Cours Complexité/Calculabilité

Licence L3 Année 2021-2022

Version 1.0

Université de Montpellier
Place Eugène Bataillon
34095 Montpellier Cedex 5

RODOLPHE GIROUDEAU ET JEAN-CLAUDE KÖNIG
161, RUE ADA
34392 MONTPELLIER CEDEX 5
MAIL : $\{rgirou, konig\}$@LIRMM.FR

Avertissement & planning

Ce manuscrit est le cours de complexité de la licence informatique L3 de l'université de Montpellier. Les notions abordées dans ce cours portent sur les classes de complexité ($\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complet, ...), le principe de la réduction polynomiale ...
Les livres sur lesquels le cours est basé sont les suivants :

- Introduction to algorithms [2]. Ce livre sera utile tout le long de votre formation.
- Complexité et approximation polynomiale [5]. Ce livre est à la base de cours.
- Complexity and approximation : combinatorial optimization problems and their approximability properties [1].

Tout le long de votre formation, vous serez amenés à rédiger des rapports scientifiques comme ce manuscrit. Il est écrit en LaTeX et les figures sont en TikZ.
Ce manuscrit n'a aucune vocation à être distribué ou déposé sur un site autre que le site Moodle de l'université de Montpellier. Il est truffé de coquilles, d'erreurs, de fautes d'orthographes, je vous remercie de me les signaler.

Planning : voir l'application PROSE sur le site ENT.

Table des matières

Table des figures

Liste des Algorithmes

Chapitre

1 Langages, Instances, Problèmes et algorithmes

Sommaire

Résumé

Dans ce chapitre nous présentons la notion de langages, d'instances d'un problème combinatoire. Nous caractérisons également la notion de problèmes de décision, d'optimisation, . . .

1.1 Introduction

Dans la partie précédente, nous avons mis en évidence des problèmes sans solution algorithmique générale (problème de l'arrêt notamment). Pour d'autres problèmes des solutions algorithmiques existent.
Prenons par exemple le problème de l'existence d'un couplage de taille k.

COUPLAGE DANS UN GRAPHE

ENTRÉE : Soit $G = (V, E)$ non orienté, $k \in \mathbb{N}$.
QUESTION :Existe-t'il un ensemble d'arêtes de taille k deux à deux disjointes ?

Un algorithme simple consiste à donner tous les sous-ensembles de k arêtes et pour chacun d'eux vérifier si le sous-ensemble d'arêtes est bien un couplage. Il est facile de constater que cet algorithme va poser un problème malgré le fait que vérifier si un sous-ensemble d'arêtes est un couplage est un problème rapide à résoudre (il suffit de vérifier que l'ensemble des extrémités des arêtes n'a pas de doublons et donc un algorithme en $k * \log(k)$ peut faire l'affaire). En effet le nombre de sous-ensembles de taille k peut-être colossale (petit exercice estimer le temps d'exécution au pire de l'algorithme si $m = 1000$, $k = 40$, le problème précédent se résout en 1 nano-seconde).
Heureusement il existe un algorithme rapide : on calcule le couplage maximum en $O(nm)$, si la cardinalité est supérieure à k la réponse en oui sinon elle est non. En clair, il existe une réponse à la question de l'existence avec un algorithme de complexité en temps polynomiale en fonction de la taille des données (le stockage du graphe nécessite au moins $m + n$ bits et l'algorithme s'exécute en un temps au pire proportionnel à nm). Ce problème est dit de la classe $\mathcal{P}$.
Même réflexion avec le stable de taille k.

STABLE

ENTRÉE : $G = (V, E)$, $k \in \mathbb{N}$.
TÂCHE : Trouver un ensemble maximum de sommets indépendants.

Remarque savoir si un sous-ensemble de taille k est un stable est aussi un problème rapide. Le problème est donc dit $\mathcal{NP}$ (comme le problème différent). Plus précisément si la réponse est OUI il existe un stable de taille k, et il est facile de convaincre rapidement quelqu'un (ou prouver que la réponse est OUI rapidement). Remarquons que prouver peut être long (pour convaincre que la réponse est OUI il suffit par exemple de chercher un stable de taille k).
Remarquons aussi que la stratégie est de prouver que la réponse est NON n'est

pas évidente (comment faire autrement que montrer que tous les sous-ensembles de taille k ne sont pas des stables).
La classe $\mathcal{NP}$ est plus grande que la classe $\mathcal{P}$. Strictement ? C'est la grande question. La plupart des chercheurs du domaine pense que la réponse est OUI : $\mathcal{NP}$ est strictement plus grande que $\mathcal{P}$.
Pourquoi la notion de polynomialité est-elle importante ? Cette frontière entre ce qui est réalisable de façon opérationnelle ou pas est-elle purement théorique ? Quel est le sens pratique ?

1.2 Problème de décision versus problème d'optimisation

Les problèmes qui nous intéressent peuvent être classés en trois catégories :

- les problèmes de décision (voir la section 1.2.1),
- les problèmes du calcul de la valeur optimale (voir la section 1.2.3),
- les problèmes d'optimisation (voir la section 1.2.4).

1.2.1 Les problèmes de décision

Pour les problèmes de décision, on cherche à décider de l'existence d'une solution vérifiant certaines propriétés. La solution d'un problème de décision est :

- soit « **oui** une telle solution existe » ;
- soit « **non** une telle solution n'existe pas ».

EXEMPLE
Le problème de décision le plus connu est le problème de satisfaisabilité (noté par SATISFAISABILITÉ) d'une forme conjonctive normale : A une variable binaire x, nous associons deux littéraux x et $\bar{x}$, c'est à dire, la variable elle-même et sa négation ; une clause est une disjonction de littéraux.

SATISFAISABILITÉ

ENTRÉE : Etant donné un ensemble U de n variables binaires $x_1, \ldots, x_n$ et un ensemble $C = \{C_1, \ldots, C_m\}$ de m clauses.
QUESTION :Existe-t'il une affectation des variables de vérité telle que C soit satisfait ?

Etant donné une instance ϕ de SATISFAISABILITÉ, si ϕ admet un modèle, alors la solution (la bonne réponse) est **oui** ; sinon, la solution est **non**. Ici toutes les solutions ont la même valeur.

Un autre type de problème de décision est celui où nous cherchons plus particulièrement à affirmer ou à infirmer l'existence d'une solution dont la valeur est supérieure ou inférieure à certain seuil. La solution de tels problèmes est, encore une fois :

— soit « **oui** une telle solution existe » ;

— soit « **non** une telle solution n'existe pas ».

EXEMPLE
Considérons le problème classique du voyageur de commerce :

VOYAGEUR DE COMMERCE

ENTRÉE : Un ensemble de m villes X, un ensemble de routes entre les villes E, une fonction de coût $v : E \to \mathbb{R}$ où $v(x, y)$ est le coût de déplacement de x à y.

TÂCHE : Trouver une tournée de coût (chemin Hamiltonien de coût minimum) ?

Nous intéressons non plus à la recherche d'une solution optimale mais de répondre « existe-il un cycle Hamiltonien de distance inférieure ou égale à K ? » Ici, encore une fois, la solution est **oui** si un tel cycle existe, **non** s'il n'existe pas.

DÉFINITION 1.2.1 : Un problème de décision est un problème pour lequel on cherche à décider de d l'existence d'une solution vérifiant certaines propriétés (portant, soit sur sa structure, soit sur sa valeur, soit ...).

1.2.2 Un cadre pour les langages formels

On utilise indifféremment la terminologie problème ou langage dans tout ce qui suit.
L'un des intérêts pratiques de s'en tenir aux problèmes de décision est qu'ils facilitent l'emploi des mécanismes de la théorie des langages formels. Il n'est pas inutile ici de revoir quelques définitions de cette théorie. Un alphabet Σ est un

ensemble fini de symboles. Un langage L sur S est un ensemble quelconque de chaînes construites à partir de symboles de S.

Problèmes vs langages Cela découle des considérations suivantes : à un problème (de décision) est associé un langage et réciproquement. En effet, à un problème est associé généralement implicitement une fonction de codage (par exemple pour les graphes, une façon de coder les graphes) qui permet de coder les instances, c'est-à-dire les éléments de E, par un mot sur un certain alphabet Σ. On peut donc voir E comme un sous-ensemble de Σ^*, où Σ est un certain alphabet : au problème de décision P, on associe le langage $L(P)$ correspondant à l'ensemble des mots codant une instance de E, qui appartient à E^+ avec E^+ l'ensemble des instances positives :

$$L(P) = \{w | w \in E^+\}$$

Réciproquement, on peut voir tout langage L comme un problème de décision, en le formulant de cette façon :

PROBLÈME ASSOCIÉ AU LANGAGE L

ENTRÉE : Un mot w.
QUESTION : Décider si $w \in L$

EXEMPLE
Si $\Sigma = \{0, 1\}$, l'ensemble $L = \{10, 11, 101, 111, 1011, 1101, 10001, \ldots\}$ est le langage des représentations binaires des nombres premiers. La chaîne vide est notée ϵ, et le langage vide est noté $\varnothing$. Le langage de toutes les chaînes sur Σ est noté Σ^*. Par exemple, si $\Sigma = \{0, 1\}$, alors $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$ est l'ensemble de toutes les chaînes binaires. Tout langage L sur Σ est un sous-ensemble de Σ^*.

Il existe de nombreuses opérations sur les langages. Les opérations ensemblistes, comme l'union et l'intersection, se déduisent directement des définitions de la théorie des ensembles. On définit le complément de L par $\bar{L} = \Sigma^* - L$. La concaténation de deux langages L_1 et L_2 est le langage

$$L = \{x_1 x_2 : x_1 \in L_1 \text{ et } x_2 \in L_2\}.$$

La fermeture ou Kleene étoile d'un langage L est le langage

$$L^* = \{\epsilon\} \cup L \cup L^2 \cup L^3 \cup \ldots,$$

Ou L^k est le langage obtenu par k concaténations successives de L avec lui-même. Du point de vue de la théorie des langages, l'ensemble des instances d'un problème de décision Q est tout simplement l'ensemble Σ^*, où $\Sigma = \{0, 1\}$. Puisque Q est entièrement caractérisé par les instances du problème qui produisent une réponse 1 (oui), on peut voir Q comme un langage L sur $\Sigma = \{0, 1\}$, où

$$L = \{x \in \Sigma^* : Q(x) = 1\}$$

Par exemple, le problème de décision CHEMIN a pour langage correspondant

$$\begin{aligned} \text{CHEMIN} \quad = \quad & \{(G, u, v, k) : G = (S, A) \text{ est un graphe non orienté} \\ & u, v \in S, k \geq 0 \text{ est un entier, et il existe une chaîne de} \\ & u \text{ vers } v \text{ dans } G \text{ composé d'au plus } k \text{ arêtes}\} \end{aligned}$$

(Par commodité, on utilise parfois le même nom, ici CHEMIN, pour indiquer à la fois un problème de décision et le langage correspondant).
La théorie des langages formels permet d'exprimer de façon concise la relation entre les problèmes de décision et les algorithmes qui les résolvent.
On dit qu'un algorithme A accepte une chaîne $x \in \{0, 1\}^*$ si, étant donné une entrée x, l'algorithme produit $A(x) = 1$. Le langage accepté par un algorithme A est l'ensemble des chaînes acceptées par l'algorithme, soit $L = \{x \in \{0, 1\}^* : A(x) = 1\}$. Un algorithme A rejette une chaîne x si $A(x) = 0$.
Même si le langage L est accepté par un algorithme A, l'algorithme ne rejettera pas forcément une chaîne $x \notin L$.

EXEMPLE

Notons qu'un algorithme pourra boucler indéfiniment. Un langage L est décidé par un algorithme A si toute chaîne binaire de L est acceptée par A et si toute chaîne binaire n'appartenant pas à L est rejetée par A.

— Un langage L est accepté en temps polynomial par un algorithme A s'il est accepté par A et si, en outre, il existe une constante k telle que, pour toute chaîne x de longueur n appartenant à L, l'algorithme A accepte x en temps $O(n^k)$.

— Un langage L est décidé en temps polynomial par un algorithme A s'il existe une constante k telle que, pour toute chaîne $x \in \{0, 1\}^*$ de longueur n, l'algorithme décide correctement si $x \in L$ en temps $O(n^k)$. Donc, pour accepter un langage, un algorithme n'a besoin de s'intéresser qu'aux chaînes de L, alors que pour décider un langage, il doit accepter ou rejeter correctement toutes les chaînes de $\{0, 1\}^*$.

1.2.3 Les problèmes de calcul de la valeur optimale

Dans ce type de problèmes nous cherchons à calculer la valeur d'une solution optimale (et pas la solution elle-même).
Pour le problème VOYAGEUR DE COMMERCE, par exemple, le problème du calcul de la valeur optimale associé revient à calculer la longueur du plus petit cycle Hamiltonien (et pas la détermination du cycle lui-même). Trouver la valeur d'une solution optimale peut être, selon les problèmes, plus simple ou équivalent que de calculer la solution elle-même.

1.2.4 Les problèmes d'optimisation

Nous considérons la catégorie de problèmes où l'on cherche à déterminer la meilleure solution parmi celles vérifiants certaines propriétés imposées par la définition même du problème. De façon générale, une instance I d'un problème d'optimisation peut être vue comme un programme mathématique de la forme :

$$\begin{cases} opt\ v(x) \\ x \in C_I \end{cases}$$

où x est un vecteur décrivant la solution [1], $v(x)$ est la fonction objectif, C_I est l'ensemble des contraintes du problème, spécifiée pour l'instance I, et $opt \in \{\min, \max\}$. Une solution optimale de I est un vecteur $x^* \in argopt\{v(x), x \in C_I\}$. La quantité $v(x^*)$ est appelée valeur objectif ou valeur du problème. Une solution $x \in C_I$ est appelée solution réalisable.

EXEMPLE

STABLE

ENTRÉE : $G = (V, E)$ d'ordre n avec m arêtes.
TÂCHE : Trouver un ensemble maximum de sommets indépendants.

Une instance générique de ce problème peut être exprimée de la façon suivante :

1. Pour les problèmes de l'optimisation combinatoire qui nous intéressent, on peut supposer que les composantes de ce vecteur sont 0 ou 1 ou, à la rigueur des entiers.

$$\begin{cases} \max x \\ Ax \leq 1 \end{cases}$$

où A est la matrice d'incidence de G et $x \in \{0,1\}$. : $x_i = 1$ si le sommet $v_i \in V$ est pris dans la solution ; sinon $x_i = 0$. Les m contraintes $Ax \leq 1$ expriment le fait que, pour chaque arête, une et une seule parmi ses extrémités peut être incluse dans le stable. Les solutions réalisables sont toutes les stables de G et la solution optimale un stable maximum de G correspondant à un vecteur réalisable comportant un nombre maximum de 1.

Il est facile de voir que la solution d'un problème d'optimisation englobe celle d'un problème de calcul de la valeur optimale (sous réserve que l'on sache calculer cette valeur). De plus, un problème de calcul de la valeur optimale peut être associé à un problème d'optimisation. Notons enfin que les problèmes d'optimisation ont toujours une variante décisionnelle.

Chapitre

2 Classes de complexité : $\mathcal{P}$, $\mathcal{NP}$ et $\mathcal{NP}$-complet

Sommaire

Résumé

Dans ce chapitre nous étudierons les classes de complexité de base dans le cadre de la théorie de la complexité. Ces classes sont $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complet, et leurs classes complémentaires.

2.1 Introduction

Nous présentons les éléments essentiels pour la classification des problèmes au sens de la complexité. L'outil nécessaire et fondamentale est la notion de réduction (polynomiale, logarithmique), qui permet de relié deux problèmes à priori totalement différents.

2.2 Réduction polynomiale

Nous nous intéressons dans cette section aux problèmes dans l'espace $\mathcal{NP}\backslash\mathcal{P}$. Ces problèmes sont évidemment plus difficiles algorithmiquement que les problèmes dans $\mathcal{P}$ puisque comme nous l'avons mentionné précédemment nous ne connaissons pas d'algorithmes polynomiaux les résolvant. Nous allons voir que beaucoup de d'entre-eux sont très fortement liés par le biais du concept de réduction.
Le principe de réduction d'un problème π à un problème π' consiste à regarder modulo une petite transformation, le problème π comme étant un cas particulier de π'.
Si la transformation est polynomiale et que l'on sait résoudre π' en temps polynomial, nous saurons alors résoudre π à son tour, toujours en temps polynomial. La réduction est ainsi un moyen de transporter un résultat de résolution d'un problème à un autre ; elle est de la même façon un outil de classement des problèmes selon le niveau de difficulté de leur résolution.
Il y a eu, au cours du développement de la théorie de la complexité, grand nombre de définitions de réductions selon ce que l'on voulait préserver, transférer d'un problème à l'autre (décision ou optimisation, construction ou évaluation, structure du problème, ...) Mais on retrouve, à la base de toute réduction, les mêmes principes fondamentaux.
La figure 2.1 illustre le principe d'une réduction en temps polynomial d'un langage L_1 à un autre langage L_2. Chaque langage est un sous-ensemble de $\{0,1\}^*$. La fonction de réduction f fournit une application en temps polynomial telle que si $x \in L_1$, alors $f(x) \in L_2$. De plus, si $x \notin L_1$, alors $f(x) \notin L_2$. Donc la fonction de réduction fait correspondre une instance x quelconque du problème représenté par L_2. L'obtention d'une réponse à la question $f(x) \in L_2$ fournit directement une réponse à la question $x \in L_1$.

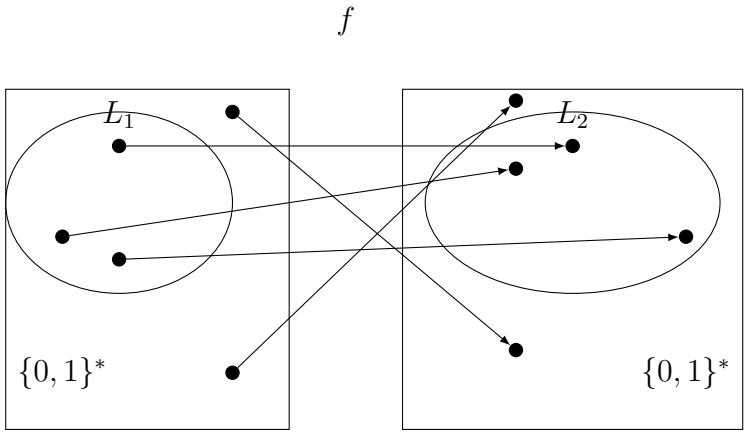


FIGURE 2.1 – Une illustration d'une réduction en temps polynomial d'un langage L_1 vers un langage L_2 via une fonction de réduction f. Pour une entrée $x \in \{0,1\}^*$ quelconque, la question de savoir si $x \in L_1$ a la même réponse que la question de savoir si $f(x) \in L_2$.

DÉFINITION 2.2.1 : Une réduction many-one en temps polynomial d'un problème B (sur l'alphabet Σ_B) à un problème A (sur l'alphabet Σ_A) est une fonction $f : \Sigma_B^* \to \Sigma_A^*$ calculable en temps polynomial telle que :

$$\forall x \in \Sigma_B^*, x \in B \Leftrightarrow f(x) \in A$$

Si une telle fonction f existe, on dira que B se réduit à A (via f) et on notera $B \leq A$.

Une illustration est donnée par les figures 2.1 pour un point de vue des langages, 2.2 pour un point de vue problèmes, et par la figure 2.3 du point de vue algorithmique.

Du point de vue des problèmes, f est une réduction polynomiale du problème π au problème π' si elle est une réduction polynomiale entre les langages correspondants. La réduction f transforme toutes les instances positives π en instances positives de π', et toutes instances négatives de π en instances négatives de π'.

L'existence d'une réduction polynomiale de π vers π' montre que π' est au moins aussi difficile que π. En effet si π peut être résolu en temps polynomial, alors π' peut l'être aussi ; si par contre π requiert un temps exponentiel, alors π' ne peut être résolu par un algorithme polynomial.

Notons bien que le sens premier de la réduction de π vers π' est encore plus fort : une réduction prouve qu'à une transformation polynomiale près des instances,

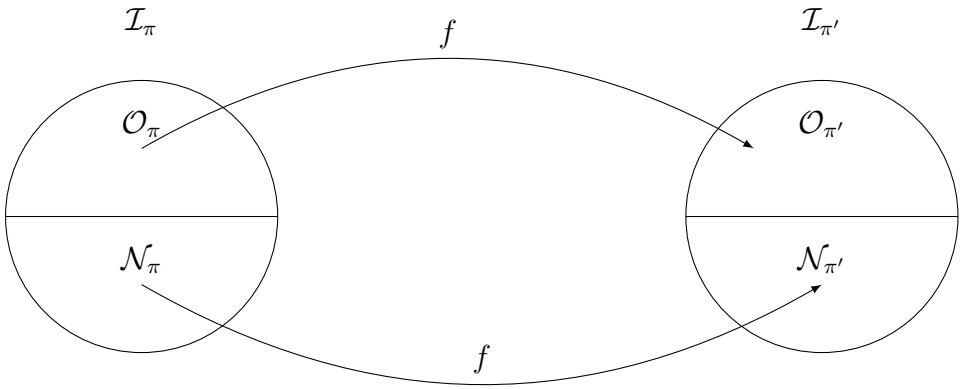
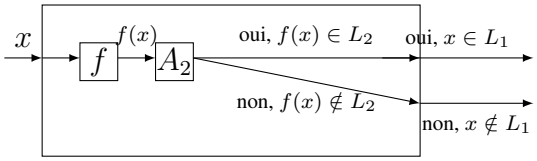


FIGURE 2.2 – Réduction du point de vue des problèmes.



Algorithme à temps polynomial décidant A_1

FIGURE 2.3 – Utilisation d'une réduction à temps polynomial pour résoudre un problème de décision Π en temps polynomial à partir d'un algorithme de décision à temps polynomial associé à un autre problème Π'. En temps polynomial, on transforme une instance x de Π en une instance x' de Π', on résout B en temps polynomial, puis on utilise la réponse pour x' comme réponse pour x.

c'est-à-dire à un codage naturel près de π, le problème π est simplement un sous-problème de π'.
D'un point de vue algorithmique, une réduction est un "pré-processing" des instances du problème π, qui permet d'utiliser tout algorithme polynomial de résolution pour π' pour résoudre π en temps polynomial (voir exercice de TD).

2.2.1 La réduction de Karp

Nous nous intéressons dans ce paragraphe à une réduction particulière, originellement présenté par Karp, ayant pour objectif de lier deux **problèmes de décision** par rapport à la possibilité de leur résolution optimale en temps polynomial.
Avant de formaliser le concept de la réduction de Karp, nous allons le présenter informellement par le biais d'un exemple. Plus précisément, nous allons montrer comment la résolution polynomiale du voyageur de commerce aurait entraîné la résolution polynomiale du problème du cycle Hamiltonien. Ce problème est définit de la manière suivante :

CIRCUIT HAMILTONIEN

ENTRÉE : Soit $G = (V, E)$ non orienté
QUESTION :Existe-t'il un chemin Hamiltonien ?

Remarquons que l'Hamiltonicité n'est pas une propriété triviale pour un graphe. Il existe un nombre infini de graphes Hamiltoniens (c'est à dire qu'il possèdent un cycle Hamiltonien) et un nombre infini de graphes qui ne possèdent pas de tel cycle. Considérons un graphe G' quelconque, fixons deux de ses sommets a et b et rajoutons le graphe G'' comme dans la figure 2.4
Appelons G le graphe résultant. Il est facile de voir que G n'est pas Hamiltonien, même si G' l'est.
Il est aisé de démontrer que le cycle Hamiltonien est dans $\mathcal{NP}$. En effet, si un ensemble C d'arêtes se présente comme solution candidate, il faut les compter et les teste si, en les mettant bout-à-bout, le résultat est un cycle Hamiltonien ; ces tests se font, comme l'on a vu en $O(n^2)$.
Dans ce qui suit, nous allons procéder de la façon suivante :

1. nous transformons en temps polynomial une instance $G = (V, E)$, $|V| = n$, de cycle Hamiltonien en une instance du voyageur de commerce.
2. on suppose que qu'il existe un algorithme polynomial qui résout correctement le problème du voyageur de commerce : on montre comment, en ex-

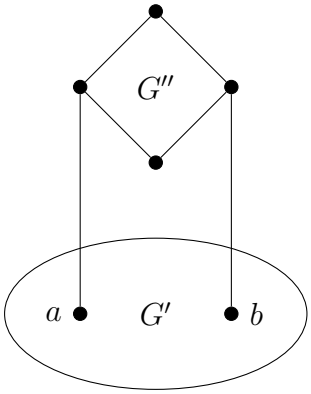


FIGURE 2.4 – Une classe de graphes non-Hamiltonien.

ploitant cette résolution, nous pouvons résoudre correctement le problème de cycle Hamiltonien en temps polynomial.

Pour le point 1, nous complétons le graphe G pour obtenir un graphe complet K_n, et on pose :

$$d(i,j) = \begin{cases} 1 \text{ si } (v_i, v_j) \in E \\ 2 \text{ si } (v_i, v_j) \notin E \end{cases}$$

Enfin, nous posons $K = n$. L'instance du voyageur de commerce ci-dessus est construite en $O(n^2)$.
Supposons maintenant pour le point 2 que notre algorithme polynomial hypothétique répond oui pour le voyageur de commerce, c'est à dire, il existe un tour de valeur n dans k_n et donc ne passant que par des arêtes de G. On en déduit alors $O(1)$ (en lisant la réponse) que la solution pour le cycle Hamiltonien est également oui ; Inversement, si l'algorithme répond non, ceci veut dire qu'il n'existe pas de cycle Hamiltonien ne passant que par des arêtes de G ; donc la réponse est également non pour cycle Hamiltonien, et celle-ci est toujours déduite en $O(1)$.
Puisque la transformation de G en K_n et la transformation d'une solution pour VOYAGEUR DE COMMERCE en une solution pour CIRCUIT HAMILTONIEN se font en temps polynomial, en supposant que l'algorithme qui fournit la solution pour VOYAGEUR DE COMMERCE est polynomial, on en déduit un algorithme qui résout de manière polynomiale le problème de CIRCUIT HAMILTONIENen faisant appel à un algorithme polynomial pour la version décisionnelle du VOYAGEUR DE COMMERCE .

La preuve ci-dessus est un exemple d'application de ce qui appelé, dans la littérature, une réduction de Karp (notée parfois K-réduction). Formellement, cette réduction est définie comme suit.

DÉFINITION 2.2.2 : Etant donné deux problèmes de décision π_1 et π_2, une réduction de Karp (ou transformation polynomiale) est une fonction $f : \mathcal{I}_{\pi_1} \rightarrow \mathcal{I}_{\pi_1}$, calculable en temps polynomial, telle que étant donné une solution pour $f(I)$, nous sommes capables de trouver une solution I en temps polynomial en $|I|$ (la taille de l'instance I).

Remarquons que la définition 2.2.2 est différente de la définition usuelle de la réduction de Karp : étant donné deux problèmes de décision π_1 et π_2 une réduction de Karp (ou transformation polynomiale) est une fonction $f : \mathcal{I}_{\pi_1} \rightarrow \mathcal{I}_{\pi_2}$, calculable en temps polynomial telle que $I \in \mathcal{O}_{\pi_1}$, si et seulement si $f(I) \in \mathcal{O}_{\pi_2}$. Cette différence est due au fait qu'ici nous voyons une solution pour un problème de décision non seulement comme une réponse oui ou non mais, pour le cas oui on l'associe à un certificat (voir paragraphe 2.3.1).

2.2.2 La réduction de Turing

En fait, dans la littérature, une forme plus large de la réduction de Karp est utilisée. Remarquons que, de façon générale, une réduction de Karp d'un problème de décision π_1 à un problème de décision π_2 met en évidence un algorithme pour π_1 qui fait appel à un algorithme pour π_2. Nous pouvons encore expliciter cette procédure comme suie : étant n'importe quelle instance $I_1 \in \mathcal{I}_{\pi_1}$, l'algorithme construit $I_2 \in \mathcal{I}_{\pi_2}$; il fait appel à l'algorithme (c'est l'algorithme hypothétique supposé auparavant) pour π_2 en lui demandant de calculer une solution sur I_2, puis il transforme cette solution en solution pour π_1 sur I_1. Si l'algorithme hypothétique pour π_2 est polynomial, l'algorithme pour π_1 est aussi polynomial.
A la suite de cette remarque, nous pouvons maintenant spécifier une explicitation de la réduction de Karp, appelée réduction de Turing dans la littérature et notée $\leq_T$ dans ce qui suite. En vue de la spécification de la réduction de Turing, complétons légèrement notre façon de voir le problème. D'ores et déjà, nous définissons un problème π comme un couple $(\mathcal{I}_\pi, Sol_\pi)$ tel que :

— $\mathcal{I}_\pi$ est l'ensemble des instances de π (on considère que chaque instance $I \in \mathcal{I}_\pi$ est reconnaissable en temps polynomial en $|I|$).

— Sol_π est l'ensemble de solutions pour π (nous notons par $Sol_\pi(I)$ l'ensemble de solution de l'instance $I \in Sol_\pi$).

DÉFINITION 2.2.3 : Une réduction de Turing d'un problème π_1 à un problème π_2 est un algorithme A_1 qui résout π_1 en faisant appel (éventuellement plusieurs fois) à un algorithme A_2 pour π_2 de manière telle que si A_2 est polynomial alors A_1 l'est aussi.

En pratique, l'application suivante de la définition 2.2.3 est utilisée. Soient deux problèmes $\pi_1 = (\mathcal{I}_{\pi_1}, Sol_{\pi_1})$ et $\pi_2 = (\mathcal{I}_{\pi_2}, Sol_{\pi_2})$ et soit (f, g) un couple de fonctions, calculables en temps polynomial, où :

— $f : \mathcal{I}_{\pi_1} \rightarrow \mathcal{I}_{\pi_2} : \forall I \in \mathcal{I}_{\pi_1}, f(I) \in \mathcal{I}_{\pi_2}$

— $g : \mathcal{I}_{\pi_1} \times Sol_{\pi_2} \rightarrow Sol_{\pi_1}; \forall (I, S) \in (\mathcal{I}_{\pi_1} \times Sol_{\pi_2}(f(I))), g(I, S) \in Sol_{\pi_1}(I)$

Supposons enfin l'existence d'un algorithme polynomiale A pour le problème pour le problème π_2. L'algorithme $g \circ A \circ f$ est une réduction (polynomiale) de Turing.

2.2.3 Propriétés structurelles des réductions

Etudions maintenant les propriétés structurelles liées à la notion de réduction (transitivité, fermeture, ...).

THÉORÈME 2.2. 1 : Les réductions de Karp et de Turing sont transitives. En d'autres termes si $\Pi_1 \leq_{T,K} \Pi_2$ et $\Pi_2 \leq_{T,K} \Pi_3$ alors $\Pi_1 \leq_{T,K} \Pi_3$.

Remarquons que, en utilisant la réduction de Turing, la preuve avant marche toujours. Il suffit que, avec la réponse oui, l'algorithme hypothétique pour le voyageur de commerce (noté VOYAGEUR DE COMMERCE dans la suite) fournit également le tour lui même. Modulo cette hypothèse, nous avons prouvé que le problème CIRCUIT HAMILTONIEN$\leq$ VOYAGEUR DE COMMERCE .

De cette preuve, puisqu'un algorithme polynomial pour VOYAGEUR DE COMMERCE aurait suffit pour résoudre non seulement celui-ci mais aussi CIRCUIT HAMILTONIEN, on déduit aussi le corollaire suivant.

COROLLAIRE 2.2. 1 : VOYAGEUR DE COMMERCE (version optimisation) est plus difficile (puissant), par rapport à une résolution polynomiale, que CIRCUIT HAMILTONIEN.

DÉFINITION 2.2.4 : Une classe de complexité $\mathcal{C}$ est dite fermée/close pour une réduction $\leq_r$ si pour n'importe quelle paire de problèmes de décision Π_1 et Π_2 tel que $\Pi_1 \leq_r \Pi_2$, et $\Pi_2 \in \mathcal{C}$ implique que $\Pi_1 \in \mathcal{C}$.

Aussi, remarquons que tant la réduction de Turing que la réduction de Karp préservant l'appartenance à la classe $\mathcal{P}$ dans le sens où si Π se T ou se K-réduit à Π' et $\Pi' \in \mathcal{P}$, alors $\Pi \in \mathcal{P}$.

THÉORÈME 2.2. 2 : Les classes $\mathcal{P}$ et $\mathcal{NP}$ sont closes pour la réduction $\leq$.

PREUVE

— Pour la classe $\mathcal{P}$: soit $\Pi \in \mathcal{P}$ et $\Pi' \leq \Pi$ via f, montrons que $\Pi' \in \mathcal{P}$. Soit k un entier suffisamment grand pour que f soit calculable en temps n^k et Π soit décidable en temps n^k. Voici un algorithme pour Π' sur l'entrée $x \in \Sigma^*_{\Pi'}$:
Calculer $f(x)$ et décider si $f(x) \in \Pi$.
Cet algorithme décide Π' par définition de la réduction f de Π' à Π, et il fonctionne en temps polynomial : le calcul de f prend un temps $\leq n^k$; puisque $|f(x)| \leq n^k$, décider si $f(x) \in \Pi$ prend un temps $\leq (n^k)^k = n^{k^2}$.
Le temps total est $O(n^{k^2})$, donc $\Pi' \in P$.

— Pour la classe $\mathcal{NP}$, le raisonnement est exactement le même. Si Π est reconnu par une machine non déterministe polynomial N_Π, la machine no déterministe pour Π fonctionne ainsi sur l'entrée x : calculer $f(x)$ et exécuter $N_\Pi(f(x))$.

□

La clôture de la classe $\mathcal{P}$ pour $\leq$ signifie qu'un problème Π' plus simple qu'un problème Π efficacement résoluble est lui-même efficacement résoluble, conformément à l'intuition.

LEMME 2.2.1 : La relation $\leq$ est reflexive et transitive.

PREUVE

— Réflexivité : soit Π un problème , alors $\Pi \leq \Pi$ via l'identité.

— Transitivité : soit Π, Π' et Π'' des problèmes tels que $\Pi \leq \Pi'$ et $\Pi' \leq \Pi''$ via g. Alors $x \in \Pi$ si et seulement si $f(x) \in \Pi'$ si et seulement si $g(f(x)) \in \Pi''$, donc $\Pi \leq \Pi''$ via $g \circ f$.

□

DÉFINITION 2.2.5 : Si deux problèmes Π et Π' vérifient $\Pi \leq \Pi'$ et $\Pi' \leq \Pi$ on notera $\Pi \equiv \Pi'$ et on dira que Π et Π' sont équivalents pour les réductions many-one polynomiales.

Un exemple de problèmes équivalents CLIQUE$\equiv$ STABLE (voir théorème 2.6.4). Pour certaines classes usuelles, on ne sait pas si elles contiennent des problèmes complets. Cependant, la magie de cette notion de complétude est l'existence de nombreux problèmes naturels complets pour les classes vues jusqu'à présent, notamment $\mathcal{NP}$. Mais commençons par voir que cette notion est inutile pour la classe $\mathcal{P}$.

LEMME 2.2.2 : Tout problème Π non trivial (c'est-à-dire $\Pi \neq \varnothing$ et $\Pi \neq \Sigma_\Pi^*$) est $\mathcal{P}$-difficile pour les réductions many-one en temps polynomial.

PREUVE
Soit Π non trivial $\pi_0 \notin \Pi$ et $\pi_1 \in \Pi$. Soit $\Pi' \in \mathcal{P}$: montrons que $\Pi' \leq \Pi$. La réduction f de Π' à Π est alors définie comme suit :

$$f(\pi) = \begin{cases} \pi_1 \text{ si } x \in \Pi' \\ \pi_0 \text{ sinon} \end{cases}$$

Puisque $\Pi' \in \mathcal{P}$, cette fonction f est calculable en temps polynomial. Par ailleurs, si $\pi \in \Pi'$ alors $f(\pi) = \pi_1 \in \Pi$ et si $\pi \notin \Pi'$ alors $f(\pi) = \pi_0 \notin \Pi$: ainsi $\pi \in \Pi'$ si et seulement si $f(\pi) \in \Pi$, donc f est une réduction de Π' à Π.

□

Ce résultat vient du fait que la réduction est trop puissante pour la classe $\mathcal{P}$: pour avoir une pertinence, il faudra donc réduire la puissance des réductions. Nous verrons par la suite que nous pouvons considérer pour cela les réduction many-one en espace logarithmique. Il s'agit d'un phénomène général : plus la classe est petite, plus il faut utiliser des réductions faibles pour comparer les problèmes de la classe.

2.3 La classe $\mathcal{P}$ des problèmes

Les problèmes algorithmiques les plus naturels peuvent être classifiés en deux catégories :

— les problèmes dits **polynomiaux** ; ce sont les problèmes donc résolus à l'optimum par des algorithmes dont la complexité est polynomiale, c'est à dire, en $O(n^k)$ où k est une constante indépendante de n ; ils forment la classe $\mathcal{P}$.

— les problèmes dits `non-polynomiaux` ; ce sont les problèmes dont les meilleurs algorithmes (les résolvant à l'optimum) sont de complexité « super-polynomiale », c'est à dire, en $O(f(n)^{g(n)}$ où f et g sont des fonctions croissantes en n et $\lim_{n\to} g(n) = \infty$.

Par exemple, le langage CHEMIN peut être accepté en temps polynomial. Un algorithme acceptant en temps polynomial vérifie que G encode un graphe non orienté, vérifie que u et v sont des sommets de G, utilise une recherche en largeur pour calculer le plus court chemin de u à v dans G, puis compare le nombre d'arêtes du plus court chemin obtenu à k. Si G encode un graphe non orienté et

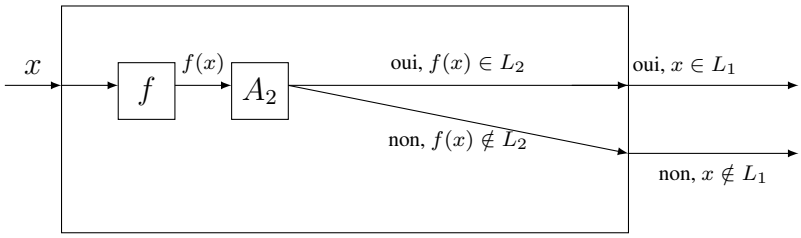
que le chemin de u à v a au plus k arêtes, l'algorithme affiche 1 puis s'arrête. Autrement, l'algorithme tourne indéfiniment.
Toutefois, cet algorithme ne décide pas CHEMIN, puisqu'il n'affiche jamais 0 explicitement pour les instances dans lesquelles le plus court chemin a plus de k arêtes. Un algorithme de décision pour CHEMIN doit rejeter explicitement les chaînes qui n'appartiennent pas à CHEMIN. Pour un problème de décision tel que CHEMIN, ce type d'algorithme de décision est facile à concevoir : au lieu de tourner indéfiniment quand il n'existe pas de chemin de u à v ayant au plus k arcs, l'algorithme affiche 0 et s'arrête. Pour d'autres problèmes, comme le problème de l'arrêt d'une machine de Turing, il existe un algorithme d'acceptation mais pas d'algorithme de décision.
On peut définir de manière informelle une classe de complexité comme un ensemble de langages, dont l'appartenance est déterminée par une mesure de la complexité, tel le temps d'exécution d'un algorithme, qui détermine si une chaîne donnée x appartient au langage L.
En utilisant le cadre de la théorie des langages, on peut proposer une autre définition de la classe de complexité $\mathcal{P}$:

$\mathcal{P} = \{L \subseteq \{0,1\}^* :$ il existe un algorithme A qui décide L en temps polynomial, $\}$

En fait, $\mathcal{P}$ est aussi la classe des langages qui peuvent être acceptés en temps polynomial.

THÉORÈME 2.3. 1 : Considérons le langage suivant : $\mathcal{P}= \{L :$ L est accepté par un algorithme à temps polynomial$\}$.

PREUVE
Puisque la classe des langages décidés par des algorithmes à temps polynomial est un sous-ensemble de la classe des langages acceptés par des algorithmes à temps polynomial, il suffit de montrer que, si L est accepté par un algorithme à temps polynomial, il est décidé par un algorithme à temps polynomial. Soit L le langage accepté par un certain algorithme A à temps polynomial.
On utilise une démonstration classique par « simulation » pour construire un autre algorithme à temps polynomial A' qui décide L. Comme A accepte L en temps $O(n^k)$, pour une certaine constante k, il existe également une constante c telle que A accepte L en au plus $T = cn^k$ étapes.
Pour une chaîne d'entrée x quelconque, l'algorithme A' simule l'action de A pendant le temps T. A la fin du délai T, l'algorithme A' observe le comportement de A.



Algorithme à temps polynomial décidant A_1

FIGURE 2.5 – Illustration de la preuve du lemme 2.3.1. L'algorithme F est un algorithme de réduction qui calcule la fonction de réduction qui calcule la fonction de réduction f de L_1 à L_2 en temps polynomial et A_2 est un algorithme polynomial qui décide L_2. On peut voir si l'algorithme A_1 est capable de décider si $x \in L_1$ en utilisant F pour transformer une entrée x quelconque en $f(x)$, puis en utilisant A_2 pour décider si $f(x) \in L_2$.

— Si A a accepté x, alors A' accepte x en affichant un 1.

— Si A n'a pas accepté x, alors A' rejette x en affichant un 0.

Le travail supplémentaire dû à la simulation de A par A' n'augmente pas le temps d'exécution de plus d'un facteur polynomial, et donc A' est un algorithme à temps polynomial qui décide L.

□

LEMME 2.3.1 : Si $L_1,\ L_2 \subseteq \{0,1\}^*$ sont des langages tel que $L_1 \leq L_2$, alors $L_2 \in \mathcal{P}$ implique $L_1 \in \mathcal{P}$

PREUVE

Soit A_2 un algorithme polynomial qui décide de L_2, et soit F un algorithme de réduction à temps polynomial qui calcule la fonction de réduction f. Nous allons construire un algorithme A_1 polynomial qui décide de L_1.
La figure 2.5 illustre la construction de A_1. Pour une entrée $x \in \{0,1\}^*$ donnée, l'algorithme A_1 utilise F pour transformer x en $f(x)$, puis fait appel à A_2 pour tester si $f(x) \in L_2$. Le résultat de A_2 sera la valeur fournie comme résultat par A_1.
La validité de A_1 se déduit de la définition 2.2.1.
L'algorithme s'exécute en temps polynomial, puisque F et A_2 s'exécutent en temps polynomial

□

2.3.1 La classe des problèmes $\mathcal{NP}$

2.3.1.1 Algorithme de vérification

Nous supposerons dorénavant qu'une solution d'un problème de décision est « oui une telle solution existe et la voilà » ou « non une telle solution n'existe pas ». En d'autres termes, si pour résoudre un problème on avait recours à un « oracle », celui-ci nous fournirait en réponse certes un oui ou non mais dans le premier cas il fournirait également un témoin (vérificateur ou certificat) de la solution. Ce témoin n'est rien d'autre qu'une proposition de solution que l'oracle « affirme » comme étant une réelle solution pour notre problème.
On définit un algorithme de vérification comme un algorithme A à deux arguments, où un argument est une chaîne ordinaire x et l'autre une chaîne binaire y appelée certificat. Un algorithme A à deux arguments vérifie une chaîne x s'il existe un certificat y tel que $A(x, y) = 1$. Le langage vérifié par un algorithme de vérification A est

$$L = \{x \in \{0,1\}^* : \exists y \in \{0,1\}^* | A(x, y) = 1\}$$

Intuitivement, un algorithme A vérifie un langage L si, pour toute chaîne $x \in L$, il existe un certificat y que A peut utiliser pour démontrer que $x \in L$. Par ailleurs, pour toute chaîne $x \notin L$, il ne doit y avoir aucun certificat prouvant que $x \in L$.

DÉFINITION 2.3.1 : La classe de complexité $\mathcal{NP}$ est la classe des langages pouvant être validés par un algorithme polynomial. Plus précisément, un langage L appartient à $\mathcal{NP}$ si et seulement si il existe un algorithme polynomial A à deux entrées et une constante c telle que

$$L = \{x \in \{0,1\}^* : \exists y \text{ avec } |y| = O(|x|^c) | A(x, y) = 1\}$$

On dit que l'algorithme A vérifie le langage L en temps polynomial.

Considérons maintenant les problèmes de décision, dans le sens de la définition 1.2.1 pour lesquels la vérification de la solution proposée peut être effectué en temps polynomial. Les problèmes vérifiant cette propriété forment la classe $\mathcal{NP}$.

DÉFINITION 2.3.2 : Un problème de décision Π est dans $\mathcal{NP}$ si toute solution de Π est vérifiable en temps polynomial.

EXEMPLE
Le problème SATISFAISABILITÉ est dans $\mathcal{NP}$. En effet, étant donné une affectation de valeurs 0 et 1 aux variables d'une instance ϕ de SATISFAI-

SABILITÉ, nous pouvons, avec au plus nm applications du connecteur $\wedge$, décider si l'affectation suggérée est un modèle pour ϕ.

EXEMPLE

VOYAGEUR DE COMMERCE

ENTRÉE : Un ensemble de m villes X, un ensemble de routes entre les villes E, une fonction de coût $v : E \to \mathbb{R}$ où $v(x, y)$ est le coût de déplacement de x à y, $K \in \mathbb{N}$.

QUESTION :Existe-t-'il un tour de K_n de valeur inférieure ou égale à K ?

Nous avons clairement que VOYAGEUR DE COMMERCE $\in \mathcal{NP}$.
En effet, considérons un ensemble d'arêtes de T. Pour déterminer si T est un tour réel dans K_n, on exécute la procédure itérative suivante :

— on marque un sommet du graphe au hasard,

— on cherche dans la solution une arête qui a ce sommet comme extrémité et on la marque ;

— on marque aussi la deuxième extrémité de cette arête ;

— on se place sur ce nouveau sommet marqué et on cherche, parmi les arêtes non marquées, une arête qui l'a comme extrémité, ...

Si on arrête, sans être revenu au sommet initial ou si tous les sommets du graphe ou toutes les arêtes de la solution ne sont pas marqués, ce n'est pas un cycle Hamiltonien et on répond non. Sinon, nous avons un vrai tour et, dans ce cas, il suffit de faire la somme des valeurs de ses arêtes pour calculer sa valeur. Si la somme est inférieure ou égale à K, on répond par oui ; sinon, on répond par non. Le travail que nous avons fourni pour la vérification de T est en $O(n^2)$ (pour un sommet on cherche au pire tous ces voisins pour le test de la valeur, on fait n additions).

2.3.1.2 Quelques mots sur la structure $\mathcal{NP}$

Revenons à la définition 2.3.2 et considérons un problème de décision Π et une instance générique $\mathcal{I}_\pi$ de Π définie sur une structure de données S (par exemple un graphe) et une constante de décision K. De la définition 2.3.2 deux constations fondamentales :

1. s'il existe une solution entraînant une réponse oui pour Π sur I et si cette solution est présentée pour la vérification, alors la réponse de tout algorithme correct de vérification sera toujours oui ;
2. si une telle solution n'existe pas, alors toute proposition de solution présentée pour vérification entraînera la réponse non de la part de tout algorithme correct de vérification.

Considérons maintenant la variant décisionnelle suivante du voyageur de commerce,

CO-VOYAGEUR DE COMMERCE

ENTRÉE : Un ensemble de m villes X, un ensemble de routes entre les villes E, une fonction de coût $v : E \to \mathbb{R}$ où $v(x, y)$ est le coût de déplacement de x à y, and $K \in \mathbb{N}$

QUESTION :Est-il vrai qu'il n'existe pas de cycle Hamiltonien de distance totale inférieure ou égale à K ?

Comment pouvoir certifier que la réponse, sur une instance de ce problème est oui ? Cette question entraîne celle de l'appartenance de ce problème à la classe $\mathcal{NP}$. Nous rencontrons la même situation pour un très grand nombre de problèmes dans $\mathcal{NP}\backslash\mathcal{P}$ (en supposant que $\mathcal{NP}\neq\mathcal{P}$).

2.3.2 La classe $\mathcal{NPO}$

Considérons les problèmes d'optimisation pour lesquels les trois propriétés suivantes sont vérifiées :

1. la réalisabilité d'une solution est vérifiable en temps polynomial
2. la valeur d'une solution réalisable est calculable en temps polynomial ;
3. une solution réalisable est calculable en temps polynomial.

DÉFINITION 2.3.3 : Les problèmes d'optimisation qui vérifient les propriétés 1, 2 et 3 forment la classe $\mathcal{NP}$.

EXEMPLE

COUPLAGE DANS UN GRAPHE

ENTRÉE : $G = (V, E)$ d'ordre n
TÂCHE : Trouver un couplage de taille maximum.

Démontrons que ce problème vérifie les propriétés 1, 2 et 3 :

1. Etant donné une solution, c'est à dire un ensembles d'arêtes, vérifier si elles forment un couplage revient à les rester deux à deux pour contrôler si elles ont ou non une extrémité commune ; si oui, l'ensemble ne constitue pas un couplage ; cette recherche peut être faite en $O(n^2)$ étapes au maximum.
2. La valeur d'une solution réalisable correspond à la taille du couplage ; comme il s'agit d'un couplage réalisable, il possède au maximum $n/2$ arêtes et leur comptage se fait en $O(n)$.
3. Une solution réalisable peut être obtenue en sélectionnant une unique arête au hasard dans le graphe ; cette arête est un couplage à elle seule et la complexité de son choix $O(1)$.

Les trois conditions sont donc réalisées et par conséquent le problème du couplage maximum est dans $\mathcal{NPO}$. Evidemment on peut voir aisément que la variante décisionnelle du couplage maximum appartient à $\mathcal{NP}$.

EXEMPLE

VOYAGEUR DE COMMERCE appartient à $\mathcal{NPO}$. La vérification des propriétés 1 et 2 est montrée précédemment. Démontrons maintenant que la propriété 3 est également vérifiée. Pour déterminer une solution réalisable sur K_n

— on sélectionne un sommet au hasard et on le marque,

— on sélectionne un sommet au hasard parmi ses $n-1$ voisins non-marqués,

— on le marque ainsi que l'arête correspondante, ... ;

— on itère ainsi jusqu'au dernier sommet non marqué puis on revient au sommet de départ (le premier marqué, ceci est toujours possible car le graphe est complet)

La complexité de cette construction est $O(n^2)$.

THÉORÈME 2.3. 2 : La variante décisionnelle d'un problème d'optimisation dans $\mathcal{NPO}$ appartient à $\mathcal{NP}$.

La théorie de la complexité est basée sur les problèmes de décision (et sur les variantes décisionnelles des problèmes d'optimisation).

2.4 $\mathcal{P}$ versus $\mathcal{NP}$

Nous avons au début du chapitre définit la classe $\mathcal{P}$ comme étant la classe de problème résolubles à l'optimum en temps polynomial. Par ailleurs, la classe $\mathcal{NP}$ a

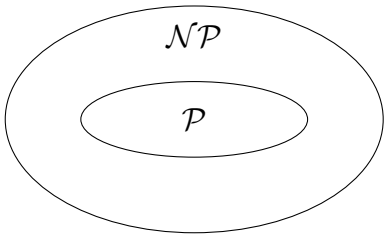


FIGURE 2.6 – $\mathcal{P}$ et $\mathcal{NP}$ (sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$)

été définie (définition 2.3.2) sans référence explicite à la résolution optimale de ses problèmes mais plutôt par référence à la vérification d'une solution donnée.
Bien évidemment, la condition de l'appartenance à $\mathcal{P}$ étant plus forte que celle de l'appartenance à $\mathcal{NP}$ (qui sait résoudre peut vérifier), nous avons $\mathcal{P} \subseteq \mathcal{NP}$.
Une question majeure se pose donc sur la complexité des problèmes dans $\mathcal{NP} \setminus \mathcal{P}$: quelles sont leurs complexités ?
Il est vrai que les meilleurs algorithmes connus pour ces problèmes (des problèmes très naturels du point de vue des applications : VOYAGEUR DE COMMERCE , STABLE... sont de tels problèmes) sont tous super-polynomiaux et toute tentative pour concevoir des algorithmes polynomiaux a été, jusqu'à présent, vouée à l'échec.
Le meilleur résultat général sur la complexité de la résolution des problèmes de $\mathcal{NP}$ est exprimé par le théorème suivant :

THÉORÈME 2.4. 1 : Considérons un problème $\Pi \in \mathcal{NP}$ et une instance générique $\mathcal{I}_\pi$ de Π avec $|I| = n$. Il existe un polynôme p tel que Π peut être résolu par un algorithme de complexité $O(2^{p(n)})$.

La nature des problèmes $\mathcal{NP} \setminus \mathcal{P}$ est-elle foncièrement différente de celle des problèmes de $\mathcal{P}$? Ces problèmes sont-ils réellement intrinsèquement difficiles ou est-ce un manque (crucial pour l'instant) d'outils mathématiques efficaces qui nous empêche de les résoudre polynomialement ? Ces questions passionnantes constituent le cadre de la recherche en théorie de la complexité et appellent des réponses.
Le schéma de la figure 2.13 n'est donc qu'une conjecture. La quasi-totalité des chercheurs en complexité est intimement convaincue de la véracité de cette conjecture mais elle reste toujours une supposition. La question $\mathcal{P} \neq \mathcal{NP}$ est la plus grande question ouverte en informatique et une des plus connues en mathématiques.

2.5 Une notion de complétude pour $\mathcal{NP}$

2.5.1 Définitions

La complétude se réfère toujours à trois éléments :

— une classe de problèmes $\mathcal{C}$,

— une seconde classe $\mathcal{C}'$ plus faible que $\mathcal{C}$ (souvent $\mathcal{C}'$ est un sous-ensemble de $\mathcal{C}$

— et une réduction $\mathcal{R}$ préservant l'appartenance à $\mathcal{C}'$ (ceci est le casa des réductions de Karp et de Turing et de la classe $\mathcal{P}$).

Un problème Π de $\mathcal{C}$ est dit $\mathcal{C}$-complet si tout problème de $\mathcal{C}'$ se $\mathcal{R}$-réduit à Π : l'idée est que, si l'on prouve pour seulement l'un de ces problèmes complets son appartenance à la classe $\mathcal{C}'$, alors on l'aura prouvé pour tout problème de $\mathcal{C}$; par ailleurs, si $\mathcal{C}'$ est un sous-ensemble de $\mathcal{C}$, alors $\mathcal{C}'$ et $\mathcal{C}$ coïncident.

DÉFINITION 2.5.1 : Soit $\mathcal{C}'$ et $\mathcal{C}$ deux classes de problèmes $\mathcal{NPO}$ et $\mathcal{R}$ une réduction préservant l'appartenance à $\mathcal{C}'$. Alors $\mathcal{C}$-complet$= \{\Pi \in \mathcal{C}' : \forall \Pi' \in \mathcal{C}, \Pi' \leq_{\mathcal{R}} \Pi\}$.

Parfois (généralement quand $\mathcal{C}' \subseteq \mathcal{C}$ et qu'il n'y a pas d'ambiguïté la référence à la réduction ou à l'ensemble $\mathcal{C}'$ est omise).
Ici, nous nous intéressons à une complétude particulière avec $\mathcal{C}' = \mathcal{P}$, $\mathcal{C} = \mathcal{NP}$ et $\mathcal{R}$ la réduction de Karp. Informellement, cette complétude signifie que si un seul problème $\mathcal{NP}$-complet est démontré soluble en temps polynomial, alors $\mathcal{P} = \mathcal{NP}$. Supposons qu'en utilisant la réduction de Turing (ou de Karp) nous parvenons à prouver qu'il existe un problème $\Pi_1^* \in \mathcal{NP}$ tel que, pour tout autre problème $\Pi \in \mathcal{NP}$,

$$\Pi \quad \leq_T \quad \Pi_1^* \tag{2.1}$$

D'une telle preuve on en déduit que Π_1^* est le problème le plus difficile de $\mathcal{NP}$. Allons plus loin et supposons que, dans un second temps, nous prouvons également qu'il existe aussi un deuxième problème Π_2^* pour lequel pour tout autre problème $\Pi \in \mathcal{NP}$:

$$\Pi \quad \leq_T \quad \Pi_2^* \tag{2.2}$$

Comme précédemment, nous déduisons que :

— Π_2^* est aussi le problème le plus difficile de $\mathcal{NP}$;

— Π_1^* et Π_2^* sont les deux problèmes les plus difficiles de $\mathcal{NP}$ et sont aussi de difficulté « équivalente » par rapport à leurs résolutions polynomiales ; en effet, d'après l'équation (2.1) $\Pi_1^* \leq_T \Pi_1^*$ et d'après l'équation (2.2) nous avons $\Pi_1^* \leq_T \Pi_2^*$;

— si un algorithme polynomial existe pour n'importe lequel d'entre eux, il servira pour résoudre aussi l'autre en temps polynomial.

Si on poursuit le même processus en mettant en évidence d'autres problèmes « étoilés » nous nous trouvons devant une sous-classe de $\mathcal{NP}$ dont les problèmes :

— sont les plus difficiles de $\mathcal{NP}$;

— sont tous de difficulté équivalente par rapport à leurs résolutions polynomiales ;

— il suffirait de concevoir un algorithme pour l'un d'entre eux pour que tous les problèmes fassent appel à lui pour être résolus en temps polynomial.

Ces problèmes sont appelés $\mathcal{NP}$-complets et la classe de ces problèmes classe $\mathcal{NP}$-complet. La définition est une formulation de ce que nous venons de dire.

DÉFINITION 2.5.2 : Un problème de décision Π est $\mathcal{NP}$-complet si et seulement si il vérifie les deux conditions suivantes :

1. $\Pi \in \mathcal{NP}$;
2. $\forall \Pi' \in \mathcal{NP}, \Pi' \leq_T \Pi$

Un problème qui vérifie la condition 2 (sans vérifier obligatoirement la condition 1 est appelé $\mathcal{NP}$-dur ($\mathcal{NP}$-hard). Par conséquent, une définition alternative à la définition 2.5.2 est la suivante :

DÉFINITION 2.5.3 : Un problème de décision Π est $\mathcal{NP}$-complet si et seulement si il appartient à $\mathcal{NP}$ et est $\mathcal{NP}$-dur.

Il existe une autre définition : On remplace $\mathcal{NP}$-dur par $\mathcal{NP}$-difficile, mais je préfère réserver la notion $\mathcal{NP}$-difficile aux problèmes d'optimisation.

2.5.2 Propriétés autour de la complétude

Plus concrètement pour démontrer qu'un problème Π est $\mathcal{NP}$-complet, il suffit de démontrer les deux propriétés suivantes :

1. $\Pi \in \mathcal{NP}$;

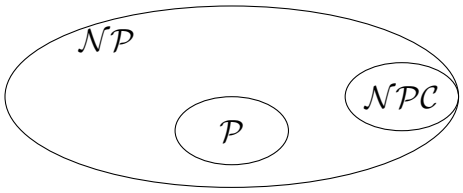


FIGURE 2.7 – Comment la plupart des théoriciens de l'informatique voient les relations entre $\mathcal{P}$, $\mathcal{NP}$, et $\mathcal{NPC}$ sont toutes deux entièrement contenues dans $\mathcal{NP}$, et $\mathcal{P} \cap \mathcal{NPC} = \varnothing$.

2. un problème quelconque $\Pi' \in \mathcal{NP}$ se réduit à Π.

La propriété 1 implique que Π est au moins aussi difficile que n'importe quel problème $\mathcal{NP}$-complet. En effet, Π' étant $\mathcal{NP}$-complet, pour tout autre problème $\Pi'' \in \mathcal{NP}, \Pi'' \leq_T \Pi'$; d'autre part, par la propriété 2, $\Pi' \leq_T \Pi$ et, par la théorème 2.2.1 pour tout $\Pi'' \in \mathcal{NP}, \Pi'' \leq_T \Pi$. Ceci, avec la propriété 2, font que Π remplit les conditions de la définition 2.5.3, c'est à dire $\Pi \in \mathcal{NP}$-complet
Dans l'hypothèse d'existence de problèmes $\mathcal{NP}$-complets, nous pouvons affiner davantage le monde de $\mathcal{NP}$ telle que la figure 2.13 le présente. Ce raffinement, toujours sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$ est montré par la figure 2.13. Bien évidemment si $\mathcal{P} = \mathcal{NP}$ les trois classes de la figure 2.13 coïncident.
De plus, sous l'hypothèse $\mathcal{P} \neq \mathcal{NP}$, les classes $\mathcal{P}$ et $\mathcal{NP}$-complet ne s'intersectent pas.
Notons aussi que, sous la même hypothèse, alors $\mathcal{NP} \neq \mathcal{P} \cup \mathcal{NP}$-complet. En effet, notons par $\mathcal{NP}$-intermédiaire la classe $\mathcal{NP} \setminus (\mathcal{P} \cup \mathcal{NP}\text{-complet})$. Il s'agit, informellement, de la classe des problèmes de difficulté intermédiaire, c'est à dire des problèmes plus difficiles que ceux de $\mathcal{P}$ mais plus facile que ceux de $\mathcal{NP}$-complet.
Plus formellement, pour deux classes de complexité $\mathcal{C}$ et $\mathcal{C}'$ telle que $\mathcal{C}' \subseteq \mathcal{C}$, et une réduction $\mathcal{R}$ préservant l'appartenance à $\mathcal{C}'$, un problème est $\mathcal{C}$-intermédiaire s'il n'est pas ni $\mathcal{C}$-complet sous $\mathcal{R}$, ni dans $\mathcal{C}'$.
Il a été prouvé dès 1975 que sous la K-réduction la classe $\mathcal{NP}$ intermédiaire n'est pas vide. Par contre, nous ne connaissons pas s'il existe de problèmes intermédiaires sous la T-réduction.
Notons enfin que, comme nous l'avons vu tout le long de ce chapitre, la notion de $\mathcal{NP}$-complétude va de pair avec les problèmes de décision. Lorsqu'on traite des problèmes d'optimisation, le terme approprié, utilisé dans la littérature, est

$\mathcal{NP}-$difficile. La définition suivante spécifie la notion de $\mathcal{NP}$-difficulté pour un problème d'optimisation.

DÉFINITION 2.5.4 : Un problème de $\mathcal{NPO}$ est $\mathcal{NP}$-difficile si et seulement si sa variante décisionnelle est un problème $\mathcal{NP}$-complet.

Un des intérêts des problèmes $\mathcal{NP}$-complets réside dans le résultat suivant.

LEMME 2.5.1 :
Les affirmations suivantes sont équivalentes :

1. $\mathcal{P}$=$\mathcal{NP}$
2. Tout problème $\mathcal{NP}$-complet est dans $\mathcal{P}$.
3. Il existe un problème $\mathcal{NP}$-complet dans $\mathcal{P}$.

PREUVE

— $1 \Rightarrow 2$ est évidente.

— $2 \Rightarrow 3$ est évidente car il existe au moins un problème $\mathcal{NP}$-complet

— $3 \Leftarrow 1$. Soit Π un langage de $\mathcal{NP}$, montrons que $\Pi \in \mathcal{P}$. Soit Π' le problème $\mathcal{NP}$-complet donné par 3 : d'une part, $\Pi' \in \mathcal{P}$ par 3; d'autre part, $\Pi \leq \Pi'$ par la complétude de Π'. Puisque $\mathcal{P}$ est clos (Voir Théorème 2.2.2) par réduction many-one, on en déduit que $\Pi \in \mathcal{P}$.

□

2.5.3 Quelques conséquences si $\mathcal{P}$=$\mathcal{NP}$

Prenons l'hypothèse forte que $\mathcal{P}$=$\mathcal{NP}$ et regardons les implications engendrées par cette hypothèse forte.

2.5.3.1 Factorisation

Le problème de factoriser consiste, connaissant un nombre n, de trouver un de ses diviseurs. Ce problème est fondamental en cryptographie; en particulier, le protocole RSA utilise le fait qu'il n'existe pas, à l'heure actuelle, d'algorithme polynomial pour factoriser. La théorie des classes de complexité ne s'applique pas directement au problème de la factorisation : En effet, factoriser n'est pas un problème de décision. Par contre, considérons le problème suivant :

$$L_{fac} = \{(n,k) | n \text{ a un diviseur strictement plus petit que } k\}$$

Par exemple le couple $(29874949, 15) \in L_{fac}$ puisque 29874949 est divisible par 13, qui est plus petit que 15, mais en revanche $(29874949, 11) \notin L_{fac}$. Pour analyser ce problème, nous allons besoin

$$L_p = \{n | n \text{ est premier } \}.$$

THÉORÈME 2.5. 1 : **Agrawal, Kayal, Saxena** 2002 Le problème $L_p \in \mathcal{P}$.

Autrement dit, on peut tester en temps polynomial si un nombre est premier.

LEMME 2.5.2 : Nous avons $L_{fac} \in \mathcal{NP}$.

PREUVE
Evident.

□

LEMME 2.5.3 : Nous avons $L_{fac} \in co\mathcal{NP}$.

PREUVE
Il faut donc donner un algorithme dans $\mathcal{NP}$ qui décide, étant donné n et k, si n n'a pas de diviseur plus petit que k. Pour cela, il suffit de considérer l'algorithme suivant, sur l'entrée n et k :

- Choisir des entiers a_i inférieurs à n.
- Vérifier qu'ils sont tous premiers.
- Vérifier que le produit des a_i vaut n.
- Vérifier qu'aucun des a_i n'est inférieur à k.

Autrement dit, deviner la décomposition en facteurs premiers de n, et vérifier qu'aucun des facteurs premiers n'est inférieur à k. Cet algorithme est bien dans $\mathcal{NP}$. Pour cela, il faut borner le nombre de a_i à prendre à la première étape (si on en prend trop, l'algorithme ne sera plus polynomial).
Cependant, le nombre de facteurs premiers d'un entier n est borné par $\log n$ (puisque tous ses facteurs premiers sont supérieurs à 2, il ne peut en avoir plus de $\log n$). Lors de la première étape, il suffit donc de choisir au plus $\log n$ entiers a_i, c'est-à-dire un nombre linéaire en la taille de l'entrée (qui est $\log n + \log k$).
La première étape de vérification utilise le théorème 2.5.2.

□

THÉORÈME 2.5. 2 : Supposons que $\mathcal{P}= \mathcal{NP}$, il existe un algorithme polynomial pour factoriser.

PREUVE
$L_{fac} \in \mathcal{NP}$ donc sous l'hypothèse, il existe un algorithme polynomial pour décider, étant donné, n et k si n a un diviseur plus petit que k. On va utiliser cet algorithme pour trouver le plus petit diviseur par dichotomie.
Pour cela considérons l'algorithme 2.5.3.1 :

Algorithme 2.1 Algorithme de factoriser.

```
if n est premier then
    RETOURNER premier
else
    a = 2, b = n
end if
while a ≠ b do
    Poser c : (a + b)/2
    if n a un diviseur plus petit que c then
        b = c;
    else
        a = c
    end if
end while
```

Cet algorithme effectue un nombre d'étapes de l'ordre de $\log n$, donc polynomial en la longueur de l'entrée, et chaque étape s'effectue, par hypothèse, en temps polynomial.

□

2.6 Le problème SATISFAISABILITÉ

Le deuxième type de problème que nous utiliserons est celui des problèmes de SATISFAISABILITÉ. Ce type de problème implique des formules propositionnelles. Une VARIABLE BOOLÉENNE est une variable à laquelle on peut assigner soit la valeur VRAI, soit la valeur FAUX.
Un LITTÉRAL est une proposition qui est soit égale à une variable booléenne, soit égale à une négation d'une variable booléenne.
Une CLAUSE est une disjonction de littéraux, par exemple, étant donné trois variables booléennes x_1, x_2 et x_3, le prédicat $x_1 \vee \neg x_2 \vee x_3$ est une clause impliquant les trois littéraux $x_1, \neg x_2$ et x_3.

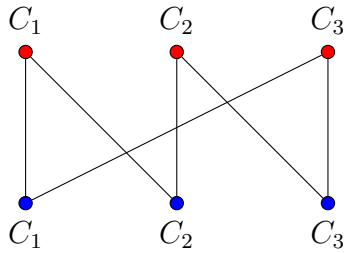


FIGURE 2.8 – Exemple de graphe G_φ pour la formule $\varphi = (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3)$.

Une FORMULE BOOLÉENNE sous FORME NORMALE CONJONCTIVE est une formule propositionnelle composée de conjonctions de clauses. Ainsi la formule $\varphi = (x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3)$ est une formule booléenne composée des trois clauses $C_1 = x_1 \vee x_2, C_2 = \neg x_2 \vee x_3$ et $C_3 = x_1 \vee \neg x_3$.
Pour une variable booléenne x_i, on note ψ_i la liste des clauses dans laquelle la variable apparaît, c'est-à-dire les clauses contenant le littéral x_i ou le littéral $\neg x_i$.
Une assignation β des variables de φ est une attribution de valeurs VRAI ou FAUX à chacune des variables booléennes qui composent la formule. On note par $\beta(x_i)$ la valeur attribuée à la variable x_i dans β.
La formule φ est *satisfaisable* s'il existe une assignation qui satisfait φ et *insatisfaisable* dans le cas contraire. Savoir si oui ou non une formule est satisfaisable est un problème très important en informatique fondamentale. Il peut être formulé comme suit.

SATISFAISABILITÉ

ENTRÉE : Une formule booléenne φ sous forme normale conjonctive.

QUESTION : φ est elle satisfaisable ?

Pour une formule booléenne sous forme normale conjonctive φ, on note G_φ le graphe tel que l'ensemble des sommets de G_φ est composé par l'ensemble des variables et des clauses de φ et l'ensemble des arêtes est donné par $E(G_\varphi) = \{x_i C_\ell \mid C_\ell \in \psi_i\}$ (on ajoute une arête entre une variable et les clauses dans lesquelles elle apparaît).
La figure 2.8 donne un exemple d'un tel graphe.
Le problème SATISFAISABILITÉa été très étudié et de nombreuses variantes existent pour lesquelles des contraintes sont données sur les instances.

— k-SATISFAISABILITÉ : chaque clause composant la formule contient exac-

tement k littéraux. Par exemple la formule $(x_1 \vee \neg x_2) \wedge (x_3 \vee x_4)$ est une formule 2-SAT.

— Monotone : chaque clause contient exclusivement des littéraux positifs ou des littéraux négatifs. Par exemple la formule $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$ est une formule monotone.

— Planaire : **formule SAT planaire** le graphe de la formule est planaire. C'est le cas de la formule illustrée par la figure 2.8 car l'arête entre C_3 et x_1 peut être courbée pour ne pas croiser d'autres arêtes.

— SATISFAISABILITÉ(b) : **formule SAT(b)** le nombre d'occurrences de chaque variable x_i est borné par b, autrement dit $|\psi_i| \leq b$. Ainsi la formule $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_1 \vee x_4)$ est une instance de SAT(3).

Notons qu'il est tout à fait possible de combiner ces contraintes : on peut ainsi formuler le problème 3-SATISFAISABILITÉ(4) Planaire Monotone. Pour les formules insatisfaisables, il peut être intéressant de trouver une assignation qui maximise le nombre de clauses satisfaites. On peut donc formuler le problème d'optimisation suivant.

SATISFAISABILITÉ MAXIMUM

ENTRÉE : Une formule booléenne sous forme normale conjonctive φ.

QUESTION :Une assignation β des variables de φ telle que le nombre de clauses satisfaites par β soit maximum.

On peut là aussi utiliser les mêmes variantes que pour le problème de décision. Une autre variante intéressante concernant ce problème d'optimisation est celle où les clauses sont pondérées. Formellement, une formule booléenne pondérée est composée du 2-uplet (φ, W) tel que φ est une formule booléenne sous forme normale conjonctive et W est une fonction attribuant un poids positif à chacune des clauses. La version impliquant une formule booléenne pondérée est décrite comme suit.

SATISFAISABILITÉ MAXIMUM PONDÉRÉ

ENTRÉE : Une formule booléenne pondérée (φ, W).

QUESTION :Trouver une assignation β des variables de φ telle que la somme des poids des clauses satisfaites par β soit maximum.

On dénote par $\mathrm{W}(\varphi)$ la somme totale des poids des clauses et par $\mathrm{W}(\beta)$ la somme des poids des clauses satisfaites par l'assignation β.
Pour les formules booléennes dans le cas général, c'est-à-dire non nécessairement sous forme normale conjonctive, on peut utiliser une représentation sous la forme

d'un graphe particulier appelé [circuit booléen]circuit booléen. Étant donné une formule booléenne φ, un circuit booléen C de φ est un graphe orienté tel que l'ensemble des sommets de C peuvent être partitionnés en trois ensembles I,O et P tels que :

— chaque sommet de I correspond à une variable de φ et il n'existe pas d'arc entrant incident à un sommet de I ;

— O est composé d'un unique sommet, appelé *sortie* et il existe un unique arc sortant incident à ce sommet ;

— chaque sommet p de P, appelé *porte*, correspond à une fonction logique $p : \{0,1\}^n \mapsto \{0,1\}$.

Pour une assignation β des variables booléennes de φ, on peut évaluer si φ est satisfaite par β en évaluant successivement les valeurs de sortie des portes en partant des sommets de I et en suivant l'orientation des arcs. Les variables assignées à VRAI envoient la valeur 1 sur leurs arcs sortants et les variables assignées à FAUX envoient la valeur 0 sur leurs arcs sortants. Les portes envoient sur leurs arcs sortants le retour de leur fonction logique, en prenant en entrée les valeurs données sur leurs arcs entrants. L'assignation β satisfait φ si et seulement si la valeur 1 est envoyée à la sortie de C, dans ce cas on dit également que β satisfait C. Un problème que l'on peut formuler sur les circuits booléens est le suivant.

CIRCUIT BOOLÉEN

ENTRÉE : Un circuit booléen C.
QUESTION :Une assignation β qui satisfait C.

Notons qu'il existe plusieurs circuits booléens possibles pour représenter une même formule φ, en fonction des portes utilisées dans le circuit. On appelle *hauteur* du circuit booléen la longueur du plus long chemin partant d'un sommet de I et allant vers la sortie. La *trame* (*weft* en anglais) d'un circuit booléen correspond au nombre de portes ayant au moins trois arcs entrants incidents. On peut la plupart du temps diminuer la valeur de la trame en augmentant la hauteur du circuit booléen. Des exemples de circuits booléens sont donnés par la figure 4.2.
Le premier problème démontré comme étant $\mathcal{NP}$-complet par Cook en 1971 est le problème SATISFAISABILITÉ. Puisqu'il s'agissait du premier problème $\mathcal{NP}$-complet, le processus décrit par les définitions 2.5.2 et 2.5.3, supposant l'existence préalable d'un problème $\mathcal{NP}$-complet ne pouvait pas être applicable. Par conséquent, la réduction (qui est souvent appelée réduction générique) démontrant la $\mathcal{NP}$-complétude de SATISFAISABILITÉ utilise la définition de la $\mathcal{NP}$-

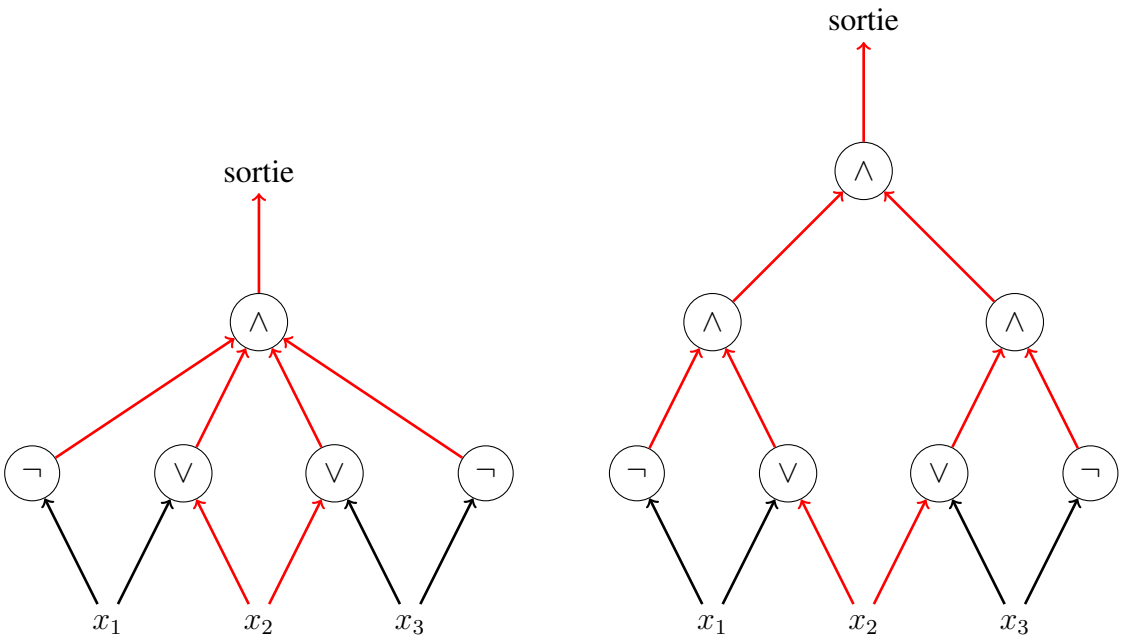


FIGURE 2.9 – Deux exemples de circuits booléens pour la formule booléenne $\neg x_1 \wedge (x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge \neg x_3$. Les flèches sur les arcs représentent leurs orientations. Lors de l'évaluation, les arcs rouges ont la valeur 1 et les arcs noirs ont la valeur 0. Le circuit de gauche a une hauteur égale à trois et une trame égale à un. Le circuit de droite a une hauteur égale à quatre et une trame égale à zéro.

complétude (définition 2.5.2) et repose sur la théorie des langages récursifs et les machines de Turing. L'idée générale de la réduction générique est la suivante :
Pour un problème (langage) générique de $\mathcal{NP}$ on décrit par une CNF le fonctionnement d'un algorithme non-déterministe (machine de Turing) qui résout (décide) ce problème (langage).
De cette manière, Cook parvient à démontrer que si SATISFAISABILITÉ est résoluble en temps polynomial, il en sera de même pour tout problème de $\mathcal{NP}$.
Nous avons donc le théorème suivant :

THÉORÈME 2.6. 1 : [Cook-Levin] Le problème SATISFAISABILITÉ est $\mathcal{NP}$-complet.

2.6.1 Le problème 3-SATISFAISABILITÉ

Le deuxième problème démontré $\mathcal{NP}$-complet est une variante de SATISFAISABILITÉ.

3-SATISFAISABILITÉ

ENTRÉE : Une formule booléenne φ sous forme normale conjonctive avec de clauses de taille trois.
QUESTION : φ est elle satisfaisable ?

CONSTRUCTION 2.6.1 :
L'idée de base pour la transformation d'une instance ϕ de SATISFAISABILITÉen instance ϕ' de 3-SATISFAISABILITÉ est la suivante. Considérons une clause $C_i = \{l_{i_1}, l_{i_2}, \ldots, l_{i_k}\}$ de ϕ :

- si $k = 1$, alors nous introduisons deux nouvelles variables y_{i_1}, et y_{i_2} et construisons la formule $\phi'_i = (l_{i_1} \vee y_{i_1} \vee y_{i_2}) \wedge (l_{i_1} \vee y_{i_1} \vee \bar{y}_{i_2}) \wedge (l_{i_1} \vee \bar{y}_{i_1} \vee y_{i_2}) \wedge (l_{i_1} \vee \bar{y}_{i_1} \vee \bar{y}_{i_2})$.
- si $k = 2$, alors nous introduisons une nouvelle variable y_{i_1} et construisons la formule $\phi'_i = (l_{i_1} \vee l_{i_2} \vee y_{i_1}) \wedge (l_{i_1} \vee l_{i_2} \vee \bar{y}_{i_1})$,
- si $k = 3$, alors la clause reste inchangée :
- si $k > 3$, nous introduisons $k - 3$ nouvelles variables $y_{i_1},\ y_{i_2}, \ldots, y_{i_{k-3}}$ et construisons alors la formule

$$\phi' = (l_{i_1} \vee l_{i_2} \vee y_{i_1}) \wedge_{1 \leq j \leq k-4} (\bar{y}_{i_j} \vee l_{j+2} \vee y_{i_{j+1}}) \wedge (\bar{y}_{i_{k-3}} \vee l_{ik-1} \vee l_{i_k})$$

La formule ϕ' est la conjonction des formules ϕ'_i obtenues comme décrit ci-dessus. On vérifie aisément que ϕ' est une instance de 3-SATISFAISABILITÉ .

La réduction précédente est calculable en temps polynomial.

THÉORÈME 2.6. 2 : [Karp] Le problème 3-SATISFAISABILITÉ est $\mathcal{NP}$-complet.

PREUVE
Nous démontrons maintenant que ϕ' est satisfiable si et seulement si ϕ l'est aussi.

- Supposons qu'il existe une solution pour le problème SATISFAISABILITÉ, c'est-à-dire une affectation des valeurs de vérité telles que les clauses de SATISFAISABILITÉsoient satisfaites. Montrons qu'il existe une solution pour le problème 3-SATISFAISABILITÉ .

  Soit T une affectation de valeurs de vérité sur les variables de ϕ qui satisfit cette formule. Notons aussi que pour chaque clause nous avons introduit des nouvelles variables y qui lui sont propres. Il suffit donc de démontrer comment nous pouvons étendre T pour satisfaire chacune des formules $\phi'_i, i = 1, \ldots, m$ de façon cohérente. Fixons un $i \in \{1, \ldots, m\}$; nous avons alors les cas suivants :

  - si ϕ' a été construite suivant les cas $k = 1$ ou $k = 2$ ci-dessus, alors ϕ'_i est déjà satisfaire par T et nous pouvons étendre T arbitrairement ;

- si ϕ'_i a été construite suivant le cas $k = 3$ ci-dessus, alors ϕ'_i est la clause C_i de ϕ et est déjà satisfaire par T ;
- si ϕ'_i a été construite suivant le cas $k \geq 4$ ci-dessus et puisque T satisfait la clause C_i d'origine, alors soit l le plus petit entier tel que l_x est mis à 1 par T :
  - si $x = 1$ ou $x = 2$, alors nous posons $y_{i_j} = 0, 1 \leq j \leq k-3$;
  - si $x = k-1$ ou $x = k$, alors nous posons $y_{i_j} = 1, 1 \leq j \leq k-3$;
  - dans tous les autres cas, nous posons $y_{i_j} = 1, 1 \leq j \leq k-2$, et $y_{i_j} = 0, l-2 < j \leq k-3$.

  On vérifie aisément que l'affectation T' ainsi construite satisfait chaque ϕ'_i

- Supposons maintenant qu'il existe une solution pour le problème 3-SATISFAISABILITÉ c'est-à-dire une affectation des valeurs de vérité telles que les clauses de 3-SATISFAISABILITÉ soient satisfaites. Montrons qu'il existe une solution pour le problème SATISFAISABILITÉ.
  Si une affectation T' satisfait ϕ', alors sa restriction aux variables de ϕ satisfait ϕ et la preuve du théorème 2.6.2 est complète.

□

2.6.2 Le problème 2-SATISFAISABILITÉ

Dans cette section, nous allons montrer que le problème est polynomial lorsque le nombre de littéraux est deux.
L'idée est de remarquer qu'une clause de taille deux peut toujours s'écrire comme une implication logique. Par exemple la clause $(x_0 \vee x_2)$ peut s'écrire $(\bar{x}_0 \rightarrow x_2)$ ou encore $(\bar{x}_2 \rightarrow x_0)$.

CONSTRUCTION 2.6.2 :
Soit ϕ une instance de 2-SATISFAISABILITÉ , un ensemble de clauses avec deux littéraux chacun. Nous allons définir un graphe $G(\phi)$ de la manière suivante :

- les sommets sont les variables de ϕ et leur négation ;
- $\exists(\alpha, \beta)$ si et seulement si $\exists(\neg\alpha \vee \beta)$ ou $(\alpha \vee \neg\beta)$ dans ϕ. Intuitivement, les arcs représentent l'implication logique $\Rightarrow$ de ϕ.

Nous pouvons remarque une certaine symétrie dans $G(\phi)$; si (α, β) est un arc alors $(\neg\alpha, \neg\beta)$ est également un arc de $G(\phi)$.
La construction est clairement effectuée en temps polynomial.

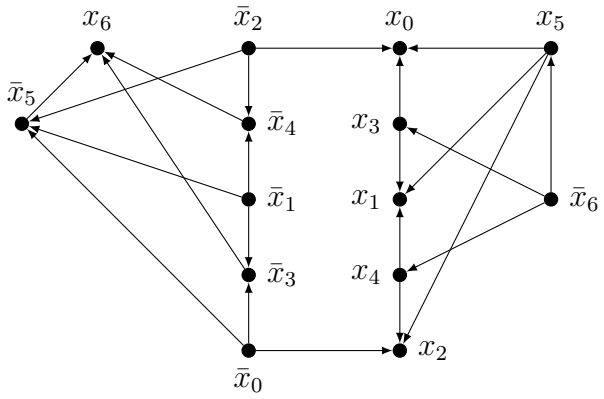


FIGURE 2.10 – Exemple de construction du graphe $G(\phi)$.

EXEMPLE
Considérons la formule suivante $\phi = (x_0 \vee x_2) \wedge (x_0 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_4) \wedge (x_2 \vee \bar{x}_4) \wedge (x_0 \vee \bar{x}_5) \wedge (x_1 \vee \bar{x}_5) \wedge (x_2 \vee \bar{x}_5) \wedge (x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$.
Une illustration de la construction 2.6.2 est donnée par la figure 2.10.

THÉORÈME 2.6. 3 : [Aspvall, Plass, Tarjan] Le problème 2-SATISFAISABILITÉ est polynomial, *i.e.* ϕ est insatisfiable si et seulement si $\exists x$ tels que il existe des chemins de x à $\neg x$ et de $\neg x$ à x dans $G(\phi)$.

PREUVE

- Supposons qu'il existe des chemins de ce type, et que ϕ est satisfiable par une affectation consistante des variables de vérités. Soit $T(x) = true$. Sachant qu'il existe un chemin de x vers $\neg x$ et que $T(x) = true$ et $T(\neg x) = false$, il existe forcément un arc (α, β) sur ce chemin tel que $T(\alpha) = true$ et $T(\beta) = false$. Cependant sachant que (α, β) est un arc $G(\phi)$, on peut conclure que $(\neg\alpha, \beta)$ est une clause de ϕ. Cette clause est non satisfaite par T, une contradiction.

  Remarquons par des arguments équivalents si $T(x) = false$, nous obtenons la même chose.

- Réciproquement, supposons qu'il n'existe aucune variable avec de tels chemins dans $G(\phi)$. Nous allons construire une affectation consistante des variables tel que aucun arc e de $G(\phi)$ allant de true à false.

Nous allons répéter plusieurs fois l'opération suivante :

Prenons un sommet α dont l'affectation positive n'a pas encore affectée et pas de chemin de α à $\neg\alpha$.

Nous considérons tous les sommets atteignables à partir de α dans $G(\alpha)$, et affectons la valeur true.

Nous affectons aussi la valeur false pour les variables apparaissant sous forme négative (les sommets pour lesquels $\neq \alpha$ est atteignable).

Cette itération est bien-définie car, si il existe des chemins de α vers β et $\neg\beta$, alors ce sont également de $\neg\alpha$ vers aussi β et $\neg\beta$ (par symétrie de $G(\phi)$), et par la même des chemins de α vers $\neg\alpha$, une contradiction avec notre hypothèse. Donc, si il avait un chemin de α à un sommet déjà affecté false, une l'étape précédente, alors α est un prédécesseur d'un sommet qui a reçu l'affectation false.

Nous répétons cette étape tant qu'il existe des sommets admettant une affectation positive. Sachant que nous avons supposé qu'il n'existe pas de chemin de x à $\neg x$ et réciproquement, tous les sommets reçoivent une affectation positive.
Il est important de noter qu'à chaque itération, quand un sommet reçoit une affectation positive tous les successeurs reçoivent également cette affectation.. Cette affectation positive satisfait ϕ.

□

2.6.3 Problèmes $\mathcal{NP}$-complets autour des graphes

THÉORÈME 2.6. 4 : Les problèmes VERTEX-COVER , CLIQUE et STABLE sont des problèmes $\mathcal{NP}$-complets.

CONSTRUCTION 2.6.3 :
La construction est basé sur une réduction entre 3-SATISFAISABILITÉ à VERTEX-COVER .
Soit π une instance de 3-SATISFAISABILITÉ , construisons une instance π' de VERTEX-COVER de la manière suivante : ce graphe est composé de deux ensembles de composantes reliés par des arêtes.

- le premier ensemble est constitué de $2n$ sommets $x_1, \bar{x}_2, \ldots, x_n, \bar{x}_n$, et de n arêtes $(x_i, \bar{x}_i), i = 1, \ldots, n$, qui relient les sommets par paires.Chaque sommet correspond à un littéral, sert à définir les valeurs de vérité de la solution.
- Le deuxième est constitué de m triangles (c'est-à-dire, de m cliques à 3 sommets) sommets disjoints. Pour une clause C_i, nous notons par c_{i_1}, c_{i_2} et c_{i_3} les sommets du triangle correspondant. Cet ensemble de triangles correspond aux clauses et chaque sommet y est associé à un littéral de cette clause. Ces triangles servent à vérifier la satisfaction des clauses.
- On ajoute $3m$ arêtes « unificatrices » qui relient chaque sommet de chaque « triangle »-clause- au « sommet-libéral » qui correspond. Remarquons que le nombre de voisins est de 3, un pour chaque sommet.
- Finalement, on pose $k = n + 2m$.

La construction de l'instance π' se fait donc en temps polynomial puisque $|V| = 2n + 3m$ et $|E| = n + 6m$.

EXEMPLE
Considérons l'instance ϕ suivante :

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$$

Les littéraux associés aux quatre variables sont : $x_1, \bar{x}_1, x_2, \bar{x}_2, x_3, \bar{x}_3, x_4, \bar{x}_4$.
Le graphe G est donné par la figure 2.11 avec $k = 12$.

PREUVE
Le problème VERTEX-COVER appartient à $\mathcal{NP}$. Nous allons maintenant démontrer que G admet une couverture de sommets de taille inférieure ou égale à $n + 2m$ si et seulement si la formule ϕ est satisfiable.

- Supposons qu'il existe une affectation qui satisfasse la formule ϕ, montrons qu'il existe une solution S pour le problème VERTEX-COVER de taille $n + 2m$ comme suit :

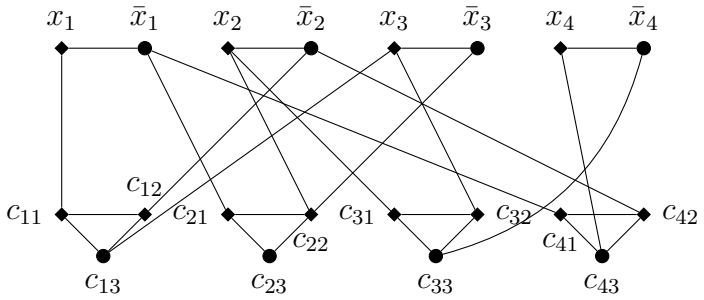


FIGURE 2.11 – Exemple d'utilisation de la construction 2.6.3 pour la formule $\phi = \{C_1, C_2, C_3, C_4\}$ avec $C_1 = (x_1 \vee \bar{x}_2 \vee x_3), C_2 = (\bar{x}_1 \vee x_2 \vee \bar{x}_3), C_3 = (x_2 \vee x_3 \vee \bar{x}_4), C_4 = (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$.

— Pour toute variable x_i, si $x_i = 1$, alors l'extrémité x_i de l'arête $(x_i, \bar{x}_i)$ est mise dans S ; sinon, l'extrémité $\bar{x}_i$ de l'arête $(x_i, \bar{x}_i)$ y est mise ; nous couvrons ainsi les arêtes du type $(x_i, \bar{x}_i), i = 1, \ldots, n$, et une arête unificatrice par triangle.

— soit T_i (correspondent à la clause $C_i, i = 1, \ldots, m$, un triangle et soit (l_k, c_{i_1}) l'arête unificatrice couverte par la mise à 1 de l_k ; nous mettons alors S les sommets c_{i_2} et c_{i_3} ; ces sommets couvrent à la fois les arêtes de T_i et les deux arêtes unificatrices ayant comme extrémités c_{i_2} et c_{i_3}, respectivement ;

— en étirant cette opération pour chaque triangle, uns solution S de taille $n + 2m$ est finalement construit en temps polynomial.

- Inversement, supposons qu'il existe une solution S pour le problème VERTEX-COVER , et montrons qu'il existe une affectation qui satisfasse la formule ϕ.

  Remarquons que chaque triangle (resp. pair $(x, \bar{x})$) il faut au moins deux sommets (resp. un sommet) couvrir les arêtes.

  Nous posons $x_i = 1$ si l'extrémité x_i de l'arête $(x_i, \bar{x}_i)$ est prise dans S ; si l'extrémité $\bar{x}_i$ y est incluse, alors on pose $\bar{x}_i = 1$, c'est-à-dire, $x_i = 0$.

  Montrons que cette affectation de valeurs de vérité aux variables satisfait ϕ. En effet, puisque seulement une extrémité de chaque arête $(x_i, \bar{x}_i)$ est prise dans S, *un seul littéral pour chaque variable est mis à* 1, et par la même l'affectation en question est cohérente (à chaque littéral est affecté une et une seule valeur de vérité).

Par ailleurs, considérons un triangle T_i de G correspondant à la clause C_i, notons ses sommets par c_{i_1}, c_{i_2} et c_{i_3} et supposons que les deux derniers appartiennent à S. Supposons aussi que l'arête unificatrice ayant comme extrémité le sommet c_{i_1} est l'arête (c_{i_1}, l_k), l_k étant un des littéraux associés à la variable x_k. Puisque $c_{i_1} \notin S$, l_k y appartient, c'est-à-dire, $l_k = 1$; l'existence de l'arête (c_{i_1}, l_k) signifiant que le littéral l_k appartient à la clause C_i, celle-ci est avérée par la mise de l_k à 1.

En itérant cet argument pour toutes les clauses, la nécessité de la condition est démontrée. De plus, remarquons que l'obtention de l'affectation de valeurs de vérité aux variables de ϕ est faite en temps polynomial.

Pour les problèmes CLIQUE et STABLE, il est important de noter qu'un VERTEX-COVER quelconque dans le complémentaire dans le graphe G obtenu par la construction 2.6.3. Pour CLIQUE et STABLE, on peut remarquer que si il existe une clique de taille k dans un graphe G il existe un stable de taille $n - k$ dans le graphe $\bar{G}$ (graphe complémentaire de G, avec $G + \bar{G} = K_n$).
Ainsi, CLIQUE$\equiv$ STABLE comme indiqué par la définition 2.2.5.

□

THÉORÈME 2.6. 5 : Le problème CIRCUIT HAMILTONIEN est $\mathcal{NP}$-complet.

CONSTRUCTION 2.6.4 :
La construction est basé sur une réduction entre 3-SATISFAISABILITÉ à CIRCUIT HAMILTONIEN. La construction est assez sophistiquée. Je vous invite à regarder la vidéo : https ://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/threeSATtohamiltonianCycle.html
Il est important de comprendre la construction proposée. Elle servira pour la preuve du théorème 4.2.2.

PREUVE
Le problème CIRCUIT HAMILTONIEN appartient à $\mathcal{NP}$.

-
-

□

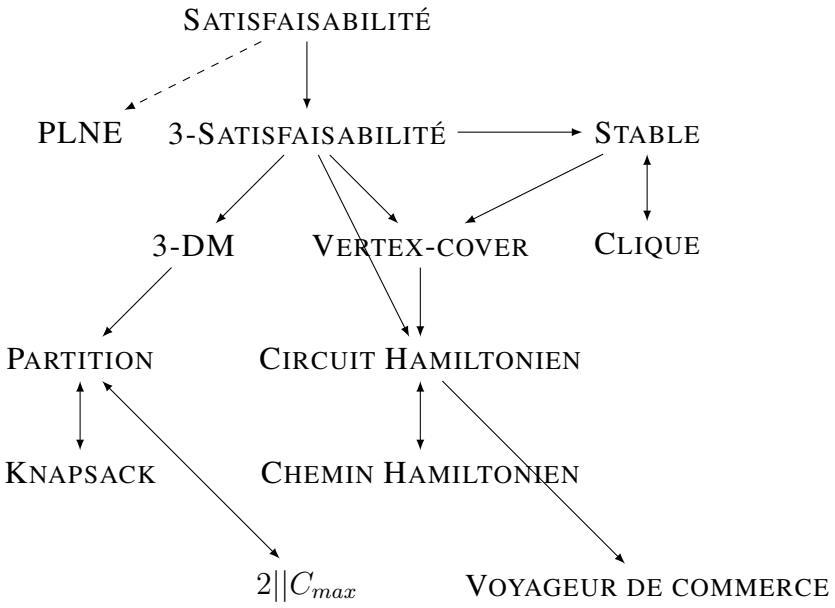


FIGURE 2.12 – Quelques réductions classiques.

2.7 Les classes $co\mathcal{NP}$ et $co\mathcal{NP}$-complet

De façon générale, notons par $\mathcal{I}_\pi$ l'ensemble de toutes les instances d'un problème de décision $\pi \in \mathcal{NP}$ et par $\mathcal{O}_\pi$ le sous-ensemble de $\mathcal{I}_\pi$ pour lequel la solution est oui, c'est à dire l'ensemble des oui-instances (ou instances positives) de π.

DÉFINITION 2.7.1 : Notons par $\bar{\pi}$ le problème ayant comme oui-instances l'ensemble : $\mathcal{I}_\pi \backslash \mathcal{O}_\pi$. Tous ces problèmes forment la classe $co\mathcal{NP}$, c'est à dire $co\mathcal{NP} = \{\bar{\pi} : \pi \in \mathcal{NP}\}$.

CONJECTURE 2.7.1 : La conjecture, largement partagée par la communauté des chercheurs en complexité par rapport à la relation entre $\mathcal{NP}$ et $co\mathcal{NP}$, est que $\mathcal{NP} \neq co\mathcal{NP}$, mais la question est-ce $\mathcal{NP} = co\mathcal{NP}$? reste ouverte.

Cette conjecture est considérée comme étant plus forte que $\mathcal{NP} \neq \mathcal{P}$, dans le sens où il se pourrait que $\mathcal{NP} \neq \mathcal{P}$ même si $\mathcal{NP} = co\mathcal{NP}$ (mais si $\mathcal{NP} = \mathcal{P}$, alors $\mathcal{NP} = co\mathcal{NP}$).

Bien évidemment pour tout problème de décision $\pi \in \mathcal{P}$, le problème $\bar{\pi}$ appartient également à $\mathcal{P}$; par conséquent , $\mathcal{P} \subseteq \mathcal{NP} \cap co\mathcal{NP}$.

THÉORÈME 2.7. 1 : $\mathcal{P} \subseteq \mathcal{NP}$ et $\mathcal{P} \subseteq co\mathcal{NP}$.

PREUVE
Soit $\pi \in \mathcal{P}$ et soit A un algorithme permettant de résoudre chaque instance de π. Pour une instance $\pi \in \mathcal{O}_\pi$, on peut vérifier que $\pi \in \mathcal{O}_\pi$ en appliquant A, qui donne la réponse oui en temps polynomial. Ceci prouve que $\mathcal{P} \subseteq \mathcal{NP}$. La preuve pour $\mathcal{P} \subseteq co\mathcal{NP}$ est similaire. □

THÉORÈME 2.7. 2 : ($\mathcal{NP}$-complet $\cap \mathcal{P} \neq \varnothing$) $\Leftrightarrow$ ($\mathcal{P} = \mathcal{NP}$)

PREUVE

— $\Leftarrow$ Supposons $\mathcal{P}=\mathcal{NP}$.
On a alors $\mathcal{P}= \mathcal{NP}$-complet car $\mathcal{NP}$-complet $\subseteq \mathcal{NP}=\mathcal{P}$ et si $\Pi \in \mathcal{P}$, on a $\Pi \in \mathcal{NP}$ et $\Pi' \in \mathcal{NP}= \mathcal{P}$ (puisqu'on peut résoudre Π' en faisant un nombre polynomial d'opérations élémentaires $+$ un appel à la boîte noire qui résout Π.
Comme $\mathcal{P} \neq \varnothing$, on a donc $\mathcal{NP}$-complet $\cap \mathcal{P}=\mathcal{P} \neq \varnothing$.

— $\Rightarrow$ Supposons que $\mathcal{NP}$-complet $\cap \mathcal{P}=\mathcal{P} \neq \varnothing$.
On sait déjà que $\mathcal{P} \subseteq \mathcal{NP}$. Soit $\Pi \in \mathcal{NP}$. Il suffit de montrer que $\Pi \in \mathcal{P}$. Choisissons Π' dans $\mathcal{NP}$-complet $\cap \mathcal{P}$. On a alors $\Pi \leq_T \Pi'$ car Π' est $\mathcal{NP}$-complet. Mais on sait par le lemme 2.3.1 que $\Pi \leq_T \Pi'$ et $\Pi \in \mathcal{P}$ implique $\Pi \in \mathcal{P}$.

□

Prouver que $\mathcal{NP} \neq co\mathcal{NP}$ serait plus fort que prouver $\mathcal{P} \neq \mathcal{NP}$ pour la raison suivante :

LEMME 2.7.1 : Si $\mathcal{NP} \neq co\mathcal{NP}$, alors $\mathcal{P} \neq \mathcal{NP}$.

PREUVE
Nous allons prouver le contraire : $\mathcal{P}= \mathcal{NP}$ implique que $\mathcal{NP}=co\mathcal{NP}$. Le point central c'est que la classe $\mathcal{P}$ est fermé par complémentaire. Ainsi, si $\mathcal{P}= \mathcal{NP}$, alors $\mathcal{NP}$ serait aussi clôt par complémentaire.

— Plus formellement, partons de l'hypothèse que $\mathcal{P}= \mathcal{NP}$, nous avons :

$$\Pi \in \mathcal{NP} \Rightarrow \Pi \in \mathcal{P} \Rightarrow \bar{\Pi} \in \mathcal{P} \Rightarrow \bar{\Pi} \in \mathcal{NP} \Rightarrow \Pi \in co\mathcal{NP}$$

Et

— de manière similaire :

$$\Pi \in co\mathcal{NP} \Rightarrow \bar{\Pi} \in \mathcal{NP} \Rightarrow \bar{\Pi} \in \mathcal{P} \Rightarrow \Pi \in \mathcal{NP} \Rightarrow \Pi \in \mathcal{NP}$$

Ainsi, nous pourrions avoir $\mathcal{NP} \subseteq co\mathcal{NP}$ et $co\mathcal{NP} \subseteq \mathcal{NP}$, d'où $\mathcal{NP} = co\mathcal{NP}$. □

CONJECTURE 2.7.2 : $\mathcal{P} \neq \mathcal{NP}$.

CONJECTURE 2.7.3 : (Edmonds) $\mathcal{P} = \mathcal{NP} \cap co\mathcal{NP}$.

Les problèmes Π appartenant à $\mathcal{NP} \cap co\mathcal{NP}$ sont appelés les problèmes biens formés.

LEMME 2.7.2 : Si Π est $\mathcal{NP}$-complet alors $\bar{\Pi}$ est $co\mathcal{NP}$-complet.

PREUVE

Comme $\Pi \in \mathcal{NP}$, par définition $\bar{\Pi}$ est dans $co\mathcal{NP}$. Il reste à prouver qu'il est $co\mathcal{NP}$-dur.
Soit $\Pi' \in co\mathcal{NP}$: par définition $\bar{\Pi}' \in \mathcal{NP}$, donc $\bar{\Pi}'$ se réduit à Π via une réduction f calculable en temps polynomial. Donc $x \in \bar{\Pi}'$ ssi $f(x) \in \Pi$.
Ainsi, nous obtenons que $x \in \Pi'$ ssi $f(x) \in \bar{\Pi}$ c'est-à-dire que Π' se réduit à $\bar{\Pi}$.
Donc $\bar{\Pi}$ est $\mathcal{NP}$-complet. □

Si $\Pi \in \mathcal{NP}$ alors $\forall x \in \Pi$ possède un certificat succinct. Si $\bar{\Pi} \in co\mathcal{NP}$ alors $\forall x \in \bar{\Pi}$ possède un disqualificateur succinct.

TAUTOLOGIE

ENTRÉE : Soit ϕ une formule booléennes.
QUESTION : ϕ est-elle satisfaite pour n'importe quelle affectation ?

THÉORÈME 2.7. 3 : Le problème TAUTOLOGIE est $co\mathcal{NP}$-complet.

PREUVE

Si ϕ est une formule booléenne à n variables, alors c'est une tautologie si et seulement si elle est satisfaite par une affectation de n variables. Il existe donc une machine de Turing à temps polynomiale V telle que pour tout $x \in \{0,1\}^n$, V évaluera ϕ avec x comme affectation à ses variables. La formule ϕ est une tautologie si elle vaut vrai (1) pour tout x. Donc TAUTOLOGIE $\in co\mathcal{NP}$.
Nous montrons maintenant que chaque langue $L \in co\mathcal{NP}$ est réductible à la TAUTOLOGIE . On prend $\bar{L}$, le complément de L. Si $L \in co\mathcal{NP}$, alors $L \in \mathcal{NP}$. Donc, par réduction nous obtenons ϕ_x. On sait, $x \in L$ si et seulement si $x \notin \bar{L}$ si et seulement si ϕ_x est insatisfiable. Donc, $x \in L$ si et seulement si $\neg\phi_x$ est une tautologie. Donc la réduction est, pour tout $x \in \{0,1\}^*$, crée ϕ_x par réduction et prenez la négation de la formule. □

Une « non »-instance du problème dans $co\mathcal{NP}$ possède un preuve courte appartenant à une « non »-instance.

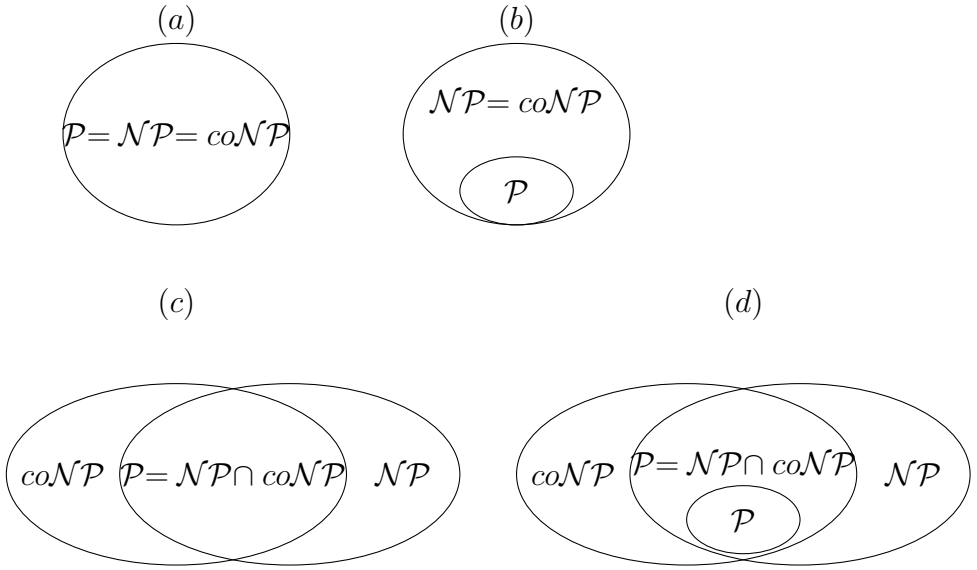


FIGURE 2.13 – Quatre possibilités de relations entre les classes de complexité. Dans chaque diagramme, un renfermant une autre indique une relation de sous-ensemble propre. (a) $\mathcal{P}$=$\mathcal{NP}$=co$\mathcal{NP}$. La plupart des chercheurs considèrent cette possibilité comme étant la plus improbable. (b) Si $\mathcal{NP}$ est fermé pour le complément, alors $\mathcal{NP}$= co$\mathcal{NP}$, mais il n'est pas nécessaire que $\mathcal{P}$= $\mathcal{NP}$. (c) $\mathcal{P}$= $\mathcal{NP}$∩ co$\mathcal{NP}$, mais $\mathcal{NP}$ n'est pas fermé pour le complément. (d) $\mathcal{NP}$ $\neq$ co$\mathcal{NP}$ et $\mathcal{P}$ $\neq$ $\mathcal{NP}$∩ co$\mathcal{NP}$. La plupart des chercheurs considère cette possibilité comme étant la plus probable.

On en déduit que les problèmes CO-SATISFAISABILITÉ, CO-CLIQUE, etc., sont $co\mathcal{NP}$-complets.

2.8 La classe $\mathcal{NP}$-intermédiaire

Dans la classe $\mathcal{NP}$, nous avons donc d'un côté des problèmes $\mathcal{NP}$-complets qui sont difficiles si $\mathcal{P} \neq \mathcal{NP}$, et de l'autre des problèmes $\mathcal{P}$ qui sont faciles.
Si $\mathcal{P} \neq \mathcal{NP}$, y a-t-il quelque chose entre les deux ? En d'autres termes, si $\mathcal{P} \neq \mathcal{NP}$ existe-t-il des problèmes de $\mathcal{NP}$ qui soient hors de $\mathcal{P}$ mais pas $\mathcal{NP}$-complet (problèmes « intermédiaires ») ?

THÉORÈME 2.8. 1 : Si $\mathcal{P} \neq \mathcal{NP}$ alors il existe un problème $\Pi \in \mathcal{NP}$ tel que :

- $\Pi \notin \mathcal{P}$
- Π n'est pas $\mathcal{NP}$-complet.

PREUVE
La preuve est technique et est donnée par Ladner. □

Jusqu'à présent on n'a exhibé aucun problème naturel dans cette classe, mais le problème ISOMORPHISME DE GRAPHE qui est clairement $\mathcal{NP}$ semble être un bon candidat.

ISOMORPHISME DE GRAPHE

ENTRÉE : Soient G_1 et G_2 deux graphes non orientés.
QUESTION : G_1 et G_2 sont-ils isomorphes ?

CONJECTURE 2.8.1 : Si $\mathcal{P} \neq \mathcal{NP}$, alors ISOMORPHISME DE GRAPHE n'est pas $\mathcal{NP}$-dur, ni polynomial.

2.9 Conclusion

Dans ce chapitre nous avons décrit des classes de complexité classiques, et nous avons procédé à plusieurs réductions.

Chapitre

3 Classes intérieures à $\mathcal{P}$: $\mathcal{L}, \mathcal{NL}$

Sommaire

Résumé

Dans ce chapitre, nous étudions des classes de complexité plus petites que $\mathcal{P}$ permettant une classification plus fine des problèmes combinatoires.

3.1 Introduction

A l'opposé du chapitre précédent dans lequel nous avons étudié des classes plus grande que la classe $\mathcal{P}$, dans celui-ci nous proposons d'étudier les classes de complexité que $\mathcal{P}$. Pour cela nous définissons la réduction logarithmique.

3.2 Les problèmes $\mathcal{P}$-complets

De manière similaire à la caractérisation de la classe $\mathcal{NP}$-complétude, on peut définir la $\mathcal{P}$-complétude. C'est la propriété de la classe $\mathcal{P}$qui ne sont les plus difficiles à résoudre, c'est-à-dire vers lesquels tous les problèmes de $\mathcal{P}$peuvent être réduits. Il faut néanmoins faire attention au genre de réductions employée, car son coût ne doit pas masquer celui de la résolution des problèmes.

DÉFINITION 3.2.1 : Une réduction many-one en espace logarithmique d'un problème B (sur l'alphabet Σ_B) à un problème A (sur l'alphabet Σ_A) est une fonction $f : \Sigma_B^* \to \Sigma_A^*$ calculable en espace logarithmique telle que :

$$\forall x \in \Sigma_B^*, x \in B \Leftrightarrow f(x) \in A.$$

Si une telle fonction f existe, on dira que B se réduit à A (via f) en espace logarithmique et on notera $B \leq_{\log} A$.

Après la réduction calculatoire proposée dans le cours de calculabilité la réduction polynomiale (voir section 2.2), la réduction logarithmique est le troisième outil d'analyse d'un ensemble de fonctions. Le premier permet de distinguer ce qui est calculable de ce qui ne l'est pas. A l'intérieur du calculable, le second distingue ce qui est praticable de ce qui ne l'est pas. Et à l'intérieur du praticable, le troisième distingue ce qui est vraiment facile du reste.
Chaque niveau d'analyse est plus fin que le précédent et nécessite une réduction adaptée où la fonction de traduction à la bonne force. C'est ainsi qu'on passe de la réduction calculatoire à la réduction polynomiale, puis à la réduction logarithmique. On peut définir d'autres réductions fonctionnant selon les mêmes principes pour caractériser d'autres classes de fonctions. La réduction linéaire permet ainsi de caractériser des classes de problèmes de complexités analogues puisque ne différant que d'un facteur constant.

DÉFINITION 3.2.2 : **$\mathcal{P}$-complétude** On dit qu'un problème π est $\mathcal{P}$-complet si et seulement si :

1. $\pi \in \mathcal{P}$, et
2. $\forall \pi' \in \mathcal{P}$, $\pi' \leq_{\log} \pi$.

De nombreux problèmes sont $\mathcal{P}$-complets :

1. Evaluation de circuit booléen :

   VALEUR D'UN CIRCUIT

   ENTRÉE : Soit un réseau de portes logiques C composées par des portes $\vee$, $\wedge$ et $\neg$, $x_1,\ x_2, \ldots, x_n$ et une sortie y.
   QUESTION :Est-ce que $y = true$ sur l'entrée $x_1,\ x_2, \ldots, x_n$?

   considérons un réseau de portes logiques ($\wedge$, $\vee$, ...), sans boucle, et des valeurs à donner aux entrées. Le but est de calculer la valeur de sorites. Ce problème est le contraire du problème SATISFAISABILITÉ puisque dans ce dernier on fixe la sorite et on cherche dessaleurs d'entrée qui donnent cette sortie. Le problème de l'évaluation de circuit booléen est $\mathcal{P}$-complet et joue un rôle similaire à celui que SATISFAISABILITÉ joue pour les problèmes $\mathcal{NP}$-complets. C'est le premier problème qu'on prouve complet en modélisant chaque calcul polynomial en circuit booléen. Les autres problèmes $\mathcal{P}$-complets sont prouvés tels que par réduction vers chacun d'eux du problème de l'évaluation de circuit booléen, ou d'un autre problème déjà prouvé $\mathcal{P}$-complet.

2. Programmation linéaire :

   Soit A une matrice $n \times d$, b un vecteur $n \times 1$, et c un vecteur $1 \times d$. Le but est de trouver un vecteur x de taille $d \times 1$ à valeurs rationnelles tel que $Ax \leq b$ et cx est le plus grand possible. Dans ce problème, A est un ensemble de contraintes qui pèsent sur x, et c représente une fonction d'objectif qu'il faut maximiser. Ce problème est $\mathcal{P}$-complet. Sa variante où les solutions sont à valeurs entières devient $\mathcal{NP}$-complets.

3. Flot maximal.

   Soit un réseau de transport dont chaque liaison a une capacité, soit la donnée d'un sommet source et d'un sommet puits, et une spécification de flot f (par exemple une quantité de marchandise à faire transiter au travers du réseau). Vérifier si ce réseau peut supporter un flot supérieur ou égal à f entre la source elle puits. Ce problème, qu'on peut aussi rattacher au domaine de la recherche opérationnelle, est $\mathcal{P}$-complet. On le rencontre dès qu'il est

question de gérer un trafic sur un réseau (routier, électrique, informatique, ...

4. Analyse syntaxique : Soit une grammaire G, et une séquence m. On note $\mathcal{L}(G)$ le langage engendré par G (i.e. toutes les phrases autorisées par la grammaire). Le but est de déterminer si $m \in \mathcal{L}(G)$. Ce problème $\mathcal{P}$-complet est au cœur des interfaces et des outils d'analyse de programmes.

3.3 Classes plus petite que $\mathcal{P}$

3.3.1 Espace logarithmique

Cette remarque nous amène à la définition de la classe très usuelle des langages reconnus en espace logarithmique. Par abus de langage, on notera $\log n$ pour désigner la fonction sur les entiers $\lfloor \log n \rfloor$ afin de respecter la définition ci-dessus, bien que cela ne change pas grand-chose puisque l'espace est mesuré à une constante près.

DÉFINITION 3.3.1 : La classe $\mathcal{L}$ est l'ensemble des langages reconnus en espace logarithmique $O(\log n)$, c'est-à-dire $\mathcal{L}=\mathcal{PSPACE}(\log n)$.

La restriction sur l'espace semble si forte qu'on peut se demander ce qu'il est possible de calculer en espace logarithmique. Voici quelques exemples.

1. Machine qui ajoute 1 à un nombre binaire donné en entrée. Cette machine est en fait un automate fini puisqu'elle n'a pas besoin de ruban de travail : elle ne dispose que de son ruban d'entrée en lecture seule et de son ruban de sortie en écriture seule. Son fonctionnement est le suivant :
   - elle se déplace sur la dernière lettre de son ruban d'entrée ;
   - de droite à gauche, tant qu'elle lit un 1 sur son ruban d'entrée, elle écrit 0 sur son ruban de sortie (la retenue se propage) ;
   - dès qu'elle lit 0 en entrée elle écrit 1 en sortie (la retenue) et continue son parcours de droite à gauche, en recopiant maintenant chaque symbole sur la sortie (il n'y a plus de retenue) ;
   - si elle arrive au début de l'entrée sans avoir vu de 0, alors elle ajoute un 1 en tête (la retenue) sur le ruban de sortie.
2. La décrémentation est calculée de manière identique.

LEMME 3.3.1 : Tout problème A non trivial (c'est-à-dire $A \neq \varnothing$ et $A \neq \Sigma_A^*$) est $\mathcal{P}$-difficile pour les réductions many-one en temps polynomial.

PREUVE

Soit A non trivial, $x_0 \notin A$ et $x_1 \in A$. Soit $B \in \mathcal{P}$: montrons que $B \leq A$. La réduction f de B à A est alors définie comme suit :

$$f(x) = \begin{cases} x_1 \text{ si } x \in B, \\ x_0 \text{ sinon} \end{cases}$$

Puisque $B \in \mathcal{P}$, cette fonction f est calculable en temps polynomial. Par ailleurs, si $x \in B$ alors $f(x) = x_1 \in A$ et si $x \notin B$ alors $f(x) = x_0 \notin A$: ainsi, $x \in B$ si et seulement si $f(x) \in A$, donc f est une réduction de B à A.

□

Ce résultat vient du fait que la réduction est trop puissante pour la classe $\mathcal{P}$: pour avoir une pertinence, il faudra donc réduire la puissance des réductions. Nous verrons par la suite. que nous pouvons considérer pour cela les réduction many-one en espace logarithmique. Il s'agit d'un phénomène général : plus la classe est petite, plus il faut utiliser des réductions faibles pour comparer les problèmes de la classe.

3.4 Autres classes de complexité

Dans cette section nous considérons des classes de complexité plus petite que $\mathcal{P}$. Nous allons caractériser les classes $\mathcal{NL}$ et $co\mathcal{NL}$. La classe $\mathcal{NL}$ est une classe relativement petite parmi les classes usuelles. On a notamment $\mathcal{NL} \subseteq \mathcal{P}$. $\mathcal{NL}$, c'est l'ensemble des problèmes de décision qui peuvent être décidés par des machines de Turing non déterministes dont l'espace de travail est borné par une fonction logarithmique.

DÉFINITION 3.4.1 : Soit $\mathcal{SPACE}f(n)$ la classe de problèmes qui peuvent être résolus, sur une instance de taille n, en utilisant un espace mémoire en $O(f(n))$

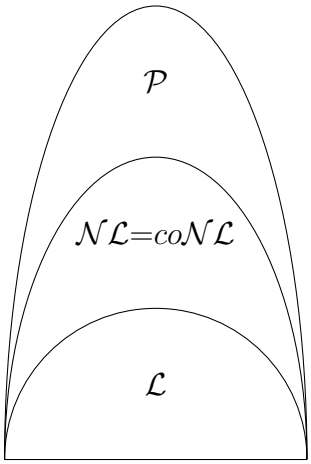


FIGURE 3.1 – Une comparaison des classes de complexité

DÉFINITION 3.4.2 : La classe $\mathcal{PSPACE}$ est la classe des langages décidés par une machine de Turing déterministe dont la complexité en espace (le nombre de cases du ruban utilisées) est bornée par un polynôme. Autrement dit la classe de tous les problèmes qui peuvent résolus en utilisant en espace mémoire polynomial en taille de leurs instances :

$$\mathcal{PSPACE} = \bigcup_{k>0} \mathcal{SPACE}n^k$$

On pose $\mathcal{L}$= $\mathcal{SPACE}(\log n)$.

DÉFINITION 3.4.3 : La classe $\mathcal{NPSPACE}$ est la classe des langages acceptés par une machine deTuring déterministe dont la complexité en espace (le nombre de cases du ruban utilisées) est bornée par un polynôme. On pose $\mathcal{NL}$= $\mathcal{NPSPACE}(\log n)$.

Il est établi que $\mathcal{PSPACE}$=$\mathcal{NPSPACE}$ cette dernière classe n'est donc en général jamais mentionnée.
Une conséquence de $\mathcal{PSPACE}$= $\mathcal{NPSPACE}$ est que $\mathcal{NP}\subseteq \mathcal{PSPACE}$ puisque l'espace utilisé par une machine de Turing est toujours inférieur au temps qu'elle utilise.
Comme pour la classe $\mathcal{NP}$, on peut définir la classe des problèmes complets dans $\mathcal{PSPACE}$. Un problème $\mathcal{PSPACE}$-complet est certainement aussi difficile, et `a priori` plus difficile, qu'un problème $\mathcal{NP}$-complet.

On cherche maintenant à étudier les "petites" classes de complexité : $\mathcal{L}$, $\mathcal{P}$ et $\mathcal{NL}$. On commence par présenter un théorème que nous ne prouverons pas mais qui est important pour comprendre la situation.
Contrairement à la classe $\mathcal{PSPACE}$ qui est égale à $\mathcal{NPSPACE}$ par le théorème de Savitch, on conjecture généralement que $\mathcal{L} \neq \mathcal{NL}$ (mais, comme d'habitude, c'est seulement une conjecture). Cependant, nous disposons d'un résultat plus faible. On peut bien sûr définir la classe $co\mathcal{NL}$: elle contient tous les langages qui sont le complément d'un langage dans $\mathcal{NL}$. Il se trouve que $\mathcal{NL} = co\mathcal{NL}$.
Le problème $\mathcal{NL}$-complet que nous allons voir se rapporte à l'accessibilité dans un graphe orienté G pour lequel on note S l'ensemble des sommets et A l'ensemble des arcs.

ACCESSIBILITÉ

ENTRÉE : un graphe $G = (V, E)$ orienté et deux sommets x et y appartenant à V.
QUESTION :Existe-t-il un chemin de x à y ?

THÉORÈME 3.4. 1 : ACCESSIBILITÉ est $\mathcal{NL}$-complet pour les réductions en espace logarithmique.

PREUVE

Algorithme 3.1 Algorithme pour ACCESSIBILITÉ sur l'entrée (G, s, t)

```
u := s ;
for i allant de 0 à |S| − 1 do
  if u = t then
    accepter ;
  else
    Choisir de manière non déterministe un voisin sortant v de u ;
    u := v ;
  end if
end for
Rejeter
```

— S'il existe un chemin de s à t , alors il existe un chemin simple de s à t (c'est-à-dire sans boucle) et celui-ci est de taille $< |S|$ (le nombre de sommets). Donc il existera une suite de choix non déterministes dans l'algorithme arrivant à t en au plus $|S| - 1$ étapes, c'est- à-dire que l'algorithme acceptera. Si par contre t n'est pas accessible depuis s, alors la boucle se terminera sans avoir trouvé t et tous les chemins rejetteront. Cet algorithme

non déterministe décide donc ACCESSIBILITÉ. Cet algorithme nécessite de conserver en mémoire le compteur i variant de 0 à $|S| - 1$: en binaire, il prend $1 + \log(|S| - 1)$ cases. Il faut également écrire les deux sommets u et v : on les désigne par leur numéro en binaire, ce qui prend encore un espace $1 + \log |S|$. Une fois v deviné, vérifier qu'il s'agit d'un voisin sortant de u se fait en parcourant l'entrée, de même que vérifier si $v = t$. Au final, cet algorithme non déterministe fonctionne en espace logarithmique, donc ACCESSIBILITÉ$\in \mathcal{NL}$.

— Pour la $\mathcal{NL}$-difficulté, nous allons réduire tout problème de $\mathcal{NL}$ à ACCESSIBILITÉ. Soit $A \in \mathcal{NL}$ et N une machine non déterministe pour A fonctionnant en espace $\leq \alpha \log n$, c'est-à-dire que $x \in A$ ssi $N(x)$ a un chemin acceptant. Pour une entrée x, on note G_x le graphe des configurations potentielles de $N(x)$ en espace $\alpha\gamma \log n$, où γ est une constante concernant le déplacement de la tête sur le ruban d'entrée. À partir de G_x on définit un nouveau graphe G'_x en ajoutant un sommet t et en reliant toutes les configurations acceptantes de G_x à t. Pour s, on prend la configuration initiale. Ainsi, $N(x)$ accepte ssi t est accessible depuis s dans G_x. Notre réduction f de A à ACCESSIBILITÉ associe (G'_x, s, t) à x. Ainsi nous obtenons $x \in A$ ssi $f(x) \in$ ACCESSIBILITÉ.

On peut montrer que G_x est calculable en espace logarithmique. Pour en déduire G'_x, il suffit de relier toutes les configurations acceptantes à t, ce qui nécessite simplement de parcourir chaque sommet et se fait en espace logarithmique également. Ainsi, f est calculable en espace déterministe logarithmique, ce qui conclut.

□

Le problème ACCESSIBILITÉ pour les graphes orientés n'est pas probablement dans la classe $\mathcal{NL}$-complet car ce problème dans la classe $\mathcal{L}$ grâce à un algorithme sophistiqué.
Considérons maintenant le problème CO-ACCESSIBILITÉdéfini de la manière suivante :

CO-ACCESSIBILITÉ

ENTRÉE : un graphe $G = (V, E)$ orienté et deux sommets x et y appartenant à V.
QUESTION :y est-il inaccessible ?

En utilisant le même procédé que pour la preuve du théorème 3.4.1 CO-ACCESSIBILITÉ est $co\mathcal{NL}$-complet.

LEMME 3.4.1 : CO-ACCESSIBILITÉ$\in \mathcal{NL}$.

PREUVE

Pour décider si t n'est pas accessible depuis s dans G, on utilise un algorithme pour compter le nombre k de sommets accessibles depuis s. Puis on énumère tous les sommets u et pour chacun on teste de manière non déterministe s'il est accessible par un chemin de taille $\leq n$. Pendant ce processus on a également compté le nombre de sommets accessibles en incrémentant un compteur c. À la fin du calcul, si $c = k$ alors on a bien visité tous les sommets accessibles donc on accepte ssi t n'a jamais été atteint depuis s.
Pour résoudre CO-ACCESSIBILITÉ, l'algorithme 3.2 (algorithme non déterministe) que l'on utilise sur l'entrée (G, s, t) :

Algorithme 3.2 Algorithme pour CO-ACCESSIBILITÉsur l'entrée (G, s, t)

$c := 0$ (compteur pour le nombre de sommets atteints) ;
$b := 0$ (booléen pour savoir si t a été atteint) ;
compter le nombre k de sommets accessibles depuis s grâce à un algorithme (rejeter si le calcul n'est pas acceptant) ;
for Tout sommet u **do**
  Tester s'il existe un chemin de s à u de taille $\leq n$ en se déplaçant dans G de manière non déterministe
  if Le chemin a atteint u **then**
    $c := c + 1$
    if $u = t$ **then**
      $b := 1$;
    end if
  end if
end for
if $c = k$ et $b = 0$ **then**
  Accepter ;
else
  Rejeter ;
end if

Un calcul ne peut être acceptant que si $c = k$, c'est-à-dire que tous les sommets accessibles depuis s ont été atteints. Il y a toujours au moins un tel calcul. Si t ne fait pas partie des sommets atteints, alors $b = 0$ et on accepte, ce qui correspond bien au cas où t n'est pas accessible depuis s. Si t a été atteint, alors il est accessible depuis s et $b = 1$ pour tous les calculs tels que $c = k$, donc tous les calculs

rejettent. Au final, cet algorithme décide bien CO-ACCESSIBILITÉ. L'espace qu'il utilise est logarithmique puisqu'il suffit d'énumérer tous les sommets, de maintenir un compteur et d'exécuter des tests requérant un espace logarithmique. Donc CO-ACCESSIBILITÉ$\in \mathcal{NL}$.

□

THÉORÈME 3.4. 2 : Les classes de complexité en espace $\mathcal{NL}$ et $co\mathcal{NL}$ sont égales.

PREUVE

— $\subseteq$ Soit A dans $\mathcal{NL}$. Comme ACCESSIBILITÉ est $\mathcal{NL}$-complet, A se réduit à ACCESSIBILITÉ en espace logarithmique.

Par le lemme 3.4.1, comme ACCESSIBILITÉ est dans $co\mathcal{NL}$, alors A est dans $co\mathcal{NL}$.

— $\supseteq$ Soit A dans $co\mathcal{NL}$. Comme ACCESSIBILITÉ est $\mathcal{NL}$-complet, A se réduit à ACCESSIBILITÉ en espace logarithmique. Par le lemme 3.4.1 comme ACCESSIBILITÉ est dans $co\mathcal{NL}$, alors $\bar{A}$ est dans $co\mathcal{NL}$. Donc A est dans $\mathcal{NL}$.

□

Chapitre

4 Autres classes de complexité

Sommaire

Résumé

Nous verrons d'autres classes de complexité un peu moins connu.

4.1 Introduction

Les problèmes d'optimisation n'ont pas été classés de manière satisfaisante dans la théorie de $\mathcal{P}$ et $\mathcal{NP}$; ce sont ces problèmes qui motivent les extensions immédiates de cette théorie au-delà de $\mathcal{NP}$.
Prenons le problème du voyageur de commerce comme exemple de travail. Dans le problème VOYAGEUR DE COMMERCE , on nous donne la matrice de distance d'un ensemble de villes ; nous voulons trouver le tour le plus proche des villes. Nous n'avons étudié la complexité du VOYAGEUR DE COMMERCE dans le cadre de $\mathcal{P}$ et $\mathcal{NP}$qu'indirectement : nous avons défini la version de décision VOYAGEUR DE COMMERCE D, et l'avons prouvée $\mathcal{NP}$-complète (faire l'épreuve). Nous introduisons deux autres variantes

EXACT VOYAGEUR DE COMMERCE

ENTRÉE : Un ensemble de m villes X, un ensemble de routes entre les villes E, une fonction de coût $v : E \to \mathbb{R}$ où $v(x, y)$ est le coût de déplacement de x à y, et $B \in$

QUESTION :Trouver une tournée de coût exactement B ?

VOYAGEUR DE COMMERCE DE COÛT

ENTRÉE : Un ensemble de m villes X, un ensemble de routes entre les villes E, une fonction de coût $v : E \to \mathbb{R}$ où $v(x, y)$ est le coût de déplacement de x à y, et $B \in \mathbb{N}^*$

QUESTION :Déterminer le coût d'une tournée optimale ?

Les quatre variantes peuvent être ordonnées selon une complexité croissante : VOYAGEUR DE COMMERCE D, EXACT VOYAGEUR DE COMMERCE, VOYAGEUR DE COMMERCE DE COÛT, VOYAGEUR DE COMMERCE .
Chaque problème de cette progression peut être réduit au suivant. Pour les trois derniers problèmes, c'est trivial ; pour les deux premiers il faut remarquer que la réduction prouvant que VOYAGEUR DE COMMERCE D est $\mathcal{NP}$-complete (voir exercice TD) peut être utilisée pour réduire CHEMIN HAMILTONIEN à EXACT VOYAGEUR DE COMMERCE (le graphe a un chemin Hamiltonien si et seulement si longueur exactement $n + 1$). Et puisque CHEMIN HAMILTONIEN est $\mathcal{NP}$-complet VOYAGEUR DE COMMERCE D est dans $\mathcal{NP}$, nous devons conclure qu'il y a une réduction de VOYAGEUR DE COMMERCE D à EXACT VOYAGEUR DE COMMERCE.
En fait, nous savons que ces quatre problèmes sont polynomialement équivalents (puisque le premier et le dernier le sont, rappelons l'exemple 10.4). Autrement dit, il existe un algorithme de temps polynomial pour un si et seulement s'il y en

a pour les quatre. Certes, du point de vue de la motivation pratique de la théorie de la complexité (à savoir, pour identifier les problèmes susceptibles de nécessiter un temps exponentiel), cette caractérisation grossière devrait suffire.
Cependant, les réductions et l'exhaustivité permettent une catégorisation des problèmes beaucoup plus raffinée et intéressante. En ce sens, de ces quatre variantes du VOYAGEUR DE COMMERCE , nous ne connaissons la complexité précise que du problème $\mathcal{NP}$-complet VOYAGEUR DE COMMERCE D. Dans cette section, nous verrons que les trois autres versions de VOYAGEUR DE COMMERCE sont complètes pour une extension très naturelle de $\mathcal{NP}$.

4.2 La classe $\mathcal{DP}$

Est-ce que EXACT VOYAGEUR DE COMMERCE appartient à $\mathcal{NP}$?. Soit une matrice de distance et nous allégeons que l'optimalité admet un coût de B, comment pouvons-nous certifier succinctement que la valeur de l'optimum est bien B ?
Le lecteur est invité à réfléchir à cette question ; aucune solution évidente ne vient à l'esprit. Ce serait tout aussi impressionnant si nous pouvions certifier que le coût optimal n'est pas de B ; en d'autres termes, EXACT VOYAGEUR DE COMMERCE ne semble même pas être dans $co\mathcal{NP}$. En fait, le résultat de cette section suggérera que si EXACT VOYAGEUR DE COMMERCE est dans $\mathcal{NP} \cup co\mathcal{NP}$, cela aurait des conséquences vraiment remarquables ; le monde de la complexité devrait être très différent de ce que l'on croit actuellement. Cependant, EXACT VOYAGEUR DE COMMERCE est étroitement lié à $\mathcal{NP}$ et $co\mathcal{NP}$ d'au moins une manière importante : considéré comme un langage, c'est l'intersection d'un langage en $\mathcal{NP}$ (le langage VOYAGEUR DE COMMERCE) et un en $co\mathcal{NP}$ (le langage CO-VOYAGEUR DE COMMERCE , demandant si le coût optimal est d'au moins B). En d'autres termes, une entrée est une instance "oui" de EXACT VOYAGEUR DE COMMERCE si et seulement si c'est une instance "oui" de VOYAGEUR DE COMMERCE , et une instance "oui" de CO-VOYAGEUR DE COMMERCE . Cela appelle une définition :

DÉFINITION 4.2.1 : Un langage L est dans la classe $\mathcal{DP}$ si et seulement si il y a deux langages $L_1 \in \mathcal{NP}$ et $L_2 \in co\mathcal{NP}$ tel que $L = L_1 \cap L_2$.

Il est important de noter que la classe $\mathcal{DP}$ n'est pas la classe $\mathcal{NP} \cap co\mathcal{NP}$. Ces deux classes sont totalement différentes.
D'une part, il est peu probable que $\mathcal{DP}$ soit contenu dans $\mathcal{NP} \cup co\mathcal{NP}$, alors par rapport à $\mathcal{NP} \cap co\mathcal{NP}$.

Une autre différence porte sur la possibilité de la complétude pour $\mathcal{DP}$. Pour cela, considérons le problème SAT-UNSAT.

SAT-UNSAT

ENTRÉE : Soient deux expressions sous-forme congestive normale avec trois littéraux par clause ϕ et ϕ'.

QUESTION :Existe-t'il une affectation telle que ϕ est satisfiable et ϕ' non satisfiable ?

THÉORÈME 4.2. 1 : SAT-UNSAT est $\mathcal{DP}$-complet.

CONSTRUCTION 4.2.1 :
Pour montrer que SAT-UNSAT appartient à la classe $\mathcal{DP}$, nous devons exhiber deux langages $L_1 \in \mathcal{NP}$ et $L_2 \in co\mathcal{NP}$ tel que l'ensemble des instances positives de SAT-UNSAT est $L_1 \cap L_2$. Ceci est facile : $L_1 = \{(\phi; \phi') : \phi$ est satisfiable$\}$ et $L_2 = \{(\phi; \phi') : \phi'$ est insatisfiable$\}$.

PREUVE

Pour monter la complétude, soit L un langage de $\mathcal{DP}$. Nous devons montrer que L se réduit à SAT-UNSAT. Ce que nous savons du langage L c'est qu'il y a deux langages $L_1 \in \mathcal{NP}$ et $L_2 \in co\mathcal{NP}$ tel que $L = L_1 \cap L_2$.
Sachant que SATISFAISABILITÉest $\mathcal{NP}$-complet, nous savons qu'il existe une réduction R_1 de L_1 à SATISFAISABILITÉ, et une réduction R_2 du complémentaire de L_2 à SATISFAISABILITÉ.
La réduction de L à SAT-UNSAT est donnée avec pour entrée x :

$$R(x) = (R_1(x), R_2(x)).$$

Nous avons que $R(x)$ est une instance "oui" de SAT-UNSAT si et seulement si $x \in L_1$ et $x \in L_2$, ou de manière équivalente $x \in L$.

□

THÉORÈME 4.2. 2 : EXACT VOYAGEUR DE COMMERCE est $\mathcal{DP}$-complete.

CONSTRUCTION 4.2.2 : Nous montrons que qu'il existe une réduction de SAT-UNSAT à EXACT VOYAGEUR DE COMMERCE. Soit (ϕ, ϕ') de SAT-UNSAT, nous appliquons aux deux expressions ϕ et ϕ' la réduction à CHEMIN HAMILTONIENpour obtenir deux graphes G et G' respectivement, ayant pour le deux un HP cassé avec en garantie.
Nous combinons ensuite les deux graphes dans un cycle en identifiant le nœud 2 de G avec le nœud 1 de G', et vice-versa (voir figure 4.1). Soit n le nombre de nœuds dans le nouveau graphe.
Nous définissons ensuite les distances entre les nœuds du graphe combiné pour obtenir une instance du VOYAGEUR DE COMMERCE . La distance entre les nœuds i et j est définie comme suit : si $[i, j]$ est une arête du graphe G ou du graphe G', alors la distance est un. Si $[i, j]$ n'est pas une arête, mais que i et j sont tous deux des nœuds du graphe G' alors sa distance est de deux ; tous les autres non-bords ont une distance de 4.

PREUVE

La preuve est basée sur la construction 4.2.2. Nous avons déjà argumenter que EXACT VOYAGEUR DE COMMERCE appartient à $\mathcal{DP}$. Pour prouve la complétude, nous allons procéder à la réduction entre SAT-UNSAT à EXACT VOYAGEUR DE COMMERCE.
Soit (ϕ, ϕ') une instance de SAT-UNSAT.
Nous utilisons la construction 2.6.4, celle donnée entre 3-SATISFAISABILITÉ à CIRCUIT HAMILTONIEN (voir théorème 2.6.5) pour produire à partir de (ϕ, ϕ') deux graphes (G, G'), possédant chaque un chemin Hamiltonien si et seulement si l'expression correspondante est satisfiable. Cependant notre construction ajoute une nouvelle caractéristique essentielle : les graphes G et G' possèdent toujours a chemin Hamiltonien cassé, qui deux chemins sommets-disjoints couvrant tous les sommets.
Une dernière modification concernant l'expression tel que chaque expression possède un affectation presque valide, qui est une affectation qui satisfait toutes les classes sauf une. Ceci est facile à obtenir, il suffit d'ajouter une nouveau littéral appelé z à toutes les clauses et ajoutons la clause $(\neg z)$. Ainsi, en affectation la valeur vraie à toutes les variables et nous satisfaisons toutes es clauses sauf une, la nouvelle. Maintenant nous transformons les clauses en des clauses de taille trois en remplaçant la clause $(x_1 \vee x_2 \vee x_3 \vee z)$ par deux clauses $(x_1 \vee x_2 \vee w)$ et $(\neg w \vee x_3 \vee z)$.
Si nous effectuons maintenant la réduction du théorème à partir d'un ensemble de clauses qui a une telle assignation de vérité presque satisfaisante, appelons-le T, il est facile de voir que le graphe résultant a toujours un chemin Hamiltonien

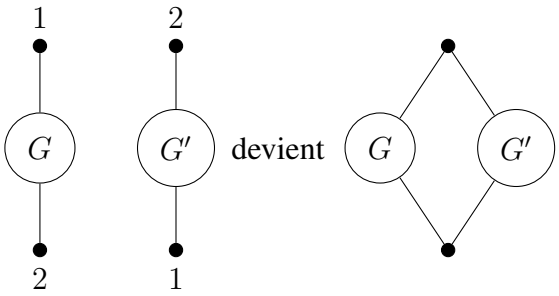


FIGURE 4.1 – Illustration de la Construction 4.2.2.

cassé : il commence au nœud 1, il parcourt toutes les variables selon T, et continue aux clauses sauf celle qui peut être insatisfaite, où le chemin est rompu une fois (vous pouvez examiner le "gadget de contrainte" de la figure 9.6 pour vérifier qu'il provoque au plus une telle rupture). Le chemin continue ensuite normalement jusqu'au nœud 2.
C'est fondamental le côté HC cassé : quand ϕ est non satisfiable, alors on peut garantir deux chemins disjoints qui couvrent. Dans ce cas contraire on ne peut garantir qu'il ya deux chemins.
Quelle est la durée de la visite la plus courte de cette instance de VOYAGEUR DE COMMERCE . Évidemment, cela dépend du fait que ϕ et ϕ' sont satisfaisables ou non. S'ils sont tous les deux satisfaisables, alors le coût optimal en n, le nombre de nœuds dans le graphe combiné (il y a un cycle hamiltonien dans le graphe combiné). S'ils ne sont pas satisfaits tous les deux, alors le coût optimal est $n + 3$ (le tour optimal combine les deux chemins hamiltoniens cassés, et donc un non-bord de G et un non-bord de G' auront à utiliser). Si ϕ est satisfiable et ϕ' ne l'est pas, alors le coût optimal est $n + 2$ (il faudra utiliser un bord non de G', mais pas de G. Et si ϕ est insatisfiable et ϕ' est satisfiable, alors le coût optimal est $n + 1$. Il s'ensuit que (ϕ, ϕ') est une instance oui de SAT-UNSAT si et seulement si le coût optimal est $n + 2$. Prendre B égal à ce nombre complète notre réduction de SAT-UNSAT à EXACT VOYAGEUR DE COMMERCE.

□

Mais $\mathcal{DP}$ est plus riche que cela. Par exemple ne relation avec SAT-UNSAT, D'autres problèmes appartiennent à la classe $\mathcal{DP}$:

— CRITICAL-SAT

CRITICAL-SAT

ENTRÉE : Soit une expression sous-forme conjonctive normale ϕ.

QUESTION :Existe-t'il une affectation telle que ϕ est insatisfiable et la suppression de n'importe clause rend ϕ satisfiable

— UNIQUE-SAT

UNIQUE-SAT

ENTRÉE : Soit une expression sous-forme conjonctive normale ϕ.

QUESTION :Existe-t'il une unique affectation telle que ϕ soit satisfiable.

— VOYAGEUR DE COMMERCE CRITIQUE

VOYAGEUR DE COMMERCE CRITIQUE

ENTRÉE : Soit $G = (V, E)$

QUESTION :Existe-t'il aucun chemin Hamiltonien, mais l'ajout de n'importe quelle arête créée un chemin Hamiltonien.

— CRITICAL-3 COULEURS

CRITICAL-3 COULEURS

ENTRÉE : Soit $G = (V, E)$

QUESTION :Existe-t'il aucune 3-coloration, mais la suppression de n'importe quel sommet rend le graphe 3-coloriable ?

Remarquons que les trois problèmes dits critiques appartiennent également à $\mathcal{DP}$-complet. D'autre part UNIQUE-SAT, et aussi d'autres problèmes traitant du problème : est-ce qu'une instance possède une unique solution, ne possède pas encore de classification dans des classes plus faibles. Il est difficile de croire que qu'ils appartiennent à $\mathcal{DP}$-complet.

4.3 La classe $\mathcal{PLS}$ pour la recherche locale

4.3.1 Optimum local

Enfin, les *algorithmes de recherche locale* sont des algorithmes permettant de trouver un optimum local, qui là aussi n'est pas nécessairement optimale. L'ap-

partenance pour un problème de recherche locale à une classe de complexité dépendra du nombre de solutions explorées avant d'obtenir un optimum local.
Le dernier type de problèmes que nous aborderons est celui des problèmes de recherche locale. Un problème de recherche locale est défini comme suit.

DÉFINITION 4.3.1 : Un problème de recherche locale est un 2-uplet $(\Pi, \mathcal{N})$ où

— Π est un problème d'optimisation ;

— $\mathcal{N} : \Sigma_\Pi \times sol \mapsto \mathcal{P}(sol)$ est une application, qui étant donné une instance $I \in \Sigma_\Pi$ et une solution $x \in sol(I)$, construit un ensemble de solutions inclus dans $sol(I)$.

On préférera noter $\Pi/\mathcal{N}$ le problème de recherche locale $(\Pi, \mathcal{N})$. L'application $\mathcal{N}$ est appelée *fonction de voisinage*. Pour une instance I et une solution x, on appelle *voisinage* de x l'ensemble $\mathcal{N}(I, x)$ et une solution $x' \in \mathcal{N}(I, x)$ est une *solution voisine* de x. Notons qu'une solution x' peut être voisine d'une solution x sans que x ne soit une solution voisine de x'. Une solution est un *optimum local* si elle est meilleure que toute solution de son voisinage. On pourra préciser en qualifiant cette solution de *minimum local* si Π est un problème de minimisation ou de *maximum local* si Π est un problème de maximisation. À titre d'exemple, on peut exprimer une fonction de voisinage pour les problèmes de SATISFAISABILITÉ, appelée *Flip*.

DÉFINITION 4.3.2 : flip Soit φ une instance SATISFAISABILITÉet une assignation β des variables booléennes de φ. Une assignation β' voisine de β selon la fonction de voisinage Flip est construite en changeant la valeur d'exactement une variable booléenne.

Considérons l'algorithme standard de recherche locale défini par l'algorithme 4.1.

Algorithme 4.1 Algorithme de recherche local standard.

$x' :=$ la meilleure solution dans $\mathcal{N}(I, x) \cup \{x\}$;
if $x \neq x'$ **then**
  RETOURNER $recherche\_locale(\Pi/\mathcal{N}, I, x')$;
end if
RETOURNER x ;

Cet algorithme permet de trouver un optimum local en partant d'une solution initiale. Si la solution x donnée en entrée est un optimum local, alors elle est retournée. Sinon, on prend la meilleure solution parmi son voisinage et on fait un appel récursif sur celle-ci. Pour un algorithme de recherche local, on va mesurer le

nombre d'appels récursifs à effectuer dans le pire des cas. D'un point de vue applicatif, la recherche locale peut être utile pour améliorer une solution construite à l'aide d'un algorithme d'approximation. De ce point de vue, il n'est pas forcément nécessaire de minimiser le nombre d'appels récursifs avant de trouver un optimum local. En effet, si l'on trouve un optimum local trop rapidement, cela veut dire que la solution initiale n'a pas été grandement améliorée.
À titre d'exemple, faisons une recherche locale sur l'instance de MAXIMUM 2-SATISFAISABILITÉ PONDÉRÉ suivante :

$$\underbrace{(\neg x_1 \vee \neg x_1)}_{\mathrm{w}(C_1)=2} \wedge \underbrace{(\neg x_2 \vee \neg x_2)}_{\mathrm{w}(C_2)=2} \wedge \underbrace{(\neg x_3 \vee \neg x_3)}_{\mathrm{w}(C_3)=3} \wedge \underbrace{(x_1 \vee x_3)}_{\mathrm{w}(C_4)=3} \wedge \underbrace{(x_2 \vee x_3)}_{\mathrm{w}(C_5)=3} \wedge \underbrace{(\neg x_4 \vee \neg x_4)}_{\mathrm{w}(C_6)=1}$$

Pour décrire une solution, on peut utiliser un 4-uplet dans $\{1,0\}^4$ où pour chaque variable booléenne x_i, la position i du 4-uplet contient un 1 si l'assignation de la variable booléenne est égale à VRAI et un 0 sinon. On considère une solution initiale $S_{init} = (1,1,1,1)$ où toutes les variables booléennes sont assignées à VRAI. Cette solution S_{init} a un poids égal à six. Déroulons les étapes de l'algorithme de recherche locale.

1. Les solutions voisines de S_{init} sont $(0,1,1,1), (1,0,1,1), (1,1,0,1)$ et $(1,1,1,0)$. La meilleure solution parmi ces solutions voisines est $S_1 = (1,1,0,1)$ qui a un poids de neuf.
2. Les solutions voisines de S_1 sont $(0,1,0,1)$, $(1,0,0,1)$, $(1,1,1,1)$ et $(1,1,0,0)$. La meilleure solution parmi ces solutions voisines est $S_2 = (1,1,0,0)$ qui a un poids de dix.
3. Les solutions voisines de S_2 sont $(0,1,0,0)$, $(1,0,0,0)$, $(1,1,1,0)$ et $(1,1,0,1)$. Parmi ces solutions voisines, aucune n'a un poids plus grand que celui de S_2. On a donc trouvé un minimum local.

Ainsi, en trois appels récursifs de la fonction de recherche locale, on a trouvé un optimum local. Notons que ce n'est cependant pas une solution optimale pour cette instance : en effet la meilleure solution est $(0,0,1,0)$ qui a un poids de onze.

4.3.2 La classe $\mathcal{PLS}$

La classe de complexité $\mathcal{PLS}$ (pour *polynomial local search* en anglais) contient des problèmes de recherche locale et est définie de la façon suivante.

DÉFINITION 4.3.3 : Un problème de recherche locale $\Pi/\mathcal{N}$ appartient à la classe $\mathcal{PLS}$, s'il existe trois applications polynomiales A_L, B_L et C_L telles que :

- pour une instance $I \in \Sigma_\Pi$, l'application $A_L : \Sigma_\Pi \mapsto sol(I)$ donne une solution initiale pour I ;
- pour toute solution $x \in sol(I)$, l'application $B_L : \Sigma_\Pi \times sol(I) \mapsto \mathbb{R}$ calcule le score de x ;
- pour toute solution x, l'application $C_L : \Sigma_\Pi \times sol(I) \mapsto sol(I)$ détermine si S un optimum local et, si ce n'est pas le cas, retourne la solution avec le meilleur score dans son voisinage.

Dit de façon un peu informelle, les problèmes de recherche locale dans $\mathcal{PLS}$sont ceux où l'on peut construire une première solution initiale en temps polynomial et tels que leurs fonctions de voisinage construisent des voisinages de taille polynomiale.
Par exemple, considérons le problème de recherche local MAXIMUM 2-SATISFAISABILITÉ PONDÉRÉ/Flip. On peut construire une fonction donnant une solution initiale en donnant à toutes les variables la valeur VRAI (ou même n'importe quelle valeur au hasard). Pour calculer le score d'une assignation, il suffit de parcourir chacune des clauses et des variables booléennes et de faire les sommes des poids des clauses satisfaites, ce qui se fait en temps polynomial. Enfin, une assignation a exactement autant de solutions voisines que de variables booléennes présentes dans la formule, on peut donc clairement parcourir tout le voisinage en temps polynomial pour déterminer la meilleure solution parmi les voisins. On peut donc construire trois applications polynomiales A_L, B_L et C_L comme décrites dans la définition 4.3.3. Et donc MAXIMUM 2-SATISFAISABILITÉ PONDÉRÉ/Flip appartient à la classe $\mathcal{PLS}$.

4.3.3 Complétude et difficulté de la classe $\mathcal{PLS}$

Pour cette classe $\mathcal{PLS}$, l'ensemble des problèmes complets va contenir les problèmes de recherche locale où calculer un optimum local est le plus difficile. On peut voir ces problèmes comme ceux qui nécessitent le plus d'appels récursifs de la fonction de recherche locale avant d'obtenir un optimum local en utilisant l'algorithme de recherche locale standard. La réduction utilisée est la $\mathcal{PLS}$-réduction, définie comme suit.

DÉFINITION 4.3.4 : Soient $\Pi_1/\mathcal{N}_1$ et $\Pi_2/\mathcal{N}_2$ deux problèmes de recherche locale. $\Pi_1/\mathcal{N}_1$ est $\mathcal{PLS}$-réductible à $\Pi_2/\mathcal{N}_2$ s'il existe une application $f : \Sigma_{\Pi_1} \mapsto \Sigma_{\Pi_2}$ telle que pour toute instance $I \in \Sigma_{\Pi_1}$:

— $f(I)$ est calculable en temps polynomial ;

— il existe une application $g : sol(f(I)) \mapsto sol(I)$ telle que pour toute solution $x \in sol(f(I))$ telle que x est un optimum local, $g(x)$ est un optimum local.

Historiquement, le premier problème $\mathcal{PLS}$-complet apparaît dans un article de Johnson, Papadimitriou et Yannakakis. Il s'agit du problème de trouver une assignation des variables dans un circuit booléen [3]. Le problème MAXIMUM 2-SATISFAISABILITÉ PONDÉRÉ/Flip a également été montré $\mathcal{PLS}$-complet [4].

4.3.4 Preuves

Le premier problème difficile appartenant à $\mathcal{PLS}$-complet et le problème noté CIRCUIT/FLIP.

CIRCUIT/FLIP

ENTRÉE : Un circuit acyclique de booléens avec en entrée $x_1, x_2, \ldots, x_n$ et $y_1,\ y_2, \ldots, y_m$

TÂCHE : Une assignation de $x_1, x_2, \ldots, x_n$.

Le coût de la solution donnée par l'affectation $x_1, x_2, \ldots, x_n$ est la sortie $y_1,\ y_2, \ldots, y_m$ lu comme un nombre entier

$$c(x_1, \ldots, x_n) = \sum_{i=1}^{m} 2^{(i-1)}(y_i)$$

Le voisinage d'une solution $s = x_1, \ldots, x_n$ peut-être obtenu par le flip (par la négation) d'un x_i au hasard (changement de bit). Alors s et toutes les voisins $r \in \mathcal{N}(I, s)$ possède une distance de Hamming de un, $H(s, r) = 1$.

EXEMPLE

Considérons la figure 4.2 pour illustrer le problème avec la formule suivante :

$$\begin{pmatrix} x_1 \vee x_2 \\ (x_1 \vee x_2) \wedge \neg x_3 \end{pmatrix}$$

LEMME 4.3.1 : MAX CIRCUIT/FLIP est équivalent à MIN CIRCUIT/FLIP.

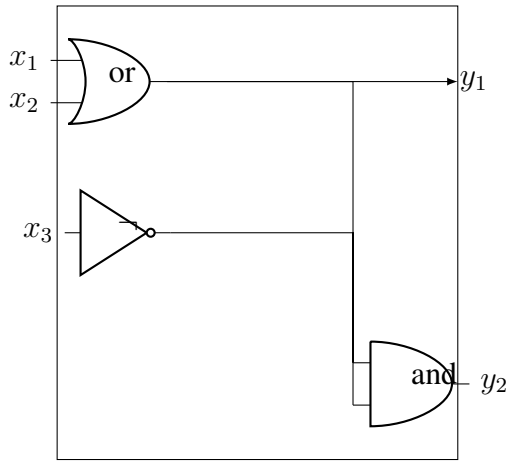


FIGURE 4.2 – Trois arbres couvrants.

PREUVE
MAX CIRCUIT/FLIP et MIN CIRCUIT/FLIP peuvent être réduits l'un à l'autre. Ceci peut être réalisé en ajoutant une couche supplémentaire de logique au circuit qui annule chaque sortie. Soit g l'identité et définissons f pour transformer les instances en ajoutant une porte négative devant chaque variable de sortie y_i, cela peut être fait en temps polynomial. De toute évidence, les optima locaux et globaux sont les mêmes.

□

THÉORÈME 4.3. 1 : MAX CIRCUIT/FLIP et MIN CIRCUIT/FLIP sont $\mathcal{PLS}$-complets.

PREUVE
La preuve est donnée dans [3]. □

POS-NAE-kSAT

ENTRÉE : Une formule logique avec n variables logiques $x_1, x_2, \ldots, x_n$ et m clauses $c_1, c_2, \ldots, c_m$ chacune contenant k littéraux positifs et m poids $w_1, w_2, \ldots, w_m$ associés à chacune des m clauses.

TÂCHE : Maximiser la valeur de l'affectation

On définit la valeur d'une affectation comme la somme pondérée de clauses satisfaites.

4.3.5 D'autres problèmes dans $\mathcal{PLS}$-complets

THÉORÈME 4.3. 2 : POS-NAE-3SAT$\leq_{PLS}$ POS-NAE-2SAT.

PREUVE

CONSTRUCTION 4.3.1 : Pour chaque clause $c_i = (x_1, x_2, x_3)$ de poids w POS-NAE-3SAT, l'algorithme A introduit les clauses $(x_1, x_2), (x_2, x_3)$ et (x_1, x_3) chacune de poids $w/2$.

La valeur d'une affectation dans l'instance POS-NAE-2SATest identique à sa valeur dans l'instance POS-NAE-3SATétant donné que l'instance de POS-NAE-3SATest satisfaite si et seulement si deux de trois clauses de l'instance POS-NAE-2SATsont satisfaites. Ainsi, les optimaux locaux de deux instances coïncident (l'algorithme B est trivial).

□

Bibliographie

[1] Giorgio Ausiello, Alberto Marchetti-Spaccamela, Pierluigi Crescenzi, Giorgio Gambosi, Marco Protasi, and Viggo Kann. *Complexity and approximation : combinatorial optimization problems and their approximability properties*. Springer, 1999.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[3] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search ? *J. Comput. Syst. Sci.*, 37(1) :79–100, 1988.

[4] Mark W. Krentel. On finding and verifying locally optimal solutions. *SIAM J. Comput.*, 19(4) :742–749, 1990.

[5] Vangelis Th. Paschos. *Complexité et approximation polynomiale*. Hermès, 2004.