Partie IV : Complexité Paramétrée

G. Fertin
guillaume.fertin@univ-nantes.fr

Université de Nantes, LS2N
Bât 34 – Bureau 301

M1 Informatique – 2019/2020

Sommaire

Introduction

Ce qu'on a vu jusqu'à présent pour résoudre un problème NP-complet :

- démontrer que P=NP (exact et polynomial)
- petits cas et classes d'instances (exact et polynomial)
- approximation (pas exact mais écart à l'optimal contrôlé ; polynomial)

Introduction

Ce qu'on a vu jusqu'à présent pour résoudre un problème NP-complet :

- démontrer que P=NP (exact et polynomial)
- petits cas et classes d'instances (exact et polynomial)
- approximation (pas exact mais écart à l'optimal contrôlé ; polynomial)
- ici : FPT (exact et donc exponentiel)

ATTENTION Exponentiel $\rightarrow$ non praticable pour les instances "intéressantes"

FPT : catégorie particulière d'algorithme exponentiel

FPT ? Kezako ?

Intuitivement :

- Pb = notre problème ; n = la taille des données de Pb
- choisir un paramètre k de Pb

FPT ? Kezako ?

Intuitivement :

- Pb = notre problème ; n = la taille des données de Pb
- choisir un paramètre k de Pb
- chercher un algorithme exponentiel en k uniquement
- si on trouve un tel algorithme, alors Pb est FPT = Fixed-Parameter Tractable

FPT ? Kezako ?

Formellement :

Fixed-Parameter Tractability

- Parameter : k
- Tractability : polynomialité (en temps)
- Fixed : si $k = O(1)$, alors le problème est polynomial

FPT ? Kezako ?

Formellement :

Fixed-Parameter Tractability

- Parameter : k
- Tractability : polynomialité (en temps)
- Fixed : si $k = O(1)$, alors le problème est polynomial

$\Rightarrow$ on veut que l'algorithme qui résout Pb soit en

$$O(f(k) \cdot n^c)$$

où $f(k)$ est un fonction qui ne dépend que de k

Intérêt du FPT

- L'exponentielle est confinée en k
- $\Rightarrow$ si k est petit, l'algorithme FPT qui résout Pb peut aller "assez loin"

Intérêt du FPT

- L'exponentielle est confinée en k
- $\Rightarrow$ si k est petit, l'algorithme FPT qui résout Pb peut aller "assez loin"

Exemple (fictif) : si $k << n$, mieux vaut $O(2^k \cdot n^2)$ que $O(2^n)$

FPT – Bibliographie

k	$f(k) = 2^k$	$f(k) = 1.49^k$	$f(k) = 1.32^k$	$f(k) = 1.28^k$
10	$\sim 10^3$	~ 54	~ 16	~ 12
20	$\sim 10^6$	~ 2909	~ 258	~ 140
30	$\sim 10^9$	$\sim 1.5x10^5$	~ 4140	~ 1650
40	$\sim 10^{12}$	$\sim 8.4x10^6$	$\sim 6.7x10^4$	$\sim 2x10^4$
50	$\sim 10^{15}$	$\sim 4.5x10^8$	$\sim 1.1x10^6$	$\sim 2.3x10^5$
75	$\sim 10^{22}$	$\sim 9.7x10^{12}$	$\sim 1.1x10^9$	$\sim 1.1x10^8$
100	$\sim 10^{30}$	$\sim 2x10^{17}$	$\sim 1.1x10^{12}$	$\sim 5.3x10^{10}$
500	$\sim 10^{150}$	$\sim 3.9x10^{86}$	$\sim 1.9x10^{60}$	$\sim 4.1x10^{53}$

FPT – Bibliographie

k	$f(k)=2^k$	$f(k)=1.49^k$	$f(k)=1.32^k$	$f(k)=1.28^k$
10	$\sim 10^3$	~ 54	~ 16	~ 12
20	$\sim 10^6$	~ 2909	~ 258	~ 140
30	$\sim 10^9$	$\sim 1.5x10^5$	~ 4140	~ 1650
40	$\sim 10^{12}$	$\sim 8.4x10^6$	$\sim 6.7x10^4$	$\sim 2x10^4$
50	$\sim 10^{15}$	$\sim 4.5x10^8$	$\sim 1.1x10^6$	$\sim 2.3x10^5$
75	$\sim 10^{22}$	$\sim 9.7x10^{12}$	$\sim 1.1x10^9$	$\sim 1.1x10^8$
100	$\sim 10^{30}$	$\sim 2x10^{17}$	$\sim 1.1x10^{12}$	$\sim 5.3x10^{10}$
500	$\sim 10^{150}$	$\sim 3.9x10^{86}$	$\sim 1.9x10^{60}$	$\sim 4.1x10^{53}$

- trois de ces 4 fonctions $f(k)$ (2^k, 1.32^k, 1.28^k)
- sont trois algorithmes FPT pour résoudre le MINIMUM VERTEX-COVER (MIN-VC)
- avec k=taille du Vertex Cover que l'on cherche à atteindre

Considérations générales

- $f(k)$ devient un frein lorsque sa valeur est $\sim 10^9$
- un problème Pb peut être FPT pour un paramètre k_1 et pas FPT pour un autre paramètre k_2
- toujours préciser quel paramètre on étudie : couple (Pb, k)

Considérations générales

- $f(k)$ devient un frein lorsque sa valeur est $\sim 10^9$
- un problème Pb peut être FPT pour un paramètre k_1 et pas FPT pour un autre paramètre k_2
- toujours préciser quel paramètre on étudie : couple (Pb, k)

- branche de l'algorithmique datant des années 1990
- essor conséquent dans les années 2000
- nouvelles techniques, nouveaux algorithmes

Plan à suivre

- choix du paramètre k
- techniques pour montrer qu'un (Pb,k) est FPT (+ exemples)
- techniques visant à montrer qu'un (Pb,k) n'est pas FPT (+ exemples)

Sommaire

Choix de k : l'exemple de SAT

SAT

Instance : une formule ϕ sous FNC, contenant m clauses définies à partir de n variables $x_1, x_2 \ldots x_n$

Question : ϕ peut-elle être satisfaite ?

Choix de k : l'exemple de SAT

SAT
Instance : une formule ϕ sous FNC, contenant m clauses définies à partir de n variables $x_1, x_2 \dots x_n$
Question : ϕ peut-elle être satisfaite ?

Example

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$$

Plusieurs paramètres possibles $\Rightarrow$ (SAT,k)

Choix de k : l'exemple de SAT

- k=Taille de la clause
- (plus précisément : k = longueur maximale d'une clause)

Choix de k : l'exemple de SAT

- k=Taille de la clause
- (plus précisément : k = longueur maximale d'une clause)
- Discussion :
  - 2-SAT est dans P
  - k-SAT est NP-complet pour tout $k \geq 3$

Choix de k : l'exemple de SAT

- k=Taille de la clause
- (plus précisément : k = longueur maximale d'une clause)
- Discussion :
  - 2-SAT est dans P
  - k-SAT est NP-complet pour tout $k \geq 3$

(SAT,k) n'est pas FPT

$\Rightarrow$ sinon (par ex. avec $k = 3$) on prouve que P=NP !

Choix de k : l'exemple de SAT

Autres possibilités :

- k=Nombre de variables (donc $k = n$)
  - sur l'exemple, $n = 3$
  - 2^n cas à tester ; chaque test (vérifier si ϕ est satisfaite) en temps polynomial
  - $\Rightarrow$ (SAT,n) est FPT

Choix de k : l'exemple de SAT

Autres possibilités :

- k=Nombre de variables (donc $k = n$)
  - sur l'exemple, $n = 3$
  - 2^n cas à tester ; chaque test (vérifier si ϕ est satisfaite) en temps polynomial
  - $\Rightarrow$ (SAT,n) est FPT
- k=Nombre de clauses (donc $k = m$)
  - sur l'exemple, $m = 5$
  - un algorithme FPT avec $f(m) = 1.49^m$ existe (non démontré)
  - $\Rightarrow$ (SAT,m) est FPT

Choix de k : l'exemple de SAT

Autre possibilité :

- k=Longueur de la formule ($k = \ell$) = nombre total de littéraux utilisés
  - sur l'exemple, $\ell = 11$
  - un algorithme FPT avec $f(\ell) = 1.08^{\ell}$ existe (non démontré)
  - $\Rightarrow$ (SAT,ℓ) est FPT

Choix de k : l'exemple de SAT

Autre possibilité :

- k=Longueur de la formule ($k = \ell$) = nombre total de littéraux utilisés
  - sur l'exemple, $\ell = 11$
  - un algorithme FPT avec $f(\ell) = 1.08^\ell$ existe (non démontré)
  - $\Rightarrow$ (SAT,ℓ) est FPT

Remarque : grandes instances $\Rightarrow$ n, m et ℓ sont grands !

Peu de chances que le FPT soit praticable à partir d'une certaine taille

Choix de k : l'exemple de SAT

Plus généralement :

- Problèmes de minimisation : paramètre "naturel" = taille de la solution recherchée

Choix de k : l'exemple de SAT

Plus généralement :

- Problèmes de minimisation : paramètre "naturel" = taille de la solution recherchée
  - MIN-VC : paramètre = nombre de sommets formant un Vertex Cover

Choix de k : l'exemple de SAT

Plus généralement :

- Problèmes de minimisation : paramètre "naturel" = taille de la solution recherchée
  - MIN-VC : paramètre = nombre de sommets formant un Vertex Cover
  - MIN-BIN-PACKING : paramètre = nombre de "bins" utilisés

Choix de k : l'exemple de SAT

Plus généralement :

- Problèmes de minimisation : paramètre "naturel" = taille de la solution recherchée
  - MIN-VC : paramètre = nombre de sommets formant un Vertex Cover
  - MIN-BIN-PACKING : paramètre = nombre de "bins" utilisés
- Problèmes de maximisation : paramètre "naturel" = "complément" de la taille de la solution

Choix de k : l'exemple de SAT

Plus généralement :

- Problèmes de minimisation : paramètre "naturel" = taille de la solution recherchée
  - MIN-VC : paramètre = nombre de sommets formant un Vertex Cover
  - MIN-BIN-PACKING : paramètre = nombre de "bins" utilisés
- Problèmes de maximisation : paramètre "naturel" = "complément" de la taille de la solution
  - MAX-CUT : paramètre = nombre d'arêtes n'appartenant pas à la coupure

Choix de k : l'exemple de SAT

Plus généralement :

- Problèmes de minimisation : paramètre "naturel" = taille de la solution recherchée
  - MIN-VC : paramètre = nombre de sommets formant un Vertex Cover
  - MIN-BIN-PACKING : paramètre = nombre de "bins" utilisés
- Problèmes de maximisation : paramètre "naturel" = "complément" de la taille de la solution
  - MAX-CUT : paramètre = nombre d'arêtes n'appartenant pas à la coupure
  - MAX-SAT : paramètre = nombre de clauses non satisfaites

Choix de k : l'exemple de SAT

Plus généralement :

- Problèmes de minimisation : paramètre "naturel" = taille de la solution recherchée
  - MIN-VC : paramètre = nombre de sommets formant un Vertex Cover
  - MIN-BIN-PACKING : paramètre = nombre de "bins" utilisés
- Problèmes de maximisation : paramètre "naturel" = "complément" de la taille de la solution
  - MAX-CUT : paramètre = nombre d'arêtes n'appartenant pas à la coupure
  - MAX-SAT : paramètre = nombre de clauses non satisfaites
- Problèmes de Décision : très variable, mais garder à l'esprit que k ne doit pas être trop grand

Sommaire

Pour montrer que (Pb,k) est FPT

Plusieurs techniques pour démontrer qu'un (Pb,k) est FPT :

- Arbre de recherche borné
- Kernelization
- Compression Itérative
- Programmation Dynamique
- Color Coding

Arbre de Recherche Borné

Principe Général :

- Arbre de recherche (test de toutes les possibilités)
- taille de l'arbre de recherche qui ne dépend que de k (le paramètre)
- souvent : utilise des propriétés du problème (et plus précisément de la solution voulue)

Arbre de Recherche Borné

Principe Général :

- Arbre de recherche (test de toutes les possibilités)
- taille de l'arbre de recherche qui ne dépend que de k (le paramètre)
- souvent : utilise des propriétés du problème (et plus précisément de la solution voulue)

Example (CLUSTER EDITING (CE))

CLUSTER EDITING (CE)
Instance : Un graphe $G = (V, E)$, un entier k
Question : Peut-on, après insertion ou suppression d'au plus k arêtes, obtenir un graphe G' qui soit une union disjointe de cliques ?

Arbre de Recherche Borné

- motivation : clustering est très utilisé en bio-informatique et en traitement d'images
- insertion/suppression d'arêtes car la donnée d'entrée (le graphe) est bruitée
- il faut la "réparer" pour obtenir une donnée fiable (= des clusters bien identifiés = des cliques)

Arbre de Recherche Borné

- motivation : clustering est très utilisé en bio-informatique et en traitement d'images
- insertion/suppression d'arêtes car la donnée d'entrée (le graphe) est bruitée
- il faut la "réparer" pour obtenir une donnée fiable (= des clusters bien identifiés = des cliques)

But : montrer que (CE,k) est FPT, en utilisant un arbre de recherche borné

Arbre de Recherche Borné

Theorem

Un graphe $G = (V, E)$ est une union disjointe de cliques si et seulement si on ne peut pas trouver 3 sommets u, v, w tels que (u, v) et $(u, w) \in E$, alors que $(v, w) \notin E$

Preuve :

Arbre de Recherche Borné

Theorem

Un graphe $G = (V, E)$ est une union disjointe de cliques si et seulement si on ne peut pas trouver 3 sommets u, v, w tels que (u, v) et $(u, w) \in E$, alors que $(v, w) \notin E$

Preuve :

$(\Rightarrow)$ si G est une union disjointe de cliques, tout triplet de sommets u, v, w induit 0, 1 ou 3 arêtes :

Arbre de Recherche Borné

Theorem

Un graphe $G = (V, E)$ est une union disjointe de cliques si et seulement si on ne peut pas trouver 3 sommets u, v, w tels que (u, v) et $(u, w) \in E$, alors que $(v, w) \notin E$

Preuve :
$(\Rightarrow)$ si G est une union disjointe de cliques, tout triplet de sommets u, v, w induit 0, 1 ou 3 arêtes :

- u, v, w dans 3 cliques différentes : 0 arête
- 2 sommets dans une clique K_a, le 3e dans un autre clique K_b : 1 arête
- u, v, w dans la même clique : 3 arêtes
- $\Rightarrow$ 2 arêtes induites est impossible

Arbre de Recherche Borné

Theorem

Un graphe $G = (V, E)$ est une union disjointe de cliques si et seulement si on ne peut pas trouver 3 sommets u, v, w tels que (u, v) et $(u, w) \in E$, alors que $(v, w) \notin E$

Arbre de Recherche Borné

Theorem

Un graphe $G = (V, E)$ est une union disjointe de cliques si et seulement si on ne peut pas trouver 3 sommets u, v, w tels que (u, v) et $(u, w) \in E$, alors que $(v, w) \notin E$

Preuve :

$(\Leftarrow)$ Si G n'est pas une union disjointe de cliques :

- il y a deux sommets u et v dans la même composante connexe, tels que $(u, v) \notin E$

Arbre de Recherche Borné

Theorem

Un graphe $G = (V, E)$ est une union disjointe de cliques si et seulement si on ne peut pas trouver 3 sommets u, v, w tels que (u, v) et $(u, w) \in E$, alors que $(v, w) \notin E$

Preuve :

$(\Leftarrow)$ Si G n'est pas une union disjointe de cliques :

- il y a deux sommets u et v dans la même composante connexe, tels que $(u, v) \notin E$
- si u et v sont voisins d'un même sommet w : OK (il existe u, v, w tels que...)

Arbre de Recherche Borné

Theorem

Un graphe $G = (V, E)$ est une union disjointe de cliques si et seulement si on ne peut pas trouver 3 sommets u, v, w tels que (u, v) et $(u, w) \in E$, alors que $(v, w) \notin E$

Preuve :

$(\Leftarrow)$ Si G n'est pas une union disjointe de cliques :

- il y a deux sommets u et v dans la même composante connexe, tels que $(u, v) \notin E$
- si u et v sont voisins d'un même sommet w : OK (il existe u, v, w tels que...)
- sinon, le chemin le plus court, P, qui relie u à v est de longueur ≥ 3 : $P = (u, u_1, u_2 \ldots v)$
- $\rightarrow (u, u2) \notin E$, sinon P ne serait pas le chemin le plus court

Arbre de Recherche Borné

Theorem

Un graphe $G = (V, E)$ est une union disjointe de cliques si et seulement si on ne peut pas trouver 3 sommets u, v, w tels que (u, v) et $(u, w) \in E$, alors que $(v, w) \notin E$

Preuve :

($\Leftarrow$) Si G n'est pas une union disjointe de cliques :

- il y a deux sommets u et v dans la même composante connexe, tels que $(u, v) \notin E$
- si u et v sont voisins d'un même sommet w : OK (il existe u, v, w tels que...)
- sinon, le chemin le plus court, P, qui relie u à v est de longueur ≥ 3 : $P = (u, u_1, u_2 \ldots v)$
- $\rightarrow (u, u2) \notin E$, sinon P ne serait pas le chemin le plus court
- $\Rightarrow$ les 3 sommets recherchés sont u, u_1, u_2

Arbre de Recherche Borné

On se base sur les décisions suivantes :

1. si G est une union disjointe de cliques : STOP et OK

Arbre de Recherche Borné

On se base sur les décisions suivantes :

1. si G est une union disjointe de cliques : STOP et OK
2. si $k \geq 0$: pas de solution trouvée

Arbre de Recherche Borné

On se base sur les décisions suivantes :

1. si G est une union disjointe de cliques : STOP et OK
2. si $k \geq 0$: pas de solution trouvée
3. sinon, il existe trois sommets u, v, w tels que (u, v) et (u, w) sont dans E, alors que (v, w) n'y est pas (voir Théorème)

Arbre de Recherche Borné

Dans le cas 3., continuer la recherche récursivement (3 cas possibles) :

- $E' = E \cup \{(v, w)\}$, $k' = k - 1$ (on a inséré une arête)

Arbre de Recherche Borné

Dans le cas 3., continuer la recherche récursivement (3 cas possibles) :

- $E' = E \cup \{(v, w)\}$, $k' = k - 1$ (on a inséré une arête)
- $E' = E - \{(u, v)\}$, $k' = k - 1$ (on a supprimé une arête)

Arbre de Recherche Borné

Dans le cas 3., continuer la recherche récursivement (3 cas possibles) :

- $E' = E \cup \{(v,w)\}$, $k' = k - 1$ (on a inséré une arête)
- $E' = E - \{(u,v)\}$, $k' = k - 1$ (on a supprimé une arête)
- $E' = E - \{(u,w)\}$, $k' = k - 1$ (on a supprimé une arête)

Arbre de Recherche Borné

A chaque itération :

- k est décrémenté
- il y a 3 possibilités

Arbre de Recherche Borné

A chaque itération :

- k est décrémenté
- il y a 3 possibilités
- $\Rightarrow$ le nombre de feuilles de l'arbre de recherche est $\leq 3^k$
- $\Rightarrow$ l'arbre est de taille $O(3^k)$

Arbre de Recherche Borné

A chaque itération :

- k est décrémenté
- il y a 3 possibilités
- $\Rightarrow$ le nombre de feuilles de l'arbre de recherche est $\leq 3^k$
- $\Rightarrow$ l'arbre est de taille $O(3^k)$
- Gagné ! : l'arbre de recherche a une taille qui ne dépend que du paramètre k

Autre technique : la kernelization

Principe Général :

- Traiter rapidement (c'est-à-dire en temps polynomial) les parties de l'instance qui sont simples à résoudre

Autre technique : la kernelization

Principe Général :

- Traiter rapidement (c'est-à-dire en temps polynomial) les parties de l'instance qui sont simples à résoudre
- $\rightarrow$ l'instance I de départ est transformée une instance I' (en temps polynomial)
- I' est le "noyau" (=kernel), la partie difficile à résoudre

Autre technique : la kernelization

Principe Général :

- Traiter rapidement (c'est-à-dire en temps polynomial) les parties de l'instance qui sont simples à résoudre
- $\rightarrow$ l'instance I de départ est transformée une instance I' (en temps polynomial)
- I' est le "noyau" (=kernel), la partie difficile à résoudre
- si I' ne dépend (pour sa résolution) que du paramètre k, alors le problème (Pb,k) est FPT
- le passage de I à I' se fait par des règles de réduction des données (data reduction)

Autre technique : la kernelization

Principe Général :

- Traiter rapidement (c'est-à-dire en temps polynomial) les parties de l'instance qui sont simples à résoudre
- $\rightarrow$ l'instance I de départ est transformée une instance I' (en temps polynomial)
- I' est le "noyau" (=kernel), la partie difficile à résoudre
- si I' ne dépend (pour sa résolution) que du paramètre k, alors le problème (Pb,k) est FPT
- le passage de I à I' se fait par des règles de réduction des données (data reduction)

Remarques :

- Data Reduction = principe général, pas seulement utile/utilisé dans le cadre du FPT
- ex : le logiciel CPLX (dans le cadre de l'ILP) utilise massivement des règles de Data Reduction

Kernelization

Example (VERTEX COVER (VC))

VERTEX COVER (VC, version décision du problème MIN-VC)
Instance : Un graphe $G = (V, E)$, un entier k
Question : Existe-t-il un Vertex Cover de G, de cardinalité $\leq k$?

Kernelization

Example (VERTEX COVER (VC))

VERTEX COVER (VC, version décision du problème MIN-VC)
Instance : Un graphe $G = (V, E)$, un entier k
Question : Existe-t-il un Vertex Cover de G, de cardinalité $\leq k$?

Rappel : Vertex Cover = couverture par les sommets = ensemble de sommets V' tel que :

- pour toute arête $(u, v) \in E$
- au moins un des deux sommets parmi u et v appartient à V'

Kernelization

But :

- se donner une ou des règle(s) de réduction
- pour se ramener à un graphe dont la taille (nombre de sommets et nombre d'arêtes) ne dépend que de k

Kernelization

But :

- se donner une ou des règle(s) de réduction
- pour se ramener à un graphe dont la taille (nombre de sommets et nombre d'arêtes) ne dépend que de k
- ici, deux règles : VC_1 et VC_2

Kernelization

Règle VC_1

S'il existe un sommet u de degré 1 dans G, dont le voisin est v :

- mettre v dans VC
- $k \rightarrow k - 1$
- $G \rightarrow G - \{u, v,$ toutes les arêtes incidentes à $v\}$

Kernelization

Règle VC_1

S'il existe un sommet u de degré 1 dans G, dont le voisin est v :

- mettre v dans VC
- $k \to k - 1$
- $G \to G - \{u, v,$ toutes les arêtes incidentes à $v\}$

Justification Règle VC_1 :

- $(u, v) \in E \to$ au moins une des deux extrémités est présente dans tout VC
- u ne couvre que (u, v) (car il est de degré 1)
- v couvre potentiellement d'autres arêtes
- $\to$ v suffit, et il est au moins "aussi bon" que u

Kernelization

Remarque :

- si une solution existe, une règle de réduction ne fournit pas toutes les solutions, mais une seule

Kernelization

Remarque :

- si une solution existe, une règle de réduction ne fournit pas toutes les solutions, mais une seule
- exemple : Règle VC_1 : il peut exister des VC de cardinalité $\leq k$ qui contiennent un ou des sommet(s) de degré 1

Kernelization

Règle VC_2

S'il existe un sommet de v degré $\geq k+1$ dans G :

- mettre v dans VC
- $k \leftarrow k-1$
- $G \leftarrow G$–$\{v$, toutes les arêtes incidentes à $v\}$

Kernelization

Règle VC_2

S'il existe un sommet de v degré $\geq k+1$ dans G :

- mettre v dans VC
- $k \leftarrow k-1$
- $G \leftarrow G-\{v$, toutes les arêtes incidentes à $v\}$

Justification Règle VC_2 :

- si on ne met pas v dans le VC $\rightarrow$ il faut y mettre l'ensemble de ses $p \geq k+1$ voisins
- $\Rightarrow$ pas de VC de cardinalité $\leq k$
- seule possibilité pour une réponse positive : mettre v dans le VC

Autre technique : la kernelization

Idée :

- appliquer itérativement les règles VC_1 et VC_2
- ordre arbitraire
- STOP quand plus aucune règle ne peut s'appliquer

Autre technique : la kernelization

Idée :

- appliquer itérativement les règles VC_1 et VC_2
- ordre arbitraire
- STOP quand plus aucune règle ne peut s'appliquer

$\rightarrow$ on obtient une instance réduite I', composée de :

- un nouveau paramètre $k' \leq k$
- un nouveau graphe G' à n' sommets et m' arêtes

Autre technique : la kernelization

But : montrer que la taille de G' ne dépend que de k (le paramètre initial)

Autre technique : la kernelization

But : montrer que la taille de G' ne dépend que de k (le paramètre initial)

- on peut supposer que G' n'a pas de sommet isolé (ils n'interviennent pas dans le VC, donc on les ignore)

Autre technique : la kernelization

But : montrer que la taille de G' ne dépend que de k (le paramètre initial)

- on peut supposer que G' n'a pas de sommet isolé (ils n'interviennent pas dans le VC, donc on les ignore)
- G' est de degré maximum k, donc un sommet du VC ne peut pas couvrir plus de k arêtes
- il ne reste que k' sommets à mettre dans le VC $\Rightarrow$ pas plus de $k' \cdot k$ arêtes dans G' (sinon : réponse “NON”)

Autre technique : la kernelization

But : montrer que la taille de G' ne dépend que de k (le paramètre initial)

- on peut supposer que G' n'a pas de sommet isolé (ils n'interviennent pas dans le VC, donc on les ignore)
- G' est de degré maximum k, donc un sommet du VC ne peut pas couvrir plus de k arêtes
- il ne reste que k' sommets à mettre dans le VC $\Rightarrow$ pas plus de $k' \cdot k$ arêtes dans G' (sinon : réponse "NON")
- comme $k' \leq k \Rightarrow G'$ possède $\leq k^2$ arêtes

Autre technique : la kernelization

Conclusion :

- tous les sommets de G' sont de degré ≤ 2

Autre technique : la kernelization

Conclusion :

- tous les sommets de G' sont de degré ≤ 2
- pour tout graphe, Σ degrés $= 2\times$nombre d'arêtes
- $\rightarrow$ $n' \leq k^2$ (n' = nombre de sommets de G')

Autre technique : la kernelization

Conclusion :

- tous les sommets de G' sont de degré ≤ 2
- pour tout graphe, Σ degrés $= 2\times$nombre d'arêtes
- $\rightarrow$ $n' \leq k^2$ (n' = nombre de sommets de G')
- Gagné ! : nombre de sommets et nombre d'arêtes de G' ne dépendent que de k

Autre technique : la kernelization

Pour résoudre le problème (VC, k) sur G' : méthode “brute force” (=exhaustive)

Autre technique : la kernelization

Pour résoudre le problème (VC,k) sur G' : méthode "brute force" (=exhaustive)

Plus précisément :

- choisir k' éléments parmi n' (rappels : $k' \leq k$ et $n' \leq k^2$)
- tester pour chaque ensemble de sommets généré si c'est un VC

Sommaire

Pour prouver que (Pb,k) n'est pas FPT

Techniques sont semblables à celles qui concernent l'inapproximabilité :

- par raisonnement, ou
- par réduction

Pour prouver que (Pb,k) n'est pas FPT

Techniques sont semblables à celles qui concernent l'inapproximabilité :

- par raisonnement, ou
- par réduction

Raisonnement : le plus souvent :

- Rappel : HRC = Hypothèses Raisonnables de Complexité $\rightarrow$ on suppose P$\neq$NP

Pour prouver que (Pb,k) n'est pas FPT

Techniques sont semblables à celles qui concernent l'inapproximabilité :

- par raisonnement, ou
- par réduction

Raisonnement : le plus souvent :

- Rappel : HRC = Hypothèses Raisonnables de Complexité $\rightarrow$ on suppose P$\neq$NP
- (Pb,k) veut dire que si $k = O(1)$, alors Pb peut se résoudre (de manière exacte) en temps polynomial
- si on sait que Pb est NP-complet même pour $k = O(1)$, on conclut (sous HRC) que (Pb,k) n'est pas FPT

Pour prouver que (Pb, k) n'est pas FPT

Example (Coloration de Graphes (k-COL))

k-COL

Instance : Graphe $G = (V, E)$, entier k

Question : Existe-t-il une coloration propre de G utilisant au plus k couleurs ?

Pour prouver que (Pb, k) n'est pas FPT

- on sait que 2-COL est polynomial (VRAI si et seulement si G est biparti)
- on sait que k-COL est NP-complet pour tout $k \geq 3$

Pour prouver que (Pb,k) n'est pas FPT

- on sait que 2-COL est polynomial (VRAI si et seulement si G est biparti)
- on sait que k-COL est NP-complet pour tout $k \geq 3$
- $\rightarrow$ sous HRC, (k-COL,k) n'est pas FPT (sinon 3-COL peut se résoudre en temps polynomial $\rightarrow$ contredit HRC)

Pour prouver que (Pb, k) n'est pas FPT

- on sait que 2-COL est polynomial (VRAI si et seulement si G est biparti)
- on sait que k-COL est NP-complet pour tout $k \geq 3$
- $\rightarrow$ sous HRC, (k-COL, k) n'est pas FPT (sinon 3-COL peut se résoudre en temps polynomial $\rightarrow$ contredit HRC)

Autre exemple : k-SAT (même raison : 2-SAT est dans P, k-SAT est NP-complet pour tout $k \geq 3$). Voir transparent 12.

Pour prouver que (Pb, k) n'est pas FPT

Par réduction :

- en réalité, FPT est une classe de complexité (comme P, NP, APX, etc.)

Pour prouver que (Pb, k) n'est pas FPT

Par réduction :

- en réalité, FPT est une classe de complexité (comme P, NP, APX, etc.)
- (Pb, k) est dans FPT s'il existe un algorithme en $O(f(k) \cdot n^c)$ pour résoudre Pb
- FPT $\leftrightarrow$ P

Pour prouver que (Pb, k) n'est pas FPT

Par réduction :

- en réalité, FPT est une classe de complexité (comme P, NP, APX, etc.)
- (Pb, k) est dans FPT s'il existe un algorithme en $O(f(k) \cdot n^c)$ pour résoudre Pb
- FPT $\leftrightarrow$ P
- Classe W[1]-dur $\leftrightarrow$ NP-dur : (Pb, k) est W[1]-dur si un autre problème W[1]-dur (Pb', k) peut se réduire en lui par une réduction paramétrée
- Réduction paramétrée : réduction "classique" avec contraintes sur le paramètre k

Sommaire

In a nutshell

- Présentation de diverses stratégies visant à contourner la difficulté des problèmes NP-complets

In a nutshell

- Présentation de diverses stratégies visant à contourner la difficulté des problèmes NP-complets
- Compromis temps d'exécution/qualité de la solution. Stratégies possibles :
  - Montrer que P=NP
  - Petits cas, classes d'instances
  - Algorithmes d'approximation (FPTAS/PTAS/APX)
  - Algorithmes FPT

In a nutshell

- Présentation de diverses stratégies visant à contourner la difficulté des problèmes NP-complets
- Compromis temps d'exécution/qualité de la solution. Stratégies possibles :
  - Montrer que P=NP
  - Petits cas, classes d'instances
  - Algorithmes d'approximation (FPTAS/PTAS/APX)
  - Algorithmes FPT
- Branche très active de l'algorithmique, en plein essor depuis la fin des années 1990
- Toujours en développement, et de plus en plus en ce qui concerne le FPT

FPT – Bibliographie

- R. Niedermeier. Invitation to Fixed-Parameter Algorithms. Oxford University Press, 2006.
- R.G. Downey, M. R. Fellows : Fundamentals of Parameterized Complexity Springer-Verlag, 2013.
- Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, Saket Saurabh. Parameterized Algorithms. Springer-Verlag, 2015.