Algorithmique avancée

Marc-Antoine Weisser

CentraleSupélec

Saison 2020-21

Retour sur le cours 2

- Limiter la complexité en espace à un impact sur les méthodes de résolution.
- La mémoïsation (ou plus généralement les tables de correspondances) est une technique très utilisée.
- Example avec la programmation dynamique du Knapsack.
- [Pas vu] Les classes de complexité en espace.

Exemple

$$f(i,\ b) = \begin{cases} 0 \text{ si } i = 0 \\ \max(f(i-1,b),\ v_i + \underbrace{f(i-1, b-w_i)}_{\text{si } b-w_i \geq 0}) \end{cases}$$

Limite de poids (i)	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1,\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2,\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5,\ v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6,\ v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7,\ v_5 = 28$	0	1	6	7	7	18	22	28	29	34	35	40

Pseudo polynomial

Codage de l'entrée

- La taille de l'entrée est exprimée en fonction du nombre d'objets de la magnitude des valeurs W, w_i et v_i.
- Au contraire du nombre d'objets, la magnitude des objets, en particulier la taille, dépend du codage (en base 2, en base 10, ou en base 1).
- En fonction de cette base, la taille des instances varie :
  - en base 2 : $n \cdot \log_2(W)$;
  - en base 10 : $n \cdot \log_{10}(W)$;
  - en base 1 : $n \cdot \log_1(W) = nW$.

Complexité

La complexité est polynomial quand le codage est unaire, elle est exponentielle dans les autres cas.

Pseudo-polynomialité

Définition

- Les problèmes qu'on peut résoudre en temps polynomial lorsque leurs entrées sont codées en unaire sont appelés problèmes faiblement *NP*-complets.
- On parle d'algorithme pseudo-polynomiaux.

Lorsque la magnitude est fixé, on peut résoudre ces problèmes en temps polynomial.

Fortement *NP*-complet

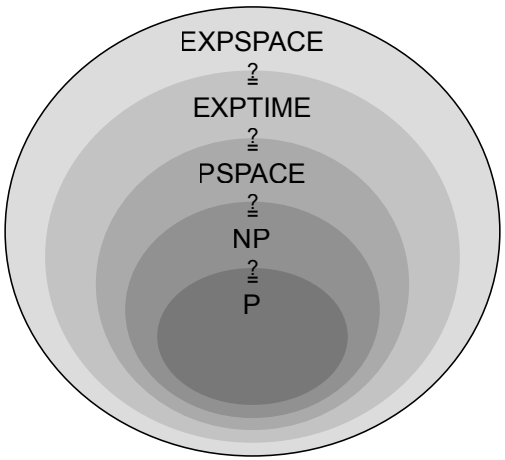
- Un problème est fortement *NP*-complet s'il reste *NP*-complet lorsque ces valeurs numériques sont bornées polynomialement par la taille des instances.

- Faiblement *NP*-complet : Knapsack, Subset Sum, ...
- Fortement *NP*-complet : Clique, Steiner, SAT, Voyageur de commerce, ...

Première partie I

Classe de complexité en espace

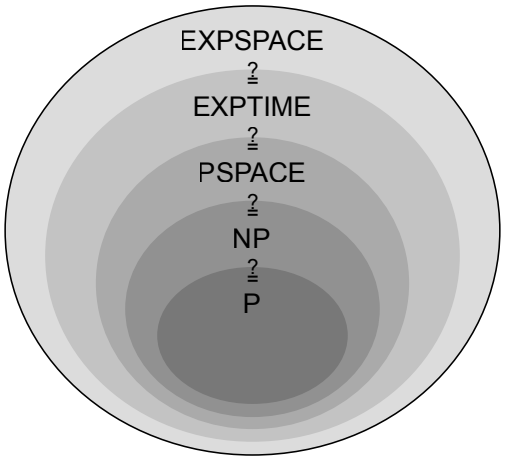
The big picture



Classe P

- L'ensemble des problèmes pouvant être résolu en temps $poly(n)$, avec n la taille de l'instance (par une machine de Turing détermiste).

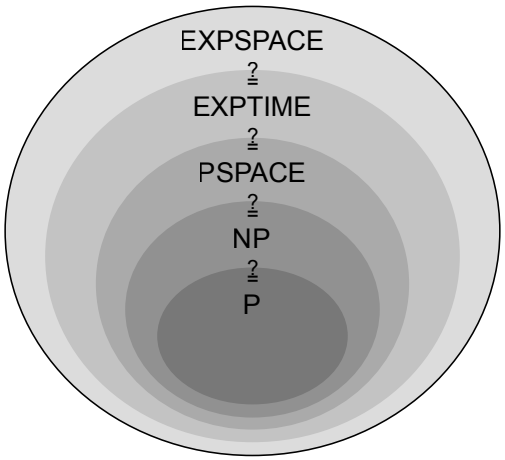
The big picture



Classe NP

- L'ensemble des problèmes pouvant être vérifier en temps $poly(n)$ (par une machine de Turing détermiste).
- L'ensemble des problèmes pouvant être résolu en temps $poly(n)$ (par une machine de Turing non détermiste).
- *NP* pour "non-determistic Turing Machine".

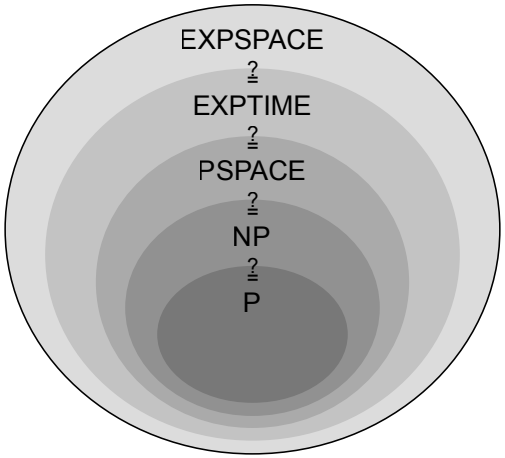
The big picture



Classe PSPACE

- L'ensemble des problèmes décidés par une machine de Turing déterministe (ou non) avec un espace polynomial.
- Formellement, le nombre de cases du ruban de travail visité par une machine de Turing.

The big picture

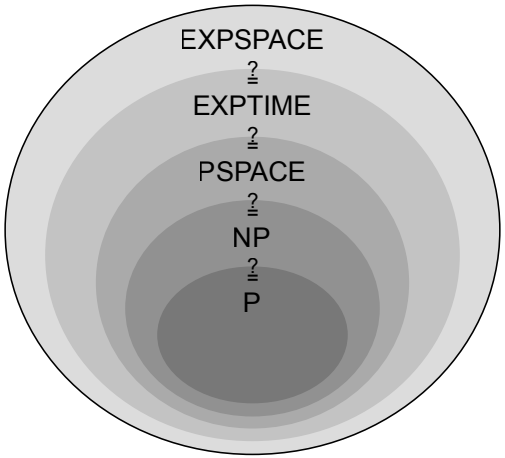


Exemple de problèmes PSPACE-complet

- QBF – Quantified Boolean Formula : Résoudre une formule logique à plusieurs niveaux de quantificateurs :

$$\forall x \; \exists y \; \exists z \; ((x \vee z) \wedge y)$$

The big picture

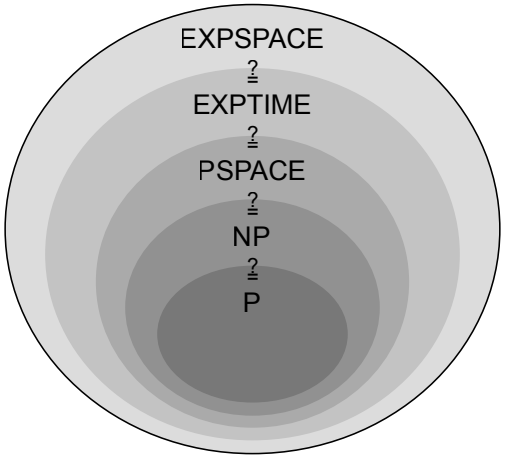


Exemple de problèmes PSPACE-complet

- Expression régulière : Est-ce qu'une expression génère tous les mots d'un langage ?

ˆ [mdp]*a+

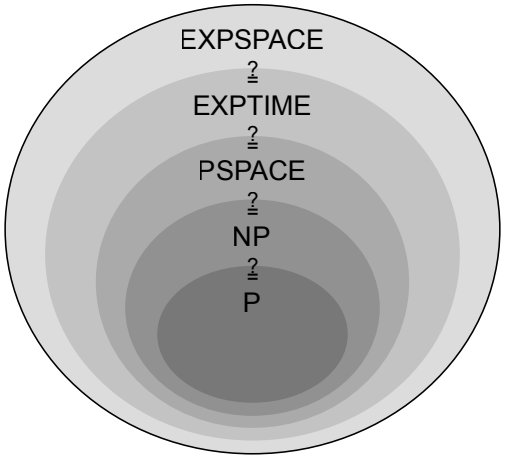
The big picture



Exemple de problèmes PSPACE-complet

- Certains jeux (combinatoires ou pas) : Reversi, Rush Hour, Sokoban, Lemmings, Super Mario Bros, ...

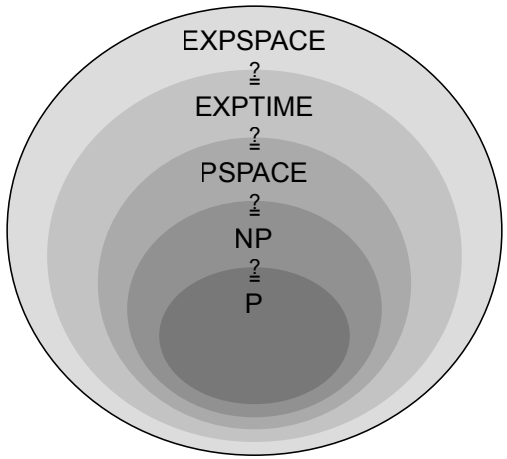
The big picture



EXPTIME / EXPSPACE

- EXPTIME est l'ensemble des problèmes que l'on peut résoudre en temps exponentiel.
- EXPSPACE est l'ensemble des problèmes que l'on peut résoudre en espace exponentiel.

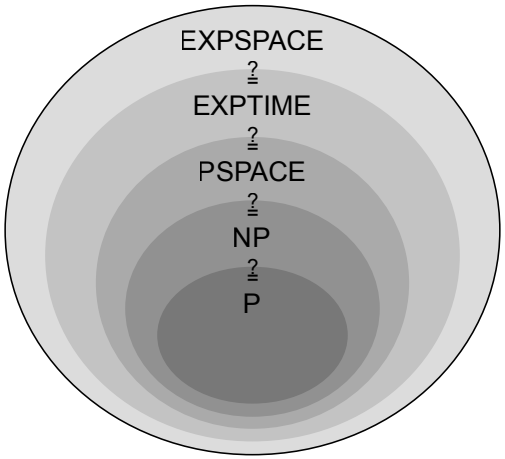
The big picture



Problèmes EXPTIME-complet / EXPSPACE-complet

- EXPTIME : Échecs, Dames, Go
- EXPSPACE : Expression régulière avec exponentiation

The big picture



$$P \subseteq NP \subseteq PSPACE$$
$$PSPACE \subseteq EXPTIME \subseteq EXPSPACE$$

$$PSPACE \subsetneq EXPSPACE$$
$$P \subsetneq EXPTIME$$

Réduction pour les échecs

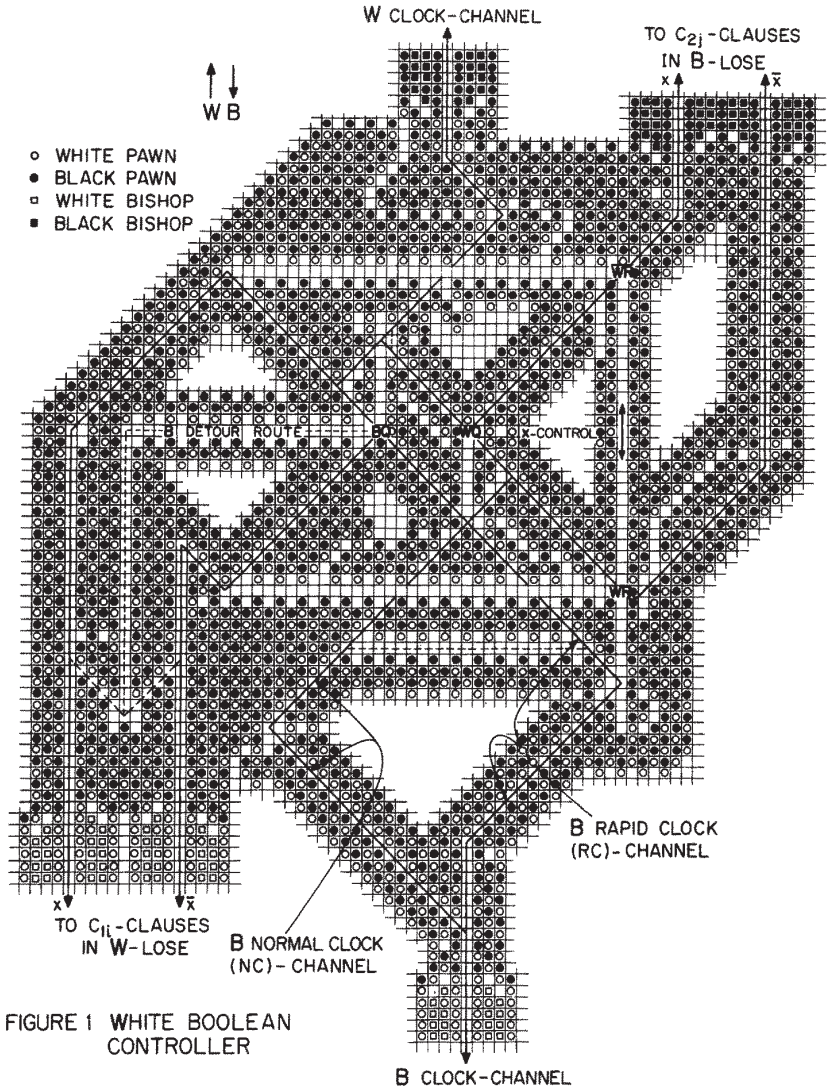


FIGURE 1 WHITE BOOLEAN CONTROLLER

Deuxième partie II

PTAS – FPTAS

Définition

Données :

- $W \in \mathbb{N}$, la capacité du sac ;
- $O : \{1, \ldots, n\}$, un ensemble d'objets ;
- $w_i \in \{1, \ldots, W\}$, la taille des objets ;
- $v_i \in \mathbb{N}$, la valeur des objets.

Question : Trouver $O' \subseteq O$, tel que $\sum_{i \in O'} w_i \leq W$ et $\sum_{i \in O'} v_i$ soit maximum.

Le problème de décision associé est *NP*-complet.

PTAS – Introduit en TD

Principe de l'algorithme vu en TD

- Choisir un $\epsilon > 0$.
- Compléter chaque combinaison de sous-ensemble d'objets de taille $1/\epsilon$, par le glouton sur les éléments restant.
- Renvoyer la meilleure solution rencontrée.

Polynomial Time Approximation Scheme

$\mathcal{A}$ est un PTAS si pour tout $\epsilon > 0$ fixé alors pour toute instance de taille n, $\mathcal{A}$ est une $(1+\epsilon)$-approximation en temps $\mathcal{O}(n^{f(1/\epsilon)})$.

Remarques

- Définition pour la minimisation, dans le cas de la maximisation on a une $(1-\epsilon)$-approximation.
- Si ϵ est fixé, alors la complexité est polynomial.

Fully Polynomial Time Approximation Scheme

$\mathcal{A}$ est un FPTAS si pour tout $\epsilon > 0$ fixé, alors pour toute instance de taille n, $\mathcal{A}$ est une $(1+\epsilon)$-approximation en temps $\mathcal{O}(poly(n, 1/\epsilon))$.

Remarques

- Que ϵ soit fixé ou non, le temps est polynomial.

Programmation dynamique : quelle est la valeur maximum d'un sac avec un poids d'au plus b ?

$$f(i,\, b) = \begin{cases} 0 \text{ si } i = 0 \\ \max(f(i-1,b), v_i + \underbrace{f(i-1, b-w_i)}_{\text{si } b-w_i \geq 0}) \end{cases}$$

Le nombre de colonnes est la taille du sac.

Poids max	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1, v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2, v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5, v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6, v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7, v_5 = 28$	0	1	6	7	7	18	22	28	29	34	35	40

Programmation dynamique : quel est le **poids minimum** d'un sac pour avoir une **valeur d'au moins** v ?

$$f'(i,\ v) = \begin{cases} +\infty \text{ si } i = 0 \\ \min(f'(i-1, v), w_i + \underbrace{f'(i-1, v-v_i)}_{\text{si } v - v_i \geq 0}) \end{cases}$$

Le nombre de colonnes est la somme des valeurs des objets.

Valeur min	0	1	2	3	4	5	6	7	.....	75
$w_1 = 1, v_1 = 1$	0	1	1	1	1	1	1	1	.....	1
$w_2 = 2, v_2 = 6$	0	1	1	1	1	1	2	3	.....	3
$w_3 = 5, v_3 = 18$	0	1	1	1	1	1	2	3	.....	8
$w_4 = 6, v_4 = 22$	0	1	1	1	1	1	2	3	.....	14
$w_5 = 7, v_5 = 28$	0	1	1	1	1	1	2	3	.....	21

Limiter le nombre de colonnes ?

La complexité de l'algorithme de programmation dynamique est en $\mathcal{O}(nV)$ avec $V = \sum v_i$.

- Besoin : accélérer l'algorithme en réduisant le nombre de colonnes.
- Idée : perdre en précision, réduire le nombre de valeurs possibles pour un objet.
- Enjeu : maitriser l'impact sur la qualité de la solution.

Algorithme

- Construire une instance réduite du problème dans laquelle
  - $v'_i = \lfloor \frac{n}{\epsilon} \cdot \frac{v_i}{v_{max}} \rfloor$,
  - $V' = \sum v'_i$,
  - w_i et W restent identiques.
- Appliquer la programmation dynamique en $\mathcal{O}(nV')$.

La transformation des valeurs a pour but de discrétiser les valeurs en n/ϵ marches.

Complexité de la progammation dynnamique :

$$\mathcal{O}(nV')$$

V' est borné :

$$V' \leq n \cdot \max(v_i') \leq n \cdot \lfloor \frac{n}{\epsilon} \cdot \frac{v_{max}}{v_{max}} \rfloor \leq \frac{n^2}{\epsilon}$$

La complexité est polynomial en n et ϵ :

$$\mathcal{O}\left(\frac{n^3}{\epsilon}\right)$$

Théorème

- Soit A une solution optimale au problème non-discrétisé.
- Si la programmation dynamique appliquée à une instance discrétisée renvoie une solution A', alors $(1 - \epsilon)v(A) \leq v(A')$.

Éléments connus

1. Soit $\alpha = \frac{n}{\epsilon v_{max}}$ et $v'_i = \lfloor \alpha v_i \rfloor$

Idée

Pour sous-ensemble d'objets S, avec la discrétisation, on se trompe d'au plus un pour chaque élément de S :

$$\alpha v(S) - |S| \leq v'(S) \leq \alpha v(S)$$

$$v(S) - \frac{|S|}{\alpha} \leq \frac{1}{\alpha} v'(S) \leq v(S)$$

Éléments connus

1. Soit $\alpha = \frac{n}{\epsilon v_{max}}$ et $v'_i = \lfloor \alpha v_i \rfloor$
2. $\forall S, v(S) - \frac{|S|}{\alpha} \leq \frac{1}{\alpha} v'(S) \leq v(S)$

Idée

Soit A' une solution optimale au problème discrétisé, renvoyée par la Prog Dyn.

$$\frac{1}{\alpha} v'(A') \leq v(A')$$

Éléments connus

1. Soit $\alpha = \frac{n}{\epsilon v_{max}}$ et $v'_i = \lfloor \alpha v_i \rfloor$
2. $\forall S, v(S) - \frac{|S|}{\alpha} \leq \frac{1}{\alpha} v'(S) \leq v(S)$
3. $\frac{1}{\alpha} v'(A') \leq v(A')$

Idée

Soit A une solution optimale au problème non discrétisé.

$$v'(A) \leq v'(A')$$
$$v(A') \leq v(A)$$

$$\frac{1}{\alpha} v'(A) \leq \frac{1}{\alpha} v'(A') \leq v(A')$$

Éléments connus

1. Soit $\alpha = \frac{n}{\epsilon v_{max}}$ et $v'_i = \lfloor \alpha v_i \rfloor$
2. $\forall S, v(S) - \frac{|S|}{\alpha} \leq \frac{1}{\alpha} v'(S) \leq v(S)$
3. $\frac{1}{\alpha} v'(A) \leq \frac{1}{\alpha} v'(A') \leq v(A')$

Idée

En combinant 2 et 3 on a

$$v(A) - \frac{|A|}{\alpha} \leq \frac{1}{\alpha} v'(A) \leq \frac{1}{\alpha} v'(A') \leq v(A')$$

$$v(A) - \frac{n}{\alpha} \leq v(A) - \frac{|A|}{\alpha} \leq v(A')$$

Éléments connus

1. Soit $\alpha = \frac{n}{\epsilon v_{max}}$ et $v'_i = \lfloor \alpha v_i \rfloor$
2. $\forall S, v(S) - \frac{|S|}{\alpha} \leq \frac{1}{\alpha} v'(S) \leq v(S)$
3. $\frac{1}{\alpha} v'(A) \leq \frac{1}{\alpha} v'(A') \leq v(A')$

Idée

Avec 1 et $v(A) \geq v_{max}$ on a

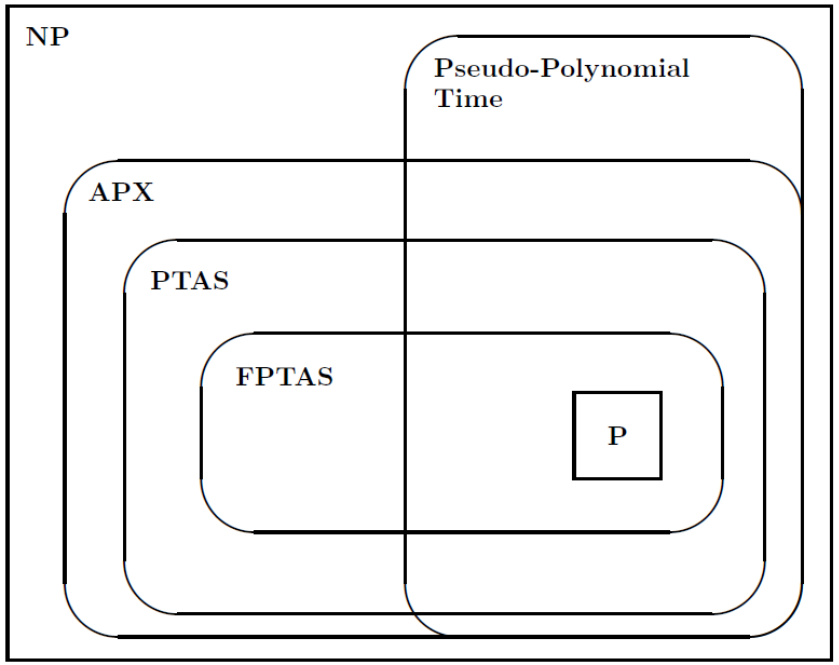
$$v(A) - \frac{n}{\alpha} \leq v(A) - \frac{|A|}{\alpha} \leq v(A')$$

$$v(A)(1-\epsilon) \leq v(A) - \frac{n}{\alpha} \leq v(A) - \frac{|A|}{\alpha} \leq v(A')$$

La programmation dynamique sur les valeurs discrétisées est donc une FPTAS :

- en temps polynomial $\mathcal{O}\left(\frac{n^3}{\epsilon}\right)$,
- elle permet de se rapprocher arbitrairement de l'optimum $(1-\epsilon)$.

Classes de complexité relative à l'approximation



APX

L'ensemble des problèmes d'optimisation pouvant être approché avec un rapport constant.

Troisième partie III

Fixed Parameter Tractable

Surveillance...

Scénario

- On souhaite installer des caméras pour surveiller la circulation dans des couloirs. Afin que l'on dépense le moins, on choisit d'installer les caméra uniquement aux intersections.
- On peut observer la circulation dans un couloir si l'une de ces extrémités est surveillée.

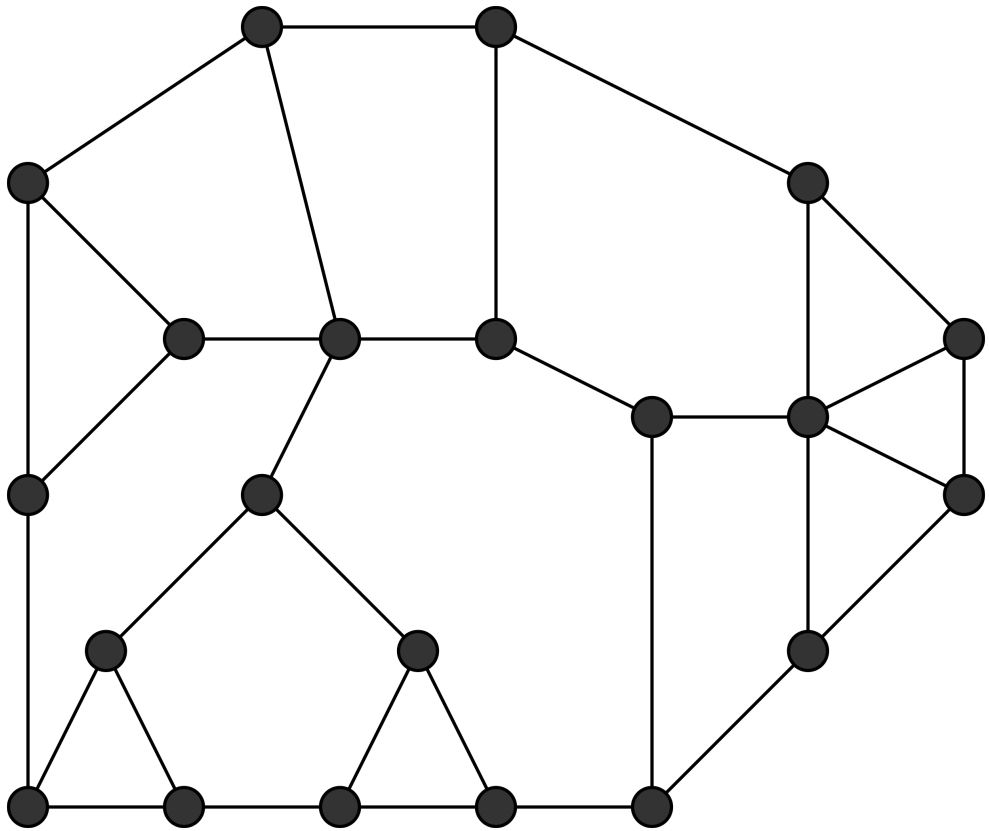
Définition

- Entrée : étant donnés un graphe non orienté $G = (V, E)$ et un entier K
- Question : existe-t-il un $V' \subseteq V$ tel que $|V'| \leq K$ et toute arête $(u, v) \in E$ a l'une de ces extrémités dans V'.

Résultats connus

- Mauvaise nouvelle : problème *NP*-Complet, inapproximable en dessous de 1,1666.
- Bonne nouvelle : approximable en $2 - \frac{\log\log(n)}{2\log(n)}$.

Exemple



Recherche de petite couverture de sommets

Couverture pour k fixé

- Énumérer tout les $\binom{n}{k}$ sous-ensembles de taille k et tester si l'un couvre toutes les arêtes.
- Complexité en $\mathcal{O}(n^{k+1})$

- Rapidement avec k et n cet algorithme devient inefficace.
- Comment faire mieux ?

Idées

Pour chaque arête $(u, v) \in E$, toute couverture de G contient u ou v.

Algorithme – $VC(G, k)$

- Si G n'a pas d'arête renvoyer (Vrai, $\emptyset$).
- si $k = 0$ return Faux.
- soit (u, v) une arête choisie arbitrairement.
- si $C = VC(G - u, k - 1)$ alors renvoyer (Vrai, $C \cup \{u\}$).
- si $C = VC(G - v, k - 1)$ alors renvoyer (Vrai, $C \cup \{v\}$).
- Renvoyer Faux.

Validité

On peut prouver la validité de l'algorithme par induction.

Complexité

- Avec le bon codage de graphe, on peut obtenir $\mathcal{O}(n \cdot 2^k)$.
- C'est un algorithme dont la complexité est paramétrée.

Le meilleur algorihme connu est en $\mathcal{O}(1.2738^k + (k \cdot n))$.

Algorithme FPT

Un algorithme $\mathcal{A}$ est FPT, vis à vis du paramètre κ, si pour toute instance de taille n, la complexité est au plus $f(\kappa) \cdot poly(n)$.

Classe FPT

- Un problème π est FPT vis à vis d'un paramètre κ s'il existe un algorithme FPT vis à vis de κ. On le note (π, κ).
- La classe FPT regroupe l'ensemble des problèmes FPT.

Exemples

- Arbre de Steiner avec k terminaux
- Chemin élémentaire de longueur k

Importance de la classe FPT

- Obtenir des algorithmes pour des problèmes NP dont la complexité est maitrisée.
- Comprendre ce qui rend difficile un problème.

- En pratique on trouve souvent des applications pour lesquels fixer un paramètre est pertinant (ex. réseau, chimie, biologie).

Réduction FPT

Soient (π, κ) et (π', κ') deux problèmes paramétriques. On dit qu'il existe une réduction FPT de (π, κ) vers (π', κ') s'il existe une fonction $R : (\pi, \kappa) \to (\pi', \kappa')$ telle que

- pour toute instance positive $I \in (\pi, \kappa)$ alors $R(I) \in (\pi', \kappa')$ est positive ;
- R est calculable en temps FPT vis à vis de κ ($f(\kappa) \cdot poly(|I|)$) ;
- Il existe une fonction calculable $g : \mathbb{N} \to \mathbb{N}$ telle que $\kappa'(R(I)) \leq g(\kappa(I))$.

La classe FPT est close sous les réductions FPT.

XP

XP est l'ensemble des problèmes $(\pi, kappa)$ qui peuvent être résolu en temps $\mathcal{O}(|I|^{f(\kappa(I)})$ pour toute instance $I \in (\pi, \kappa)$.

Hiérarchie W

- La hiérarchie W est un ensemble de classe de complexité.
- Un problème paramétré appartient à la classe $W[i]$ si chaque instance peut se réduire en temps FPT a un circuit booléan de trame (weft) i.
- On a W[0] = FPT et $W[i] \subseteq W[j], \forall i \leq j$.

Exemples

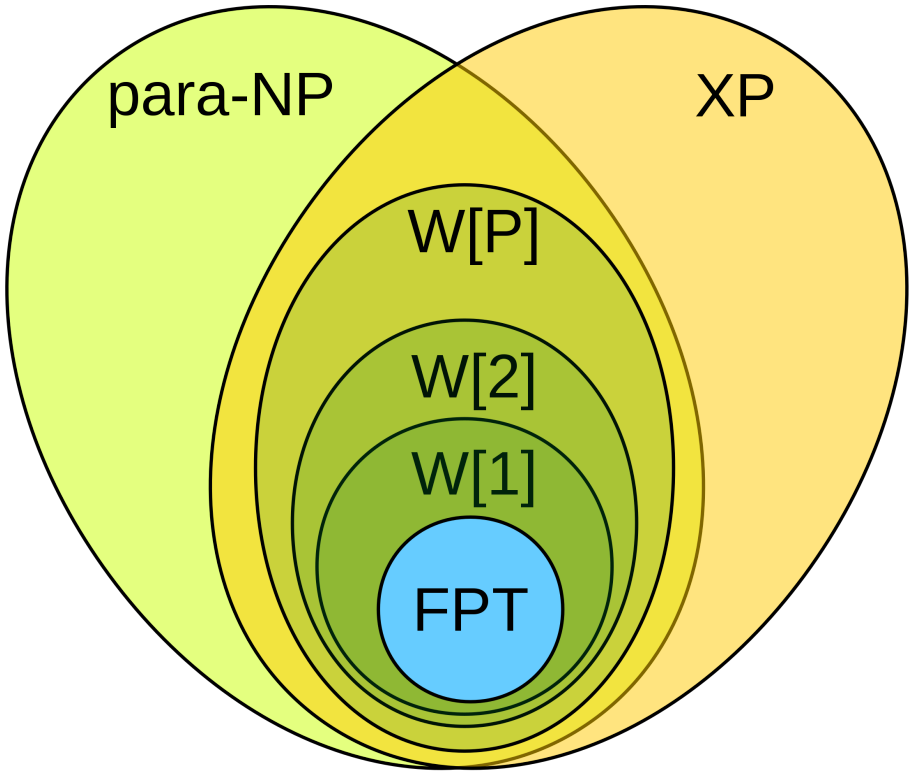
$W[1]$

- Clique de taille k
- Ensemble indépendant de taille k

$W[2]$

- Dominant de taille k
- Set Cover

Imbrication des classes



Quatrième partie IV

Ouverture

Imbrication des classes