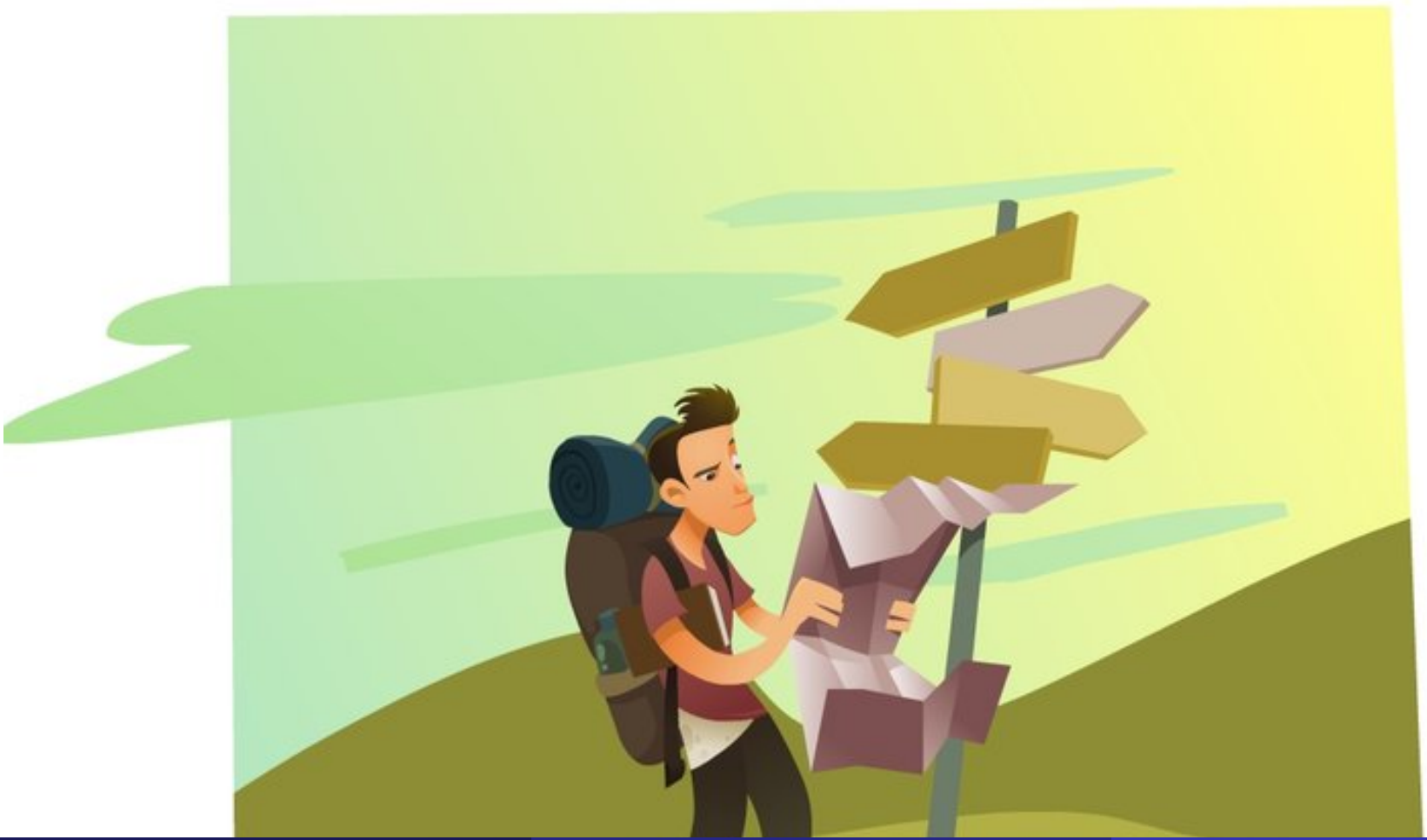
Complexité paramétrée

Florent Capelli

Journées Algorithmes et Programmation,
05 Mai 2016.

Un problème pour les vacances

- Un voyageur veut visiter toutes les villes d'un pays
- Il a 15 jours de vacances et une carte
- A-t-il le temps ?

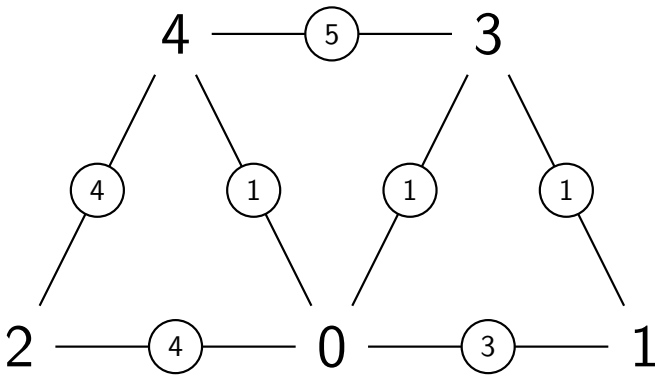


Une formalisation du problème

Étant donnés :

- un graphe $G = (V, E)$ et un sommet v_0
- un poids $w : E \to \mathbb{Q}_+$ sur chaque arête
- un objectif $k \in \mathbb{Q}_+$

Existe-t-il un chemin dans G de poids inférieur à k commençant en v_0 et passant par tous les sommets ?

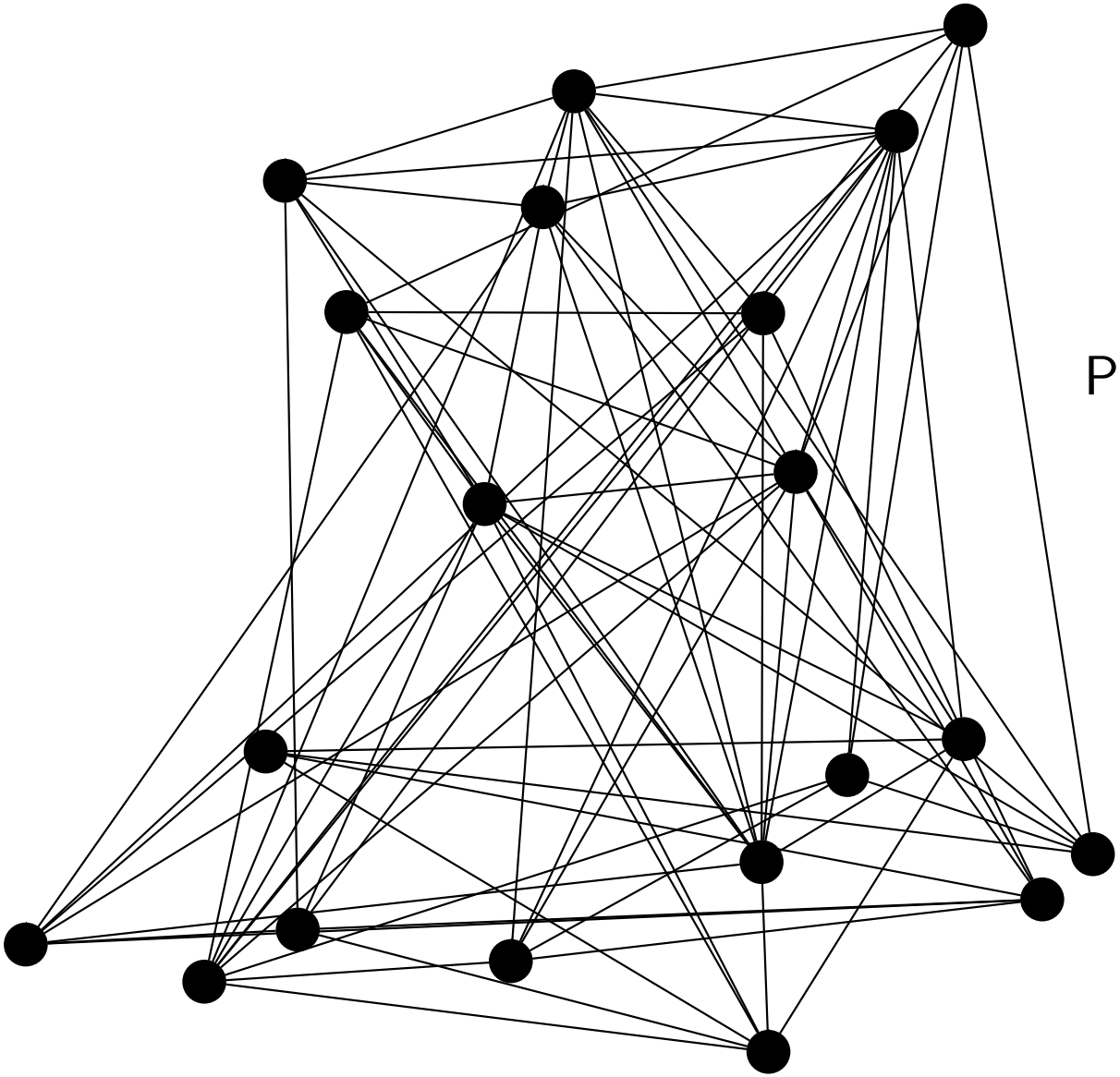


- $k < 15$? Oui : 0,1,3,4,2.
- $k < 10$? Oui : 0,3,1,3,0,4,2.
- $k < 9$? Non...

Ce problème est NP-**difficile.**

Une bonne modélisation du problème ?

Avez-vous déjà vu des routes construites ainsi ?



Problèmes :

- pas planaire
- distance euclidienne ?
- Structure pas vraiment réaliste

Voyageur de commerce euclidien

Même problème sauf que :

- le graphe est donné comme une liste de points dans le plan et d'arêtes
- le poids d'une arête est sa norme euclidienne

Voyageur de commerce euclidien

Même problème sauf que :

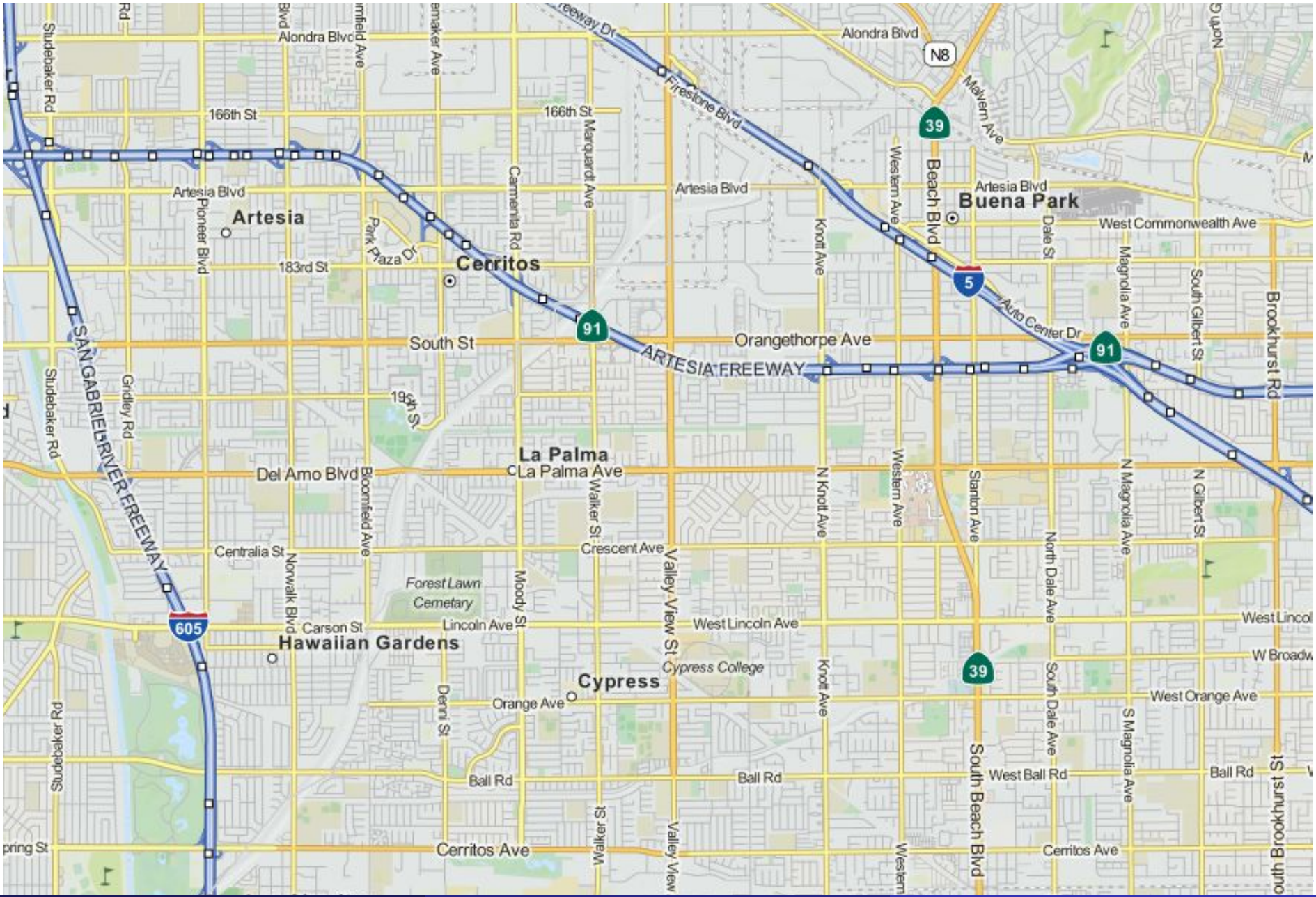
- le graphe est donné comme une liste de points dans le plan et d'arêtes
- le poids d'une arête est sa norme euclidienne

Ce nouveau problème est :

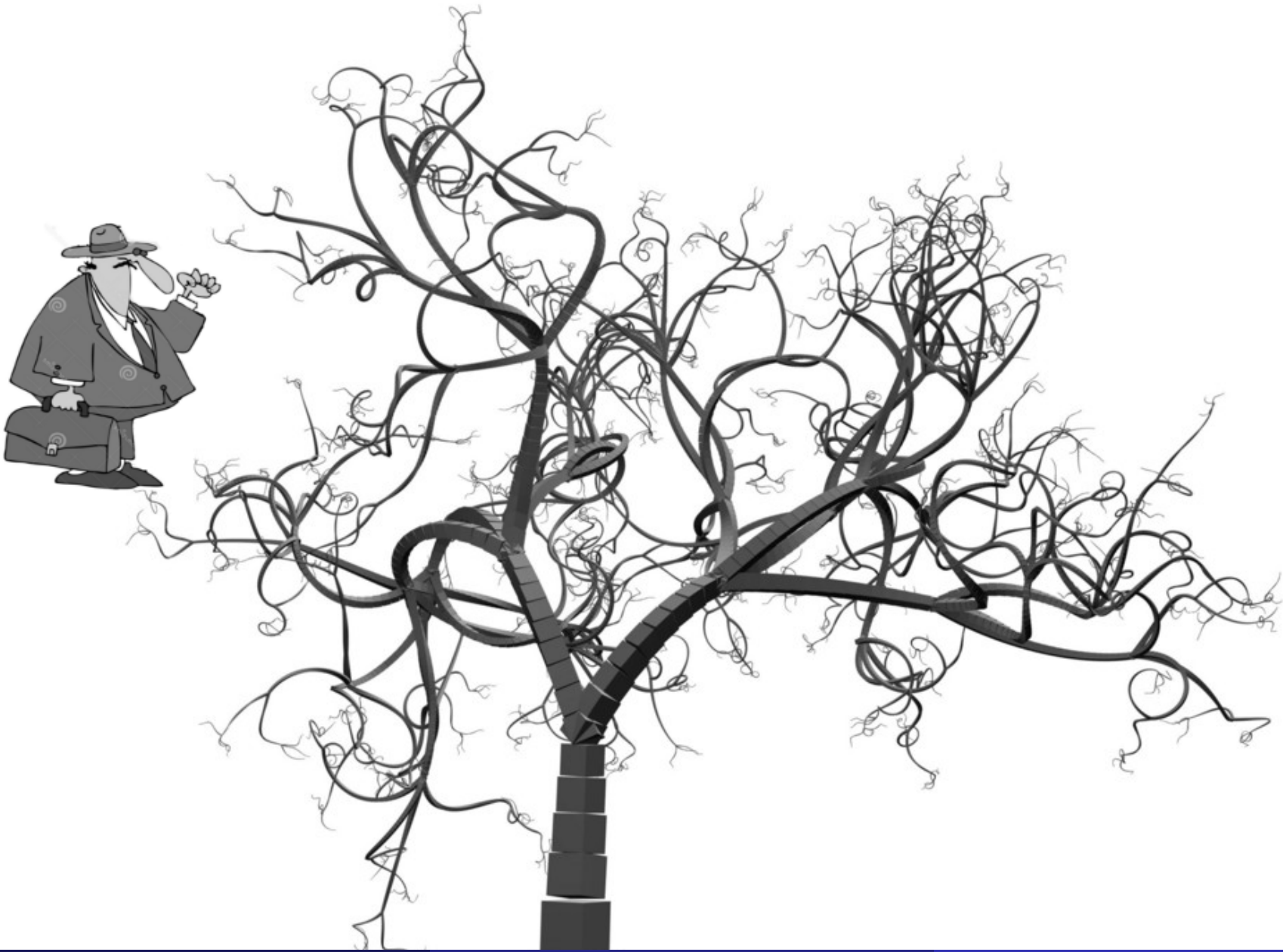
- Toujours NP-complet ...
- ... mais on sait approximer.

Peut-on exploiter la structure du graphe sous-jacent pour trouver l'optimum plus rapidement ?

Los Angeles

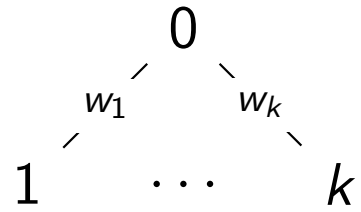

Alondra Blvd
N8
Firestone Blvd
166th St
Artesia Blvd
Artesia
Pioneer Blvd
183rd St
Park Plaza Dr
Cerritos
Carmenita Rd
Marquardt Ave
Studebaker Rd
SAN GABRIEL RIVER FREEWAY
605
Gridley Rd
South St
91
ARTESIA FREEWAY
Orangethorpe Ave
Beach Blvd
39
Malvern Ave
Western Ave
Knott Ave
Buena Park
Dale St
West Commonwealth Ave
Magnolia Ave
South Gilbert St
Brookhurst Rd
Auto Center Dr
5
19th St
La Palma
La Palma Ave
Del Amo Blvd
Bloomfield Ave
Walker St
Centralia St
Norwalk Blvd
Crescent Ave
Valley View St
Stanton Ave
North Dale Ave
Forest Lawn Cemetary
Moody St
Carson St
Lincoln Ave
West Lincoln Ave
Hawaiian Gardens
Cypress College
Cypress
Orange Ave
Denni St
West Orange Ave
South Dale Ave
South Beach Blvd
West Ball Rd
Ball Rd
Cerritos Ave
Valley View

Voyageur de commerce sur un arbre



Voyageur de commerce sur un arbre

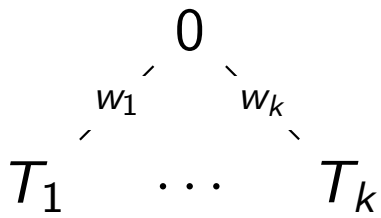
Supposons que le graphe sous-jacent soit un arbre. Comment trouver l'optimum ?



- Un chemin est forcément de la forme : $0, i_1, 0, i_2, 0, \ldots, i_m$
- L'optimal visite exactement une fois chaque sommet $i > 0$
- Le poids de l'optimal est $2 \cdot (\sum_{i=1}^{k} w_i) - w$ où $w = \max w_i$

Voyageur de commerce sur un arbre

Supposons que le graphe sous-jacent soit un arbre. Comment trouver l'optimum ?



- Un chemin est forcément de la forme : $0, C_1, 0, C_2, 0, \ldots, C_m$ où C_i est un chemin dans T_i
- L'optimal visite exactement une fois chaque sous-arbre T_i
- Le poids de l'optimal est $2 \cdot (\sum_{i=1}^{k} w_i + w(T_i)) - w$ où $w = \max w_i + w(T_i)$

Pas vraiment réaliste

- Peu d'endroits où les routes forment un arbre
- On préférerait un algorithme qui marche *tout le temps* mais plus ou moins bien selon la "complexité" du graphe
- On verra plus tard comment "mesurer" la distance d'un graphe à un arbre et comment l'utiliser pour ce problème

Complexité paramétrée

- Notion de **paramètre** pour mesurer la structure de l'entrée:

$$\kappa : \text{entrees} \to \mathbb{N}$$

- Problème paramétré : un problème + un paramètre κ calculable en temps polynomial
- **But :** trouver un algo polynomial lorsque κ est borné

Exemples

$G = (V, E)$ un graphe.
Clique $= W \subseteq V$ tq $\forall i, j \in W, (i, j) \in E$

Exemples

$G = (V, E)$ un graphe.
Clique $= W \subseteq V$ tq $\forall i, j \in W, (i, j) \in E$

Problème CLIQUE : étant donnés G et k, trouver une clique de taille k dans G

Problème p-CLIQUE : étant donnés G et k, trouver une clique de taille k dans G
Paramètre : k
$\rightarrow$ algorithme en temps $O(\binom{n}{k} k^2)$ en bruteforçant les sous-ensembles de taille k

Exemples

Vertex cover $= W \subseteq V$ tq $\forall e \in E, e \cap W \neq \emptyset$

Exemples

Vertex cover $= W \subseteq V$ tq $\forall e \in E, e \cap W \neq \emptyset$

Problème VERTEX-COVER : étant donnés G et k, trouver un vertex cover de taille k dans G

Problème p-VERTEX-COVER : étant donnés G et k, trouver un vertex cover de taille k dans G
Paramètre : k
$\rightarrow$ algorithme en temps $O(2^k kn)$ (on y reviendra).

Différentes qualités

- Vertex cover : algorithme en temps $f(k) \cdot n$ pour $f(k) = k2^k$
- Clique : algorithme en temps n^k
- dépendance en k fondamentalement différente

Définition

Un problème paramétrée par κ est FPT (fixed-parameter tractable) s'il peut être résolu en temps $f(\kappa(n)) \cdot \text{poly}(n)$ pour f calculable.

$$\text{FPT} = \text{“facile”}$$

- Vertex cover : FPT, encore faisable pour $k = 10$, $n = 1000$.
- Clique : bruteforce pas FPT, inutile pour $k = 10, n = 20$.

Algorithme FPT pour Vertex Cover

Observation

Si $G = (V, E)$ a un vertex cover de taille k, alors il a au plus $k \cdot (|V| - 1)$ arêtes.

En effet, soit S un vertex cover de taille k:

- par définition, chaque arête $e \in E$ est incidente à S
- donc e de la forme (x, s) avec $s \in S$
- pour chaque $s \in S$, au plus $|V| - 1$ choix pour x

Algorithme FPT pour Vertex Cover

Cela suggère l'algorithme VC(G, k):

- Si G a plus de $k(|V| - 1)$ arêtes, renvoyer FAUX.
- Si G n'a pas d'arêtes, renvoyer VRAI.
- Sinon, choisir $e = (u, v) \in E$.
- Renvoyer VC($G - u, k - 1$) $\vee$ VC($G - v, k - 1$).

Algorithme FPT pour Vertex Cover

Cela suggère l'algorithme VC(G, k):

- Si G a plus de $k(|V| - 1)$ arêtes, renvoyer FAUX.
- Si G n'a pas d'arêtes, renvoyer VRAI.
- Sinon, choisir $e = (u, v) \in E$.
- Renvoyer VC($G - u, k - 1$) $\vee$ VC($G - v, k - 1$).

On a une complexité de $O(2^k kn)$:

- Profondeur des appels récursifs: au plus k
- Donc la fonction est appelée au plus 2^k fois
- Chaque appel $\simeq$ O(nombre d'arêtes), ie au plus kn.

But de la complexité paramétrée

- Algorithmes paramétrés : notion naturelle
- Besoin d'un cadre théorique ?

But de la complexité paramétrée

- Algorithmes paramétrés : notion naturelle
- Besoin d'un cadre théorique ?
- Analogie avec la théorie de la complexité "classique" :
  - étudier la complexité d'un algorithme : théorie de la complexité inutile
  - mais aide à savoir à quel point un problème est difficile
- Pareil ici : outils pour nous dire si un paramètre est pertinent ou non

Notion de réduction

- En complexité classique, notion de réduction très importante
- Permet de comparer la difficulté des problèmes

Definition

Un problème P_1 se réduit à un problème P_2, noté $P_1 \leq P_2$, s'il existe f calculable en temps polynomial telle que $x \in P_1$ ssi $f(x) \in P_2$.

- Intuitivement : P_1 est plus facile que P_2

Exemples de réduction

Ensemble indépendant $=$ ensemble de sommets W tq $\forall i, j \in W$, $(i, j) \notin E$

Problème ENSEMBLE-INDEPENDANT : étant donné $k \in \mathbb{N}$ et G, trouver un ensemble indépendant de taille k dans G.

On a:

- ENSEMBLE-INDEPENDANT $\leq$ CLIQUE par $(G, k) \mapsto (\bar{G}, k)$

Exemples de réduction

Ensemble indépendant = ensemble de sommets W tq $\forall i, j \in W$, $(i, j) \notin E$

Problème ENSEMBLE-INDEPENDANT : étant donné $k \in \mathbb{N}$ et G, trouver un ensemble indépendant de taille k dans G.

On a:

- ENSEMBLE-INDEPENDANT $\leq$ CLIQUE par $(G, k) \mapsto (\bar{G}, k)$
- ENSEMBLE-INDEPENDANT $\leq$ VERTEX-COVER par $(G, k) \mapsto (G, |V| - k)$ car le complémentaire d'un VC est un EI

Utilisation pour la complexité paramétrée

k-**Coloriage** d'un graphe $G = (V, E)$: fonction $w : V \to [k]$ tq $\forall (u, v) \in E,\ w(u) \neq w(v)$

Problème COLORIAGE: étant donné G et $k \in \mathbb{N}$, trouver un k-coloriage de G.
Paramètre : k.

Utilisation pour la complexité paramétrée

k-**Coloriage** d'un graphe $G = (V, E)$: fonction $w : V \to [k]$ tq $\forall (u, v) \in E,\ w(u) \neq w(v)$

Problème COLORIAGE: étant donné G et $k \in \mathbb{N}$, trouver un k-coloriage de G.
Paramètre : k.

- N'admet probablement pas d'algo FPT car NP-complet pour $k = 3$
- Très peu probable qu'il admette un algo en $n^{f(k)}$
- Peut-on trouver un outil plus fin pour différencier les problèmes admettant un algorithme en $n^{f(k)}$ mais pas FPT ?

Réduction FPT

- Généraliser cette notion à la complexité paramétrée
- Réduire à la fois le problème de décision ET le paramètre

Definition

Un problème paramétré (P_1, κ_1) se réduit à un problème (P_2, κ_2), noté $(P_1, \kappa_1) \leq (P_2, \kappa_2)$, s'il existe f telle que:

- f est calculable en temps $g(\kappa_1(n)) \cdot \text{poly}(n)$ pour une fonction g calculable
- $x \in P_1$ ssi $f(x) \in P_2$
- $\kappa_2(f(x)) \leq g(\kappa_1(x))$ pour une fonction g calculable

- Si (P_2, κ_2) est FPT et $(P_1, \kappa_1) \leq (P_2, \kappa_2)$, alors (P_1, κ_1) est FPT.

Exemples

Problème p-ENSEMBLE-INDÉPENDANT : trouver une clique de taille k dans un graphe
Paramètre : k

p-CLIQUE $\leq$ p-ENSEMBLE-INDÉPENDANT paramétré par k par la réduction FPT:

$$f : (k, G) \mapsto (k, \overline{G})$$

Exemples

Problème p-ENSEMBLE-INDÉPENDANT : trouver une clique de taille k dans un graphe
Paramètre : k

p-CLIQUE $\leq$ p-ENSEMBLE-INDÉPENDANT paramétré par k par la réduction FPT:

$$f : (k, G) \mapsto (k, \overline{G})$$

En effet:

- f est calculable en temps polynomial
- G admet une clique de taille k ssi $\bar{G}$ admet un IS de taille k.
- $k \leq k$

Co-exemple

p-ENSEMBLE-INDÉPENDANT à p-VERTEX-COVER par

$$(G, k) \mapsto (G, |V| - k)$$

- Pas FPT!
- $|V| - k$ pas borné par une fonction en k
- tant mieux! Réduction + algo FPT pour VC
  $\rightarrow O(2^{|V|-k}(|V| - k)|V|)$, pas FPT !
- Mais ENSEMBLE-INDÉPENDANT paramétré par $|V| - k$ est FPT!

La classe W[1] et l'hypothèse de l'ingénieur

Definition

Un problème paramétré (P, κ) est dans la classe $W[1]$ ssi il existe une réduction de (P, κ) à p-CLIQUE.

- On a FPT $\subseteq W[1]$ (pas évident!)
- Hypothèse de l'ingénieur : FPT $\neq W[1]$
- Analogue de P $\neq$ NP
- Si p-CLIQUE $\leq (P, \kappa)$, alors P est W[1]-dur, probablement pas FPT!
- p-ENSEMBLE-INDEPENDANT est W[1]-dur!

Exemple : CLIQUE-MULTICOLORE

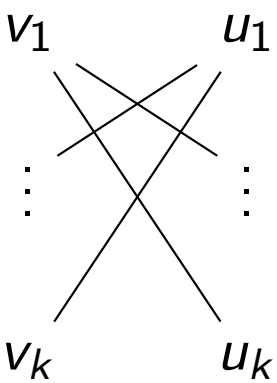
Problème p-CLIQUE-MULTICOLORE : étant donné un graphe colorié avec k couleurs, trouver une clique ayant un sommet de chaque couleur.
Paramètre : k

p-CLIQUE-MULTICOLORE est W[1]-dur.

Réduction à p-CLIQUE:

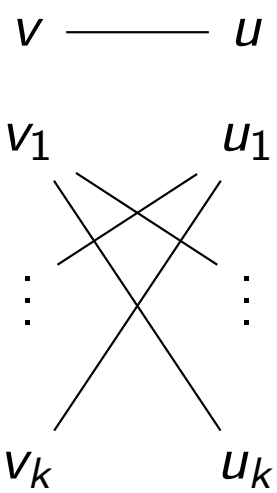
- Depuis G et k, construire $G' = (V', E')$ où

v —— u



- $V' = V \times [k]$ ie chaque sommet copié k fois
- Arête $(u, v) \in E$ transformé en arêtes (u_i, v_j) pour $i \neq j$
- u_i est colorié en i

Exemple : CLIQUE-MULTICOLORE



- Une k-clique $u^1, \ldots, u^k$ dans G donne la k-clique multicolore $u_1^1, \ldots, u_k^k$ dans G'
- Une k-clique multicolore dans G' est de la forme $u^1, \ldots, u^k$ avec:
  - u^i de couleur i
  - u^i, u^j sont des copies de deux sommets distincts de G
  - ces sommets forment un k-clique de G

Hypothèse de l'ingénieur

Question W[1] vs FPT?

- Si P = NP alors W[1] = FPT car p-CLIQUE a un algorithme polynomial
- Malheureusement rien de connu dans l'autre sens
- Conséquences de W[1] = FPT ?!

Le problème SAT

- On a une formule CNF F sur les variables X, ie de la forme

$$\bigwedge_i C_i$$

- Les C_i sont des *clauses*, de la forme

$$\bigvee \ell_{i,j}$$

- les $\ell_{i,j}$ sont des litéraux de la forme x ou $\neg x$ pour $x \in X$
- On cherche $\tau : X \to \{0,1\}$ qui rend la formule vrai
- Par exemple

$$F = (x \vee \neg y \vee t) \wedge (\neg x \vee z \vee w) \wedge (\neg t \vee \neg w)$$

est satisfaite pour $\{x \mapsto 0, y \mapsto 1, t \mapsto 0, z \mapsto 1\}$

Le problème SAT

- Problème SAT : décider si, étant donné F une CNF, F est satisfaisable
- Problème NP-complet, le premier naturel à avoir été découvert
- Meilleurs algorithmes exacts actuels : $O(2^n)$ avec n le nombre de variables...
- ... pas mieux que le bruteforce

Le problème k-SAT

- Algorithmes exacts actuels résolvent k-SAT
- k-SAT = SAT restreint au k-CNF
- où une k-CNF est une CNF avec des clauses ayant au plus k litéraux
- k-SAT est NP-complet pour $k \geq 3$
- Meilleurs algo pour k-SAT : complexité $O(2^{n-u_k/k})$ avec $\lim_{k\to\infty} u_k = 0$

SETH

- Même un algorithme en temps $O(2^{10^{-10}(10^{10}-1)n})$ serait une grande avancée
- $s_k = \inf\{s \mid$ il existe un algorithme en temps $O(2^{sn})$ pour k-SAT$\}$
- s_k croissante et borné (par 1), converge vers s_∞
- "Strong Exponential Time Hypothesis" (SETH) : $s_\infty = 1$
- En gros : bruteforce pour SAT est optimal à une constante près.

ETH

- SETH un peu trop forte
- Plus raisonnable “Exponential Time Hypothesis” : $s_\infty > 0$
- En gros : pas d'algorithme en temps $2^{o(n)}$ pour SAT
- Équivalent (pas évident) à $s_3 > 0$.

ETH et la complexité paramétrée

Si FPT = W[1] alors ETH est fausse. Quasiprouvons-le !

- Dans ETH, on peut interchanger variables et clauses (pas évident)
- k-CLIQUE en temps $f(k)n^c \rightarrow$ 3-SAT en temps $2^{o(m)}$ (m clauses)

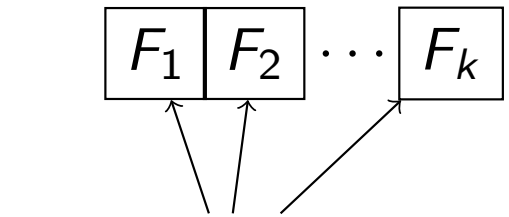
ETH et la complexité paramétrée

Si FPT = W[1] alors ETH est fausse. Quasiprouvons-le !

- Dans ETH, on peut interchanger variables et clauses (pas évident)
- k-CLIQUE en temps $f(k)n^c \to$ 3-SAT en temps $2^{o(m)}$ (m clauses)

Preuve:

- Soit F une 3-CNF avec m clauses et k qu'on fixera plus tard



m/k clauses
$\leq 3m/k$ variables

- G graphe avec un sommet pour chaque solution de F_i
- Arête entre une solution de F_i et une solution de F_j ssi compatibles
- k-clique dans G = solution de F
- G est de taille $\leq 2^{3m/k}k$

ETH et la complexité paramétrée

- k-clique dans G = solution de F
- G est de taille $2^{3m/k}k$
- On peut résoudre 3-SAT sur F en temps

$$O(f(k)2^{3mc/k})$$

- Choisissons $k = f^{-1}(m) = \max\{i \mid f(i) \leq m\}$
- f^{-1} est croissante et tend vers l'infini donc

$$3mc/k = 3mc/f^{-1}(m) = o(m)$$

- on résoud 3-SAT en temps $O(m2^{o(m)}) = O(2^{o(m)})$

Rappels

Jusqu'ici on a vu :

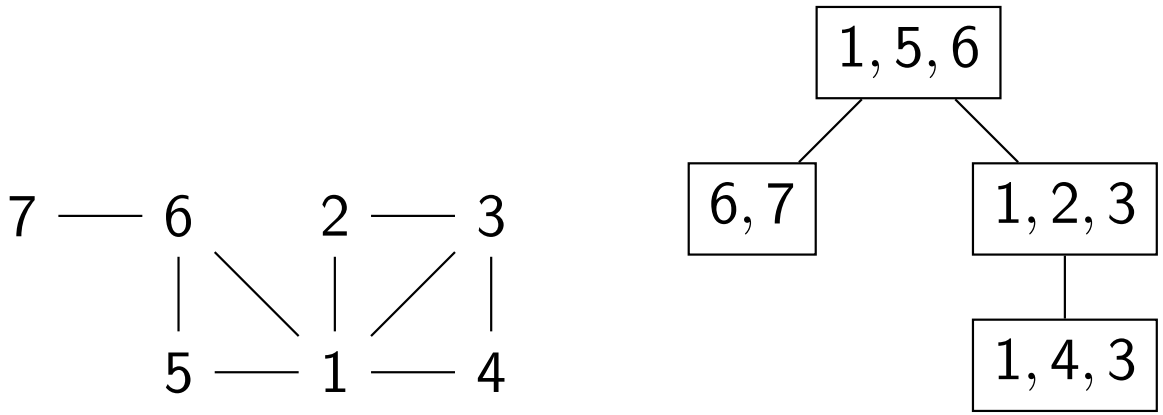
- ce qu'est un problème paramétré
- algorithmes FPT, cas "faciles" pour la complexité paramétrée
- réductions FPT entre les problèmes
- la classe W[1], utile pour montrer qu'un problème n'admet pas d'algorithme FPT
- Les conséquences de W[1] = FPT sur la complexité de SAT

Structure d'un graphe

- Jusqu'ici : paramètres basiques
- Quel paramètre pour SAT ? Pour le voyageur de commerce ?
- On veut des paramètres qui quantifient la structure d'un graphe
- Riche littérature et nombreux paramètres pour différents problèmes
- Ce talk : seulement la treewidth

Décomposition arborescente

- But : mesurer la distance d'un graphe à un arbre
- Exemple : un cycle est presque un arbre. Pas une clique.
- Idée : découper le graphe et le ranger dans un arbre

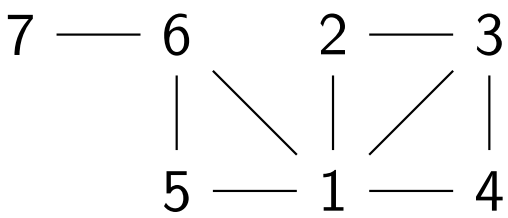


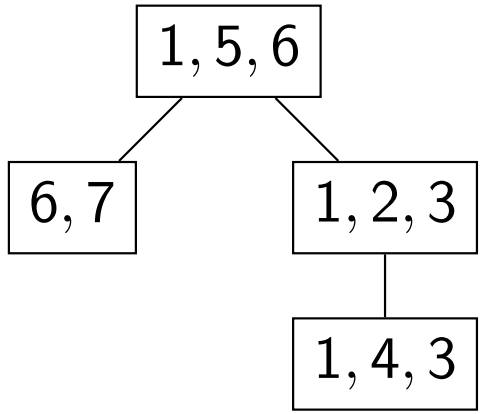
Une décomposition arborescente pour un graphe $G = (V, E)$ est un arbre T tel que :

- sommets de T sont étiquetés par des sous-ensembles de V, *les sacs*
- chaque arête de E est couverte par un sommet t de T
- pour chaque sommet $v \in V$, les sommets de T qui contiennent v forment un sous-arbre connecté de T

Tree width

- Tree width d'une décomposition = taille du plus gros sac -1
- Tree width d'un graphe = tree width minimal de ses décompositions

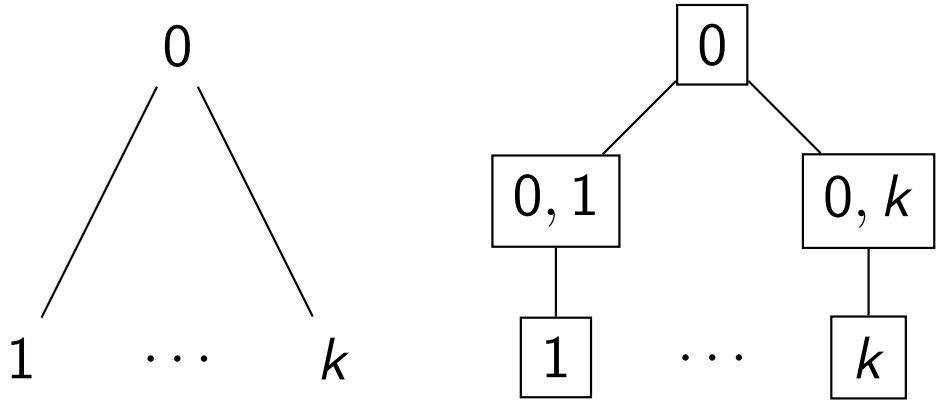




- Cette décomposition est de tree width 2

Exemples classiques

- $tw(G) \leq |V| - 1$
- Les arbres sont de tree width 1 :



- Les cycles sont de tree width 2 :
  $(1,2) \to (1,2,3) \to (1,3,4) \to \cdots \to (1,n-1,n) \to (1,n)$
- Une k-clique est de tree width $k-1$
- Une grille $n \times n$ est de tree width n

Calcul de la tree width

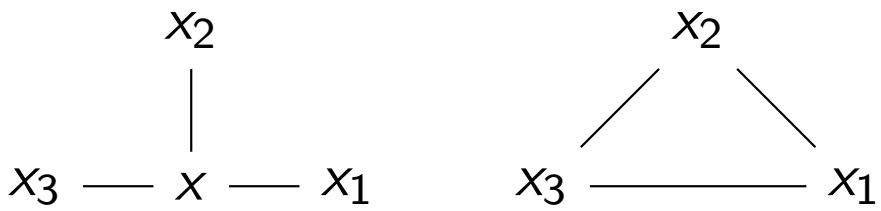
- Décider si un graphe est de treewidth $\leq k$ est NP-complet
- Par contre, ce problème paramétré par k est FPT !
- Donne la décomposition au passage
- Conséquence intéressante : pour montrer qu'un problème paramétré par la tree width est FPT, on peut supposer qu'on a la décomposition
- Algorithme un peu technique mais linéaire en la taille du graphe (Bodlander)

Tree width et algorithmes

- Nombreux problèmes paramétrés par la tree width sont FPT
- Technique souvent similaire : *programmation dynamique* sur la décomposition
- Exemples :
  - trouver une 3-coloration d'un graphe de tree width k en temps $2^{O(k)}|V|$
  - trouver un vertex cover
  - Ça marche même pour le comptage et l'énumération!
  - résoudre voyageur de commerce (en pratique, on a quand même de grande tree width)

Une autre caractérisation

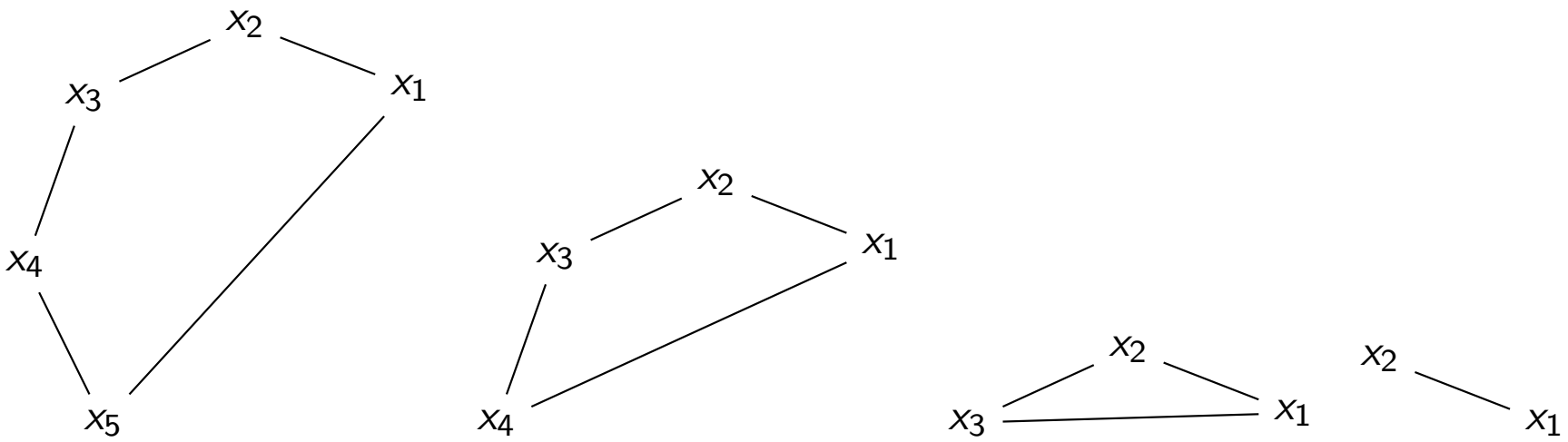
- Autre caractérisation (moins courante) de la tree width
- Généralise l'effeuillage d'un arbre
- Pour un graphe G et un sommet x, on note : G/x le graphe où on enlève x et remplace le voisinage de x par une clique



- Si x est de degré 1, ça revient à enlever x
- G est de treewidth k ssi il existe une élimination des sommets $x_1, \ldots, x_n$ tel que x_i est de degré au plus k dans G_i où $G_1 = G$ et $G_{i+1} = G_i/x_i$
- Appelons ça un effeuillage de largeur k

Exemple : les cycles sont de tree width 2

Effeuillage de largeur 2:



Structure pour SAT

- Comment parler de la structure d'une formule CNF ?
- En associant un graphe !
- Plusieurs façons de faire, ici **graphe primal** : sommets du graphe = variables de la formule et $x \longleftrightarrow y$ si x et y apparaissent dans la même clause.

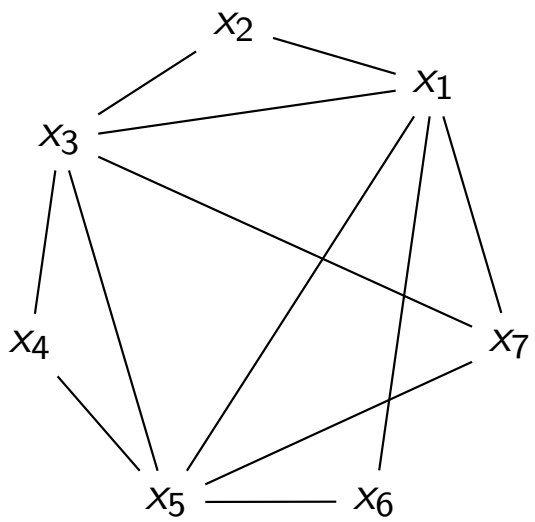


Figure: $(x_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee \neg x_5) \wedge (x_1 \vee x_5 \vee x_6) \wedge (x_1 \vee \neg x_3 \vee x_5 \vee \neg x_7)$

SAT et tree width bornée

- SAT paramétré par la tree width est FPT
- On va le prouver avec l'effeuillage
- Résolution : dire que $(x \vee C) \wedge (\neg x \vee D)$ est satisfiable ssi $C \vee D$ l'est
- Plus généralement : soit x une variable, F une formule avec

$$F = \bigwedge_i (x \vee C_i) \wedge \bigwedge_j (\neg x \vee D_j) \wedge G$$

  avec $x \notin \mathrm{var}(G)$
- On note $res(F, x) = (\bigwedge_{i,j} C_i \vee D_j) \wedge G$
- $res(F, x)$ est satisfaisable ssi F est satisfaisable
- Problème : le nombre de clauses introduites peut exploser

SAT et tree width bornée

- Soit G le graphe primal de F de tree width k
- Le graphe primal de $res(F, x)$ est un sous-graphe de G/x
- Si x est de degré k dans G, clauses contenant x dans F : au plus k variables distinctes
- on introduit au plus 3^k nouvelles clauses dans $res(F, x)$
- En suivant un effeuillage de largeur k, on résoud SAT en temps $O(3^k(m + n))$.

SAT en pratique

- SAT-solvers : outils pour résoudre SAT en pratique
- Peuvent résoudre des instances industrielles avec plusieurs millions de clauses et de variables
- Butent sur des instances de moins de cents variables
- Instances industrielles sont plus structurées mais quelle structure ?

Slide effrayante