# Complexité paramétrée (6)

## Décomposition et largeur arborescente

Christophe PAUL
(CNRS - LIRMM)

Algorithmes pour les graphes de largeur arborescente bornée

- Ensemble Indépendant sur les arbres
- Décomposition arborescente
- Ensemble Indépendent paramétré par $tw(G)$
- Chemin Hamiltonien paramétré par $tw(G)$

Calcul et approximation de la largeur arborescente

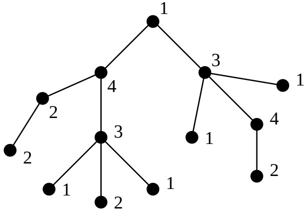
Théorème de Courcelle

- Logique du second ordre monadique
- Méta-théorème

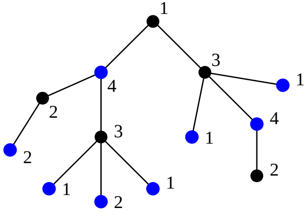
Réduction à largeur arborescente bornée

- Obstructions
- Sommet / arête inutile

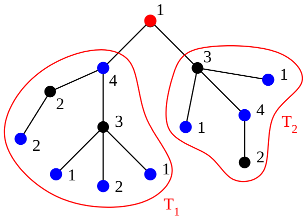
# Ensemble Indépendant (valué) sur les arbres



# Ensemble Indépendant (valué) sur les arbres



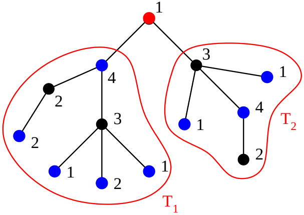
# Ensemble Indépendant (valué) sur les arbres




### Observations

1. tout sommet d'un arbre est un séparateur
2. l'union d'ensembles indépendants de composantes connexes distinctes est un ensemble indépendant

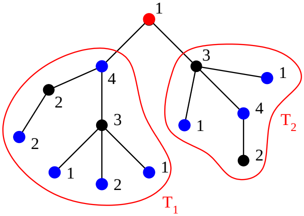
# Ensemble Indépendant (valué) sur les arbres




Soit $x$ la racine de $T$ et $x_1 \ldots x_l$ ses fils:

- $wIS(T, x) \longrightarrow$ ensemble indépendant max. contenant $x$
- $wIS(T, \overline{x}) \longrightarrow$ ensemble indépendant max. ne contenant pas $x$

# Ensemble Indépendant (valué) sur les arbres

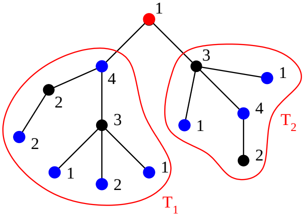



Soit $x$ la racine de $T$ et $x_1 \ldots x_l$ ses fils:

- $wIS(T, x) \longrightarrow$ ensemble indépendant max. contenant $x$
- $wIS(T, \overline{x}) \longrightarrow$ ensemble indépendant max. ne contenant pas $x$

$$\left\{ \begin{array}{lcl} wIS(T, x) & = & \sum_{i \in [l]} wIS(T, \overline{x_i}) \\ & & \end{array} \right.$$

# Ensemble Indépendant (valué) sur les arbres

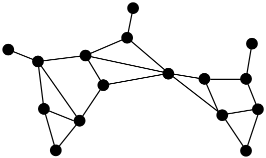



Soit $x$ la racine de $T$ et $x_1 \dots x_l$ ses fils:

- $wIS(T, x) \longrightarrow$ ensemble indépendant max. contenant $x$
- $wIS(T, \overline{x}) \longrightarrow$ ensemble indépendant max. ne contenant pas $x$

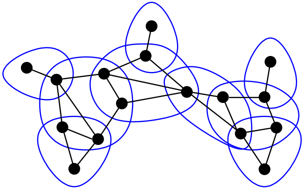
$$\begin{cases} wIS(T,x) & = & \sum_{i \in [l]} wIS(T, \overline{x_i}) \\ wIS(T,\overline{x}) & = & \sum_{i \in [l]} \max\{wIS(T,x_i), wIS(T,\overline{x_i})\} \end{cases}$$

# Décomposition arborescente



Une décomposition arborescente d'un graphe $G = (V, E)$ est une paire $(T, \{X_t : t \in T\})$ avec $T$ est un arbre et $\forall t \in T$, $V_t \subseteq V$, telle que

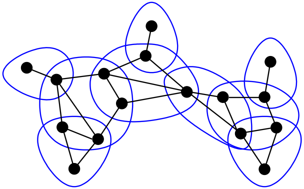
# Décomposition arborescente



Une décomposition arborescente d'un graphe $G = (V, E)$ est une paire $(T, \{X_t : t \in T\})$ avec $T$ est un arbre et $\forall t \in T$, $V_t \subseteq V$, telle que

- [couverture des sommets] $\forall x \in V$, $\exists t \in T$ tel que $x \in X_t$

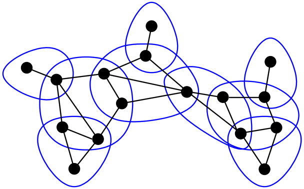
# Décomposition arborescente



Une décomposition arborescente d'un graphe $G = (V, E)$ est une paire $(T, \{X_t : t \in T\})$ avec $T$ est un arbre et $\forall t \in T$, $V_t \subseteq V$, telle que

- [couverture des sommets] $\forall x \in V$, $\exists t \in T$ tel que $x \in X_t$
- [couverture des arêtes] $\forall (x, y) \in E$, $\exists t \in T$ tel que $x, y \in X_t$

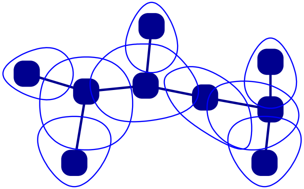
# Décomposition arborescente



Une décomposition arborescente d'un graphe $G = (V, E)$ est une paire $(T, \{X_t : t \in T\})$ avec $T$ est un arbre et $\forall t \in T$, $V_t \subseteq V$, telle que

- [couverture des sommets] $\forall x \in V$, $\exists t \in T$ tel que $x \in X_t$
- [couverture des arêtes] $\forall (x, y) \in E$, $\exists t \in T$ tel que $x, y \in X_t$
- [consistance] si $x \in V$ appartient à $X_{t_1} \cap X_{t_2}$, alors $\forall t \in T$ sur le chemin entre $t_1$ et $t_2$ dans $T$, $x \in X_t$.

# Décomposition arborescente



Une décomposition arborescente d'un graphe $G = (V, E)$ est une paire $(T, \{X_t : t \in T\})$ avec $T$ est un arbre et $\forall t \in T$, $V_t \subseteq V$, telle que

- [couverture des sommets] $\forall x \in V$, $\exists t \in T$ tel que $x \in X_t$
- [couverture des arêtes] $\forall (x, y) \in E$, $\exists t \in T$ tel que $x, y \in X_t$
- [consistance] si $x \in V$ appartient à $X_{t_1} \cap X_{t_2}$, alors $\forall t \in T$ sur le chemin entre $t_1$ et $t_2$ dans $T$, $x \in X_t$.

# Décomposition arborescente

- La largeur d'une décomposition arborescente $\mathcal{T}_G = (T, \{X_t : t \in T\})$ de $G$ est $width(\mathcal{T}_G) = \max_{t \in T} |X_t| - 1$

# Décomposition arborescente

- La largeur d'une décomposition arborescente $\mathcal{T}_G = (T, \{X_t : t \in T\})$ de $G$ est $width(\mathcal{T}_G) = \max_{t \in T} |X_t| - 1$
- La largeur arborescente d'un graphe $G$ est $$tw(G) = \min_{\mathcal{T}_G} width(\mathcal{T}_G)$$

# Décomposition arborescente

- La largeur d'une décomposition arborescente $\mathcal{T}_G = (T, \{X_t : t \in T\})$ de $G$ est $width(\mathcal{T}_G) = \max_{t \in T} |X_t| - 1$

- La largeur arborescente d'un graphe $G$ est
$$tw(G) = \min_{\mathcal{T}_G} width(\mathcal{T}_G)$$

Observation Soit $\mathcal{T}_G = (T, \{X_t : t \in T\})$ une décomposition arborescente de $G$

1. tout nœud $X_t$ est un séparateur de $G$.

# Décomposition arborescente

- La largeur d'une décomposition arborescente $\mathcal{T}_G = (T, \{X_t : t \in T\})$ de $G$ est
  $width(\mathcal{T}_G) = \max_{t \in T} |X_t| - 1$
- La largeur arborescente d'un graphe $G$ est
  $tw(G) = \min_{\mathcal{T}_G} width(\mathcal{T}_G)$

Observation Soit $\mathcal{T}_G = (T, \{X_t : t \in T\})$ une décomposition arborescente de $G$

1. tout nœud $X_t$ est un séparateur de $G$.
2. si $X_t$ et $X_{t'}$ sont deux nœuds tels que $t$ et $t'$ sont adjacents dans $T$, alors $X_t \cap X_{t'}$ est un séparateur de $G$.

# Décomposition arborescente

- La largeur d'une décomposition arborescente $\mathcal{T}_G = (T, \{X_t : t \in T\})$ de $G$ est
  $width(\mathcal{T}_G) = \max_{t \in T} |X_t| - 1$

- La largeur arborescente d'un graphe $G$ est
  $$tw(G) = \min_{\mathcal{T}_G} width(\mathcal{T}_G)$$

Observation Soit $\mathcal{T}_G = (T, \{X_t : t \in T\})$ une décomposition arborescente de $G$

1. tout nœud $X_t$ est un séparateur de $G$.
2. si $X_t$ et $X_{t'}$ sont deux nœuds tels que $t$ et $t'$ sont adjacents dans $T$, alors $X_t \cap X_{t'}$ est un séparateur de $G$.

Notations : Si on enracine $\mathcal{T}_G = (T, \{X_t : t \in T\})$, alors :

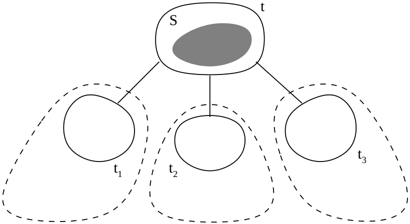
- $V_t$ : l'ensemble de sommets présents dans les descendants de $t$
- $G_t$ : le sous-graphe $G[V_t]$

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

$$\boxed{\forall S \subseteq X_t,\ IS(S,t)\ = \text{indép. max. de } G_t \text{ tq } X_t \cap IS(S,t) = S}$$

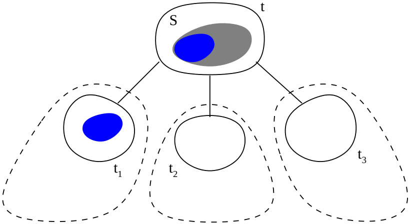
# Ensemble Indépendant (pondéré) paramétré $tw(G)$

$$\boxed{\forall S \subseteq X_t,\ IS(S,t) \ = \text{indép. max. de } G_t \text{ tq } X_t \cap IS(S,t) = S}$$



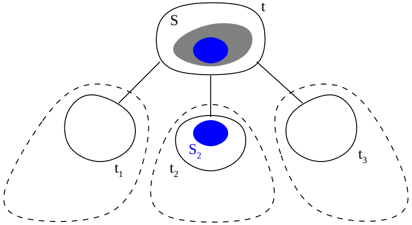
# Ensemble Indépendant (pondéré) paramétré $tw(G)$

$$\boxed{\forall S \subseteq X_t,\ IS(S,t)\ = \text{indép. max. de } G_t \text{ tq } X_t \cap IS(S,t) = S}$$



# Ensemble Indépendant (pondéré) paramétré $tw(G)$

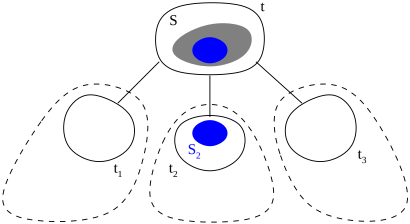
$$\boxed{\forall S \subseteq X_t,\ IS(S,t) \ = \text{indép. max. de } G_t \text{ tq } X_t \cap IS(S,t) = S}$$




Lemme : Si $S \subseteq X_t$ et $S_j = S \cap X_{t_j}$ alors $IS(S,t) \cap V_{t_j} = IS(S_j, t_j)$

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

$\boxed{\forall S \subseteq X_t,\ IS(S,t) \ = \text{indép. max. de } G_t \text{ tq } X_t \cap IS(S,t) = S}$

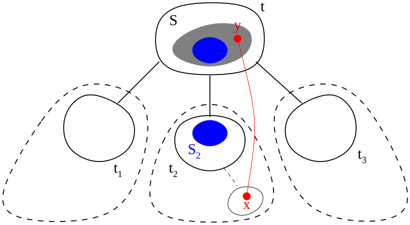



Lemme : Si $S \subseteq X_t$ et $S_j = S \cap X_{t_j}$ alors $IS(S,t) \cap V_{t_j} = IS(S_j, t_j)$

Hyp.: $IS(S,t) \cap V_{t_j}$ n'est pas max.

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

$$\boxed{\forall S \subseteq X_t,\ IS(S,t) \ = \text{indép. max. de } G_t \text{ tq } X_t \cap IS(S,t) = S}$$



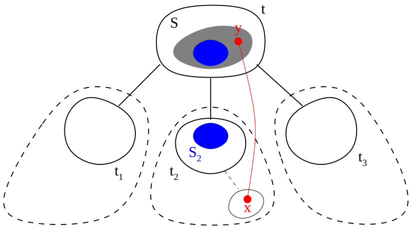

Lemme : Si $S \subseteq X_t$ et $S_j = S \cap X_{t_j}$ alors $IS(S,t) \cap V_{t_j} = IS(S_j, t_j)$

Hyp.: $IS(S,t) \cap V_{t_j}$ n'est pas max.

$\Rightarrow$ $\exists y \in S \setminus S_j$ et $\exists x \in IS(S_j, t_j) \setminus X_{t_j}$ tels que $xy \in E$

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

$$\boxed{\forall S \subseteq X_t,\ IS(S,t) \ = \text{indép. max. de } G_t \text{ tq } X_t \cap IS(S,t) = S}$$




Lemme : Si $S \subseteq X_t$ et $S_j = S \cap X_{t_j}$ alors $IS(S,t) \cap V_{t_j} = IS(S_j, t_j)$

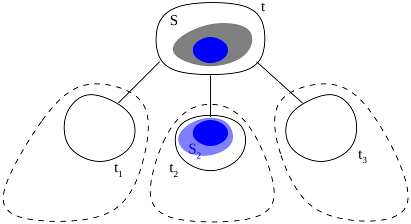
Hyp.: $IS(S,t) \cap V_{t_j}$ n'est pas max.

$\Rightarrow$ $\exists y \in S \setminus S_j$ et $\exists x \in IS(S_j, t_j) \setminus X_{t_j}$ tels que $xy \in E$

- contradiction : $X_{t_j}$ est un séparateur

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

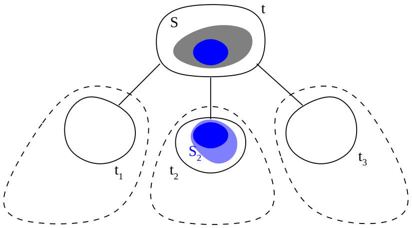
Idée de l'algo de programmation dynamique




Calcul de $IS(S,t)$ connaissant $IS(S_j^i, t_j)$, $\forall j \in [l]$, $\forall S_j^i \subseteq X_{t_j}$

# ENSEMBLE INDÉPENDANT (pondéré) paramétré $tw(G)$

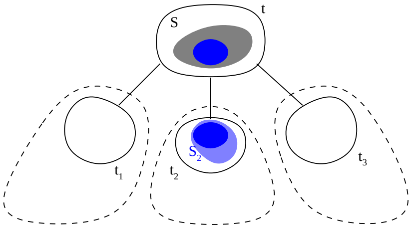
Idée de l'algo de programmation dynamique




Calcul de $IS(S,t)$ connaissant $IS(S_j^i, t_j)$, $\forall j \in [l]$, $\forall S_j^i \subseteq X_{t_j}$

# ENSEMBLE INDÉPENDANT (pondéré) paramétré $tw(G)$

Idée de l'algo de programmation dynamique

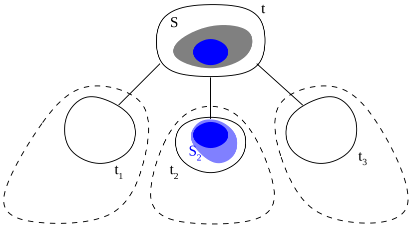



Calcul de $IS(S,t)$ connaissant $IS(S_j^i, t_j)$, $\forall j \in [l]$, $\forall S_j^i \subseteq X_{t_j}$

- vérifier que $S_j^i \cap X_t = S \cap X_{t_j} = S_j$ et $S_j \subseteq S_j^i$

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

Idée de l'algo de programmation dynamique

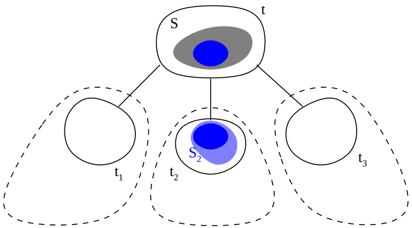



Calcul de $IS(S,t)$ connaissant $IS(S^i_j, t_j)$, $\forall j \in [l]$, $\forall S^i_j \subseteq X_{t_j}$

- vérifier que $S^i_j \cap X_t = S \cap X_{t_j} = S_j$ et $S_j \subseteq S^i_j$
- vérifier que $S^i_j$ est un indépendant

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

Idée de l'algo de programmation dynamique




Calcul de $IS(S,t)$ connaissant $IS(S_j^i, t_j)$, $\forall j \in [l]$, $\forall S_j^i \subseteq X_{t_j}$

- vérifier que $S_j^i \cap X_t = S \cap X_{t_j} = S_j$ et $S_j \subseteq S_j^i$
- vérifier que $S_j^i$ est un indépendant

$$IS(S,t) = \left\{ \sum_{i \in [l]} \max \begin{array}{l} |S| + \\ \{IS(S_j^i, t_j) - |S_j| \ : \\ S_j^i \cap X_t = S_j \ \& \ S_j \subseteq S_j^i \text{ indépendent}\} \end{array} \right.$$

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

$$IS(S,t) = \left\{ \begin{array}{ll} \sum_{i\in[l]} \max & \begin{array}{l} |S| + \\ \{IS(S_j^i, t_j) - |S_j| \ : \\ S_j^i \cap X_t = S_j \ \& \ S_j \subseteq S_j^i \text{ indépendent}\} \end{array} \end{array} \right.$$

Analyse de complexité:

- calcul de $IS(S,t)$ : $O(2^k.k^2.l)$

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

$$IS(S,t) = \left\{ \sum_{i \in [l]} \max \begin{array}{l} |S| + \\ \{IS(S^i_j, t_j) - |S_j| \ : \\ S^i_j \cap X_t = S_j \ \& \ S_j \subseteq S^i_j \text{ indépendent}\} \end{array} \right.$$

Analyse de complexité:

- calcul de $IS(S,t)$ : $O(2^k.k^2.l)$
- calcul de $IS(S,t)$ pour tout $S \subseteq X_t$ : $O(2^k.2^k.k^2.l)$

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

$$IS(S,t) = \left\{ \begin{array}{ll} & |S| + \\ \sum_{i\in[l]} \max & \{IS(S_j^i, t_j) - |S_j| \ : \\ & S_j^i \cap X_t = S_j \ \& \ S_j \subseteq S_j^i \text{ indépendent}\} \end{array} \right.$$

Analyse de complexité:

- calcul de $IS(S,t)$ : $O(2^k.k^2.l)$
- calcul de $IS(S,t)$ pour tout $S \subseteq X_t$ : $O(2^k.2^k.k^2.l)$
- calcul de la solution : $O(4^k.k^2.n)$

# Ensemble Indépendant (pondéré) paramétré $tw(G)$

$$IS(S,t) = \left\{ \sum_{i\in[l]} \max \begin{array}{l} |S| + \\ \{IS(S_j^i, t_j) - |S_j| \;: \\ S_j^i \cap X_t = S_j \;\&\; S_j \subseteq S_j^i \text{ indépendent}\} \end{array} \right.$$

Analyse de complexité:

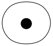
- calcul de $IS(S,t)$ : $O(2^k.k^2.l)$
- calcul de $IS(S,t)$ pour tout $S \subseteq X_t$ : $O(2^k.2^k.k^2.l)$
- calcul de la solution : $O(4^k.k^2.n)$

- il faut ajouter le temps de calcul d'une décomposition arborescente optimale!!!

# Décomposition arborescente simple

Une décomposition arborescente enracinée $\mathcal{T}_G = (T, \{X_t : t \in T\})$ est simple si tout nœud $t$ est d'un des quatre types suivant:

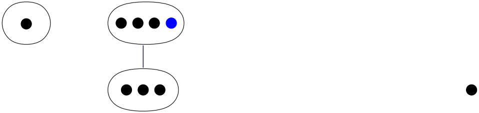
- Feuille : pas de fils et $|X_t| = 1$



# Décomposition arborescente simple

Une décomposition arborescente enracinée $\mathcal{T}_G = (T, \{X_t : t \in T\})$ est simple si tout nœud $t$ est d'un des quatre types suivant:

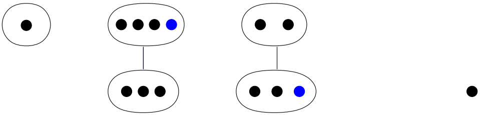
- Feuille : pas de fils et $|X_t| = 1$
- Ajout : un fils unique $t'$ et $X_t = X_{t'} \cup \{v\}$ avec $v \notin X_{t'}$



# Décomposition arborescente simple

Une décomposition arborescente enracinée $\mathcal{T}_G = (T, \{X_t : t \in T\})$ est simple si tout nœud $t$ est d'un des quatre types suivant:

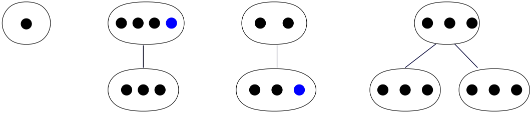
- Feuille : pas de fils et $|X_t| = 1$
- Ajout : un fils unique $t'$ et $X_t = X_{t'} \cup \{v\}$ avec $v \notin X_{t'}$
- Suppression : un fils unique $t'$ et $X_t = X_{t'} \setminus \{v\}$ avec $v \in X_{t'}$



# Décomposition arborescente simple

Une décomposition arborescente enracinée $\mathcal{T}_G = (T, \{X_t : t \in T\})$ est simple si tout nœud $t$ est d'un des quatre types suivant:

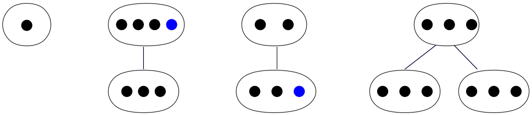
- Feuille : pas de fils et $|X_t| = 1$
- Ajout : un fils unique $t'$ et $X_t = X_{t'} \cup \{v\}$ avec $v \notin X_{t'}$
- Suppression : un fils unique $t'$ et $X_t = X_{t'} \setminus \{v\}$ avec $v \in X_{t'}$
- Fusion : deux fils $t_1$ et $t_2$ et $X_t = X_{t_1} = X_{t_2}$.



# Décomposition arborescente simple

Une décomposition arborescente enracinée $\mathcal{T}_G = (T, \{X_t : t \in T\})$ est simple si tout nœud $t$ est d'un des quatre types suivant:

- Feuille : pas de fils et $|X_t| = 1$
- Ajout : un fils unique $t'$ et $X_t = X_{t'} \cup \{v\}$ avec $v \notin X_{t'}$
- Suppression : un fils unique $t'$ et $X_t = X_{t'} \setminus \{v\}$ avec $v \in X_{t'}$
- Fusion : deux fils $t_1$ et $t_2$ et $X_t = X_{t_1} = X_{t_2}$.



Observation :

Une décomposition arborescente $\mathcal{T}$ de largeur $k$ avec $c$ nœuds peut être transformée en temps $O(kn)$ en une décomposition arborescente simple de largeur $k$ avec $k^2.c$ nœuds.

# Algorithme amélioré pour Ensemble Indépendant

Calcul de $IS(S,t)$ pour tout $S \subseteq X_t$

- $t$ est une feuille : trivial

# Algorithme amélioré pour Ensemble Indépendant

Calcul de $IS(S,t)$ pour tout $S \subseteq X_t$

- $t$ est une feuille : trivial
- $t$ est un nœud ajout : $X_t = X_{t'} \cup \{v\}$

$$IS(S,t) = \begin{cases} IS(S,t') & \text{si } v \notin S \\ IS(S \setminus \{v\}, t') + \omega(v) & \text{si } v \in S \text{ et } S \text{ indépendent} \\ -\infty & \text{sinon} \end{cases}$$

# Algorithme amélioré pour Ensemble Indépendant

Calcul de $IS(S,t)$ pour tout $S \subseteq X_t$

- $t$ est une feuille : trivial
- $t$ est un nœud ajout : $X_t = X_{t'} \cup \{v\}$

$$IS(S,t) = \begin{cases} IS(S,t') & \text{si } v \notin S \\ IS(S \setminus \{v\}, t') + \omega(v) & \text{si } v \in S \text{ et } S \text{ indépendent} \\ -\infty & \text{sinon} \end{cases}$$

- $t$ est un nœud suppression : $X_t = X_{t'} \setminus \{v\}$

$$IS(S,t) = \max\{IS(S,t'), IS(S \cup \{v\}, t')\}$$

# Algorithme amélioré pour Ensemble Indépendant

Calcul de $IS(S,t)$ pour tout $S \subseteq X_t$

- $t$ est une feuille : trivial
- $t$ est un nœud ajout : $X_t = X_{t'} \cup \{v\}$

$$IS(S,t) = \begin{cases} IS(S,t') & \text{si } v \notin S \\ IS(S \setminus \{v\}, t') + \omega(v) & \text{si } v \in S \text{ et } S \text{ indépendent} \\ -\infty & \text{sinon} \end{cases}$$

- $t$ est un nœud suppression : $X_t = X_{t'} \setminus \{v\}$

$$IS(S,t) = \max\{IS(S,t'), IS(S \cup \{v\}, t')\}$$

- $t$ est un nœud fusion : $X_t = X_{t_1} = X_{t_2}$

$$IS(S,t) = IS(S,t_1) + IS(S,t_2) - \omega(S)$$

# Algorithme amélioré pour ENSEMBLE INDÉPENDANT

Calcul de $IS(S,t)$ pour tout $S \subseteq X_t$

- $t$ est une feuille : trivial
- $t$ est un nœud ajout : $X_t = X_{t'} \cup \{v\}$

$$IS(S,t) = \begin{cases} IS(S,t') & \text{si } v \notin S \\ IS(S \setminus \{v\}, t') + \omega(v) & \text{si } v \in S \text{ et } S \text{ indépendent} \\ -\infty & \text{sinon} \end{cases}$$

- $t$ est un nœud suppression : $X_t = X_{t'} \setminus \{v\}$

$$IS(S,t) = \max\{IS(S,t'), IS(S \cup \{v\}, t')\}$$

- $t$ est un nœud fusion : $X_t = X_{t_1} = X_{t_2}$

$$IS(S,t) = IS(S,t_1) + IS(S,t_2) - \omega(S)$$

$$\boxed{\text{Complexité : } O(2^k.n)}$$

# Chemin Hamiltonien paramétré par $tw(G)$

Algorithmes pour les graphes de largeur arborescente bornée

- ENSEMBLE INDÉPENDANT sur les arbres
- Décomposition arborescente
- ENSEMBLE INDÉPENDENT paramétré par $tw(G)$
- CHEMIN HAMILTONIEN paramétré par $tw(G)$

Calcul et approximation de la largeur arborescente

Théorème de Courcelle

- Logique du second ordre monadique
- Méta-théorème

Réduction à largeur arborescente bornée

- Obstructions
- Sommet / arête inutile

# Théorème de Bodlaender

Théoreme [Arnborg, Corneil, Proskurowski]
Etant donnés une graphe $G$ et un entier $k$, décider si $tw(G) \leqslant k$ est un problème NP-difficile.

Théorème [Bodlaender]
Etant donnés une graphe $G$ et un entier $k$ fixé, il existe un algorithme de complexité $O(2^{k^3}.n)$ qui décide si $tw(G) \leqslant k$

# Théorème de Bodlaender

Théoreme [Arnborg, Corneil, Proskurowski]
Etant donnés une graphe $G$ et un entier $k$, décider si $tw(G) \leqslant k$ est un problème NP-difficile.

Théorème [Bodlaender]
Etant donnés une graphe $G$ et un entier $k$ fixé, il existe un algorithme de complexité $O(2^{k^3}.n)$ qui décide si $tw(G) \leqslant k$

Théorème [Diestel et al.]
Il existe un algorithme de complexité $O(3^{3k}.k.n)$ qui calcule une décomposition arborescente $\mathcal{T}$ tel que $width(\mathcal{T}) \leqslant 4k+1$ si $tw(G) \leqslant k$

# Théorème de Bodlaender

Théoreme [Arnborg, Corneil, Proskurowski]
Etant donnés une graphe $G$ et un entier $k$, décider si $tw(G) \leqslant k$ est un problème NP-difficile.

Théorème [Bodlaender]
Etant donnés une graphe $G$ et un entier $k$ fixé, il existe un algorithme de complexité $O(2^{k^3}.n)$ qui décide si $tw(G) \leqslant k$

Théorème [Diestel et al.]
Il existe un algorithme de complexité $O(3^{3k}.k.n)$ qui calcule une décomposition arborescente $\mathcal{T}$ tel que $width(\mathcal{T}) \leqslant 4k+1$ si $tw(G) \leqslant k$

Problème ouvert : existence d'un noyau polynomial

# Algorithme de Diestel et al.

## Quelques définitions

- $Y$ et $Z$, $|Y| = |Z|$ sont séparables si $V$ contient un ensemble $S$ tel que $|S| < |Y|$ et $G - S$ ne contient pas de chemin entre $Y \setminus S$ et $Z \setminus S$.

# Algorithme de Diestel et al.

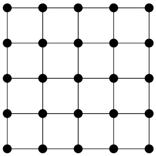
## Quelques définitions

- $Y$ et $Z$, $|Y| = |Z|$ sont séparables si $V$ contient un ensemble $S$ tel que $|S| < |Y|$ et $G - S$ ne contient pas de chemin entre $Y \setminus S$ et $Z \setminus S$.
- Un ensemble $X$ est $k$-lié si $|X| \geqslant k$ et $\forall\ Y, Z \subseteq X$, $|Y| = |Z| \leqslant k$, $Y$ and $Z$ sont non-séparables

# Algorithme de Diestel et al.

## Quelques définitions

- $Y$ et $Z$, $|Y| = |Z|$ sont séparables si $V$ contient un ensemble $S$ tel que $|S| < |Y|$ et $G - S$ ne contient pas de chemin entre $Y \setminus S$ et $Z \setminus S$.
- Un ensemble $X$ est $k$-lié si $|X| \geqslant k$ et $\forall\ Y, Z \subseteq X$, $|Y| = |Z| \leqslant k$, $Y$ and $Z$ sont non-séparables

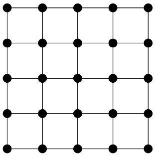


La grille $k \times k$ est $k$-liée

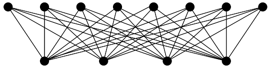
# Algorithme de Diestel et al.

## Quelques définitions

- $Y$ et $Z$, $|Y| = |Z|$ sont séparables si $V$ contient un ensemble $S$ tel que $|S| < |Y|$ et $G - S$ ne contient pas de chemin entre $Y \setminus S$ et $Z \setminus S$.
- Un ensemble $X$ est $k$-lié si $|X| \geqslant k$ et $\forall\ Y, Z \subseteq X$, $|Y| = |Z| \leqslant k$, $Y$ and $Z$ sont non-séparables



La grille $k \times k$ est $k$-liée



Dans le biparti complet $K_{2k,k}$:
$A$ est $2k$-lié
$B$ est $k$-lié

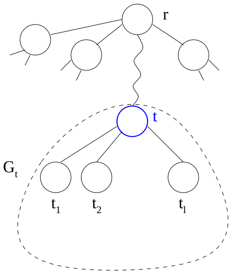
# Algorithme de Diestel et al.

Lemme : Si $G$ contient un ensemble $(k+1)$-lié $X$ tq $|X| \geqslant 3k$, alors $tw(G) \geqslant k$

# Algorithme de Diestel et al.

Lemme : Si $G$ contient un ensemble $(k+1)$-lié $X$ tq $|X| \geqslant 3k$, alors $tw(G) \geqslant k$

Soit $\mathcal{T}_G$ une décomposition arborescente tq $width(\mathcal{T}_G) < k$

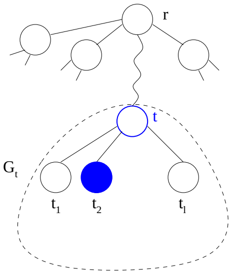



Soit $t$ le nœud le plus "bas" tq $|V_t \cap X| \geqslant 2k$

# Algorithme de Diestel et al.

Lemme : Si $G$ contient un ensemble $(k+1)$-lié $X$ tq $|X| \geqslant 3k$, alors $tw(G) \geqslant k$

Soit $\mathcal{T}_G$ une décomposition arborescente tq $width(\mathcal{T}_G) < k$




Soit $t$ le nœud le plus "bas" tq $|V_t \cap X| \geqslant 2k$

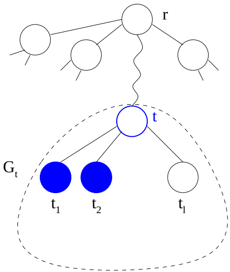
Si $\exists i \in [l]$ tq $|V_{t_i} \cap X| \geqslant k$, alors
$Y \subseteq V_{t_i} \cap X$, $|Y| = k$ et $Z \subset V \setminus V_{t_i}$, $|Z| = k$

Mais $S = X_{t_i} \cap X_t$ sépare $Y$ de $Z$ et
$|S| \leqslant k - 1$.

# Algorithme de Diestel et al.

Lemme : Si $G$ contient un ensemble $(k+1)$-lié $X$ tq $|X| \geqslant 3k$, alors $tw(G) \geqslant k$

Soit $\mathcal{T}_G$ une décomposition arborescente tq $width(\mathcal{T}_G) < k$




Soit $t$ le nœud le plus "bas" tq $|V_t \cap X| \geqslant 2k$

Sinon $W = V_{t_1} \cup \cdots \cup V_{t_i}$ avec $|W \cap X| \geqslant k$ et $|(W \setminus V_{t_j}) \cap X| < k$ pour $1 \leqslant j \leqslant i$

$Y \subseteq W \cap X$, $|Y| = k$ et $Z \subset V \setminus W$, $|Z| = k$

Mais $S = X_t$ sépare $Y$ de $Z$ et $|S| \leqslant k-1$.

# Algorithme de Diestel et al.

Lemme : Etant donné un ensemble de sommets $X$ d'un graphe $G$, on peut décider si $X$ est $k$-lié (pour $k \leqslant |X|$) en temps $O(f(k).n^{O(1)})$.

# Algorithme de Diestel et al.

Lemme : Etant donné un ensemble de sommets $X$ d'un graphe $G$, on peut décider si $X$ est $k$-lié (pour $k \leqslant |X|$) en temps $O(f(k).n^{O(1)})$.

- pour toute paire de sous-ensembles $Y$ et $Z$ de $X$ telle que $|Y| = |Z| \leqslant k$, on teste si $Y$ et $Z$ sont séparables (algorithme de flot)

# Algorithme de Diestel et al.

Lemme : Etant donné un ensemble de sommets $X$ d'un graphe $G$, on peut décider si $X$ est $k$-lié (pour $k \leqslant |X|$) en temps $O(f(k).n^{O(1)})$.

- pour toute paire de sous-ensembles $Y$ et $Z$ de $X$ telle que $|Y| = |Z| \leqslant k$, on teste si $Y$ et $Z$ sont séparables (algorithme de flot)
- complexité : $O(4^k.n^{O(1)})$

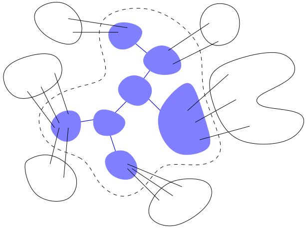
# Algorithme de Diestel et al.

Lemme : Etant donné un ensemble de sommets $X$ d'un graphe $G$, on peut décider si $X$ est $k$-lié (pour $k \leqslant |X|$) en temps $O(f(k).n^{O(1)})$.

- pour toute paire de sous-ensembles $Y$ et $Z$ de $X$ telle que $|Y| = |Z| \leqslant k$, on teste si $Y$ et $Z$ sont séparables (algorithme de flot)
- complexité : $O(4^k.n^{O(1)})$

Remarque : Si $X$ n'est pas $k$-lié, on trouve deux sous-ensembles séparables.

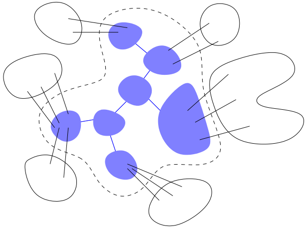
# Algorithme de Diestel et al.



Principe

- On ajoute les sommets dans $U$ de manière gloutonne

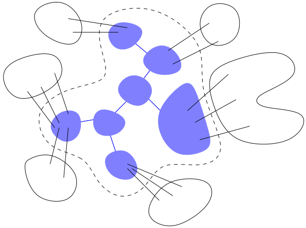
# Algorithme de Diestel et al.



Principe

- On ajoute les sommets dans $U$ de manière gloutonne
- On maintient une décomposition arborescente $\mathcal{T}_U$ tq $width(\mathcal{T}_U) \leqslant 4k$

# Algorithme de Diestel et al.

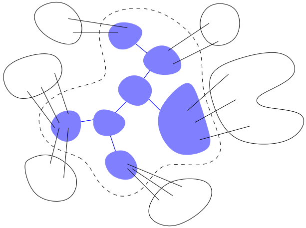


Principe

- On ajoute les sommets dans $U$ de manière gloutonne
- On maintient une décomposition arborescente $\mathcal{T}_U$ tq $width(\mathcal{T}_U) \leqslant 4k$

- Invariant : Chaque composante connexe $C$ de $G - U$ possède au plus $3k$ voisins dans $U$,
  et il existe un nœud $t$ tel que $V_t$ les contient tous.

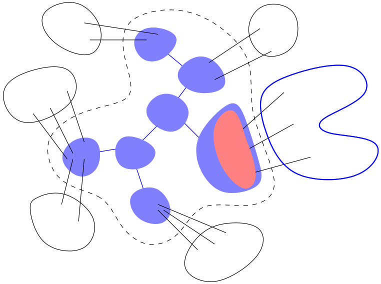
# Algorithme de Diestel et al.



Principe

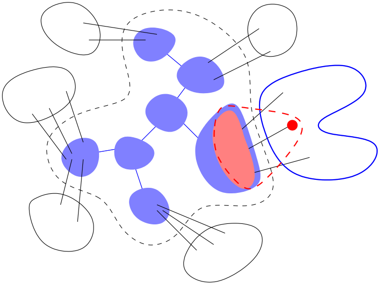
- On ajoute les sommets dans $U$ de manière gloutonne
- On maintient une décomposition arborescente $\mathcal{T}_U$ tq $width(\mathcal{T}_U) \leqslant 4k$
- Invariant : Chaque composante connexe $C$ de $G - U$ possède au plus $3k$ voisins dans $U$,
  et il existe un nœud $t$ tel que $V_t$ les contient tous.
- Initialement on choisit $V_1$ de taille $3k$.

# Algorithme de Diestel et al.



Soit $X$ les voisins d'une composante $C$ et $t$ le nœud tq $X \subseteq X_t$.

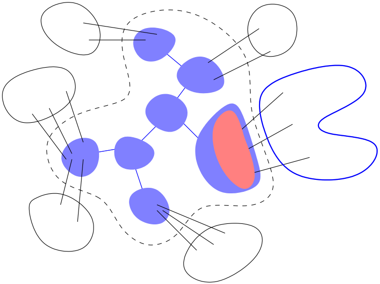
# Algorithme de Diestel et al.



Soit $X$ les voisins d'une composante $C$ et $t$ le nœud tq $X \subseteq X_t$.

- Si $|X| < 3k$ : on ajoute un nœud $t'$ voisin de $t$ tel que $X_{t'} = \{x\} \cup X$ avec $x \in C$ un voisin de $X_t$

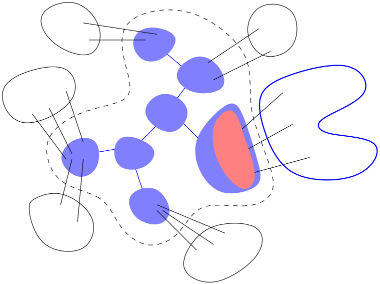
# Algorithme de Diestel et al.



Soit $X$ les voisins d'une composante $C$ et $t$ le nœud tq $X \subseteq X_t$.

- Si $|X| = 3k$ : on teste si $X$ est $(k+1)$-lié

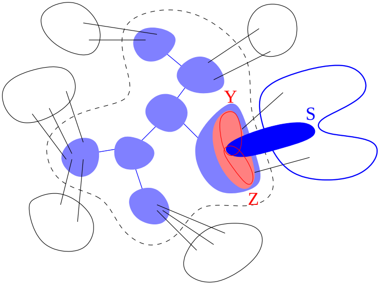
# Algorithme de Diestel et al.



Soit $X$ les voisins d'une composante $C$ et $t$ le nœud tq $X \subseteq X_t$.

- Si $|X| = 3k$ : on teste si $X$ est $(k+1)$-lié
  1. Si $X$ est $(k+1)$-lié alors $tw(G) \geqslant k+1$

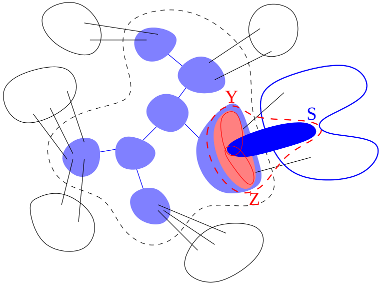
# Algorithme de Diestel et al.




Soit $X$ les voisins d'une composante $C$ et $t$ le nœud tq $X \subseteq X_t$.

- Si $|X| = 3k$ : on teste si $X$ est $(k+1)$-lié
  1. Si $X$ est $(k+1)$-lié alors $tw(G) \geqslant k+1$
  2. Sinon on trouve $Y$, $Z$ et $S$ tq $|S| < |Y| = |Z| \leqslant k+1$ et $S$ sépare $Y$ et $Z$

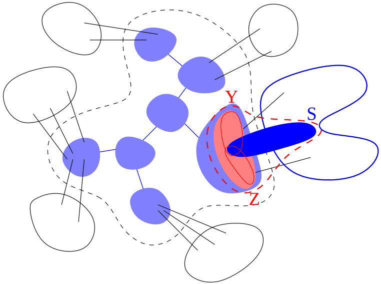
# Algorithme de Diestel et al.




Soit $X$ les voisins d'une composante $C$ et $t$ le nœud tq $X \subseteq X_t$.

- Si $|X| = 3k$ : on teste si $X$ est $(k+1)$-lié
  1. Si $X$ est $(k+1)$-lié alors $tw(G) \geqslant k+1$
  2. Sinon on trouve $Y$, $Z$ et $S$ tq $|S| < |Y| = |Z| \leqslant k+1$ et $S$ sépare $Y$ et $Z$
     On crée un nœud $t'$ voisin de $t$ tel que $X_{t'} = (S \cap C) \cup X$

# Algorithme de Diestel et al.




Soit $X$ les voisins d'une composante $C$ et $t$ le nœud tq $X \subseteq X_t$.

- Si $|X| = 3k$ : on teste si $X$ est $(k+1)$-lié
  1. Si $X$ est $(k+1)$-lié alors $tw(G) \geqslant k+1$
  2. Sinon on trouve $Y$, $Z$ et $S$ tq $|S| < |Y| = |Z| \leqslant k+1$ et $S$ sépare $Y$ et $Z$
     On crée un nœud $t'$ voisin de $t$ tel que $X_{t'} = (S \cap C) \cup X$

     Obs: les voisins de chaque nouvelle composante $C' \subseteq C$ sont dans $(X \setminus Z) \cup (S \cap C)$ ou dans $(X \setminus Y) \cup (S \cap C)$ $\Rightarrow$ au plus $3k$ voisins

Algorithmes pour les graphes de largeur arborescente bornée

- ENSEMBLE INDÉPENDANT sur les arbres
- Décomposition arborescente
- ENSEMBLE INDÉPENDENT paramétré par $tw(G)$
- CHEMIN HAMILTONIEN paramétré par $tw(G)$

Calcul et approximation de la largeur arborescente

Théorème de Courcelle

- Logique du second ordre monadique
- Méta-théorème

Réduction à largeur arborescente bornée

- Obstructions
- Sommet / arête inutile

# Logique du second ordre monadique sur les graphes

On représente un graphe $G = (V, E)$ à l'aide de la structure $\mathcal{G} = (U, Vertex, Edge, I)$ où

- $U = V \cup E$ est l'univers
- *Vertex* et *Edge* sont des relations unaires permettant de distinguer les sommets et les arêtes
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ est la relation d'incidence.

# Logique du second ordre monadique sur les graphes

On représente un graphe $G = (V, E)$ à l'aide de la structure $\mathcal{G} = (U, Vertex, Edge, I)$ où

- $U = V \cup E$ est l'univers
- *Vertex* et *Edge* sont des relations unaires permettant de distinguer les sommets et les arêtes
- $I = \{(v, e) \mid v \in V, e \in E, v \in e\}$ est la relation d'incidence.

Une formule MSOL est construite à partir

- des connecteurs logiques $\vee$, $\wedge$, $\Rightarrow$, $\neg$, $=$, $\neq$
- prédicats $adj(u, v)$ et $inc(e, v)$
- des quantificateurs $\exists$, $\forall$ sur des variables de sommets / arêtes ou d'ensembles de sommets / arêtes
- des relations $\in$, $\subseteq$ sur les ensemble de sommets / arêtes

# Logique du second ordre monadique sur les graphes

Exemple : $G$ est un graphe connexe

- pour toute bipartition de $V$, il existe une arête tranverse

# Logique du second ordre monadique sur les graphes

Exemple : $G$ est un graphe connexe

- pour toute bipartition de $V$, il existe une arête tranverse

$\forall V_1, V_2, [\ \forall v \in V, (v \in V_1 \vee v \in V_2) \wedge (v \in V_1 \Rightarrow v \notin V_2) \wedge (v \in V_2 \Rightarrow v \notin V_1)\ ]$
$\wedge\ \ \exists v_1 \in V_1, \exists v_2 \in V_2, \exists e \in E, inc(v_1, e) \wedge inc(v_2, e)$

# Logique du second ordre monadique sur les graphes

Exemple : $G$ est un graphe connexe

- pour toute bipartition de $V$, il existe une arête tranverse

$\forall V_1, V_2, [\ \forall v \in V, (v \in V_1 \vee v \in V_2) \wedge (v \in V_1 \Rightarrow v \notin V_2) \wedge (v \in V_2 \Rightarrow v \notin V_1)\ ]$
$\wedge\ \ \exists v_1 \in V_1, \exists v_2 \in V_2, \exists e \in E, inc(v_1, e) \wedge inc(v_2, e)$

Exercice : Exprimer qu'un graphe $G$

- possède un vertex cover, un ensemble indépendent, un ensemble dominant de taille $k$...
- est $k$-colorable
- possède un cycle hamiltonien

Théorème [Courcelle]

Toute propriété de graphe exprimable en MSOL peut être testée en temps $O(f(k).n)$ sur les graphes de largeur arborescente au plus $k$ (pour $k$ fixé).

Théorème [Courcelle]
Toute propriété de graphe exprimable en MSOL peut être testée en temps $O(f(k).n)$ sur les graphes de largeur arborescente au plus $k$ (pour $k$ fixé).

Corollaire : Vertex Cover, Ensemble Indépendant, Dominating Set paramétrés par la largeur arborescente sont des problèmes FPT

Théorème [Courcelle]
Toute propriété de graphe exprimable en MSOL peut être testée en temps $O(f(k).n)$ sur les graphes de largeur arborescente au plus $k$ (pour $k$ fixé).

Corollaire : Vertex Cover, Ensemble Indépendant, Dominating Set paramétrés par la largeur arborescente sont des problèmes FPT

Remarque : la fonction $f(k)$ dépend de la structure de la formule

**Théorème [Courcelle]**
Toute propriété de graphe exprimable en MSOL peut être testée en temps $O(f(k).n)$ sur les graphes de largeur arborescente au plus $k$ (pour $k$ fixé).

**Corollaire :** VERTEX COVER, ENSEMBLE INDÉPENDANT, DOMINATING SET paramétrés par la largeur arborescente sont des problèmes FPT

**Remarque :** la fonction $f(k)$ dépend de la structure de la formule

**Utilisation du théorème de Courcelle**

1. Montrer que le problème est exprimable en MSOL
2. Montrer que si $tw(G)$ est trop grande (par rapport au paramètre $k$), alors
   - l'instance est négative ($G$ contient une obstruction)
   - ou l'instance peut-être réduite (on diminue $tw(G)$)

Algorithmes pour les graphes de largeur arborescente bornée

- Ensemble Indépendant sur les arbres
- Décomposition arborescente
- Ensemble Indépendent paramétré par $tw(G)$
- Chemin Hamiltonien paramétré par $tw(G)$

Calcul et approximation de la largeur arborescente

Théorème de Courcelle

- Logique du second ordre monadique
- Méta-théorème

Réduction à largeur arborescente bornée

- Obstructions
- Sommet / arête inutile

# Dominating set dans les graphes planaires (bidim)