

An Introduction to Parameterized Complexity

Lecture 1: Fixed-Parameter Tractability

<https://clegra.github.io/paracompl/paracompl.html>

Clemens Grabmayer

Ph.D. Program Advanced Course

Gran Sasso Science Institute

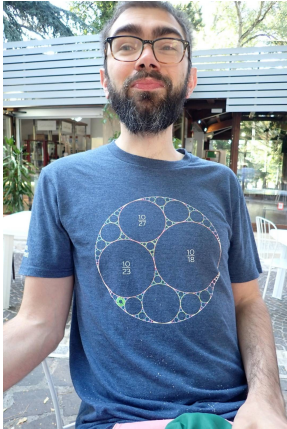
L'Aquila, Italy

Monday, July 14, 2025

Course overview

Monday, July 14 10.30 – 12.30	Tuesday, July 15 10.30 – 12.30	Wednesday, July 16 10.30 – 12.30	Thursday, July 17 10.30 – 12.30	Friday, July 18
<i>Algorithmic Techniques</i>		<i>Formal-Method & Algorithmic Techniques</i>		
Introduction & basic FPT results motivation for FPT kernelization, Crown Lemma, Sunflower Lemma	Notions of bounded graph width path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. width	Algorithmic Meta-Theorems 1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width	FPT-Intractability Classes & Hierarchies motivation for FP-intractability results, FPT-reductions, class XP (slice-wise polynomial), W- and A-Hierarchies, placing problems on these hierarchies	
				14.30 – 16.30
				examples, question hour

Course developers



Hugo Gilbert
course 2019/20 (Hugo & Clemens)



CG & Alessandro Aloisio
course 2020/21 (Alessandro & C)

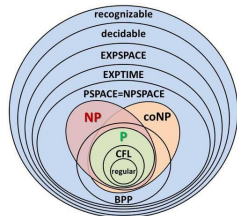
Course overview

Monday, July 14 10.30 – 12.30	Tuesday, July 15 10.30 – 12.30	Wednesday, July 16 10.30 – 12.30	Thursday, July 17 10.30 – 12.30	Friday, July 18
<i>Algorithmic Techniques</i>		<i>Formal-Method & Algorithmic Techniques</i>		
Introduction & basic FPT results motivation for FPT kernelization, Crown Lemma, Sunflower Lemma	Notions of bounded graph width path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. width	Algorithmic Meta-Theorems 1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width	FPT-Intractability Classes & Hierarchies motivation for FP-intractability results, FPT-reductions, class XP (slice-wise polynomial), W- and A-Hierarchies, placing problems on these hierarchies	
				14.30 – 16.30
				examples, question hour

Motivation

Classical complexity theory

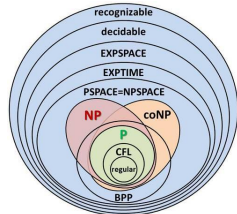
- ▶ analyses problems by **resource** (**space** or **time**)
needed to solve them on a **reasonable machine model**
- ▶ as a function of the **input size** $n = |x|$ (Hartmanis/Stearns, 1965)
- ⇒ variety of **complexity classes**
(P, LOGSPACE, NP, PSPACE, ...)
- ⇒ **tractable problems**
= polynomial-time computable (in P)
- ⇒ **theory of intractability**
(reductions, NP-completeness)



Motivation

Classical complexity theory

- ▶ analyses problems by **resource** (space or time)
needed to solve them on a **reasonable machine model**
- ▶ as a function of the **input size** $n = |x|$ (Hartmanis/Stearns, 1965)
- ⇒ variety of **complexity classes**
(P, LOGSPACE, NP, PSPACE, ...)
- ⇒ **tractable problems**
= polynomial-time computable (in P)
- ⇒ **theory of intractability**
(reductions, NP-completeness)



Drawback

- ▶ measures problem size $n = |x|$
only in terms of input instances x ,
and **ignores structural information** about instances
- ▶ sometimes problems are **easier to solve**
for instances if additional structure information is available

Motivation

Classical complexity theory

- ▶ analyses problems by **resource** (**space** or **time**)
needed to solve them on a **reasonable machine model**
 - ▶ as a function of the **input size** $n = |x|$ (Hartmanis/Stearns, 1965)
- ⇒ variety of **complexity classes** (P, LOGSPACE, NP, PSPACE, ...)
- ⇒ **tractable problems** = polynomial-time computable (in P)
- ⇒ **theory of intractability** (reductions, NP-completeness)

Motivation

Classical complexity theory

- ▶ analyses problems by **resource** (**space** or **time**)
needed to solve them on a **reasonable machine model**
- ▶ as a function of the **input size** $n = |x|$ (Hartmanis/Stearns, 1965)
- \Rightarrow variety of **complexity classes** (P, LOGSPACE, NP, PSPACE, ...)
- \Rightarrow **tractable problems** = polynomial-time computable (in P)
- \Rightarrow **theory of intractability** (reductions, NP-completeness)

Parameterized complexity

- ▶ measures complexity also in terms of a parameter $k = \kappa(x)$
that may depend on the input x in an arbitrary way
- \Rightarrow **fixed-parameter tractable problems**
relaxes polynomial time solvability to algorithms whose
non-polynomial behavior $f(k) \cdot p(n)$ is restricted by parameter k
- \Rightarrow complexity classes (FPT, XP, W[P], W- and A-hierarchies)
- \Rightarrow **theory of fixed-parameter intractability**

Parameterized (versus classical) problems

Definition

A **classical (decision) problem** is a pair $\langle \Sigma, Q \rangle$ where:

- ▷ $Q \subseteq \Sigma^*$ the set of *problem yes-instances* over a finite alphabet Σ

Parameterized (versus classical) problems

Definition

A **classical (decision) problem** is a pair $\langle \Sigma, Q \rangle$ where:

- ▷ $Q \subseteq \Sigma^*$ the set of *problem yes-instances* over a finite alphabet Σ

A **parameterized (decision) problem** is a triple $\langle \Sigma, Q, \kappa \rangle$ where:

- ▷ $Q \subseteq \Sigma^*$ the set of *problem yes-instances* over a finite alphabet Σ ,
- ▷ $\kappa : \Sigma^* \rightarrow \mathbb{N}$ a function, *the parameterization*.

Parameterized (versus classical) problems

Definition

A **classical (decision) problem** is a pair $\langle \Sigma, Q \rangle$ where:

- ▷ $Q \subseteq \Sigma^*$ the set of *problem yes-instances* over a finite alphabet Σ

A **parameterized (decision) problem** is a triple $\langle \Sigma, Q, \kappa \rangle$ where:

- ▷ $Q \subseteq \Sigma^*$ the set of *problem yes-instances* over a finite alphabet Σ ,
- ▷ $\kappa : \Sigma^* \rightarrow \mathbb{N}$ a function, *the parameterization*.

We regularly shorten $\langle \Sigma, Q, \kappa \rangle$ to a pair $\langle Q, \kappa \rangle$.

Parameterized (versus classical) problems

Definition

A **classical (decision) problem** is a pair $\langle \Sigma, Q \rangle$ where:

- ▷ $Q \subseteq \Sigma^*$ the set of *problem yes-instances* over a finite alphabet Σ

A **parameterized (decision) problem** is a triple $\langle \Sigma, Q, \kappa \rangle$ where:

- ▷ $Q \subseteq \Sigma^*$ the set of *problem yes-instances* over a finite alphabet Σ ,
- ▷ $\kappa : \Sigma^* \rightarrow \mathbb{N}$ a function, *the parameterization*.

We regularly shorten $\langle \Sigma, Q, \kappa \rangle$ to a pair $\langle Q, \kappa \rangle$.

Assumption

The parameterization κ can be **efficiently** computed.

Parameterized problems (examples)

A Parameterized Clique Problem

p-CLIQUE:

Given: a graph G and an integer k .

Question: Does there exists a clique of size k in G ?

Parameter: k .

Parameterized problems (examples)

A Parameterized Clique Problem

p-CLIQUE:

Given: a graph G and an integer k .

Question: Does there exists a clique of size k in G ?

Parameter: k .

A Parameterized Hitting Set Problem

p-HITTING SET

Given: a universe $U = \{x_1, \dots, x_n\}$, a collection of sets $\mathcal{S} = (S_1, \dots, S_m)$ where $S_i \subseteq U$ and an integer k ,

Question: Does there exists a set $S \subseteq U$ such that $|S| \leq k$ and $S \cap S_i \neq \emptyset, \forall i \in \{1, \dots, m\}$.

Parameterized problems (examples)

A Parameterized Clique Problem

p-CLIQUE:

Given: a graph G and an integer k .

Question: Does there exists a clique of size k in G ?

Parameter: k .

A Parameterized Hitting Set Problem

p-HITTING SET

Given: a universe $U = \{x_1, \dots, x_n\}$, a collection of sets $\mathcal{S} = (S_1, \dots, S_m)$ where $S_i \subseteq U$ and an integer k ,

Question: Does there exists a set $S \subseteq U$ such that $|S| \leq k$ and $S \cap S_i \neq \emptyset, \forall i \in \{1, \dots, m\}$.

Parameter: $\max |S_i|$.

Parameterized problems (examples)

A Parameterized Clique Problem

p-CLIQUE:

Given: a graph G and an integer k .

Question: Does there exists a clique of size k in G ?

Parameter: k .

A Parameterized Hitting Set Problem

p-HITTING SET

Given: a universe $U = \{x_1, \dots, x_n\}$, a collection of sets $\mathcal{S} = (S_1, \dots, S_m)$ where $S_i \subseteq U$ and an integer k ,

Question: Does there exists a set $S \subseteq U$ such that $|S| \leq k$ and $S \cap S_i \neq \emptyset, \forall i \in \{1, \dots, m\}$.

Parameter: $\max |S_i|$.

- ▶ NP-hard even if $\max |S_i| = 2$,

Parameterized problems (examples)

A Parameterized Clique Problem

p-CLIQUE:

Given: a graph G and an integer k .

Question: Does there exists a clique of size k in G ?

Parameter: k .

A Parameterized Hitting Set Problem

p-HITTING SET

Given: a universe $U = \{x_1, \dots, x_n\}$, a collection of sets $S = (S_1, \dots, S_m)$ where $S_i \subseteq U$ and an integer k ,

Question: Does there exists a set $S \subseteq U$ such that $|S| \leq k$ and $S \cap S_i \neq \emptyset, \forall i \in \{1, \dots, m\}$.

Parameter: $\max |S_i|$.

- ▶ NP-hard even if $\max |S_i| = 2$,
- ▶ is **fixed-parameter tractable**.

The art of parameterization

What is a **good parameter**?

The art of parameterization

What is a **good parameter**?

- ▶ We should have reasons to believe that the parameter is “**small**” for some applications.

The art of parameterization

What is a **good parameter**?

- ▶ We should have reasons to believe that the parameter is “**small**” for some applications.
- ▶ It is better if the parameter is **intuitive**.

The art of parameterization

What is a **good parameter**?

- ▶ We should have reasons to believe that the parameter is “**small**” for some applications.
- ▶ It is better if the parameter is **intuitive**.
- ▶ It is better if the parameter is **efficiently computable**.

The art of parameterization

What is a **good parameter**?

- ▶ We should have reasons to believe that the parameter is “**small**” for some applications.
- ▶ It is better if the parameter is **intuitive**.
- ▶ It is better if the parameter is **efficiently computable**.

There is a hierarchy on parameters.

The art of parameterization

There are many **different types** of parameters!

The art of parameterization

There are many **different types** of parameters!

- ▶ The **size of the solution** we are looking for.

The art of parameterization

There are many **different types** of parameters!

- ▶ The **size of the solution** we are looking for.
- ▶ The **size of some parts** of the instance.
E.g., the number of voters in an election problem.

The art of parameterization

There are many **different types** of parameters!

- ▶ The **size of the solution** we are looking for.
- ▶ The **size of some parts** of the instance.
E.g., the number of voters in an election problem.
- ▶ Some more **structural property** of the instance.
E.g., the diameter of a graph.

The art of parameterization

There are many **different types** of parameters!

- ▶ The **size of the solution** we are looking for.
- ▶ The **size of some parts** of the instance.
E.g., the number of voters in an election problem.
- ▶ Some more **structural property** of the instance.
E.g., the diameter of a graph.
- ▶ It can be a **combination** of values, a **difference**, ...

The art of parameterization

- ▶ **Graph problems**: maximum degree, treewidth, diameter...

The art of parameterization

- ▶ **Graph problems**: maximum degree, treewidth, diameter...
- ▶ **Social choice problems**: number of voters, candidates, correlation of preferences...

The art of parameterization

- ▶ **Graph problems**: maximum degree, treewidth, diameter...
- ▶ **Social choice problems**: number of voters, candidates, correlation of preferences...
- ▶ **Boolean formulas**: number of variables, number of clauses...

The art of parameterization

- ▶ **Graph problems**: maximum degree, treewidth, diameter...
- ▶ **Social choice problems**: number of voters, candidates, correlation of preferences...
- ▶ **Boolean formulas**: number of variables, number of clauses...
- ▶ **Problems on strings**: maximum length of a string, size of the alphabet...

Fixed Parameter Tractability (Class FPT)

Definition

A parameterized problem $\langle Q, \kappa \rangle$ is *fixed-parameter tractable* if:

$\exists f : \mathbb{N} \rightarrow \mathbb{N}$ computable $\exists p \in \mathbb{N}[X]$ polynomial
 $\exists \mathbb{A}$ algorithm, takes inputs in Σ^* and $\forall x \in \Sigma^*$
 $\left[\mathbb{A} \text{ decides if } x \in Q \text{ in time } \leq f(\kappa(x)) \cdot p(|x|) \right].$

FPT := complexity class of all fixed-parameter tractable problems.

Fixed Parameter Tractability (Class FPT)

Definition

A parameterized problem $\langle Q, \kappa \rangle$ is *fixed-parameter tractable* if:

$$\begin{aligned} &\exists f : \mathbb{N} \rightarrow \mathbb{N} \text{ computable } \exists p \in \mathbb{N}[X] \text{ polynomial} \\ &\quad \exists \mathbb{A} \text{ algorithm, takes inputs in } \Sigma^* \text{ and } \forall x \in \Sigma^* \\ &\quad \quad [\mathbb{A} \text{ decides if } x \in Q \text{ in time } \leq f(\kappa(x)) \cdot p(|x|)]. \end{aligned}$$

FPT := complexity class of all fixed-parameter tractable problems.

Assumption for a robust fpt-theory:

κ is *polynomially computable*, or itself *fpt-computable*.

Fixed Parameter Tractability (Class FPT)

Definition

A parameterized problem $\langle Q, \kappa \rangle$ is *fixed-parameter tractable* if:

$$\begin{aligned} &\exists f : \mathbb{N} \rightarrow \mathbb{N} \text{ computable } \exists p \in \mathbb{N}[X] \text{ polynomial} \\ &\quad \exists \text{ algorithm, takes inputs in } \Sigma^* \text{ and } \forall x \in \Sigma^* \\ &\quad \quad [\text{algorithm decides if } x \in Q \text{ in time } \leq f(\kappa(x)) \cdot p(|x|)]. \end{aligned}$$

FPT := complexity class of all fixed-parameter tractable problems.

Assumption for a robust fpt-theory:

κ is *polynomially computable*, or itself *fpt-computable*.

Goal in parameterized algorithmics:

- \Rightarrow design FPT algorithms,
- \Rightarrow try to make both factors $f(\kappa(x))$ and $p(|x|)$ as small as possible.
- \Rightarrow or show (if possible) that finding such factors is impossible

Slices of FPT problems are in P

The ℓ -th slice of a parameterized problem $\langle Q, \kappa \rangle$:

$$\langle Q, \kappa \rangle_{\ell} := \{x \in Q \mid \kappa(x) = \ell\} \quad (\text{as classical problem}).$$

Proposition

If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle_{\ell} \in \text{P}$ for all $\ell \in \mathbb{N}$.

Slices of FPT problems are in P

The ℓ -th slice of a parameterized problem $\langle Q, \kappa \rangle$:

$$\langle Q, \kappa \rangle_{\ell} := \{x \in Q \mid \kappa(x) = \ell\} \quad (\text{as classical problem}).$$

Proposition

If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle_{\ell} \in \text{P}$ for all $\ell \in \mathbb{N}$.

Proof.

If $\langle Q, \kappa \rangle \in \text{FPT}$, then there are a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$, a polynomial p , and an algorithm \mathbb{A} that decides $x \in \Sigma^*$ in running time $\leq f(\kappa(x)) \cdot p(|x|)$ time.

Slices of FPT problems are in P

The ℓ -th slice of a parameterized problem $\langle Q, \kappa \rangle$:

$$\langle Q, \kappa \rangle_{\ell} := \{x \in Q \mid \kappa(x) = \ell\} \quad (\text{as classical problem}).$$

Proposition

If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle_{\ell} \in \text{P}$ for all $\ell \in \mathbb{N}$.

Proof.

If $\langle Q, \kappa \rangle \in \text{FPT}$, then there are a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$, a polynomial p , and an algorithm \mathbb{A} that decides $x \in \Sigma^*$ in running time $\leq f(\kappa(x)) \cdot p(|x|)$ time. This algorithm can also be used to decide the ℓ -th slice in time $\leq f(\ell) \cdot p(|x|)$, which for fixed ℓ is a polynomial. \square

A problem not in FPT (unless $P = NP$)

The ℓ -th slice of a parameterized problem $\langle Q, \kappa \rangle$:

$$\langle Q, \kappa \rangle_{\ell} := \{x \in Q \mid \kappa(x) = \ell\} \quad (\text{as classical problem}).$$

Proposition

If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle_{\ell} \in P$ for all $\ell \in \mathbb{N}$.

Application

p -COLORABILITY

Instance: a graph \mathcal{G} and $k \in \mathbb{N}$.

Parameter: k .

Problem: Decide whether \mathcal{G} is k -colorable.

Known: 3-COLORABILITY \in NP-complete (Lovàsz, Stockmeyer, 1973).

A problem not in FPT (unless $P = NP$)

The ℓ -th slice of a parameterized problem $\langle Q, \kappa \rangle$:

$$\langle Q, \kappa \rangle_{\ell} := \{x \in Q \mid \kappa(x) = \ell\} \quad (\text{as classical problem}).$$

Proposition

If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle_{\ell} \in P$ for all $\ell \in \mathbb{N}$.

Application

p -COLORABILITY

Instance: a graph \mathcal{G} and $k \in \mathbb{N}$.

Parameter: k .

Problem: Decide whether \mathcal{G} is k -colorable.

Known: 3-COLORABILITY \in NP-complete (Lovász, Stockmeyer, 1973).

Since 3-COLORABILITY = p -COLORABILITY₃,

it follows that p -COLORABILITY \notin FPT (unless $P = NP$).

Slice-wise polynomial problems (Class XP)

Definition

A parameterized problem $\langle Q, \kappa \rangle$ is *slice-wise polynomial* if:

$\exists f, g : \mathbb{N} \rightarrow \mathbb{N}$ computable

$\exists \mathbb{A}$ algorithm, takes inputs in Σ^* and $\forall x \in \Sigma^*$

$\left[\mathbb{A} \text{ decides if } x \in Q \text{ in time } \leq f(\kappa(x)) \cdot |x|^{g(\kappa(x))} \right].$

XP := complexity class of slice-wise polynomial problems.

Slices of XP problems are in P

The ℓ -th slice of a parameterized problem $\langle Q, \kappa \rangle$:

$$\langle Q, \kappa \rangle_{\ell} := \{x \in Q \mid \kappa(x) = \ell\} \quad (\text{as classical problem}).$$

Proposition

If $\langle Q, \kappa \rangle \in \text{XP}$, then $\langle Q, \kappa \rangle_{\ell} \in \text{P}$ for all $\ell \in \mathbb{N}$.

Slices of XP problems are in P

The ℓ -th slice of a parameterized problem $\langle Q, \kappa \rangle$:

$$\langle Q, \kappa \rangle_{\ell} := \{x \in Q \mid \kappa(x) = \ell\} \quad (\text{as classical problem}).$$

Proposition

If $\langle Q, \kappa \rangle \in \text{XP}$, then $\langle Q, \kappa \rangle_{\ell} \in \text{P}$ for all $\ell \in \mathbb{N}$.

Proof.

If $\langle Q, \kappa \rangle \in \text{XP}$, then there are a function $f : \mathbb{N} \rightarrow \mathbb{N}$ computable, a polynomial p , and an algorithm \mathbb{A} that decides $x \in \Sigma^*$ in running time $\leq f(\kappa(x)) \cdot |x|^{p(\kappa(x))}$ time.

Slices of XP problems are in P

The ℓ -th slice of a parameterized problem $\langle Q, \kappa \rangle$:

$$\langle Q, \kappa \rangle_{\ell} := \{x \in Q \mid \kappa(x) = \ell\} \quad (\text{as classical problem}).$$

Proposition

If $\langle Q, \kappa \rangle \in \text{XP}$, then $\langle Q, \kappa \rangle_{\ell} \in \text{P}$ for all $\ell \in \mathbb{N}$.

Proof.

If $\langle Q, \kappa \rangle \in \text{XP}$, then there are a function $f : \mathbb{N} \rightarrow \mathbb{N}$ computable, a polynomial p , and an algorithm \mathbb{A} that decides $x \in \Sigma^*$ in running time $\leq f(\kappa(x)) \cdot |x|^{p(\kappa(x))}$ time. This algorithm can be used to decide the ℓ -th slice in time $\leq f(\ell) \cdot |x|^{p(\ell)}$, which for fixed ℓ is a polynomial. \square

A problem not in XP (unless $P = NP$)

The ℓ -th slice of a parameterized problem $\langle Q, \kappa \rangle$:

$$\langle Q, \kappa \rangle_{\ell} := \{x \in Q \mid \kappa(x) = \ell\} \quad (\text{as classical problem}).$$

Proposition

If $\langle Q, \kappa \rangle \in \text{XP}$, then $\langle Q, \kappa \rangle_{\ell} \in \text{P}$ for all $\ell \in \mathbb{N}$.

Application

p -COLORABILITY

Instance: a graph \mathcal{G} and $k \in \mathbb{N}$.

Parameter: k .

Problem: Decide whether \mathcal{G} is k -colorable.

Known: 3-COLORABILITY \in NP-complete (Lovàsz, Stockmeyer, 1973).

A problem not in XP (unless $P = NP$)

The ℓ -th slice of a parameterized problem $\langle Q, \kappa \rangle$:

$$\langle Q, \kappa \rangle_{\ell} := \{x \in Q \mid \kappa(x) = \ell\} \quad (\text{as classical problem}).$$

Proposition

If $\langle Q, \kappa \rangle \in \text{XP}$, then $\langle Q, \kappa \rangle_{\ell} \in P$ for all $\ell \in \mathbb{N}$.

Application

p -COLORABILITY

Instance: a graph \mathcal{G} and $k \in \mathbb{N}$.

Parameter: k .

Problem: Decide whether \mathcal{G} is k -colorable.

Known: 3-COLORABILITY \in NP-complete (Lovász, Stockmeyer, 1973).

Since 3-COLORABILITY = p -COLORABILITY₃,

it follows that p -COLORABILITY $\notin \text{XP}$ (unless $P = NP$).

Aims of the course

- 1 Acquire a **basic notions** of parameterized complexity.
- 2 Obtain an introduction to some techniques to derive **FPT or XP results**.
- 3 Obtain an introduction to a variety of techniques to prove **algorithmic lower bounds** and in particular prove **parameterized hardness results**.

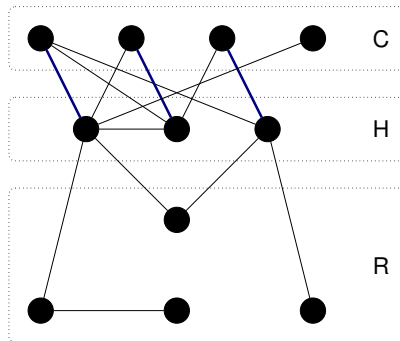
Course overview

Monday, July 14 10.30 – 12.30	Tuesday, July 15 10.30 – 12.30	Wednesday, July 16 10.30 – 12.30	Thursday, July 17 10.30 – 12.30	Friday, July 18
<i>Algorithmic Techniques</i>		<i>Formal-Method & Algorithmic Techniques</i>		
Introduction & basic FPT results motivation for FPT kernelization, Crown Lemma, Sunflower Lemma	Notions of bounded graph width path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. width	Algorithmic Meta-Theorems 1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width	FPT-Intractability Classes & Hierarchies motivation for FP-intractability results, FPT-reductions, class XP (slice-wise polynomial), W- and A-Hierarchies, placing problems on these hierarchies	
				14.30 – 16.30
				examples, question hour

Today

Monday, July 14 10.30 – 12.30	Tuesday, July 15 10.30 – 12.30	Wednesday, July 16 10.30 – 12.30	Thursday, July 17 10.30 – 12.30	Friday, July 18
<i>Algorithmic Techniques</i>		<i>Formal-Method & Algorithmic Techniques</i>		
Introduction & basic FPT results	Notions of bounded graph width	Algorithmic Meta-Theorems	FPT-Intractability Classes & Hierarchies	
motivation for FPT kernelization, Crown Lemma, Sunflower Lemma	path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. width	1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width	motivation for FP-intractability results, FPT-reductions, class XP (slice-wise polynomial), W- and A-Hierarchies, placing problems on these hierarchies	
				14.30 – 16.30
				examples, question hour

From today's lecture



A **crown decomposition** of a graph G is a partitioning (C, H, R) of $V(G)$, such that:

- 1 C is nonempty.
- 2 C is an independent set.
- 3 H separates C and R .
- 4 G contains a matching of H into C .

Crown Lemma (\Leftarrow results by König, Hall)

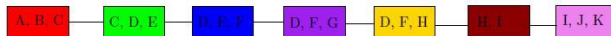
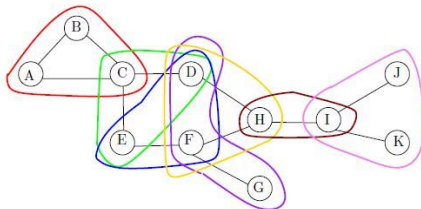
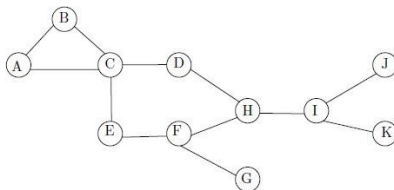
Let G be a graph with no isolated vertices and with at least $3k + 1$ vertices. There is a polynomial-time algorithm that:

- ▶ either finds a matching of size $k + 1$ in G ;
- ▶ or finds a crown decomposition of G .

Tomorrow

Monday, July 14 10.30 – 12.30	Tuesday, July 15 10.30 – 12.30	Wednesday, July 16 10.30 – 12.30	Thursday, July 17 10.30 – 12.30	Friday, July 18
<i>Algorithmic Techniques</i>		<i>Formal-Method & Algorithmic Techniques</i>		
Introduction & basic FPT results	Notions of bounded graph width	Algorithmic Meta-Theorems	FPT-Intractability Classes & Hierarchies	
motivation for FPT kernelization, Crown Lemma, Sunflower Lemma	path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. width	1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width	motivation for FP-intractability results, FPT-reductions, class XP (slice-wise polynomial), W- and A-Hierarchies, placing problems on these hierarchies	
				14.30 – 16.30
				examples, question hour

In tomorrow's lecture: a path decomposition of a graph



Wednesday

Monday, July 14 10.30 – 12.30	Tuesday, July 15 10.30 – 12.30	Wednesday, July 16 10.30 – 12.30	Thursday, July 17 10.30 – 12.30	Friday, July 18
<i>Algorithmic Techniques</i>		<i>Formal-Method & Algorithmic Techniques</i>		
Introduction & basic FPT results	Notions of bounded graph width	Algorithmic Meta-Theorems	FPT-Intractability Classes & Hierarchies	
motivation for FPT kernelization, Crown Lemma, Sunflower Lemma	path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. width	1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width	motivation for FP-intractability results, FPT-reductions, class XP (slice-wise polynomial), W- and A-Hierarchies, placing problems on these hierarchies	
				14.30 – 16.30
				examples, question hour

In Wednesday's lecture: Monadic second-order logic

$$\psi_3 := \exists C_1 \exists C_2 \exists C_3 \left(\left(\forall x \bigvee_{i=1}^3 C_i(x) \right) \wedge \forall x \forall y \left(E(x, y) \rightarrow \bigwedge_{i=1}^3 \neg (C_i(x) \wedge C_i(y)) \right) \right)$$

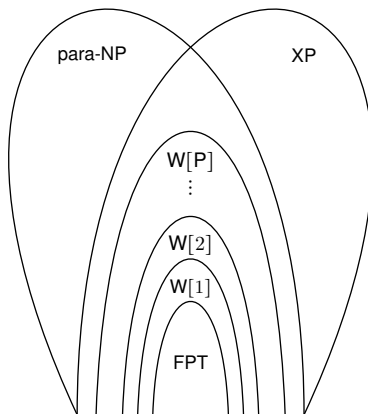
$$\mathcal{A}(\mathcal{G}) \models \psi_3 \iff \mathcal{G} \text{ has is } 3\text{-colorable.}$$

Thursday

Monday, July 14 10.30 – 12.30	Tuesday, July 15 10.30 – 12.30	Wednesday, July 16 10.30 – 12.30	Thursday, July 17 10.30 – 12.30	Friday, July 18
<i>Algorithmic Techniques</i>		<i>Formal-Method & Algorithmic Techniques</i>		
Introduction & basic FPT results	Notions of bounded graph width	Algorithmic Meta-Theorems	FPT-Intractability Classes & Hierarchies	
motivation for FPT kernelization, Crown Lemma, Sunflower Lemma	path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. width	1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width	motivation for FP-intractability results, FPT-reductions, class XP (slice-wise polynomial), W- and A-Hierarchies, placing problems on these hierarchies	
				14.30 – 16.30
				examples, question hour

From Thursday's lecture: W-Hierarchy

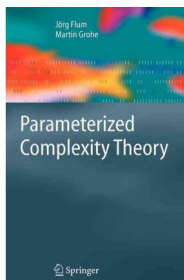
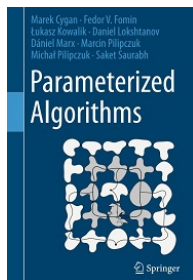
‘There is no definite single class that can be viewed as “the parameterized NP”. Rather, there is a whole hierarchy of classes playing this role. (Flum, Grohe [FG06])



Course overview

Monday, July 14 10.30 – 12.30	Tuesday, July 15 10.30 – 12.30	Wednesday, July 16 10.30 – 12.30	Thursday, July 17 10.30 – 12.30	Friday, July 18
<i>Algorithmic Techniques</i>		<i>Formal-Method & Algorithmic Techniques</i>		
Introduction & basic FPT results motivation for FPT kernelization, Crown Lemma, Sunflower Lemma	Notions of bounded graph width path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. width	Algorithmic Meta-Theorems 1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width	FPT-Intractability Classes & Hierarchies motivation for FP-intractability results, FPT-reductions, class XP (slice-wise polynomial), W- and A-Hierarchies, placing problems on these hierarchies	
				14.30 – 16.30
				examples, question hour

Books



Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh, *Parameterized Algorithms*, 1st ed., Springer, 2015.



Jörg Flum and Martin Grohe, *Parameterized Complexity Theory*, Springer, 2006.

Kernelization

- ▶ Idea
- ▶ Definition

Kernelization

- ▶ Idea
- ▶ Definition
- ▶ Kernel examples for:
 - ▶ point line cover problem
 - ▶ vertex cover problem

Kernelization

- ▶ Idea
- ▶ Definition
- ▶ Kernel examples for:
 - ▶ point line cover problem
 - ▶ vertex cover problem
- ▶ Kernelization \Leftrightarrow FPT

Kernelization

- ▶ Idea
- ▶ Definition
- ▶ Kernel examples for:
 - ▶ point line cover problem
 - ▶ vertex cover problem
- ▶ Kernelization \Leftrightarrow FPT
- ▶ Crown lemma and crown decomposition
 - ▶ smaller kernel for vertex cover problem
 - ▶ kernel for dual colorability problem

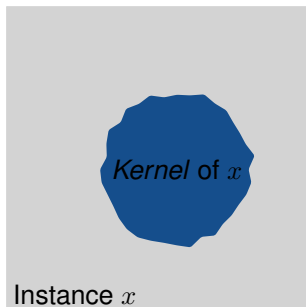
Kernelization

- ▶ Idea
- ▶ Definition
- ▶ Kernel examples for:
 - ▶ point line cover problem
 - ▶ vertex cover problem
- ▶ Kernelization \Leftrightarrow FPT
- ▶ Crown lemma and crown decomposition
 - ▶ smaller kernel for vertex cover problem
 - ▶ kernel for dual colorability problem
- ▶ Sunflower lemma
 - ▶ kernel for hitting set problem

Kernelization methods (informally)

Kernelization is:

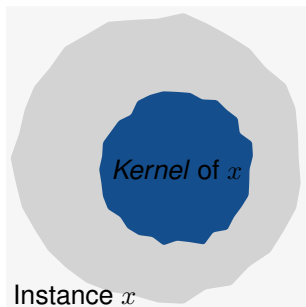
- ▶ a systematic study of polynomial-time preprocessing algorithms,
- ▶ an important tool in the design of parameterized algorithms.



Kernelization methods (informally)

Kernelization is:

- ▶ a systematic study of **polynomial-time preprocessing algorithms**,
- ▶ an important tool in the design of parameterized algorithms.



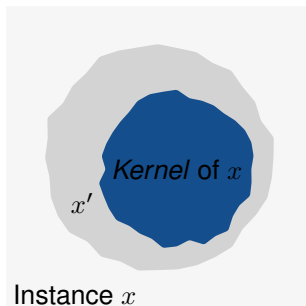
→ Application of rule 1

- ▶ Often a collection of efficient **preprocessing rules**.

Kernelization methods (informally)

Kernelization is:

- ▶ a systematic study of **polynomial-time preprocessing algorithms**,
- ▶ an important tool in the design of parameterized algorithms.



→ Application of rule 1

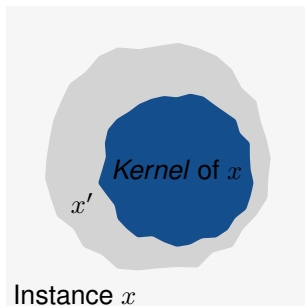
→ Application of rule 2

- ▶ Often a collection of efficient **preprocessing rules**.
- ▶ Transform an instance x into a smaller equivalent instance x' .

Kernelization methods (informally)

Kernelization is:

- ▶ a systematic study of **polynomial-time preprocessing algorithms**,
- ▶ an important tool in the design of parameterized algorithms.



→ Application of rule 1

→ Application of rule 2

- ▶ Often a collection of efficient **preprocessing rules**.
- ▶ Transform an instance x into a smaller equivalent instance x' .
- ▶ Hopefully, $|x'| \leq g(\kappa(x))$.
→ use a (non-efficient) exact algorithm.

Kernelization (formally)

Definition

Let $\langle Q, \kappa \rangle$ be a parameterized problem over Σ .

A **kernelization** of $\langle Q, \kappa \rangle$ is a function $K: \Sigma^* \rightarrow \Sigma^*$ such that:

- (K1) For all $x \in \Sigma^* : (x \in Q \iff K(x) \in Q)$.
- (K2) K is polynomial-time computable.
- (K3) There is a computable function $h: \mathbb{N} \rightarrow \mathbb{N}$ such that for all $x \in \Sigma^* : |K(x)| \leq h(\kappa(x))$.

Kernelization (formally)

Definition

Let $\langle Q, \kappa \rangle$ be a parameterized problem over Σ .

A **kernelization** of $\langle Q, \kappa \rangle$ is a function $K: \Sigma^* \rightarrow \Sigma^*$ such that:

- (K1) For all $x \in \Sigma^* : (x \in Q \iff K(x) \in Q)$.
- (K2) K is polynomial-time computable.
- (K3) There is a computable function $h: \mathbb{N} \rightarrow \mathbb{N}$
such that for all $x \in \Sigma^* : |K(x)| \leq h(\kappa(x))$.

We say that such a kernelization K is **polynomial** (resp. **linear**)
(and that Q **has a polynomial** (resp. **linear**) kernel)
if the function h is **polynomial** (resp. **linear**).

Kernelization (formally)

Definition

Let $\langle Q, \kappa \rangle$ be a parameterized problem over Σ .

A **kernelization** of $\langle Q, \kappa \rangle$ is a function $K: \Sigma^* \rightarrow \Sigma^*$ such that:

- (K1) For all $x \in \Sigma^* : (x \in Q \iff K(x) \in Q)$.
- (K2) K is polynomial-time computable.
- (K3) There is a computable function $h: \mathbb{N} \rightarrow \mathbb{N}$ such that for all $x \in \Sigma^* : |K(x)| \leq h(\kappa(x))$.

We say that such a kernelization K is **polynomial** (resp. **linear**) (and that Q **has a polynomial** (resp. **linear**) kernel) if the function h is **polynomial** (resp. **linear**).

Lemma

If $\langle Q, \kappa \rangle$ admits a kernel and is decidable, then $\langle Q, \kappa \rangle \in \text{FPT}$.

Kernelization (formally)

Definition

Let $\langle Q, \kappa \rangle$ be a parameterized problem over Σ .

A *kernelization* of $\langle Q, \kappa \rangle$ is a function $K: \Sigma^* \rightarrow \Sigma^*$ such that:

- (K1) For all $x \in \Sigma^* : (x \in Q \iff K(x) \in Q)$.
- (K2) K is polynomial-time computable.
- (K3) There is a computable function $h: \mathbb{N} \rightarrow \mathbb{N}$
such that for all $x \in \Sigma^* : |K(x)| \leq h(\kappa(x))$.

We say that such a kernelization K is *polynomial* (resp. *linear*)
(and that Q *has a polynomial* (resp. *linear*) kernel)
if the function h is *polynomial* (resp. *linear*).

Lemma

If $\langle Q, \kappa \rangle$ admits a kernel and is decidable, then $\langle Q, \kappa \rangle \in \text{FPT}$.

Lemma

If $\langle Q, \kappa \rangle \in \text{FPT}$, the $\langle Q, \kappa \rangle$ admits a kernel.

The (parameterized) Point Line Cover Problem

p-POINT-LINE-COVER:

Given: n points in the plane and an integer k .

Parameter: The integer k .

Question: Do there exist k lines that cover all points?

The (parameterized) Point Line Cover Problem

p-POINT-LINE-COVER:

Given: n points in the plane and an integer k .

Parameter: The integer k .

Question: Do there exist k lines that cover all points?

Rule 1:

If we have a line that hits $k + 1$ or more points, **then**:

- i) include it in the solution;
- ii) remove the points hit by the line;
- iii) set $k := k - 1$.

The (parameterized) Point Line Cover Problem

p-POINT-LINE-COVER:

Given: n points in the plane and an integer k .

Parameter: The integer k .

Question: Do there exist k lines that cover all points?

Rule 1:

If we have a line that hits $k + 1$ or more points, then:

- i) include it in the solution;
- ii) remove the points hit by the line;
- iii) set $k := k - 1$.

Observation: Let (x, κ) be a **yes instance** of the p-Point-Line-Cover such that Rule 1 cannot be applied. Then $n \leq \kappa^2$ holds.

The (parameterized) Point Line Cover Problem

p-POINT-LINE-COVER:

Given: n points in the plane and an integer k .

Parameter: The integer k .

Question: Do there exist k lines that cover all points?

Rule 1:

If we have a line that hits $k + 1$ or more points, then:

- i) include it in the solution;
- ii) remove the points hit by the line;
- iii) set $k := k - 1$.

Observation: Let (x, κ) be a **yes instance** of the p-Point-Line-Cover such that Rule 1 cannot be applied. Then $n \leq k^2$ holds.

Rule 2:

If we cannot apply Rule 1, and we have more than k^2 points, then say **no**, and return a **trivial no instance**.

The (parameterized) Point Line Cover Problem

p-POINT-LINE-COVER:

Given: n points in the plane and an integer k .

Parameter: The integer k .

Question: Do there exist k lines that cover all points?

Rule 1:

If we have a line that hits $k + 1$ or more points, **then:**

- i) include it in the solution;
- ii) remove the points hit by the line;
- iii) set $k := k - 1$.

Observation: Let (x, κ) be a **yes instance** of the p-Point-Line-Cover such that Rule 1 cannot be applied. **Then** $n \leq k^2$ holds.

Rule 2:

If we cannot apply Rule 1, and we have more than k^2 points, **then** say **no**, and return a **trivial no instance**.

Proposition

p-POINT-LINE-COVER \in **FPT**: it admits a kernel of size with k^2 points.

The (parameterized) Vertex Cover Problem

p-VERTEX-COVER:

Given: A graph G , and an integer k .

Parameter: The integer k .

Question: Does there exists a vertex cover of size at most k ?

Definition

Let G be a graph and $S \subseteq V(G)$. The set S is called a **vertex cover** if for every edge of G at least one of its endpoints is in S .

The (parameterized) Vertex Cover Problem

p-VERTEX-COVER:

Given: A graph G , and an integer k .

Parameter: The integer k .

Question: Does there exists a vertex cover of size at most k ?

Definition

Let G be a graph and $S \subseteq V(G)$. The set S is called a **vertex cover** if for every edge of G at least one of its endpoints is in S .

Exercise

Find an $O(k^2)$ kernel for p-VERTEX-COVER.

The (parameterized) Vertex Cover Problem *(Buss kernel)*

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G \setminus v, k)$

The (parameterized) Vertex Cover Problem *(Buss kernel)*

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G \setminus v, k)$

Rule 2: If there is a vertex v of degree at least $k + 1$, then delete v (and its incident edges) from G and decrement the parameter k by 1.
The new instance is $(G \setminus v; k - 1)$

The (parameterized) Vertex Cover Problem *(Buss kernel)*

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G \setminus v, k)$

Rule 2: If there is a vertex v of degree at least $k + 1$, then delete v (and its incident edges) from G and decrement the parameter k by 1.
The new instance is $(G \setminus v; k - 1)$

Observations

- ▶ After exhaustive application of Rule 1 and Rule 2 all vertices have degree between 1 and k .

The (parameterized) Vertex Cover Problem *(Buss kernel)*

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G \setminus v, k)$

Rule 2: If there is a vertex v of degree at least $k + 1$, then delete v (and its incident edges) from G and decrement the parameter k by 1.
The new instance is $(G \setminus v; k - 1)$

Observations

- ▶ After exhaustive application of Rule 1 and Rule 2 all vertices have degree between 1 and k .
- ▶ If G has maximum degree d , k vertices can cover $\leq k \cdot d$ edges.

The (parameterized) Vertex Cover Problem *(Buss kernel)*

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G \setminus v, k)$

Rule 2: If there is a vertex v of degree at least $k + 1$, then delete v (and its incident edges) from G and decrement the parameter k by 1.
The new instance is $(G \setminus v; k - 1)$

Observations

- ▶ After exhaustive application of Rule 1 and Rule 2 all vertices have degree between 1 and k .
- ▶ If G has maximum degree d , k vertices can cover $\leq k \cdot d$ edges.
- ▶ If G has a vertex cover of $\leq k$ vertices after exhaustive application of Rules 1 & 2, then G has $\leq k^2$ edges (and $\leq k^2 + k$ vertices).

The (parameterized) Vertex Cover Problem *(Buss kernel)*

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G \setminus v, k)$

Rule 2: If there is a vertex v of degree at least $k + 1$, then delete v (and its incident edges) from G and decrement the parameter k by 1.
The new instance is $(G \setminus v; k - 1)$

Observations

- ▶ If G has a vertex cover of $\leq k$ vertices after exhaustive application of Rules 1 & 2, then G has $\leq k^2$ edges (and $\leq k^2 + k$ vertices).

The (parameterized) Vertex Cover Problem *(Buss kernel)*

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G \setminus v, k)$

Rule 2: If there is a vertex v of degree at least $k + 1$, then delete v (and its incident edges) from G and decrement the parameter k by 1.
The new instance is $(G \setminus v; k - 1)$

Observations

- ▶ If G has a vertex cover of $\leq k$ vertices after exhaustive application of Rules 1 & 2, then G has $\leq k^2$ edges (and $\leq k^2 + k$ vertices).

Rule 3: Let (G, k) be an instance to which Rules 1 & 2 are not applicable. If G has $> k^2 + k$ vertices, or $> k^2$ edges, then (G, k) is a **no-instance** that can be replaced by a **trivial no-instance**.

The (parameterized) Vertex Cover Problem *(Buss kernel)*

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G \setminus v, k)$

Rule 2: If there is a vertex v of degree at least $k + 1$, then delete v (and its incident edges) from G and decrement the parameter k by 1.
The new instance is $(G \setminus v; k - 1)$

Observations

- ▶ If G has a vertex cover of $\leq k$ vertices after exhaustive application of Rules 1 & 2, then G has $\leq k^2$ edges (and $\leq k^2 + k$ vertices).

Rule 3: Let (G, k) be an instance to which Rules 1 & 2 are not applicable. If G has $> k^2 + k$ vertices, or $> k^2$ edges, then (G, k) is a **no-instance** that can be replaced by a **trivial no-instance**.

Theorem (Samuel Buss)

p-VERTEX-COVER \in **FPT**, because it admits a kernel with at most $O(k^2)$ vertices and $O(k^2)$ edges.

Kernelization \Rightarrow FPT

Exercise

If $\langle Q, \kappa \rangle$ admits a kernel and is decidable, then $\langle Q, \kappa \rangle \in \text{FPT}$.

Kernelization \Rightarrow FPT

Exercise

If $\langle Q, \kappa \rangle$ admits a kernel and is decidable, then $\langle Q, \kappa \rangle \in \text{FPT}$.

Definitions

A *kernelization* of $\langle Q, \kappa \rangle$ is a function $K: \Sigma^* \rightarrow \Sigma^*$ such that:

- (K1) For all $x \in \Sigma^*$: $(x \in Q \iff K(x) \in Q)$.
- (K2) K is polynomial-time computable.
- (K3) There is a computable function $h: \mathbb{N} \rightarrow \mathbb{N}$ such that for all $x \in \Sigma^*$: $|K(x)| \leq h(\kappa(x))$.

A parameterized problem $\langle Q, \kappa \rangle$ is *fixed-parameter tractable* if:

$$\begin{aligned} &\exists f: \mathbb{N} \rightarrow \mathbb{N} \text{ computable } \exists p \in \mathbb{N}[X] \text{ polynomial} \\ &\quad \exists \text{ algorithm, takes inputs in } \Sigma^* \text{ and } \forall x \in \Sigma^* \\ &\quad \left[\text{algorithm decides if } x \in Q \text{ in time } \leq f(\kappa(x)) \cdot p(|x|) \right]. \end{aligned}$$

FPT := complexity class of all fixed-parameter tractable problems.

Kernelization \Rightarrow FPT

Lemma

If $\langle Q, \kappa \rangle$ admits a kernel and is decidable, then $\langle Q, \kappa \rangle \in \text{FPT}$.

Kernelization \Rightarrow FPT

Lemma

If $\langle Q, \kappa \rangle$ admits a kernel and is decidable, then $\langle Q, \kappa \rangle \in \text{FPT}$.

$\langle Q, \kappa \rangle$ a parameterized problem, $Q \subseteq \Sigma^*$

Definition: $K: \Sigma^* \rightarrow \Sigma^*$ a kernelization for $\langle Q, \kappa \rangle$ if:

(K1) $\forall x \in \Sigma^* (x \in Q \Leftrightarrow K(x) \in Q)$

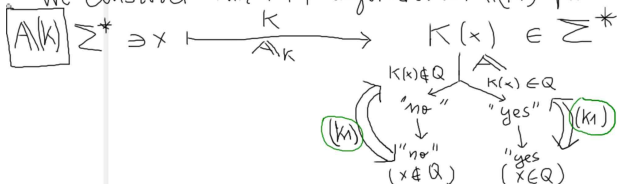
(K2) K is poly time computable

(K3) $\exists h: \mathbb{N} \rightarrow \mathbb{N} \forall x \in \Sigma^* (|K(x)| \leq h(\kappa(x)))$.

Proposition: If $\langle Q, \kappa \rangle$ is decidable, and has kernelization K , then $\langle Q, \kappa \rangle \in \text{FPT}$

Proof. Since $\langle Q, \kappa \rangle$ is decidable, there is an algorithm A that decides instances $x \in \Sigma^*$ in time $\leq f(|x|)$ steps for some computable function $f: \mathbb{N} \rightarrow \mathbb{N}$.

Then, assuming a polynomial algorithm A_K for K (time bounded by $P(x)$) we construct an FPT algorithm $A(K)$ for $\langle Q, \kappa \rangle$:



$$\begin{aligned}
 \text{Running time } A(K) &= \text{time}(A_K) + \text{time}(A(K(x))) \\
 &= p(|x|) + f(\kappa(K(x))) \\
 &\stackrel{\text{by (K2)}}{=} p(|x|) + f(h(\kappa(x))) \\
 &= (f \circ h)(\kappa(x)) \cdot (1+p)(|x|) \\
 &= f(\kappa(x)) \cdot \text{poly}(|x|) \in \text{FPT}.
 \end{aligned}$$

FPT \Rightarrow Kernelization

Lemma

*If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle$ admits a **kernel**.*

Proof.

Let \mathbb{A} be an algorithm that solves $\langle Q, \kappa \rangle$ in time $f(\kappa(x)) \cdot p(x)$, for all $x \in \Sigma^*$, where $f : \mathbb{N} \rightarrow \mathbb{N}$ computable, and $p(n)$ a polynomial.

FPT \Rightarrow Kernelization

Lemma

*If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle$ admits a *kernel*.*

Proof.

Let \mathbb{A} be an algorithm that solves $\langle Q, \kappa \rangle$ in time $f(\kappa(x)) \cdot p(x)$, for all $x \in \Sigma^*$, where $f : \mathbb{N} \rightarrow \mathbb{N}$ computable, and $p(n)$ a polynomial. We can assume $p(n) \geq \max\{n, 1\}$ for all $n \in \mathbb{N}$.

FPT \Rightarrow Kernelization

Lemma

If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle$ admits a kernel.

Proof.

Let \mathbb{A} be an algorithm that solves $\langle Q, \kappa \rangle$ in time $f(\kappa(x)) \cdot p(x)$, for all $x \in \Sigma^*$, where $f : \mathbb{N} \rightarrow \mathbb{N}$ computable, and $p(n)$ a polynomial. We can assume $p(n) \geq \max\{n, 1\}$ for all $n \in \mathbb{N}$.

If $Q = \emptyset$ or $Q = \Sigma^*$, then we can defined $K(x) := \epsilon$.

FPT \Rightarrow Kernelization

Lemma

*If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle$ admits a **kernel**.*

Proof.

Let \mathbb{A} be an algorithm that solves $\langle Q, \kappa \rangle$ in time $f(\kappa(x)) \cdot p(x)$, for all $x \in \Sigma^*$, where $f : \mathbb{N} \rightarrow \mathbb{N}$ computable, and $p(n)$ a polynomial. We can assume $p(n) \geq \max\{n, 1\}$ for all $n \in \mathbb{N}$.

If $Q = \emptyset$ or $Q = \Sigma^*$, then we can defined $K(x) := \epsilon$. Otherwise we have $\emptyset \subsetneq Q \subsetneq \Sigma^*$, and we choose some $x_0 \in Q$, and $x_1 \in \Sigma^* \setminus Q$.

FPT \Rightarrow Kernelization

Lemma

If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle$ admits a *kernel*.

Proof.

Let \mathbb{A} be an algorithm that solves $\langle Q, \kappa \rangle$ in time $f(\kappa(x)) \cdot p(x)$, for all $x \in \Sigma^*$, where $f : \mathbb{N} \rightarrow \mathbb{N}$ computable, and $p(n)$ a polynomial. We can assume $p(n) \geq \max\{n, 1\}$ for all $n \in \mathbb{N}$.

If $Q = \emptyset$ or $Q = \Sigma^*$, then we can defined $K(x) := \epsilon$. Otherwise we have $\emptyset \subsetneq Q \subsetneq \Sigma^*$, and we choose some $x_0 \in Q$, and $x_1 \in \Sigma^* \setminus Q$.

We define the **polynomial-time computable function** $K : \Sigma^* \rightarrow \Sigma^*$ by:

$$K(x) := \begin{cases} x_0 & \dots \mathbb{A} \text{ accepts } x \text{ in } \leq p(|x|) \cdot p(|x|) \text{ steps,} \\ x_1 & \dots \mathbb{A} \text{ rejects } x \text{ in } \leq p(|x|) \cdot p(|x|) \text{ steps,} \\ x & \dots \mathbb{A} \text{ does not terminate in } \leq p(|x|) \cdot p(|x|) \text{ steps.} \end{cases}$$

FPT \Rightarrow Kernelization

Lemma

If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle$ admits a *kernel*.

Proof.

Let \mathbb{A} be an algorithm that solves $\langle Q, \kappa \rangle$ in time $f(\kappa(x)) \cdot p(x)$, for all $x \in \Sigma^*$, where $f : \mathbb{N} \rightarrow \mathbb{N}$ computable, and $p(n)$ a polynomial. We can assume $p(n) \geq \max\{n, 1\}$ for all $n \in \mathbb{N}$.

If $Q = \emptyset$ or $Q = \Sigma^*$, then we can defined $K(x) := \epsilon$. Otherwise we have $\emptyset \subsetneq Q \subsetneq \Sigma^*$, and we choose some $x_0 \in Q$, and $x_1 \in \Sigma^* \setminus Q$.

We define the **polynomial-time computable function** $K : \Sigma^* \rightarrow \Sigma^*$ by:

$$K(x) := \begin{cases} x_0 & \dots \mathbb{A} \text{ accepts } x \text{ in } \leq p(|x|) \cdot p(|x|) \text{ steps,} \\ x_1 & \dots \mathbb{A} \text{ rejects } x \text{ in } \leq p(|x|) \cdot p(|x|) \text{ steps,} \\ x & \dots \mathbb{A} \text{ does not terminate in } \leq p(|x|) \cdot p(|x|) \text{ steps.} \end{cases}$$

In the last case ($K(x) = x$) we have $p(|x|) \cdot p(|x|) \leq f(\kappa(x)) \cdot p(|x|)$,

FPT \Rightarrow Kernelization

Lemma

If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle$ admits a *kernel*.

Proof.

Let \mathbb{A} be an algorithm that solves $\langle Q, \kappa \rangle$ in time $f(\kappa(x)) \cdot p(x)$, for all $x \in \Sigma^*$, where $f: \mathbb{N} \rightarrow \mathbb{N}$ computable, and $p(n)$ a polynomial. We can assume $p(n) \geq \max\{n, 1\}$ for all $n \in \mathbb{N}$.

If $Q = \emptyset$ or $Q = \Sigma^*$, then we can defined $K(x) := \epsilon$. Otherwise we have $\emptyset \subsetneq Q \subsetneq \Sigma^*$, and we choose some $x_0 \in Q$, and $x_1 \in \Sigma^* \setminus Q$.

We define the **polynomial-time computable function** $K: \Sigma^* \rightarrow \Sigma^*$ by:

$$K(x) := \begin{cases} x_0 & \dots \mathbb{A} \text{ accepts } x \text{ in } \leq p(|x|) \cdot p(|x|) \text{ steps,} \\ x_1 & \dots \mathbb{A} \text{ rejects } x \text{ in } \leq p(|x|) \cdot p(|x|) \text{ steps,} \\ x & \dots \mathbb{A} \text{ does not terminate in } \leq p(|x|) \cdot p(|x|) \text{ steps.} \end{cases}$$

In the last case ($K(x) = x$) we have $p(|x|) \cdot p(|x|) \leq f(\kappa(x)) \cdot p(|x|)$, and hence $|K(x)| = |x| \leq p(|x|) \leq f(\kappa(x))$.

FPT \Rightarrow Kernelization

Lemma

If $\langle Q, \kappa \rangle \in \text{FPT}$, then $\langle Q, \kappa \rangle$ admits a *kernel*.

Proof.

Let \mathbb{A} be an algorithm that solves $\langle Q, \kappa \rangle$ in time $f(\kappa(x)) \cdot p(x)$, for all $x \in \Sigma^*$, where $f : \mathbb{N} \rightarrow \mathbb{N}$ computable, and $p(n)$ a polynomial. We can assume $p(n) \geq \max\{n, 1\}$ for all $n \in \mathbb{N}$.

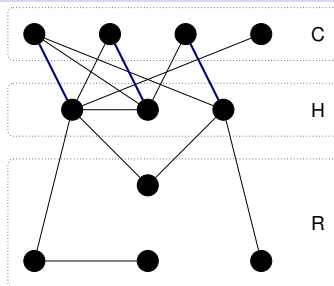
If $Q = \emptyset$ or $Q = \Sigma^*$, then we can defined $K(x) := \epsilon$. Otherwise we have $\emptyset \subsetneq Q \subsetneq \Sigma^*$, and we choose some $x_0 \in Q$, and $x_1 \in \Sigma^* \setminus Q$.

We define the *polynomial-time computable function* $K : \Sigma^* \rightarrow \Sigma^*$ by:

$$K(x) := \begin{cases} x_0 & \dots \mathbb{A} \text{ accepts } x \text{ in } \leq p(|x|) \cdot p(|x|) \text{ steps,} \\ x_1 & \dots \mathbb{A} \text{ rejects } x \text{ in } \leq p(|x|) \cdot p(|x|) \text{ steps,} \\ x & \dots \mathbb{A} \text{ does not terminate in } \leq p(|x|) \cdot p(|x|) \text{ steps.} \end{cases}$$

In the last case ($K(x) = x$) we have $p(|x|) \cdot p(|x|) \leq f(\kappa(x)) \cdot p(|x|)$, and hence $|K(x)| = |x| \leq p(|x|) \leq f(\kappa(x))$. Therefore K is a *kernel*. \square

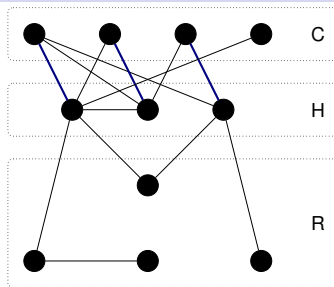
Crown Decomposition and Crown Lemma



A **crown decomposition** of a graph G is a partitioning (C, H, R) of $V(G)$, such that:

- 1 C is nonempty.
- 2 C is an independent set.
- 3 H separates C and R .
- 4 G contains a matching of H into C .

Crown Decomposition and Crown Lemma



A **crown decomposition** of a graph G is a partitioning (C, H, R) of $V(G)$, such that:

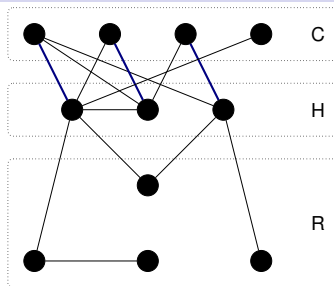
- 1 C is nonempty.
- 2 C is an independent set.
- 3 H separates C and R .
- 4 G contains a matching of H into C .

Crown Lemma (\Leftarrow results by König, Hall)

Let G be a graph with no isolated vertices and with at least $3k + 1$ vertices. There is a polynomial-time algorithm that:

- ▶ either finds a matching of size $k + 1$ in G ;
- ▶ or finds a crown decomposition of G .

Crown Decomposition and Crown Lemma



A **crown decomposition** of a graph G is a partitioning (C, H, R) of $V(G)$, such that:

- 1 C is nonempty.
- 2 C is an independent set.
- 3 H separates C and R .
- 4 G contains a matching of H into C .

Crown Lemma (\Leftarrow results by Kőnig, Hall)

Let G be a graph with no isolated vertices and with at least $3k + 1$ vertices. There is a polynomial-time algorithm that:

- ▶ either finds a matching of size $k + 1$ in G ;
- ▶ or finds a crown decomposition of G .

Exercise

Apply the Crown Lemma to the Vertex Cover Problem.

The (par.) Vertex Cover Problem (smaller kernel)

p-VERTEX-COVER:

Given: A graph G , and an integer k .

Parameter: The integer k .

Question: Does there exists a vertex cover of size at most k ?

The (par.) Vertex Cover Problem (smaller kernel)

p-VERTEX-COVER:

Given: A graph G , and an integer k .

Parameter: The integer k .

Question: Does there exist a vertex cover of size at most k ?

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G - v, k)$

The (par.) Vertex Cover Problem (smaller kernel)

p-VERTEX-COVER:

Given: A graph G , and an integer k .

Parameter: The integer k .

Question: Does there exist a vertex cover of size at most k ?

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G - v, k)$

Rule 2: If $|V(G)| \geq 3k + 1$, apply the Crown Lemma.

- ▶ If it returns a matching of size $k + 1$,
then conclude that (G, k) is a **no-instance**
- ▶ If it returns a **crown decomposition** $V(G) = C \cup H \cup R$:
 - ▶ Pick the vertices in H in the solution

The (par.) Vertex Cover Problem (smaller kernel)

p-VERTEX-COVER:

Given: A graph G , and an integer k .

Parameter: The integer k .

Question: Does there exist a vertex cover of size at most k ?

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G - v, k)$

Rule 2: If $|V(G)| \geq 3k + 1$, apply the Crown Lemma.

- ▶ If it returns a matching of size $k + 1$,
then conclude that (G, k) is a **no-instance**
- ▶ If it returns a **crown decomposition** $V(G) = C \cup H \cup R$:
 - ▶ Pick the vertices in H in the solution
 - ▶ Reduce (G, k) to $(G - H, k - |H|)$

The (par.) Vertex Cover Problem (smaller kernel)

p-VERTEX-COVER:

Given: A graph G , and an integer k .

Parameter: The integer k .

Question: Does there exists a vertex cover of size at most k ?

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G - v, k)$

Rule 2: If $|V(G)| \geq 3k + 1$, apply the Crown Lemma.

- ▶ If it returns a matching of size $k + 1$,
then conclude that (G, k) is a **no-instance**
- ▶ If it returns a **crown decomposition** $V(G) = C \cup H \cup R$:
 - ▶ Pick the vertices in H in the solution
 - ▶ Reduce (G, k) to $(G - H, k - |H|)$
 - ▶ Reduce $(G - H, k - |H|)$ to $(G - H - C, k - |H|)$
by using Rule 1 (note that vertices in C are isolated)

The (par.) Vertex Cover Problem (smaller kernel)

p-VERTEX-COVER:

Given: A graph G , and an integer k .

Parameter: The integer k .

Question: Does there exists a vertex cover of size at most k ?

Rule 1: If G contains an isolated vertex v , delete v from G .
The new instance is $(G - v, k)$

Rule 2: If $|V(G)| \geq 3k + 1$, apply the Crown Lemma.

- ▶ If it returns a matching of size $k + 1$,
then conclude that (G, k) is a **no-instance**
- ▶ If it returns a **crown decomposition** $V(G) = C \cup H \cup R$:
 - ▶ Pick the vertices in H in the solution
 - ▶ Reduce (G, k) to $(G - H, k - |H|)$
 - ▶ Reduce $(G - H, k - |H|)$ to $(G - H - C, k - |H|)$
by using Rule 1 (note that vertices in C are isolated)

Theorem

p-VERTEX-COVER admits a kernel with at most $3k$ vertices.

The (parameterized) Dual-Coloring Problem

p-COLORABILITY:

Given: A graph $G = \langle V, E \rangle$ on n vertices and an integer k .

Parameter: The integer k .

Question: Is G k -colorable?

Definition

Let $k \in \mathbb{N}$. A graph $G = \langle V, E \rangle$ is k -colorable if there is a function $C : V \rightarrow \{1, \dots, k\}$ such that $C(u) \neq C(v)$ for all edges $\{u, v\} \in E$.

The (parameterized) Dual-Coloring Problem

p-DUAL-COLORABILITY:

Given: A graph $G = \langle V, E \rangle$ on n vertices and an integer k .

Parameter: The integer k .

Question: Is G $(n - k)$ -colorable?

Definition

Let $k \in \mathbb{N}$. A graph $G = \langle V, E \rangle$ is k -colorable if there is a function $C : V \rightarrow \{1, \dots, k\}$ such that $C(u) \neq C(v)$ for all edges $\{u, v\} \in E$.

The (parameterized) Dual-Coloring Problem

p-DUAL-COLORABILITY:

Given: A graph $G = \langle V, E \rangle$ on n vertices and an integer k .

Parameter: The integer k .

Question: Is G $(n - k)$ -colorable?

Definition

Let $k \in \mathbb{N}$. A graph $G = \langle V, E \rangle$ is k -colorable if there is a function $C : V \rightarrow \{1, \dots, k\}$ such that $C(u) \neq C(v)$ for all edges $\{u, v\} \in E$.

Exercise

Obtain a kernel with $O(k)$ vertices using crown decomposition.

The Dual-Coloring Problem

Rule 1: Let $I \subseteq V(G)$ be the isolated vertices. Remove I from G , and color them with one color. The new instance is $(G - I, k)$

The Dual-Coloring Problem

Rule 1: Let $I \subseteq V(G)$ be the isolated vertices. Remove I from G , and color them with one color. The new instance is $(G - I, k)$

Rule 2: Consider graph $\overline{G}(V, \overline{E})$ obtained from G by saying that $e \in \overline{E}$ iff $e \notin E$.

If $|V(G)| > 3k$, apply the Crown Lemma to \overline{G} .

- ▶ If it returns a matching of size $k + 1$, then conclude that (G, k) is a **yes-instance**

The Dual-Coloring Problem

Rule 1: Let $I \subseteq V(G)$ be the isolated vertices. Remove I from G , and color them with one color. The new instance is $(G - I, k)$

Rule 2: Consider graph $\overline{G}(V, \overline{E})$ obtained from G by saying that $e \in \overline{E}$ iff $e \notin E$.

If $|V(G)| > 3k$, apply the Crown Lemma to \overline{G} .

- ▶ If it returns a matching of size $k + 1$,
then conclude that (G, k) is a **yes-instance**
- ▶ If it returns **crown decomposition** $V(G) = V(\overline{G}) = C \cup H \cup R$:

The Dual-Coloring Problem

Rule 1: Let $I \subseteq V(G)$ be the isolated vertices. Remove I from G , and color them with one color. The new instance is $(G - I, k)$

Rule 2: Consider graph $\overline{G}(V, \overline{E})$ obtained from G by saying that $e \in \overline{E}$ iff $e \notin E$.

If $|V(G)| > 3k$, apply the Crown Lemma to \overline{G} .

- ▶ If it returns a matching of size $k + 1$,
then conclude that (G, k) is a **yes-instance**
- ▶ If it returns **crown decomposition** $V(G) = V(\overline{G}) = C \cup H \cup R$:
 - ▶ The vertices in H can be saved.

The Dual-Coloring Problem

Rule 1: Let $I \subseteq V(G)$ be the isolated vertices. Remove I from G , and color them with one color. The new instance is $(G - I, k)$

Rule 2: Consider graph $\overline{G}(V, \overline{E})$ obtained from G by saying that $e \in \overline{E}$ iff $e \notin E$.

If $|V(G)| > 3k$, apply the Crown Lemma to \overline{G} .

- ▶ If it returns a matching of size $k + 1$,
then conclude that (G, k) is a **yes-instance**
- ▶ If it returns **crown decomposition** $V(G) = V(\overline{G}) = C \cup H \cup R$:
 - ▶ The vertices in H can be saved.
 - ▶ Reduce (G, k) to $(G - H - C, k - |H|)$ if $|H| < k$, and otherwise to a **yes-instance**

The Dual-Coloring Problem

Rule 1: Let $I \subseteq V(G)$ be the isolated vertices. Remove I from G , and color them with one color. The new instance is $(G - I, k)$

Rule 2: Consider graph $\overline{G}(V, \overline{E})$ obtained from G by saying that $e \in \overline{E}$ iff $e \notin E$.

If $|V(G)| > 3k$, apply the Crown Lemma to \overline{G} .

- ▶ If it returns a matching of size $k + 1$, then conclude that (G, k) is a **yes-instance**
- ▶ If it returns **crown decomposition** $V(G) = V(\overline{G}) = C \cup H \cup R$:
 - ▶ The vertices in H can be saved.
 - ▶ Reduce (G, k) to $(G - H - C, k - |H|)$ if $|H| < k$, and otherwise to a **yes-instance**
 - ▶ Note that the vertices in C belong to a clique in $G(V, E)$, that is we need $|C|$ colors, and that we need different colors for R .

The Dual-Coloring Problem

Rule 1: Let $I \subseteq V(G)$ be the isolated vertices. Remove I from G , and color them with one color. The new instance is $(G - I, k)$

Rule 2: Consider graph $\overline{G}(V, \overline{E})$ obtained from G by saying that $e \in \overline{E}$ iff $e \notin E$.

If $|V(G)| > 3k$, apply the Crown Lemma to \overline{G} .

- ▶ If it returns a matching of size $k + 1$, then conclude that (G, k) is a **yes-instance**
- ▶ If it returns **crown decomposition** $V(G) = V(\overline{G}) = C \cup H \cup R$:
 - ▶ The vertices in H can be saved.
 - ▶ Reduce (G, k) to $(G - H - C, k - |H|)$ if $|H| < k$, and otherwise to a **yes-instance**
 - ▶ Note that the vertices in C belong to a clique in $G(V, E)$, that is we need $|C|$ colors, and that we need different colors for R .

Theorem

p -DUAL-COLORING admits a kernel with at most $3k$ vertices.

Sunflower Lemma

Definition

A **sunflower** with k **petals** and a **core** Y is a collection of sets S_1, \dots, S_k such that $S_i \cap S_j = Y$ for all $i \neq j$. The sets $S_i \setminus Y$ are petals and they must be non-empty.

Sunflower Lemma

Definition

A **sunflower** with k **petals** and a **core** Y is a collection of sets S_1, \dots, S_k such that $S_i \cap S_j = Y$ for all $i \neq j$. The sets $S_i \setminus Y$ are petals and they must be non-empty.

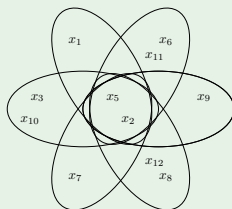
A sunflower with 6 petals and a core $Y = \{x_2, x_5\}$.

$$S_1 = \{x_2, x_3, x_5, x_{10}\}$$

$$S_2 = \{x_1, x_2, x_5\}$$

$$S_3 = \{x_2, x_5, x_6, x_{11}\}$$

...



Sunflower Lemma

Definition

A **sunflower** with k **petals** and a **core** Y is a collection of sets S_1, \dots, S_k such that $S_i \cap S_j = Y$ for all $i \neq j$. The sets $S_i \setminus Y$ are petals and they must be non-empty.

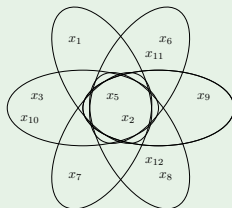
A sunflower with 6 petals and a core $Y = \{x_2, x_5\}$.

$$S_1 = \{x_2, x_3, x_5, x_{10}\}$$

$$S_2 = \{x_1, x_2, x_5\}$$

$$S_3 = \{x_2, x_5, x_6, x_{11}\}$$

...



Sunflower Lemma (Erdős, Rado)

Let \mathcal{A} be a family of sets (without duplicates) over a universe U such that each set in \mathcal{A} has cardinality $= d$.

If $|\mathcal{A}| > d!(k-1)^d$, then \mathcal{A} contains a sunflower with k petals which can be computed in time polynomial in $|\mathcal{A}|$, $|U|$, and k .

Application to d -Hitting Set

Sunflower Lemma (Erdős, Rado)

Let \mathcal{A} be a family of sets (without duplicates) over a universe U such that each set in \mathcal{A} has cardinality $= d$.

If $|\mathcal{A}| > d!(k-1)^d$, then \mathcal{A} contains a sunflower with k petals which can be computed in time polynomial in $|\mathcal{A}|$, $|U|$, and k .

Application to d -Hitting Set

Sunflower Lemma (Erdős, Rado)

Let \mathcal{A} be a family of sets (without duplicates) over a universe U such that each set in \mathcal{A} has cardinality $= d$.

If $|\mathcal{A}| > d!(k-1)^d$, then \mathcal{A} contains a sunflower with k petals which can be computed in time polynomial in $|\mathcal{A}|$, $|U|$, and k .

Parameterized d -Hitting Set Problem

p-d-HITTING-SET:

Given: A family \mathcal{A} of sets over a universe U , where each set has cardinality $\leq d$ and a positive integer k ,

Parameter: The integer k .

Question: Does there exist a subset $H \subseteq U$ of size at most k such that H intersects each set in \mathcal{A} ?

Application to d -Hitting Set

Sunflower Lemma (Erdős, Rado)

Let \mathcal{A} be a family of sets (without duplicates) over a universe U such that each set in \mathcal{A} has cardinality $= d$.

If $|\mathcal{A}| > d!(k-1)^d$, then \mathcal{A} contains a sunflower with k petals which can be computed in time polynomial in $|\mathcal{A}|$, $|U|$, and k .

Parameterized d -Hitting Set Problem

p-d-HITTING-SET:

Given: A family \mathcal{A} of sets over a universe U , where each set has cardinality $\leq d$ and a positive integer k ,

Parameter: The integer k .

Question: Does there exist a subset $H \subseteq U$ of size at most k such that H intersects each set in \mathcal{A} ?

Exercise

Apply the sunflower lemma.

Application to d -Hitting Set

Sunflower Lemma (Erdős, Rado)

Let \mathcal{A} be a family of sets (without duplicates) over a universe U such that each set in \mathcal{A} has cardinality $= d$.

If $|\mathcal{A}| > d!(k-1)^d$, then \mathcal{A} contains a sunflower with k petals which can be computed in time polynomial in $|\mathcal{A}|$, $|U|$, and k .

Parameterized d -Hitting Set Problem

p - d -HITTING-SET:

Given: A family \mathcal{A} of sets over a universe U , where each set has cardinality $\leq d$ and a positive integer k ,

Parameter: The integer k .

Question: Does there exist a subset $H \subseteq U$ of size at most k such that H intersects each set in \mathcal{A} ?

Theorem

p - d -HITTING-SET has a kernel with $\leq d!k^d d$ sets & $\leq d!k^d d^2$ elements.

Application to d -Hitting Set

Observation

If \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of $k+1$ sets, then every hitting set H of \mathcal{A} with $|H| \leq k$ must intersect the core Y of \mathcal{S} . Otherwise it is a **no-instance**, because H cannot intersect each of the $k+1$ petals $S_i \setminus Y$.

Application to d -Hitting Set

Observation

If \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of $k+1$ sets, then every hitting set H of \mathcal{A} with $|H| \leq k$ must intersect the core Y of \mathcal{S} . Otherwise it is a **no-instance**, because H cannot intersect each of the $k+1$ petals $S_i \setminus Y$.

Rule HS.1: Let (U, \mathcal{A}, k) be an instance of d -HITTING SET.

Assume that \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of cardinality $k+1$ with core Y .

Then return (U', \mathcal{A}', k) , where $\mathcal{A}' := (\mathcal{A} \setminus \mathcal{S}) \cup Y$,

$$U' := \bigcup \mathcal{A}' = \bigcup_{X \in \mathcal{A}'} X.$$

Proof (kernel of p - d -HITTING-SET with $\leq d!k^d d$ sets and $\leq d!k^d d^2$ elements).

If for some $d' \in \{1, \dots, d\}$, the number of sets in \mathcal{A} of size $= d'$ is more than $d'!k^{d'}$, then the sunflower lemma yields a sunflower of size $k+1$.

Application to d -Hitting Set

Observation

If \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of $k+1$ sets, then every hitting set H of \mathcal{A} with $|H| \leq k$ must intersect the core Y of \mathcal{S} . Otherwise it is a **no-instance**, because H cannot intersect each of the $k+1$ petals $S_i \setminus Y$.

Rule **HS.1**: Let (U, \mathcal{A}, k) be an instance of d -HITTING SET.

Assume that \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of cardinality $k+1$ with core Y .

Then return (U', \mathcal{A}', k) , where $\mathcal{A}' := (\mathcal{A} \setminus \mathcal{S}) \cup Y$,

$$U' := \bigcup \mathcal{A}' = \bigcup_{X \in \mathcal{A}'} X.$$

Proof (kernel of p -d-HITTING-SET with $\leq d!k^d d$ sets and $\leq d!k^d d^2$ elements).

If for some $d' \in \{1, \dots, d\}$, the number of sets in \mathcal{A} of size $= d'$ is more than $d'!k^{d'}$, then the sunflower lemma yields a sunflower of size $k+1$.

Rule **HS.1** applies.

Application to d -Hitting Set

Observation

If \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of $k+1$ sets, then every hitting set H of \mathcal{A} with $|H| \leq k$ must intersect the core Y of \mathcal{S} . Otherwise it is a **no-instance**, because H cannot intersect each of the $k+1$ petals $S_i \setminus Y$.

Rule HS.1: Let (U, \mathcal{A}, k) be an instance of d -HITTING SET.

Assume that \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of cardinality $k+1$ with core Y .

Then return (U', \mathcal{A}', k) , where $\mathcal{A}' := (\mathcal{A} \setminus \mathcal{S}) \cup Y$,

$$U' := \bigcup \mathcal{A}' = \bigcup_{X \in \mathcal{A}'} X.$$

Proof (kernel of p - d -HITTING-SET with $\leq d!k^d d$ sets and $\leq d!k^d d^2$ elements).

If for some $d' \in \{1, \dots, d\}$, the number of sets in \mathcal{A} of size $= d'$ is more than $d'!k^{d'}$, then the sunflower lemma yields a sunflower of size $k+1$.

Rule **HS.1** applies. By applying this rule exhaustively, we obtain a new family of sets \mathcal{A}' with $\leq d'!k^{d'}$ sets of size $= d'$ for every $d' \in \{1, \dots, d\}$.

Application to d -Hitting Set

Observation

If \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of $k+1$ sets, then every hitting set H of \mathcal{A} with $|H| \leq k$ must intersect the core Y of \mathcal{S} . Otherwise it is a **no-instance**, because H cannot intersect each of the $k+1$ petals $S_i \setminus Y$.

Rule HS.1: Let (U, \mathcal{A}, k) be an instance of d -HITTING SET.

Assume that \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of cardinality $k+1$ with core Y .

Then return (U', \mathcal{A}', k) , where $\mathcal{A}' := (\mathcal{A} \setminus \mathcal{S}) \cup Y$,

$$U' := \bigcup \mathcal{A}' = \bigcup_{X \in \mathcal{A}'} X.$$

Proof (kernel of p - d -HITTING-SET with $\leq d!k^d d$ sets and $\leq d!k^d d^2$ elements).

If for some $d' \in \{1, \dots, d\}$, the number of sets in \mathcal{A} of size $= d'$ is more than $d'!k^{d'}$, then the sunflower lemma yields a sunflower of size $k+1$.

Rule **HS.1** applies. By applying this rule exhaustively, we obtain a new family of sets \mathcal{A}' with $\leq d'!k^{d'}$ sets of size $= d'$ for every $d' \in \{1, \dots, d\}$. Hence $|\mathcal{A}'| \leq d!k^d d$ and $|U'| = d!k^d d^2$.

Application to d -Hitting Set

Observation

If \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of $k+1$ sets, then every hitting set H of \mathcal{A} with $|H| \leq k$ must intersect the core Y of \mathcal{S} . Otherwise it is a **no-instance**, because H cannot intersect each of the $k+1$ petals $S_i \setminus Y$.

Rule HS.1: Let (U, \mathcal{A}, k) be an instance of d -HITTING SET.

Assume that \mathcal{A} contains a sunflower $\mathcal{S} = \{S_1, \dots, S_{k+1}\}$ of cardinality $k+1$ with core Y .

Then return (U', \mathcal{A}', k) , where $\mathcal{A}' := (\mathcal{A} \setminus \mathcal{S}) \cup Y$,
 $U' := \bigcup \mathcal{A}' = \bigcup_{X \in \mathcal{A}'} X$.

Proof (kernel of p - d -HITTING-SET with $\leq d!k^d d$ sets and $\leq d!k^d d^2$ elements).

If for some $d' \in \{1, \dots, d\}$, the number of sets in \mathcal{A} of size $= d'$ is more than $d'!k^{d'}$, then the sunflower lemma yields a sunflower of size $k+1$.

Rule **HS.1** applies. By applying this rule exhaustively, we obtain a new family of sets \mathcal{A}' with $\leq d'!k^{d'}$ sets of size $= d'$ for every $d' \in \{1, \dots, d\}$. Hence $|\mathcal{A}'| \leq d!k^d d$ and $|U'| = d!k^d d^2$.

If $\emptyset \in \mathcal{A}'$ (a sunflower had an empty core), then it is a **no instance**. \square

Course overview

Monday, July 14 10.30 – 12.30	Tuesday, July 15 10.30 – 12.30	Wednesday, July 16 10.30 – 12.30	Thursday, July 17 10.30 – 12.30	Friday, July 18
<i>Algorithmic Techniques</i>		<i>Formal-Method & Algorithmic Techniques</i>		
Introduction & basic FPT results motivation for FPT kernelization, Crown Lemma, Sunflower Lemma	Notions of bounded graph width path-, tree-, clique width, FPT-results by dynamic programming, transferring FPT results betw. width	Algorithmic Meta-Theorems 1st-order logic, monadic 2nd-order logic, FPT-results by Courcelle's Theorems for tree and clique-width	FPT-Intractability Classes & Hierarchies motivation for FP-intractability results, FPT-reductions, class XP (slice-wise polynomial), W- and A-Hierarchies, placing problems on these hierarchies	
				14.30 – 16.30
				examples, question hour