Complexité paramétrée (3)

Recherche de noyaux - bornes inférieures

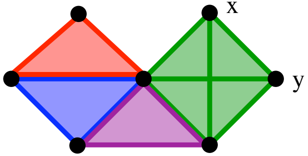
Christophe PAUL
(CNRS - LIRMM)

Observation

Le théorème prouvant, pour un problème paramétré (Q, κ), l'équivalence entre l'existence d'un noyau et son appartenance à la classe de complexité FPT, ne permet que de déduire des noyaux de tailles exponentiels.

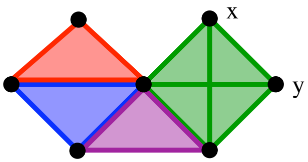
- Peut-on démontrer que certains problèmes n'admettent pas de noyau polynomial ?

L'exemple de 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 ?

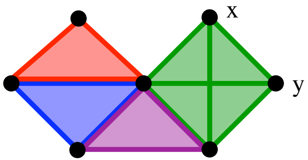
L'exemple de Edge Clique-Cover



Théorème [Gramm, Guo, Hüffner, Niedermeier]

Edge Clique-Cover admet un noyau de taille 2^k sommets.

L'exemple de EDGE CLIQUE-COVER



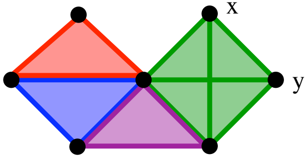
Théorème [Gramm, Guo, Hüffner, Niedermeier]

EDGE CLIQUE-COVER admet un noyau de taille 2^k sommets.

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



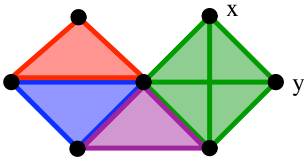
Théorème [Gramm, Guo, Hüffner, Niedermeier]
EDGE CLIQUE-COVER admet un noyau de taille 2^k sommets.

Preuve : Supposons qu'il existe k cliques $C_1 \ldots C_k$ 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



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

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

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

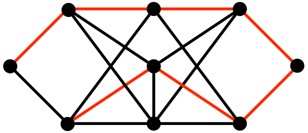
Observation : $\boxed{\forall i \in [k]\ C[x, i] = C[y, i] \text{ ssi } N[x] = N[y]}$

Problème ouvert

$\boxed{\text{Edge Clique-Cover admet-il un noyau polynomial ?}}$

Longest Path

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

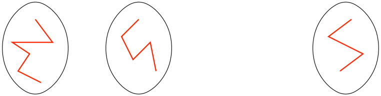


Longest Path est NP-Complet (cf. chemin hamiltonien) mais peut-être résolu en temps $O(c^k.n^{O(1)})$ grâce à Color Coding.

Longest Path

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

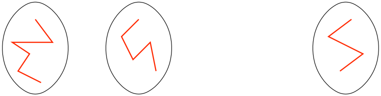
- construisons une instance (G, k) avec t instances différentes
  $$(G, k) = (G_1, k) \oplus (G_2, k) \oplus \ \ldots \ \oplus (G_t, k)$$



Longest Path

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

- 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 que $\mathcal{A}$ identifie en temps polynomial quel G_i, parmi les 2^{2^n} possibles, 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 $x_i \in \Sigma^*$, $\forall i \in [t]$;

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 $(x_1, \ldots x_t)$, 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 $\max_{i \in [t]} |x_i|$.

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) 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 $\max_{i \in [t]} |x_i|$.

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 $(x_i, k) \in \Sigma^* \times \mathbb{N}$ $\forall i \in [t]$;

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 $\sum_{i=1}^{t} |x_i| + k$,

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.

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.

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

Théorème [Bodlaender, Downey, Fellows, Hermelin]

Soit $(Q, \kappa) \in \Sigma^* \times \mathbb{N}$ un problème paramétré $\textsc{ou}$-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 $\textsc{ou}$-distillable.

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

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

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.

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

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

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

- 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).

$$
\begin{tikzcd}[column sep=small]
 & \boxed{(x_1\#1^{k_1},\dots,x_t\#1^{k_t})} \arrow[d] & \\
 & \boxed{((x_1,k_1)\dots(x_t,k_t))} \arrow[dl] \arrow[dr] & \\
\boxed{\{(x_i,k_i)\mid k_i=p_1\}} \arrow[d, "\mathcal{C}"'] & \dots & \boxed{\{(x_i,k_i)\mid k_i=p_r\}} \arrow[d, "\mathcal{C}"] \\
\boxed{(y_1,k'_1)} \arrow[dr, "\mathcal{K}"'] & & \boxed{(y_r,k'_r)} \arrow[dl, "\mathcal{K}"] \\
 & \boxed{((z_1,k''_1),\dots,(z_r,k''_r))} \arrow[d] & \\
 & \boxed{(z_1\#1^{k''_1},\dots,z_r\#1^{k''_r})} \arrow[d, "\Phi"] & \\
 & \boxed{(\Phi(z_1\#1^{k''_1}),\dots\Phi(z_r\#1^{k''_r}))} \arrow[d, "\Psi"] & \\
 & \boxed{\Psi(\bigvee_{i\in[r]}\Phi(z_i\#1^{k''_i}))} &
\end{tikzcd}
$$

OU-composition

kernelization

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.

$\boxed{\text{Sous-arborescence avec } k \text{ feuilles}}$

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.

$\boxed{\text{Sous-arborescence avec } k \text{ feuilles}}$

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.

$\boxed{\text{Sous-arborescence avec } k \text{ feuilles}}$

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

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:

$$\boxed{\textsc{Path Packing}}$$

$\longrightarrow$ 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.

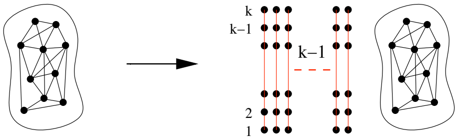
Utilisation

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

$\boxed{\textsc{Path Packing}}$

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

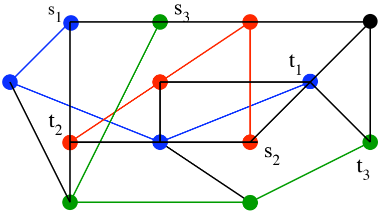
Remarque : PATH-PACKING n'est pas OU-composable !



$\boxed{\textsc{Longest Path} \leqslant_{TPP} \textsc{Path Packing}}$

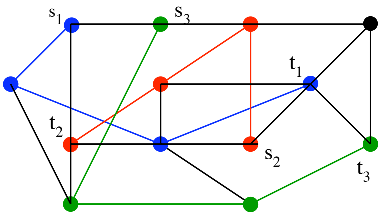
Disjoint Paths

- 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 (P, π)
- Montrer que (P, π) est FPT et OU-composable, que P (non-paramétré) est NP-Complet
- Montrer que $(P, \pi) \leqslant_{TPP}$ Disjoint Paths

Disjoint Paths

Méthode

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

$\boxed{\text{DISJOINT FACTORS}}$

Un mot W sur $\Sigma = \{1, \dots k\}$ possède la propriété des facteurs disjoints si W contient k facteurs disjoints $F_1, \dots F_k$ 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

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

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

$$\boxed{((W_1,\ k)\ ,\ (W_2,\ k)\ ,\ (W_3,\ k)\ ,\ (W_4,\ k))}$$

$$\swarrow \qquad\qquad\qquad\qquad \searrow$$

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

$$\searrow \qquad\qquad\qquad\qquad \swarrow$$

$$\boxed{(k+2)(k+1)W_1(k+1)W_2(k+1)(k+2)(k+1)W_3(k+1)W_4(k+1)(k+2)}$$

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

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

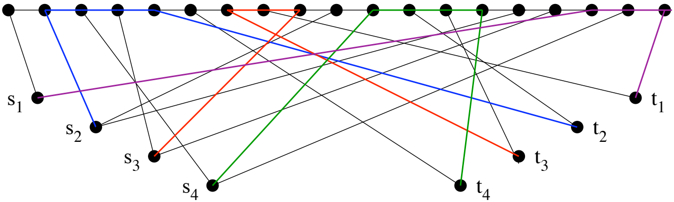
Remarques:

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

Disjoint Paths

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

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



Théorème [Bodlaender, Thomassé, Yeo]

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

Compositions croisées

Idée de base / objectifs

- unifier les deux techniques existantes (Ou-composition et TPP)
- permettre de composer un problème vers un autre problème
- être capable de composer des problèmes NP-complets vers des problèmes FPT

Compositions croisées

Idée de base / objectifs

- unifier les deux techniques existantes (Ou-composition et TPP)
- permettre de composer un problème vers un autre problème
- être capable de composer des problèmes NP-complets vers des problèmes FPT

Une relation d'équivalence $\mathcal{R}$ sur Σ^* est une relation d'équivalence polynomiale si

1. $\exists$ un algorithme qui décide si $x\mathcal{R}y$ en temps $(|x| + |y|)^{O(1)}$
2. $\forall S \subseteq \Sigma^*$, $\mathcal{R}$ partitionne S en au plus $(max_{x \in S}|X|)^{O(1)}$ classes

Compositions croisées

Idée de base / objectifs

- unifier les deux techniques existantes (Ou-composition et TPP)
- permettre de composer un problème vers un autre problème
- être capable de composer des problèmes NP-complets vers des problèmes FPT

Une relation d'équivalence $\mathcal{R}$ sur Σ^* est une relation d'équivalence polynomiale si

1. $\exists$ un algorithme qui décide si $x\mathcal{R}y$ en temps $(|x| + |y|)^{O(1)}$
2. $\forall S \subseteq \Sigma^*$, $\mathcal{R}$ partitionne S en au plus $(max_{x \in S}|X|)^{O(1)}$ classes

- Exemple : nombre de sommets et valeur du paramètres égaux

Compositions croisées

Soient $L \subseteq \Sigma^*$ et $(Q, \kappa) \subseteq \Sigma^* \times \mathbb{N}$. Il existe une composition croisée de L vers Q si il existe une relation d'équivalence polynomiale $\mathcal{R}$ sur Σ^* et un algorithme $\mathcal{A}$ qui

- reçoit une séquence $(x_1, \ldots x_t)$, tel que $\forall 1 \leqslant i < j \leqslant t$, $x_i \mathcal{R} x_j$
- a une complexité polynomiale en $\sum_{i=1}^{t} |x_i|$
- retourne $(x, k) \in \Sigma^* \times \mathbb{N}$ tel que
  1. $(x, k) \in (Q, \kappa) \Leftrightarrow \exists i \in [t], x_i \in L$
  2. k est polynomial en $max_{i=1}^{t}|x_i| + \log t$.

Compositions croisées

Soient $L \subseteq \Sigma^*$ et $(Q, \kappa) \subseteq \Sigma^* \times \mathbb{N}$. Il existe une composition croisée de L vers Q si il existe une relation d'équivalence polynomiale $\mathcal{R}$ sur Σ^* et un algorithme $\mathcal{A}$ qui

- reçoit une séquence $(x_1, \dots x_t)$, tel que $\forall 1 \leqslant i < j \leqslant t$, $x_i \mathcal{R} x_j$
- a une complexité polynomiale en $\sum_{i=1}^{t} |x_i|$
- retourne $(x, k) \in \Sigma^* \times \mathbb{N}$ tel que
  1. $(x, k) \in (Q, \kappa) \Leftrightarrow \exists i \in [t], x_i \in L$
  2. k est polynomial en $max_{i=1}^{t} |x_i| + \log t$.

Théorème [Bodlaender, Jansen, Kratsch]
Soit $\mathcal{A}$ une composition croisée d'un problème NP-complet $L \subseteq \Sigma^*$ vers un problème paramétré (Q, κ) tel que $\tilde{Q} \in$ NP-complet. Si (Q, κ) admet un noyau polynomial, alors L admet une OU-distillation.

Compositions croisées

Soient $L \subseteq \Sigma^*$ et $(Q, \kappa) \subseteq \Sigma^* \times \mathbb{N}$. Il existe une composition croisée de L vers Q si il existe une relation d'équivalence polynomiale $\mathcal{R}$ sur Σ^* et un algorithme $\mathcal{A}$ qui

- reçoit une séquence $(x_1, \ldots x_t)$, tel que $\forall 1 \leqslant i < j \leqslant t$, $x_i \mathcal{R} x_j$
- a une complexité polynomiale en $\sum_{i=1}^{t} |x_i|$
- retourne $(x, k) \in \Sigma^* \times \mathbb{N}$ tel que
  1. $(x, k) \in (Q, \kappa) \Leftrightarrow \exists i \in [t], x_i \in L$
  2. k est polynomial en $max_{i=1}^{t} |x_i| + \log t$.

Théorème [Bodlaender, Jansen, Kratsch]
Soit $\mathcal{A}$ une composition croisée d'un problème NP-complet $L \subseteq \Sigma^*$ vers un problème paramétré (Q, κ) tel que $\tilde{Q} \in$ NP-complet. Si (Q, κ) admet un noyau polynomial, alors L admet une OU-distillation.

- preuve similaire à celle de la OU-composition

Compositions croisées

$$\begin{tikzcd}
 & \boxed{(x_1\#1^{k_1},\dots,x_t\#1^{k_t})} \arrow[dl] \arrow[dr] \arrow[d, phantom, "\mathcal{R}"] & \\
\boxed{X_1 = \{(x_1,k_1^1),\dots\}} \arrow[d, "\mathcal{A}"'] \arrow[r, phantom, "\text{composition croisée}"] & \dots & \boxed{X_r = \{(x_1^r,k_1^R),\dots\}} \arrow[d, "\mathcal{A}"] \\
\boxed{(y_1,k_1')} \arrow[d, "\mathcal{K}"'] & \text{kernelization} & \boxed{(y_r,k_r')} \arrow[d, "\mathcal{K}"] \\
\boxed{(z_1,k_1'')} \arrow[dr] & & \boxed{(z_r,k_r''} \arrow[dl] \\
 & \boxed{(z_1\#1^{k_1''},\dots,z_r\#1^{k_r''})} \arrow[d, "\Phi\,:\,\tilde{Q}\,\longrightarrow\,SAT"] & \\
 & \boxed{(\Phi(z_1\#1^{k_1''}),\dots\Phi(z_r\#1^{k_r''})} \arrow[d, "\Psi\,:\,SAT\,\longrightarrow\,L"] & \\
 & \boxed{z = \Psi(\bigvee_{i\in[r]}\Phi(z_i\#1^{k_i''}))} &
\end{tikzcd}$$

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

- On pose $m = \max_{i \in [t]} |x_i|$ et $t \leqslant (|\Sigma| + 1)^m$ (sinon certaines instances sont des copies que l'on supprime)

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

- On pose $m = \max_{i\in[t]} |x_i|$ et $t \leqslant (|\Sigma| + 1)^m$ (sinon certaines instances sont des copies que l'on supprime)
- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$

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

- On pose $m = \max_{i\in[t]} |x_i|$ et $t \leqslant (|\Sigma| + 1)^m$ (sinon certaines instances sont des copies que l'on supprime)
- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en
$$\sum_{i\in[t]} |x_i|$$

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

- On pose $m = \max_{i\in[t]} |x_i|$ et $t \leqslant (|\Sigma|+1)^m$ (sinon certaines instances sont des copies que l'on supprime)
- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en
$$\sum_{i\in[t]} |x_i|$$
- Il reste à prouver que la taille de l'instance z de $\tilde{Q}$ est polynomiale en $n = \max_{i\in[t]} |x_i\#1^{k_i}|$

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

- On pose $m = \max_{i\in[t]} |x_i|$ et $t \leqslant (|\Sigma| + 1)^m$ (sinon certaines instances sont des copies que l'on supprime)
- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en
$$\sum_{i\in[t]} |x_i|$$
- Il reste à prouver que la taille de l'instance z de $\tilde{Q}$ est polynomiale en $n = \max_{i\in[t]} |x_i\#1^{k_i}|$
  - r est polynomial en m

    $\mathcal{R}$ est une relation d'équivalence polynomiale

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

- On pose $m = \max_{i\in[t]} |x_i|$ et $t \leqslant (|\Sigma| + 1)^m$ (sinon certaines instances sont des copies que l'on supprime)
- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en
$$\sum_{i\in[t]} |x_i|$$
- Il reste à prouver que la taille de l'instance z de $\tilde{Q}$ est polynomiale en $n = \max_{i\in[t]} |x_i\#1^{k_i}|$
    - r est polynomial en m
    $\mathcal{R}$ est une relation d'équivalence polynomiale
    - $\forall i \in [r]$, k_i' est polynomial en $m + \log t$
    $\mathcal{A}$ est une composition croisée

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

- On pose $m = \max_{i\in[t]} |x_i|$ et $t \leqslant (|\Sigma| + 1)^m$ (sinon certaines instances sont des copies que l'on supprime)
- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i \# 1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1 \# 1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en
$$\sum_{i\in[t]} |x_i|$$
- Il reste à prouver que la taille de l'instance z de $\tilde{Q}$ est polynomiale en $n = \max_{i\in[t]} |x_i \# 1^{k_i}|$
    - r est polynomial en m
    $\mathcal{R}$ est une relation d'équivalence polynomiale
    - $\forall i \in [r]$, k_i' est polynomial en $m + \log t$
    $\mathcal{A}$ est une composition croisée
    - $\forall i \in [r]$, $|zi|$ et k_i'' est polynomial en $m + \log t$
    $\mathcal{K}$ est une kernelization

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

- On pose $m = \max_{i\in[t]} |x_i|$ et $t \leqslant (|\Sigma| + 1)^m$ (sinon certaines instances sont des copies que l'on supprime)
- $\Psi(\bigvee_{i\in[r]}(\Phi(z_i\#1^{k_i''})) \in \tilde{Q} \iff \exists i \in [r], (x_1\#1^{k_1}) \in \tilde{Q}$
- La complexité de l'algorithme est polynomiale en
$$\sum_{i\in[t]} |x_i|$$
- Il reste à prouver que la taille de l'instance z de $\tilde{Q}$ est polynomiale en $n = \max_{i\in[t]} |x_i\#1^{k_i}|$
  - r est polynomial en m
    $\mathcal{R}$ est une relation d'équivalence polynomiale
  - $\forall i \in [r]$, k_i' est polynomial en $m + \log t$
    $\mathcal{A}$ est une composition croisée
  - $\forall i \in [r]$, $|zi|$ et k_i'' est polynomial en $m + \log t$
    $\mathcal{K}$ est une kernelization
  - $|z|$ est polynomial en $r \times (m + \log t)$ donc polynomial en m

Corollaire :

Soit $\mathcal{A}$ une composition croisée d'un problème NP-complet $L \subseteq \Sigma^*$ vers un problème paramétré (Q, κ) tel que $\tilde{Q} \in$ NP-complet.
Sauf si $PH = \Sigma_p^3$, (Q, κ) n'admet pas de noyau polynomial.

Corollaire :

Soit $\mathcal{A}$ une composition croisée d'un problème NP-complet $L \subseteq \Sigma^*$ vers un problème paramétré (Q, κ) tel que $\tilde{Q} \in$ NP-complet.
Sauf si $PH = \Sigma_p^3$, (Q, κ) n'admet pas de noyau polynomial.

Observations :

1. Si le problème paramétré (P, κ) est OU-composable, alors il existe une composition croisée de $\tilde{P}$ vers (P, κ).
   - la relation d'équivalence polynomiale regroupe les instances "valides" $x\#1^k$ selon la valeur du paramètre k

Corollaire :

Soit $\mathcal{A}$ une composition croisée d'un problème NP-complet $L \subseteq \Sigma^*$ vers un problème paramétré (Q, κ) tel que $\tilde{Q} \in$ NP-complet.
Sauf si $PH = \Sigma_p^3$, (Q, κ) n'admet pas de noyau polynomial.

Observations :

1. Si le problème paramétré (P, κ) est OU-composable, alors il existe une composition croisée de $\tilde{P}$ vers (P, κ).
   - la relation d'équivalence polynomiale regroupe les instances "valides" $x\#1^k$ selon la valeur du paramètre k

2. Soient (P, κ_P) et (Q, κ_Q) deux problèmes paramétrés tels que (P, κ_P) est OU-composable et $(P, \kappa_P) \leqslant_{TPP} (Q, \kappa_Q)$, alors il existe une composition croisée de $\tilde{P}$ vers (Q, κ_Q).

Dominating set

- Un graphe $G = (V, E)$ contient-il un ensemble dominant $S \subseteq V$ de taille au plus k:

$$\forall x \in V, x \notin S \Rightarrow \exists y \in S, (x, y) \in E$$

Dominating set

- Un graphe $G = (V, E)$ contient-il un ensemble dominant $S \subseteq V$ de taille au plus k:

$$\forall x \in V, x \notin S \Rightarrow \exists y \in S, (x, y) \in E$$

Résultats connus

- $W[2]$-complet si paramétré par la taille de la solution

$\Rightarrow$ pas de noyau exponentiel !

Dominating set

- Un graphe $G = (V, E)$ contient-il un ensemble dominant $S \subseteq V$ de taille au plus k:

$$\forall x \in V, x \notin S \Rightarrow \exists y \in S, (x, y) \in E$$

Résultats connus

- $W[2]$-complet si paramétré par la taille de la solution
  $\Rightarrow$ pas de noyau exponentiel !
- FPT et OU-composable si paramétré par *treewidth*(G)
  $\Rightarrow$ pas de noyau polynomial !

Dominating set

- Un graphe $G = (V, E)$ contient-il un ensemble dominant $S \subseteq V$ de taille au plus k:

$$\forall x \in V, x \notin S \Rightarrow \exists y \in S, (x, y) \in E$$

Résultats connus

- $W[2]$-complet si paramétré par la taille de la solution
  $\Rightarrow$ pas de noyau exponentiel !
- FPT et OU-composable si paramétré par *treewidth*(G)
  $\Rightarrow$ pas de noyau polynomial !
- FPT si paramétré par la taille d'un vertex cover

DOMINATING SET

- Un graphe $G = (V, E)$ contient-il un ensemble dominant $S \subseteq V$ de taille au plus k:

$$\forall x \in V, x \notin S \Rightarrow \exists y \in S, (x, y) \in E$$

Résultats connus

- $W[2]$-complet si paramétré par la taille de la solution
  $\Rightarrow$ pas de noyau exponentiel !
- FPT et OU-composable si paramétré par $treewidth(G)$
  $\Rightarrow$ pas de noyau polynomial !
- FPT si paramétré par la taille d'un vertex cover

MAIS

$$vertex\ cover(G) \geqslant treewidth(G)$$

$\boxed{\exists?\ \text{un noyau polynomial pour la paramétrisation par vertex cover}}$

Dominating set

Théorème [Wratigan]

Il existe une composition croisée de Vertex Cover vers Dominating Set paramétré par vertex cover.

Dominating set

Théorème [Wratigan]

Il existe une composition croisée de Vertex Cover vers Dominating Set paramétré par vertex cover.

Soit $(G_1\#1^{k_1},\dots G_t\#1^{k_t})$ une série d'instances de Vertex Cover

- les deux problèmes sont NP-Complets
- $G_i\#1^{k_i}\ \mathcal{R}\ G_j\#1^{k_j}$ si $|V(G_i)|=|V(G_j)|=n$ et $k_i=k_j=k$

Dominating set

Théorème [Wratigan]
Il existe une composition croisée de Vertex Cover vers Dominating Set paramétré par vertex cover.

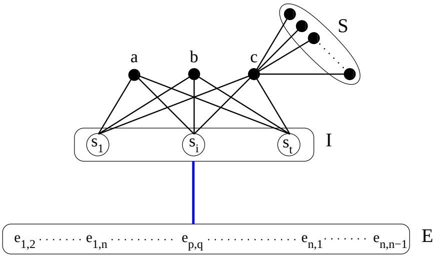
Soit $(G_1\#1^{k_1},\dots G_t\#1^{k_t})$ une série d'instances de Vertex Cover

- les deux problèmes sont NP-Complets
- $G_i\#1^{k_i}\ \mathcal{R}\ G_j\#1^{k_j}$ si $|V(G_i)| = |V(G_j)| = n$ et $k_i = k_j = k$
- On construit une instance (G, k_G, Z) de Dominating Set paramétré par vertex cover avec Z un vertex cover de G tel que

G admet un ensemble dominant de taille k_G

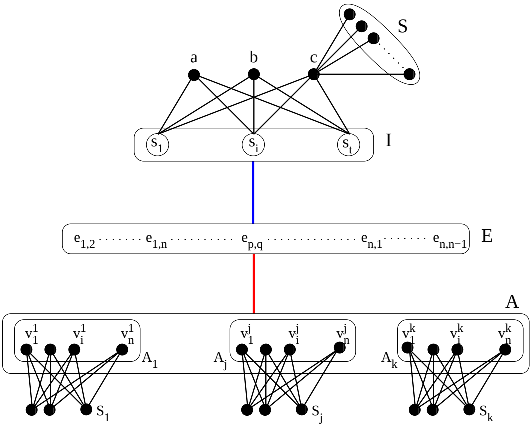
ssi

$\exists j \in [t]$ tel que G_i possède un vertex cover de taille k_j



- $|S| = k + 3$

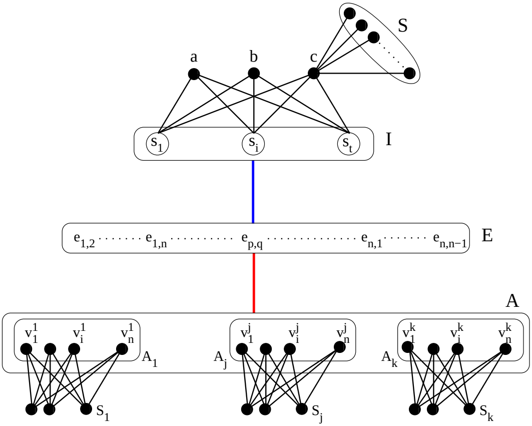
$(s_i, e_{p,q}) \in E(G) \Leftrightarrow (p, q) \notin E(G_i)$



- $|S| = k + 3$
- $\forall j \in [k],\ |S_j| = k + 3$

$(s_i, e_{p,q}) \in E(G) \Leftrightarrow (p,q) \notin E(G_i)$

$(v_i^j, e_{p,q}) \in E(G) \Leftrightarrow i = p \vee i = q$

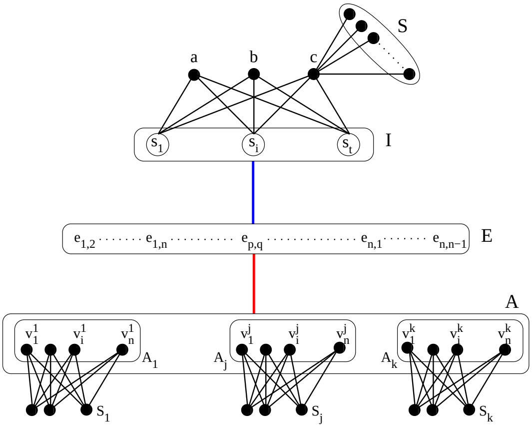


- $|S| = k+3$
- $\forall j \in [k],\ |S_j| = k+3$

$(s_i, e_{p,q}) \in E(G) \Leftrightarrow (p,q) \notin E(G_i)$

$(v_i^j, e_{p,q}) \in E(G) \Leftrightarrow i = p \vee i = q$

G_i possède un vertex cover $S = \{u_1, \dots u_k\}$ de taille k
$\Rightarrow \{c, s_i\} \cup \{v_{u_i}^i : i \in [l]\}$ est un $k+2$-dominant de G



- $|S| = k+3$
- $\forall j \in [k],\ |S_j| = k+3$

$(s_i, e_{p,q}) \in E(G) \Leftrightarrow (p,q) \notin E(G_i)$

$(v^j_i, e_{p,q}) \in E(G) \Leftrightarrow i = p \vee i = q$

G possède un $(k+2)$ dominant
$\Rightarrow \exists i \in [k]$ tel que G_i possède un vertex cover de taille k