Complexité et algorithmes paramétrés (2)
-
Kernelization et techniques algorithmiques

Christophe PAUL
CNRS - LIRMM

EJC Informatique Mathématique du GDR IM
Perpignan, avril 2013

Algorithmes paramétrés

- Arbre de recherche bornée - VERTEX COVER
- Compression itérative - FEEDBACK VERTEX SET
- Algorithmes randomisés - FEEDBACK VERTEX SET
- Color Coding - LONGEST PATH

Kernelization - Bornes inférieures

- Algorithmes de OU-composition
- Transformation polynomiales paramétrées

Kernelization - Bornes supérieures

- VERTEX COVER et programmation linéaire
- Un noyau linéaire pour FAST à l'aide de couplages

Vertex Cover (1)

Règle de branchement : Soit (G, k) une instance de Vertex Cover telle que $k > 0$ et $\exists (u, v) \in E(G)$, alors brancher sur :

1. $(G - u, k - 1)$
2. $(G - v, k - 1)$

Règle de réduction : Soit (G, k) une instance de Vertex Cover.

1. Si $k = 0$ et $E(G) \neq \emptyset$, retourner Faux
2. Si $E(G) = \emptyset$, retourner Vrai

Théorème : Vertex Cover (paramétré par la taille k de la solution) admet un algorithme de recherche bornée de complexité $2^k \cdot (n + m)$.

Vertex Cover (1)

Règle de branchement : Soit (G, k) une instance de Vertex Cover telle que $k > 0$ et $\exists (u, v) \in E(G)$, alors brancher sur :

1. $(G - u, k - 1)$
2. $(G - v, k - 1)$

Règle de réduction : Soit (G, k) une instance de Vertex Cover.

1. Si $k = 0$ et $E(G) \neq \emptyset$, retourner Faux
2. Si $E(G) = \emptyset$, retourner Vrai

Théorème : Vertex Cover (paramétré par la taille k de la solution) admet un algorithme de recherche bornée de complexité $2^k \cdot (n + m)$.

Peut-on faire mieux ? (Diminuer la taille de l'arbre de branchement)

Vertex Cover (2)

Règle de branchement (2) : Soient (G, k) une instance de Vertex Cover et x un sommet de degré $d(x) \geqslant 1$, alors brancher sur

1. $(G - x, k - 1)$ – x est sélectionné dans la solution
2. $(G - N[x], k - d(x))$ – $N(x)$ est sélectionné dans la solution

Vertex Cover (2)

Règle de branchement (2) : Soient (G, k) une instance de Vertex Cover et x un sommet de degré $d(x) \geqslant 1$, alors brancher sur

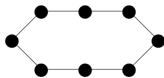
1. $(G - x, k - 1)$ – x est sélectionné dans la solution
2. $(G - N[x], k - d(x))$ – $N(x)$ est sélectionné dans la solution

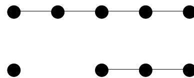
Observation :

- Si on peut garantir l'existence d'un sommet de "grand" degré, alors une branche de l'arbre sera plus courte.

Vertex Cover (3)

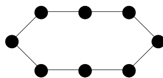
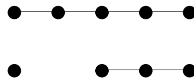
Observation : Le problème Vertex Cover est polynomial sur les graphes de degré maximum 2.





Vertex Cover (3)

Observation : Le problème Vertex Cover est polynomial sur les graphes de degré maximum 2.

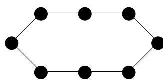



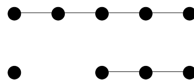
Algorithme Vertex Cover(G, k)

- Si G contient un sommet x tel que $d(x) \geqslant 3$, alors appliquer la règle de branchement (2)
- Sinon résoudre Vertex Cover en temps polynomial.

VERTEX COVER (3)

Observation : Le problème VERTEX COVER est polynomial sur les graphes de degré maximum 2.





Algorithme VERTEX COVER(G, k)

- Si G contient un sommet x tel que $d(x) \geqslant 3$, alors appliquer la règle de branchement (2)
- Sinon résoudre VERTEX COVER en temps polynomial.

Quelle est la taille de l'arbre de branchement ?

Vertex Cover (4) – analyse de l'arbre de branchement

$$T_{vc}(k+3) \geqslant T_{vc}(k+2) + T_{vc}(k) + 1 \quad (1)$$

$$T_{vc}(1) = T_{vc}(2) = 1 \qquad T_{vc}(0) = 0$$

Vertex Cover (4) – analyse de l'arbre de branchement

$$T_{vc}(k+3) \geqslant T_{vc}(k+2) + T_{vc}(k) + 1 \quad (1)$$

$$T_{vc}(1) = T_{vc}(2) = 1 \qquad T_{vc}(0) = 0$$

En fixant $T_{vc}(k) = c^k - 1$, nous devons résoudre

$$c^3 = c^2 + 1 \quad (2)$$

VERTEX COVER (4) – analyse de l'arbre de branchement

$$T_{vc}(k+3) \geqslant T_{vc}(k+2) + T_{vc}(k) + 1 \quad (1)$$

$$T_{vc}(1) = T_{vc}(2) = 1 \qquad T_{vc}(0) = 0$$

En fixant $T_{vc}(k) = c^k - 1$, nous devons résoudre

$$c^3 = c^2 + 1 \quad (2)$$

On prend la plus petite racine positive, ici $c = 5^{\frac{1}{4}} \leqslant 1,47$

Théorème : VERTEX COVER (paramétré par la taille de la solution) admet un algorithme de recherche bornée de complexité $1,47^k \cdot (n+m)$.

Règles de branchement

Une règle de branchement pour un problème paramétré κ-**P** est un algorithme polynomial qui étant donnée une instance $(x, \kappa(x))$, $(k > 1)$ retourne un ensemble $\mathcal{I} = \{(x_1, \kappa(x_1)), \ldots, (x_c, \kappa(x_\ell))\}$ de $\ell > 0$ instances telles que

1. $\forall i \in [\ell]$, $|x_i| \leqslant |x|$ et $k_i < k$ et
2. $(x, \kappa(x)) \in \kappa\text{-}\mathbf{P} \iff \exists i \in [\ell],\ (x_i, \kappa(x_i)) \in \kappa\text{-}\mathbf{P}$

Règles de branchement

Une règle de branchement pour un problème paramétré κ-**P** est un algorithme polynomial qui étant donnée une instance $(x, \kappa(x))$, $(k > 1)$ retourne un ensemble $\mathcal{I} = \{(x_1, \kappa(x_1))$, ..., $(x_c, \kappa(x_\ell))\}$ de $\ell > 0$ instances telles que

1. $\forall i \in [\ell]$, $|x_i| \leqslant |x|$ et $k_i < k$ et
2. $(x, \kappa(x)) \in \kappa\text{-}\mathbf{P} \iff \exists i \in [\ell],\ (x_i, \kappa(x_i)) \in \kappa\text{-}\mathbf{P}$

A chaque règle de branchement est associé un vecteur de branchement (ordonné par valeurs décroissantes)

$$(v_1 = \kappa(x) - \kappa(x_1), \ldots, v_\ell = \kappa(x) - \kappa(x_\ell))$$

Lemme : Une règle de branchement de vecteur $(v_1, \ldots, v_\ell)$ développe un arbre de recherche de taille c^k, où c est la plus petite racine positive de

$$a^k = a^{k-v_1} + \cdots + a^{k-v_\ell}$$

(Étrange) Vertex Cover

Etant donné un graphe G paramétré par $k \in \mathbb{N}$, G admet-il un vertex cover de taille 2^k ?

(Étrange) VERTEX COVER

Etant donné un graphe G paramétré par $k \in \mathbb{N}$, G admet-il un vertex cover de taille 2^k ?

Observation : Si le problème (Étrange) VERTEX COVER peut être résolu par une règle de branchement, alors $\mathbf{P} = \mathbf{NP}$

(Étrange) VERTEX COVER

Etant donné un graphe G paramétré par $k \in \mathbb{N}$, G admet-il un vertex cover de taille 2^k ?

Observation : Si le problème (Étrange) VERTEX COVER peut être résolu par une règle de branchement, alors $\mathbf{P} = \mathbf{NP}$

- chaque application de la règle de branchement divise valeur du paramètre par un multiple de 2

(Étrange) Vertex Cover

Etant donné un graphe G paramétré par $k \in \mathbb{N}$, G admet-il un vertex cover de taille 2^k ?

Observation : Si le problème (Étrange) Vertex Cover peut être résolu par une règle de branchement, alors $\mathbf{P} = \mathbf{NP}$

- chaque application de la règle de branchement divise valeur du paramètre par un multiple de 2
- La hauteur de l'arbre de recherche est donc $O(\log k)$
- et la taille de l'arbre de recherche est polynomial en k ! ! !

Algorithmes paramétrés

- Arbre de recherche bornée - Vertex Cover
- Compression itérative - Feedback Vertex Set
- Algorithmes randomisés - Feedback Vertex Set
- Color Coding - Longest Path

Kernelization - Bornes inférieures

- Algorithmes de ou-composition
- Transformation polynomiales paramétrées

Kernelization - Bornes supérieures

- Vertex Cover et programmation linéaire
- Un noyau linéaire pour fast à l'aide de couplages

Compression itérative - Principe

Utilisée pour les problèmes de **minimisation** dont le paramètre est la taille k de la solution.

1. **Etape de compression :**
   Etant donnée une solution de taille $k+1$, trouver un algo **FPT** qui
   - soit on construit une solution de taille k
   - ou prouve qu'il n'y a pas de solution de taille k

Compression itérative - Principe

Utilisée pour les problèmes de **minimisation** dont le paramètre est la taille k de la solution.

1. **Etape de compression :**
   Etant donnée une solution de taille $k+1$, trouver un algo **FPT** qui
   - soit on construit une solution de taille k
   - ou prouve qu'il n'y a pas de solution de taille k
2. **Itération :**
   L'algorithme considère les sous-instances $X_i = X[x_1, \ldots x_i]$ les unes après les autres. Et à partir d'une solution S_i pour X_i :

Compression itérative - Principe

Utilisée pour les problèmes de **minimisation** dont le paramètre est la taille k de la solution.

1. **Etape de compression :**
   Etant donnée une solution de taille $k+1$, trouver un algo **FPT** qui
   - soit on construit une solution de taille k
   - ou prouve qu'il n'y a pas de solution de taille k
2. **Itération :**
   L'algorithme considère les sous-instances $X_i = X[x_1, \ldots x_i]$ les unes après les autres. Et à partir d'une solution S_i pour X_i :
   - construit une solution triviale pour G_{i+1} de taille $|S_{i+1}|$
   - applique l'étape de compression pour essayer d'améliorer cette solution

Compression itérative - Feedback Vertex Set

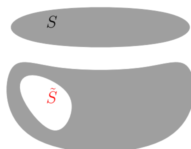
Feedback Vertex Set

- Etant donnés un graphe G et un entier k. Existe-t-il un ensemble S d'au plus k sommets tel que $G - S$ soit acyclique ?

Compression itérative - Feedback Vertex Set

Feedback Vertex Set

- Etant donnés un graphe G et un entier k. Existe-t-il un ensemble S d'au plus k sommets tel que $G - S$ soit acyclique ?



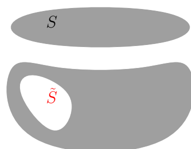
Disjoint Feedback Vertex Set

- Etant donnés un graphe G et un ensemble S de taille k tel que $G - S$ soit acyclique. Existe-t-il un ensemble $\tilde{S}$ de sommets tel que $G - \tilde{S}$ soit acyclique et $S \cap \tilde{S} = \emptyset$ et $|\tilde{S}| < |S|$?

Compression itérative - Feedback Vertex Set

Feedback Vertex Set

- Etant donnés un graphe G et un entier k. Existe-t-il un ensemble S d'au plus k sommets tel que $G - S$ soit acyclique ?



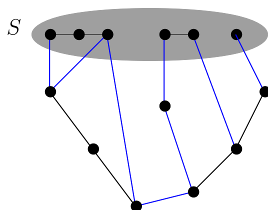
Disjoint Feedback Vertex Set

- Etant donnés un graphe G et un ensemble S de taille k tel que $G - S$ soit acyclique. Existe-t-il un ensemble $\tilde{S}$ de sommets tel que $G - \tilde{S}$ soit acyclique et $S \cap \tilde{S} = \emptyset$ et $|\tilde{S}| < |S|$?

Lemme : Si l'on peut résoudre Disjoint Feedback Vertex Set en temps $O^*(c^k)$ (avec $c \in \mathbb{N}^+$), alors on peut résoudre Feedback Vertex Set en temps $O^*((c+1)^k)$.

Compression itérative - Feedback Vertex Set

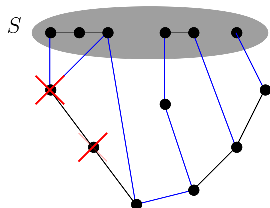
Soit (G, S, k) une instance de Disjoint Feedback Vertex Set



Règle réduction 1 : Si x possède deux voisins dans une composante connexe de $G[S]$, alors retourner $(G - \{x\}, S, k - 1)$

Compression itérative - Feedback Vertex Set

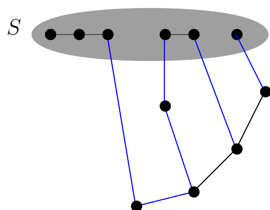
Soit (G, S, k) une instance de Disjoint Feedback Vertex Set



Règle réduction 1 : Si x possède deux voisins dans une composante connexe de $G[S]$, alors retourner $(G - \{x\}, S, k - 1)$

Compression itérative - Feedback Vertex Set

Soit (G, S, k) une instance de Disjoint Feedback Vertex Set

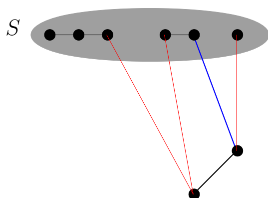


Règle réduction 1 : Si x possède deux voisins dans une composante connexe de $G[S]$, alors retourner $(G - \{x\}, S, k - 1)$

Règle réduction 2 : Si x possède un seul voisin y dans S et un seul voisin z dans $G - S$, alors retourner $(G - x \cup \{yz\}, S, k)$

Compression itérative - Feedback Vertex Set

Soit (G, S, k) une instance de Disjoint Feedback Vertex Set



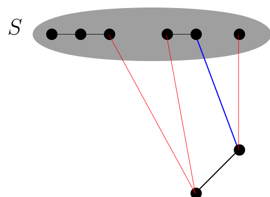
Règle réduction 1 : Si x possède deux voisins dans une composante connexe de $G[S]$, alors retourner $(G - \{x\}, S, k - 1)$

Règle réduction 2 : Si x possède un seul voisin y dans S et un seul voisin z dans $G - S$, alors retourner $(G - x \cup \{yz\}, S, k)$

Règle de branchement : Si x possède deux voisins dans des composantes connexes différentes de $G - S$, alors brancher sur

$(G - x, S, k - 1)$ et $(G, S \cup \{x\}, k)$

Compression itérative - Feedback Vertex Set

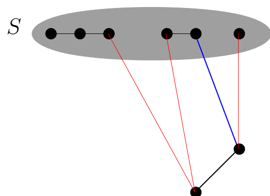


Théorème : L'alogrithme de réduction et branchement résoud Disjoint Feedback Vertex Set en temps $O^*(4^k)$.

- Pour analyser la taille de l'arbre de branchement, on utilise la mesure

$$\mu = k + \#cc(G[S]) \leqslant 2k$$

Compression itérative - Feedback Vertex Set



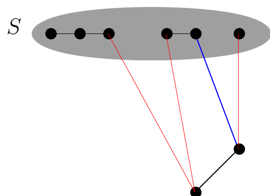
Théorème : L'alogrithme de réduction et branchement résoud Disjoint Feedback Vertex Set en temps $O^*(4^k)$.

- Pour analyser la taille de l'arbre de branchement, on utilise la mesure

$$\mu = k + \#cc(G[S]) \leqslant 2k$$

- Si x est supprimé (ou ajouté dans $\tilde{S}$), μ décroit
  Notons que l'ajout de x dans $G[S]$ ne crée pas de cycle

Compression itérative - Feedback Vertex Set



Théorème : L'alogrithme de réduction et branchement résoud Disjoint Feedback Vertex Set en temps $O^*(4^k)$.

- Pour analyser la taille de l'arbre de branchement, on utilise la mesure

$$\mu = k + \#cc(G[S]) \leqslant 2k$$

- Si x est supprimé (ou ajouté dans $\tilde{S}$), μ décroit
  Notons que l'ajout de x dans $G[S]$ ne crée pas de cycle
- Sinon x est ajouté à S et $\#cc(G[S]$ décroit

Algorithmes paramétrés

- Arbre de recherche bornée - VERTEX COVER
- Compression itérative - FEEDBACK VERTEX SET
- Algorithmes randomisés - FEEDBACK VERTEX SET
- Color Coding - LONGEST PATH

Kernelization - Bornes inférieures

- Algorithmes de OU-composition
- Transformation polynomiales paramétrées

Kernelization - Bornes supérieures

- VERTEX COVER et programmation linéaire
- Un noyau linéaire pour FAST à l'aide de couplages

Feedback Vertex Set randomisé

Soit G un graphe sans boucle, ni sommet de degré 0, 1 ou 2.

- les sommets avec boucles sont supprimés car, ils appartiennent à tous FVS ;
- les sommets de degré 0 ou 1 n'apparaîssent dans aucun cycle, ils peuvent être supprimés ;
- il exsite toujours un FVS ne contenant pas de sommet de degré 2 : s'il en existe un, on peut contracter une arête incidente.

Feedback Vertex Set randomisé

Soit G un graphe sans boucle, ni sommet de degré 0, 1 ou 2.

Lemme : Soient S un FVS de G. Notons $X = V(G) \setminus S$ et $E_X = E(G) \cap (X \times X)$. Alors

$$|E_X| \leqslant |E(G)|/2$$

Notons $E_{S,X} = E(G) \cap (S \times X)$

Feedback Vertex Set randomisé

Soit G un graphe sans boucle, ni sommet de degré 0, 1 ou 2.

Lemme : Soient S un FVS de G. Notons $X = V(G) \setminus S$ et $E_X = E(G) \cap (X \times X)$. Alors

$$|E_X| \leqslant |E(G)|/2$$

Notons $E_{S,X} = E(G) \cap (S \times X)$

- Puisque G est de degré minimum 3 :

$$3 \cdot |X| \leqslant \Sigma_{x \in X} d(x) = 2 \cdot |E_X| + |E_{S,X}|$$

Feedback Vertex Set randomisé

Soit G un graphe sans boucle, ni sommet de degré 0, 1 ou 2.

Lemme : Soient S un FVS de G. Notons $X = V(G) \setminus S$ et $E_X = E(G) \cap (X \times X)$. Alors

$$|E_X| \leqslant |E(G)|/2$$

Notons $E_{S,X} = E(G) \cap (S \times X)$

- Puisque G est de degré minimum 3 :

$$3 \cdot |X| \leqslant \Sigma_{x \in X} d(x) = 2 \cdot |E_X| + |E_{S,X}|$$

- Puisque S est un FVS : $|E_X| < |X|$ donc

$$3 \cdot |E_X| < 2 \cdot |E_X| + |E_{S,X}|$$

FEEDBACK VERTEX SET randomisé (2)

Théorème [Becker, Bar-Yehuda, Geiger 2000]

Le problème FEEDBACK VERTEX SET admet un algorithme de complexité $O(c4^k \cdot kn)$ (avec $c > 0$) qui retourne un FVS de taille k d'un graphe G (s'il en existe un) avec probabilité au moins $1 - (1 - \frac{1}{4^k})^{c4^k}$.

Feedback Vertex Set randomisé (2)

Théorème [Becker, Bar-Yehuda, Geiger 2000]
Le problème Feedback Vertex Set admet un algorithme de complexité $O(c4^k \cdot kn)$ (avec $c > 0$) qui retourne un FVS de taille k d'un graphe G (s'il en existe un) avec probabilité au moins $1 - (1 - \frac{1}{4^k})^{c4^k}$.

- Soit S un ensemble de sommets obtenu en tirant k fois uniformément une arête puis un sommet de l'arête

$$\mathbb{P}(S \text{ n'est pas un FVS}) = 1 - \frac{1}{4^k}$$

Feedback Vertex Set randomisé (2)

Théorème [Becker, Bar-Yehuda, Geiger 2000]
Le problème Feedback Vertex Set admet un algorithme de complexité $O(c4^k \cdot kn)$ (avec $c > 0$) qui retourne un FVS de taille k d'un graphe G (s'il en existe un) avec probabilité au moins $1 - (1 - \frac{1}{4^k})^{c4^k}$.

- Soit S un ensemble de sommets obtenu en tirant k fois uniformément une arête puis un sommet de l'arête

$$\mathbb{P}(S \text{ n'est pas un FVS}) = 1 - \frac{1}{4^k}$$

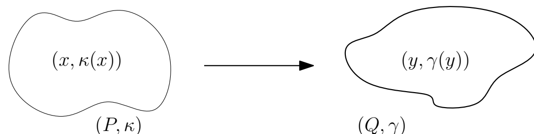
- Soient $S_1 \dots S_\ell$ avec $\ell \in [c4^k]$

$$\mathbb{P}(\forall i \in [c4^k],\ S_i \text{ n'est pas un FVS}) = (1 - \frac{1}{4^k})^{c4^k}$$

$$\mathbb{P}(\exists i \in [c4^k],\ S_i \text{ est un FVS}) = 1 - (1 - \frac{1}{4^k})^{c4^k}$$

Color Coding : Idée – Principe [Alon, Yuster, Zwick]

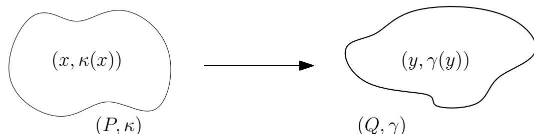
Color Coding : Transformer un problème (P, κ) en un problème "plus facile" (Q, γ) à l'aide de coloration aléatoire



- (Q, γ), la version colorée de (P, κ), teste l'existence d'une solution multicolore utilisant une et une seule fois chaque couleur.

Color Coding : Idée – Principe [Alon, Yuster, Zwick]

Color Coding : Transformer un problème (P, κ) en un problème "plus facile" (Q, γ) à l'aide de coloration aléatoire



- (Q, γ), la version colorée de (P, κ), teste l'existence d'une solution multicolore utilisant une et une seule fois chaque couleur.

Analyse

- estimer la probabilité qu'une solution à (P, κ) soit multcolore (pour en déduire le nombre d'itérations nécessaires du principe pour l'obtenir sûrement)
- Test efficace de l'existence d'une solution multicolore ?

Longest Path (1)

Tester l'existence d'un un chemin de longueur k dans graphe G

Lemme : Soient P_k de longueur k dans un graphe G et $\omega : V(G) \to [k]$ une coloration aléatoire de G. Alors

$$\mathbb{P}(P_k^{\omega}) = \frac{k!}{k^k} > \frac{1}{e^k}$$

avec P_k^{ω} le chemin multicolore

Longest Path (1)

Tester l'existence d'un un chemin de longueur k dans graphe G

Lemme : Soient P_k de longueur k dans un graphe G et $\omega : V(G) \to [k]$ une coloration aléatoire de G. Alors

$$\mathbb{P}(P_k^\omega) = \frac{k!}{k^k} > \frac{1}{e^k}$$

avec P_k^ω le chemin multicolore

- k^k colorations possibles pour P_k
- $k!$ d'entre elles sont multicolores

Longest Path (1)

Tester l'existence d'un un chemin de longueur k dans graphe G

Lemme : Soient P_k de longueur k dans un graphe G et $\omega : V(G) \to [k]$ une coloration aléatoire de G. Alors

$$\mathbb{P}(P_k^\omega) = \frac{k!}{k^k} > \frac{1}{e^k}$$

avec P_k^ω le chemin multicolore

Lemme : Un algorithme testant l'existence d'un chemin P_k multicolore en temps $f(k)$ implique un algorithme de complexité $e^k \cdot f(k) \cdot n^{O(1)}$ pour Longest Path.

LONGEST PATH (1)

Tester l'existence d'un un chemin de longueur k dans graphe G

Lemme : Soient P_k de longueur k dans un graphe G et $\omega : V(G) \rightarrow [k]$ une coloration aléatoire de G. Alors

$$\mathbb{P}(P_k^\omega) = \frac{k!}{k^k} > \frac{1}{e^k}$$

avec P_k^ω le chemin multicolore

Lemme : Un algorithme testant l'existence d'un chemin P_k multicolore en temps $f(k)$ implique un algorithme de complexité $e^k \cdot f(k) \cdot n^{O(1)}$ pour LONGEST PATH.

- Soit G^ω un graphe coloré $\qquad \mathbb{P}(P_k^\omega \notin G^\omega) < 1 - \frac{1}{e^k}$

Longest Path (1)

Tester l'existence d'un un chemin de longueur k dans graphe G

Lemme : Soient P_k de longueur k dans un graphe G et $\omega : V(G) \rightarrow [k]$ une coloration aléatoire de G. Alors

$$\mathbb{P}(P_k^\omega) = \frac{k!}{k^k} > \frac{1}{e^k}$$

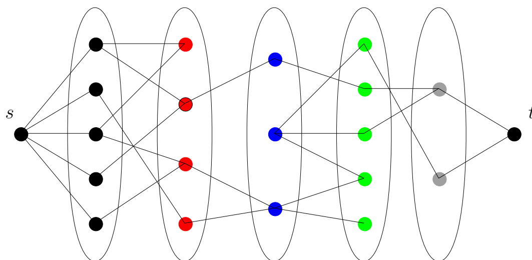
avec P_k^ω le chemin multicolore

Lemme : Un algorithme testant l'existence d'un chemin P_k multicolore en temps $f(k)$ implique un algorithme de complexité $e^k \cdot f(k) \cdot n^{O(1)}$ pour Longest Path.

- Soit G^ω un graphe coloré $\mathbb{P}(P_k^\omega \notin G^\omega) < 1 - \frac{1}{e^k}$
- Après e^k colorations G^{ω_i}, $\mathbb{P}(\forall i, P_k^\omega \notin G^{\omega_i}) < (1 - \frac{1}{e^k})^{e^k} = \frac{1}{e}$

Longest Path (2)

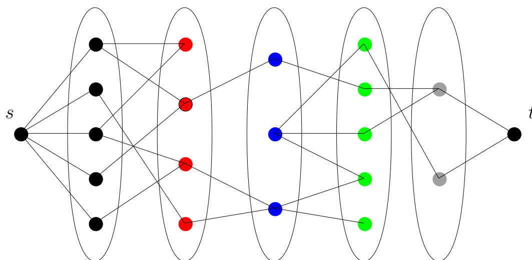
- Tester l'existence d'un chemin multicolore à l'aide de "flots"



- Tester les $k!$ permutations de couleurs
- Supprimer les arêtes monochromatiques et celles entre deux couleurs non-consécutives
- Tester l'existence d'un chemin (orienté) entre s et t

Longest Path (2)

- Tester l'existence d'un chemin multicolore à l'aide de "flots"



- Tester les $k!$ permutations de couleurs
- Supprimer les arêtes monochromatiques et celles entre deux couleurs non-consécutives
- Tester l'existence d'un chemin (orienté) entre s et t

Complexité : $O(k! \cdot |E(G)|)$

Longest Path (2)

- Tester l'existence d'un chemin multicolore à l'aide de programmation dynamique

On définit $2^k \cdot |V(G)|$ variables booléennes telles que

$$T(x, C) = \text{VRAI}$$

ssi il existe un chemin mulitcolore de s à x sur les couleurs de C.

Longest Path (2)

- Tester l'existence d'un chemin multicolore à l'aide de programmation dynamique

On définit $2^k \cdot |V(G)|$ variables booléennes telles que

$$T(x, C) = \text{VRAI}$$

ssi il existe un chemin mulitcolore de s à x sur les couleurs de C.

- $T(s, \emptyset) = \text{VRAI}$
- $T(x, C) = \bigvee_{xy \in E} T(y, C \setminus \{\omega(x)\})$

Longest Path (2)

- Tester l'existence d'un chemin multicolore à l'aide de programmation dynamique

On définit $2^k \cdot |V(G)|$ variables booléennes telles que

$$T(x, C) = \text{VRAI}$$

ssi il existe un chemin mulitcolore de s à x sur les couleurs de C.

- $T(s, \emptyset) = \text{VRAI}$
- $T(x, C) = \bigvee_{xy \in E} T(y, C \setminus \{\omega(x)\})$

Complexité pour remplir la table : $O(2^k \cdot (|V(G)| + |E(G)|)$

Longest Path (3)

Observation : il est possible de dérandomiser l'algorithme

Théorème [Alon, Yuster, Zwick]
Longest Path admet un algorithme FPT (randomisé) de complexité $2^{O(k)} \cdot (|V(G)| + |E(G)|)$.

Exercice :

1. Proposer un algorithme de color-coding pour k-Disjoint Triangle
2. Proposer un algorithme de color-coding pour le problème Vertex Cover.
3. Pourquoi color-coding ne s'applique pas pour Clique ?

Algorithmes paramétrés

- Arbre de recherche bornée - VERTEX COVER
- Compression itérative - FEEDBACK VERTEX SET
- Algorithmes randomisés - FEEDBACK VERTEX SET
- Color Coding - LONGEST PATH

Kernelization - Bornes inférieures

- Algorithmes de OU-composition
- Transformation polynomiales paramétrées

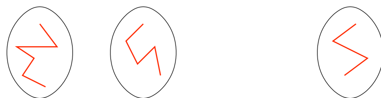
Kernelization - Bornes supérieures

- VERTEX COVER et programmation linéaire
- Un noyau linéaire pour FAST à l'aide de couplages

Non-existence de noyau polynomial

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

- construisons une instance (G, k) avec t instances différentes
  $$(G, k) = (G_1, k) \oplus (G_2, k) \oplus \ \ldots \ \oplus (G_t, k)$$



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 de décider si une des instances possède un chemin de longueur k en disposant de moins de 1 bits par instance en moyenne ?

Algorithmes de distillation

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

- reçoit une séquence $(x_1, \ldots x_t)$ d'instances de Q, $\forall i \in [t]$;

Algorithmes de distillation

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

- reçoit une séquence $(x_1, \ldots x_t)$ d'instances de Q, $\forall i \in [t]$;
- 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) est un algorithme qui :

- reçoit une séquence $(x_1, \ldots x_t)$ d'instances de Q, $\forall i \in [t]$;
- possède une complexité polynomiale en $\sum_{i=1}^{t} |x_i|$,
- retourne une instance y de Q 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) est un algorithme qui :

- reçoit une séquence $(x_1, \ldots x_t)$ d'instances de Q, $\forall i \in [t]$;
- possède une complexité polynomiale en $\sum_{i=1}^{t} |x_i|$,
- retourne une instance y de Q 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.

Algorithmes de distillation

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

- reçoit une séquence $(x_1, \dots x_t)$ d'instances de Q, $\forall i \in [t]$;
- possède une complexité polynomiale en $\sum_{i=1}^{t} |x_i|$,
- retourne une instance y de Q 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 κ-Q est un algorithme $\mathcal{A}$ qui

- reçoit une séquence $((x_1, k), \ldots (x_t, k))$, d'instances paramétrées ;

Algorithme de OU-composition

Un algorithme de OU-composition pour un problème paramétré κ-Q est un algorithme $\mathcal{A}$ qui

- reçoit une séquence $((x_1, k), \ldots (x_t, k))$, d'instances paramétrées ;
- 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 κ-Q est un algorithme $\mathcal{A}$ qui

- reçoit une séquence $((x_1, k), \ldots (x_t, k))$, d'instances paramétrées ;
- a une complexité polynomiale en $\sum_{i=1}^{t} |x_i| + k$,
- retourne (y, k') 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 κ-Q est un algorithme $\mathcal{A}$ qui

- reçoit une séquence $((x_1, k), \ldots (x_t, k))$, d'instances paramétrées ;
- a une complexité polynomiale en $\sum_{i=1}^{t} |x_i| + k$,
- retourne (y, k') 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 κ-Q un problème paramétré OU-composable tel que le problème $\tilde{Q}$ (non-paramétré) est NP-Complet.
Si κ-Q admet un noyau polynomial, alors $\tilde{Q}$ est OU-distillable.

Théorème [Bodlaender, Downey, Fellows, Hermelin]

Soit κ-Q un problème paramétré OU-composable tel que le problème $\tilde{Q}$ (non-paramétré) est NP-Complet.
Si κ-Q admet un noyau polynomial, alors $\tilde{Q}$ est OU-distillable.

- $(x, \kappa(x))$ instance de κ-Q $\longrightarrow$ $x\#1^k$ instance de $\tilde{Q}$

Théorème [Bodlaender, Downey, Fellows, Hermelin]

Soit κ-Q un problème paramétré OU-composable tel que le problème $\tilde{Q}$ (non-paramétré) est NP-Complet.
Si κ-Q admet un noyau polynomial, alors $\tilde{Q}$ est OU-distillable.

- $(x, \kappa(x))$ instance de κ-Q $\longrightarrow$ $x\#1^k$ instance de $\tilde{Q}$
- Puisque $\tilde{Q}$ est un problème NP-Complet, il existe deux transformations polynomiales

$$\Phi : \tilde{Q} \longrightarrow \mathrm{SAT} \qquad \Psi : \mathrm{SAT} \longrightarrow \tilde{Q}$$

Théorème [Bodlaender, Downey, Fellows, Hermelin]

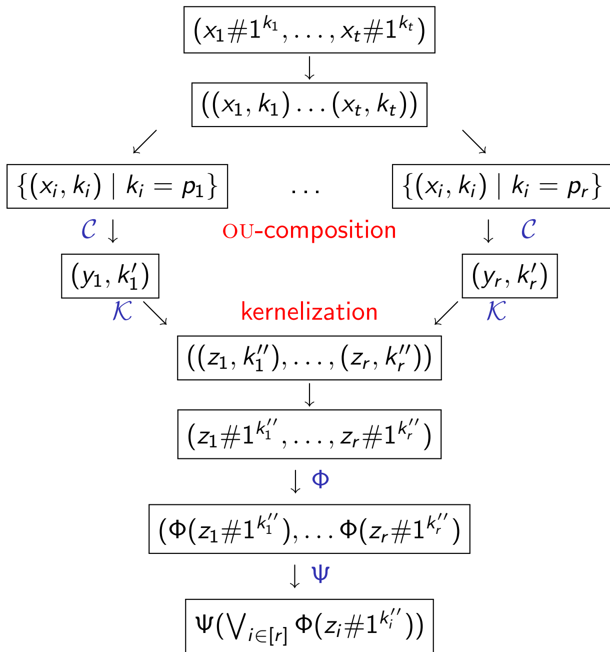
Soit κ-Q un problème paramétré OU-composable tel que le problème $\tilde{Q}$ (non-paramétré) est NP-Complet.
Si κ-Q admet un noyau polynomial, alors $\tilde{Q}$ est OU-distillable.

- $(x, \kappa(x))$ instance de κ-Q $\longrightarrow$ $x\#1^k$ instance de $\tilde{Q}$

- Puisque $\tilde{Q}$ est un problème NP-Complet, il existe deux transformations polynomiales

$$\Phi : \tilde{Q} \longrightarrow \text{SAT} \qquad \Psi : \text{SAT} \longrightarrow \tilde{Q}$$

- on va construire un algorithme $\mathcal{A}$ de OU-distillation pour $\tilde{Q}$ à partir de Φ, Ψ, de l'algorithme de OU-composition $\mathcal{C}$ et de l'algorithme $\mathcal{K}$ calculant le noyau de (Q, κ).


$(x_1\#1^{k_1}, \ldots, x_t\#1^{k_t})$
$((x_1, k_1) \ldots (x_t, k_t))$
$\{(x_i, k_i) \mid k_i = p_1\}$
$\ldots$
$\{(x_i, k_i) \mid k_i = p_r\}$
$\mathcal{C}$
OU-composition
$\mathcal{C}$
(y_1, k_1')
(y_r, k_r')
$\mathcal{K}$
kernelization
$\mathcal{K}$
$((z_1, k_1''), \ldots, (z_r, k_r''))$
$(z_1\#1^{k_1''}, \ldots, z_r\#1^{k_r''})$
Φ
$(\Phi(z_1\#1^{k_1''}), \ldots \Phi(z_r\#1^{k_r''})$
Ψ
$\Psi(\bigvee_{i\in[r]} \Phi(z_i\#1^{k_i''}))$

L'algorithme décrit est un algorithme de OU-distillation pour $\tilde{Q}$.

- $\Psi(\bigvee_{i \in [r]}(\Phi(z_i \# 1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1 \# 1^{k_1}) \in \tilde{Q}$

L'algorithme décrit est un algorithme de OU-distillation pour $\tilde{Q}$.

- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en $\sum_{i\in[t]} |x_i|$

L'algorithme décrit est un algorithme de OU-distillation pour $\tilde{Q}$.

- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en $\sum_{i\in[t]} |x_i|$
- Il reste à prouver que la taille de l'instance de $\tilde{Q}$ retournée est polynomiale en $n = \max_{i\in[t]} |x_i\#1^{k_i}|$

L'algorithme décrit est un algorithme de OU-distillation pour $\tilde{Q}$.

- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en $\sum_{i\in[t]} |x_i|$
- Il reste à prouver que la taille de l'instance de $\tilde{Q}$ retournée est polynomiale en $n = \max_{i\in[t]} |x_i\#1^{k_i}|$
  - $r \leqslant k = \max_{i\in[r]} k_r \leqslant n$

L'algorithme décrit est un algorithme de OU-distillation pour $\tilde{Q}$.

- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en $\sum_{i\in[t]} |x_i|$
- Il reste à prouver que la taille de l'instance de $\tilde{Q}$ retournée est polynomiale en $n = \max_{i\in[t]} |x_i\#1^{k_i}|$
  - $r \leqslant k = \max_{i\in[r]} k_r \leqslant n$
  - $\forall i \in [r],\ k_i'$ est borné par un polynôme en $k_i \leqslant k \leqslant n$ ($\mathcal{C}$ est un algorithme de OU-composition)

L'algorithme décrit est un algorithme de OU-distillation pour $\tilde{Q}$.

- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en $\sum_{i\in[t]} |x_i|$
- Il reste à prouver que la taille de l'instance de $\tilde{Q}$ retournée est polynomiale en $n = \max_{i\in[t]} |x_i\#1^{k_i}|$
  - $r \leqslant k = \max_{i\in[r]} k_r \leqslant n$
  - $\forall i \in [r],\ k_i'$ est borné par un polynôme en $k_i \leqslant k \leqslant n$ ($\mathcal{C}$ est un algorithme de OU-composition)
  - Donc pour tout $i \in [r]$, la taille de $z_i\#1^{k_i''}$ est bornée par un polynôme en n ($\mathcal{K}$ est une kernalization)

L'algorithme décrit est un algorithme de OU-distillation pour $\tilde{Q}$.

- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en $\sum_{i\in[t]} |x_i|$
- Il reste à prouver que la taille de l'instance de $\tilde{Q}$ retournée est polynomiale en $n = \max_{i\in[t]} |x_i\#1^{k_i}|$
  - $r \leqslant k = \max_{i\in[r]} k_r \leqslant n$
  - $\forall i \in [r],\ k_i'$ est borné par un polynôme en $k_i \leqslant k \leqslant n$ ($\mathcal{C}$ est un algorithme de OU-composition)
  - Donc pour tout $i \in [r]$, la taille de $z_i\#1^{k_i''}$ est bornée par un polynôme en n ($\mathcal{K}$ est une kernalization)
  - Donc la taille de $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''}))$ est bornée par un polynôme en n (Φ et Ψ sont des transformations polynomiales)

Conséquences

Corollaire

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

Conséquences

Corollaire

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

Sous-arborescence avec k feuilles

Etant donné un graphe orienté $D = (V, \overrightarrow{E})$, existe-t-il une arborescence $\overrightarrow{T}$ dans D avec k feuilles ? $\longrightarrow$ algo en $O(4^k n^{O(1)})$

Conséquences

Corollaire

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

Sous-arborescence avec k feuilles

Etant donné un graphe orienté $D = (V, \overrightarrow{E})$, existe-t-il une arborescence $\overrightarrow{T}$ dans D avec k feuilles ?

$\longrightarrow$ algo en $O(4^k n^{O(1)})$

Lemme

Sauf si $PH = \Sigma_p^3$, le problème k-SOUS-ARBORESCENCE n'admet pas de noyau polynomial.

Conséquences

Corollaire

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

Sous-arborescence avec k feuilles

Etant donné un graphe orienté $D = (V, \overrightarrow{E})$, existe-t-il une arborescence $\overrightarrow{T}$ dans D avec k feuilles ? $\longrightarrow$ algo en $O(4^k n^{O(1)})$

Lemme

Sauf si $PH = \Sigma_p^3$, le problème k-SOUS-ARBORESCENCE n'admet pas de noyau polynomial.

Remarque La version enracinée (on fixe une racine r) admet un noyau cubique !

Transformations polynomiales et paramétrées

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

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

On note : $\pi\text{-}\mathbf{P} \leqslant_{TPP} \kappa\text{-}\mathbf{Q}$

Transformations polynomiales et paramétrées

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

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

On note : π-**P** $\leqslant_{TPP}$ κ-**Q**

Théorème [Bodlaender, Thomassé, Yeo]
Soient π-**P** et κ-**Q** deux problèmes paramétrés tels que P est NP-Complet et Q appartient à NP.
Si π-**P** $\leqslant_{TPP}$ κ-**Q** et si π-**P** n'admet pas de noyau polynomial, alors κ-**Q** n'admet pas de noyau polynomial.

Exercice : Faire la preuve du théorème précédent.

(**Idée :** *en supposant que* (Q, κ) *admet un noyau polynomial, alors on construit un algorithme polynomial qui réduit* (P, π) *à un noyau polynomial.*)

Exercice : Faire la preuve du théorème précédent.

Utilisation

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

Exercice : Faire la preuve du théorème précédent.

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

Exercice : Faire la preuve du théorème précédent.

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 !

Exercice : Faire la preuve du théorème précédent.

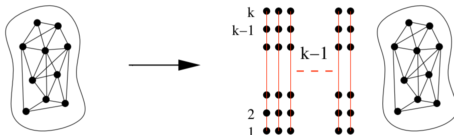
Utilisation

- 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

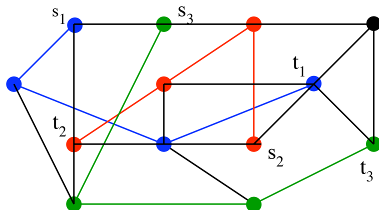
Remarque : PATH-PACKING n'est pas OU-composable !



LONGEST PATH $\leqslant_{TPP}$ PATH PACKING

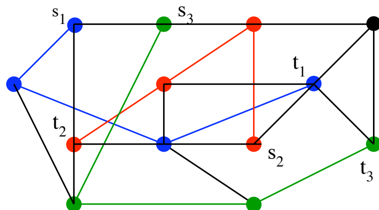
Disjoint Paths

- Un graphe G et k paires de sommets $(s_1, t_1), \ldots (s_k, t_k)$
- G contient-il k chemins $P_1, \ldots P_k$, sommets disjoints tels que P_i est un chemin entre s_i et t_i $(i \in [k])$?



Disjoint Paths

- Un graphe G et k paires de sommets $(s_1, t_1), \dots (s_k, t_k)$
- G contient-il k chemins $P_1, \dots P_k$, sommets disjoints tels que P_i est un chemin entre s_i et t_i $(i \in [k])$?



Remarque

- Disjoint Paths est FPT et NPC [Roberston et Seymour]
- Mais Disjoint Paths n'est pas ou-composable

Disjoint Paths

Méthode

- Introduction d'un problème intermédiaire π-**P**
- Montrer que π-**P** est FPT et OU-composable, que P (non-paramétré) est NP-Complet
- Montrer que π-**P** $\leqslant_{TPP}$ Disjoint Paths

Disjoint Paths

Méthode

- Introduction d'un problème intermédiaire π-**P**
- Montrer que π-**P** est FPT et OU-composable, que P (non-paramétré) est NP-Complet
- Montrer que π-**P** $\leqslant_{TPP}$ Disjoint Paths

Disjoint Factors

Un mot W sur $\Sigma = \{1, \ldots k\}$ possède la propriété des facteurs disjoints si W contient k facteurs disjoints $F_1, \ldots F_k$ tels que $\forall i \in [k]$, F_i commence et termine par la lettre i et $|F_i| \geqslant 2$.

Disjoint Paths

Méthode

- Introduction d'un problème intermédiaire π-**P**
- Montrer que π-**P** est FPT et OU-composable, que P (non-paramétré) est NP-Complet
- Montrer que π-**P** $\leqslant_{TPP}$ Disjoint Paths

Disjoint Factors

Un mot W sur $\Sigma = \{1, \ldots k\}$ possède la propriété des facteurs disjoints si W contient k facteurs disjoints $F_1, \ldots F_k$ tels que $\forall i \in [k]$, F_i commence et termine par la lettre i et $|F_i| \geqslant 2$.

$$1 \, \boxed{2\,4\,3\,2} \, 4 \, \boxed{3\,1\,3} \, 2 \, \boxed{4\,2\,3\,4} \, 2\,3 \, \boxed{1\,4\,1}$$

Disjoint Paths

Exercice

1. Montrer que Disjoint Factors est FPT
   (idée : par programmation dynamique en temps $O(nk \cdot 2^k)$)
2. Montrer que Disjoint Factors est NP-Complet
   (idée : réduction depuis 3-sat)

Disjoint Paths

Exercice

1. Montrer que Disjoint Factors est FPT
   (idée : par programmation dynamique en temps $O(nk \cdot 2^k)$)
2. Montrer que Disjoint Factors est NP-Complet
   (idée : réduction depuis 3-SAT)

Lemme [BTY] : Disjoint Factors est OU-composable

Disjoint Paths

Exercice

1. Montrer que Disjoint Factors est FPT
   (idée : par programmation dynamique en temps $O(nk \cdot 2^k)$)
2. Montrer que Disjoint Factors est NP-Complet
   (idée : réduction depuis 3-SAT)

Lemme [BTY] : Disjoint Factors est OU-composable

$$\boxed{((W_1,\ k)\ ,\ (W_2,\ k))}$$

$$\boxed{((k+1).W_1.(k+1).W_2.(k+1),\ (k+1))}$$

Disjoint Paths

Exercice

1. Montrer que Disjoint Factors est FPT
   (idée : par programmation dynamique en temps $O(nk \cdot 2^k)$)
2. Montrer que Disjoint Factors est NP-Complet
   (idée : réduction depuis 3-SAT)

Lemme [BTY] : Disjoint Factors est OU-composable

$$((W_1,\ k)\ ,\ (W_2,\ k)\ ,\ (W_3,\ k)\ ,\ (W_4,\ k))$$

$$\swarrow \qquad \searrow$$

$$(k+1)W_1(k+1)W_2(k+1) \qquad (k+1)W_3.(k+1)W_4(k+1)$$

$$\searrow \qquad \swarrow$$

$$(k+2)(k+1)W_1(k+1)W_2(k+1)(k+2)(k+1)W_3(k+1)W_4(k+1)(k+2)$$

Disjoint Paths

Exercice

1. Montrer que Disjoint Factors est FPT
   (idée : par programmation dynamique en temps $O(nk \cdot 2^k)$)
2. Montrer que Disjoint Factors est NP-Complet
   (idée : réduction depuis 3-SAT)

Lemme [BTY] : Disjoint Factors est OU-composable

$$((W_1, k) \dots (W_t, k)) \longrightarrow (W', k + \lceil \log_2 t \rceil)$$

Remarques :

DISJOINT PATHS

Exercice

1. Montrer que DISJOINT FACTORS est FPT
   (idée : par programmation dynamique en temps $O(nk \cdot 2^k)$)
2. Montrer que DISJOINT FACTORS est NP-Complet
   (idée : réduction depuis 3-SAT)

Lemme [BTY] : DISJOINT FACTORS est OU-composable

$$((W_1, k) \dots (W_t, k)) \longrightarrow (W', k + \lceil \log_2 t \rceil)$$

Remarques :

- $t \leqslant 2^k$, sinon le problème peut se résoudre en temps polynomial car $2^k \leqslant \sum_1^t |W_i|$.

Disjoint Paths

Exercice

1. Montrer que Disjoint Factors est FPT
   (idée : par programmation dynamique en temps $O(nk \cdot 2^k)$)
2. Montrer que Disjoint Factors est NP-Complet
   (idée : réduction depuis 3-SAT)

Lemme [BTY] : Disjoint Factors est OU-composable

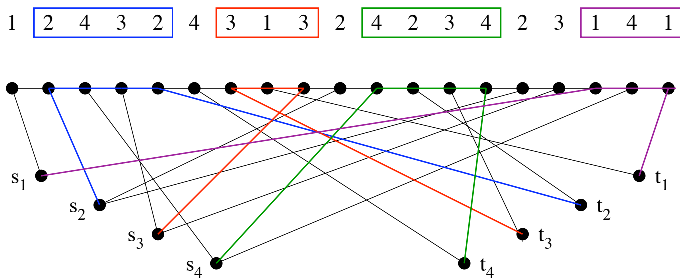
$$((W_1, k) \ldots (W_t, k)) \longrightarrow (W', k + \lceil \log_2 t \rceil)$$

Remarques :

- $t \leqslant 2^k$, sinon le problème peut se résoudre en temps polynomial car $2^k \leqslant \sum_1^t |W_i|$.
- $k + \lceil \log_2 t \rceil \leqslant 2k$

Disjoint Paths

Lemme [BTY] : Disjoint Factors $\leqslant_{TPP}$ Disjoint Paths



Théorème [Bodlaender, Thomassé, Yeo]

Disjoint Paths n'admet pas de noyau polynomial à moins que $PH = \Sigma_p^3$

Algorithmes paramétrés

- Arbre de recherche bornée - VERTEX COVER
- Compression itérative - FEEDBACK VERTEX SET
- Algorithmes randomisés - FEEDBACK VERTEX SET
- Color Coding - LONGEST PATH

Kernelization - Bornes inférieures

- Algorithmes de OU-composition
- Transformation polynomiales paramétrées

Kernelization - Bornes supérieures

- VERTEX COVER et programmation linéaire
- Un noyau linéaire pour FAST à l'aide de couplages

Vertex Cover et programmation linéaire

Soit un graphe $G = (V, E)$. Alors le programme linéaire $L_{vc}(G)$ a une solution optimale semi-entière.

$$L_{vc}(G) = \min \sum_{v \in V} x_v \text{ tel que } \begin{cases} x_u + x_v \geqslant 1 & \forall uv \in E \\ 0 \leqslant x_v \leqslant 1 & \forall v \in V \end{cases}$$

VERTEX COVER et programmation linéaire

Soit un graphe $G = (V, E)$. Alors le programme linéaire $L_{vc}(G)$ a une solution optimale semi-entière.

$$L_{vc}(G) = \min \sum_{v \in V} x_v \text{ tel que } \begin{cases} x_u + x_v \geqslant 1 & \forall uv \in E \\ 0 \leqslant x_v \leqslant 1 & \forall v \in V \end{cases}$$

Soit $(x_v)_{v \in V}$ une solution optimale demi-entière de $L_{vc}(G)$. Pour $r \in \{0, \frac{1}{2}, 1\}$, on note

- $V_r = \{v \in V \mid x_v = r\}$ et
- $G_r = G[V_r]$.

Vertex Cover et programmation linéaire

Lemme

Soient un graphe $G = (V, E)$ et $(x_v)_{v \in V}$ une solution optimale demi-entière de $L_{vc}(G)$. Alors

1. $VC(G_{\frac{1}{2}}) \geqslant |V_{\frac{1}{2}}| \ / \ 2$
2. $VC(G_{\frac{1}{2}}) = VC(G) - |V_1|$

Vertex Cover et programmation linéaire

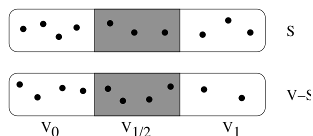
Lemme

Soient un graphe $G = (V, E)$ et $(x_v)_{v \in V}$ une solution optimale demi-entière de $L_{vc}(G)$. Alors

1. $VC(G_{\frac{1}{2}}) \geqslant |V_{\frac{1}{2}}| \ / \ 2$
2. $VC(G_{\frac{1}{2}}) = VC(G) - |V_1|$

Preuve

(i) Si S est un V.C. de G, alors $S_r = S \cap V_r$ est un V.C. de G_r.



Vertex Cover et programmation linéaire

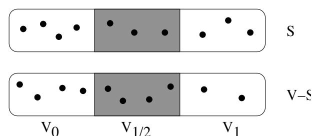
Lemme

Soient un graphe $G = (V, E)$ et $(x_v)_{v \in V}$ une solution optimale demi-entière de $L_{vc}(G)$. Alors

1. $VC(G_{\frac{1}{2}}) \geqslant |V_{\frac{1}{2}}| \ / \ 2$
2. $VC(G_{\frac{1}{2}}) = VC(G) - |V_1|$

Preuve

(i) Si S est un V.C. de G, alors $S_r = S \cap V_r$ est un V.C. de G_r.



(ii) Si S' est un V.C. de $G_{\frac{1}{2}}$, alors $S' \cup V_1$ est un V.C. de G.

VERTEX COVER et programmation linéaire

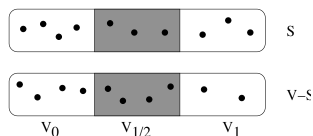
Lemme

Soient un graphe $G = (V, E)$ et $(x_v)_{v \in V}$ une solution optimale demi-entière de $L_{vc}(G)$. Alors

1. $VC(G_{\frac{1}{2}}) \geqslant |V_{\frac{1}{2}}| \ / \ 2$
2. $VC(G_{\frac{1}{2}}) = VC(G) - |V_1|$

Preuve

(i) Si S est un V.C. de G, alors $S_r = S \cap V_r$ est un V.C. de G_r.



(ii) Si S' est un V.C. de $G_{\frac{1}{2}}$, alors $S' \cup V_1$ est un V.C. de G.

$$|S'| + |V_1| \geqslant VC(G) \geqslant \sum_{v \in V} x_v = \frac{1}{2}|V_{\frac{1}{2}}| + |V_1|$$

Vertex Cover et programmation linéaire

Théorème : k-Vertex Cover possède un kernel de taille au plus $2k$.

VERTEX COVER et programmation linéaire

Théorème : k-VERTEX COVER possède un kernel de taille au plus $2k$.

Preuve

- $k - |V_1| < 0$: G ne possède pas de VC de taille k

Vertex Cover et programmation linéaire

Théorème : k-Vertex Cover possède un kernel de taille au plus $2k$.

Preuve

- $k - |V_1| < 0$: G ne possède pas de VC de taille k
- $k - |V_1| = 0$: Si $G_{\frac{1}{2}}$ possède une arête alors G ne possède pas de solution sinon G possède un VC de taille k

Vertex Cover et programmation linéaire

Théorème : k-Vertex Cover possède un kernel de taille au plus $2k$.

Preuve

- $k - |V_1| < 0$: G ne possède pas de VC de taille k
- $k - |V_1| = 0$: Si $G_{\frac{1}{2}}$ possède une arête alors G ne possède pas de solution sinon G possède un VC de taille k
- $k - |V_1| > 0$ et $|V_{\frac{1}{2}}| > 2(k - |V_1|)$: $G_{\frac{1}{2}}$ ne possède pas de VC de taille $k - |V_1| \Rightarrow G$ ne possède pas de VC de taille k.

Vertex Cover et programmation linéaire

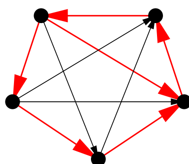
Théorème : k-Vertex Cover possède un kernel de taille au plus $2k$.

Preuve

- $k - |V_1| < 0$: G ne possède pas de VC de taille k
- $k - |V_1| = 0$: Si $G_{\frac{1}{2}}$ possède une arête alors G ne possède pas de solution sinon G possède un VC de taille k
- $k - |V_1| > 0$ et $|V_{\frac{1}{2}}| > 2(k - |V_1|)$: $G_{\frac{1}{2}}$ ne possède pas de VC de taille $k - |V_1| \Rightarrow G$ ne possède pas de VC de taille k.
- Sinon, le noyau est l'instance $(G_{\frac{1}{2}}, k - |V_1|)$.

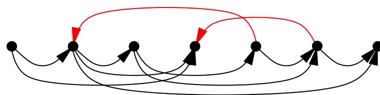
FAST

Observation : un tournoi est transitif (ou acyclique) ssi il ne contient pas de triangle (orienté).



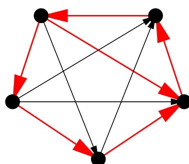
Définitions / terminologie

- On note $T_\sigma = (V, A, \sigma)$ un tournoi dont les sommets sont ordonnés selon une permutation σ



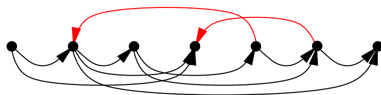
FAST

Observation : un tournoi est transitif (ou acyclique) ssi il ne contient pas de triangle (orienté).



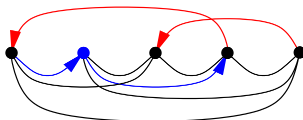
Définitions / terminologie

- On note $T_\sigma = (V, A, \sigma)$ un tournoi dont les sommets sont ordonnés selon une permutation σ



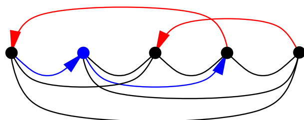
- Un arc $\overrightarrow{uv}$ de T_σ est un arc retour si $u <_\sigma v$.
- Soit $\overrightarrow{uv}$ un arc retour $\quad span(\overrightarrow{uv}) = \{w \in V : u <_\sigma w <_\sigma v\}$

FAST - certificats

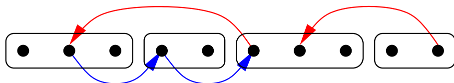


- Si $\overrightarrow{uv}$ est un arc retour de T_σ et $w \in span(\overrightarrow{uv})$ n'est incident à aucun arc retour, $c(\overrightarrow{uv}) = \{u, w, v\}$ est un certificat pour $\overrightarrow{uv}$
- Si F est un ensemble d'arcs retours, un F-certificat est un ensemble $c(F) = \{c(f) : f \in F\}$ de certificats arc-disjoints.

FAST - certificats



- Si $\overrightarrow{uv}$ est un arc retour de T_σ et $w \in span(\overrightarrow{uv})$ n'est incident à aucun arc retour, $c(\overrightarrow{uv}) = \{u, w, v\}$ est un certificat pour $\overrightarrow{uv}$
- Si F est un ensemble d'arcs retours, un F-certificat est un ensemble $c(F) = \{c(f) : f \in F\}$ de certificats arc-disjoints.
- Une partition ordonnée $\mathcal{P} = \{V_1, \ldots V_l\}$ d'un tournoi T_σ est saine si l'ensemble F des arcs retours externes peut être certifié uniquement avec des arcs externes.



FAST - règles de réduction

1. [Sommet inutile] Supprimer les sommets n'appartenant à aucun triangle orienté

2. [Partition saine] Si $\mathcal{P}$ est une partition saine de T_σ, retourner les arcs retours externes et décrémenter k en conséquence.

FAST - règles de réduction

1. [Sommet inutile] Supprimer les sommets n'appartenant à aucun triangle orienté

2. [Partition saine] Si $\mathcal{P}$ est une partition saine de T_σ, retourner les arcs retours externes et décrémenter k en conséquence.

- Comment calculer une partition saine en temps polynomial ?
- Montrer que ces deux règles permettent d'obtenir un noyau de taille $4k$

FAST - conflict packing

Un conflict packing est un ensemble maximal $\mathcal{C}$ de certificats arcs disjoints. On note $V(\mathcal{C})$ les sommets couverts par $\mathcal{C}$.

FAST - conflict packing

Un conflict packing est un ensemble maximal $\mathcal{C}$ de certificats arcs disjoints. On note $V(\mathcal{C})$ les sommets couverts par $\mathcal{C}$.

Lemme 1

Si $\mathcal{C}$ est un conflict packing d'une instance positive (T, k) de FAST, alors $|V(\mathcal{C})| \leqslant 3k$

FAST - conflict packing

Un conflict packing est un ensemble maximal $\mathcal{C}$ de certificats arcs disjoints. On note $V(\mathcal{C})$ les sommets couverts par $\mathcal{C}$.

Lemme 1

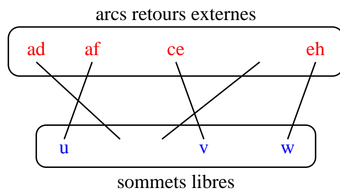
Si $\mathcal{C}$ est un conflict packing d'une instance positive (T, k) de FAST, alors $|V(\mathcal{C})| \leqslant 3k$

Lemme 2

Si $\mathcal{C}$ est un conflict packing d'une instance (T, k) de FAST, alors $\exists$ une permutation σ tq $\forall \overrightarrow{uv}$ arc retour de T_σ, $\{u, v\} \subseteq V(\mathcal{C})$

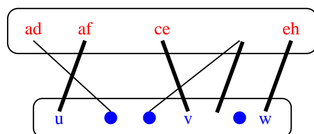
FAST

Lemme 3 : Si le tournoi T possède au moins $4k$ sommets, alors on peut calculer une partition saine en temps polynomial.



FAST

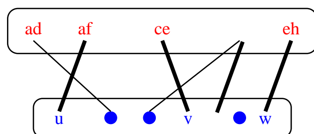
Lemme 3 : Si le tournoi T possède au moins $4k$ sommets, alors on peut calculer une partition saine en temps polynomial.



- il existe un couplage, et donc un vertex cover S de taille k
- $\mathcal{P}$ est la partition obtenue en isolant les sommets libres n'appartenant pas à S (il en existe car $|V| > 4k$).

FAST

Lemme 3 : Si le tournoi T possède au moins $4k$ sommets, alors on peut calculer une partition saine en temps polynomial.

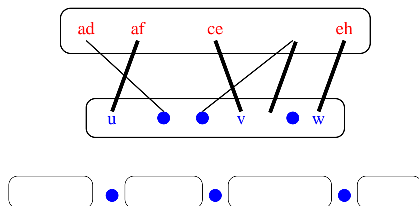


- il existe un couplage, et donc un vertex cover S de taille k
- $\mathcal{P}$ est la partition obtenue en isolant les sommets libres n'appartenant pas à S (il en existe car $|V| > 4k$).

$\mathcal{P}$ est une partition saine.

FAST

Lemme 3 : Si le tournoi T possède au moins $4k$ sommets, alors on peut calculer une partition saine en temps polynomial.



- il existe un couplage, et donc un vertex cover S de taille k
- $\mathcal{P}$ est la partition obtenue en isolant les sommets libres n'appartenant pas à S (il en existe car $|V| > 4k$).

$\mathcal{P}$ est une partition saine.

Théorème : FAST (paramétré par la taille k de la solution) admet un noyau de taille au plus $4k$

A venir

- Décomposition arborescente et programmation dynamique