Bornes inférieures algorithmiques et limites de la kernelisation

Christophe PAUL
(CNRS - LIRMM)

15ème Journées Combinatoire du Littoral Méditerranéen
11 mars 2015
JCALM – Sophia Antipolis

PART 1. Bornes inférieures passées sur (S)-ETH

PART 2. Bornes inférieures pour la kernelization

Complexités rencontrées fréquemment

Algorithmes FPT :

- $2^{O(k)} \cdot n^{O(1)}$ $\rightarrow$ Vertex Cover
- $2^{O(k \cdot \log(k))} \cdot n^{O(1)}$ $\rightarrow$ Hamiltonian Cycle
- $2^{2^{2^{O(k)}}} \cdot n^{O(1)}$ paramétré par $k = \mathsf{tw}(G)$.
  $\rightarrow$ par le théorème de Courcelle

Complexités rencontrées fréquemment

Algorithmes FPT :

- $2^{O(k)} \cdot n^{O(1)}$ → Vertex Cover
- $2^{O(k \cdot \log(k))} \cdot n^{O(1)}$ → Hamiltonian Cycle paramétré par $k = \mathrm{tw}(G)$.
- $2^{2^{2^{O(k)}}} \cdot n^{O(1)}$ → par le théorème de Courcelle

Algorithmes XP (problèmes W[1]-Hard) :

- $n^{O(k)}$ → Independent Set
- $n^{O(\log \log k)}$ ...

Complexités rencontrées fréquemment

Algorithmes FPT :

- $2^{O(k)} \cdot n^{O(1)}$ → Vertex Cover
- $2^{O(k \cdot \log(k))} \cdot n^{O(1)}$ → Hamiltonian Cycle paramétré par $k = \mathrm{tw}(G)$.
- $2^{2^{2^{O(k)}}} \cdot n^{O(1)}$ → par le théorème de Courcelle

Algorithmes XP (problèmes W[1]-Hard) :

- $n^{O(k)}$ → Independent Set
- $n^{O(\log \log k)}$ ...

Algorithmes exponentiels (problèmes NP) :

- $2^{O(n)}$ → Independent Set
- $2^{O(\sqrt{n})}$ → Planar Independent Set

Complexités rencontrées fréquemment

Algorithmes FPT :

- $2^{O(k)} \cdot n^{O(1)}$ → Vertex Cover
- $2^{O(k \cdot \log(k))} \cdot n^{O(1)}$ → Hamiltonian Cycle paramétré par $k = \mathrm{tw}(G)$.
- $2^{2^{2^{O(k)}}} \cdot n^{O(1)}$ → par le théorème de Courcelle

Algorithmes XP (problèmes W[1]-Hard) :

- $n^{O(k)}$ → Independent Set
- $n^{O(\log \log k)}$ ...

Algorithmes exponentiels (problèmes NP) :

- $2^{O(n)}$ → Independent Set
- $2^{O(\sqrt{n})}$ → Planar Independent Set

Question :

Ces complexités sont optimales sous quelle hypothèse ?

Hypothèse $P \neq NP$

Question : La conjecture $P \neq NP$ implique-t'elle que tout problème $\Pi \in NP \setminus P$ nécessite un temps exponentiel ?

Hypothèse $P \neq NP$

Question : La conjecture $P \neq NP$ implique-t'elle que tout problème $\Pi \in NP \setminus P$ nécessite un temps exponentiel ?

- NON : $\Pi \in NP \setminus P$ peut avoir une complexité de la forme $O(n^{\log n})$
- La théorie de la NP-Complétude ne permet pas l'étude de bornes inférieures fines sur les complexités algorithmiques.

Hypothèse $P \neq NP$

Question : La conjecture $P \neq NP$ implique-t'elle que tout problème $\Pi \in NP \setminus P$ nécessite un temps exponentiel ?

- NON : $\Pi \in NP \setminus P$ peut avoir une complexité de la forme $O(n^{\log n})$
- La théorie de la NP-Complétude ne permet pas l'étude de bornes inférieures fines sur les complexités algorithmiques.

Objectifs :

- Expliquer les différences (ou équivalences) de difficultés entre problèmes de NP
- Identifier des barrières de complexité

Exponential Time Hypothesis – ETH

Observation : CNF-SAT peut être résolu en temps $O^*(2^N)$
(où N est le nombre de variables)

$\rightsquigarrow$ Peut-on résoudre CNF-SAT en temps $O^*(1,99^N)$?

Exponential Time Hypothesis – ETH

Observation : CNF-SAT peut être résolu en temps $O^*(2^N)$
(où N est le nombre de variables)

$\rightsquigarrow$ Peut-on résoudre CNF-SAT en temps $O^*(1,99^N)$?

[Impagliazzo, Paturi, Zane]

CNF-SAT ne peut être résolu en temps $O^*(2^{o(N)})$

$\rightsquigarrow$ Quelles sont les conséquences d'une telle hypothèse ?

Exponential Time Hypothesis – ETH

Observation : CNF-SAT peut être résolu en temps $O^*(2^N)$
(où N est le nombre de variables)

$\rightsquigarrow$ Peut-on résoudre CNF-SAT en temps $O^*(1,99^N)$?

[Impagliazzo, Paturi, Zane]

CNF-SAT ne peut être résolu en temps $O^*(2^{o(N)})$

$\rightsquigarrow$ Quelles sont les conséquences d'une telle hypothèse ?

- algorithmes exponentiels exacts, XP, FPT
- algorithmes d'approximation. . .
- algorithmes polynomiaux

(Strong) Exponential Time Hypothesis – (S)ETH

Pour tout entier $q \geqslant 3$, nous notons
$\delta_q = \inf\{c \mid \exists$ un algorithme de complexité $O^*(2^{cN})$ pour q-SAT$\}$

Conjectures

ETH $\boxed{\delta_3 > 0}$ SETH $\boxed{\lim_{q\to\infty} \delta_q = 1}$

(Strong) Exponential Time Hypothesis – (S)ETH

Pour tout entier $q \geqslant 3$, nous notons
$\delta_q = \inf\{c \mid \exists$ un algorithme de complexité $O^*(2^{cN})$ pour q-SAT$\}$

Conjectures

ETH $\boxed{\delta_3 > 0}$ SETH $\boxed{\lim_{q\to\infty} \delta_q = 1}$

- ETH $\Rightarrow$ 3-SAT ne se résoud pas en temps $2^{o(n)}$
- SETH $\Rightarrow$ CNF-SAT ne se résoud pas en temps $O^*((2-\epsilon)^n)$

(Strong) Exponential Time Hypothesis – (S)ETH

Pour tout entier $q \geqslant 3$, nous notons
$\delta_q = \inf\{c \mid \exists$ un algorithme de complexité $O^*(2^{cN})$ pour q-SAT$\}$

Conjectures

ETH $\boxed{\delta_3 > 0}$ SETH $\boxed{\lim_{q\to\infty} \delta_q = 1}$

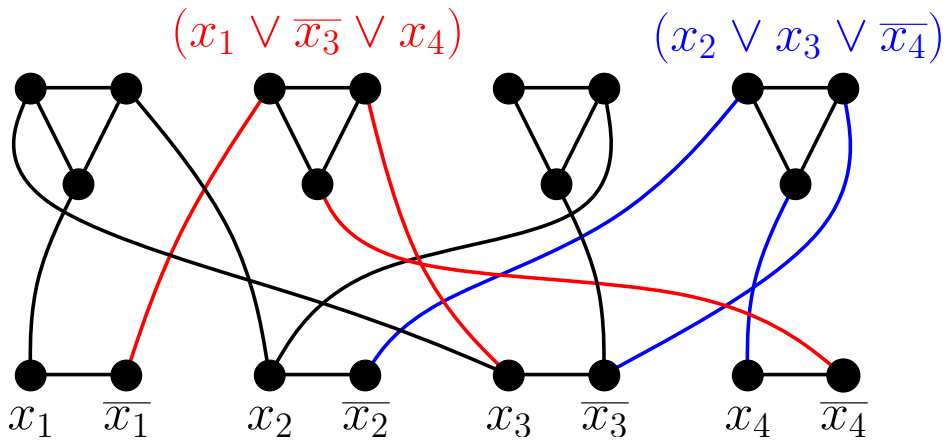
- ETH $\Rightarrow$ 3-SAT ne se résoud pas en temps $2^{o(n)}$
- SETH $\Rightarrow$ CNF-SAT ne se résoud pas en temps $O^*((2-\epsilon)^n)$

Observation : Pour une formule CNF-SAT sur N variables et M clauses, nous pouvons avoir : $N \ll M$

Remarque : ETH étudie la complexité (de CNF-SAT) en fonction de l'espace de recherche, NP en fonction de la taille de la donnée.

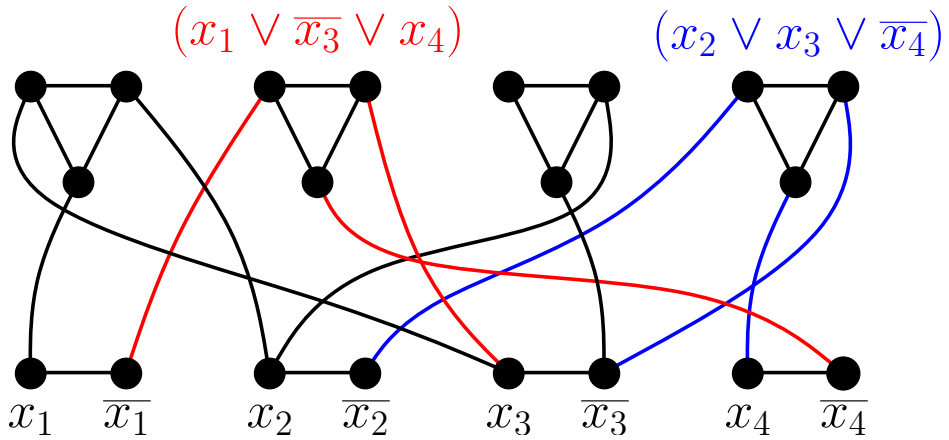
ETH et réductions depuis 3-SAT

Réduction $\mathcal{R}$ d'une instance φ de 3-SAT à N variables et M clauses en une instance G de Vertex Cover de taille $O(N+M)$.



ETH et réductions depuis 3-SAT

Réduction $\mathcal{R}$ d'une instance φ de 3-SAT à N variables et M clauses en une instance G de Vertex Cover de taille $O(N+M)$.



Observation : $\mathcal{R}$ est une réduction linéaire mais $\boxed{|G| = O(N^3)}$

ETH $\Rightarrow$ Vertex Cover ne se résoud pas en temps $2^{o(N^{1/3})}$

Meilleure borne inférieure possible ?

(Strong) Exponential Time Hypothesis – (S)ETH

Observation : $\delta_3 \leqslant \delta_4 \leqslant \cdots \leqslant \delta_q$

- Some values : $\delta_3 \leqslant 0.388, \quad \delta_4 \leqslant 0.555, \quad \delta_q \leqslant 1 - \Theta(\frac{1}{q})$

Theorem [Impagliazzo, Paturi, Zane] :

$$\boxed{k\text{-SAT en temps } 2^{o(M)} \Rightarrow k\text{-SAT en temps } 2^{o(N)}}$$

(Strong) Exponential Time Hypothesis – (S)ETH

Observation : $\delta_3 \leqslant \delta_4 \leqslant \cdots \leqslant \delta_q$

- Some values : $\delta_3 \leqslant 0.388, \quad \delta_4 \leqslant 0.555, \quad \delta_q \leqslant 1 - \Theta(\frac{1}{q})$

Theorem [Impagliazzo, Paturi, Zane] :

$$\boxed{k\text{-SAT en temps } 2^{o(M)} \Rightarrow k\text{-SAT en temps } 2^{o(N)}}$$

- une instance $\varphi^{(4)}$ de 4-SAT se réduit à une instance $\varphi^{(3)}$ 3-SAT instance avec $O(M)$ variables
- un algorithme de complexité $2^{o(N)}$ pour 3-SAT implique un algorithme $2^{o(N)}$ pour 4-SAT

(Strong) Exponential Time Hypothesis – (S)ETH

Observation : $\delta_3 \leqslant \delta_4 \leqslant \cdots \leqslant \delta_q$

- Some values : $\delta_3 \leqslant 0.388, \quad \delta_4 \leqslant 0.555, \quad \delta_q \leqslant 1 - \Theta(\frac{1}{q})$

Theorem [Impagliazzo, Paturi, Zane] :

k-SAT en temps $2^{o(M)} \Rightarrow$ k-SAT en temps $2^{o(N)}$

- une instance $\varphi^{(4)}$ de 4-SAT se réduit à une instance $\varphi^{(3)}$ 3-SAT instance avec $O(M)$ variables
- un algorithme de complexité $2^{o(N)}$ pour 3-SAT implique un algorithme $2^{o(N)}$ pour 4-SAT

ETH $\Rightarrow$ Vertex Cover ne se résoud pas en temps $2^{o(n)}$

Lemme de sparsification

Lemme [Impagliazzo, Paturi, Zane] :
$\forall\ \epsilon > 0$ et $q \geqslant 0$, il existe une constante $C(\epsilon, q)$ telle que :

$$\forall q\text{-CNF } \varphi \text{ à } N \text{ variables} \longrightarrow \varphi_\epsilon = \bigvee_{i=1}^{t} \varphi_i \text{ telle que}$$

- φ est satisfiable ssi φ_ϵ est satisfiable
- $t \leqslant 2^{\epsilon \cdot N}$
- $\forall i \in [t], \varphi_i$ est une q-CNF sur le même ensemble de variables et au plus $C(\epsilon, q) \cdot N$ clauses.

La complexité de l'algorithme $\mathcal{A}_s(\epsilon, \varphi)$ calculant φ_ϵ est $O^*(2^{\epsilon \cdot N})$.

Conséquence du lemme de sparsification (1)

Théorème : Si ETH est vérifiée alors $\exists c > 0$ tel que 3-SAT ne peut pas être résolu en temps $O^*\left(2^{c(N+M)}\right)$.

($\Rightarrow$ 3-SAT ne peut pas être résolu en temps $O^*\left(2^{o(N+M)}\right)$)

Conséquence du lemme de sparsification (1)

Théorème : Si ETH est vérifiée alors $\exists c > 0$ tel que 3-SAT ne peut pas être résolu en temps $O^*\left(2^{c(N+M)}\right)$.

($\Rightarrow$ 3-SAT ne peut pas être résolu en temps $O^*\left(2^{o(N+M)}\right)$)

Preuve (par contradiction) :

Soit $\mathcal{A}_c$ un algorithme résolvant 3-SAT en temps $O^*\left(2^{c(N+M)}\right)$

Soit $\epsilon = d/2$ et $C = C(\epsilon, 3)$

Considérons l'algorithme $\mathcal{B}_d(\varphi)$ suivant

1. $\varphi_\epsilon = \bigvee_{i=1}^{t} \varphi_i = \mathcal{A}_s(\epsilon, \varphi)$
2. $\forall i$, appliquer $\mathcal{A}_{c'}(\varphi_i)$ avec $c' = \frac{d}{2\cdot(C+1)}$

Conséquence du lemme de sparsification (1)

Théorème : Si ETH est vérifiée alors $\exists c > 0$ tel que 3-SAT ne peut pas être résolu en temps $O^*\left(2^{c(N+M)}\right)$.

($\Rightarrow$ 3-SAT ne peut pas être résolu en temps $O^*\left(2^{o(N+M)}\right)$)

Preuve (par contradiction) :
Soit $\mathcal{A}_c$ un algorithme résolvant 3-SAT en temps $O^*\left(2^{c(N+M)}\right)$
Soit $\epsilon = d/2$ et $C = C(\epsilon, 3)$
Considérons l'algorithme $\mathcal{B}_d(\varphi)$ suivant

1. $\varphi_\epsilon = \bigvee_{i=1}^{t} \varphi_i = \mathcal{A}_s(\epsilon, \varphi)$
2. $\forall i$, appliquer $\mathcal{A}_{c'}(\varphi_i)$ avec $c' = \frac{d}{2\cdot(C+1)}$

- Observation 1 : φ est satisfiable ssi $\exists i$, φ_i satisfiable.

Conséquence du lemme de sparsification (1)

Théorème : Si ETH est vérifiée alors $\exists c > 0$ tel que 3-SAT ne peut pas être résolu en temps $O^*\left(2^{c(N+M)}\right)$.

($\Rightarrow$ 3-SAT ne peut pas être résolu en temps $O^*\left(2^{o(N+M)}\right)$)

Preuve (par contradiction) :
Soit $\mathcal{A}_c$ un algorithme résolvant 3-SAT en temps $O^*\left(2^{c(N+M)}\right)$
Soit $\epsilon = d/2$ et $C = C(\epsilon, 3)$
Considérons l'algorithme $\mathcal{B}_d(\varphi)$ suivant

1. $\varphi_\epsilon = \bigvee_{i=1}^{t} \varphi_i = \mathcal{A}_s(\epsilon, \varphi)$ $\rightsquigarrow O^*\left(2^{\frac{d \cdot N}{2}}\right)$
2. $\forall i$, appliquer $\mathcal{A}_{c'}(\varphi_i)$ avec $c' = \frac{d}{2 \cdot (C+1)}$ $\rightsquigarrow$ $O^*\left(2^{\frac{d}{2 \cdot (C+1)} \cdot (C+1) \cdot N}\right)$

- Observation 1 : φ est satisfiable ssi $\exists i$, φ_i satisfiable.
- Observation 2 : La complexité de $\mathcal{B}_d$ est $\rightsquigarrow O^*(2^{d \cdot N})$

Conséquence du lemme de sparsification (2)

Théorème : $\boxed{\text{SETH} \Rightarrow \text{ETH}}$

Preuve (par contradiction) : Supposons que $\delta_3 = 0$.

$\Rightarrow$ $\forall c > 0$, $\exists$ $\mathcal{A}_c$ un algorithme pour 3-SAT en temps $O^*(2^{c \cdot N})$

Conséquence du lemme de sparsification (2)

Théorème : $\boxed{\text{SETH} \Rightarrow \text{ETH}}$

Preuve (par contradiction) : Supposons que $\delta_3 = 0$.

$\Rightarrow$ $\forall c > 0$, $\exists\ \mathcal{A}_c$ un algorithme pour 3-SAT en temps $O^*(2^{c \cdot N})$

- Soit φ une formule q-CNF-SAT
  1. $\mathcal{A}_s(\epsilon, \varphi) = \bigvee_{i=1}^{t} \varphi_i$ pour $\epsilon > 0$
  2. $\forall i$, transformer φ_i en une formule $\tilde{\varphi}_i$ 3-SAT ayant au plus $(1 + q \cdot C(\epsilon, q)) \cdot N$ variables
  3. $\forall i$, appliquer $\mathcal{A}_\delta(\tilde{\varphi}_i)$ pour $0 < \delta < 1$

Conséquence du lemme de sparsification (2)

Théorème : $\boxed{\text{SETH} \Rightarrow \text{ETH}}$

Preuve (par contradiction) : Supposons que $\delta_3 = 0$.

$\Rightarrow$ $\forall c > 0$, $\exists\ \mathcal{A}_c$ un algorithme pour 3-SAT en temps $O^*(2^{c \cdot N})$

- Soit φ une formule q-CNF-SAT
  1. $\mathcal{A}_s(\epsilon, \varphi) = \bigvee_{i=1}^{t} \varphi_i$ pour $\epsilon > 0$
  2. $\forall i$, transformer φ_i en une formule $\tilde{\varphi}_i$ 3-SAT ayant au plus $(1 + q \cdot C(\epsilon, q)) \cdot N$ variables
  3. $\forall i$, appliquer $\mathcal{A}_\delta(\tilde{\varphi}_i)$ pour $0 < \delta < 1$

- Observation 1 : φ est satisfiable ssi $\exists i$, $\tilde{\varphi}_i$ satisfiable.

Conséquence du lemme de sparsification (2)

Théorème : $\boxed{\text{SETH} \Rightarrow \text{ETH}}$

Preuve (par contradiction) : Supposons que $\delta_3 = 0$.

$\Rightarrow$ $\forall c > 0$, $\exists\ \mathcal{A}_c$ un algorithme pour 3-SAT en temps $O^*(2^{c \cdot N})$

- Soit φ une formule q-CNF-SAT
  1. $\mathcal{A}_s(\epsilon, \varphi) = \bigvee_{i=1}^{t} \varphi_i$ pour $\epsilon > 0$
  2. $\forall i$, transformer φ_i en une formule $\tilde{\varphi}_i$ 3-SAT ayant au plus $(1 + q \cdot C(\epsilon, q)) \cdot N$ variables
  3. $\forall i$, appliquer $\mathcal{A}_\delta(\tilde{\varphi}_i)$ pour $0 < \delta < 1$
- Observation 1 : φ est satisfiable ssi $\exists i$, $\tilde{\varphi}_i$ satisfiable.
- Observation 2 : La complexité de l'algorithme décrit est

$$O^*\left(2^{\beta \cdot N}\right) \text{ avec } \beta = \epsilon + \delta \cdot (1 + q \cdot C(\epsilon, q))$$

Conséquence du lemme de sparsification (2)

Théorème : $\boxed{\text{SETH} \Rightarrow \text{ETH}}$

Preuve (par contradiction) : Supposons que $\delta_3 = 0$.

$\Rightarrow$ $\forall c > 0$, $\exists\ \mathcal{A}_c$ un algorithme pour 3-SAT en temps $O^*(2^{c \cdot N})$

- Soit φ une formule q-CNF-SAT
  1. $\mathcal{A}_s(\epsilon, \varphi) = \bigvee_{i=1}^{t} \varphi_i$ pour $\epsilon > 0$
  2. $\forall i$, transformer φ_i en une formule $\tilde{\varphi}_i$ 3-SAT ayant au plus $(1 + q \cdot C(\epsilon, q)) \cdot N$ variables
  3. $\forall i$, appliquer $\mathcal{A}_\delta(\tilde{\varphi}_i)$ pour $0 < \delta < 1$
- Observation 1 : φ est satisfiable ssi $\exists i$, $\tilde{\varphi}_i$ satisfiable.
- Observation 2 : La complexité de l'algorithme décrit est

$$O^*\left(2^{\beta \cdot N}\right) \text{ avec } \beta = \epsilon + \delta \cdot (1 + q \cdot C(\epsilon, q))$$

- On peut choisir ϵ et δ aussi proche de 0 que possible $\Rightarrow \delta_q = 0$

Conséquence du lemme de sparsification

Observation : Pour exclure un algorithme en $O^*\left(2^{o(f(|x|))}\right)$ pour un problème A, il suffit de fournir une réduction polynomiale

$\mathcal{R}$: 3-SAT $\longrightarrow$ A telle que
$|\mathcal{R}(\varphi)| = O(g(|x|))$ avec g la fonction inverse de f

Conséquence du lemme de sparsification

Observation : Pour exclure un algorithme en $O^*\left(2^{o(f(|x|))}\right)$ pour un problème A, il suffit de fournir une réduction polynomiale

$\mathcal{R}$: 3-SAT $\rightarrow$ A telle que
$|\mathcal{R}(\varphi)| = O(g(|x|))$ avec g la fonction inverse de f

Théorème : Les problèmes suivant admettent une réduction linéaire depuis 3-SAT

- Vertex Cover, Dominating Set, Feedback Vertex Set, 3-Coloring, Hamiltonian Cycle.

ETH $\Rightarrow$ ces problèmes n'ont pas d'algorithme en $O^*\left(2^{o(n+m)}\right)$

Conséquence du lemme de sparsification

Observation : Pour exclure un algorithme en $O^*\left(2^{o(f(|x|))}\right)$ pour un problème A, il suffit de fournir une réduction polynomiale

$\mathcal{R}$: 3-SAT $\rightarrow$ A telle que
$|\mathcal{R}(\varphi)| = O(g(|x|))$ avec g la fonction inverse de f

Théorème : Les problèmes suivant admettent une réduction polynomiale $\mathcal{R}$ depuis 3-SAT

- Planar Vertex Cover, Planar Dominating Set, Planar Feedback Vertex Set

telle que $\mathcal{R}(\varphi)$ est un graphe de taille $O((N+M)^2)$.
ETH $\Rightarrow$ ces problèmes n'ont pas d'algorithme en $O^*\left(2^{o(\sqrt{n})}\right)$

ETH et problèmes W[1]-Difficiles

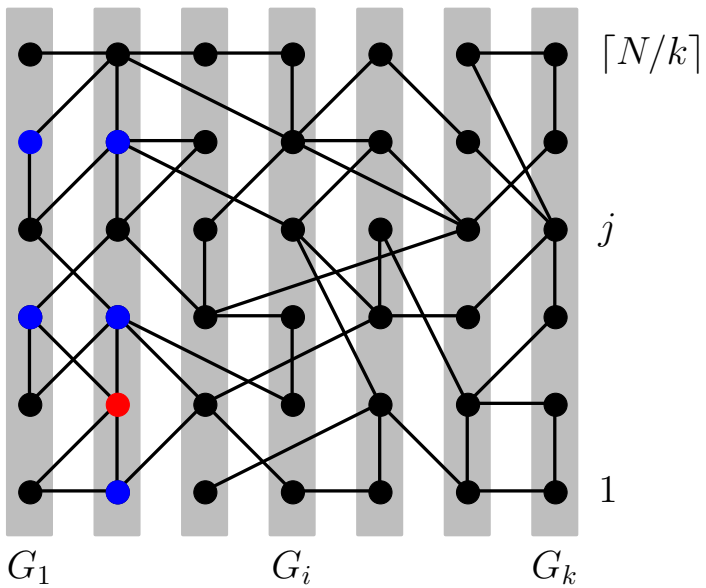
Théorème : S'il existe un algorithme de complexité $f(k) \cdot n^{o(k)}$ pour Clique ou Independent Set, alors ETH n'est pas valide.

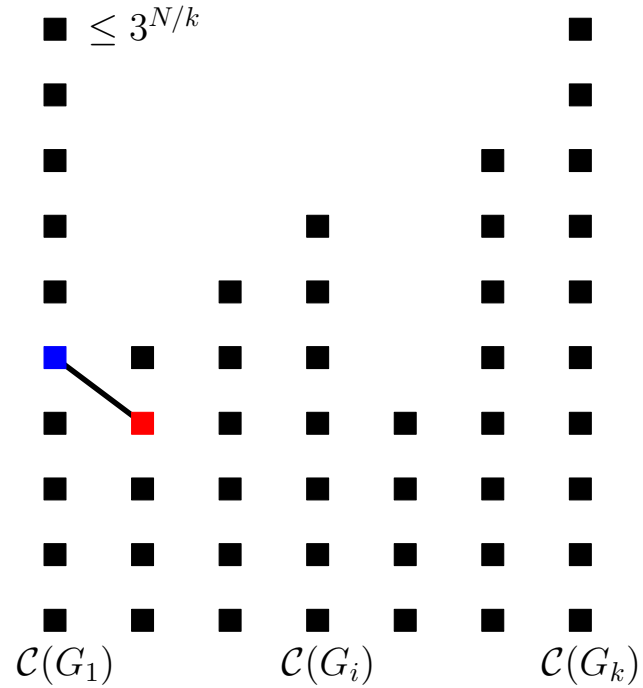
Corollaire : $\boxed{\text{ETH} \Rightarrow \text{FPT} \neq \text{W[1]-Difficile}}$

ETH et problèmes W[1]-Difficiles

Théorème : S'il existe un algorithme de complexité $f(k) \cdot n^{o(k)}$ pour Clique ou Independent Set, alors ETH n'est pas valide.

Soit $\mathcal{R}$ la réduction linéaire de 3-SAT vers 3-Coloration
Soit $\mathcal{A}$ un algorithme en $2^k \cdot n^{o(k)}$ pour Clique.

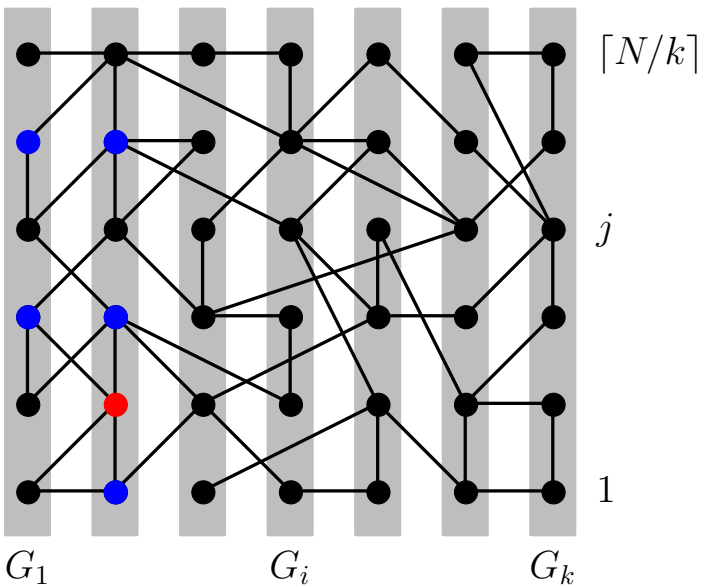




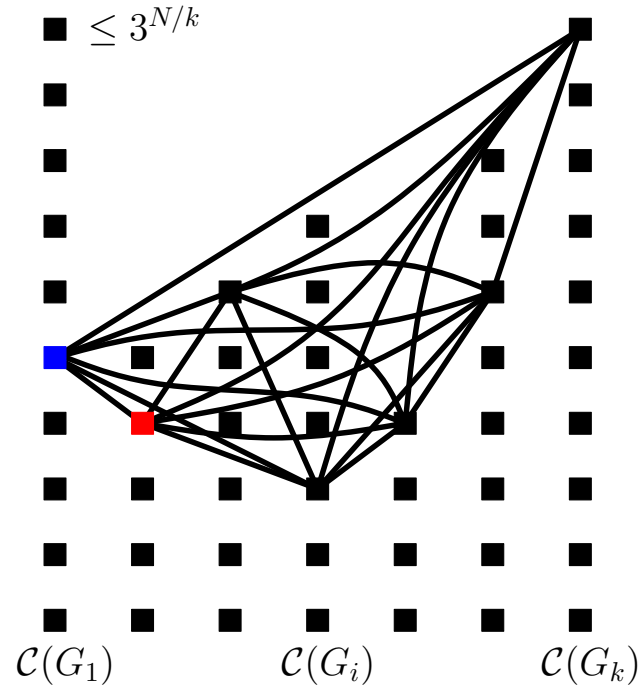
ETH et problèmes W[1]-Difficiles

Théorème : S'il existe un algorithme de complexité $f(k) \cdot n^{o(k)}$ pour Clique ou Independent Set, alors ETH n'est pas valide.

Soit $\mathcal{R}$ la réduction linéaire de 3-SAT vers 3-Coloration
Soit $\mathcal{A}$ un algorithme en $2^k \cdot n^{o(k)}$ pour Clique.



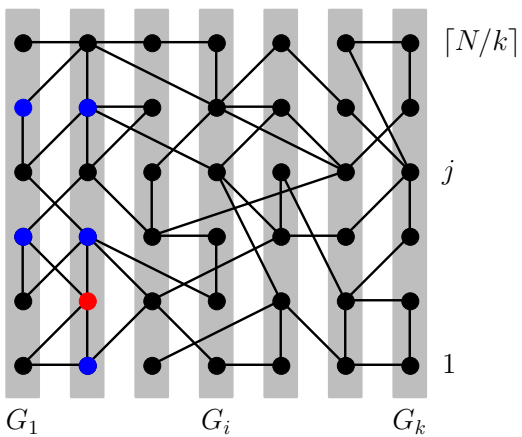
$$\chi(G) = 3 \Leftrightarrow \omega(\mathcal{C}(G)) = k$$



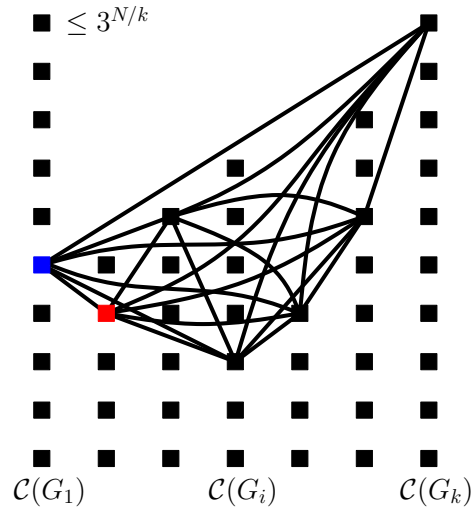
ETH et problèmes W[1]-Difficiles

Théorème : S'il existe un algorithme de complexité $f(k) \cdot n^{o(k)}$ pour Clique ou Independent Set, alors ETH n'est pas valide.

Soit $\mathcal{R}$ la réduction linéaire de 3-SAT vers 3-Coloration
Soit $\mathcal{A}$ un algorithme en $2^k \cdot n^{o(k)}$ pour Clique.



$\chi(G) = 3 \Leftrightarrow \omega(\mathcal{C}(G)) = k$



complexité de $\mathcal{A} \circ \mathcal{R}$: $n = |V(\mathcal{C}(G))| = k \cdot 3^{\lceil N/k \rceil}$ et $k = \log N$

$$\rightsquigarrow \boxed{2^k \cdot n^{o(k)} = N \cdot (\log N)^{o(\log N)} \cdot 3^{N \cdot o(\log N)/\log N} = O^*\left(2^{o(N)}\right)}$$

ETH et problèmes FPT

Observation : Pour exclure un algorithme en $O^*\left(2^{o(f(k)}\right)$ pour un problème A paramétré par k, il suffit de fournir une réduction paramétrée

$\mathcal{R}$: 3-SAT $\longrightarrow$ A telle que
$k(\mathcal{R}(\varphi)) = O(g(n+m))$ avec g la fonction inverse de f

- seule la dépendance en k doit être contrôlée, la taille de l'instance réduite peut-être polynomiale.

ETH et problèmes FPT

Observation : Pour exclure un algorithme en $O^*\left(2^{o(f(k)}\right)$ pour un problème A paramétré par k, il suffit de fournir une réduction paramétrée

$\mathcal{R}$: 3-SAT $\rightarrow$ A telle que
$k(\mathcal{R}(\varphi)) = O(g(n+m))$ avec g la fonction inverse de f

- seule la dépendance en k doit être contrôlée, la taille de l'instance réduite peut-être polynomiale.

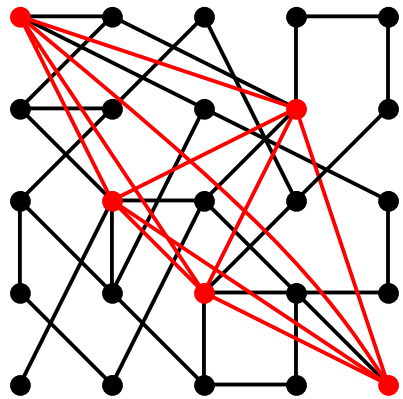
Théorème : Les problèmes paramétrés suivant admettent une réduction paramétrée linéaire depuis 3-SAT

- Vertex Cover, Feedback Vertex Set

ETH $\Rightarrow$ ces problèmes n'ont pas d'algorithme en $O^*\left(2^{o(k)}\right)$

Problèmes FPT super-exponentiels

Théorème : Sauf si ETH est fausse, le problème paramétré $(k \times k)$-Clique n'admet pas d'algorithme en $O^*\left(2^{o(k \log k)}\right)$.

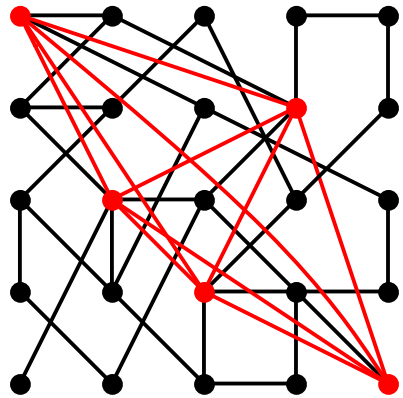


$\rightsquigarrow$ La preuve est une adaptation de celle vue précédemment

$\rightsquigarrow$ $(k \times k)$-Clique admet un algorithme en $O^*(k^k)$

Problèmes FPT super-exponentiels

Théorème : Sauf si ETH est fausse, le problème paramétré $(k \times k)$-Clique n'admet pas d'algorithme en $O^*\left(2^{o(k \log k)}\right)$.



$\rightsquigarrow$ La preuve est une adaptation de celle vue précédemment

$\rightsquigarrow$ $(k \times k)$-Clique admet un algorithme en $O^*(k^k)$

Problèmes de bases pour les réductions

- $(k \times k)$-Permutation Clique
- $(k \times k)$-(Permutation) Hitting Set
- $(k \times k)$-(Permutation) Hitting Set with Thin Sets

Bornes inférieures FPT basées sur SETH

Théorème : Independent Set paramétré par tw est résolu en $O^*(2^{\mathsf{tw}(G)})$ par programmation dynamique.

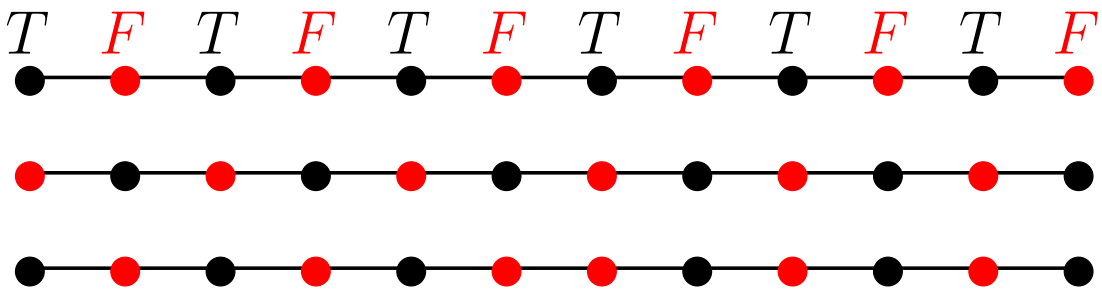
$\rightsquigarrow$ Peut-on obtenir un algorithme en $O^*\left((1.9999...)^{\mathsf{tw}(G)}\right)$?

Bornes inférieures FPT basées sur SETH

Théorème : Independent Set paramétré par tw est résolu en $O^*(2^{\mathsf{tw}(G)})$ par programmation dynamique.

$\rightsquigarrow$ Peut-on obtenir un algorithme en $O^*\left((1.9999\ldots)^{\mathsf{tw}(G)}\right)$?

- Independent Set sur les chemins pairs



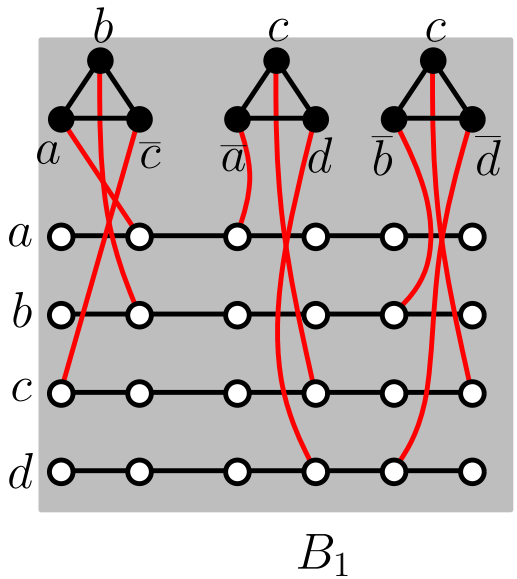
Bornes inférieures FPT basées sur SETH

Théorème : Independent Set paramétré par tw est résolu en $O^*(2^{\mathsf{tw}(G)})$ par programmation dynamique.

$\rightsquigarrow$ Peut-on obtenir un algorithme en $O^*\left((1.9999\ldots)^{\mathsf{tw}(G)}\right)$?

- k-SAT $\longrightarrow$ Independent Set paramétré par tw

$$\varphi = (a \vee b \vee \overline{c}) \wedge (\overline{a} \vee c \vee d) \wedge (\overline{b} \vee c \vee \overline{d})$$



B_1

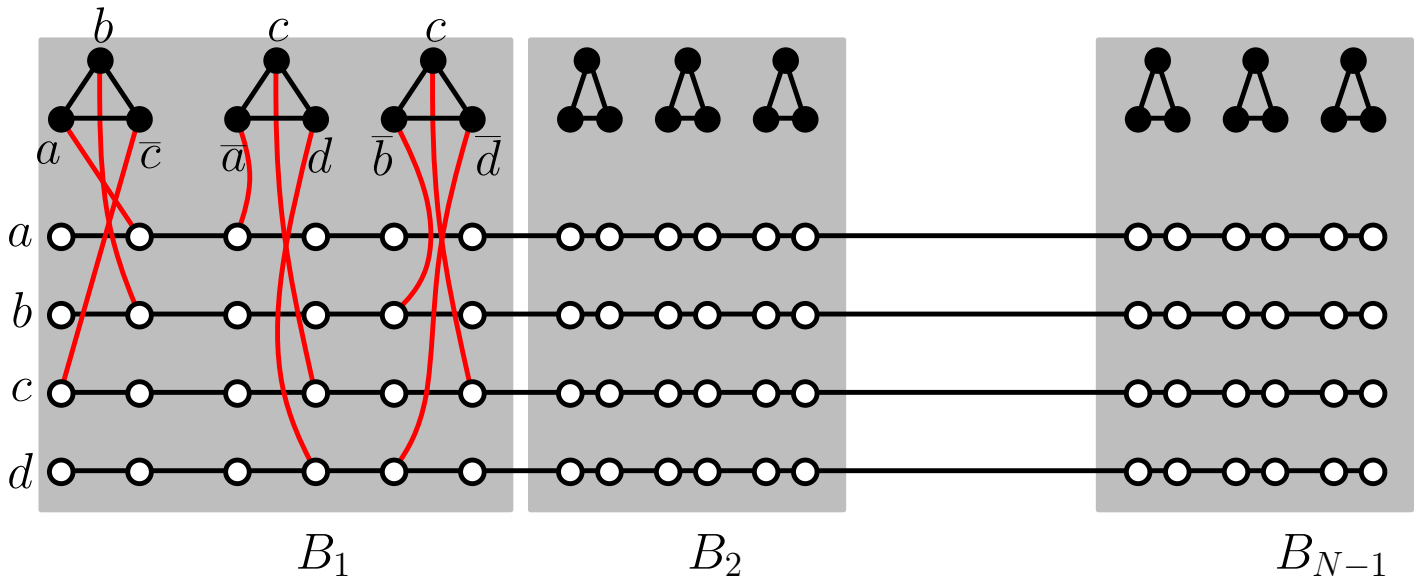
Bornes inférieures FPT basées sur SETH

Théorème : Independent Set paramétré par tw est résolu en $O^*(2^{\mathsf{tw}(G)})$ par programmation dynamique.

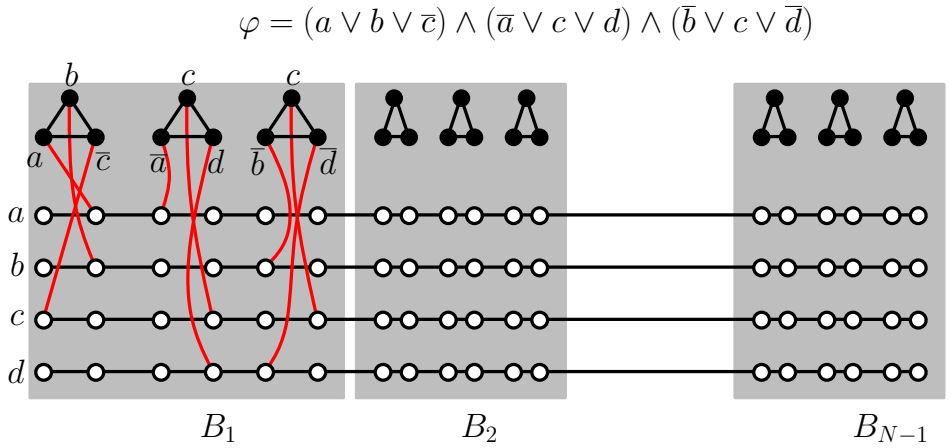
$\rightsquigarrow$ Peut-on obtenir un algorithme en $O^*\left((1.9999\ldots)^{\mathsf{tw}(G)}\right)$?

- k-SAT $\longrightarrow$ Independent Set paramétré par tw

$$\varphi = (a \vee b \vee \overline{c}) \wedge (\overline{a} \vee c \vee d) \wedge (\overline{b} \vee c \vee \overline{d})$$



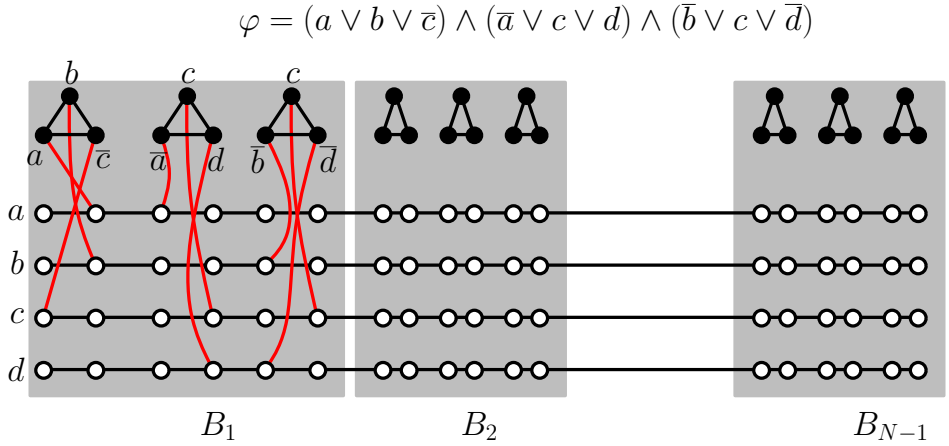
Bornes inférieures FPT basées sur SETH



Observation

- Il existe un block parmi les N blocks contenant uniquement des chemins Vrai ou Faux
- $\mathrm{pw}(H(\varphi)) \leqslant N + k$

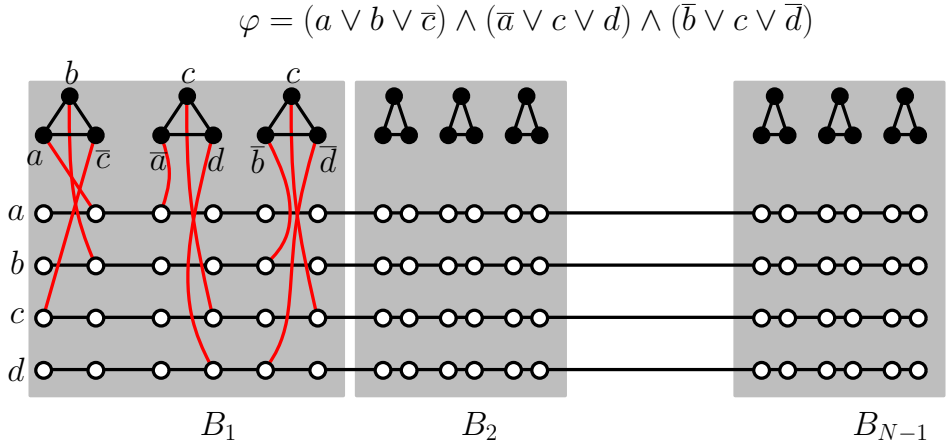
Bornes inférieures FPT basées sur SETH



Observation

- Il existe un block parmi les N blocks contenant uniquement des chemins VRAI ou FAUX
- $\mathsf{pw}(\mathsf{H}(\varphi)) \leqslant \mathsf{N} + \mathsf{k}$
- Un algorithme $\mathcal{A}$ en $O^*\left((2-\epsilon)^{\mathsf{tw}(\mathsf{G})}\right)$ pour tw-INDEPENDENT SET $\Longrightarrow$ un algorithme $\mathcal{A}'$ en $O^*\left((2-\epsilon)^{N}\right)$ pour k-SAT

Bornes inférieures FPT basées sur SETH



Observation

- Il existe un block parmi les N blocks contenant uniquement des chemins Vrai ou Faux
- $\mathsf{pw}(\mathsf{H}(\varphi)) \leqslant \mathsf{N} + \mathsf{k}$
- Un algorithme $\mathcal{A}$ en $O^*\left((2-\epsilon)^{\mathsf{tw}(\mathsf{G})}\right)$ pour tw-Independent Set $\Longrightarrow$ un algorithme $\mathcal{A}'$ en $O^*\left((2-\epsilon)^{N}\right)$ pour k-SAT

Théorème : S'il existe un algorithme de complexité $O^*\left((2-\epsilon)^{\mathsf{tw}(\mathsf{G})}\right)$ pour Independent Set, alors SETH n'est pas valide.

Conclusion

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

Autres implications de ETH-SETH

- bornes inférieures pour l'approximation
  ($\rightsquigarrow$ cf exposé de Nicolas Nisse)
- bornes inférieures pour des algorithmes polynomiaux (SETH)
- d'autres à venir ?

Quelques références

- D. Lokshtanov, D. Marx, S. Saurabh. *Lower bounds based on the exponential time hypothesis.* Bulletin of the EATCS 105:41-72, 2011
- M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk and S. Saurabh. *Parameterized Algorithms*. Book from the FPT school in Beldewo, 2014
- and many slides on the net : D. Lokshtanov, M. Pilipczuk[2], A. Drucker...

Kernelization

Théorème [folkore] : Un problème paramétré (P, k) est FPT ssi il admet une kernelisation (un noyau).

Une kernelisation d'un problème paramétré est un algorithme polynomial $\mathcal{K} : (x, k) \longrightarrow (x', k')$ tel que :

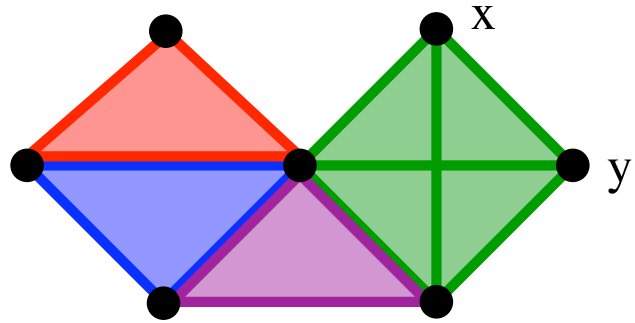
- (x, k) et (x', k') sont des instances équivalentes,
- $|x'| \leqslant g(k)$ et $k' \leqslant k$

Kernelization

Théorème [folkore] : Un problème paramétré (P, k) est FPT ssi il admet une kernelisation (un noyau).

Une kernelisation d'un problème paramétré est un algorithme polynomial $\mathcal{K} : (x, k) \longrightarrow (x', k')$ tel que :

- (x, k) et (x', k') sont des instances équivalentes,
- $|x'| \leqslant g(k)$ et $k' \leqslant k$

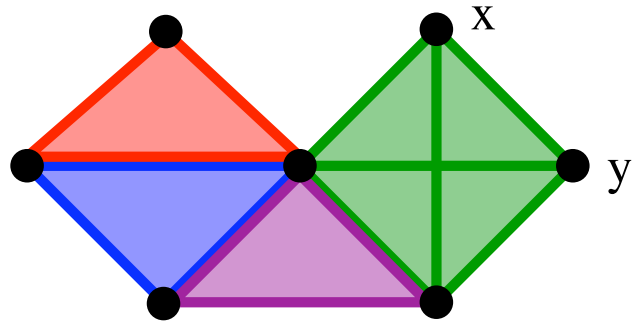


Edge Clique-Cover

- Un graphe $G = (V, E)$ et un paramètre $k \in \mathbb{N}$
- Peut-on couvrir les arètes E par au plus k cliques ?

Théorème [Gramm, Guo, Hüffner, Niedermeier]
Edge Clique-Cover admet un noyau de taille 2^k sommets.

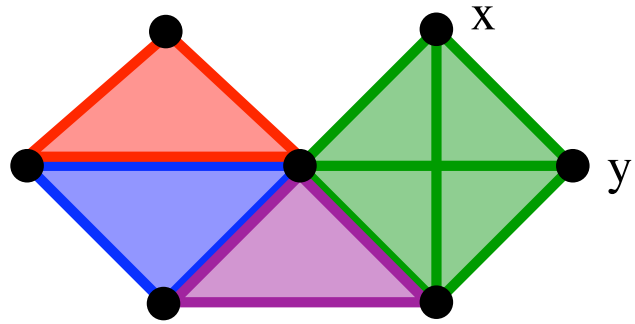
L'exemple de Edge Clique-Cover



Règles de réduction

1. Supprimer les sommets isolés
2. S'il existe deux sommets x et y tels que $N[x] = N[y]$, supprimer l'un des deux sommets.

L'exemple de Edge Clique-Cover



Règles de réduction

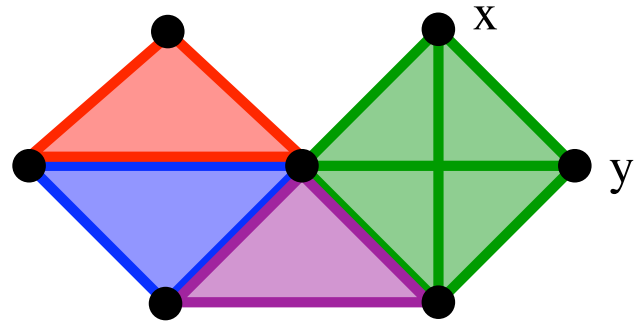
1. Supprimer les sommets isolés
2. S'il existe deux sommets x et y tels que $N[x] = N[y]$, supprimer l'un des deux sommets.

Preuve : Supposons qu'il existe k cliques $C_1 \dots C_k$ couvrant E.
A chaque sommet x on asscocie un verteur C de k bits tel que

$$C[x, i] = 1 \iff x \in C_i$$

Observation : $\boxed{\forall i \in [k]\ C[x, i] = C[y, i] \text{ ssi } N[x] = N[y]}$

L'exemple de Edge Clique-Cover



Règles de réduction

1. Supprimer les sommets isolés
2. S'il existe deux sommets x et y tels que $N[x] = N[y]$, supprimer l'un des deux sommets.

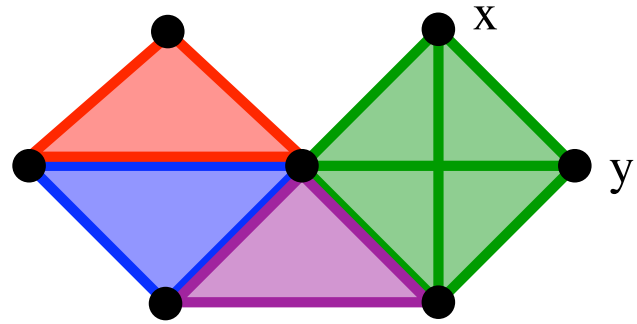
Preuve : Supposons qu'il existe k cliques $C_1 \dots C_k$ couvrant E.
A chaque sommet x on asscocie un verteur C de k bits tel que

$$C[x, i] = 1 \iff x \in C_i$$

Observation : $\boxed{\forall i \in [k]\ C[x, i] = C[y, i] \text{ ssi } N[x] = N[y]}$

Edge Clique-Cover admet-il un noyau polynomial ?

L'exemple de Edge Clique-Cover



Règles de réduction

1. Supprimer les sommets isolés
2. S'il existe deux sommets x et y tels que $N[x] = N[y]$, supprimer l'un des deux sommets.

Preuve : Supposons qu'il existe k cliques $C_1 \dots C_k$ couvrant E.
A chaque sommet x on asscocie un verteur C de k bits tel que

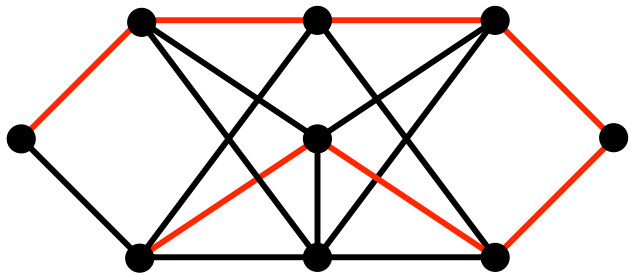
$$C[x, i] = 1 \iff x \in C_i$$

Observation : $\boxed{\forall i \in [k]\ C[x, i] = C[y, i] \text{ ssi } N[x] = N[y]}$

Edge Clique-Cover admet-il un noyau polynomial ? NON

Longest Path

- Un graphe $G = (V, E)$ et un paramètre $k \in \mathbb{N}$
- G contient-il un chemin de taille k ?



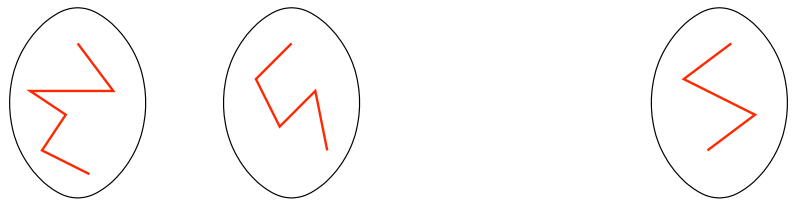
Longest Path est NP-Complet (cf. chemin hamiltonien) mais peut-être résolu en temps $O(c^k.n^{O(1)})$ grâce à Color Coding.

Longest Path

Hypothèse Il existe un algorithme de kernelization $\mathcal{A}$ pour Longest Path qui retourne un noyau polynomial de taille k^c bits.

- construisons une instance (G, k) avec t instances différentes

$$(G, k) = (G_1, k) \oplus (G_2, k) \oplus \ \ldots \ \oplus (G_t, k) \quad \text{avec } \log(t-1) = k^c$$

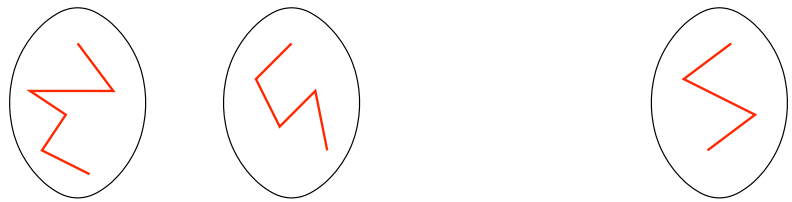


Longest Path

Hypothèse Il existe un algorithme de kernelization $\mathcal{A}$ pour Longest Path qui retourne un noyau polynomial de taille k^c bits.

- construisons une instance (G, k) avec t instances différentes

$$(G, k) = (G_1, k) \oplus (G_2, k) \oplus \ \ldots \ \oplus (G_t, k) \quad \text{avec } \log(t-1) = k^c$$



Observation : (G, k) admet un chemin de longueur k ssi $\exists i$ tq G_i admet un chemin de taille k.

Question :

Est-il possible que $\mathcal{A}$ identifie en temps polynomial quel G_i (s'il en existe un) possède un chemin de longueur k ?

Algorithmes de distillation

Un algorithme de OU-distillation $\mathcal{A}$ pour un problème de décision Q (i.e. non paramétré) sur l'alphabet Σ est un algorithme qui :

- reçoit une séquence $(x_1, \ldots x_t)$, avec $x_i \in \Sigma^*$, $\forall i \in [t]$;
- possède une complexité polynomiale en $\sum_{i=1}^{t} |x_i|$,

Algorithmes de distillation

Un algorithme de OU-distillation $\mathcal{A}$ pour un problème de décision Q (i.e. non paramétré) sur l'alphabet Σ est un algorithme qui :

- reçoit une séquence $(x_1, \ldots x_t)$, avec $x_i \in \Sigma^*$, $\forall i \in [t]$;
- possède une complexité polynomiale en $\sum_{i=1}^{t} |x_i|$,
- retourne $y \in \Sigma^*$ tel que
  1. $y \in Q \Leftrightarrow \exists i \in [t], x_i \in Q$;
  2. $|y|$ est polynomial en $\max_{i \in [t]} |x_i|$.

Algorithmes de distillation

Un algorithme de OU-distillation $\mathcal{A}$ pour un problème de décision Q (i.e. non paramétré) sur l'alphabet Σ est un algorithme qui :

- reçoit une séquence $(x_1, \dots x_t)$, avec $x_i \in \Sigma^*$, $\forall i \in [t]$;
- possède une complexité polynomiale en $\sum_{i=1}^{t} |x_i|$,
- retourne $y \in \Sigma^*$ tel que
  1. $y \in Q \Leftrightarrow \exists i \in [t], x_i \in Q$;
  2. $|y|$ est polynomial en $\max_{i \in [t]} |x_i|$.

Conjecture [Bodlaender, Downey, Fellows, Hermelin]
Aucun problème NP-Complet n'est OU-distillable.

Théorème [Fortnow et Santhanam]
S'il existe un problème NP-complet OU-distillable, alors $PH = \Sigma_p^3$

Algorithme de OU-composition

Un algorithme de OU-composition pour un problème paramétré $(Q, \kappa) \subseteq \Sigma^* \times \mathbb{N}$ est un algorithme $\mathcal{A}$ qui

- reçoit une séquence $((x_1, k), \ldots (x_t, k))$, avec $(x_i, k) \in \Sigma^* \times \mathbb{N}$ $\forall i \in [t]$;
- a une complexité polynomiale en $\sum_{i=1}^{t} |x_i| + k$,

Algorithme de OU-composition

Un algorithme de OU-composition pour un problème paramétré $(Q, \kappa) \subseteq \Sigma^* \times \mathbb{N}$ est un algorithme $\mathcal{A}$ qui

- reçoit une séquence $((x_1, k), \dots (x_t, k))$, avec $(x_i, k) \in \Sigma^* \times \mathbb{N}$ $\forall i \in [t]$;
- a une complexité polynomiale en $\sum_{i=1}^{t} |x_i| + k$,
- retourne $(y, k') \in \Sigma^* \times \mathbb{N}$ tel que
  1. $(y, k') \in (Q, \kappa) \Leftrightarrow \exists i \in [t], (x_i, k) \in (Q, \kappa)$;
  2. k' est polynomial en k.

Algorithme de OU-composition

Un algorithme de OU-composition pour un problème paramétré $(Q, \kappa) \subseteq \Sigma^* \times \mathbb{N}$ est un algorithme $\mathcal{A}$ qui

- reçoit une séquence $((x_1, k), \ldots (x_t, k))$, avec $(x_i, k) \in \Sigma^* \times \mathbb{N}$ $\forall i \in [t]$;
- a une complexité polynomiale en $\sum_{i=1}^{t} |x_i| + k$,
- retourne $(y, k') \in \Sigma^* \times \mathbb{N}$ tel que
  1. $(y, k') \in (Q, \kappa) \Leftrightarrow \exists i \in [t]$, $(x_i, k) \in (Q, \kappa)$;
  2. k' est polynomial en k.

Observation : L'existence d'un noyau polynomial pour LONGEST PATH impliquerait l'existence d'un algorithme de OU-composition.

Théorème [Bodlaender, Downey, Fellows, Hermelin]

Soit $(Q, \kappa) \in \Sigma^* \times \mathbb{N}$ un problème paramétré OU-composable tel que le problème $\tilde{Q} \in \Sigma^*$ (non-paramétré) est NP-Complet.
Si (Q, κ) admet un noyau polynomial, alors $\tilde{Q}$ est OU-distillable.

Théorème [Bodlaender, Downey, Fellows, Hermelin]

Soit $(Q, \kappa) \in \Sigma^* \times \mathbb{N}$ un problème paramétré OU-composable tel que le problème $\tilde{Q} \in \Sigma^*$ (non-paramétré) est NP-Complet.
Si (Q, κ) admet un noyau polynomial, alors $\tilde{Q}$ est OU-distillable.

Corollaire

Sauf si $PH = \Sigma_p^3$, Longest Path n'admet pas de noyau polynomial.

Transformations polynomiales et paramétrées

Soient (P, π) et (Q, κ) deux problèmes paramétrés.
Une transformation polynomiale et paramétrée (TPP) de (P, π) vers (Q, κ) est un algorithme polynomial $\mathcal{A}$ qui :

- à toute instance (x, k) de (P, π) associe une instance (x', k') de (Q, κ) ;
- $\boxed{(x, k) \in (P, \pi) \Longleftrightarrow (x', k') \in (Q, \pi) \quad \text{et} \quad k' \leqslant poly(k)}$

On note $(P, \pi) \leqslant_{TPP} (Q, \kappa)$

Transformations polynomiales et paramétrées

Soient (P, π) et (Q, κ) deux problèmes paramétrés.
Une transformation polynomiale et paramétrée (TPP) de (P, π) vers (Q, κ) est un algorithme polynomial $\mathcal{A}$ qui :

- à toute instance (x, k) de (P, π) associe une instance (x', k') de (Q, κ) ;
- $\boxed{(x, k) \in (P, \pi) \Longleftrightarrow (x', k') \in (Q, \pi) \quad \text{et} \quad k' \leqslant poly(k)}$

On note $(P, \pi) \leqslant_{TPP} (Q, \kappa)$

Théorème [Bodlaender, Thomassé, Yeo]

Soient (P, π) et (Q, κ) deux problèmes paramétrés tels que P est NP-Complet et Q appartient à NP.
Si $(P, \pi) \leqslant_{TPP} (Q, \kappa)$ et si (P, π) n'admet pas de noyau polynomial, alors (Q, κ) n'admet pas de noyau polynomial.

Utilisation

- construction de noyaux polynomiaux
- preuve de non-existence de noyau polynomial :

Utilisation

- construction de noyaux polynomiaux
- preuve de non-existence de noyau polynomial :

PATH PACKING

$\longrightarrow$ Tester si un graphe contient k chemins sommets disjoints de taille k

Utilisation

- construction de noyaux polynomiaux
- preuve de non-existence de noyau polynomial :

PATH PACKING

$\longrightarrow$ Tester si un graphe contient k chemins sommets disjoints de taille k

Remarque : PATH-PACKING n'est pas OU-composable !

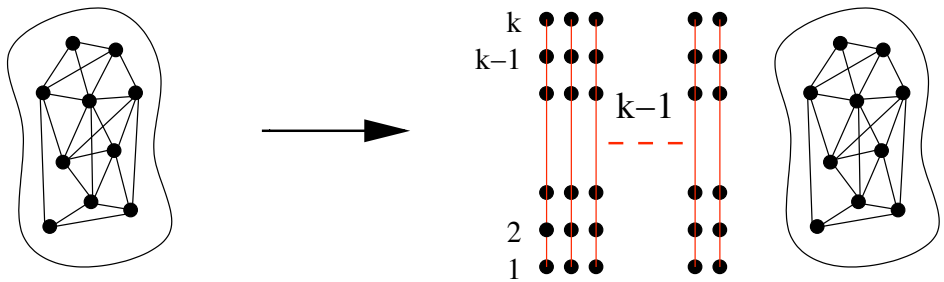
Utilisation

- construction de noyaux polynomiaux
- preuve de non-existence de noyau polynomial :

Path Packing

→ Tester si un graphe contient k chemins sommets disjoints de taille k

Remarque : Path-Packing n'est pas or-composable !



Longest Path $\leqslant_{TPP}$ Path Packing

Autres méthodes

- Compositions croisées : d'un problème NP-Complet vers un problème paramétré
  $\rightsquigarrow$ combinaison des deux techniques précédentes

- ET-compositions et ET-distillations

- Compositions faibles
  $\rightsquigarrow$ k-Vertex Cover n'admet pas de noyau de taille $O(k^{2-\epsilon})$ $(\forall \epsilon > 0)$ sauf si co-NP$\subseteq$NP/poly.