1 Introduction to parameterized algorithm

1.1 FPT algorithms

Let \( \Pi \) be a problem. Let \( I \) be an instance of \( \Pi \) and \( k \) be an integer. An algorithm \( A \) is FPT if it runs in time \( f(k) \cdot P(n) \) where \( P \) is a polynomial function.

Various parameters are considered in the literature, in these lectures, we will concentrate on two main parameters: the solution size and the so-called treewidth of your graph.

The goal of FPT algorithms (and of the parameterized complexity in general) consists in trying to limit the combinatorial explosion to some parameter (hopefully small). If the function \( f \) is “decent” and we are given an instance with a small parameter, then the algorithm might be particularly efficient!

**Vertex Cover.** Let us provide an FPT algorithm for Vertex Cover. Let \( G \) be a graph and \( k \) be the solution size. Let \( e = (u, v) \) be an edge of \( G \). Since we are looking for a vertex cover, we know that \( u \) or \( v \) have to be in the vertex cover (maybe both). We then consider two subinstances \((G \setminus u, k - 1)\) and \((G \setminus v, k - 1)\). We apply inductively the algorithm on these two cases.

We claim that the complexity is \( O(P(n) \cdot 2^k) \). Indeed, the complexity of the algorithm is given by \( f(k, n) \leq 2f(k - 1, n) \). One can easily check that the function \( f \) satisfies the above conditions.

In order to see it in another (very useful for more complex problems) way, let us define the concept of **branching tree.** It is a way of evaluating the complexity of an FPT algorithm.

- Each node corresponds to an instance considered in our algorithm.
- We set an arc between two instances \( I, I' \) if \( I' \) is considered as a call from \( I \).

**Lemma 1.** If:

- The width of the tree is \( \leq g(k) \)
- The depth of the tree is \( \leq h(k) \)
- Each leaf can be decided in FPT time \( f(n, k) \)
- Each subinstance can be computed \( f(n, k) \)

then the problem is FPT and its running time is \( \ast(g(k)^{h(k)} \cdot f(n, k)) \).
Better algorithm for Vertex Cover? One can ask if a better algorithm can be found for VC. The answer is positive. Let us consider the following branching rule: if $G$ admits a vertex $v$ of degree at least 3, then we branch of the instance $(G \setminus v, k - 1)$ and $(G \setminus N[v], k - |N(v)|)$.

We claim that when this rule is no long applicable then the resulting instance can be decided in polynomial time. A careful analysing of the running time of this algorithm (by counting the number of nodes of the branching tree) guarantees that we have a running time of $O^*(2^{1.47k})$.

1.2 Kernalization.

A kernel of an instance $(I, k)$ of a parameterized problem is an instance $(I', k')$ that can be computed in polynomial time from $(I, k)$ and such that:

- $(I, k)$ is positive if and only if $(I', k')$ is positive and,
- $k' \leq f(k)$ for some function $f$ (that does not depend on the input).
- $|I| \leq g(k)$ for some function $g$ (that does not depend on the input).

A kernel is polynomial if both $f$ and $g$ are polynomial function. In most of the existing kernel algorithms, $k' \leq k$ (the value of the parameter is usually decreasing).

**Theorem 2.** A problem admits a FPT algorithm if and only if it admits a kernel.

**Proof.** If a problem admits a kernel, then it indeed admits a FPT algorithm. We first compute a kernel in time $P(n, k)$ and then decide the problem on an instance of size $f(k)$, which can be done in a time that does not depend on $n$ anymore.

Let us now prove the contrary. Assume that there exists an FPT algorithm running in time $f(k) \cdot P(n)$. Now we consider two cases, either $n \leq f(k)$ and then the FPT algorithm is actually a polynomial time algorithm. So we can now assume that $n \geq f(k)$. Then, in particular, the whole instance is a kernel of size at most $f(k)$, which completes the proof.

So we mainly focus on trying to find polynomial kernels. Let us briefly describe the main ingredient used to obtain kernels, namely “reduction rules”. A reduction rule consists in finding in the graph a “structure” that can be remplaced by another one which is “simpler” (usually simpler means smaller, but it might also mean structurally simpler). Usually, in order to prove that a problem admits a polynomial kernel, we need two steps:

1. Find some reduction rules (that can be applied in polynomial time) and prove their correctness.
2. Prove that when no reduction rule can be applied, the size of the instance is bounded by $P(k)$ (or that otherwise the instance is negative / positive).

Vertex Cover.

Let us first start with the following reduction rule:

**Reduction Rule 1.** If a vertex has degree more than $k$, then delete it and reduce by $k$ by 1.

**Lemma 3.** If Reduction Rule 1 is no longer appliable either $G$ has size at most $k^2$ or the instance is negative.
The proof is left as an exercise.
Can we do better? The answer is positive using LP. You (may) know that extremal points of the Vertex Cover polynomial are semi-integral. In other words, all the fractional solutions of the MVC problem have value in \( \{0, \frac{1}{2}, 1\} \). It leads to the following reduction rule:

**Reduction Rule 2.** Find an optimal solution of the fractional relaxation of the MVC. Delete all the vertices with value 0 and 1. And decrease the value of the parameter by the number of vertices with value 1.

**Lemma 4.**
- The resulting instance is equivalent to the original one.
- The resulting instance has size at most \( 2k \).

**d-Hitting Set** Let us consider a problem generalizing vertex cover, called the Hitting Set problem. Let \( H = (V,E) \) be a hypergraph. A hypergraph is a \( d \)-hypergraph if all the hyperedges have size at most \( d \). A hitting set of a hypergraph is a subset \( S \) of vertices such that for every hyperedge \( e \), \( e \cap S \) is not empty. (Why does it generalizes vertex cover?). In what follows our goal consists in proving that the Hitting Set problem in \( d \)-hypergraphs admits a polynomial kernel.

**Exercise 5.** Generalize the Reduction Rule 1.1 for \( d \)-Hitting Set.

## 2 Classical methods

### 2.1 Color Coding.

Done with Louis Esperet.

### 2.2 Iterative compression

This method was introduced by Reed at al. (to find odd cycle transversal). We illustrate it on the problem of feedback vertex set problem. A feedback vertex set is a subset of vertices \( X \) such that the graph induced by \( V \setminus X \) is acyclic.

Let us now introduce the key notion of iterative compression.

**Lemma 6** (Reed et al.). Let \( (\Pi, k) \) be an hereditary
\(^1\) parameterized problem. Let \( \text{DISJOINT} \Pi \) be the problem where the instance consists in a solution of \( \Pi \) of size \((k + 1)\) plus a solution of \( S \) of \( \Pi \) of size at most \((k + 1)\). And the goal is to find a solution of \( \Pi \) of size \( k \) disjoint from \( S \).

Let \( (I, k) \) be an instance of \( \Pi \). If \( \text{DISJOINT} \Pi \) admits an FPT algorithm running in time \( a^k \cdot n^c \) then \( \Pi \) admits an FPT algorithm running in time \( a^{k+1} \cdot n^{c+1} \).

**Proof.** Let us prove it by induction on the size of the instance. (For simplicity, let us assume that the instance is a graph). Let \( G_i \) be the current instance and \( S_i \) be a solution of \( G_i \) of size at most \( k \). If such a solution does not exist, the hereditary property ensures that the whole instance is negative and we can output no. So let \( S_i \) be a solution of \( G_i \). Now let us add a new element \( v_{i+1} \). \( S_i \cup v_{i+1} \) is still a solution of \( G_{i+1} \).

Now let us try to find a solution of \( G_{i+1} \) of size \( k \). We branch over all the possible intersection of the solution \( S' \) with \( S_i \cup v_{i+1} \). In the branches corresponding to \( S' \cap (S_i \cup v_{i+1}) \neq \emptyset \), then by induction there is a running time of at most \( a^{k-\ell} n^c \) if the intersection has size \( \ell \) (indeed we

\(^1\)Hereditary means here that if \( S \) is a solution for \( \Pi \) then it is a solution for “smaller” instances.
reduced the parameter by \( \ell \) and we are looking for a solution disjoint from what remain. So for all these branches, the total running time is at most
\[
\sum_{\ell=1}^{k} \binom{k+1}{\ell} a^{k-\ell} n^c \leq ((a + 1)^k - 1) \cdot n^c
\]
And finally by assumption, the last instance can be solved in time \( f(k) \cdot n^c \) (since it is an instance of DISJOINT II).

Let us now illustrate it on the problem of FVS.

**Lemma 7.** The problem DISJOINT FEEDBACK VERTEX SET can be decided in time \( \text{TODO} \).

The rest of this part is devoted to prove the lemma.

Let \( I = (G, k, S) \) be an instance of DISJOINT FEEDBACK VERTEX SET where \( G \) is a graph, \( k \) is a parameter and \( S \) is a solution of size at most \( k + 1 \). Note that we can assume \( G[S] \) induces a forest since otherwise, the instance \( I \) is a no instance.

So from now on we can assume that \( S \) induces a forest. We now consider the following two reduction rules:

**Reduction 1.** If there exists a vertex \( v \in V \) of degree two with one neighbor of \( x \) in \( S \) and one neighbor \( y \) in \( V \setminus S \) then return the instance \((G, k, S \cup \{v\})\).

**Proof:** If there is a solution containing \( x \), then it is also a solution if we replace \( x \) by \( y \). So if there exists a solution, there exists a solution that does not contain \( x \) and then the new instance has a solution.

**Reduction 2.** If there exists a vertex \( v \in V \) incident to two vertices of \( S \) in the same connected component of \( G[S] \), then return the instance \((G \setminus v, k - 1, S)\).

**Proof:** We are looking for a solution that does not contain any vertex \( v \) of \( S \). And there exists a cycle containing only vertices of \( S \) plus \( v \) so if we want to eliminate this cycle, we need to select \( v \).

**Branching Rule 3.** If there exists a vertex \( v \in V \) incident to two vertices of \( S \) in distinct component of \( G[S] \) then branch over the instances \((G \setminus v, k - 1, S)\) and \((G, k, S \cup v)\).

**Proof:** The proof is trivial. Either \( v \) is in the solution and we are in the first case, or it is not in the solution and we are in the second case.

Using these reduction and branching rules, we can prove the following

**Lemma 8.** Disjoint FVS admits an FPT algorithm running in time \( O(2^{k+cc} n^c) \) where \( cc \) denotes the number of connected component of \( S \) in the original instance and \( c \) is a universal constant.

**Proof.** We prove it by induction on \( n + k \). Let \( G \) be an instance of size \( n \) where \( k \) is the value of the parameter and \( cc \) the number of connected component induced by \( S \). Note that since \( S \) is a solution, \( G \setminus S \) is a solution. So \( G \setminus S \) induces a forest. Let \( v \) be a leaf of this forest. Either Reduction 1 or Reduction 2 can be applied, and then, by induction we have an algorithm running in time at most \( P(n) + f(k, n - 1, cc) \) and then the conclusion holds (note that in both cases, the number of connected component in \( S \) did not decrease). So we can assume that none of the two reduction rules apply. Note that if \( v \) has no neighbor in \( S \), then we can delete it since it is in no cycle. So we can assume that \( d(v) \geq 3 \) and \( v \) has at least two neighbors in \( S \). So we can apply Branching Rule 3. By induction, the problem can be decided in time \( P(n) + f(k - 1, n - 1, cc) + f(k, n - 1, cc - 1) \). By induction, the running time is at most:
\[
O(2^{2+1}cc + 2^{k-2cc-1})n^c) = O(2^{k+cc}n^c)
\]
and then the conclusion holds.

Since \( cc \leq k \) in our case, we get a \( O^*(4^k) \) algorithm to compute a Disjoint FVS if \( S \) has size at most \( k + 1 \).
3 Lower bounds

3.1 For polynomial kernels

An OR-distillation for a problem Π is an algorithm such that:

• has an input a sequence \((x_1, \ldots, x_t)\) of instances of Π
• has running time \(\text{Poly}(\sum |x_i|)\)
• returns an instance \(y\) of Π such that \(y \in \Pi\) if and only if there exists an \(i\) such that \(x_i \in \Pi\) (and \(|y| = P(\sum |x_i|)\)).

The OR-distillation are unlikely to happen because of the following result:

**Theorem 9.** If an NP-complete problem admits an OR-distillation then \(\text{coNP} \subseteq \text{NP/Poly}\).

An OR-composition for a problem Π is an algorithm such that:

• has an input a sequence \((x_1, k), \ldots, (x_t, k)\) of parameterized instances of Π
• has running time \(\text{Poly}(\sum |x_i| + k)\)
• returns an instance \((y, k')\) of Π such that \(y \in \Pi\) if and only if there exists an \(i\) such that \(x_i \in \Pi\) (and \(|y| = P(\sum |x_i|)\)) and \(k'\) is polynomial in \(k\).

**Theorem 10.** If an NP-complete problem Π admits an OR-composition and a polynomial kernel parameterized by \(k\), then Π admits an OR-distillation.

**Proof.** Consider the OR-composition. Kernalize it. We can prove that it corresponds to an OR-distillation. \(\square\)

Applications.

**Theorem 11.** Longest Path does not admit a polynomial kernel (unless \(\text{coNP} \subseteq \text{NP/poly}\)).

3.2 For FPT algorithms

Quick summary: We have to make the same type of reduction as for NP-completeness except that:

• We get an FPT time to do it.
• The parameter in the final instance has to be bounded by \(f(k)\) where \(k\) is the original parameter.

4 Treewidth

4.1 Tree Decompositions and Brambles

Treewidth provides a measure of how closely a graph shares some structural separation properties possessed by trees. Formally, given an undirected graph \(G = (V, E)\) we may represent it by a tree \(T = (N, L)\) and a labeling \(\gamma : N \to 2^V\).\(^2\) The labeling assigns to each node \(t \in T\) a

\(^2\)For clarity, we will refer to vertices and edges in \(G\) and nodes and links in \(T\).
subset \((t) = V_t\) of vertices of \(G\) also called the bag of \(t\). For each \(v \in V\) we denote by \(T_v\) the set of nodes in \(T\) for which \(v\) is included in the label, i.e. \(T_v = \{ t : v \in V_t \}\). We say that a tree and labeling, \((T, )\), is a tree decomposition of \(G\) if:

(i) For each vertex \(v\) of \(G\), the set \(T_v\) is a non-empty and connected subgraph of \(T\).
(ii) For each edge \(e = (u, v)\) in \(G\), the subtrees \(T_u\) and \(T_v\) intersect in \(T\).

The width of a tree decomposition \((T, )\) of \(G\) is the size of the largest label of a node in \(T\) minus one.\(^3\) The treewidth, \(tw(G)\), is the minimum width of a tree decomposition of \(G\).

Another way to see treewidth is related to "completion into chordal graphs". We have already seen that chordal graphs are intersection of subtrees of trees. It immediately provides a tree decomposition where the tree corresponds to the tree defining chordal graphs and a vertex is put in the bag of a node \(w\) if the subtree of the vertex contains the node \(w\). One can prove (using bramble below) that the treewidth of the graph is exactly the one obtained using this algorithm. Moreover, for every graph of treewidth \(t\), the graph is then a "subgraph" of a chordal graph of maximum clique \(t + 1\) corresponding to the graph where all the vertices in the bags of the tree decomposition are transformed into cliques.

There are combinatorial structures called brambles that provide a dual notion for tree decompositions. A bramble is a collection \(F = \{F_1, F_2, \ldots, F_p\}\) of sets such that:

(i) Each \(F_i \subseteq V\) induces a connected subgraph of \(G\), and
(ii) Every pair \(F_i\) and \(F_j\) in \(F\) either intersect (share a vertex) or are adjacent (there is an edge with one endpoint in \(F_i\) and one endpoint in \(F_j\)).

The hitting size \(\beta(F)\) of a bramble \(F\) is the minimum size of a subset of vertices that intersects each set \(F_i\) in \(F\). The bramble number \(\beta(G) = \max F \beta(F)\) is the maximum hitting size of any bramble in \(G\). Seymour and Thomas proved the following minmax theorem:

**Theorem 12.** [Seymour, Thomas] The bramble number \(\beta(G)\) is equal to the treewidth \(tw(G)\) − 1.

Brambles (rather than tree decompositions) directly relate to coalition games. Moreover, the relationship is actually through combinatorial structures we call thickets.

**Exercise 13.** What is the treewidth of trees? Cliques? Grids?

**Exercise 14.** Prove that the treewidth of a chordal graph is \(\omega(G) - 1\) were \(\omega\) denotes the size of a maximal clique.

We will also need the followign lemma that is not hard to prove:

**Lemma 15.** Prove that there exists a tree decomposition of \(G\) containing at most \(|V(G)|\) nodes.

### 4.2 Treewidth and FPT algorithm

Since their introduction, treewidth appeared to be a central object in graph theory and algorithmic. In parameterized, it plays a central role because of the following theorem due to Courcelle:

**Theorem 16** (Courcelle). Every problem \(\Pi\) that can be expressed in Monadic Second Order Logic can be solved in \(f(tw) \cdot n\). In other words, it is FPT parameterized by treewidth.

Two remarks. The first one, positive, is that the algorithm is not only polynomial when the treewidth is bounded but even linear ! It is very surprising when you know the whole general of MSOL. The second remark, quite negative, is that the function hidden in \(f\) is ugly. It is a very bad function, even for simple formula... That’s the price to pay for generality !

Now what is the monoic second order logic? Informally

\(^3\)The decision to subtract one was made to ensure trees have treewidth one. Unfortunately, as is apparent, this choice leads to an unaesthetic “plus one” in many theorems concerning treewidth.
• First order: We can quantify on variables.
• Second order: We can quantify over functions.
• MSOL is a fragment of second order logic (ie a subset). We can only quantify over sets.

For instance, the Dominating Set problem (does it exist a dominating set of size at most $k$) can be expressed in MSOL: $\exists X, |X| \leq k, \forall v \in V, (v \in X \lor \exists y \in X \text{ such that } (x, y) \in E)$.

The proof of Courcelle theorem is based on a (complicated) dynamic programming. But the technicalities essentially comes from its generality. Let us prove it for Dominating Set instead!

Basic ideas of the proof. We make a dynamic programming algorithm bottom up from the leaves (we root the tree arbitrarily). Let $u$ be a node of $T$ and let us denote by $V_u$ the subset of vertices in the subgraph rooted in $u$. And let us denote by $X_u$ the set of vertices of the bag of $u$. Note that by definition of treewidth, all the vertices that appear in bags which are not in the subtree rooted in $u$ are either in $X_u$ or in $V \setminus V_u$ (since the set of bags containing a vertex have to be connected).

Now we want to compute all the following $3^{tw}$ values on each node of the tree: For each vector of $\{0, 1, 2\}^{|X_u|}$ we want to determine the minimum number of vertices of $V_u \setminus X_u$ such that:

• All the vertices of $X_u$ with value 1 are selected.
• All the vertices of $V_u \setminus X_u$ have to be dominated.
• All the vertices of $X_u$ of value 2 have to be dominated.

Note that the vertices of value 0 are not necessarily dominated. Somehow we ”guess“ that these vertices will be dominated later by vertices in the parents.

Note that in total, the number of values we have to compute is $3^k \cdot V(T)$. Since we have seen that we can assume that $T$ has at most $n$ vertices, it would give the conclusion. Let us prove that every value can be easily computed.

Lemma 17. Let $u$ be a node and $v_1, \ldots, v_k$ be its children. If we can access to all the values of $v_1, \ldots, v_k$ in constant time then all the values for $u$ can be computed in FPT time parameterized by the treewidth.

Exercise: Prove it!