

Efficient Simplification: The (im)possibilities

IMPECS 2010

Discovering definitions.

preprocessing, simplification, data reduction

kernelization

KERNELS

Upper Bounds

**Combinatorial Tools
and applications towards
obtaining kernels.**

**Compositions, reductions,
and beyond:
techniques with applications.**

Lower Bounds

Consider the following nightmare from high school algebra homework:

Evaluate:

$$\begin{aligned} &394x^6 + 98x^5y^2 - 821x^7y^8 + 24x^3 \\ &\quad -14x^3 + 600y^8x^7 + 6x^6 \\ &\quad -100x^5y^2 - 400x^6 + 221x^7y^8 \end{aligned}$$

at $x = (7 + 9i)$ and $y = (24 - 5i)$.

Evaluate:

$$\begin{aligned} &394x^6 + 98x^5y^2 - 821x^7y^8 + 24x^3 \\ &\quad - 14x^3 + 600y^8x^7 + 6x^6 \\ &\quad - 100x^5y^2 - 400x^6 + 221x^7y^8 \end{aligned}$$

at $x = (7 + 9i)$ and $y = (24 - 5i)$.

Evaluate:

$$\begin{aligned} &+ 98x^5y^2 - 821x^7y^8 + 24x^3 \\ &- 14x^3 + 600y^8x^7 \\ &- 100x^5y^2 \qquad \qquad + 221x^7y^8 \end{aligned}$$

at $x = (7 + 9i)$ and $y = (24 - 5i)$.

Evaluate:

$$\begin{aligned} &+ 98x^5y^2 - 821x^7y^8 + 24x^3 \\ &- 14x^3 + 600y^8x^7 \\ &- 100x^5y^2 + 221x^7y^8 \end{aligned}$$

at $x = (7 + 9i)$ and $y = (24 - 5i)$.

Evaluate:

$$\begin{aligned} &+ 98x^5y^2 & + 24x^3 \\ &- 14x^3 \\ &- 100x^5y^2 \end{aligned}$$

at $x = (7 + 9i)$ and $y = (24 - 5i)$.

Evaluate:

$$+ 10x^3$$

$$- 2x^5y^2$$

at $x = (7 + 9i)$ and $y = (24 - 5i)$.

Evaluate:

$$\begin{aligned} &394x^6 + 98x^5y^2 - 821x^7y^8 + 24x^3 \\ &\quad -14x^3 + 600y^8x^7 + 6x^6 \\ &\quad -100x^5y^2 - 400x^6 + 221x^7y^8 \end{aligned}$$

at $x = (7 + 9i)$ and $y = (24 - 5i)$.

Evaluate:

$$+ 10x^3$$

$$- 2x^5y^2$$

at $x = (7 + 9i)$ and $y = (24 - 5i)$.

Simplification is a good idea.

Simplification is a good idea.

When you cannot simplify, your problem might be declared *irreducible*.

Simplification is a good idea.

When you cannot simplify, your problem might be declared *irreducible*.

We want to think about *coming up with a definition* for the notion of a *good* simplification procedure.

A simplification procedure is **powerful** if it works...

A simplification procedure is **powerful** if it works...

...all the time — on any input, however complicated.

A simplification procedure is **efficient** if it beats...

A simplification procedure is **efficient** if it beats...

...a brute-force approach hands-down — the faster, the better.

A simplification procedure is good if it is
powerful and efficient.

Are there problems for which we can devise **provably good** simplification procedures?

Notice that if there are problems with good simplification procedures, then such simplifications would be in great demand. Imagine solving all your algebra homework quickly, every single time!

Notice that if there are problems with good simplification procedures, then such simplifications would be in great demand. Imagine solving all your algebra homework **quickly, every single time!**

The performance guarantees make the simplification really special.

Are there problems for which we **provably cannot** devise good simplification procedures?

Notice that showing such a negative result entails demonstrating some “inherent hardness” of the problem, because of which no simplification procedure can offer a performance guarantee.

What does it mean to “simplify”?

The natural idea of simplification is “to reduce”.

The natural idea of simplification is “to reduce”.

Make it smaller.

The natural idea of simplification is “to reduce”.

Make it smaller.

Let's try to formalize this.

Let us use n to denote the “size” of the problem.

Let us use n to denote the “size” of the problem.

Clearly, it is n that we wish to make smaller.

Proposed Definition #1

A simplification routine should:

- run in polynomial time,
- take as input a problem instance of size n , and
- return an equivalent instance of size $n' < n$.

Theorem

A NP-complete problem cannot be simplified unless $P = NP$.

Theorem

A NP-complete problem cannot be simplified unless $P = NP$.

Proof.

Let X be an instance of a NP-complete problem, and suppose it admits a simplification routine.

$$X \longrightarrow X_1 \longrightarrow \cdots \longrightarrow X_n$$

In polynomial time, we have simplified too severely — we have solved the problem! This gives us $P = NP$. □

Proposed Definition #2

Demanding that we reduce the instance with every run of the simplification: unreasonable.

Demanding that we reduce the instance with every run of the simplification: unreasonable.

Let us try demanding that a simplification process reduces *until some point*, after which we might obtain something that cannot be reduced further.

A simplification routine should:

- run in polynomial time,
- take as input a problem instance of size n , and
- return an equivalent instance of size $n' < n$.

A simplification routine should:

- run in polynomial time,
- take as input a problem instance of size n , and
- return an equivalent instance of size 100.

A simplification routine should:

- run in polynomial time,
- take as input a problem instance of size n , and
- return an equivalent instance of size 100.

Does this work?

A simplification routine should:

- run in polynomial time,
- take as input a problem instance of size n , and
- return an equivalent instance of size 100.

NO.

A simplification routine should:

- run in polynomial time,
- take as input a problem instance of size n , and
- return an equivalent instance of size p .

A simplification routine should:

- run in polynomial time,
- take as input a problem instance of size n , and
- return an equivalent instance of size $f(p)$.

A simplification routine should:

- run in polynomial time,
- take as input a problem instance of size n , and
- return an equivalent instance of size $f(p)$.

There are 25 people at a professor's party.

The professor would like to locate a group of *six* people who are popular.

i.e, everyone is a friend of at least one of the six.

Being a busy man, he asks two of his students to find such a group.

The first grimaces and starts making a list of $\binom{25}{6}$ possibilities.

The second knows that no one in the party has more than three friends¹.

¹Academicians tend to be lonely.

There are 25 people at a professor's party.

The professor would like to locate a group of *six* people who are popular.

i.e, everyone is a friend of at least one of the six.

Being a busy man, he asks two of his students to find such a group.

The first grimaces and starts making a list of $\binom{25}{6}$ possibilities.

The second knows that no one in the party has more than three friends¹.

¹Academicians tend to be lonely.

There is a graph on n vertices.

The professor would like to locate a group of *six* people who are popular.

i.e, everyone is a friend of at least one of the six.

Being a busy man, he asks two of his students to find such a group.

The first grimaces and starts making a list of $\binom{25}{6}$ possibilities.

The second knows that no one in the party has more than three friends¹.

¹Academicians tend to be lonely.

There is a graph on n vertices.

Is there a dominating set of size at most k ?

i.e, everyone is a friend of at least one of the six.

Being a busy man, he asks two of his students to find such a group.

The first grimaces and starts making a list of $\binom{25}{6}$ possibilities.

The second knows that no one in the party has more than three friends¹.

¹Academicians tend to be lonely.

There is a graph on n vertices.

Is there a dominating set of size at most k ?

(Every vertex is a neighbor of at least one of the k vertices.)

Being a busy man, he asks two of his students to find such a group.

The first grimaces and starts making a list of $\binom{25}{6}$ possibilities.

The second knows that no one in the party has more than three friends¹.

¹Academicians tend to be lonely.

There is a graph on n vertices.

Is there a dominating set of size at most k ?

(Every vertex is a neighbor of at least one of the k vertices.)

The problem is NP-complete,

The first grimaces and starts making a list of $\binom{25}{6}$ possibilities.

The second knows that no one in the party has more than three friends¹.

¹Academicians tend to be lonely.

There is a graph on n vertices.

Is there a dominating set of size at most k ?

(Every vertex is a neighbor of at least one of the k vertices.)

The problem is NP-complete,

but is trivial on “large” graphs of bounded degree,

The second knows that no one in the party has more than three friends¹.

¹Academicians tend to be lonely.

There is a graph on n vertices.

Is there a dominating set of size at most k ?

(Every vertex is a neighbor of at least one of the k vertices.)

The problem is NP-complete,

but is trivial on “large” graphs of bounded degree,

as you can say NO whenever $n > kb$.

Notation

We denote a parameterized problem as a pair (Q, κ) consisting of a classical problem $Q \subseteq \{0, 1\}^*$ and a parameterization $\kappa : \{0, 1\}^* \rightarrow \mathbb{N}$.

Problem Simplification

involves pruning down

a large input

into an equivalent,

significantly smaller object,

quickly.

Problem Simplification

is a function $f : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \times \mathbb{N}$, such that

a large input

into an equivalent,

significantly smaller object,

quickly.

Problem Simplification

is a function $f : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \times \mathbb{N}$, such that

for all (x, k) , $|x| = n$

into an equivalent,

significantly smaller object,

quickly.

Problem Simplification

is a function $f : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \times \mathbb{N}$, such that

for all (x, k) , $|x| = n$

$f(x), k' \in L$ iff $(x, k) \in L$,

significantly smaller object,

quickly.

Problem Simplification

is a function $f : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \times \mathbb{N}$, such that

for all (x, k) , $|x| = n$

$f(x), k' \in L$ iff $(x, k) \in L$,

$|f(x)| = g(k)$,

quickly.

Problem Simplification

is a function $f : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \times \mathbb{N}$, such that

for all (x, k) , $|x| = n$

$f(x), k' \in L$ iff $(x, k) \in L$,

$|f(x)| = g(k)$,

and f is polytime computable.

A *kernelization* procedure

is a function $f : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \times \mathbb{N}$, such that

for all (x, k) , $|x| = n$

$f(x), k' \in L$ iff $(x, k) \in L$,

$|f(x)| = g(k)$,

and f is polytime computable.

A *kernelization* procedure

is a function $f : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \times \mathbb{N}$, such that

for all (x, k) , $|x| = n$

Kernel

$(f(x), k') \in L$ iff $(x, k) \in L$,

$$|f(x)| = g(k),$$

and f is polytime computable.

A *kernelization* procedure

is a function $f : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \times \mathbb{N}$, such that

for all (x, k) , $|x| = n$

$f(x), k' \in L$ iff $(x, k) \in L$,

$$|f(x)| = g(k),$$

Size of the Kernel

and f is polytime computable.

Theorem

Having a kernelization procedure implies, and is implied by,
parameterized tractability.

Theorem

Having a kernelization procedure implies, and is implied by,
parameterized tractability.

Definition

A parameterized problem L is *fixed-parameter tractable* if there exists an algorithm that decides in $f(k) \cdot n^{O(1)}$ time whether $(x, k) \in L$, where $n := |x|$, $k := \kappa(x)$, and f is a computable function that does not depend on n .

Theorem

A problem admits a kernel if, and only if,
it is fixed-parameter tractable.

Definition

A parameterized problem L is *fixed-parameter tractable* if there exists an algorithm that decides in $f(k) \cdot n^{O(1)}$ time whether $(x, k) \in L$, where $n := |x|$, $k := \kappa(x)$, and f is a computable function that does not depend on n .

Theorem

A problem admits a kernel if, and only if,
it is fixed-parameter tractable.

Given a kernel, a FPT algorithm is immediate (even brute-force on the kernel will lead to such an algorithm).

Theorem

A problem admits a kernel if, and only if,
it is fixed-parameter tractable.

On the other hand, a FPT runtime of $f(k) \cdot n^c$ gives us a $f(k)$ -sized kernel.

We run the algorithm for n^{c+1} steps and either have a trivial kernel if the algorithm stops, else:

$$n^{c+1} < f(k) \cdot n^c$$

Theorem

A problem admits a kernel if, and only if,
it is fixed-parameter tractable.

On the other hand, a FPT runtime of $f(k) \cdot n^c$ gives us a $f(k)$ -sized kernel.

We run the algorithm for n^{c+1} steps and either have a trivial kernel if the algorithm stops, else:

$$n < f(k)$$

“Efficient” Kernelization

What is a reasonable notion of efficiency for kernelization?

“Efficient” Kernelization

What is a reasonable notion of efficiency for kernelization?

The smaller, the better.

“Efficient” Kernelization

What is a reasonable notion of efficiency for kernelization?

The smaller, the better.

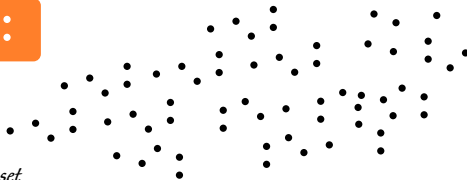
In particular,

Polynomial-sized kernels $\prec$ _{are better than} Exponential-sized Kernels

The problem of finding Dominating Set of size k on graphs where the degree is bounded by b , *parameterized by* k , has a linear kernel. This is an example of a polynomial-sized kernel.

Examples of Polynomial Kernels

Vertex Cover



independent set

Vertex Cover

Input: A graph $G = (V, E)$ and an integer k .

Vertex Cover

Input: A graph $G = (V, E)$ and an integer k .

Question: Does there exist a subset S of at most k vertices such that:
for every edge (u, v) , either $u \in S$ or $v \in S$?

Vertex Cover

Input: A graph $G = (V, E)$ and an integer k .

Question: Does there exist a subset S of at most k vertices such that:
for every edge (u, v) , either $u \in S$ or $v \in S$?

Parameter: k

Vertex Cover

UPPER BOUNDS

Let us simplify!

Vertex Cover

A vertex v of degree more than k is a part of *any* vertex cover of size at most k .

Vertex Cover

A vertex v of degree more than k is a part of *any* vertex cover of size at most k .

There is always a vertex cover that does not pick isolated vertices.

Vertex Cover

A vertex v of degree more than k is a part of *any* vertex cover of size at most k .

There is always a vertex cover that does not pick isolated vertices.

A graph with more than k^2 edges, without isolated vertices and vertices of high degree, has no vertex cover of size at most k .

3-Hitting Set

Input: A set U and a family $\mathcal{F}$ of subsets of U , size three each.

3-Hitting Set

Input: A set U and a family $\mathcal{F}$ of subsets of U , size three each.

Question: Does there exist a subset S of U , of size at most k , such that:
for every set $X \in \mathcal{F}$, $X \cap S \neq \emptyset$?

3-Hitting Set

Input: A set U and a family $\mathcal{F}$ of subsets of U , size three each.

Question: Does there exist a subset S of U , of size at most k , such that:
for every set $X \in \mathcal{F}$, $X \cap S \neq \emptyset$?

Parameter: k

3-Hitting Set

Input: $\{a, b, x\}, \{x, s, r\}, \{s, i, l\}, \{p, q, a\}, \{z, r, c\}$

3-Hitting Set

Input: $\{a, b, x\}, \{x, s, r\}, \{s, i, l\}, \{p, q, a\}, \{z, r, c\}$

Solution: $\{a, s, r\}$ “hits” every set.

3-Hitting Set

Input: $\{a, b, x\}, \{x, s, r\}, \{s, i, l\}, \{p, q, a\}, \{z, r, c\}$

Solution: $\{a, s, r\}$ “hits” every set.

Observation: No hitting set of size two.

3-Hitting Set

Exercise: Is there an $O(k^3)$ kernel? An $O(k^2)$ kernel?

Connected Vertex Cover

Input: A graph $G = (V, E)$ and an integer k .

Connected Vertex Cover

Input: A graph $G = (V, E)$ and an integer k .

Question: Does there exist a subset S of at most k vertices such that: for every edge (u, v) , either $u \in S$ or $v \in S$, **and $G[S]$ is connected?**

Connected Vertex Cover

Input: A graph $G = (V, E)$ and an integer k .

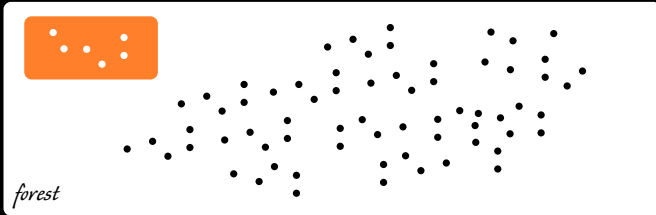
Question: Does there exist a subset S of at most k vertices such that: for every edge (u, v) , either $u \in S$ or $v \in S$, **and $G[S]$ is connected?**

Parameter: k

Connected Vertex Cover

Exercise: Is there an $O(k^2)$ kernel? An $O(k^{O(1)})$ kernel?

Feedback Vertex Set



Feedback Vertex Set

Input: A graph $G = (V, E)$ and an integer k .

Feedback Vertex Set

Input: A graph $G = (V, E)$ and an integer k .

Question: Does there exist a subset S of at most k vertices such that:
 $G[V \setminus S]$ is a forest?

Feedback Vertex Set

Input: A graph $G = (V, E)$ and an integer k .

Question: Does there exist a subset S of at most k vertices such that:
 $G[V \setminus S]$ is a forest?

Parameter: k

Feedback Vertex Set

Notice that in this problem, we are looking to *hit cycles*.

Feedback Vertex Set

Notice that in this problem, we are looking to *hit cycles*.

Since cycles never involve vertices of degree zero or one, such vertices do not participate in an optimal FVS.

Feedback Vertex Set

Notice that in this problem, we are looking to *hit cycles*.

Since cycles never involve vertices of degree zero or one, such vertices do not participate in an optimal FVS.

The immediate simplification is to remove isolated (degree 0) and pendant (degree 1) vertices.

Feedback Vertex Set

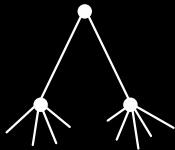
What about vertices of degree two?

Feedback Vertex Set

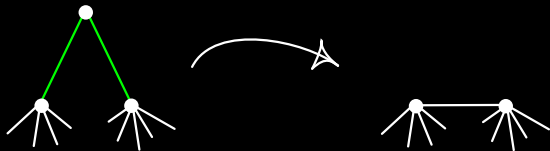
What about vertices of degree two?

These are also simple to handle...

Feedback Vertex Set



Feedback Vertex Set



Feedback Vertex Set

Notice that we now have a graph whose *minimum* degree is at least three.

Feedback Vertex Set

Notice that we now have a graph whose *minimum* degree is at least three.

What about vertices of high degree?

Feedback Vertex Set

Recall that we were lucky with high degree vertices for vertex cover.

Feedback Vertex Set

Recall that we were lucky with high degree vertices for vertex cover.

Let's be optimistic and *assume* that we will get lucky again...

Feedback Vertex Set

So suppose X is a procedure for simplifying vertices of degree more than s .

Feedback Vertex Set

So suppose X is a procedure for simplifying vertices of degree more than s .

What can we say about a graph that is simplified to the extent of being degree at least three, and not having vertices of degree more than s ?

Feedback Vertex Set

$$1.5n \leq \# \text{of edges}$$

Minimum degree three...

FVS

forest

Feedback Vertex Set

$$1.5n \leq \# \text{of edges} \leq sk + k^2 + (n - k - 1)$$

FVS

forest

Feedback Vertex Set

$$1.5n \leq \# \text{of edges} \leq sk + k^2 + (n - k - 1)$$

$$n \leq 2(sk + (k - 1) + k^2)$$

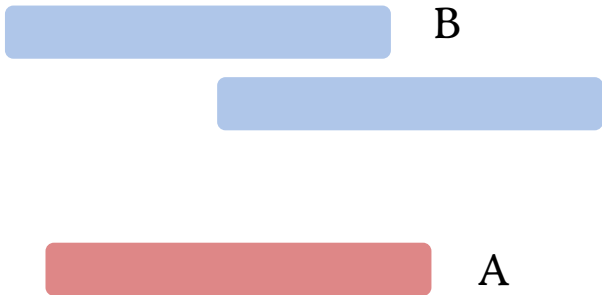
FVS

forest

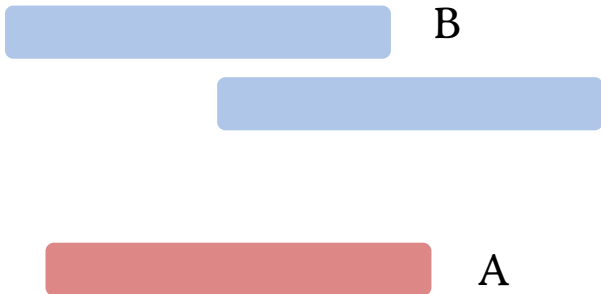
²⁾

Feedback Vertex Set

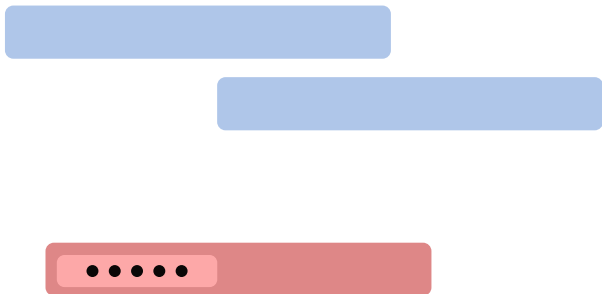
Now we simply have to target high-degree vertices, and try to discover the procedure X that we assumed before.



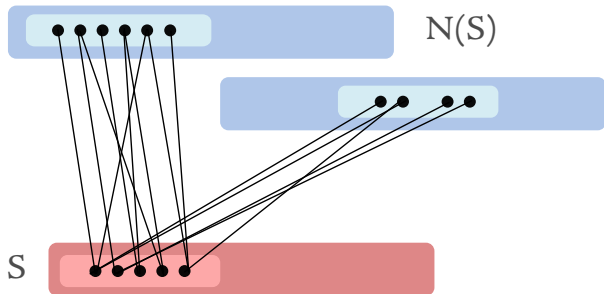
Consider a bipartite graph one of whose parts (say B) is at least twice as big as the other (call this A).



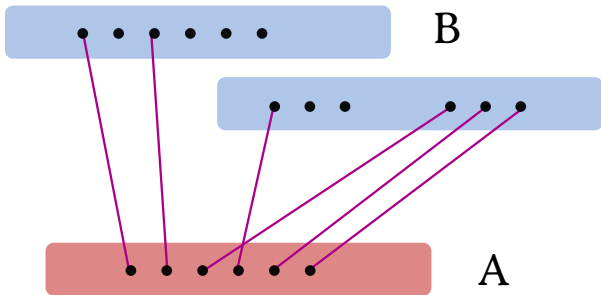
Assume that there are no isolated vertices in B.



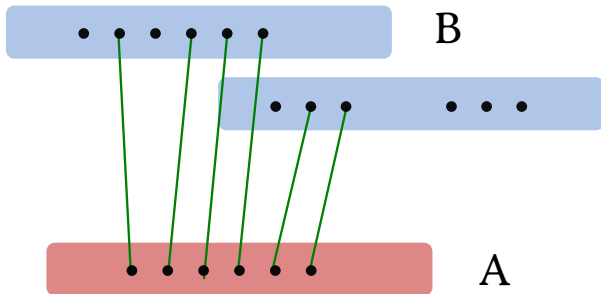
Suppose, further, that for every subset S in A ,



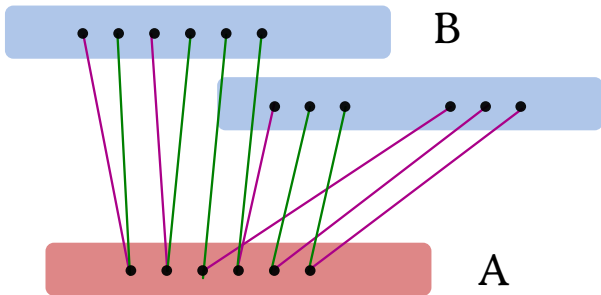
Suppose, further, that for every subset S in A , $N(S)$ is at least twice as large as $|S|$.



Then there exist two matchings saturating A ,



Then there exist two matchings saturating A ,



Then there exist two matchings saturating A ,
and disjoint in B .

Claim:

If $N(A) \geq 2|A|$, then there exists a subset S of A such that:

there 2 matchings saturating the subset S
that are vertex-disjoint in B .

Claim:

If $N(A) \geq 2|A|$, then there exists a subset S of A such that:

there 2 matchings saturating the subset S
that are vertex-disjoint in B .

Claim:

If $|B| \geq 2|A|$, then there exists a subset S of A such that:

there 2 matchings saturating the subset S
that are vertex-disjoint in B ,

provided B does not have any isolated vertices.

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

In this case, $S = A$ and we are done.

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.
Throw it away.

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.

Throw it away.

Delete the vertices of S and the edges incident on it,

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.

Throw it away.

and remove any isolated vertices created in B .

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.
Throw it away.

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.

Throw it away.

In the graph that is left, $|B|$ is still
larger than $2|A|$.

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.

Throw it away.

In the graph that is left, $|B|$ is still
larger than $2|A|$.

$$N(S) \leq 2|S|$$

If $N(S) \geq 2|S|$ for all subsets S , then
 there are 2 matchings,
 vertex-disjoint in B , saturating A .

Else: there exists a subset S for
 which $N(S)$ is smaller than $2|S|$.

Throw it away.

In the graph that is left, $|B|$ is still
 larger than $2|A|$.

$$- N(S) \geq - 2|S|$$

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.

Throw it away.

In the graph that is left, $|B|$ is still
larger than $2|A|$.

$$|B| - N(S) \geq 2|A| - 2|S|$$

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.

Throw it away.

In the graph that is left, $|B|$ is still
larger than $2|A|$.

$$|B| - N(S) \geq 2|A| - 2|S|$$

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.

Throw it away.

In the graph that is left, $|B|$ is still
larger than $2|A|$.

$$|B| - N(S) \geq 2(|A| - |S|)$$

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.
Throw it away.

Done: by repetition.

$$|B| - N(S) \geq 2(|A| - |S|)$$

If $N(S) \geq 2|S|$ for all subsets S , then
there are 2 matchings,
vertex-disjoint in B , saturating A .

Else: there exists a subset S for
which $N(S)$ is smaller than $2|S|$.

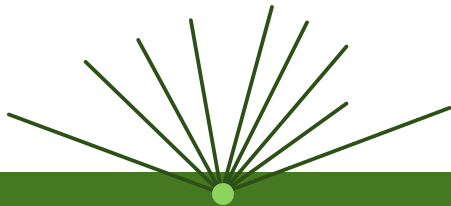
Throw it away.

Done: by repetition.

**(And you'll never clear everything
out, because $|B|$ is consistently
more than $2|A|$.)**

$$|B| - N(S) \geq 2(|A| - |S|)$$

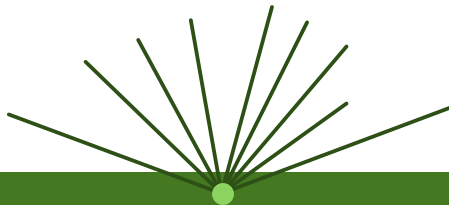
Ingredients



a high-degree vertex, v

Ingredients

a small hitting set,
sans v



a high-degree vertex, v

Ingredients

Given a high-degree vertex v , finding a small hitting set that does not contain v .

Given a high-degree vertex v , finding a small **hitting set** that does not contain v .

A subset whose removal makes the graph acyclic.

Given a high-degree vertex v , finding a **small** hitting set that does not contain v .

A polynomial function of k .

Given a **high-degree** vertex v , finding a small hitting set that does not contain v .

At least twice the size of the hitting set.

Given a high-degree vertex v , **finding** a small hitting set that does not contain v .

Find an **approximate** hitting set S .

Given a high-degree vertex v , **finding** a small hitting set that does not contain v .

If S does not contain v , we are done.

Given a high-degree vertex v , **finding** a small hitting set that does not contain v .

Else: $v \in S$. Delete $S \setminus v$ from G .

Given a high-degree vertex v , **finding** a small hitting set that does not contain v .

The only remaining cycles pass through v .

Given a high-degree vertex v , **finding** a small hitting set that does not contain v .

Find an optimal cut set for paths from $N(v)$ to $N(v)$.

Given a high-degree vertex v , **finding** a small hitting set that does not contain v .

Why is this cut set small enough?

Given a high-degree vertex v , **finding** a small hitting set that does not contain v .

When is this cut set small enough?

Given a high-degree vertex v , finding a small hitting set that does not contain v .

When is this cut set small enough?

When the largest collection of vertex disjoint paths from $N(v)$ to $N(v)$ is small.

Given a high-degree vertex v , **finding** a small hitting set that does not contain v .

When is this cut set small enough?

When the largest collection of vertex disjoint paths from $N(v)$ to $N(v)$ is *not* small...

Given a high-degree vertex v , **finding** a small hitting set that does not contain v .

When is this cut set small enough?

When the largest collection of vertex disjoint paths from $N(v)$ to $N(v)$ is *not* small... we get a reduction rule.

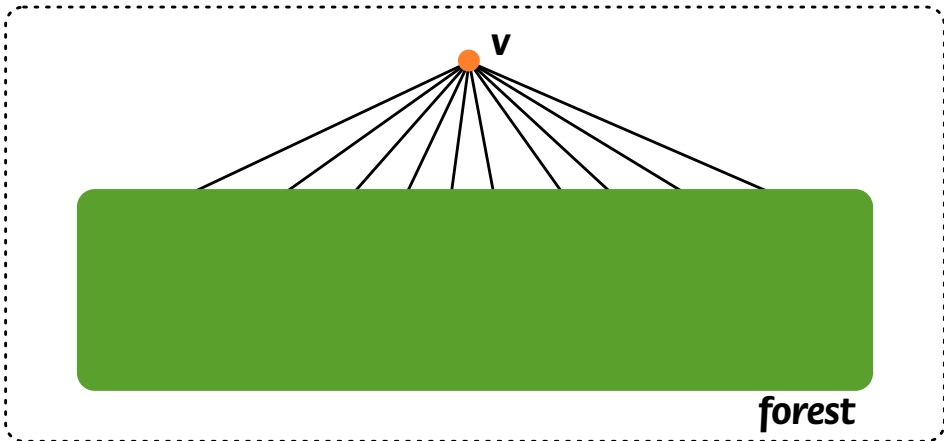
Given a high-degree vertex v , **finding** a small hitting set that does not contain v .

When is this cut set small enough?

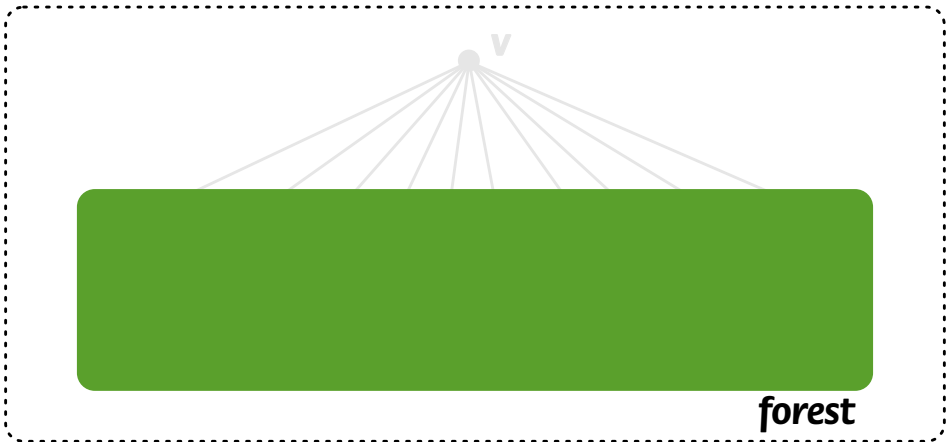
More than k vertex-disjoint paths from $N(v)$ to $N(v)$
→ v belongs to *any* hitting set of size at most k .

Given a high-degree vertex v , finding a small hitting set that does not contain v .

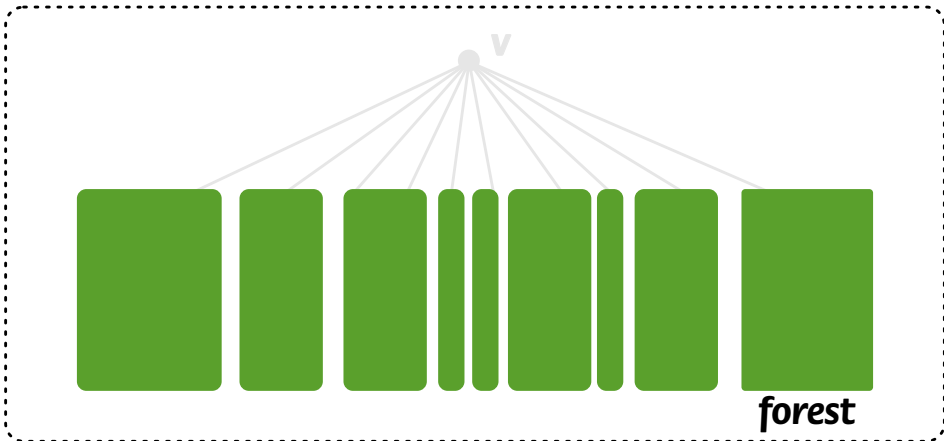
So either v “forced”, or we have hitting set of suitable size.
Notice that we arrive at either situation in polynomial time.



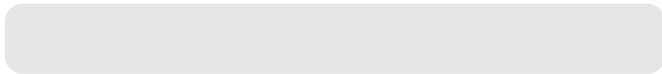
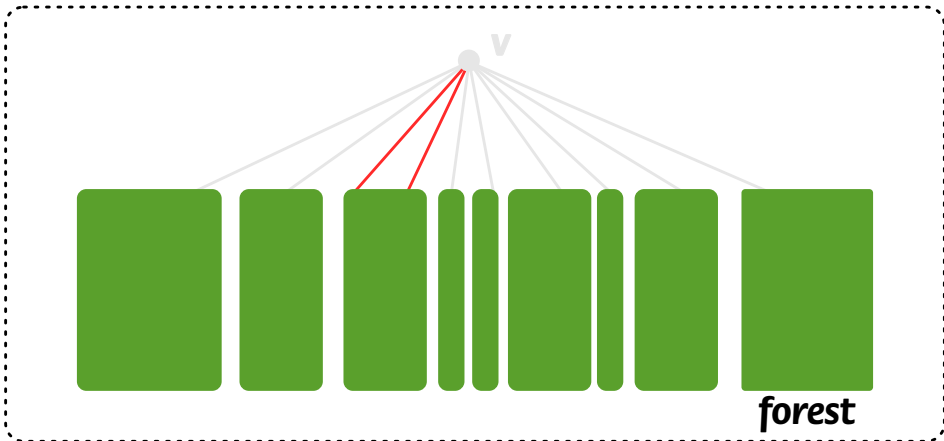
hitting set that excludes v



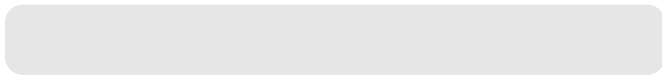
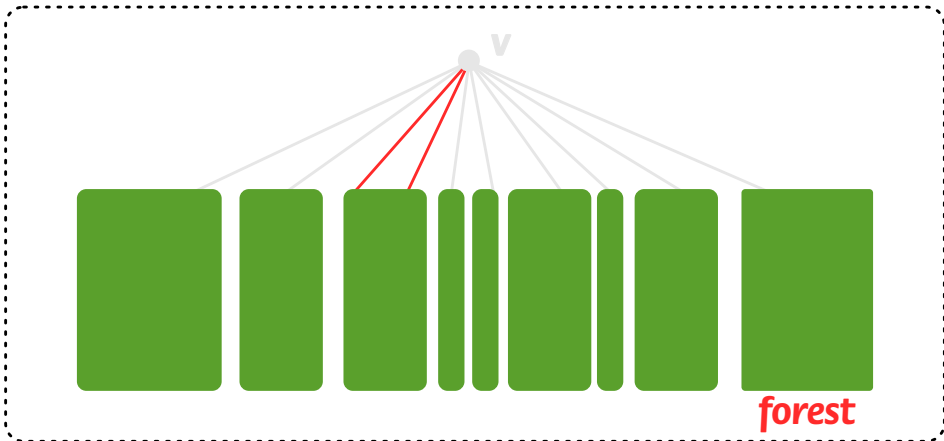
hitting set that excludes v



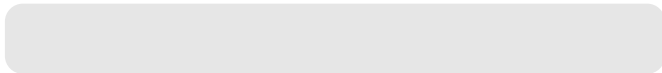
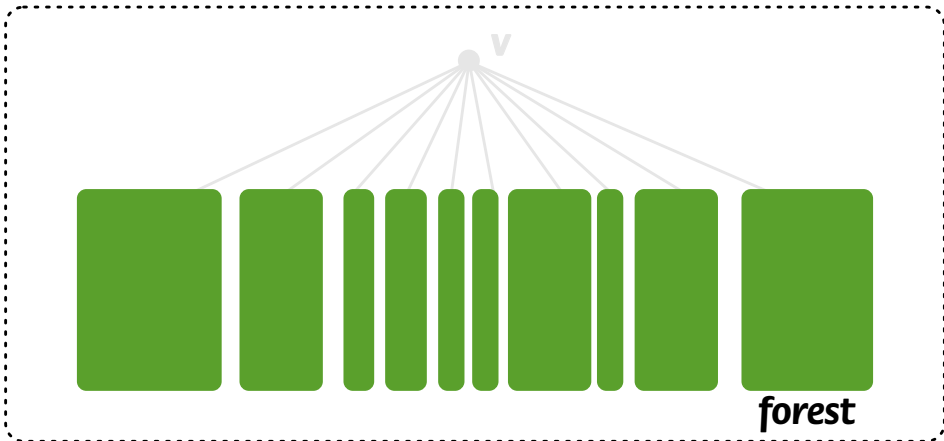
hitting set that excludes v



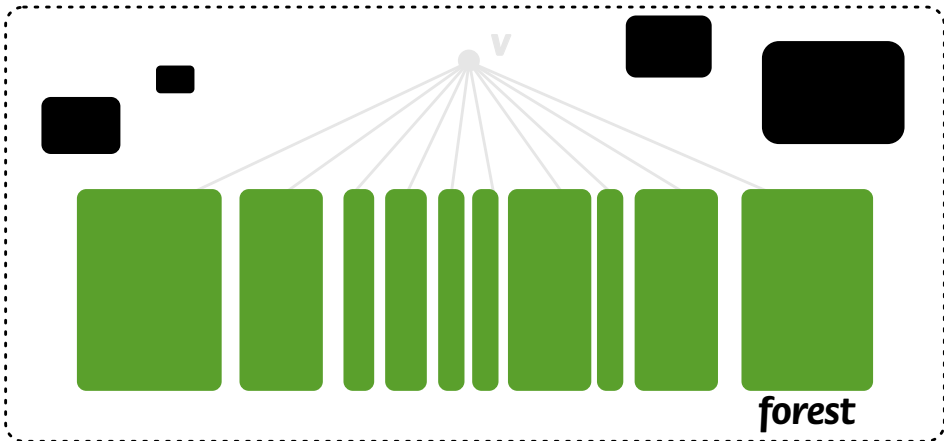
hitting set that excludes v



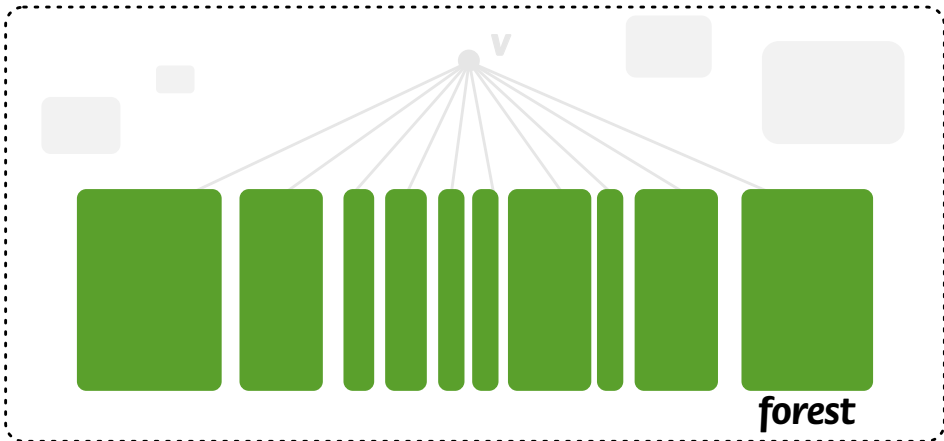
hitting set that excludes v



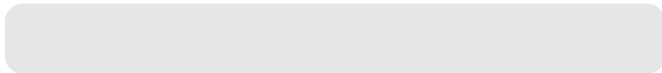
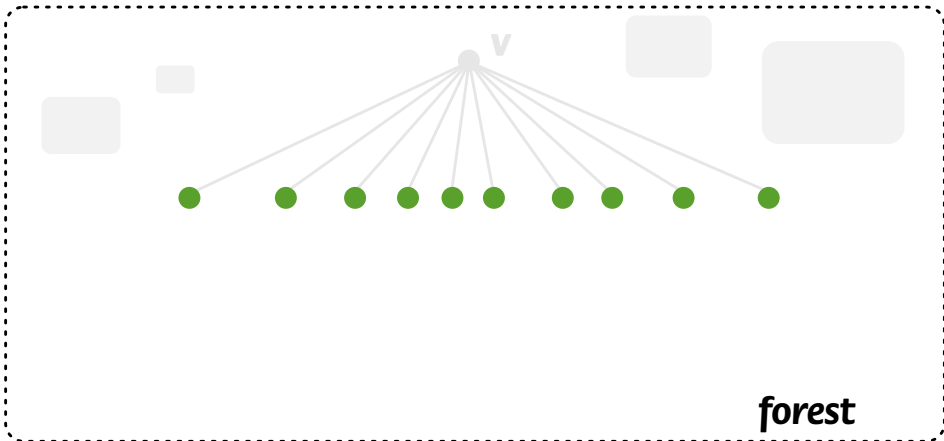
hitting set that excludes v



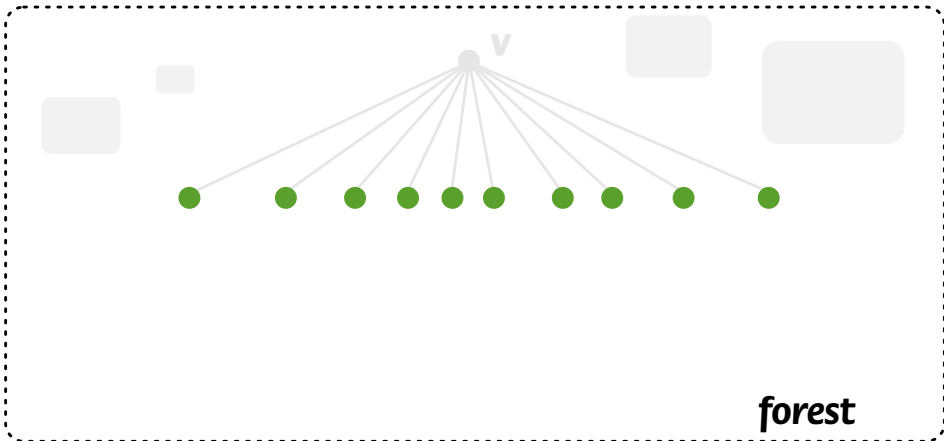
hitting set that excludes v



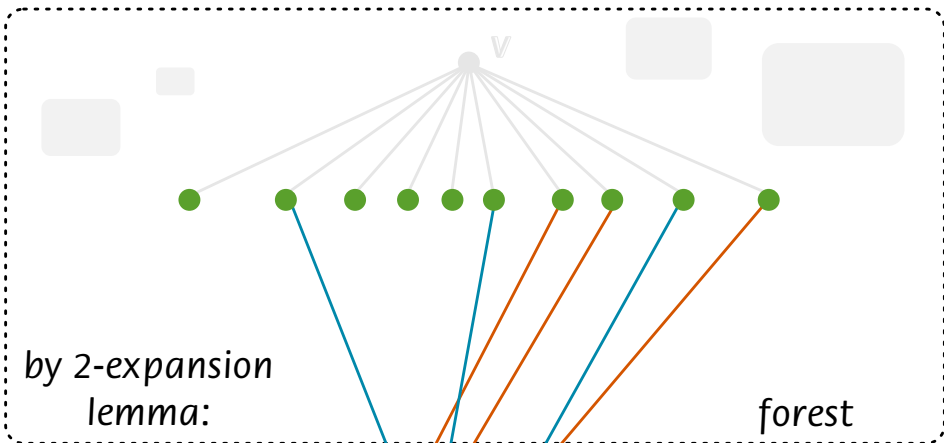
hitting set that excludes v



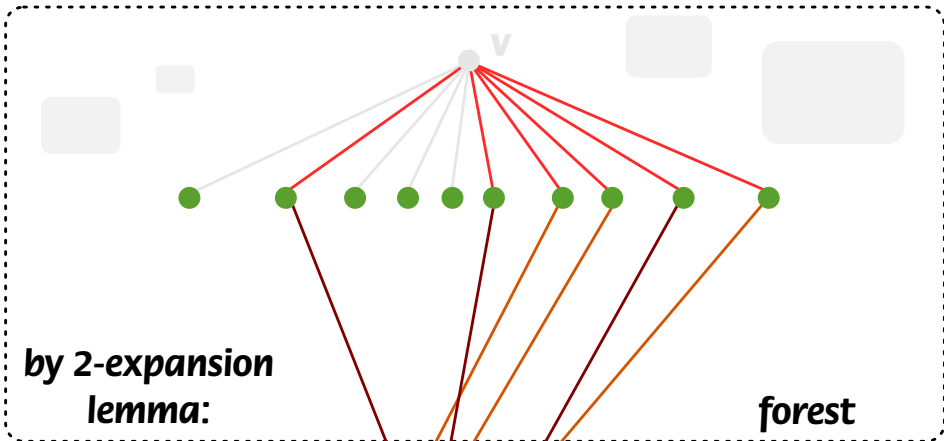
hitting set that excludes v



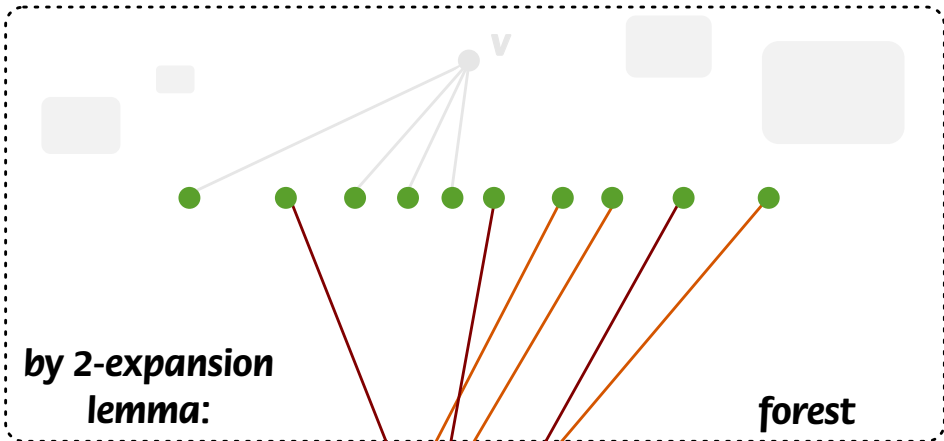
hitting set that excludes v



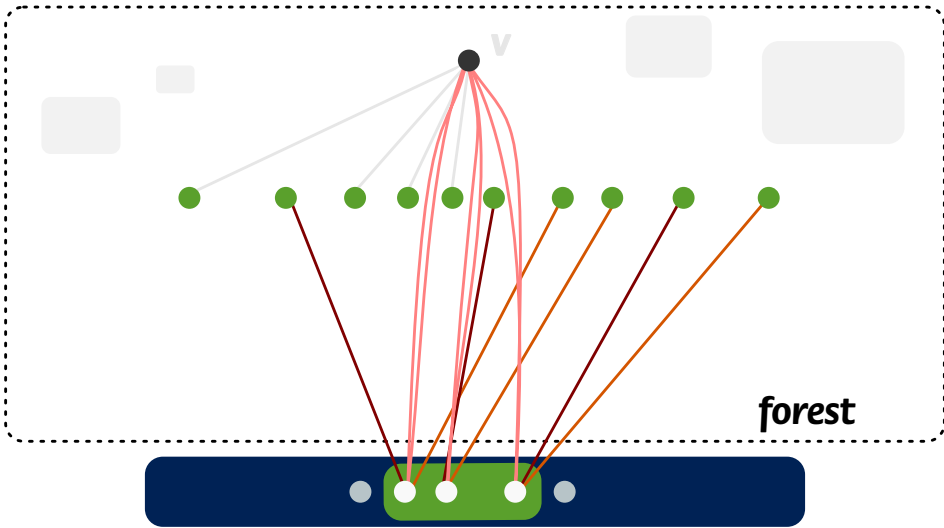
hitting set that excludes v



hitting set that excludes v



hitting set that excludes v



hitting set that excludes v

The Forward Direction

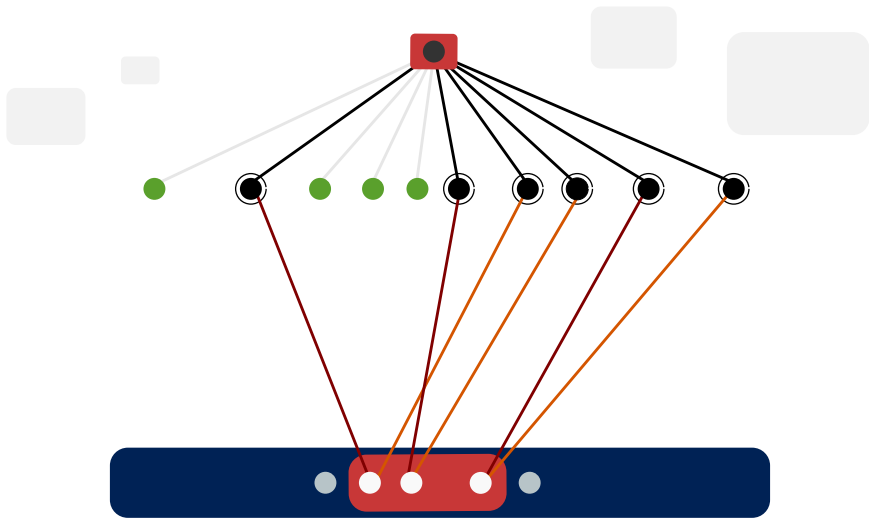
The Forward Direction

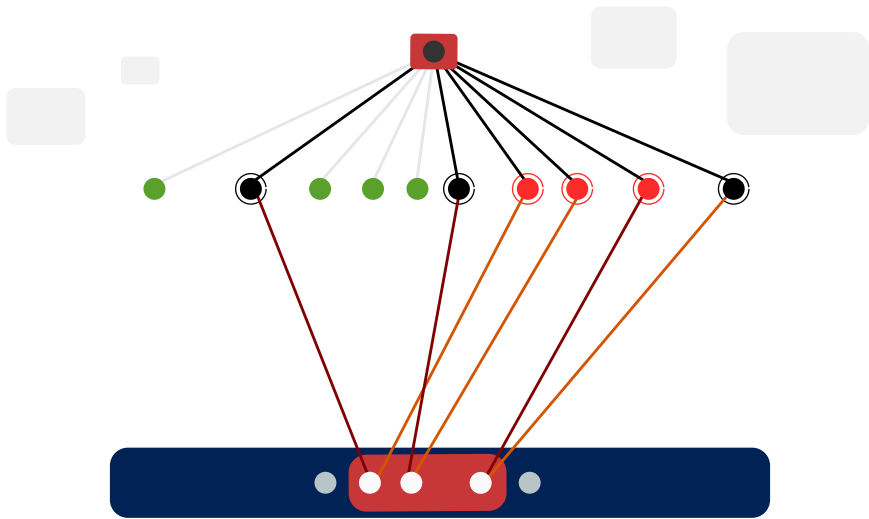
$$\text{FVS} \leq k \text{ in } G \Rightarrow \text{FVS} \leq k \text{ in } H$$

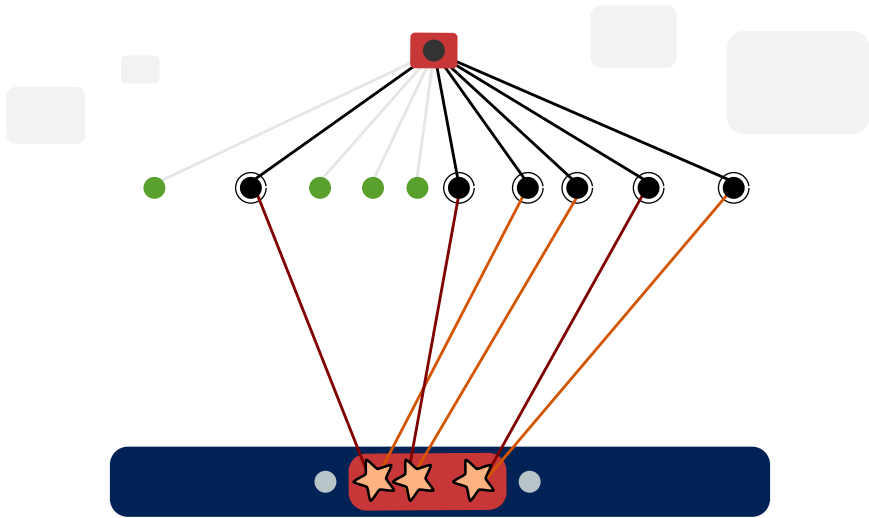
If G has a FVS that either contains v or all of S , we are in good shape.

Consider now a FVS that:

- Does not contain v ,
- and omits at least one vertex of S .







Notice that this does not lead to a larger FVS:

Notice that this does not lead to a larger FVS:

For every vertex v in S that a FVS of G leaves out,

Notice that this does not lead to a larger FVS:

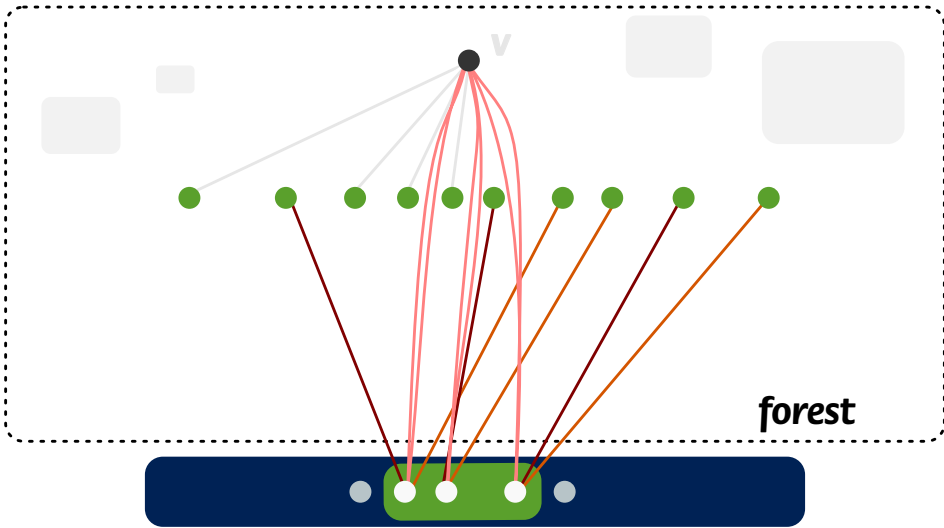
For every vertex v in S that a FVS of G leaves out,

it must pick a vertex u that kills no more than all of S .

The Reverse Direction

The Reverse Direction

$$\text{FVS} \leq k \text{ in } G \Leftarrow \text{FVS} \leq k \text{ in } H$$



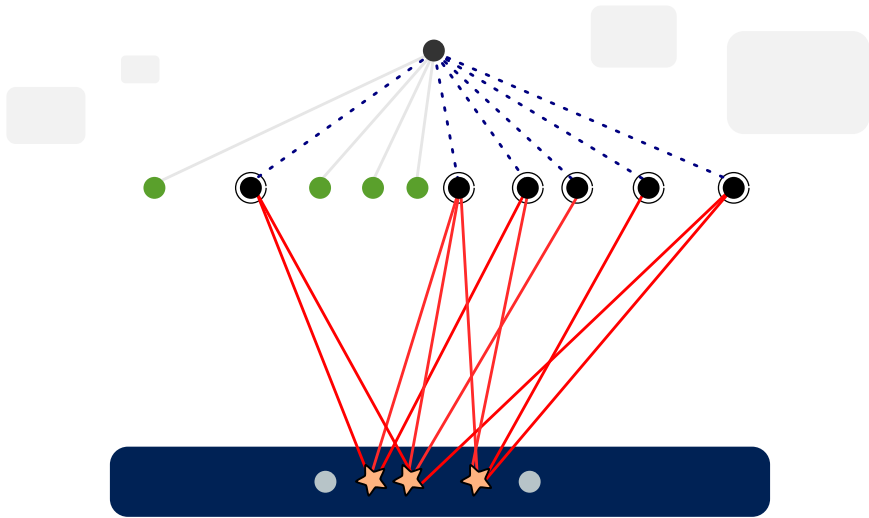
hitting set that excludes v

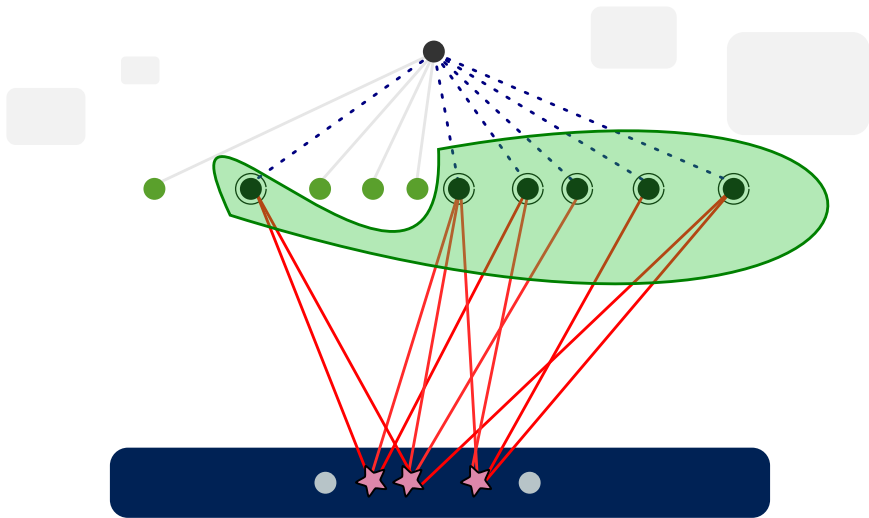
The Only Danger

Cycles that:

- pass through v ,
- non-neighbors of v in H^1
- and do not pass through S .

¹neighbors in G , however





Feedback Vertex Set

Having worked out the details, it turns out that vertices whose degree is more than $8k$ can be handled, so this leads to an $O(k^2)$ kernel.

Demonstrating non-existence of Polynomial Kernels

LOWER BOUNDS

Consider a number of instances of the boolean satisfiability problem:

$$\phi_1 \quad , \quad \phi_2 \quad , \quad \phi_3 \quad , \quad \dots \quad , \quad \phi_t$$

LOWER BOUNDS

Consider a number of instances of the boolean satisfiability problem:

$$\phi_1 \quad , \quad \phi_2 \quad , \quad \phi_3 \quad , \quad \dots \quad , \quad \phi_t$$

Can we come up with a short formula ψ that is equivalent to:

$$\phi_1 \quad \vee \quad \phi_2 \quad \vee \quad \phi_3 \quad \vee \quad \dots \quad \vee \quad \phi_t$$

LOWER BOUNDS

Consider a number of instances of the boolean satisfiability problem:

Let's say all instances are of size n

ϕ_1 , ϕ_2 , ϕ_3 , ... , ϕ_t

Can we come up with a **short** formula ψ that is equivalent to:

$\phi_1 \vee \phi_2 \vee \phi_3 \vee \dots \vee \phi_t$

Polynomial in n , and *independent* of t

LOWER BOUNDS

Consider a number of instances of the boolean satisfiability problem:

Let's say all instances are of size n

ϕ_1 , ϕ_2 , ϕ_3 , ... , ϕ_t

Can we come up with a short formula ψ that is equivalent to:

$\phi_1 \vee \phi_2 \vee \phi_3 \vee \dots \vee \phi_t$

Polynomial in n , and *independent* of t

It turns out that: ψ cannot be obtained in polynomial time.

LOWER BOUNDS

Consider a number of instances of the boolean satisfiability problem:

Let's say all instances are of size n

$$(\phi_1, p), \quad (\phi_2, p), \quad (\phi_3, p), \quad \dots, \quad (\phi_t, p)$$

Can we come up with a ~~short~~ formula ψ that is equivalent to:

$$(\phi_1, p) \vee (\phi_2, p) \vee (\phi_3, p) \vee \dots \vee (\phi_t, p)$$

It turns out that: ψ cannot be obtained in polynomial time.

LOWER BOUNDS

Consider a number of instances of the boolean satisfiability problem:

Let's say all instances are of size n

$(\phi_1, p), (\phi_2, p), (\phi_3, p), \dots, (\phi_t, p)$

Can we come up with a ~~short~~ formula ψ that is equivalent to:

$(\phi_1, p) \vee (\phi_2, p) \vee (\phi_3, p) \vee \dots \vee (\phi_t, p)$

If we can write a long formula ψ : we cannot simplify it, because:

It turns out that: a short ψ cannot be obtained in polynomial time.

LOWER BOUNDS

A composition algorithm $\mathcal{A}$ for a problem is designed to act as a fast Boolean OR of multiple problem-instances.

It receives as input a sequence of instances.

It produces as output a yes-instance with a small parameter if and only if at least one of the instances in the sequences is also a yes-instance.

LOWER BOUNDS

A composition algorithm $\mathcal{A}$ for a problem is designed to act as a fast Boolean OR of multiple problem-instances.

It receives as input a sequence of instances.

$\bar{x} = (x_1, \dots, x_t)$ with $x_i \in \{0, 1\}^*$ for $i \in [t]$, such that
 $k_1 = k_2 = \dots = k_t = k$

It produces as output a yes-instance with a small parameter if and only if at least one of the instances in the sequences is also a yes-instance.

LOWER BOUNDS

A composition algorithm $\mathcal{A}$ for a problem is designed to act as a fast Boolean OR of multiple problem-instances.

It receives as input a sequence of instances.

$\bar{x} = (x_1, \dots, x_t)$ with $x_i \in \{0, 1\}^*$ for $i \in [t]$, such that
 $k_1 = k_2 = \dots = k_t = k$

It produces as output a yes-instance with a **small parameter** if and only if at least one of the instances in the sequences is also a yes-instance.

$$k' = k^{O(1)}$$

LOWER BOUNDS

A composition algorithm $\mathcal{A}$ for a problem is designed to act as a **fast** Boolean OR of multiple problem-instances.

It receives as input a sequence of instances.

$\bar{x} = (x_1, \dots, x_t)$ with $x_i \in \{0, 1\}^*$ for $i \in [t]$, such that
 $k_1 = k_2 = \dots = k_t = k$

It produces as output a yes-instance with a **small parameter** if and only if at least one of the instances in the sequences is also a yes-instance.

$$k' = k^{O(1)}$$

Running time polynomial in $\sum_{i \in [t]} |x_i|$

LOWER BOUNDS

The Recipe for Hardness

Composition Algorithm + Polynomial Kernel



Distillation Algorithm



$$\text{PH} = \Sigma_3^{\text{P}}$$

LOWER BOUNDS

The Recipe for Hardness

Composition Algorithm + Polynomial Kernel



Distillation Algorithm



$$\text{PH} = \Sigma_3^{\text{P}}$$

LOWER BOUNDS

Theorem. Let (P, k) be a compositional parameterized problem such that P is NP-complete. If P has a polynomial kernel, then P also has a distillation algorithm.

Transformations

Let (P, κ) and (Q, γ) be parameterized problems.

We say that there is a polynomial parameter transformation from P to Q if there exists a polynomial time computable function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$, and a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$, such that, if $f(x) = y$, we have:

$$x \in P \text{ if and only if } y \in Q,$$

Transformations

Let (P, κ) and (Q, γ) be parameterized problems.

We say that there is a polynomial parameter transformation from P to Q if there exists a polynomial time computable function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$, and a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$, such that, if $f(x) = y$, we have:

$x \in P$ if and only if $y \in Q$,

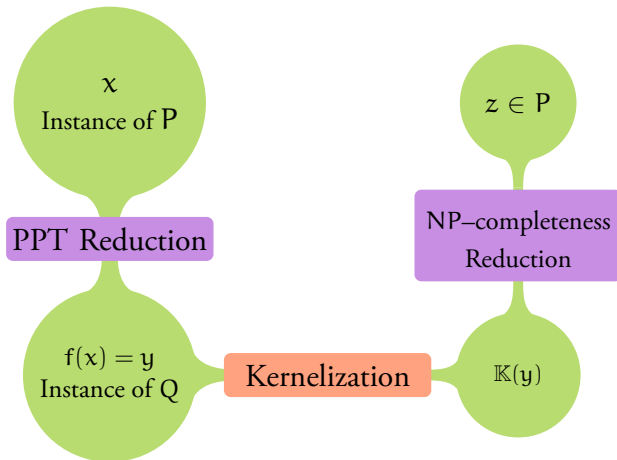
and

$$\gamma(y) \leq p(\kappa(x))$$

LOWER BOUNDS

Theorem: Suppose P is NP-complete, and $Q \in \text{NP}$. If f is a polynomial time and parameter transformation from P to Q and Q has a polynomial kernel, then P has a polynomial kernel.

Theorem: Suppose P is NP-complete, and $Q \in \text{NP}$. If f is a polynomial time and parameter transformation from P to Q and Q has a polynomial kernel, then P has a polynomial kernel.



LOWER BOUNDS

Recall

A composition for a parameterized language (Q, κ) is required to “merge” instances

$$x_1, x_2, \dots, x_t,$$

into a single instance x in polynomial time, such that $\kappa(x)$ is polynomial in $k := \kappa(x_i)$ for any i .

The output of the algorithm belongs to $(Q, \kappa(x))$
if, and only if
there exists at least one $i \in [t]$ for which $x_i \in (Q, \kappa)$.

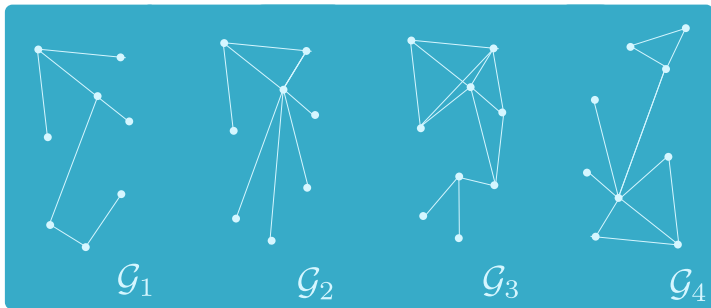
LOWER BOUNDS

k-Path



LOWER BOUNDS

k-Path



LOWER BOUNDS

Boolean satisfiability

ϕ_1 ϕ_2 ϕ_3 ϕ_4 ϕ_5

LOWER BOUNDS

Boolean satisfiability

$$\phi_1 \vee \phi_2 \vee \phi_3 \vee \phi_4 \vee \phi_5$$

LOWER BOUNDS

Weighted Satisfiability

Given a CNF formula ϕ ,
Is there a satisfying assignment of weight at most k ?

LOWER BOUNDS

Weighted Satisfiability

Given a CNF formula ϕ ,
Is there a satisfying assignment of weight at most k ?

When the length of the longest clause is bounded by b ,
there is an easy branching algorithm with runtime $\mathcal{O}(b^k \cdot p(n))$.

LOWER BOUNDS

Weighted Satisfiability

Given a CNF formula ϕ ,
Is there a satisfying assignment of weight at most k ?

Parameter: $b+k$.

When the length of the longest clause is bounded by b ,
there is an easy branching algorithm with runtime $\mathcal{O}(b^k \cdot p(n))$.

LOWER BOUNDS

Input: $\alpha_1, \alpha_2, \dots, \alpha_t$.

LOWER BOUNDS

Input: $\alpha_1, \alpha_2, \dots, \alpha_t$.

$\kappa(\alpha_i) = b + k$.

LOWER BOUNDS

Input: $\alpha_1, \alpha_2, \dots, \alpha_t$.

$\kappa(\alpha_i) = b + k$.

$n := \max_{i \in [n]} n_i$

LOWER BOUNDS

Input: $\alpha_1, \alpha_2, \dots, \alpha_t$.

$\kappa(\alpha_i) = b + k$.

$n := \max_{i \in [n]} n_i$

If $t > b^k$, then solve every α_i individually.

LOWER BOUNDS

Input: $\alpha_1, \alpha_2, \dots, \alpha_t$.

$\kappa(\alpha_i) = b + k$.

$n := \max_{i \in [n]} n_i$

If $t > b^k$, then solve every α_i individually.

Total time = $t \cdot b^k \cdot p(n)$

LOWER BOUNDS

Input: $\alpha_1, \alpha_2, \dots, \alpha_t$.

$\kappa(\alpha_i) = b + k$.

$n := \max_{i \in [n]} n_i$

If $t > b^k$, then solve every α_i individually.

Total time $< t \cdot t \cdot p(n)$

LOWER BOUNDS

Input: $\alpha_1, \alpha_2, \dots, \alpha_t$.

$\kappa(\alpha_i) = b + k$.

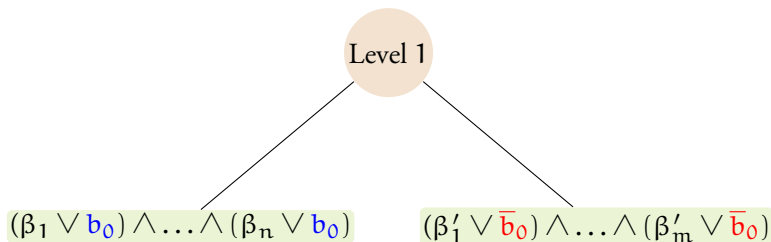
$n := \max_{i \in [n]} n_i$

If $t > b^k$, then solve every α_i individually.

Total time $< t \cdot t \cdot p(n)$

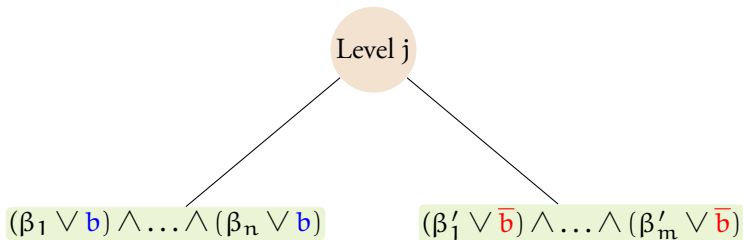
If not, $t < b^k$ – this gives us a bound on the number of instances.

LOWER BOUNDS

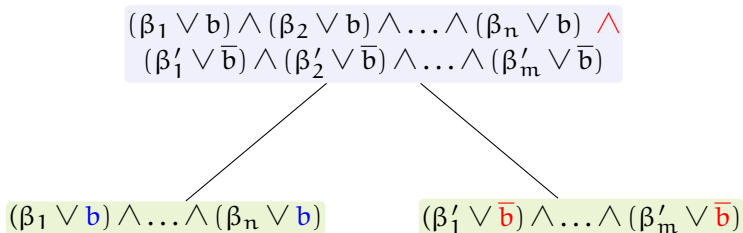


This is the scene at the leaves, where the β_j s are the clauses in α_i for some i and the β'_j s are clauses of α_{i+1} .

LOWER BOUNDS



LOWER BOUNDS



Take the conjunction of the formulas stored at the child nodes.

$$(\beta_1 \vee b \vee b_j) \wedge (\beta_2 \vee b \vee b_j) \wedge \dots \wedge (\beta_n \vee b \vee b_j) \wedge \\ (\beta'_1 \vee \bar{b} \vee b_j) \wedge (\beta'_2 \vee \bar{b} \vee b_j) \wedge \dots \wedge (\beta'_m \vee \bar{b} \vee b_j)$$

$$(\beta_1 \vee b) \wedge \dots \wedge (\beta_n \vee b)$$

$$(\beta'_1 \vee \bar{b}) \wedge \dots \wedge (\beta'_m \vee \bar{b})$$

if the parent is a “left child”.

LOWER BOUNDS

$$(\beta_1 \vee b \vee \bar{b}_j) \wedge (\beta_2 \vee b \vee \bar{b}_j) \wedge \dots \wedge (\beta_n \vee b \vee \bar{b}_j) \wedge$$

$$(\beta'_1 \vee \bar{b} \vee \bar{b}_j) \wedge (\beta'_2 \vee \bar{b} \vee \bar{b}_j) \wedge \dots \wedge (\beta'_m \vee \bar{b} \vee \bar{b}_j)$$

$$(\beta_1 \vee b) \wedge \dots \wedge (\beta_n \vee b)$$

$$(\beta'_1 \vee \bar{b}) \wedge \dots \wedge (\beta'_m \vee \bar{b})$$

if the parent is a “right child”.

LOWER BOUNDS

adding a suffix to “control the weight”

$$\begin{aligned} & \alpha \\ & \wedge \\ & (\bar{c}_0 \vee \bar{b}_0) \wedge (c_0 \vee b_0) \wedge \\ & (\bar{c}_1 \vee \bar{b}_1) \wedge (c_1 \vee b_1) \wedge \\ & \dots \\ & (\bar{c}_i \vee \bar{b}_i) \wedge (c_i \vee b_i) \wedge \\ & \dots \\ & (\bar{c}_{l-1} \vee \bar{b}_{l-1}) \wedge (c_{l-1} \vee b_{l-1}) \end{aligned}$$

LOWER BOUNDS

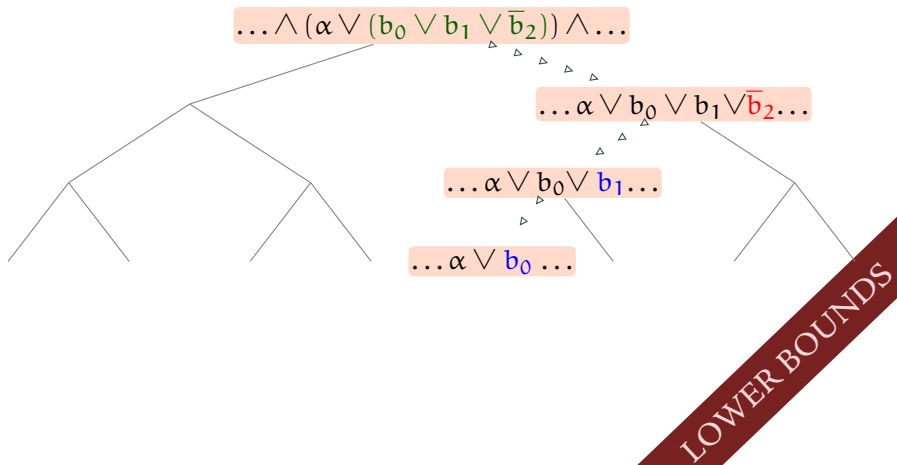
Claim

The composed instance α has a satisfying assignment of **weight $2k$**



at least one of the input instances admit a satisfying assignment of **weight k** .

Proof of Correctness



Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

LOWER BOUNDS

Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

A factor of a word $w_1 \cdots w_r \in L_k^*$ is a substring $w_i \cdots w_j \in L_k^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter.

Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

A factor of a word $w_1 \cdots w_r \in L_k^*$ is a substring $w_i \cdots w_j \in L_k^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter.

123235443513

LOWER BOUNDS

Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

A factor of a word $w_1 \cdots w_r \in L_k^*$ is a substring $w_i \cdots w_j \in L_k^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter.

123235443513

LOWER BOUNDS

Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

A factor of a word $w_1 \cdots w_r \in L_k^*$ is a substring $w_i \cdots w_j \in L_k^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter.

123235443513

LOWER BOUNDS

Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

A factor of a word $w_1 \cdots w_r \in L_k^*$ is a substring $w_i \cdots w_j \in L_k^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter.

123235443513

LOWER BOUNDS

Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

A factor of a word $w_1 \cdots w_r \in L_k^*$ is a substring $w_i \cdots w_j \in L_k^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter.

123235443513

LOWER BOUNDS

Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

A factor of a word $w_1 \cdots w_r \in L_k^*$ is a substring $w_i \cdots w_j \in L_k^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter.

123235443513

LOWER BOUNDS

Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

A factor of a word $w_1 \cdots w_r \in L_k^*$ is a substring $w_i \cdots w_j \in L_k^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter.

123235443513

Disjoint factors do not overlap in the word.

LOWER BOUNDS

Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

A factor of a word $w_1 \cdots w_r \in L_k^*$ is a substring $w_i \cdots w_j \in L_k^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter.

123235443513

Disjoint factors do not overlap in the word.

Does the word have all the k factors, mutually disjoint?

LOWER BOUNDS

Disjoint Factors

Let L_k be the alphabet consisting of the letters $\{1, 2, \dots, k\}$.

A factor of a word $w_1 \cdots w_r \in L_k^*$ is a substring $w_i \cdots w_j \in L_k^*$, with $1 \leq i < j \leq r$, which starts and ends with the same letter.

123235443513

Disjoint factors do not overlap in the word.

Does the word have all the k factors, mutually disjoint?

Parameter: k

LOWER BOUNDS

A $k! \cdot \mathcal{O}(n)$ algorithm is immediate.

LOWER BOUNDS

A $k! \cdot \mathcal{O}(n)$ algorithm is immediate.

A $2^k \cdot p(n)$ algorithm can be obtained by Dynamic Programming.

LOWER BOUNDS

A $k! \cdot \mathcal{O}(n)$ algorithm is immediate.

A $2^k \cdot p(n)$ algorithm can be obtained by Dynamic Programming.

Let t be the number of instances input to the composition algorithm.
Again, the non-trivial case is when $t < 2^k$.

LOWER BOUNDS

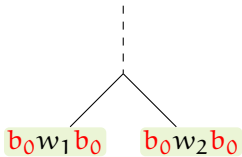
A $k! \cdot \mathcal{O}(n)$ algorithm is immediate.

A $2^k \cdot p(n)$ algorithm can be obtained by Dynamic Programming.

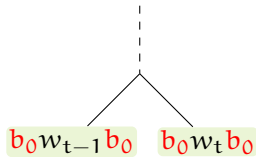
Let t be the number of instances input to the composition algorithm.
Again, the non-trivial case is when $t < 2^k$.

Let $w_1, w_2, \dots, w_t$ be words over L_k^* .

Leaf Nodes

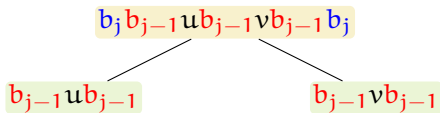


...



LOWER BOUNDS

Level j .



LOWER BOUNDS

Claim

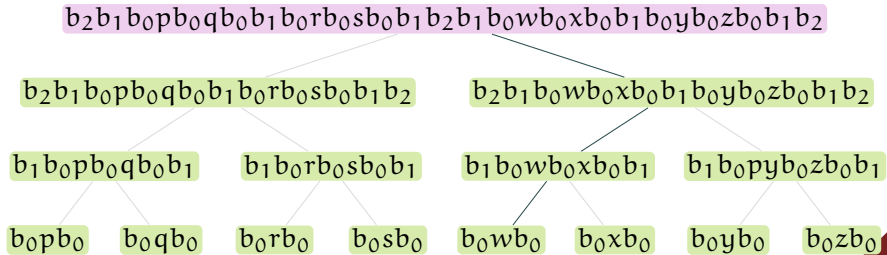
The composed word has all the $2k$ disjoint factors



at least one of the input instances has all the k disjoint factors.

LOWER BOUNDS

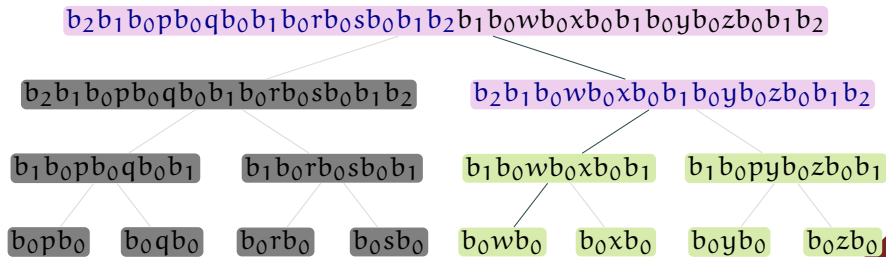
Proof of Correctness



Input words: p, q, r, s, w, x, y, z .

LOWER BOUNDS

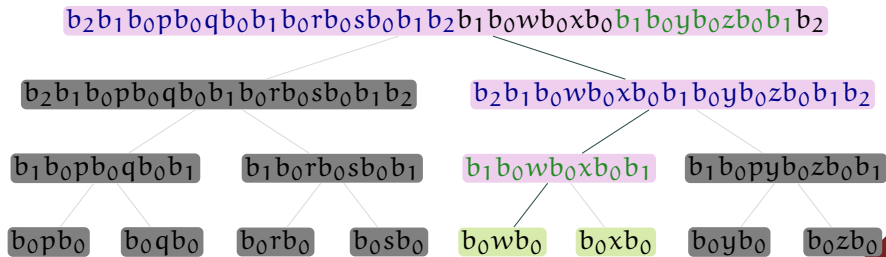
Proof of Correctness



Input words: p, q, r, s, w, x, y, z .

LOWER BOUNDS

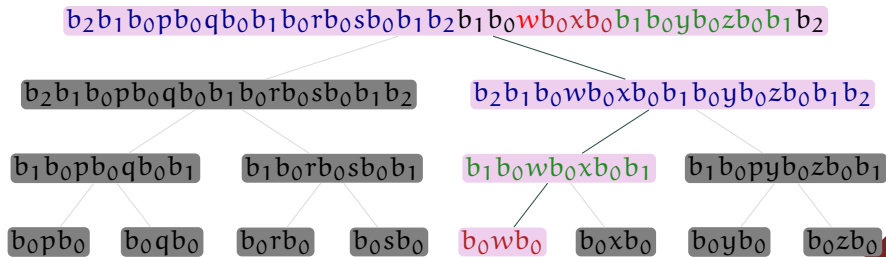
Proof of Correctness



Input words: p, q, r, s, w, x, y, z .

LOWER BOUNDS

Proof of Correctness



Input words: p, q, r, s, w, x, y, z .

LOWER BOUNDS

Vertex-Disjoint Cycles

Input: $G = (V, E)$

LOWER BOUNDS

Vertex-Disjoint Cycles

Input: $G = (V, E)$

Question: Are there k vertex-disjoint cycles?

LOWER BOUNDS

Vertex-Disjoint Cycles

Input: $G = (V, E)$

Question: Are there k vertex-disjoint cycles?

Parameter: k .

LOWER BOUNDS

Vertex-Disjoint Cycles

Input: $G = (V, E)$

Question: Are there k vertex-disjoint cycles?

Parameter: k .

Related problems:

FVS, has a $\mathcal{O}(k^2)$ kernel.

Edge-Disjoint Cycles, has a $\mathcal{O}(k^2 \log^2 k)$ kernel.

LOWER BOUNDS

Vertex-Disjoint Cycles

Input: $G = (V, E)$

Question: Are there k vertex-disjoint cycles?

Parameter: k .

Related problems:

FVS, has a $\mathcal{O}(k^2)$ kernel.

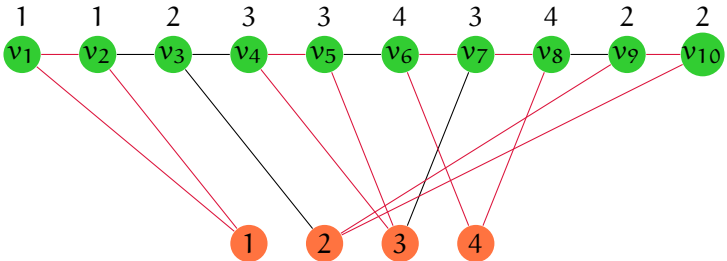
Edge-Disjoint Cycles, has a $\mathcal{O}(k^2 \log^2 k)$ kernel.

LOWER BOUNDS

In contrast, Disjoint Factors transforms into Vertex-Disjoint Cycles in polynomial time.

LOWER BOUNDS

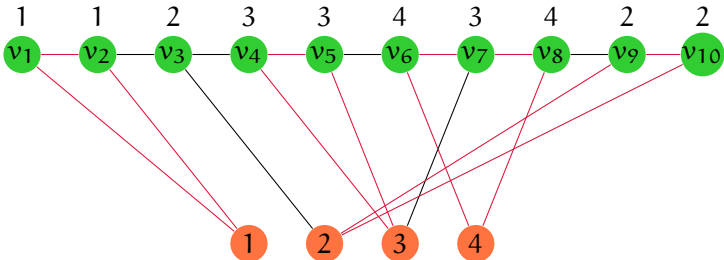
$$w = 1123343422$$



Disjoint Factors $\preceq_{\text{ppt}}$ Disjoint Cycles

LOWER BOUNDS

$$w = 1123343422$$



w has all k disjoint factors $\iff G_w$ has k vertex-disjoint cycles.

LOWER BOUNDS

Colors and IDs

STORIES FOR ANOTHER DAY

Colors and IDs

(Advanced techniques for the design of composition algorithms.)

STORIES FOR ANOTHER DAY

Finer lower bounds on kernel size.

STORIES FOR ANOTHER DAY

Dealing with “hardness” of polynomial kernelization.

STORIES FOR ANOTHER DAY

Dealing with “hardness” of polynomial kernelization.

(Many poly kernels, approximative kernels, randomized kernels, etc.)

STORIES FOR ANOTHER DAY

For the many problems whose kernelization complexity is unknown,
prove positive or negative results.

STORIES FOR ANOTHER DAY

For the many problems whose kernelization complexity is unknown,
prove positive or negative results.

(Odd Cycle Traversal, Feedback Vertex Set on Directed Graphs)

STORIES FOR ANOTHER DAY

Discover implications of the AND conjecture, and other ways of showing lower bounds.

STORIES FOR ANOTHER DAY

FIN