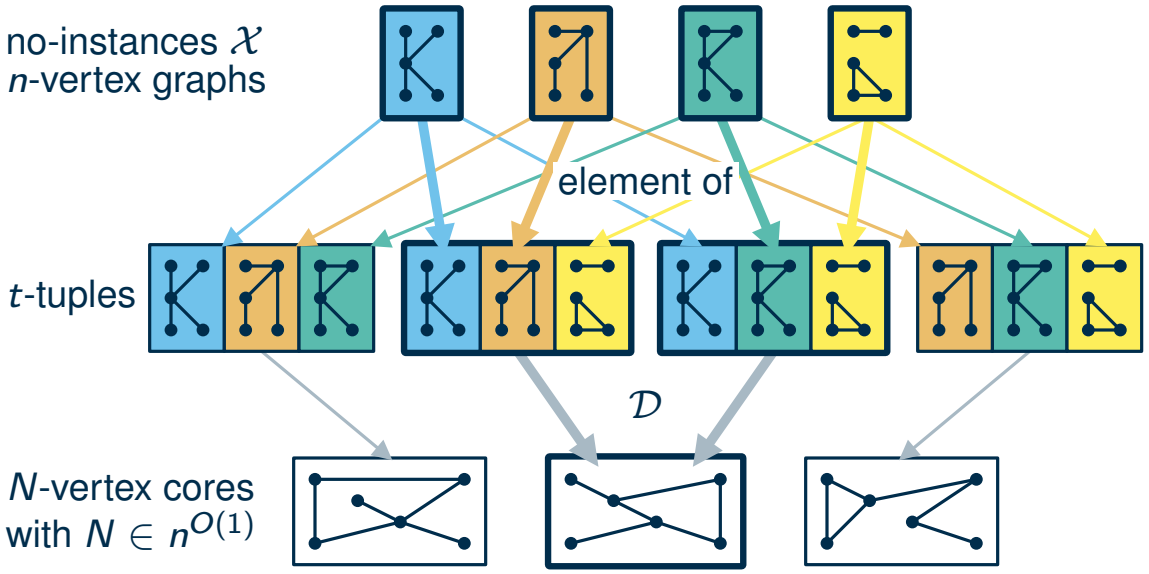
Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.



Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

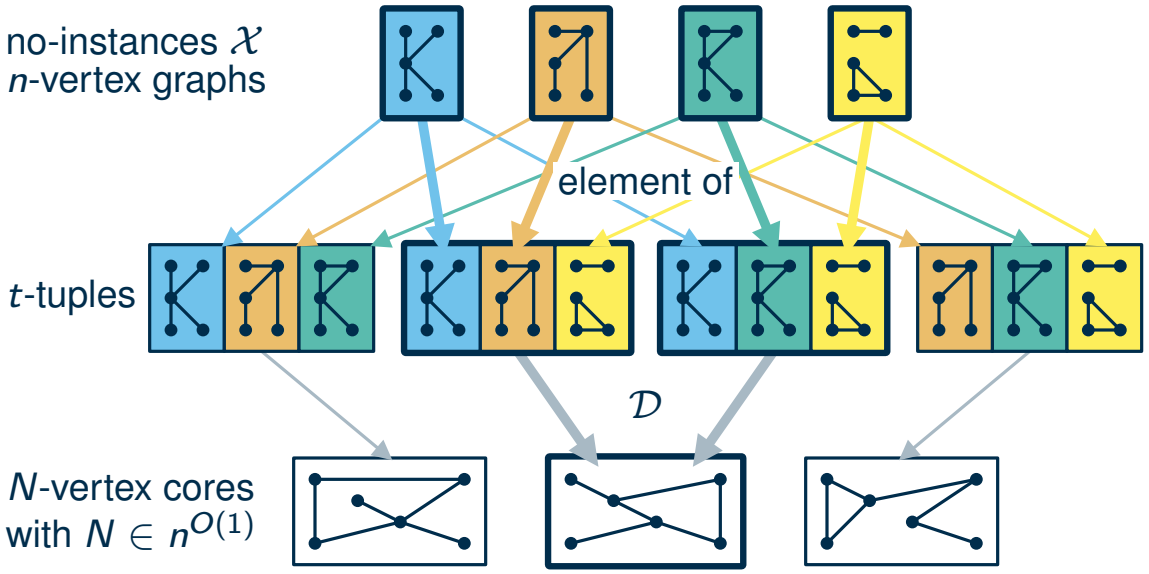


Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

Proof

- total number of possible cores: $\leq 2^{N^2}$ **Why?**

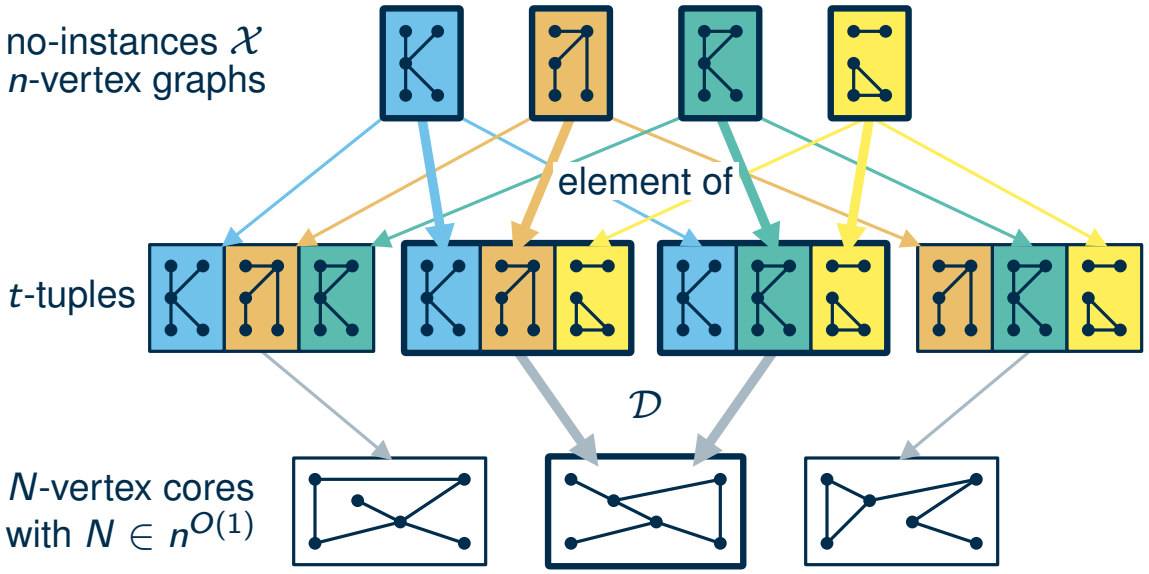


Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

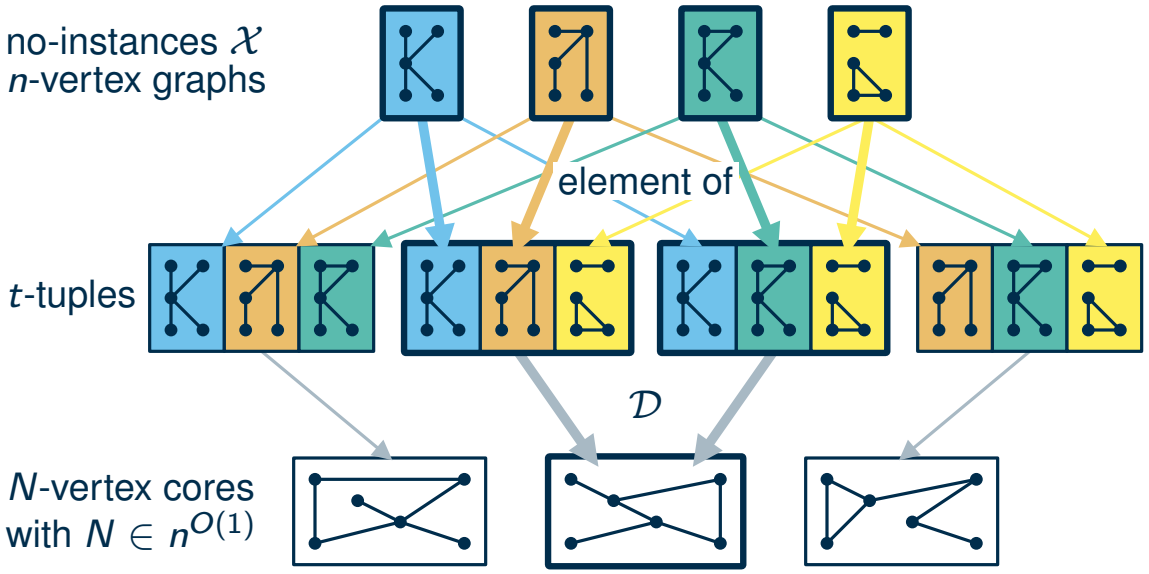
Proof

- total number of possible cores: $\leq 2^{N^2}$ Why?
- number of t-tuples: $|\mathcal{X}|^t$



Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

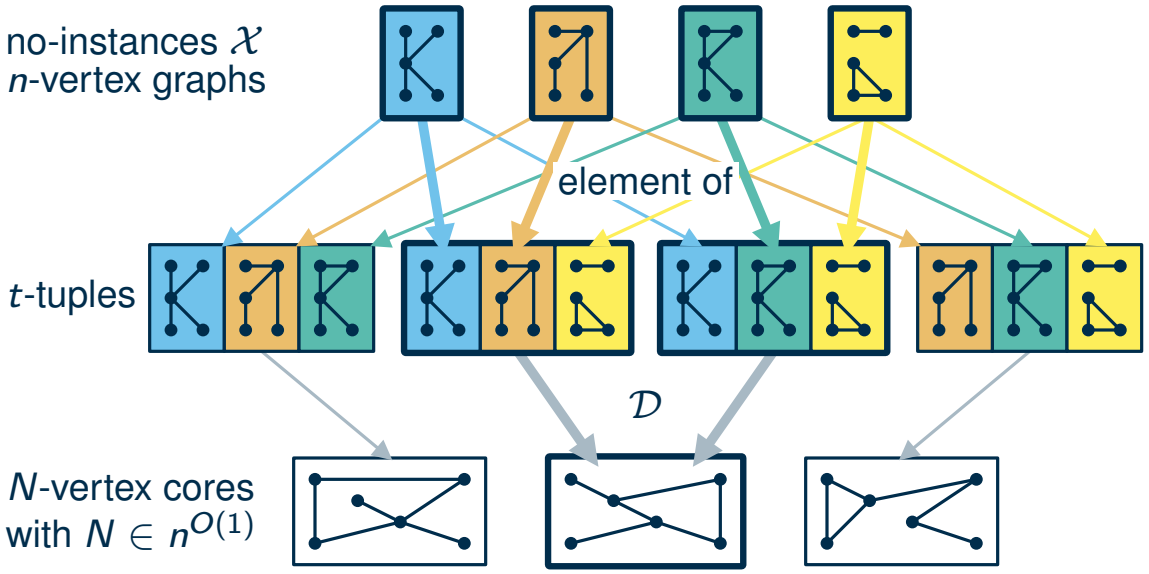


Proof

- total number of possible cores: $\leq 2^{N^2}$ Why?
- number of t-tuples: $|\mathcal{X}|^t$
- average number of t-tuples mapped to each core: $\geq \frac{|\mathcal{X}|^t}{2^{N^2}} = \left(\frac{|\mathcal{X}|}{2}\right)^t$ for $t = N^2$

Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

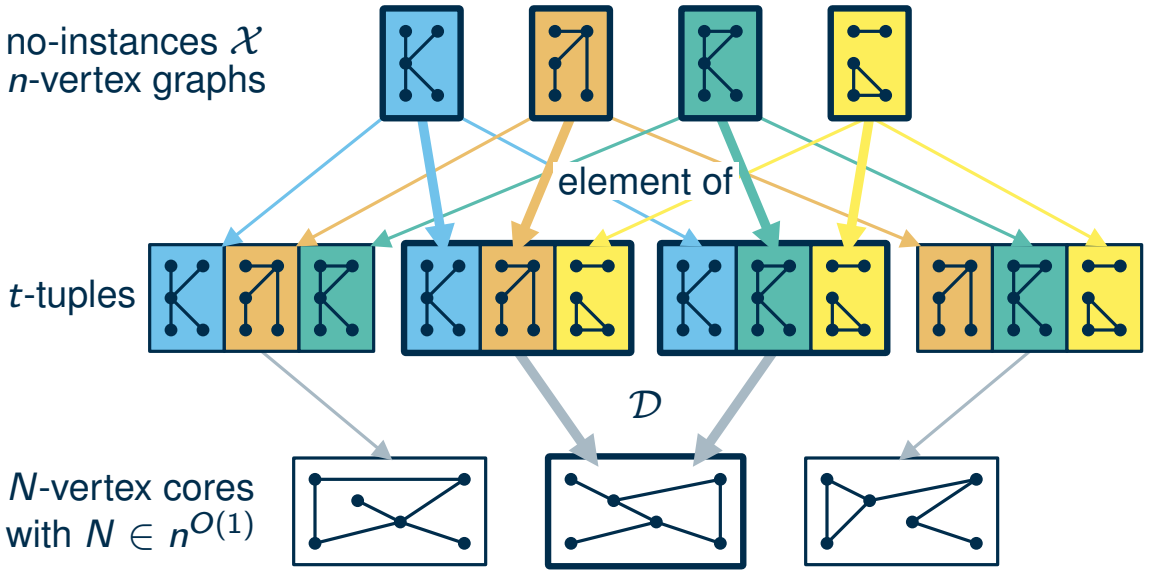


Proof

- total number of possible cores: $\leq 2^{N^2}$ Why?
- number of t-tuples: $|\mathcal{X}|^t$
- average number of t-tuples mapped to each core: $\geq \frac{|\mathcal{X}|^t}{2^{N^2}} = \left(\frac{|\mathcal{X}|}{2}\right)^t$ for $t = N^2$
- add core to α_n that covers at least $\left(\frac{|\mathcal{X}|}{2}\right)^t$ tuples

Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

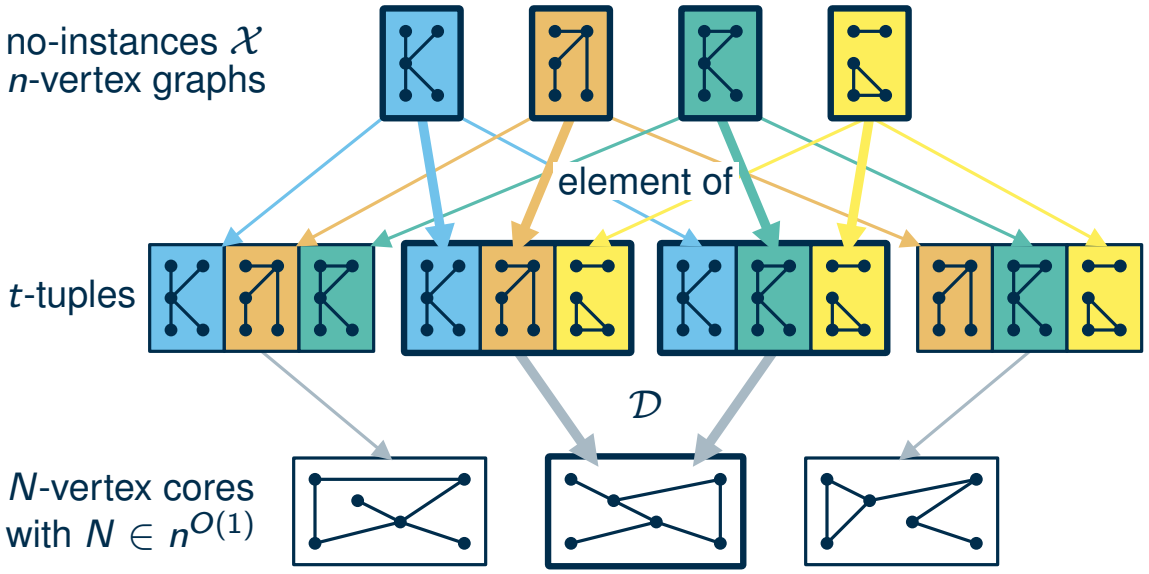


Proof

- total number of possible cores: $\leq 2^{N^2}$ Why?
- number of t-tuples: $|\mathcal{X}|^t$
- average number of t-tuples mapped to each core: $\geq \frac{|\mathcal{X}|^t}{2^{N^2}} = \left(\frac{|\mathcal{X}|}{2}\right)^t$ for $t = N^2$
- add core to α_n that covers at least $\left(\frac{|\mathcal{X}|}{2}\right)^t$ tuples
- k^t different t-tuples contain at least k different graphs $\rightarrow |\mathcal{X}|/2$ no-instances covered

Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

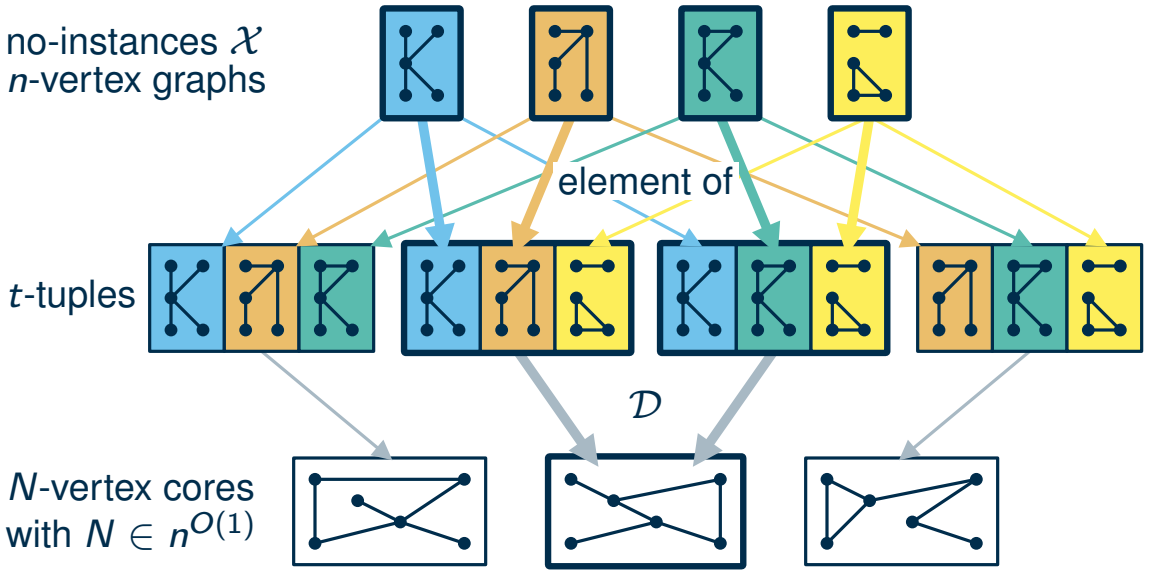


Proof

- total number of possible cores: $\leq 2^{N^2}$ Why?
- number of t-tuples: $|\mathcal{X}|^t$
- average number of t-tuples mapped to each core: $\geq \frac{|\mathcal{X}|^t}{2^{N^2}} = \left(\frac{|\mathcal{X}|}{2}\right)^t$ for $t = N^2$
- add core to α_n that covers at least $\left(\frac{|\mathcal{X}|}{2}\right)^t$ tuples
- k^t different t-tuples contain at least k different graphs $\rightarrow |\mathcal{X}|/2$ no-instances covered
- repeat with remaining no-instances $\rightarrow$ done after $\log |\mathcal{X}|$ steps

Existence of good advice

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.



Proof

- total number of possible cores: $\leq 2^{N^2}$ Why?
- number of t-tuples: $|\mathcal{X}|^t$
- average number of t-tuples mapped to each core: $\geq \frac{|\mathcal{X}|^t}{2^{N^2}} = \left(\frac{|\mathcal{X}|}{2}\right)^t$ for $t = N^2$
- add core to α_n that covers at least $\left(\frac{|\mathcal{X}|}{2}\right)^t$ tuples
- k^t different t-tuples contain at least k different graphs $\rightarrow |\mathcal{X}|/2$ no-instances covered
- repeat with remaining no-instances $\rightarrow$ done after $\log |\mathcal{X}|$ steps $\Rightarrow |\alpha_n| = n^{O(1)} \log |\mathcal{X}| \in n^{O(1)}$

Why?

Wrapping up

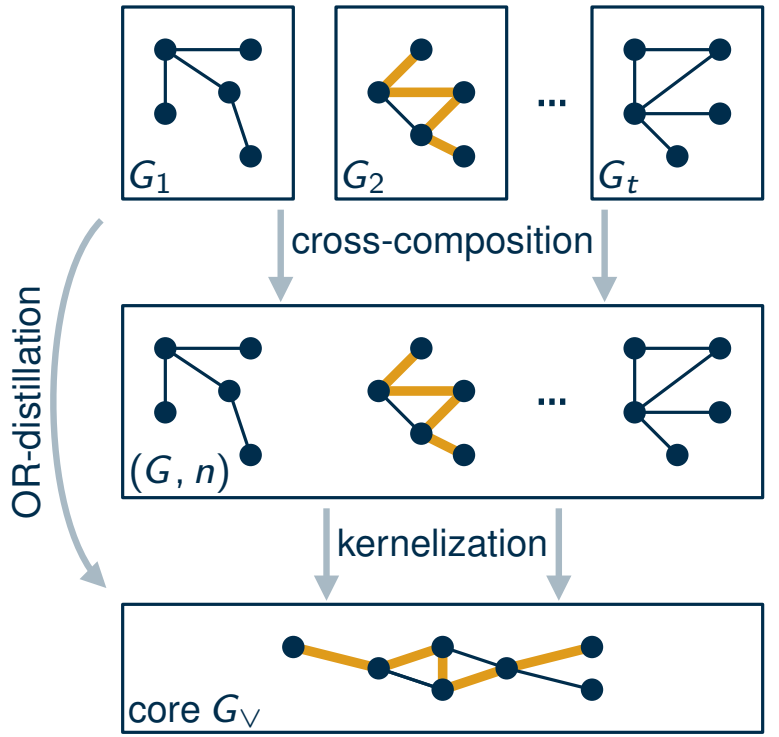
Theorem
If LONGEST PATH has a polynomial kernel, then NP $\subseteq$ coNP/poly. (via: HAMILTONIAN PATH $\in$ coNP/poly)

Wrapping up

Theorem
If LONGEST PATH has a polynomial kernel, then NP $\subseteq$ coNP/poly. (via: HAMILTONIAN PATH $\in$ coNP/poly)

Proof

- polynomial kernel for LONGP $\Rightarrow$ OR-distillation $\mathcal{D}$



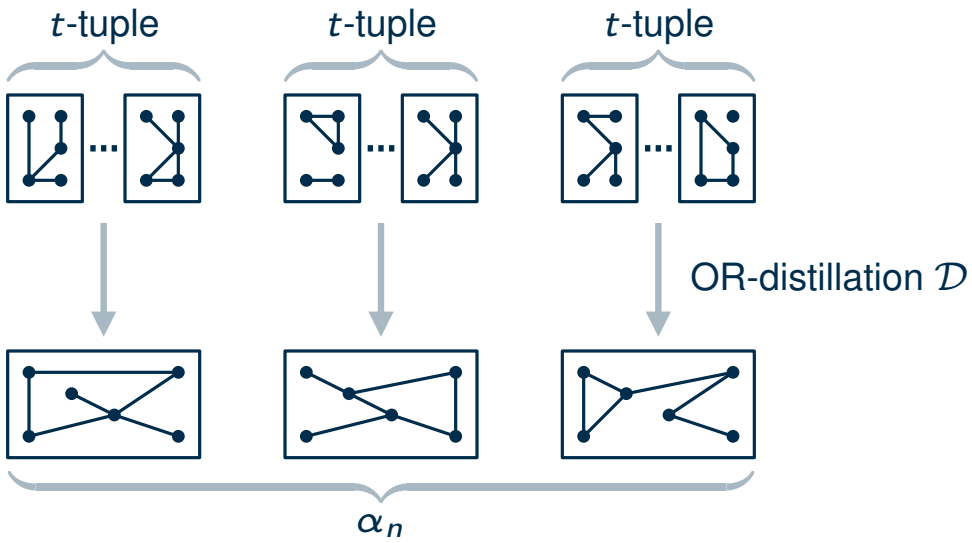
Wrapping up

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

Theorem
If LONGEST PATH has a polynomial kernel, then NP $\subseteq$ coNP/poly. (via: HAMILTONIAN PATH $\in$ coNP/poly)

Proof

- polynomial kernel for LONGP $\Rightarrow$ OR-distillation $\mathcal{D}$
- lemma applies $\rightarrow$ defines t and advice α_n



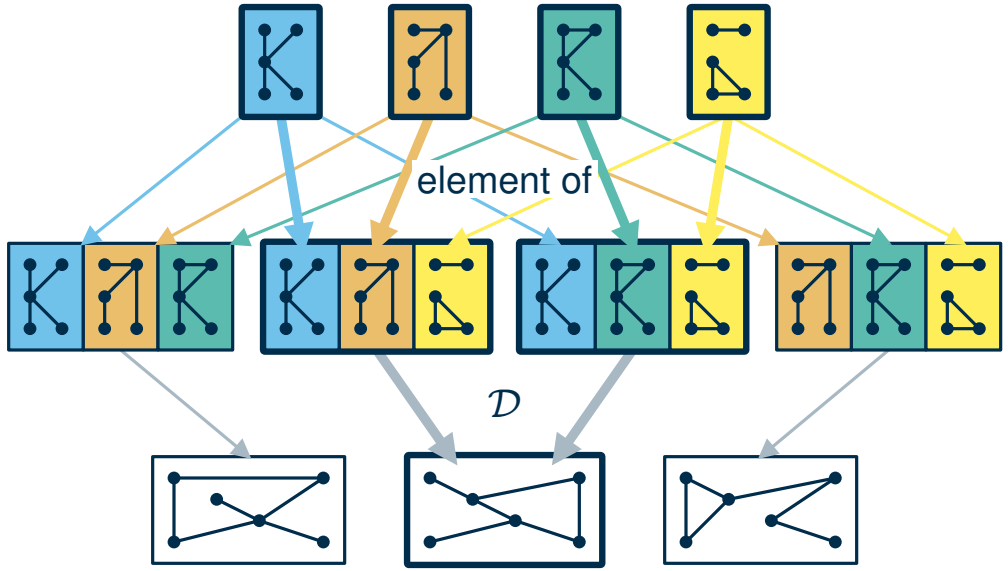
Wrapping up

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

Theorem
If LONGEST PATH has a polynomial kernel, then NP $\subseteq$ coNP/poly. (via: HAMILTONIAN PATH $\in$ coNP/poly)

Proof

- polynomial kernel for LONGP $\Rightarrow$ OR-distillation $\mathcal{D}$
- lemma applies $\rightarrow$ defines t and advice α_n



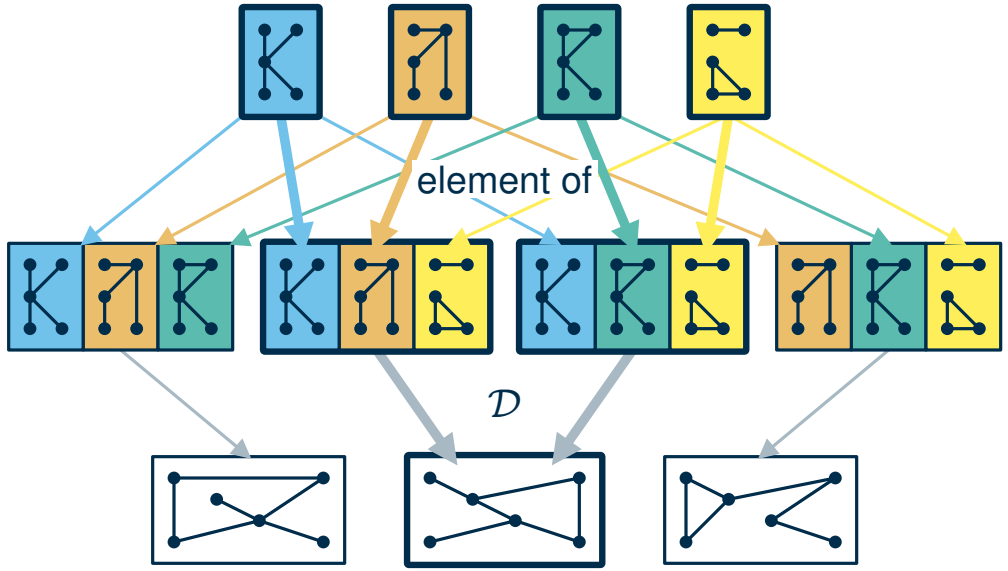
Wrapping up

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

Theorem
If LONGEST PATH has a polynomial kernel, then NP $\subseteq$ coNP/poly. (via: HAMILTONIAN PATH $\in$ coNP/poly)

Proof

- polynomial kernel for LONGP $\Rightarrow$ OR-distillation $\mathcal{D}$
- lemma applies $\rightarrow$ defines t and advice α_n
- coNP/poly-algo for HAMP on n-vertex graph G:
  - guess a t-tuple $(G_1, \ldots, G_t)$ containing G
  - run $\mathcal{D}$ on $(G_1, \ldots, G_t) \rightarrow$ core C
  - return **no** if $C \in \alpha_n$, and **yes** otherwise



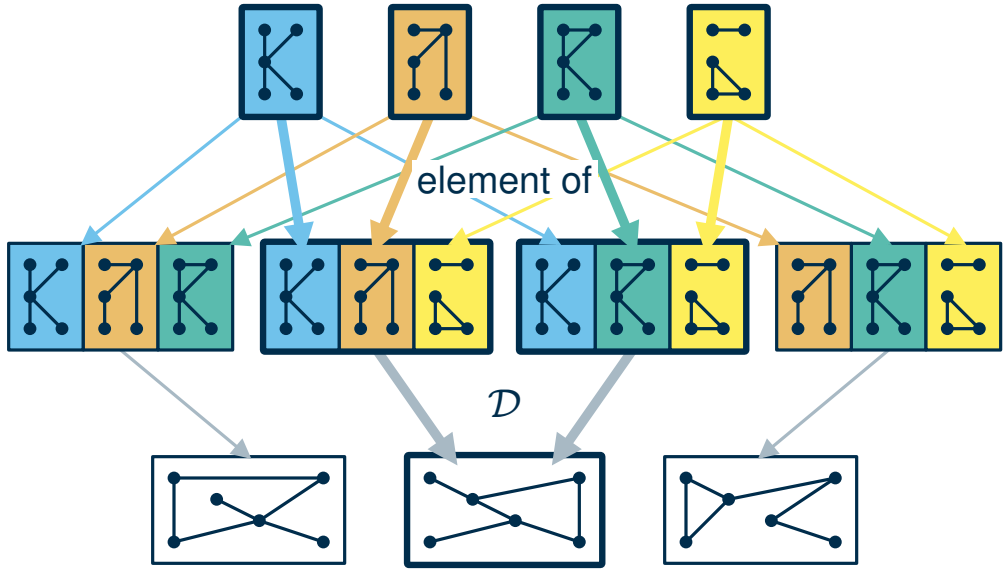
Wrapping up

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

Theorem
If LONGEST PATH has a polynomial kernel, then NP $\subseteq$ coNP/poly. (via: HAMILTONIAN PATH $\in$ coNP/poly)

Proof

- polynomial kernel for LONGP $\Rightarrow$ OR-distillation $\mathcal{D}$
- lemma applies $\rightarrow$ defines t and advice α_n (poly size)
- coNP/poly-algo for HAMP on n-vertex graph G:
  - guess a t-tuple $(G_1, \ldots, G_t)$ containing G (poly size)
  - run $\mathcal{D}$ on $(G_1, \ldots, G_t) \rightarrow$ core C (runs in poly time)
  - return **no** if $C \in \alpha_n$, and **yes** otherwise



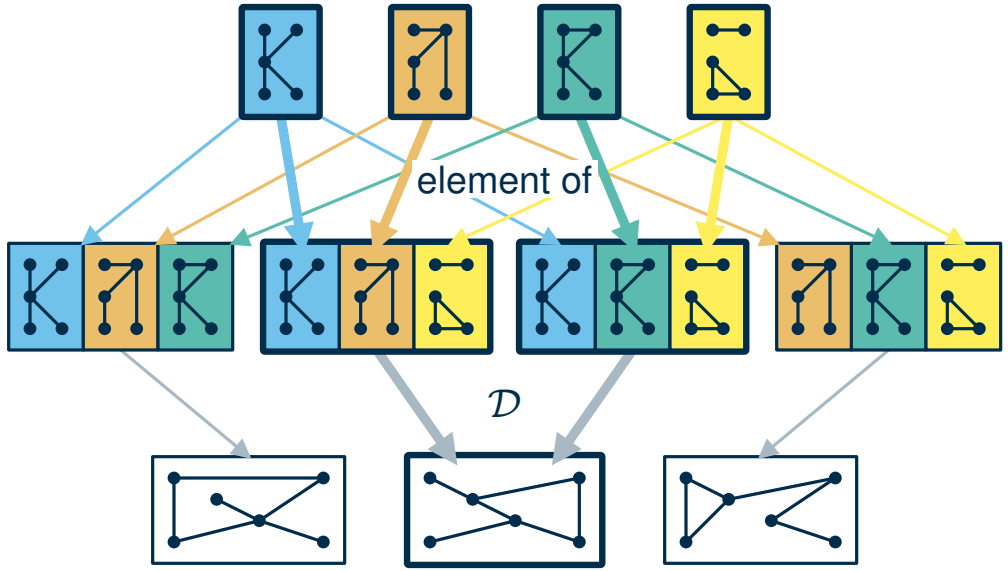
Wrapping up

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

Theorem
If LONGEST PATH has a polynomial kernel, then NP $\subseteq$ coNP/poly. (via: HAMILTONIAN PATH $\in$ coNP/poly)

Proof

- polynomial kernel for LONGP $\Rightarrow$ OR-distillation $\mathcal{D}$
- lemma applies $\rightarrow$ defines t and advice α_n (poly size)
- coNP/poly-algo for HAMP on n-vertex graph G:
  - guess a t-tuple $(G_1, \ldots, G_t)$ containing G (poly size)
  - run $\mathcal{D}$ on $(G_1, \ldots, G_t) \rightarrow$ core C (runs in poly time)
  - return **no** if $C \in \alpha_n$, and **yes** otherwise
  - if G is a yes-instance, we return **yes**



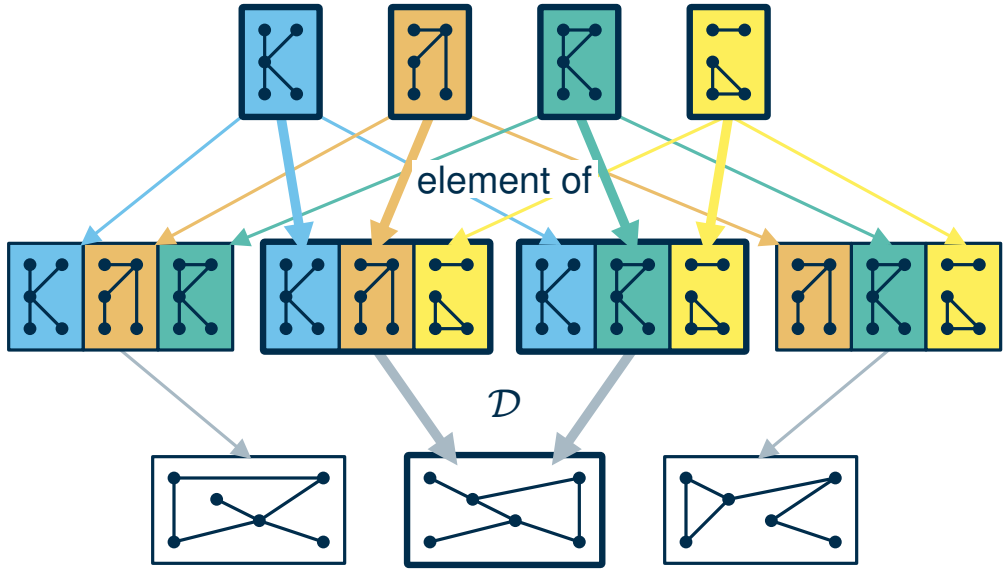
Wrapping up

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

Theorem
If LONGEST PATH has a polynomial kernel, then NP $\subseteq$ coNP/poly. (via: HAMILTONIAN PATH $\in$ coNP/poly)

Proof

- polynomial kernel for LONGP $\Rightarrow$ OR-distillation $\mathcal{D}$
- lemma applies $\rightarrow$ defines t and advice α_n (poly size)
- coNP/poly-algo for HAMP on n-vertex graph G:
  - guess a t-tuple $(G_1, \ldots, G_t)$ containing G (poly size)
  - run $\mathcal{D}$ on $(G_1, \ldots, G_t) \rightarrow$ core C (runs in poly time)
  - return **no** if $C \in \alpha_n$, and **yes** otherwise
  - if G is a yes-instance, we return **yes**
  - if G is a no-instance, we can guess a tuple that maps to a core in α_n and return **no**

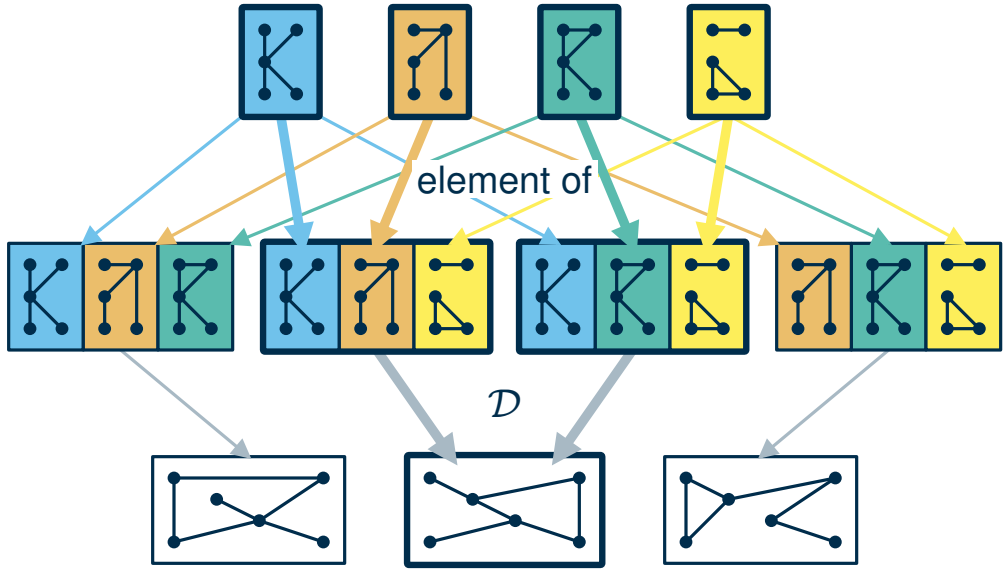


Wrapping up

Lemma
There is a set α_n of $n^{O(1)}$ cores and a $t \in n^{O(1)}$, such that every no-instance of HAMP has a t-tuple that is mapped to a core in α_n by $\mathcal{D}$.

Theorem
If LONGEST PATH has a polynomial kernel, then NP $\subseteq$ coNP/poly. (via: HAMILTONIAN PATH $\in$ coNP/poly)

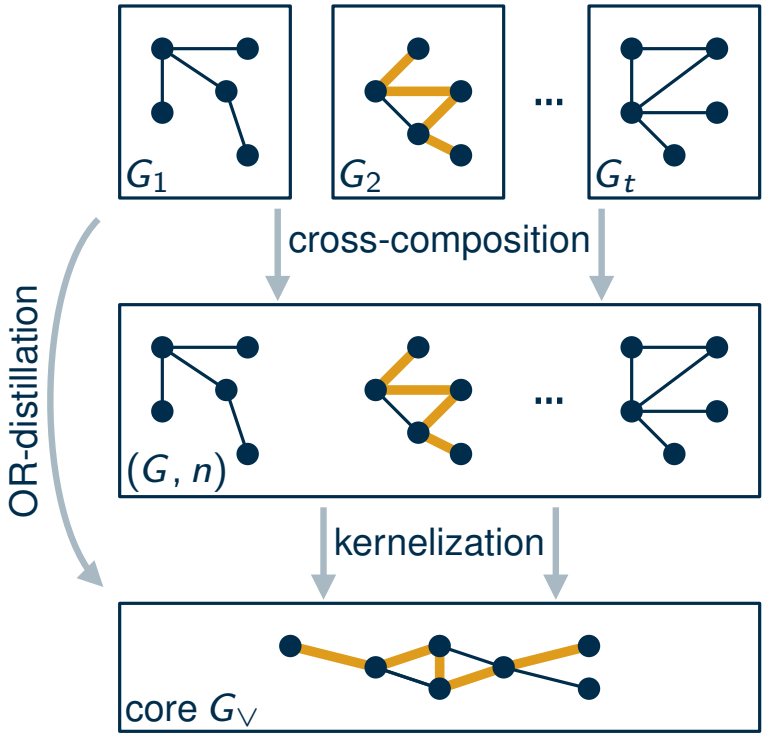
Proof



- polynomial kernel for LONGP $\Rightarrow$ OR-distillation $\mathcal{D}$
- lemma applies $\rightarrow$ defines t and advice α_n (poly size)
- coNP/poly-algo for HAMP on n-vertex graph G:
  - guess a t-tuple $(G_1, \ldots, G_t)$ containing G (poly size)
  - run $\mathcal{D}$ on $(G_1, \ldots, G_t) \rightarrow$ core C (runs in poly time)
  - return **no** if $C \in \alpha_n$, and **yes** otherwise
  - if G is a yes-instance, we return **yes**
  - if G is a no-instance, we can guess a tuple that maps to a core in α_n and return **no**
- from HAMILTONIAN PATH being NP-hard and HAMP $\in$ coNP/poly it follows NP $\subseteq$ coNP/poly

Framework

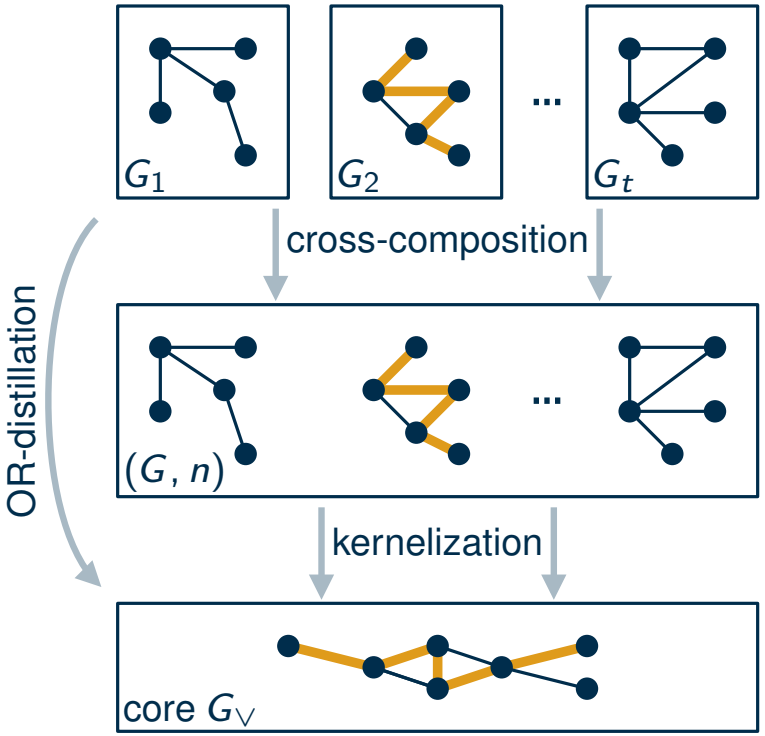
Our proof: polynomial kernel for LongP $\Rightarrow$ OR-distillation for HamP $\Rightarrow$ HamP $\in$ coNP/poly



Framework

specifically done for HAMP and LONGP — nothing special about HAMP here

Our proof: polynomial kernel for LONGP $\Rightarrow$ OR-distillation for HAMP $\Rightarrow$ HAMP $\in$ coNP/poly



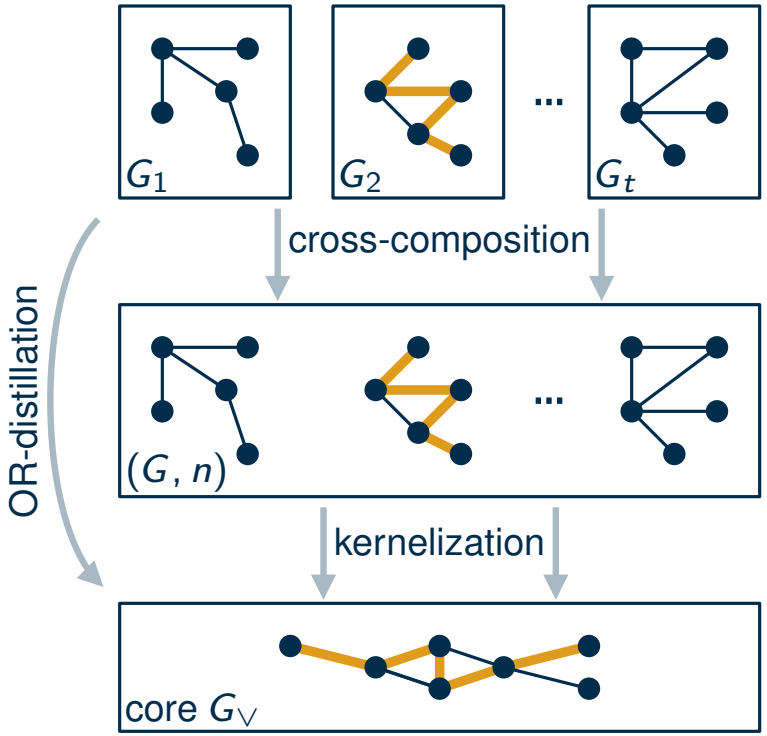
Framework

specifically done for HAMP and LONGP ↓ nothing special about HAMP here ↓

Our proof: polynomial kernel for LONGP $\Rightarrow$ OR-distillation for HAMP $\Rightarrow$ HAMP $\in$ coNP/poly

Definition
Let L and R be languages. An **OR-distillation** of L into R is an algorithm that takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum_{i\in[t]} |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max_{i\in[t]} |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i\in[t]} x_i \in L$.

Theorem
If there is an OR-distillation of L into R then $L \in$ coNP/poly.



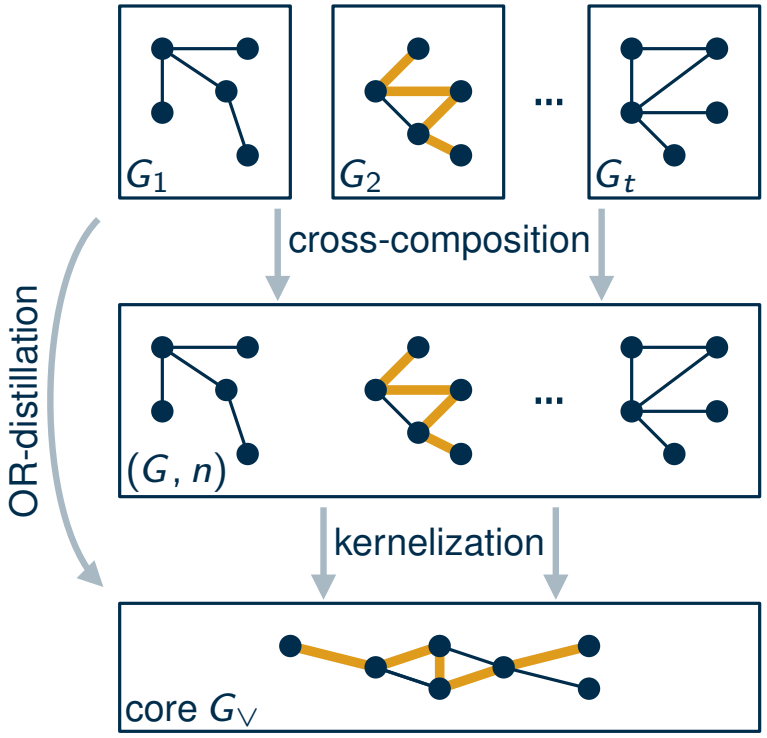
Framework

specifically done for HAMP and LONGP ↓ nothing special about HAMP here ↓

Our proof: polynomial kernel for LONGP $\Rightarrow$ OR-distillation for HAMP $\Rightarrow$ HAMP $\in$ coNP/poly

Definition
Let L and R be languages. An **OR-distillation** of L into R is an algorithm that takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum_{i\in[t]} |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max_{i\in[t]} |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i\in[t]} x_i \in L$.

Theorem
If there is an OR-distillation of L into R then $L \in$ coNP/poly.



Notes

- "similar" can mean different things (e.g., n vertices)

(formalizable via equivalence relation with polynomially many classes)

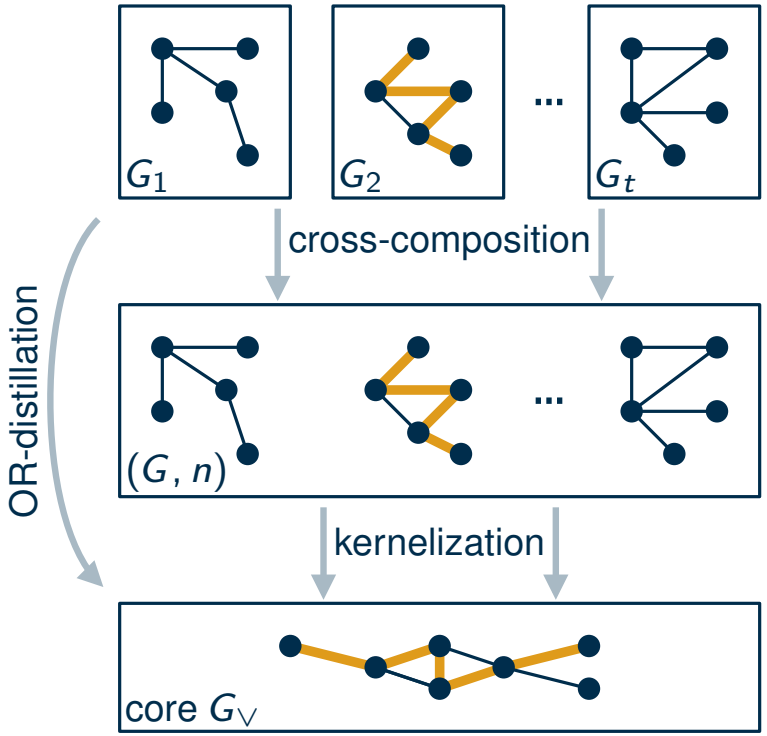
Framework

specifically done for HAMP and LONGP — nothing special about HAMP here

Our proof: polynomial kernel for LONGP $\Rightarrow$ OR-distillation for HAMP $\Rightarrow$ HAMP $\in$ coNP/poly

Definition
Let L and R be languages. An **OR-distillation** of L into R is an algorithm that takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum_{i\in[t]} |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max_{i\in[t]} |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i\in[t]} x_i \in L$.

Theorem
If there is an OR-distillation of L into R then $L \in$ coNP/poly.



Notes

- "similar" can mean different things (e.g., n vertices) (formalizable via equivalence relation with polynomially many classes)
- R does not even need to be decidable $\rightarrow$ essentially an information-theoretic result

Taking a break and splitting sets

Input: Hypergraph

- vertex set V consists of:
  - $v_0, \ldots, v_3$
  - x_0, x_1, y_0, y_1, z_0, z_1
- 11 hyperedges: see picture

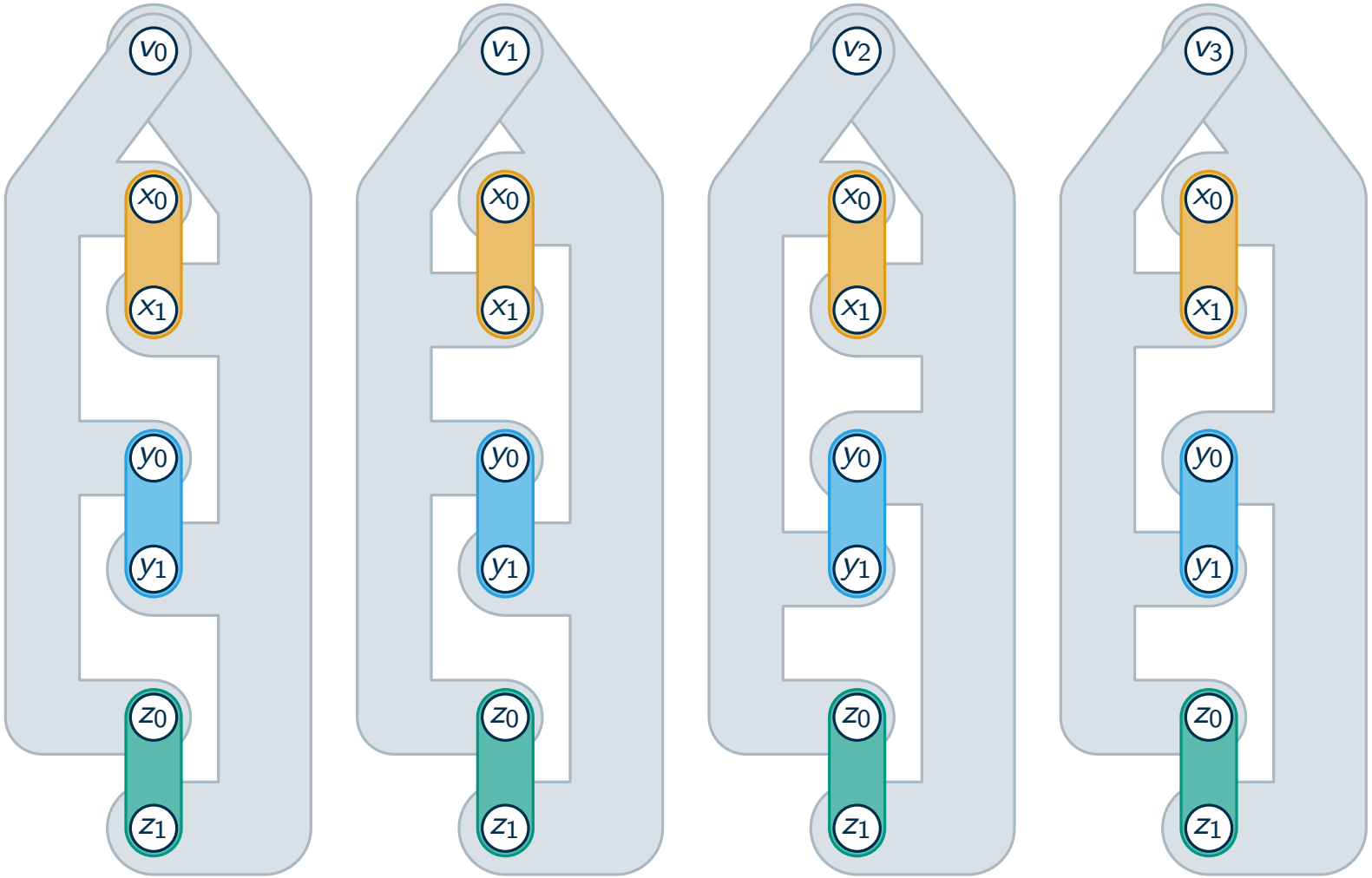
(x_i, y_i, z_i and their edges are shown multiple times)

SET SPLITTING

- find vertex set $X \subseteq V$ that splits **every** hyperedge

(every edge has a vertex from X and from $V \setminus X$)

Is this a yes-instance?



Taking a break and splitting sets

Input: Hypergraph

- vertex set V consists of:
  - $v_0, \ldots, v_3$
  - x_0, x_1, y_0, y_1, z_0, z_1
- 11 hyperedges: see picture

(x_i, y_i, z_i and their edges are shown multiple times)

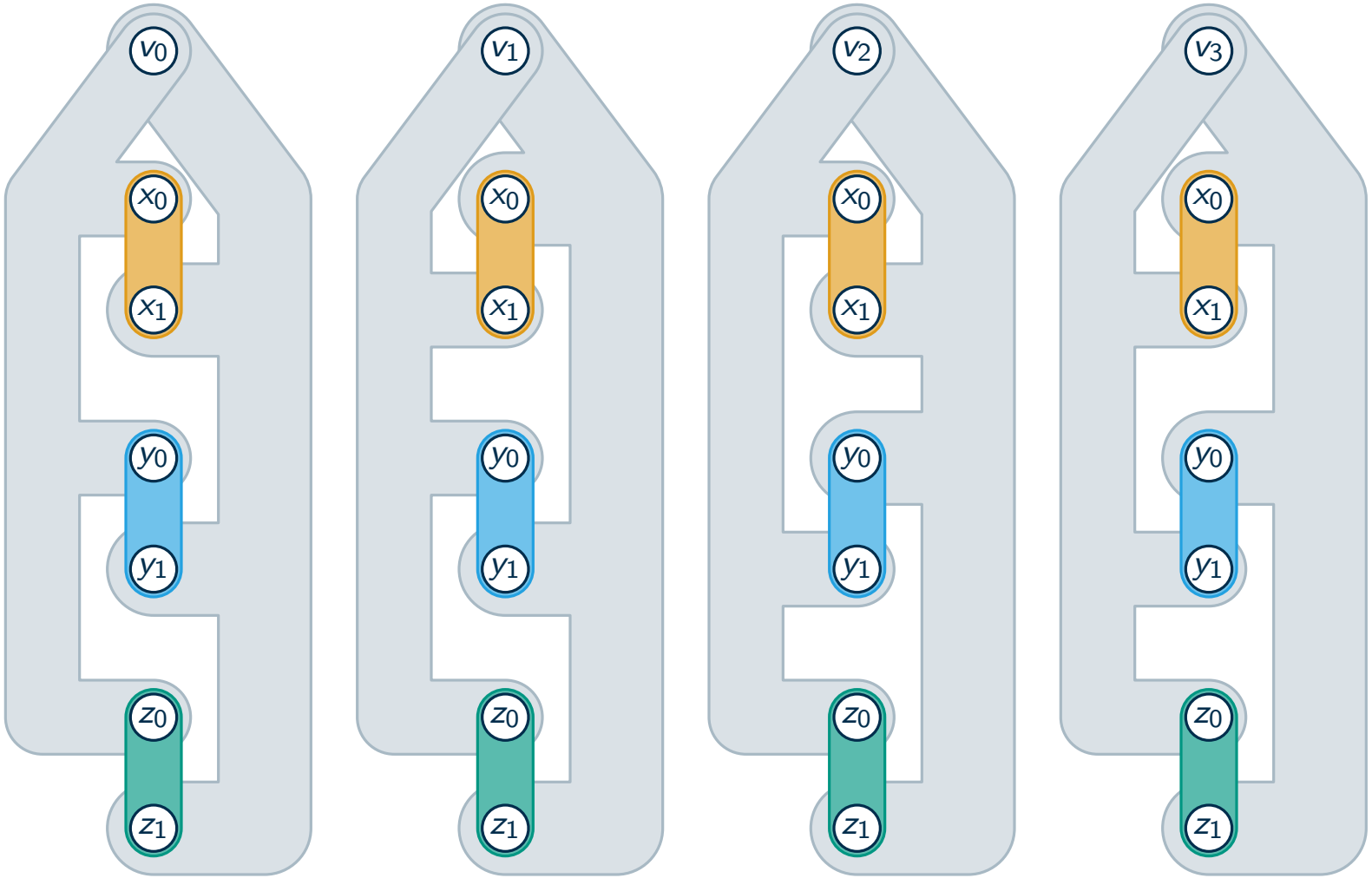
Set Splitting

- find vertex set $X \subseteq V$ that splits **every** hyperedge

(every edge has a vertex from X and from $V \setminus X$)

Selection Gadget

- choosing for the x, y, and z-edges "selects" one column
- no selection works



Taking a break and splitting sets

Input: Hypergraph

- vertex set V consists of:
  - $v_0, \ldots, v_3$
  - x_0, x_1, y_0, y_1, z_0, z_1
- 11 hyperedges: see picture

(x_i, y_i, z_i and their edges are shown multiple times)

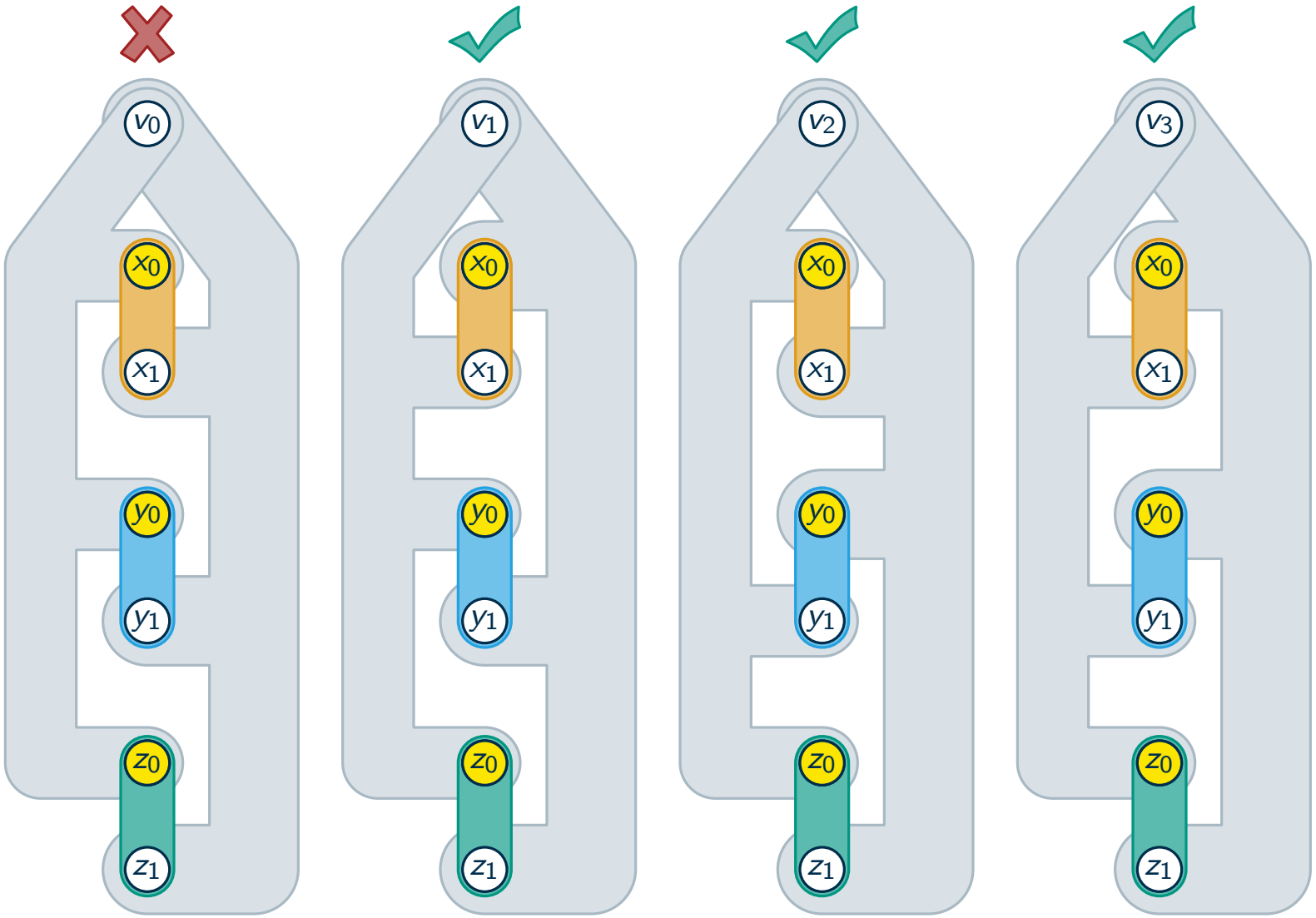
Set Splitting

- find vertex set $X \subseteq V$ that splits **every** hyperedge

(every edge has a vertex from X and from $V \setminus X$)

Selection Gadget

- choosing for the x, y, and z-edges "selects" one column
- no selection works



Taking a break and splitting sets

Input: Hypergraph

- vertex set V consists of:
  - $v_0, \ldots, v_3$
  - x_0, x_1, y_0, y_1, z_0, z_1
- 11 hyperedges: see picture

(x_i, y_i, z_i and their edges are shown multiple times)

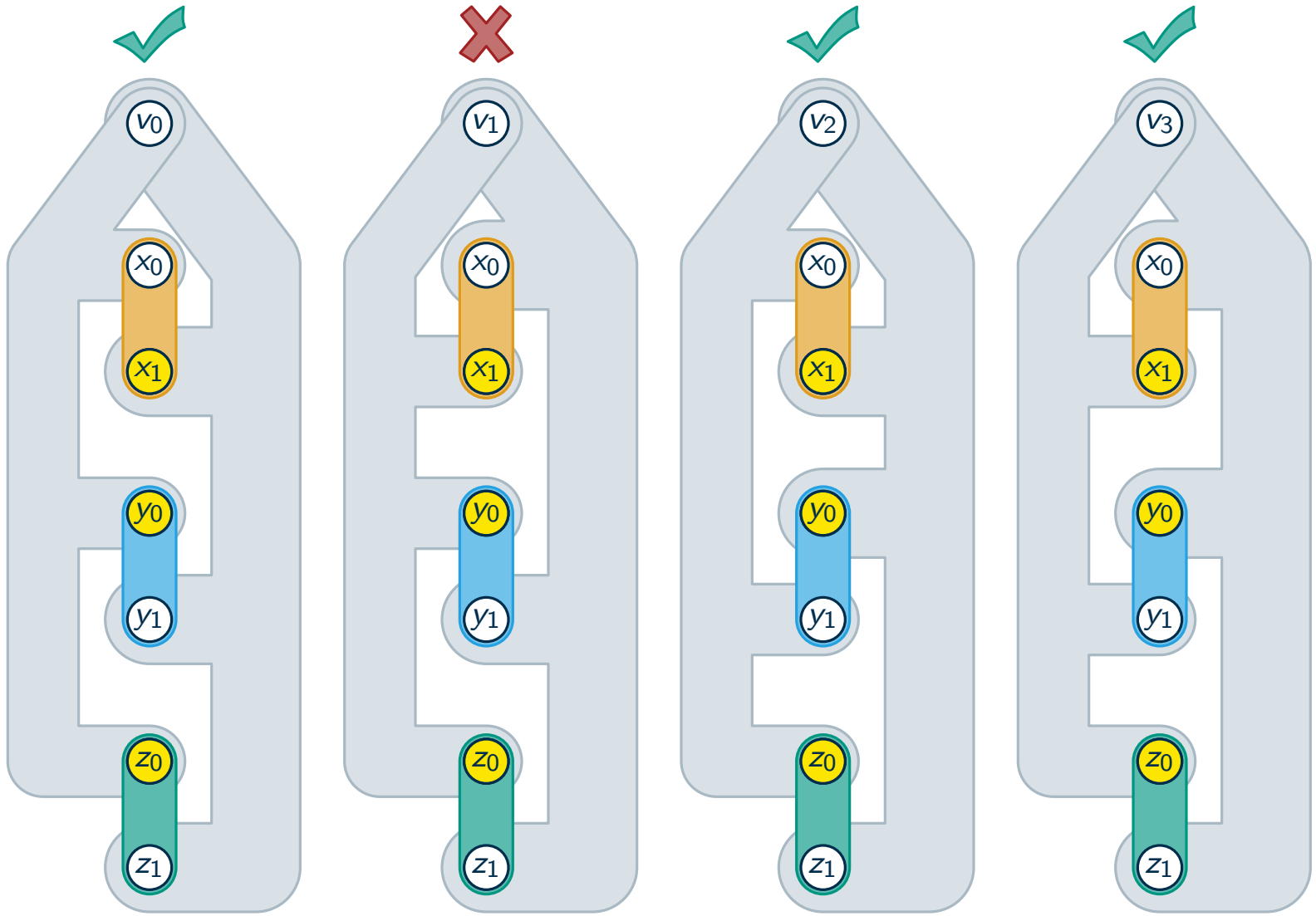
Set Splitting

- find vertex set $X \subseteq V$ that splits **every** hyperedge

(every edge has a vertex from X and from $V \setminus X$)

Selection Gadget

- choosing for the x, y, and z-edges "selects" one column
- no selection works



Taking a break and splitting sets

Input: Hypergraph

- vertex set V consists of:
  - $v_0, \ldots, v_3$
  - x_0, x_1, y_0, y_1, z_0, z_1
- 11 hyperedges: see picture

(x_i, y_i, z_i and their edges are shown multiple times)

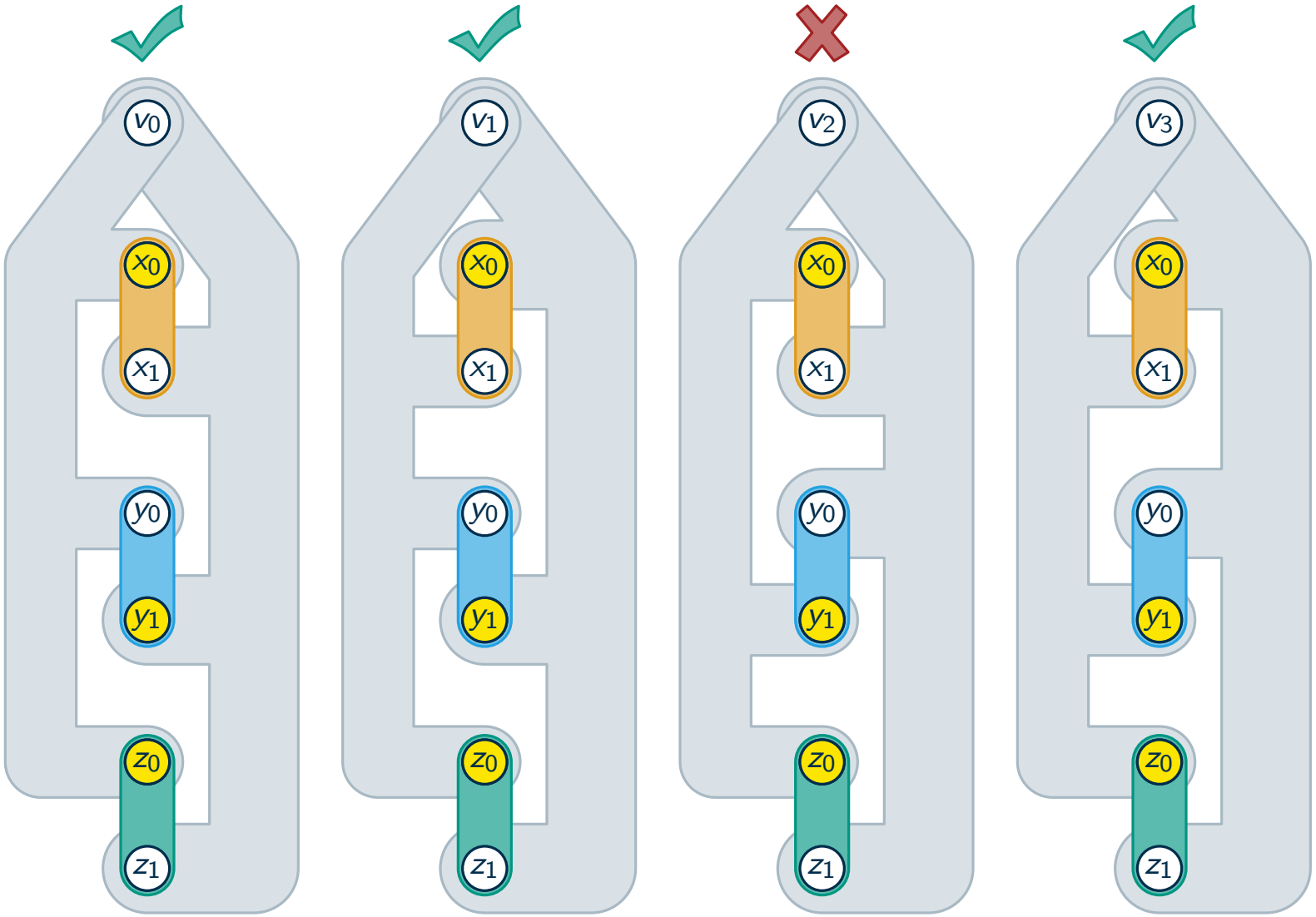
Set Splitting

- find vertex set $X \subseteq V$ that splits **every** hyperedge

(every edge has a vertex from X and from $V \setminus X$)

Selection Gadget

- choosing for the x, y, and z-edges "selects" one column
- no selection works



Taking a break and splitting sets

Input: Hypergraph

- vertex set V consists of:
  - $v_0, \ldots, v_3$
  - x_0, x_1, y_0, y_1, z_0, z_1
- 11 hyperedges: see picture

(x_i, y_i, z_i and their edges are shown multiple times)

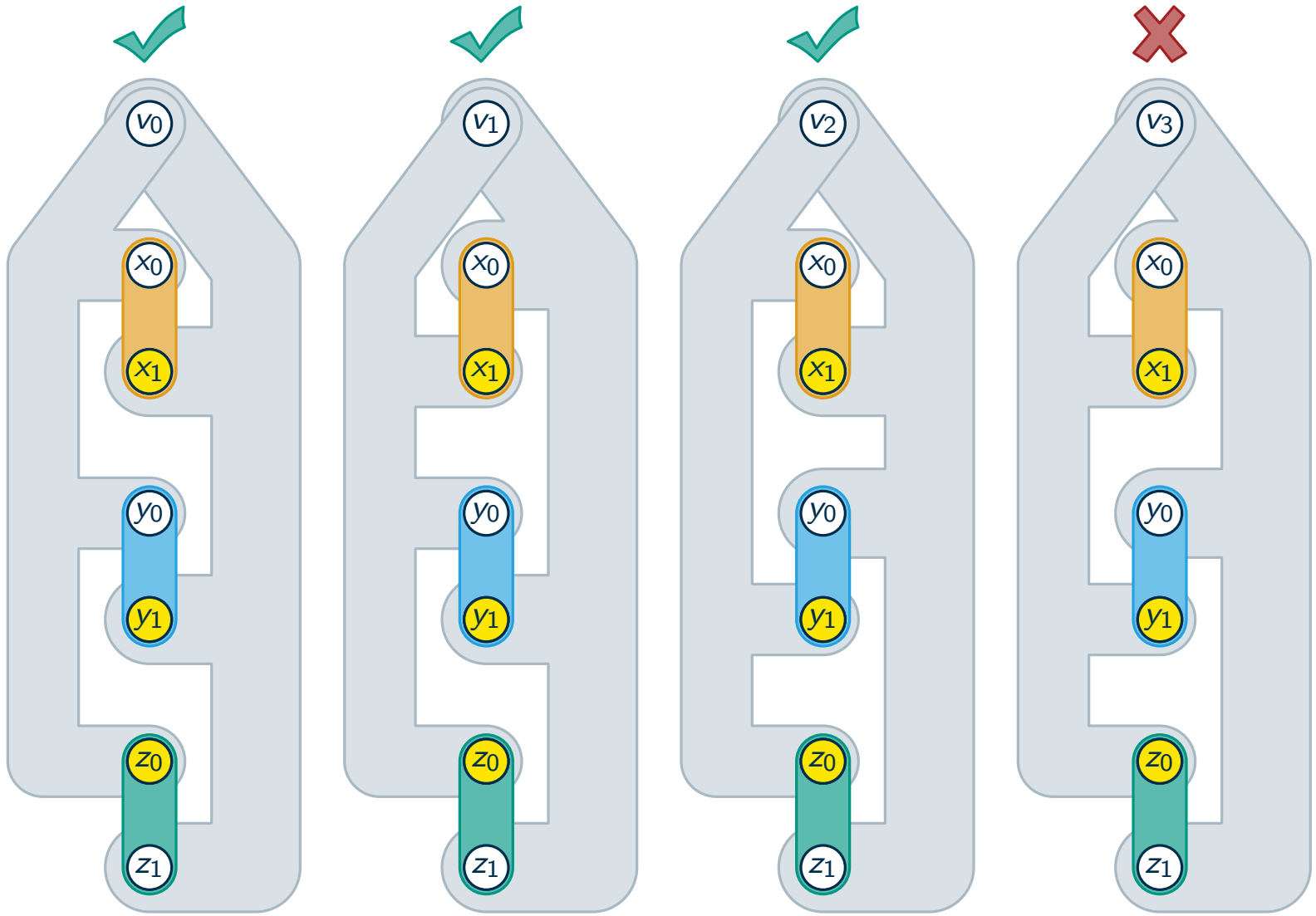
Set Splitting

- find vertex set $X \subseteq V$ that splits **every** hyperedge

(every edge has a vertex from X and from $V \setminus X$)

Selection Gadget

- choosing for the x, y, and z-edges "selects" one column
- no selection works



SET SPLITTING has no polynomial kernel

Problem: SET SPLITTING
Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$. Is there a set $X \subseteq V$ that splits all hyperedges of $\mathcal{H}$?

Goal: show that SET SPLITTING parameterized by $n = |V|$ has no polynomial kernel (unless...)

Set Splitting has no polynomial kernel

Problem: Set Splitting
Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$. Is there a set $X \subseteq V$ that splits all hyperedges of $\mathcal{H}$?

Goal: show that Set Splitting parameterized by $n = |V|$ has no polynomial kernel (unless...)

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

Plan

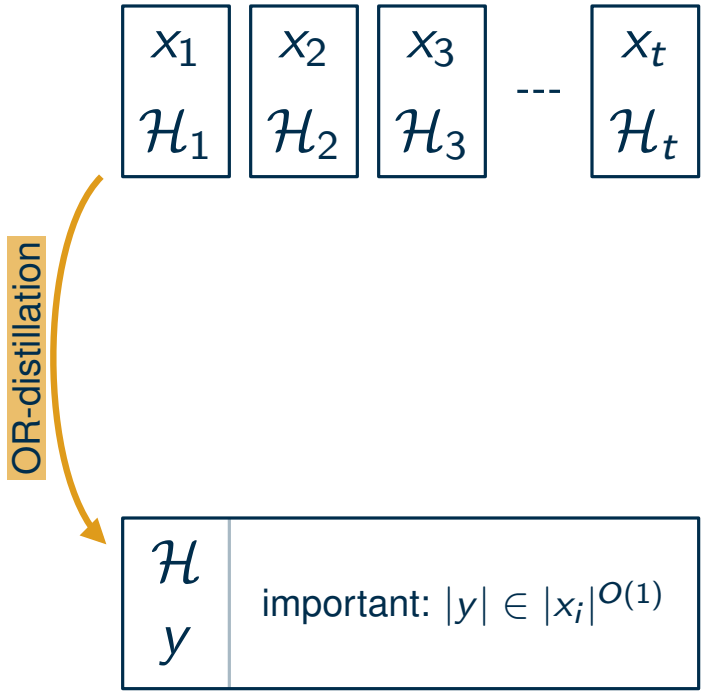
- assume there is a kernelization algorithm producing a polynomial kernel
- show that this implies an OR-distillation from Set Splitting into itself
- as Set Splitting is NP-hard, this would imply NP $\subseteq$ coNP/poly

OR-Distillation of Set Splitting into itself

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

Given

- t instances of Set Splitting $\mathcal{H}_1, \ldots, \mathcal{H}_t$, each on n vertices



Todo

- create instance $\mathcal{H}$ of size $\max |x_i|^{O(1)}$ that "or"s $\mathcal{H}_1, \ldots, \mathcal{H}_t$

OR-Distillation of Set Splitting into itself

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

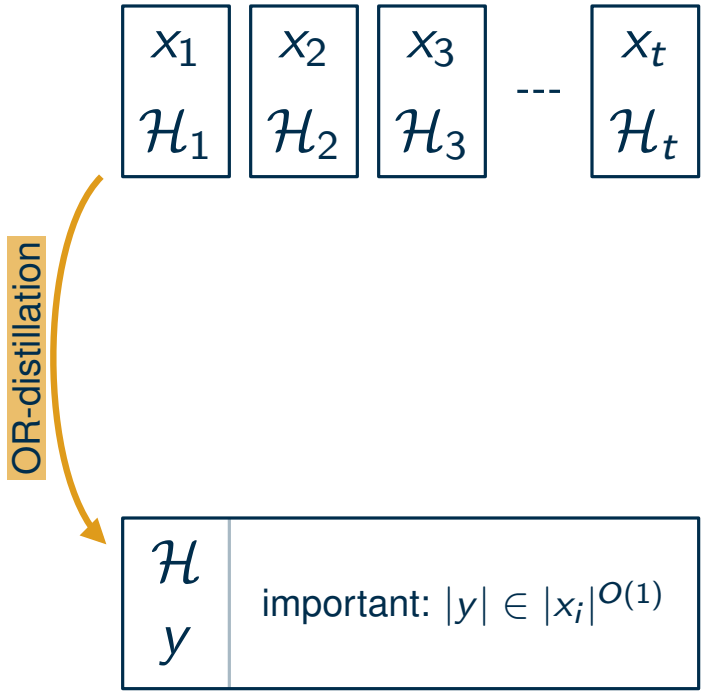
Given

- t instances of Set Splitting $\mathcal{H}_1, \ldots, \mathcal{H}_t$, each on n vertices

includes vertices and edges
#edges can be exponential in #vertices

Todo

- create instance $\mathcal{H}$ of size $\max |x_i|^{O(1)}$ that "or"s $\mathcal{H}_1, \ldots, \mathcal{H}_t$



OR-Distillation of Set Splitting into itself

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

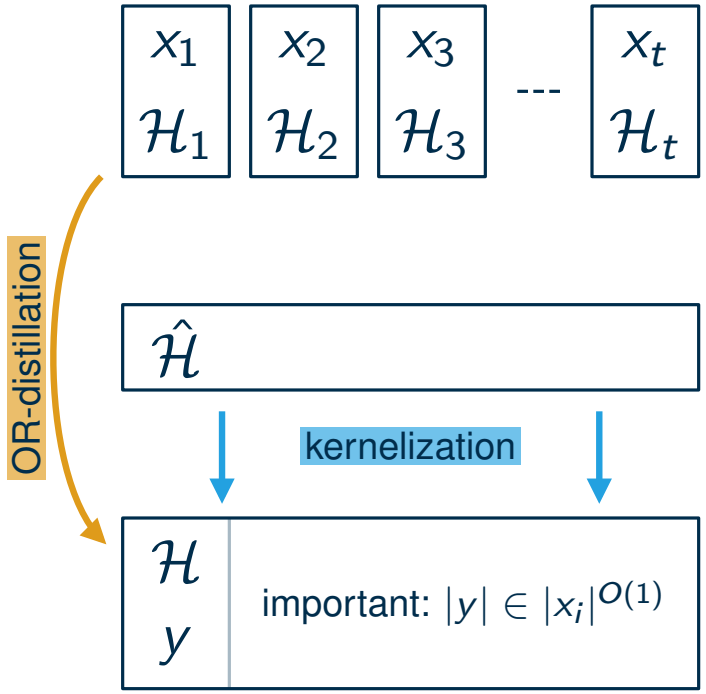
Given

- t instances of Set Splitting $\mathcal{H}_1, \ldots, \mathcal{H}_t$, each on n vertices
- kernelization algorithm: reduces instance $\hat{\mathcal{H}}$ with k vertices to instance of size $k^{O(1)}$

includes vertices and edges
#edges can be exponential in #vertices

Todo

- create instance $\mathcal{H}$ of size $\max |x_i|^{O(1)}$ that "or"s $\mathcal{H}_1, \ldots, \mathcal{H}_t$



OR-Distillation of Set Splitting into itself

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

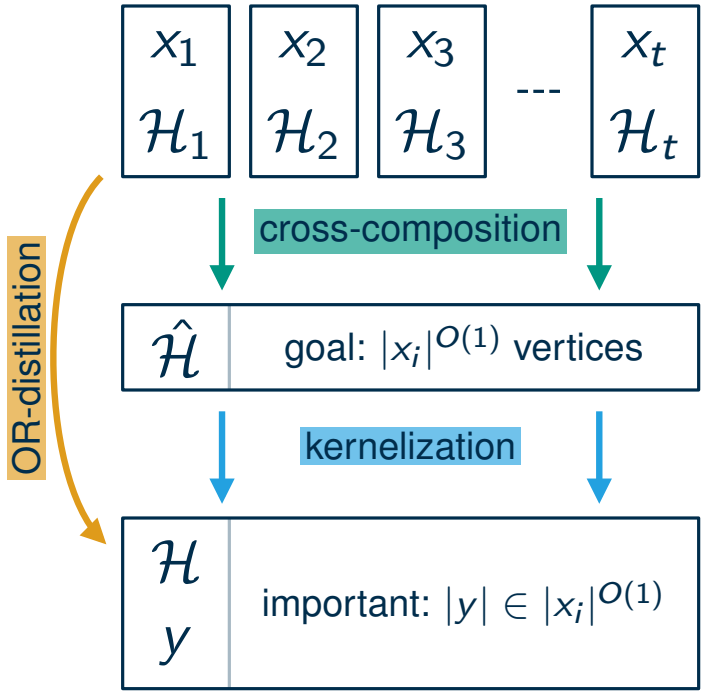
Given

- t instances of Set Splitting $\mathcal{H}_1, \ldots, \mathcal{H}_t$, each on n vertices
- kernelization algorithm: reduces instance $\hat{\mathcal{H}}$ with k vertices to instance of size $k^{O(1)}$

Todo

includes vertices and edges
#edges can be exponential in #vertices

- create instance $\mathcal{H}$ of size $\max |x_i|^{O(1)}$ that "or"s $\mathcal{H}_1, \ldots, \mathcal{H}_t$
- create instance $\hat{\mathcal{H}}$ with $\max |x_i|^{O(1)}$ vertices that "or"s $\mathcal{H}_1, \ldots, \mathcal{H}_t$



OR-Distillation of Set Splitting into itself

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

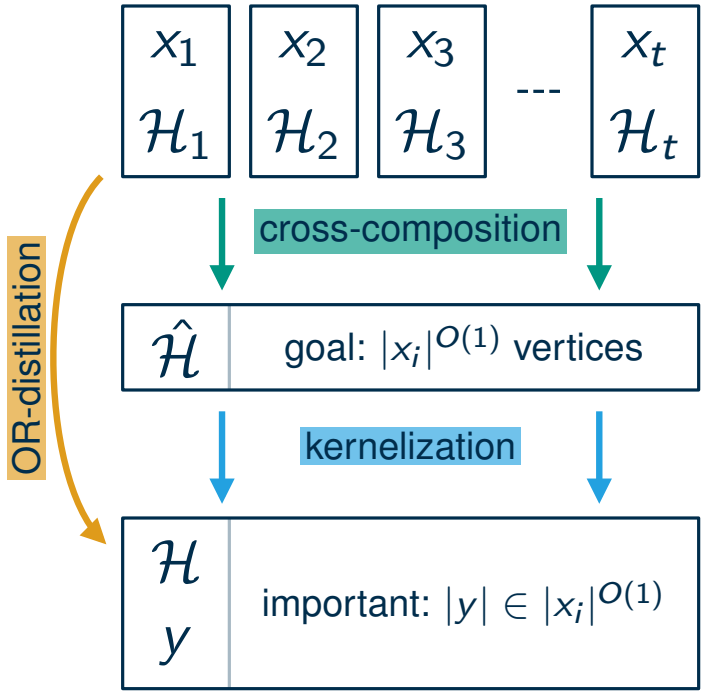
Given

- t instances of Set Splitting $\mathcal{H}_1, \ldots, \mathcal{H}_t$, each on n vertices
- kernelization algorithm: reduces instance $\hat{\mathcal{H}}$ with k vertices to instance of size $k^{O(1)}$

Todo

includes vertices and edges
#edges can be exponential in #vertices

- create instance $\mathcal{H}$ of size $\max |x_i|^{O(1)}$ that "or"s $\mathcal{H}_1, \ldots, \mathcal{H}_t$
- create instance $\hat{\mathcal{H}}$ with $\max |x_i|^{O(1)}$ vertices that "or"s $\mathcal{H}_1, \ldots, \mathcal{H}_t$
- important: number of vertices of $\hat{\mathcal{H}}$ must not depend on t



Cross-Composition

Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the “or” of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

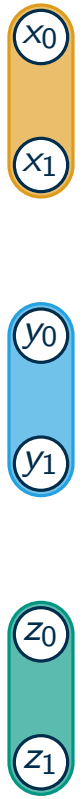
Cross-Composition

Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$



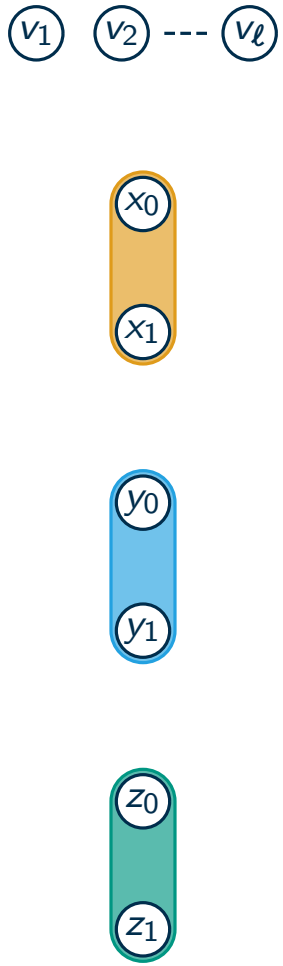
Cross-Composition

Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$



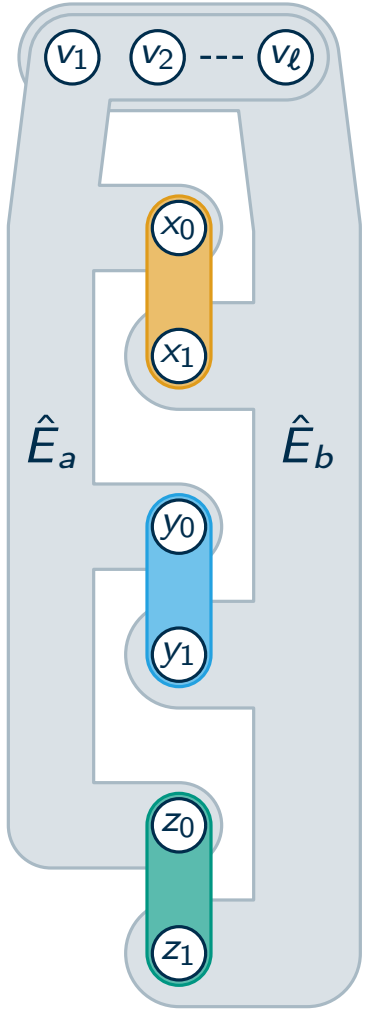
Cross-Composition

Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$
- create edges $\hat{E}_a = E \cup \{x_0, y_0, z_0\}$ and $\hat{E}_b = E \cup \{x_1, y_1, z_1\}$



Cross-Composition

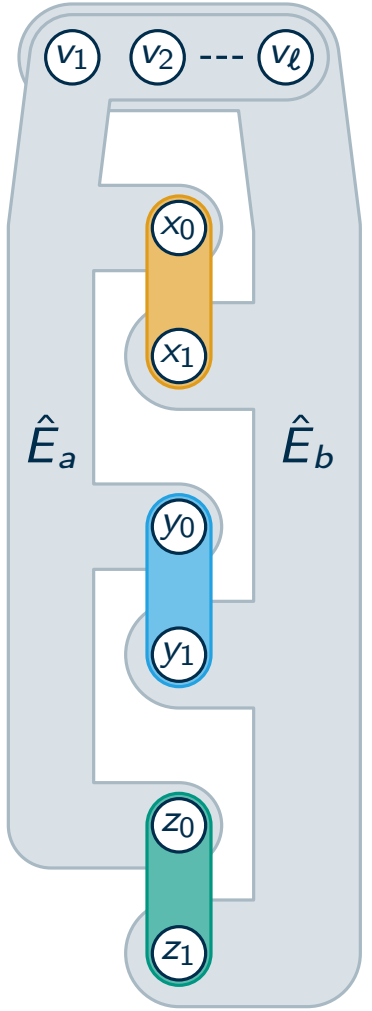
Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$
- create edges $\hat{E}_a = E \cup \{x_0, y_0, z_0\}$ and $\hat{E}_b = E \cup \{x_1, y_1, z_1\}$
- for edges in $\mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$: use $\{x_1, y_0, z_0\}, \{x_0, y_1, z_0\}, \{x_1, y_1, z_0\}$ for $\hat{E}_a$

(and their complements for $\hat{E}_b$)



Cross-Composition

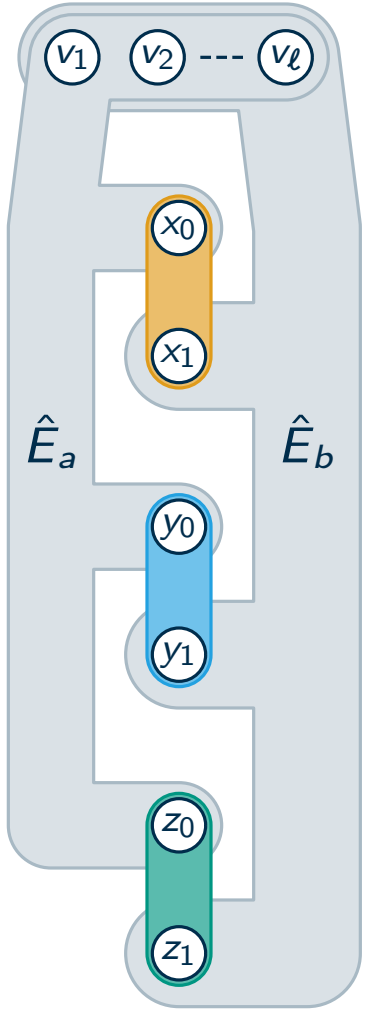
Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$
- create edges $\hat{E}_a = E \cup \{x_0, y_0, z_0\}$ and $\hat{E}_b = E \cup \{x_1, y_1, z_1\}$
- for edges in $\mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$: use $\{x_1, y_0, z_0\}, \{x_0, y_1, z_0\}, \{x_1, y_1, z_0\}$ for $\hat{E}_a$

(and their complements for $\hat{E}_b$)



Observation for a solution candidate $X \subseteq \hat{V}$

Cross-Composition

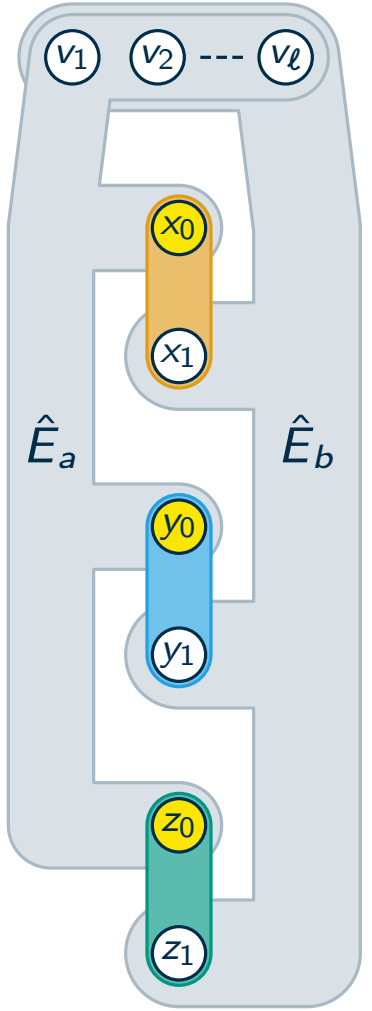
Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$
- create edges $\hat{E}_a = E \cup \{x_0, y_0, z_0\}$ and $\hat{E}_b = E \cup \{x_1, y_1, z_1\}$
- for edges in $\mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$: use $\{x_1, y_0, z_0\}, \{x_0, y_1, z_0\}, \{x_1, y_1, z_0\}$ for $\hat{E}_a$

(and their complements for $\hat{E}_b$)



Observation for a solution candidate $X \subseteq \hat{V}$

- X contains exactly one of x_i, one y_i, and one z_i

Cross-Composition

Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

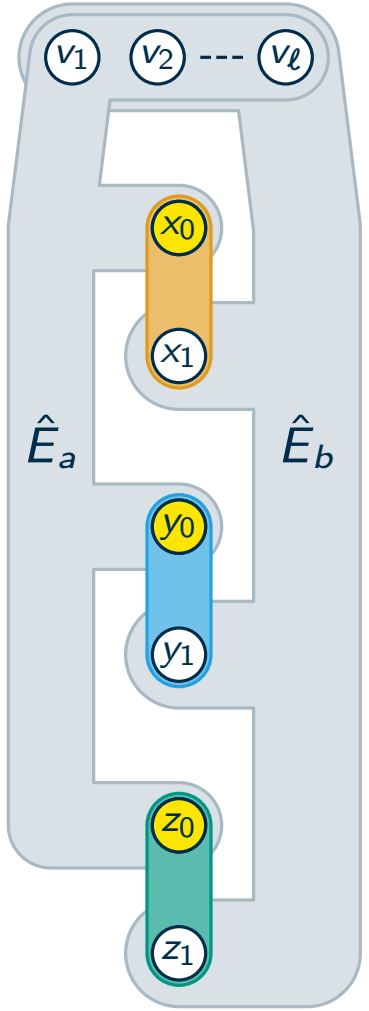
- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$
- create edges $\hat{E}_a = E \cup \{x_0, y_0, z_0\}$ and $\hat{E}_b = E \cup \{x_1, y_1, z_1\}$
- for edges in $\mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$: use $\{x_1, y_0, z_0\}, \{x_0, y_1, z_0\}, \{x_1, y_1, z_0\}$ for $\hat{E}_a$

(and their complements for $\hat{E}_b$)

Observation for a solution candidate $X \subseteq \hat{V}$

- X contains exactly one of x_i, one y_i, and one z_i
- say $\mathcal{H}_1$ is **selected** if $x_0, y_0, z_0 \in X$ or $x_1, y_1, z_1 \in X$

} exactly one of $\mathcal{H}_1 \ldots \mathcal{H}_4$ is selected



Cross-Composition

Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

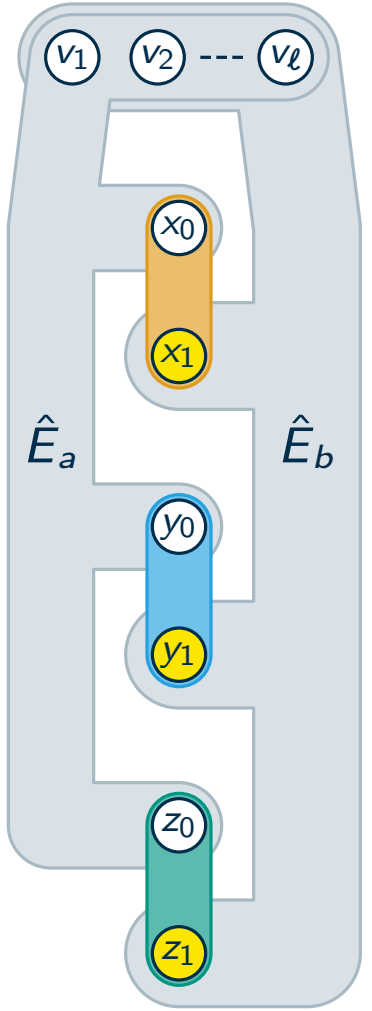
- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$
- create edges $\hat{E}_a = E \cup \{x_0, y_0, z_0\}$ and $\hat{E}_b = E \cup \{x_1, y_1, z_1\}$
- for edges in $\mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$: use $\{x_1, y_0, z_0\}, \{x_0, y_1, z_0\}, \{x_1, y_1, z_0\}$ for $\hat{E}_a$

(and their complements for $\hat{E}_b$)

Observation for a solution candidate $X \subseteq \hat{V}$

- X contains exactly one of x_i, one y_i, and one z_i
- say $\mathcal{H}_1$ is **selected** if $x_0, y_0, z_0 \in X$ or $x_1, y_1, z_1 \in X$

} exactly one of $\mathcal{H}_1 \ldots \mathcal{H}_4$ is selected



Cross-Composition

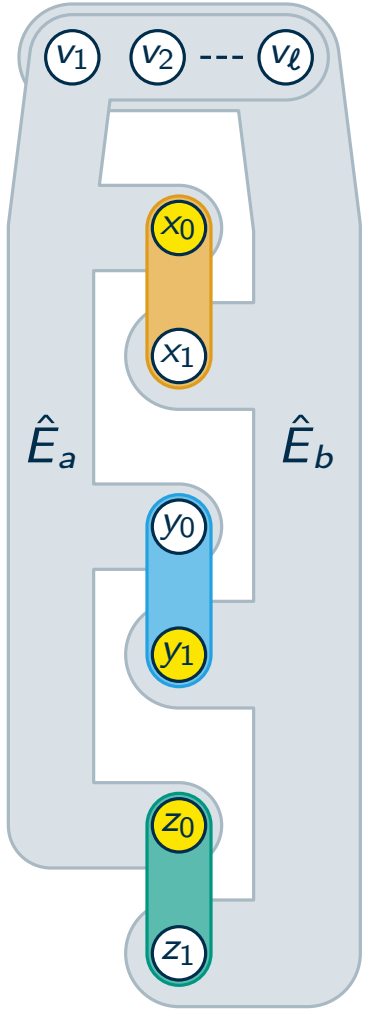
Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$
- create edges $\hat{E}_a = E \cup \{x_0, y_0, z_0\}$ and $\hat{E}_b = E \cup \{x_1, y_1, z_1\}$
- for edges in $\mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$: use $\{x_1, y_0, z_0\}, \{x_0, y_1, z_0\}, \{x_1, y_1, z_0\}$ for $\hat{E}_a$

(and their complements for $\hat{E}_b$)



Observation for a solution candidate $X \subseteq \hat{V}$

- X contains exactly one of x_i, one y_i, and one z_i
- say $\mathcal{H}_1$ is **selected** if $x_0, y_0, z_0 \in X$ or $x_1, y_1, z_1 \in X$

} exactly one of $\mathcal{H}_1 \ldots \mathcal{H}_4$ is selected

- X splits $\hat{E}_1$ and $\hat{E}_2 \quad \Leftrightarrow \quad \mathcal{H}_1$ not selected or X splits E

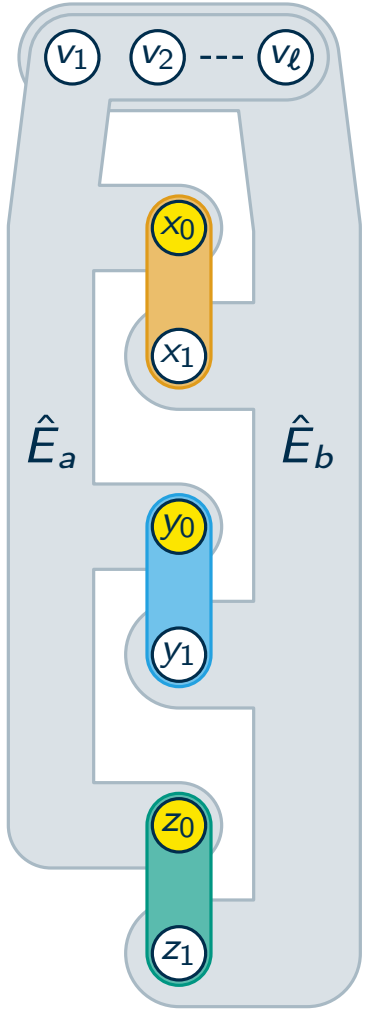
Cross-Composition

Goal (reminder):
- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$
- create edges $\hat{E}_a = E \cup \{x_0, y_0, z_0\}$ and $\hat{E}_b = E \cup \{x_1, y_1, z_1\}$
- for edges in $\mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$: use $\{x_1, y_0, z_0\}, \{x_0, y_1, z_0\}, \{x_1, y_1, z_0\}$ for $\hat{E}_a$

(and their complements for $\hat{E}_b$)



Observation for a solution candidate $X \subseteq \hat{V}$

- X contains exactly one of x_i, one y_i, and one z_i
- say $\mathcal{H}_1$ is **selected** if $x_0, y_0, z_0 \in X$ or $x_1, y_1, z_1 \in X$

} exactly one of $\mathcal{H}_1 \ldots \mathcal{H}_4$ is selected

- X splits $\hat{E}_1$ and $\hat{E}_2 \quad \Leftrightarrow \quad \mathcal{H}_1$ not selected or X splits E

Cross-Composition

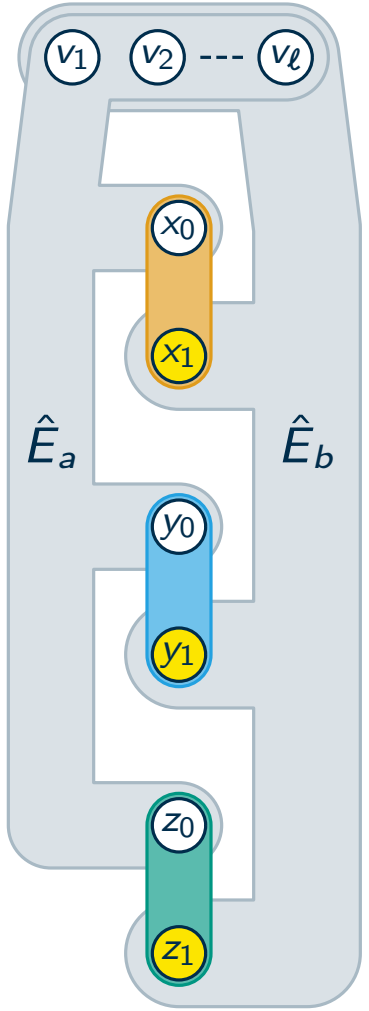
Goal (reminder):

- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$
- create edges $\hat{E}_a = E \cup \{x_0, y_0, z_0\}$ and $\hat{E}_b = E \cup \{x_1, y_1, z_1\}$
- for edges in $\mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$: use $\{x_1, y_0, z_0\}, \{x_0, y_1, z_0\}, \{x_1, y_1, z_0\}$ for $\hat{E}_a$

(and their complements for $\hat{E}_b$)



Observation for a solution candidate $X \subseteq \hat{V}$

- X contains exactly one of x_i, one y_i, and one z_i
- say $\mathcal{H}_1$ is **selected** if $x_0, y_0, z_0 \in X$ or $x_1, y_1, z_1 \in X$

} exactly one of $\mathcal{H}_1 \ldots \mathcal{H}_4$ is selected

- X splits $\hat{E}_1$ and $\hat{E}_2 \quad \Leftrightarrow \quad \mathcal{H}_1$ not selected or X splits E

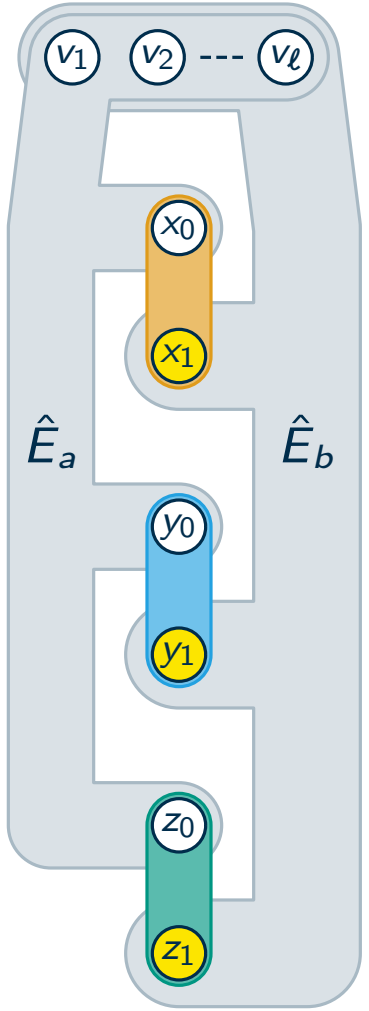
Cross-Composition

Goal (reminder):
- given $\{\mathcal{H}_1, \ldots, \mathcal{H}_t\}$ (n vertices each)
- build $\hat{\mathcal{H}}$ with $n^{O(1)}$ vertices
- $\hat{\mathcal{H}}$ is the "or" of $\mathcal{H}_1, \ldots, \mathcal{H}_t$

Construction with selector gadget for $t = 4$

- let $V = \{v_1, \ldots, v_n\}$ for all $\mathcal{H}_i \to \hat{V} = V \cup \{x_0, x_1, y_0, y_1, z_0, z_1\}$
- in $\hat{\mathcal{H}}$, create edges $\{x_0, x_1\}$, $\{y_0, y_1\}$, $\{z_0, z_1\}$
- consider a single edge $E = \{v_1, \ldots, v_\ell\} \in \mathcal{H}_1$
- create edges $\hat{E}_a = E \cup \{x_0, y_0, z_0\}$ and $\hat{E}_b = E \cup \{x_1, y_1, z_1\}$
- for edges in $\mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4$: use $\{x_1, y_0, z_0\}, \{x_0, y_1, z_0\}, \{x_1, y_1, z_0\}$ for $\hat{E}_a$

(and their complements for $\hat{E}_b$)



Observation for a solution candidate $X \subseteq \hat{V}$

- X contains exactly one of x_i, one y_i, and one z_i
- say $\mathcal{H}_1$ is **selected** if $x_0, y_0, z_0 \in X$ or $x_1, y_1, z_1 \in X$

} exactly one of $\mathcal{H}_1 \ldots \mathcal{H}_4$ is selected

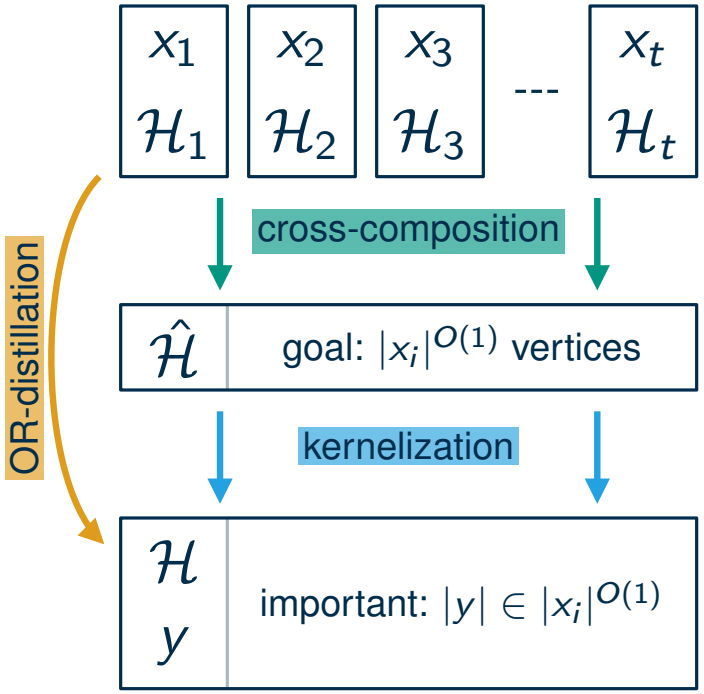
- X splits $\hat{E}_1$ and $\hat{E}_2 \quad \Leftrightarrow \quad \mathcal{H}_1$ not selected or X splits E
- $\hat{\mathcal{H}}$ has a solution $\quad \Leftrightarrow \quad$ one of $\mathcal{H}_1, \ldots, \mathcal{H}_4$ has a solution

Concluding remarks on Set Splitting

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

The cross-composition we have seen

- creates $\hat{\mathcal{H}}$ in time $(\sum |\mathcal{H}_i|)^{O(1)}$ form $\mathcal{H}_1, \ldots, \mathcal{H}_t$ (n vertices each)

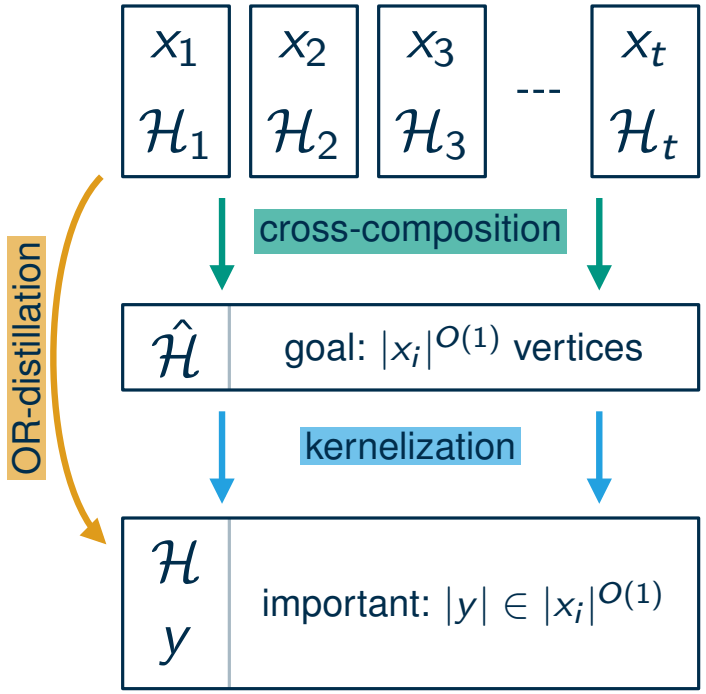


Concluding remarks on SET SPLITTING

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

The cross-composition we have seen

- creates $\hat{\mathcal{H}}$ in time $(\sum |\mathcal{H}_i|)^{O(1)}$ form $\mathcal{H}_1, \ldots, \mathcal{H}_t$ (n vertices each)
- assumes t is a power of 2

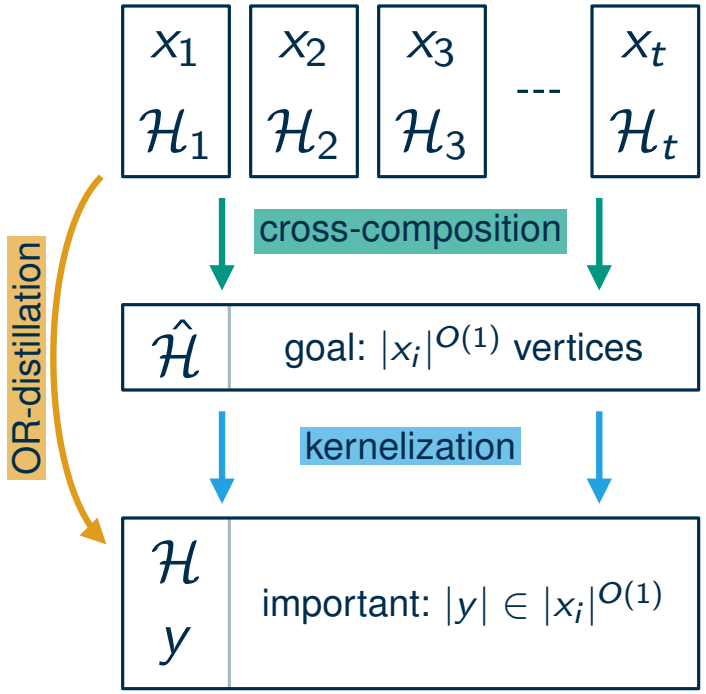


Concluding remarks on SET SPLITTING

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

The cross-composition we have seen

- creates $\hat{\mathcal{H}}$ in time $(\sum |\mathcal{H}_i|)^{O(1)}$ form $\mathcal{H}_1, \ldots, \mathcal{H}_t$ (n vertices each)
- assumes t is a power of 2 (add additional no-instances if not)

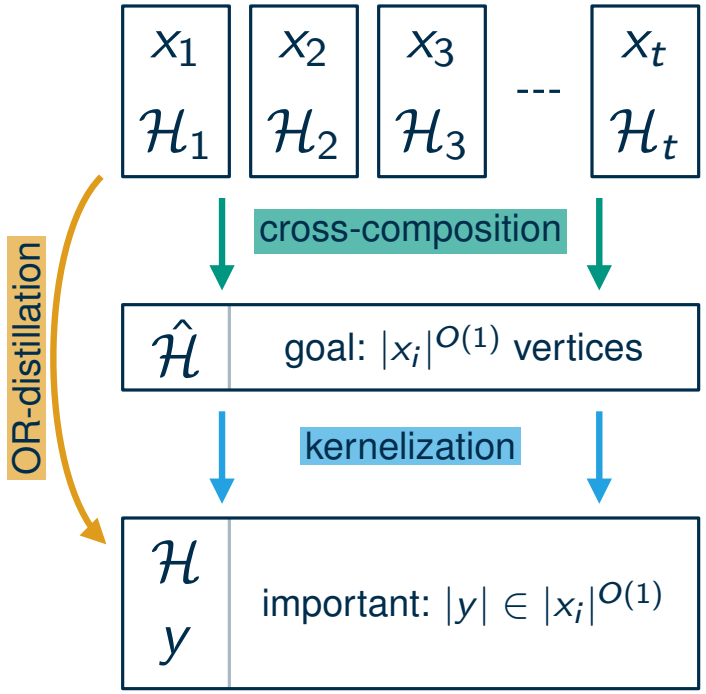


Concluding remarks on SET SPLITTING

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

The cross-composition we have seen

- creates $\hat{\mathcal{H}}$ in time $(\sum |\mathcal{H}_i|)^{O(1)}$ form $\mathcal{H}_1, \ldots, \mathcal{H}_t$ (n vertices each)
- assumes t is a power of 2 (add additional no-instances if not)
- creates $\hat{\mathcal{H}}$ with $n + 2 \log_2 t + 2$ vertices

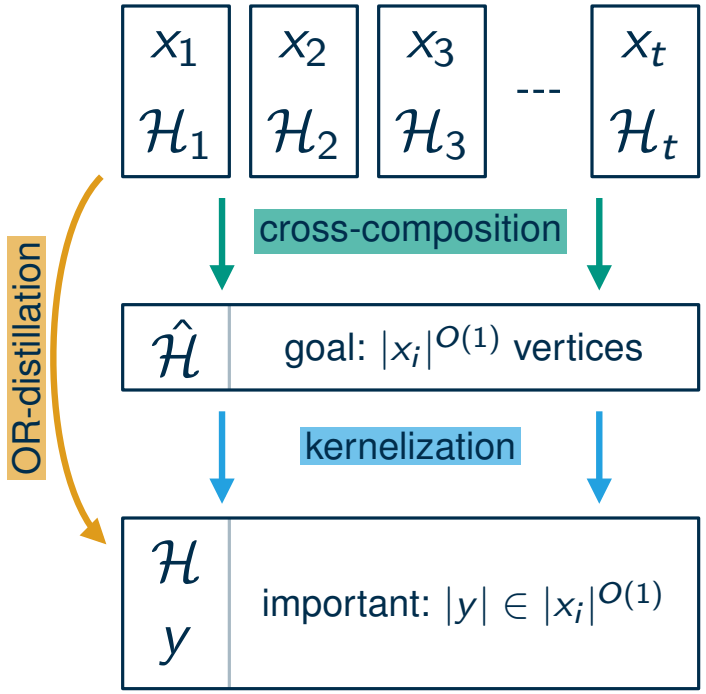


Concluding remarks on SET SPLITTING

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

The cross-composition we have seen

- creates $\hat{\mathcal{H}}$ in time $(\sum |\mathcal{H}_i|)^{O(1)}$ form $\mathcal{H}_1, \ldots, \mathcal{H}_t$ (n vertices each)
- assumes t is a power of 2 (add additional no-instances if not)
- creates $\hat{\mathcal{H}}$ with $n + 2\log_2 t + 2$ vertices
  - only $2^{O(\ell)}$ strings of length $\leq \ell \Rightarrow \log t \in \max |x_i|^{O(1)}$

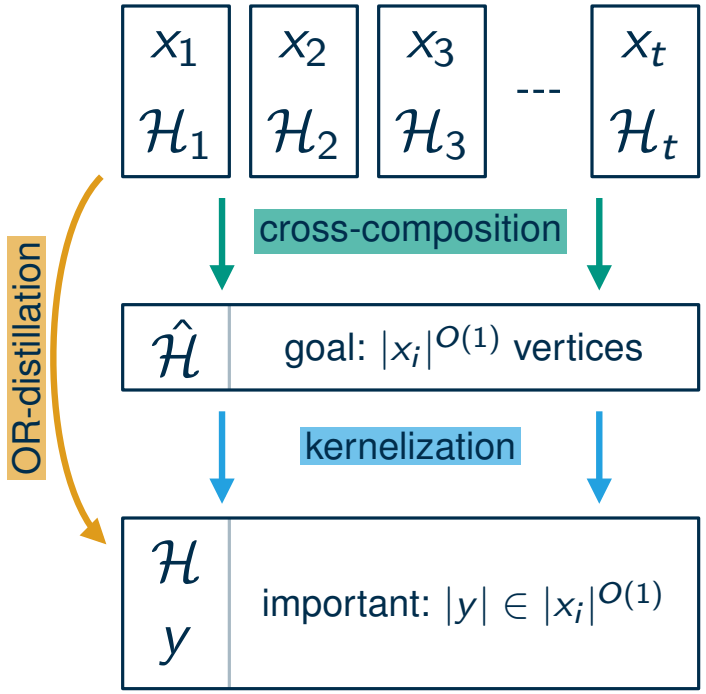


Concluding remarks on Set Splitting

Definition
An **OR-distillation** of L into R takes t similar inputs $x_1, \ldots, x_t$, runs in time $(\sum |x_i|)^{O(1)}$, and produces an output y such that: $|y| \in \max |x_i|^{O(1)}$ and $y \in R \Leftrightarrow \bigvee_{i \in [t]} x_i \in L$.

The cross-composition we have seen

- creates $\hat{\mathcal{H}}$ in time $(\sum |\mathcal{H}_i|)^{O(1)}$ form $\mathcal{H}_1, \ldots, \mathcal{H}_t$ (n vertices each)
- assumes t is a power of 2 (add additional no-instances if not)
- creates $\hat{\mathcal{H}}$ with $n + 2\log_2 t + 2$ vertices
  - only $2^{O(\ell)}$ strings of length $\leq \ell \Rightarrow \log t \in \max |x_i|^{O(1)}$



Theorem
If Set Splitting has a polynomial kernel then NP $\subseteq$ coNP/poly.

Wrap-Up

Kernelization

- being in FPT and having a kernelization algorithm is equivalent
- having a polynomial kernel is a strictly stronger requirement (unless NP $\subseteq$ coNP/poly)

Wrap-Up

Kernelization

- being in FPT and having a kernelization algorithm is equivalent
- having a polynomial kernel is a strictly stronger requirement (unless NP $\subseteq$ coNP/poly)
- thus: kernel size gives a finer separation within FPT

Wrap-Up

Kernelization

- being in FPT and having a kernelization algorithm is equivalent
- having a polynomial kernel is a strictly stronger requirement (unless NP $\subseteq$ coNP/poly)
- thus: kernel size gives a finer separation within FPT
- important: all this only works if the definition of kernelization requires polynomial (not FPT) time

Wrap-Up

Kernelization

- being in FPT and having a kernelization algorithm is equivalent
- having a polynomial kernel is a strictly stronger requirement (unless NP $\subseteq$ coNP/poly)
- thus: kernel size gives a finer separation within FPT
- important: all this only works if the definition of kernelization requires polynomial (not FPT) time

Proving kernel lower bounds

- cross-composition + kernelization gives OR-distillation
- OR-distillation for NP-hard problem implies NP $\subseteq$ coNP/poly
- proving hardness: find a cross-composition

Wrap-Up

Kernelization

- being in FPT and having a kernelization algorithm is equivalent
- having a polynomial kernel is a strictly stronger requirement (unless NP $\subseteq$ coNP/poly)
- thus: kernel size gives a finer separation within FPT
- important: all this only works if the definition of kernelization requires polynomial (not FPT) time

Proving kernel lower bounds

- cross-composition + kernelization gives OR-distillation
- OR-distillation for NP-hard problem implies NP $\subseteq$ coNP/poly
- proving hardness: find a cross-composition
- there are also AND-distillations