Parameterized hardness

Rémi Watrigant

Université Claude Bernard Lyon 1

Journées Calamar
7, 8 avril 2025, ENS de Lyon

Menu

1. Parameterized problems
2. W-hierarchy and friends $\rightarrow$ time complexity
3. XNLP $\rightarrow$ time and space complexity
4. Kernels $\rightarrow$ preprocessing complexity

The usual suspects

Clique

Input: graph G, integer k
Goal: decide whether G has $\geqslant k$ pairwise adjacent vertices

Independent Set

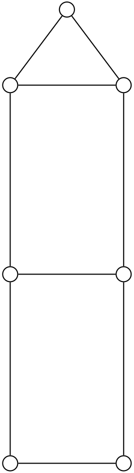
Input: graph G, integer k
Goal: decide whether G has $\geqslant k$ pairwise non-adjacent vertices

Vertex Cover

Input: graph G, integer k
Goal: decide whether G has $\leqslant k$ vertices incident to all edges

Dominating Set

Input: graph G, integer k
Goal: decide whether G has $\leqslant k$ vertices dominating all $V(G)$



The usual suspects

Clique

Input: graph G, integer k
Goal: decide whether G has $\geqslant k$ pairwise adjacent vertices

Independent Set

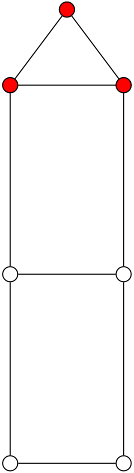
Input: graph G, integer k
Goal: decide whether G has $\geqslant k$ pairwise non-adjacent vertices

Vertex Cover

Input: graph G, integer k
Goal: decide whether G has $\leqslant k$ vertices incident to all edges

Dominating Set

Input: graph G, integer k
Goal: decide whether G has $\leqslant k$ vertices dominating all $V(G)$



The usual suspects

Clique

Input: graph G, integer k
Goal: decide whether G has $\geqslant k$ pairwise adjacent vertices

Independent Set

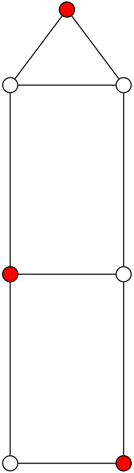
Input: graph G, integer k
Goal: decide whether G has $\geqslant k$ pairwise non-adjacent vertices

Vertex Cover

Input: graph G, integer k
Goal: decide whether G has $\leqslant k$ vertices incident to all edges

Dominating Set

Input: graph G, integer k
Goal: decide whether G has $\leqslant k$ vertices dominating all $V(G)$



The usual suspects

CLIQUE

Input: graph G, integer k
Goal: decide whether G has $\geqslant k$ pairwise adjacent vertices

INDEPENDENT SET

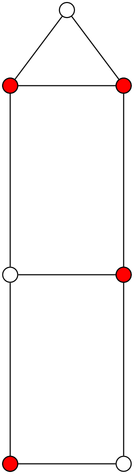
Input: graph G, integer k
Goal: decide whether G has $\geqslant k$ pairwise non-adjacent vertices

VERTEX COVER

Input: graph G, integer k
Goal: decide whether G has $\leqslant k$ vertices incident to all edges

DOMINATING SET

Input: graph G, integer k
Goal: decide whether G has $\leqslant k$ vertices dominating all $V(G)$



The usual suspects

Clique

Input: graph G, integer k
Goal: decide whether G has $\geqslant k$ pairwise adjacent vertices

Independent Set

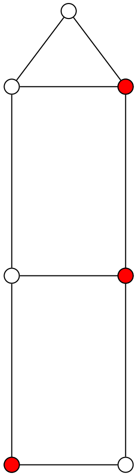
Input: graph G, integer k
Goal: decide whether G has $\geqslant k$ pairwise non-adjacent vertices

Vertex Cover

Input: graph G, integer k
Goal: decide whether G has $\leqslant k$ vertices incident to all edges

Dominating Set

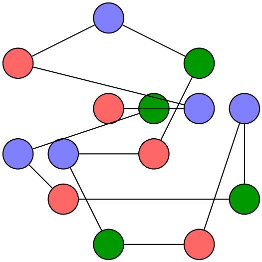
Input: graph G, integer k
Goal: decide whether G has $\leqslant k$ vertices dominating all $V(G)$



The usual suspects

q-Coloring

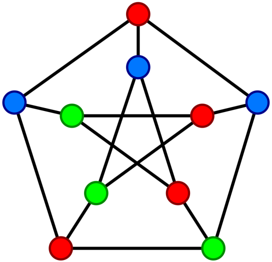
Input: graph G
Goal: decide whether $V(G)$ can be partitioned into q independent sets



The usual suspects

q-Coloring

Input: graph G
Goal: decide whether $V(G)$ can be partitioned into q independent sets



Example from Wikipedia

Parameterized problems

Why parameterized complexity?

$\rightarrow$ need for a "multivariate" analysis of problem hardness:

NP-hardness theory tells that for some problems, one cannot expect an algorithm deciding if an instance x is positive in time $|x|^{O(1)}$

$|x|$ hides many different parts of the instance that might be interesting:

- deciding whether G has a vertex cover of size at most k can be done in time $O(2^k(n+m))$
- deciding whether G has a clique of size at least k can be done in time $O(n^k k^2)$
- deciding whether G has an independent set of size at least k can be done in time $O((\Delta+1)^k(n+m))$ if G has maximum degree Δ
- deciding whether G is 3-colorable can be done in $2^{O(t)}n$ if G has treewidth at most t

Parameterized problems

- Σ = finite alphabet to encode problem inputs (ex: $\Sigma = \{0, 1\}$)

Parameterized problems

- Σ = finite alphabet to encode problem inputs (ex: $\Sigma = \{0, 1\}$)

A **parameterized problem** is a subset $Q \subseteq \Sigma^* \times \mathbb{N}$

- $(x, k) \in \Sigma^* \times \mathbb{N}$ is an instance
- $(x, k) \in Q$ iff it is a *positive instance* with *parameter value* k

decision problems only

In the remainder: $|x| = n$

Parameterized problems

- Σ = finite alphabet to encode problem inputs (ex: $\Sigma = \{0, 1\}$)

A **parameterized problem** is a subset $Q \subseteq \Sigma^* \times \mathbb{N}$

- $(x, k) \in \Sigma^* \times \mathbb{N}$ is an instance
- $(x, k) \in Q$ iff it is a *positive instance* with *parameter value* k

decision problems only

In the remainder: $|x| = n$

Several kinds of parameters (examples for graphs):

- related to the **solution**:
  - finding a structure of size/weight k in a graph
    $\rightarrow$ when turning an optimization problem into a decision problem
  - structure of the solution: ex: size of partition, property of a decomposition, ...
- related to the structure of the **input instance**: degree, *-width, "distance" to a known class, ...
- a **combination** of several parameters

Three worlds

Parameterized problems whose "unparameterized version" is NP-hard: 3 choices:

The bad:

There is a value of the parameter for which the problem is NP-hard
ex: Coloring parameterized by the number of colors (3-Coloring is NP-hard)

para-NP-hard

The ugly:

There is an algorithm running in time $O(n^{f(k)})$ for a computable function f
ex: Clique parameterized by the size of the clique

XP

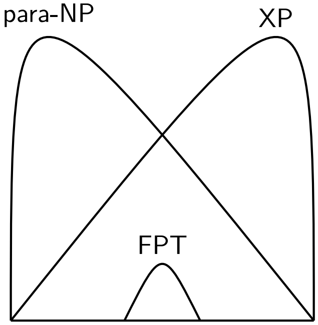
The good:

There is an algorithm running in time $f(k)n^{O(1)}$ for a computable function f
ex: Vertex Cover parameterized by the size of the vertex cover

Fixed-Parameter Tractable (FPT)

The picture so far

- **FPT**: solvable in deterministic $f(k)n^{O(1)}$ time
- **para-NP**: solvable in non-deterministic $f(k)n^{O(1)}$ time
- **XP**: solvable in deterministic $n^{f(k)}$ time



Known relations:

- para-NP = FPT $\Leftrightarrow$ P=NP
- FPT $\subsetneq$ XP   (relies on the fact that $DTIME(n^c) \subsetneq DTIME(n^{c+1})$)

Reduction

Parameterized reduction

Let $Q, R \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems
Parameterized reduction from Q to R: an algorithm which maps (x, k) to (x', k') such that:

- $(x, k) \in Q \Leftrightarrow (x', k') \in R$
- runs in time $f(k)n^{O(1)}$ for a computable function f
- $k' \leqslant g(k)$ for a computable function g

Theorem

If there is a parameterized reduction from Q to R, and R is **FPT**, then Q is **FPT**

Remark: the second condition implies that a parameterized reduction is different from a "classical" polynomial reduction

(but most parameterized reductions run in polynomial time)

Problems as hard as CLIQUE

- Reduction from CLIQUE to INDEPENDENT SET:
  $(G, k) \quad \rightarrow \quad (\overline{G}, k)$ (take the complement)

✓parameterized reduction

Problems as hard as CLIQUE

- Reduction from CLIQUE to INDEPENDENT SET:
  $(G, k) \quad \rightarrow \quad (\overline{G}, k)$ (take the complement)

✓parameterized reduction

- Reduction from INDEPENDENT SET to VERTEX COVER:
  $(G, k) \quad \rightarrow \quad (G, |V(G)| - k)$ (C is a vertex cover $\Leftrightarrow$ $V \setminus C$ is an independent set)

✗not a parameterized reduction

Problems as hard as CLIQUE

- Reduction from CLIQUE to INDEPENDENT SET:
  $(G, k) \quad \rightarrow \quad (\overline{G}, k)$ (take the complement)

✓parameterized reduction

- Reduction from INDEPENDENT SET to VERTEX COVER:
  $(G, k) \quad \rightarrow \quad (G, |V(G)| - k)$ (C is a vertex cover $\Leftrightarrow V \setminus C$ is an independent set)

✗not a parameterized reduction

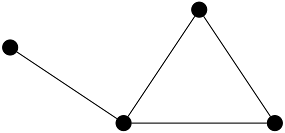
Working hypothesis: CLIQUE, INDEPENDENT SET $\notin$ FPT

Some interesting reductions 1/2

CLIQUE to MULTICOLORED CLIQUE (parameterized reduction)

MULTICOLORED CLIQUE

Input: a graph G, a partition V_1, V_2, ..., V_k of $V(G)$
Question: is there a clique C such that $|C \cap V_i| = 1$ for all $i = 1...k$?



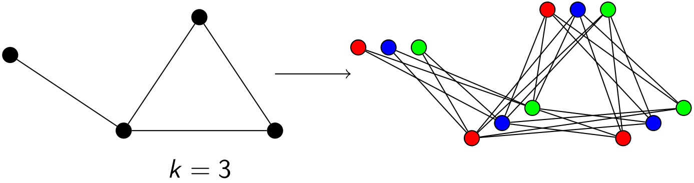
$k = 3$

Some interesting reductions 1/2

CLIQUE to MULTICOLORED CLIQUE (parameterized reduction)

MULTICOLORED CLIQUE

Input: a graph G, a partition V_1, V_2, ..., V_k of $V(G)$
Question: is there a clique C such that $|C \cap V_i| = 1$ for all $i = 1...k$?



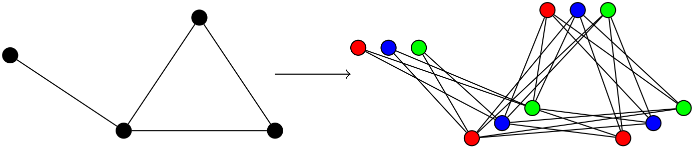
G has a clique of size k $\Leftrightarrow$ G' has a multicolored clique of size k

Some interesting reductions 1/2

CLIQUE to MULTICOLORED CLIQUE (parameterized reduction)

MULTICOLORED CLIQUE

Input: a graph G, a partition V_1, V_2, ..., V_k of $V(G)$
Question: is there a clique C such that $|C \cap V_i| = 1$ for all $i = 1...k$?



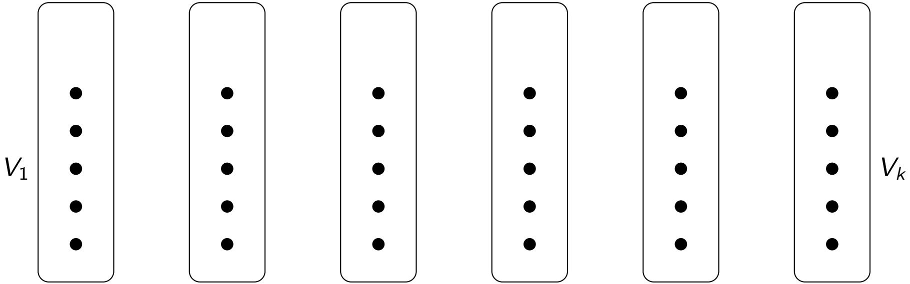
$k = 3$

G has a clique of size k $\Leftrightarrow$ G' has a multicolored clique of size k

- multicolored versions of problems are convenient starting points for parameterized reductions
- taking the complement: MULTICOLORED INDEPENDENT SET is as hard as CLIQUE and INDEPENDENT SET

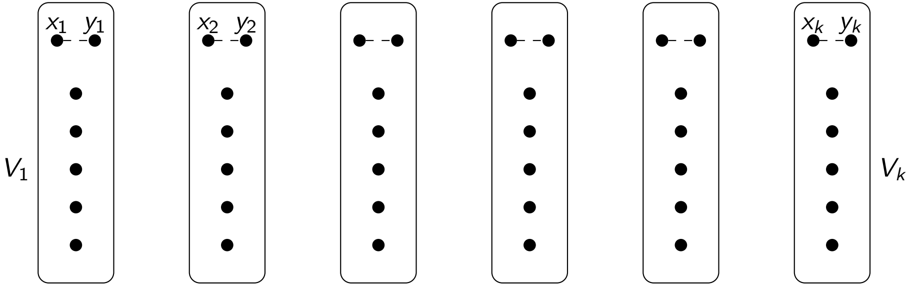
Some interesting reductions 2/2

Multicolored Independent Set to Dominating Set (parameterized reduction)



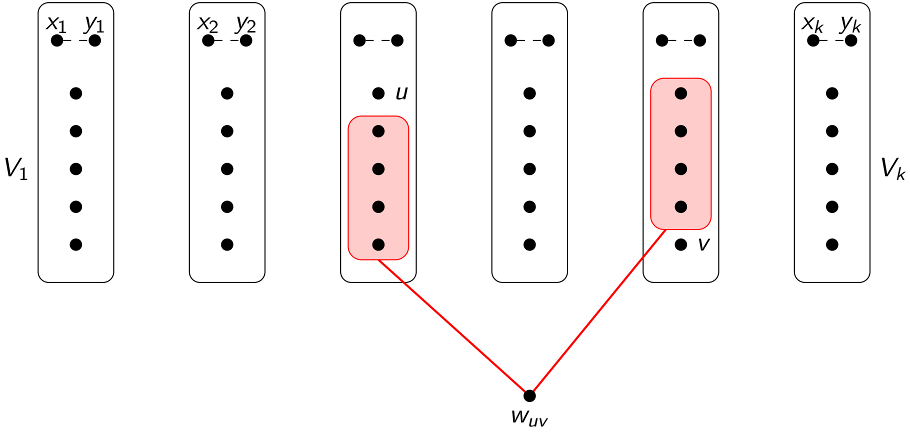
Some interesting reductions 2/2

MULTICOLORED INDEPENDENT SET to DOMINATING SET (parameterized reduction)



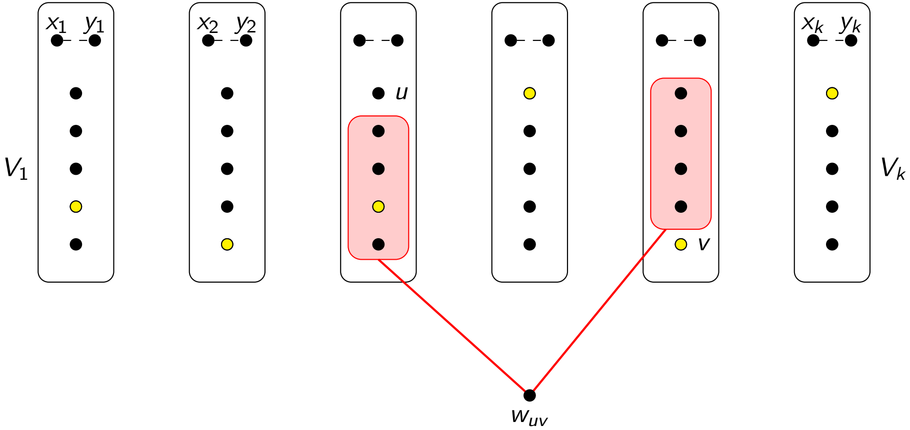
Some interesting reductions 2/2

MULTICOLORED INDEPENDENT SET to DOMINATING SET (parameterized reduction)



Some interesting reductions 2/2

Multicolored Independent Set to Dominating Set (parameterized reduction)



Some interesting reductions 2/2

Recap:

- parameterized reduction from CLIQUE to MULTICOLORED CLIQUE
- MULTICOLORED CLIQUE parameterized equivalent to MULTICOLORED INDEPENDENT SET
- parameterized reduction from MULTICOLORED INDEPENDENT SET to DOMINATING SET

Some interesting reductions 2/2

Recap:

- parameterized reduction from CLIQUE to MULTICOLORED CLIQUE
- MULTICOLORED CLIQUE parameterized equivalent to MULTICOLORED INDEPENDENT SET
- parameterized reduction from MULTICOLORED INDEPENDENT SET to DOMINATING SET

Implies a parameterized reduction from CLIQUE to DOMINATING SET
$\rightarrow$ so, under our working hypothesis, DOMINATING SET $\notin$ FPT

Some interesting reductions 2/2

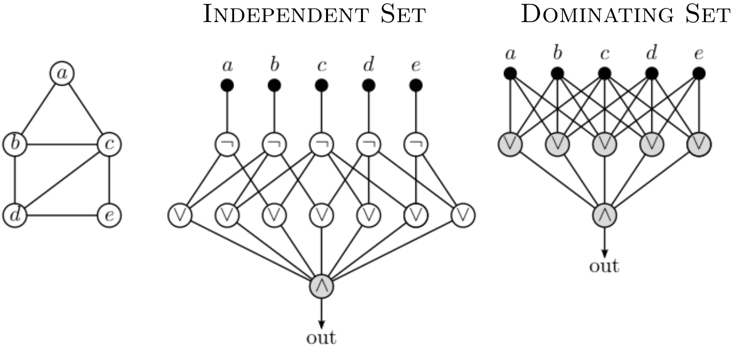
Recap:

- parameterized reduction from CLIQUE to MULTICOLORED CLIQUE
- MULTICOLORED CLIQUE parameterized equivalent to MULTICOLORED INDEPENDENT SET
- parameterized reduction from MULTICOLORED INDEPENDENT SET to DOMINATING SET

Implies a parameterized reduction from CLIQUE to DOMINATING SET
$\rightarrow$ so, under our working hypothesis, DOMINATING SET $\notin$ FPT

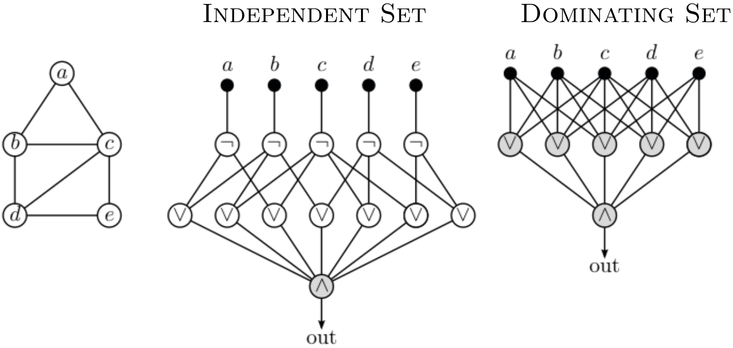
Interestingly, there is no known parameterized reduction backward, from Dominating Set to Clique
$\Rightarrow$ these two problems are not "equivalent" from the parameterized point of view

The W-hierarchy: the circuit point of view



- Goal: set at most k variables to 1
- Distinguish between **large** and **small** nodes
- Weft: maximum number of large nodes from an input to the output

The W-hierarchy: the circuit point of view



- Goal: set at most k variables to 1
- Distinguish between **large** and **small** nodes
- Weft: maximum number of large nodes from an input to the output

Definition

A parameterized problem is in $W[t]$ if it can be represented by a circuit of weft t (and bounded total depth)

The W-hierarchy

The following problems are complete for $W[1]$:

- WEIGHTED 2-SAT (set at most k variables to *true*)
- INDEPENDENT SET, CLIQUE
- MULTICOLORED INDEPENDENT SET, MULTICOLORED CLIQUE
- SHORT TURING MACHINE ACCEPTANCE: decide if a non-deterministic Turing machine halts in $\leqslant k$ steps

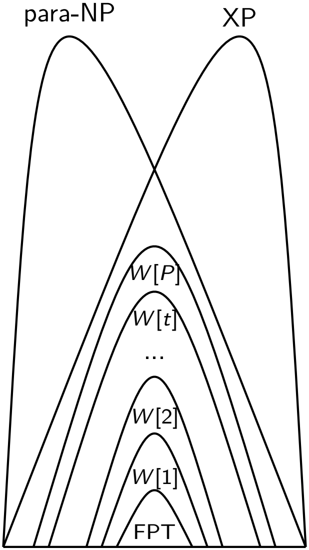
The following problems are complete for $W[2]$:

- DOMINATING SET
- Set systems: given $S_1, \cdots, S_m$ subsets of $\mathcal{U}$:
  - SET COVER: find $\leqslant k$ sets whose union is $\mathcal{U}$
  - HITTING SET find $\leqslant k$ elements of $\mathcal{U}$ intersecting each S_i

Canonical $W[t]$-complete problem:

- WEIGHTED t-NORMALIZED SAT: Boolean formula with t alternances of conjunctions and disjunctions

The picture



$W[P]$: weighted Boolean circuits (no weft restriction)

Further intuition that $FPT \neq W[1]$

FPT = W[1] contradicts the Exponential Time Hypothesis:

Theorem

An $f(k)n^{o(k)}$ algorithm for Clique implies a $2^{o(n)}$ algorithm for 3-Coloring

(which implies a $2^{o(n)}$ algorithm for 3-Sat and contradicts the ETH)

Sketch of proof:

Further intuition that $FPT \neq W[1]$

FPT = W[1] contradicts the Exponential Time Hypothesis:

Theorem

An $f(k)n^{o(k)}$ algorithm for CLIQUE implies a $2^{o(n)}$ algorithm for 3-COLORING

(which implies a $2^{o(n)}$ algorithm for 3-SAT and contradicts the ETH)

Sketch of proof: Let G with n vertices

- Carefully choose k to be roughly $f^{-1}(n)$
- Split $V(G)$ into k groups $V_1, \ldots, V_k$ of size at most $\lceil n/k \rceil$
- build a graph H:
  - for each $i = 1...k$, for each 3-coloring of V_i, add a vertex
  - connect two vertices if the two corresponding colorings are compatible
- Clearly, H has a k-clique $\Leftrightarrow$ G has a 3-coloring

Further intuition that $FPT \neq W[1]$

FPT = W[1] contradicts the Exponential Time Hypothesis:

Theorem

An $f(k)n^{o(k)}$ algorithm for CLIQUE implies a $2^{o(n)}$ algorithm for 3-COLORING

(which implies a $2^{o(n)}$ algorithm for 3-SAT and contradicts the ETH)

Sketch of proof: Let G with n vertices

- Carefully choose k to be roughly $f^{-1}(n)$
- Split $V(G)$ into k groups $V_1, \ldots, V_k$ of size at most $\lceil n/k \rceil$
- build a graph H:
  - for each $i = 1...k$, for each 3-coloring of V_i, add a vertex
  - connect two vertices if the two corresponding colorings are compatible
- Clearly, H has a k-clique $\Leftrightarrow$ G has a 3-coloring

Analysis:

- $|V(H)| \leqslant k \cdot 3^{\lceil n/k \rceil} = 2^{o(n)}$
- check that $f(k)|V(H)|^{o(k)}$ is $2^{o(n)}$

3. Some recent developments: XNLP

Some recent developments: XNLP

- Roughly, problems in $W[P]$ are: "choose at least/at most k elements out of n such that ..."
  $\Rightarrow$ they have certificates of size $O(k \log(n))$

Some recent developments: XNLP

- Roughly, problems in $W[P]$ are: "choose at least/at most k elements out of n such that ..."
  $\Rightarrow$ they have certificates of size $O(k \log(n))$

Bandwidth

Input: a graph $G = (V, E)$, $k \in \mathbb{N}$
Parameter: k
Question: is there a bijection $f : V \to \{1, \dots, |V|\}$ such that for every $uv \in E$, $|f(v) - f(u)| \leqslant k$?

Some recent developments: XNLP

- Roughly, problems in $W[P]$ are: "choose at least/at most k elements out of n such that ..."
  $\Rightarrow$ they have certificates of size $O(k \log(n))$

Bandwidth

Input: a graph $G = (V, E)$, $k \in \mathbb{N}$
Parameter: k
Question: is there a bijection $f : V \to \{1, \ldots, |V|\}$ such that for every $uv \in E$, $|f(v) - f(u)| \leqslant k$?

- For Bandwidth it seems that $\Omega(n)$ is needed for a certificate so likely $\notin W[P]$
- Belongs to XP: dynamic programming, remember the last $2k$ chosen vertices
  better: $O(n^k)$ time algorithm [Gurari, Sudborough, 1984]
- $W[t]$-hard for every $t \in \mathbb{N}$ [Bodlaender, claimed in 1994, proved in 2020]
- in which class does Bandwidth belong to?

Some recent developments: XNLP

XNLP

Parameterized problems which can be solved in non-deterministic FPT time and $O(f(k)\log(n))$ space

- Introduced by Elberfeld et al. in 2014, revisited by Bodlaender et al. in 2020
- BANDWIDTH $\in$ XNLP:
  - at each step, guess the next vertex
  - keep in memory the last $2k$ vertices with their order

$\rightarrow$ polynomial time, n guesses, $O(k\log(n))$ space

Some recent developments: XNLP

XNLP

Parameterized problems which can be solved in non-deterministic FPT time and $O(f(k)\log(n))$ space

- Introduced by Elberfeld et al. in 2014, revisited by Bodlaender et al. in 2020
- BANDWIDTH $\in$ XNLP:
  - at each step, guess the next vertex
  - keep in memory the last $2k$ vertices with their order

  $\rightarrow$ polynomial time, n guesses, $O(k\log(n))$ space
- BANDWIDTH is XNLP-complete [Bodlaender et al., 2021]
  (even for very restricted caterpillar graphs)

Some recent developments: XNLP

XNLP

Parameterized problems which can be solved in non-deterministic FPT time and $O(f(k)\log(n))$ space

- Introduced by Elberfeld et al. in 2014, revisited by Bodlaender et al. in 2020
- Bandwidth $\in$ XNLP:
  - at each step, guess the next vertex
  - keep in memory the last $2k$ vertices with their order

  $\rightarrow$ polynomial time, n guesses, $O(k\log(n))$ space
- Bandwidth is XNLP-complete [Bodlaender et al., 2021]
  (even for very restricted caterpillar graphs)
- XNLP captures many parameterized problems with "linear" structure which can be solved using dynamic programming:
- XNLP-completeness implies W[t]-hardness for every t
- hardness is obtained via parameterized logspace reductions ($O(f(k) + \log(n))$ space)

Some recent developments: XNLP

Some XNLP-complete problems:

Standard parameterization (solution):

- Longest Common Subsequence
- Chained version of Clique, Weighted CNF-Sat

Structural "linear" parameter:

- List Coloring, Precoloring Extension parameterized by pathwidth
- Independent Set, Dominating Set, Feedback Vertex Set, 5-coloring parameterized by linear mim-width

Some recent developments: XNLP

Some XNLP-complete problems:

Standard parameterization (solution):

- Longest Common Subsequence
- Chained version of Clique, Weighted CNF-Sat

Structural "linear" parameter:

- List Coloring, Precoloring Extension parameterized by pathwidth
- Independent Set, Dominating Set, Feedback Vertex Set, 5-coloring parameterized by linear mim-width

Slice-wise polynomial space conjecture

XNLP-hard problems do not admit algorithms running in $n^{f(k)}$ time and $f(k)n^{O(1)}$ space

→ they morally have to use dynamic algorithm with tables of size $n^{f(k)}$

Some recent developments: XNLP

Some XNLP-complete problems:

Standard parameterization (solution):

- Longest Common Subsequence
- Chained version of Clique, Weighted CNF-Sat

Structural "linear" parameter:

- List Coloring, Precoloring Extension parameterized by pathwidth
- Independent Set, Dominating Set, Feedback Vertex Set, 5-coloring parameterized by linear mim-width

Slice-wise polynomial space conjecture

XNLP-hard problems do not admit algorithms running in $n^{f(k)}$ time and $f(k)n^{O(1)}$ space

$\rightarrow$ they morally have to use dynamic algorithm with tables of size $n^{f(k)}$

- Also: XALP class, to capture problems with tree-structured parameters

4. Kernels

Kernels

Definition

A kernel for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is a polynomial-time algorithm which transforms (x, k) into (x', k') such that:

- (x, k) is a yes-instance $\Leftrightarrow$ (x', k') is a yes-instance
- $k' \leqslant k$
- $|x'| \leqslant f(k)$ for some computable function called the **size of the kernel**

Kernels

Definition

A kernel for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is a polynomial-time algorithm which transforms (x, k) into (x', k') such that:

- (x, k) is a yes-instance $\Leftrightarrow$ (x', k') is a yes-instance
- $k' \leqslant k$
- $|x'| \leqslant f(k)$ for some computable function called the **size of the kernel**

- One way to design FPT algorithm: kernel $\Rightarrow$ FPT
  - use the kernel (x, k) to get (x', k')
  - brute-force (x', k')

Kernels

Definition

A kernel for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is a polynomial-time algorithm which transforms (x, k) into (x', k') such that:

- (x, k) is a yes-instance $\Leftrightarrow$ (x', k') is a yes-instance
- $k' \leqslant k$
- $|x'| \leqslant f(k)$ for some computable function called the **size of the kernel**

- One way to design FPT algorithm: kernel $\Rightarrow$ FPT
  - use the kernel (x, k) to get (x', k') poly-time
  - brute-force (x', k') time $g(|x'|) \leqslant g(f(k))$

$\Rightarrow$ FPT running time

Kernels

Definition

A kernel for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is a polynomial-time algorithm which transforms (x, k) into (x', k') such that:

- (x, k) is a yes-instance $\Leftrightarrow$ (x', k') is a yes-instance
- $k' \leqslant k$
- $|x'| \leqslant f(k)$ for some computable function called the **size of the kernel**

- One way to design FPT algorithm: kernel $\Rightarrow$ FPT
  - use the kernel (x, k) to get (x', k') poly-time
  - brute-force (x', k') time $g(|x'|) \leqslant g(f(k))$

  $\Rightarrow$ FPT running time

- The converse also holds! FPT algorithm $\Rightarrow$ kernel:
  - Assume we have an algorithm $\mathcal{A}$ which solves (x, k) in time $f(k)|x|^c$
  - Let (x, k)
    - if $|x| \leqslant f(k)$: it is already a kernel
    - otherwise $|x| > f(k)$ but then $\mathcal{A}$ runs in polynomial-time (output a dummy instance)

Kernels

- The previous proof shows that all FPT problems admit a kernel (of possibly exponential size)
- Some problems admit a kernel of polynomial size

Kernels

- The previous proof shows that all FPT problems admit a kernel (of possibly exponential size)
- Some problems admit a kernel of polynomial size

The classical example: Buss' kernel for VERTEX COVER

Let (G, k). Question: at most k vertices incident to all edges?

- Observation 1: if a vertex is incident to $k+1$ edges, it must be in a solution $\rightarrow$ remove it, decrease k by 1
- Observation 2: if there is an isolated vertex, remove it (useless)

Kernels

- The previous proof shows that all FPT problems admit a kernel (of possibly exponential size)
- Some problems admit a kernel of polynomial size

The classical example: Buss' kernel for VERTEX COVER

Let (G, k). Question: at most k vertices incident to all edges?

- Observation 1: if a vertex is incident to $k+1$ edges, it must be in a solution $\rightarrow$ remove it, decrease k by 1
- Observation 2: if there is an isolated vertex, remove it (useless)

Apply the two *reduction rules* above as long as you can. At the end:

- maximum degree at most k, no isolated vertices

Remark: if G has a vertex cover of size at most k, it has at most k^2 edges
$\Rightarrow$ we may assume that G has $\leqslant k^2$ edges = **quadratic kernel**

Kernels

- The previous proof shows that all FPT problems admit a kernel (of possibly exponential size)
- Some problems admit a kernel of polynomial size

The classical example: Buss' kernel for VERTEX COVER

Let (G, k). Question: at most k vertices incident to all edges?

- Observation 1: if a vertex is incident to $k+1$ edges, it must be in a solution $\rightarrow$ remove it, decrease k by 1
- Observation 2: if there is an isolated vertex, remove it (useless)

Apply the two *reduction rules* above as long as you can. At the end:

- maximum degree at most k, no isolated vertices

Remark: if G has a vertex cover of size at most k, it has at most k^2 edges
$\Rightarrow$ we may assume that G has $\leqslant k^2$ edges = **quadratic kernel**

- For some problems, only kernels of exponential size
  How to rule out the existence of polynomial kernels?

Kernels: ruling out polynomial kernels

One such problem:

Longest Path

Input: A graph G, an integer k
Parameter: k
Goal: Decide whether G has a path of length at least k

Kernels: ruling out polynomial kernels

One such problem:

Longest Path

Input: A graph G, an integer k
Parameter: k
Goal: Decide whether G has a path of length at least k

Intuition for not having a polynomial kernel:

- suppose Longest Path has a kernel of size n^c
- take n^{c+1} graphs $G_1, \ldots, G_{n^{c+1}}$ on n vertices for which you want to test the existence of a path of length k
- let G^* be the disjoint union of all G_i's
- (G^*, k) yes-instance of Longest Path $\Leftrightarrow$ at least one G_i has a path of length k.
- apply the kernel on $(G^*, k^*) \rightarrow (G', k')$ with $|G'| \leqslant n^c$
  $\rightarrow$ **we have "forgotten" some instances in the process!**

(does not imply P=NP, but something weird)

Kernels: ruling out polynomial kernels

Cross-composition

Let L be a problem, Q be a parameterized problem
A cross-composition from L to Q is an algorithm which takes a sequence of instances $x_1, \dots, x_t$, all of size n, and output (y, k) such that:

- it runs in polynomial time in $\Sigma_{i=1}^{t}|x_i|$
- k is polynomial in n and $\log(t)$
- (y, k) is positive for Q iff at least one x_i is positive for L

Remark:

- size of y can be huge (polynomial in t)
- instances of the sequence can share some properties (e.g. same number of edges)

Kernels: ruling out polynomial kernels

Cross-composition

Let L be a problem, Q be a parameterized problem
A cross-composition from L to Q is an algorithm which takes a sequence of instances $x_1, \dots, x_t$, all of size n, and output (y, k) such that:

- it runs in polynomial time in $\Sigma_{i=1}^{t} |x_i|$
- k is polynomial in n and $\log(t)$
- (y, k) is positive for Q iff at least one x_i is positive for L

Remark:

- size of y can be huge (polynomial in t)
- instances of the sequence can share some properties (e.g. same number of edges)

Theorem [Bodlaender, Jansen, Kratsch, 2012]

If an NP-hard problem L cross-composes into a parameterized problem Q, then coNP $\subseteq$ NP/poly

coNP $\subseteq$ NP/poly

Weaker assumption than NP $\neq$ coNP:

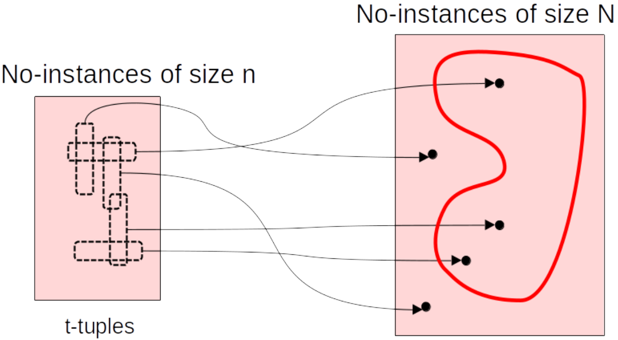
- **coNP**: solvable in co-nondeterministic polynomial time
  $\rightarrow$ if instance is negative, there is a computation path that rejects
  $\rightarrow$ if instance is positive, all computation paths accept
- **NP/poly**: NP using advices of polynomial size
  $\rightarrow$ for every size n, we have access to a string S_n of size $poly(n)$ for solving the instance

coNP $\subseteq$ NP/poly

Weaker assumption than NP $\neq$ coNP:

- **coNP**: solvable in co-nondeterministic polynomial time
  $\rightarrow$ if instance is negative, there is a computation path that rejects
  $\rightarrow$ if instance is positive, all computation paths accept
- **NP/poly**: NP using advices of polynomial size
  $\rightarrow$ for every size n, we have access to a string S_n of size $poly(n)$ for solving the instance

Composition + polynomial kernel $\Rightarrow$ **Distillation algorithm $\mathcal{A}$ for a problem L**



Advice S_n: set of $poly(n)$ no-instances of size N which covers all no-instances of size n

coNP $\subseteq$ NP/poly

Weaker assumption than NP $\neq$ coNP:

- **coNP**: solvable in co-nondeterministic polynomial time
  $\rightarrow$ if instance is negative, there is a computation path that rejects
  $\rightarrow$ if instance is positive, all computation paths accept
- **NP/poly**: NP using advices of polynomial size
  $\rightarrow$ for every size n, we have access to a string S_n of size $poly(n)$ for solving the instance

Composition + polynomial kernel $\Rightarrow$ **Distillation algorithm $\mathcal{A}$ for a problem L**

Given $x \in \Sigma^*$, guess a tuple $(x_1, ..., x_t)$ with $x = x_i$ for some i
$\rightarrow$ Check if $\mathcal{A}(x_1, \ldots, x_t) \in S_n$

- if $x \notin L$, there is a guess which will produce an element of S_n
- if $x \in L$, no guess will produce an element of S_n

$\rightarrow$ we decide $\overline{L}$

Kernels: ruling out polynomial kernels

Different ways for obtaining kernel lower bounds for a problem:

- Check if the problem composes to itself
  - disjoint union
  - + possibly some instance selector gadget

- Turn the known NP-hardness reductions into compositions

- Reduce from a problem with a known kernel lower bound
  $\rightarrow$ restriction of parameterized reduction (bound on size of parameter)

  - SAT parameterized by the number of variables
  - Colored Red-Blue Dominating Set parameterized by number of colors + vertices to dominate
  - d-Hitting Set has no kernel of size $O(k^{d-\varepsilon})$ (unless...)
  - ...

Kernels: ruling out polynomial kernels

Some variants of cross-compositions:

- it can run in co-nondeterministic time (may use non-constructive results)
  $\rightarrow$ RAMSEY does not have PK (unless...)

- instead of encoding the OR of the input problem, it may encode the AND

- if the output parameter k is $\leqslant poly(n) \cdot t^{1/d}$
  $\Rightarrow$ no kernel of bitsize $O(n^{d-\varepsilon})$
  $\rightarrow$ VERTEX COVER has no kernel of bitsize $O(n^{2-\varepsilon})$ (unless...)

Thanks! Questions?