4. Complexité

4.1. Introduction

On ne considère dans ce chapitre que des problèmes décidables.
On ne considère dans ce chapitre que des machines de Turing qui s'arrêtent toujours.

4.1.1. Objectifs

4.1.2. Définition des complexités

Le **temps d'une étape de calcul d'une machine de Turing M** est 1.
Le **temps d'un calcul a m étapes (donc $m+1$ configurations) d'une machine de Turing M** est m.
L'espace d'une configuration $|C|$ d'une machine de Turing, est le nombre de caractères utilisés sur la bande jusqu'au dernier non #.
L'espace d'un calcul d'une machine de Turing, est l'espace le plus grand utilisé par une configuration de ce calcul $\max_i |C_i|$.
La complexité en temps d'une machine de Turing M pour une entrée w, notée $t_M(w)$ est le temps le plus grand des exécutions d'entrée w.
La complexité en espace d'une machine de Turing M pour une entrée w, notée $s_M(w)$ est l'espace le plus grand des exécutions d'entrée w.
La complexité en temps d'une machine de Turing M pour une taille d'entrée n, notée $t_M(n)$ est le temps le plus grand des exécutions d'entrées de longueur n. $t_M(n) = \max_{|w|=n} t_M(w)$.
La complexité en espace d'une machine de Turing M pour une taille d'entrée n, notée $s_M(n)$ est l'espace le plus grand des exécutions d'entrées de longueur n. $s_M(n) = \max_{|w|=n} s_M(w)$.
Donc $t_M : N \to N$, $s_M : N \to N$. En général on s'intéresse au comportement asymptotique.
La déf donnée pour l'espace implique $s_M(n) \geq n$ puisque la configuration initiale comprend l'entrée.
Inapproprié pour l'étude des machines avec peu d'espace. Il faut introduire des machines avec bandes séparées pour l'entrée et la sortie, et ne considérer que l'espace de travail matérialisé par les autres bandes. Pour le temps on a pas ce problème.
Le langage $a\Sigma^*$ est décidable en tant constant (donc sous-linéaire).
Cependant pour la majorité des problèmes intéressants, la machine doit au moins lire l'entrée.
Pour beaucoup de résultats il est nécessaire de supposer $t_M(n) \geq n$ mais ce n'est pas restrictif.
Inégalités fondamentales d'une MT. Pour une machine de Turing M (qui s'arrête toujours), on a
1) $\forall n \in N \;\; s(n) \leq \max(t(n), n)$ (donc $s(n) \leq t(n)$ dès que $s(n) \geq n$)
2) $\forall n \in N \;\; t(n) \leq 2^{Ks(n)}$ avec $K \in R_+$

4.2. Complexité en temps

4.2.1. Théorème d'accélération

On peut toujours accélérer une machine de Turing d'un facteur constant en démultipliant l'alphabet et en traitant des blocs en une fois. (à condition que le temps soit suffisant pour lire l'entrée).
Ainsi lorsqu'on étudie la complexité en temps, les constantes multiplicatives ne sont pas significatives, on les exprime sous la forme asymptotique $O\big(f(n)\big)$, avec $f : N \to R_+$

4.2.2. Changements de modèles

Chaque transition d'une MT bi-infinie est simulée par exactement une transition d'une MT infinie équivalente, donc il n'y a pas de différence dans la complexité en temps.
Une MT à k bandes M est équivalente à une MT à une bande M' telle que $t_{M'}(n) = O\left(t_M^2(n)\right)$

Borne optimale : on peut construire une MT à 2 bandes qui accepte les palindromes en temps linéaire, mais une MT à 1 bande qui accepte les palindromes, prend un temps au moins quadratique.
Une MT à k bandes M est équivalente à une MT à 2 bandes M' telle que $t_{M'}(n) \leq t_M(n) \log t_M(n)$
Une MT (ND) est équivalente à une MTD M' tel que $t_{M'}(n) = O\left(t_M(n) k^{t_M(n)}\right) = 2^{O(t_M(n))}$ avec k le degré maximum dans le graphe des configurations de M.

4.2.3. Classes de complexité en temps

Pour $f: N \to R_+$, $\boldsymbol{TIME}(\boldsymbol{f}(\boldsymbol{n}))$ est l'ensemble des problèmes décidés par une MTD (à plusieurs bandes) en temps $O(f(n))$
Pour $f: N \to R_+$, $\boldsymbol{NTIME}(\boldsymbol{f}(\boldsymbol{n}))$ est l'ensemble des problèmes décidés par une MTND (à plusieurs bandes) en temps $O(f(n))$
$\boldsymbol{P} = \bigcup_{k \in N} TIME(n^k)$ $\boldsymbol{NP} = \bigcup_{k \in N} NTIME(n^k)$
$\boldsymbol{EXPTIME} = \bigcup_{k \in N} TIME\left(2^{n^k}\right)$ $\boldsymbol{NEXPTIME} = \bigcup_{k \in N} NTIME\left(2^{n^k}\right)$
Le fait que ces définitions portent sur les MT multi bandes est sans grande importance car ces classes sont grossières on obtiendrait les mêmes classes si mono bande.
Cependant essentiel de distinguer les MTND des MTD.
Hiérarchie des classes de complexité en temps. $P \subseteq \underset{coNP}{NP} \subseteq EXPTIME \subseteq \underset{coNEXPTIME}{NEXPTIME}$ (par déf, ou inégalités temps espace pour $NP \subseteq EXPTIME$). On peut les représenter par un joli diagramme.
Pour une classe de complexité C, la **classe de complexité duale $\boldsymbol{coC}$** est celle des problèmes complémentaires.
Toutes les classes de complexité définies pour des MTD sont autoduales, par exemple P, $EXPTIME$.
Le théorème d'Immerman et Szelepcsényi établit que toutes les complexités en espace sont autoduales.
On sait par le théorème de hiérarchie, que P est inclus strictement dans $EXPTIME$, mais on ne sait pas si les autres inclusions sont strictes. Savoir si $P \subseteq NP$ est stricte ou non est un problème majeur.
$P = NP$ ssi $EXPTIME = NEXPTIME$
Exemples de problèmes :
PATH : existe-t-il un chemin de s a t dans un graphe G. $\mathrm{PATH} \in P$
Tout langage algébrique est dans P. (peut être décidé en temps cubique, par CKY).
Algorithme CKY décide si w engendre par une grammaire en forme normale quadratique.*
HAM-PATH : existe-t-il un chemin hamiltonien de s a t dans un graphe G. $\mathrm{HAMPATH} \in \mathrm{NP}$
SAT : est-ce qu'une formule du calcul propositionnel est satisfiable ?. $\mathrm{SAT} \in NP$
Un **vérificateur en temps polynomial** pour un langage L, est une MTD qui accepte les entrees de la forme $\langle w, c \rangle$ en temps polynomial en $|w|$ telle que $L = \{w \in \Sigma^* | \exists c \ \ M \text{ accepte } \langle w, c \rangle\}$
L'élément c représente la preuve que $w \in L$. Le vérificateur vérifie que c est bien une preuve.
Cette définition impose que c peut etre choisi de taille polynomiale en w. En effet, la MTD n'utilise qu'une partie de c de taille polynomiale en w, donc on peut supprimer la partie qui reste.
Equivalence NP vérificateur. Un langage $L \in NP$ ssi il existe un vérificateur en temps polynomial de L.
Un langage $L \in NEXPTIME$ ssi il existe un vérificateur en temps exponentiel de L.
Utile pour prouver l'appartenance d'un problème a une classe non déterministe.
L'étoile de Kleene d'un langage dans P est dans P
L'étoile de Kleene d'un langage dans NP est dans NP

4.2.4. NP-complétude

Pour deux problèmes A, B représentés par $L_A \subseteq \Sigma_A^*, L_B \subseteq \Sigma_B^*$, une **réduction polynomiale de A à B** correspond à une fonction $f: \Sigma_A^* \to \Sigma_B^*$ calculable en temps polynomial par une MTD telle que $w \in L_A \Leftrightarrow f(w) \in L_B$. L'existence d'une réduction polynomiale de A à B se note $\boldsymbol{A \leq_P B}$

$\leq_P$ est transitive et réflexive. Intuitivement $A \leq_P B$ signifie que B est plus compliqué que A.

En composant la machine qui calcule f, avec celle qui decide B en temps polynomial on voit que :

Un problème qui se réduit en temps polynomial a un problème P, est P. $A \leq_P B$ et $B \in P \Rightarrow A \in P$

Un problème A est **NP-difficile** ssi tout problème B de NP se reduit a A en temps polynomial ($B \leq_P A$).

Un problème **NP-complet** est un problème NP-difficile et NP.

Intuitivement, un problème est NP-difficile ssi une solution de ce problème permet de résoudre tous les problèmes NP. Les problèmes NP-complets sont donc compris comme les problèmes les plus durs de NP.

Un problème NP-difficile qui se réduit en temps polynomial a un autre problème, entraine que l'autre problème est aussi NP-difficile.

4.2.5. NP-complétude de SAT et 3SAT

La NP-complétude est pertinente car il existe de nombreux problèmes NP-complets, cela est étonnant car on aurait pu a priori imaginer une hiérarchie infinie dans NP. Les premiers problèmes NP-complets historiques sont SAT et 3SAT. La NP-complétude se montre souvent par réduction à l'un de ces deux problèmes fondamentaux. 3SAT est très pratique car se prête bien aux réductions.

SAT : est-ce qu'une formule du calcul propositionnel est satisfiable

3SAT : est-ce qu'une formule en forme normale conjonctive avec au plus 3 littéraux par clause du calcul propositionnel est satisfiable, ou la forme requise est $\varphi = c_1 \wedge \ldots \wedge c_k$ avec $c_i = l_1 \vee l_2 \vee l_3$ avec l_i variable x_k ou négation d'une variable $\overline{x_k}$.

Cook et Levin 1971*. SAT et 3SAT sont NP-complets. (tout problème $NP \leq_P SAT$, et $SAT \leq_P 3SAT$)

4.2.6. Exemples de problèmes NP-complets

HAMPATH est NP-complet. $(3SAT \leq HAMPATH)$

CLIQUE*. existe-t-il une clique de taille k dans un graphe non orienté? est NP-complet.

Une **vertex-cover** d'un graphe est un ensemble de sommets dont tout arc du graphe touche l'un d'eux.

VERTEX-COVER*. Existe-t-il une vertex-cover de taille k dans un graphe non orienté? est NP-complet.

SUBSET-SUM*. Pour une suite finie d'entiers d'entrée, et un entier s, trouver une facon (des indices) de choisir les entiers d'entrée de sorte à ce que leur somme vaille s. SUBSET-SUM est NP-complet.

Un k-coloriage d'un graphe non orienté correspond à une fonction des nœuds du graphe dans un ensemble fini de couleurs et tel que deux sommets adjacents ne sont jamais de la même couleur.

k-COLOR : existe-t-il dans un graphe non orienté, un k-coloriage ?

2-COLOR est dans P. 3-COLOR est NP-complet.

SYMSAT : existe-t-il une affectation symétrique d'une formule en forme normale conjonctive avec au plus 3 littéraux par clause ? SYMSAT est NP-complet.

SETSPLIT : pour un ensemble fini et k sous-ensembles $F_1, \ldots, F_k$, existe-t-il une partition de l'ensemble E en deux E_1, E_2 tel que chacune des 2 partition intersecte tous les sous-ensembles : $\forall i, \forall j\ E_i \cap F_j \neq \emptyset$

SETSPLIT est NP-complet

Le langage défini par une expression rationnelle sans étoile.

Le problème de savoir si le langage d'une expression sans étoile est différent d'un langage A^n fixé est

NP-complet.
Un automate déterministe et complet est synchronisant ssi $\exists w \in \Sigma^* \ |Q \cdot w| = 1$
La longueur du plus petit mot synchronisant est appelée **le délai de synchronisation de l'automate**.
Exo automate synchronisant : TODO

4.3. Complexité en espace

4.3.1. Changement de modèle

Les changement de modèle ont moins d'incidence sur la complexité en espace (qu'en temps).
Une MT à k bandes M est équivalente à une MT à une bande M' qui utilise le même espace.
Savitch 1970*. Pour $s: N \to R_+$ telle que $\exists N \forall n \geq N \ s(n) \geq n$,
une MT (ND) qui fonctionne en espace $s(n)$ équivaut à une MTD qui fonctionne en espace $O\big(s^2(n)\big)$

4.3.2. Classes de complexité en espace

Pour $f: N \to R_+$, $\boldsymbol{SPACE\big(f(n)\big)}$ est l'ensemble des problèmes décidés par une MTD (à plusieurs bandes) en espace $O\big(f(n)\big)$
Pour $f: N \to R_+$, $\boldsymbol{NSPACE\big(f(n)\big)}$ est l'ensemble des problèmes décidés par une MTND (à plusieurs bandes) en espace $O\big(f(n)\big)$
$\boldsymbol{PSPACE} = \bigcup_{k \in N} SPACE\big(n^k\big) = \bigcup_{k \in N} NSPACE\big(n^k\big)$ (par Savitch)
$\boldsymbol{EXPSPACE} = \bigcup_{k \in N} TIME\left(2^{n^k}\right) = \bigcup_{k \in N} NSPACE\left(2^{n^k}\right)$ (par Savitch)
Le fait que ces définitions portent sur les MT multi bandes est sans importance.
Il n'y a plus besoin de distinguer les MTND des MTD par Savitch.

4.3.3. Complexités en temps et en espace

$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME \subseteq NEXPTIME \subseteq EXPSPACE$
Le théorème de hiérarchie montre $P \neq EXPTIME, PSPACE \neq EXPSPACE$.
Hiérarchie des classes de complexité. $P \subseteq \underset{coNP}{NP} \subseteq PSPACE \subseteq EXPTIME \subseteq \underset{coNEXPTIME}{NEXPTIME} \subseteq EXPSPACE$

4.3.4. Exemples de problèmes dans PSPACE

Universalité d'un NFA. Est-ce qu'un automate accepte tous les mots sur son alphabet ? est PSPACE.
Une **formule quantifiée close** est une formule du calcul des prédicats + propositionnel, avec quantificateurs existentiels/universels dont toutes les variables sont liées.
QSAT. Est-ce qu'une formule quantifiée close est vraie ? QSAT joue un rôle pour PSPACE analogue a SAT pour NP
Une formule φ de logique propositionnelle sur des variables $x_1, \dots, x_n$ solution de SAT ssi
$\exists x_1 \exists x_2 \dots \exists x_n \varphi$ solution de QSAT.
QSAT est PSPACE.

4.3.5. PSPACE-complétude

Un problème A est **PSPACE-difficile** ssi tout problème B de PSPACE se reduit a A en <u>temps</u> polynomial ($B \leq_P A$).
Un problème **PSPACE-complet** est un problème PSPACE-difficile et PSPACE.
QSAT est PSPACE-complet.*
Un problème PSPACE-complet qui se réduit en temps polynomial a un autre problème PSPACE, entraine que l'autre problème est aussi PSPACE-complet.

4.3.6. Espace logarithmique

$\boldsymbol{L} = PSPACE(\log n)$ et $\boldsymbol{NL} = NSPACE(\log n)$

Hiérarchie des complexités logarithmiques. $L \subseteq NL = coNL \subseteq P$

Le langage sur un alphabet binaire $\{0,1\}$ dont le nombre de 0 egale le nombre de 1, est dans L.

PATH (dans un graphe orienté) est dans la classe NL.

Reingold 2005. Le problème analogue a PATH dans un graphe non orienté, est L. (preuve très difficile).

Une **fonction $\boldsymbol{f}: \boldsymbol{\Sigma_A^* \to \Sigma_B^*}$ est calculable en espace logarithmique** ssi il existe une MTD (avec bande de sortie) en espace logarithmique qui calcule $f(w)$ pour toute entree w. Comme la taille de $f(w)$ n'est pas prise en compte dans l'espace de la machine, cette taille n'est pas nécessairement logarithmique. Par contre $L \subseteq P$ montre que $|f(w)|$ est polynomial en $|w|$.

Pour deux problèmes A, B représentés par $L_A \subseteq \Sigma_A^*$, $L_B \subseteq \Sigma_B^*$, une **réduction logarithmique de $\boldsymbol{A}$ à $\boldsymbol{B}$** correspond à une fonction $f: \Sigma_A^* \to \Sigma_B^*$ calculable en espace logarithmique par une MTD telle que $w \in L_A \Leftrightarrow f(w) \in L_B$. L'existence d'une réduction polynomiale de A à B se note $\boldsymbol{A \leq_{log} B}$

La composee de deux fonctions calculables en espace logarithmique est calculable en espace logarithmique.

$\leq_{log}$ est transitive et réflexive.

Un problème qui se réduit en espace logarithmique a un problème L, est L. $A \leq_{log} B$ et $B \in L \Rightarrow A \in L$

La satisfiabilité d'une formule en forme normale conjonctive avec 2 littéraux par clause (2SAT) se réduit en espace logarithmique au problème PATH.* $2SAT \leq_{log} PATH$

Corollaire : $2SAT \in P$, donc très différent de 3SAT qui est NP-complet.

Il peut être décidé en temps linéaire si un mot appartient au langage de Dyck D_n^*.

Il peut être décidé en espace logarithmique si un mot appartient au langage de Dyck D_n^*.

4.3.7. NL-complétude

Un problème A est **NL-difficile** ssi tout problème B de NL se réduit a A en espace logarithmique ($B \leq_{log} A$).

Un problème **NL-complet** est un problème NL-difficile et NL.

PATH est NL-complet.*

2SAT est NL-complet. (on montre $PATH \leq_{log} co2SAT$, car $NL = coNL$)

« Est-ce qu'un DFA accepte un mot w ?» est dans la classe L.

« Est-ce qu'un NFA accepte un mot w ? » est NL-complet.

PATH est décidable par une TMD en espace $log^2\ n$

$NL \subseteq L^2$ avec $L^2 = SPACE(\log^2 n)$

Un langage est accepté par un automate multi-tête ssi il est accepté par une MT en espace logarithmique.

4.4. Théorème de hiérarchie

Une **fonction $\boldsymbol{f}: \boldsymbol{N \to N}$ est constructible en temps (resp. en espace)** ssi il existe une MT pour une entrée 1^n le mot $1^{f(n)}$ en temps $O(f(n))$ (resp. en espace $O(f(n))$)

Pour la définition de constructible en espace pour f telle que $f(n) \leq n$, il faut utiliser des MT avec une bande d'entrée et une bande de sortie séparées comme cela a été fait pour l'espace logarithmique.

La classe des fonctions constructibles en temps ou en espace contient toutes les fonctions usuelles $n \mapsto E(\log n)$, $n \mapsto E(\sqrt{n})$, …, et est close par somme, produit, exponentiation, composition, et toutes les opérations raisonnables.

Théorème de hiérarchie. Si une fonction $g: N \to N$ est négligeable devant une fonction $f: N \to N$ constructible en temps (resp. en espace), l'inclusion $TIME\big(g(n)\big) \subset TIME(f(n))$ (resp. $SPACE\big(g(n)\big) \subset SPACE(f(n))$) est stricte.

Corollaire : $P \subset EXPTIME, PSPACE \subset EXPSPACE$.

4.5. Machines alternantes

4.5.1. Définitions et exemples

4.5.2. Complémentation

4.5.3. Automates alternants

4.5.4. Classes de complexité

4.6. Compléments.