

# Parameterized Algorithms

## FPT-Basics

Thomas Bläsius

# Before we start



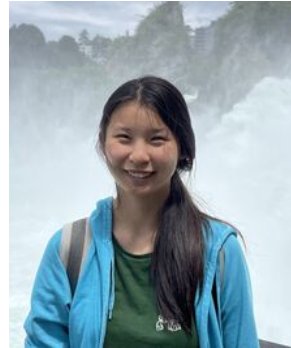
Thomas



Elly



Jean-Pierre



Wendy



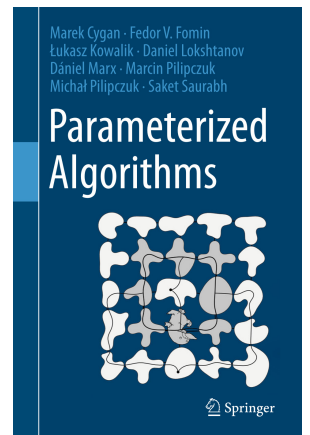
You

## Materials & infos

- slides, exercise sheets on our homepage: [https://scale.iti.kit.edu/teaching/2026ss/param\\_algo/start](https://scale.iti.kit.edu/teaching/2026ss/param_algo/start)
- Book: Parameterized Algorithms
- Discord: <https://discord.gg/y8EksCf4av> (or if you are already on our server: send `!help` `join` to the scale-bot)

## Requirements

- good algorithmic understanding
- no (little) prior knowledge



# Rough Schedule

week $i$							week $i + 1$							week $i + 2$							week $i + 3$						
Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
$(i \text{ even})$							exercise sheet $\frac{i}{2}$														exercise sheet $\frac{i}{2} + 1$						

(we are in week 1 right now)

## Lecture

- lecture with slides
- new topics

## Exercise sheet

- hand in in groups of two or three
- graded by us

## Active session

- if it's not a Holiday
- training additional skills
- curiosities

## Exercise session (Week $i + 1$ )

- with Wendy, Jean-Pierre, and Elly
- recap
- support solving exercise sheets
- additional exercises

## Exam

- oral exam (20 min)
- admission only with exercise certificate

# Exercise certificate

hand in here:

[https://ilias.studium.kit.edu/ilias.php?baseClass=ilrepositorygui&ref\\_id=2902870](https://ilias.studium.kit.edu/ilias.php?baseClass=ilrepositorygui&ref_id=2902870)

**Goal:**  $\frac{1}{2}$  of the points in total **and**  $\frac{1}{4}$  on every exercise sheet

## What if I don't find the solution?

- you get points for explaining what you tried and why it did not work
- and: there are many ways to get support
  - talk to your peers
  - ask in the exercise session or on discord

## What if I can't manage to hand in an exercise sheet?

- sometimes, life can get in the way (for all sorts of reasons, e.g., sickness)
- talk to us, we'll find a solution

we don't want to make your life hard and we also don't bite  
we just want you to learn something and have fun doing so

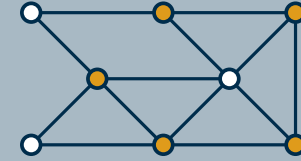
## Our goal

- you spend some time with the content of the lecture and write down your solution
- then, the exercise certificate should not be a big obstacle

# Parameterized perspective

## Problem: $k$ -VERTEX COVER

Does the graph  $G = (V, E)$  have a vertex cover of size  $k$ ?



(vertex set  $V' \subseteq V$  with  $e \cap V' \neq \emptyset$  for all  $e \in E$ )

## Can we solve VERTEX COVER efficiently?

- $k$ -VERTEX COVER is NP-complete

## What if the considered graphs are well-behaved?

- What if  $G$  has small maximum degree?
- What if the minimum vertex cover of  $G$  is small?
- What if  $G$  has small chromatic number?
- Or small clique number? Small treewidth? Small ... ?

## Goal

- analyze running time not only depending on input size, but also take parameter into account

# Fundamental definition: FPT

## Definition

A **parameterized problem** is a language  $L \subseteq \Sigma^* \times \mathbb{N}$ . An algorithm solves  $L$  if it correctly decides  $(x, k) \in L$  for every instance  $(x, k) \in \Sigma^* \times \mathbb{N}$ . The **parameter** of  $(x, k)$  is  $k$ .

## Definition

A parameterized problem is **fixed parameter tractable (FPT)** if it can be solved in  $O(f(k)n^c)$  time, where  $f$  is a computable function,  $n$  is the input size, and  $c$  is a constant.

Which running times belong to FPT-algorithms?

$$A = 2^k(n^4 + k^2) \quad D = k^2 n! \quad G = k! n \log n$$

$$B = k^{\log n} \quad E = n^{17} k^{k!} \quad H = n^k n^2$$

$$C = 2^{k^2 + 2 \log n} \quad F = n^{\log k} \quad I = 2^{k \log n}$$

## Observation for FPT algorithms

- $k$  constant  $\Rightarrow$  polynomial running time
- $k$  constant  $\Rightarrow$  asymptotic running time independent of  $k$

This second point is important! It does not hold for  $n^k$ .

# Example: Parameterized VERTEX COVER

## Definition

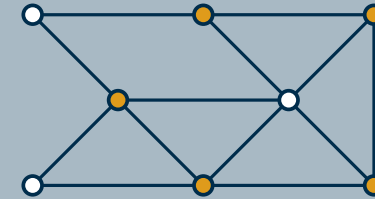
A **parameterized problem** is a language  $L \subseteq \Sigma^* \times \mathbb{N}$ . An algorithm solves  $L$  if it correctly decides  $(x, k) \in L$  for every instance  $(x, k) \in \Sigma^* \times \mathbb{N}$ . The **parameter** of  $(x, k)$  is  $k$ .

## Definition

A parameterized problem is **fixed parameter tractable (FPT)** if it can be solved in  $O(f(k)n^c)$  time, where  $f$  is a computable function,  $n$  is the input size, and  $c$  is a constant.

## Problem: VERTEX COVER

Given a graph  $G = (V, E)$  and a parameter  $k$ . Does  $G$  have a vertex cover of size  $k$ ?



(vertex set  $V' \subseteq V$  with  $e \cap V' \neq \emptyset$  for all  $e \in E$ )

## Complexity

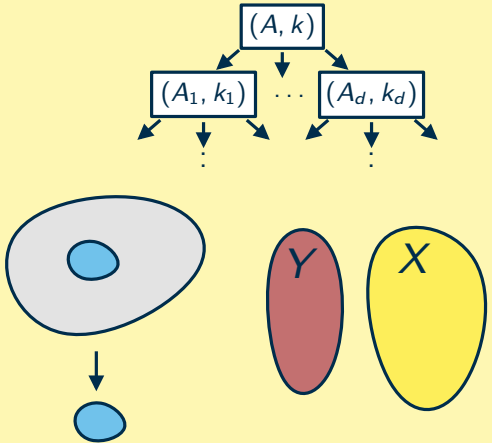
- VERTEX COVER is NP-complete
- enumerating all  $\binom{n}{k}$  subsets of size  $k$  (brute-force):  $O(n^k m)$
- is VERTEX COVER  $\in$  FPT?

poly for constant  $k$

# Content

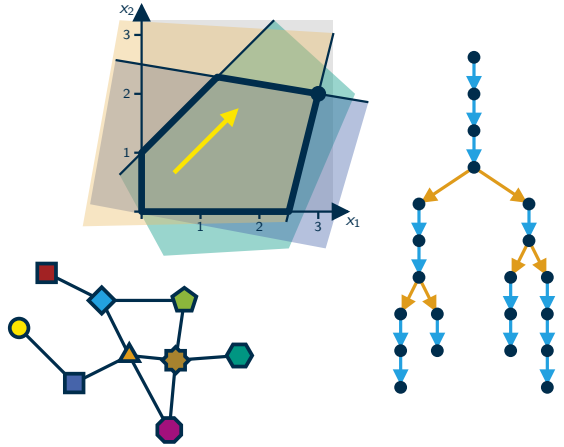
## Basic toolbox

- bounded search trees
- kernelization
- iterative compression



## Extended toolbox

- linear programs
- branch-and-reduce
- color coding



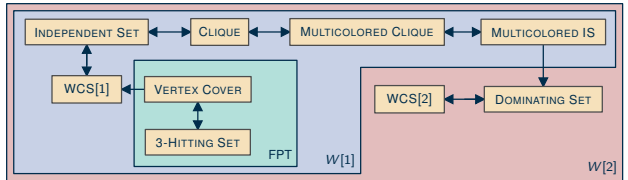
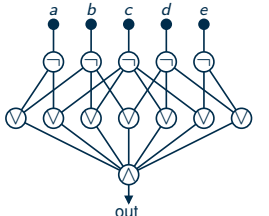
## Tree width

- dynamic programming
- chordal and planar graphs
- Courcelle's theorem



## Lower bounds

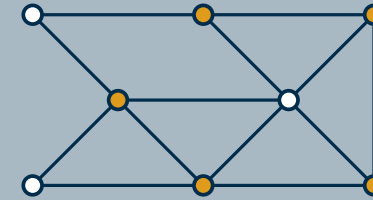
- kernel lower bounds
- parameterized reductions
- circuits and the W-hierarchy
- ETH and SETH



# Bounded search tree

## Problem: VERTEX COVER

Given a graph  $G = (V, E)$  and a parameter  $k$ . Does  $G$  have a vertex cover of size  $k$ ?



(vertex set  $V' \subseteq V$  with  $e \cap V' \neq \emptyset$  for all  $e \in E$ )

Is there a vertex cover of size at most  $k = 3$ ?



edge  $\{u, v\}$  is uncovered  
 $\Rightarrow u$  or  $v$  must be selected  
 $\rightarrow$  binary decision



solution of size 3

## Simple branching algorithm

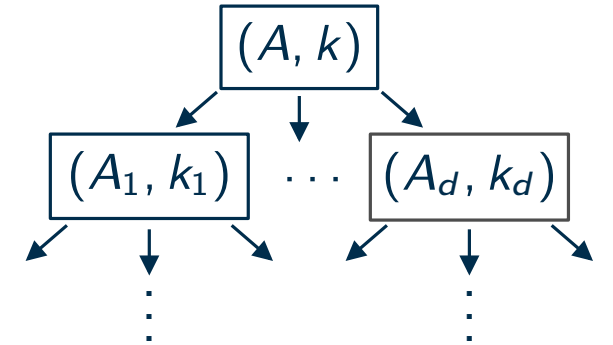
- start:  $V' = \emptyset \rightarrow$  no edge covered
- step: branch on uncovered edge
- stop if: all edges covered or already  $k$  vertices selected

running time:  $O(2^k m)$

# Basic FPT-techniques

## Bounded search tree

- for instance  $(A, k)$  compute in time  $n^c$  instances  $(A_1, k_1), \dots, (A_d, k_d)$ , such that:  
 $(A, k)$  yes instance  $\Leftrightarrow (A_i, k_i)$  yes instance for an  $i \in [1, d]$
- bound  $d$  by a function  $f_1(k)$
- bound tree height by  $f_2(k)$  (example: parameter shrinks in each step)
- $\Rightarrow$  FPT-algo with running time  $O(f_1(k)^{f_2(k)} n^c)$



# Kernelization for VERTEX COVER

**Reduction rule 1:** delete an isolated vertex

**Reduction rule 2**

- add a vertex  $v$  with  $\deg(v) > k$  to the VC
- new instance: remove  $v$ , reduce  $k$  by 1

**Problem: VERTEX COVER**

Given a graph  $G = (V, E)$  and a parameter  $k$ .

Does  $G$  have a vertex cover of size  $k$ ?

(vertex set  $V' \subseteq V$  with  $e \cap V' \neq \emptyset$  for all  $e \in E$ )



**Lemma**

Reduction rule 1 is safe.

(new instance is YES-instance  $\Leftrightarrow$  old instance is YES-instance)

**Proof:** obvious

# Kernelization for VERTEX COVER

**Reduction rule 1:** delete an isolated vertex

## Reduction rule 2

- add a vertex  $v$  with  $\deg(v) > k$  to the VC
- new instance: remove  $v$ , reduce  $k$  by 1

### Problem: VERTEX COVER

Given a graph  $G = (V, E)$  and a parameter  $k$ .

Does  $G$  have a vertex cover of size  $k$ ?

(vertex set  $V' \subseteq V$  with  $e \cap V' \neq \emptyset$  for all  $e \in E$ )



## Lemma

Reduction rule 2 is safe.

(new instance is YES-instance  $\Leftrightarrow$  old instance is YES-instance)

## Proof

- either  $v$  or every neighbor of  $v$  must be part of the vertex cover
- if we don't choose  $v$ , the vertex cover will be too large ( $\deg(v) > k$ )

# Kernelization for VERTEX COVER

**Reduction rule 1:** delete an isolated vertex

**Reduction rule 2**

- add a vertex  $v$  with  $\deg(v) > k$  to the VC
- new instance: remove  $v$ , reduce  $k$  by 1

**Problem: VERTEX COVER**

Given a graph  $G = (V, E)$  and a parameter  $k$ .

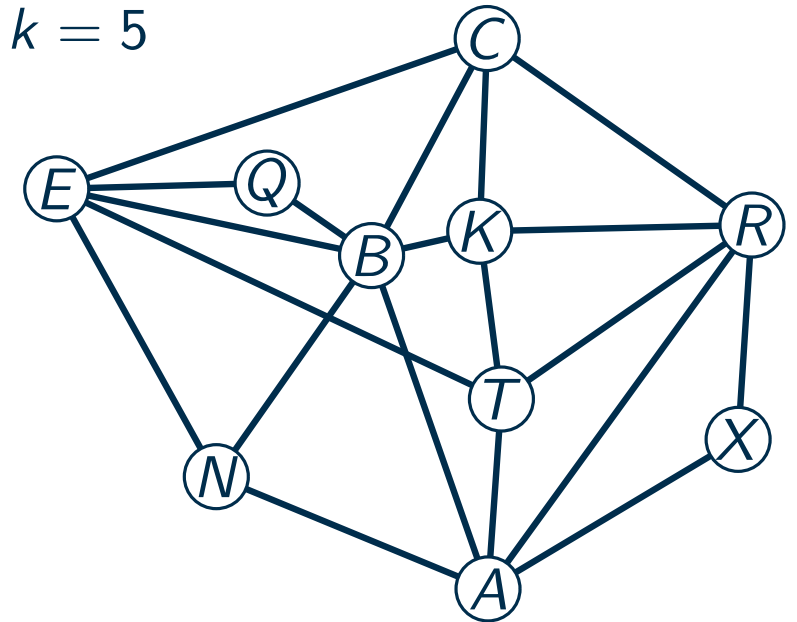
Does  $G$  have a vertex cover of size  $k$ ?

(vertex set  $V' \subseteq V$  with  $e \cap V' \neq \emptyset$  for all  $e \in E$ )



**Apply reduction rule 2 exhaustively**

$k = 5$



**Solution:**

# Kernelization for VERTEX COVER

**Reduction rule 1:** delete an isolated vertex

**Reduction rule 2**

- add a vertex  $v$  with  $\deg(v) > k$  to the VC
- new instance: remove  $v$ , reduce  $k$  by 1

**Reduction rule 3**

- rules 1 and 2 not applicable and  $m > k^2$ : reduce to NO-instance of constant size

## Lemma

Reduction rule 3 is safe.

(new instance is YES-instance  $\Leftrightarrow$  old instance is YES-instance)

## Proof

- rule 2 not applicable  $\Rightarrow \deg(v) \leq k$  for all  $v \in V$
- every vertex covers at most  $k$  edges
- $k$  vertices cover at most  $k^2$  edges  $\Rightarrow$  NO-instance, as  $m > k^2$

### Problem: VERTEX COVER

Given a graph  $G = (V, E)$  and a parameter  $k$ .  
Does  $G$  have a vertex cover of size  $k$ ?  
(vertex set  $V' \subseteq V$  with  $e \cap V' \neq \emptyset$  for all  $e \in E$ )



# Kernelization for VERTEX COVER

**Reduction rule 1:** delete an isolated vertex

## Reduction rule 2

- add a vertex  $v$  with  $\deg(v) > k$  to the VC
- new instance: remove  $v$ , reduce  $k$  by 1

## Reduction rule 3

- rules 1 and 2 not applicable and  $m > k^2$ : reduce to NO-instance of constant size

## Kernelization

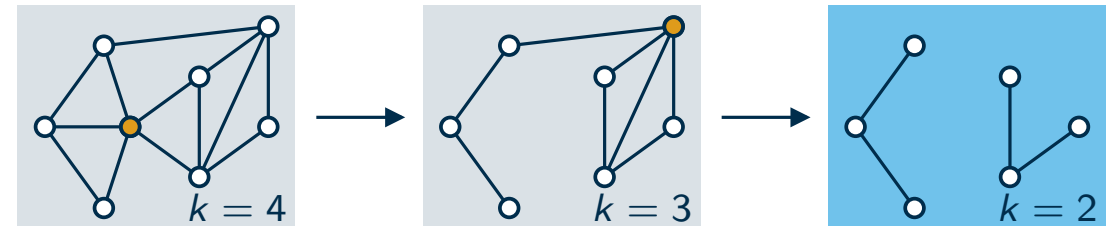
- apply reduction rules exhaustively
- it remains a **problem core** of size  $O(k^2)$

### Theorem

VERTEX COVER has a kernel with  $O(k^2)$  vertices and edges. It can be computed in  $O(m)$  time.

### Problem: VERTEX COVER

Given a graph  $G = (V, E)$  and a parameter  $k$ . Does  $G$  have a vertex cover of size  $k$ ?  
(vertex set  $V' \subseteq V$  with  $e \cap V' \neq \emptyset$  for all  $e \in E$ )



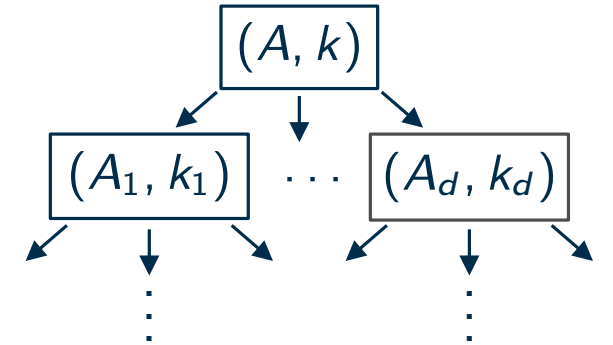
### FPT algorithm

- brute-force on the kernel:  $O(2^{k^2} k^2)$
- or better: search tree on the kernel

# Basic FPT-techniques

## Bounded search tree

- for instance  $(A, k)$  compute in time  $n^c$  instances  $(A_1, k_1), \dots, (A_d, k_d)$ , such that:  
 $(A, k)$  yes instance  $\Leftrightarrow (A_i, k_i)$  yes instance for an  $i \in [1, d]$
- bound  $d$  by a function  $f_1(k)$
- bound tree height by  $f_2(k)$  (example: parameter shrinks in each step)
- $\Rightarrow$  FPT-algo with running time  $O(f_1(k)^{f_2(k)} n^c)$



## Kernelization

- successively apply safe reduction rules
- show: what remains is a core, whose size only depends on  $k$

# VERTEX COVER COMPRESSION

## Problem: VERTEX COVER COMPRESSION

Given a graph  $G$ , a parameter  $k$ , and a VC  $Z$  of size  $k + 1$ . Is there are VC of size  $k$ ?

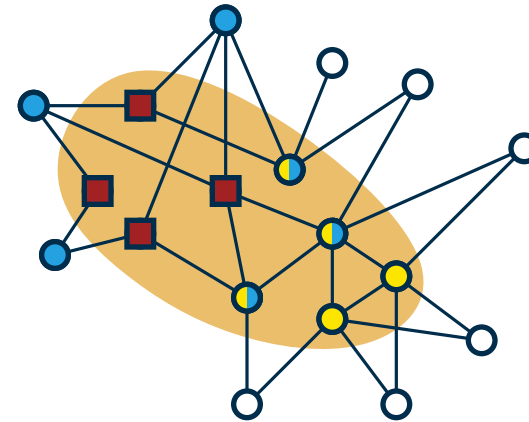
### Idea

- guess a subset  $X \subseteq Z$ ;  $Y = Z \setminus X$
- is there a VC  $Z'$  mit  $|Z'| \leq k$  und  $Z' \cap Z = X$ ?

**Case 1:**  $G[Y]$  has edges  $\Rightarrow Z'$  does not exist

**Case 2:** otherwise the following holds:

- $Z'$  must contain all neighbors  $N(Y)$  of  $Y$
- and  $X \cup N(Y)$  is a VC
- the VC  $Z'$  exists  $\Leftrightarrow |X \cup N(Y)| \leq k$



### Algorithm

- apply this procedure to every subset  $X \subseteq Z$
- there are  $2^{k+1}$  such subsets
- $O(m)$  for each  $X \Rightarrow$  overall  $O(2^k m)$  time

# Iterative compression

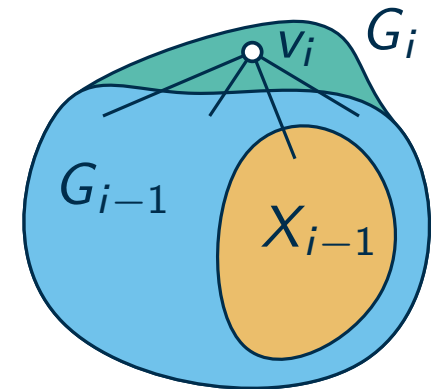
## Problem: VERTEX COVER COMPRESSION

Given a graph  $G$ , a parameter  $k$ , and a VC  $Z$  of size  $k + 1$ . Is there are VC of size  $k$ ?

**Just seen:**  $O(2^k m)$ -algorithm for VERTEX COVER COMPRESSION

## Algorithm for VERTEX COVER

- iterate over the subgraphs  $G_i$  induced by the vertices  $v_1, \dots, v_i$
- start: in  $G_k$ , all vertices form a VC of size  $k$
- assumption for  $i > k$ : for  $G_{i-1}$ , we know a VC  $X_{i-1}$  of size  $k$
- $\Rightarrow X_{i-1} \cup \{v_i\}$  is a VC of size  $k + 1$  in  $G_i$
- use algo for VERTEX COVER COMPRESSION:
  - case 1:  $G_i$  has a VC of size  $k \Rightarrow$  continue with  $G_{i+1}$
  - case 2:  $G_i$  has no VC of size  $k \Rightarrow G$  has no VC of size  $k \Rightarrow$  stop



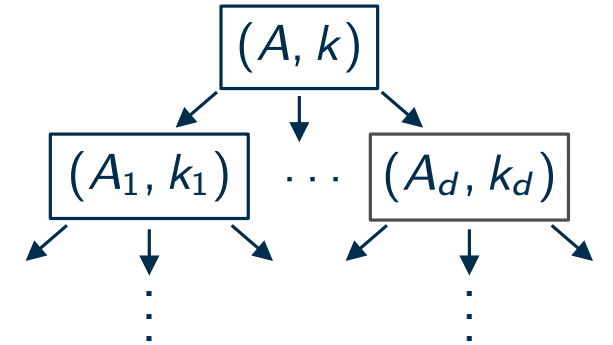
Running time?

$O(2^k nm)$

# Basic FPT-techniques

## Bounded search tree

- for instance  $(A, k)$  compute in time  $n^c$  instances  $(A_1, k_1), \dots, (A_d, k_d)$ , such that:  
 $(A, k)$  yes instance  $\Leftrightarrow (A_i, k_i)$  yes instance for an  $i \in [1, d]$
- bound  $d$  by a function  $f_1(k)$
- bound tree height by  $f_2(k)$  (example: parameter shrinks in each step)
- $\Rightarrow$  FPT-algo with running time  $O(f_1(k)^{f_2(k)} n^c)$



## Kernelization

- successively apply safe reduction rules
- show: what remains is a core, whose size only depends on  $k$

## Iterative compression

- compression algo: solve the problem under the assumption of knowing a slightly larger solution
- grow the instance step by step and maintain a small solution via repeated compression

# Wrap-Up

**VERTEX COVER  $\in$  FPT**  
(when parameterized with the solution size!)

# FAQ: How does FPT relate to NP?

## Formally correct answer: It does not

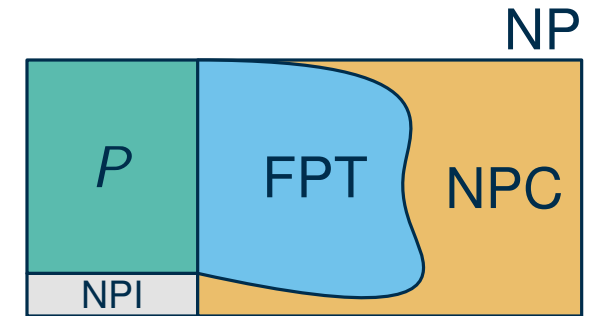
- $NP \cap FPT = \emptyset$  (NP is a set of problems, FPT is a set of parameterized problems)
- if  $L$  is decidable, then there are parameterizations  $L'$  and  $L''$  with:
  - $L' \in FPT$  (running time of the algorithm as parameter)
  - $L'' \notin FPT$ , unless  $L \in P$  (constant parameter)

## Typical situation: What is the goal of FPT?

- solve problems in NPC in reasonable time
- we mainly consider problems in NPC
- good intuition: FPT builds a bridge between P and NPC (although this is a technically malformed statement)

## Bonus info: para-NP

- para-NP is the equivalent of NP for parameterized problems
- FPT relates to para-NP like P to NP



# FAQ: Why must $f$ be computable?

## Definition

A parameterized problem is **fixed parameter tractable (FPT)** if it can be solved in  $O(f(k)n^c)$  time, where  $f$  is a computable function,  $n$  is the input size, and  $c$  is a constant.

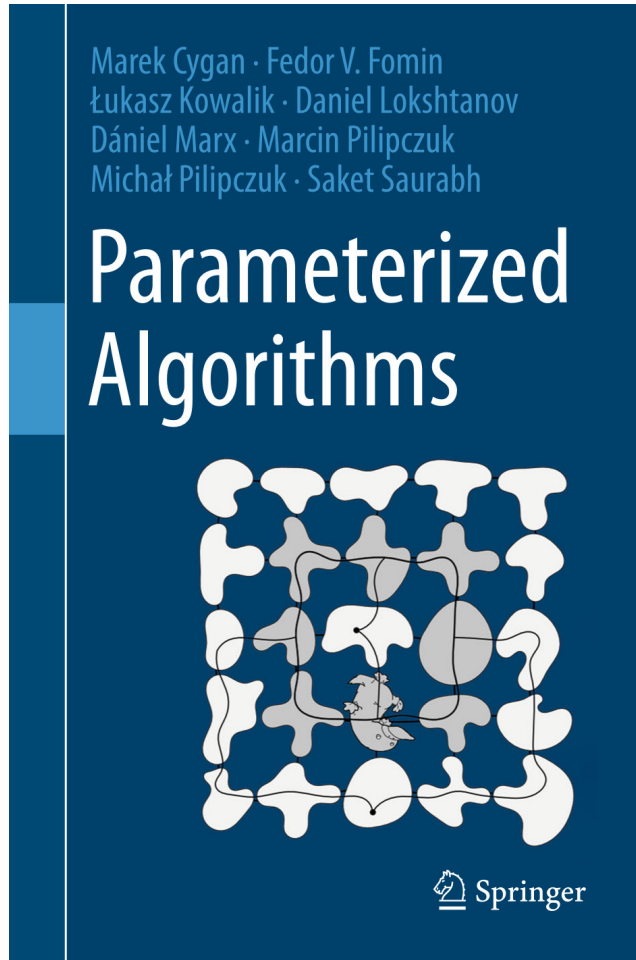
## How can $f$ be not computable?

- your algorithm: solves some parameterized problem for input  $(G, k)$
- you show: for every fixed  $k$ , it runs in  $O(n^c)$  time
- your proof uses something like the graph minors theorem (every class of graphs closed under taking minors is characterized by a finite set of forbidden minors)
- you don't know what  $f$  is, but maybe you can show that busy beaver is an upper bound for  $f$

## Weaker variants of FPT

- $f$  not being computable makes your result slightly weaker
- even weaker: non-uniform (i.e., different algorithms for different  $k$ )
- also see: Section 2.2 of Downey & Fellows book

# Recommended Books



## Parameterized Algorithms

- excellent book
- (almost) all discussed topics can be found there
- free for KIT students: [link.springer.com/book/10.1007/978-3-319-21275-3](https://link.springer.com/book/10.1007/978-3-319-21275-3)

## Fundamentals of Parameterized Complexity

- slightly older book
- also free for KIT students:  
<https://link.springer.com/book/10.1007/978-1-4471-5559-1>

