

# Algorithmes du monde réel

C. Gavoille, A. Muscholl, M. Zeitoun, A. Zvonkine

Master 2, Univ. Bordeaux 1, 2008–2009

## Modalités du cours

- ▶ 12 cours, 3 groupes de TD (débutent semaine du 22/9/08).
- ▶ C. Gavoille, A. Muscholl, M. Zeitoun, S. Zvonkine.

## Objectifs

1. **Identifier** les **problèmes** n'ayant pas d'algorithme efficace : distinguer les problèmes de complexité **polynomiale** de ceux qui ont une complexité (probablement) plus élevée.
2. **Développer** des **algorithmes** résolvant efficacement ces problèmes.

### ↪ Concessions

- ▶ Heuristiques,
- ▶ Algorithmes d'approximation,
- ▶ Algorithmes probabilistes,
- ▶ Algorithmes sur des classes restreintes mais utiles de données,
- ▶ Semi-algorithmes, sans garantie de terminaison.
- ▶ ...

## Objectifs

- ▶ Il ne s'agit pas d'un second cours sur la calculabilité.
- ▶ Les notions de théorie de la complexité (1<sup>er</sup> cours) serviront à distinguer les problèmes difficiles sur lesquels on travaillera.
- ▶ L'essentiel du cours se concentre ensuite sur les algorithmes.

## Plan

Problèmes difficiles : NP-complétude

## Complexité en temps des algorithmes

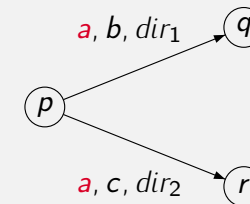
- ▶ Complexité d'un **algorithme** : borne **supérieure** sur le temps d'exécution.
- ▶ Fonction qui à l'entier  $n$  associe le nombre d'opérations élémentaires effectuées par l'algorithme
  - ▶ dans le cas le pire,
  - ▶ sur une donnée de **taille**  $n$ .

## Complexité en temps des algorithmes

- ▶ Formellement, on la définit grâce aux machines de Turing (MT).
- ▶ Les machines de Turing considérées s'arrêtent sur toute entrée.

## Rappels : machines de Turing non-déterministes

- ▶ Une machine de Turing est **non déterministe** s'il est parfois possible de choisir entre plusieurs transitions à appliquer.



Si dans l'état  $p$  on lit  $a$  : 2 transitions sont possibles.

- ▶ Une MT non déterministe peut avoir **plusieurs** calculs sur un même mot  $\rightsquigarrow$  arbre de calcul.
- ▶ Ce n'est pas un modèle de calcul réaliste.
- ▶ Le **temps de calcul** d'une machine non déterministe sur l'entrée  $x$  est la longueur du **plus long** calcul sur  $x$ .

## Complexité en temps

- ▶  $M$  : machine de Turing, déterministe ou non.
- ▶ Le temps du calcul  $\gamma = q_0x \rightarrow C_1 \rightarrow C_2 \rightarrow \dots \rightarrow C_n$  est  $n$ .
- ▶ Soit  $\text{step}_M(x)$  le **temps maximal** d'un calcul de  $M$  sur l'entrée  $x$ .
- ▶ La **complexité** de  $M$  est la fonction

$$f_M(n) = \max\{\text{step}_M(w) \mid |w| = n\}.$$

- ▶  $f_M(n)$  mesure le temps passé par  $M$  sur une entrée de taille  $n$ , **dans le pire des cas**.

## Complexité des problèmes : classe P

- ▶ Classe des problèmes ayant un algorithme polynomial : **PTIME**, ou **P**.
- ▶ Un problème de décision est dans **P** s'il existe une machine de Turing
  1. **déterministe**,
  2. qui sur une instance positive, s'arrête sur OK,
  3. qui sur une instance négative, s'arrête sur KO,
  4. dont le temps de calcul est polynomial en fonction de la taille de l'entrée.
- ▶ Exemple : algorithme naïf pour tester la primalité d'un entier codé en binaire  $\rightsquigarrow$  exponentiel.
- ▶ Peut-on faire mieux?

## Complexité des problèmes : classe NP

- ▶ Un problème de décision est dans **NP** s'il existe une machine de Turing
  1. **déterministe ou non**,
  2. qui sur une instance positive, a **au moins** un calcul s'arrêtant sur OK.
  3. qui sur une instance négative, a **tous** ses calculs s'arrêtant sur KO.
  4. dont le temps de calcul est polynomial en fonction de la taille de l'entrée.
- ▶ Exemple : algorithme naïf pour tester la **non**-primalité d'un entier codé en binaire  $\rightsquigarrow$  polynomial, mais non-déterministe.

## Les classes P et NP

$$\text{NP} = \{\mathcal{L}(M) \mid \exists p \text{ polynôme tel que } f_M(n) \leq p(n)\}.$$

$$\text{P} = \{\mathcal{L}(M) \mid \exists p \text{ polynôme tel que } f_M(n) \leq p(n) \text{ et } M \text{ déterministe}\}.$$

- ▶ On a bien sûr  $\text{P} \subseteq \text{NP}$ .
- ▶ Déterminer si cette inclusion est stricte est un problème ouvert.
- ▶ **Certains des problèmes de NP sont considérés très difficiles. Ce sont les problèmes NP-complets dont on va voir des exemples.**

## Les classes P et NP

- ▶ Pour montrer qu'un problème est NP, on utilise souvent
  - ▶ une phase non-déterministe, où la machine **devine** le résultat.
  - ▶ une phase déterministe, où elle **vérifie** si ce résultat est **correct**.
- ▶ Ex : machine non-déterministe testant si un entier  $n$  n'est pas premier.
  - ▶ On devine un diviseur  $k$  de  $n$ .
  - ▶ Si  $2 \leq k \leq n - 1$  et  $k|n$  : OK.
  - ▶ Sinon : KO.
- ▶ Fonctionne en temps polynomial, donc le problème est dans NP.

## Les classes P et NP

- ▶ L'appartenance à P ne dépend pas du nombre de bandes utilisées.

### Théorème

Si  $L$  est décidé par une MT à  $k$  bandes déterministe en temps  $f(n)$ , alors  $L$  est décidé par une MT usuelle en  $O(f^2(n))$ .

- ▶ On peut passer d'une MT non déterministe à une MT déterministe au prix d'une **exponentielle** en complexité en temps.

### Théorème

Si  $L$  est décidé par une MT non-déterministe en temps  $f(n)$ , alors  $L$  est décidé par une MT usuelle en temps  $2^{O(f(n))}$ .

## Une autre formulation de NP

- P** problèmes pour lesquels une solution peut être **déterminée** en temps polynomial.
- NP** problèmes pour lesquels une solution peut être **vérifiée** en temps polynomial.

## Certificats et vérificateurs

- ▶ Un **vérificateur en temps polynomial** pour un langage  $L$  est une machine **déterministe**  $M$  acceptant des couples  $(w, c)$ , telle que
  1.  $L = \{w \mid \exists c, (w, c) \in L(M)\}$ , et
  2.  $M$  a un temps de calcul polynomial en  $|w|$  sur l'entrée  $(w, c)$ .
- ▶ La partie  $c$  du couple peut être vue comme un certificat, une preuve que l'instance  $w$  est bien dans  $L$ .

### Théorème

Un langage  $L$  est dans NP si et seulement s'il existe un vérificateur en temps polynomial pour  $L$ .

## Problèmes NP : Exemples

- ▶ SAT.
- ▶ 3-SAT.
- ▶ 3-coloration.
- ▶ Voyageur de commerce.
- ▶ Chemin Hamiltonien.
- ▶ Couverture de sommets.
- ▶ Clique.
- ▶ Ensemble indépendant.
- ▶ Somme d'entiers.
- ▶ Partition.
- ▶ Mariages à 3 dimensions.

## Optimisation vs. décision

Parfois, on a besoin d'évaluer la complexité de problèmes d'optimisation. À chaque solution du problème est associé un coût.

**Optimisation** Trouver la solution de coût optimal.

**Décision** Existe-t-il une solution de coût  $\leq k$  ?

- ▶ **Exemple** Voyageur de commerce.
- ▶ La solution du problème d'optimisation donne une réponse immédiate au problème de décision.
- ▶ Inversement, on peut en général se ramener du problème de décision au problème d'optimisation par recherche dichotomique.

## Réductions polynomiales

- ▶ Soient  $A$  et  $B$  deux langages de  $\Sigma^*$ .
- ▶ En pratique,  $A$  et  $B$  = instances positives de 2 problèmes  $P_A, P_B$ .
- ▶ Une **réduction polynomiale** de  $A$  vers  $B$  est une fonction  $f : \Sigma^* \rightarrow \Sigma^*$  telle que
  - ▶  $f$  est **calculable par une MT déterministe en temps polynomial**.
  - ▶ On a l'équivalence suivante :

$$x \in A \iff f(x) \in B.$$

- ▶ On note  $A \leq_P B$  ( $A$  se réduit à  $B$  de façon polynomiale).
- ▶ L'existence d'une réduction de  $A$  vers  $B$  assure que
  - ▶ si  $B \in P$ , alors  $A \in P$ .
  - ▶ si  $B \in NP$ , alors  $A \in NP$ .

## Réductions polynomiales

- ▶ Si  $A \leq_P B$  et si  $B$  est dans  $P$ , alors  $A$  aussi.
- ▶ En effet, par définition de  $\leq_P$ , il existe  $f : \Sigma^* \rightarrow \Sigma^*$  telle que
  - ▶  $f$  est **calculable par une MT déterministe en temps polynomial**, et

$$x \in A \iff f(x) \in B.$$

- ▶ Algorithme polynomial pour tester si  $x \in A$  :
  1. calculer  $f(x)$  (polynomial par hypothèse),
  2. tester si  $f(x) \in B$ , polynomial en  $|f(x)|$  puisque  $B \in P$ , donc en  $|x|$ .

## Problèmes NP complets

- ▶ Un langage  $L$  (ou problème) est **NP-complet** si
  1.  $L \in \text{NP}$ , et
  2. pour tout langage  $K \in \text{NP}$ , on a  $K \leq_P L$ .
- ▶ Intuitivement : **NP-complets** = problèmes NP « les plus durs ».
- ▶ On va montrer qu'il existe des problèmes NP-complets.
- ▶ Aucun algorithme polynomial connu pour aucun problème NP-complet.
- ▶ Si **un** problème NP-complet avait un algorithme polynomial, alors **tous** auraient aussi un algorithme polynomial.

## Notion de problème NP-complet : utilité

- ▶ Lorsqu'on montre qu'un problème est NP-complet, cela signifie qu'il est peu probable qu'on trouvera un algorithme efficace (polynomial).
- ▶ On cherche plutôt des algorithmes approchés (heuristiques), probabilistes, ou à complexité paramétrique faible.

## Théorème de Cook-Levin

- ▶ Le problème **SAT** est le suivant :
  - ▶ **Donnée** : une formule Booléenne sur des variables  $\{x_1, x_2, \dots, \}$  donnée comme conjonction de clauses.
  - ▶ **Question** : existe-t-il une assignation de chaque variable  $x_i$  par **vrai** ou **faux** qui rend la formule vraie ?
- ▶ **Théorème (Cook, Levin)** **SAT** est NP-complet.

## Théorème de Cook-Levin — principe de la preuve

- ▶ On part d'un langage quelconque de NP, décidé par une MT  $M$ .
- ▶ On construit à partir de  $M$  et d'un mot d'entrée  $w$  une formule  $\varphi_{M,w}$  de taille polynomiale, et telle que

$\varphi_{M,w}$  est satisfaisable  $\iff M$  accepte  $w$ .

## Théorème de Cook-Levin — idée de preuve

- ▶ Tout calcul de  $M$  sur  $w$  prend  $p(n)$  étapes, où  $p$  est un polynôme et  $n = |w|$ .
- ▶ Donc tout calcul de  $M$  sur  $w$  utilise au plus  $p(n) + 1$  cases.
- ▶ On introduit des variables, avec leur signification intuitive
  - ▶  $Q(i, k)$  : vrai ssi l'état à l'instant  $i$  est  $q_k$ .
  - ▶  $P(i, j)$  : vrai ssi la position de la tête de lecture à l'instant  $i$  est  $j$ .
  - ▶  $L(i, j, a)$  : vrai ssi la lettre à l'instant  $i$  dans la case  $j$  est  $a$ .
- ▶ **Note.** Seulement un nombre **polynomial** de variables !

## Théorème de Cook-Levin — idée de preuve

- ▶ Grâce aux variables  $Q, P, L$ , on encode le calcul : À tout instant
  - ▶ on se trouve dans un et un seul état,
  - ▶ une et une seule case est lue,
  - ▶ il y a une et une seule lettre dans chaque case.
- ▶ On code également que
  - ▶ au temps 0, la configuration est  $q_0 w$ .
  - ▶ au temps  $p(n)$ , la machine est dans l'état  $q_{OK}$ .
  - ▶ entre le temps  $i$  et le temps  $i + 1$ , on applique une transition.

## 3-SAT

Le problème **3-SAT** est le suivant :

- ▶ **Donnée** : une formule Booléenne sur des variables  $\{x_1, x_2, \dots\}$  donnée comme conjonction de **3-clauses**.
- ▶ **Question** : existe-t-il une assignation de chaque variable  $x_i$  par **vrai** ou **faux** qui rend la formule vraie ?

Une **3-clause** est une clause avec exactement 3 littéraux.

**Exemple**  $x \vee \neg y \vee z$ .

## Réduction SAT vers 3-SAT

3-SAT est bien dans **NP**. Réduction pour la NP-complétude :

- ▶ À toute instance  $\varphi$  de SAT, on associe une instance  $\tilde{\varphi}$  de 3-SAT tq.  
 $\varphi$  est satisfaisable  $\iff \tilde{\varphi}$  est satisfaisable  
et tq. on peut construire  $\tilde{\varphi}$  en temps polynomial par rapport à  $|\varphi|$ .

## Réduction SAT vers 3-SAT

Si  $\varphi = c_1 \wedge \dots \wedge c_k$  où chaque  $c_i$  est une clause, on construit  $\tilde{\varphi}$  de la forme  $\varphi_1 \wedge \dots \wedge \varphi_k$ , où

- ▶ Chaque  $\varphi_i$  est une conjonction de 3-clauses,
- ▶  $\varphi_i$  utilise les variables de  $c_i$ , + éventuellement de nouvelles variables.
- ▶ Si une affectation des variables rend  $c_i$  vraie, on peut la compléter pour rendre  $\varphi_i$  vraie.
- ▶ Inversement, si une affectation des variables rend  $\varphi_i$  vraie, sa restriction aux variables de  $c_i$  rend  $c_i$  vraie.

## Réduction SAT vers 3-SAT : construction de $\varphi_i$

- ▶ Si  $c_i = \ell_1$  (un littéral), on ajoute 2 variables  $y_i, z_i$  et
$$\varphi_i = (\ell_1 \vee y_i \vee z_i) \wedge (\ell_1 \vee \neg y_i \vee z_i) \wedge (\ell_1 \vee y_i \vee \neg z_i) \wedge (\ell_1 \vee \neg y_i \vee \neg z_i).$$
- ▶ Si  $c_i = \ell_1 \vee \ell_2$  (2 littéraux), on ajoute 1 variable  $y_i$  et
$$\varphi_i = (y_i \vee \ell_1 \vee \ell_2) \wedge (\neg y_i \vee \ell_1 \vee \ell_2).$$
- ▶ Si  $c_i$  est une 3-clause :  $\varphi_i = c_i$ .
- ▶ Si  $c_i = \ell_1 \vee \dots \vee \ell_k$  avec  $k \geq 4$ , on ajoute  $k - 3$  variables  $t_1, \dots, t_{k-3}$ 
$$\varphi_i = (t_1 \vee \ell_1 \vee \ell_2) \wedge (\neg t_1 \vee \ell_3 \vee \ell_2) \wedge (\neg t_2 \vee \ell_4 \vee \ell_3) \wedge \dots \wedge (\neg t_{k-3} \vee \ell_{k-1} \vee \ell_k)$$

## Réduction SAT vers 3-SAT

- ▶ Si une affectation des variables rend chaque  $c_i$  vraie, on la complète facilement pour rendre chaque  $\varphi_i$  vraie.
- ▶ Inversement, on vérifie que si une affectation des variables rend chaque  $\varphi_i$  vraie, la restriction de cette affectation aux variables de  $c_i$  rend  $c_i$  vraie.
- ▶ Donc  $\varphi$  est satisfaisable  $\iff \tilde{\varphi}$  est satisfaisable.
- ▶ Par ailleurs, la taille de  $\tilde{\varphi}$  est linéaire par rapport à celle de  $\varphi$ , et on construit  $\tilde{\varphi}$  en temps polynomial.

## Chemin Hamiltonien dans un graphe orienté

Le problème **Chemin Hamiltonien** dans un graphe orienté est le suivant :

- ▶ **Donnée** : un graphe orienté  $G$ .
- ▶ **Question** : existe-t-il dans  $G$  un chemin Hamiltonien, c'est-à-dire qui visite une et une seule fois chaque sommet ?

Ce problème est bien dans NP : on devine un chemin de  $|V(G)|$  sommets et on vérifie qu'il est Hamiltonien.

## Réduction 3-SAT vers Chemin Hamiltonien

- ▶ À toute instance  $\varphi$  de 3-SAT, on associe une instance  $G_\varphi$  de Chemin Hamiltonien tq.

$\varphi$  est satisfaisable  $\iff G_\varphi$  a un chemin Hamiltonien.

et tq. on peut construire  $G_\varphi$  en temps polynomial par rapport à  $|\varphi|$ .

Exemple :  $(x_0 \vee x_1 \vee x_2) \wedge (x_0 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3)$ .

## Chemin Hamiltonien dans un graphe non orienté

Le problème **Chemin Hamiltonien** dans un graphe orienté est le suivant :

- ▶ **Donnée** : un graphe orienté  $G$ .
- ▶ **Question** : existe-t-il dans  $G$  un chemin Hamiltonien, c'est-à-dire qui visite une et une seule fois chaque sommet ?

Ce problème est bien dans NP : on devine un chemin de  $|V(G)|$  sommets et on vérifie qu'il est Hamiltonien.

## Chemin Hamiltonien dans un graphe non orienté

On réduit le problème **Chemin Hamiltonien orienté** en **Chemin Hamiltonien non orienté** :

A tout graphe orienté  $G = (V, E)$ , on associe le graphe non orienté  $H = (S, A)$  avec  $S = V \times \{0, 1, 2\}$ , et comme arêtes :  $(v, 0) - (v, 1) - (v, 2)$ , et pour tout arc  $v \rightarrow w$  dans  $G$  :  $(v, 2) - (w, 0)$ .

On vérifie :

Il existe un chemin Hamiltonien dans  $G$

$\iff$

Il existe un chemin Hamiltonien dans  $H$ .

et  $H$  peut être construit en temps polynomial en  $|G|$ .

## 3-coloration

Le problème **3-coloration** est le suivant :

- ▶ **Donnée** : un graphe non orienté  $G$ .
- ▶ **Question** : existe-t-il une 3-coloration de  $G$  ?

À nouveau, le problème est dans la classe NP.

## Réduction 3-SAT vers 3-coloration

- ▶ À toute instance  $\varphi$  de 3-SAT, on associe une instance  $G_\varphi$  de 3-coloration tq.  
 $\varphi$  est satisfaisable  $\iff G_\varphi$  admet une 3-coloration.  
et tq. on peut construire  $G_\varphi$  en temps polynomial par rapport à  $|\varphi|$ .

On utilise des sous-graphes (appelés *gadgets*) pour coder

- ▶ les littéraux vrais dans une évaluation qui satisfait  $\varphi$ ,
- ▶ les opérateurs logiques  $\wedge$  et  $\vee$ .

## Couverture de sommets

Une **couverture de sommets** d'un graphe est un ensemble  $S$  de sommets tel que toute arête a l'une de ses extrémités dans  $S$ .

Le problème **Couverture de sommets** est le suivant :

- ▶ **Donnée** : un graphe non orienté  $G$  et un entier  $K > 0$ .
- ▶ **Question** : existe-t-il dans  $G$  une couverture de sommets de taille  $K$  ?

Ce problème est bien dans NP : on devine la couverture et on vérifie.

## Clique et ensemble indépendant

Dans un graphe  $G$  non orienté

- ▶ Une **clique** pour  $G$  est un ensemble de sommets tous reliés 2 à 2.
- ▶ Un **ensemble indépendant** pour  $G$  est un ensemble de sommets sans arête entre 2 d'entre eux.

Le problème **Clique** est le suivant :

- ▶ **Donnée** : un graphe  $G$  non orienté et un entier  $K > 0$ .
- ▶ **Question** : existe-t-il une clique de  $G$  de taille  $K$  ?

Le problème **Ensemble indépendant** est le suivant :

- ▶ **Donnée** : un graphe  $G$  non orienté et un entier  $K > 0$ .
- ▶ **Question** : existe-t-il un ensemble indépendant de  $G$  de taille  $K$  ?

## Clique et ensemble indépendant

### Lemme

Pour un graphe non orienté  $G = (S, A)$  et  $X \subseteq S$ , les propriétés suivantes sont équivalentes :

1.  $X$  est une couverture de sommets.
2.  $S \setminus X$  est un ensemble indépendant pour  $G$ .
3.  $S \setminus X$  est une clique pour le complément de  $G$ .

On vérifie que **Couverture de sommets**, **Clique** et **Ensemble indépendant** sont dans NP.

Lemme  $\implies$  si l'un d'eux est NP-complet, les autres aussi.

## Réduction 3-SAT vers clique

- ▶ À toute instance  $\varphi$  de 3-SAT, on associe une instance  $G_\varphi, K_\varphi$  de **Clique** tq.  
 $\varphi$  est satisfaisable  $\iff G_\varphi$  a une clique de taille  $K_\varphi$ .  
et tq. on peut construire  $G_\varphi, K_\varphi$  en temps polynomial par rapport à  $|\varphi|$ .

## Réduction 3-SAT vers clique

- ▶ Soit  $\varphi = (\ell_0 \vee \ell_1 \vee \ell_2) \wedge \dots \wedge (\ell_{3k-3} \wedge \ell_{3k-2} \wedge \ell_{3k-1})$ .
- ▶ Le graphe  $G_\varphi$  a  $3k$  sommets  $\ell_1, \dots, \ell_{3k}$ .
- ▶ Deux sommets  $\ell_i, \ell_k$  sont reliés si
  - ▶ ils ne proviennent pas de la même clause ( $i/3 \neq k/3$ ), et si
  - ▶ ils ne sont pas de la forme  $\ell, \neg\ell$ .
- ▶ On choisit l'entier  $K_\varphi$  égal à  $k$ .
- ▶ On vérifie que  $G_\varphi$  a une clique de taille  $K_\varphi$  ssi  $\varphi$  est satisfaisable.

## Somme d'entiers

Le problème **Somme d'entiers** est le suivant :

- ▶ **Donnée** : des entiers  $x_1, \dots, x_k > 0$  et un entier  $s$ .
- ▶ **Question** : existe-t-il  $1 \leq i_1 < i_2 < \dots < i_p \leq k$  tels que

$$x_{i_1} + \dots + x_{i_p} = s.$$

C'est clairement dans NP : on devine  $i_1, \dots, i_p$  et on teste.

On montre que c'est NP-complet par une réduction 3-SAT  $\leq$  Somme d'entiers.

## Partition

Le problème **Partition** est le suivant :

- ▶ **Donnée** : des entiers  $x_1, \dots, x_k > 0$ .
- ▶ **Question** : existe-t-il  $I \subseteq \{1, \dots, k\}$  tel que

$$\sum_{i \in I} x_i = \sum_{i \notin I} x_i.$$

C'est clairement dans NP : on devine  $I$  et on teste.

## Réduction Somme d'entiers vers Partition

Soit  $x_1, \dots, x_k, s$  une instance de Somme d'entiers. Soit  $x = \sum x_i$ . On construit (en temps polynomial) l'instance  $x_1, \dots, x_k, x - 2s$  de Partition.

- ▶ Si Somme d'entiers a une solution sur  $x_1, \dots, x_k, s$ , il existe  $I$  tel que  $\sum_{i \in I} x_i = s$ . Mais alors,  $\sum_{i \notin I} x_i = x - s = \sum_{i \in I} x_i + (x - 2s)$ . Partition a bien une solution sur l'instance construite.
- ▶ Inversement, si Partition a une solution sur  $x_1, \dots, x_k, x - 2s$ , il existe  $I$  tel que  $\sum_{i \notin I} x_i = \sum_{i \in I} x_i + (x - 2s)$ , d'où  $\sum_{i \in I} x_i = s$  : Somme d'entiers a donc une solution sur l'instance originale.