Introduction to Parameterized Complexity

Ignasi Sau

CNRS, LIRMM, Montpellier, France

UFMG
Belo Horizonte, February 2018

Outline of the talk

1. Why parameterized complexity?
2. Basic definitions
3. Kernelization
4. Some techniques

Next section is...

Some history of complexity: NP-completeness

- Cook-Levin Theorem (1971): the SAT problem is NP-complete.

Some history of complexity: NP-completeness

- Cook-Levin Theorem (1971): the SAT problem is NP-complete.
- Karp (1972): list of 21 *important* NP-complete problems.

Some history of complexity: NP-completeness

- Cook-Levin Theorem (1971): the SAT problem is NP-complete.
- Karp (1972): list of 21 *important* NP-complete problems.
- Nowadays, literally thousands of problems are known to be NP-hard: unless $\mathsf{P} = \mathsf{NP}$, they cannot be solved in polynomial time.

Some history of complexity: NP-completeness

- Cook-Levin Theorem (1971): the SAT problem is NP-complete.
- Karp (1972): list of 21 *important* NP-complete problems.
- Nowadays, literally thousands of problems are known to be NP-hard: unless $\mathsf{P} = \mathsf{NP}$, they cannot be solved in polynomial time.
- But what does it mean for a problem to be NP-hard?

  No algorithm solves all instances optimally in polynomial time.

Some history of complexity: NP-completeness

- Cook-Levin Theorem (1971): the SAT problem is NP-complete.
- Karp (1972): list of 21 *important* NP-complete problems.
- Nowadays, literally thousands of problems are known to be NP-hard: unless $\mathsf{P} = \mathsf{NP}$, they cannot be solved in polynomial time.
- But what does it mean for a problem to be NP-hard?

  No algorithm solves $\boxed{\text{all}}$ instances optimally in polynomial time.

Are all instances really hard to solve?

Maybe there are relevant subsets of instances that can be solved efficiently.

Are all instances really hard to solve?

Maybe there are relevant subsets of instances that can be solved efficiently.

- VLSI design: the number of circuit layers is usually ≤ 10. The problem is NP-complete, but if we fix the number of layers, it is tractable.

Are all instances really hard to solve?

Maybe there are relevant subsets of instances that can be solved efficiently.

- VLSI design: the number of circuit layers is usually ≤ 10. The problem is NP-complete, but if we fix the number of layers, it is tractable.
- Computational biology: Real instances of DNA chain reconstruction have special properties (treewidth ≤ 11) that allow for efficient algorithms.

Are all instances really hard to solve?

Maybe there are relevant subsets of instances that can be solved efficiently.

- VLSI design: the number of circuit layers is usually ≤ 10. The problem is NP-complete, but if we fix the number of layers, it is tractable.
- Computational biology: Real instances of DNA chain reconstruction have special properties (treewidth ≤ 11) that allow for efficient algorithms.
- Robotics: The number of degrees of freedom in motion planning problems is ≤ 10. These problems become tractable under this restriction.

Are all instances really hard to solve?

Maybe there are relevant subsets of instances that can be solved efficiently.

- VLSI design: the number of circuit layers is usually ≤ 10. The problem is NP-complete, but if we fix the number of layers, it is tractable.
- Computational biology: Real instances of DNA chain reconstruction have special properties (treewidth ≤ 11) that allow for efficient algorithms.
- Robotics: The number of degrees of freedom in motion planning problems is ≤ 10. These problems become tractable under this restriction.
- Compilers: Checking compatibility of type declarations is hard, but usually the depth of type declarations is ≤ 10: the problem becomes tractable.

Are all instances really hard to solve?

Maybe there are relevant subsets of instances that can be solved efficiently.

- VLSI design: the number of circuit layers is usually ≤ 10. The problem is NP-complete, but if we fix the number of layers, it is tractable.
- Computational biology: Real instances of DNA chain reconstruction have special properties (treewidth ≤ 11) that allow for efficient algorithms.
- Robotics: The number of degrees of freedom in motion planning problems is ≤ 10. These problems become tractable under this restriction.
- Compilers: Checking compatibility of type declarations is hard, but usually the depth of type declarations is ≤ 10: the problem becomes tractable.

$\boxed{\text{Summary}}$ In many applications, not only the total size of the instance matters, but also the value of an additional parameter.

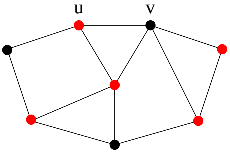
Three famous NP-hard problems

Let us focus on the following three problems by considering a parameter k:

Three famous NP-hard problems

Let us focus on the following three problems by considering a parameter k:

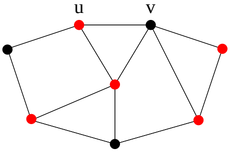
- k-VERTEX COVER: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, covering $E(G)$?



Three famous NP-hard problems

Let us focus on the following three problems by considering a parameter k:

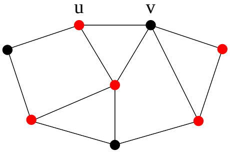
- k-Vertex Cover: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, covering $E(G)$?



- k-Independent Set: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise non-adjacent vertices?

Three famous NP-hard problems

Let us focus on the following three problems by considering a parameter k:



- k-Vertex Cover: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, covering $E(G)$?

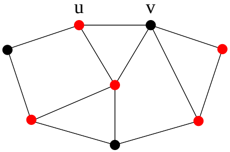
- k-Independent Set: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise non-adjacent vertices?

- Vertex k-Coloring: Can the vertices of a graph be colored with at most k colors, so that any two adjacent vertices get different colors?

Three famous NP-hard problems

Let us focus on the following three problems by considering a parameter k:

- k-Vertex Cover: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, covering $E(G)$?



- k-Independent Set: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise non-adjacent vertices?

- Vertex k-Coloring: Can the vertices of a graph be colored with at most k colors, so that any two adjacent vertices get different colors?

These three problems are NP-hard, but are they $\boxed{\text{equally}}$ hard?

Vertex k-Coloring: Can the vertices of a graph be colored with at most k colors, so that any two adjacent vertices get different colors?

VERTEX k-COLORING: Can the vertices of a graph be colored with at most k colors, so that any two adjacent vertices get different colors?

The problem is already NP-hard for fixed $k = 3$.

VERTEX k-COLORING

VERTEX k-COLORING: Can the vertices of a graph be colored with at most k colors, so that any two adjacent vertices get different colors?

The problem is already NP-hard for fixed $k = 3$.

For fixed k, there is no poly-time algorithm (unless $P = NP$).

k-INDEPENDENT SET

k-INDEPENDENT SET: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise non-adjacent vertices?

k-Independent Set

k-Independent Set: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise non-adjacent vertices?

We can generate all subsets of k vertices, and check whether they induce an independent set or not.

k-Independent Set

k-Independent Set: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \geq k$, of pairwise non-adjacent vertices?

We can generate all subsets of k vertices, and check whether they induce an independent set or not.

$$\boxed{\text{Running time: } O\left(\binom{n}{k} \cdot k^2\right) = O(n^k \cdot k^2)}$$

k-VERTEX COVER: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, covering all the edges of G?

k-Vertex Cover

k-Vertex Cover: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, covering all the edges of G?

Easy branching rule: Let (G, k) be an instance and let $e = \{u, v\}$ be an edge of G.

k-Vertex Cover

k-Vertex Cover: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, covering all the edges of G?

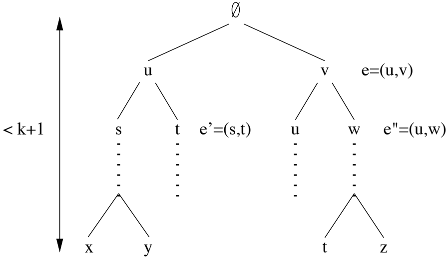
Easy branching rule: Let (G, k) be an instance and let $e = \{u, v\}$ be an edge of G. Then branch into the two smaller instances

$$(G - u, k - 1) \text{ and } (G - v, k - 1)$$

k-VERTEX COVER: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, covering all the edges of G?

Easy branching rule: Let (G, k) be an instance and let $e = \{u, v\}$ be an edge of G. Then branch into the two smaller instances

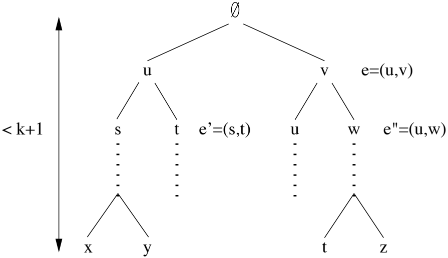
$$(G - u, k - 1) \text{ and } (G - v, k - 1)$$



k-VERTEX COVER: Does a graph G contain a set $S \subseteq V(G)$, with $|S| \leq k$, covering all the edges of G?

Easy branching rule: Let (G, k) be an instance and let $e = \{u, v\}$ be an edge of G. Then branch into the two smaller instances

$$(G - u, k - 1) \text{ and } (G - v, k - 1)$$



Running time: $O(2^k \cdot (m + n))$

Here, $n = |V(G)|$ and $m = |E(G)|$

We get three very different running times

Summarizing:

- VERTEX k-COLORING: NP-hard for fixed $k = 3$.
- k-INDEPENDENT SET: Solvable in time $O(k^2 \cdot n^k)$
- k-VERTEX COVER: Solvable in time $O(2^k \cdot (m + n))$

We get three very different running times

Summarizing:

- Vertex k-Coloring: NP-hard for fixed $k = 3$.
- k-Independent Set: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.
- k-Vertex Cover: Solvable in time $O(2^k \cdot (m + n)) = f(k) \cdot n^{O(1)}$.

We get three very different running times

Summarizing:

- VERTEX k-COLORING: NP-hard for fixed $k = 3$.
- k-INDEPENDENT SET: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.
- k-VERTEX COVER: Solvable in time $O(2^k \cdot (m + n)) = f(k) \cdot n^{O(1)}$.

The behavior of these three NP-hard problems is very different.

Comparison between $O(2^k \cdot n)$ and $O(n^{k+1})$

The behavior of these two types of functions is dramatically different:

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	625	2.500	5.625
$k = 3$	15.625	125.000	421.875
$k = 5$	390.625	6.250.000	31.640.623
$k = 10$	$1,9 \times 10^{12}$	$9,8 \times 10^{14}$	$3,7 \times 10^{16}$
$k = 20$	$1,8 \times 10^{26}$	$9,5 \times 10^{31}$	$2,1 \times 10^{35}$

The ratio $\frac{n^{k+1}}{2^k \cdot n}$ for several values of n and k.

Next section is...

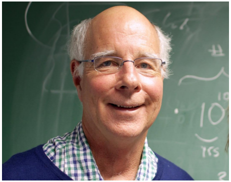
The area of parameterized complexity

$\boxed{\text{Idea}}$ Measure the complexity of an algorithm in terms of the input size and an additional parameter.

The area of parameterized complexity

$\boxed{\text{Idea}}$ Measure the complexity of an algorithm in terms of the input size and an additional parameter.

This theory started in the late 80's, by Downey and Fellows:



Today, it is a well-established area with hundreds of articles published every year in the most prestigious TCS journals and conferences.

Parameterized problems

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$,
where Σ is a fixed, finite alphabet.

Parameterized problems

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$,
where Σ is a fixed, finite alphabet.

For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the parameter.

Parameterized problems

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a fixed, finite alphabet.

For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the parameter.

Examples:

- Decide whether a graph G has an independent set (or clique) of size at least k.
- Decide whether a graph G has a vertex cover of size at most k.

Parameterized problems

A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$,
where Σ is a fixed, finite alphabet.

For an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, k is called the parameter.

Examples:

- Decide whether a graph G has an independent set (or clique) of size at least k.
- Decide whether a graph G has a vertex cover of size at most k.
- Decide whether a graph G has a clique of size at least k, parameterized by the maximum degree Δ of G.
- Decide whether a graph G has a clique of size at least k, parameterized by the treewidth $\mathrm{tw}(G)$ of G.

Classes FPT and XP

A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is fixed-parameter tractable (FPT) if there exists an algorithm $\mathcal{A}$ (FPT algorithm), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ decides whether $(x, k) \in L$ in time bounded by

$$f(k) \cdot |(x, k)|^c.$$

Classes FPT and XP

A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is fixed-parameter tractable (FPT) if there exists an algorithm $\mathcal{A}$ (FPT algorithm), a computable function $f : \mathbb{N} \to \mathbb{N}$, and a constant c such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ decides whether $(x, k) \in L$ in time bounded by

$$f(k) \cdot |(x, k)|^c.$$

A parameterized problem $L \subseteq \Sigma^* \times \mathbb{N}$ is slice-wise polynomial (XP) if there exists an algorithm $\mathcal{A}$ (XP algorithm) and two computable functions $f, g : \mathbb{N} \to \mathbb{N}$ such that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ decides whether $(x, k) \in L$ in time bounded by

$$f(k) \cdot |(x, k)|^{g(k)}.$$

Now we can classify the previous problems

- k-VERTEX COVER: Solvable in time $O(2^k \cdot (m+n)) = f(k) \cdot n^{O(1)}$.

  $\boxed{\text{The problem is FPT.}}$

- k-INDEPENDENT SET: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

  $\boxed{\text{The problem is XP.}}$

- VERTEX k-COLORING: NP-hard for fixed $k = 3$.

Now we can classify the previous problems

- k-Vertex Cover: Solvable in time $O(2^k \cdot (m+n)) = f(k) \cdot n^{O(1)}$.

  The problem is FPT.

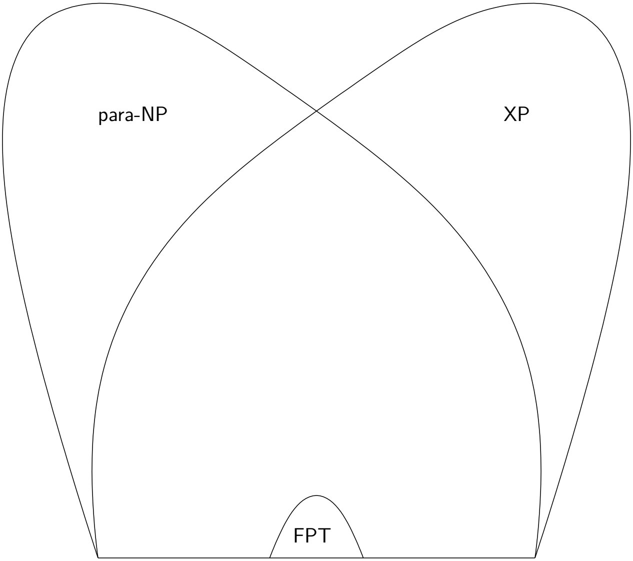
- k-Independent Set: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

  The problem is XP.

- Vertex k-Coloring: NP-hard for fixed $k = 3$.

  Such problems are called para-NP-hard.

Summary: FPT, XP, and para-NP



Are all parameterized problems FPT?

k-Independent Set: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Are all parameterized problems FPT?

k-CLIQUE: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Are all parameterized problems FPT?

k-Clique: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why k-Clique may not be FPT?

Are all parameterized problems FPT?

k-CLIQUE: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why k-CLIQUE may not be FPT?

So far, nobody has managed to find an FPT algorithm.

(also, nobody has found a poly-time algorithm for 3-SAT)

Are all parameterized problems FPT?

k-CLIQUE: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why k-CLIQUE may not be FPT?

So far, nobody has managed to find an FPT algorithm.

(also, nobody has found a poly-time algorithm for 3-SAT)

Working hypothesis of parameterized complexity: $\boxed{k\text{-CLIQUE is not FPT}}$

Are all parameterized problems FPT?

k-CLIQUE: Solvable in time $O(k^2 \cdot n^k) = f(k) \cdot n^{g(k)}$.

Why k-CLIQUE may not be FPT?

So far, nobody has managed to find an FPT algorithm.

(also, nobody has found a poly-time algorithm for 3-SAT)

Working hypothesis of parameterized complexity: $\boxed{k\text{-CLIQUE is not FPT}}$

(in classical complexity: 3-SAT cannot be solved in poly-time)

How to transfer the hardness among parameterized problems?

How to transfer the hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A parameterized reduction from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

How to transfer the hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A parameterized reduction from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

1. (x, k) is a $\textsc{Yes}$-instance of $A \Leftrightarrow (x', k')$ is a $\textsc{Yes}$-instance of B,

How to transfer the hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A parameterized reduction from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

1. (x, k) is a YES-instance of $A \Leftrightarrow (x', k')$ is a YES-instance of B,
2. $k' \leq g(k)$ for some computable function g, and

How to transfer the hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A parameterized reduction from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

1. (x, k) is a Yes-instance of $A \Leftrightarrow (x', k')$ is a Yes-instance of B,
2. $k' \leq g(k)$ for some computable function g, and
3. the running time is $f(k) \cdot |x|^{O(1)}$ for some computable function f.

How to transfer the hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A parameterized reduction from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

1. (x, k) is a $\textsc{Yes}$-instance of $A \Leftrightarrow (x', k')$ is a $\textsc{Yes}$-instance of B,
2. $k' \leq g(k)$ for some computable function g, and
3. the running time is $f(k) \cdot |x|^{O(1)}$ for some computable function f.

W[1]-hard problem: $\exists$ parameterized reduction from k-$\textsc{Clique}$ to it.

How to transfer the hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A parameterized reduction from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

1. (x, k) is a YES-instance of $A \Leftrightarrow (x', k')$ is a YES-instance of B,
2. $k' \leq g(k)$ for some computable function g, and
3. the running time is $f(k) \cdot |x|^{O(1)}$ for some computable function f.

W[1]-hard problem: $\exists$ parameterized reduction from k-CLIQUE to it.

W[2]-hard problem: $\exists$ param. reduction from k-DOMINATING SET to it.

How to transfer the hardness among parameterized problems?

Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A parameterized reduction from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

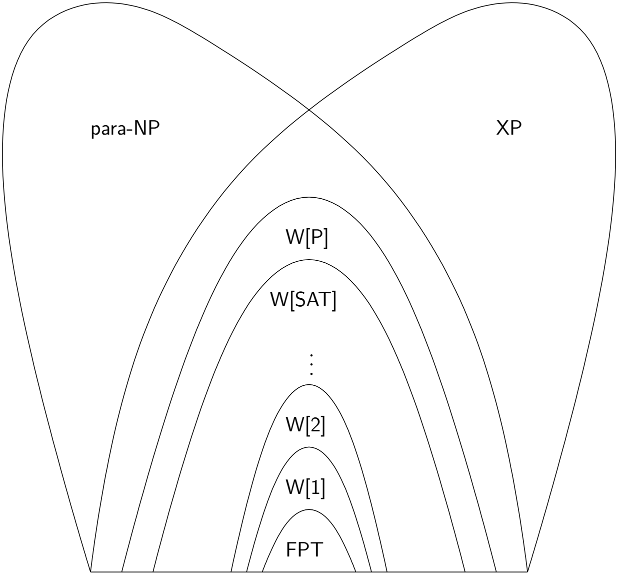
1. (x, k) is a YES-instance of $A \Leftrightarrow (x', k')$ is a YES-instance of B,
2. $k' \leq g(k)$ for some computable function g, and
3. the running time is $f(k) \cdot |x|^{O(1)}$ for some computable function f.

W[1]-hard problem: $\exists$ parameterized reduction from k-CLIQUE to it.

W[2]-hard problem: $\exists$ param. reduction from k-DOMINATING SET to it.

Being W[i]-hard is a strong evidence of not being FPT.

Hierarchy of classes of parameterized problems



Next section is...

1. Why parameterized complexity?
2. Basic definitions
3. Kernelization
4. Some techniques

Kernelization

$\boxed{\text{Idea}}$ polynomial-time preprocessing.

Kernelization

$\boxed{\text{Idea}}$ polynomial-time preprocessing.

A kernel for a parameterized problem L is an algorithm A that, given an instance (x, k) of L, works in polynomial time and returns an equivalent instance (x', k') of L such that $|x'| + k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

Kernelization

$\boxed{\text{Idea}}$ polynomial-time preprocessing.

A kernel for a parameterized problem L is an algorithm A that, given an instance (x, k) of L, works in polynomial time and returns an equivalent instance (x', k') of L such that $|x'| + k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

The function g is called the size of the kernel.

Kernelization

$\boxed{\text{Idea}}$ polynomial-time preprocessing.

A kernel for a parameterized problem L is an algorithm A that, given an instance (x, k) of L, works in polynomial time and returns an equivalent instance (x', k') of L such that $|x'| + k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

The function g is called the size of the kernel.

If g is a polynomial, then we speak about a polynomial kernel.

$\boxed{\text{Idea}}$ polynomial-time preprocessing.

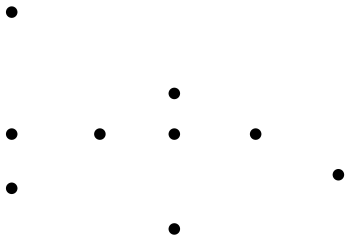
A kernel for a parameterized problem L is an algorithm A that, given an instance (x, k) of L, works in polynomial time and returns an equivalent instance (x', k') of L such that $|x'| + k' \leq g(k)$ for some computable function $g : \mathbb{N} \to \mathbb{N}$.

The function g is called the size of the kernel.

If g is a polynomial, then we speak about a polynomial kernel.

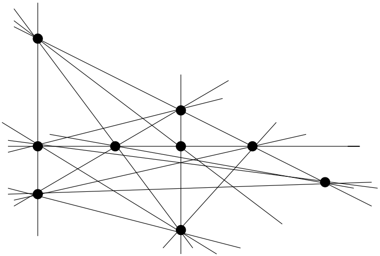
Folklore: $\boxed{\text{A problem is FPT} \Leftrightarrow \text{it admits a kernel}}$

Example of kernel for a geometric problem



Can a given set S of points in the plane be covered by at most k lines?

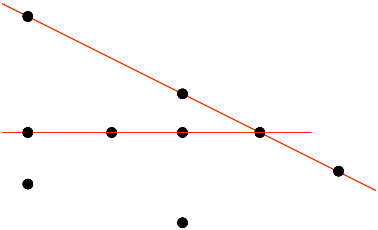
Example of kernel for a geometric problem



Can a given set S of points in the plane be covered by at most k lines?

Observation 1: We can just consider the lines generated by pairs of points in S

Example of kernel for a geometric problem

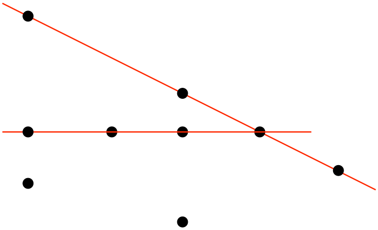


Can a given set S of points in the plane be covered by at most k lines?

Observation 2: If a line L contains at least $k+1$ points, then it necessarily belongs to the solution (if it exists) (in the example, $k = 3$)

$\Rightarrow$ delete L and update $k \to k-1$

Example of kernel for a geometric problem



Can a given set S of points in the plane be covered by at most k lines?

Observation 2: If a line L contains at least $k+1$ points, then it necessarily belongs to the solution (if it exists) (in the example, $k = 3$)

$\Rightarrow$ delete L and update $k \rightarrow k - 1$

$\Rightarrow$ The reduced instance must contain at most k^2 points (if more, answer is "NO")

Do all FPT problems admit polynomial kernels?

Folklore: $\boxed{\text{A problem is FPT} \Leftrightarrow \text{it admits a kernel}}$

Do all FPT problems admit polynomial kernels?

Folklore: $\boxed{\text{A problem is FPT} \Leftrightarrow \text{it admits a kernel}}$

Do all FPT problems admit polynomial kernels?

Do all FPT problems admit polynomial kernels?

Folklore: $\boxed{\text{A problem is FPT} \Leftrightarrow \text{it admits a kernel}}$

Do all FPT problems admit polynomial kernels? NO!

Theorem

Deciding whether a graph has a PATH *with $\geq k$ vertices* is FPT *but does not admit a polynomial kernel*,

Do all FPT problems admit polynomial kernels?

Folklore: $\boxed{\text{A problem is FPT} \Leftrightarrow \text{it admits a kernel}}$

Do all FPT problems admit polynomial kernels? NO!

Theorem

Deciding whether a graph has a $\textsc{Path}$ *with* $\geq k$ *vertices* is FPT *but does not admit a polynomial kernel, unless* NP $\subseteq$ coNP/poly.

Next section is...

Typical approach to deal with a parameterized problem

Parameterized problem L

k-Clique

k-Vertex Cover

k-Path

Vertex k-Coloring

Typical approach to deal with a parameterized problem

Parameterized problem L

k-Clique

k-Vertex Cover

k-Path

Vertex k-Coloring

XP

para-NP-hard

Typical approach to deal with a parameterized problem

Parameterized problem L

k-Clique
k-Vertex Cover
k-Path
Vertex k-Coloring

XP

k-Clique
k-Vertex Cover
k-Path

para-NP-hard

Vertex k-Coloring

Typical approach to deal with a parameterized problem

Parameterized problem L

k-Clique
k-Vertex Cover
k-Path
Vertex k-Coloring

XP

k-Clique
k-Vertex Cover
k-Path

para-NP-hard

Vertex k-Coloring

W[1]-hard

FPT

Typical approach to deal with a parameterized problem

Parameterized problem L

k-Clique
k-Vertex Cover
k-Path
Vertex k-Coloring

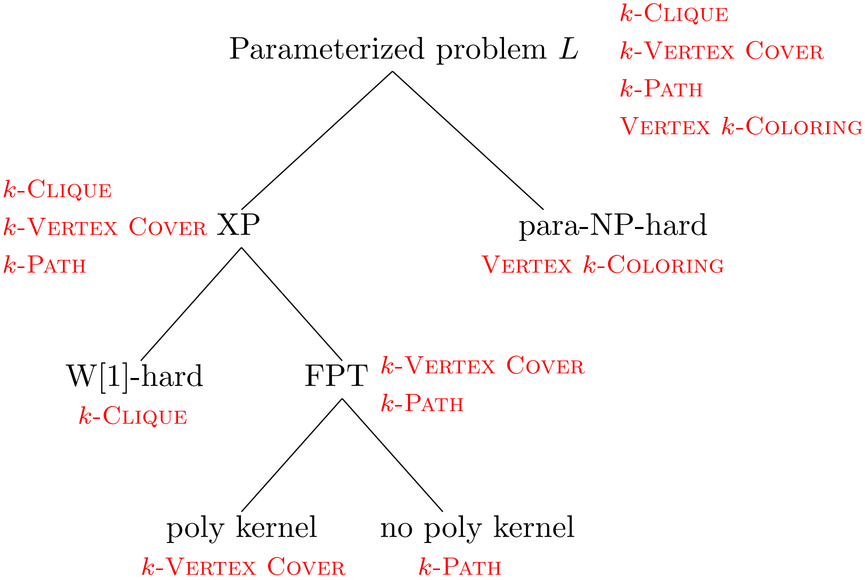
- XP
  - k-Clique
  - k-Vertex Cover
  - k-Path
  - W[1]-hard
    - k-Clique
  - FPT
    - k-Vertex Cover
    - k-Path
- para-NP-hard
  - Vertex k-Coloring

Typical approach to deal with a parameterized problem

k-Clique
k-Vertex Cover
k-Path
Vertex k-Coloring

- Parameterized problem L
  - XP — k-Clique, k-Vertex Cover, k-Path
    - W[1]-hard — k-Clique
    - FPT — k-Vertex Cover, k-Path
      - poly kernel
      - no poly kernel
  - para-NP-hard — Vertex k-Coloring

Typical approach to deal with a parameterized problem



How to prove that a problem is FPT?

How to prove that a problem is FPT?

There exist a bunch of techniques to obtain FPT algorithms:

- Bounded search trees
- Iterative compression
- Randomized methods (color coding, etc.)
- Tree decompositions and dynamic programming
- Important separators
- Representative sets (matroids)

Meta-techniques

There also exist meta-theorems to prove that whole families of problems are FPT.

Meta-techniques

There also exist meta-theorems to prove that whole families of problems are FPT.

Typical statement:

Every parameterized problem that satisfies property Π is FPT

There also exist meta-theorems to prove that whole families of problems are FPT.

Typical statement:

Every parameterized problem that satisfies property Π is FPT on the class of graphs $\mathcal{G}$.

There also exist meta-theorems to prove that whole families of problems are FPT.

Typical statement:

Every parameterized problem that satisfies property Π is FPT on the class of graphs $\mathcal{G}$.

Let us see two examples of famous meta-theorems.

Meta-theorem 1: Courcelle's theorem

Meta-theorem 1: Courcelle's theorem

Monadic Second Order Logic (MSOL):
Graph logic that allows quantification over sets of vertices and edges.

Meta-theorem 1: Courcelle's theorem

Monadic Second Order Logic (MSOL):
Graph logic that allows quantification over sets of vertices and edges.

Example: $\texttt{DomSet}(S):\quad [\ \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G)\ ]$

Meta-theorem 1: Courcelle's theorem

Monadic Second Order Logic (MSOL):
Graph logic that allows quantification over sets of vertices and edges.

Example: $\texttt{DomSet}(S): \quad [\ \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G)\ ]$

Treewidth:
Invariant that measures the topological resemblance of a graph to a tree.

Meta-theorem 1: Courcelle's theorem

Monadic Second Order Logic (MSOL):
Graph logic that allows quantification over sets of vertices and edges.

Example: $\texttt{DomSet}(S): \quad [\, \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G) \,]$

Treewidth:
Invariant that measures the topological resemblance of a graph to a tree.

Theorem (Courcelle)

Every problem expressible in MSOL can be solved in time $f(\mathrm{tw}) \cdot n$ on graphs on n vertices and treewidth at most tw.

Monadic Second Order Logic (MSOL):
Graph logic that allows quantification over sets of vertices and edges.

Example: $\texttt{DomSet}(S): \quad [\, \forall v \in V(G) \setminus S, \exists u \in S : \{u, v\} \in E(G) \,]$

Treewidth:
Invariant that measures the topological resemblance of a graph to a tree.

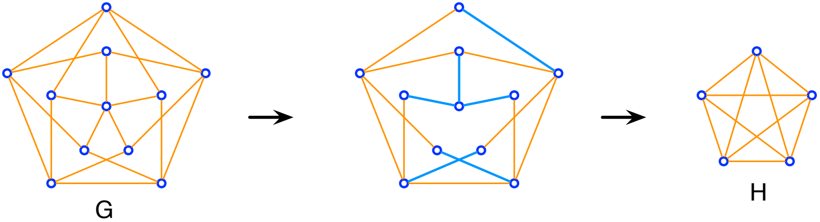
Theorem (Courcelle)

Every problem expressible in MSOL can be solved in time $f(\mathrm{tw}) \cdot n$ on graphs on n vertices and treewidth at most tw.

Examples: VERTEX COVER, DOMINATING SET, HAMILTONIAN CYCLE.

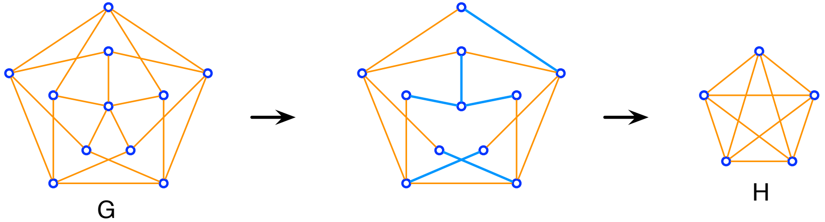
Meta-theorem 2: Graph minors

Meta-theorem 2: Graph minors



H is a minor of a graph G if H can be obtained from a subgraph of G by contracting edges.

Meta-theorem 2: Graph minors

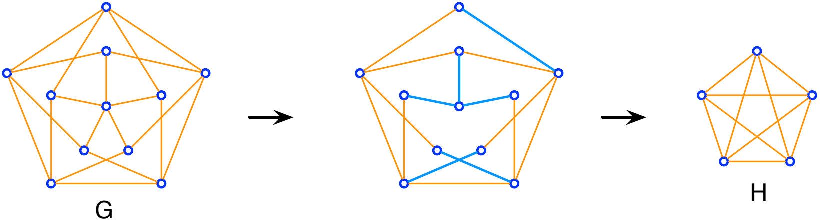


H is a minor of a graph G if H can be obtained from a subgraph of G by contracting edges.

A parameterized problem is minor-closed if

$$H \text{ is a minor of } G \;\Rightarrow\; \mathtt{param}(H) \leq \mathtt{param}(G).$$

Meta-theorem 2: Graph minors



H is a minor of a graph G if H can be obtained from a subgraph of G by contracting edges.

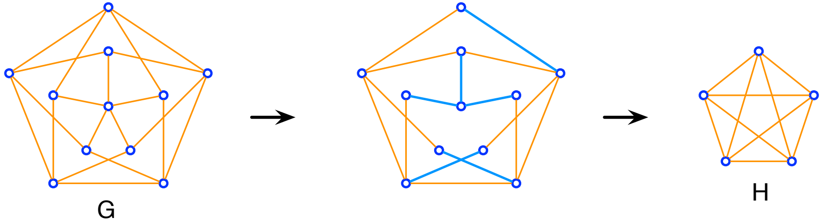
A parameterized problem is minor-closed if

$$H \text{ is a minor of } G \;\Rightarrow\; \texttt{param}(H) \le \texttt{param}(G).$$

Theorem (Robertson and Seymour)

Every minor-closed graph problem is FPT.

Meta-theorem 2: Graph minors



H is a minor of a graph G if H can be obtained from a subgraph of G by contracting edges.

A parameterized problem is minor-closed if

$$H \text{ is a minor of } G \;\Rightarrow\; \mathtt{param}(H) \leq \mathtt{param}(G).$$

Theorem (Robertson and Seymour)

Every minor-closed graph problem is FPT.

Examples: Vertex Cover, Feedback Vertex Set, Longest Path.

Is it enough to prove that a problem is FPT?

Typically, these meta-theorems allow to prove that a problem is FPT...

Is it enough to prove that a problem is FPT?

Typically, these meta-theorems allow to prove that a problem is FPT...

but the running time can be huge!

Is it enough to prove that a problem is FPT?

Typically, these meta-theorems allow to prove that a problem is FPT...

but the running time can be huge!

$$f(k) \cdot n^{O(1)}$$

Is it enough to prove that a problem is FPT?

Typically, these meta-theorems allow to prove that a problem is FPT...

but the running time can be huge!

$$f(k) \cdot n^{O(1)} = 2^{3^{4^{5^{6^{7^{8^{k}}}}}}} \cdot n^{O(1)}$$

Is it enough to prove that a problem is FPT?

Typically, these meta-theorems allow to prove that a problem is FPT...

but the running time can be huge!

$$f(k) \cdot n^{O(1)} = 2^{3^{4^{5^{6^{7^{8^{k}}}}}}} \cdot n^{O(1)}$$

Major goal: find the smallest possible function $f(k)$.

Is it enough to prove that a problem is FPT?

Typically, these meta-theorems allow to prove that a problem is FPT...

but the running time can be huge!

$$f(k) \cdot n^{O(1)} = 2^{3^{4^{5^{6^{7^{8^{k}}}}}}} \cdot n^{O(1)}$$

Major goal: find the smallest possible function $f(k)$.

This is one of the most active areas in parameterized complexity.

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

SETH: The SAT problem on n variables cannot be solved in time $(2 - \varepsilon)^n$

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

SETH: The SAT problem on n variables cannot be solved in time $(2-\varepsilon)^n$

$$\text{SETH} \Rightarrow \text{ETH}$$

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

SETH: The SAT problem on n variables cannot be solved in time $(2-\varepsilon)^n$

$$\mathrm{SETH} \quad \Rightarrow \quad \mathrm{ETH} \quad \Rightarrow \quad \mathsf{FPT} \neq \mathsf{W}[1]$$

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

SETH: The SAT problem on n variables cannot be solved in time $(2-\varepsilon)^n$

$$\text{SETH} \Rightarrow \text{ETH} \Rightarrow \text{FPT} \neq \text{W[1]} \Rightarrow \text{P} \neq \text{NP}$$

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

SETH: The SAT problem on n variables cannot be solved in time $(2-\varepsilon)^n$

$$\text{SETH} \quad \Rightarrow \quad \text{ETH} \quad \Rightarrow \quad \text{FPT} \neq \text{W[1]} \quad \Rightarrow \quad \text{P} \neq \text{NP}$$

Typical statements:

ETH $\Rightarrow$ k-Vertex Cover cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$.

Lower bounds on the running times of FPT algorithms

- Suppose that we have an FPT algorithm in time $k^{O(k)} \cdot n^{O(1)}$.
- Is it possible to obtain an FPT algorithm in time $2^{O(k)} \cdot n^{O(1)}$?
- Is it possible to obtain an FPT algorithm in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Very helpful tool: (Strong) Exponential Time Hypothesis – (S)ETH

ETH: The 3-SAT problem on n variables cannot be solved in time $2^{o(n)}$

SETH: The SAT problem on n variables cannot be solved in time $(2-\varepsilon)^n$

$$\text{SETH} \;\Rightarrow\; \text{ETH} \;\Rightarrow\; \text{FPT} \neq \text{W[1]} \;\Rightarrow\; \text{P} \neq \text{NP}$$

Typical statements:

ETH $\Rightarrow$ k-Vertex Cover cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$.

ETH $\Rightarrow$ Planar k-Vertex Cover cannot in time $2^{o(\sqrt{k})} \cdot n^{O(1)}$.

How to prove that a problem admits a (polynomial) kernel?

How to prove that a problem admits a (polynomial) kernel?

There exist a bunch of techniques to obtain (polynomial) kernels:

- Sunflower lemma
- Crown decomposition
- Linear programming
- Protrusion decomposition
- Matroids

Meta-kernelization

As in the case of FPT algorithms, there exist meta-kernelization results.

Meta-kernelization

As in the case of FPT algorithms, there exist meta-kernelization results.

Typical statement:

Every parameterized problem that satisfies property Π is admits a linear/polynomial kernel on the class of graphs $\mathcal{G}$.

Meta-kernelization

As in the case of FPT algorithms, there exist meta-kernelization results.

Typical statement:

Every parameterized problem that satisfies property Π is admits a linear/polynomial kernel on the class of graphs $\mathcal{G}$.

This has been also a very active area in parameterized complexity, specially on sparse graphs: planar graphs, graphs on surfaces, minor-free graphs, ...

Meta-kernelization results on sparse graphs

H-topological-minor-free

$\cup$

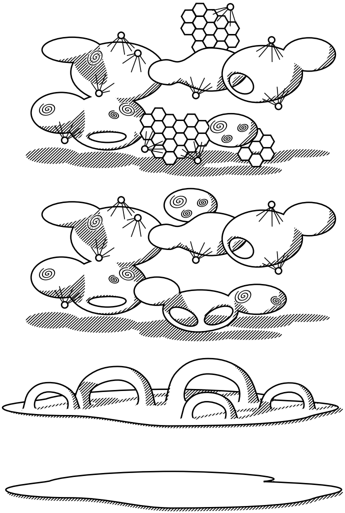
H-minor-free

$\cup$

bounded genus

$\cup$

planar



treewidth-bounding

bidimensional, separation property

quasi-compact

“distance-property”

[Figure by Felix Reidl]

Bibliography

- Parameterized Complexity, R. G. Downey and M. R. Fellows, 1999.

Bibliography

- Parameterized Complexity, R. G. Downey and M. R. Fellows, 1999.
- Invitation to Fixed-Parameter Algorithms, R. Niedermeier, 2006.
- Parameterized Complexity Theory, J. Flum and M. Grohe, 2006.
- Fundamentals of Parameterized Complexity, R. G. Downey, M. R. Fellows, 2013.
- Parameterized Algorithms, M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, 2015.

Gràcies!