Classes de complexités

Montaigne 2023-2024

– mpi23@arrtes.net –

La **calculabilité** permet de caractériser les problèmes susceptibles d'être résolus par des algorithmes. Mais elle ne dit rien de leurs performances.

La **complexité** interroge les contraintes de ressources, temporelles ou spatiales, nécessaires pour les mettre en œuvre.

Il apparaît alors que certains problèmes, **a priori** différents, présentent des similitudes de performance qui mènent à les **relier** par une opération appelée **réduction polynomiale**, offrant de fait une vision plus globale des algorithmes en termes de **classes de complexités**.

Notion de problème

Problème de décision

Un **problème de décision** est un type particulier de problème qui répond de manière binaire à des questions. La définition suivante, rencontrée au chapitre sur la **décidabilité**, formalise ce concept.

Définition 1

- Un **problème de décision** sur un domaine d'entrées E est défini par une fonction totale f de E vers $\mathbb{B}$, ensemble des booléens.
- Un **algorithme** A **résout** un problème de décision f si, pour toute entrée $e \in E$, l'algorithme A appliqué à e termine en un temps fini et produit le résultat $f(e)$.
- Chaque élément $e \in E$ du domaine d'entrées est appelé une **instance** du problème.

Problème de décision

De manière équivalente, un problème de décision sur un domaine d'entrées E peut être être défini par le sous-ensemble $P \subseteq E$ des instances $e \in P$ pour lesquelles $f(e) = \mathrm{V}$.
P est l'ensemble d'**instances positives**.

Ce chapitre place les problèmes de décision au centre des problématiques mais dautres types de problèmes existent pour lesquels la réponse n'est pas binaire. Toutefois il est possible de les relier à des problèmes de décision.

Problème de décision

Quelques problèmes de décision classiques décidables et leurs domaines d'entrées associés.

- Y a-t-il un doublon dans un tableau a d'entiers ?
  Domaine des entrées : tableaux d'entiers, sans restriction de taille.
- Un nombre n est-il premier ?
  Domaine des entrées : nombres entiers positifs, sans restriction.
- Y a-t-il un chemin entre les sommets s et t d'un graphe g ?
  Domaine des entrées : triplets formés d'un graphe et de deux sommets.
- Un graphe g peut-il être colorié avec quatre couleurs ?
  Domaine des entrées : graphes non orientés.
- Un mot m est-il accepté par un automate fini a ?
  Domaine des entrées : paires formées d'un automate fini et d'un mot.
- Une grammaire algébrique $\mathcal{G}$ peut-elle générer le mot vide ?
  Domaine des entrées : grammaires algébriques.

Problème de recherche

Outre l'existence d'une solution à un problème, on peut vouloir rechercher une solution. Le domaine des entrées est alors l'ensemble des configurations possibles, le domaine des résultats l'ensemble des opérations menant à une ou plusieurs solutions valides.

Définition 2

- Un **problème de recherche** sur un domaine d'entrées E et un domaine de solutions S est défini par la donnée d'une relation binaire $\mathcal{R} \subseteq E \times S$.
- Un **algorithme** A résout un problème de recherche $\mathcal{R}$ si, pour toute entrée $e \in E$, l'algorithme A appliqué à e produit une solution s telle que $\mathcal{R}(e, s)$, lorsqu'une telle solution existe.

Problème de recherche

Cette définition couvre des situations où le problème admet plusieurs solutions ou aucune solution pour une entrée donnée.

Dans le cas où la solution pour chaque entrée est unique, c'est-à-dire le cas où la relation $\mathcal{R}$ est fonctionnelle, on parle de **problème de calcul** : un unique résultat est **produit** à partir d'une entrée. On dit également **calculé** à partir d'une entrée.

Techniquement, un problème de décision peut être considéré comme un cas particulier de problème de recherche dans lequel l'ensemble S des solutions est réduit aux booléens et la relation $\mathcal{R}$ est fonctionnelle et totale.

Problème de recherche

Cuelques problèmes de recherche et les domaines de résultats associés.

- Trouver un doublon dans un tableau a d'entiers.
  Domaine des solutions : nombres entiers.
- Trouver un chemin entre les sommets s et t d'un graphe g.
  Domaine des solutions : séquences de sommets.
- Trouver une coloration des sommets d'un graphe g.
  Domaine des solutions : fonctions partielles des entiers (numéros des sommets) vers les entiers (numéros des couleurs).

Problèmes d'existence et de vérification

D'un problème de recherche, on peut extraire plusieurs problèmes de décision.

Définition 3

Soit un problème de recherche défini par une relation $\mathcal{R} \subseteq E \times S$.

- Le **problème d'existence d'une solution** associé à $\mathcal{R}$ est le problème de décision sur le domaine d'entrées E défini par la fonction $f_{\text{exist}} : E \to \mathbb{B}$ telle que $f_{\text{exist}}(e) = \text{V}$ si et seulement s'il existe un $s \in S$ avec $\mathcal{R}(e, s)$.
- Le **problème de vérification d'une solution** associé à $\mathcal{R}$ est le problème de décision dont le domaine d'entrées est formé des paires (e, s), $e \in E$ étant une entrée et $s \in S$ étant une solution candidate, défini par la fonction caractéristique de $\mathcal{R}$, c'est-à-dire la fonction $f_{check} : E \times S \to \mathbb{B}$ telle que $f_{check}(e, s) = \text{V}$ si et seulement si $\mathcal{R}(e, s)$.

Problème d'optimisation

Quand un problème admet potentiellement plusieurs solutions, on peut chercher à en identifier une qui satisfait un critère particulier. On la qualifie de **meilleure** solution possible. Il s'agit alors d'un raffinement d'un problème de recherche appelé **problème d'optimisation**.

Définition 4

- Un **problème d'optimisation** sur un domaine d'entrées E et un domaine de solutions S est défini par une relation $\mathcal{R} \subseteq E \times S$ et une **fonction de coût** $c : S \to \mathbb{R}^+$ telles que, étant donné un $e \in E$ et une solution s telle que $\mathcal{R}(e, s)$, $c(s)$ est minimale.
- Un **algorithme** A **résout** un problème d'optimisation si, pour toute entrée $e \in E$ et s'il existe au moins une solution, l'algorithme A appliqué à e produit une solution de l'instance e de coût minimal.

Problème d'optimisation

Deux problèmes d'optimisation déjà rencontrés dans le cours d'informatique.

- Trouver un **plus court chemin** entre les sommets s et t d'un graphe g. Les solutions sont des chemins, et la fonction de coût est la somme des poids des arcs d'une solution.
- Trouver une **coloration d'un graphe** g utilisant un nombre minimal de couleurs. Les solutions sont des coloriages, et la fonction de coût est le nombre de couleurs utilisées par une solution.

Problème de seuil

Tout problème d'optimisation peut se ramener à un problème de décision en fixant un **seuil** et en reformulant le problème.

Définition 5

Soit un **problème d'optimisation** sur un domaine d'entrées E et un domaine de solutions S, défini par une relation $\mathcal{R} \subseteq E \times S$ et une fonction de coût $c : S \to \mathbb{R}^+$. Pour tout choix d'un **seuil** $c_0 \in \mathbb{R}^+$, la fonction $f_{c_0} : E \to \mathbb{B}$ définie par :

$$f_{c_0}(e) = \mathrm{V} \quad \Leftrightarrow \quad \exists s \in S \, \mathcal{R}(e, s) \text{ et } c(s) \leqslant c_0$$

définit un **problème de décision**.

Problème de seuil

Les problèmes de décision suivants peuvent être associés aux problèmes d'optimisation vus plus haut.

- Existe-t-il un chemin de longueur inférieure ou égale à 100 entre les sommets s et t d'un graphe g ?
- Un graphe g peut-il être colorié avec trois couleurs ?
- Un graphe g peut-il être colorié avec quatre couleurs ?

Modèle de calcul

Modèle de calcul

Le chapitre précédent à défini un **algorithme** comme un programme C ou OCaml s'exécutant sur une machine n'ayant pas de limite de mémoire.

Cette définition est ici conservée en ajoutant des critères d'évaluation des ressources de temps et d'espace utilisées par ces algorithmes. Ce qui constitue notre **modèle de calcul**.

Mode d'énoncé des complexités temporelles

Pour évaluer la complexité temporelle d'un algorithme, rappelons qu'un problème algorithmique est défini sur un **domaine d'entrées infini**.

Par ailleurs, on mesure l'**ordre de grandeur asymptotique** du nombre d'**opérations élémentaires**, évalué en fonction de la **taille de l'entrée**.

Dans l'étude des classes de complexité, on considère systématiquement la **complexité dans le pire cas** des algorithmes considérés. En effet, de la même manière que la décidabilité demande à un algorithme de toujours renvoyer une réponse en un temps fini, on attend des algorithmes qu'ils renvoient toujours une réponse dans la limite de temps impartie.

Opérations élémentaires

Les **opérations élémentaires** sont celles effectivement réalisables en un **temps constant** comme les accès en lecture ou en écriture à la mémoire, les opérations arithmétiques élémentaires, les comparaisons de valeurs élémentaires.

Sont en revanche exclus, par exemple, l'accès à un élément d'une liste chaînée autre que sa tête, la concaténation de deux chaînes de caractères.

Une opération arithmétique sur un **grand** entier, c'est-à-dire un entier dépassant des capacités des entiers machine, est également généralement considérée comme non-élémentaire.

Taille de l'entrée

La **taille d'une entrée** est donnée par l'espace mémoire nécessaire au stockage de cette entrée.

Une mesure exacte nécessiterait de connaître la manière dont les données sont représentées et organisées en mémoire.

Mais dans la recherche d'ordres de grandeur des complexités, l'ordre de grandeur, et non le nombre exact de bits utilisés, de la taille d'une entrée donnée suffit.

Taille de l'entrée

En pratique :

- une donnée sur un domaine fini, comme un booléen, a une **taille unitaire**,
- une collection, comme un tableau, a une **taille proportionnelle** au produit du nombre d'éléments dans la structure par la taille d'un élément individuel.

Taille de l'entrée

Des subtilités apparaissent lorsque les entrées comportent des nombres. Par exemple, dans le problème consistant à déterminer si un nombre n donné en entrée est premier, quelle est la taille d'une telle entrée ?

En se limitant aux entiers machine, il serait possible de leur donner une taille unitaire. Mais on serait alors dans le cas d'un problème sur un domaine fini, ce que nous avons exclu de notre étude. C'est pourquoi nous considérons plutôt les entiers mathématiques qui forment un domaine infini.

Taille de l'entrée

Dès lors, quelle quantité de mémoire permet la représentation d'un entier de taille arbitraire ?

Sauf à utiliser une représentation particulièrement inefficace, la taille de l'entier n **n'est pas** n. En général, un entier n de taille arbitraire est représenté par la séquence des chiffres utilisés dans son écriture dans une certaine base. Quelle que soit celle-ci, la taille de sa représentation est proportionnelle au **logarithme** de n.

En ne s'intéressant qu'aux ordres de grandeur, le choix de la base n'a donc aucun impact sur les résultats de complexité obtenus. Aussi peut-on retenir qu'en toutes circonstances, la taille d'un entier n est proportionnelle au logarithme de n.

Classe P

Complexité polynomiale

Un algorithme est de **complexité polynomiale** si, pour un certain entier k, celle-ci est en $O(n^k)$. Plusieurs raisons justifient l'intérêt de tels algorithmes.

Les algorithmes de complexité polynomiale sont **plutôt efficaces** et les **algorithmes non polynomiaux** sont **certainement inefficaces**.

Un algorithme de complexité linéaire $O(n)$ est toutefois plus efficace qu'un algorithme de complexité quadratique[1] $O(n^2)$ ou encore qu'un algorithme de complexité polynomiale en $O(n^{100})$, **certainement** peu efficace dès que n prend des valeurs importantes.

1. Il convient toutefois de prendre garde aux constantes cachées dans la notation grand-O.

Complexité polynomiale

Certains algorithmes non polynomiaux peuvent se révéler être **assez efficaces**, en moyenne, ou pour des jeux restreints de données. Néanmoins, pour ces derniers, même avec des jeux de données de taille relativement faible, leur temps de calcul **peut devenir prohibitif**, les rendant inutilisables en pratique.

Complexité polynomiale

La notion de complexité polynomiale est **robuste** car indépendante de toute technologie.

La définition d'un temps de calcul ne dépend que du modèle de calcul adopté.

Comme il est possible de prouver l'équivalence entre tous les modèles connus[2], on peut donc également montrer que la complexité se transforme de manière polynomiale.

2. À l'exception du modèle quantique...

Complexité polynomiale

Les algorithmes polynomiaux forment une **classe stable**. La composition de deux algorithmes polynomiaux reste polynomiale et un algorithme construit polynomialement à partir d'appels à des fonctions de complexité polynomiale reste polynomial.

En cherchant à **classer** les **problèmes de décision** en fonction de la complexité de leurs algorithmes, ceux qui peuvent être résolus en temps polynomial en la taille de l'entrée, considérés comme **raisonnables**, ont donc une place bien particulière dans l'univers des algorithmes. Ils définissent une **classe de complexité** notée P.

Classe P

Définition 6

La **classe** P est l'ensemble des **problèmes de décision** qui peuvent être résolus par un algorithme de complexité temporelle majorée asymptotiquement par un polynôme en la taille de l'entrée.

Cette définition appelle deux remarques importantes.

- Un élément de la classe P est un problème de décision et non un algorithme. C'est l'existence d'un algorithme de la bonne complexité qui signe l'appartenance du problème à la classe P.
- Cette définition ne s'applique qu'à des **problèmes de décision**. Les autres problèmes, comme les problèmes d'optimisation, doivent être reformulés en termes de problèmes de décision.

Classe P

La définition de la classe P demande de pouvoir majorer la complexité d'un algorithme par un polynôme, sans aucune contrainte sur la forme de ce polynôme, et en particulier sans aucune contrainte sur son degré. Cette souplesse dans la définition a trois conséquences notables.

Classe P

La classe P est extrêmement **robuste** : on peut combiner à l'envi des algorithmes de complexité polynomiale, l'algorithme obtenu est encore polynomial.

Cettre propriété est largement utilisée dans les prochaines sections.

Classe P

Cette robustesse fait que l'analyse de complexité et la mesure de la taille des entrées n'ont généralement pas besoin d'être très précises pour établir qu'un algorithme est polynomial. Peu importe que l'on majore la complexité par un polynôme de degré deux fois supérieur à ce qui serait nécessaire, le résultat est toujours un polynôme.

En particulier, si un algorithme prend en entrée un tableau de N entiers pris dans l'intervalle $[0, N[$, la taille réelle de cette entrée est de l'ordre de $N \log N$ pour tenir compte de la taille de chacun des entiers du tableau, mais on peut faire l'analyse de complexité en simplifiant cette taille en N sans risque d'altérer la conclusion.

Classe P

La classe P contient également, du moins en théorie, des problèmes dont la complexité optimale est donnée par un polynôme de degré élevé. Et malgré son appartenance à P, on ne peut pas dire d'un tel problème qu'il puisse être résolu **en pratique**. Même pour des tailles d'instances N modestes, l'exécution d'un algorithme de complexité $\Theta(N^{100})$ nécessite vite des temps de calcul inenvisageables.

Classe P

L'existence d'un bon algorithme suffit à caractériser l'appartenance à P. Mais il est tout à fait envisageable qu'un problème de la classe P admette également des solutions dont le temps d'exécution est exponentiel, ou pire, en la taille de l'entrée. Les exemples précédents ont illustré ce fait.

En conséquence, pour un problème de décision donné, ne connaître que des algorithmes de complexité exponentielle ne suffit pas à déduire que ce problème n'appartient pas à la classe P : un algorithme polynomial existe peut-être, qui n'a pas encore été découvert.

Classe P

Démontrer l'impossibilité d'un algorithme polynomial pour un problème de décision est un problème potentiellement plus délicat, similaire en nature à une démonstration d'indécidabilité.

En sortant du cadre des problèmes de décision, on connaît déjà un exemple de problème algorithmique pour lequel l'impossibilité d'une solution polynomiale est facile à justifier : la **déterminisation** d'un automate fini. En effet, la taille de l'automate déterministe à construire est potentiellement exponentielle en celle de l'automate non déterministe pris en entrée. Le temps nécessaire à sa construction est donc nécessairement au moins exponentiel.

Problème du diviseur commun

Désignons par COMMDIV le problème qui consiste à déterminer si un couple d'entiers possède un diviseur commun non trivial [3].

Une **instance** de ce problème est un couple d'enties (a, b) et le **problème de décision** COMMDIV est de savoir si, en dehors de 1, a et b possèdent un diviseur commun.

3. Différent de 1 !

Problème du diviseur commun

La fonction `com_div(a,b)` ci-dessous en langage C renvoie `1` si `a` et `b` admettent un diviseur commun non trivial, `0` sinon. Une telle fonction existe et s'appuie sur le résultat renvoyé par la fonction `gcd` qui met en œuvre de l'algorithme d'Euclide : `gcd(a,b)` calcule le `pgcd` de `a` et `b`, avec `a > b`.

```
int gcd(int a, int b) {
  while (b != 0) {
    int tmp = b;
    b = a % b;
    a = tmp;
  }
  return a;
}

bool com_div(int a, int b) {
  return gcd(a, b) > 1;
}
```

Problème du diviseur commun

Si `a` et `b` sont représentés sur n bits, la complexité[4] de `gcd` est en $O(n)$. Un fois `gcd(a,b)` calculé, la fonction `com_div` constitue une mise en œuvre algorithmique de résolution du problème initial.

Pour résoudre ce problème, on résout d'abord un problème visiblement plus difficile, à savoir calculer un pgcd.

Un programme naïf, qui incrémente un diviseur d en partant de la valeur 2 jusqu'à ce que d divise à la fois a et b ou jusqu'à atteindre le plus petit de ces deux entiers, permet le calcul du pgcd des deux entiers. Mais cet algorithme est de complexité exponentielle en la taille n des données. Si a et b sont représentés sur n bits, le nombre d'itérations peut être de l'ordre de 2^n. En conséquence, cet algorithme ne permet pas de savoir si le problème de décision initial est dans la classe P.

4. Voir le théorème de Lamé pour une justification du résultat.

Problème du chemin dans un graphe

Considérons à présent le problème GRAPHPATH qui, étant donné un graphe G et deux de ses sommets s et t, détermine s'il existe un chemin reliant s et t dans G.

Une **instance** de ce problème est la donnée du graphe G et des sommets s, t. Le **problème de décision** GRAPHPATH est de savoir si un chemin $s \to t$ existe dans G.

Problème du chemin dans un graphe

Une solution naïve examine successivement tous les chemins de s, jusqu'à en trouver un d'extrémité t.

Cette procédure n'est en général pas de complexité polynomiale car le nombre de chemins dans un graphe peut être une fonction exponentielle de la taille du graphe. Chaque fois qu'un sommet est choisi pour construire un chemin, de nouveaux choix doivent être opérés entre les différents arcs issus de ce sommet.

Problème du chemin dans un graphe

Au lieu de parcourir tous les chemins, on peut se contenter de marquer les sommets i pour lesquels existe un arc $s \to i$, puis de marquer les sommets j qu'on peut atteindre à partir de ceux déjà marqués. Cette procédure est répétée jusqu'à marquer t ou jusqu'à stabilisation de l'ensemble des sommets marqués.

Le nombre de sommets marqués augmente au moins de 1 entre deux étapes consécutives de l'algorithme[5]) donc le nombre d'étapes est borné par le nombre n de sommets du graphe. Et chaque étape, à savoir examiner les arcs issus des sommets déjà marqués, est polynomiale. En conséquence, cet algorithme est de complexité polynomiale en la taille du graphe qui caractérise bien la taille des données.

5. Sinon celui-ci est terminé.

Problème du chemin dans un graphe

On peut s'interroger sur l'influence du choix de la structure de données adoptée pour représenter le graphe influe. Trouver une structure optimale constitue d'ailleurs une question importante pour une résolution pratique du problème.

Néanmoins la complexité reste polynomiale indépendamment du choix de la structure, même avec une représentation brutale de la liste des arcs [6].

6. Sans accès direct aux arcs de source donnée !

Problème P sur les entiers

Un algorithme déterminant si un nombre $p \in \mathbb{N}$ est premier en un temps proportionnel à $\sqrt{p}$ existe : il suffit de tester tous les diviseurs entre 2 et $\sqrt{p}$. Si on se rappelle que la taille d'une instance $p \in \mathbb{N}$ est de l'ordre de $n = \lfloor \log_2 p \rfloor$ alors la complexité de l'algorithme est **exponentielle** en la taille de p puisqu'en en $2^{n/2}$. Un tel algorithme est parfois qualifié de **pseudo-polynomial**.

La question de l'appartenance véritable du test de primalité à la classe P est longtemps restée ouverte et n'a été tranchée qu'en 2002 avec la publication de l'algorithme AKS, dont la complexité est effectivement polynomiale en le logarithme de l'entier p à tester.

Réduction polynomiale

La stabilité des polynômes par composition permet de transférer des propriétés de complexité d'un problème à un autre s'il existe une réduction calculatoire de complexité polynomiale. On parle alors de **réduction polynomiale**.

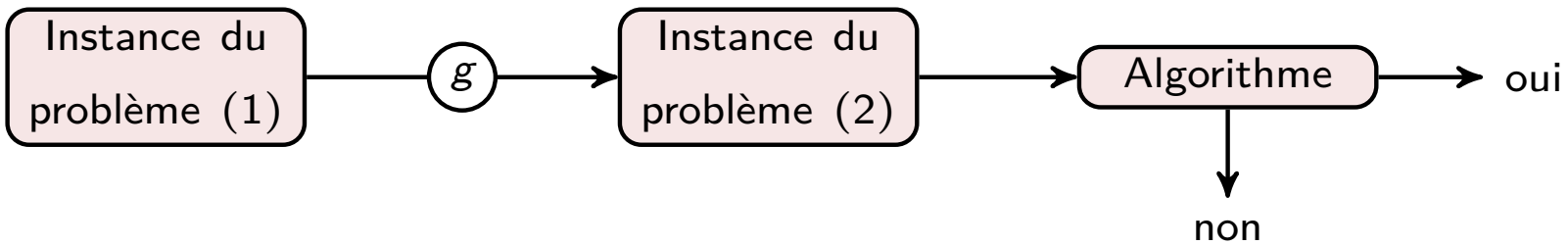
Définition 7

Soient deux problèmes de décision définis par des fonctions :

$$f_1 : E_1 \to \mathbb{B} \qquad f_2 : E_2 \to \mathbb{B}$$

On dit que f_1 se **réduit polynomialement** à f_2, et on note $f_1 \leqslant_p f_2$, s'il existe une fonction $g : E_1 \to E_2$ de complexité temporelle polynomiale en la taille de son entrée telle que pour tout $e \in E_1$ on a $f_1(e) = f_2(g(e))$.

On peut notamment établir l'appartenance d'un problème (1) à la classe P, en le réduisant polynomialement à un autre problème (2) qui serait déjà connu comme appartenant à P. On dit alors que le problème (1) est plus facile que le problème (2), ce que la notation $f_1 \leqslant_p f_2$ illustre assez bien.



Petite subtilité dans ce raisonnement : on se repose sur le fait que la taille de l'instance construite par la fonction de réduction est nécessairement elle-même polynomiale en la taille de l'entrée d'origine.

Appartenance à P par réduction polynomiale

Théorème 8

Soient deux problèmes de décision définis par des fonctions :

$$f_1 : E_1 \to \mathbb{B} \qquad f_2 : E_2 \to \mathbb{B}$$

Si f_2 appartient à la classe P et si $f_1 \leqslant_p f_2$, alors f_1 appartient à la classe P.

Démonstration

Notons `g` l'algorithme de réduction de f_1 vers f_2. Soit `f2` un algorithme solution de f_2 dont la complexité est majorée par un polynôme P_2 en la taille de l'entrée. On construit un algorithme `f1` pour f_1, en traduisant une entrée $e_1 \in E_1$ de f_1 en une entrée $e_2 \in E_2$ pour f_2 à l'aide de `g`, puis en renvoyant le résultat de l'application de `f2` à e_2. Ce nouvel algorithme n'est rien d'autre que la composition des deux algorithmes polynomiaux `f2` et `g`.

```
let f1 e1 = f2 (g e1)
```

Trois éléments permettent alors d'assurer que la complexité temporelle de `f1` est bien polynomiale en la taille de e_1.

- Par définition, la complexité temporelle de la traduction `g` est majorée par un polynôme P_g en la taille $|e_1|$ de e_1.
- Par conséquent, la taille $|e_2|$ de l'instance e_2 obtenue après traduction est également nécessairement polynomiale, majorée par un polynôme P_T en $|e_1|$.
- La complexité temporelle de cette exécution de `f2` est ainsi majorée [a] par $P_2(|e_2|)$, et donc par $P_2(P_T(|e_1|))$.

Finalement, le coût total est majoré par $P_g(|e_1|) + P_2(P_T(|e_1|))$, qui est bien un polynôme en la taille de e_1.

---

a. Sans perte de généralité, on suppose que P_2 est croissant.

Problème 2SAT

Le **problème 2SAT** admet une solution algorithmique polynomiale par réduction polynomiale à un problème de détermination des composantes fortement connexes d'un graphe.

Noter que, dans le cas d'une réduction polynomiale, la réduction elle-même n'appartient pas à la classe P, puisqu'il ne s'agit pas d'un problème de décision.

Classe NP

Classe NP

Les questions algorithmiques pour lesquelles on ne connaît pas de solution polynomiale sont nombreuses, et se rencontrent fréquemment. Il en va par exemple de la résolution de casse-têtes comme l'âne rouge ou le Sudoku, de jeux comme Othello, les échecs ou le go, de la résolution de formules SAT, du coloriage de graphes.

Il existe au-delà de P toute une hiérarchie de classes de complexité de plus en plus vastes, contenant des problèmes réputés plus difficiles, et en particulier les exemples précédents.

Classe NP

La classe NP contient certains de ces problèmes et est liée à des problèmes de recherche que l'on ne sait pas nécessairement résoutre facilement, mais dont le problème de vérification associé est simple.

Ainsi, si quelqu'un se présente à vous avec une prétendue solution s pour une instance donnée e, alors vous pouvez facilement déterminer si s est effectivement une solution valide pour l'instance e. Dans cette description informelle, il faut comprendre « facilement » comme « en temps polynomial ». Par exemple : résoutre une grille de Sudoku peut être difficile, mais vérifier la validité d'une solution est immédiat.

Classe NP

Autrement dit, nous parlons ici de **problèmes de recherche dont le problème de vérification associé appartient à P**. Cependant, la définition rigoureuse de la classe NP précise deux éléments par rapport à la description précédente.

- La classe NP, comme la classe P, est une classe de problèmes de décision.
- La définition de NP ne se restreint pas aux problèmes de décision issus de problèmes de recherche, mais couvre plus généralement tout problème de décision ayant les deux propriétés suivantes :
  - toute réponse positive admet une justification de taille polynomiale,
  - étant données une instance et une prétendue justification de sa positivité, on peut vérifier que la justification est valide, en temps polynomial.

Classe NP

Définition 9

Un problème de décision défini par une fonction $f : E \to \mathbb{B}$ est dans la **classe NP** si :

- il existe un ensemble C de **certificats** et une fonction $g : E \times C \to \mathbb{B}$ telle que $f(e) = \mathrm{V}$ si et seulement s'il existe un $c \in C$ de taille polynomiale en la taille de e tel que $g(e, c) = \mathrm{V}$;
- le problème de décision défini par g, appelé problème de **vérification** d'un certificat, est dans la classe P.

Problème SAT

Le problème SAT appartient à la classe NP.

- On prend comme ensemble de certificats l'ensemble des valuations, et on définit la fonction g telle que $g(e, c) = \mathrm{V}$ si et seulement si la valuation c rend vraie la formule e.
- Si une formule e est effectivement satisfiable, alors il existe des valuations c telles que $g(e, c) = \mathrm{V}$. En particulier il en existe au moins une qui ne mentionne que les variables effectivement dans e, dont la taille est donc majorée par celle de e.
- Étant données une formule et une valuation, on peut déterminer si la formule est vraie en un temps linéaire.

Problème SAT

En OCaml, on pourrait définir les certificats par des tableaux de type `bool array`. L'algorithme de vérification serait donné par la fonction `eval` ci-dessous.

```
let rec eval v f = match f with
  | True  -> true
  | False -> false
  | Var i -> assert (1 <= i && i < Array.length v);
             v.(i)
  | Not f -> not (eval v f)
  | Bin (And, f1, f2) -> eval v f1 && eval v f2
  | Bin (Or,  f1, f2) -> eval v f1 || eval v f2
  | Bin (Imp, f1, f2) -> not (eval v f1) || eval v f2
```

Sudoku généralisé

Le jeu de Sudoku généralisé d'ordre n comporte une grille de côté n^2 subdivisée en n^2 carrés de taille $n \times n$, à remplir avec n^2 symboles, chacun ne devant apparaître qu'une fois dans chaque ligne, chaque colonne, et chaque carré. Le Sudoku usuel est celui d'ordre 3.

Ce jeu est un problème de recherche. On veut trouver, s'il en existe, une manière valide de compléter une grille partiellement remplie fournie en entrée. Le problème associé d'existence d'une solution est dans la classe NP. Une solution peut être donnée sous la forme d'une grille complétée, dont la taille est comparable à celle de l'entrée. Étant donnée une solution potentielle c à une grille e, on peut effectivement vérifier en temps qu'une polynomial d'une part que c est valide, et d'autre part qu'elle complète bien e.

Sudoku généralisé

Pour la vérification de la validité de c, un programme comme le suivant convient (avec des notations évidentes).

```
bool check(int grid[81], int p) {
  for (int c = 0; c < 81; c++)
    if (c != p && same_zone(p,c)
               && grid[p] == grid[c])
      return false;
  return true;
}
```

Pour la vérification que c complète bien e, il suffit de tester point à point les cases non vides de e.

Contre-exemple probable

On pourrait tenter de justifier que le problème de l'existence d'une solution à partir d'une configuration de l'**âne rouge** généralisé est également dans la classe NP, en prenant comme certificats les séquences de coups permettant de résoutre ce jeu.

Étant donnée une liste de coups, on peut facilement les reproduire et vérifier qu'ils mènent à une position gagnante. Cependant, on ne sait pas établir que de tels certificats respectent la contrainte de taille polynomiale !

En réalité, le problème de l'âne rouge généralisé est fortement soupçonné de ne pas appartenir à la classe NP, de la même manière que l'on soupçonne que certains problèmes de la classe NP n'appartiennent pas à la classe P.

Inclusion de P dans NP

Les classes P et NP sont deux classes de problèmes de décisions apparentées mais leur relation exacte n'a pas encore pu démontrée. On établit que la classe NP contient intégralement la classe P.

Théorème 10

Les classes P et NP satisfont l'inclusion $P \subseteq NP$.

Démonstration

Soit un problème de décision appartenant à la classe P, défini par une fonction $f : E \to \mathbb{B}$. Prenons comme ensemble de certificats le singleton $C = \{0\}$ et définissons $g : E \times C \to \mathbb{B}$ par $g(e, 0) = f(e)$. La taille du certificat unique 0 est bien bornée par un polynôme, et le problème de vérification défini par g est équivalent au problème de décision f, qui appartient justement à la classe P.

Inclusion de P dans NP

On soupçonne que l'inclusion de P dans NP est stricte, c'est-à-dire qu'il existe dans la classe NP des problèmes qui ne peuvent pas être résolus par des algorithmes polynomiaux. Nous verrons même dans la section suivante que l'on connaît un grand nombre de problèmes NP supposés non polynomiaux.

Cependant, personne n'a encore pu démontrer ceci rigoureusement. Jusqu'à preuve du contraire, il reste donc également possible que ces deux classes de complexité soient égales.

Ce problème de l'égalité ou de la non-égalité des classes P et NP, encore ouvert cinquante ans après avoir été énoncé, est considéré comme l'un des problèmes ouverts majeurs de l'informatique théorique. Il a des ramifications dans de nombreux domaines scientifiques et en particulier en cryptographie.

NP-complétude

Problème NP-complet

Au sein de la classe NP, on peut caractériser les problèmes dont la difficulté est maximale : un problème est de difficulté maximale si une solution de ce problème permet de construire « facilement » une solution de n'importe quelle autre problème de la classe NP. Comme précédemment, cette notion de facilité couvre une notion de complexité polynomiale.

Définition 11

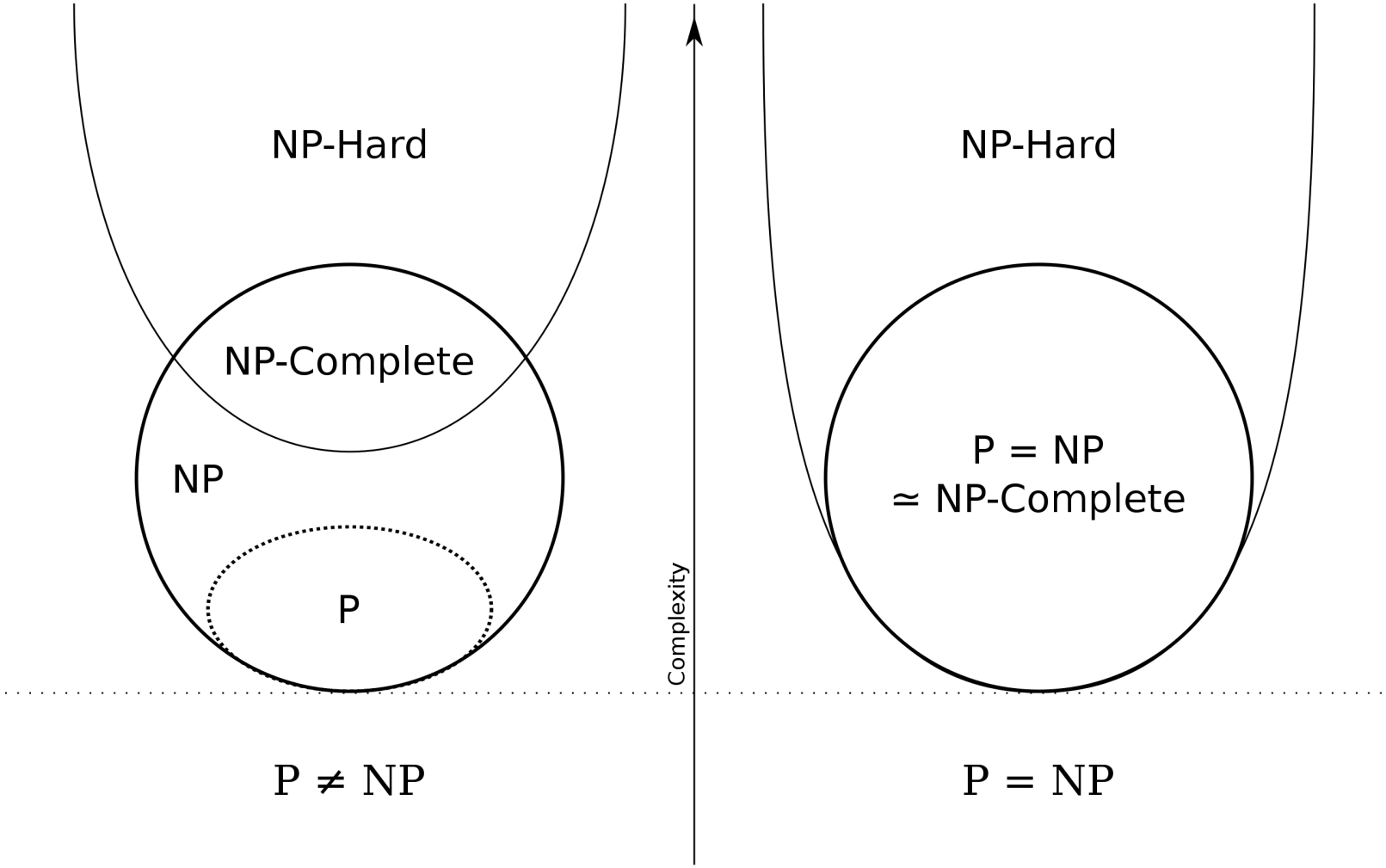
Un **problème NP-difficile** est un problème algorithmique auquel peut être réduit polynomialement tout problème NP.
Un **problème NP-complet** est un problème de décision appartenant à la classe NP, qui est également NP-difficile.

Problème NP-complet

Un problème NP-difficile est un problème f qui est donc au moins aussi dur que tout problème NP, sans préjuger du fait que f appartienne à NP ni même qu'il s'agisse d'un problème de décision.

Les problèmes NP-complets sont à l'intersection de NP et des problèmes NP-difficiles.

Problème NP-complet



Behnam Esfahbod, CC BY-SA 3.0
https ://creativecommons.org/licenses/by-sa/3.0, via Wikimedia Commons

Solution polynomiale

Cette classe particulière de problèmes présente un intérêt fondamental : si un tel problème de difficulté maximale au sein de NP appartenait à la classe P, alors on pourrait en déduire que l'intégralité de la classe NP est incluse dans P, et la question « P = NP ? » serait immédiatement résolue positivement.

Théorème 12

S'il existe un algorithme A de complexité temporelle polynomiale en la taille de son entrée qui résoud un problème de décision f NP-complet, alors tout problème de la classe NP admet une solution polynomiale.

Démonstration

Supposons qu'existent un tel algorithme A pour un tel problème f. Soit f' un problème de décision appartenant à la classe NP. Alors par définition de la NP-complétude, f' se réduit polynomialement à f par une certaine fonction g. On résoud ainsi f' en temps polynomial en composant les algorithmes polynomiaux g et A.

Réduction polynomiale

Inversement, on peut utiliser des réductions polynomiales pour transférer des résultats de NP-complétude d'un problème à un autre.

Théorème 13

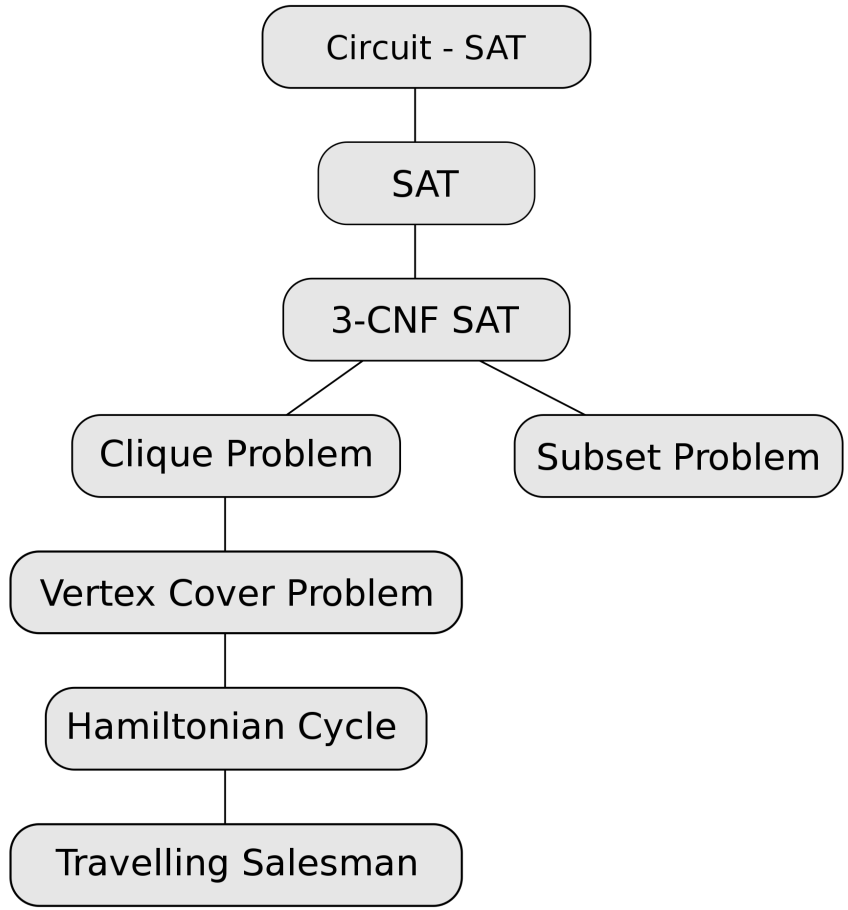
Soient deux problèmes de décision définis par des fonctions $f_1 : E_1 \to \mathbb{B}$ et $f_2 : E_2 \to \mathbb{B}$. Si f_1 est NP-complet et se réduit polynomialement à f_2, alors f_2 est également NP-complet.

Démonstration

Par définition de la NP-complétude, tous les problèmes NP se réduisent polynomialement à f_1. Or, la composition de deux réductions polynomiales est encore une réduction polynomiale. Ainsi, tous les problèmes NP se réduisent polynomialement à f_2, qui est bien NP-complet.

Problème NP-complet

Le schéma ci-contre indique, pour une partie de ces problèmes, dans quel ordre on prouve en général la NP-complétude.
Cet ordre de réduction n'est pas le seul possible, puisque par définition, il existe toujours une réduction entre deux problèmes NP-complets quelconques.
Son intérêt est de simplifier les démonstrations.



Gian Luca RuggeroActam
Public domain, via Wikimedia Commons

Problème SAT

Si l'on connaît déjà un problème NP-complet, on peut utiliser des réductions polynomiales pour démontrer que de nombreux autres problèmes sont également NP-complets.

En revanche, démontrer la NP-complétude d'un premier problème f, à partir de la seule définition, est difficile : il faut réussir à réduire n'importe quel autre problème f' de la classe NP à f, c'est-à-dire réduire un problème de décision f' à f sans autre connaissance sur f' que le fait qu'il admet des certificats vérifiables en temps polynomial. Une telle preuve a été faite en 1971 pour le problème SAT.

Théorème 14 (Cook Levin)

théorème de Cook-Levin Le problème SAT est NP-complet.

Problème SAT

On a établi l'appartenance de SAT à la classe NP. Le reste de la preuve du théorème, à savoir démontrer que SAT est NP-difficile, est hors-programme. Et pour cause, notre définition d'un algorithme comme un programme C ou OCaml exécuté sur une machine à mémoire illimitée est trop imprécise pour permettre une preuve rigoureuse.

Il est possible d'esquisser une preuve dont le principe reste accessible mais sa lecture peut être omise.

Problème SAT

On considère un problème NP arbitraire et l'algorithme polynomial de vérification associé. Partant d'une entrée e pour ce problème, on construit une formule SAT de taille polynomiale en la taille de e, satisfiable si et seulement s'il existe un certificat c tel que l'algorithme de vérification appliqué à (e, c) répond V. L'astuce consiste à produire une formule qui décrit exhaustivement le calcul de l'algorithme de vérification. Les idées sont les suivantes.

Problème SAT

Tout d'abord, on remarque que toute exécution de l'algorithme de vérification sur une entrée e de taille n et un certificat c légitime pour e utilise une quantité de mémoire polynomiale en n, pendant un nombre d'étapes de calcul également polynomial en n. En effet, la taille des certificats est bornée par un polynôme en n, et la complexité temporelle de la vérification est elle-même polynomiale en n et la taille du certificat, c'est-à-dire polynomiale en n.

De ce dernier point on déduit également que la vérification ne peut utiliser effectivement qu'une quantité polynomiale de mémoire.

Problème SAT

Ainsi, on peut décrire exhaustivement l'exécution d'une vérification pour une entrée $e \in E$ en considérant un nombre polynomial de bits de mémoire pendant un nombre polynomial d'étapes de calcul, c'est-à-dire à une surface d'espace-temps polynomiale.

On fixe pour cela des variables propositionnelles $X_{i,k}$, où i désigne l'un des bits de la zone de mémoire utilisée et k l'une des étapes de l'exécution. Interprétation : la variable $X_{i,k}$ vaut Vsi le bit numéro i vaut 1 à l'étape k du calcul. Ces variables sont en nombre polynomial en n.

Problème SAT

Il ne reste qu'à écrire les formules SAT décrivant deux choses :

- l'état initial de la mémoire (variables $X_{i,0}$), contenant le programme exécuté, la représentation de l'entrée e et la représentation d'un certificat indéterminé ;
- l'évolution de chaque bit entre une étape et la suivante, c'est-à-dire de $X_{i,k+1}$ en fonction des $X_{j,k}$ décrivant l'instruction exécutée et les autres données participant à l'opération.

Les formules SAT précises reflètent avec un certain détail l'architecture de la machine sur laquelle s'exécute l'algorithme. Dans notre cas, où cette architecture n'a pas été détaillée, nous ne pouvons guère aller plus loin.