Complexité de Problèmes
Les Propriétés NP-dures et NP-complètes
AAC

S.Tison-Lille1
Master1 Informatique

RAPPEL DU DERNIER COURS

- Une propriété *NP* est une propriété "easy to check"

RAPPEL DU DERNIER COURS

- Une propriété NP est une propriété "easy to check"
  L est dit NP si il existe un polynôme Q, et un algorithme polynomial A à deux entrées et à valeurs booléennes tels que: $L = \{u / \exists c, A(c,u) = Vrai, |c| \leq Q(|u|)\}$

RAPPEL DU DERNIER COURS

- Une propriété *NP* est une propriété "easy to check"
- Définition Alternative:

RAPPEL DU DERNIER COURS

- Une propriété *NP* est une propriété "easy to check"
- Définition Alternative: Une propriété *NP* est une propriété décidable par un algorithme non déterministe polynomial.

RAPPEL DU DERNIER COURS

- Une propriété *NP* est une propriété "easy to check"
- Définition Alternative: Une propriété *NP* est une propriété décidable par un algorithme non déterministe polynomial.
- *P* est inclus dans *NP*

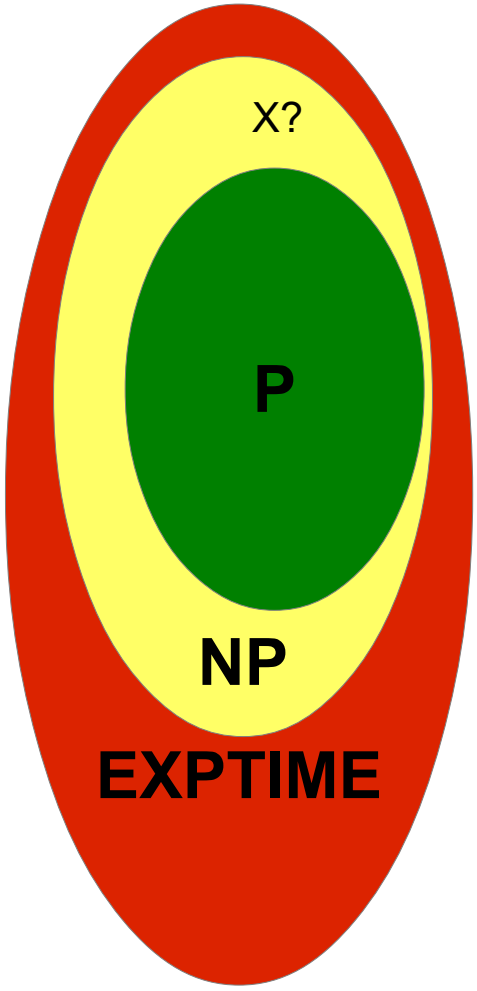
RAPPEL DU DERNIER COURS

- Une propriété *NP* est une propriété "easy to check"
- Définition Alternative: Une propriété *NP* est une propriété décidable par un algorithme non déterministe polynomial.
- *P* est inclus dans *NP*
- *NP* est inclus dans *ExpTime*

RAPPEL DU DERNIER COURS

- Une propriété NP est une propriété "easy to check"
- Définition Alternative: Une propriété NP est une propriété décidable par un algorithme non déterministe polynomial.
- P est inclus dans NP
- NP est inclus dans $ExpTime$
- $NP/=P$ conjecturé depuis 1971 mais non prouvé

La hiérarchie



Menu du jour

- Réductions Polynomiales

Menu du jour

- Réductions Polynomiales
  Comment transformer un problème en un autre.

MENU DU JOUR

- Réductions Polynomiales
  Comment transformer un problème en un autre.
- La notion de propriété $NP-$dure.

MENU DU JOUR

- Réductions Polynomiales
  Comment transformer un problème en un autre.
- La notion de propriété $NP-$dure.
  Qu'est-ce qu'une propriété qui contient toute la difficulté de la classe NP?

La problématique

Je cherche un algorithme pour vérifier une propriété; j'ai montré qu'elle est NP mais je ne trouve pas d'algorithme P.

La problématique

Je cherche un algorithme pour vérifier une propriété; j'ai montré qu'elle est NP mais je ne trouve pas d'algorithme P.

La propriété est-elle "dure" ou suis-je "nulle"? (le ou est non exclusif...)

La problématique

Je cherche un algorithme pour vérifier une propriété; j'ai montré qu'elle est NP mais je ne trouve pas d'algorithme P.

La propriété est-elle "dure" ou suis-je "nulle"? (le ou est non exclusif...)

Si j'arrive à prouver qu'il n' y a pas d'algorithme polynomial, je prouve $P \neq NP$: ça risque d'être difficile...

La problématique

Je cherche un algorithme pour vérifier une propriété; j'ai montré qu'elle est NP mais je ne trouve pas d'algorithme P.

La propriété est-elle "dure" ou suis-je "nulle"? (le ou est non exclusif...)

Si j'arrive à prouver qu'il n' y a pas d'algorithme polynomial, je prouve $P \neq NP$: ça risque d'être difficile...

Je peux peut-être montrer qu'elle est $NP-$dure.

La problématique

Une propriété $NP-$dure contient d'une certaine façon toute la difficulté de la classe NP.

La problématique

Une propriété $NP-$dure contient d'une certaine façon toute la difficulté de la classe NP.

Si on trouve un algorithme polynomial pour une telle propriété, on en aurait un pour toute propriété NP.

La problématique

Une propriété $NP-$dure contient d'une certaine façon toute la difficulté de la classe NP.

Si on trouve un algorithme polynomial pour une telle propriété, on en aurait un pour toute propriété NP.

Montrer qu'une propriété est $NP-$dure, justifie en quelque sorte de ne pas avoir trouvé d'algorithme polynomial.

Un outil: Les réductions polynomiales

La notion de réduction permet de traduire qu'un problème n'est pas "plus dur" qu'un autre.

UN OUTIL: LES RÉDUCTIONS POLYNOMIALES

La notion de réduction permet de traduire qu'un problème n'est pas "plus dur" qu'un autre.

Si un problème A se réduit en un problème B, le problème A est (au moins) aussi facile que B - si la réduction est "facile"-. On peut a priori utiliser la réduction de deux façons:

UN OUTIL: LES RÉDUCTIONS POLYNOMIALES

La notion de réduction permet de traduire qu'un problème n'est pas "plus dur" qu'un autre.

Si un problème A se réduit en un problème B, le problème A est (au moins) aussi facile que B - si la réduction est "facile"-. On peut a priori utiliser la réduction de deux façons:

- pour montrer qu'un problème est dur: si A est réputé dur, B l'est aussi;

UN OUTIL: LES RÉDUCTIONS POLYNOMIALES

La notion de réduction permet de traduire qu'un problème n'est pas "plus dur" qu'un autre.

Si un problème A se réduit en un problème B, le problème A est (au moins) aussi facile que B - si la réduction est "facile"-. On peut a priori utiliser la réduction de deux façons:

- pour montrer qu'un problème est dur: si A est réputé dur, B l'est aussi;
- pour montrer qu'un problème est facile: si B est facile, A l'est aussi.

C'est en fait surtout le premier raisonnement que l'on va utiliser.

UN EXEMPLE "INFORMEL"

Soient les deux problèmes de décision suivants:

Un exemple "informel"

Soient les deux problèmes de décision suivants:

Problème 1

Donnée: N, un nombre de participants et une liste de paires de participants: les paires d'ennemis
Q? Peut-on faire p équipes de telle sorte qu'aucune équipe ne contienne une paire d'ennemis.

Un exemple "informel"

Soient les deux problèmes de décision suivants:

Problème 1

Donnée: N, un nombre de participants et une liste de paires de participants: les paires d'ennemis
Q? Peut-on faire p équipes de telle sorte qu'aucune équipe ne contienne une paire d'ennemis.

Problème 2

Donnée: Un graphe G, un entier k: un nombre de couleurs.
Q? G est -il k-coloriable?

Un exemple "informel"

Soient les deux problèmes de décision suivants:

Problème 1

Donnée: N, un nombre de participants et une liste de paires de participants: les paires d'ennemis
Q? Peut-on faire p équipes de telle sorte qu'aucune équipe ne contienne une paire d'ennemis.

Problème 2

Donnée: Un graphe G, un entier k: un nombre de couleurs.
Q? G est -il k-coloriable?

Comment transformer un problème en un autre?

UN EXEMPLE: LA TRANSFORMATION...

Supposons qu'on dispose d'un algorithme pour le pb de coloriage de graphes.
Comment s'en servir pour le pb des équipes?
On transforme une instance du Pb1 en une instance du Pb2:

UN EXEMPLE: LA TRANSFORMATION...

Supposons qu'on dispose d'un algorithme pour le pb de coloriage de graphes.
Comment s'en servir pour le pb des équipes?
On transforme une instance du Pb1 en une instance du Pb2:

- Les sommets du graphe sont les personnes.

UN EXEMPLE: LA TRANSFORMATION...

Supposons qu'on dispose d'un algorithme pour le pb de coloriage de graphes.
Comment s'en servir pour le pb des équipes?
On transforme une instance du Pb1 en une instance du Pb2:

- Les sommets du graphe sont les personnes.
- Il y a un arc entre deux sommets si les personnes sont ennemies.

UN EXEMPLE: LA TRANSFORMATION...

Supposons qu'on dispose d'un algorithme pour le pb de coloriage de graphes.
Comment s'en servir pour le pb des équipes?
On transforme une instance du Pb1 en une instance du Pb2:

- Les sommets du graphe sont les personnes.
- Il y a un arc entre deux sommets si les personnes sont ennemies.
- $k = p$

UN EXEMPLE: LA TRANSFORMATION...

Supposons qu'on dispose d'un algorithme pour le pb de coloriage de graphes.
Comment s'en servir pour le pb des équipes?
On transforme une instance du Pb1 en une instance du Pb2:

- Les sommets du graphe sont les personnes.
- Il y a un arc entre deux sommets si les personnes sont ennemies.
- $k = p$

Alors, G est k-coloriable Ssi on peut faire p équipes!

...EST POLYNOMIALE

De plus, la construction de G se fait polynomialement.
Donc si on a un algorithme polynomial pour le $p-$coloriage de graphe, on en a aussi un pour le problème d'équipes:

...EST POLYNOMIALE

De plus, la construction de G se fait polynomialement.
Donc si on a un algorithme polynomial pour le $p-$coloriage de graphe, on en a aussi un pour le problème d'équipes:

Soit:
`boolean Sol2(Pb2 inst)` qui retourne en temps polynomial `Vrai` Ssi `inst` est une instance positive de `Pb2`, alors:

...EST POLYNOMIALE

De plus, la construction de G se fait polynomialement.
Donc si on a un algorithme polynomial pour le $p-$coloriage de graphe, on en a aussi un pour le problème d'équipes:

Soit:
`boolean Sol2(Pb2 inst)` qui retourne en temps polynomial `Vrai` Ssi `inst` est une instance positive de `Pb2`, alors:

```
boolean Sol1(Pb1 inst) {
    return Sol2(red_1_to_2(inst))}
```

retourne en temps polynomial `Vrai` Ssi `inst` est une instance positive de `Pb1`.

...EST POLYNOMIALE

Si on a un algorithme polynomial pour le $p-$coloriage de graphe, on en a aussi un pour le problème d'équipes.

...EST POLYNOMIALE

Si on a un algorithme polynomial pour le $p-$coloriage de graphe, on en a aussi un pour le problème d'équipes.

Par contraposée:

...EST POLYNOMIALE

Si on a un algorithme polynomial pour le $p-$coloriage de graphe, on en a aussi un pour le problème d'équipes.

Par contraposée:

Si il n'y a pas d'algorithme polynomial pour le problème d'équipes, il n'y en a pas non plus pour le $p-$coloriage.

On peut aussi faire une réduction dans l'autre sens...

On peut aussi faire une réduction dans l'autre sens...

- Les personne sont les sommets de G.

On peut aussi faire une réduction dans l'autre sens...

- Les personne sont les sommets de G.
- Deux personnes sont "ennemies" si elles sont eliées par un arc de G.

On peut aussi faire une réduction dans l'autre sens...

- Les personne sont les sommets de G.
- Deux personnes sont "ennemies" si elles sont eliées par un arc de G.

Alors on peut faire p équipes Ssi le graphe de départ était p-coloriable.
Là encore, la réduction est polynomiale.

On peut aussi faire une réduction dans l'autre sens...

- Les personne sont les sommets de G.
- Deux personnes sont "ennemies" si elles sont eliées par un arc de G.

Alors on peut faire p équipes Ssi le graphe de départ était p-coloriable.
Là encore, la réduction est polynomiale.

Donc si il y a un algo polynomial pour le problème d'équipes, il y en a un pour le $p-$coloriage de graphes.

On peut aussi faire une réduction dans l'autre sens...

- Les personne sont les sommets de G.
- Deux personnes sont "ennemies" si elles sont eliées par un arc de G.

Alors on peut faire p équipes Ssi le graphe de départ était p-coloriable.
Là encore, la réduction est polynomiale.

Donc si il y a un algo polynomial pour le problème d'équipes, il y en a un pour le $p-$coloriage de graphes.

Si il n'y en a pas pour le problème de $p-$coloriage, il n'y en a pas non plus pour le problème d'équipes.

Définition formelle

Définition

Soient L et L' deux langages de Σ^*, correspondant à deux propriétés.

Une **réduction polynomiale** *de L dans L' est une application red calculable polynomiale de Σ^* dans Σ^* telle que :*

$$u \in L \text{ Ssi } red(u) \in L'.$$

Notation: $L \leq_P L'$.

Réduction polynomiale = transformation d'un problème dans un autre.

Par exemple, une réduction d'un problème pb1 d'équipes en un problème pb2 de coloriage de graphes sera une transformation:

- qui associe à toute instance i du pb1 d'équipes, une instance $red(i)$ du pb2 de coloriage de graphes

Réduction polynomiale = transformation d'un problème dans un autre.

Par exemple, une réduction d'un problème pb1 d'équipes en un problème pb2 de coloriage de graphes sera une transformation:

- qui associe à toute instance i du pb1 d'équipes, une instance $red(i)$ du pb2 de coloriage de graphes
- telle que $red(i)$ a une solution Ssi l'instance i avait une solution.

Réduction polynomiale = transformation d'un problème dans un autre.

Par exemple, une réduction d'un problème pb1 d'équipes en un problème pb2 de coloriage de graphes sera une transformation:

- qui associe à toute instance i du pb1 d'équipes, une instance $red(i)$ du pb2 de coloriage de graphes
- telle que $red(i)$ a une solution Ssi l'instance i avait une solution.
- cette transformation est polynomiale, i.e. calculable par un algorithme polynomial.

Réduction polynomiale = transformation d'un problème dans un autre.

Par exemple, une réduction d'un problème pb1 d'équipes en un problème pb2 de coloriage de graphes sera une transformation:

- qui associe à toute instance i du pb1 d'équipes, une instance $red(i)$ du pb2 de coloriage de graphes
- telle que $red(i)$ a une solution Ssi l'instance i avait une solution.
- cette transformation est polynomiale, i.e. calculable par un algorithme polynomial.

Remarque: on en déduit donc que la taille de $red(i)$ est bornée polynomialement par rapport à celle de i.

ENCORE UN EXEMPLE

Soient les deux propriétés suivantes:

```
Clique:
Entrée:
G=(S,A) --un graphe non orienté
k       -- un entier
Sortie:
 Oui, Ssi G contient une clique de cardinal k.

Independant
Entrée:
G=(S,A) --un graphe non orienté
k       -- un entier
Sortie:
Oui, Ssi G contient un ensemble indépendant de card
```

EXEMPLE: SUITE...

On peut choisir comme réduction de `Clique` dans `Independant` l'application *red* qui à $(G = (S, A), k)$ associe $(G' = (S, A'), k)$ avec $A' = \{(x, y)/(x, y) \notin A \text{ et } (x, y) \notin A\}$.

EXEMPLE: SUITE...

On peut choisir comme réduction de `Clique` dans `Independant` l'application *red* qui à $(G = (S, A), k)$ associe $(G' = (S, A'), k)$ avec $A' = \{(x, y)/(x, y) \notin A \; et \; (x, y) \notin A\}$.

Montrer que c'est bien une réduction polynomiale de `Clique` dans `Independant` consiste en:

EXEMPLE: SUITE...

On peut choisir comme réduction de `Clique` dans `Independant` l'application *red* qui à $(G = (S, A), k)$ associe $(G' = (S, A'), k)$ avec $A' = \{(x, y)/(x, y) \notin A \text{ et } (x, y) \notin A\}$.

Montrer que c'est bien une réduction polynomiale de `Clique` dans `Independant` consiste en:

- Montrer que c'est correct i.e., pour tout (G, k), $Clique(G, k)$ Ssi $Independant(G', k)$.

EXEMPLE: SUITE...

On peut choisir comme réduction de `Clique` dans `Independant` l'application *red* qui à $(G = (S, A), k)$ associe $(G' = (S, A'), k)$ avec $A' = \{(x, y)/(x, y) \notin A \text{ et } (x, y) \notin A\}$.

Montrer que c'est bien une réduction polynomiale de `Clique` dans `Independant` consiste en:

- Montrer que c'est correct i.e., pour tout (G, k), $Clique(G, k)$ Ssi $Independant(G', k)$.
- Montrer que la transformation est polynomiale.

EXEMPLE: SUITE...

red transforme l'instance de `Clique` $(G = (S, A), k)$, en l'instance d'`Independent` $(G' = (S, A'), k)$ avec $A' = \{(x, y)/(x, y) \notin A \text{ et } (x, y) \notin A\}$.

- Correct? i.e., pour tout (G, k), $Clique(G, k)$ Ssi $Independent(G', k)$? Oui:
  Si $(G = (S, A), k)$ a une clique C de cardinal k, C est un ensemble indépendant de cardinal k de G'.
  réciproquement: si $(G' = (S, A'), k)$ a un ensemble indépendant C de cardinal k, C est une clique de cardinal k de G.

EXEMPLE: SUITE...

red transforme l'instance de `Clique` $(G = (S, A), k)$, en l'instance d'`Independent` $(G' = (S, A'), k)$ avec $A' = \{(x, y)/(x, y) \notin A \text{ et } (x, y) \notin A\}$.

- Correct? i.e., pour tout (G, k), $Clique(G, k)$ Ssi $Independent(G', k)$? Oui:
  Si $(G = (S, A), k)$ a une clique C de cardinal k, C est un ensemble indépendant de cardinal k de G'.
  réciproquement: si $(G' = (S, A'), k)$ a un ensemble indépendant C de cardinal k, C est une clique de cardinal k de G.
- La transformation est polynomiale: l'algorithme qui calcule $red(G)$ est bien polynomial (en $O(card(S)^2)$.

En résumé

Pour prouver qu'une propriété $P1$ se réduit polynomialement en une autre $P2$, il faut:

En résumé

Pour prouver qu'une propriété $P1$ se réduit polynomialement en une autre $P2$, il faut:

- proposer une réduction, i.e. un algo *red* qui à une instance I de $P1$, associe une instance $red(I)$ de $P2$.

En résumé

Pour prouver qu'une propriété $P1$ se réduit polynomialement en une autre $P2$, il faut:

- proposer une réduction, i.e. un algo *red* qui à une instance I de $P1$, associe une instance $red(I)$ de $P2$.
- prouver qu'elle est correcte, i.e. qu'une instance I de P1 est positive **Si et Seulement Si** $red(I)$ est une instance positive de $P2$.

En résumé

Pour prouver qu'une propriété $P1$ se réduit polynomialement en une autre $P2$, il faut:

- proposer une réduction, i.e. un algo *red* qui à une instance I de $P1$, associe une instance $red(I)$ de $P2$.
- prouver qu'elle est correcte, i.e. qu'une instance I de P1 est positive **Si et Seulement Si** $red(I)$ est une instance positive de $P2$.
- prouver qu'elle est polynomiale.

Pourquoi prouver qu'un problème se réduit en un autre?

Proposition: Si L' est dans P et si $L \leq_P L'$, alors L est dans P.

POURQUOI PROUVER QU'UN PROBLÈME SE RÉDUIT EN UN AUTRE?

Proposition: Si L' est dans P et si $L \leq_P L'$, alors L est dans P.

$u \in L$ Ssi $red(u) \in L'$: donc, pour décider de l'appartenance à L, il suffit d'appliquer la transformation, et de décider de l'appartenance du transformé à L':

POURQUOI PROUVER QU'UN PROBLÈME SE RÉDUIT EN UN AUTRE?

Proposition: Si L' est dans P et si $L \leq_P L'$, alors L est dans P.

$u \in L$ Ssi $red(u) \in L'$: donc, pour décider de l'appartenance à L, il suffit d'appliquer la transformation, et de décider de l'appartenance du transformé à L':

Si $AppL'$ est un algo polynomial pour tester l'appartenance à L',

`boolean AppL(u) {return AppL'(red(u));}` est un algo polynomial pour tester l'appartenance à L'.

POURQUOI PROUVER QU'UN PROBLÈME SE RÉDUIT EN UN AUTRE?

Proposition: Si L' est dans P et si $L \leq_P L'$, alors L est dans P.

$u \in L$ Ssi $red(u) \in L'$: donc, pour décider de l'appartenance à L, il suffit d'appliquer la transformation, et de décider de l'appartenance du transformé à L':

Si $AppL'$ est un algo polynomial pour tester l'appartenance à L',

`boolean AppL(u) {return AppL'(red(u));}` est un algo polynomial pour tester l'appartenance à L'.

Si L' est P, L est P.

si L n'est pas P, L' n'est pas P.

RÉDUCTIONS EN CHAÎNE:

Propriété: La relation $\leq_P$ est transitive: si $L \leq_P L'$ et $L' \leq_P L''$, alors $L \leq_P L$".

Si $red1$ est une réduction polynomiale de L dans L', et $red2$ est une réduction polynomiale de L' vers L" , alors $red2 \circ red1$ est une réduction polynomiale de L vers L".

Les propriétés NP-dures, NP-complètes

Définition

Une propriété R est dite **NP-dure** *Si toute propriété NP se réduit polynomialement en R.*

Les propriétés NP-dures, NP-complètes

Définition

Une propriété R est dite **NP-dure** *Si toute propriété NP se réduit polynomialement en R.*

Donc, d'après ce qui précède, si une propriété $NP - dure\ R$ était P, on aurait $NP = P$: R contient donc d'une certaine façon toute la difficulté de NP!

Les propriétés NP-dures, NP-complètes

Définition

Une propriété R est dite **NP-dure** *Si toute propriété NP se réduit polynomialement en R.*

Donc, d'après ce qui précède, si une propriété $NP - dure\ R$ était P, on aurait $NP = P$: R contient donc d'une certaine façon toute la difficulté de NP!

Définition

*Une propriété est dite $NP-$***complète** *Si elle est NP et $NP-dure$.*

La découverte de Cook

Cook fut le premier à découvrir... l'existence de propriétés NP-dures:
Théorème de Cook: $3-CNF-SAT$ est $NP-$complète.

Rappel: $3-CNF-SAT$ est le problème de la satisfiabilité d'une formule sous forme conjonctive où chaque clause contient 3 littéraux.

$(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg X_4) \wedge (\neg x_1 \vee \neg x_2 \neg x_3)$

Comment montrer qu'une propriété est NP-dure?

Pour montrer que R est NP-dure:

Comment montrer qu'une propriété est NP-dure?

Pour montrer que R est NP-dure:

- On peut montrer "directement" que toute propriété NP se réduit polynomialement en R.

Comment montrer qu'une propriété est NP-dure?

Pour montrer que R est NP-dure:

- On peut montrer "directement" que toute propriété NP se réduit polynomialement en R.
- mais il suffit de montrer qu'une propriété connue $NP-$dure se réduit polynomialement en R.
  En effet, comme $\leq_P$ est transitif, on a:

Comment montrer qu'une propriété est NP-dure?

Pour montrer que R est NP-dure:

- On peut montrer "directement" que toute propriété NP se réduit polynomialement en R.
- mais il suffit de montrer qu'une propriété connue $NP-$dure se réduit polynomialement en R.
  En effet, comme $\leq_P$ est transitif, on a:

  Proposition: si $L \leq_P L'$ et L est $NP-$dure, alors L' est $NP-$dure.

POURQUOI MONTRER QU'UNE PROPRIÉTÉ EST $NP-$DURE?

D'après la proposition précédente, si une propriété $NP-$dure se révélait être P, on aurait $P = NP$, ce qui est peu probable et en tout cas a résisté aux efforts de nombreux chercheurs!

Donc, montrer qu'une propriété est $NP-$dure justifie raisonnablement qu'on n'ait pas trouvé d'algorithme polynomial pour décider de cette propriété!

NP VERSUS $NP-$DURE

Par contre, montrer qu'une propriété est NP, c'est montrer qu'elle n'est pas "si dure" que ça même si elle n'est peut-être pas P...

EXEMPLE: CLIQUE EST *NP*−DUR

```
Clique:
Entrée:
G=(S,A) --un graphe non orienté
k       -- un entier

Sortie:
Oui, Ssi G contient une clique de cardinal k.
```

EXEMPLE: CLIQUE EST $NP-$DUR?

EXEMPLE: CLIQUE EST $NP-$DUR?

$3-CNF-SAT$ est $NP-$dur.

EXEMPLE: CLIQUE EST $NP-$DUR?

$3-CNF-SAT$ est $NP-$dur.

On va réduire polynomialement $3-CNF-Sat$ dans $Clique$:

EXEMPLE: CLIQUE EST $NP-$DUR?

$3-CNF-SAT$ est $NP-$dur.

On va réduire polynomialement $3-CNF-Sat$ dans $Clique$:

3-CNF-SAT:
Donnée: $\Phi = C_1 \wedge C_2 \ldots; \wedge C_p$ avec $C_j = l_j^1 \vee l_j^2 \vee l_j^3$

EXEMPLE: CLIQUE EST $NP-$DUR?

$3-CNF-SAT$ est $NP-$dur.

On va réduire polynomialement $3-CNF-Sat$ dans $Clique$:

3-CNF-SAT:
Donnée: $\Phi = C_1 \wedge C_2 ....; \wedge C_p$ avec $C_j = l_j^1 \vee l_j^2 \vee l_j^3$

Sortie: Oui, si il existe une valuation v telle que $v(\Phi) = Vrai$.

3 – *CNF – SAT* se réduit polynomialement en *Clique*: la réduction

3 – *CNF – SAT* se réduit polynomialement en *Clique*: la réduction

A une formule sous forme 3-CNF Φ, on va associer un graphe G_Φ et un entier p tel que la formule Φ est satisfiable Ssi le graphe G_Φ a une clique de cardinal p.

3 – *CNF – SAT* se réduit polynomialement en *Clique*: la réduction

A une formule sous forme 3-CNF Φ, on va associer un graphe G_Φ et un entier p tel que la formule Φ est satisfiable Ssi le graphe G_Φ a une clique de cardinal p.

Construction du graphe:

3 – *CNF – SAT* se réduit polynomialement en *Clique*: la réduction

A une formule sous forme 3-CNF Φ, on va associer un graphe G_Φ et un entier p tel que la formule Φ est satisfiable Ssi le graphe G_Φ a une clique de cardinal p.

Construction du graphe:

- les sommets = les littéraux

3 – *CNF – SAT* se réduit polynomialement en *Clique*: la réduction

A une formule sous forme 3-CNF Φ, on va associer un graphe G_Φ et un entier p tel que la formule Φ est satisfiable Ssi le graphe G_Φ a une clique de cardinal p.

Construction du graphe:

- les sommets = les littéraux
- les arcs: un arc entre deux littéraux de clauses différentes et compatibles, i.e. pas d'arc si ils sont dans la même clause ou si l'un est de la forme x et l'autre $\neg x$.

3 – *CNF – SAT* se réduit polynomialement en *Clique*: la réduction

A une formule sous forme 3-CNF Φ, on va associer un graphe G_Φ et un entier p tel que la formule Φ est satisfiable Ssi le graphe G_Φ a une clique de cardinal p.

Construction du graphe:

- les sommets = les littéraux
- les arcs: un arc entre deux littéraux de clauses différentes et compatibles, i.e. pas d'arc si ils sont dans la même clause ou si l'un est de la forme x et l'autre $\neg x$.
- p: le nombre de clauses de Φ

La réduction est correcte?

G_Φ a un clique d'ordre p Ssi Φ est satisfiable?
Preuve:
. Φ satisfiable: $\exists v$ telle que $v(\Phi) = Vrai$.

La réduction est correcte?

G_Φ a un clique d'ordre p Ssi Φ est satisfiable?
Preuve:
. Φ satisfiable: $\exists v$ telle que $v(\Phi) = Vrai$.

Donc, pour chaque j il existe $l_j^{i_j}$ tel que $v(l_j^{i_j}) = Vrai$.

La réduction est correcte?

G_Φ a un clique d'ordre p Ssi Φ est satisfiable?
Preuve:
. Φ satisfiable: $\exists v$ telle que $v(\Phi) = Vrai$.

Donc, pour chaque j il existe $l_j^{i_j}$ tel que $v(l_j^{i_j}) = Vrai$.

Alors les $l_j^{i_j}$ forment une clique de G_Φ par définition des arcs !

La réduction est correcte?

G_Φ a un clique d'ordre p Ssi Φ est satisfiable?
Preuve:
. Φ satisfiable: $\exists v$ telle que $v(\Phi) = Vrai$.

Donc, pour chaque j il existe $l_j^{i_j}$ tel que $v(l_j^{i_j}) = Vrai$.

Alors les $l_j^{i_j}$ forment une clique de G_Φ par définition des arcs !

.Dans l'autre sens: G_Φ a une clique:

La réduction est correcte?

G_Φ a un clique d'ordre p Ssi Φ est satisfiable?
Preuve:
. Φ satisfiable: $\exists v$ telle que $v(\Phi) = Vrai$.

Donc, pour chaque j il existe $l_j^{i_j}$ tel que $v(l_j^{i_j}) = Vrai$.

Alors les $l_j^{i_j}$ forment une clique de G_Φ par définition des arcs !

.Dans l'autre sens: G_Φ a une clique:

Posons $v(x) = vrai$ si il y a un x dans la clique, $v(x) = faux$ si il y a un $\neg x$ dans la clique, $v(x) = Vrai$ (ou $Faux$)si on n'a ni l'un ni l'autre.

La réduction est correcte?

G_Φ a un clique d'ordre p Ssi Φ est satisfiable?
Preuve:
. Φ satisfiable: $\exists v$ telle que $v(\Phi) = Vrai$.

Donc, pour chaque j il existe $l_j^{i_j}$ tel que $v(l_j^{i_j}) = Vrai$.

Alors les $l_j^{i_j}$ forment une clique de G_Φ par définition des arcs !

.Dans l'autre sens: G_Φ a une clique:

Posons $v(x) = vrai$ si il y a un x dans la clique, $v(x) = faux$ si il y a un$\neg x$ dans la clique, $v(x) = Vrai$ (ou $Faux$)si on n'a ni l'un ni l'autre.

v est bien définie: on ne peut avoir à la fois x et $\neg x$ dans la clique.

La réduction est correcte?

G_Φ a un clique d'ordre p Ssi Φ est satisfiable?
Preuve:
. Φ satisfiable: $\exists v$ telle que $v(\Phi) = Vrai$.

Donc, pour chaque j il existe $l_j^{i_j}$ tel que $v(l_j^{i_j}) = Vrai$.

Alors les $l_j^{i_j}$ forment une clique de G_Φ par définition des arcs !

.Dans l'autre sens: G_Φ a une clique:

Posons $v(x) = vrai$ si il y a un x dans la clique, $v(x) = faux$ si il y a un$\neg x$ dans la clique, $v(x) = Vrai$ (ou $Faux$)si on n'a ni l'un ni l'autre.

v est bien définie: on ne peut avoir à la fois x et $\neg x$ dans la clique.

Par construction, un sommet par clause: v valide au moins un littéral par lause; $v(\Phi) = Vrai$: Φ est bien satisfiable

LA RÉDUCTION EST BIEN POLYNOMIALE?

- taille de la formule: 3*p

LA RÉDUCTION EST BIEN POLYNOMIALE?

- taille de la formule: 3*p
- taille du graphe: 3*p sommets, au plus de l'ordre de $(3p)^2$ arcs:

LA RÉDUCTION EST BIEN POLYNOMIALE?

- taille de la formule: 3*p
- taille du graphe: 3*p sommets, au plus de l'ordre de $(3p)^2$ arcs:
- la construction du graphe est bien polynomiale.

$3 - CNF - SAT \leq_P Clique$: BILAN

- On a bien réduit polynomialement $3 - CNF - SAT$ dans *Clique*: $3 - CNF - SAT \leq_P Clique$

$3-CNF-SAT \leq_P Clique$: BILAN

- On a bien réduit polynomialement $3-CNF-SAT$ dans $Clique$: $3-CNF-SAT \leq_P Clique$
- $3-CNF-SAT$ est $NP-$dur.

$3-CNF-SAT \leq_P Clique$: BILAN

- On a bien réduit polynomialement $3-CNF-SAT$ dans $Clique$: $3-CNF-SAT \leq_P Clique$
- $3-CNF-SAT$ est $NP-$dur.
- donc $Clique$ est $NP-$ dur

$3-CNF-SAT \leq_P Clique$: BILAN

- On a bien réduit polynomialement $3-CNF-SAT$ dans $Clique$: $3-CNF-SAT \leq_P Clique$
- $3-CNF-SAT$ est $NP-$dur.
- donc $Clique$ est $NP-$ dur
- $Clique$ est même $NP-$ complète car $NP + NP-$dure.

$3-CNF-SAT \leq_P Clique$: BILAN

- On a bien réduit polynomialement $3-CNF-SAT$ dans $Clique$: $3-CNF-SAT \leq_P Clique$
- $3-CNF-SAT$ est $NP-$dur.
- donc $Clique$ est $NP-$ dur
- $Clique$ est même $NP-$ complète car NP + $NP-$dure.

Remarque: On avait prouvé que $Clique$ se réduit en $Independent$, donc

$3-CNF-SAT \leq_P Clique$: BILAN

- On a bien réduit polynomialement $3-CNF-SAT$ dans $Clique$: $3-CNF-SAT \leq_P Clique$
- $3-CNF-SAT$ est $NP-$dur.
- donc $Clique$ est $NP-$ dur
- $Clique$ est même $NP-$ complète car NP + $NP-$dure.

Remarque: On avait prouvé que $Clique$ se réduit en $Independent$, donc $Independent$ aussi est $NP-$dur!

$3-CNF-SAT \leq_P Clique$: BILAN

- On a bien réduit polynomialement $3-CNF-SAT$ dans $Clique$: $3-CNF-SAT \leq_P Clique$
- $3-CNF-SAT$ est $NP-$dur.
- donc $Clique$ est $NP-$ dur
- $Clique$ est même $NP-$ complète car NP + $NP-$dure.

Remarque: On avait prouvé que $Clique$ se réduit en $Independent$, donc $Independent$ aussi est $NP-$dur!

Question? A-t-on $Clique$ qui se réduit polynomialement dans $3-CNF-SAT$?

$3-CNF-SAT \leq_P Clique$: BILAN

- On a bien réduit polynomialement $3-CNF-SAT$ dans $Clique$: $3-CNF-SAT \leq_P Clique$
- $3-CNF-SAT$ est $NP-$dur.
- donc $Clique$ est $NP-$ dur
- $Clique$ est même $NP-$ complète car NP + $NP-$dure.

Remarque: On avait prouvé que $Clique$ se réduit en $Independent$, donc $Independent$ aussi est $NP-$dur!

Question? A-t-on $Clique$ qui se réduit polynomialement dans $3-CNF-SAT$?

Rép: oui, $Clique$ est NP, $3-CNF-SAT$ est $NP-$dure!

Problèmes NP-durs / Propriétés

Ce qui précède ne s'applique qu'aux propriétés.

Problèmes NP-durs / Propriétés

Ce qui précède ne s'applique qu'aux propriétés.

Parler de problèmes NP, si ce ne sont pas des problèmes de décision, n'a guère de sens;

PROBLÈMES NP-DURS / PROPRIÉTÉS

Ce qui précède ne s'applique qu'aux propriétés.

Parler de problèmes NP, si ce ne sont pas des problèmes de décision, n'a guère de sens;

Par contre on peut parler de problèmes $NP-$durs:

Problèmes NP-durs / Propriétés

Ce qui précède ne s'applique qu'aux propriétés.

Parler de problèmes NP, si ce ne sont pas des problèmes de décision, n'a guère de sens;

Par contre on peut parler de problèmes $NP-$durs:

Définition

Si l'existence d'un algorithme polynomial pour le problème R impliquerait $P = NP$, on dit que R est $NP-$dur.

Problèmes NP-durs et optimisation

A tout pb d'optimisation de type "trouver une solution de coût minimal (resp. de gain maximal), on peut associer une propriété:

PROBLÈMES NP-DURS ET OPTIMISATION

A tout pb d'optimisation de type "trouver une solution de coût minimal (resp. de gain maximal), on peut associer une propriété:

" existe-t-il une solution de coût inférieur (resp. de gain supérieur) à une valeur donnée en entrée";

PROBLÈMES NP-DURS ET OPTIMISATION

A tout pb d'optimisation de type "trouver une solution de coût minimal (resp. de gain maximal), on peut associer une propriété:

" existe-t-il une solution de coût inférieur (resp. de gain supérieur) à une valeur donnée en entrée";

Si le pb de décision est $NP-$dur, le pb d'optimisation le sera aussi.

PROBLÈMES NP-DURS ET OPTIMISATION

A tout pb d'optimisation de type "trouver une solution de coût minimal (resp. de gain maximal), on peut associer une propriété:

" existe-t-il une solution de coût inférieur (resp. de gain supérieur) à une valeur donnée en entrée";

Si le pb de décision est $NP-$dur, le pb d'optimisation le sera aussi.

Si on avait un algorithme P pour le pb d'optimisation, on en aurait un pour celui de décision donc pour toute propriété NP, puisqu'il est $NP-$dur!

Problèmes NP-durs: Un exemple

Soit le problème du coloriage de graphes: pour un graphe, trouver un coloriage correct utilisant le moins de couleurs possibles.

Problèmes NP-durs: Un exemple

Soit le problème du coloriage de graphes: pour un graphe, trouver un coloriage correct utilisant le moins de couleurs possibles.

On peut lui associer la propriété du $k - coloriage$: étant donné un graphe et un entier k, existe-t-il un coloriage correct de G avec k couleurs?

Problèmes NP-durs: Un exemple

Soit le problème du coloriage de graphes: pour un graphe, trouver un coloriage correct utilisant le moins de couleurs possibles.

On peut lui associer la propriété du $k - coloriage$: étant donné un graphe et un entier k, existe-t-il un coloriage correct de G avec k couleurs?

Cette propriété étant $NP-$dure, on peut dire que le problème de coloriage de graphe est $NP-$dur.

Problèmes NP-durs: Un exemple

Soit le problème du coloriage de graphes: pour un graphe, trouver un coloriage correct utilisant le moins de couleurs possibles.

On peut lui associer la propriété du $k - coloriage$: étant donné un graphe et un entier k, existe-t-il un coloriage correct de G avec k couleurs?

Cette propriété étant $NP-$dure, on peut dire que le problème de coloriage de graphe est $NP-$dur.

Si il existe un algorithme polynomial pour colorier un graphe avec le moins de couleurs possible, on aurait un algorithme polynomial pour la propriété du $k - coloriage$ et donc on aurait $P = NP$

LE CATALOGUE "LES PROPRIÉTÉS $NP-$COMPLÈTES"

Pour prouver qu'une propriété R est $NP-$dure, on peut choisir une propriété connue NP-dure et essayer de la réduire dans R: le problème est de bien la choisir ....

LE CATALOGUE "LES PROPRIÉTÉS $NP-$COMPLÈTES"

Pour prouver qu'une propriété R est $NP-$dure, on peut choisir une propriété connue NP-dure et essayer de la réduire dans R: le problème est de bien la choisir ....

Il existe un catalogue des propriétés $NP-$dures, le livre de Garey et Johnson:" Computers and Intractability: A guide to the theory of NP-completeness " qui contient beaucoup de problèmes mais date de 1979;

LE CATALOGUE "LES PROPRIÉTÉS $NP-$COMPLÈTES"

Pour prouver qu'une propriété R est $NP-$dure, on peut choisir une propriété connue NP-dure et essayer de la réduire dans R: le problème est de bien la choisir ....

Il existe un catalogue des propriétés $NP-$dures, le livre de Garey et Johnson:" Computers and Intractability: A guide to the theory of NP-completeness " qui contient beaucoup de problèmes mais date de 1979;

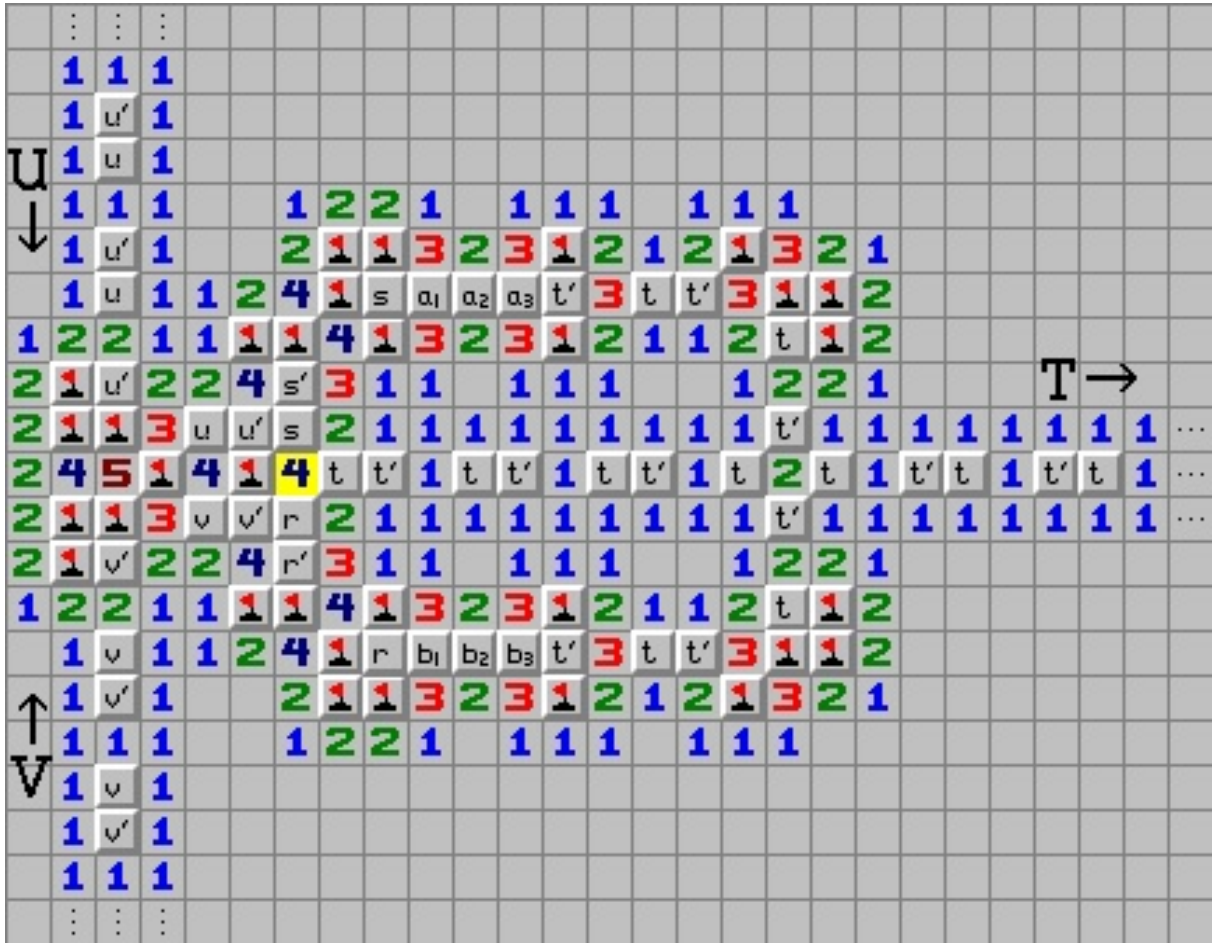
Il existe des sites Web qui essaient de tenir à jour le catalogue comme le compendium de problèmes d'optimisations NP: http://www.nada.kth.se/ viggo/problemlist/compendium.html.

UN EXTRAIT DU CATALOGUE DE $NP-$COMPLÈTES:

- 3-CNF-SAT:
- 3-coloriage de graphes:
- existence d'une clique de taille k dans un graphe.
- le pb du BinPacking
- Le pb du sac à dos (non fractionnable).
- le pb de l'existence d'un circuit hamiltonien.
- Le pb du voyageur de commerce.
- la programmation linéaire en entiers.
- Le pb du recouvrement d'ensembles.
- Le pb du démineur ...

Le démineur

Minesweeper is NP-complete! by Richard Kaye 2000



Classic Nintendo Games are (NP-)Hard

by Greg Aloupis, Erik D. Demaine, Alan Guo, March 9, 2012

We prove NP-hardness results for five of Nintendo's largest video game franchises: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokemon. Our results apply to Super Mario Bros. 1, 3, Lost Levels, and Super Mario World; Donkey Kong Country 13; all Legend of Zelda games except Zelda II: The Adventure of Link; all Metroid games; and all Pokemon role-playing games.

Perl Regular Expression Matching is NP-Hard

by M-J. Dominus

Perl takes exponential time (in the length of the string to be matched) to check whether or not a string matches certain regex.
One is tempted to wonder whether this difficulty is inherent, or whether there might be a clever way of speeding up the matching algorithm. The answer is that there is probably no clever way to speed up the algorithm, and that the exponential running time of the matching algorithm is probably unavoidable. We show that regex matching is NP-hard when regexes are allowed to have backreferences.

ATTENTION AUX FAUX FRÈRES!

ATTENTION à la spécification des pbs!
Il suffit de changer à peine un problème pour qu'il ne soit plus $NP-$dur:
Le 3-coloriage est dur ($NP-$dur) le 2-coloriage de graphe est facile (P).

ATTENTION AUX FAUX FRÈRES!

ATTENTION à la spécification des pbs!
Il suffit de changer à peine un problème pour qu'il ne soit plus $NP-$dur:
Le 3-coloriage est dur ($NP-$dur) le 2-coloriage de graphe est facile (P).

La programmation linéaire en entiers est $NP-$dure, la programmation linéaire en réels est P.

COMPLEXITÉ DES PROBLÈMES: BILAN

- *NP* : "easy to check"

Complexité des problèmes: bilan

- NP : "easy to check"
- $NP - dur$: contient toute la complexité de NP, "aussi dur" que n'importe quel NP.

COMPLEXITÉ DES PROBLÈMES: BILAN

- NP : "easy to check"
- $NP - dur$: contient toute la complexité de NP, "aussi dur" que n'importe quel NP.
- pour montrer que R est $NP-$dur: on cherche à réduire polynomialement un pb connu $NP-$ dur dans R.

- nouvelle série de TP commence cette semaine sur le voyageur de commerce ou Travelling Salesman: `http://www.travellingsalesmanmovie.com/`
- DS lundi 19 novembre