Complexité et Algorithmes
Partie II : Complexité des Problèmes

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

Université de Nantes, LS2N
Bât 34 – Bureau 301

M1 Informatique – 2019-2020

Sommaire

Différence entre Algorithme et Problème

Complexité d'un algorithme

- Jusque là: complexité **des algorithmes**
- Càd: étant donné un algorithme A, quel est son coût (en temps) ?

Algorithme vs Problème

- Problème = question générale
- Algorithme = une façon de résoudre un problème
- $\Rightarrow$ Que signifie la complexité d'un problème ?

Complexité d'un Problème

Définition

Complexité d'un problème = complexité du meilleur[1] algorithme qui le résout

[1] meilleur = le plus rapide en temps d'exécution

Divers types de Problèmes

Les classes de problèmes les plus fréquentes

- **Problèmes de calcul**: sortie = une ou plusieurs valeur(s)
  ex: calcul des racines d'un polynôme du second degré
- **Problèmes d'énumération**: sortie = un ensemble de réponses
  ex: PATTERN MATCHING
- **Problèmes d'optimisation**: max/min-imiser une valeur
  ex: ROBOT SOUDEUR
- **Problèmes de décision**: sortie = oui/non
  ex: primalité d'un nombre

Sommaire

Problèmes de Décision et d'Optimisation

Remarque

A partir de maintenant: exclusivement problèmes **de décision** et **d'optimisation**

Problèmes de Décision et d'Optimisation

Pourquoi ?

- Problèmes de décision (PbD) "simples"... à poser
- Problèmes d'optimisation (PbO) très liés aux PbD (on le verra plus tard)
- Les PbD peuvent sembler limités mais
  1. il y en a déjà beaucoup à étudier, et
  2. ce qu'on va dire sur eux est généralisable aux autres classes de problèmes

Problèmes de Décision

Réponse = oui/non ⇒ L'énoncé est une question

Quelques exemples

- Étant donné un entier n, n est-il premier ?
- Étant donnés un graphe G et un entier k, G admet-il un cycle de longueur $\geq k$?
- Étant donné un graphe G planaire, G peut-il être (sommet-)colorié avec 4 couleurs, sans que deux sommets voisins aient la même couleur[1] ?

[1] on parle alors de coloration propre

Problèmes de Décision

Réponse = oui/non ⇒ L'énoncé est une question

Quelques exemples

- Étant donnés deux programmes (informatiques) P et P', P et P' sont-ils équivalents[1] ?
- Étant donné un programme P, P termine-t-il ?

[1] càd, pour la même instance d'entrée, P et P' font systématiquement la même chose

Problèmes de Décision

Description standard

- Formellement, on décrit un PbD sous la forme suivante :
  - **NOM**: identifie le problème (en majuscules)
  - **Instance**: description des données d'entrée du problème
  - **Question**: la question posée

Problèmes de Décision

Description standard PbD – Exemples

PRIMES :
Instance : un entier n
Question : n est-il premier ?

k-**CYCLE** :
Instance : un graphe G, un entier k
Question : G admet-il un cycle de longueur $\geq k$?

Problèmes de Décision

Questions

Les problèmes suivants sont-ils des PbD ?

- Déterminer si un entier n est pair ou impair $\rightarrow$ décision
- Déterminer le PGCD de deux entiers m et n $\rightarrow$ optimisation
- Déterminer le nombre minimum de couleurs nécessaire pour colorier de façon propre les sommets d'un graphe G donné $\rightarrow$ optimisation

Liens Optimisation/Décision

Optimisation n'est pas décision, mais...

- Tout PbO peut se ramener à un PbD
- Signification:

Algo Polynomial pour PbO $\Leftrightarrow$ Algo Polynomial pour PbD

Liens Optimisation/Décision

Illustration sur un exemple

PbO $\leftrightarrow$ PbD

- Coloration d'un graphe G: affecter une couleur à chaque sommet de G
- Coloration propre de G: 2 sommets voisins portent toujours des couleurs différentes
- **Problème**: étant donné un graphe G, déterminer le nombre minimum de couleurs nécessaire pour colorier proprement G
- Problème d'optimisation, qu'on appellera **O-COL**

Liens PbO/PbD

Passage PbO/PbD

O-COL :
Instance : un graphe G
Sortie : le nombre minimum c de couleurs nécessaires pour colorier proprement G

D-COL :
Instance : un graphe G, un entier k
Question : G peut-il être proprement colorié en $\leq k$ couleurs ?

Liens Optimisation/Décision

Polynomial(PbO) $\Rightarrow$ Polynomial(PbD)

- Résolution de **O-COL** $\rightarrow$ donne un nombre c_{opt} de couleurs
- Concernant **D-COL**:
  - Si $c_{opt} \leq k \Rightarrow$ réponse OUI
  - Si $c_{opt} > k \Rightarrow$ réponse NON

Liens Optimisation/Décision

Polynomial(PbD) $\Rightarrow$ Polynomial(PbO)

- $n =$ nombre de sommets de G
- Remarque: pour tout graphe G, $1 \leq c_{opt} \leq n$[1]
- Idée: répondre à **O-COL** en faisant des appels à **D-COL**

[1] 1 pour un graphe sans arête ; n pour la clique

Liens Optimisation/Décision

Algorithme O-COL(G: graphe)

- int i:=0
- bool ok:=faux
- **Tant que** ok=faux et i$\leq$ n-1 **faire**
- i:=i+1
- ok:=D-COL(G,i)
- **Fin Tant que**
- Return i;

Liens Optimisation/Décision

Polynomial(PbD) $\Rightarrow$ Polynomial(PbO)

- Complexité de **O-COL** $= O(n\times$ Complexité de **D-COL**$)$
- taille des données $= n$ (G possède n sommets et au plus $\frac{n(n-1)}{2}$ arêtes)

Conclusion: **D-COL** polynomial $\Rightarrow$ **O-COL** polynomial

Remarque: on peut faire mieux (dichotomie) $\Rightarrow \sim \log_2 n$ appels à **D-COL** au pire

Liens Optimisation/Décision

En règle générale

Algo polynomial pour PbO $\Rightarrow$ Algo polynomial pour PbD

Comparaison entre

- k = paramètre de PbD **et**
- OPT = valeur optimale de PbO

Algo polynomial pour PbD $\Rightarrow$ Algo polynomial pour PbO

Recherche dichotomique pour déterminer OPT

Remarques

- On parle de PbD associé à un PbO
- $\Rightarrow$ A partir de maintenant, **PbD** uniquement (sauf mention contraire)

Sommaire

Motivations

Savoir où on va !

Il y a des problèmes pour lesquels on ne trouve pas d'algorithmes polynomiaux

- diagnostiquer qu'on se trouve confronté à de tels problèmes
- ne pas chercher un algorithme efficace $\rightarrow$ réduire ses exigences

La classe P

P =polynomial

En première approche, deux classes de complexité pour les problèmes:

- La **classe** P: tous les problèmes pour lesquels il existe un algorithme de **coût polynomial**
- La classe des autres

La classe P

Définition P

Un PbD appartient à P s'il existe un algorithme A et une constante c, tel que **pour toute instance I** du PbD :

- A résout le PbD en temps polynomial : $O(n^c)$
- où n est la **taille des données**

Problèmes $\in$ P

Exemples de PbO et PbD $\in$ P

- PbO:
  - Problème du Plus Court Chemin (Algorithme de Dijkstra)
  - Problème du PGCD (Algorithme d'Euclide)
- PbD:
  - PRIMES (résultat datant de 2002)
  - 2-SAT (résultat datant de 1967 ; on y reviendra)

La classe P

En résumé sur P

- Tout PbD $\in$ P est dit facile (ou *tractable*, ou traitable)
- Tout PbD $\notin$ P est dit difficile (ou *intractable*, ou intraitable)

Exemples de PbD $\notin$ P

- Terminaison des programmes
- Équivalence des programmes

La classe P

Facile ou difficile ?

Problème $\in$ P ou $\notin$ P ? Pas toujours simple...

- Si $\in$ P:
  - fournir un algorithme
  - montrer qu'il est correct
  - montrer qu'il est polynomial en la taille des données
- Si $\notin$ P:
  - montrer qu'aucun algorithme polynomial n'existe
  - en général très compliqué
  - sort du cadre de ce cours

P or not P?

Quand on ne sait pas...

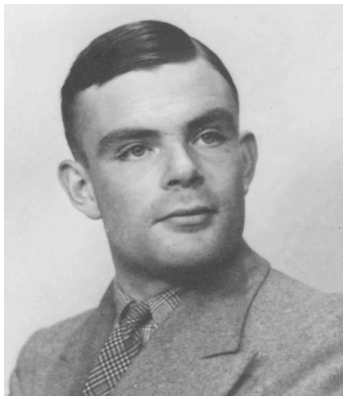
- Pour beaucoup de PbD, on ne sait rien:
  - aucun algo polynomial connu $\Rightarrow$ tous sont exponentiels...
  - ...mais aucune preuve que le PbD $\notin$ P !
- Idée: inventer une classe intermédiaire: NP
- ATTENTION: NP $\leftrightarrow$ Nondeterministic Polynomial

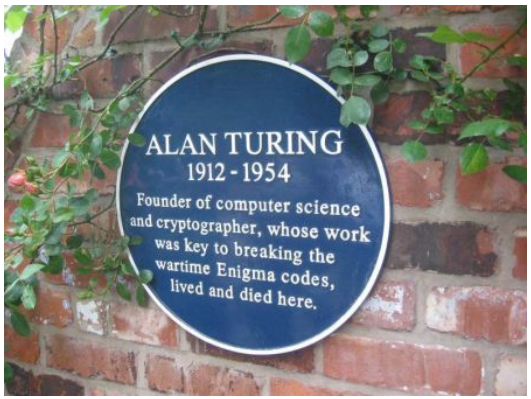
NP **ne veut pas dire NON POLYNOMIAL !**

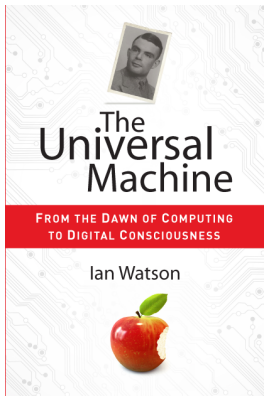
Récréation (1/2)

Alan Turing

- Mathématicien et logisticien anglais (1912-1954)
- Un des pères de l'informatique



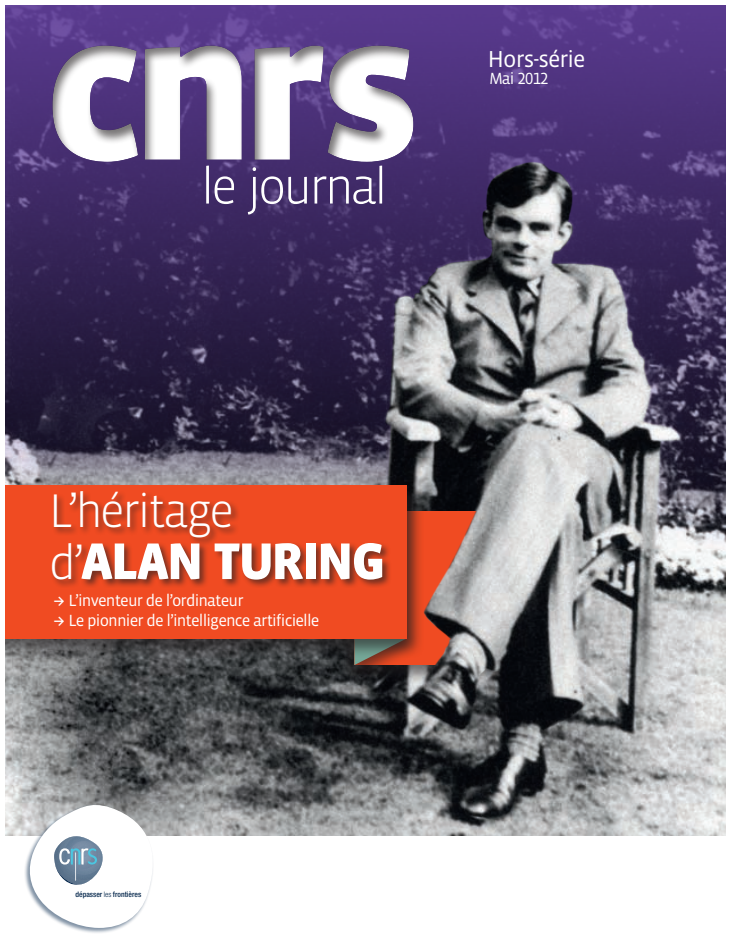




- Également célèbre pour:
  - avoir "cassé" le code de la machine Enigma pendant la 2e guerre mondiale
  - s'être suicidé en mangeant une pomme empoisonnée
- $\rightarrow$ la complexité des problèmes est directement reliée aux travaux de Turing

Récréation (2/2)

100e anniversaire en 2012



Retour à la classe NP

Définition NP

PbD $\in$ NP si:

- pour chaque instance positive I (càd: réponse OUI)
- il existe un certificat $C(I)$ (de sa positivité) vérifiant:
  1. taille de $C(I)$: **polynomiale** en la taille des données de PbD
  2. vérification à partir de $C(I)$: en temps **polynomial**

La classe NP

Moins formellement

- NP = classe des PbD pour lesquels il est "facile" de **vérifier** qu'une réponse fournie est correcte
- Le but est de **vérifier** qu'une solution fournie est correcte, **pas de trouver** la solution
- Exigences diminuées

La classe NP

En résumé

- PbD $\in$ P $\leftrightarrow$ solution **facile à trouver**
- PbD $\in$ NP $\leftrightarrow$ solution **facile à vérifier**

La classe NP

NP: Exemple

Problème **D-COL**

- Instance: un graphe G, un entier k
- Question: G peut-il être proprement colorié en $\leq k$ couleurs ?

Montrons que **D-COL**$\in$ NP :

- Certificat (pour toute instance I) ? liste sommet $\leftrightarrow$ couleur pour chacun des n sommets de G
- Taille du certificat ? $O(n)$, donc polynomial
- Vérification :
  1. nombre de couleurs $\leq k$: $O(n)$
  2. chaque sommet a exactement une couleur : $O(n)$
  3. deux sommets voisins n'ont pas la même couleur : $O(n^2)$
- Taille et vérification polynomiales $\Rightarrow$ **D-COL**$\in$ NP

La classe NP

NP, et alors ?

Remarque: Si PbD$\in$ P, alors PbD$\in$ NP:

- trouver est plus dur que vérifier
- si trouver est facile, alors vérifier l'est aussi
- $\Rightarrow$ P $\subseteq$ NP

Quid des autres PbD (ceux dont on ne sait rien) ?

- But: comparer les problèmes $\in$ NP entre eux
- Identifier les plus difficiles de NP

$\Rightarrow$ notion de **réduction d'un problème vers un autre**

Réduction d'un problème vers un autre

Définition Réduction

- Soit Pb et Pb' deux PbD
- Réduction de Pb' vers Pb si:
  - Partant de **toute** instance I' de Pb'...
  - ...on peut construire **une** instance I de Pb
  - Contraintes:
    1. $I' \rightarrow I$ en temps polynomial
    2. I et I' ont toujours la même réponse (OUI ou NON)
- On écrira $Pb' \geq_{\mathrm{P}} Pb$

Réduction

Idée principale

But: comparer les PbD de NP entre eux

Supposons que $Pb' \geq_{\mathrm{P}} Pb$. Alors:

- Si $Pb \in \mathrm{P}$, alors $Pb' \in \mathrm{P}$

$\Rightarrow$ **Pb** est au moins aussi difficile que **Pb'**

Les problèmes NP-complets

NP + réduction $\Rightarrow$ les problèmes les plus difficiles de NP

Nouvelle classe: NP-complet

Définition NP-complet

Un PbD est NP-complet si :

1. PbD $\in$ NP **et**
2. chaque problème NP peut se réduire vers lui

Abus de langage: on dit "est NP-complet" au lieu de $\in$ NP-complet

Les problèmes NP-complets

Autrement dit

- Un PbD NP-complet est un problème NP au moins aussi difficile que tout autre problème NP
- Les PbD NP-complets sont les problèmes **les plus difficiles de** NP
- Remarque : les PbD NP-complets sont **nombreux** !

Les problèmes NP-complets

Theorem

Soient P_1 et P_2 deux PbD tels que:

- *P_1 est* NP*-complet*
- $P_2 \in$ NP
- $P_1 \geq_{\mathrm{P}} P_2$

Alors P_2 est NP*-complet*

Preuve

- Pour tout PbD $Pb \in$ NP, on a $Pb \geq_{\mathrm{P}} P_1$ (car P_1 est NP-complet)
- Or, $P_1 \geq_{\mathrm{P}} P_2 \Rightarrow Pb \geq_{\mathrm{P}} P_2$
- De plus, $P_2 \in$ NP $\Rightarrow$ P_2 est NP-complet

Marche à suivre

Pour montrer qu'un PbD Pb est NP-complet, il faut montrer:

1. $Pb \in \text{NP}$
2. il existe un PbD NP-complet P_1 tel que $P_1 \geq_{\text{P}} Pb$

Remarques:

- Dû à Définition + Théorème
- Plus simple que la Définition seule

Sommaire

L'œuf et la poule

Montrer qu'un PbD est NP-complet implique de réduire un problème NP-complet vers lui...

Remarque

Il n'est a priori pas évident qu'il existe au moins un problème NP-complet

Satisfiabilité des formules booléennes

Satisfiabilité des formules booléennes

- Variables booléennes $X = \{x_1, x_2, \ldots, x_n\}$
- Clause: disjonction de littéraux, par ex.

$$(x_4 \vee \overline{x_5} \vee x_1 \vee \overline{x_6})$$

- Forme normale conjonctive (FNC): conjonction de clauses, par ex.

$$\ldots \wedge (x_2 \vee x_3) \wedge (x_4 \vee \overline{x_5} \vee x_1 \vee \overline{x_6}) \wedge (x_2 \vee \overline{x_5} \vee \overline{x_1}) \wedge \ldots$$

- Une affectation est une fonction $\nu : X \to \{\texttt{vrai}, \texttt{faux}\}$
- Une formule est satisfiable s'il existe une affection qui satisfait la formule (càd qui la rend vraie)

Satisfiabilité des formules booléennes

Satisfiabilité des formules booléennes

- Toutes les formules booléennes ne sont pas satisfiables, par ex.

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

- Soit ϕ une formule booléenne sous FNC. Alors ϕ est satisfiable si et seulement si chaque clause est satisfaite
- Toute formule booléenne peut être mise sous FNC
- Il existe 2^n affectations différentes

Le problème SAT est NP-complet

SAT

Instance: Une formule booléenne ϕ sous FNC

Question: La formule ϕ est-elle satisfiable ?

Theorem (Cook 1971)

Le problème SAT *est* NP*-complet*

Le problème k-SAT est NP-complet

k-SAT

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

Question: La formule ϕ est-elle satisfiable ?

Theorem

Pour tout $k \geq 3$, *le problème* k-SAT *est* NP-*complet*

Quelques remarques sur le problème SAT

Cas particuliers polynomiaux

- Le problème 2-$\mathrm{SAT} \in$ P
- Une formule booléenne de Horn est une conjonction de clauses telle que chaque clause contient au plus une variable positive, par ex.

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

Quelques remarques sur le problème SAT

Cas particuliers polynomiaux

HORN-SAT :
Instance : Une formule booléenne de Horn ϕ sous FNC
Question : La formule ϕ est-elle satisfiable ?

Le problème $\mathrm{HORN\text{-}SAT} \in \mathrm{P}$

Quelques variantes difficiles du problème SAT

1-SUR-3-SAT

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

Question: Existe-t-il une affectation qui satisfait exactement un littéral par clause ?

Theorem

Le problème 1-SUR-3-SAT *est* NP-*complet*

Remarque: Réduction depuis SAT

$$\text{SAT} \geq_{\text{P}} \text{1-SUR-3-SAT}$$

Quelques variantes difficiles du problème SAT

NOT-ALL-EQUAL-3-SAT

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

Question: Existe-t-il une affectation qui satisfait au moins 1 littéral par clause et au plus 2 ?

Theorem

Le problème NOT-ALL-EQUAL-3-SAT *est* NP*-complet*

Exemple de réduction

Ensemble stable[1] dans un graphe G = ensemble de sommets de G ne partageant aucune arête

ENSEMBLE STABLE

Instance: Un graphe G, un entier k

Question: Existe-t-il un ensemble stable de taille $\geq k$ dans G ?

- PbO associé: problème de maximisation
- Modélise des (in)compatibilités

[1] en anglais: Independent Set

Notre première preuve de NP-complétude

Theorem

Le problème Ensemble Stable *est* NP*-complet*

Proof.

1. Ensemble Stable $\in$ NP puis
2. 3-SAT $\geq_{\mathsf{P}}$ Ensemble Stable

□

Notre première preuve de NP-complétude

Ensemble Stable $\in$ NP

Proof.

- certificat polynomial: liste des sommets de l'ensemble stable proposé (qu'on appellera S)
- vérification en temps polynomial:
  1. S a au moins k sommets $\rightarrow O(k)$
  2. pas d'arêtes entre les sommets de $S \rightarrow O(k^2)$

$$\Rightarrow \text{Ensemble Stable} \in \text{NP}$$

□

Notre première preuve de NP-complétude

3-SAT $\geq_P$ Ensemble Stable

Proof.

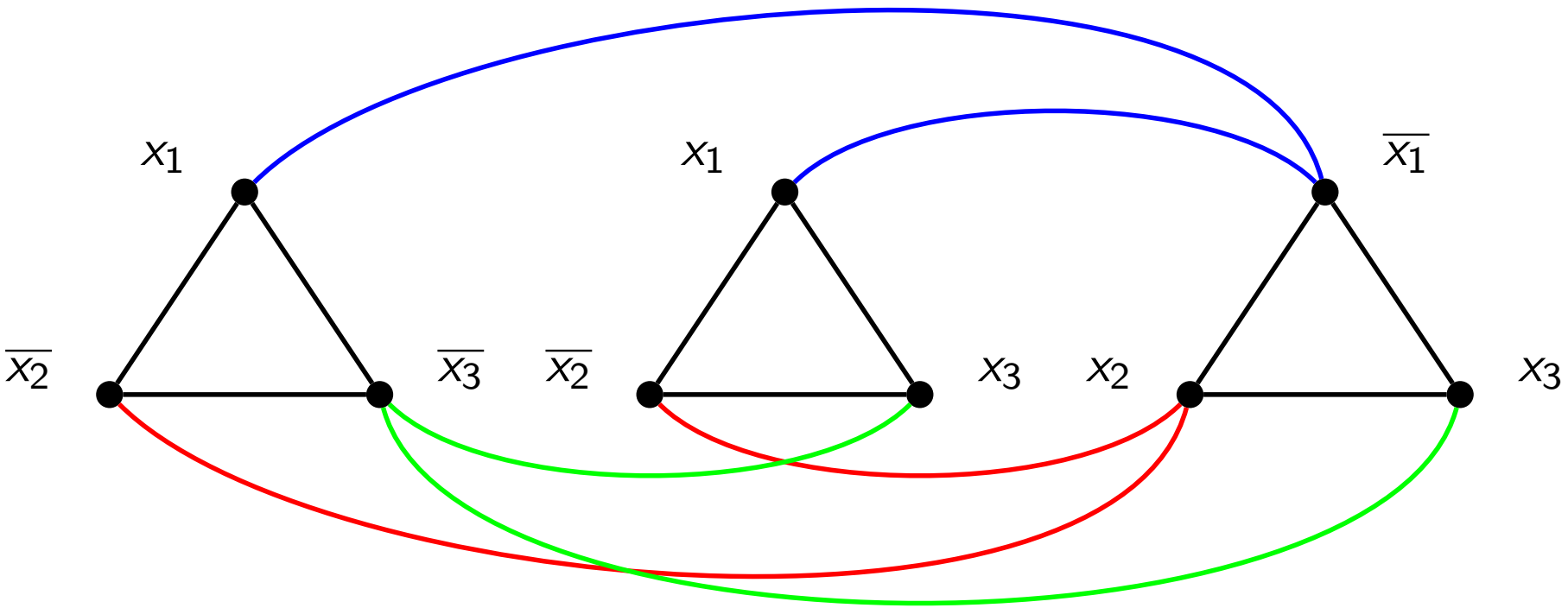
- Réduction polynomiale depuis le problème 3-SAT
- $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ une formule booléenne sous FNC
- Il faut construire un graphe G et un entier positif k tels que

ϕ est satisfiable $\Leftrightarrow$ il existe un ensemble stable de taille $\geq k$ dans G

□

Le problème Ensemble Stable est NP-complet

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

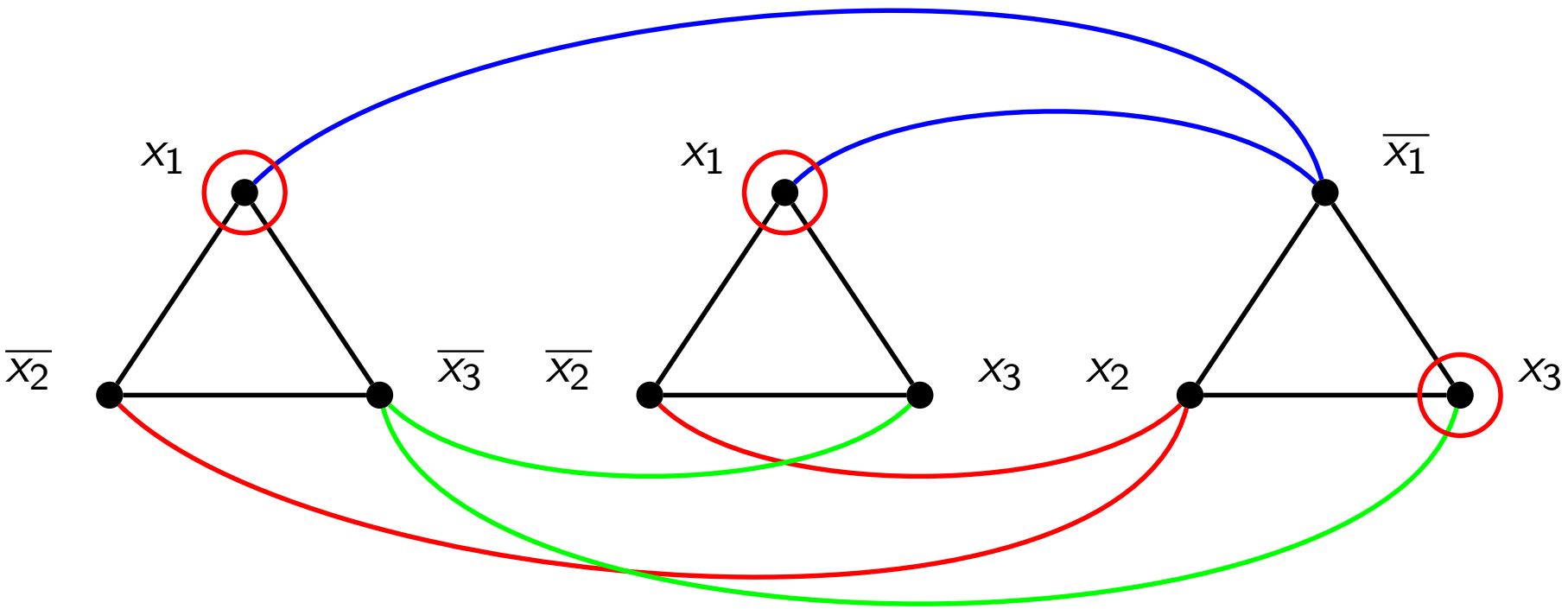


k = nombre de clauses = 3

Le problème Ensemble Stable est NP-complet

($\Rightarrow$) Si ϕ est satisfiable, alors il existe un ensemble stable de taille $k \geq 3$ dans G (en fait, $k = 3$)

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



Le problème Ensemble Stable est NP-complet

($\Leftarrow$) S'il existe un ensemble stable de taille $k \geq 3$ (k = le nombre de clauses de ϕ), alors ϕ est satisfiable

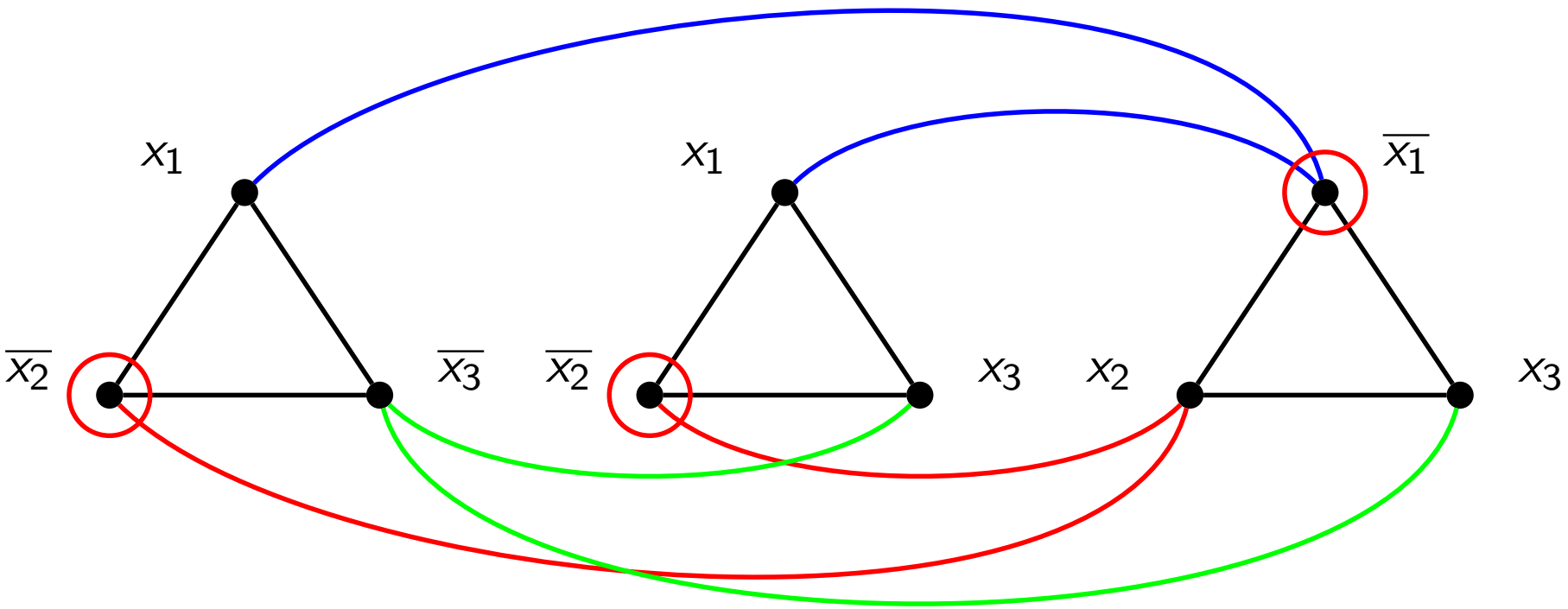
Remarques:

- Ensemble stable $\rightarrow$ au maximum un sommet par triangle
  - $\Rightarrow$ Ens. stable $\leq k$
  - $\Rightarrow$ Ens. stable $= k$
  - $\Rightarrow$ exactement 1 sommet par triangle
- dans l'Ens. stable, impossible d'avoir x_i et $\overline{x_i}$ (par construction)

Le problème Ensemble Stable est NP-complet

($\Leftarrow$) S'il existe un ensemble stable de taille $k \geq 3$ ($k =$ le nombre de clauses de ϕ), alors ϕ est satisfiable

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



Attention à NP-complet!

Ce que veut dire NP-complet

Un PbD est NP-complet veut dire:

- qu'il existe au moins une instance difficile
- ...mais pas forcément toutes !!!
- Des cas particuliers peuvent être polynomiaux:
  - Petits cas (ex: 2-SAT)
  - Classes d'instances (ex: HORN-SAT)
  - etc.

Attention aux valeurs de k (Petits cas)

Theorem

Si $k = O(1)$, alors le problème Ensemble Stable *peut se résoudre en temps polynomial*

Exemples: $k = 2$, $k = 3$

Proof.

On considère tous les ensembles de k sommets dans G

On teste la stabilité de chaque ensemble

Algo en:

$$O(\underbrace{\binom{n}{k}}_{\text{nb d'ensembles}} \cdot \underbrace{k^2}_{\text{test}}) = O(n^k \cdot k^2) \qquad \text{Polynomial !}$$

Attention à la structure de G (Classes d'instances)

Theorem

Si G est un chemin, alors le problème Ensemble Stable *peut se résoudre en temps polynomial*

Proof.

- Ensemble stable: un sommet sur deux dans le chemin
- $\rightarrow$ nécessairement optimal

□

Sommaire

Pour revenir à la classe NP-complet

A quoi ça peut bien servir ?

1. Si un seul PbD NP-complet est polynomial $\Rightarrow$ tous les problèmes NP aussi !
2. Inversement, si un seul PbD NP est intraitable $\Rightarrow$ tous les problèmes NP-complets aussi !

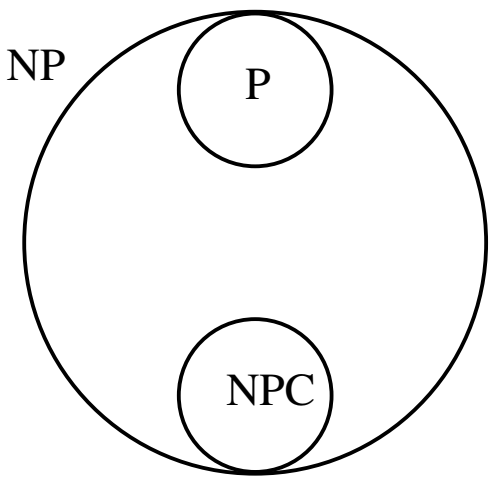
Quelques propriétés des NP-complets

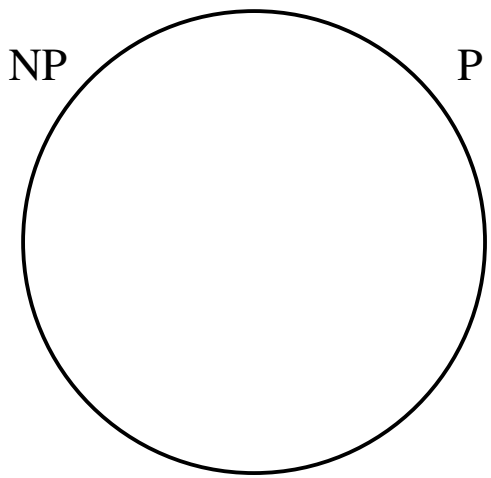
Problèmes NP-complets: où en est-on ?

- Problèmes tous très difficiles à résoudre
- Aucun algorithme polynomial n'a été trouvé pour ces problèmes
- L'impossibilité de trouver des algorithmes polynomiaux n'a pas été prouvée non plus !
- L'existence ou non d'algorithmes polynomiaux pour les problèmes NP-complets est l'un des plus grands problèmes encore ouverts en informatique

Relations entre P et NP ...

- Les problèmes NP-complets sont les problèmes les plus difficiles de la classe NP
- $P \subseteq NP$
- mais qu'est-ce qui est correct ?





La grande question

P =NP?

- Recherches innombrables sur le sujet depuis des dizaines d'années
- Fait partie des 7 problèmes du millénaire du Clay Mathematics Institute[1] (1 million de dollars par problème résolu)
- Les implications sont multiples et réelles ! Exemple: codage RSA (transactions cryptées sur Internet)
- On ne sait toujours pas:
  - si les PbD $\in$ NP sont tous polynomiaux...
  - ou s'ils sont tous intraitables

[1] http://www.claymath.org/millennium-problems

La grande question

P =NP?

L'opinion généralement partagée est que :

- P $\subset$ NP, càd les problèmes NP-complets ne sont pas polynomiaux
- le PbD "P =NP ?" est lui-même NP-complet... (?!?)

Que faire face à un problème inconnu ?

Notre problème *Pb*

- soit on pense que le problème *Pb* est **facile**
  $\Rightarrow$ on cherche un algorithme *correct* et polynomial (le meilleur possible!) qui le résout
- soit on pense que le problème *Pb* est **difficile**
  $\Rightarrow$ on cherche à montrer qu'il est NP-complet, càd
  - toute solution proposée peut être polynomialement **vérifiable** (appartenance à NP)
  - prendre un problème NP-complet et le **réduire** polynomialement à *Pb*

Que faire face à un problème Polynomial ?

Si *Pb* est polynomial

- Trouver le "meilleur" algorithme
- S'assurer qu'il est correct!
- Meilleur = le plus rapide
- Meilleur = le moins gourmand en mémoire
- La priorité est souvent mise sur le temps

Que faire face à un problème NP-complet?

Si *Pb* est NP-complet

- Premier constat: ne pas s'acharner à trouver un algorithme exact et rapide
- Baisser ses exigences:
  - soit sur la **rapidité** d'exécution
  - soit sur l'**exactitude** de la réponse
  - soit sur l'**ensemble des instances** autorisées

$\Rightarrow$ c'est ce que nous verrons dans la suite