Structural Parameters

Algorithmics, 186.814, VU 6.0

Jan Dreier

Recap

- A parameterized problem is a problem whose instances come with a numerical parameter.
  - For example, an instance of a parameterized graph problem could look like (G, k).

Recap

- A parameterized problem is a problem whose instances come with a numerical parameter.
  - For example, an instance of a parameterized graph problem could look like (G, k).
- For an NP-complete parameterized problem $\mathcal{P}$:
  1. $\mathcal{P}$ could be FPT, meaning that it can be solved in time $f(k) \cdot \text{poly}(n)$ (FPT algorithm),

Recap

- A parameterized problem is a problem whose instances come with a numerical parameter.
  - For example, an instance of a parameterized graph problem could look like (G, k).
- For an NP-complete parameterized problem $\mathcal{P}$:
  1. $\mathcal{P}$ could be FPT, meaning that it can be solved in time $f(k) \cdot \text{poly}(n)$ (FPT algorithm),
  2. $\mathcal{P}$ could be W[1]-complete, W[2]-complete, ..., meaning that it cannot be solved by an FPT algorithm but can be solved in time $n^{f(k)}$ (XP algorithm),

Recap

- A parameterized problem is a problem whose instances come with a numerical parameter.
  - For example, an instance of a parameterized graph problem could look like (G, k).
- For an NP-complete parameterized problem $\mathcal{P}$:
  1. $\mathcal{P}$ could be FPT, meaning that it can be solved in time $f(k) \cdot \text{poly}(n)$ (FPT algorithm),
  2. $\mathcal{P}$ could be W[1]-complete, W[2]-complete, ..., meaning that it cannot be solved by an FPT algorithm but can be solved in time $n^{f(k)}$ (XP algorithm),
  3. $\mathcal{P}$ could be para-$\mathbf{NP}$-hard, meaning that it can't even be solved in time $n^{f(k)}$.

Recap

- A parameterized problem is a problem whose instances come with a numerical parameter.
  - For example, an instance of a parameterized graph problem could look like (G, k).
- For an NP-complete parameterized problem $\mathcal{P}$:
  1. $\mathcal{P}$ could be FPT, meaning that it can be solved in time $f(k) \cdot \text{poly}(n)$ (FPT algorithm),
  2. $\mathcal{P}$ could be W[1]-complete, W[2]-complete, ..., meaning that it cannot be solved by an FPT algorithm but can be solved in time $n^{f(k)}$ (XP algorithm),
  3. $\mathcal{P}$ could be para-$\mathbf{NP}$-hard, meaning that it can't even be solved in time $n^{f(k)}$.
- Any parameterized problem $\mathcal{P} \in$ FPT can be kernelized:
  - A kernelization algorithm runs in polynomial time and shrinks the instance until its size is bounded by a function $h(k)$
  - $\mathcal{P}$ can either admit a polykernel (h is a polynomial function), or not.

One More Kernelization Example

Consider the following problem:

Point Line Cover

Instance: A set of n points in the plane and an integer k.

Parameter: k.

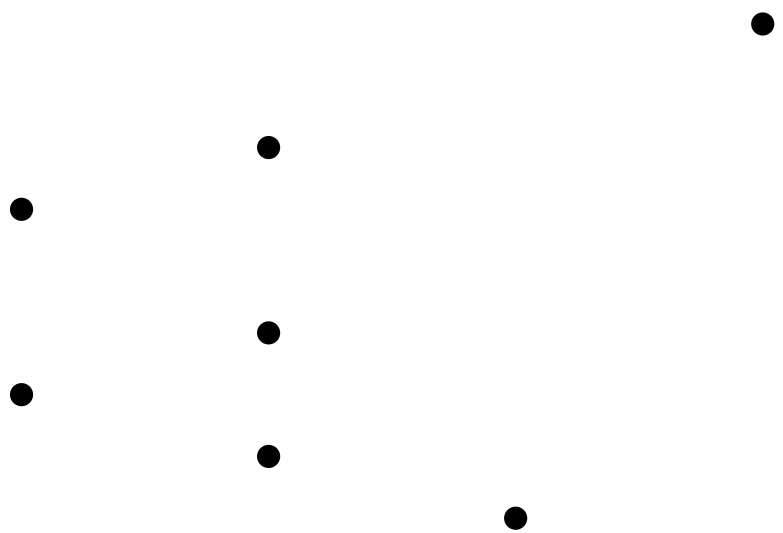
Question: Do there exist at most k lines on the plane that contain all the input points?

One More Kernelization Example

Consider the following problem:

> Point Line Cover
>
> *Instance*: A set of n points in the plane and an integer k.
>
> *Parameter*: k.
>
> *Question*: Do there exist at most k lines on the plane that contain all the input points?

Example: $k = 3$

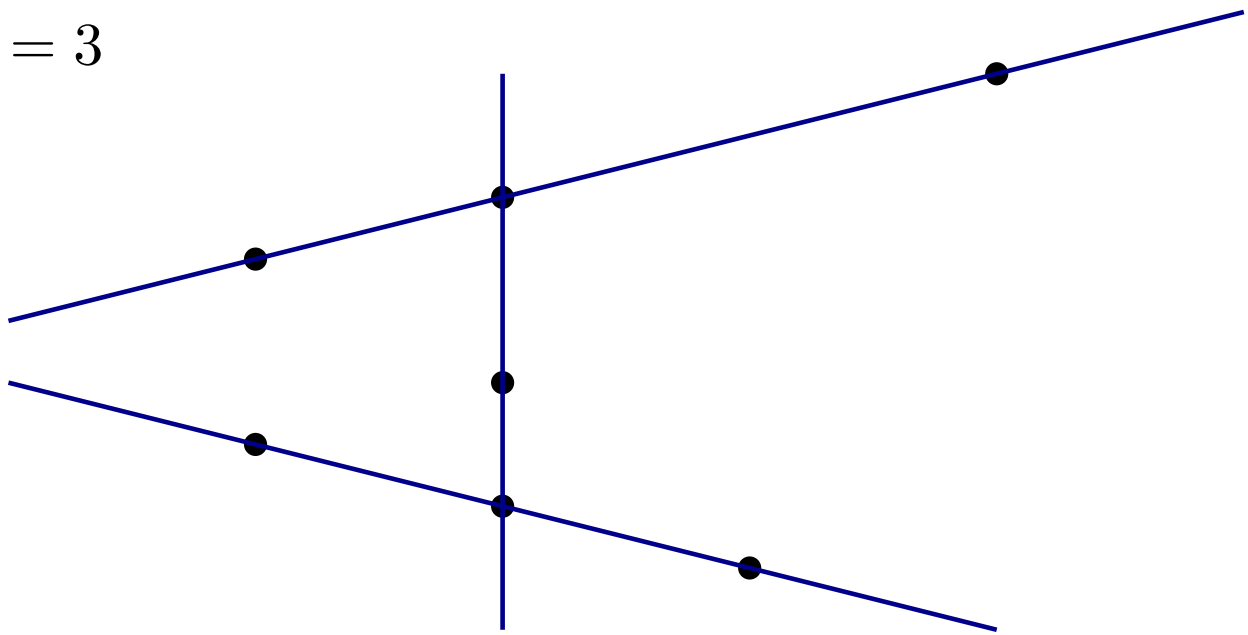


One More Kernelization Example

Consider the following problem:

> Point Line Cover
>
> *Instance*: A set of n points in the plane and an integer k.
>
> *Parameter*: k.
>
> *Question*: Do there exist at most k lines on the plane that contain all the input points?

Example: $k = 3$



Polynomial Kernel for Point Line Cover

1. If there are at least 2 points, then one needs to consider only n^2 possible lines (between pairs of points).
   - Never helps to use a line that hits exactly 1 point.

Polynomial Kernel for Point Line Cover

1. If there are at least 2 points, then one needs to consider only n^2 possible lines (between pairs of points).
   - Never helps to use a line that hits exactly 1 point.
2. What if a line hits at least $k+1$ points?
   - Then this line **must** be in the solution. Otherwise, we'd need at least $k+1$ lines to hit these points.

   Rule 1: If there is a line that hits at least $k+1$ points, then delete these points and set $k := k-1$.

Polynomial Kernel for Point Line Cover

1. If there are at least 2 points, then one needs to consider only n^2 possible lines (between pairs of points).
   - Never helps to use a line that hits exactly 1 point.
2. What if a line hits at least $k+1$ points?
   - Then this line **must** be in the solution. Otherwise, we'd need at least $k+1$ lines to hit these points.

   Rule 1: If there is a line that hits at least $k+1$ points, then delete these points and set $k := k-1$.
3. What if we cannot apply **Rule 1**?
   - Each remaining possible line hits at most k points.
   - Then if there's more than k^2 points, we can never hit all of them with k lines.
   - Either we already have a k^2 kernel, or we have a no-instance.

Polynomial Kernel for Point Line Cover

1. If there are at least 2 points, then one needs to consider only n^2 possible lines (between pairs of points).
   - Never helps to use a line that hits exactly 1 point.
2. What if a line hits at least $k+1$ points?
   - Then this line **must** be in the solution. Otherwise, we'd need at least $k+1$ lines to hit these points.

   Rule 1: If there is a line that hits at least $k+1$ points, then delete these points and set $k := k-1$.
3. What if we cannot apply **Rule 1**?
   - Each remaining possible line hits at most k points.
   - Then if there's more than k^2 points, we can never hit all of them with k lines.
   - Either we already have a k^2 kernel, or we have a no-instance.

- What about an FPT algorithm?
  - First kernelize, then apply brute-force branching.

Today's Topic: Parameter Choice

So far, we always considered problems of the form "Does I have [...] of size at least or at most k?" and parameterized by the **solution size** k.

- Vertex Cover, Independent Set, d-Hitting Set, Coloring, ...

Today's Topic: Parameter Choice

So far, we always considered problems of the form "Does I have [...] of size at least or at most k?" and parameterized by the **solution size** k.

- Vertex Cover, Independent Set, d-Hitting Set, Coloring, ...

But what about cases where this simply isn't possible?

- Independent Set is W[1]-hard parameterized by solution size.
- Coloring is para-**NP**-hard parameterized by the number of colors.
- Problems like SAT and Hamiltonian Cycle simply don't include any numbers we can choose as parameters.
- What if the problem-specific parameter is simply too big to be of use?

Today's Topic: Parameter Choice

So far, we always considered problems of the form "Does I have [...] of size at least or at most k?" and parameterized by the **solution size** k.

- Vertex Cover, Independent Set, d-Hitting Set, Coloring, ...

But what about cases where this simply isn't possible?

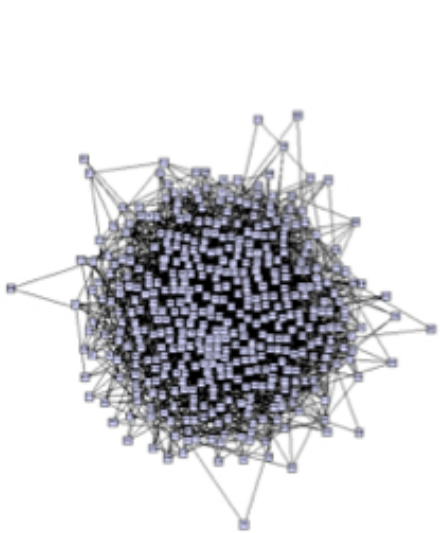
- Independent Set is W[1]-hard parameterized by solution size.
- Coloring is para-**NP**-hard parameterized by the number of colors.
- Problems like SAT and Hamiltonian Cycle simply don't include any numbers we can choose as parameters.
- What if the problem-specific parameter is simply too big to be of use?

Different parameterization: exploit the structure of instances! For problems "Does I have [...]?" parameterize by structure of I.

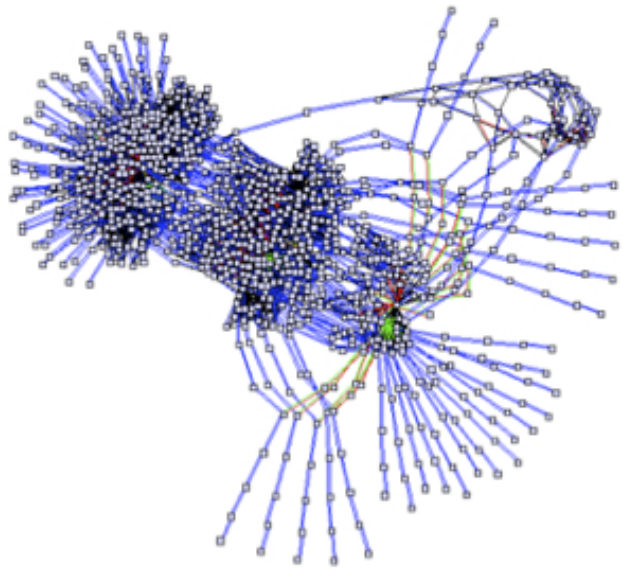
Structure in Real-World Problems

Real-world instances are usually *well-structured*

- *well-structured* could mean many things, depending on the problem and setting



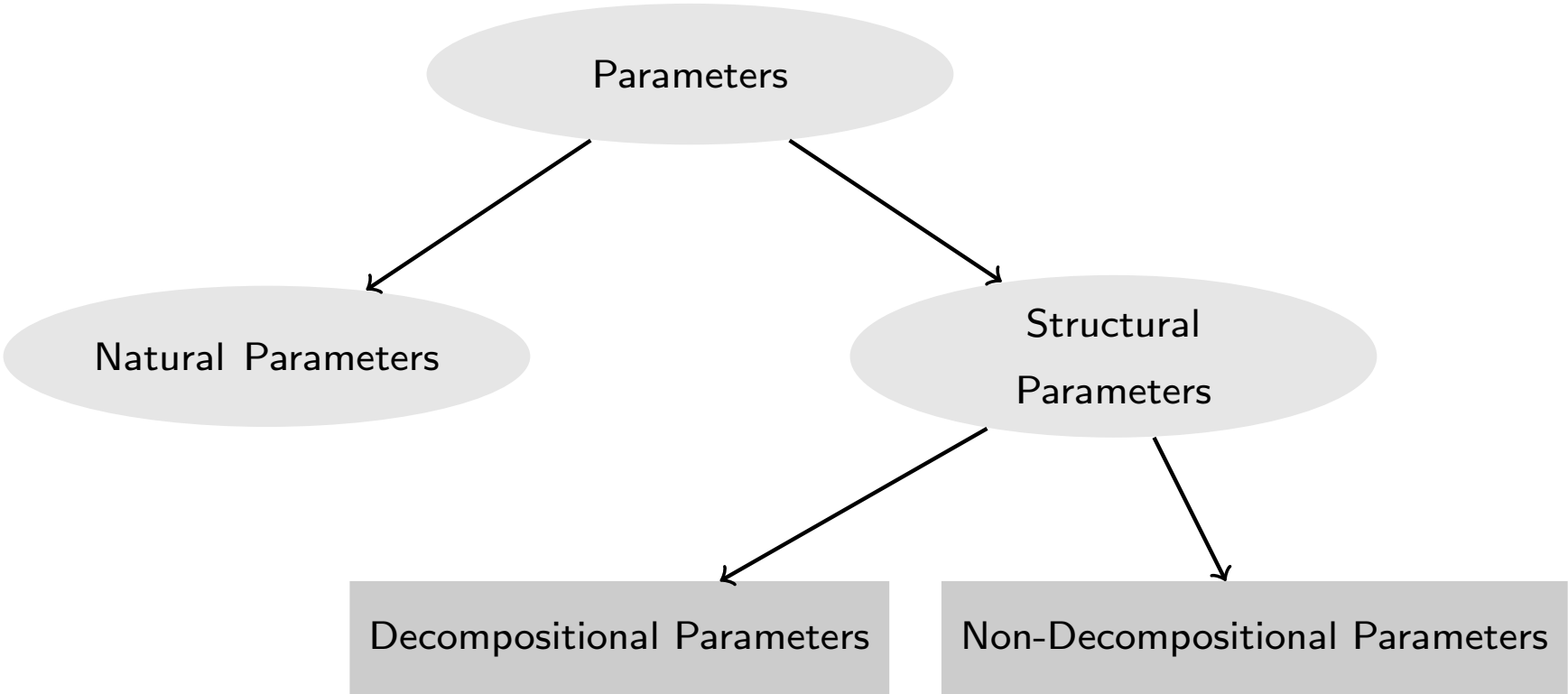
Random instance



Real-world instance

Parameters can be used to exploit structure for solving NP-hard problems.

An Overview of Parameterizations



- **Natural Parameters:** what we did until now
- **Decompositional Parameters:** based on graph decompositions, will be covered in next block
- **Non-Decompositional Parameters:** topic of this lecture

This Lecture

Today we will:

- try out a few simple structural parameters
- practice FPT algorithms
- practice kernelization

This Lecture

Today we will:

- try out a few simple structural parameters
- practice FPT algorithms
- practice kernelization

Some forms of structure are more algorithmically useful than others!

Parameterizing by Maximum Degree

Consider the maximum degree of the graph. Can this be used as a structural parameter?

- Well-defined, easy to quantify
- Easy to compute

Parameterizing by Maximum Degree

Consider the maximum degree of the graph. Can this be used as a structural parameter?

- Well-defined, easy to quantify
- Easy to compute
- Most classical **NP**-hard problems remain **NP**-hard on bounded degree graphs
  - (unparameterized) Vertex Cover, Independent Set, Dominating Set, Hamiltonian Cycle, ...

$\rightarrow$ On its own, degree is mostly not considered a good structural parameter.

Parameterizing by Maximum Degree + Solution Size

Remember that we solved Independent Set on graphs of degree 3 in time $4^k \cdot \text{poly}(n)$ (either a vertex or one of its neighbors needs to be in a solution).

Parameterizing by Maximum Degree + Solution Size

Remember that we solved Independent Set on graphs of degree 3 in time $4^k \cdot \text{poly}(n)$ (either a vertex or one of its neighbors needs to be in a solution).

You may even use the sum of two numbers as a parameter.

Independent Set using Degree and Solution Size

Instance: A graph G of degree at most d and an integer k.

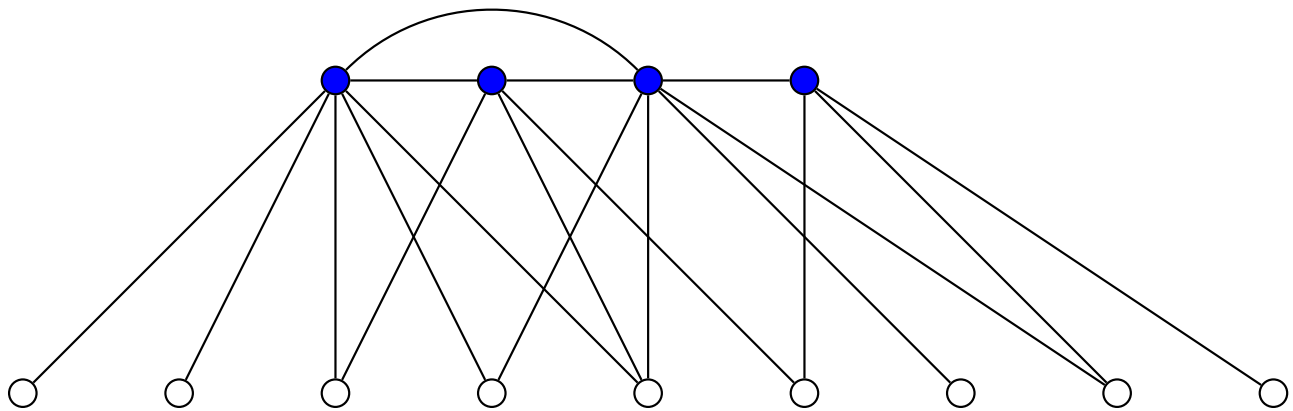
Parameter: $k + d$.

Task: Choose at least k vertices in G which are pairwise *independent*.

This problem can be solved in time $(d+1)^k \cdot \text{poly}(n)$ using our standard branching algorithm. It is therefore FPT.

Parameterizing by Vertex Cover

Having a vertex cover in a graph does provide a lot of insight into the structure of the graph.
Recall: A vertex set X is a vertex cover of a graph G iff each edge is incident to at least one vertex from X.



Example: vertex cover marked in blue.

Using Vertex Cover

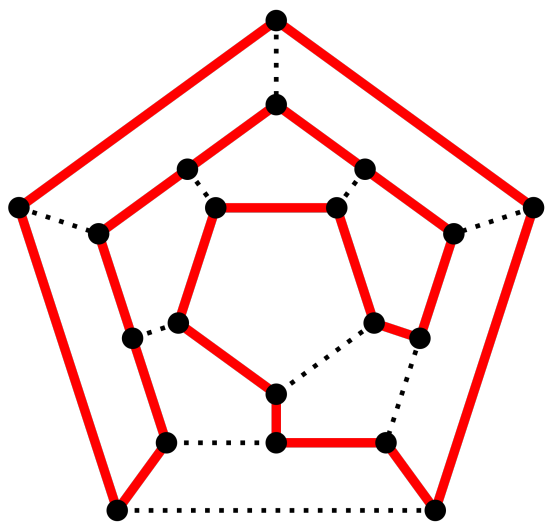
So, for now assume that we are given a vertex cover "for free" together with the input. Can we use its size as a parameter to solve NP-hard problems?

Using Vertex Cover

So, for now assume that we are given a vertex cover "for free" together with the input. Can we use its size as a parameter to solve NP-hard problems?

Hamiltonian Cycle
Instance: A graph G.
Question: Does there exist a *Hamiltonian cycle* in G?

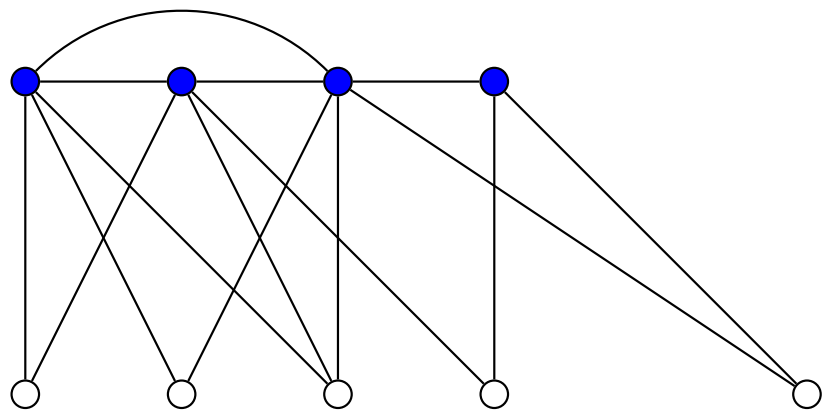
Hamiltonian cycle: A cycle which goes through each vertex in G.



Using Vertex Cover for Hamiltonian Cycle

So, the parameterized problem we are formally looking at is this:

Hamiltonian Cycle Using Vertex Cover
Instance: A graph G and a vertex cover X of G.
Parameter: $k = \|X\|$.
Question: Does there exist a *Hamiltonian cycle* in G?



Is this problem FPT?

A Simple Lemma

Lemma

Any cycle in G only contains at most $k = |X|$ vertices outside of the vertex cover X.

Proof.

- There cannot be an edge between any two vertices outside of X.
- So, a cycle cannot contain two consecutive vertices outside of X.
- Hence at least one half (rounded up) of the vertices in a cycle must be in X.
- Since $|X| = k$, we have that the cycle contains at most k vertices outside of X. □

Using Vertex Cover for Hamiltonian Cycle

Hamiltonian Cycle Using Vertex Cover

Instance: A graph G and a vertex cover X of G.

Parameter: $k = |X|$.

Question: Does there exist a *Hamiltonian cycle* in G?

A trivial kernelization algorithm:

1. If $|V| > 2k$, answer "no".
2. Otherwise we have a kernel with $2k$ vertices and at most k^2 edges:

Using Vertex Cover for Hamiltonian Cycle

Hamiltonian Cycle Using Vertex Cover

Instance: A graph G and a vertex cover X of G.

Parameter: $k = |X|$.

Question: Does there exist a *Hamiltonian cycle* in G?

A trivial kernelization algorithm:

1. If $|V| > 2k$, answer "no".
2. Otherwise we have a kernel with $2k$ vertices and at most k^2 edges:

Correctness: By the Lemma (repeated below), it follows that there cannot be a cycle which contains more than $2k$ vertices.

Lemma

Any cycle in G only contains at most k vertices outside of the vertex cover X.

Recap

Theorem

Hamiltonian Cycle Using Vertex Cover *admits a polynomial kernel.*

- Hamiltonian Cycle can be solved efficiently **if you are given a small vertex cover** as part of your input.
- Many other problems can be considered, and many forms of structure other than vertex cover can be exploited.
  - $\rightarrow$ when solving a hard problem, think whether you can exploit the structure of your instances
- Let's try out some other problems parameterized by vertex cover.

Chromatic Number Parameterized by Vertex Cover

Chromatic Number Using Vertex Cover

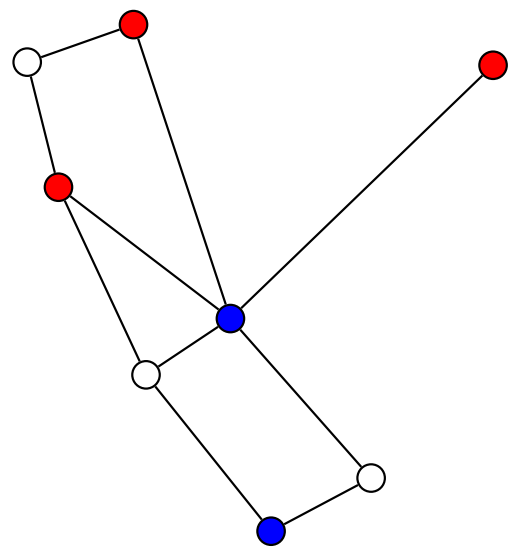
Instance: A graph G, an integer q and a vertex cover X of G.

Parameter: $k = |X|$.

Question: Does there exist a proper q-coloring of G?

Proper q-coloring: a coloring of the vertices using at most q colors; neighbors cannot have the same color.

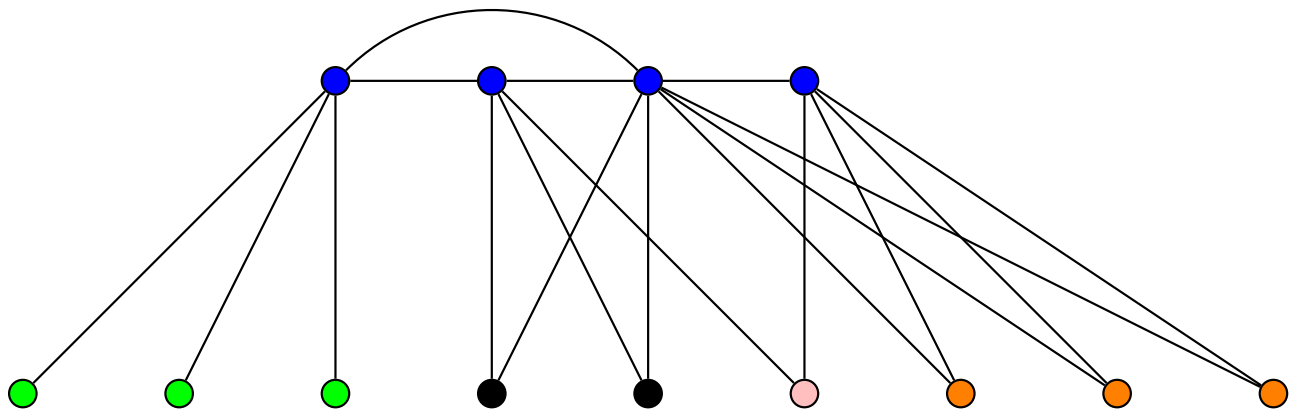
Example with 3 colors:



Vertex Types

When using a vertex cover X, it is often useful to separate vertices outside of X into types.

- two vertices have the same type iff they have the same neighbors in X.

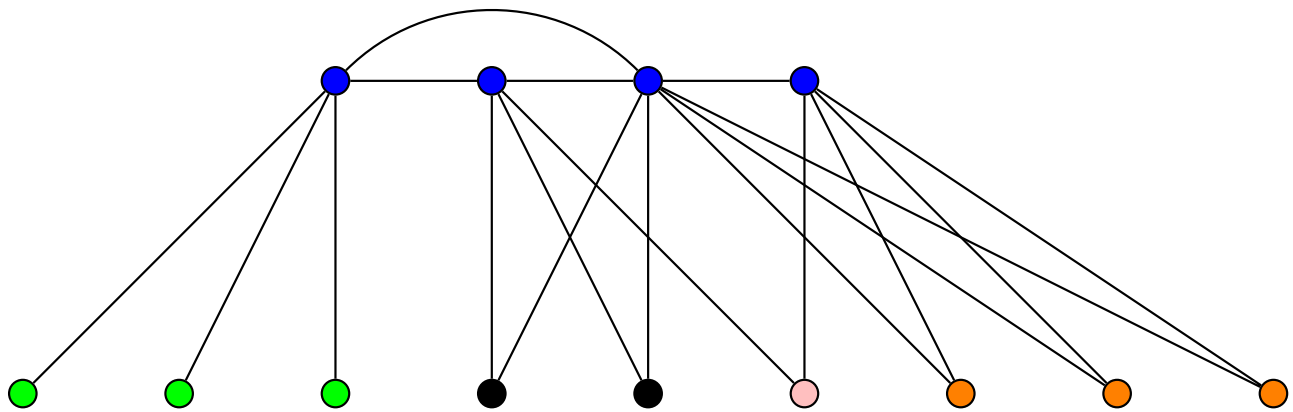


Example: vertex cover depicted in blue, and each of the 4 types identified by a unique color.

Vertex Types

When using a vertex cover X, it is often useful to separate vertices outside of X into types.

- two vertices have the same type iff they have the same neighbors in X.



Example: vertex cover depicted in blue, and each of the 4 types identified by a unique color.

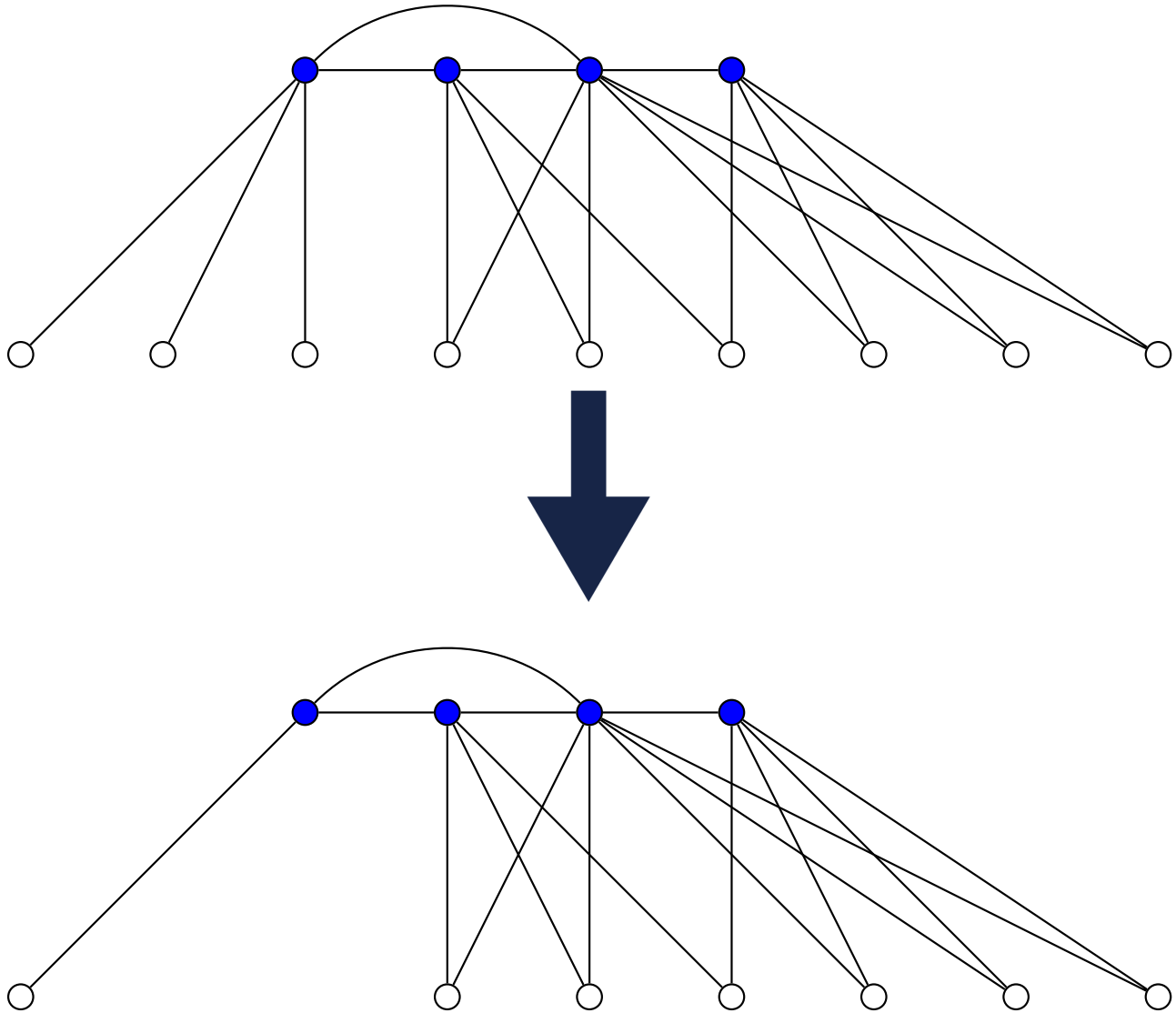
Observe: A graph can only have at most 2^k types, where $k = |X|$.

Using Types for Chromatic Number

Consider the following **reduction rule:**

- If a type T contains more than 1 vertex, delete all but a single vertex in that type.

Visualization of reduction rule



Using Types for Chromatic Number

Why is this rule **safe**?

- If you have a q-coloring of the original graph G, then you can use the same coloring for the subgraph G' obtained by reducing the type T.
- On the other hand, if you have a q-coloring of the subgraph G', then you can duplicate the color used for T to also color G.

Chromatic Number – Wrapping Up

Chromatic Number Parameterized by Vertex Cover

Instance: A graph G, an integer q and a vertex cover X of G.

Parameter: $k = |X|$.

Question: Does there exist a proper q-coloring of G?

- After applying reduction rule, we have at most 1 vertex in each type.
- Since there are at most 2^k types, this means that G has at most $2^k + k$ vertices.
  - At most 1 vertex in each type, and k vertices in the vertex cover X.

Chromatic Number – Wrapping Up

Chromatic Number Parameterized by Vertex Cover

Instance: A graph G, an integer q and a vertex cover X of G.

Parameter: $k = |X|$.

Question: Does there exist a proper q-coloring of G?

- After applying reduction rule, we have at most 1 vertex in each type.
- Since there are at most 2^k types, this means that G has at most $2^k + k$ vertices.
  - At most 1 vertex in each type, and k vertices in the vertex cover X.
- If the number of colors $q > 2^k + k$, we can answer "yes" now.

Chromatic Number – Wrapping Up

Chromatic Number Parameterized by Vertex Cover

Instance: A graph G, an integer q and a vertex cover X of G.

Parameter: $k = |X|$.

Question: Does there exist a proper q-coloring of G?

- After applying reduction rule, we have at most 1 vertex in each type.
- Since there are at most 2^k types, this means that G has at most $2^k + k$ vertices.
  - At most 1 vertex in each type, and k vertices in the vertex cover X.
- If the number of colors $q > 2^k + k$, we can answer "yes" now.
- Both parts of the input (q and G) are now bounded by a function of k; we have a (non-polynomial) kernel, and so we conclude:

Chromatic Number – Wrapping Up

Chromatic Number Parameterized by Vertex Cover

Instance: A graph G, an integer q and a vertex cover X of G.

Parameter: $k = |X|$.

Question: Does there exist a proper q-coloring of G?

- After applying reduction rule, we have at most 1 vertex in each type.
- Since there are at most 2^k types, this means that G has at most $2^k + k$ vertices.
  - At most 1 vertex in each type, and k vertices in the vertex cover X.
- If the number of colors $q > 2^k + k$, we can answer "yes" now.
- Both parts of the input (q and G) are now bounded by a function of k; we have a (non-polynomial) kernel, and so we conclude:

Theorem

Chromatic Number Using Vertex Cover *is FPT.*

Chromatic Number – Wrapping Up

Chromatic Number Parameterized by Vertex Cover

Instance: A graph G, an integer q and a vertex cover X of G.

Parameter: $k = |X|$.

Question: Does there exist a proper q-coloring of G?

- After applying reduction rule, we have at most 1 vertex in each type.
- Since there are at most 2^k types, this means that G has at most $2^k + k$ vertices.
  - At most 1 vertex in each type, and k vertices in the vertex cover X.
- If the number of colors $q > 2^k + k$, we can answer "yes" now.
- Both parts of the input (q and G) are now bounded by a function of k; we have a (non-polynomial) kernel, and so we conclude:

Theorem

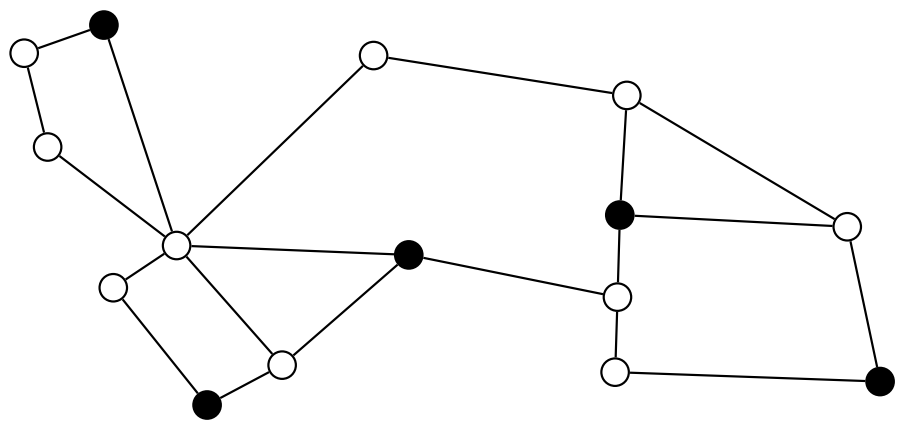
Chromatic Number Using Vertex Cover *is FPT.*

Remark: Chromatic Number Using Vertex Cover does not admit a polynomial kernel (unless the polynomial hierarchy collapses).

Independent Set Using Vertex Cover

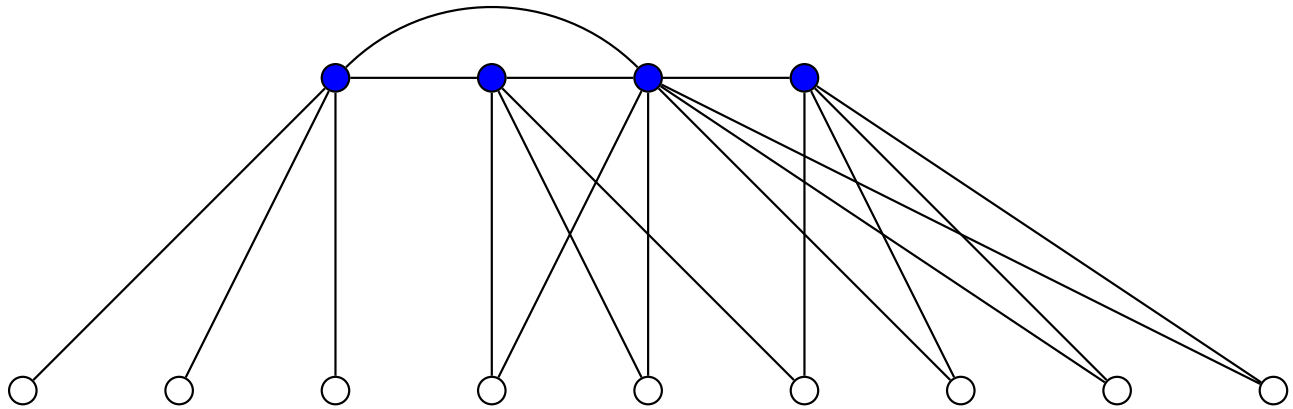
Independent Set Using Vertex Cover
Instance: A graph G, an integer p and a vertex cover X of G.
Parameter: $k = \|X\|$.
Question: Does there exist an independent set of size at least p in G?

Independent Set: a vertex set I such that there are no edges between vertices in I.



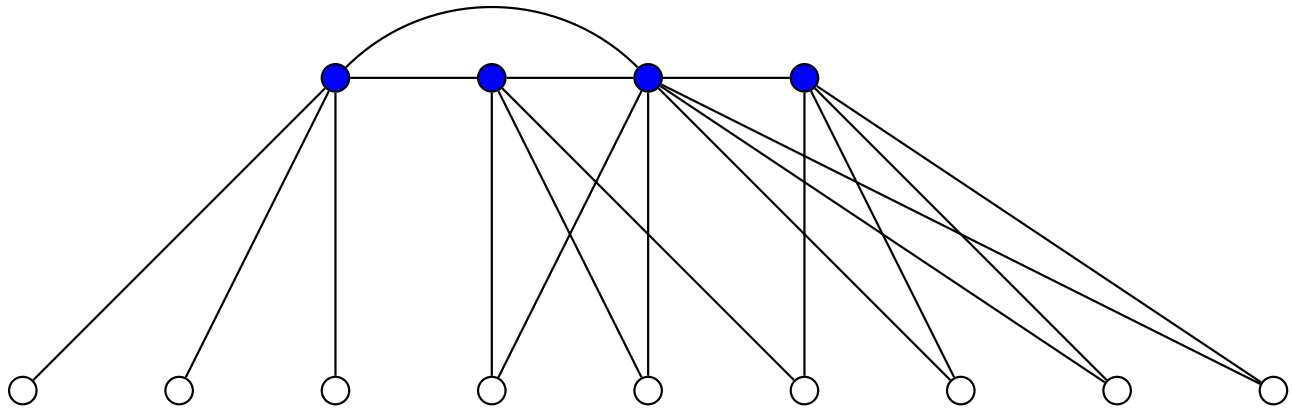
Is Independent Set Using Vertex Cover FPT?

Solving Independent Set Using Vertex Cover



- Several approaches can be used (including types).
- We will solve the problem using **branching**.

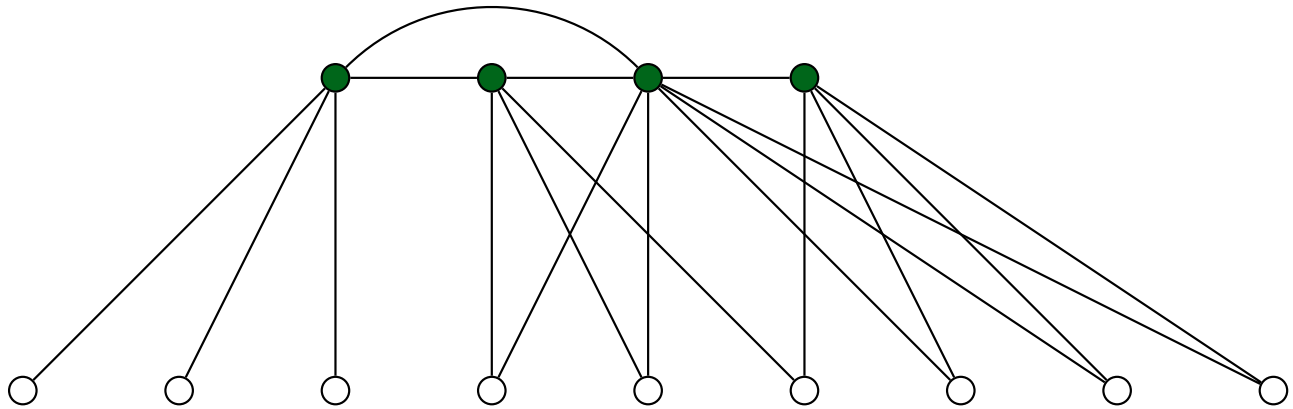
Solving Independent Set Using Vertex Cover



- Several approaches can be used (including types).
- We will solve the problem using **branching**.

Recall that X is the vertex cover (of size k) and I is a sought-after independent set of size at least p.
Each vertex $x \in X$ can either be in I ($x \in I$) or be outside of I ($x \notin I$).
$\rightarrow$ there are exactly 2^k possibilities of how I intersects with X.

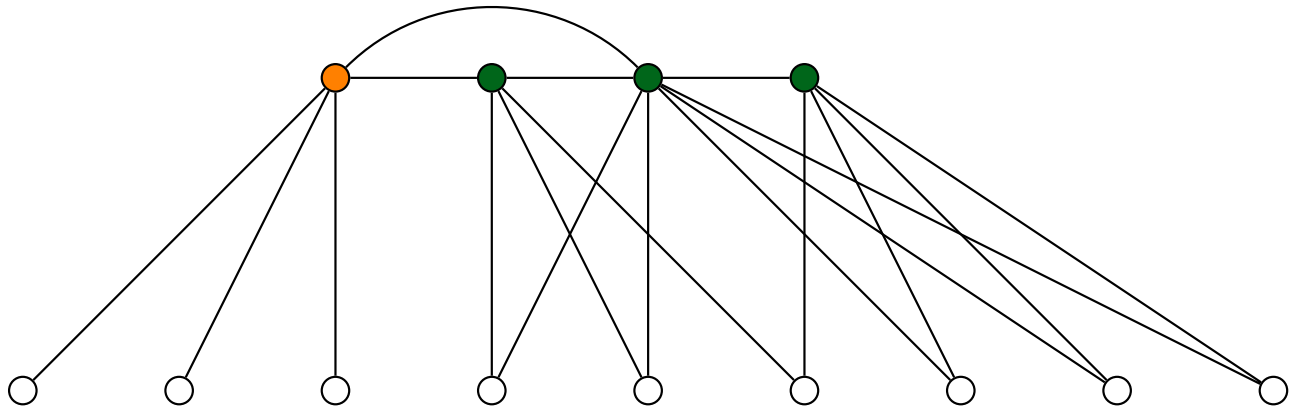
Solving Independent Set Using Vertex Cover



- Several approaches can be used (including types).
- We will solve the problem using **branching**.

Recall that X is the vertex cover (of size k) and I is a sought-after independent set of size at least p.
Each vertex $x \in X$ can either be in I ($x \in I$) or be outside of I ($x \notin I$).
$\rightarrow$ there are exactly 2^k possibilities of how I intersects with X.

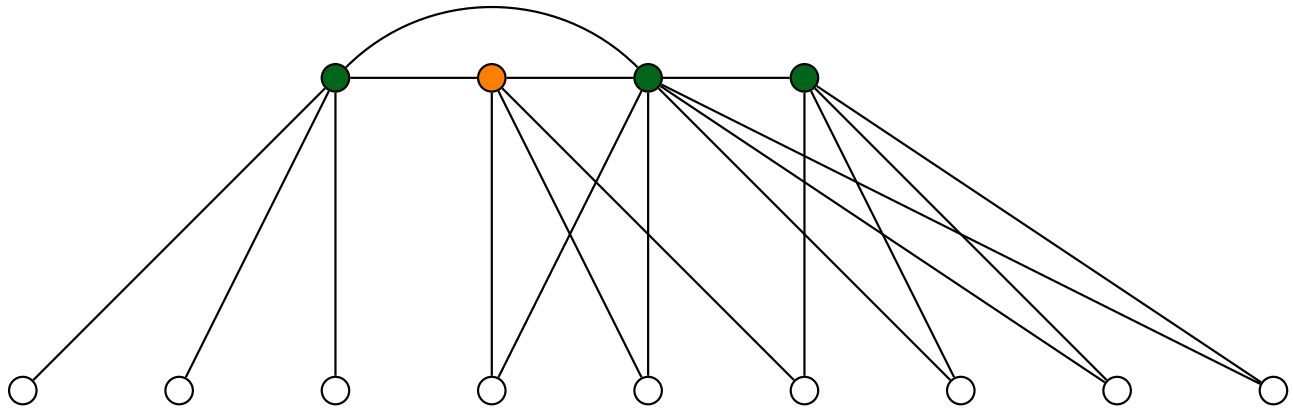
Solving Independent Set Using Vertex Cover



- Several approaches can be used (including types).
- We will solve the problem using **branching**.

Recall that X is the vertex cover (of size k) and I is a sought-after independent set of size at least p.
Each vertex $x \in X$ can either be in I ($x \in I$) or be outside of I ($x \notin I$).
$\rightarrow$ there are exactly 2^k possibilities of how I intersects with X.

Solving Independent Set Using Vertex Cover

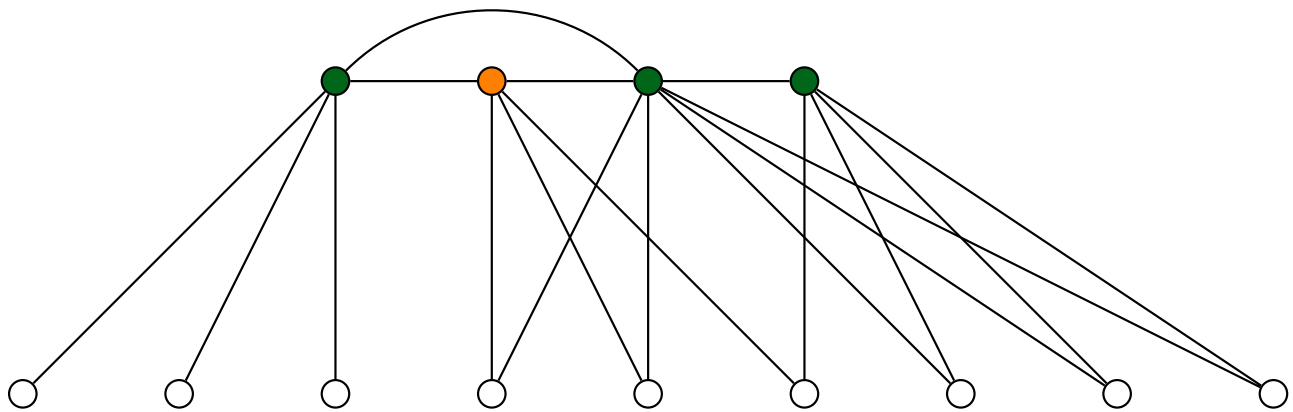


- Several approaches can be used (including types).
- We will solve the problem using **branching**.

Recall that X is the vertex cover (of size k) and I is a sought-after independent set of size at least p.
Each vertex $x \in X$ can either be in I ($x \in I$) or be outside of I ($x \notin I$).
$\rightarrow$ there are exactly 2^k possibilities of how I intersects with X.

For each branch, we:

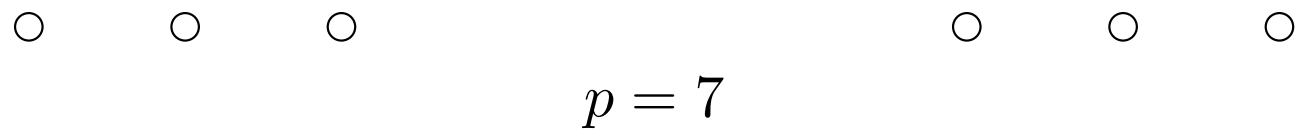
1. delete $X \cap I$ and all of their neighbors (we "guessed" they're in the independent set)
2. reduce p by $|X \cap I|$ (we added this many vertices into I)
3. delete the remaining vertices in X (we "guessed" they're not in the independent set)
4. all the remaining vertices can go into I. Check whether the branch satisfies $|I| \geq p$.



$p = 8$

For each branch, we:

1. delete $X \cap I$ and all of their neighbors (we "guessed" they're in the independent set)
2. reduce p by $|X \cap I|$ (we added this many vertices into I)
3. delete the remaining vertices in X (we "guessed" they're not in the independent set)
4. all the remaining vertices can go into I. Check whether the branch satisfies $|I| \geq p$.

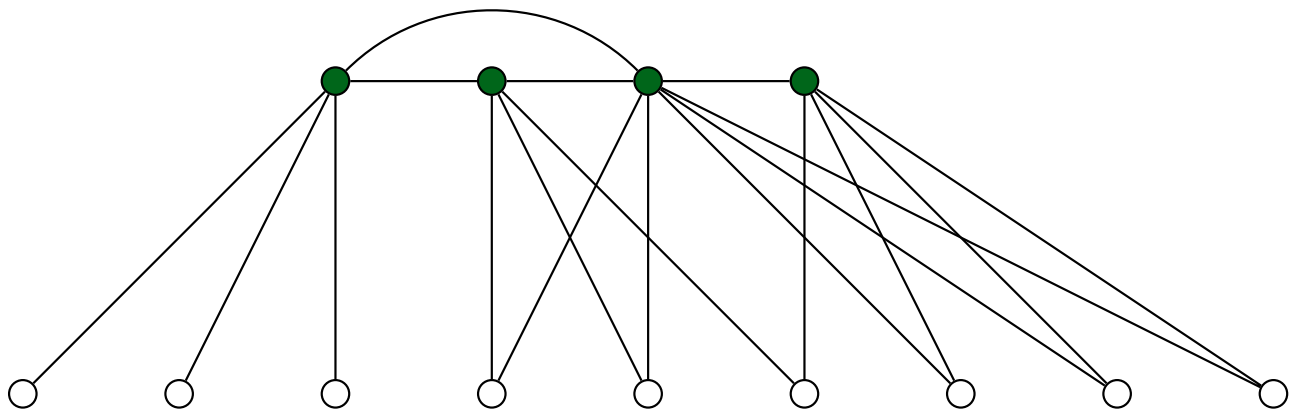


$$p = 7$$

Can only add 6 more vertices into $I \rightarrow$ not a solution branch.

For each branch, we:

1. delete $X \cap I$ and all of their neighbors (we "guessed" they're in the independent set)
2. reduce p by $|X \cap I|$ (we added this many vertices into I)
3. delete the remaining vertices in X (we "guessed" they're not in the independent set)
4. all the remaining vertices can go into I. Check whether the branch satisfies $|I| \geq p$.



$p = 8$

This is a solution branch.

Independent Set Using Vertex Cover – Summing Up

The above gives a $2^k \cdot n$ algorithm for the problem.

- observe that we are only branching **once** – at the beginning.

Theorem

Independent Set Using Vertex Cover *is FPT.*

Remark: in case you are wondering, the problem does have a polykernel (but we won't show it here).

Other Forms of Structure

Recall the SAT problem:

SAT
Instance: A CNF formula F.
Question: Is there an assignment satisfying F?

CNF Formula: A conjunction of *clauses*, each containing positive and negative occurrences of variables (individually called *literals*).

$$(x \vee y \vee \neg z) \wedge (z \vee w) \wedge (\neg x \vee y \vee z \vee \neg w)$$

Assignment: A mapping from the variables to $\{0, 1\}$ (false/true).

For example, the assignment $x, y \mapsto 0,\ z, w \mapsto 1$ does not satisfy F:

$$(x \vee y \vee \neg z) \wedge (z \vee w) \wedge (\neg x \vee y \vee z \vee \neg w)$$

A simple structural parameter

Observe: if all clauses contain only 1 variable, then F is trivial.

- Each such unit clause forces a specific assignment of the variable.
- Let us call a formula which contains only clauses with at most one literal a unit formula.

$$(x) \wedge (z) \wedge (\neg y) \wedge (\neg z)$$

A simple structural parameter

Observe: if all clauses contain only 1 variable, then F is trivial.

- Each such unit clause forces a specific assignment of the variable.
- Let us call a formula which contains only clauses with at most one literal a unit formula.

$$(x) \wedge (z) \wedge (\neg y) \wedge (\neg z)$$

What if we parameterize by the number of variables that need to be deleted to obtain a unit formula?

- Deleting a variable means removing it from all clauses (regardless of whether it occurs positively or negatively).

$$(x \vee \neg a) \wedge (z \vee b \vee \neg a) \wedge (\neg y \vee c) \wedge (\neg z \vee \neg b) \wedge (b \vee \neg c)$$

Example: Removing the red variables a, b, c markes the formula unit.

Problem Definition

SAT Using Deletion to Unit Clauses

Instance: A CNF formula F and a set X containing k variables whose deletion makes F unit.

Parameter: k.

Question: Is there an assignment satisfying F?

$$(x \vee \neg a) \wedge (z \vee b \vee \neg a) \wedge (\neg y \vee c) \wedge (\neg z \vee \neg b) \wedge (b \vee \neg c)$$

A formula F with a variable set $X = \{a, b, c\}$ marked in red.

Is SAT Using Deletion to Unit Clauses FPT?

Solving SAT Using Deletion to Unit Clauses

If there is a satisfying assignment, then we can guess how it assigns X.
We branch over all assignments of X.

- Each variable in X should be assigned either to *true* (1) or *false* (0) $\rightarrow 2^k$ options.

Solving SAT Using Deletion to Unit Clauses

If there is a satisfying assignment, then we can guess how it assigns X.
We branch over all assignments of X.

- Each variable in X should be assigned either to *true* (1) or *false* (0) $\rightarrow 2^k$ options.

$$(x \vee \neg a) \wedge (z \vee b \vee \neg a) \wedge (\neg y \vee c) \wedge (\neg z \vee \neg b) \wedge (b \vee \neg c)$$

Example: assume we assign $a, b, c \mapsto 0$.

Solving SAT Using Deletion to Unit Clauses

If there is a satisfying assignment, then we can guess how it assigns X. We branch over all assignments of X.

- Each variable in X should be assigned either to *true* (1) or *false* (0) $\rightarrow 2^k$ options.

$$(x \vee \neg a) \wedge (z \vee b \vee \neg a) \wedge (\neg y \vee c) \wedge (\neg z \vee \neg b) \wedge (b \vee \neg c)$$

Example: assume we assign $a, b, c \mapsto 0$. Blue represents *already satisfied*, while red represents *not satisfied yet*.

$$(x \vee \neg 0) \wedge (z \vee 0 \vee \neg 0) \wedge (\neg y \vee 0) \wedge (\neg z \vee \neg 0) \wedge (0 \vee \neg 0)$$

In each branch, we can delete clauses which are already satisfied and literals that are already unsatisfied. The resulting formula is unit.

$$(\neg y)$$

Evaluate in linear time. Return true if some branch has satisfying assignment.

Solving SAT Using Deletion to Unit Clauses – Recap

1. Branch over truth assignments in $X \to 2^k$ branches.
2. Each branch can be solved in linear time.

Theorem

SAT Using Deletion to Unit Clauses *is FPT.*

Solving SAT Using Deletion to Unit Clauses – Recap

1. Branch over truth assignments in $X \to 2^k$ branches.
2. Each branch can be solved in linear time.

Theorem

SAT Using Deletion to Unit Clauses *is FPT.*

Remark: in case you are wondering, this problem also does not admit a polynomial kernel (unless the polynomial hierarchy collapses).

Should Structure be Part of the Input?

- In all the examples until now, we assumed that the structure (vertex cover, deletion set) was given as part of the input.
- But what if you don't have access to the structure? As long as it is in the input, can you still use it? Of course, but you may need to compute it or approximate it.

Hidden Structure Problems

A new version of a problem we solved previously:

Hamiltonian Cycle Parameterized by Vertex Cover'

Instance: A graph G.

Parameter: The size k of a minimum vertex cover in G.

Question: Does there exist a Hamiltonian Cycle in G?

Hidden Structure Problems

A new version of a problem we solved previously:

Hamiltonian Cycle Parameterized by Vertex Cover'
Instance: A graph G.
Parameter: The size k of a minimum vertex cover in G.
Question: Does there exist a Hamiltonian Cycle in G?

When aiming for FPT algorithms: Use an FPT algorithm to find the structure.

1. Set $i = 1$. In time $f(i) \cdot \text{poly}(n)$ try to find vertex cover of size i. If not successful, set $i := i + 1$ and repeat.
2. This finds a vertex cover of size k in time $\sum_{i=1}^{k} f(i) \cdot \text{poly}(n)$.
3. Use the vertex cover to solve the problem (as was shown earlier).

Hidden Structure Problems – Polykernels

Let us try to find a polykernel for the hidden structure version.

Hamiltonian Cycle Parameterized by Vertex Cover'
Instance: A graph G.
Parameter: The size k of a minimum vertex cover in G.
Question: Does there exist a Hamiltonian Cycle in G?

Usually, we will want to use the structure of the vertex cover in the algorithm.

Is it sufficient to know that Vertex Cover admits a polykernel?

Hidden Structure Problems – Polykernels

Let us try to find a polykernel for the hidden structure version.

Hamiltonian Cycle Parameterized by Vertex Cover'
Instance: A graph G.
Parameter: The size k of a minimum vertex cover in G.
Question: Does there exist a Hamiltonian Cycle in G?

Usually, we will want to use the structure of the vertex cover in the algorithm.

Is it sufficient to know that Vertex Cover admits a polykernel? **No**.

1. The polykernel for vertex cover assumes you know k.
2. A polykernel does not give you an *actual vertex cover* you can use.
3. Moreover, a polykernel for Vertex Cover does not necessarily preserve the existence of a solution (e.g., a Hamiltonian cycle).

So, can we do anything?

Hidden Structure Problems – Polykernels

Let us try to find a polykernel for the hidden structure version.

Hamiltonian Cycle Parameterized by Vertex Cover'
Instance: A graph G.
Parameter: The size k of a minimum vertex cover in G.
Question: Does there exist a Hamiltonian Cycle in G?

Usually, we will want to use the structure of the vertex cover in the algorithm.

Is it sufficient to know that Vertex Cover admits a polykernel? **No**.

1. The polykernel for vertex cover assumes you know k.
2. A polykernel does not give you an *actual vertex cover* you can use.
3. Moreover, a polykernel for Vertex Cover does not necessarily preserve the existence of a solution (e.g., a Hamiltonian cycle).

So, can we do anything? **Yes**, we can use polynomial-time approximation.

Approximating Vertex Cover

Our previous kernels assumed we had **some** vertex cover. Not necessarily an optimal one.

The subproblem we want to solve in polynomial time is this:

Approximating Vertex Cover

Instance: A graph G.

Task: Let k be the size of a minimum vertex cover in G.
Compute a vertex cover of size at most poly(k).

Approximating Vertex Cover

Our previous kernels assumed we had **some** vertex cover. Not necessarily an optimal one.

The subproblem we want to solve in polynomial time is this:

> Approximating Vertex Cover
>
> *Instance*: A graph G.
>
> *Task*: Let k be the size of a minimum vertex cover in G.
> Compute a vertex cover of size at most poly(k).

- A solution will give us a "pretty good" vertex cover to use for the kernelization.
- For vertex cover, we can get a simple $2k$-approximation.

Approximating Vertex Cover

Approximating Vertex Cover

Instance: A graph G.

Task: Let k be the size of a minimum vertex cover in G.
Compute a vertex cover of size at most poly(k).

Algorithm:

1. Pick any edge e.
2. e must be covered, so at least one of its incident vertices must be in the vertex cover. So, put **both** into the approximate vertex cover.
3. We can now delete both endpoints of e (they're already in the cover).
4. Repeat until all edges are gone.

Approximating Vertex Cover

Approximating Vertex Cover

Instance: A graph G.

Task: Let k be the size of a minimum vertex cover in G.
Compute a vertex cover of size at most poly(k).

Algorithm:

1. Pick any edge e.
2. e must be covered, so at least one of its incident vertices must be in the vertex cover. So, put **both** into the approximate vertex cover.
3. We can now delete both endpoints of e (they're already in the cover).
4. Repeat until all edges are gone.

Remark: This algorithm is actually *the best known* approximation algorithm for vertex cover.

Back to Kernelization

Ok, so we can get a $2k$-approximate vertex cover. How to use it?

Hamiltonian Cycle Parameterized by Vertex Cover'
Instance: A graph G.
Parameter: The size k of a minimum vertex cover in G.
Question: Does there exist a Hamiltonian Cycle in G?

Back to Kernelization

Ok, so we can get a $2k$-approximate vertex cover. How to use it?

Hamiltonian Cycle Parameterized by Vertex Cover'
Instance: A graph G.
Parameter: The size k of a minimum vertex cover in G.
Question: Does there exist a Hamiltonian Cycle in G?

1. Run approximation algorithm to get a vertex cover X' of size at most $2k$.
2. Use this vertex cover X' to get a kernel (as before).

Before, we had vertex cover has size k and a kernel has size poly(k). Now, the kernel has size poly$(2k)$. This is still polynomial in k.

Back to Kernelization

Ok, so we can get a $2k$-approximate vertex cover. How to use it?

Hamiltonian Cycle Parameterized by Vertex Cover'
Instance: A graph G.
Parameter: The size k of a minimum vertex cover in G.
Question: Does there exist a Hamiltonian Cycle in G?

1. Run approximation algorithm to get a vertex cover X' of size at most $2k$.
2. Use this vertex cover X' to get a kernel (as before).

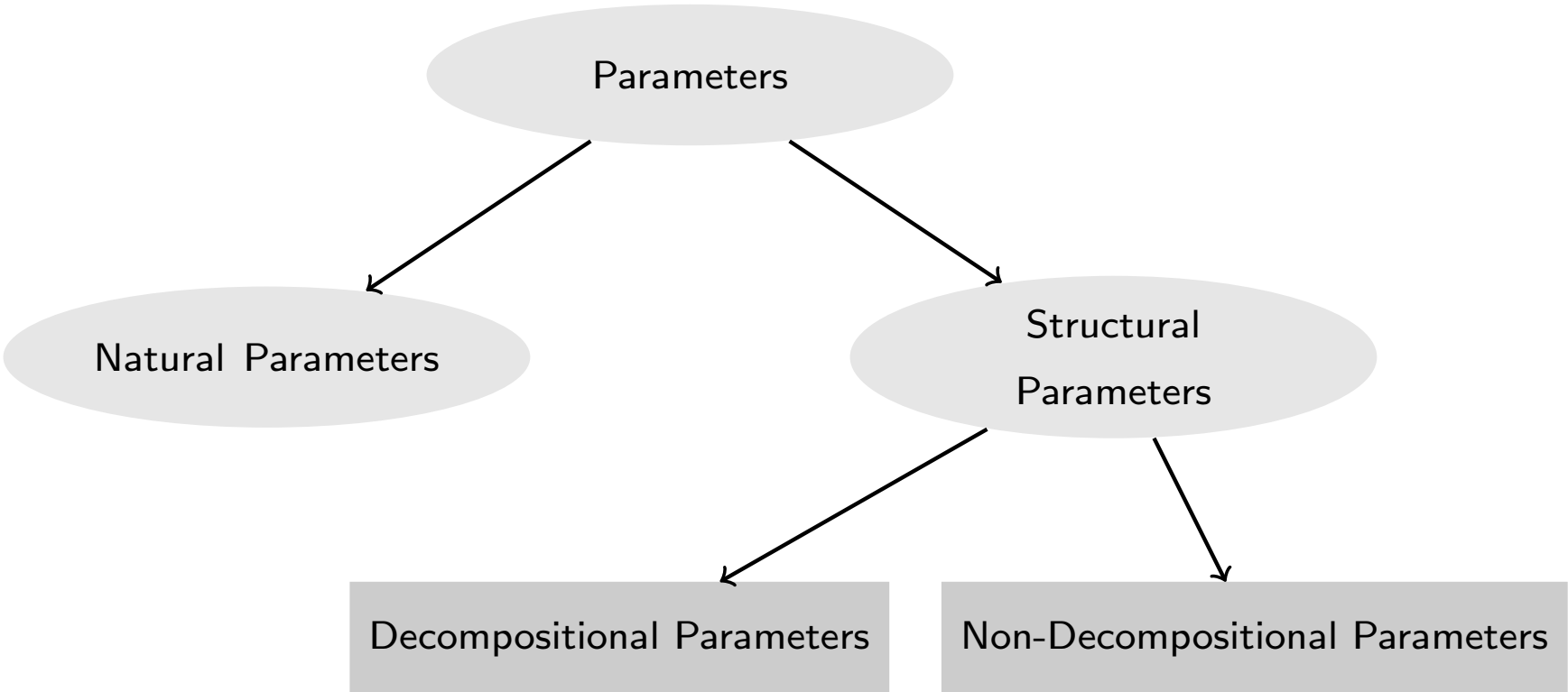
Before, we had vertex cover has size k and a kernel has size poly(k). Now, the kernel has size poly$(2k)$. This is still polynomial in k.

This is a common trick: If a structural parameter is too expensive to compute exactly, approximate it.

Recap

- Structural parameters can be used for both kernelization and FPT algorithms.
- This allows algorithms to naturally exploit useful structure in the instance
  - The performance of algorithms depends on how "well-structured" the instance is.
  - Depending on the setting, it might be necessary to compute the structure before it can be used.
- We introduced a few basic techniques used to design FPT algorithms and kernels parameterized by structural parameters.

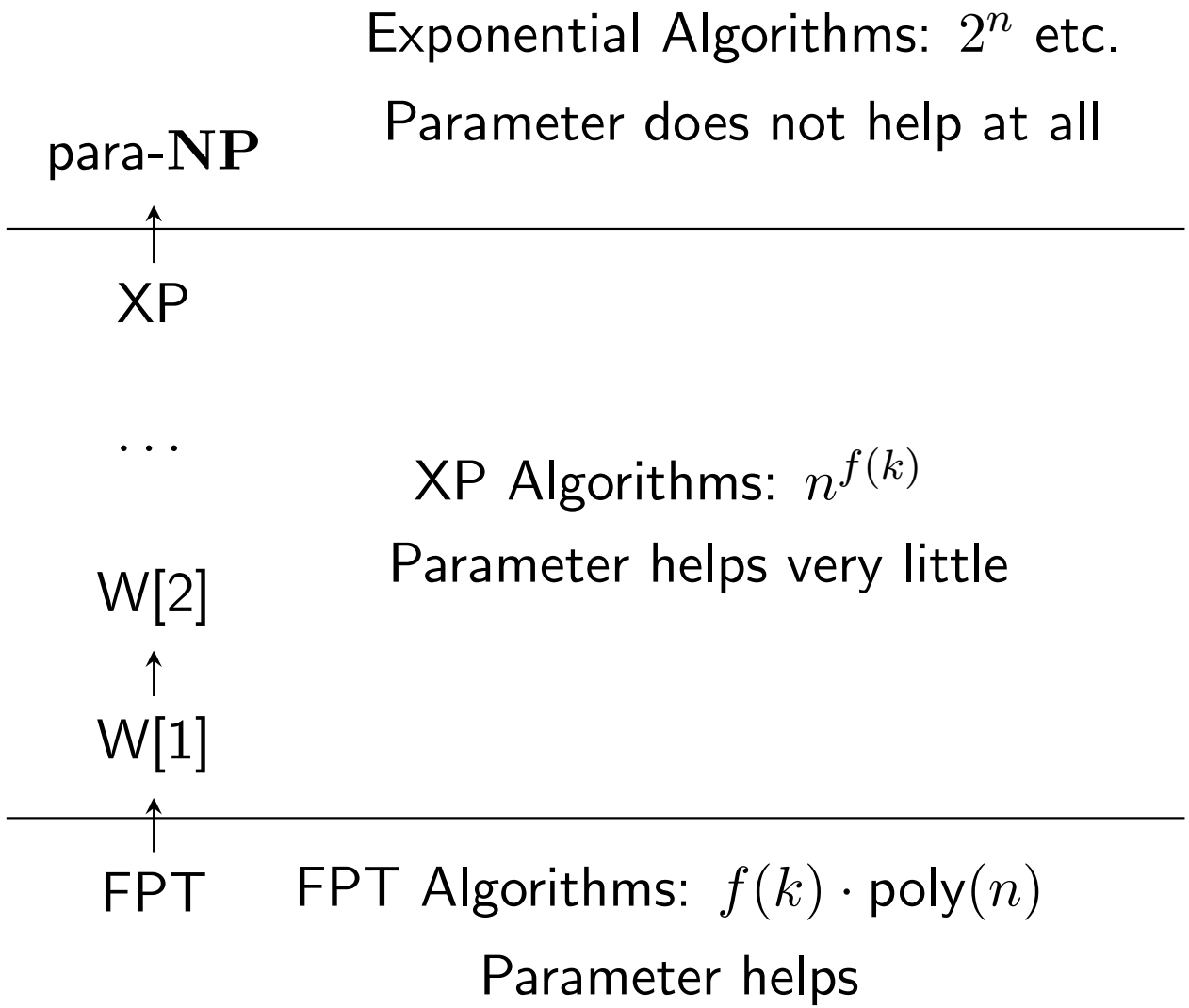
Recap: Overview of Parameterizations



- **Natural Parameters:** What we did the last two lectures.
- **Decompositional Parameters:** Next block.
- **Non-Decompositional Parameters:** What we did today.

Recap: Parameterized Complexity Classes and Algorithms

Classification for $\mathbf{NP}$-complete problems.



Many other (more advanced) algorithmic techniques exist for designing FPT algorithms

FPT Course

Course dedicated to this topic:

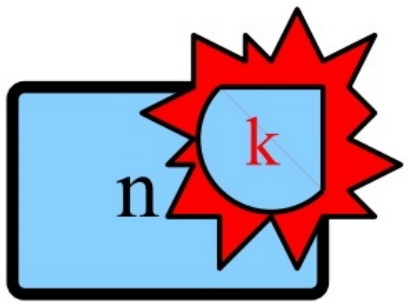
Fixed-Parameter Algorithms and Complexity

- Covers advanced algorithmic techniques for designing FPT algorithms (and kernels):
  - Iterative Compression, Integer Linear Programming, ...
- Introduces techniques to show that a problem is not FPT.
- Seminar-like format.
- Will also cover recent scientific breakthroughs in the field.
- Taught by Robert Ganian.

When? This semester, 9-30 January:

- **Mondays (13:00-16:30) and**
- **Thursdays (13:00-16:30)**.

Questions?



instead of