Complexité et Algorithmes
Partie III : Problèmes NP-complets – Stratégies de Résolution

G. Fertin
guillaume.fertin@univ-nantes.fr

Université de Nantes, LS2N
Bât 34 – Bureau 301

M1 Informatique – 2019-2020

Sommaire

Un module à problèmes

Le contexte

- Dans tout ce qui va suivre:
  - problèmes de décision (PbD) ou
  - problèmes d'optimisation (PbO)

- PbD: **NOM/Instance/Question**
- PbO: **NOM/Instance/Solution/Mesure**

PbO et PbD: nomenclature

Example (PbD)

SAT

Instance: Une formule booléenne ϕ sous FNC

Question: La formule ϕ est-elle satisfiable ?

Example (PbO)

Maximum Independent Set (Max-IS) (= Ensemble Stable)

Instance: Un graphe $G = (V, E)$

Solution: Un ensemble stable $V' \subseteq V$ de G

Mesure: $|V'|$

A propos des PbO et PbD

Des problèmes qui posent problème

- La très grande majorité des PbO/PbD "intéressants" sont NP-complets
- Un problème Pb NP-complet signifie que:
  1. Il existe au moins une instance de Pb qui est difficile à résoudre
  2. Il n'existe pas d'algorithme polynomial (en temps) pour résoudre Pb (sauf si P = NP)
  3. Pb est au moins aussi difficile que les problèmes SAT, Maximum Independent Set, Cycle Hamiltonien, ...
  4. Si nous proposons un algorithme en temps polynomial pour ce problème, alors P = NP

A propos des PbO et PbD

Des problèmes qui posent problème

- La très grande majorité des PbO/PbD "intéressants" sont NP-complets
- En l'état actuel des connaissances:
  - résultat optimal $\Rightarrow$ temps d'exécution déraisonnable
  - temps d'exécution raisonnable $\Rightarrow$ résultat pas optimal

Problème NP-complet: que faire ?

Le résoudre quand même !

3 points de vue

Utopiste $\rightarrow$ "je résous tous les problèmes d'un coup"

$\Rightarrow$ il "suffit" de montrer que P =NP (!)

1. prendre un problème NP-complet
2. trouver un algorithme polynomial qui le résout
3. au passage, empocher 1 million de dollars (Clay Mathematics Institute Millenium Prize)

Problème NP-complet: que faire ?

Le résoudre quand même !

3 points de vue

Optimiste $\rightarrow$ "j'y arriverai quand même"

- NP-complet signifie qu'au moins une instance est "difficile"...
- ...mais pas forcément toutes !

Petits Cas et Classes d'Instance

Le point de vue optimiste

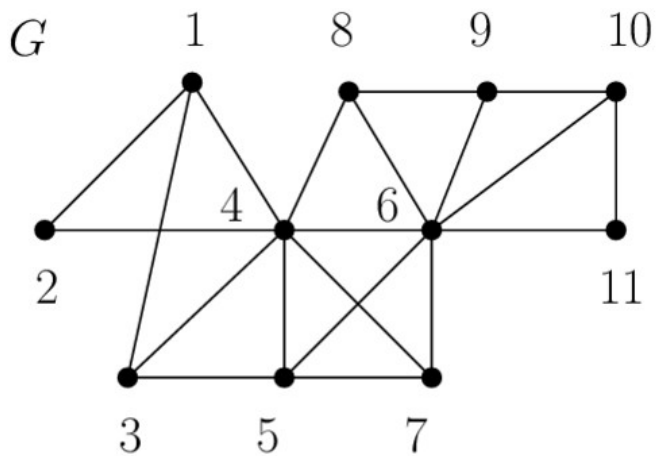
- Pour certaines instances, le problème (pourtant NP-complet) pourrait être résolu en temps polynomial
- Exemples:
  - Petits cas: un/plusieurs paramètres de l'instance est/sont petit/s
  - Classes d'instances: restriction à des instances vérifiant une/plusieurs propriété/s
- Ces instances sont-elles nombreuses ? "intéressantes" ?

Petits Cas et Classes d'Instance

Définitions

$G = (V, E)$ est un graphe ; u est un sommet de G

- Degré de u: $d(u) =$ nombre d'arêtes incidentes à u
- Degré de G: $\Delta(G) = \max_{u \in V} d(u)$
- Clique dans G: sous-ensemble de sommets tous connectés deux à deux



Petits Cas et Classes d'Instance

Example (Problèmes de Graphes)

Exemples de paramètres "Petits Cas" :

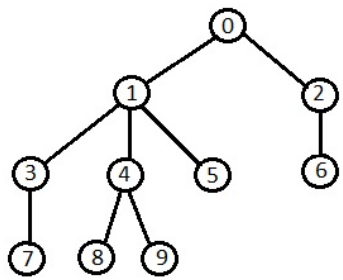
- Degré maximum $\Delta(G) = O(1)$
- Degré moyen $\overline{\Delta(G)} = O(1)$
- Taille de la plus grande clique $\omega(G) = O(1)$

Petits Cas et Classes d'Instance

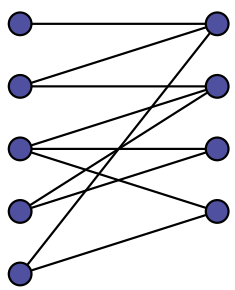
Example (Problèmes de Graphes)

Classes d'Instances:

- Arbres (propriétés = connexe et pas de cycle)



- Graphes bipartis (propriété = pas de cycle impair)

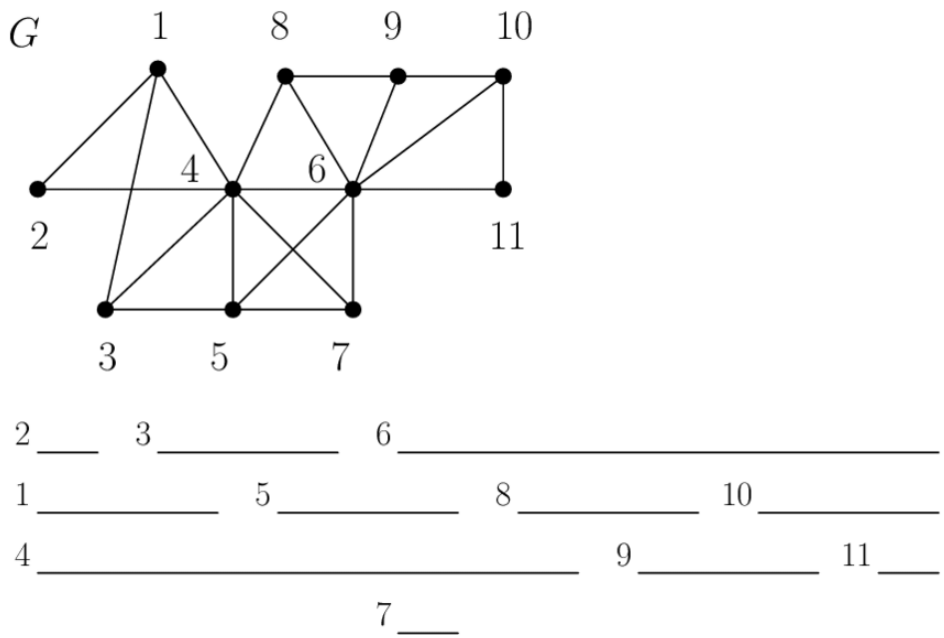


Petits Cas et Classes d'Instance

Example (Problèmes de Graphes)

Classes d'Instances:

- Graphes d'intervalles (propriété = graphe d'intersections d'intervalles)



Problème NP-complet: que faire ?

Le résoudre quand même !

3 points de vue

Raisonnable $\rightarrow$ "trouvons un compromis"

- Compromis qualité de la solution vs temps d'exécution
- Deux possibilités:
  1. Priorité sur le temps d'exécution (temps OK, optimal pas OK)
  2. Priorité sur la qualité de la solution (optimal OK, temps pas OK)

Priorité sur le temps d'exécution

On exige une solution rapide, donc pas optimale
$\Rightarrow$ temps d'exécution polynomial

- Heuristiques: bon fonctionnement empirique, mais aucune garantie sur le résultat $\rightarrow$ benchmarks
- Algorithmes probabilistes: résultat optimal avec une certaine probabilité (ex: 50% de chances d'avoir le résultat optimal)
- **Approximations**: résultat dont l'écart à l'optimal est contrôlé et garanti

Priorité sur la qualité de la solution

On exige une solution optimale, donc pas rapide
$\Rightarrow$ temps d'exécution exponentiel

- Idée: "contrôler" l'exponentielle
- Question: existe-t-il un paramètre k de l'instance du problème tel que
  1. on trouve un algo qui résout optimalement le problème
  2. dont l'exponentielle ne dépend que du paramètre k
  3. k est "petit" dans la pratique ?
- C'est ce qu'on appelle la **Complexité Paramétrée**

Dans la suite de ce cours

Ce que nous verrons

- Optimiste: Petits Cas et Classes d'Instance
- Raisonnable (temps $>$ optimalité): Approximations (et si on a le temps)
- Raisonnable (optimalité $>$ temps): Complexité Paramétrée

Bibliographie (non exhaustive)

- Approximations: "Complexity and Approximation" – Ausiello, Crescenzi, Gambosi, Kann, Marchetti-Spaccamela and Protasi – Springer 2003
- Complexité Paramétrée: "Fundamentals of Parameterized Complexity" - Downey and Fellows – Springer, 2013

Sommaire

Petits Cas et Classes d'Instances

Rappel: petits cas... mais suffisamment nombreux et intéressants !

Example (Quelques Résultats)

- PbD:
  - 2-SAT
  - HORN-SAT
- PbO:
  - MIN-COL $\Delta(G) = 2$
  - MIN-COL Graphes Bipartis
  - MIN-COL Graphes d'Intervalle

Résolution de Petits Cas: 2-SAT

Rappels

k-SAT

Instance: Une formule booléenne ϕ sous FNC, où chaque clause contient k littéraux

Question: La formule ϕ est-elle satisfiable ?

Theorem

k-SAT *est* NP*-complet pour tout* $k \geq 3$

Theorem

2-SAT *est dans* P

2-SAT est dans P

Rappels

- Instance de 2-SAT: formule booléenne de la forme

$$\phi = (a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge \ldots \wedge (a_q \vee b_q)$$

- les a_i et les b_i, $1 \leq i \leq q$, appartiennent à un ensemble de littéraux

$$\mathcal{X} = \{x_1, x_2, \ldots x_n, \overline{x_1}, \overline{x_2} \ldots \overline{x_n}\}$$

2-SAT est dans P

Propriété

Logique booléenne: quelques rappels

- $(a_i \vee b_i) \Leftrightarrow (\overline{\overline{a_i}} \vee b_i)$
- rem: $(\overline{s} \vee t) \Leftrightarrow (s \Rightarrow t) \Leftrightarrow (\overline{t} \Rightarrow \overline{s})$,
- on en déduit que

$$(a_i \vee b_i) \Leftrightarrow (\overline{a_i} \Rightarrow b_i) \Leftrightarrow (\overline{b_i} \Rightarrow a_i)$$

2-SAT est dans P

Idée: représenter ϕ par un graphe G_ϕ: le graphe des implications

Graphe des implications

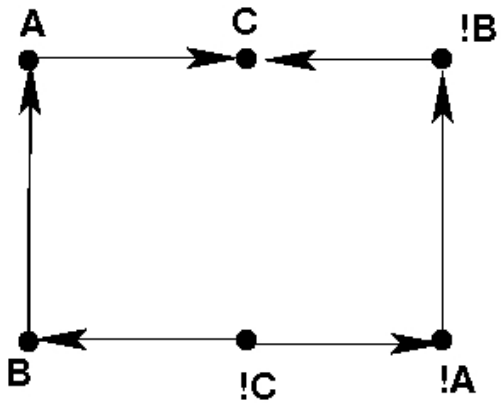
- sommets de G_ϕ = éléments de $\mathcal{X}$
- pour toute clause $(a_i \vee b_i)$ dans ϕ, G_ϕ contient deux arcs:
  - $\overline{a_i} \to b_i$ **et**
  - $\overline{b_i} \to a_i$
  - un arc dans G_ϕ = une implication

2-SAT est dans P

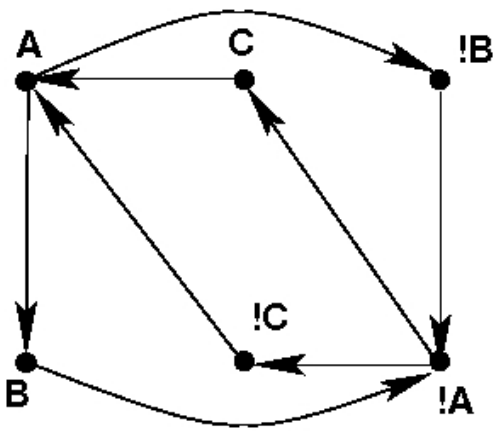
Example (Deux Formules)

$$\phi_1 = (A \vee \overline{B}) \wedge (B \vee C) \wedge (\overline{A} \vee C)$$

$$\phi_2 = (\overline{A} \vee B) \wedge (\overline{A} \vee \overline{B}) \wedge (A \vee \overline{C}) \wedge (A \vee C)$$



Graphe G1



Graphe G2

2-SAT est dans P

Théorème

ϕ est satisfiable si et seulement si il n'existe aucun circuit[1] dans G_ϕ passant par x_i et $\overline{x_i}$, quel que soit $1 \leq i \leq n$

Arguments de preuve:

1. chemin de x_i vers $\overline{x_i}$: incohérent donc $x_i \neq \texttt{Vrai}$
2. chemin de $\overline{x_i}$ vers x_i: incohérent donc $x_i \neq \texttt{Faux}$

[1] circuit = cycle orienté (qui suit les flèches)

2-SAT est dans P

Propriété

Déterminer l'existence (ou non) d'un circuit entre deux sommets u et v dans un graphe orienté G est un problème polynomial

Argument:

1. calculer les composantes fortement connexes (CFC) de G
2. circuit entre u et $v \Leftrightarrow$ une CFC contient à la fois u et v

2-SAT est dans P

Algo 2-SAT

1. construire le graphe de comparaison G_ϕ
2. pour toute paire $x_i, \overline{x_i}$, tester l'existence d'un circuit entre x_i et $\overline{x_i}$
   - un circuit existe $\Rightarrow$ ϕ non satisfiable
   - aucun circuit n'existe $\Rightarrow$ ϕ satisfiable

Remarque: si ϕ est satisfiable, fournir une affectation `Vrai/Faux` à chaque x_i: en temps polynomial

Résolution de Petits Cas: HORN-SAT

Définition

SAT

Instance: Une formule booléenne ϕ sous FNC

Question: La formule ϕ est-elle satisfiable ?

Rappel: SAT est comme k-SAT, mais on ne spécifie pas le k

Résolution de Petits Cas: HORN-SAT

Définition

Formule de Horn = FNC où chaque clause contient au plus une variable positive

Example

$$\phi = (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_2)$$

Résolution de Petits Cas: HORN-SAT

Theorem

Le problème HORN-SAT *est dans* P

Définitions

- Littéral ℓ: une variable x ou $\overline{x}$
- Clause unitaire CU: clause ne contenant qu'*un seul* littéral

Algo de résolution de HORN-SAT

Algo formules de Horn

1. Tant qu'il existe une CU: (ℓ)
   - si $\ell = x_i$, $x_i \leftarrow$ `Vrai`, sinon $x_i \leftarrow$ `Faux`
   - retirer de ϕ toutes les clauses contenant ℓ
   - retirer $\overline{\ell}$ des clauses qui le contiennent
2. Dans l'expression ϕ' qui reste, tout mettre à `Faux`
3. Si ϕ' satisfaite, OK; sinon, ϕ' insatisfiable

Algo de résolution de HORN-SAT

Argument

- CU "forcent" les valeurs des variables
- $\rightarrow$ propagation dans la formule
- Si problème $\rightarrow$ **STOP**: instatisfiable
- Sinon, toutes les clauses de taille $\geq 2 \rightarrow$ au moins un littéral "négatif"
- $\Rightarrow$ tout mettre à `Faux` satisfait ce qui reste

Example

$$\phi = (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_2)$$

Résolution de Petits Cas: Min-Col Bipartis

Définitions

Min-Col

Instance: Un graphe G

Solution: Une coloration propre des sommets de G

Mesure: k, le nombre de couleurs utilisées

Graphe Biparti $G = (V, E)$: G est biparti si:

- il existe une partition de V en V_1 et V_2
- telle que chaque arête de E ait une extrémité dans V_1 et l'autre dans V_2

Résolution de Petits Cas: Min-Col Bipartis

Theorem

Tout graphe biparti peut être proprement colorié en au plus 2 couleurs

Algorithme (graphe $G = (V_1, V_2, E)$):

- si $E = \emptyset \rightarrow$ 1 couleur suffit
- si $E \neq \emptyset \rightarrow$ 2 couleurs sont nécessaires et suffisantes
  - Pour tout $v \in V_1 \rightarrow$ bleu
  - Pour tout $v \in V_2 \rightarrow$ rouge

Résolution de Petits Cas: Min-Col Intervalles

Définitions

Min-Col

Instance: Un graphe G

Solution: Une coloration propre des sommets de G

Mesure: k, le nombre de couleurs utilisées

Graphe d'Intervalles $G = (V, E)$: G est d'intervalles si à chaque sommet v on peut associer un intervalle I_v tel que

$$\text{pour tous sommets } u, v \in V,\ uv \in E \text{ ssi } I_u \cap I_v \neq \emptyset$$

Exemples: C_4 et bull-graph

Résolution de Petits Cas: Min-Col Intervalles

Definition

Soit $G = (V, E)$ un graphe:

- une clique dans G = ensemble $V' \subseteq V$ tel que pour tous $u, v \in V'$, $uv \in E$
- $\omega(G)$ = la taille de la plus grande clique de G

Theorem

Tout graphe d'intervalles peut être proprement colorié en $\omega(G)$ couleurs

Résolution de Petits Cas: Min-Col Intervalles

Theorem

Tout graphe d'intervalles peut être proprement colorié en $\omega(G)$ couleurs

Remarque: $\omega(G)$ couleurs sont nécessaires !... donc c'est l'optimal

Theorem

Pour tout graphe d'intervalles G, déterminer $\omega(G)$ peut se faire en temps polynomial

Sommaire

Approximation

Rappels

- Problèmes NP-complets
- Ici, uniquement PbO
- Priorité: temps $>$ optimalité
- on veut un algorithme en temps polynomial qui retourne une solution approchée **garantie**

Remarques

- Comment garantir l'écart à l'optimal ? écart ou ratio d'approximation
- Est-ce toujours possible ? nouvelles classes de complexité

Garantir l'approximation

Additif vs Multiplicatif

- Additif: la solution trouvée : optimal $\pm$ écart

$$Sol \in [opt - ecart; opt + ecart]$$

- Multiplicatif: la solution trouvée : optimal$\times$ratio

$$Sol \in [opt; opt \times ratio](minimisation)$$

Plus formellement

- Problème d'optimisation Pb
- Algorithme polynomial A: pour toute instance $I \in \text{inst}(Pb)$, retourne une solution $sol_A(I)$
- Solution optimale $\text{opt}(I)$

Garantir l'approximation

Écart à l'optimal (additif)

- écart à l'optimal de A: $e(A)$ ($e(A)$=le plus souvent un nombre)
- inst(Pb) = ensemble des instances autorisées pour Pb
- Algorithme d'approximation A d'écart $e(A)$ si

$$e(A) = \max_{I \in \text{inst}(Pb)} |\text{opt}(I) - sol_A(I)|$$

Garantir l'approximation

Ratio d'approximation (multiplicatif)

- ratio d'approximation de A: $r(A)$ ($r(A)$=le plus souvent un nombre)
- inst(Pb) = ensemble des instances autorisées pour Pb
- Algorithme d'approximation de ratio $r(A)$ si

$$r(A) = \max_{I \in \text{inst}(Pb)} \left\{ \frac{\text{opt}(I)}{sol_A(I)}, \frac{sol_A(I)}{\text{opt}(I)} \right\}$$

(Gauche pour maximisation, Droite pour minimisation)

Ratio d'approximation

A propos du ratio

- Définition unifiée
- Problèmes de maximisation $\Rightarrow \dfrac{\text{opt}(I)}{sol_A(I)}$
- Problèmes de minimisation $\Rightarrow \dfrac{sol_A(I)}{\text{opt}(I)}$
- $\Rightarrow$ pour tous les problèmes,

$$r(A) \geq 1$$

- Remarque: $r(A) = 1$ si et seulement si A est un algorithme exact

Approximation

Exemple Approximation Additive

Min Edge Coloring (Min-ECol)
Instance: un graphe G
Solution: une coloration propre des arêtes de G
Mesure: k, le nombre de couleurs utilisées

Théorème (Vizing - 1964)

Tout graphe G de degré maximum Δ peut être proprement arête colorié en $\Delta + 1$ couleurs.

Remarque: algorithme de coloration A (polynomial) associé au théorème ci-dessus

Approximation

Approximation Additive

- Vizing $\rightarrow$ pour tout graphe G, $sol_A(G) \leq \Delta + 1$
- Définition Edge Coloring $\rightarrow$ pour tout graphe G, $\text{opt}(G) \geq \Delta$
- $e(A) = \max_{G \in \text{inst}(Min-ECol)} |\text{opt}(G) - sol_A(G)|$
- $\Rightarrow e(A) = 1$

Approximation

Additif vs Multiplicatif

- Très peu de problèmes NP-complets pour lesquels on trouve un écart (additif)
- A l'inverse, beaucoup de problèmes NP-complets pour lesquels on trouve un ratio (multiplicatif)
- $\Rightarrow$ à partir de maintenant, **ratio** (multiplicatif) uniquement

Approximation

Attention!

Tous les problèmes NP-complets ne sont pas égaux devant l'approximation

Approximation

ε = constante > 0 arbitrairement petite
c, c' = constantes n = taille des données

Différents types d'approximations

- Schéma d'approximation en temps (pleinement) polynomial $\rightarrow r(A) = 1 + \varepsilon$
- Ratio d'approximation constant $\rightarrow r(A) = O(1)$
- Ratio d'approximation logarithmique $\rightarrow r(A) = O(\log n)$
- Ratio d'approximation poly-logarithmique $\rightarrow$ $r(A) = O((\log n)^c \cdot n^{c'})$
- Ratio d'approximation polynomial $\rightarrow r(A) = O(n^c)$

Min-Col - Ratio polynomial

Min-Col
Instance: Un graphe G
Solution: Une coloration propre des sommets de G
Mesure: k, le nombre de couleurs utilisées

Théorème
Min-Col est NP-complet

Min-Col - Ratio polynomial

Théorème

Soit G un graphe à n sommets. Il existe un algorithme d'approximation de ratio n pour le problème Min-Col.

Preuve:

- Algo A de coloration: une couleur unique à chaque sommet
- $\Rightarrow sol_A(G) = n$
- Rem: $\text{opt}(G) \geq 1$ (ex: graphe *vide*, càd sans arête)
- $\Rightarrow r(A) \leq \frac{n}{1}$ pour toutes les instances

Couverture par les Sommets - Ratio 2

Définitions

- Soit $G = (V, E)$ un graphe
- Couverture par les Sommets = Vertex Cover = VC = ensemble de sommets $V' \subseteq V$
- Vertex Cover: $\forall\, uv \in E$, au moins un des sommets u, v est dans V'
- $\rightarrow$ V' couvre toutes les arêtes de G

Couverture par les Sommets - Ratio 2

Min Vertex Cover (Min-VC) (=Couverture par les Sommets)
Instance: Un graphe $G = (V, E)$
Solution: Un Vertex Cover $V' \subseteq V$ de G
Mesure: $k = |V'|$ le nombre de sommets de V'

Couverture par les Sommets - Ratio 2

Théorème

Min-VC est NP-complet

Théorème

Il existe un algorithme d'approximation de ratio 2 pour Min-VC

Couverture par les Sommets - Ratio 2

Algorithme approx-2-VC

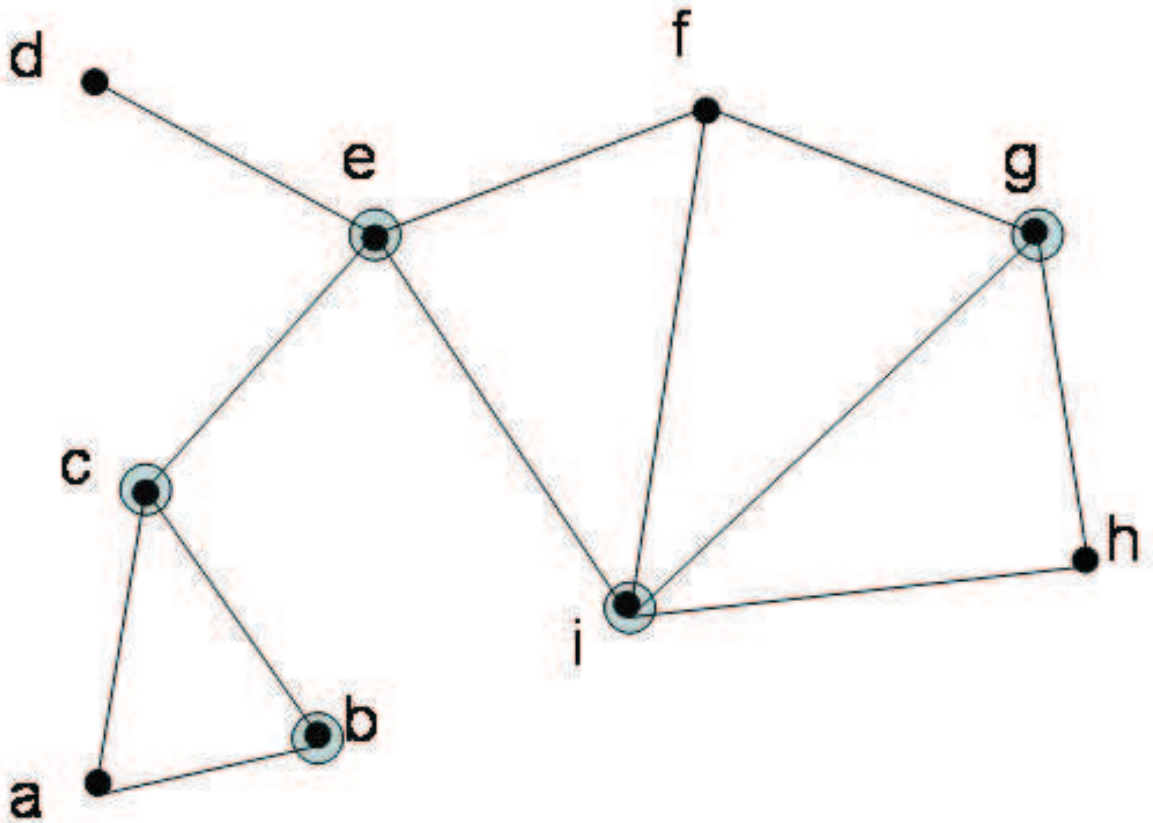
Entrée: graphe $G = (V, E)$

1. $V' \leftarrow \emptyset$
2. Tant que $E \neq \emptyset$ faire
   - choisir une arête uv dans E
   - $V' \leftarrow V' \cup \{u\} \cup \{v\}$
   - $E' \leftarrow$ arêtes incidentes à u et v
   - $E \leftarrow E - E'$

Remarque: algo glouton (greedy en anglais)

Algo approx-2-VC

Example



Couverture par les Sommets - Ratio 2

Théorème

approx-2-VC est un algorithme de 2-approximation pour Min-VC

Analyse de approx-2-VC

- les arêtes choisies pendant l'exécution sont indépendantes (pas de sommets en commun)
- pour couvrir chacune de ces arêtes, au moins un sommet
- $\Rightarrow OPT \geq \frac{|V'|}{2}$, d'où $\frac{|V'|}{OPT} \leq 2$

Remarques:

- algo très simple (glouton)
- analyse simple
- ...et pourtant ce ratio 2 est le meilleur qu'on connaisse!

Max-3-SAT - Ratio 2

Max-3-SAT

Instance: Une formule booléenne ϕ sous FNC, où chaque clause contient au plus 3 littéraux

Solution: Une affectation `Vrai`/`Faux` à chaque variable

Mesure: k, le nombre de clauses satisfaites

Remarque: Attention, ici on veut maximiser le nombre de clauses satisfaites $\Rightarrow$ PbO

Théorème

Il existe un algorithme d'approximation de ratio 2 pour Max-3-SAT

Max-3-SAT - Ratio 2

Algorithme approx-2-Max-3-SAT

Entrée: formule 3-SAT sous FNC ϕ

1. $nb_V \leftarrow$ nombre de clauses de ϕ satisfaites quand toutes les variables sont à `Vrai`
2. $nb_F \leftarrow$ nombre de clauses de ϕ satisfaites quand toutes les variables sont à `Faux`
3. Si $nb_V \geq nb_F$ alors mettre toutes les variables à `Vrai`
4. Sinon mettre toutes les variables à `Faux`

Max-3-SAT - Ratio 2

Théorème

approx-2-Max-3-SAT est un algorithme de 2-approximation pour le problème Max-3-SAT

Analyse de approx-2-Max-3-SAT

$C =$ nb de clauses dans ϕ

- nb clauses satisfaites $C_s = \max\{n_F, n_T\}$
- Remarque: $n_F + n_T \geq C$
- $\Rightarrow C_s \geq \frac{C}{2}$
- Or $OPT \leq C$

$$\frac{OPT}{C_S} \leq 2$$

Comment obtenir une r-approximation ?

Pas de recette miracle

- Pas de méthode standard :-(
- (...comme pour les réductions de type "NP-complet")
- avoir l'intuition qu'un algorithme A peut fonctionner
- souvent (pas toujours): l'algorithme A est simple
- essai/erreur + expérience

Comment obtenir une r-approximation ?

Cela dit...

Une fois l'algorithme A déterminé (ici, pb de minimisation)

1. Montrer que pour toute instance I, $sol_A(I) \leq X$
2. Montrer que pour toute instance I, $\text{opt}(I) \geq Y$
3. Croiser les doigts pour que $\frac{X}{Y} = \text{constante}$
4. Si c'est le cas, $r = \frac{X}{Y}$

$\Rightarrow$ Deux conseils:

- trouver une bonne borne inf. pour $\text{opt}(I) \rightarrow$ bonne analyse du problème
- trouver une bonne borne sup. pour $sol_A(I) \rightarrow$ bonne analyse de l'algo A

Ce que [ne] veut [pas] dire r-approximation

Ce que ça veut dire

- quelle que soit l'instance, on a toujours une solution $\leq r \cdot opt$
- cette solution est obtenue en temps polynomial

Ce que [ne] veut [pas] dire r-approximation

Ici, pb de minimisation

Ce que ça ne veut pas dire

- que c'est toujours $r \cdot opt$!
- Par exemple:
  - très souvent, *sol* proche de opt
  - très rarement, *sol* proche de $r \cdot$ opt
- mais l'analyse de l'algo ne permet pas de dire mieux que r...
- ...dans les faits, peut être un ratio $r' << r$

$\Rightarrow$ même si r est "grand", un algo de r-approximation reste intéressant

Sommaire

Approximation et Classes de Complexité

Que vient faire la complexité là-dedans ?

- PbO initial NP-complet
- on cherche un algo d'approximation polynomial et de ratio r
- $\Rightarrow$ Idée: il peut être difficile (au sens de la complexité) d'approximer en-dessous d'un certain ratio
- $\Rightarrow$ **complexité pour l'approximation**
- Création de nouvelles classes de complexité

La classe **FPTAS**

FPTAS: Fully Polynomial-Time Approximation Scheme

Rappel: n = taille des données

Definition

FPTAS = classe des PbO approximables:

1. avec ratio $r = 1 + \varepsilon$ pour tout $\varepsilon > 0$, et
2. en temps polynomial en n **et** $\frac{1}{\varepsilon}$

La classe **FPTAS**

A propos de **FPTAS**

- C'est le mieux que l'on puisse espérer:
  - compromis temps/optimalité
  - dépendant d'un seul paramètre ε
  - complexité polynomiale pour toute valeur fixée de ε
- mais... peu de problèmes sont dans cette classe :-(

La classe **PTAS**

PTAS: Polynomial-Time Approximation Scheme

Rappel: n = taille des données

Definition

PTAS: classe des PbO approximables:

1. avec ratio $r = 1 + \varepsilon$ pour tout $\varepsilon > 0$, et
2. en temps polynomial en n

La classe **PTAS**

Remarques

- **FPTAS** $\subseteq$ **PTAS**
- compromis temps/optimalité
- dépendant d'un seul paramètre ε
- exponentiel en $\frac{1}{\varepsilon} \rightarrow$ peu pratique pour de très petites valeurs de ε

La classe **APX**

APX: Approximable

Rappel: n = taille des données

Definition

APX: classe des PbO approximables avec ratio r constant

Ex: Min-VC et Max-3-SAT sont dans **APX**

Remarques

- implicite: algorithme d'approximation polynomial en n
- **FPTAS** $\subseteq$ **PTAS** $\subseteq$ **APX**

Un exemple de problème dans **FPTAS**

Problème du Sac à Dos (KNAPSACK) - Définition informelle

- on a N objets, et 1 sac à dos
- chaque objet possède:
  - un poids $w_i > 0$
  - une valeur $v_i > 0$
- le sac à dos peut porter un poids maximum W
- **but**: remplir le sac à dos de façon à maximiser la valeur des objets qu'il contient

Un exemple de problème dans **FPTAS**

KNAPSACK

Instance: Un ensemble $X = \{x_1, x_2 \dots x_N\}$ où chaque x_i est un couple (v_i, w_i) d'entiers ; un entier W

Solution: Un sous-ensemble $S \subseteq X$ tel que $\sum_{x_i \in S} w_i \leq W$

Mesure: $V = \sum_{x_i \in S} v_i$

- KNAPSACK est un PbO
- Valeur totale V à maximiser

KNAPSACK est dans **FPTAS**

Example (KNAPSACK)

	valeur v_i	poids w_i
x_1	13	1
x_2	65	2
x_3	181	5
x_4	221	6
x_5	284	7

et $W = 11$

$\Rightarrow$ OPT=402 en choisissant x_3 et x_4

Theorem

Le problème KNAPSACK *est* NP*-complet*

KNAPSACK est dans **FPTAS**

Theorem

Pour tout $\varepsilon > 0$, il existe un algorithme de $(1-\varepsilon)$-approximation pour KNAPSACK, *et dont la complexité en temps est en* $O(N^3 \cdot \frac{1}{\varepsilon})$

Remarque: ici, ratio< 1 ($r = 1 - \varepsilon < 1$)... mais c'est pour faciliter la présentation!

Équivalent à ratio $1 + \varepsilon'$ avec $\varepsilon' = \frac{\varepsilon}{1-\varepsilon}$.

Theorem

KNAPSACK *est dans* **FPTAS**

KNAPSACK est dans **FPTAS**

Programmation Dynamique

- Principe de la Prog. Dyn. = calculs réalisés sur la base de calculs antérieurs
- Souvent: une table de Prog. Dyn. doit être remplie
- Solution: un élément particulier de la table
- Taille de la table de Prog. Dyn. $\sim$ complexité de l'algorithme

KNAPSACK est dans **FPTAS**

Algorithme de Programmation Dynamique

$PM[p, v]$ = poids minimum du sac à dos

- quand sa valeur totale est égale à v et
- quand les éléments sont choisis parmi $x_1, x_2 \ldots x_p$
- Cas 1: on utilise x_p
  - s'appuie sur $x_1, x_2 \ldots x_{p-1}$
  - à partir de la valeur $v - v_p$
  - $\rightarrow PM[p-1, v-v_p]$
- Cas 2: on n'utilise pas x_p
  - s'appuie sur $x_1, x_2 \ldots x_{p-1}$
  - à partir de la valeur v
  - $\rightarrow PM[p-1, v]$

KNAPSACK est dans **FPTAS**

$$PM[p,v] = \begin{cases} 0 & \text{si } p = 0 \\ PM[p-1,v] & \text{si } v_p > v \\ min\{w_p + PM[p-1, v-v_p]; PM[p-1,v]\} & \text{sinon} \end{cases}$$

Valeur max = maximum des V parmi les $PM[N,V] \leq W$

Remarques

- N= nombre d'objets, V^*= valeur optimale pour KNAPSACK
- Taille de la table de Prog. Dyn.: NV^*
- $\Rightarrow$ algorithme en $O(NV^*)$
- Taille des données: $N \max\{\log w_i\} + N \max\{\log v_i\} + \log W$
- $\Rightarrow$ exponentiel en la taille des données!!!

KNAPSACK est dans **FPTAS**

FPTAS: intuition

Partant d'une instance I

- diviser les valeurs par un même facteur X pour diminuer l'échelle sur ces valeurs
- arrondir $\rightarrow$ instance I' où $v'_i = \lfloor \frac{v_i}{X} \rfloor$ pour tout $1 \leq i \leq N$
- utiliser l'algo de Prog. Dyn. en $O(NV^*)$ sur I'
- remonter la solution (exacte) pour I' vers une solution (approchée) pour I

KNAPSACK est dans **FPTAS**

Example (Instance I)

	valeur v_i	poids w_i
x_1	134 221	1
x_2	656 342	2
x_3	1 810 013	5
x_4	2 217 800	6
x_5	2 843 199	7

KNAPSACK est dans **FPTAS**

$\Rightarrow$ avec $X = 10^4$ et $v'_i = \lfloor \frac{v_i}{X} \rfloor$

Example (Instance I')

	valeur v'_i	poids w'_i
x'_1	13	1
x'_2	65	2
x'_3	181	5
x'_4	221	6
x'_5	284	7

KNAPSACK est dans **FPTAS**

Notations

- V = la plus grande valeur de l'instance I
- $1-\varepsilon$ = ratio d'approximation souhaité
- $\Rightarrow$ on pose

$$X = \frac{\varepsilon \cdot V}{N}$$

Example (Notre choix: $X = 10^4$. Quel ε ?)

- Dans l'exemple, on a choisi $X = 10^4$
- $\Rightarrow \varepsilon = \frac{N \cdot X}{V} = \frac{5 \cdot 10^4}{2\,843\,199} = 0.0176$
- $\Rightarrow 1-\varepsilon \sim 0.983$
- on est à $\geq 98.3\%$ de l'optimal

KNAPSACK est dans **FPTAS**

Preuve de l'appartenance à **FPTAS**

- Supposons que $w_i \leq W$ pour tout i
- ...sinon on réduit le problème en éliminant de tels éléments
- On en déduit que

$$V \leq V^* \leq N \cdot V$$

- l'élément de valeur V (seul) est une solution
- on ne peut pas dépasser N fois l'élément de valeur V

KNAPSACK est dans **FPTAS**

Notations

- S^* = ensemble des x_i choisis pour résoudre optimalement I
- Valeur optimale: V^*
- S'^* = ensemble des x'_i choisis pour résoudre optimalement I'
- Valeur optimale: V'^*
- V: plus grande valeur de I
- V': plus grande valeur de I'

KNAPSACK est dans **FPTAS**

Constatations

$v'_i = \lfloor \frac{v_i}{X} \rfloor \Rightarrow \frac{v_i - X}{X} < v'_i \leq \frac{v_i}{X}$

$\Rightarrow v_i \geq X \cdot v'_i$

Donc

$$(1) \quad \sum_{x'_i \in S'^*} v_i \geq \sum_{x'_i \in S'^*} X \cdot v'_i$$

KNAPSACK est dans **FPTAS**

Constatations

$\sum_{x'_i \in S'^*} v'_i \geq \sum_{x_j \in S^*} v'_j$ car S'^* optimal

Donc

$$\Rightarrow (2) \quad \sum_{x'_i \in S'^*} X \cdot v'_i \geq \sum_{x_j \in S^*} X \cdot v'_j$$

KNAPSACK est dans **FPTAS**

Constatations

$v'_i = \lfloor \frac{v_i}{X} \rfloor \Rightarrow \frac{v_i - X}{X} < v'_i \leq \frac{v_i}{X}$

$\Rightarrow X \cdot v'_i > v_i - X$

Donc

$$(3) \sum_{x_j \in S^*} X \cdot v'_j \geq \sum_{x_j \in S^*} (v_j - X)$$

KNAPSACK est dans **FPTAS**

En résumé

$$(1) \quad \sum_{x'_i \in S'^*} v_i \geq \sum_{x'_i \in S'^*} X \cdot v'_i$$

$$\Rightarrow (2) \quad \sum_{x'_i \in S'^*} X \cdot v'_i \geq \sum_{x_j \in S^*} X \cdot v'_j$$

$$(3) \sum_{x_j \in S^*} X \cdot v'_j \geq \sum_{x_j \in S^*} (v_j - X)$$

(1) puis (2) puis (3) donnent:

$$\sum_{x'_i \in S'^*} v_i \geq \sum_{x_j \in S^*} (v_j - X)$$

KNAPSACK est dans **FPTAS**

Nous y sommes presque... on sait:

$$\sum_{x_i' \in S'^*} v_i \geq \sum_{x_j \in S^*} (v_j - X)$$

Or, $\sum_{x_j \in S^*} X \leq X \cdot N$

Donc

$$\sum_{x_i' \in S'^*} v_i \geq (\sum_{x_j \in S^*} v_j) - X \cdot N$$

KNAPSACK est dans **FPTAS**

Rappel (slide précédent) $\sum_{x'_i \in S'^*} v_i \geq (\sum_{x_j \in S^*} v_i) - X \cdot N$
On a une solution V_{sol} qui vérifie

$$V_{sol} \geq V^* - X \cdot N$$

Preuve de l'appartenance à **FPTAS**

- Comme $X = \frac{\varepsilon \cdot V}{N}$, on a $V_{sol} \geq V^* - \varepsilon \cdot V$
- Or $V \leq V^*$
- Donc $V_{sol} \geq (1 - \varepsilon) \cdot V^*$
- $\Rightarrow$ Algo d'approximation de ratio $1 - \varepsilon$

Montre que KNAPSACK est (au moins) dans **PTAS**

Reste à étudier la complexité de l'algorithme pour montrer que KNAPSACK est dans **FPTAS**

KNAPSACK est dans **FPTAS**

Complexité en temps de KNAPSACK

- Prog. Dyn. $\Rightarrow O(NV'^*)$
- Or, $V'^* \leq NV'$
- $\Rightarrow O(NV'^*) = O(N^2V')$
- Mais $V' = \lfloor \frac{V}{X} \rfloor$
- Donc $V' \leq \frac{V}{X} = \frac{N}{\varepsilon}$
- $\Rightarrow$ Complexité en temps de

$$O(N^3 \cdot \frac{1}{\varepsilon})$$

Et si ça se passe mal ?

La classe **APX**-dur

- classes **FPTAS**, **PTAS** $\leftrightarrow$ résultat positif, ratio aussi petit qu'on veut
- ...mais pas toujours possible !

On se place sous des hypothèses raisonnables de complexité (HRC), càd

$$\boxed{\mathit{HRC} \leftrightarrow \mathrm{P} \neq \mathrm{NP}}$$

Definition

Problème **APX**-dur: sous HRC, le problème n'est pas dans **PTAS**

Inapproximabilité

APX-dur: ce qui arrive souvent

- pas d'espoir de ratio $r = 1 + \varepsilon$
- parfois même pas approximable sous un ratio r' constant
- ...on parle de ratio d'inapproximation r'
- dit autrement, sous HRC, il n'existe pas d'algo d'approximation polynomial de ratio $\leq r'$ (r'=constante)
- ...mais le problème peut être dans **APX** quand même (ratio r OK), càd:
  - ratio $\leq r'$: difficile (sous HRC)
  - ratio $\geq r$: polynomial
- on parle de problème **APX**-complet
- Problèmes **APX**-complets: les plus difficiles de la classe **APX**

Inapproximabilité

Definition

APX-complet = dans **APX** et **APX**-dur

Retour sur NP

- NP-complet = dans NP et NP-dur
- NP-dur $\rightarrow$ par réduction

Une façon de voir les choses

- **APX** $\leftrightarrow$ NP
- **APX**-dur $\leftrightarrow$ NP-dur (**APX**-dur $\rightarrow$ par réduction)
- **APX**-complet $\leftrightarrow$ NP-complet

Inapproximabilité (exemple 1)

Theorem

Le problème Min-VC *n'est pas approximable sous un ratio* 1.1666

Remarques

- Min-VC n'est pas dans **PTAS** $\Rightarrow$ Min-VC est **APX**-dur
- Comme Min-VC est 2-approximable:
  - Min-VC est **APX** $\Rightarrow$ Min-VC est **APX**-complet
  - Meilleur ratio possible pour Min-VC: entre 1.1666 et 2

Inapproximabilité (exemple 2)

Rappel:

MAXIMUM INDEPENDENT SET (MAX-IS) (= Ensemble Stable)
Instance: Un graphe $G = (V, E)$
Solution: Un ensemble stable $V' \subseteq V$ de G
Mesure: $|V'|$

Theorem
Le problème MAX-IS *n'est pas approximable sous un ratio* $\mathcal{O}(n^{1-\varepsilon})$ *pour tout* $\varepsilon > 0$.

$\Rightarrow$ MAX-IS n'est pas dans **APX**

Et si ça se passe mal ?

Ce qu'on peut faire pour

- Montrer qu'un problème est **APX**-dur (càd, pas dans **PTAS**):
  - raisonnement ou
  - réduction qui préserve l'approximation
- Montrer qu'un problème n'est pas dans **APX**:
  - raisonnement ou
  - réduction qui préserve l'approximation

Preuve par raisonnement

Raisonnement

- Notre problème: Pb ; un autre: Pb'
- (souvent) Preuve par contradiction. par ex:
  - Pb' étant NP-complet...
  - ...supposons que Pb soit r-approximable...
  - ...alors (après argumentaire) Pb' serait polynomial
  - $\Rightarrow$ Pas possible sous HRC
  - donc Pb est **APX**-dur (si $r = 1 + \varepsilon$), voire n'appartient pas à **APX** (si r =constante)

Preuve par raisonnement - Exemple

Cycle hamiltonien dans un graphe G = cycle passant une et une seule fois par chaque sommet de G

Cycle Hamiltonien
Instance: Un graphe $G = (V, E)$
Question: Existe-t-il un cycle hamiltonien dans G ?

Traveling Salesman Problem (TSP)
Instance: Un graphe complet à n sommets K_n, un poids w_e sur chaque arête e du graphe
Solution: Un cycle hamiltonien CH dans G
Mesure: Longueur de $CH = \sum_{e \in CH} w_e$

Rem: TSP est un PbO, Cycle Hamiltonien est un PbD

Theorem

Cycle Hamiltonien *est* NP-*complet*

Preuve par raisonnement - Exemple

On va prouver ceci:

Theorem

Sous HRC, TSP *n'est pas dans* **APX**

Par contradiction

- on suppose qu'il existe un algo de r-approx. pour TSP $r \geq 1$
- on va en déduire que dans ce cas Cycle Hamiltonien est dans P
- or Cycle Hamiltonien est NP-complet
- sauf si P =NP, il y a contradiction
- $\Rightarrow$ supposition initiale fausse
- $\Rightarrow$ TSP n'admet pas de ratio d'approx. $r \geq 1$
- $\Rightarrow$ TSP n'est pas dans **APX**

Preuve par raisonnement - Exemple

- une instance $G = (V, E)$ à n sommets de Cycle Hamiltonien
- on construit pour TSP
  - graphe K_n
  - poids: $w_e = 1$ si $e \in E$, $w_e = rn + 1$ sinon
- si G a un cycle hamiltonien $\mathcal{C} \rightarrow \mathcal{C}$ de poids n pour TSP
- sinon $\rightarrow$ tout cycle hamiltonien a un poids $> rn$ pour TSP

Preuve par raisonnement - Exemple

En conclusion:

- algo d'approx. de $\mathrm{TSP} \rightarrow$ observation de la solution *sol*
- si *sol* $>$ *rn* alors TSP optimal $> n \Rightarrow$ pas de cycle hamiltonien dans G
- si *sol* $\leq$ *rn* alors TSP optimal $= n \Rightarrow$ un cycle hamiltonien dans G
- $\Rightarrow$ algo d'approx. pour TSP rend Cycle Hamiltonien polynomial
- $\rightarrow$ contradiction

Preuve par réduction

Réduction préservant l'approximation

- Similaire à la réduction classique (pour démontrer qu'un problème est NP-complet)
- Contraintes supplémentaires:
  - réduction d'un PbO vers un PbO
  - le paramètre doit être "linéairement conservé"

Preuve par réduction

MAXIMUM INDEPENDENT SET (MAX-IS) (= Ensemble Stable)
Instance: Un graphe $G = (V, E)$
Solution: Un ensemble stable $V' \subseteq V$ de G
Mesure: $|V'|$

Theorem

Le problème MAX-IS *n'est pas approximable sous un ratio* $\mathcal{O}(n^{1-\varepsilon})$ *pour tout* $\varepsilon > 0$.

MAXIMUM CLIQUE (MAX-CLIQUE)
Instance: Un graphe $G = (V, E)$
Solution: Une clique (V', E') dans G
Mesure: $|V'|$

Example

Réduction préservant l'approximation de MAX-IS vers MAX-CLIQUE

Conclusion

- problèmes NP-complets
- comment les résoudre polynomialement ?
- petits cas = instances à un ou plusieurs paramètre/s constant/s
- classes d'instances = instances restreintes vérifiant une ou plusieurs propriété/s
- algos d'approximation (PbO seulement)
  - nouvelles classes de complexité
  - "positives" : **FPTAS**, **PTAS**, **APX**
  - "négatives" : **APX**-dur, pas dans **APX**
- bien d'autres techniques (heuristiques, ILP, prog. par contraintes, probabiliste, complexité paramétrée, etc.)