Classes de complexité

February 14, 2025

1 Classes de complexité pour un problème de décision

1.1 Complexité

Définition 11.1

Soit $\mathcal{P}$ un problème abstrait défini par une fonction $f : I \to S$. On considère un algorithme $\mathcal{A}$ "abstrait" qui le résout.
On y associe un problème *concret* en imposant des choix sur *l'encodage* des entrées et des sorties.
La *complexité du problème concret* correspond à la complexité de l'implémentation de l'algorithme (sa réalisation concrète) pour le choix d'encodage.
C'est-à-dire un ordre de grandeur du nombre d'opérations élémentaires réalisées par l'implémentation de l'algorithme pour une entrée en fonction de la *taille de l'entrée* (espace occupé en mémoire pour son encodage).

Remarque 11.2 - *cas du graphe*

La taille de l'encodage pour un graphe :

- matrice d'adjacence : $\mathcal{O}(|S|^2)$
- liste d'adjacence : $\mathcal{O}(|S| + |A|)$

Exemple 11.3 - *taille de l'endage des entiers*

Un entier n s'encode en $\mathcal{O}(\log_2(n))$, taille de l'écriture binaire.
Une complexité en $\mathcal{O}(n)$ où n est l'entier en entrée, est donc exponentielle pour un algorithme dont l'entrée est n, est $\mathcal{O}(n) = \mathcal{O}(2^{\log_2(n)})$. C'est donc exponentiel en la taille de l'entrée !

Remarque 11.4 - *relative aux problèmes*

Selon l'importance des entiers dans le problème, on peut "ignorer" qe les entier prennent beaucoup de place :

- pour un graphe pondéré, la taille des entiers est ignorée.
- pour le problème du sac à dos, la taille des entiers est prise en compte.

Exemple 11.5

- **Instance** : un entier n
- **Question** : $v_2(n) \geq v_3(n)$? (en parlant de valuation 2-adique)

```
1  Algo(n):
2      tmp = n
3      tant que tmp%2 == 0 ou tmp%3==0:
4          si tmp%6==0:
5              tmp = tmp/6
6          sinon si tmp%2==0:
7              renvoyer Vrai
8          sinon si tmp%3==0:
9              renvoyer Faux
10     renvoyer Vrai // v_2(n) = v_3(n)
```

La complexité temporelle est en $\mathcal{O}(\log_6(n)) = \mathcal{O}(\log_2(n)) = \mathcal{O}(|n|)$. C'est une complexité linéaire en la taille de l'entrée : $|n| = \log_2(n)$.

Moralité : toujours se ramener à une complexité en fonction de la taille de l'entrée.

Remarque 11.6

La taille de l'entrée est à rapprocher de la taille du mot de $\{0;1\}$ que l'on écrit initialement sur une machine de Turing exécutant l'algorithme sur ce mot.

1.2 Classe P

Définition 11.7 - *classe P*

On appelle *classe P* (ou *classe PTIME*) l'ensemble de *problèmes de décision* que l'on peut résoudre à l'aide d'un algorithme de complexité polynomiale en la taille de l'entrée (pour un choix d'encodage).

Remarque 11.8

Généralement, on donne la complexité sous la forme $\mathcal{O}(|e|^k)$ avec k un entier (degré de l'expression polynomiale de la complexité) et $|e|$ désignant la taille de l'entier.

Remarque 11.9

Les problèmes de la classe P sont raisonnablement traitables. (Attention, on parle toujours uniquement de problème de décision).
Pour considérer des problèmes d'optimisation, on se ramène à des problèmes de décision en introduisant un seuil à dépasser pour la fonction de coût.

Remarque 11.10 - *classe EXP*

La *classe EXP* (ou *classe EXPTIME*) est la classe de problèmes de décision pour lesquels il existe un algorithme de résolution de complexité exponentielle :

$$\mathcal{O}(P(|e|)\exp(|e|))$$

où P est un polynôme réel et $|e|$ est la taille de l'entrée.

Remarque 11.11 - *comparaison entre EXP et P*

On a P $\subset$ EXP car $e^n \gg P(n)$ pour $n \to +\infty$.

Remarque 11.12

La complexité, soit le nombre d'opérations élémentaires, dépend du modèle de calcul choisi. Il en existe plusieurs.
En MPI, on se place dans le cadre d'un modèle de calcul intuitif, qui correspond au fonctionnement d'un ordinateur exécutant un programmation en C ou en OCaml. Les opérations élémentaires sont alors les suivantes :

- écriture
- lecture
- opération arithmétique
- etc (cf MP2I)

De plus on fait l'hypothèse que la machine dispose d'une *mémoire infinie* :

- la pile d'appels ne déborde jamais
- il n'y a pas de limite à la quantité de données allouable sur la pile, le tas ou bien le segment de données.

Un autre modèle de calcul classique est celui des machines de Turing : on compte le nombre de transitions empruntées au cours d'un calcul.

Exemple 11.13 - *problème du dernier exemple*

Le problème précédent (valuations) est de classe P car linéaire et particulier polynomial.

Exemple 11.14 - *problème 2-SAT*

- **Instance** : φ une formule de la logique propositionnelle sous forme normale conjonctive dont les clauses sont de taille au plus 2.
- **Question** : φ est-elle satisfiable ?

On a 2-SAT $\in$ P. En effet, on peut construire le graphe d'implications puis lui appliquer l'algorithme de Kosaraju qui renvoie les composantes fortement connexes du graphe d'implications. On vérifie que pour $x \in \mathcal{V}_\varphi$, x et $\neg x$ ne sont pas dans la même composante fortement connexe. $= C_1 \wedge \cdots \wedge C_k$ de taille $\mathcal{O}(k) = |\varphi|$.

Construction du graphe :

- au plus $4k$ sommets (un par littéral)
- au plus $2k$ arêtes (deux par clause)

se fait en $\mathcal{O}(k)$
Kosaraju : $\mathcal{O}(|\mathcal{A}| + |\mathcal{S}|)$ (deux parcours), se fait en
à finir

1.3 Classe NP

Définition 11.15

Pour un problème de décision, on appelle *certificat* une donnée que l'on peut ajouter à l'entrée permettant de vérifier que l'entrée est une instance positive.

Exemple 11.16

Pour le problème SAT, la donnée d'une valuation est un certificat. En effet, on vérifie que la valuation satisfait la formule pour vérifier qu'elle est satisfiable donc que c'est une instance positive. Pour toute formule φ, il existe une valuation v telle que l'algorithme vérificateur confirme que φ est satisfiable est satisfable.

Exemple 11.17 - *lié au problème K-COLOR*

- **Instance** : $G = (S, A)$ un graphe, $k \in \mathbb{N}$.
- **Question** : sur l'existence de $c : S \to [\![1,\, k]\!]$ un coloriage des sommets à k couleurs tel que :

$$\forall (u, v) \in A,\, c(u) \neq c(v)$$

On peut chosiir la donnée d'une fonction $c : S \to [\![1,\, k]\!]$ comme certificat.

Définition 11.18 - *classe NP*

La classe NP est l'ensemble des problèmes de décision pour lesquels il existe un *algorithme vérificateur* permettant de *vérifier* qu'une instance est positive à partir d'un couple (instance, certificat) en temps polynomial en la taille de l'instance, avec un certificat de taille polynomiale en fonction de cette taille.

Exemple 11.19 - *problème SAT*

Soit φ une instance positive (une formule satisfiable) et v un certificat associé à φ (un modèle de φ) :

$$|v| = |\mathcal{V}_\varphi| = |\varphi|$$

L'algorithme vérificateur repose sur un parcours de l'arbre représentant φ en $\mathcal{O}\Big(|\varphi|\Big)$

Remarque 11.20

L'algorithme vérificateur prend en entrée :

- un encodage de l'instance
- un encodage du certificat

deux chaînes de caractères ! Si on choisit un certificat booléen, pour un instance i^+ positive, on note `verif` l'algorithme tel que :

- `verif(`i^+`,true)` renvoie `true`
- `verif(`i^+`,false)` renvoie un booléen arbitraire

et pour i^- une instance négative :

- `verif(`i^-`,true)` renvoie `false`
- `verif(`i^-`,false)` renvoie `false`

Si on a un tel algorithme, alors on sait résoudre le problème avec cet algorithme et le certificat n'est pas nécessaire. Donc on ne pas prendre directement la solution comme un certificat ? A montrer à ChatGPT, il saura m'expliquer.

Définition 11.21 - *formelle de classe NP*

Soit $\mathcal{P}$ un problème de décision défini par sa fonction de prédicat $f : I \to \{Vrai, Faux\}$. $\mathcal{P}$ est de classe NP lorsqu'il existe :

1. Σ un alphabet permettant d'encoder un ensemble C de certificats
2. $\mathcal{P}'$ un problème de prédicat :
$$f' : I \times \Sigma^* \to S = \{Vrai, Faux\}$$
3. g une fonction polynomiale telle que pour tout $e \in I$, $f(e)$ est Vrai ssi :
$$\exists c \in C,\ |c| \leq g(\texttt{e}) \quad \text{et} \quad f'(e, |c|) = Vrai$$
($|\cdot|$ désigne la taille de représentation)
4. le problème $\mathcal{P}'$ est dans $\mathcal{P}$.

L'algorithme en temps polynomial qui résout $\mathcal{P}'$ est appelé *algorithme vérificateur*.

Remarque 11.22 - *sur la définition*

Dans la 3. : $|c| \leq g(|e|)$ signifie que le certificat est de taille polynomiale en fonction de la taille de l'instance.

Remarque 11.23

Pour montrer en pratique qu'un problème est dans NP :

- on exhibe un certificat de taille polynomiale en fonction de l'instance
- on donne un algorithme vérificateur de complexité polynomiale en fonction de l'instance.

Exemple 11.24

On considère le problème CYCLE HAMILTONIEN :

- **Instance** : $G = (S, A)$ un graphe
- **Question** : Existe-t-il un cycle passant exactement par tous les sommets de S ?
- **Certificat** : une permutation σ de S.

```
1 | VERIFICATEUR(G,σ):
2 |     n = |S|
3 |     vérifier que σ est dans S_n.
4 |     pour i=1 jusqu'à n-1:
5 |         Si (σ(i),σ(i+1)) ∉ A:
6 |             renvoyer false
7 |     renvoyer ((σ(n),σ(1)) ∈ A)
```

Cet algorithme est de complexité en $\mathcal{O}(|\mathcal{S}|)$ donc en temps polynomial en fonction de la taille de G en mémoire : $|S| + |A|$ et $|\sigma| = |S|$ également polynomiale.

Proposition 11.25 - *classe P implique classe NP*

P $\subset$ NP.

Démonstration

On peut prendre n'importe quel certificat, on utilise l'algorithme résolvant $\mathcal{P}$ en temps polynomial comme algorithme vérificateur.

Remarque 11.26 - *sens réciproque*

Le sens réciproque est un problème ouvert. On ne connaît aucun exemple de problème de classe NP pour lequel on a une démonstration du fait qu'il n'est pas de classe P.

1.4 Le problème d'optimisation

Pour étudier la classe de complexité d'un problème d'opitimisation , one le transforme en problème de décision en ajoutant un *seuil* à l'entrée. On répond alors à la question :
"Existe-t-il une solution réalisable dont la valeur atteinte par la fonction de coût est supérieur (maximisation) ou inférieur (minimisation) à un seuil s ?"

Le problème de décision associé est "moins compliqué". En particulier, si pour le problème de décision, on ne sait pas trouver d'algorithme en temps polynomial, on ne saura pas non plus pour le problème d'optimisation : il est légitime de s'intéresse à des *algorithmes d'approximation*.

Remarque 11.27

Si le problème de décision est de classe P, on peut s'approcher le plus possible de la valeur optimale par dichotomie en utilisant plusieurs appels à l'algorithme de résolution en temps polynomial.

On introduit la notation hors programme suivante.
La classe des problèmes d'optimisation dont le problème de décision associé est de classe P s'appelle la classe PO. De même, pour la classe NP, la classe NPO.

2 Réduction polynomiale

Définition 11.28 - *réduction polynomiale*

Soit deux problèmes de décision $\mathcal{P}_1$ et $\mathcal{P}_2$ définis par f_1 et f_2 leurs fonctions de prédicats associées définies sur un I_1 et I_2 respectivement. On dit que $\mathcal{P}_1$ *se réduit à* $\mathcal{P}_2$ *de façon polynomiale* (noté $\mathcal{P}_1 \leq_m^{\mathrm{P}} \mathcal{P}_2$) lorsqu'il existe $g : I_1 \to I_2$ calculable avec un algorithme de complexité polynomiale telle que :

$$\forall e \in I_1,\ f_1(e) = f_2\Big(g(e)\Big)$$

Remarque 11.29

On a la même définition de réduction many-to-one que lorsqu'on discute de décidabilité, on uniquement l'hypothèse supplémentaire de l'existence d'un algorithme polynomial qui transforme les instances de $\mathcal{P}_1$ en des instances de $\mathcal{P}_2$.

Proposition 11.30 - $\leq_m^{\mathrm{P}}$ *est presque une relation d'ordre*

La relation $\leq_m^{\mathrm{P}}$ est réflexive, transitive, mais non antisymétrique.

Voir démo 1

Proposition 11.31

Soit $\mathcal{P}_1$ et $\mathcal{P}_2$ deux problèmes de décision. Si $\mathcal{P}_1 \leq_m^{\mathrm{P}} \mathcal{P}_2$ et $\mathcal{P}_2$ est de classe P, alors $\mathcal{P}_1$ est de classe P.

Démonstration

Il suffit de combiner l'algorithme calculant g et l'algorithme qui résout $\mathcal{P}_2$ en temps polynomial.

Voir suite par Gwénolé.

3 NP-completude, NP-difficile

3.1 Définitions et problème SAT

Définition 11.32

Un problème de décision est *NP-difficile* ou *NP-dur* lorsqu'il est possible de se ramener à tout problème de classe NP par une réduction polynomiale depuis ce problème.
Moralement, il est plus difficile que tous les problèmes NP, mais on peut s'y ramener.

Définition 11.33

Soit $\mathcal{P}$ un problème de décision. $\mathcal{P}$ est NP-difficile si et seulement si :

$$\forall \mathcal{P}' \in \mathrm{NP},\ \mathcal{P}' \leq_m^{\mathrm{P}} \mathcal{P}$$

Si de plus $\mathcal{P}$ est dans NP, alors $\mathcal{P}$ est *NP-complet*.

Remarque 11.34

Pour montrer qu'un problème est NP-diffficile, on montre que SAT se réduit polynomialement à ce problème : SAT $\leq_m^{\mathrm{P}} \mathcal{P}$. On utilise donc la transitivité de $\leq_m^{\mathrm{P}}$.