

# Algorithmique 2

L3 RI

## Table des matières

<b>1</b>	<b>NP-Complétude</b>	<b>1</b>
1.1	Définitions . . . . .	1
1.2	Satisfiabilité d'une formule . . . . .	2
1.3	Graphes . . . . .	2
<b>2</b>	<b>Algorithmes d'approximation</b>	<b>3</b>
2.1	Définitions . . . . .	3
2.2	Exemples . . . . .	3
<b>3</b>	<b>Algorithmes probabilistes</b>	<b>4</b>
3.1	Premiers exemples . . . . .	4
3.2	Classes de complexité des algorithmes probabilistes . . . . .	4
3.3	Déterminisation ou dérandomisation . . . . .	4
3.4	Schéma d'approximation probabiliste entièrement polynomial . . . . .	4
<b>4</b>	<b>Géométrie algorithmique</b>	<b>5</b>
4.1	Enveloppe convexe . . . . .	5
4.2	Points les plus rapprochés . . . . .	6
<b>5</b>	<b>Algorithmes distribués</b>	<b>6</b>
5.1	Généralités . . . . .	6
5.2	Problème du consensus . . . . .	7
5.3	Consensus dans le cas synchrone . . . . .	7
5.4	Consensus dans le cas asynchrone . . . . .	7

## 1 NP-Complétude

### 1.1 Définitions

**Réduction** Soient  $L_1, L_2$  des langages. Une réduction polynomiale de  $L_1$  à  $L_2$  est une fonction  $f$  calculable en temps polynomial telle que :

$$f(x) \in L_2 \iff x \in L_1$$

On note  $L_1 \leq L_2$  ( $L_1$  plus facile que  $L_2$  = Si on sait résoudre  $L_2$ , on sait résoudre  $L_1$ )

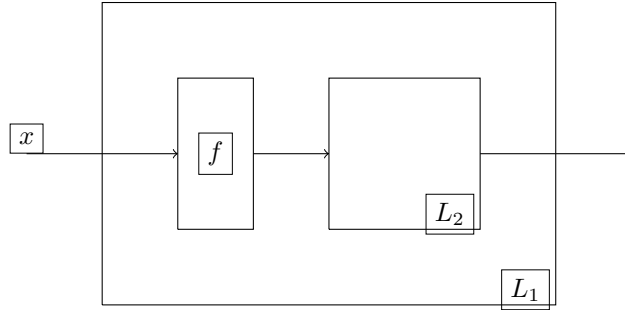


FIGURE 1 – Schéma de réduction de  $L_1$  à  $L_2$

**Classe NP** Soit  $L$  un langage.  $L$  est dit dans la classe NP s'il existe une machine de Turing non-déterministe en temps polynomial par rapport à la taille de l'entrée qui décide  $L$ .

**NP-Complétude** Soit  $L$  un langage.  $L$  est dit NP-dur si pour tout  $L' \in \text{NP}$ ,  $L' \leq L$ .  $L$  est dit NP-complet si  $L \in \text{NP}$  et  $L$  est NP-dur.

**Remarques** Si un problème NP-complet est dans P, alors  $P = \text{NP}$ . Pour montrer que  $L'$  est NP-dur, il suffit de montrer qu'il existe  $L$  NP-dur tel que  $L \leq L'$ .

## 1.2 Satisfiabilité d'une formule

### Problème de décision : SAT

*Entrée*  $\varphi$  formule de la logique propositionnelle

*Sortie* Oui si  $\varphi$  est satisfiable, c'est-à-dire s'il existe une valuation  $v$  des variables qui rend  $\varphi$  vraie

**Théorème de Cook** SAT est NP-complet. (*Réduction de  $L$  à SAT pour  $L \in \text{NP}$  en considérant une machine de Turing  $\mathcal{M}$  non-déterministe qui décide  $L$ . On construit une formule qui traduit une exécution de  $\mathcal{M}$ .*)

### Problème de décision : $i$ -SAT

Restriction de SAT à des formules en forme normale conjonctive (CNF) tel que chaque clause contient au plus  $i$  littéraux

**Théorème** 3-SAT est NP-complet. (*Réduction de SAT à 3-SAT.*)

**Théorème** 2-SAT est dans P. (*Réduction de 2-SAT à la détermination des CFC d'un graphe.*)

### Problème de décision : MAX-2-SAT

*Entrée*  $\varphi$  formule en 2-forme normale conjonctive et  $k \in \mathbb{N}$

*Sortie* Oui s'il existe une valuation qui satisfait au moins  $k$  clauses de  $\varphi$

**Théorème** MAX-2-SAT est NP-complet. (*Réduction de 3-SAT à MAX-2-SAT.*)

## 1.3 Graphes

### Problème de décision : ENS\_INDEP

*Entrée*  $G = (V, E)$  un graphe non-orienté et  $k \in \mathbb{N}$

*Sortie* Oui s'il existe  $V' \subseteq V$  tel que  $|V'| = k$  et  $(V' \times V') \cap E = \emptyset$

**Théorème** ENS\_INDEP est NP-complet. (*Réduction de 3-SAT à ENS\_INDEP.*)

### Problème de décision : MAX\_CUT

*Entrée*  $G = (V, E)$  un graphe non-orienté et  $k \in \mathbb{N}$

*Sortie* Oui s'il existe  $S_1$  et  $S_2$ ,  $S = S_1 \uplus S_2$  et  $\#\{(u, v) \in E \mid u \in S_1 \text{ et } v \in S_2\} \geq k$

**Théorème** MAX\_CUT est NP-complet. (*Réduction de MAX-2-SAT à MAX\_CUT.*)

## 2 Algorithmes d'approximation

### 2.1 Définitions

#### Problème de décision (DEC) $\neq$ Problème d'optimisation (OPT)

- Un problème OPT (par exemple «  $x$  tel  $f(x)$  minimal ») peut être traduit en un problème DEC (« Pour un  $b$  donné, existe-t'il  $x$  tel que  $f(x) \leq b$ ? »).
- OPT permet de résoudre DEC
- Souvent, DEC permet de résoudre OPT par dichotomie.

#### Instance d'un problème d'optimisation

- ensemble de solution admissibles
- coût pour chaque solution admissibles
- hypothèse : il y a une solution de coût optimal  $C^*$

**Garantie de performance** Un algorithme d'approximation a une garantie de performance  $\rho(n)$  (appelé facteur d'approximation) si

$$\frac{C(\text{solution calculée})}{C^*} \leq \rho(n)$$

pour un problème de minimisation, ou

$$\frac{C^*}{C(\text{solution calculée})} \leq \rho(n)$$

pour un problème de maximisation.

### 2.2 Exemples

#### Problème d'optimisation : VERTEX COVER

*Entrée*  $G = (V, E)$  un graphe non orienté

*Sortie*  $S \subset V$  de cardinal minimal tel que  $S$  couvre les arêtes de  $G$ , ie  $\forall (v, v') \in E, v \notin S \implies v' \in S$

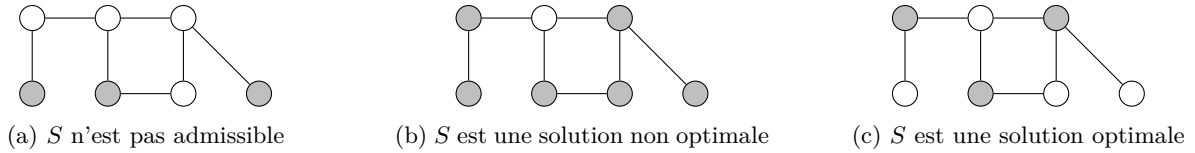


FIGURE 2 –  $S$  est constitué des nœuds grisés

**Théorème** Le problème de décision associé à VERTEX COVER est NP-complet. ( $VERTEX COVER \equiv ENS\_INDEP$ )

**Théorème** Un algorithme glouton permet d'obtenir une 2-approximation pour VERTEX COVER..

#### Problème d'optimisation : SET COVER

*Entrée*  $X$  ensemble fini d'éléments,  $\mathcal{F}$  une famille de sous ensembles de  $X$ .

*Sortie*  $S \subset \mathcal{F}$  de cardinal minimal tel que  $S$  couvre tout  $X$ , ie  $\forall x \in X, \exists \mathcal{E} \in S$  tel que  $x \in \mathcal{E}$

**Théorème** VERTEX COVER  $\leq$  SET COVER

**Théorème** Un algorithme glouton permet d'obtenir une  $\ln(n)$ -approximation pour VERTEX COVER.

#### Problème d'optimisation : VOYAGEUR DU COMMERCE

*Entrée*  $n$  villes, distances entre chaque ville

*Sortie* Une tournée qui minimise la distance parcourue.

**Théorème** Si  $P \neq NP$ , alors pour toutes constante  $\rho \geq 1$ , il n'existe pas d'algorithme d'approximation polynomial de garantie de performance  $\rho$  pour le voyageur de commerce.

## 3 Algorithmes probabilistes

Introduire de l'aléatoire pour :

- Réduire la complexité
- Simplifier les algorithmes.

Contrepartie : solution approchée (avec bonne proba d'être près de l'optimale) ou complexité bonne en moyenne mais pas dans le pire des cas.

### 3.1 Premiers exemples

**Random Quick Sort** Comme le tri rapide déterministe mais le pivot est choisi aléatoirement. La complexité est quadratique dans le pire des cas mais en  $\mathcal{O}(n \cdot \ln(n))$  en moyenne.

Pour le montrer, on calcule l'espérance de  $X_{i,j}$  qui vaut 1 si  $i$  et  $j$  sont comparées et 0 sinon. Si  $X$  est le temps d'exécution, on a :

$$\mathbb{E}(X) = \sum_{1 \leq i < j \leq n} \mathbb{E}(X_{i,j})$$

**Random Min Cut** **TODO**

### 3.2 Classes de complexité des algorithmes probabilistes

**ZPP : Zero Error Probabilistic Polynomial** Pour toute entrée de taille  $n$ , si il termine l'algo renvoie la réponse exacte et le temps moyen d'exécution est borné par  $P(n)$ , où  $P$  est un polynôme.

*Exemple* : Random quick sort.

**RP : Randomized Polynomial** Pour toute entrée de taille  $n$ , l'algo  $A$  s'exécute en temps borné par  $P(n)$ , où  $P$  est un polynôme, et si  $x \in L$ ,  $A$  répond « oui » avec probabilité  $\frac{1}{2}$ , sinon il répond non avec probabilité 1.

*Exemple* : Random min cut.

**Théorème**  $RP \subset NP$ .

**Théorème**  $P \subset ZPP$ .

**Théorème**  $ZPP = RP \cap \text{co-RP}$ .

*Rappel* :  $L \in \text{co-RP} \iff \Sigma^* \setminus L \in \text{RP}$ .

### 3.3 Déterminisation ou dérandomisation

**But** Passer d'un algorithme probabiliste à un algorithme déterministe en conservant les mêmes facteurs d'approximation.

### 3.4 Schéma d'approximation probabiliste entièrement polynomial

En anglais : Fully Polynomial Randomized Approximation (FPRA)

**Définition** Si  $k$  est une valeur,  $X$  est  $(\varepsilon, \delta)$ -approximation de  $k$  si  $\mathbb{P}(|X - k| \leq \varepsilon \cdot k) \geq 1 - \delta$ .

**Schéma d'approximation probabiliste entièrement polynomial** C'est un algorithme  $A$  qui prend  $\varphi, \varepsilon, \delta$  en entrée et calcule  $A(\varphi)$  une  $(\varepsilon, \delta)$ -approximation de la valeur recherchée, en temps polynomial en  $|\varphi|$ ,  $\log(\frac{1}{\varphi})$  et  $\frac{1}{\varepsilon}$ .

**Bornes de Chernoff** Soient  $0 < p, \delta, \varepsilon < 1$  et  $N \geq \frac{3 \cdot \log(\frac{2}{\delta})}{p \cdot \varepsilon^2}$ .

Si  $X_1, \dots, X_N$  sont des variables aléatoires indépendantes et à valeur dans  $\{0, 1\}$ , alors

$$\mathbb{P} \left( \left| \left( \frac{1}{N} \sum_{i=1}^N X_i \right) - p \right| \leq \varepsilon \cdot p \right) \geq 1 - \delta.$$

## 4 Géométrie algorithmique

### 4.1 Enveloppe convexe

#### Calcul de l'enveloppe convexe

*Entrée* Un ensemble  $S$  de points  $(x, y)$  du plan

*Sortie*  $C \subseteq S$  une énumération dans le sens trigonométrique des points extrémaux de l'enveloppe convexe de  $S$

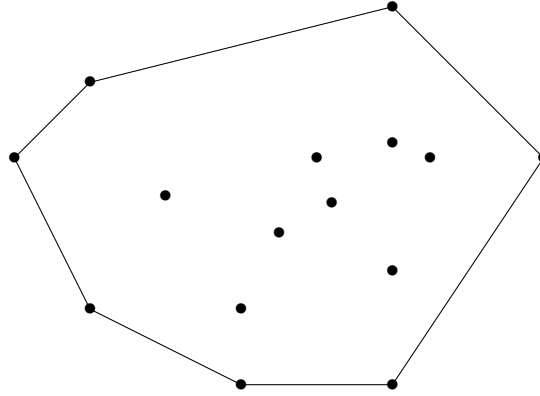


FIGURE 3 – Exemple d’enveloppe convexe d’un ensemble de points

**Position d’un point par rapport à une droite** On suppose qu’il n’existe pas trois points alignés.  $p_2$  est à gauche de la droite orientée  $(p_0\vec{p}_1)$  si et seulement si :

$$\det(p_0\vec{p}_1, p_0\vec{p}_2) > 0$$

**Marche de Jarvis** Algorithme du paquet cadeau

- Partir du point de plus petite ordonnée  $p_0$
- Prendre un point suivant, parcourir tous les points et le remplacer s’il en existe un plus à gauche de la droite orientée  $(p_{actuel} \vec{p}_{suivant})$
- Recommencer jusqu’à retomber sur  $p_0$

**Complexité de la marche de Jarvis** Si  $n$  est la taille de  $S$  et  $h$  le nombre de points dans l’enveloppe convexe :  $O(nh)$

La complexité dépend de la sortie (*output sensitive*).

**Balayage de Graham** On utilise une pile pour représenter l’enveloppe.  $p_0$  point d’ordonnée minimale.  $p_1 \dots p_n$  énumération des autres points, triés par angle polaire croissant (par rapport à  $p_0$ ). On est alors certains que les 3 premiers (et les deux derniers) points sont dans l’enveloppe convexe.

- Au début, empiler  $p_0, p_1$  puis  $p_2$
- Pour  $i$  de 1 à  $n$ 
  - Tant que  $p_i$  est à droite du vecteur formé par les deux points en haut de la pile (ie  $p_i$  est à l’extérieur de la proposition d’enveloppe que représente la pile), on dépile.
  - Puis on empile  $p_i$ .
- On retourne la pile.

**Complexité du balayage de Graham**  $\mathcal{O}(n \log(n))$  pour trier les points. Dans la boucle, on empile ou dépile au maximum  $n$  fois.

Donc complexité en  $\mathcal{O}(n \log(n))$ .

## 4.2 Points les plus rapprochés

### Points les plus rapprochés

*Entrée* Un ensemble  $S$  de points  $(x, y)$  du plan  
*Sortie*  $d^*$  la distance minimale entre deux points de  $S$

**Algorithme** Principe de diviser pour régner, on divise le plan en deux. On obtient récursivement  $\delta_D$  et  $\delta_G$ . Soit  $\delta = \min(\delta_G, \delta_D)$ . On considère l'ensemble  $M$  des points dans la bande de largeur  $2\delta$ .  $M = p_1 \dots p_k$  énumérés par ordonnées croissantes.

**Lemme** S'il existe  $i < j$  tel que  $d(p_i, p_j) < \delta$  alors il existe  $i' < j' \leq i' + 8$  tel que  $d(p_{i'}, p_{j'}) < \delta$ .

En effet, on considère un rectangle qu'on divise en huit parties. On peut alors prendre 9 points et appliquer le principe des tiroirs.

**Complexité** Pré-tris en  $O(n \log n)$ . Au total, avec le *Master Theorem* :  $O(n \log n)$

## 5 Algorithmes distribués

### 5.1 Généralités

Un système distribué est constitué de **processus** communiquant par des objets de partage ou par **envoi de messages**.

- **Communications**
  - Asynchrones (pas de confirmation de réception du message)
  - Par rendez-vous (Transmission uniquement si le récepteur est prêt, problème d'interblocage)
- **Primitives**
  - Point à point
  - broadcast
- **Topologie**
  - Graphe fortement connexe
  - Graphe complet
- **Degré de synchronie**
  - Synchrone (horloges locales des processus sont synchronisées, délai d'acheminement des messages borné par  $\Delta$ , vitesse relative des processus bornée par  $\phi$ )
  - Asynchrone ( $\Delta$  et  $\phi$  n'existent pas a priori)
- **Types de défaillances**
  - Processus (crash, omission, byzantin)
  - Liens de communication, perte de messages, altération, duplication, création

### 5.2 Problème du consensus

Chaque processus  $p_i$  a une valeur  $v_i$  et une variable de décision  $d_i$  initialement à  $\perp$  qui est *write once*.

Un algorithme résout le consensus s'il garantit :

- **Accord** : Si deux processus décident, alors ils décident la même valeur
  - **Terminaison** : Tout processus correct va finalement décider
  - **Validité** : Si  $v$  est une valeur de décision d'un processus correct, alors  $v$  était une valeur initiale
- Si on ôte une des trois propriétés, le problème du consensus affaibli devient trivial.

### 5.3 Consensus dans le cas synchrone

Si on formalise, un processus  $p_i$  a un ensemble d'états ( $states_i$ ), des états initiaux possibles ( $start_i \subseteq states_i$ ) une fonction  $msg_i$  qui à un état et un voisin associe un message (ou *null*) et une fonction de transition. Un round est constitué d'une application de la fonction  $msg_i$  et d'un envoi des messages aux destinataires, puis d'une réception des messages et une mise à jour de l'état courant.

Un algorithme résout le problème du consensus et tolère  $t$  défaillances si pour toute exécution  $\sigma$  telle que  $f(\sigma) \leq t$  (nombre de défaillances au cours de  $\sigma$ ), on a les trois propriétés : accord, terminaison, validité.

**Algorithme Flood Set** Chaque processus commence avec  $W_i = \{v_i\}$ . Pour tout round d'indice inférieur à  $t$ , chaque processus envoie  $W_i$  à tout le monde, et ajoute à  $W_i$  ce qu'il reçoit des autres. Quand l'indice du round est  $t + 1$ , on décide  $d_i = \min_i W_i$ . L'algorithme Flood Set résout le problème du consensus pour les systèmes synchrones et tolère  $t$  défaillances de type crash.

**Preuve** Terminaison : décision au tour  $t + 1$ . Validité : à tout instant,  $W_i$  contient des valeurs initiales. Accord : Il existe un round sans crash, à l'issue de ce round tous les processus ont le même  $W_i$ , et l'égalité est conservée.

## 5.4 Consensus dans le cas asynchrone

**Théorème de Fischer Lynch Paterson** Il n'existe pas d'algorithme résolvant le consensus dans le cas asynchrone qui tolère même une seule défaillance de type crash.

**TODO** formalisation, propriété du diamant, bivalence, étapes de la preuve (cf TD6)