Algorithmique et Complexité
Cours 6/7 : Théorie de la complexité

CentraleSupélec – Gif

ST2 – Gif

Problèmes en algorithmique

Ce que nous avons vu :

- problèmes de décision (existence d'un chemin, ...) ;
- problèmes d'optimisation (arbre couvrant de poids minimum, flot maximum, alignement de séquence,...) ;
- des algorithmes de complexité polynomiale $\mathcal{O}(n^c)$ avec n la taille de l'instance et c une constante.

Problèmes en algorithmique

Ce que nous avons vu :

- problèmes de décision (existence d'un chemin, ...) ;
- problèmes d'optimisation (arbre couvrant de poids minimum, flot maximum, alignement de séquence,...) ;
- des algorithmes de complexité polynomiale $\mathcal{O}(n^c)$ avec n la taille de l'instance et c une constante.

Questions

- Que peut-on calculer ?

Problèmes en algorithmique

Ce que nous avons vu :

- problèmes de décision (existence d'un chemin, ...) ;
- problèmes d'optimisation (arbre couvrant de poids minimum, flot maximum, alignement de séquence,...) ;
- des algorithmes de complexité polynomiale $\mathcal{O}(n^c)$ avec n la taille de l'instance et c une constante.

Questions

- Que peut-on calculer ?
- Que peut-on calculer efficacement ?
  $\longrightarrow$ Existe-t-il toujours des algorithmes polynomiaux ?

Problèmes en algorithmique

Ce que nous avons vu :

- problèmes de décision (existence d'un chemin, ...) ;
- problèmes d'optimisation (arbre couvrant de poids minimum, flot maximum, alignement de séquence,...) ;
- des algorithmes de complexité polynomiale $\mathcal{O}(n^c)$ avec n la taille de l'instance et c une constante.

Questions

- Que peut-on calculer ?
- Que peut-on calculer efficacement ?
  $\longrightarrow$ Existe-t-il toujours des algorithmes polynomiaux ?
- Comment formaliser la notion de complexité ?
- Comment classer les problèmes par complexité ?

Plan

Abstraction d'un ordinateur : la Machine de Turing, 1936

Ressources

1. fichier de données comme entrée
2. instructions du programme
3. mémoire
4. registres et pile d'exécution
5. fichier de données comme sortie

Abstraction d'un ordinateur : la Machine de Turing, 1936

Ressources

1. fichier de données comme entrée
2. instructions du programme
3. mémoire
4. registres et pile d'exécution
5. fichier de données comme sortie

Machine de Turing

- La *Machine de Turing* est une formalisation de cette structure.

Abstraction d'un ordinateur : la Machine de Turing, 1936

Ressources	$\longrightarrow$	Formalisation
1. fichier de données comme entrée		ruban d'entrée
2. instructions du programme		
3. mémoire		
4. registres et pile d'exécution		
5. fichier de données comme sortie		

Machine de Turing

- La *Machine de Turing* est une formalisation de cette structure.

Abstraction d'un ordinateur : la Machine de Turing, 1936

Ressources	$\longrightarrow$	Formalisation
1. fichier de données comme entrée		ruban d'entrée
2. instructions du programme		table d'actions
3. mémoire		
4. registres et pile d'exécution		
5. fichier de données comme sortie		

Machine de Turing

- La *Machine de Turing* est une formalisation de cette structure.

Abstraction d'un ordinateur : la Machine de Turing, 1936

Ressources	$\longrightarrow$	Formalisation
1. fichier de données comme entrée		ruban d'entrée
2. instructions du programme		table d'actions
3. mémoire		ruban de travail
4. registres et pile d'exécution		
5. fichier de données comme sortie		

Machine de Turing

- La *Machine de Turing* est une formalisation de cette structure.

Abstraction d'un ordinateur : la Machine de Turing, 1936

Ressources	$\longrightarrow$	Formalisation
1. fichier de données comme entrée		ruban d'entrée
2. instructions du programme		table d'actions
3. mémoire		ruban de travail
4. registres et pile d'exécution		registre d'état
5. fichier de données comme sortie		

Machine de Turing

- La *Machine de Turing* est une formalisation de cette structure.

Abstraction d'un ordinateur : la Machine de Turing, 1936

Ressources	$\longrightarrow$	Formalisation
1 fichier de données comme entrée		ruban d'entrée
2 instructions du programme		table d'actions
3 mémoire		ruban de travail
4 registres et pile d'exécution		registre d'état
5 fichier de données comme sortie		ruban de sortie

Machine de Turing

- La *Machine de Turing* est une formalisation de cette structure.

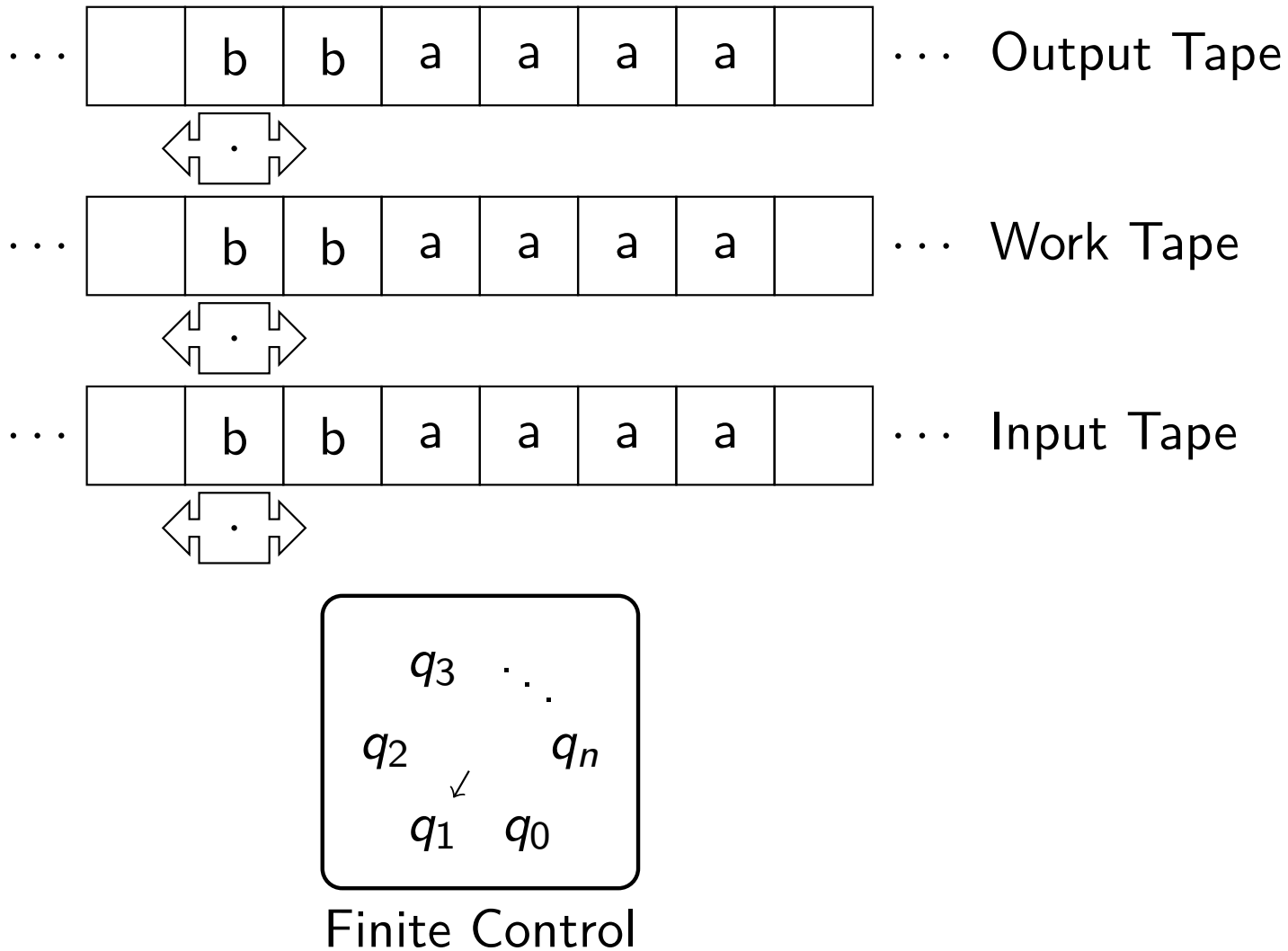
Abstraction d'un ordinateur : la Machine de Turing, 1936

Ressources	$\longrightarrow$	Formalisation
1. fichier de données comme entrée		ruban d'entrée
2. instructions du programme		table d'actions
3. mémoire		ruban de travail
4. registres et pile d'exécution		registre d'état
5. fichier de données comme sortie		ruban de sortie

Machine de Turing

- La *Machine de Turing* est une formalisation de cette structure.
- Il existe plusieurs définitions/variantes de la Machine de Turing (nombre de rubans, alphabet...).

Illustration d'une Machine de Turing



Détails des éléments d'une Machine de Turing

Chaque machine possède :

- un registre d'état qui mémorise l'état courant de la machine de Turing (le nombre d'états possibles est fini) ;
- trois rubans d'entrée, de travail et de sortie divisés en cases pouvant stocker des symboles (alphabet fini) ;
- trois têtes pouvant lire ou écrire sur les rubans et se déplacer d'une case vers la gauche ou vers la droite ;

Détails des éléments d'une Machine de Turing

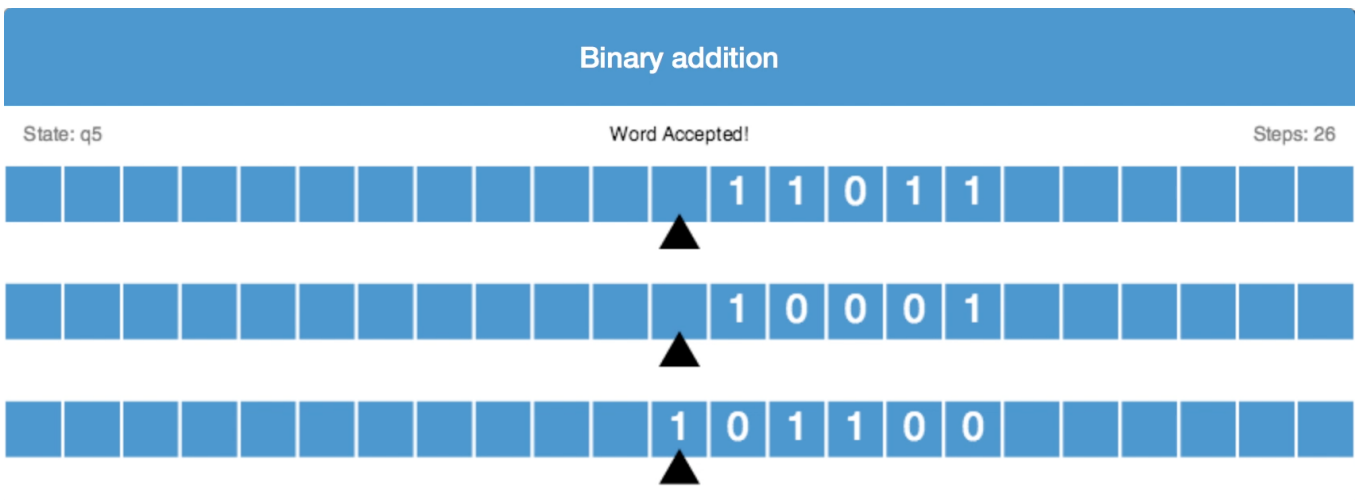
Chaque machine possède :

- un registre d'état qui mémorise l'état courant de la machine de Turing (le nombre d'états possibles est fini) ;
- trois rubans d'entrée, de travail et de sortie divisés en cases pouvant stocker des symboles (alphabet fini) ;
- trois têtes pouvant lire ou écrire sur les rubans et se déplacer d'une case vers la gauche ou vers la droite ;
- une table d'actions qui en fonction :
  - de l'état courant
  - des symboles lus sur les rubans

  indique :
  - quel symbole écrire sur chaque ruban
  - quel déplacement appliquer aux têtes de lecture (gauche/droite)
  - quel est le nouvel état.

Démonstration du fonctionnement d'une Machine de Turing

- Addition de 27 et 17 en binaire : 11011#10001
- Simulateur de MT disponible en ligne : `https://turingmachinesimulator.com/`



La thèse de Church Turing

- D'autres modèles de calcul existent. Jusqu'à présent, tous ont pu être simulés par une Machine de Turing.

La thèse de Church Turing

- D'autres modèles de calcul existent. Jusqu'à présent, tous ont pu être simulés par une Machine de Turing.

Thèse de Church Turing

Tout système de calcul physique (basé sur du silicium, de l'ADN, des neurones ou toute autre technologie extra-terrestre) peut être simulé par une Machine de Turing.

- Cette thèse n'est pas un théorème, mais elle est largement acceptée par la communauté scientifique.
- Elle implique que ce qui est calculable ne dépend pas du modèle de calcul.

Limites de ce modèle

Tout ne peut pas être calculé par une Machine de Turing.

Problème indécidable

Il existe des fonctions qui ne sont pas calculables par les Machines de Turing.

Exemple, le problème de l'arrêt (Turing, Church, 1936)

Il n'existe pas de MT M qui prend en entrée une MT M' quelconque et détermine si M' s'arrête ou non.

- Savoir si un programme se termine est indécidable.

Temps d'exécution d'une Machine de Turing

Comment mesurer le temps d'exécution d'une MT ?

- Chaque action de la machine est une étape (lecture, écriture sur un ruban, décalage d'une tête de lecture).

Temps d'exécution d'une Machine de Turing

Comment mesurer le temps d'exécution d'une MT ?

- Chaque action de la machine est une étape (lecture, écriture sur un ruban, décalage d'une tête de lecture).
- Tout calcul nécessite de lire une entrée dans son ensemble, on mesure donc le temps d'exécution en fonction de la taille de l'entrée.
  - Pour une entrée x on note $|x|$ la taille de x.

Temps d'exécution d'une Machine de Turing

Comment mesurer le temps d'exécution d'une MT ?

- Chaque action de la machine est une étape (lecture, écriture sur un ruban, décalage d'une tête de lecture).
- Tout calcul nécessite de lire une entrée dans son ensemble, on mesure donc le temps d'exécution en fonction de la taille de l'entrée.
  - Pour une entrée x on note $|x|$ la taille de x.
- On dit qu'une MT calcule une fonction f en temps $T(n)$ si le calcul de toute entrée x tel que $n = |x|$ nécessite au plus $T(|x|)$ étapes.

La classe *P*

Définition de *P*

La classe *P* est l'ensemble des problèmes qui peuvent être résolus par des Machines de Turing en temps polynomial $poly(n)$.

La classe P

Définition de P

La classe P est l'ensemble des problèmes qui peuvent être résolus par des Machines de Turing en temps polynomial $poly(n)$.

Machine simple vs machine complexe

- Toute fonction f calculable par une MT M avec k rubans et un alphabet Γ en un temps $T(n)$ peut être calculé par une MT $\tilde{M}$ avec un seul ruban et un alphabet binaire en un temps $poly(T(n))$.
- La classe P est donc indépendante du modèle de machine de Turing que l'on considère.

Questions philosophiques

Importance de la classe P

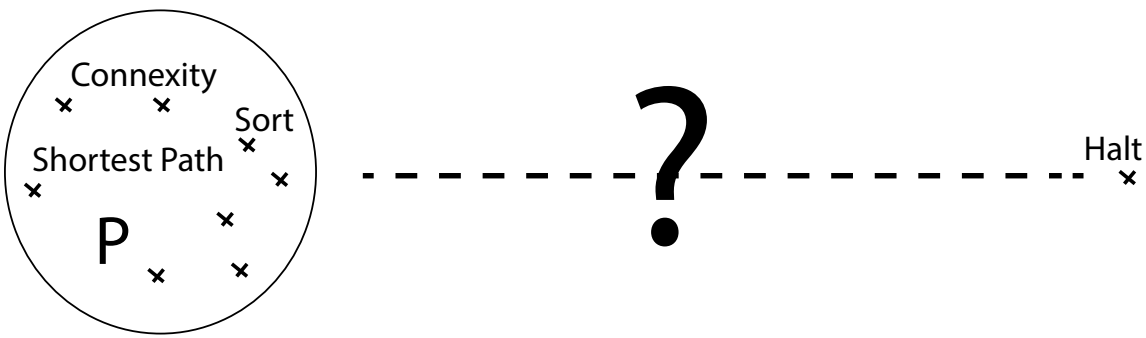
- La classe P capture les problèmes accessibles.

Critiques concernant la classe P

- Bien que l'on puisse douter de l'efficacité d'un algorithme avec une complexité de $\mathcal{O}(n^{100})$, en pratique la complexité des algorithmes de résolution des problèmes de P ne dépasse que rarement $\mathcal{O}(n^5)$.
- La complexité au pire est trop stricte.
- D'autres modèles de calcul devraient être envisagés (quantique, avec utilisation d'aléa).

Enjeu

- La classe *P* est la classe de problèmes pour lesquels il existe des algorithmes efficaces.
- Il existe une classe de problèmes pour lesquels il n'existe pas d'algorithme.
- Qui y a-t-il entre ces deux classes ?

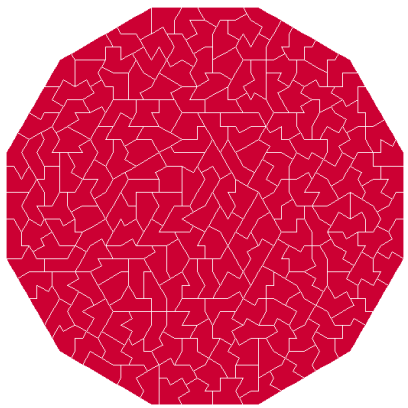


Plan

Eternity

Le puzzle Eternity

- Puzzle de 209 pièces, sorti en juin 1999, associé à une prime de £1'000'000.
- Deux mathématiciens de Cambridge ont remporté le prix en octobre 2000.



Vérifier vs Résoudre

- Dans ce puzzle, il est très simple de vérifier que la solution est correcte alors qu'il est extrêmement difficile de la trouver.
- Cette partie du cours traite de la classe des problèmes *NP*-complets qui présentent une propriété analogue.

Les problèmes de décision

Définition

Un problème de décision partitionne l'ensemble D d'instances en deux sous-ensembles :

- D^+ d'instances positives (pour lesquelles la réponse est oui) ;
- D^- d'instances négatives (pour lesquelles la réponse est non).

Résoudre un tel problème consiste à déterminer, étant donné $I \in D$, si $I \in D^+$ (ou bien $I \in D^-$).

Classe *NP* : intuition

Intuition

L'ensemble des problèmes de décision pour lesquels :
chaque instance positive $I \in D^+$ admet une solution S que l'on peut vérifier par un algorithme en temps polynomial

Classe *NP* : intuition

Intuition

L'ensemble des problèmes de décision pour lesquels :
chaque instance positive $I \in D^+$ admet une solution S que l'on peut vérifier par un algorithme en temps polynomial

- L'algorithme de vérification prend en entrée une solution et répond oui au problème de décision

Classe *NP* : intuition

Intuition

L'ensemble des problèmes de décision pour lesquels :
chaque instance positive $I \in D^+$ admet une solution S que l'on peut vérifier par un algorithme en temps polynomial

- L'algorithme de vérification prend en entrée une solution et répond oui au problème de décision
- L'algorithme de vérification est polynomial
- Aucune contrainte sur l'algorithme de résolution : il peut être exponentiel !

  (qui réponds oui/non au problème initial, sans connaître la solution)

Classe *NP* : intuition

Intuition

L'ensemble des problèmes de décision pour lesquels :
chaque instance positive $I \in D^+$ admet une solution S que l'on peut vérifier par un algorithme en temps polynomial

- L'algorithme de vérification prend en entrée une solution et répond oui au problème de décision
- L'algorithme de vérification est polynomial
- Aucune contrainte sur l'algorithme de résolution : il peut être exponentiel !

(qui réponds oui/non au problème initial, sans connaître la solution)

Pour illustrer !

On peut comprendre et vérifier la preuve d'un théorème, même si c'est bien plus difficile de la trouver soi-même.

⏩ skip formal definition

La Classe *NP* : définition formelle

Définition de la classe *NP*

Un problème de décision appartient à *NP* s'il existe une relation binaire polynomiale R et un polynôme p tels que :

$$I \in D^+ \Leftrightarrow \exists x \text{ t.q. } R(I, x) \text{ et } |x| \leq p(|I|)$$

Remarques

- x est un *certificat* prouvant que l'instance I est positive.
- R est calculable, en temps polynomial, par une machine de Turing.
- On ne considère que les instances positives.
- *NP* signifie "Nondeterministic Polynomial Turing Machine", ce nom provient de la définition originelle de la classe *NP*.

Relations entre *P* et *NP*

$$P \overset{?}{\subsetneq} NP$$

$$P \overset{?}{=} NP$$

$$P \overset{?}{\supsetneq} NP$$

Relations entre *P* et *NP*

Trivial !

$$P \subseteq NP$$

- L'algorithme de résolution est un algorithme de vérification :
  - il produit un certificat (une solution) et répond oui au problème de décision

Relations entre *P* et *NP*

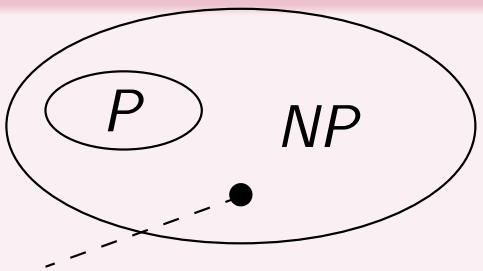
Trivial !

$$P \subseteq NP$$

- L'algorithme de résolution est un algorithme de vérification :
  - il produit un certificat (une solution) et répond oui au problème de décision

Conjecture !

$$P \subsetneq NP$$



- Trouver un problème *NP* et prouver qu'il n'appartient pas à *P*
- $P \neq NP$ est la conjecture la plus célèbre en informatique

Exemple de problème dans *NP* 1/2

Problème du Stable

Instance :

- $G = (V, E)$ un graphe ;
- $k \in \mathbb{N}$

Question : existe-t-il un stable S (independent set) tel que :

- $S \subseteq V$ (sous-graphe) avec $|S| \geq k$
- $\forall u, v \in S.\ \{u, v\} \notin E$ (le sous-graphe induit ne contient pas d'arête)

Stable est dans *NP*

On peut vérifier en temps polynomial par rapport à la taille de G, si une solution $S \subseteq V$ (certificat) est un stable de taille supérieure ou égale à k

Exemple de problème dans *NP* 2/2

Nombre composé

Instance :

- $X \in \mathbb{N}$

Question : X est-il un nombre composé, c'est-à-dire, non premier ?

Le problème du nombre composé est dans *NP*

- Il existe un certificat de taille polynomiale (n, m) avec $n \leq m < x$.
- On vérifie en temps polynomial (par rapport à la taille de x) que $x = n \times m$.

Liste de problèmes dans *NP*

Problèmes dans $P \subseteq NP$

Plus court chemin, arbre couvrant de poids minimum, flot maximum

Problèmes dans *NP*

Stable, sac à dos, sudoku

Liste de problèmes dans NP

Problèmes dans $P \subseteq NP$

Plus court chemin, arbre couvrant de poids minimum, flot maximum

Problèmes dans NP

Stable, sac à dos, sudoku

- En pratique, pour les problèmes NP de la seconde liste, on n'a jamais trouvé d'algorithmes de résolution dans P.
- La plupart des scientifiques pensent que $P \neq NP$

**Que faire des problèmes dans *NP*
qui ne semblent pas être dans *P* ?**

Peut-on classer les problèmes par difficulté ?

Plan

Problème de la clique

Problème de la clique

Instance :

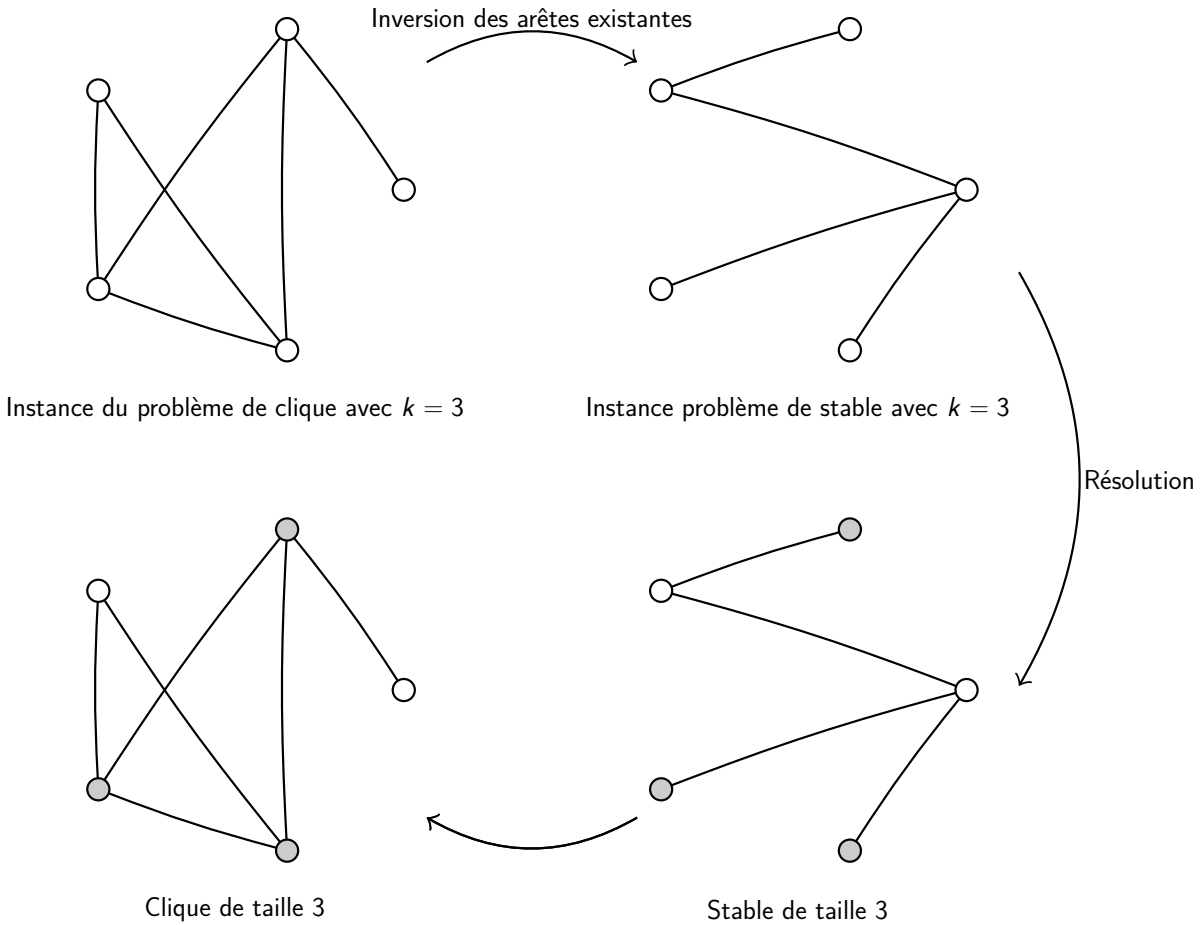
- $G = (V, E)$ un graphe ;
- $k \in \mathbb{N}$

Question : existe-t-il une clique S telle que :

- $S \subseteq V$ (sous-graphe) avec $|S| \geq k$
- $\forall u, v \in S. \ \{u, v\} \in E$ (le sous-graphe induit est complet)

Intuitivement, ce problème appartient à la classe *NP*.

Résoudre Clique par Stable



Résoudre Clique par Stable

Que montre l'algorithme précédent ?

- La transformation précédente peut-être calculée en temps polynomial donc :
  si l'on savait résoudre le problème du stable en temps polynomial, alors on saurait résoudre le problème de la clique également en temps polynomial !

Résoudre Clique par Stable

Que montre l'algorithme précédent ?

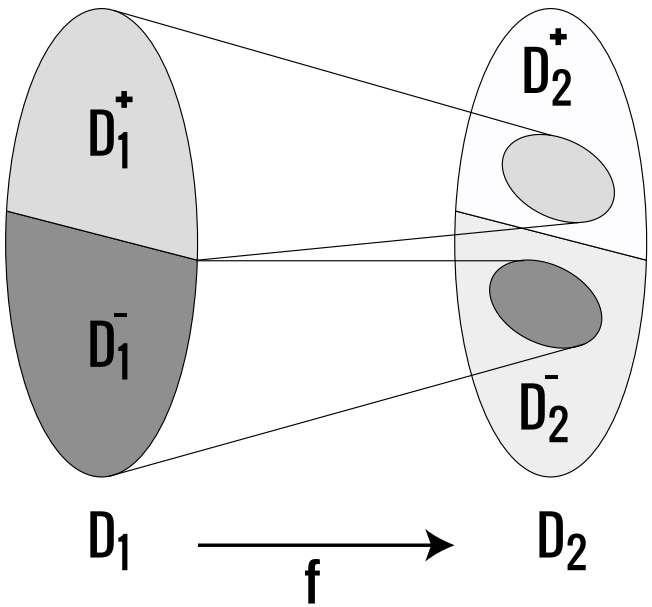
- La transformation précédente peut-être calculée en temps polynomial donc :
  si l'on savait résoudre le problème du stable en temps polynomial, alors on saurait résoudre le problème de la clique également en temps polynomial !
- La notion sous-jacente est la réduction polynomiale.

Réduction Polynomiale $D_1 \leq D_2$

Soient $D_1 = (D_1^+, D_1^-)$ et $D_2 = (D_2^+, D_2^-)$ deux problèmes de décision.

On dit que D_1 se réduit à D_2 sous une réduction de Karp ($D_1 \leq_K D_2$) s'il existe une fonction $f : D_1 \to D_2$ telle que :

- $I \in D_1^+ \iff f(I) \in D_2^+$;
- f est calculable en temps polynomial.



NP-complet

NP-difficile

Un problème D est *NP*-difficile si $D' \leq D, \forall D' \in NP$.

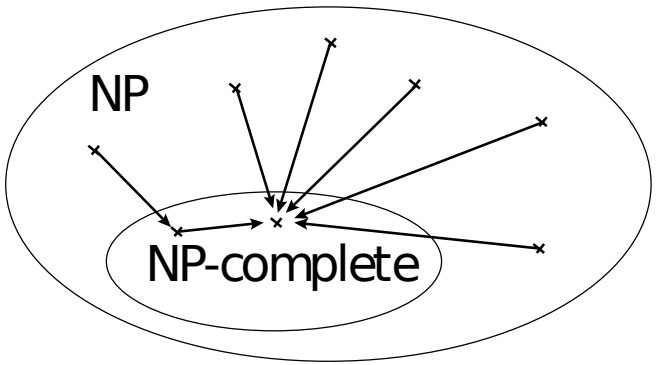
NP-complet

Un problème D est *NP*-complet si D est *NP*-difficile et $D \in NP$.

Théorèmes

- Si $D \leq D'$ et $D' \leq D''$ alors $D \leq D''$.
- Si D est *NP*-difficile et $D \in P$ alors $P = NP$.
- Si D est *NP*-complet alors $D \in P$ ssi $P = NP$.

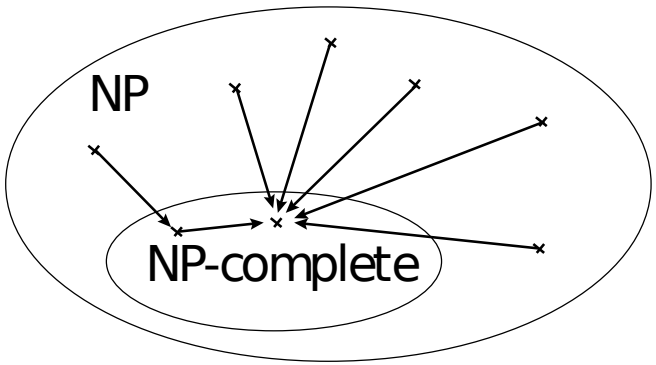
Bilan



Corollaire

On peut réduire tout problème de *NP* dans un problème *NP*-complet.

Bilan



Corollaire

On peut réduire tout problème de *NP* dans un problème *NP*-complet.

Existe-il un problème *NP*-complet ?

Satisfaction de formules booléennes

Le problème SAT

Entrée : une formule sous Forme Normale Conjonctive (FNC)

- Un ensemble U de variables
- Une collection C de clauses disjonctives de littéraux, où les littéraux sont une variable ou la négation d'une variable.

$$(U_1 \vee U_2 \vee \neg U_3) \wedge (\neg U_1 \vee \neg U_4) \wedge (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$

Satisfaction de formules booléennes

Le problème SAT

Entrée : une formule sous Forme Normale Conjonctive (FNC)

- Un ensemble U de variables
- Une collection C de clauses disjonctives de littéraux, où les littéraux sont une variable ou la négation d'une variable.

Question : Existe-t-il une affectation de valeurs aux variables telle que toutes les clauses soient satisfaites ?

$$(U_1 \vee U_2 \vee \neg U_3) \wedge (\neg U_1 \vee \neg U_4) \wedge (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$

Satisfaction de formules booléennes

Le problème SAT

Entrée : une formule sous Forme Normale Conjonctive (FNC)

- Un ensemble U de variables
- Une collection C de clauses disjonctives de littéraux, où les littéraux sont une variable ou la négation d'une variable.

Question : Existe-t-il une affectation de valeurs aux variables telle que toutes les clauses soient satisfaites ?

$$(U_1 \vee U_2 \vee \neg U_3) \wedge (\neg U_1 \vee \neg U_4) \wedge (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$

Un premier ensemble de solutions...

Satisfaction de formules booléennes

Le problème SAT

Entrée : une formule sous Forme Normale Conjonctive (FNC)

- Un ensemble U de variables
- Une collection C de clauses disjonctives de littéraux, où les littéraux sont une variable ou la négation d'une variable.

Question : Existe-t-il une affectation de valeurs aux variables telle que toutes les clauses soient satisfaites ?

$$(U_1 \vee U_2 \vee \neg U_3) \wedge (\neg U_1 \vee \neg U_4) \wedge (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$

Un autre ensemble de solutions. . .

Théorème de Cook-Levin, 1971

Théorème

- SAT est *NP*-complet

Théorème de Cook-Levin, 1971

Théorème

- SAT est *NP*-complet

Idée de la preuve

- Montrer que SAT est dans *NP* est trivial.
- On admettra qu'il est *NP*-difficile :
  étant donné un problème $D \in NP$ et une machine de Turing M qui résout D, pour toute instance I de D il est possible de construire en temps polynomial une formule SAT $\varphi(I)$ qui sera vraie ssi M vérifie I.

Existe-t-il d'autres problèmes *NP*-Complets ?

Prouver que Stable est *NP*-complet

Problème du Stable (rappel)

Instance :

- $G = (V, E)$ un graphe ;
- $k \in \mathbb{N}$

Question : existe-t-il un stable S de taille au moins k ?

Comment montrer que Stable est *NP*-complet ?

Prouver que Stable est *NP*-complet

Problème du Stable (rappel)

Instance :

- $G = (V, E)$ un graphe ;
- $k \in \mathbb{N}$

Question : existe-t-il un stable S de taille au moins k ?

Comment montrer que Stable est *NP*-complet ?

- Montrer qu'il est dans *NP* (fait)

Prouver que Stable est *NP*-complet

Problème du Stable (rappel)

Instance :

- $G = (V, E)$ un graphe ;
- $k \in \mathbb{N}$

Question : existe-t-il un stable S de taille au moins k ?

Comment montrer que Stable est *NP*-complet ?

- Montrer qu'il est dans *NP* (fait)
- Montrer qu'il est *NP*-difficile ?

Prouver que Stable est *NP*-complet

Problème du Stable (rappel)

Instance :

- $G = (V, E)$ un graphe ;
- $k \in \mathbb{N}$

Question : existe-t-il un stable S de taille au moins k ?

Comment montrer que Stable est *NP*-complet ?

- Montrer qu'il est dans *NP* (fait)
- Montrer qu'il est *NP*-difficile ?
  Pour cela, on peut se limiter à réduire un problème NP-complet à Stable car la réduction polynomiale est transitive.

SAT $\leq$ Stable

Considérons les k clauses d'une FNC :

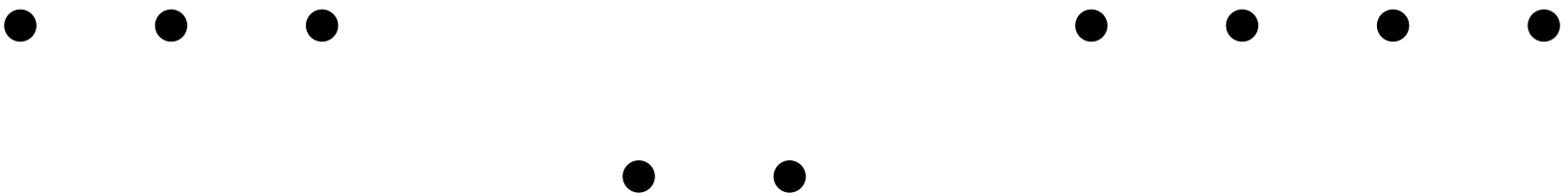
$$(U_1 \vee U_2 \vee \neg U_3) \quad \bigwedge \quad (\neg U_1 \vee \neg U_4) \quad \bigwedge \quad (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$

Méthode

SAT $\leq$ Stable

Considérons les k clauses d'une FNC :

$$(U_1 \vee U_2 \vee \neg U_3) \quad \bigwedge \quad (\neg U_1 \vee \neg U_4) \quad \bigwedge \quad (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$



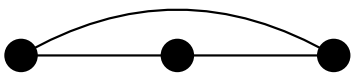
Méthode

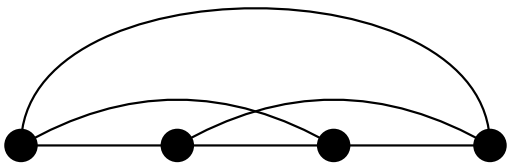
1. 1 sommet pour chaque occurrence de variable dans une clause

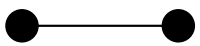
SAT $\leq$ Stable

Considérons les k clauses d'une FNC :

$$(U_1 \vee U_2 \vee \neg U_3) \quad \bigwedge \quad (\neg U_1 \vee \neg U_4) \quad \bigwedge \quad (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$







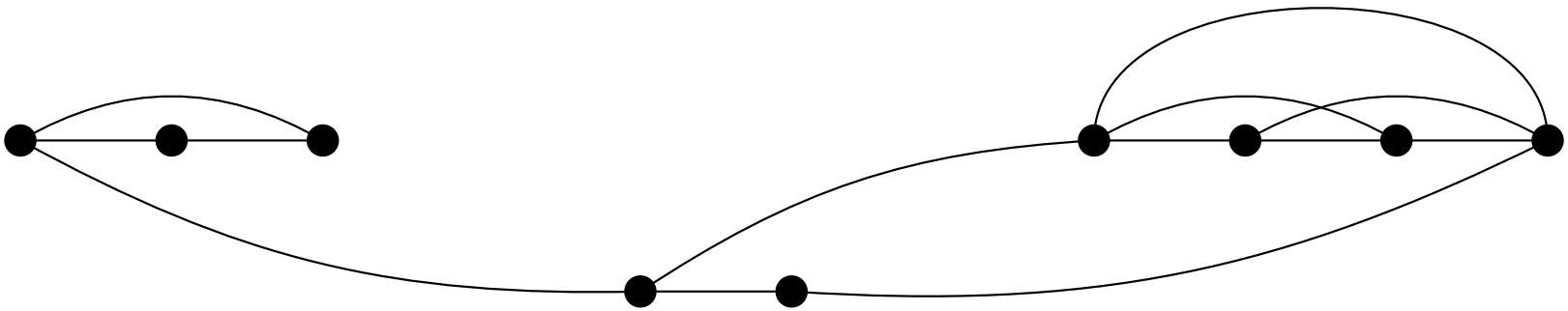
Méthode

1. 1 sommet pour chaque occurrence de variable dans une clause
2. Relier entre eux les sommets d'une même clause

SAT $\leq$ Stable

Considérons les k clauses d'une FNC :

$$(U_1 \vee U_2 \vee \neg U_3) \quad \wedge \quad (\neg U_1 \vee \neg U_4) \quad \wedge \quad (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$



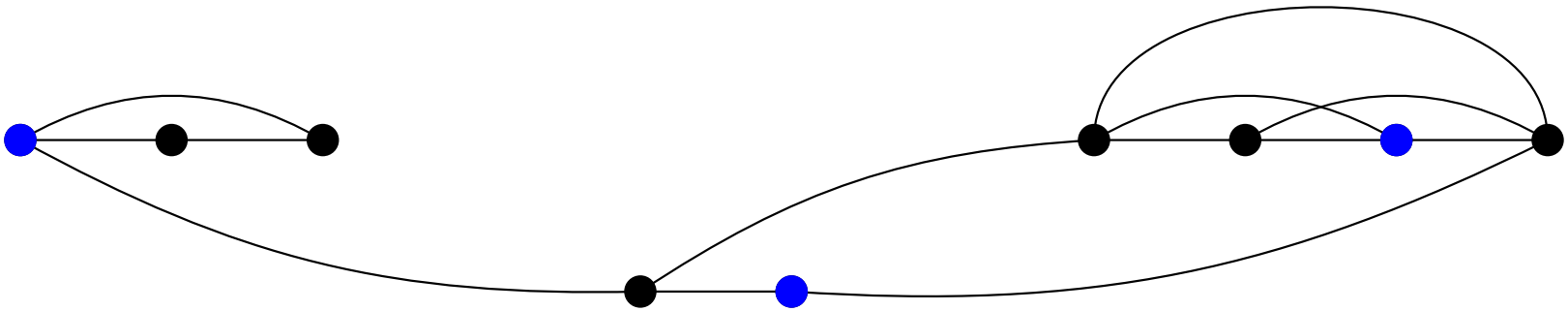
Méthode

1. 1 sommet pour chaque occurrence de variable dans une clause
2. Relier entre eux les sommets d'une même clause
3. Relier les occurrences positives et les occurrences négatives

SAT $\leq$ Stable

Considérons les k clauses d'une FNC :

$$(U_1 \vee U_2 \vee \neg U_3) \quad \wedge \quad (\neg U_1 \vee \neg U_4) \quad \wedge \quad (U_1 \vee U_2 \vee \neg U_5 \vee U_4)$$



- satisfiable $\Rightarrow$ stable de taille k ou plus
  - ➜ chaque clause est satisfaite.
  - ➜ on peut construire un stable de taille k en sélectionnant un littéral vrai dans chaque clause.
- stable de taille k $\Rightarrow$ satisfiable
  - ➜ chaque sommet du stable correspond à un littéral satisfaisant une clause différente.
  - ➜ par construction, le stable ne contient pas deux sommets dont l'un correspondant à une variable et l'autre à sa négation.

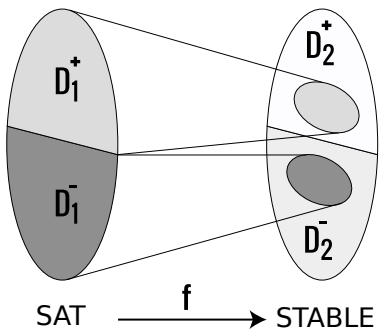
SAT $\leq$ Stable

Conclusion

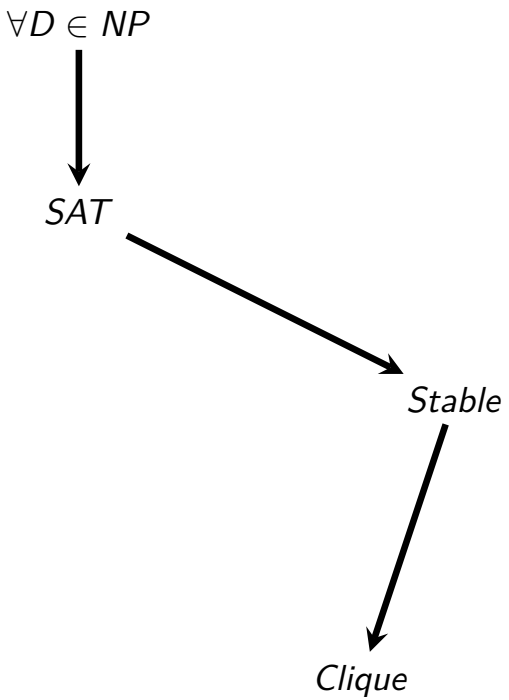
Nous avons défini une réduction f qui pour toute instance I_{SAT} de SAT :

- crée une instance de Stable $I_{Stable} = f(I_{SAT})$
- I_{SAT} est positive $\Rightarrow$ I_{Stable} est positive
- I_{SAT} est négative $\Rightarrow$ I_{Stable} est négative
  - car I_{Stable} est positive $\Rightarrow$ I_{SAT} est positive
- f est polynomiale en temps

➜ SAT $\leq$ Stable



Le réseau des réductions entre les problèmes *NP*-complets



⏩ skip next reduction

Le problème du circuit hamiltonien

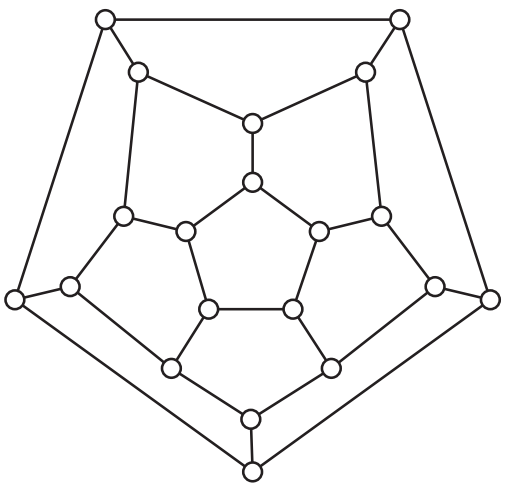
D-HAM (Directed HAM)

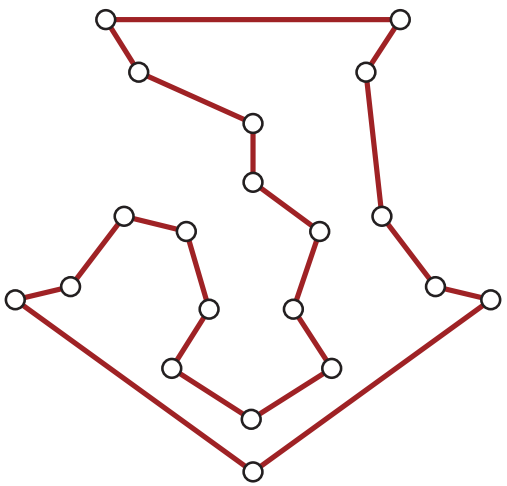
Instance :

- $G = (V, A)$ un graphe orienté

Question : existe-t-il un circuit hamiltonien, c'est-à-dire passant une et une seule fois par tous les sommets du graphe ?

Dans cet exemple, tel que présenté à l'origine par Lord Hamilton, le graphe est non orienté.





D-HAM est *NP*-complet

Premièrement : D-HAM $\in$ *NP*

Étant donné une instance de D-HAM (un graphe $G = (V, A)$) et un circuit C, il est possible de vérifier en temps polynomial si C est un circuit hamiltonien ou non. Le problème D-HAM appartient donc à la classe *NP*.

D-HAM est *NP*-complet

Premièrement : D-HAM $\in$ *NP*

Étant donné une instance de D-HAM (un graphe $G = (V, A)$) et un circuit C, il est possible de vérifier en temps polynomial si C est un circuit hamiltonien ou non. Le problème D-HAM appartient donc à la classe *NP*.

Deuxièmement : une réduction polynomiale SAT $\leq$ D-HAM

Comment réduire SAT en D-HAM ?

SAT $\leq$ D-HAM

Soit I une instance de SAT avec les variables $x_1, \ldots, x_n$ et les clauses $C_1, \ldots, C_k$.

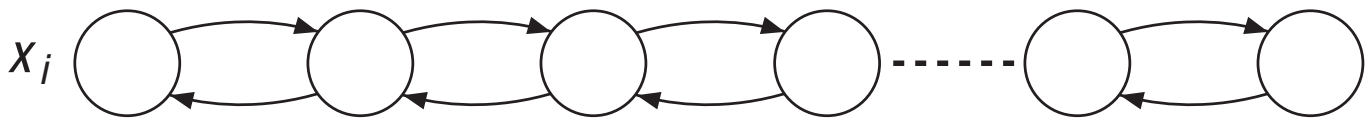
$$(x_1 \vee x_2) \wedge (\overline{x_3} \vee x_4 \vee \overline{x_5}) \wedge \ldots \wedge (\overline{x_1})$$

Idée de la réduction

- Créer des structures à base de graphe pour représenter les variables et les clauses.
- Organiser les structures ensemble pour encoder la formule.
- Montrer que la structure admet un circuit hamiltonien ssi la formule est satisfiable.

SAT $\leq$ D-HAM

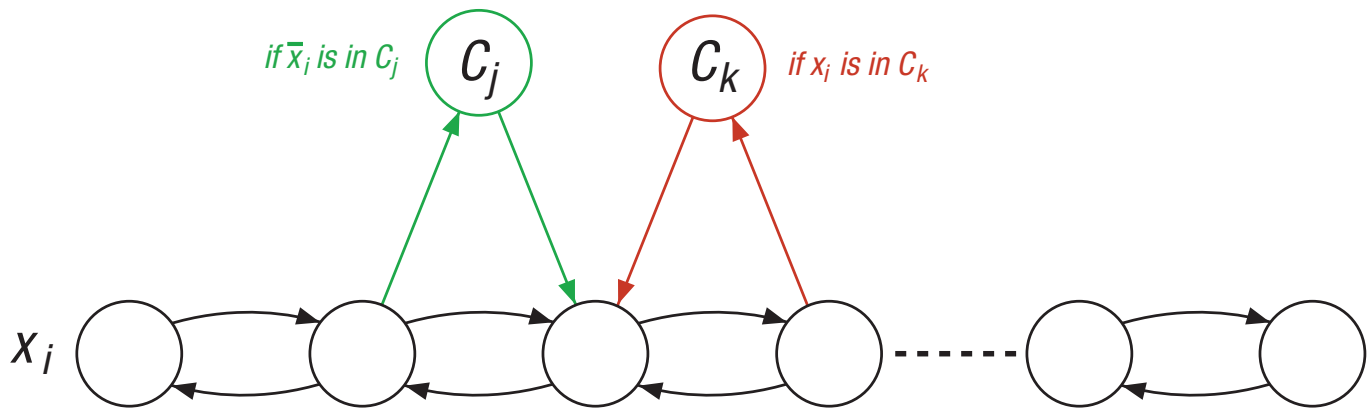
Pour chaque variable x_i, on crée une structure de la forme suivante.



Elle devra être traversée par le circuit hamiltonien. Par convention, si elle est traversée de gauche à droite, on affectera *faux* à la variable correspondante, sinon on affectera *vrai* à la variable.

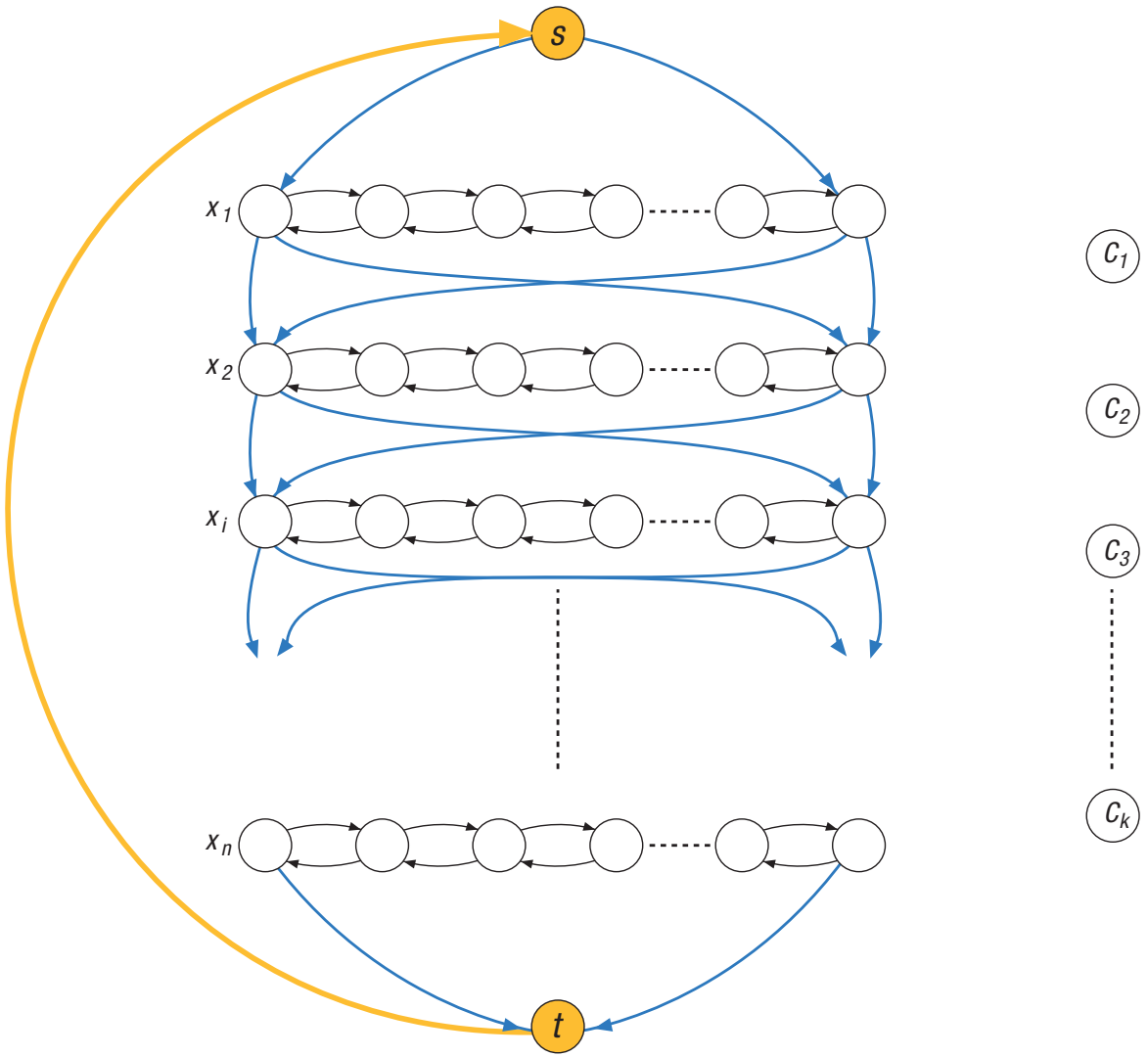
SAT $\leq$ D-HAM

On ajoute un nouveau nœud pour chaque clause et on le relie aux structures existantes.

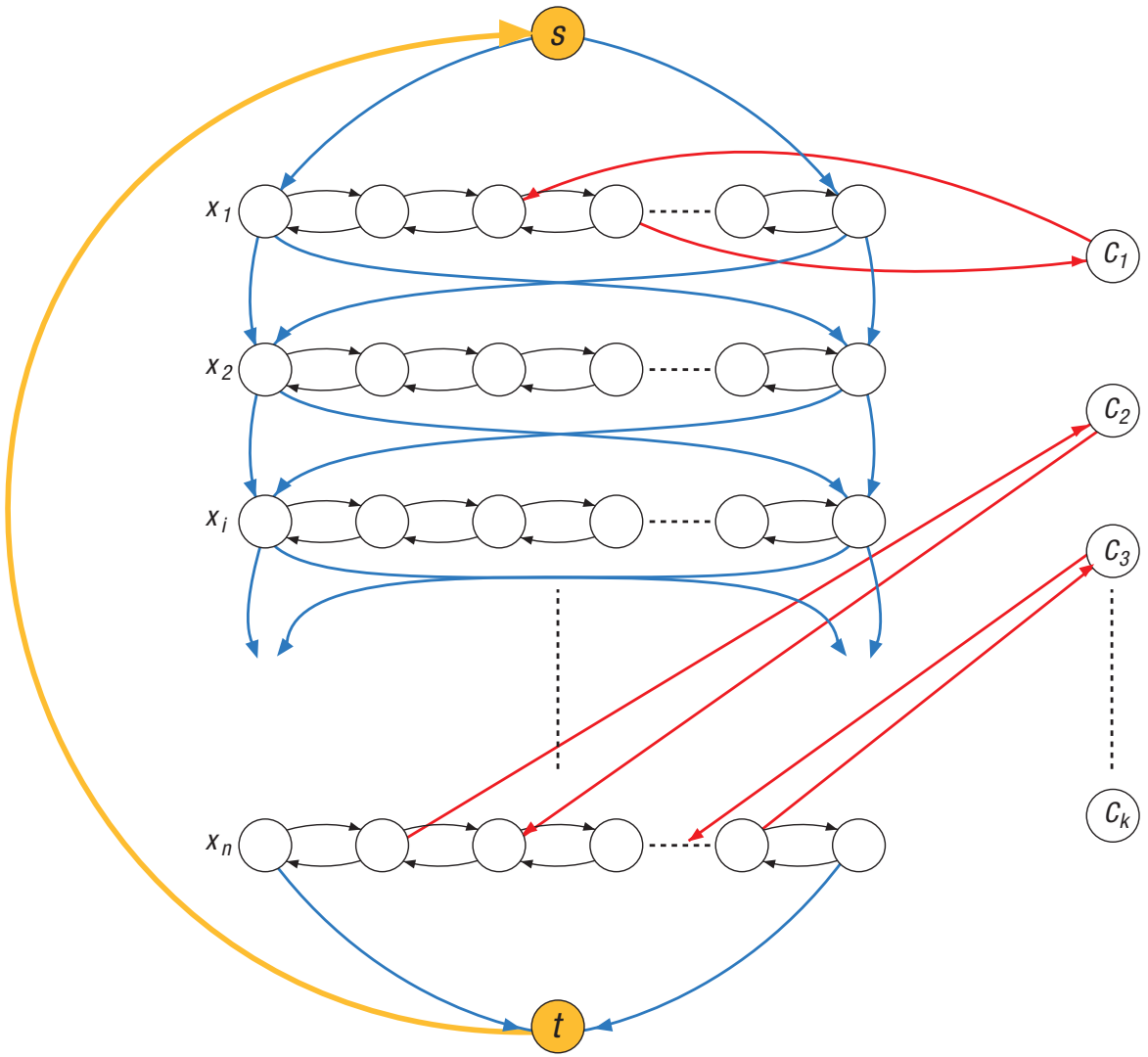


Pour chaque variables, les structures doivent être suffisamment longues pour placer les clauses (au moins 3+k noeuds). Le nombre total de noeuds (3n+kn) reste polynomial en fonction de la taille de la formule SAT.

SAT $\leq$ D-HAM



SAT $\leq$ D-HAM



SAT $\leq$ D-HAM

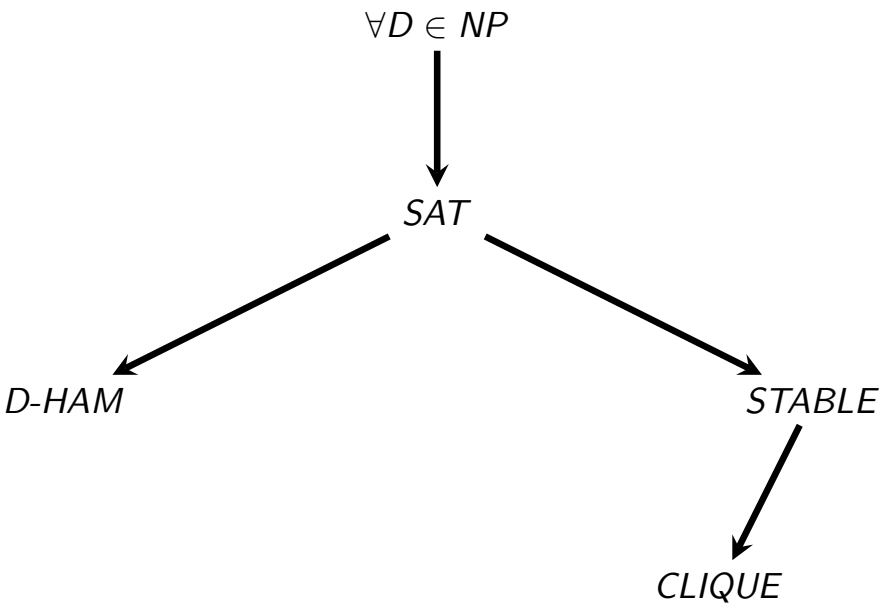
Supposons qu'il existe un circuit hamiltonien

- Ce circuit hamiltonien encode les affectations pour les variables (en fonction de la direction dans laquelle chaque structure est traversée).
- Pour que le circuit soit hamiltonien, il doit visiter toutes les clauses.
- On ne peut visiter les clauses qu'en les satisfaisant (en mettant un de ses termes à *vrai*).
- Donc s'il existe un circuit hamiltonien, il existe une affectation qui satisfait la formule.

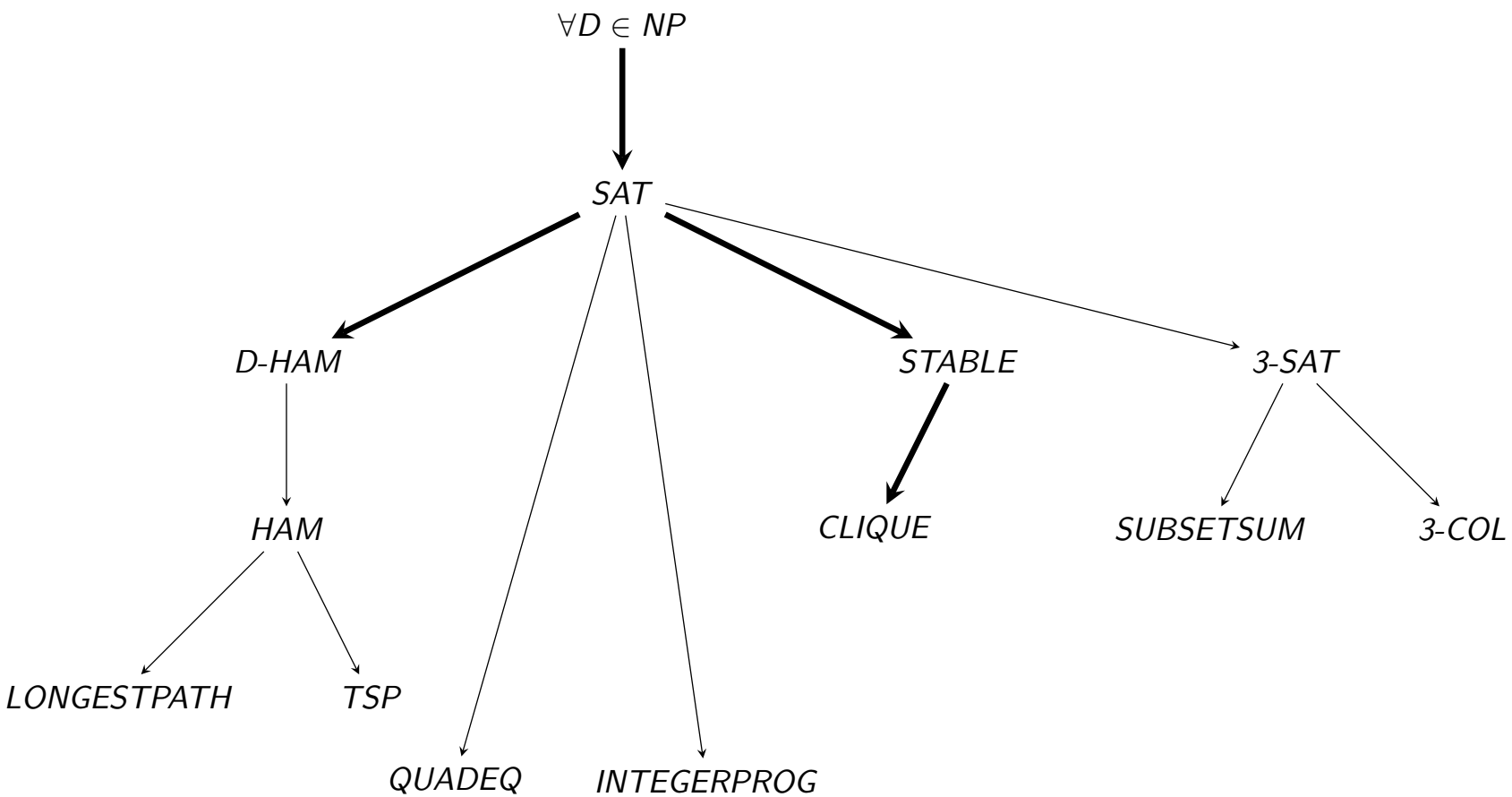
Réciproque

Une affectation décrit un parcours (attention à ne pas repasser par des clauses déjà satisfaites)

Le réseau des réductions entre les problèmes *NP*-complets



Le réseau des réductions entre les problèmes *NP*-complets



Plan

On a donc maintenant classé l'ensemble des problèmes ?

La classe *NP* ne regroupe que les problèmes de décision de la forme $D = D^+ \cup D^-$ pour lesquels il existe un certicat pour les instances $I \in D^+$...

Autres classes de complexité : *coNP*

coNP

- La classe *NP* regroupe les problèmes pour lesquels valider une solution est polynomial.
- La classe *coNP* regroupe les problèmes pour lesquels valider un contre-exemple est polynomial.
- On suppose $coNP \neq NP$
- De manière similaire, il existe des problèmes *coNP*-complets.
- $P \subseteq NP \cap coNP$

Exemples de problèmes dans *coNP*

Tautologie

- Soit une formule booléenne quelconque, on cherche à montrer qu'elle est vraie pour toute affectation.
- Il est simple d'exhiber une affectation prouvant que cela n'est pas vrai mais difficile de montrer que cela est vrai pour toutes.

Exemples de problèmes dans *coNP*

Tautologie

- Soit une formule booléenne quelconque, on cherche à montrer qu'elle est vraie pour toute affectation.
- Il est simple d'exhiber une affectation prouvant que cela n'est pas vrai mais difficile de montrer que cela est vrai pour toutes.

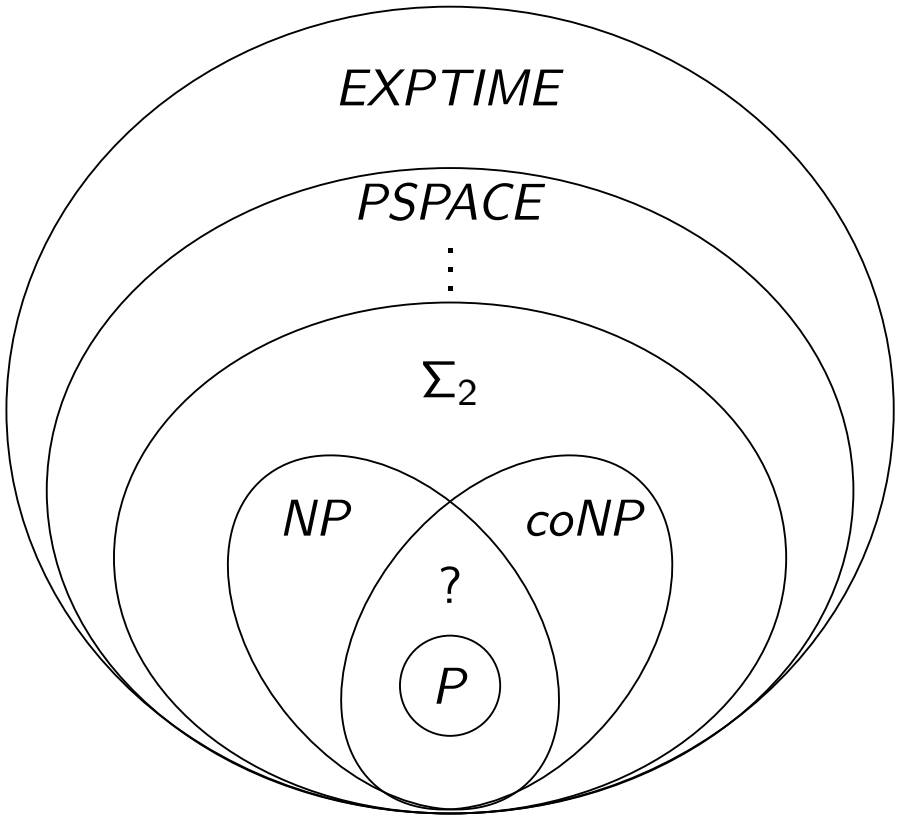
Primalité est dans *P*

- Tester la primalité d'un nombre est dans *coNP*. Si un nombre n'est pas premier, on peut vérifier en temps polynomial le certificat (un facteur).
- Exhiber un certificat montrant qu'il n'existe pas de facteurs semble plus délicat. Mais depuis 2002, la primalité est dans *P*.

Et maintenant, on a donc classé l'ensemble des problèmes ?

Quid des problèmes pour lesquels vérifier le certificat ne serait pas dans P ?

Hiérarchie des classes de complexité



Examples

- Σ_2 : pour une formule booléenne $\phi(x, y)$, satisfaire $\exists x \forall y \phi(x, y)$
- *PSPACE* : Othello (Reversi), QBF
- *EXPTIME* : Chess, Go

Ce qu'il faut retenir

- Définitions des classes P et NP
- Définition de la réduction polynomiale
- Mise en œuvre de la réduction polynomiale sur des problèmes simples (voir TD 5 et suivants)
- Problèmes classiques (SAT, Stable, HAM)
  - ✗ Ne pas retenir les réductions...
  - ✓ ... mais vous devez les comprendre et pouvoir les expliquer !