

# Cours 7: Complexité paramétrique

- Noyau linéaire pour VC
- Largeur arborescente et programmation dynamique
- Méta-algorithmes
- Calcul d'une décomposition arborescente
- Classes de complexité paramétrique

# Noyau linéaire pour Vertex Cover (VC)

La semaine dernière on a vu:

noyau linéaire de taille  $c \cdot n \Rightarrow$  algo approché à facteur  $c$

En effet, soit  $i$  le nombre de sommets mis dans le couvrant durant une réduction au  $k$ -noyau (calculé en temps polynomial)

- 1er cas:  $i > k$  et on sait que l'optimum vaut au moins  $k$
- 2ème cas, on obtient un couvrant de taille  $i + c \cdot (k - i) < c \cdot k$

**Algorithme:** Essayer  $k = 1, 2, \dots$  (ou par dichotomie) pour trouver le plus petit  $k$  tel qu'on soit dans le 1er cas.

Peut on trouver un noyau linéaire ?

# Noyau linéaire pour VC, par programmation linéaire

Vertex cover de  $G = (V, E)$  sous forme LP:

$$\min \left( \sum_{v \in V} x_v \right) \quad \text{avec} \quad \begin{cases} x_v + x_{v'} \geq 1 & (v, v') \in E \\ x_v \in \{0, 1\} & v \in V \end{cases}$$

Relaxation polynomiale:  $0 \leq x_v \leq 1$ , pour tout  $v \in V$ .

Posons:  $C_0 = \{v \mid x_v^* < \frac{1}{2}\}$ ,  $C_1 = \{v \mid x_v^* > \frac{1}{2}\}$  et  $C_{\frac{1}{2}} = \{v \mid x_v^* = \frac{1}{2}\}$

**Théorème:** Il existe une couverture optimale  $V^*$  avec  $C_1 \subseteq V^* \subseteq C_{\frac{1}{2}} \cup C_1$

On en déduit que  $VC(G) = |V^*| = |C_1| + VC(G_{\frac{1}{2}})$ .

On montre de plus que  $VC(G_{\frac{1}{2}}) \geq \frac{1}{2}|C_{\frac{1}{2}}|$

**Algorithme pour un  $k$ -noyau:** – si  $|C_1| > k$ ,  $VC(G) > k$

– si  $|C_1| = k$ , et  $G_{\frac{1}{2}}$  contient une arête,  $VC(G) > k$

– si  $k - |C_1| > \frac{1}{2}|C_{\frac{1}{2}}|$ , alors  $VC(G) \geq |C_1| + \frac{1}{2}|C_{\frac{1}{2}}| > k$ .

– sinon le noyau  $(G_{\frac{1}{2}}, k - |C_1|)$  est de taille au plus  $2k$

# En finir avec Vertex Cover ?

- Il existe un algo exact en  $O^*(1.189^n)$  (Robson, 1984)
- Algo par exploration bornée en  $O(2^k n^c)$ , puis  $f(k) = O(1.466^k)$  en PC.
- Noyau de taille  $2k \Rightarrow O(1.22^{2k} + n^c) = O(1.413^k + n^c)$

Arbres d'exploration bornée raffinés par analyse de cas plus compliquée  
 $\Rightarrow$  optimisation "automatique": degré 1, puis  $\geq 5$ , puis 2, 3, 4  $\rightarrow O(1.34^k)$   
Meilleur résultat pour VC:  $O(1.28^k)$  par exploration bornée + sophistiquée

Borne inférieure sur la taille du noyau ?

On a vu que noyau de taille  $c \cdot n$  implique facteur d'approx  $c$ .

Or on ne connaît d'approx à un facteur meilleur que 2 pour VC, et un des théorèmes les plus remarquables de ces dernières années est le théorème PCP, dont une des formulations est que vertex cover n'admet pas d'algo d'approx à un facteur meilleur que 1.26.

# Cours 7: Complexité paramétrique

- Noyau linéaire pour VC
- Largeur arborescente et programmation dynamique
- Méta-algorithmes
- Calcul d'une décomposition arborescente
- Classes de complexité paramétrique

# Programmation dynamique

Les algorithmes pseudo-polynomiaux peuvent être vus comme FPT pour le paramètre **nombre de bits de codage des entrées**.

Par exemple le sac-à-dos de capacité  $P$  avec objets  $(p_i, v_i)$ ,  $i = 1, \dots, n$ :

par programmation dynamique, en notant  $V(p, i)$  la valeur du meilleur sac de capacité  $p$  rempli avec des objets d'indices entre 1 et  $i$ , et en utilisant la récurrence 
$$V(p, i + 1) = \max(V(p - p_{i+1}, i) + v_i, V(p, i))$$

⇒ solution en temps  $O(P \cdot n) = O(2^b \cdot n)$

FPT pour le paramètre  $b$ , nombre de bits de codage des entrées

# Programmation dynamique pour Steiner tree

**Steiner tree problem:** un graphe  $G$  valué et  $k$  sommets marqués;  
construire un arbre de poids minimum joignant les sommets marqués.

**Paramètre:**  $k$  le nombre de sommets marqués. **NP-complet**

Notons: –  $d(u, v)$  le plus court chemin de  $u$  à  $v$  dans  $G$ .

–  $St(X)$  le poids du meilleur arbre de Steiner connectant les sommets de  $X$ .

Alors  $St(X \cup \{y\}) = \min \begin{cases} \min_{x \in X} St(X) + d(x, y) \\ \min_{z \in V \setminus X} St_z(X \cup \{z\}) + d(y, z) \end{cases}$

avec  $St_z(X \cup \{z\}) = \min_{Y \cup Z = X} St(Y \cup \{z\}) + St(Z \cup \{z\})$

le 1er cas correspond à l'attachement à un sommet marqué déjà traité

le 2ème cas à l'attachement à un sommet relais: on y coupe l'arbre en 2.

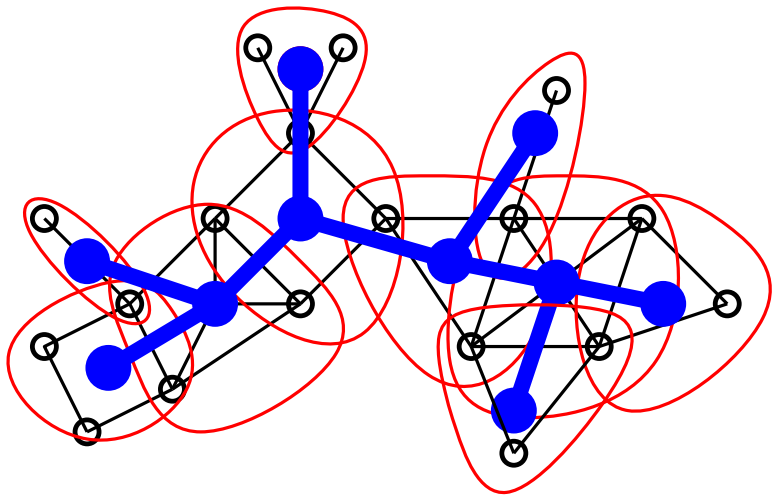
pour construire la table  $St$  pour les  $X$  de cardinal  $i$ , on minimise sur  $2^i n$   
sous-problèmes: au total  $\sum_{i < k} \binom{k}{i} 2^i n \leq 3^k n$  consultations de table.

**FPT  $O(3^k \cdot n^2 + n^2 \log n)$**

# Programmation dynamique sur les arbres

Vertex cover ou indépendant set sont faciles sur les arbres par programmation dynamique (ou plus précisément par analyse bottom-up).

On voudrait que cela reste vrai sur un graphe **proche** d'un arbre...

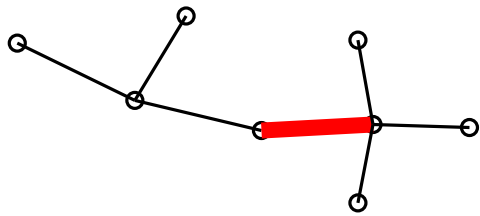


# Décomposition arborescente, largeur arborescente

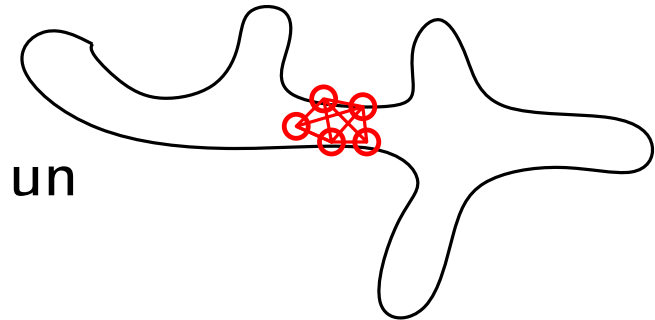
Mesurer si un graphe est plus ou moins loin d'être un arbre...

**Intérêt des arbres:** décomposer n'importe où et recommencer récursivement

Remplacer les paires séparatrices  
par des  $k$ -séparateurs



$\Rightarrow$  graphe inscrit dans un  
"arbre de largeur  $k$ "



**Arbre de décomposition d'un graphe  $G$ :**

– les nœuds sont des **sacs** formés de copies des sommets de  $G$ : tout sommet est dupliqué dans un ensemble de sacs formant un sous-arbre connexe.

– toute arête  $\{x, y\}$  de  $G$  doit être incluse dans au moins un sac

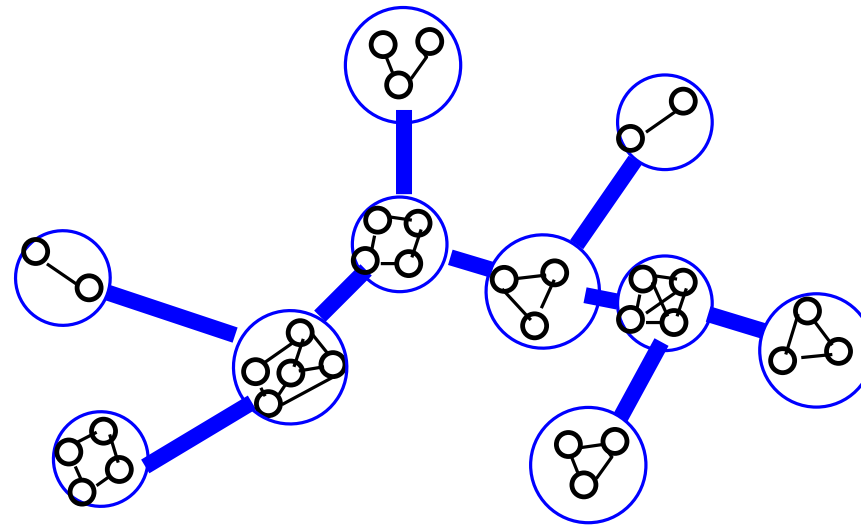
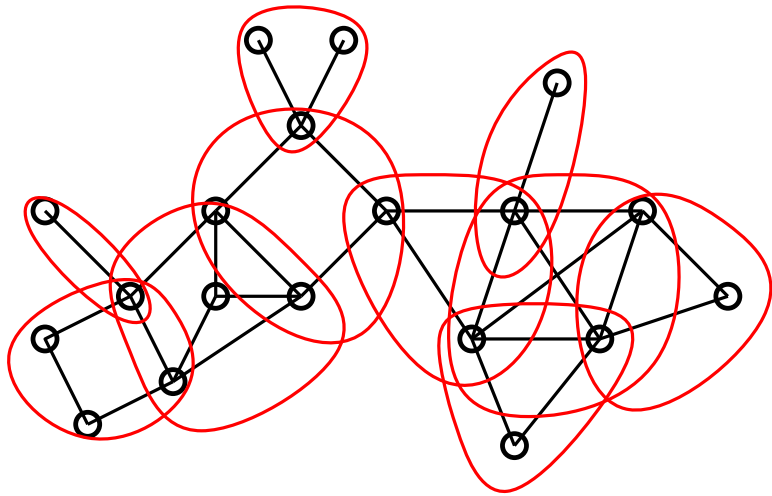
la **largeur de la décomposition** est la taille du plus gros sac moins 1

la **largeur arborescente du graphe  $G$**  est la largeur de la décomposition arborescente la moins large, notée  $tw(G)$  (treewidth)

Treewidth=1  $\Leftrightarrow$  arbres

Treewidth=2  $\leftrightarrow$  graphes séries-parallèles

# Décomposition arborescente, largeur arborescente



décomposition de largeur 4

optimale ? non

Arbre de décomposition d'un graphe  $G$ :

– les nœuds sont des sacs formés de copies des sommets de  $G$ : tout sommet est dupliqué dans un ensemble de sacs formant un sous-arbre connexe.

– toute arête  $\{x, y\}$  de  $G$  doit être incluse dans au moins un sac

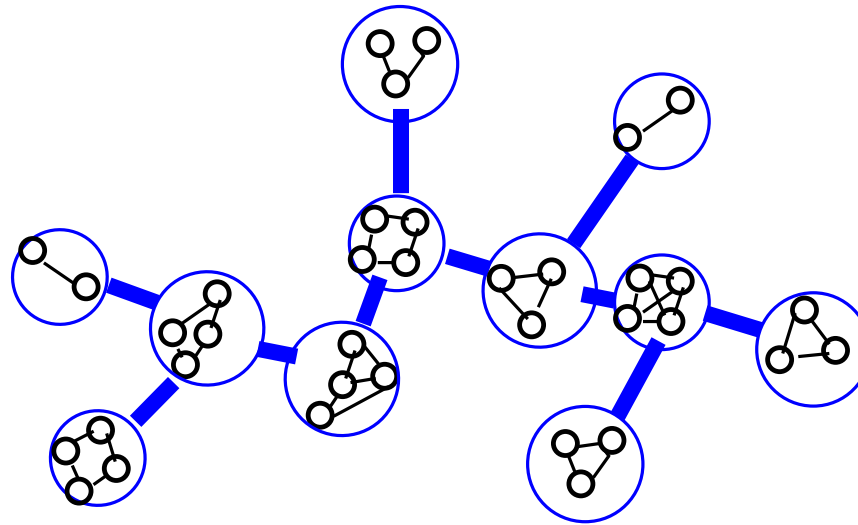
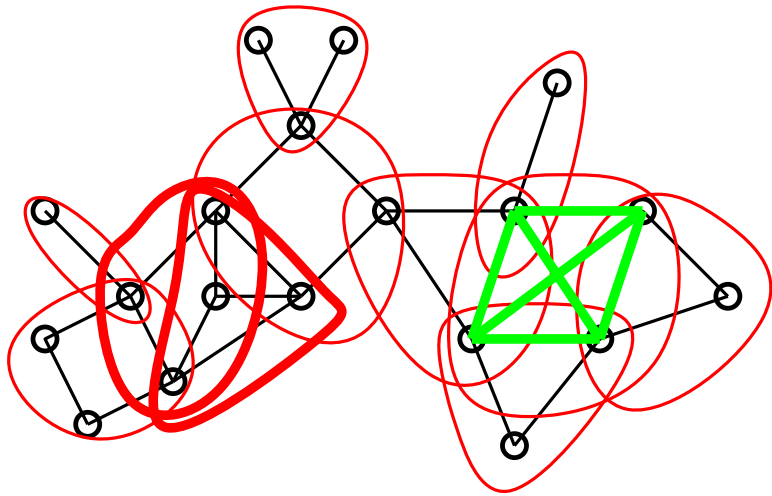
la largeur de la décomposition est la taille du plus gros sac moins 1

la largeur arborescente du graphe  $G$  est la largeur de la décomposition arborescente la moins large.

Treewidth=1  $\Leftrightarrow$  arbres

Treewidth=2  $\leftrightarrow$  graphes séries-parallèles

# Décomposition arborescente, largeur arborescente



⇒ décomposition de largeur 3 optimale ? oui, donc  $tw(G) = 3$

**Arbre de décomposition d'un graphe  $G$ :**

– les nœuds sont des **sacs** formés de copies des sommets de  $G$ : tout sommet est dupliqué dans un ensemble de sacs formant un sous-arbre connexe.

– toute arête  $\{x, y\}$  de  $G$  doit être incluse dans au moins un sac

la **largeur de la décomposition** est la taille du plus gros sac moins 1

la **largeur arborescente du graphe  $G$**  est la largeur de la décomposition arborescente la moins large.

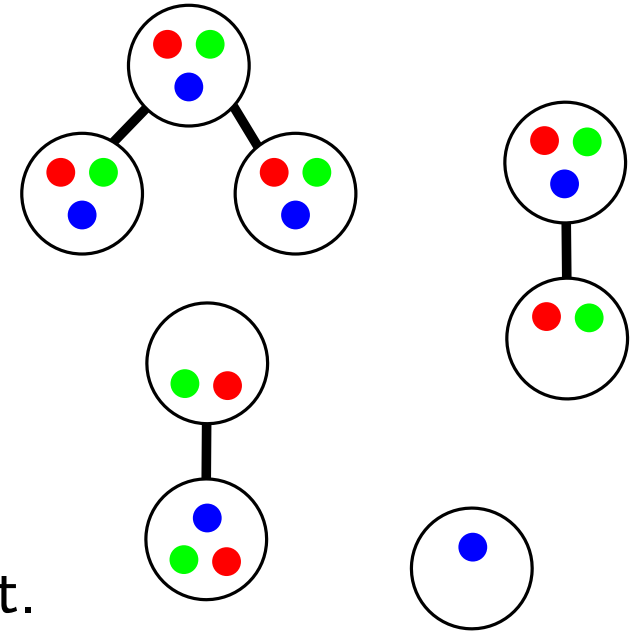
Treewidth=1  $\Leftrightarrow$  arbres

Treewidth=2  $\Leftrightarrow$  graphes séries-parallèles

# Décomposition arborescente sympatique

Une décomposition est **sympatique** si les sacs qui la constituent sont tous de types suivants:

- Des noeuds binaires, pour l'union de deux branches de même sac racine.
- Des noeuds unaires d'ajout: pour ajouter un sommet au sac fils
- Des noeuds unaires d'élimination: pour enlever un sommet du sac fils
- Des feuilles: un sac contenant un seul sommet.



Il est toujours possible de transformer une décomposition de largeur  $k$  avec  $m$  noeuds en une décomposition arborescente dite "sympatique", de même largeur, avec  $k^2m$  noeuds

**Intérêt:** moins de cas à traiter lors d'une analyse bottom up.

# Dominating set

# FPT pour la treewidth

**Donnée:** un graphe  $G = (V, E)$  et un entier  $\ell$ .

**Problème:** existe-t-il un ensemble  $D$  de  $\ell$  sommets dominants (tq tout sommet de  $G$  a un voisin dans  $D$ ) ?

NP-complet et pas de FPT connu pour le paramètre  $\ell$ .

Supposons qu'on ait une décomposition sympatique de  $G$  de largeur  $k$ .

On fait un calcul bottom-up sur l'arbre (dynamic programming):

- pour chaque sous-arbre de l'arbre calculer les configurations d'étiquettes admissibles pour les sommets du sac racine: dominant, dominé ou libre
  - un sommet introduit est dominé s'il y a un dominant dans le sac
  - un sommet exclu ne doit pas être libre
  - lors d'une union les étiquettes doivent être compatibles.

Le calcul est mené de bas en haut, pour chaque nœuds de l'arbre on calcule pour chaque configuration admissible le nombre de dominants nécessaires.

# SAT et largeur arborescente

**Donnée:** un ensemble de clauses  $C = \{C_i\}$  sur les variables  $X = \{x_j\}$ .

**Problème:**  $\exists$  une assignation des variables qui satisfasse toutes les clauses ?

Largeur arborescente d'une formule ?

la largeur arborescente du graphe d'incidence variable/clause !

**Théorème:** SAT est FPT par rapport à la largeur arborescente.

Même stratégie que pour dominant: programmation dynamique sur un arbre de décomposition sympatique

# Largeur arborescente, applications

On a appliqué la stratégie par programmation dynamique à

- $k$ -DOMINANT SET, qui est donc FPT par rapport à la treewidth
- SAT qui est FPT par rapport à la largeur arborescente du graphe d'incidence variables/clauses

La même stratégie s'applique à de nombreux problèmes

VERTEX COVER, INDEPENDANT SET,  $k$ -COLORING, HAMILTON CYCLE,...

Les programmes écrits dans un langage impératif sans GOTO ont un graphe de dépendance des flots de largeur arborescente au plus 6  
+ le  $k$ -coloriage des graphes est FPT pour la treewidth  
⇒ un algorithme efficace pour l'allocation de registres.

# Cours 7: Complexité paramétrique

- Noyau linéaire pour VC
- Largeur arborescente et programmation dynamique
- Méta-algorithmes
- Calcul d'une décomposition arborescente
- Classes de complexité paramétrique

# Logique monadique du 2ème ordre sur les graphes

Formules de MSOL:

- connecteurs logiques (et, ou, non,  $\Rightarrow$ ,  $=$ ,  $\neq$ )
- prédicat binaire pour adjacence et incidence
- quantificateur  $\exists$ ,  $\forall$  sur des variables de sommets ou d'arêtes, ou sur des variables d'ensemble de sommets ou d'arêtes.
- relations  $\in$ ,  $\subset$

**Exemple:**  $G$  est hamiltonien  $\Leftrightarrow$  existence d'un ensemble d'arêtes tq

- chaque sommet est incident à deux arêtes
- tout ensemble de sommet est connecté à son complémentaire

$\Rightarrow$  expressible en MSOL

# Méta algorithme de Courcelle et compression itérative

**Théorème:** Toute propriété de graphe exprimable en MSOL peut être testée en temps  $O(f(k).n)$  sur les graphes de largeur arborescente au plus  $k$ , où  $f$  est une fonction ne dépendant que de la structure de la formule.

**Exemple:** Hamiltonien est FPT par rapport à la treewidth.

La fonction  $f$  est très grosse donc c'est plutôt un résultat de classification. En pratique on utilise les décompositions arborescentes par prog dyn.

**Compression itérative:** utilisation de la largeur arborescente pour montrer FPT pour un autre paramètre  $k$

montrer que si  $tw(G) > c \cdot k$  alors

pas de solution de paramètre  $k$  (existence d'une obstruction)

ou possibilité de réduire l'instance (une partie de la solution est contrainte)

→ réduction de  $tw$  à  $tw(G) \leq c \cdot k$  et (FPT pour  $tw$ )  $\Rightarrow$  (FPT pour  $k$ ).

# Cours 7: Complexité paramétrique

- Noyau linéaire pour VC
- Largeur arborescente et programmation dynamique
- Méta-algorithmes
- Calcul d'une décomposition arborescente
- Classes de complexité paramétrique

# Décomposition arborescente, calcul ?

En général le calcul de la largeur arborescente est NP-complet mais il est **FPT** pour le paramètre largeur arborescente lui-même.

**Théorème:** il existe un algo de complexité  $O(2^{k^3} n)$  qui décide si la largeur arborescente est inférieure à  $k$  et dans ce cas en construit une.

Ce premier algo est complexe à décrire, il existe de nombreuses variantes qui construisent des décompositions arborescentes sous-optimale *i.e.* de largeur  $c \cdot k$  avec une meilleure complexité.

# Calcul de la largeur arborescente: approximation

**Algo simplifié:** pour tout  $k$  on montre que  $tw(G) > k$  ou on trouve une décomposition de  $w \leq 4k$  en temps  $f(k)n^{O(1)}$ .

Deux ensembles de sommets  $Y$  et  $Z$  de cardinal  $k$  sont séparables s'il existe un ensemble  $S$  d'au plus  $k - 1$  sommets qui sépare  $Y \setminus S$  de  $Z \setminus S$ .

$\Rightarrow$  comment tester si  $Y$  et  $Z$  sont séparables ? par flot, en temps poly

Un ensemble  $X$  d'au moins  $k$  sommets est  $k$ -lié si pour tout  $k' \leq k$ , toute paire de  $k'$ -uplet de sommets de  $X$  est séparable

$\Rightarrow$  comment tester si  $X$  est  $k$ -lié ? complexité ?  $O(4^k n^c)$

**Lemme:** si  $G$  contient un  $(k + 1)$ -lié  $X$  avec  $|X| \geq 3k$  alors  $tw(g) \geq k$ .

preuve par inspection dans une décomposition de largeur  $k$  du plus petit sous-arbre contenant  $2k$  des sommets de  $X$ .

- soit il y a un fils de la racine dont le sous-arbre contient  $k$  des sommets de  $X$ , et on prend l'intersection fils/racine qui sépare deux ensembles de  $k$  sommets de  $X$ .

- soit le sac racine contient  $k$  sommets de  $X$  et on prend l'intersection racine/père qui sépare deux ensembles de  $k$

- soit il y a un ensemble minimal de  $j$  fils de la racine qui au total contiennent au moins  $k + 1$  des sommets de  $X$  et on prend la racine qui sépare deux ensembles de  $k + 1$

# Calcul de la largeur arborescente: approximation

**Algorithme:** On incorpore les sommets incrémentalement dans une décomposition partielle  $T_U$  de largeur  $\leq 4k$

à tout moment on veut que chaque composante connexe  $C$  de  $G - U$  ait au plus  $3k$  voisins et qu'ils soient tous ses voisins inclus dans 1 sac de  $T_U$   
au départ on prend un premier sac de  $3k$  sommets arbitraires.

On répète: soit  $C$  une composante et  $X$  ses voisins, inclus dans le sac  $X_t$ .

$\Rightarrow$  si  $|X| < 3k$ , on prend  $x \in C$  voisin de  $X_t$  et on ajoute un sac  $X \cup \{x\}$   
voisin de  $X$  **il a au plus  $3k + 1$  elts**

$\Rightarrow$  si  $|X| = 3k$  et  $X$  est  $(k + 1)$ -lié alors on peut dire que  $tw(G) > k$ .

sinon on obtient un séparateur  $S$  de  $Y$  et  $Z$  ss-ensembles de  $X$ , avec  $|S| < |Y| = |Z| \leq k + 1$ : on ajoute un sac  $X \cup (S \cap C)$  **(de card  $\leq 4k$ )**.  
une nouvelle composante  $C' \subseteq C$  a ses voisins dans  $(X \setminus Z) \cup (S \cap C)$  ou dans  $(X \setminus Y) \cup (S \cap C)$  (sinon on aurait un chemin de  $Y$  à  $Z$  par  $C'$ )

# Cours 7: Complexité paramétrique

- Noyau linéaire pour VC
- Largeur arborescente et programmation dynamique
- Méta-algorithmes
- Calcul d'une décomposition arborescente
- Classes de complexité paramétrique

# Classes de complexité paramétrique

Introduire des classes, des notions de réduction et de complétude pour "montrer" la difficulté paramétrique de pbs.

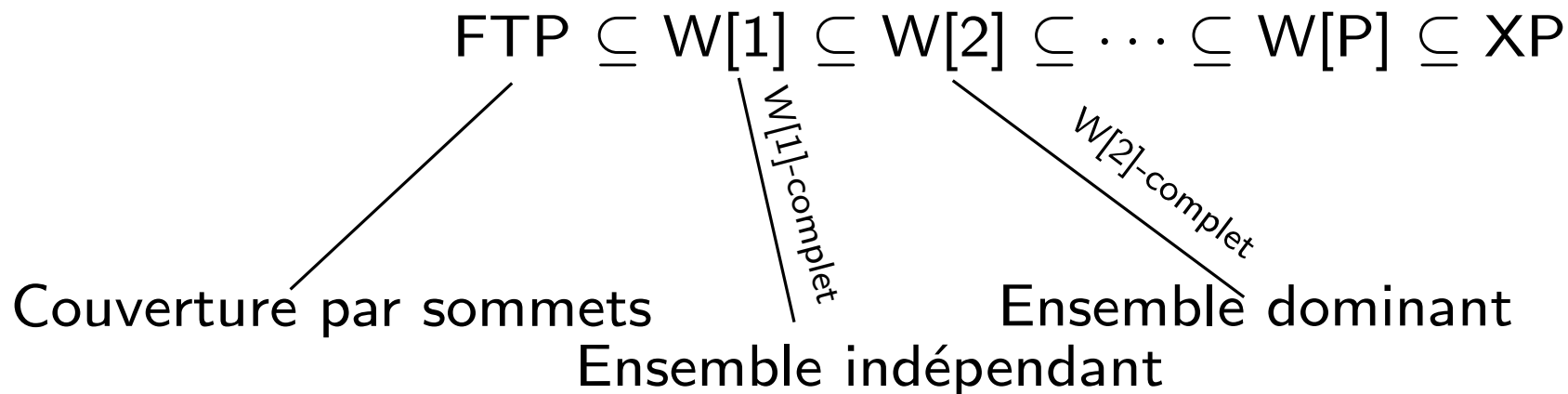
Réduction paramétrique: transformation  $(I, k) \mapsto (I', k')$  en temps poly, avec  $[(I, k) = \text{Vrai} \Leftrightarrow (I', k') = \text{Vrai}]$ , et  $k' = O(k^c)$

WEIGHTED CIRCUIT SAT[t]: étant donné un circuit de profondeur  $t$  alternant portes ET et OU, décider s'il existe une affectation de poids  $k$  la satisfaisant.

WEIGHTED CIRCUIT SAT[1] = WEIGHTED CNF-SAT

**Idée sous-jacente:** on ne croit pas que WEIGHTED CNF-SAT  $\in$  FPT

W[t]=problème paramétré se réduisant à WEIGHTED CIRCUIT SAT[t]



# Cours 7: Complexité paramétrique

- Noyau linéaire pour VC
- Largeur arborescente et programmation dynamique
- Méta-algorithmes
- Calcul d'une décomposition arborescente
- Classes de complexité paramétrique

**Retenir:** bounded search tree / branch and bound  
kernelisation pour extraire la partie dure des instances  
Utilisation de paramètres structuraux

**Aller plus loin:** une théorie de la complexité paramétrique  
Treewidth, edgewidth et théorie des mineurs exclus de  
Robertson et Seymour