Introduction à la complexité paramétrée (1)
(Complexité avancée - UMIN 345)

Christophe PAUL
(CNRS - LIRMM)

November 4, 2008

Bibliographie

- *Parameterized Complexity*, R. Downey and M. Fellows, 1999.
- *Invitation to Fixed-Parameter Algorithms*, R. Niedermeier, 2006.
- *Parameterized Complexity Theory*, J. Flum and M. Grohe, 2006.

"Mesurer la complexité seulement en fonction de la taille de la donnée signifie ignorer tout information structurelle sur l'instance donnée..."

J. Flum and M. Grohe

"Mesurer la complexité seulement en fonction de la taille de la donnée signifie ignorer tout information structurelle sur l'instance donnée..."

J. Flum and M. Grohe

"L'idée fondamentale est de restreindre l'explosion combinatoire, semble-t-il inévitable, qui est responsable de la croissance exponentielle du temps de calcul, à un paramètre spécifique au problème..."

R. Niedermeier

L'exemple de SAT

Mesure de la complexité en fonction de différents paramètres:

1. **taille des clauses :** $k =$ nombre max. de littéraux par clause
   $k = 2$: SAT $\in$ **P**
   $k \geqslant 3$: SAT $\in$ **NP**-complet

L'exemple de SAT

Mesure de la complexité en fonction de différents paramètres:

1. **taille des clauses :** $k =$ nombre max. de littéraux par clause
   $k = 2$: SAT $\in$ **P**
   $k \geqslant 3$: SAT $\in$ **NP**-complet
2. **nombre de variables :** $n =$ nombre de variables
   Il y a 2^n affectations possibles
   Si on se restreint à 3-SAT, la complexité tombe à $O(1,49^n)$

L'exemple de SAT

Mesure de la complexité en fonction de différents paramètres:

1. **taille des clauses :** $k =$ nombre max. de littéraux par clause
   $k = 2$: SAT $\in$ **P**
   $k \geqslant 3$: SAT $\in$ **NP**-complet
2. **nombre de variables :** $n =$ nombre de variables
   Il y a 2^n affectations possibles
   Si on se restreint à 3-SAT, la complexité tombe à $O(1,49^n)$
3. **nombre de clauses :** $m =$ nombre de clauses
   On obtient une complexité en $O(1,24^m)$

L'exemple de SAT

Mesure de la complexité en fonction de différents paramètres:

1. **taille des clauses :** $k =$ nombre max. de littéraux par clause
   $k = 2$: SAT $\in$ **P**
   $k \geqslant 3$: SAT $\in$ **NP**-complet
2. **nombre de variables :** $n =$ nombre de variables
   Il y a 2^n affectations possibles
   Si on se restreint à 3-SAT, la complexité tombe à $O(1,49^n)$
3. **nombre de clauses :** $m =$ nombre de clauses
   On obtient une complexité en $O(1,24^m)$
4. **longueur de la formule :** $l =$ nombre total de littéraux
   On obtient une complexité en $O(1,08^l)$

L'exemple de SAT

Mesure de la complexité en fonction de différents paramètres:

1. **taille des clauses :** $k =$ nombre max. de littéraux par clause
   $k = 2$: SAT $\in$ **P**
   $k \geqslant 3$: SAT $\in$ **NP**-complet
2. **nombre de variables :** $n =$ nombre de variables
   Il y a 2^n affectations possibles
   Si on se restreint à 3-SAT, la complexité tombe à $O(1,49^n)$
3. **nombre de clauses :** $m =$ nombre de clauses
   On obtient une complexité en $O(1,24^m)$
4. **longueur de la formule :** $l =$ nombre total de littéraux
   On obtient une complexité en $O(1,08^l)$
5. **structure de la formule :** paramètres basés par exemple sur la structure du graphe de dépendances...

L'exemple de Vertex Cover

- **Données :** Un graphe $G = (V, E)$ et un entier positif k
- **Question :** $\exists$? k sommets tels que toute arête est incidente à au moins un de ces k sommets

L'exemple de Vertex Cover

- **Données :** Un graphe $G = (V, E)$ et un entier positif k
- **Question :** $\exists$? k sommets tels que toute arête est incidente à au moins un de ces k sommets

Paramètre naturel : la taille de la solution k

L'exemple de Vertex Cover

- **Données :** Un graphe $G = (V, E)$ et un entier positif k
- **Question :** $\exists$? k sommets tels que toute arête est incidente à au moins un de ces k sommets

Paramètre naturel : la taille de la solution k

Observation

Si $S \subseteq V$ est une couverture et $e = (u, v)$ une arête de G, alors $u \in S$ ou $v \in S$

L'exemple de Vertex Cover

- **Données :** Un graphe $G = (V, E)$ et un entier positif k
- **Question :** $\exists$? k sommets tels que toute arête est incidente à au moins un de ces k sommets

Paramètre naturel : la taille de la solution k

Observation

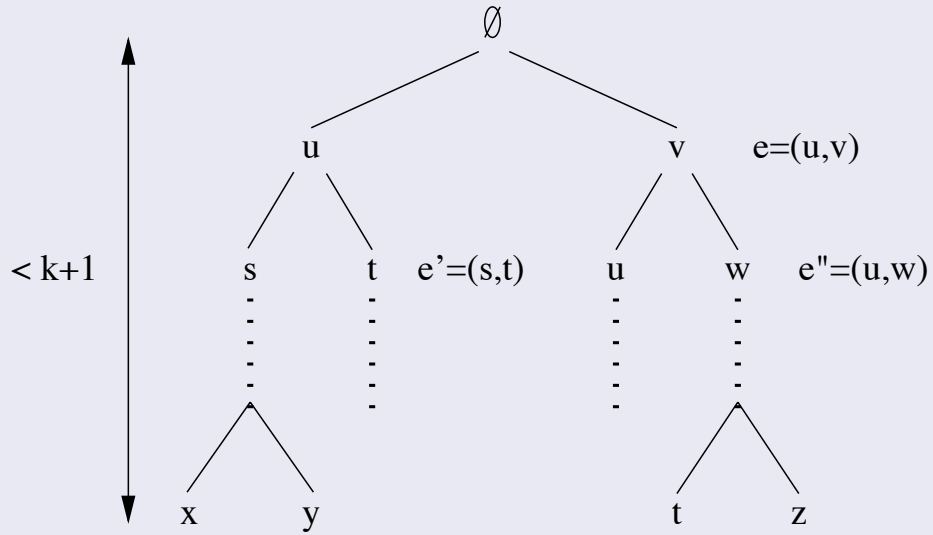
Si $S \subseteq V$ est une couverture et $e = (u, v)$ une arête de G, alors $u \in S$ ou $v \in S$

Preuve : Si $\{u, v\} \cap S \neq \emptyset$, l'arêtre e ne peut être couverte.

Un arbre de recherche pour Vertex Cover

VC(G, S) : Soit $e = (u, v)$ une arête de G,

1. $S = S \cup \{u\}$
   Si S ne couvre pas E et **si** $|S| < k$, VC$(G - u, S)$
2. $S = S \cup \{v\}$
   Si S ne couvre pas E et **si** $|S| < k$, VC$(G - v, S)$

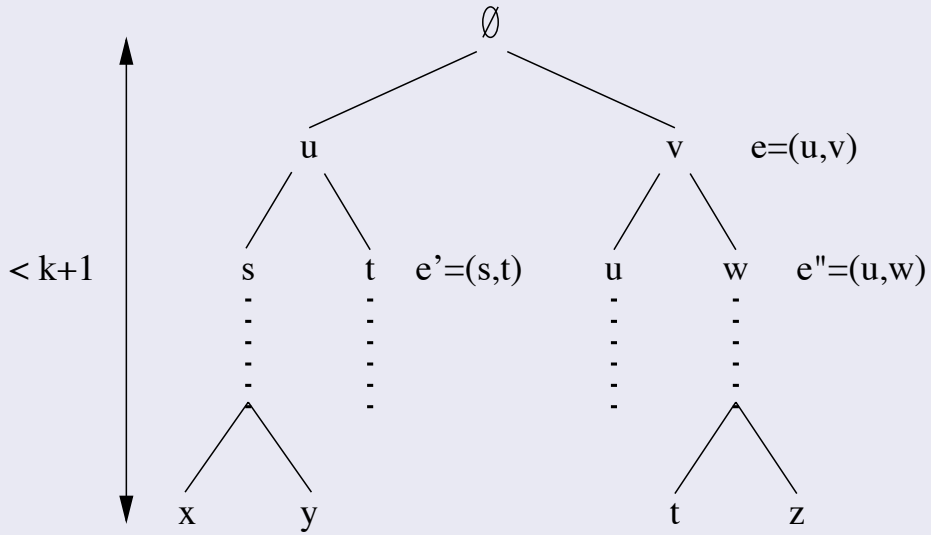


Un arbre de recherche pour Vertex Cover

VC(G, S) : Soit $e = (u, v)$ une arête de G,

1. $S = S \cup \{u\}$
   Si S ne couvre pas E et **si** $|S| < k$, VC$(G - u, S)$
2. $S = S \cup \{v\}$
   Si S ne couvre pas E et **si** $|S| < k$, VC$(G - v, S)$

Complexité : $2^k.n^{O(1)}$



L'exemple de Ensemble Indépendant

- **Données :** Un graphe $G = (V, E)$ et un entier positif k
- **Question :** $\exists$? k sommets deux à deux non-adjacents

L'exemple de Ensemble Indépendant

- **Données :** Un graphe $G = (V, E)$ et un entier positif k
- **Question :** $\exists$? k sommets deux à deux non-adjacents

Observation

- Algorithme "brut-force" : $O(n^k)$
  MAIS l'exposant n'est pas une constante!

L'exemple de Ensemble Indépendant

- **Données :** Un graphe $G = (V, E)$ et un entier positif k
- **Question :** $\exists$? k sommets deux à deux non-adjacents

Observation

G possède un Ensemble Indépendant de taille k ssi G possède un Vertex Cover de taille $n - k$.

- Algorithme "brut-force" : $O(n^k)$
  MAIS l'exposant n'est pas une constante!

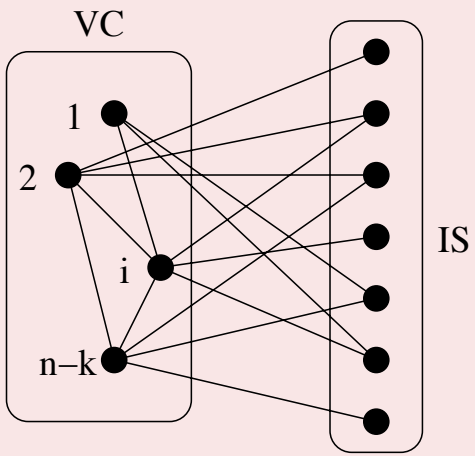
L'exemple de Ensemble Indépendant

- **Données :** Un graphe $G = (V, E)$ et un entier positif k
- **Question :** $\exists$? k sommets deux à deux non-adjacents

Observation

G possède un Ensemble Indépendant de taille k ssi G possède un Vertex Cover de taille $n - k$.

- Algorithme "brut-force" : $O(n^k)$
  MAIS l'exposant n'est pas une constante!
- L'arbre de recherche "à la Vertex Cover" donne $2^{(n-k)}.n^{O(1)}$
  MAIS l'explosion combinatoire est fonction de n



On recherche des problèmes qui

- sont **NP**;
- admettent un paramètre "*naturel*" k; et
- peuvent se résoudre par un algorithme $\mathcal{A}$ de complexité $f(k).n^{O(1)}$.

On recherche des problèmes qui

- sont **NP**;
- admettent un paramètre "*naturel*" k; et
- peuvent se résoudre par un algorithme $\mathcal{A}$ de complexité $f(k).n^{O(1)}$.

Remarque

La fonction f est quelconque et ne dépend que du paramètre k.
La fonction suivante est donc valide:

$$f(k) = 2^{2^{2^{2^{2^{2^{2^{2^{2^{2^{2^{2^{2^{k}}}}}}}}}}}}}$$

Définition

Soit Σ un alphabet fini.

1. Une **paramétrisation** de Σ^* est une fonction $\kappa : \Sigma^* \longrightarrow \mathbb{N}$ calculable en temps polynomial.

Définition

Soit Σ un alphabet fini.

1. Une **paramétrisation** de Σ^* est une fonction $\kappa : \Sigma^* \longrightarrow \mathbb{N}$ calculable en temps polynomial.
2. Un **problème paramétré** (sur Σ) est une paire (Q, κ) tel que $Q \subseteq \Sigma^*$ et κ est une paramétrisation de Σ^*.

   $x \in \Sigma^*$ est une instance de Q et $\kappa(x)$ est le paramètre correspondant.

Définition

Soit Σ un alphabet fini.

1. Une **paramétrisation** de Σ^* est une fonction $\kappa : \Sigma^* \to \mathbb{N}$ calculable en temps polynomial.
2. Un **problème paramétré** (sur Σ) est une paire (Q, κ) tel que $Q \subseteq \Sigma^*$ et κ est une paramétrisation de Σ^*.

   $x \in \Sigma^*$ est une instance de Q et $\kappa(x)$ est le paramètre correspondant.

Exemple : Vertex Cover

- *Données :* Un graphe $G = (V, E)$
- *Paramètre :* Un entier $\kappa(G)$
- *Question :* G admet-il un Vertex Cover de taille $\kappa(G)$?

Définition

Soit Σ un alphabet fini et $\kappa : \Sigma^* \longrightarrow \mathbb{N}$ une paramétrisation.

1. Un algorithme $\mathcal{A}$ sur Σ est un **algorithme paramétré** par κ s'il existe une fonction calculable $f : \mathbb{N} \longrightarrow \mathbb{N}$ tel que $\forall x \in \Sigma^*$, la complexité de $\mathcal{A}$ est:

$$f(\kappa(x)).n^{O(1)}$$

Définition

Soit Σ un alphabet fini et $\kappa : \Sigma^* \longrightarrow \mathbb{N}$ une paramétrisation.

1. Un algorithme $\mathcal{A}$ sur Σ est un **algorithme paramétré** par κ s'il existe une fonction calculable $f : \mathbb{N} \longrightarrow \mathbb{N}$ tel que $\forall x \in \Sigma^*$, la complexité de $\mathcal{A}$ est:
$$f(\kappa(x)).n^{O(1)}$$
2. Un problème (Q, κ) est **FPT** (**Fixed Parameterized Tractable**) s'il existe un algorithme $\mathcal{A}$ paramétré par κ qui décide Q.

Définition

Soit Σ un alphabet fini et $\kappa : \Sigma^* \longrightarrow \mathbb{N}$ une paramétrisation.

1. Un algorithme $\mathcal{A}$ sur Σ est un **algorithme paramétré** par κ s'il existe une fonction calculable $f : \mathbb{N} \longrightarrow \mathbb{N}$ tel que $\forall x \in \Sigma^*$, la complexité de $\mathcal{A}$ est:
$$f(\kappa(x)).n^{O(1)}$$
2. Un problème (Q, κ) est **FPT** (**Fixed Parameterized Tractable**) s'il existe un algorithme $\mathcal{A}$ paramétré par κ qui décide Q.

Observation

Tout problème $\Pi \in \mathbf{P}$ est **FPT**.

Définition

Soit (Q, κ) un problème paramétré et $l \in \mathbb{N}$. Le **niveau** t de (Q, κ) est le problème:

$$(Q, \kappa)_t = \{x \in Q \mid \kappa(x) = t\}$$

Définition

Soit (Q, κ) un problème paramétré et $l \in \mathbb{N}$. Le **niveau** t de (Q, κ) est le problème:

$$(Q, \kappa)_t = \{x \in Q \mid \kappa(x) = t\}$$

Observation

Soit (Q, κ) un problème paramétré et $t \in \mathbb{N}$. Si (Q, κ) est **FPT**, alors $(Q, \kappa)_t \in \mathbf{P}$.

Définition

Soit (Q, κ) un problème paramétré et $l \in \mathbb{N}$. Le **niveau** t de (Q, κ) est le problème:

$$(Q, \kappa)_t = \{x \in Q \mid \kappa(x) = t\}$$

Observation

Soit (Q, κ) un problème paramétré et $t \in \mathbb{N}$. Si (Q, κ) est **FPT**, alors $(Q, \kappa)_t \in$ **P** .

k-COLORATION $\in$ **FPT** ?

Définition

Soit (Q, κ) un problème paramétré et $l \in \mathbb{N}$. Le **niveau** t de (Q, κ) est le problème:

$$(Q, \kappa)_t = \{x \in Q \mid \kappa(x) = t\}$$

Observation

Soit (Q, κ) un problème paramétré et $t \in \mathbb{N}$. Si (Q, κ) est **FPT**, alors $(Q, \kappa)_t \in$ **P** .

k-COLORATION $\in$ FPT ?

Le problème 3-COLORATION$=$ $(\text{COLORATION}, \kappa)_3$ est **NP**-complet

$\Rightarrow$ k-COLORATION n'est pas **FPT**.

Une formule booléenne Φ est sous forme normale 3-disjonctive (3-DNF) si

$$\Phi = \bigvee_{i \in I} (\lambda_{i,1} \wedge \lambda_{i,2} \wedge \lambda_{i,3})$$

1. MAX-3-DNF-SAT (problème d'optimisation)
   - *Donnée :* une formule 3-DNF Φ.
   - *Objectif :* maximiser $1 + t$, avec t le nbre de termes satisfaits.

Une formule booléenne Φ est sous forme normale 3-disjonctive (3-DNF) si

$$\Phi = \bigvee_{i \in I} (\lambda_{i,1} \wedge \lambda_{i,2} \wedge \lambda_{i,3})$$

1. MAX-3-DNF-SAT (problème d'optimisation)
   - *Donnée :* une formule 3-DNF Φ.
   - *Objectif :* maximiser $1 + t$, avec t le nbre de termes satisfaits.
2. (MAX-3-DNF-SAT, k) (paramétrisation standard)
   - Existe-t-il une affectation des variables satisfaisant au moins k termes ?

Une formule booléenne Φ est sous forme normale 3-disjonctive (3-DNF) si

$$\Phi = \bigvee_{i \in I} (\lambda_{i,1} \wedge \lambda_{i,2} \wedge \lambda_{i,3})$$

1. MAX-3-DNF-SAT (problème d'optimisation)
   - *Donnée :* une formule 3-DNF Φ.
   - *Objectif :* maximiser $1 + t$, avec t le nbre de termes satisfaits.
2. (MAX-3-DNF-SAT, k) (paramétrisation standard)
   - Existe-t-il une affectation des variables satisfaisant au moins k termes ?
3. (EXACT-MAX-3-DNF-SAT, k)
   - Existe-t-il une affectation des variables satisfaisant exactement $k - 1$ termes ?

Exercice :

1. Montrer que $(\text{MAX-3-DNF-SAT}, k)$ est **FPT**.

Exercice :

1. Montrer que $(\text{MAX-3-DNF-SAT}, k)$ est **FPT**.
   - *Montrer que le nombre moyen de termes satisfaits dans une affectation aléatoire est au moins $\frac{m}{8}$ où m est le nombre total de termes*

Exercice :

1. Montrer que $(\text{MAX-3-DNF-SAT}, k)$ est **FPT**.
   - *Montrer que le nombre moyen de termes satisfaits dans une affectation aléatoire est au moins $\frac{m}{8}$ où m est le nombre total de termes*
     Soit X_i v.a. binaire telle que $X_i = 1$ ssi le terme λ_i est vrai

$$P(X_i = 1) = \frac{1}{8} \Rightarrow E[X = \sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} E[X_i] = \frac{m}{8}$$

Exercice :

1. Montrer que $(\text{MAX-3-DNF-SAT}, k)$ est **FPT**.
   - *Montrer que le nombre moyen de termes satisfaits dans une affectation aléatoire est au moins $\frac{m}{8}$ où m est le nombre total de termes*
     Soit X_i v.a. binaire telle que $X_i = 1$ ssi le terme λ_i est vrai
     $$P(X_i = 1) = \frac{1}{8} \Rightarrow E[X = \sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} E[X_i] = \frac{m}{8}$$
   - *Conclure que pour $m \geqslant 8k$, il existe une affectation qui satisfisait au moins k termes.*

Exercice :

1. Montrer que $(\text{MAX-3-DNF-SAT}, k)$ est **FPT**.
   - *Montrer que le nombre moyen de termes satisfaits dans une affectation aléatoire est au moins $\frac{m}{8}$ où m est le nombre total de termes*
     Soit X_i v.a. binaire telle que $X_i = 1$ ssi le terme λ_i est vrai
     $$P(X_i = 1) = \frac{1}{8} \Rightarrow E[X = \sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} E[X_i] = \frac{m}{8}$$
   - *Conclure que pour $m \geqslant 8k$, il existe une affectation qui satisfisait au moins k termes.*
     Si $m \geqslant 8k$ alors $E[X] \geqslant k$. Donc il existe affectation qui satisfait au moins k termes.

Exercice :

1. Montrer que (MAX-3-DNF-SAT, k) est **FPT**.
   - *Montrer que le nombre moyen de termes satisfaits dans une affectation aléatoire est au moins $\frac{m}{8}$ où m est le nombre total de termes*
     Soit X_i v.a. binaire telle que $X_i = 1$ ssi le terme λ_i est vrai
     $$P(X_i = 1) = \frac{1}{8} \Rightarrow E[X = \sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} E[X_i] = \frac{m}{8}$$
   - *Conclure que pour $m \geqslant 8k$, il existe une affectation qui satisfisait au moins k termes.*
     Si $m \geqslant 8k$ alors $E[X] \geqslant k$. Donc il existe affectation qui satisfait au moins k termes.
   - *On peut donc supposer que $m < 8k$.* Un arbre de recherche comme dans VERTEX COVER permet de répondre à la question en temps $8^{8k}.n^{O(1)}$.

Exercice :

1. Montrer que $(\text{MAX-3-DNF-SAT}, k)$ est **FPT**.
2. Montrer que sauf si **P** $=$ **NP** alors $(\text{EXACT-MAX-3-DNF-SAT}, k)$ n'est pas **FPT**

Exercice :

1. Montrer que $(\text{MAX-3-DNF-SAT}, k)$ est **FPT**.
2. Montrer que sauf si **P** $=$ **NP** alors $(\text{EXACT-MAX-3-DNF-SAT}, k)$ n'est pas **FPT**
   - *Montrer que c'est* **NP***-complet de décider s'il existe une affectation ne satisfaisant aucun terme*

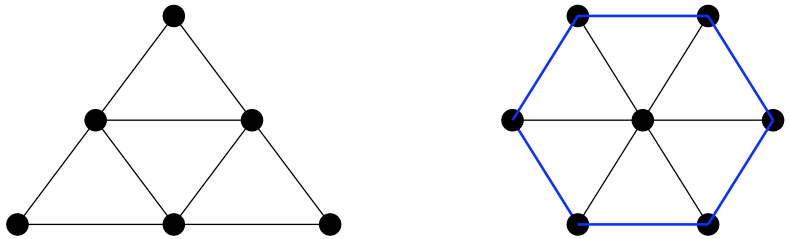
Exercice :

1. Montrer que $(\text{MAX-3-DNF-SAT}, k)$ est **FPT**.
2. Montrer que sauf si **P** $=$ **NP** alors $(\text{EXACT-MAX-3-DNF-SAT}, k)$ n'est pas **FPT**
   - *Montrer que c'est* **NP***-complet de décider s'il existe une affectation ne satisfaisant aucun terme*

     Aucun terme satisfait $\Leftrightarrow$ tout les termes satistaits dans $\neg\Phi$
     Réduction à 3-SAT

Minimum Fill-In

- **Données :** Un graphe $G = (V, E)$
- **Paramètre :** Un entier k (taille de la solution)
- **Question :** Existe-t'il $E' \subseteq V^2 \setminus E$ tel que $H = (V, E \cup E')$ ne contient pas de cycle sans corde de longueur $\leqslant 4$? (H est triangulé)

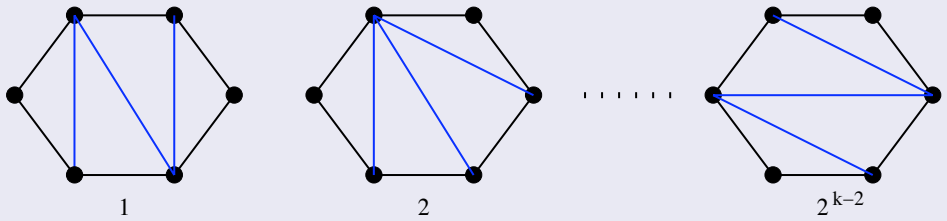


Minimum Fill-In

- **Données :** Un graphe $G = (V, E)$
- **Paramètre :** Un entier k (taille de la solution)
- **Question :** Existe-t'il $E' \subseteq V^2 \setminus E$ tel que $H = (V, E \cup E')$ ne contient pas de cycle sans corde de longueur $\leqslant 4$? (H est triangulé)

Lemme

Un cycle de longueur $k \geqslant 4$ admet 2^{k-2} complétions minimales.



Algorithme Minimum Fill-In

Minimum Fill-In(G, k)

- S'il existe un cycle C de longueur $t \geqslant 4$, alors
  - Si $t - 2 > k$, alors retourner FAUX
  - Sinon, pour chaque completion minimale E' de C, Minimum Fill-In$(G + E', k - t + 2)$
- Sinon retourner VRAI

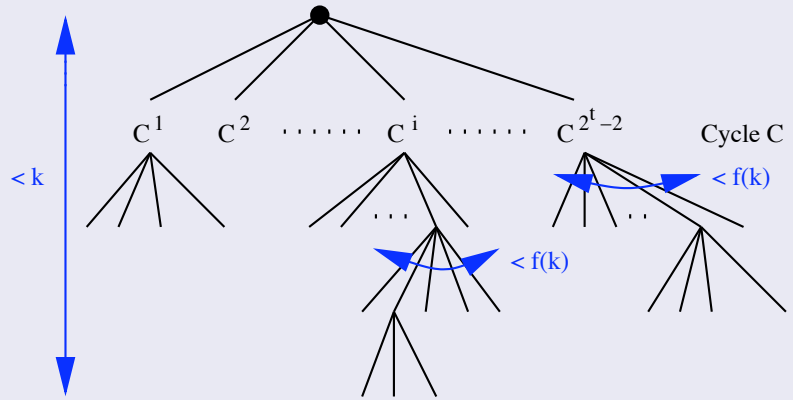
Algorithme Minimum Fill-In

Minimum Fill-In(G, k)

- S'il existe un cycle C de longueur $t \geqslant 4$, alors
  - Si $t - 2 > k$, alors retourner FAUX
  - Sinon, pour chaque completion minimale E' de C, Minimum Fill-In$(G + E', k - t + 2)$
- Sinon retourner VRAI

Analyse de complexité

1. A chaque appel, k décroit strictement
   $\Rightarrow$ profondeur au plus k



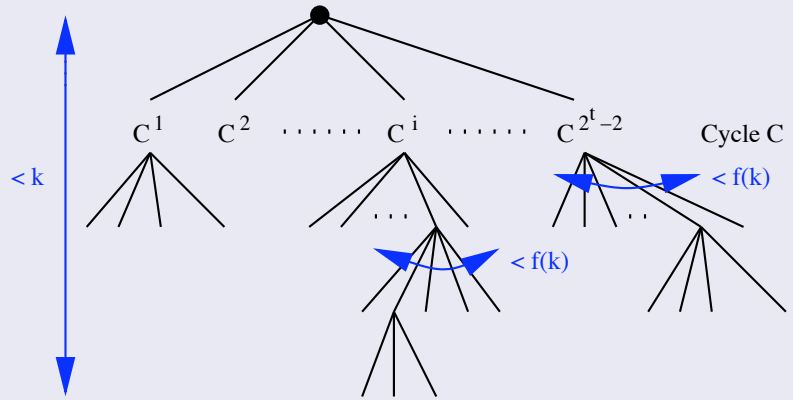
Algorithme Minimum Fill-In

Minimum Fill-In(G, k)

- S'il existe un cycle C de longueur $t \geqslant 4$, alors
  - Si $t - 2 > k$, alors retourner FAUX
  - Sinon, pour chaque completion minimale E' de C, Minimum Fill-In$(G + E', k - t + 2)$
- Sinon retourner VRAI

Analyse de complexité

1. A chaque appel, k décroit strictement
   $\Rightarrow$ profondeur au plus k
2. Le degré borné par $g(k) = 2^k$



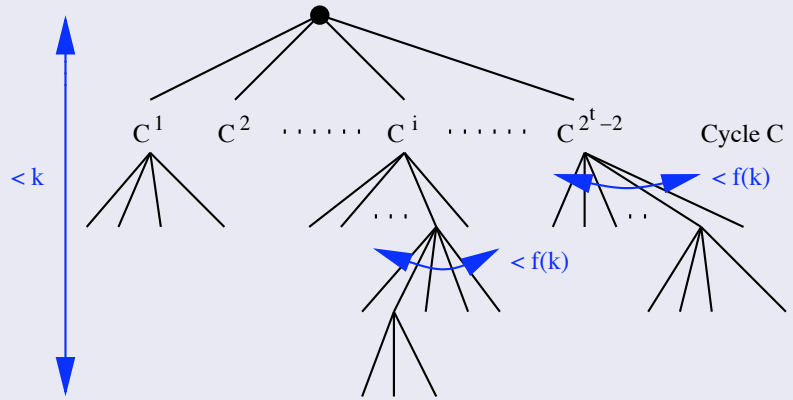
Algorithme Minimum Fill-In

Minimum Fill-In(G, k)

- S'il existe un cycle C de longueur $t \geqslant 4$, alors
  - Si $t - 2 > k$, alors retourner FAUX
  - Sinon, pour chaque completion minimale E' de C, Minimum Fill-In$(G + E', k - t + 2)$
- Sinon retourner VRAI

Analyse de complexité

1. A chaque appel, k décroit strictement
   $\Rightarrow$ profondeur au plus k
2. Le degré borné par $g(k) = 2^k$
3. $\Rightarrow$ taille de l'arbre au plus $f(k) = 2^{k^k}$



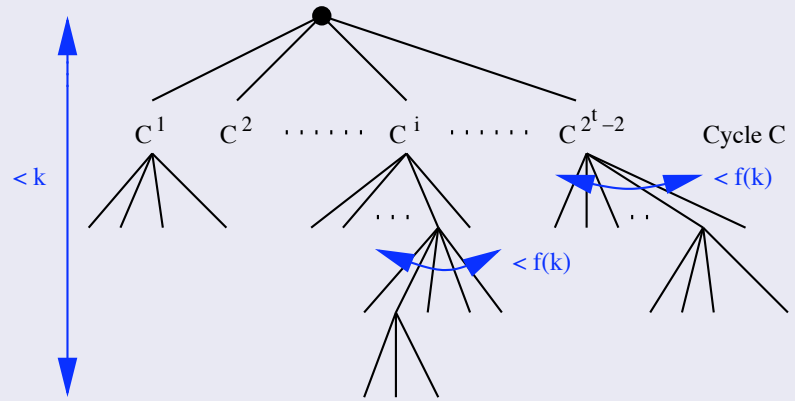
Algorithme Minimum Fill-In

Minimum Fill-In(G, k)

- S'il existe un cycle C de longueur $t \geqslant 4$, alors
  - Si $t - 2 > k$, alors retourner FAUX
  - Sinon, pour chaque completion minimale E' de C, Minimum Fill-In$(G + E', k - t + 2)$
- Sinon retourner VRAI

Analyse de complexité

1. Recherche d'un cycle sans corde: $O(n + m)$



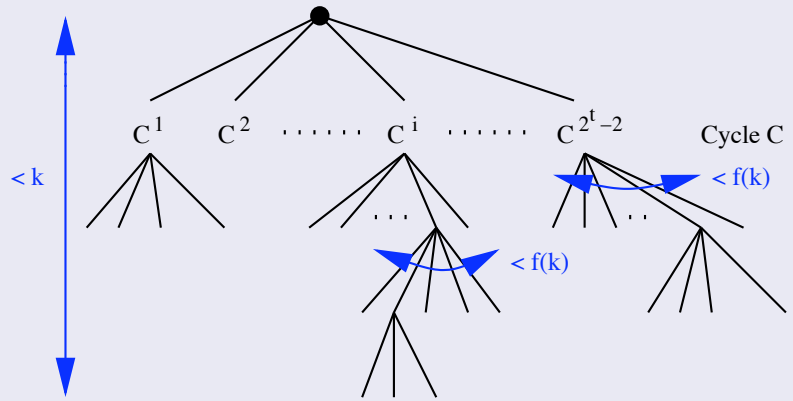
Algorithme Minimum Fill-In

Minimum Fill-In(G, k)

- S'il existe un cycle C de longueur $t \geqslant 4$, alors
  - Si $t - 2 > k$, alors retourner FAUX
  - Sinon, pour chaque completion minimale E' de C, Minimum Fill-In$(G + E', k - t + 2)$
- Sinon retourner VRAI

Analyse de complexité

1. Recherche d'un cycle sans corde: $O(n + m)$
2. Complexité totale : $O(f(k).(n + m))$



Définition

Soit $O = (sol, cost, goal)$ un pb d'optimisation **NP** sur Σ.
Soient une instance $x \in \Sigma^*$ et $y \in sol(x)$, le ratio d'approx. est:

$$r(x,y) = \max\{\frac{opt_O(x)}{cost(x,y)}, \frac{cost(x,y)}{opt_O(x)}\}$$

1. Soit $\epsilon > 0$ un réel. Un **algorithme d'ϵ-approximation polynomial** pour O est un algorithme polynomial qui, étant donné $x \in \Sigma^*$, calcule une solution $y \in sol(x)$ telle que $r(x,y) \leqslant (1+\epsilon)$.

Définition

Soit $O = (sol, cost, goal)$ un pb d'optimisation **NP** sur Σ.
Soient une instance $x \in \Sigma^*$ et $y \in sol(x)$, le ratio d'approx. est:

$$r(x,y) = \max\{\frac{opt_O(x)}{cost(x,y)}, \frac{cost(x,y)}{opt_O(x)}\}$$

1. Soit $\epsilon > 0$ un réel. Un **algorithme d'ϵ-approximation polynomial** pour O est un algorithme polynomial qui, étant donné $x \in \Sigma^*$, calcule une solution $y \in sol(x)$ telle que $r(x,y) \leqslant (1+\epsilon)$.
2. Un PTAS (**Polynomial Time Approximation Scheme**) pour O est un algorithme $\mathcal{A}$ qui étant données $(x,k) \in \Sigma \times \mathbb{N}$ tel que pour tout k fixé, $\mathcal{A}$ est une $\frac{1}{k}$-approximation polynomiale.

Définition

Soit $O = (sol, cost, goal)$ un pb d'optimisation **NP** sur Σ.
Soient une instance $x \in \Sigma^*$ et $y \in sol(x)$, le ratio d'approx. est:

$$r(x,y) = \max\{\frac{opt_O(x)}{cost(x,y)}, \frac{cost(x,y)}{opt_O(x)}\}$$

1. Soit $\epsilon > 0$ un réel. Un **algorithme d'ϵ-approximation polynomial** pour O est un algorithme polynomial qui, étant donné $x \in \Sigma^*$, calcule une solution $y \in sol(x)$ telle que $r(x,y) \leqslant (1+\epsilon)$.
2. Un PTAS (**Polynomial Time Approximation Scheme**) pour O est un algorithme $\mathcal{A}$ qui étant données $(x,k) \in \Sigma \times \mathbb{N}$ tel que pour tout k fixé, $\mathcal{A}$ est une $\frac{1}{k}$-approximation polynomiale.
3. Un PTAS est FPTAS (**Fully Polynomial Time Approximation Scheme**) si son temps d'exécution est polynomial en $|x| + k$.

Définition

Soit $O = (sol, cost, goal)$ un pb d'optimisation **NP** sur Σ. Un EPTAS (**Efficient Polynomial Time Approximation Scheme**) pour O est un PTAS $\mathcal{A}$ pour lequel il existe une fonction calculable $f : \mathbb{N} \to \mathbb{N}$ telle que le temps d'exécution de $\mathcal{A}$ sur une donnée $(x, k) \in \Sigma^* \times \mathbb{N}$ est au plus $f(k).n^{O(1)}$.

$$\text{FPTAS} \subseteq \text{EPTAS} \subseteq \text{PTAS}$$

Théorème

Si un problème $O = (sol, cost, min)$ d'optimisation **NP** admet un EPTAS $\mathcal{A}$, alors sa paramétrisation standard p-O est **FPT**.

Le problème paramétré p-O demande si $opt_O(x) \leqslant k$

Théorème

Si un problème $O = (sol, cost, min)$ d'optimisation **NP** admet un EPTAS $\mathcal{A}$, alors sa paramétrisation standard p-O est **FPT**.

Le problème paramétré p-O demande si $opt_O(x) \leqslant k$

Preuve

- Temps d'exécution de $\mathcal{A}$ sur la donnée (x, k) est $f(k).|x|^{O(1)}$

Théorème

Si un problème $O = (sol, cost, min)$ d'optimisation **NP** admet un EPTAS $\mathcal{A}$, alors sa paramétrisation standard p-O est **FPT**.

Le problème paramétré p-O demande si $opt_O(x) \leqslant k$

Preuve

- Temps d'exécution de $\mathcal{A}$ sur la donnée (x, k) est $f(k).|x|^{O(1)}$
- Soit l'algorithme **FPT** $\mathcal{A}'$
  1. calcule $y = \mathcal{A}(x, k+1)$;
  2. retourne vrai si $cost(x, y) \leqslant k$ et faux sinon

Théorème

Si un problème $O = (sol, cost, min)$ d'optimisation **NP** admet un EPTAS $\mathcal{A}$, alors sa paramétrisation standard p-O est **FPT**.

Le problème paramétré p-O demande si $opt_O(x) \leqslant k$

Preuve

- Temps d'exécution de $\mathcal{A}$ sur la donnée (x, k) est $f(k).|x|^{O(1)}$
- Soit l'algorithme **FPT** $\mathcal{A}'$
  1. calcule $y = \mathcal{A}(x, k+1)$;
  2. retourne vrai si $cost(x, y) \leqslant k$ et faux sinon
- Validité de l'algorithme:
  1. Si $cost(x, y) \leqslant k$, alors $opt_O(x) \leqslant k$

Théorème

Si un problème $O = (sol, cost, min)$ d'optimisation **NP** admet un EPTAS $\mathcal{A}$, alors sa paramétrisation standard p-O est **FPT**.

Le problème paramétré p-O demande si $opt_O(x) \leqslant k$

Preuve

- Temps d'exécution de $\mathcal{A}$ sur la donnée (x, k) est $f(k).|x|^{O(1)}$
- Soit l'algorithme **FPT** $\mathcal{A}'$
  1. calcule $y = \mathcal{A}(x, k+1)$;
  2. retourne vrai si $cost(x, y) \leqslant k$ et faux sinon
- Validité de l'algorithme:
  1. Si $cost(x, y) \leqslant k$, alors $opt_O(x) \leqslant k$
  2. Sinon

$$opt_O(x) = \frac{cost(x,y)}{r(x,y)} \geqslant \frac{cost(x,y)}{1+\frac{1}{k+1}} \geqslant \frac{k+1}{1+\frac{1}{k+1}} = \frac{(k+1)^2}{k+2} > k$$

Théorème

Si un problème $O = (sol, cost, min)$ d'optimisation **NP** admet un EPTAS $\mathcal{A}$, alors sa paramétrisation standard p-O est **FPT**.

Le problème paramétré p-O demande si $opt_O(x) \leqslant k$

Remarque : La réciproque n'est pas vraie

Il est connu que MIN-VERTEX COVER ne possède pas de PTAS (sauf si **P** $=$ **NP**) et pourtant k-VERTEX COVER est un problème **FPT**.