Cours 6: Complexité paramétrique

- Programmation dynamique et complexité paramétrique
- Largeur arborescente
- Méta-algorithmes
- Calcul d'une décomposition arborescente
- Classes de complexité paramétrique

Cours 6: Complexité paramétrique

- Programmation dynamique et complexité paramétrique
- Largeur arborescente
- Méta-algorithmes
- Calcul d'une décomposition arborescente
- Classes de complexité paramétrique

Programmation dynamique

Les algorithmes pseudo-polynomiaux peuvent être vus comme FTP 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), $i = 1, \ldots, n$:

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 $V(p, i+1) = \max(V(p - p_{i+1}, i) + v_i,\ V(p, i))$

$\Rightarrow$ 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.

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.

NP-complet

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.

Paramêtre: k le nombre de sommets marqués. NP-complet

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.

Paramêtre: k le nombre de sommets marqués. NP-complet

Notons: – $d(u, v)$ le plus court chemin de u à v dans G.
– $St_v(X)$ poids de l'arbre min joignant les sommets de X et contenant v.

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.

Paramêtre: k le nombre de sommets marqués. NP-complet

Notons: – $d(u,v)$ le plus court chemin de u à v dans G.
– $St_v(X)$ poids de l'arbre min joignant les sommets de X et contenant v.

Alors $St_v(X) = \min_{u \in V} \min_{\substack{Y \cup Z = X \\ Y \neq \emptyset,\, Z \neq \emptyset}} St_u(Y) + St_u(Z) + d(u,v)$

inégalité $\leq$: pour tous u, Y, Z on peut extraire un arbre de poids inférieur ou égal à $St_u(Y) + St_v(Z) + d(u,v)$

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.

Paramêtre: k le nombre de sommets marqués. NP-complet

Notons: – $d(u,v)$ le plus court chemin de u à v dans G.
– $St_v(X)$ poids de l'arbre min joignant les sommets de X et contenant v.

Alors $St_v(X) = \min_{u \in V} \min_{\substack{Y \cup Z = X \\ Y \neq \emptyset,\, Z \neq \emptyset}} St_u(Y) + St_u(Z) + d(u,v)$

inégalité $\leq$: pour tous u, Y, Z on peut extraire un arbre de poids inférieur ou égal à $St_u(Y) + St_v(Z) + d(u,v)$

inégalité $\geq$ étant donné un arbre min pour v, X, prendre u le sommet interne le plus proche de v et y couper l'arbre en 2.

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.

Paramêtre: k le nombre de sommets marqués. NP-complet

Notons: – $d(u,v)$ le plus court chemin de u à v dans G.
– $St_v(X)$ poids de l'arbre min joignant les sommets de X et contenant v.

Alors $St_v(X) = \min_{u \in V} \min_{\substack{Y \cup Z = X \\ Y \neq \emptyset,\, Z \neq \emptyset}} St_u(Y) + St_u(Z) + d(u,v)$

inégalité $\leq$: pour tous u, Y, Z on peut extraire un arbre de poids inférieur ou égal à $St_u(Y)+St_v(Z)+d(u,v)$

inégalité $\geq$ étant donné un arbre min pour v, X, prendre u le sommet interne le plus proche de v et y couper 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.

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.

Paramêtre: k le nombre de sommets marqués. NP-complet

Notons: – $d(u,v)$ le plus court chemin de u à v dans G.
– $St_v(X)$ poids de l'arbre min joignant les sommets de X et contenant v.

Alors $St_v(X) = \min_{u \in V} \min_{\substack{Y \cup Z = X \\ Y \neq \emptyset,\, Z \neq \emptyset}} St_u(Y) + St_u(Z) + d(u,v)$

inégalité $\leq$: pour tous u, Y, Z on peut extraire un arbre de poids inférieur ou égal à $St_u(Y)+St_v(Z)+d(u,v)$

inégalité $\geq$ étant donné un arbre min pour v, X, prendre u le sommet interne le plus proche de v et y couper l'arbre en 2.

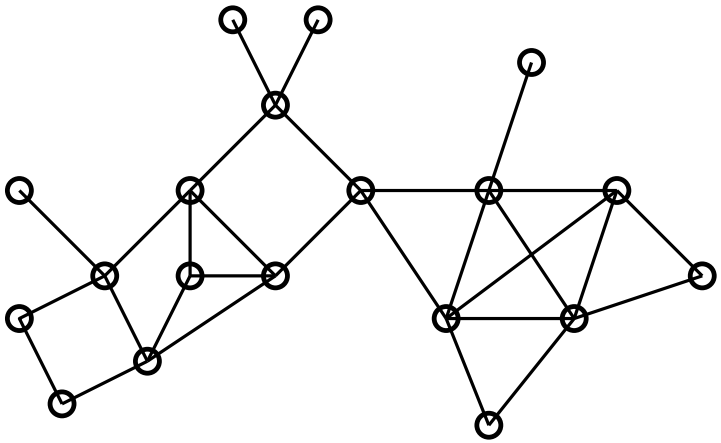
pour construire la table St pour les X de cardinal i, on minimise sur $2^i n$ sous-problèmes: au total $\sum_{i<k} \binom{k}{i} 2^i n \leq 3^k n$ consultations de table.

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

Programmation dynamique sur les arbres

On a vu que Couvrant Min ou Indépendant Max sont faciles sur les arbres par programmation dynamique (ou par analyse bottom-up).

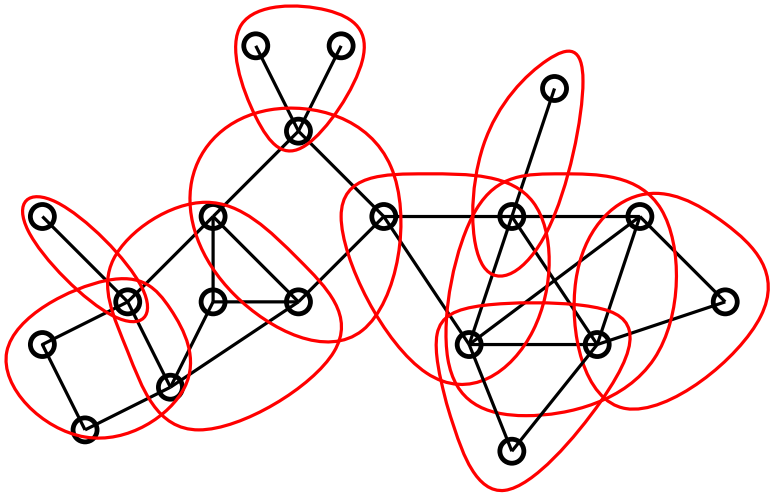
On voudrait que cela reste vrai sur un graphe proche d'un arbre...



Programmation dynamique sur les arbres

On a vu que Couvrant Min ou Indépendant Max sont faciles sur les arbres par programmation dynamique (ou par analyse bottom-up).

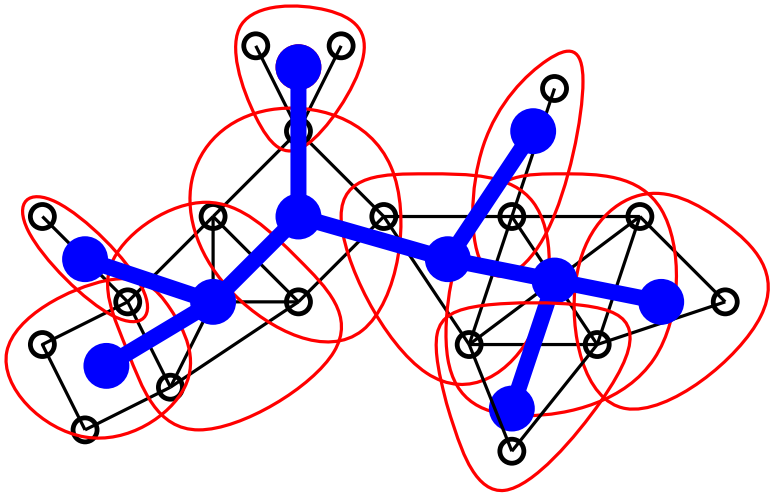
On voudrait que cela reste vrai sur un graphe proche d'un arbre...



Programmation dynamique sur les arbres

On a vu que Couvrant Min ou Indépendant Max sont faciles sur les arbres par programmation dynamique (ou par analyse bottom-up).

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



Cours 6: Complexité paramétrique

- Programmation dynamique et complexité paramétrique
- Largeur arborescente
- Méta-algorithmes
- Calcul d'une décomposition arborescente
- Classes de complexité paramétrique

Décomposition arborescente, largeur arborescente

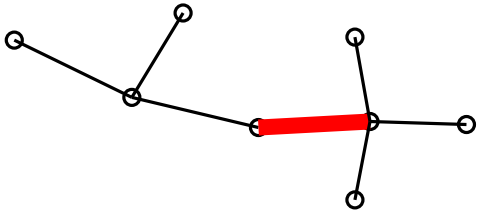
Mesurer si un graphe est plus ou moins loin d'être un arbre...

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

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

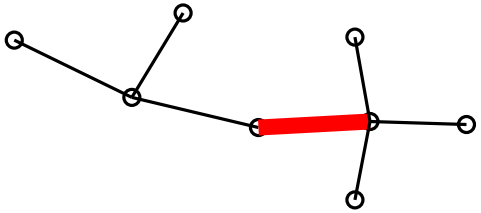


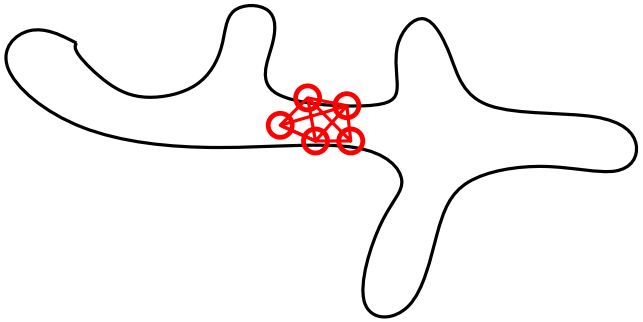
Décomposition arborescente, largeur arborescente

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



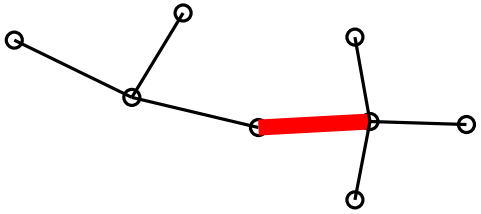


Décomposition arborescente, largeur arborescente

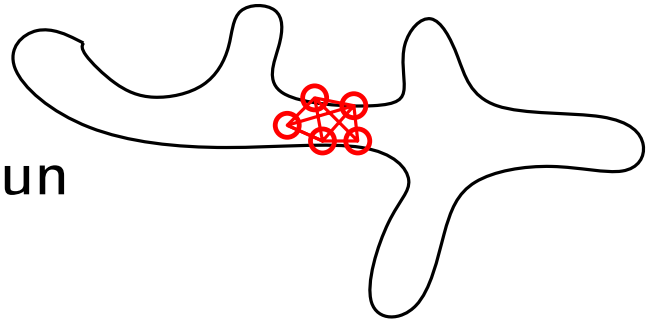
Mesurer si un graphe est plus ou moins loin d'être un arbre...

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"

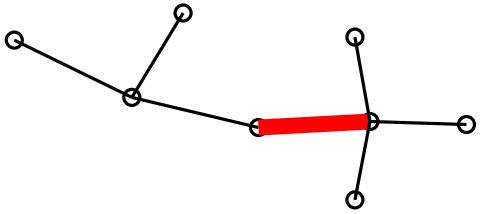


Décomposition arborescente, largeur arborescente

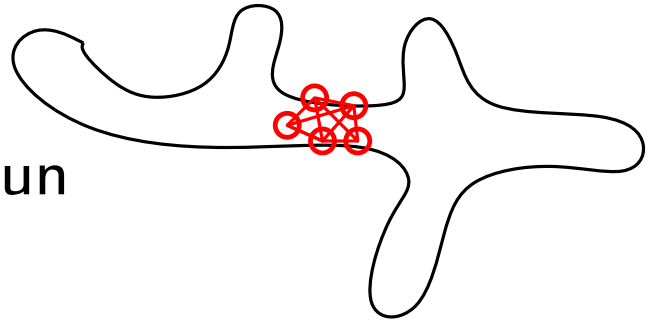
Mesurer si un graphe est plus ou moins loin d'être un arbre...

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"



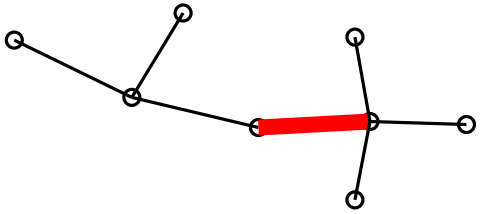
Arbre de décomposition d'un graphe $G=$ arbre dont les nœuds
sont des sacs contenant des copies des sommets de G et tel que:
– les sacs contenant un sommet x de G donné forment un sous-arbre connexe.
– toute arête $\{x, y\}$ de G est incluse dans au moins un sac

Décomposition arborescente, largeur arborescente

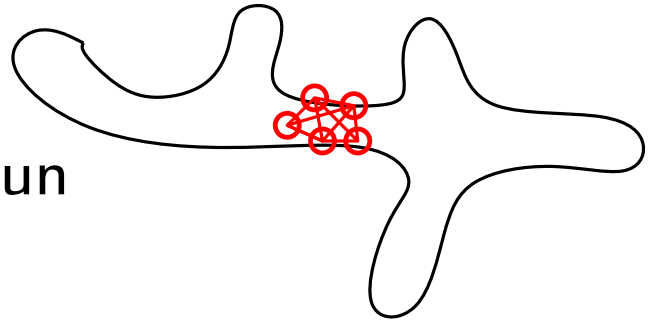
Mesurer si un graphe est plus ou moins loin d'être un arbre...

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 $G=$ arbre dont les nœuds
sont des sacs contenant des copies des sommets de G et tel que:
– les sacs contenant un sommet x de G donné forment un sous-arbre connexe.
– toute arête $\{x, y\}$ de G est incluse dans au moins un sac

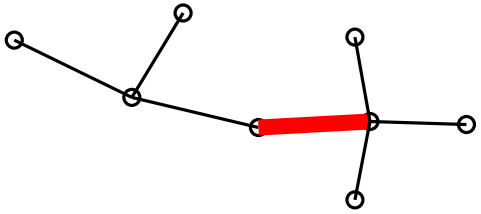
La largeur de la décomposition est la taille du plus gros sac moins 1

Décomposition arborescente, largeur arborescente

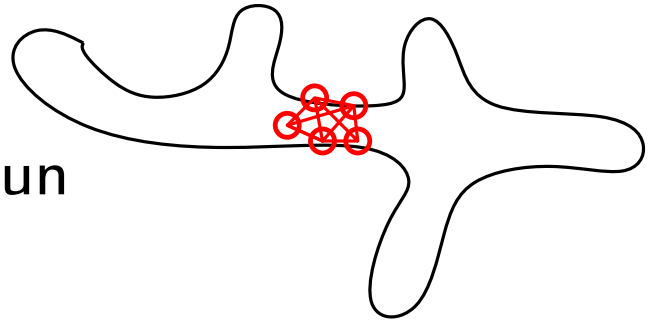
Mesurer si un graphe est plus ou moins loin d'être un arbre...

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= arbre dont les nœuds sont des sacs contenant des copies des sommets de G et tel que:
– les sacs contenant un sommet x de G donné forment un sous-arbre connexe.
– toute arête $\{x, y\}$ de G est incluse dans au moins un sac

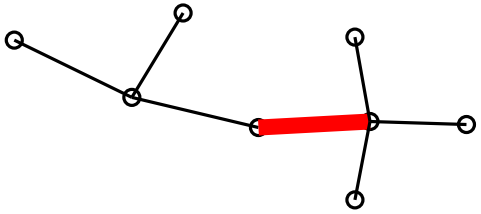
La largeur de la décomposition est la taille du plus gros sac moins 1
La largeur arborescente du graphe G est la largeur de la décomposition arborescente la moins large, notée $tw(G)$ (treewidth)

Décomposition arborescente, largeur arborescente

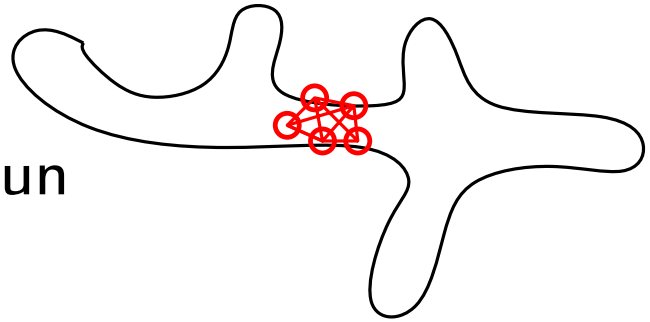
Mesurer si un graphe est plus ou moins loin d'être un arbre...

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"

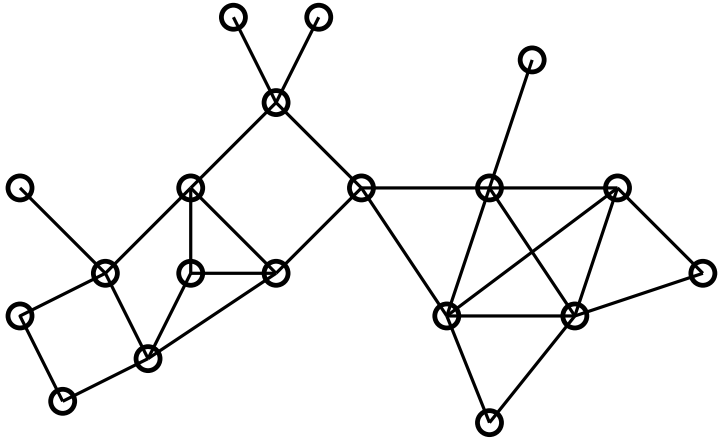


Arbre de décomposition d'un graphe G= arbre dont les nœuds sont des sacs contenant des copies des sommets de G et tel que:
– les sacs contenant un sommet x de G donné forment un sous-arbre connexe.
– toute arête $\{x, y\}$ de G est incluse dans au moins un sac

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

Décomposition arborescente, largeur arborescente

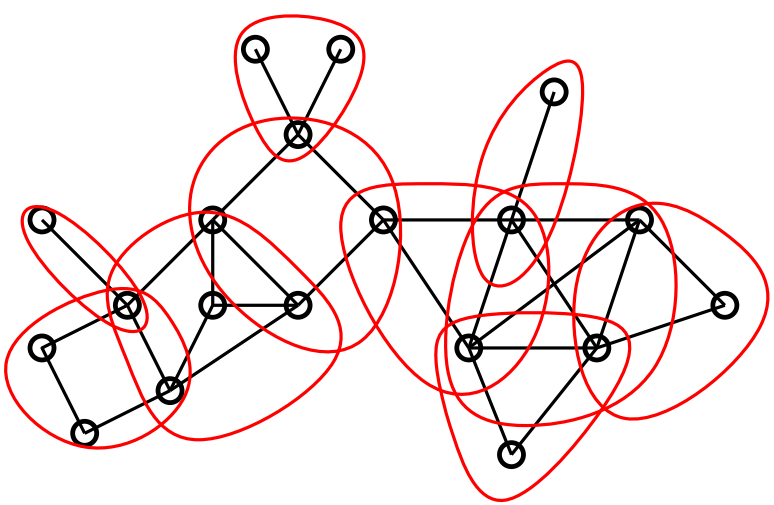


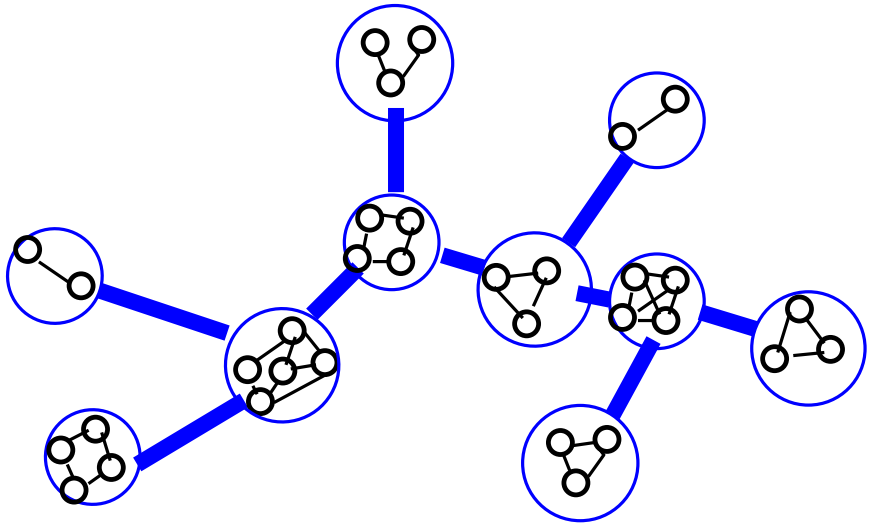
Arbre de décomposition d'un graphe $G=$ arbre dont les nœuds
sont des sacs contenant des copies des sommets de G et tel que:
– les sacs contenant un sommet x de G donné forment un sous-arbre connexe.
– toute arête $\{x, y\}$ de G est incluse dans au moins un sac

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

Décomposition arborescente, largeur arborescente





décomposition de largeur 4

Arbre de décomposition d'un graphe G= arbre dont les nœuds sont des sacs contenant des copies des sommets de G et tel que:

– les sacs contenant un sommet x de G donné forment un sous-arbre connexe.

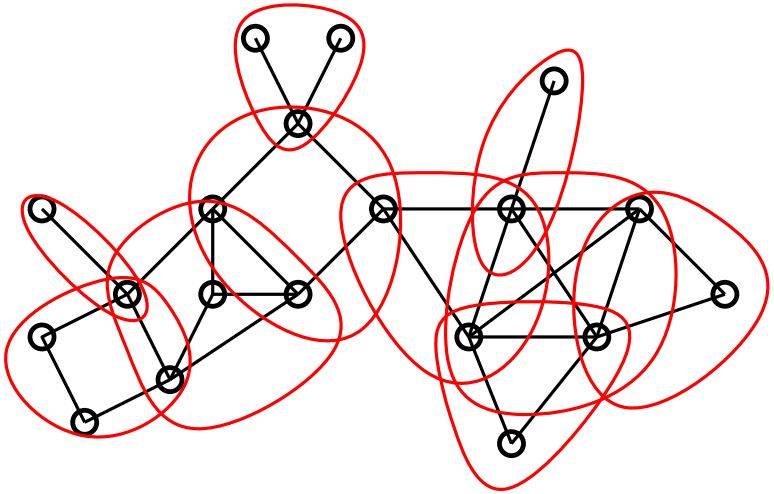
– toute arête $\{x, y\}$ de G est incluse dans au moins un sac

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

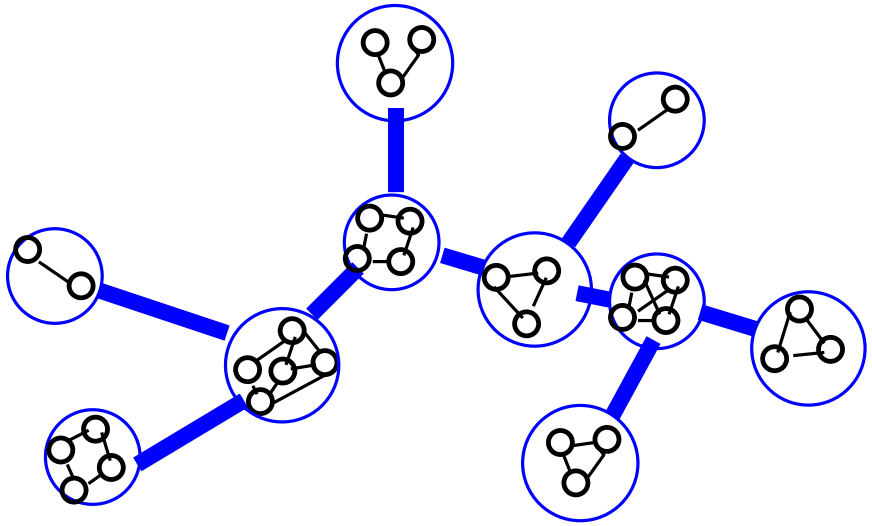
La largeur arborescente du graphe G est la largeur de la décomposition arborescente la moins large, notée $tw(G)$ (treewidth)

Treewidth=1 $\Leftrightarrow$ arbres

Décomposition arborescente, largeur arborescente



décomposition de largeur 4



optimale ?

Arbre de décomposition d'un graphe G= arbre dont les nœuds sont des sacs contenant des copies des sommets de G et tel que:

– les sacs contenant un sommet x de G donné forment un sous-arbre connexe.

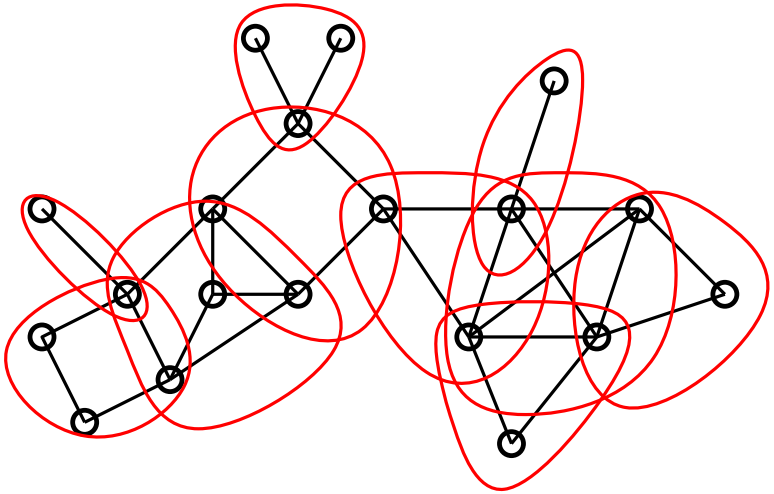
– toute arête $\{x, y\}$ de G est incluse dans au moins un sac

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

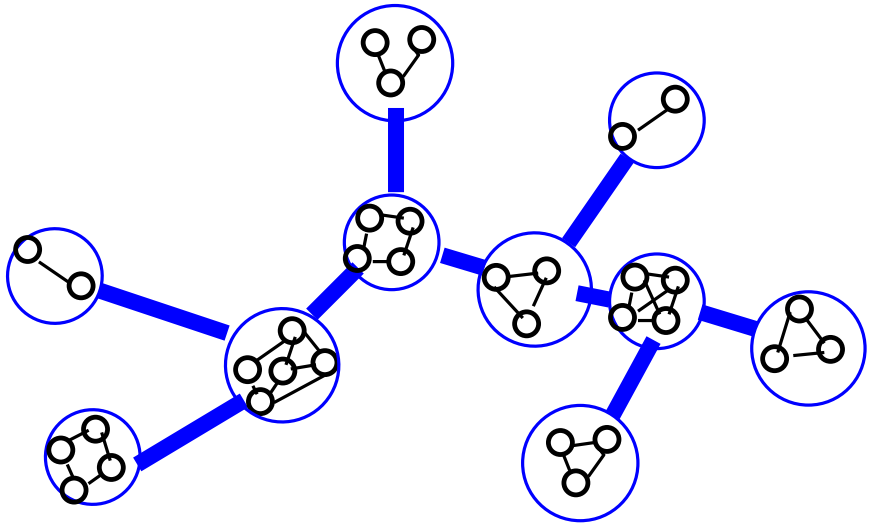
La largeur arborescente du graphe G est la largeur de la décomposition arborescente la moins large, notée $tw(G)$ (treewidth)

Treewidth=1 $\Leftrightarrow$ arbres

Décomposition arborescente, largeur arborescente



décomposition de largeur 4



optimale ? non

Arbre de décomposition d'un graphe G= arbre dont les nœuds sont des sacs contenant des copies des sommets de G et tel que:

– les sacs contenant un sommet x de G donné forment un sous-arbre connexe.

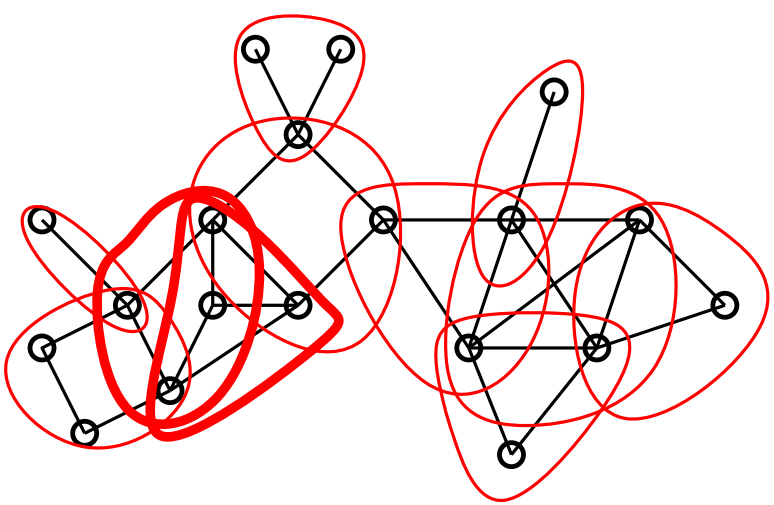
– toute arête $\{x, y\}$ de G est incluse dans au moins un sac

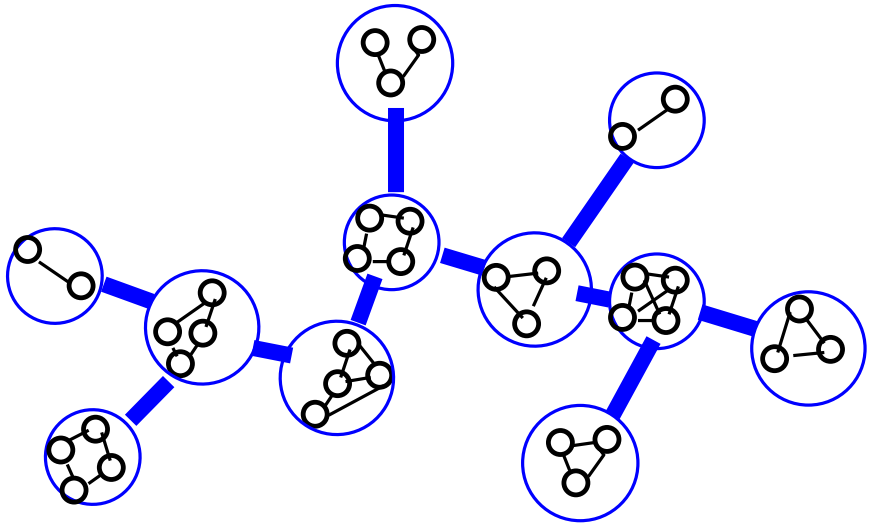
La largeur de la décomposition est la taille du plus gros sac moins 1

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

Treewidth=1 $\Leftrightarrow$ arbres

Décomposition arborescente, largeur arborescente





$\Rightarrow$ décomposition de largeur 3 optimale ?

Arbre de décomposition d'un graphe $G=$ arbre dont les nœuds sont des sacs contenant des copies des sommets de G et tel que:

– les sacs contenant un sommet x de G donné forment un sous-arbre connexe.

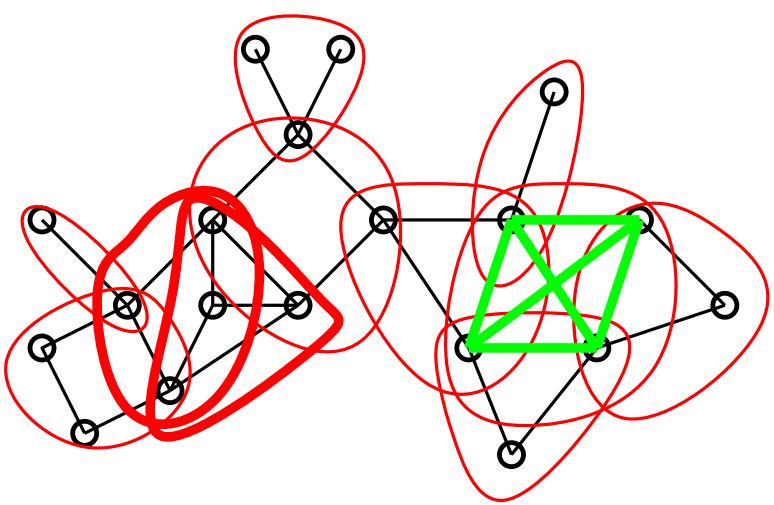
– toute arête $\{x, y\}$ de G est incluse dans au moins un sac

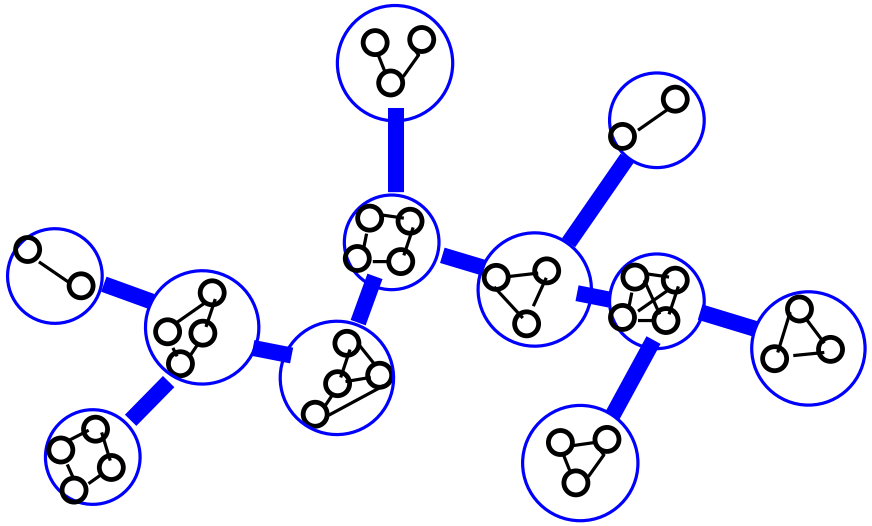
La largeur de la décomposition est la taille du plus gros sac moins 1

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

Treewidth=1 $\Leftrightarrow$ arbres

Décomposition arborescente, largeur arborescente





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

Arbre de décomposition d'un graphe G= arbre dont les nœuds sont des sacs contenant des copies des sommets de G et tel que:

– les sacs contenant un sommet x de G donné forment un sous-arbre connexe.

– toute arête $\{x, y\}$ de G est 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

Décomposition arborescente sympathique

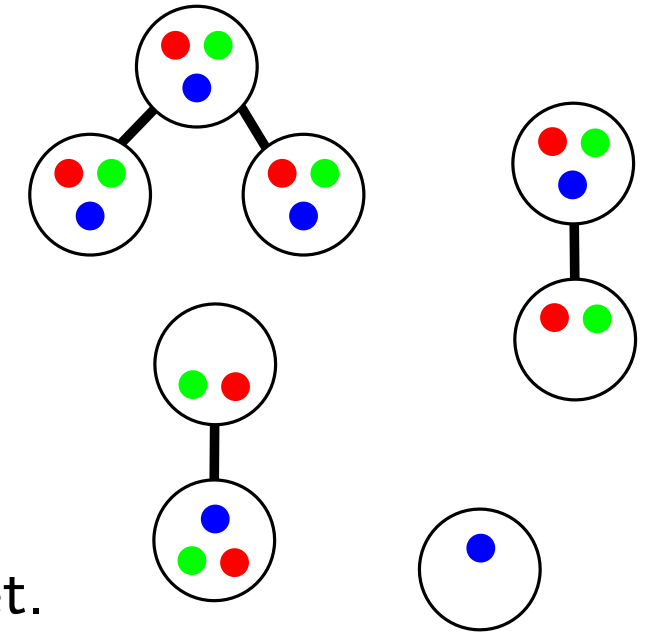
Une décomposition est sympathique si les sacs qui la constituent sont tous de types suivants:

– 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 "sympathique", de même largeur, avec k^2m noeuds

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

Dominating set

Donnée: un graphe $G = (V, E)$ et un entier ℓ.

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

NP-complet et pas de FPT connu pour le paramètre ℓ.

Dominating set

Donnée: un graphe $G = (V, E)$ et un entier ℓ.

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

NP-complet et pas de FPT connu pour le paramètre ℓ.

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

Dominating set

Donnée: un graphe $G = (V, E)$ et un entier ℓ.

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

NP-complet et pas de FPT connu pour le paramètre ℓ.

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

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

Dominating set

Donnée: un graphe $G = (V, E)$ et un entier ℓ.

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

NP-complet et pas de FPT connu pour le paramètre ℓ.

Supposons qu'on ait une décomposition sympathique 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

Dominating set

Donnée: un graphe $G = (V, E)$ et un entier ℓ.

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

NP-complet et pas de FPT connu pour le paramètre ℓ.

Supposons qu'on ait une décomposition sympathique 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.

Dominating set

Donnée: un graphe $G = (V, E)$ et un entier ℓ.

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

NP-complet et pas de FPT connu pour le paramètre ℓ.

Supposons qu'on ait une décomposition sympathique 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 nœud de l'arbre on calcule pour chaque configuration admissible le nombre de dominants nécessaires.

Dominating set FPT pour la treewidth

Donnée: un graphe $G = (V, E)$ et un entier ℓ.

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

NP-complet et pas de FPT connu pour le paramètre ℓ.

Supposons qu'on ait une décomposition sympathique 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 nœud 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 sympathique

Largeur arborescente, applications

On a appliqué la stratégie par programmation dynamique à

— k-DOMINANT SET, qui est donc FPT par rapport à la treewidth

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

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,...

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,...

Theorem (Thorup 1997)

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 du graphe d'inférence d'un graphe est FPT pour la treewidth.

$\Rightarrow$ un algorithme efficace pour l'allocation de registres.

Cours 6: Complexité paramétrique

- Programmation dynamique et complexité paramétrique
- Largeur arborescente
- 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 $\in$, $\subset$

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 $\Leftrightarrow$ existance d'un ensemble d'arêtes tq

– chaque sommet est incident à deux arêtes

– tout ensemble de sommets est connecté à son complémentaire

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 $\Leftrightarrow$ existance d'un ensemble d'arêtes tq

– chaque sommet est incident à deux arêtes

– tout ensemble de sommets est connecté à son complémentaire

$\Rightarrow$ expressible en MSOL

Méta algorithme de Courcelle

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

Méta algorithme de Courcelle

Théorème: Toute propriété de graphe exprimable en MSOL peut être testée en temps $O(f(k).n)$ sur les graphes de taille n et 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.

Méta algorithme de Courcelle

Théorème: Toute propriété de graphe exprimable en MSOL peut être testée en temps $O(f(k).n)$ sur les graphes de taille n et 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.

Méta algorithme de Courcelle

Théorème: Toute propriété de graphe exprimable en MSOL peut être testée en temps $O(f(k).n)$ sur les graphes de taille n et 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.

Cours 6: Complexité paramétrique

- Programmation dynamique et complexité paramétrique
- Largeur arborescente
- 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-optimales *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 $tw(G) \geq k$ ou on trouve une décomposition de $tw \leq 4k$ en temps $f(k)n^{O(1)}$.

Calcul de la largeur arborescente: approximation

Algo simplifié: pour tout k on montre que $tw(G) \geq k$ ou on trouve une décomposition de $tw \leq 4k$ 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 $Z \setminus S$.

Calcul de la largeur arborescente: approximation

Algo simplifié: pour tout k on montre que $tw(G) \geq k$ ou on trouve une décomposition de $tw \leq 4k$ 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 ?

Calcul de la largeur arborescente: approximation

Algo simplifié: pour tout k on montre que $tw(G) \geq k$ ou on trouve une décomposition de $tw \leq 4k$ 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

Calcul de la largeur arborescente: approximation

Algo simplifié: pour tout k on montre que $tw(G) \geq k$ ou on trouve une décomposition de $tw \leq 4k$ 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 non-séparable

Calcul de la largeur arborescente: approximation

Algo simplifié: pour tout k on montre que $tw(G) \geq k$ ou on trouve une décomposition de $tw \leq 4k$ 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 non-séparable

$\Rightarrow$ comment tester si X est k-lié ? complexité ?

Calcul de la largeur arborescente: approximation

Algo simplifié: pour tout k on montre que $tw(G) \geq k$ ou on trouve une décomposition de $tw \leq 4k$ 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 non-séparable

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

Calcul de la largeur arborescente: approximation

Algo simplifié: pour tout k on montre que $tw(G) \geq k$ ou on trouve une décomposition de $tw \leq 4k$ 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 non-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 $tw(G) \geq k$.

Calcul de la largeur arborescente: approximation

Algo simplifié: pour tout k on montre que $tw(G) \geq k$ ou on trouve une décomposition de $tw \leq 4k$ 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 non-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.

Calcul de la largeur arborescente: approximation

Algo simplifié: pour tout k on montre que $tw(G) \geq k$ ou on trouve une décomposition de $tw \leq 4k$ 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 non-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 contient au plus $k-1$ sommets et sépare deux ensembles de k sommets de X.

- 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

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

Calcul de la largeur arborescente: approximation

Algo simplifié: 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 inclus dans 1 sac de T_U

Calcul de la largeur arborescente: approximation

Algo simplifié: 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 inclus dans 1 sac de T_U

au départ on prend un premier sac de $3k$ sommets arbitraires.

Calcul de la largeur arborescente: approximation

Algo simplifié: 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 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.

Calcul de la largeur arborescente: approximation

Algo simplifié: 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 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_t il a au plus $3k$ elts

Calcul de la largeur arborescente: approximation

Algo simplifié: 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 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_t il a au plus $3k$ elts

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

Calcul de la largeur arborescente: approximation

Algo simplifié: 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 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_t il a au plus $3k$ elts

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

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 $\leq 4k$).

Calcul de la largeur arborescente: approximation

Algo simplifié: 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 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_t il a au plus $3k$ elts

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

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 6: Complexité paramétrique

- Programmation dynamique et complexité paramétrique
- Largeur arborescente
- 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.

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 $k' = O(k^c)$

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.

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

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

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]=problèmes paramétrés se réduisant à WEIGHTED CIRCUIT SAT[t]

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 $k' = O(k^c)$

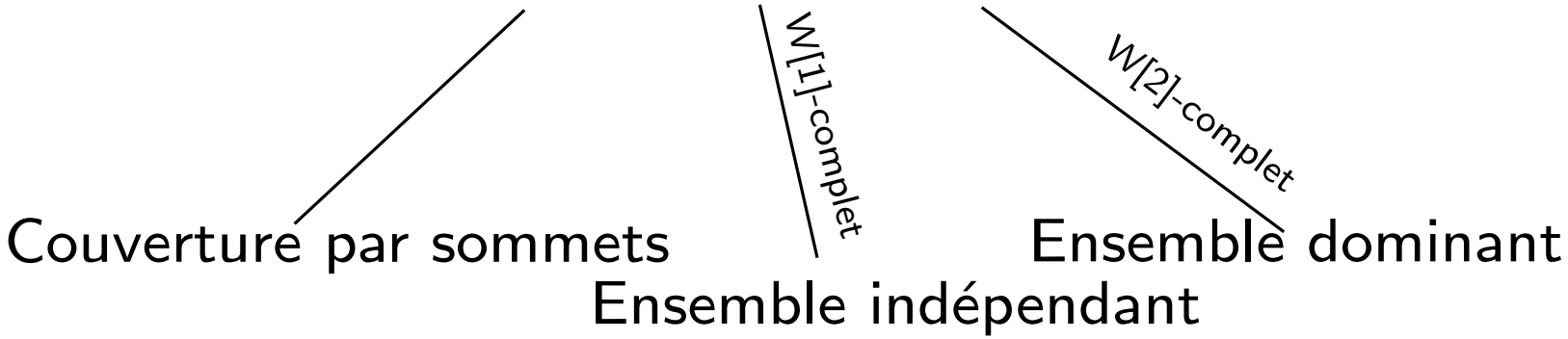
WEIGHTED CIRCUIT SAT[t]: étant donné un circuit de profondeur t alternant portes ET et OU, décider s'il existe une affectation de poids k la satisfaisant.

WEIGHTED CIRCUIT SAT[1] = WEIGHTED CNF-SAT

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

W[t]=problèmes paramétrés se réduisant à WEIGHTED CIRCUIT SAT[t]

$$\text{FTP} \subseteq \text{W[1]} \subseteq \text{W[2]} \subseteq \cdots \subseteq \text{W[P]} \subseteq \text{XP}$$



Cours 6: Complexité paramétrique

- Programmation dynamique et largeur arborescente
- 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