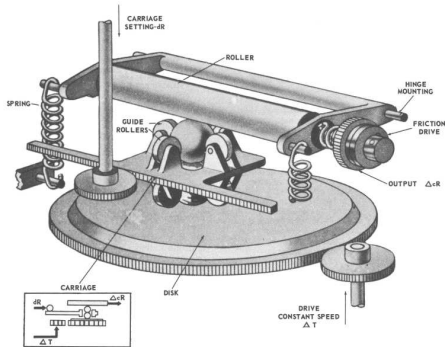


# Cours 10: Autres complexités



Olivier Bournez  
bournez@lix.polytechnique.fr

Ecole Polytechnique  
INF412

# Au menu

Au delà de P et NP

Relations entre espace et temps

Problèmes complets

D'autres complexités

Le grand tout

# Plus précisément

Au delà de P et NP

Complexité en temps : compléments

Complexité en espace

Classes les plus usuelles

# Théorèmes de hiérarchie

## Théorème (de Hiérarchie)

*Soient  $f, f' : \mathbb{N} \rightarrow \mathbb{N}$  des fonctions constructibles en temps telles que  $f(n) \log(f(n)) = o(f'(n))$ . Alors l'inclusion  $\text{TIME}(f) \subsetneq \text{TIME}(f')$  est stricte.*

# Théorèmes de hiérarchie

## Théorème (de Hiérarchie)

*Soient  $f, f' : \mathbb{N} \rightarrow \mathbb{N}$  des fonctions constructibles en temps telles que  $f(n) \log(f(n)) = o(f'(n))$ . Alors l'inclusion  $\text{TIME}(f) \subsetneq \text{TIME}(f')$  est stricte.*

- ▶ On dit qu'une fonction  $f(n) \geq n \log(n)$  est constructible en temps si la fonction qui envoie  $1^n$  sur la représentation binaire de  $1^{f(n)}$  est calculable en temps  $\mathcal{O}(f(n))$  :

# Théorèmes de hiérarchie

## Théorème (de Hiérarchie)

*Soient  $f, f' : \mathbb{N} \rightarrow \mathbb{N}$  des fonctions constructibles en temps telles que  $f(n) \log(f(n)) = o(f'(n))$ . Alors l'inclusion  $\text{TIME}(f) \subsetneq \text{TIME}(f')$  est stricte.*

- ▶ On dit qu'une fonction  $f(n) \geq n \log(n)$  est constructible en temps si la fonction qui envoie  $1^n$  sur la représentation binaire de  $1^{f(n)}$  est calculable en temps  $\mathcal{O}(f(n))$  :
  - toutes les fonctions usuelles le sont.

## Corollaire

$$\text{TIME}(n^2) \subsetneq \text{TIME}(n^{\log n}) \subsetneq \text{TIME}(2^n).$$

- On obtient :

$$P \subsetneq \text{EXPTIME}.$$

où

$$\text{EXPTIME} = \bigcup_{c \in \mathbb{N}} \text{TIME}(2^{n^c}).$$

## Corollaire

$$\text{TIME}(n^2) \subsetneq \text{TIME}(n^{\log n}) \subsetneq \text{TIME}(2^n).$$

- On obtient :

$$P \subsetneq \text{EXPTIME}.$$

où

$$\text{EXPTIME} = \bigcup_{c \in \mathbb{N}} \text{TIME}(2^{n^c}).$$

- Preuve :

- ▶ Tout polynôme devient ultimement négligeable devant  $n^{\log n}$ , et donc  $P$  est un sous-ensemble de  $\text{TIME}(n^{\log n})$ .
- ▶ Maintenant  $\text{TIME}(n^{\log n})$ , qui contient tout  $P$  est un sous-ensemble strict de, par exemple,  $\text{TIME}(2^n)$ , qui est inclus dans  $\text{EXPTIME}$ .



# Plus précisément

Au delà de P et NP

Complexité en temps : compléments

Complexité en espace

Classes les plus usuelles

## Complexité en espace mémoire

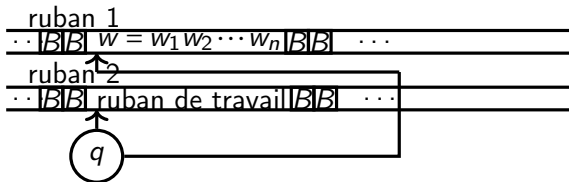
- On souhaite mesurer la mémoire utilisée par un algorithme.

- ▶ En théorie de la complexité, on parle plutôt **d'espace**, ou **d'espace mémoire**, pour désigner la mémoire.

- ▶ Soit  $t : \mathbb{N} \rightarrow \mathbb{N}$  une fonction.

On définit :  $SPACE(t(n))$  comme la classe des problèmes (langages) qui sont décidés par une machine de Turing qui utilise deux rubans :

- le premier ruban contient l'entrée est accessible **en lecture seulement** : il peut être lu, mais il ne peut pas être écrit ;
- le second est lui initialement vide et est accessible en lecture et écriture ;
- en utilisant  $\mathcal{O}(t(n))$  cases du **second** ruban, où  $n$  est la taille de l'entrée.



- PSPACE est la classe des problèmes (langages) décidés en espace polynomial :

$$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k).$$

- Cette notion d'espace mémoire ne dépend pas réellement du modèle de calcul utilisé :
  - ▶ l'utilisation des machines de Turing comme modèle de base est arbitraire.

# Plus précisément

Au delà de P et NP

Complexité en temps : compléments

Complexité en espace

Classes les plus usuelles

## Classes les plus usuelles

$$\text{LOGSPACE} = \text{SPACE}(\log n)$$

$$\text{NLOGSPACE} = \text{NSPACE}(\log n)$$

$$P = \bigcup_{c \in \mathbb{N}} \text{TIME}(n^c)$$

$$NP = \bigcup_{c \in \mathbb{N}} \text{NTIME}(n^c)$$

$$\text{PSPACE} = \bigcup_{c \in \mathbb{N}} \text{SPACE}(n^c)$$

# Au menu

Au delà de P et NP

Relations entre espace et temps

Problèmes complets

D'autres complexités

Le grand tout

# Plus précisément

## Relations entre espace et temps

### Relations triviales

Temps non-déterministe vs déterministe

Temps non-déterministe vs espace

Espace non-déterministe vs temps

Espace non-déterministe vs espace déterministe

## Relations triviales

Théorème

$$\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n)).$$

Théorème

$$\text{TIME}(f(n)) \subseteq \text{SPACE}(f(n)).$$



## Relations triviales

### Théorème

$$\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n)).$$

### Théorème

$$\text{TIME}(f(n)) \subseteq \text{SPACE}(f(n)).$$

Conséquences :

- $\text{PSPACE} \subseteq \text{NSPACE}$ ,
- $\text{P} \subseteq \text{PSPACE}$ .

■ Démonstration :

- ▶ Une machine de Turing déterministe est une machine de Turing non-déterministe particulière.
- ▶ Une machine de Turing écrit au plus un nombre fini d'emplacements (mémoire) à chaque étape.

# Plus précisément

## Relations entre espace et temps

Relations triviales

**Temps non-déterministe vs déterministe**

Temps non-déterministe vs espace

Espace non-déterministe vs temps

Espace non-déterministe vs espace déterministe

## Théorème

*Pour tout langage de  $\text{NTIME}(f(n))$ , il existe un entier  $c$  tel que ce langage soit dans  $\text{TIME}(c^{f(n)})$ .*

*Si l'on préfère :*

$$\text{NTIME}(f(n)) \subseteq \bigcup_{c \in \mathbb{N}} \text{TIME}(c^{f(n)}).$$

■ Démonstration :

- ▶ Soit  $L$  un problème de  $\text{NTIME}(f(n))$ .
- ▶ On sait qu'il existe un problème  $A$  tel que pour déterminer si un mot  $w$  de longueur  $n$  est dans  $L$ , il suffit de savoir s'il existe un certificat  $u \in \Sigma^*$  avec  $(w, u) \in A$ , ce dernier test pouvant se faire en temps  $f(n)$ , où  $n = \text{longueur}(w)$ .
- ▶ On peut se limiter aux mots  $u$  de longueur  $\mathcal{O}(f(n))$ .
- ▶ Tester si  $(w, u) \in A$  pour tous les mots  $u \in \Sigma^*$  de longueur  $f(n)$  se fait facilement en temps  $\mathcal{O}(c^{f(n)}) + \mathcal{O}(f(n)) = \mathcal{O}(c^{f(n)})$ , où  $c > 1$  est le cardinal de l'alphabet  $\Sigma$ .

## Remarque

- Pour écrire un mot  $u$  de longueur  $f(n)$ , il faut que  $f$  ne soit pas quelconque :

## Remarque

- Pour écrire un mot  $u$  de longueur  $f(n)$ , il faut que  $f$  ne soit pas quelconque :
  - ▶ par exemple calculable!!!

## Remarque

- Pour écrire un mot  $u$  de longueur  $f(n)$ , il faut que  $f$  ne soit pas quelconque :
  - ▶ par exemple calculable!!!
- En fait, on se limite aux bornes sur le temps et l'espace qui sont **de complexité propre** :
  - ▶ on suppose que la fonction  $f(n)$  est croissante, c'est-à-dire que  $f(n+1) \geq f(n)$ ,
  - ▶ et qu'il existe un algorithme qui prend en entrée  $w$  et qui produit en sortie  $1^{f(n)}$  en temps  $\mathcal{O}(n+f(n))$
  - ▶ et en espace  $\mathcal{O}(f(n))$ , où  $n = \text{longueur}(w)$ .



## Remarque

- Pour écrire un mot  $u$  de longueur  $f(n)$ , il faut que  $f$  ne soit pas quelconque :
  - ▶ par exemple calculable!!!
- En fait, on se limite aux bornes sur le temps et l'espace qui sont **de complexité propre** :
  - ▶ on suppose que la fonction  $f(n)$  est croissante, c'est-à-dire que  $f(n+1) \geq f(n)$ ,
  - ▶ et qu'il existe un algorithme qui prend en entrée  $w$  et qui produit en sortie  $1^{f(n)}$  en temps  $\mathcal{O}(n+f(n))$
  - ▶ et en espace  $\mathcal{O}(f(n))$ , où  $n = \text{longueur}(w)$ .
- Toutes les fonctions usuelles non-décroissantes, comme  $\log(n)$ ,  $n$ ,  $n^2$ ,  $\dots$ ,  $n \log n$ ,  $n!$  vérifient ces propriétés.

## Remarque

- Pour écrire un mot  $u$  de longueur  $f(n)$ , il faut que  $f$  ne soit pas quelconque :
  - ▶ par exemple calculable!!!
- En fait, on se limite aux bornes sur le temps et l'espace qui sont **de complexité propre** :
  - ▶ on suppose que la fonction  $f(n)$  est croissante, c'est-à-dire que  $f(n+1) \geq f(n)$ ,
  - ▶ et qu'il existe un algorithme qui prend en entrée  $w$  et qui produit en sortie  $1^{f(n)}$  en temps  $\mathcal{O}(n+f(n))$
  - ▶ et en espace  $\mathcal{O}(f(n))$ , où  $n = \text{longueur}(w)$ .
- Toutes les fonctions usuelles non-décroissantes, comme  $\log(n)$ ,  $n$ ,  $n^2$ ,  $\dots$ ,  $n \log n$ ,  $n!$  vérifient ces propriétés.
- Cette hypothèse sera implicite dans la suite.

# Plus précisément

## Relations entre espace et temps

Relations triviales

Temps non-déterministe vs déterministe

**Temps non-déterministe vs espace**

Espace non-déterministe vs temps

Espace non-déterministe vs espace déterministe

# Temps non-déterministe vs espace

## Théorème

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n)).$$

# Temps non-déterministe vs espace

## Théorème

$$\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n)).$$

Conséquences :

- $\text{NP} \subseteq \text{PSPACE}$

## ■ Démonstration :

- ▶ On utilise exactement le même principe que dans la preuve précédente, si ce n'est que l'on parle d'espace.
- ▶ Soit  $L$  un problème de  $\text{NTIME}(f(n))$ .
- ▶ On sait qu'il existe un problème  $A$  tel que pour déterminer si un mot  $w$  de longueur  $n$  est dans  $L$ , il suffit de savoir s'il existe  $u \in \Sigma^*$  de longueur  $f(n)$  avec  $(w, u) \in A$ .
- ▶ On utilise un espace  $\mathcal{O}(f(n))$  pour générer un à un les mots  $u \in \Sigma^*$  de longueur  $f(n)$  puis on teste pour chacun si  $(w, u) \in A$ , ce dernier test se faisant en temps  $f(n)$ , donc espace  $f(n)$ .
- ▶ Le même espace pouvant être utilisé pour chacun des mots  $u$ , au total cela se fait au total un espace  $\mathcal{O}(f(n))$  pour générer les  $u + \mathcal{O}(f(n))$  pour les tests, soit  $\mathcal{O}(f(n))$ .

# Plus précisément

## Relations entre espace et temps

Relations triviales

Temps non-déterministe vs déterministe

Temps non-déterministe vs espace

**Espace non-déterministe vs temps**

Espace non-déterministe vs espace déterministe

# Lien fondamental avec le problème REACH

- Le problème de décision REACH,
  - ▶ on se donne
    1. un graphe orienté  $G = (V, E)$ ,
    2. deux sommets  $u$  et  $v$ ,
  - ▶ et on cherche à décider s'il existe un chemin entre  $u$  et  $v$  dans  $G$ .

jouera un grand rôle.



## Lien fondamental avec le problème REACH

- A toute machine de Turing  $M$  est associé un graphe orienté  $G_M$ 
  - ▶ dont les sommets sont les configurations,
  - ▶ et tel qu'il y a une arête entre la configuration  $C$  et la configuration  $C'$  ssi  $C \vdash C'$ .
  
- Etant donnée une entrée  $w$ , on note  $C[w]$  la configuration initiale correspondante. On peut supposer qu'il y a une unique configuration terminale acceptante  $C^*$ .
  
- Soit  $L$  accepté par  $M$ .  
 $w \in L$  si et seulement si  $(G_M, C[w], C^*) \in \text{REACH}$ .

## Lien fondamental avec le problème REACH

- A toute machine de Turing  $M$  est associé un graphe orienté  $G_M$ 
  - ▶ dont les sommets sont les configurations,
  - ▶ et tel qu'il y a une arête entre la configuration  $C$  et la configuration  $C'$  ssi  $C \vdash C'$ .
    - c.-à-d. : ssi on peut passer en une étape de la configuration  $C$  en la configuration  $C'$ .
- Etant donnée une entrée  $w$ , on note  $C[w]$  la configuration initiale correspondante. On peut supposer qu'il y a une unique configuration terminale acceptante  $C^*$ .
- Soit  $L$  accepté par  $M$ .  
 $w \in L$  si et seulement si  $(G_M, C[w], C^*) \in \text{REACH}$ .

## Lien fondamental avec le problème REACH

- A toute machine de Turing  $M$  est associé un graphe orienté  $G_M$ 
  - ▶ dont les sommets sont les configurations,
  - ▶ et tel qu'il y a une arête entre la configuration  $C$  et la configuration  $C'$  ssi  $C \vdash C'$ .
    - c.-à-d. : ssi on peut passer en une étape de la configuration  $C$  en la configuration  $C'$ .
- Etant donnée une entrée  $w$ , on note  $C[w]$  la configuration initiale correspondante. On peut supposer qu'il y a une unique configuration terminale acceptante  $C^*$ .
  - ▶ si on fixe la longueur  $n$  de l'entrée  $w$ , pour un calcul en espace  $f(n)$ , il y a moins de  $\mathcal{O}\left(c^{f(n)}\right)$  sommets dans ce graphe  $G_M$ , où  $c > 1$  est le cardinal de l'alphabet de  $M$ .
- Soit  $L$  accepté par  $M$ .

$w \in L$  si et seulement si  $(G_M, C[w], C^*) \in \text{REACH}$ .

## Lien fondamental avec le problème REACH

- A toute machine de Turing  $M$  est associé un graphe orienté  $G_M$ 
  - ▶ dont les sommets sont les configurations,
  - ▶ et tel qu'il y a une arête entre la configuration  $C$  et la configuration  $C'$  ssi  $C \vdash C'$ .
    - c.-à-d. : ssi on peut passer en une étape de la configuration  $C$  en la configuration  $C'$ .
- Etant donnée une entrée  $w$ , on note  $C[w]$  la configuration initiale correspondante. On peut supposer qu'il y a une unique configuration terminale acceptante  $C^*$ .
  - ▶ si on fixe la longueur  $n$  de l'entrée  $w$ , pour un calcul en espace  $f(n)$ , il y a moins de  $\mathcal{O}(c^{f(n)})$  sommets dans ce graphe  $G_M$ , où  $c > 1$  est le cardinal de l'alphabet de  $M$ .
- Soit  $L$  accepté par  $M$ .

$w \in L$  si et seulement si  $(G_M, C[w], C^*) \in \text{REACH}$ .
- On va écrire ce que l'on sait sur le problème REACH.

- REACH se résout par exemple en temps et espace  $\mathcal{O}(n^2)$ , où  $n$  est le nombre de sommets, par un parcours en profondeur.

# Conséquence

## Théorème

*Si  $f(n) \geq \log n$ , alors*

$$\text{NSPACE}(f(n)) \subseteq \bigcup_{c \in \mathbb{N}} \text{TIME}(c^{f(n)}).$$

# Plus précisément

## Relations entre espace et temps

Relations triviales

Temps non-déterministe vs déterministe

Temps non-déterministe vs espace

Espace non-déterministe vs temps

**Espace non-déterministe vs espace déterministe**

## Proposition

$\text{REACH} \in \text{SPACE}(\log^2 n)$ .



# Astuce du théorème de Savitch

## ■ Démonstration :

- ▶ Soit  $G = (V, E)$  le graphe orienté en entrée.
- ▶ Étant donnés deux sommets  $x$  et  $y$  de ce graphe, et  $i$  un entier, on note  $CHEMIN(x, y, i)$  si et seulement si il y a un chemin de longueur inférieure à  $2^i$  entre  $x$  et  $y$ .
- ▶ On a  $(G, u, v) \in REACH$  si et seulement si  $CHEMIN(u, v, \log(n))$ , où  $n$  est le nombre de sommets.
- ▶ L'astuce est de calculer  $CHEMIN(x, y, i)$  récursivement en observant que l'on a  $CHEMIN(x, y, i)$  si et seulement si il existe un sommet intermédiaire  $z$  tel que  $CHEMIN(x, z, i-1)$  et  $CHEMIN(z, y, i-1)$ . On teste alors à chaque niveau de la récursion chaque sommet possible  $z$ .
- ▶ Pour représenter chaque sommet, il faut  $\mathcal{O}(\log(n))$  bits. Cela donne une récurrence de profondeur  $\log(n)$ , chaque étape de la récurrence nécessitant uniquement de stocker un triplet  $x, y, i$  et de tester chaque  $z$  de longueur  $\mathcal{O}(\log(n))$ . Au total, on utilise donc un espace  $\mathcal{O}(\log(n)) * \mathcal{O}(\log(n)) = \mathcal{O}(\log^2(n))$ .

## Conséquence : Théorème de Savitch

### Théorème (Savitch)

*Si  $f(n) \geq \log(n)$ , alors*

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f(n)^2).$$

- On utilise l'algorithme précédent pour déterminer s'il y a un chemin dans le graphe  $G_M$  entre  $C[w]$  et  $C^*$ .
- On remarquera que l'on a pas besoin de construire explicitement le graphe  $G_M$  mais que l'on peut utiliser l'algorithme précédent à la volée.

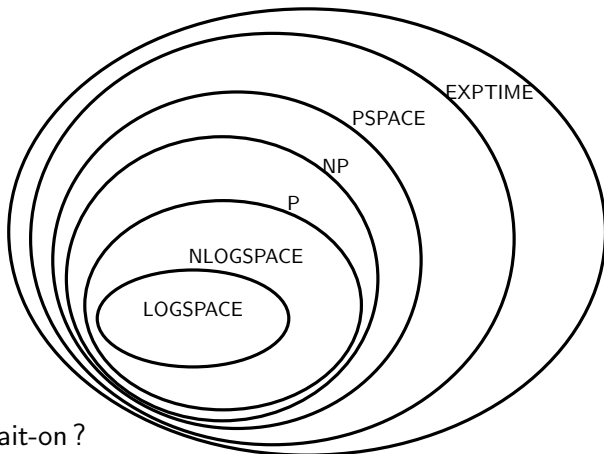
# Théorème de Savitch

Corollaire

$PSPACE = NPSPACE.$

## Le paysage de la complexité

- LOGSPACE  $\subseteq$  NLOGSPACE  $\subseteq$  P  $\subseteq$  NP  $\subseteq$  PSPACE  $\subseteq$  EXPTIME.



- Que sait-on ?
  - ▶ NLOGSPACE  $\subsetneq$  PSPACE
  - ▶ P  $\subsetneq$  EXPTIME,
  - ▶ on ne sait pas lesquelles des inclusions intermédiaires sont strictes

# Au menu

Au delà de P et NP

Relations entre espace et temps

**Problèmes complets**

D'autres complexités

Le grand tout

# Plus précisément

Problèmes complets

Digression : Exercice

## Digression : Exercice

- Produire un problème PSPACE-complet (pour  $\leq$ ).



## Digression : Exercice

- Produire un problème PSPACE-complet (pour  $\leq$ ).

▶ Problème *ARTIFICIEL2*:

**Donnée:**

- Le codage  $\langle M \rangle$  d'une machine de Turing  $M$
- $1^t$ , un entier écrit en unaire
- et un mot  $w$ .

**Réponse:** Décider si la machine  $M$  accepte le mot  $w$  en espace  $t$

## Digression : Exercice

- Produire un problème PSPACE-complet (pour  $\leq$ ).

- ▶ Problème *ARTIFICIEL2*:

**Donnée:**

- Le codage  $\langle M \rangle$  d'une machine de Turing  $M$
- $1^t$ , un entier écrit en unaire
- et un mot  $w$ .

**Réponse:** Décider si la machine  $M$  accepte le mot  $w$  en espace  $t$

- ▶ Il est dans PSPACE.

## Digression : Exercice

- Produire un problème PSPACE-complet (pour  $\leq$ ).

▶ Problème *ARTIFICIEL2*:

**Donnée:**

- Le codage  $\langle M \rangle$  d'une machine de Turing  $M$
- $1^t$ , un entier écrit en unaire
- et un mot  $w$ .

**Réponse:** Décider si la machine  $M$  accepte le mot  $w$  en espace  $t$

▶ Il est dans PSPACE.

▶ Il est PSPACE-difficile :

- Soit  $A$  un problème de PSPACE : il est reconnu par une machine de Turing  $M$  en espace  $p(n)$ .
- La fonction  $f$  qui à  $w$  associe  $(\langle M \rangle, 1^{p(|w|)}, w)$  vérifie  $w \in A$  ssi  $f(w) \in \text{ARTIFICIEL2}$ .
- La fonction  $f$  est calculable en temps polynomial.

## Digression : Exercice

- Produire un problème PSPACE-complet (pour  $\leq$ ).

▶ Problème *ARTIFICIEL2*:

**Donnée:**

- Le codage  $\langle M \rangle$  d'une machine de Turing  $M$
- $1^t$ , un entier écrit en unaire
- et un mot  $w$ .

**Réponse:** Décider si la machine  $M$  accepte le mot  $w$  en espace  $t$

▶ Il est dans PSPACE.

▶ Il est PSPACE-difficile :

- Soit  $A$  un problème de PSPACE : il est reconnu par une machine de Turing  $M$  en espace  $p(n)$ .
- La fonction  $f$  qui à  $w$  associe  $(\langle M \rangle, 1^{p(|w|)}, w)$  vérifie  $w \in A$  ssi  $f(w) \in \text{ARTIFICIEL2}$ .
- La fonction  $f$  est calculable en temps polynomial.
- On a donc bien  $A \leq \text{ARTIFICIEL2}$ .

# Un problème PSPACE-complet naturel

- Problème QBF (*Quantified Boolean Formula*):

**Donnée:** Une formule du calcul propositionnel en forme normale conjonctive  $\phi$

**Réponse:** déterminer si  $\exists x_1 \forall x_2 \exists x_3 \dots \phi(x_1, \dots, x_n)$ ? où  $x_1, x_2, \dots, x_n$  sont les variables de  $\phi$ .

## Théorème

*Le problème QBF est PSPACE-complet.*

- Le problème QBF est dans PSPACE car il est facile de construire un algorithme récursif qui le résout :
  - ▶ pour une formule de la forme  $\exists x_1 \phi$  (respectivement :  $\forall x_1 \phi$ ) on fixe la valeur de  $x_1$  à 0, on propage cette valeur pour  $x_1$  dans  $\phi$  pour obtenir  $\phi_{x_1=0}$ ,
    - et on s'appelle récursivement pour savoir si la formule  $\phi_{x_1=0}$  est satisfiable.
  - ▶ On fixe alors la valeur de  $x_1$  à 1, on propage cette valeur pour  $x_1$  dans  $\phi$  pour obtenir  $\phi_{x_1=1}$ ,
    - et on s'appelle récursivement pour savoir si la formule  $\phi_{x_1=1}$  est satisfiable.
  - ▶ On accepte si et seulement si  $\phi_{x_1=0}$  ou (resp. et)  $\phi_{x_1=1}$  sont satisfiables.
  
- L'algorithme fonctionne avec  $\mathcal{O}(n)$  appels récursifs, chacun utilisant un espace constant, et donc en espace total  $\mathcal{O}(n)$ , où  $n$  est la taille de la formule.

## ■ Pour montrer qu'il est complet

- ▶ Supposons que le problème  $L$  soit accepté en espace polynomial.
- ▶ On appelle  $G_M$  le graphe des configurations de la machine de Turing  $M$ , avec un arc entre deux configurations si l'une est successeur directe de l'autre.
- ▶ On va écrire par une formule quantifiée  $\phi_i([C],[C'])$  le fait qu'il existe un chemin de longueur inférieure à  $2^i$  dans le graphe  $G_M$  entre les configurations  $C$  et  $C'$ . Le graphe  $G_M$  est de taille  $2^{p(n)}$  pour un certain polynôme  $p$ , où  $n = longueur(w)$ .
- ▶ Cette formule s'obtient par récurrence sur  $i$ . Pour  $i = 0$ , c'est la formule qui donne les arcs du graphe  $G_M$ .
- ▶ Pour  $i > 0$ , on a envie d'écrire  $\phi_i([C],[C'])$  comme

$$\exists [C''] \phi_{i-1}([C],[C'']) \wedge \phi_{i-1}([C''],[C']).$$

Cependant si l'on fait ainsi, la formule obtenue sera de taille exponentielle en  $i$ .

- L'idée est de quantifier en réutilisant l'espace : on écrit  $\phi_i([C], [C'])$  comme

$$\exists [C''] \forall [D_1] \forall [D_2] \phi'_i([C], [C'], [D_1], [D_2], [C''])$$

où  $\phi'_i([C], [C'], [D_1], [D_2], [C''])$  teste la relation

$$\begin{aligned} &(((D_1] = [C] \wedge [D_2] = [C'']) \vee (([D_1] = [C''] \wedge [D_2] = [C'])) \\ &\Rightarrow \phi_{i-1}([D_1], [D_2]). \end{aligned}$$

- Cette fois la taille de  $\phi_i$  sera de l'ordre de celle de  $\phi_{i-1}$  plus  $\mathcal{O}(p(n))$ . Un mot  $w$  est dans le langage  $L$  si et seulement si  $\phi_{p(n)}([C[w]], [C^*])$ .
- La fonction qui à  $w$  associe la formule quantifiée  $\phi_{p(n)}([C[w]], [C^*])$  est bien calculable en temps polynomial.



## D'autres problèmes PSPACE-complets.

- Les jeux stratégiques donnent naturellement naissance à des problèmes PSPACE-complets.

## D'autres problèmes PSPACE-complets.

- Les jeux stratégiques donnent naturellement naissance à des problèmes PSPACE-complets.
  - ▶ Par exemple, le jeu GEOGRAPHY est PSPACE-complet :
    - On joue sur un graphe orienté  $G = (V, E)$ .
    - Le joueur 1 choisit un sommet  $u_1$  du graphe. Le joueur 2 doit alors choisir un sommet  $v_1$  tel qu'il y ait un arc de  $u_1$  vers  $v_1$ . C'est alors au joueur 1 de choisir un autre sommet  $u_2$  tel qu'il y ait un arc de  $v_1$  vers  $u_2$ , et ainsi de suite.
    - On n'a pas le droit de repasser deux fois par le même sommet.
    - Le premier joueur qui ne peut pas continuer le chemin  $u_1 v_1 u_2 v_2 \dots$  perd.
    - Le problème GEOGRAPHY consiste à déterminer étant donné un graphe  $G$  et un sommet de départ pour le joueur 1, s'il existe une stratégie gagnante pour le joueur 1.

## Jouer aux échecs ? au go ?

- Problème ECHEC:

**Donnée:** Une configuration aux échecs.

**Réponse:** Décider si les blancs ont une stratégie gagnante.

## Jouer aux échecs ? au go ?

- Problème ECHEC:

**Donnée:** Une configuration aux échecs.

**Réponse:** Décider si les blancs ont une stratégie gagnante.

- Décidable ? Complexité ?

## Jouer aux échecs ? au go ?

- Problème ECHEC:

**Donnée:** Une configuration aux échecs.

**Réponse:** Décider si les blancs ont une stratégie gagnante.

- Décidable ? Complexité ?

- ▶ Un échiquier est une grille  $8 \times 8$ . Il y a donc un nombre fini de configurations possibles.

# Jouer aux échecs ? au go ?

- Problème ECHEC:

**Donnée:** Une configuration aux échecs.

**Réponse:** Décider si les blancs ont une stratégie gagnante.

- Décidable ? Complexité ?

- ▶ Un échiquier est une grille  $8 \times 8$ . Il y a donc un nombre fini de configurations possibles.
- ▶ Il y a donc un nombre fini de configurations  $C_1, C_2, \dots, C_m$  gagnantes.

# Jouer aux échecs ? au go ?

## ■ Problème ECHEC:

**Donnée:** Une configuration aux échecs.

**Réponse:** Décider si les blancs ont une stratégie gagnante.

## ■ Décidable ? Complexité ?

- ▶ Un échiquier est une grille  $8 \times 8$ . Il y a donc un nombre fini de configurations possibles.
- ▶ Il y a donc un nombre fini de configurations  $C_1, C_2, \dots, C_m$  gagnantes.
- ▶ L'algorithme qui consiste à répondre *VRAI* si la configuration en entrée est  $C_1$  ou  $C_2$  ou  $\dots C_m$  fonctionne en utilisant un espace mémoire et un temps  $\mathcal{O}(1)$ .

## Jouer aux échecs ? au go ?

### ■ Problème ECHEC:

**Donnée:** Une configuration aux échecs.

**Réponse:** Décider si les blancs ont une stratégie gagnante.

### ■ Décidable ? Complexité ?

- ▶ Un échiquier est une grille  $8 \times 8$ . Il y a donc un nombre fini de configurations possibles.
- ▶ Il y a donc un nombre fini de configurations  $C_1, C_2, \dots, C_m$  gagnantes.
- ▶ L'algorithme qui consiste à répondre *VRAI* si la configuration en entrée est  $C_1$  ou  $C_2$  ou  $\dots C_m$  fonctionne en utilisant un espace mémoire et un temps  $\mathcal{O}(1)$ .

### ■ Le problème ECHEC est donc trivial<sup>1</sup> du point de vue de la complexité :

---

1. Attention : on cherche en complexité/calculabilité un algorithme qui résout le problème de décision, et on n'impose pas "plus", en particulier de savoir "produire efficacement" cet algorithme



# Jouer aux échecs ? au go ?

## ■ Problème ECHEC:

**Donnée:** Une configuration aux échecs.

**Réponse:** Décider si les blancs ont une stratégie gagnante.

## ■ Décidable ? Complexité ?

- ▶ Un échiquier est une grille  $8 \times 8$ . Il y a donc un nombre fini de configurations possibles.
- ▶ Il y a donc un nombre fini de configurations  $C_1, C_2, \dots, C_m$  gagnantes.
- ▶ L'algorithme qui consiste à répondre *VRAI* si la configuration en entrée est  $C_1$  ou  $C_2$  ou  $\dots C_m$  fonctionne en utilisant un espace mémoire et un temps  $\mathcal{O}(1)$ .

## ■ Le problème ECHEC est donc trivial<sup>1</sup> du point de vue de la complexité :

- ▶ il est dans SPACE(1), TIME(1).

---

1. Attention : on cherche en complexité/calculabilité un algorithme qui résout le problème de décision, et on n'impose pas "plus", en particulier de savoir "produire efficacement" cet algorithme

## Jouer aux échecs ? au go ?

### ■ Problème ECHEC:

**Donnée:** Une configuration aux échecs.

**Réponse:** Décider si les blancs ont une stratégie gagnante.

### ■ Décidable ? Complexité ?

- ▶ Un échiquier est une grille  $8 \times 8$ . Il y a donc un nombre fini de configurations possibles.
- ▶ Il y a donc un nombre fini de configurations  $C_1, C_2, \dots, C_m$  gagnantes.
- ▶ L'algorithme qui consiste à répondre *VRAI* si la configuration en entrée est  $C_1$  ou  $C_2$  ou  $\dots C_m$  fonctionne en utilisant un espace mémoire et un temps  $\mathcal{O}(1)$ .

### ■ Le problème ECHEC est donc trivial<sup>1</sup> du point de vue de la complexité :

- ▶ il est dans SPACE(1), TIME(1).

### ■ Même chose pour le GO qui se joue sur une grille $19 \times 19$ .

1. Attention : on cherche en complexité/calculabilité un algorithme qui résout le problème de décision, et on n'impose pas "plus", en particulier de savoir "produire efficacement" cet algorithme

## GO généralisé

- La grille est constituée de  $n \times n$  points.

## GO généralisé

- La grille est constituée de  $n \times n$  points.
- Problème GO:

**Donnée:** Une configuration de GO sur un échiquier  $n \times n$  et un entier  $1 \leq k \leq n^2$ .

**Réponse:** Décider si les blancs ont une stratégie gagnante qui mène en  $n^2 - k$  étapes à un échiquier où les noirs sont complètement éliminés.

## GO généralisé

- La grille est constituée de  $n \times n$  points.
- Problème GO:

**Donnée:** Une configuration de GO sur un échiquier  $n \times n$  et un entier  $1 \leq k \leq n^2$ .

**Réponse:** Décider si les blancs ont une stratégie gagnante qui mène en  $n^2 - k$  étapes à un échiquier où les noirs sont complètement éliminés.

### Théorème

GO est PSPACE-complet.

## GO généralisé

- La grille est constituée de  $n \times n$  points.
- Problème GO:

**Donnée:** Une configuration de GO sur un échiquier  $n \times n$  et un entier  $1 \leq k \leq n^2$ .

**Réponse:** Décider si les blancs ont une stratégie gagnante qui mène en  $n^2 - k$  étapes à un échiquier où les noirs sont complètement éliminés.

### Théorème

GO est PSPACE-complet.

- Principe :
  - ▶ GO est dans PSPACE.
  - ▶ GEOGRAPHY  $\leq$  GO :
    - voir par exemple Christos H. Papadimitriou. Computational Complexity. Addison-Wesley, 1994. p462.
    - et sur les jeux et leur complexité, plus généralement R. A. Hearn, E. D. Demaine Games, Puzzles, and Computation. A.K. Peters/CRC Press. 2009.

# Au menu

Au delà de P et NP

Relations entre espace et temps

Problèmes complets

**D'autres complexités**

Le grand tout

## Autres complexités

- Complexité parallèle.



## Autres complexités

- Complexité parallèle.
- Complexité randomisée.

## Autres complexités

- Complexité parallèle.
- Complexité randomisée.
- Complexité quantique.

## Autres complexités

- Complexité parallèle.
- Complexité randomisée.
- Complexité quantique.
- ...

# Plus précisément

## D'autres complexités

Quelques saveurs de la complexité parallèle

Quelques saveurs de la complexité randomisée

Quelques saveurs de la complexité quantique

# Quelques saveurs de la complexité parallèle

## ■ Thèse du parallélisme :

- ▶ Le temps parallèle polynomial<sup>2</sup> correspond à l'espace polynomial (PSPACE).
- ▶ Le temps parallèle polylogarithmique<sup>3</sup> correspond à l'espace polylogarithmique ( $\bigcup_{k \in \mathbb{N}} \text{SPACE}(\log^k n)$ ).

- 
2. Sans contrainte sur le nombre de processeurs, c'est-à-dire possiblement un nombre exponentiel de processeurs
  3. Possiblement un nombre polynomial de processeurs

# Problèmes “intrinsèquement non-parallélisables”

- On considère souvent dans le parallélisme que : “Se parallélise bien” = temps parallèle polylogarithmique.
  - ▶  $NC = \bigcup_{k \in \mathbb{N}} SPACE(\log^k n) \subseteq P$ .
- On ne sait pas si  $NC = P$ .
  - ▶ Si  $NC \neq P$  (comme cela est généralement conjecturé), les problèmes de  $P$  les plus difficiles à paralléliser doivent contenir les problèmes  $P$ -complets :
- En parallélisme, on dit parfois qu’un problème est **non-parallélisable** s’il est  $P$ -difficile<sup>4</sup>.

---

4. Pour un préordre  $\leq_L$  défini comme  $\leq$  (respectivement :  $\leq_m$ ) en remplaçant “calculable en temps polynomial” (resp. calculable) par “calculable en espace logarithmique”.

## Problèmes P-difficiles<sup>5</sup>

### ■ Évaluation en calcul propositionnel

**Donnée:** Une formule  $F(x_1, x_2, \dots, x_n)$  du calcul propositionnel en forme normale conjonctive, des valeurs  $x_1, \dots, x_n \in \{0, 1\}$  pour chacune des variables de la formule.

**Réponse:** Décider si la formule  $F$  s'évalue à vrai pour ces valeurs des variables.

### ■ Evaluation d'une suite booléenne

**Donnée:** Une suite de  $n$  équations booléennes du type  $e_1 = 0$ ,  $e_1 = 1$ , et  $e_k = e_j \wedge e_j$  ou  $e_k = e_j \vee e_k$ , pour  $1 \leq j < k \leq n$

**Réponse:** Calculer la valeur de  $e_n$

---

5. pour  $\leq_L$

# Plus précisément

## D'autres complexités

Quelques saveurs de la complexité parallèle

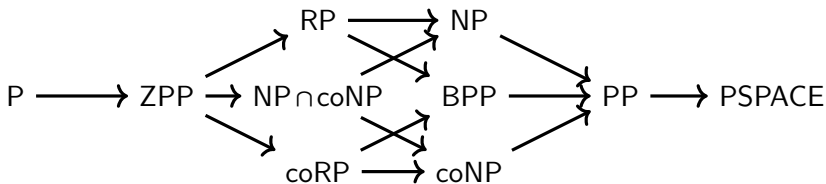
Quelques saveurs de la complexité randomisée

Quelques saveurs de la complexité quantique



## Résumé

Chaque flèche correspond à une inclusion :



- ZPP correspond aux problèmes qui sont reconnus par un algorithme qui termine toujours avec une réponse correcte et qui termine en temps moyen polynomial en la taille de l'entrée.
- BPP correspond aux problèmes qui sont reconnus par un algorithme qui termine toujours en temps polynomial mais possiblement avec une erreur bornée par  $\epsilon > 0$ .

## Exemple de problème BPP : Tester la nullité d'un polynôme

- On va présenter un test probabiliste pour lequel aucun algorithme déterministe aussi efficace n'est connu.
- Le but de ce test est de déterminer si un polynôme  $p(x_1, x_2, \dots, x_n)$  de degré faible à coefficients entiers est identiquement nul.
- On suppose que  $p$  est donné sous la forme d'un circuit arithmétique sur la structure  $(\mathbb{R}, +, -, *)$ .
  - ▶ On peut bien entendu vérifier cette propriété en développant le polynôme  $p$  et en vérifiant que chacun des termes du développement du polynôme est bien nul,
  - ▶ mais cela peut prendre un temps exponentiel dans le pire cas.

## Algorithme randomisé

- Evaluer  $p$  en des points  $a_1, a_2, \dots, a_n$  choisis aléatoirement.
- Si  $p$  est identiquement 0, alors  $p(a_1, a_2, \dots, a_n) = 0$ .
- Sinon, on s'attend à ce que  $p(a_1, a_2, \dots, a_n) \neq 0$  avec grande probabilité.

- Cela marche en fait aussi si l'on travaille sur un corps fini, si le corps est suffisamment grand.

### Proposition (Lemme de Schwartz-Zippel)

*Soit  $\mathbb{F}$  un corps, et  $S \subseteq \mathbb{F}$  un sous-ensemble arbitraire. Soit  $p(x)$  un polynôme non nul des  $n$  variables  $x = (x_1, x_2, \dots, x_n)$  de degré total  $d$  à coefficients dans  $\mathbb{F}$ . Alors l'équation  $p(x) = 0$  possède au plus  $d|S|^{n-1}$  solutions dans  $S^n$ .*

Autrement dit, soit  $\mathbb{F}$  un corps. Soit  $p(x_1, \dots, x_n)$  un polynôme non nul de degré total  $d$  à coefficients dans  $\mathbb{F}$ . Si  $p$  est évalué sur un élément choisi uniformément parmi les points de  $S^n$ , alors

$$\Pr(p(s_1, \dots, s_n) = 0) \leq \frac{d}{|S|}.$$

## Application : Problème du couplage parfait

- Un graphe biparti est un graphe dont les sommets peuvent se partitionner en deux sous-ensembles  $U$  et  $V$  tels que chaque arête ait une extrémité dans  $U$  et l'autre dans  $V$ .
- Un couplage parfait dans un graphe biparti est un sous-ensemble d'arêtes  $M$  tel que
  1. les arêtes de  $M$  ne partagent pas de sommets ;
  2. chaque sommet est l'extrémité d'une arête de  $M$ .

## Application : Problème du couplage parfait

- Un graphe biparti est un graphe dont les sommets peuvent se partitionner en deux sous-ensembles  $U$  et  $V$  tels que chaque arête ait une extrémité dans  $U$  et l'autre dans  $V$ .
- Un couplage parfait dans un graphe biparti est un sous-ensemble d'arêtes  $M$  tel que
  1. les arêtes de  $M$  ne partagent pas de sommets ;
  2. chaque sommet est l'extrémité d'une arête de  $M$ .
- On connaît un algorithme polynomial pour déterminer si un graphe biparti possède un couplage parfait.

## Application : Problème du couplage parfait

- Un graphe biparti est un graphe dont les sommets peuvent se partitionner en deux sous-ensembles  $U$  et  $V$  tels que chaque arête ait une extrémité dans  $U$  et l'autre dans  $V$ .
- Un couplage parfait dans un graphe biparti est un sous-ensemble d'arêtes  $M$  tel que
  1. les arêtes de  $M$  ne partagent pas de sommets ;
  2. chaque sommet est l'extrémité d'une arête de  $M$ .
- On connaît un algorithme polynomial pour déterminer si un graphe biparti possède un couplage parfait.
- On ne sait pas si le problème est dans NC (se résout en temps parallèle polylogarithmique).

## Algorithme NC randomisé

- On affecte à chaque arête  $(i,j)$  de  $G$  une variable  $x_{i,j}$  et on considère la matrice d'adjacence  $X$  de taille  $n \times n$  avec ces indéterminées plutôt que 1.
- Le déterminant  $\det(X)$  est un polynôme de degré  $n$  en les variables  $x_{i,j}$ , avec un terme (non nul) pour chaque couplage parfait.
- $G$  possède un couplage parfait si et seulement si le polynôme  $\det(X)$  ne s'annule pas.



# Algorithme NC randomisé

- Cela est difficile à vérifier de façon déterministe, car  $\det(X)$  peut être de grande taille.
- On peut le vérifier de façon probabiliste
  - ▶ en choisissant des valeurs pour les  $x_{ij}$  dans un corps  $\mathbb{F}$  de taille suffisamment grande (par exemple dans  $\mathbb{F}_p$ , avec  $p$  un nombre premier plus grand que  $2n$ ).
  - ▶ On calcule alors le déterminant, et on teste s'il est nul. Si le graphe ne possède pas de couplage parfait, cela se produira à coup sûr.
  - ▶ Si le graphe possède un couplage parfait, alors cela se produira avec probabilité au plus  $\frac{n}{2n} = \frac{1}{2}$ .

# Plus précisément

## D'autres complexités

Quelques saveurs de la complexité parallèle

Quelques saveurs de la complexité randomisée

Quelques saveurs de la complexité quantique

# Complexité quantique

- BQP : correspond aux problèmes solvables par un ordinateur quantique en temps polynomial, avec une probabilité d'erreur bornée par  $\epsilon > 0$ .
- Exemples :
  - ▶ Factorisation (Algorithme de Shor)
  - ▶ Logarithme Discret
  - ▶ ...

# Au menu

Au delà de P et NP

Relations entre espace et temps

Problèmes complets

D'autres complexités

**Le grand tout**

- Ceci est le dernier cours de INF412.
- MERCI.
  - ▶ pour votre attention.
  - ▶ pour vos retours.

Année 1 (L2)

Année 2 (L3)

Année 3 (M)

321  
langages

412  
logique  
modèles

442  
Données massives

421  
algorithmes

311  
tableaux  
listes  
récursivité

411  
arbres  
graphes

443  
Informatique graphique  
3D

Modal

Modal

Modal

P. A.

## Suites naturelles de ce cours en 3A

- 3A :
  - ▶ **INF551 (Logique, Preuve) "INF551 - Computational Logic : from Artificial Intelligence to Zero Bugs"**
  - ▶ PA Math-Info, autres PA.
  - ▶ Cours d'algorithmique ("INF550 - Advanced Algorithms", INF561, INF573, ...)
  - ▶ **Stages de recherche (Logique, Calculabilité, Complexité,...).**
  
- 4A
  
- 5A, 6A, ...

# La pale

- Tous les documents (poly, copie des transparents, notes) sont autorisés.
- Quand on demande un algorithme : du code haut niveau suffit.
- Le poly est un sur-ensemble de ce que nous avons vu en amphis.
- Réviser les PCs.



# Le sondage

- c'est IMPORTANT.