Algorithmes exacts modérément exponentiels
Programmation dynamique
Conclusion générale

Ioan Todinca

LIFO, Université d'Orléans et INSA CVL

École jeunes chercheurs du GdR Informatique Mathématique
Orléans, le 31 mars 2015

Plan

Programmation dynamique

- Problème du VOYAGEUR DE COMMERCE
- Calcul de la LARGEUR ARBORESCENTE (treewidth)
- Discussion

Conclusion générale du chapitre « Algorithmes exacts. . . »

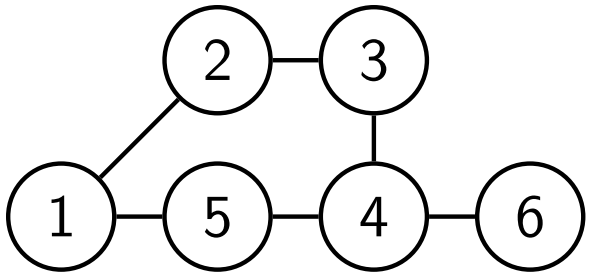
La programmation dynamique en quelques lignes

- R. Bellman, années 1950
- Votre premier contact avec l'algorithmique exacte pour problèmes NP-difficiles : SAC À DOS

- Principe :
  1. trouver une solution récursive au problème
  2. remplacer l'approche récursive par un algorithme itératif qui traite les sous-problèmes dans un ordre bien choisi et mémorise les solutions partielles
- Avantage : ne pas traiter plusieurs fois le même sous-problème
- Difficulté : il faut souvent généraliser le problème (introduction de variables)

Problème du VOYAGEUR DE COMMERCE

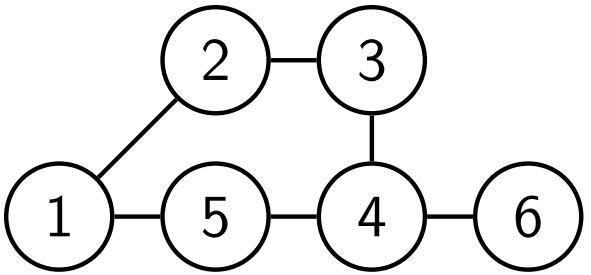
Entrée : graphe $G = (V, E)$,
poids $w : E \to \mathbb{N}$ sur les arêtes
Sortie : un *tour* T passant
au moins une fois par chaque sommet,
de poids minimum



Un algorithme facile de complexité $\mathcal{O}^*(n!)$

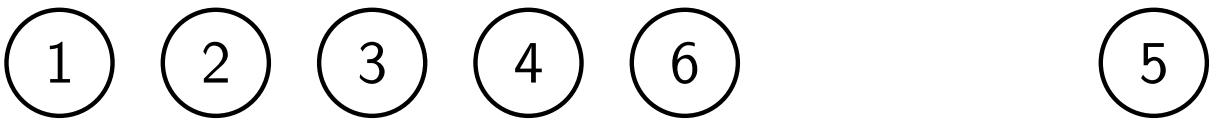
Problème du VOYAGEUR DE COMMERCE

Entrée : graphe $G = (V, E)$,
poids $w : E \to \mathbb{N}$ sur les arêtes
Sortie : un *tour* T passant
au moins une fois par chaque sommet,
de poids minimum



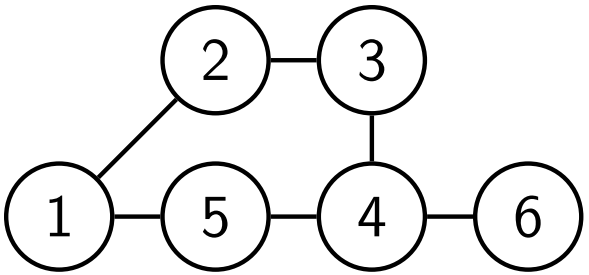
Un algorithme facile de complexité $\mathcal{O}^*(n!)$

- Deviner l'ordre $(x_1, x_2, \ldots, x_n)$ dans lequel le tour T rencontre les sommets pour la première fois



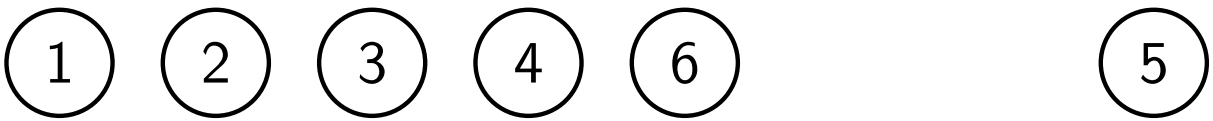
Problème du VOYAGEUR DE COMMERCE

Entrée : graphe $G = (V, E)$,
poids $w : E \to \mathbb{N}$ sur les arêtes
Sortie : un *tour* T passant
au moins une fois par chaque sommet,
de poids minimum



Un algorithme facile de complexité $\mathcal{O}^*(n!)$

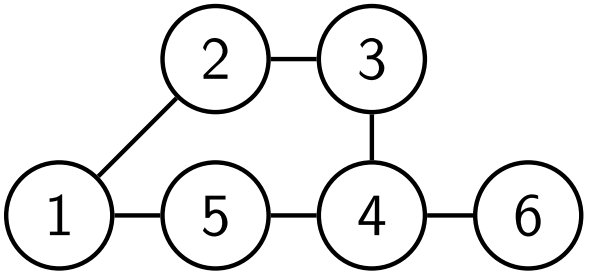
- Deviner l'ordre $(x_1, x_2, \ldots, x_n)$ dans lequel le tour T rencontre les sommets pour la première fois



$$w(T) = \sum_{i=1}^{n} dist(x_i, x_{i+1})$$

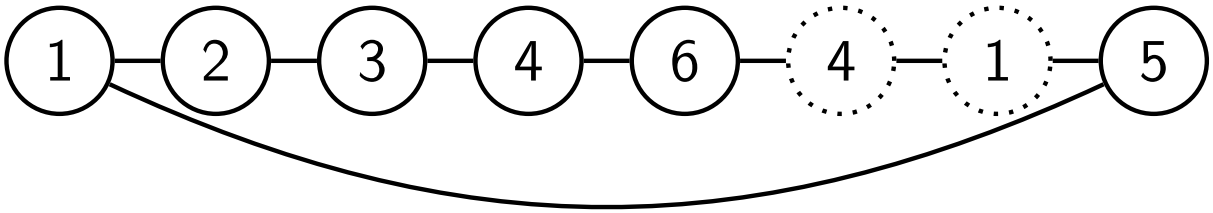
Problème du VOYAGEUR DE COMMERCE

Entrée : graphe $G = (V, E)$,
poids $w : E \to \mathbb{N}$ sur les arêtes
Sortie : un *tour* T passant
au moins une fois par chaque sommet,
de poids minimum



Un algorithme facile de complexité $\mathcal{O}^*(n!)$

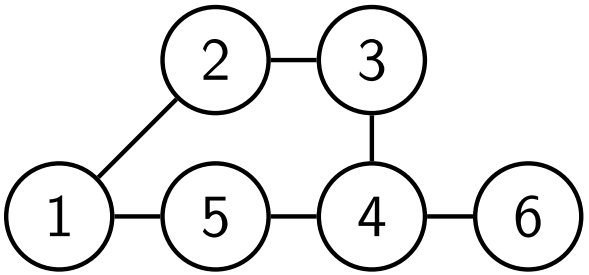
- Deviner l'ordre $(x_1, x_2, \ldots, x_n)$ dans lequel le tour T rencontre les sommets pour la première fois



$$w(T) = \sum_{i=1}^{n} dist(x_i, x_{i+1})$$

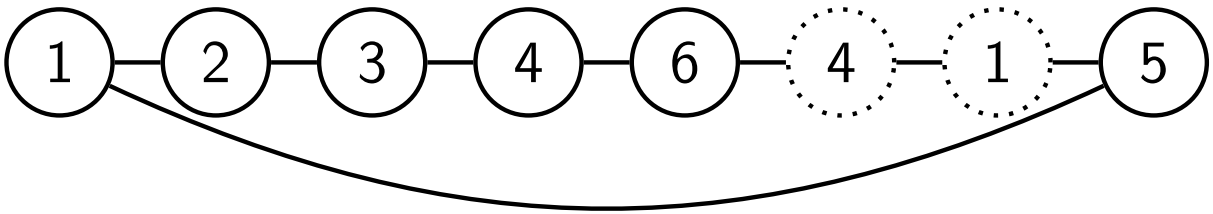
Problème du VOYAGEUR DE COMMERCE

Entrée : graphe $G = (V, E)$,
poids $w : E \to \mathbb{N}$ sur les arêtes
Sortie : un *tour* T passant
au moins une fois par chaque sommet,
de poids minimum



Un algorithme facile de complexité $\mathcal{O}^*(n!)$

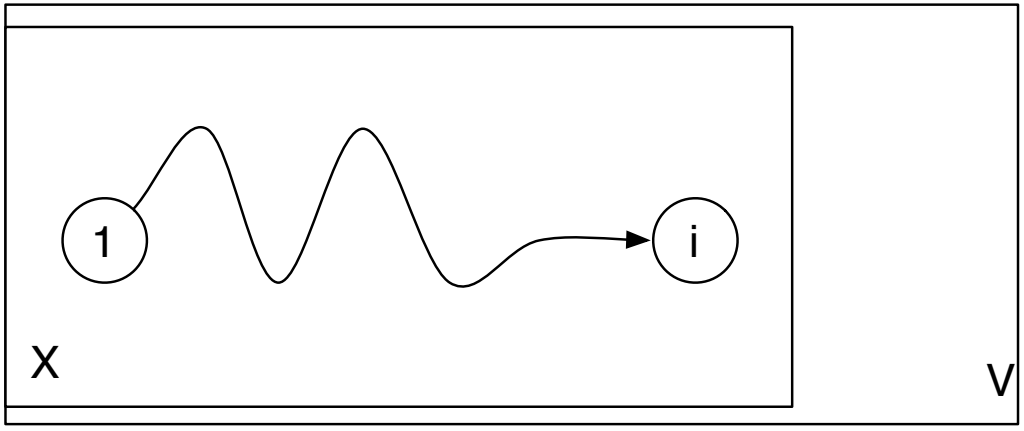
- Deviner l'ordre $(x_1, x_2, \ldots, x_n)$ dans lequel le tour T rencontre les sommets pour la première fois ($n!$ possibilités)



$$w(T) = \sum_{i=1}^{n} dist(x_i, x_{i+1})$$

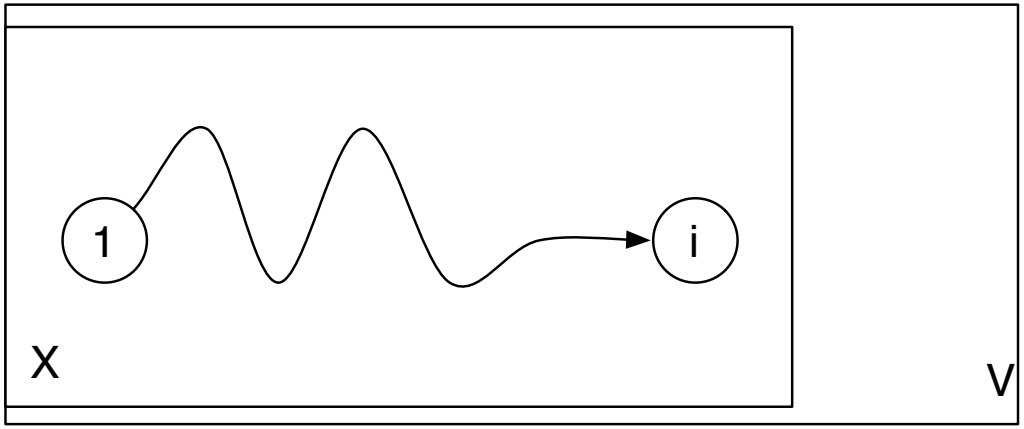
Voyageur de commerce en $\mathcal{O}^*(2^n)$

$OPT(X, i)$: le poids minimum d'un chemin de 1 à i passant par tous les sommets de X



Voyageur de commerce en $\mathcal{O}^*(2^n)$

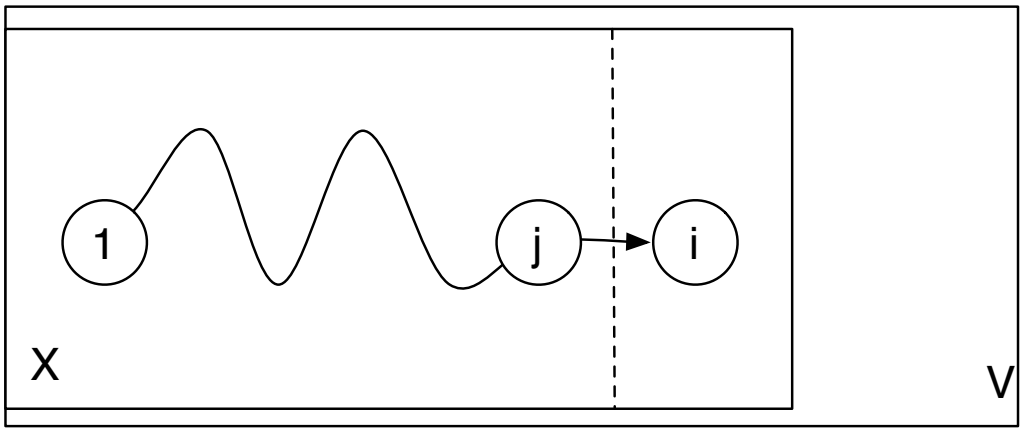
$OPT(X, i)$: le poids minimum d'un chemin de 1 à i passant par tous les sommets de X



- $OPT(\{1\}, 1) = 0$

Voyageur de commerce en $\mathcal{O}^*(2^n)$

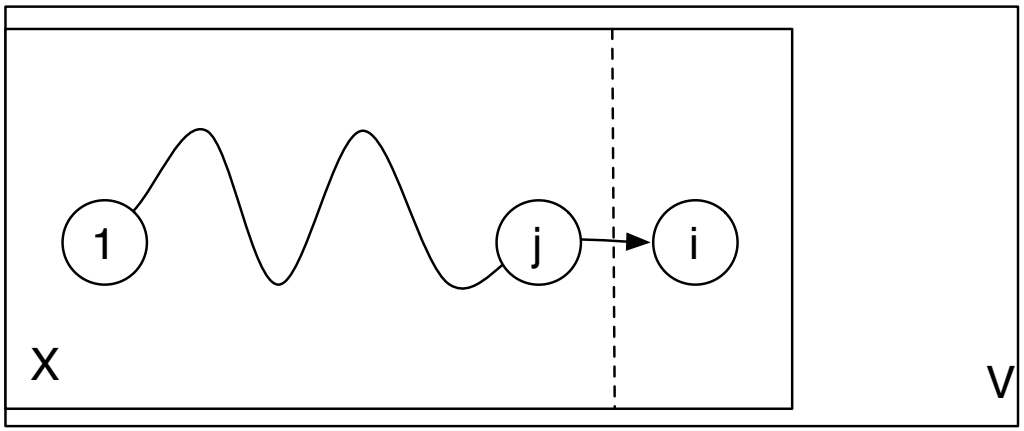
$OPT(X, i)$: le poids minimum d'un chemin de 1 à i passant par tous les sommets de X



- $OPT(\{1\}, 1) = 0$
- $OPT(X, i) = \min_{j \in X \setminus \{i,1\}} OPT(X \setminus \{i\}, j) + dist(j, i)$

Voyageur de commerce en $\mathcal{O}^*(2^n)$

$OPT(X, i)$: le poids minimum d'un chemin de 1 à i passant par tous les sommets de X

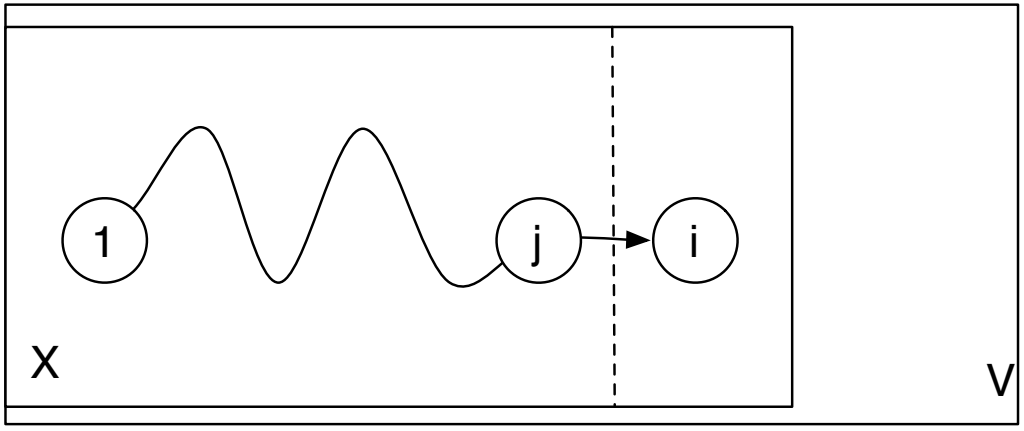


- $OPT(\{1\}, 1) = 0$
- $OPT(X, i) = \min_{j \in X \setminus \{i,1\}} OPT(X \setminus \{i\}, j) + dist(j, i)$

Calculer ces valeurs pour chaque $X \subseteq V$ par taille croissante

Voyageur de commerce en $\mathcal{O}^*(2^n)$

$OPT(X, i)$: le poids minimum d'un chemin de 1 à i passant par tous les sommets de X



- $OPT(\{1\}, 1) = 0$
- $OPT(X, i) = \min_{j \in X \setminus \{i,1\}} OPT(X \setminus \{i\}, j) + dist(j, i)$

Calculer ces valeurs pour chaque $X \subseteq V$ par taille croissante

Complexité : $\mathcal{O}^*(2^n)$ [Bellman 1962, Held, Karp 1962]

Algorithme Voyageur de commerce

Algorithme 1: VOYAGEUR DE COMMERCE

Entrées : Un graphe $G = (V, E)$, des poids $w : E \to \mathbb{N}$
Sorties : Le poids du tour minimum

$OPT[\{1\}, 1] \leftarrow 0$
pour chaque $X \subseteq V$, $\{1\} \subsetneq X \subseteq V$ **par taille croissante faire**
  pour chaque $i \in X \setminus \{1\}$ **faire**
    $OPT[X, i] \leftarrow \min_{j \in X \setminus \{1,i\}} OPT[X \setminus \{i\}, j] + dist_G(i, j)$
retourner $\min_{i \in V \setminus \{1\}} OPT[V, i] + dist_G(i, 1)$

Complexité : $\mathcal{O}^*(2^n)$ [Bellman 1962, Held, Karp 1962]

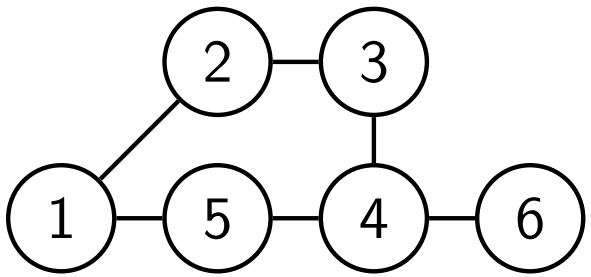
Approche de Held et Karp

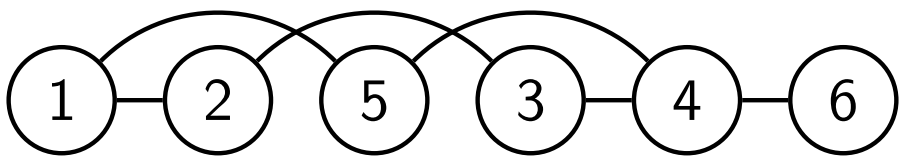
Généralisable à beaucoup de problèmes calculant un ordre total sur les sommets (graph layout)

Exercice : LARGEUR DE DÉCOUPE

Entrée : graphe $G = (V, E)$, entier k

Sortie : un ordre total sur les sommets t. q., pour tout i, il y ait au plus k arêtes entre les i premiers sommets et les $n - i$ derniers.

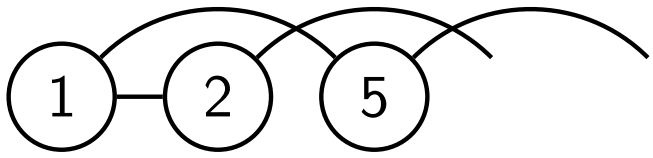




Complexité $\mathcal{O}^*(2^n)$

Largeur de découpe en $\mathcal{O}^*(2^n)$

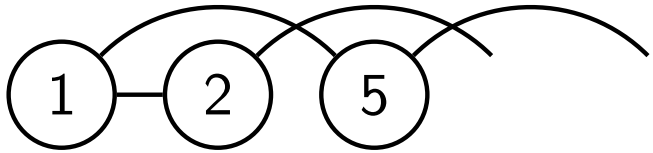
$OPT[X] = vrai$ si l'on peut commencer par les sommets de X



$OPT[X] = vrai$ si et seulement si

Largeur de découpe en $\mathcal{O}^*(2^n)$

$OPT[X] = vrai$ si l'on peut commencer par les sommets de X

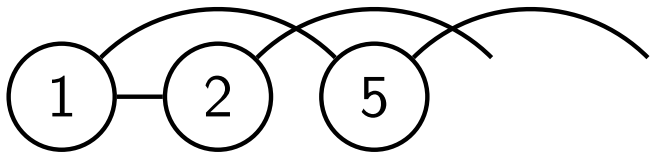


$OPT[X] = vrai$ si et seulement si

- il y a au plus k arêtes entre X et $V \setminus X$, et

Largeur de découpe en $\mathcal{O}^*(2^n)$

$OPT[X] = vrai$ si l'on peut commencer par les sommets de X

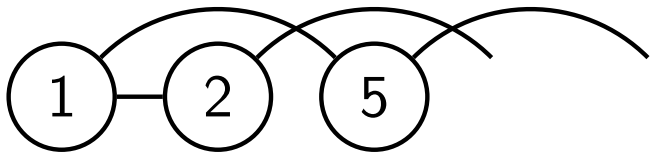


$OPT[X] = vrai$ si et seulement si

- il y a au plus k arêtes entre X et $V \setminus X$, et
- il existe $i \in X$ tel que $OPT[X \setminus \{i\}]$ soit $vrai$

Largeur de découpe en $\mathcal{O}^*(2^n)$

$OPT[X] = vrai$ si l'on peut commencer par les sommets de X



$OPT[X] = vrai$ si et seulement si

- il y a au plus k arêtes entre X et $V \setminus X$, et
- il existe $i \in X$ tel que $OPT[X \setminus \{i\}]$ soit $vrai$

Algorithme 5: Largeur de découpe(G, k)

$OPT[\emptyset] \leftarrow vrai$

pour chaque $X \subseteq V$ **par ordre croissant faire**

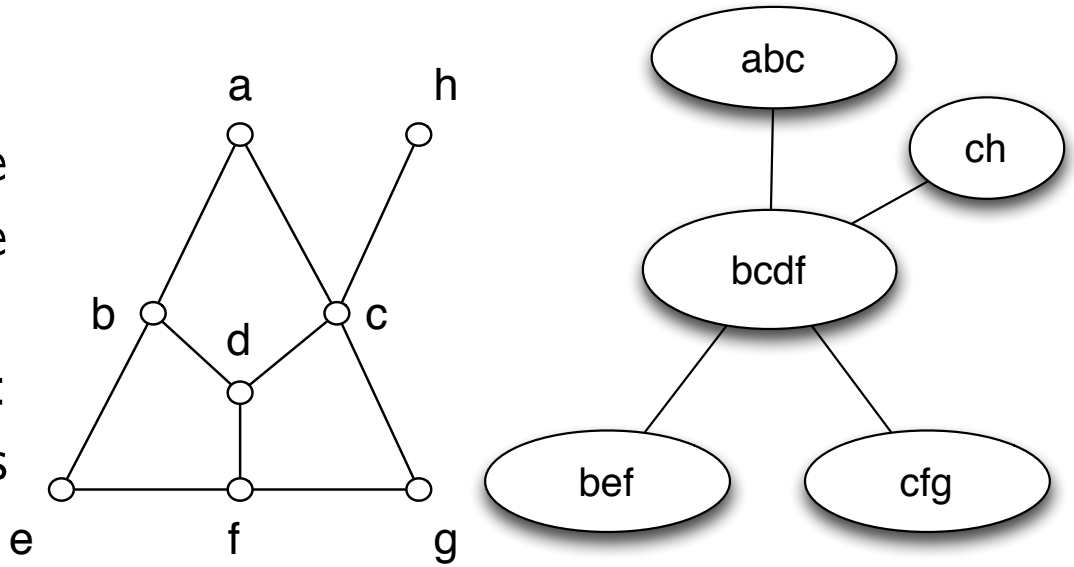
  calculer $OPT[X]$ en fonction des $OPT[X \setminus \{i\}]$

retourner $OPT[V]$

Décompositions arborescentes

[Robertson, Seymour 84/86]

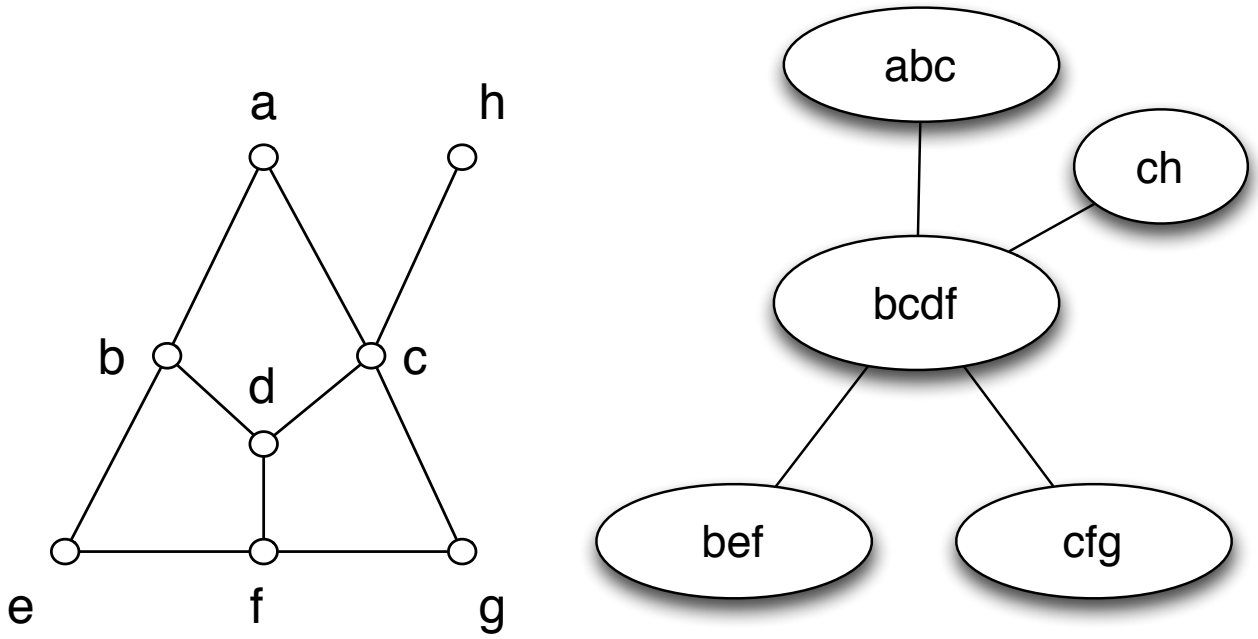
Représentation d'un graphe quelconque sous forme d'arbre « généralisé »
Largeur arborescente $tw(G)$: écart entre G et la classe des arbres



De nombreux problèmes peuvent être résolus efficacement pour des graphes de petite largeur arborescente, typiquement en $\mathcal{O}(f(tw) \cdot n)$ [Courcelle 1990]

Décomposition arborescente (tree decomposition)

Décomposition arborescente de G : arbre, les nœuds sont des sacs

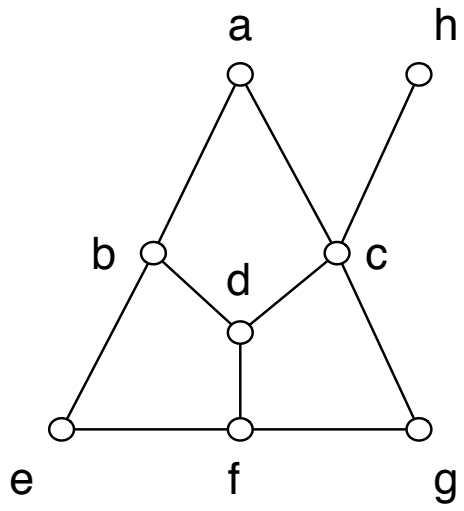


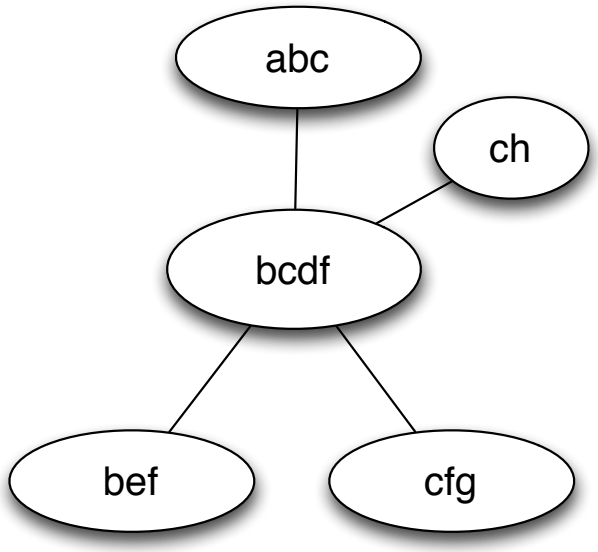
1. chaque sommet et chaque arête de G sont couverts par au moins un sac
2. pour chaque sommet x de G, les sacs le contenant induisent un sous-arbre connexe de l'arbre de décomposition

Largeur arborescente (treewidth)

Largeur d'une décomposition : taille du plus gros sac, moins un

Largeur de G : largeur minimum, parmi toutes les décompositions





classe	**largeur arborescente**
arbres, forêts	1
cycles, planaires extérieurs	2
graphes complets	$n-1$

Calcul de la largeur arborescente

Décider si $\text{tw}(G) \leq k$

1. NP-difficile [Arnborg, Corneil, Proskuroski 1987]
2. $\mathcal{O}(n^{k+const})$ [Arnborg, Corneil, Proskuroski 1987]
3. $\mathcal{O}(f(k) \cdot n)$ [Bodlaender 1996]
4. $\mathcal{O}^*(2^n)$ en adaptant [Arnborg, Corneil, Proskuroski 1987]

Calcul de la largeur arborescente

Décider si $\mathrm{tw}(G) \leq k$

1. NP-difficile [Arnborg, Corneil, Proskuroski 1987]
2. $\mathcal{O}(n^{k+const})$ [Arnborg, Corneil, Proskuroski 1987]
3. $\mathcal{O}(f(k) \cdot n)$ [Bodlaender 1996]
4. $\mathcal{O}^*(2^n)$ en adaptant [Arnborg, Corneil, Proskuroski 1987]
   ... ou [Held, Karp 1962] (!)

Calcul de la largeur arborescente

Décider si $\mathrm{tw}(G) \leq k$

1. NP-difficile [Arnborg, Corneil, Proskuroski 1987]
2. $\mathcal{O}(n^{k+const})$ [Arnborg, Corneil, Proskuroski 1987]
3. $\mathcal{O}(f(k) \cdot n)$ [Bodlaender 1996]
4. $\mathcal{O}^*(2^n)$ en adaptant [Arnborg, Corneil, Proskuroski 1987]
   ... ou [Held, Karp 1962] (!)
5. $\mathcal{O}(1.7347^n)$ [Fomin, T., Villanger 2015]

Pour commencer : intuitions

Entrée : graphe G et un ensemble de sacs Π
Sortie : une décomposition utilisant uniquement des sacs de Π

- C'est déjà l'idée de [Arnborg, Corneil, Proskuroski 1987]
- Conduirait à un algorithme en $\mathcal{O}^*(2^n)$ en prenant tous les sacs possibles

Pour commencer : intuitions

Entrée : graphe G et un ensemble de sacs Π
Sortie : une décomposition utilisant uniquement des sacs de Π

- C'est déjà l'idée de [Arnborg, Corneil, Proskuroski 1987]
- Conduirait à un algorithme en $\mathcal{O}^*(2^n)$ en prenant tous les sacs possibles . . . si on s'y prend pas trop mal

Pour commencer : intuitions

Entrée : graphe G et un ensemble de sacs Π
Sortie : une décomposition utilisant uniquement des sacs de Π

- C'est déjà l'idée de [Arnborg, Corneil, Proskuroski 1987]
- Conduirait à un algorithme en $\mathcal{O}^*(2^n)$ en prenant tous les sacs possibles . . . si on s'y prend pas trop mal
- Nous verrons plus tard comment se restreindre à des sacs « utiles »

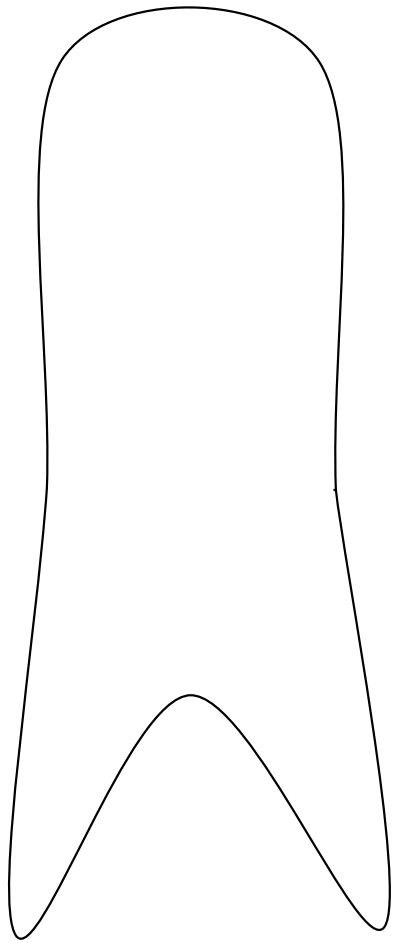
Pour commencer : intuitions

Entrée : graphe G et un ensemble de sacs Π
Sortie : une décomposition utilisant uniquement des sacs de Π

- C'est déjà l'idée de [Arnborg, Corneil, Proskuroski 1987]
- Conduirait à un algorithme en $\mathcal{O}^*(2^n)$ en prenant tous les sacs possibles . . . si on s'y prend pas trop mal
- Nous verrons plus tard comment se restreindre à des sacs « utiles »

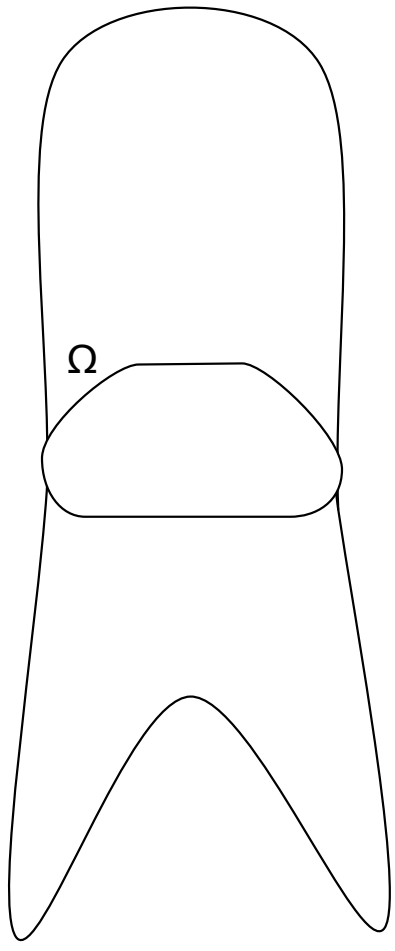
« Lego® » avec les sacs

Lego sur le graphe



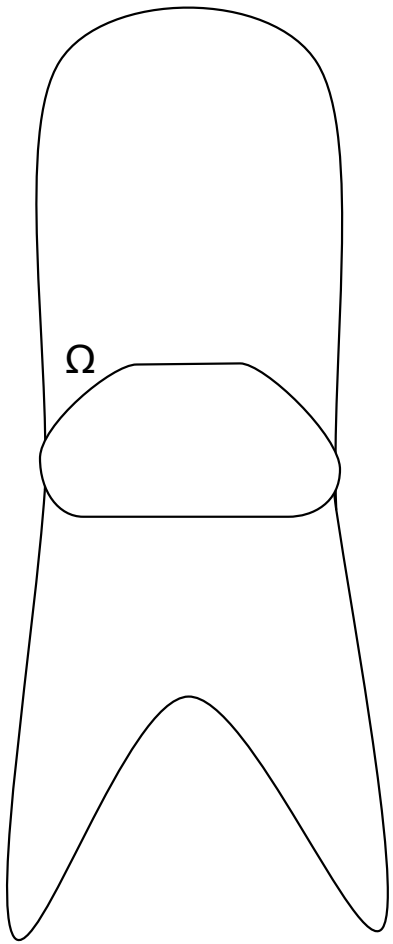
Lego sur le graphe

Pour utiliser un sac Ω, il faut pouvoir « paver » chaque composante connexe C de $G - \Omega$;



Lego sur le graphe

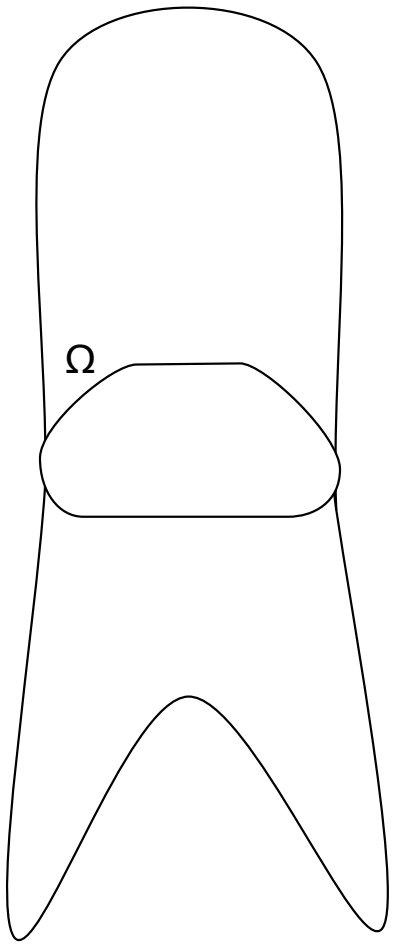
Pour utiliser un sac Ω, il faut pouvoir « paver » chaque composante connexe C de $G - \Omega$; c'est aussi une condition suffisante



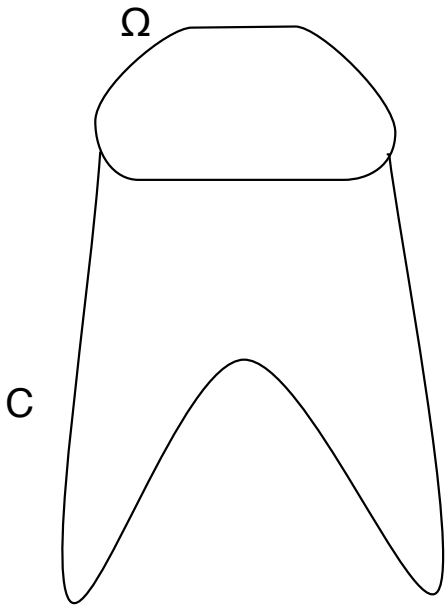
Lego sur le graphe

Pour utiliser un sac Ω, il faut pouvoir « paver » chaque composante connexe C de $G - \Omega$; c'est aussi une condition suffisante

Nous allons considérer des paires (Ω, C), appelées blocs

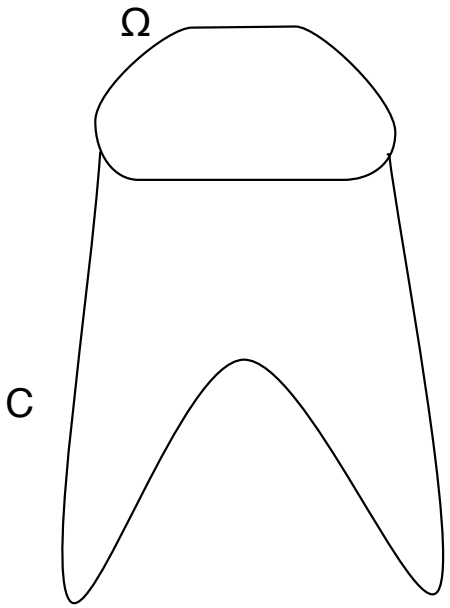


Lego sur les blocs



$\mathrm{tw}(\Omega, C) \leq k$?

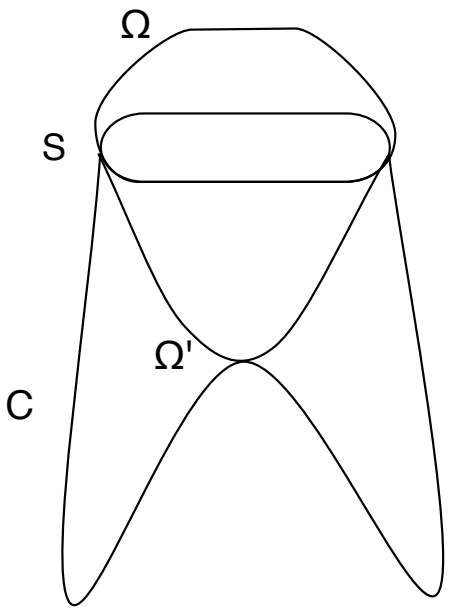
Lego sur les blocs



$\mathrm{tw}(\Omega, C) \leq k$?

Il nous faut un sac Ω' t.q.

Lego sur les blocs

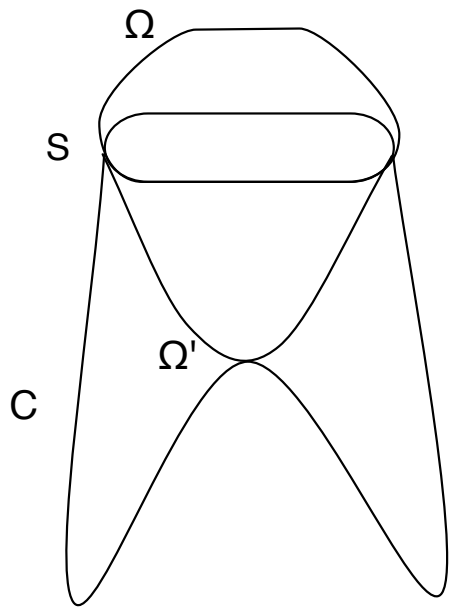


$\mathrm{tw}(\Omega, C) \leq k$?

Il nous faut un sac Ω' t.q.

- $\Omega' \subseteq \Omega \cup C$

Lego sur les blocs

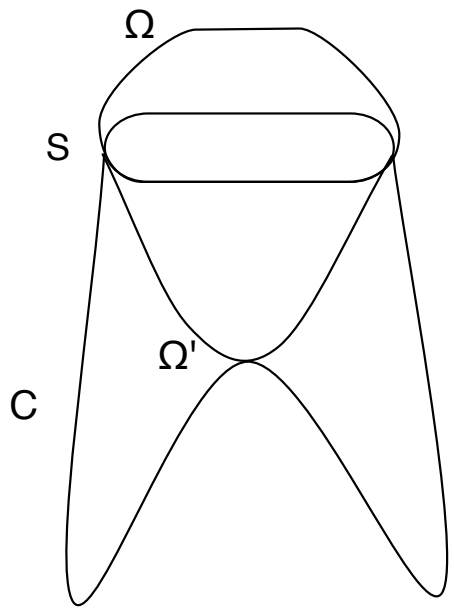


$\mathrm{tw}(\Omega, C) \leq k$?

Il nous faut un sac Ω' t.q.

- $\Omega' \subseteq \Omega \cup C$
- Ω' contienne $S = N(C)$

Lego sur les blocs



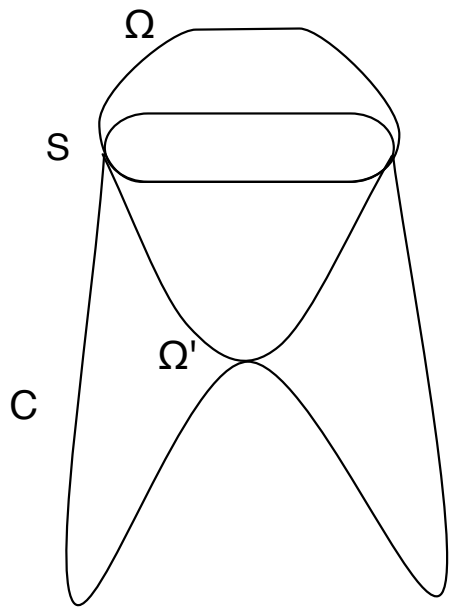
$\mathrm{tw}(\Omega, C) \leq k$?

Il nous faut un sac Ω' t.q.

- $\Omega' \subseteq \Omega \cup C$
- Ω' contienne $S = N(C)$

Ω' découpera le bloc (Ω, C) en blocs plus petits !

Lego sur les blocs



$\mathrm{tw}(\Omega, C) \leq k$?

Il nous faut un sac Ω' t.q.

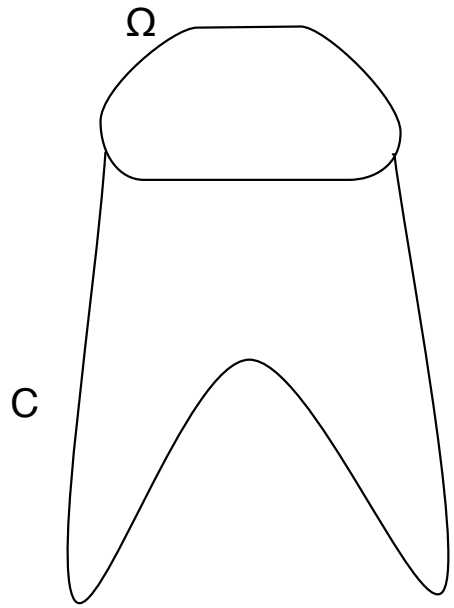
- $\Omega' \subseteq \Omega \cup C$
- Ω' contienne $S = N(C)$

Ω' découpera le bloc (Ω, C) en blocs plus petits !

Programmation dynamique

$\mathrm{tw}(\Omega, C)$?

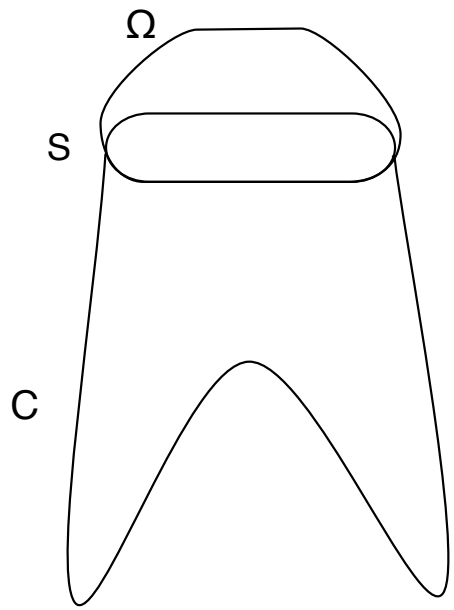
- $S = N(C)$
- $\mathrm{tw}(\Omega, C) = \max(|\Omega| - 1, \mathrm{tw}(S, C))$



Programmation dynamique

$\mathrm{tw}(\Omega, C)$?

- $S = N(C)$
- $\mathrm{tw}(\Omega, C) = \max(|\Omega| - 1, \mathrm{tw}(S, C))$

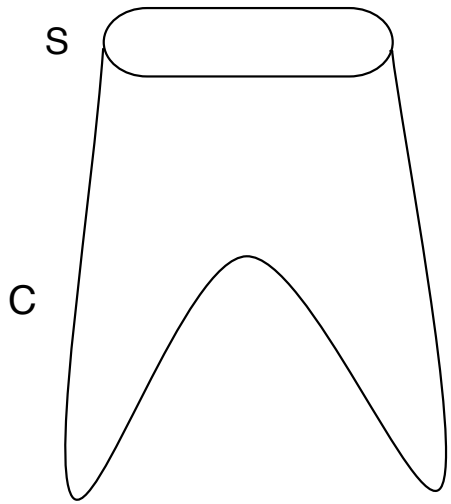


Programmation dynamique

$\mathrm{tw}(S, C)$?

$$\mathrm{tw}(S, C) = \min_{\Omega} \left(\max(|\Omega| - 1, \mathrm{tw}(S_i, C_i)\right)$$

sur tous les sacs Ω t.q. $S \subseteq \Omega \subseteq S \cup C$

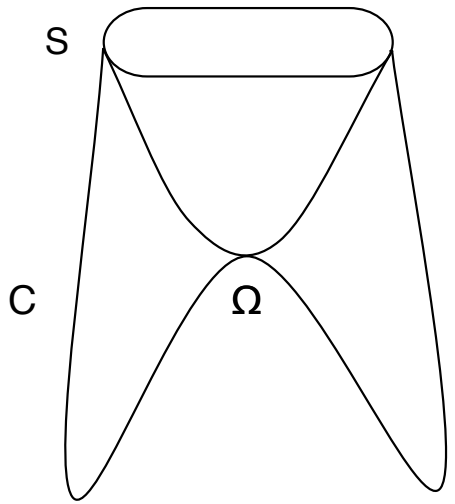


Programmation dynamique

$\mathrm{tw}(S, C)$?

$$\mathrm{tw}(S, C) = \min_{\Omega} \left(\max(|\Omega| - 1, \mathrm{tw}(S_i, C_i)\right)$$

sur tous les sacs Ω t.q. $S \subseteq \Omega \subseteq S \cup C$

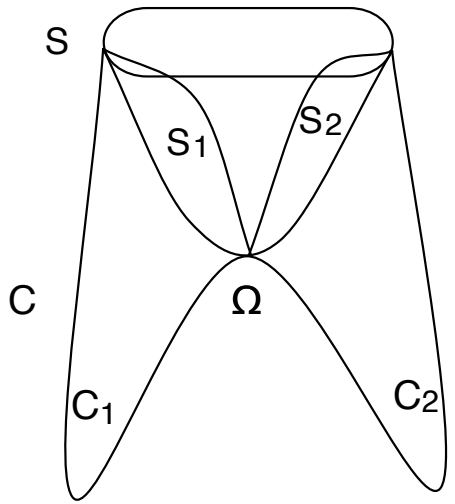


Programmation dynamique

$\mathrm{tw}(S, C)$?

$$\mathrm{tw}(S, C) = \min_{\Omega} \left(\max(|\Omega| - 1, \mathrm{tw}(S_i, C_i)\right)$$

sur tous les sacs Ω t.q. $S \subseteq \Omega \subseteq S \cup C$

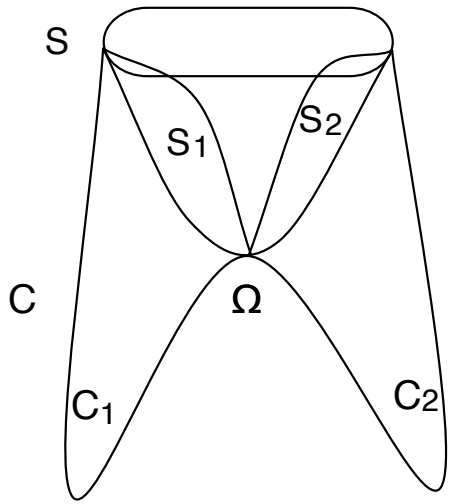


Programmation dynamique

$\mathrm{tw}(S, C)$?

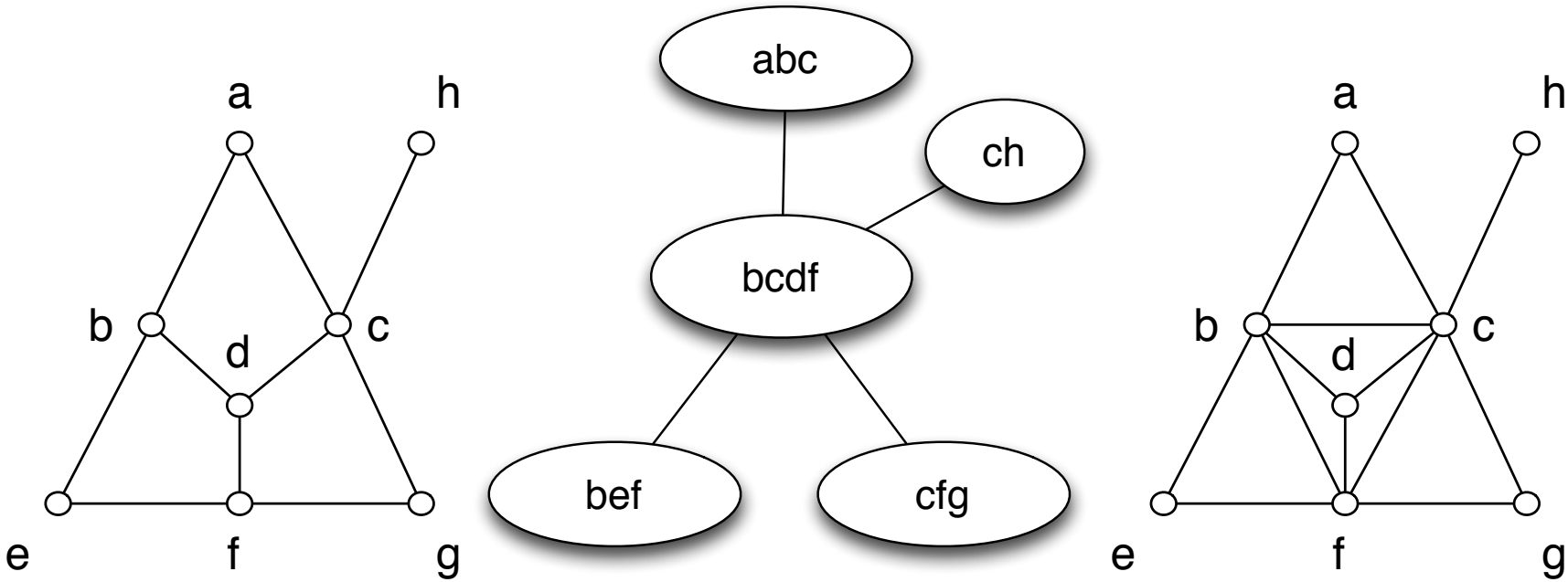
$$\mathrm{tw}(S, C) = \min_{\Omega} \left(\max(|\Omega| - 1, \mathrm{tw}(S_i, C_i)\right)$$

sur tous les sacs Ω t.q. $S \subseteq \Omega \subseteq S \cup C$



Calculer $\mathrm{tw}(S, C)$ par taille croissante de $S \cup C$

Pour finir : bien choisir ses sacs

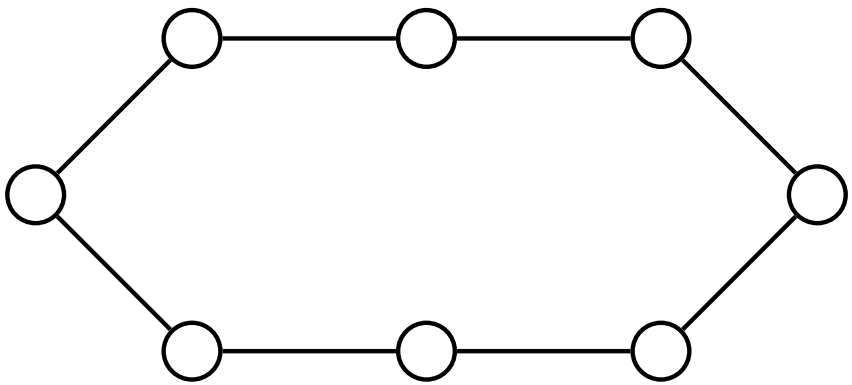


- décomposition arborescente $\Leftrightarrow$ triangulation
- sacs $\Leftrightarrow$ cliques maximales
- triangulation minimale

Cliques maximales potentielles

Definition ([Bouchitté, T. 2000])

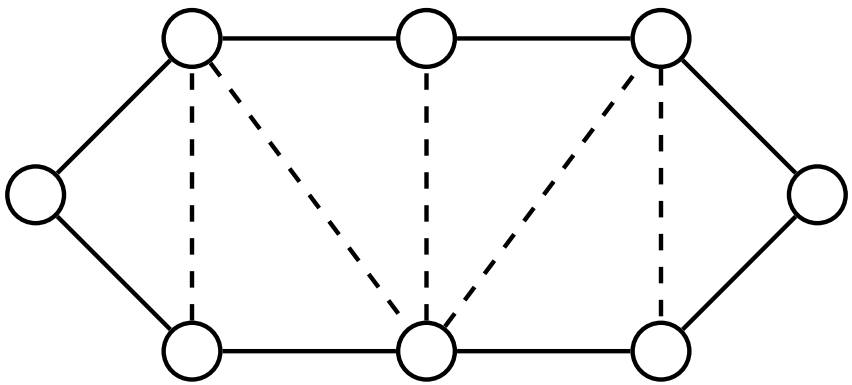
Un ensemble de sommets Ω est une clique maximal potentielle de G s'il existe une décomposition arborescente minimale TG de G telle que Ω soit un sac TG.



Cliques maximales potentielles

Definition ([Bouchitté, T. 2000])

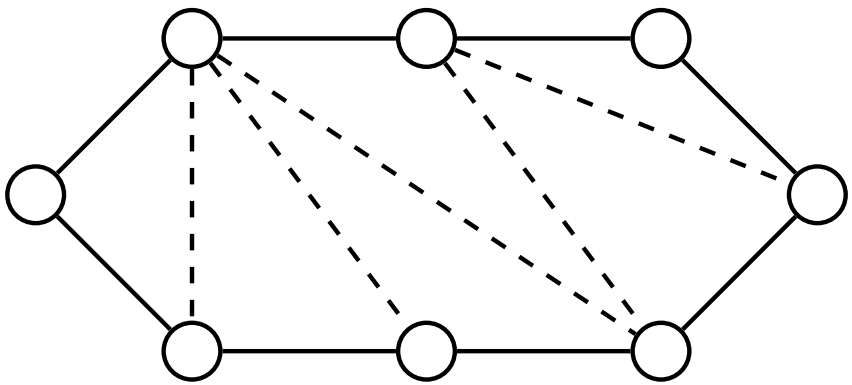
Un ensemble de sommets Ω est une clique maximale potentielle de G s'il existe une triangulation minimale TG de G tel que Ω soit une clique maximale de TG.



Cliques maximales potentielles

Definition ([Bouchitté, T. 2000])

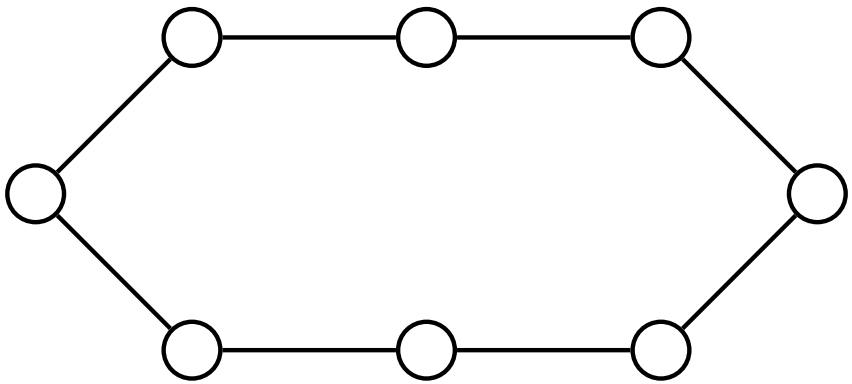
Un ensemble de sommets Ω est une clique maximale potentielle de G s'il existe une triangulation minimale TG de G tel que Ω soit une clique maximale de TG.



Cliques maximales potentielles

Definition ([Bouchitté, T. 2000])

Un ensemble de sommets Ω est une clique maximale potentielle de G s'il existe une triangulation minimale TG de G tel que Ω soit une clique maximale de TG.



Proposition ([Fomin, T, Villanger. 2015])

Le nombre de cliques maximales potentielles est $\mathcal{O}(1.7347^n)$.

Calcul de la largeur arborescente

Largeur Arborescente(G)

1. Calculer les cliques maximales potentielles ; $\mathcal{O}(1.7347^n)$
2. Utiliser la programmation dynamique sur leur blocs ; $\mathcal{O}^*(\#\text{c.m.p})$

Calcul de la largeur arborescente

Largeur Arborescente(G)

1. Calculer les cliques maximales potentielles ; $\mathcal{O}(1.7347^n)$
2. Utiliser la programmation dynamique sur leur blocs ; $\mathcal{O}^*(\#\text{c.m.p})$

Énumération des cliques maximales potentielles : caractérisation combinatoire de ces objets

Calcul de la largeur arborescente

Largeur Arborescente(G)

1. Calculer les cliques maximales potentielles ; $\mathcal{O}(1.7347^n)$
2. Utiliser la programmation dynamique sur leur blocs ; $\mathcal{O}^*(\#\text{c.m.p})$

Énumération des cliques maximales potentielles : caractérisation combinatoire de ces objets

Pour (presque) chaque c.m.p. Ω il existe un ensemble de sommets X et tel que $\Omega = N(X)$, $|X| \leq \frac{2}{3}(n - |\Omega|)$ et X est formé de deux parties connexes

Prog. dynamique sur les cliques maximales potentielles

Algorithme 6: Largeur Arborescente

Entrées : $G = (V, E)$ et ses cliques maximales potentielles
Sorties : $\mathrm{tw}(G)$

Calculer tous les blocs pleins (S, C) et les trier par taille croissante
pour chaque *bloc* (S, C) **par taille croissante faire**
    $\mathrm{tw}(S, C) \leftarrow \infty$
    pour chaque *c.m.p.* Ω *t.q.* $S \subset \Omega \subseteq S \cup C$ **faire**
        $x \leftarrow |\Omega| - 1$
        pour chaque *composante* C_i *de* $G[C \setminus \Omega]$ **faire**
            $S_i \leftarrow N(C_i)$
            $x \leftarrow \max(x, \mathrm{tw}(S_i, C_i))$
        $\mathrm{tw}(S, C) \leftarrow \min(x, \mathrm{tw}(S, C))$
retourner $\min_{S} \max_{\text{comp. } C \text{ de } G - S} \mathrm{tw}(S, C)$

Généralisation

Sur le même schéma de programmation dynamique :

- Plus long chemin induit
- Plus grande forêt induite (équivalent à Coupe-cycle minimum)
- Plus grand sous-graphe induit sans cycle de longueur $\geq l$
- Plus grand sous-graphe induit qui exclut un mineur planaire...

Généralisation

Sur le même schéma de programmation dynamique :

- Plus long chemin induit
- Plus grande forêt induite (équivalent à Coupe-cycle minimum)
- Plus grand sous-graphe induit sans cycle de longueur $\geq l$
- Plus grand sous-graphe induit qui exclut un mineur planaire...

[Fomin, T.,Villanger 2015] : plus grand sous-graphe induit de largeur arborescente $\leq cst$, satisfaisant une propriété exprimable en logique monadique du second ordre (CMSO)
Même complexité $\mathcal{O}(1.7347^n)$; méta-théorème algorithmique

Discussion : programmation dynamique

Alliance de force brute et de subtilité

- Algorithmes polynomiaux,

Discussion : programmation dynamique

Alliance de force brute et de subtilité

- Algorithmes polynomiaux, exacts modérément exponentiels,

Discussion : programmation dynamique

Alliance de force brute et de subtilité

- Algorithmes polynomiaux, exacts modérément exponentiels, paramétrés (FPT ; complexité $f(k) \cdot n^{\mathcal{O}(1)}$)

Discussion : programmation dynamique

Alliance de force brute et de subtilité

- Algorithmes polynomiaux, exacts modérément exponentiels, paramétrés (FPT ; complexité $f(k) \cdot n^{\mathcal{O}(1)}$)
- Complexité en espace : comparable à celle en temps

Discussion : programmation dynamique

Alliance de force brute et de subtilité

- Algorithmes polynomiaux, exacts modérément exponentiels, paramétrés (FPT ; complexité $f(k) \cdot n^{\mathcal{O}(1)}$)
- Complexité en espace : comparable à celle en temps
- Espace polynomial : remplacer par du « diviser pour régner » — mais c'est plus lent

Discussion : les problèmes

Voyageur de commerce

- Grand classique
- Approche de Held et Karp largement réutilisée pour d'autres problèmes

Largeur arborescente et décompositions arborescentes

- Notion « compliquée » mais cruciale combinatoire et algorithmique des graphes
- Théorie des mineurs de graphes [Robertson, Seymour, Graph Minors I – XXIII, 1983-2010]
- Algorithmique paramétrée, méta-théorèmes [Courcelle 1990]

Discussion : les problèmes

Voyageur de commerce

- Grand classique
- Approche de Held et Karp largement réutilisée pour d'autres problèmes

Largeur arborescente et décompositions arborescentes

- Notion « compliquée » mais cruciale combinatoire et algorithmique des graphes
- Théorie des mineurs de graphes [Robertson, Seymour, Graph Minors I – XXIII, 1983-2010]
- Algorithmique paramétrée, méta-théorèmes [Courcelle 1990]
- Cliques maximales potentielles : algorithmes exacts pour une large classe de problèmes

Algorithmes exacts exponentiels
– pour conclure –

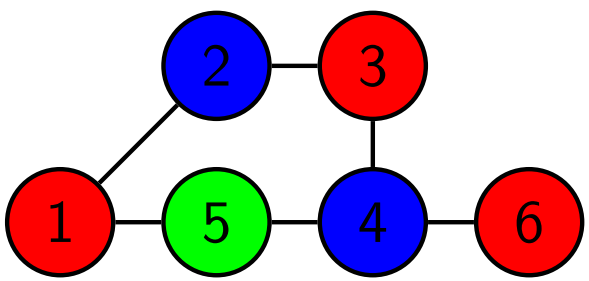
Techniques algorithmiques

- Algorithmes de branchement, « Mesurer pour conquérir »
- Trier et chercher
- Programmation dynamique

Algorithmes exacts exponentiels
– pour conclure –

Techniques algorithmiques

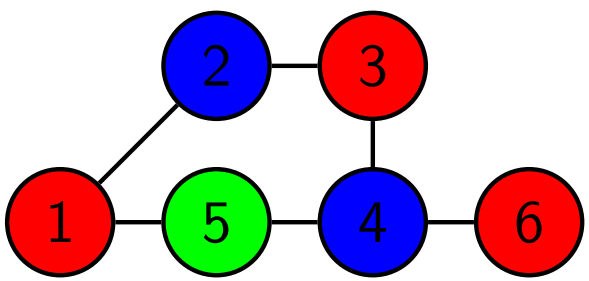
- Algorithmes de branchement, ≪ Mesurer pour conquérir ≫
- Trier et chercher
- Programmation dynamique
- Inclusion-exclusion (COLORATION en $\mathcal{O}^*(2^n)$)



Algorithmes exacts exponentiels
– pour conclure –

Techniques algorithmiques

- Algorithmes de branchement, « Mesurer pour conquérir »
- Trier et chercher
- Programmation dynamique
- Inclusion-exclusion (COLORATION en $\mathcal{O}^*(2^n)$)

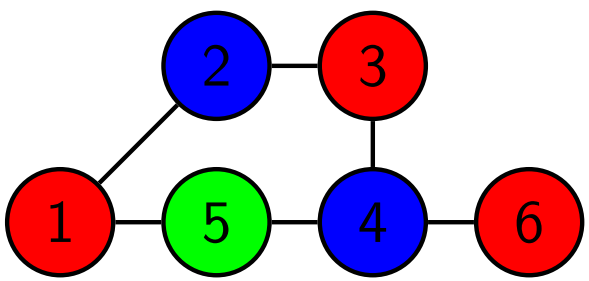


- Convolution

Algorithmes exacts exponentiels
– pour conclure –

Techniques algorithmiques

- Algorithmes de branchement, « Mesurer pour conquérir »
- Trier et chercher
- Programmation dynamique
- Inclusion-exclusion (COLORATION en $\mathcal{O}^*(2^n)$)



- Convolution
- ...

Algorithmes exacts exponentiels

. . . et complexité

- Hypothèse ETH : pas d'algorithme pour 3-SAT en $2^{o(n)}$
- SETH : pas d'algo pour SAT en $\mathcal{O}(c^n)$ avec $c < 2$
- Question de Gödel ?

. . . et d'autres domaines algorithmiques

- Approximation (question gag : 2-approximation pour $\textsc{Stable Maximum}$ en $\mathcal{O}^*(1.1^n)$?)
- Algorithmique à paramètre fixe (FPT) : paramètre k, algorithmes en $f(k) \cdot n^{\mathcal{O}(1)}$
- Heuristiques

. . . et combinatoire

- Bornes sur le nombre d'objets via algos d'énumération

Pour aller plus loin

Questions

- Coloration en $\mathcal{O}(1.99\ldots^n)$?
- Voyageur de commerce *déterministe* en $\mathcal{O}((2-\epsilon)^n)$?
- Meilleure borne sur le nombre de cliques maximales potentielles ?
- Méthodes d'analyse plus précises pour le branchement ?

Pour aller plus loin

Questions

- Coloration en $\mathcal{O}(1.99\ldots^n)$?
- Voyageur de commerce *déterministe* en $\mathcal{O}((2-\epsilon)^n)$?
- Meilleure borne sur le nombre de cliques maximales potentielles ?
- Méthodes d'analyse plus précises pour le branchement ?

Pointeurs

- Ecole AGAPE (algos de graphes parametrés et exacts), 2009 : http ://www-sop.inria.fr/mascotte/seminaires/AGAPE/notes
- [Exact Algorithms, D. Kratsch et F. V. Fomin, Springer 2010]

Pour aller plus loin

Questions

- Coloration en $\mathcal{O}(1.99\ldots^n)$?
- Voyageur de commerce *déterministe* en $\mathcal{O}((2-\epsilon)^n)$?
- Meilleure borne sur le nombre de cliques maximales potentielles ?
- Méthodes d'analyse plus précises pour le branchement ?

Pointeurs

- Ecole AGAPE (algos de graphes parametrés et exacts), 2009 : http ://www-sop.inria.fr/mascotte/seminaires/AGAPE/notes
- [Exact Algorithms, D. Kratsch et F. V. Fomin, Springer 2010]

Merci ! Vos questions ?