# Complexité

# II

## 1 Les Classes de Complexité $\mathcal{P}$ et $\mathcal{NP}$

Les quatre problèmes présentés en introduction à la section 1 sont de natures diverses. Leurs résolutions semblent plus on moins "évidentes" ou "rapides". Cette notion de difficulté de résolution a très tôt éveillé l'intérêt des chercheurs. La notion de complexité a été introduite pour qualifier la nature plus ou moins ardue des problèmes qui admettent une solution sous forme de procédure effective.

La classe de complexité qui nous intéresse *a priori*, $\mathcal{P}$, correspond aux problèmes pouvant être résolus en temps polynomial. Si tous les algorithmes polynomiaux ne sont pas efficaces (un algorithme de complexité $n^{100}$ est inutilisable en pratique), un algorithme exponentiel, de complexité $2^n$, est clairement inefficace. Malheureusement une très large proportion de problèmes "intéressants" ont peu d'espoir d'admettre un algorithme de résolution polynomial. La plupart de ces problèmes appartiennnent à une classe de complexité plus vaste, $\mathcal{NP}$. La différence entre $\mathcal{P}$ et $\mathcal{NP}$ est que pour un problème de $\mathcal{P}$ il est possible de *trouver* en temps polynomial la réponse à toute instance, tandis que pour un problème de $\mathcal{NP}$ il est possible de *vérifier* en temps polynomial qu'une réponse est correcte.

La théorie de la complexité se définit sur les problèmes de décision, et cherche à déterminer le plus petit temps d'exécution nécessaire à un algorithme pour décider d'un problème. La modélisation adoptée au chapitre précédent de la notion de résolution de problème et de l'exécution d'un algorithme conduit à reformuler formellement notre objet d'étude comme la détermination du plus petit temps de calcul nécessaire à une machine de Türing pour décider un langage. Rappelons que le passage des problèmes de décision à la reconnaissance de langages se fait par le choix d'un codage naturel des instances. La complexité d'un problème ne sera ainsi définie qu'à un polynome près, dépendant du choix du codage naturel.

Les complexités en temps et en espace (mémoire) sont ainsi définies par rapport au modèle de la machine de Türing comme le nombre de transitions et le nombre de cases mémoires utilisées. Nous exprimons la complexité en fonction de la taille de l'instance à traiter, c'est-à-dire en fonction de la taille $|w|$ du mot en entrée. Nous dirons ainsi que la machine de Türing déterministe $T$ décide un langage $L$ en temps $f(n)$ si pour tout mot d'entrée $w$ sur $\Sigma^*$, la machine $T$ accepte (ssi $w \in L$) ou rejete (ssi $w \notin L$) après au plus $f(|w|)$ transitions.

Il est important de remarquer que la complexité d'un algorithme, et donc d'un problème, est définie comme une complexité dans le pire des cas, c'est-à-dire par rapport à l'entrée la plus défavorable pour l'algorithme. On peut alors définir la classe $TIME(f(n))$ comme l'ensemble des langages pouvant être décidés en temps $\mathcal{O}(f(n))$ par une machine de Türing déterministe.

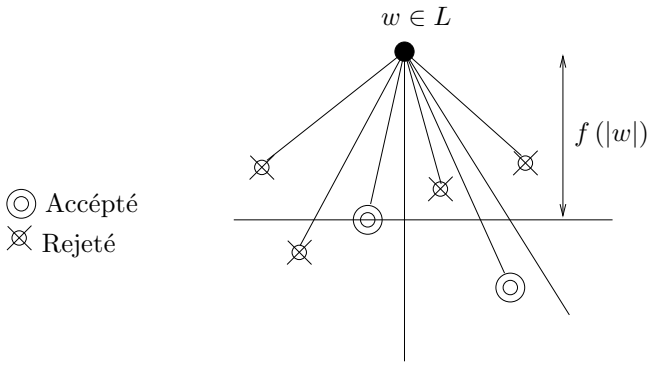
**Définition 6** *La classe $\mathcal{P}$ est l'ensemble des langages L pouvant être décidés par une machine de Türing déterministe polynomiale.*

$$\mathcal{P} = \bigcup_k TIME(n^k)$$

On dira par extension qu'un problème de décision $\Pi$ appartient à $\mathcal{P}$ si pour une fonction d'encodage naturelle, le langage $L_\Pi \in \mathcal{P}$. Par exemple, le problème Reachability décidant de l'existence d'un chemin entre 2 sommets d'un graphe est polynomial (en fait linéaire).

La classe $\mathcal{NP}$ est définie par rapport aux machines de Türing non déterministes (NTM). Rappelons qu'une NTM accepte un mot simplement si il existe une exécution acceptant ce mot. Ainsi une NTM $T$ *reconnait* le langage $L$ en temps $f(n)$ si :

- Pour tout mot $w \in L$, il existe une exécution de $T$ acceptant $w$ en au plus $f(|w|)$ transitions.
- Pour tout mot $w \notin L$, aucune exécution de $T$ (quelle que soit sa longueur) ne conduit à un état accepteur.




Reconnaître un langage en temps polynomial n'implique pas à première vue que l'on puisse le décider en temps polynomial. $\mathcal{NP}$ est ainsi l'*alter ego* de $\mathcal{P}$, marquant la différence *reconnaître* versus *décider*. Nous définissons la classe $NTIME(f(n))$ comme l'ensemble des langages pouvant être *reconnus* en temps $\mathcal{O}(f(n))$ par une machine de Türing non déterministe.

**Définition 7** *La classe $\mathcal{NP}$ est l'ensemble des langages L pouvant être reconnus*

*par une machine de Türing non déterministe polynomiale.*

$$\mathcal{NP} = \bigcup_k NTIME(n^k)$$

Il est clair que nous avons l'inclusion $\mathcal{P} \subseteq \mathcal{NP}$. En effet, la machine de Türing déterministe est un cas particulier de machine de Türing non déterministe. Le problème ouvert le plus célèbre de la complexité est de savoir si $\mathcal{P} = \mathcal{NP}$ ou si cette inclusion est stricte, $\mathcal{P} \neq \mathcal{NP}$.

Attention, soulignons que la complexité d'un problème est définie par rapport à un codage naturel de ses instances. Considérons par exemple le problème du test de primalité d'un nombre :

PRIME
INSTANCE : un entier $N$
QUESTION : $N$ est-il premier ?

Il existe un algorithme connu pour répondre à cette question en temps $O(\sqrt{N})$. Mais un codage naturel de PRIME consiste à représenter $N$ en binaire (ou décimal). Le mot d'entrée étant ainsi codé sur $\log N$ bits, la complexité de l'algorithme est en fait exponentielle. Ce problème est un exemple de problème dont il n'est pas facile simplement de prouver qu'il est dans $\mathcal{NP}$.

Une autre caractérisation possible de $\mathcal{NP}$ consiste, plutôt qu'à exhiber un algorithme polynomial non déterministe reconnaissant un langage, à exhiber un certificat de positivité *concis* (polynomial). C'est-à-dire que pour tout mot du langage, il existe un preuve d'appartenance vérifiable en temps polynomial.

**Définition 8** *$\mathcal{NP}$ est l'ensemble des langages $L$ tels que pour tout mot $w \in L$, il existe une preuve $\pi_w$ d'appartenance de $w$ à $L$ vérifiable en temps (déterministe) polynomiale en $|w|$.*

Ainsi pour le problème HAMILTONIANCIRCUIT décidant l'existence d'un cycle Hamiltonien dans un graphe :

HAMILTONIANCIRCUIT (HC)
INSTANCE : Un graphe $G = (V, E)$
QUESTION : Existe-t-il un cycle Hamiltonien, c'est-à-dire un cycle passant une fois et une seule par chaque sommet.

Le problème HC appartient à $\mathcal{NP}$. Considérons comme codage naturel la matrice d'adjacence du graphe. Un certificat de positivité consiste à donner une permutation des sommets. Ce certificat est de taille $\mathcal{O}(n \log n)$. On peut vérifier en temps $\mathcal{O}(n^2)$ que la permutation correspond à un cycle du graphe. Ce certificat est bien polynomial en la taille de l'instance $\mathcal{O}(n^2)$.

# 2 Réduction Polynomiale

Le principe de l'étude de la complexité est de classifier les problèmes par rapport au critère de temps d'exécution sur une machine de Türing. Les deux classes $\mathcal{P}$ et $\mathcal{NP}$ que nous avons définies ne semblent pas assez fines pour discriminer la difficulté des problèmes. Nous aimerions introduire une relation d'ordre sur les langages, signifiant qu'un langage est plus facile à décider qu'un autre. Cette relation d'ordre est définie par la *réduction polynômiale*.

**Définition 9** *Soient $L_1$ et $L_2$ deux langages sur un alphabet $\Sigma$. Une fonction $\tau$ de $\Sigma^*$ vers $\Sigma^*$ est une réduction de $L_1$ vers $L_2$ si et seulement si*

$$\forall x \in \Sigma^*, \quad x \in L_1 \;\Leftrightarrow\; \tau(x) \in L_2$$

Si la transformation $\tau$ est polynômiale, on dit que la réduction est polynômiale. Cette réduction est également appelée *réduction de Karp*, nous verrons plus loin qu'il existe d'autres types de réduction.

Du point de vue des problèmes, $\tau$ est une réduction polynomiale du problème $\Pi$ au problème $\Pi'$ si elle est une réduction polynomiale entre les langages correspondants. La réduction $\tau$ transforme toutes les instances positives $\Pi$ en instances positives de $\Pi'$, et toutes instances négatives de $\Pi$ en instances négatives de $\Pi'$. L'existence d'une réduction polynômiale de $\Pi$ vers $\Pi'$ montre que $\Pi'$ est au moins aussi difficile que $\Pi$. En effet si $\Pi$ peut être résolu en temps polynomial, alors $\Pi'$ peut l'être aussi ; si par contre $\Pi$ requiert un temps exponentiel, alors $\Pi'$ ne peut être résolu par un algorithme polynomial. Notons bien que le sens premier de la réduction de $\Pi$ vers $\Pi'$ est encore plus fort : une réduction prouve qu'à une transformation polynomiale près des instances, c'est-à-dire à un codage naturel près de $\Pi$, le problème $\Pi$ est simplement un sous-problème de $\Pi'$.

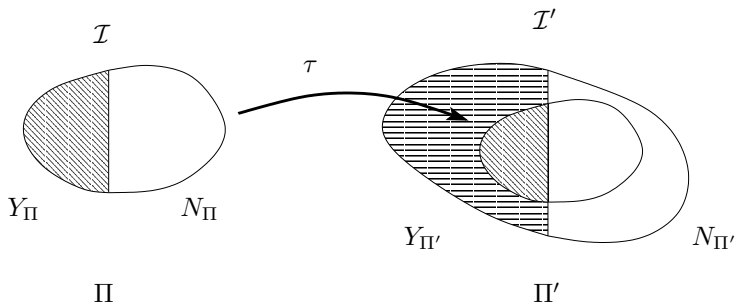



FIG. 7 – Réduction entre problèmes

D'un point de vue algorithmique, une réduction est un "pré-processing" des instances du problème $\Pi$, qui permet d'utiliser tout algorithme polynomial de

résolution pour $\Pi'$ pour résoudre $\Pi$ en temps polynomial.

On notera $\Pi \propto \Pi'$ si il existe une réduction polynomiale de $\Pi$ vers $\Pi'$. Nous dirons que $\Pi$ se réduit à $\Pi'$.

**Propriété 1** *La réduction polynômiale est un pré-ordre sur les langages : c'est une relation réflexive et transitive.*

Cette propriété n'est pas difficile à montrer. Elle est importante car elle permettra d'établir des réductions à partir de problèmes de référence. On notera par $\equiv$ l'équivalence associée : $\Pi \equiv \Pi'$ si et seulement si $\Pi \propto \Pi'$ et $\Pi' \propto \Pi$.

## 2.1 Un exemple simple de réduction

Détaillons un exemple de réduction polynomiale du problème de décision qui cherche à déterminer si un graphe possède une chaîne hamiltonnienne à partir du problème du cycle hamiltonien.

HamiltonianPath (HP)
Instance : un graphe $G = (V, E)$
Question : est-ce que le graphe possède une chaîne hamiltonienne (chaîne qui passe une fois et une seule par tous les sommets du graphe) ?

Pour montrer que nous avons la réduction HP $\propto$ HC, considèrons une instance de HP, c'est-à-dire un graphe $G$. À partir de $G$ nous construisons une instance particulière $\tau(G)$ de $HC$ en ajoutant à $G$ un nouveau sommet $\nu$ relié à tous les autres. La transformation $\tau$ est évidemment polynomiale. Montrons que c'est une réduction, c'est-à-dire que $G$ admet une chaîne hamiltonienne si et seulement si $\tau(G)$ possède un cycle hamiltonien.

Si $G$ possède une chaîne hamiltonienne $\varphi$, alors le cycle $\nu\varphi\nu$ est hamiltonien dans $\tau(G)$. Réciproquement, si $\tau(G)$ admet un cycle hamiltonien, son sous-graphe privé de $\nu$, $G$, possède une chaîne hamiltonienne.

On peut également établir que nous avons la réduction du cycle hamitonien au circuit hamiltonienne, $HC \propto HP$. Ce résultat n'est pas aussi immédiat, même s'il semble facile de déduire une chaîne hamiltonienne à partir d'un cycle hamiltonien, cela ne suffit pas à définir une réduction. Considérons la transformation $\phi$ suivante d'une instance $G$ de HC vers une instance $\phi(G)$ de HP :

On duplique un sommet quelconque $x$ du graphe $G$ en $x'$, puis on relie $x$ et $x'$ respectivement à deux nouveaux sommets $y$ et $y'$ comme c'est indiqué dans la figure 8. Cette transformation est polynomiale. C'est une réduction : $G$ admet un cycle hamiltonien si et seulement si $\phi(G)$ possède une chaîne hamiltonienne.

En effet, si $G$ possède un cycle hamiltonien, la chaîne formée de l'arête $y$ à $x$, de la partie du cycle jusqu'à un voisin de $x$, puis des arêtes de ce voisin à $x'$ et celle de $x'$ à $y'$ est une chaîne hamiltonienne de $\phi(G)$. Réciproquement, s'il existe une chaîne hamiltonienne $\varphi$ dans $\phi(G)$, elle est nécessairement de la

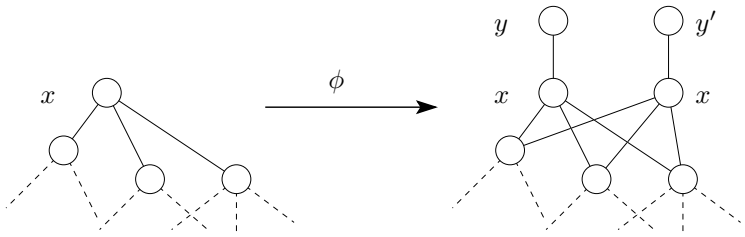



FIG. 8 – Réduction $HC \propto HP$.

forme $yx\psi x'y'$. Alors $x\psi x$ est un cycle hamiltonien sur $G$.

Nous venons de montrer que les problèmes $HC$ et $HP$ sont équivalents : nous pouvons transformer tout algorithme polynomial pour $HC$ en un algorithme polynomial pour $HP$, et inversement.

## 2.2 Exemple : Réductions de problèmes particuliers

Considérons deux problèmes voisins, les versions de décision des problèmes du voyageur de commerce (D-TSP) et de l'existence d'un circuit hamiltonnien dans un graphe (HC que nous avons déjà présenté).

D-TSP

INSTANCE : un ensemble $V$ de villes, la matrice des distances inter-villes $(d_{i,j})$ et une constante $k$.

QUESTION : Déterminer s'il existe un parcours fermé, passant par toutes les villes, de longueur inférieure à $k$.

Il est facile de démontrer que ces deux problèmes admettent un algorithme de résolution exponentiel en considérant toutes les permutations possibles (en comparant la somme de la longueur de leurs tours pour D-TSP). A ce jour, il n'existe pas d'algorithmes polynomiaux connus pour résoudre ces problèmes, et nombreux sont les informaticiens qui pensent qu'il n'en existe sans doute pas... On peut réduire HC vers D-TSP (HC $\alpha$ D-TSP).

Décrivons un algorithme polynomial qui transforme une instance quelconque de HC en une instance positive de D-TSP si et seulement si l'instance de HC est positive.

L'instance est la suivante : l'ensemble des villes correspond aux sommets du graphe $G$, les distances sont données par $d_{i,j} = 1$ si $i$ et $j$ sont reliés dans $G$, 2 sinon. La constante $k$ est égale au nombre de villes.

Cette transformation est polynomiale de manière évidente. De plus, une instance positive quelconque de HC fournit un cycle hamiltonien dans $G$. Ce cycle correspond à un parcours de longueur $n$ des $n$ villes exactement une fois chacune. Elle est positive pour D-TSP. Inversement, la réduction transforme les

instances positives de D-TSP en instances positives de HC. En effet, la solution de D-TSP possède par définition des arêtes de coût unitaire puisque la longueur totale est $k$, c'est donc aussi une solution de HC.

# 3 La classe $\mathcal{NP}$-complet

La réduction polynomiale permet de construire des classes d'équivalence. Établir qu'un langage $L$ appartient à $\mathcal{NP}$-complet consiste à montrer que $L$ est "le" langage le plus difficile de $\mathcal{NP}$, au sens où il est un élément maximum de $\mathcal{NP}$ pour la réduction polynomiale.

**Définition 10** *Un langage $L$ appartient à $\mathcal{NP}$-complet ssi il appartient à $\mathcal{NP}$ et si tout langage de $\mathcal{NP}$ se réduit polynomialement à lui :*

$$L \in \mathcal{NP} \quad \text{et} \quad \forall L' \in \mathcal{NP}, \quad L' \propto L$$

Il est facile de voir que $\mathcal{NP}$-complet est une classe d'équivalence pour la réduction polynomiale, puisque par définition si deux langages $L_1$ et $L_2$ appartiennent à $\mathcal{NP}$-complet, on a $L_1 \propto L_2$ et $L_2 \propto L_1$. Evidemment, notre définition n'indique pas s'il existe ou non des langages dans cette classe... En effet la relation $\propto$ étant simplement un pré-ordre et non un ordre total, *a priori* il peut exister deux langages maximaux pour la réduction polynomiale, qui soient par ailleurs incomparables. Dans une telle configuration $\propto$ n'admettrait pas d'élément maximum ($\mathcal{NP}$-complet $= \emptyset$) mais uniquement des éléments maximaux.

Par contre si $\mathcal{NP}$-complet n'est pas vide, cette classe contient les langages les plus "difficiles" de $\mathcal{NP}$, au sens où si l'on peut décider n'importe lequel de ces langages en temps polynomial, alors tout langage de $\mathcal{NP}$ peut être décidé en temps polynomial.

**Propriété 2** *Si un langage de $\mathcal{NP}$-complet peut être décidé par un algorithme polynomial, alors tous les langages de $\mathcal{NP}$ sont décidables en temps polynomial :*

$$\mathcal{P} \cap \mathcal{NP}\text{-complet} \neq \emptyset \Rightarrow \mathcal{P} = \mathcal{NP}$$

Si la question de savoir si $\mathcal{NP}$-complet est non vide a été résolue par le célèbre théorème de Cook que nous présentons à la section suivante, la question de savoir si $\mathcal{NP}$-complet et $\mathcal{P}$ sont disjoints est toujours d'actualité. Ladner [**?**] a montré que soit $\mathcal{P} = \mathcal{NP}$, soit il existe une infinité de classes d'équivalences distinctes sur $\mathcal{NP}$, comprenant entre autres $\mathcal{P}$ et $\mathcal{NP}$-complet. De plus, dans ce cas on peut montrer que ces classes sont denses. Cela signifie que pour tout couple de problèmes $P_1$ et $P_2$ dans $\mathcal{NP}$ non équivalents et tels que $P_1 \propto P_2$ alors, il existe un autre problème $P_{1,2}$ de $\mathcal{NP}$ tel que $P_1 \propto P_{1,2} \propto P_2$ non équivalent à $P_1$ et $P_2$. Ladner a également montré qu'il existait des problèmes de $\mathcal{NP}$ non comparables entre eux. Enfin, même si $\mathcal{NP}$ admet un plus grand et un plus petit élément (respectivement $\mathcal{P}$ et $\mathcal{NP}$-complet), il n'a pas une structure de

Treillis.

Nous pouvons donner une première structure grossière pour $\mathcal{NP}$ comme représenté dans la figure 9 ci-dessous.

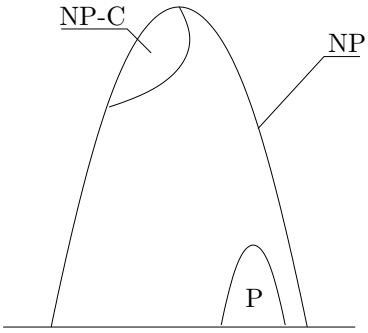



FIG. 9 – Structure possible de $\mathcal{NP}$, si $\mathcal{P} \neq \mathcal{NP}$

## 3.1 Le Théorème de Cook

Nous allons montrer dans cette section l'existence d'un premier problème de $\mathcal{NP}$-complet. Ce premier problème à avoir été démontré $\mathcal{NP}$-complet, par Cook, est un problème de logique : SATISFIABILITY.

### 3.1.1 Le problème SATISFIABILITY (SAT)

Le problème SAT est un problème de logique propositionnelle, encore appelée logique d'ordre 0 ou logique des prédicats. Rappelons qu'un prédicat est défini sur un ensemble de variables logiques, à l'aide des 3 opérations élémentaires suivantes : la négation NON ($\neg x$ que nous noterons aussi $\bar{x}$), la conjonction ET ($x \wedge y$) et la disjonction OU ($x \vee y$). Un littéral est un prédicat formé d'une seule variable ($x$) ou de sa négation $\bar{x}$.

Une clause est un prédicat particulier, formée uniquement de la disjonction de littéraux, par exemple $C = x \vee \bar{y} \vee z$. Une formule est sous forme normale conjonctive si elle s'écrit comme la conjonction de clauses. Le problème SAT consiste à décider si une formule en forme normale conjonctive est satisfiable, c'est-à-dire si il existe une assignation $\tau$ de valeur de vérité ($\{\text{VRAI}, \text{FAUX}\}$) aux variables telle que toutes les clauses sont VRAI.

SATISFIABILITY (SAT)
INSTANCE : Un ensemble de clauses $C = \{C_1, \ldots, C_m\}$
QUESTION : La formule $C_1 \wedge \cdots \wedge C_m$ est-elle satisfiable ?

Par exemple la formule $(x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y}) \wedge (x \vee \bar{z})$ est satisfaite avec l'assignation $\tau(x) =$ VRAI et $\tau(y) =$ FAUX.

Il est facile de se convaincre que le problème SAT est dans $\mathcal{NP}$. Un certificat de positivité consiste à donner une assignation des variables, codable sur un vecteur de $n$ bits, si $n$ est le nombre de variables des clauses. La vérification que toutes les clauses sont satisfaites est alors clairement polynomiale (plus précisément, dans $\mathcal{O}(nm)$). La partie suivante établie que ce problème est dans $\mathcal{NP}$-complet.

### 3.1.2 Démonstration du théorème de Cook

Nous allons maintenant démontrer le théorème de Cook, qui établit l'existence d'un problème $\mathcal{NP}$-complet :

**Théorème 1 (Cook's Theorem)** SAT $\in \mathcal{NP}$*-complet*

Compte tenu de notre remarque précédente SAT $\in \mathcal{NP}$, il nous faut prouver que :

$\forall L \in \mathcal{NP}$ il existe une réduction polynomiale de $L$ vers SAT.

La difficulté vient du fait qu'il faut prouver l'existence d'une réduction pour tous les langages $L$ de $\mathcal{NP}$. La seule indication dont on dispose est l'existence d'une TM non déterministe qui accepte $L$ en temps polynomial.

Reprécisons tout d'abord la notion de temps d'exécution sur une NTM non déterministe sur un mot $w \in L$ : c'est le minimum des temps d'exécution parmi toutes les exécutions acceptant $w$. L'idée de la preuve du théorème de Cook est de coder l'exécution d'une NTM $T$ reconnaissant $L$ comme une formule logique. Nous allons montrer pour cela comment coder les données de $L$, toutes les informations sur le ruban de $T$, mais aussi les états et les transitions.

Montrons l'existence d'une transformation qui à chaque mot $w$ et tout langage (problème) $L \in \mathcal{NP}$ associe une instance $L_{SAT}$ qui est positive si et seulement si $w \in L$. On considère une TM non déterministe $(Q, \Gamma, \Sigma, \Delta, \square, q_0, q_A)$ dont la complexité est bornée par un polynôme $p(n)$ et $w$ un mot de $\Sigma^*$ de longueur $n$. Cette machine accepte le mot $w$ si et seulement si il existe une exécution de TM sur $w$ de longueur au plus $p(n)$ qui mène vers un état accepteur. Une telle exécution peut être représentée par $p(n)$ configurations successives de TM. Chaque configuration est caractérisée par l'état, le contenu du ruban et la position de la tête de lecture. On propose de la représenter par un ensemble de tableaux :

- Un tableau $R$ à deux dimensions de taille $(p(n)+1)$ par $(p(n)+1)$ qui indique pour chaque configuration et chaque case du ruban le symbole de $\Gamma$ se trouvant dans cette case. On peut noter ici que le nombre de cases visitées est au plus $p(n)+1$ car on visite au plus une nouvelle case à chaque transition.
- Un tableau $Q$ mono-dimensionnel de taille $(p(n)+1)$ donnant l'état de chaque configuration $i$. On note $K$ le nombre d'états.

– Un tableau $P$ mono-dimensionnel de taille $(p(n)+1)$ donnant la position de la tête de lecture dans chaque configuration $i$.
– Enfin, un tableau $C$ mono-dimensionnel de taille $(p(n)+1)$ donnant le choix non déterministe effectué par la machine à chaque étape. On note $r$ le nombre maximum de ces choix.
Ce tableau a un status un peu particulier, il n'est pas strictement nécessaire à la description logique de l'exécution, mais sera utile pour vérifier que le contenu des tableaux définit bien une exécution valide.

La transformation produit une formule logique qui est satisfaite que pour un contenu de ces tableaux qui définit une exécution acceptant $w$. On introduit maintenant une variable propositionnelle par case des tableaux (il y en a un nombre polynomial $O(p(n)^2)$ :
– $r_{ij\sigma}$ pour $0 \leq i,j \leq p(n)$ et $\sigma \in \Gamma$. Cette valeur est à 1 si le symbole $\sigma$ est dans la case $(i,j)$.
– $q_{ik}$ pour $0 \leq i \leq p(n)$ et $k \in Q$. Cette valeur est 1 si l'état de la configuration $i$ est $k$ et 0 sinon.
– $p_{ij}$ pour $0 \leq i,j \leq p(n)$. Cette valeur est à 1 si la tête de lecture pointe sur la case $j$.
– $c_{is}$ pour $0 \leq i \leq p(n)$ et $1 \leq s \leq r$ si la fonction de transition s'effectue sur le choix $s$ à l'étape $i$.

Ces formules n'ont de sens que s'il n'existe qu'un seul symbole de $\Gamma$ à un moment donné sur une case, qu'un seul état et que la tête de lecture ne pointe que sur une seule case. Il est donc nécessaire de restreindre le nombre de fonctions d'interprétation possibles.
Tout d'abord, exprimons que **chaque case ne contient qu'un seul symbole** de $\Gamma$ :

$r_{ij\sigma} \Longrightarrow \overline{r_{ij\sigma'}}$ $\forall i,j$ pour $\sigma \neq \sigma'$. ce qui se traduit en forme normale conjonctive par la formule $(\bigwedge_{\sigma\neq\sigma'}(\overline{r_{ij\sigma}} \vee \overline{r_{ij\sigma'}})$.

De plus, $(\bigvee_{\sigma\in\Gamma} r_{ij\sigma})$ exprime que le contenu de la case est bien un symbole de $\Gamma$, ainsi, il suffit d'exprimer pour toutes les cases les deux conditions à la fois :

$$\bigwedge_{0\leq i,j\leq p(n)} (\bigwedge_{\sigma\neq\sigma'} (\overline{r_{ij\sigma}} \vee \overline{r_{ij\sigma'}}) \wedge (\bigvee_{\sigma\in\Gamma} r_{ij\sigma}))$$

On vérifie aisément que cette formule est sous forme normale conjonctive et que sa longueur est polynomiale, précisément, elle est dans $O(p(n)^2)$.

Il faut exprimer de la même manière que l'on a qu'une seule configuration à la fois, que la tête de lecture ne pointe que sur une seule case et qu'il n'y a qu'un seul choix possible à un moment donné pour une transition. On donne ci-dessous la formule logique qui concerne le tableau $P$ des positions de la tête de lecture dont la longueur est dans $O(p(n)^3)$ :

$$\bigwedge_{0\leq i\leq p(n)} (\bigwedge_{0\leq j,j'\leq p(n) j\neq j'} (\overline{p_{ij}} \vee \overline{p_{ij'}}) \wedge (\bigvee_{0\leq j\leq p(n)} p_{ij}))$$

Il reste maintenant à exprimer sous forme logique que toute fonction d'interprétation définit bien une exécution de la machine acceptant le mot $w$ :

- **La première configuration est la configuration initiale**. Ceci s'exprime par les conditions que les $n$ caractères du mot d'entrée $w$ figurent sur le ruban au départ (le reste étant le mot blanc), que la position de la tête de lecture soit sur la première case du mot et que l'état soit l'état initial. Ici, nous avons supposé que le mot était rangé à partir de la position $j = 0$, ce qui est un peu abusif car les déplacements de la tête de lecture peuvent se faire aussi bien vers la droite que vers la gauche. Ceci ne change évidemment pas fondamentalement la formule donnée (on pourrait par exemple considérer un tableau $R$ plus large ($-p(n)$ à $+p(n)$)) ou plus simplement considérer une machine de Türing à ruban infini d'un seul coté.

$$[\bigwedge_{0\leq j\leq n-1} r_{0jw_{j+1}} \wedge \bigwedge_{n\leq j\leq p(n)} r_{0j\square}] \wedge q_{00} \wedge p_{00}$$

- **Le passage d'une étape à la suivante est bien conforme à la fonction de transition**.
  Il nous faut exprimer deux conditions : les cases du ruban qui ne sont pas placées sous la tête de lecture ne sont pas modifiées et (ce qui est le plus délicat) que le passage d'une configuration à la suivante est bien conforme à la transition (symbole, état et choix) :

  Tout d'abord, précisons la formule qui exprime que les cases non placées sous la tête de lecture ne sont pas modifiées :

$$\bigwedge_{\substack{0\leq i\leq p(n)\\0\leq j\leq p(n)\\\sigma\in\Gamma}} [(r_{ij\sigma} \wedge \overline{p_{ij}}) \Longrightarrow r_{(i+1)j\sigma}]$$

  que l'on transforme en forme normale conjonctive :

$$\bigwedge(\overline{r_{ij\sigma}} \vee p_{ij} \vee r_{(i+1)j\sigma})$$

  Détaillons maintenant la formule qui atteste de la conformité du passage de la configuration $i$ à $i+1$ qui passe de l'état $k$ à l'unique état $k'$
  $\forall$ position $j$ à l'étape $i$ :
  $r_{ij\sigma} \wedge q_{ik} \wedge p_{ij} \wedge c_{is} \Longrightarrow\ q_{(i+1)k'}$
  Qui s'écrit en forme normale conjonctive :
  $\overline{r_{ij\sigma}} \vee \overline{q_{ik}} \vee \overline{p_{ij}} \vee \overline{c_{is}} \vee\ q_{(i+1)k'}$
  On obtient une formule similaire pour le passage du symbole $\sigma$ à $\sigma'$).
  Pour assurer que le déplacement est bien conforme, il faut considérer une

variable supplémentaire $d$ ($d = -1, 0, +1$) qui traduit de façon unique la relation de transition. Finalement, la formule est :

$$\bigwedge_{\substack{0\leq i,j\leq p(n)\\ \sigma\in\Gamma\\ 1\leq s\leq r\\ 1\leq k\leq K}} [(\overline{r_{ij\sigma}}\vee\overline{q_{ik}}\vee\overline{p_{ij}}\vee\overline{c_{is}}\vee q_{(i+1)k'})\wedge(\overline{r_{ij\sigma}}\vee\overline{q_{ik}}\vee\overline{p_{ij}}\vee\overline{c_{is}}\vee r_{(i+1)j\sigma'})\wedge(\overline{r_{ij\sigma}}\vee\overline{q_{ik}}\vee\overline{p_{ij}}\vee\overline{c_{is}}\vee p_{(i+1)(j+d)})$$

– **Finalement, on exprime que l'on atteint bien l'état accepteur en $p(n)+1$ étapes :** $\bigvee_{i\neq p(n)} q_{i,q_A}$

Toutes ces transformations sont polynomiales. La formule finale est obtenue comme la conjonction de toutes les formules précédentes.

## 3.2 Quelques problèmes $\mathcal{NP}$-complets

Nous donnons dans cette section le résumé de quelques problèmes de réference et leurs réductions. Nous avons suivi partiellement la présentation historique de Garey et Johnson, nous l'avons retenue ici pour son caractère pédagogique en remettant au goût du jour certaines preuves. Ces problèmes sont connus et les détails peuvent être trouvés dans [8]. Ces six problèmes font eux-mêmes partie des 21 problèmes originaux introduits par Karp en 1972 [**?**].

### 3.2.1 Variations autour de SAT

Le problème 3SAT est une forme simplifiée de SAT.

3SAT

INSTANCE : Une collection de clauses $C = \{c_1, c_2, ...c_m\}$ d'un ensemble fini $U$ de variables telles que $card(c_i) = 3$ pour $1 \leq i \leq m$.

QUESTION : Existe-t-il une affectation de $U$ qui satisfasse toutes les clauses de $C$ ?

La réduction de 3SAT à partir de SAT est presque immédiate. La démonstration est classique et présente dans la plupart des ouvrages. Nous ne la présentons pas en détails ici, l'idée est de transformer les clauses de SAT en clauses de cardinalité 3 en introduisant de nouvelles variables logiques.

On peut également considérer des variantes comme 3SAT-NAE (not all equal). ce problème correspond toujourts à la satisfaisabilité d'une formule logique formée de la conjonction de $m$ clauses d'exactement 3 littéraux où l'on impose de plus que dans chaque clause, au moins une valeur d'un littéral soit VRAI et au moins une soit FAUX.

Une autre variante intéressante est 4SAT-E2 où exactement 2 littéraux sont VRAI et 2 sont FAUX dans chaque clause composée d'exactement 4 littéraux.

Ces deux problèmes sont $\mathcal{NP}$-complets.

On peut se poser naturellement la question de la complexité de 2SAT (défini comme 3SAT avec des clauses formées d'exactement 2 littéraux). Si le problème 3SAT est $\mathcal{NP}$-complet, *a contrario* 2SAT est polynomial : il admet même un algorithme linéaire ! Ceci fait l'objet de l'exercice **??** à la fin du chapitre.

### 3.2.2 IndependentSet

Le problème que nous considérons maintenant un problème bien connu en théorie des graphes. On considère un graphe $G$. Un stable de $G$ est un ensemble de sommets deux à deux non adjacents. La question est de déterminer s'il existe dans $G$ un stable de cardinalité fixée. Plus formellement :

INDEPENDENTSET (IS)
INSTANCE : un graphe $G = (V, E)$ et un entier $K$
QUESTION : Existe-t-il dans $G$ un stable de cardinalité supérieure ou égale à $K$ ?

La démonstration classique s'obtient par réduction à partir de 3SAT, elle est laissée au lecteur.

### 3.2.3 VertexCover

Le problème VC (transversal) fait partie des six problèmes de base de Garey et Johnson [8]. a NP-complétude y est prouvée par une réduction à partir de 3SAT. Nous allons ici prouver qu'il est NP-complet directement à partir de IS.

On appelle *transversal* d'un graphe $G = (V, E)$ un ensemble $C$ de sommets de $V$ tel que toute arête de $E$ a au moins une de ses extrémités dans $C$. Le problème VERTEXCOVER consiste à décider si un graphe admet un transversal de cardinalité inférieure à un entier $K$ donné.

VERTEX COVER (VC)
INSTANCE : un graphe $G$ et un entier $K$.
QUESTION : Existe-t-il un transversal de cardinalité inférieure ou égale à $K$ ?

Remarquons tout d'abord trivialement que ce problème est bien dans $\mathcal{NP}$. Un ensemble $C$ de sommets est un transversal si et seulement si son complémentaire $S = V - C$ est un stable. La réduction polynomiale à partir de IS est donc immédiate.

### 3.2.4 Autres problèmes classiques

Voilà une liste de problèmes NP-complets classiques :

3DM
INSTANCE : 3 ensembles disjoints $X$, $Y$ et $Z$ de même cardinalité, un ensemble $M \subseteq X \times Y \times Z$.

QUESTION : L'ensemble $M$ contient-il un couplage parfait ? C'est-à-dire existe-il dans $M$ un ensemble de $|X|$ triplets disjoints ?

PARTITION

INSTANCE : un ensemble fini $A$ dont chaque élément possède un poids $s(a)$.

QUESTION : Existe-t-il un sous-ensemble $A'$ de $A$ tel que $\sum a \in A' s(a) = \sum a \in A - A' s(a)$ ?

On peut montrer que ce problème est NP-complet par plusieurs méthodes. Une façon est de le réduire à partir d'une variante de la variante 3SAT-NAE de SAT.

### 3.2.5 Retour sur HC

Nous avons déjà présenté le problème de la recherche d'un cycle hamiltonnien dans un graphe. Nous avons prouvé que le problème D-TSP se réduisait polynomialement à partir de HC. Il existe une démonstration classique qui montre que HC est NP-complet à partir de VC. Ceci prouve par transitivité que D-TSP est lui aussi NP-complet.

### 3.2.6 Un bilan provisoire

La figure ci-dessous donne les schémas de réduction de quelques problèmes fondamentaux à partir du problème SAT.

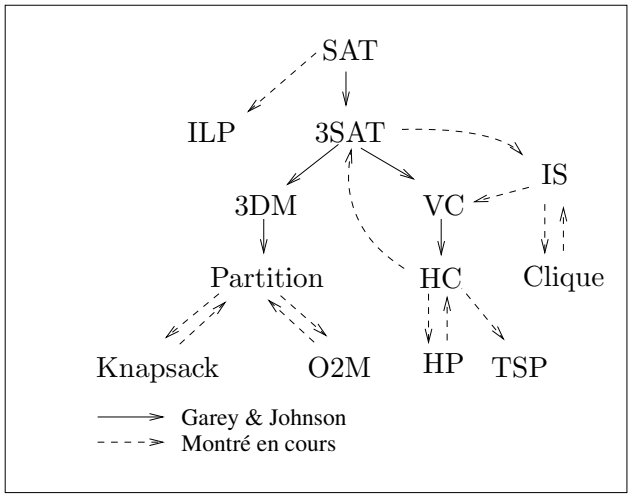



FIG. 10 – Quelques réductions à partir de SAT.

On peut construire un tableau plus complet, contenant d'autres problèmes, comme le coloriage, l'ordonnancement, etc.. Un site est mis à jour régulièrement

[compendium].

### 3.3 Frontière entre $\mathcal{P}$ et $\mathcal{NP}$

L'étude de la complexité de certains problèmes permettent de conclure qu'ils sont polynomiaux, d'autres qu'ils sont NP-complets. Pour ces derniers, en l'état des connaissances aujourd'hui, il est inenvisageable de trouver des algorithmes efficaces pour les résoudre. Il est donc important d'étudier des problèmes à la frontière, relaxer des problèmes difficiles pour obtenir des sous-problèmes polynomiaux...

Prenons l'exemple d'un problème d'ordonnancement général : le problème avec contraintes de précédence et temps unitaires est polynomial pour $m = 2$, il est NP-complet pour $m$ arbitraire. Par contre, sa complexité est inconnue pour $m = 3$. Il est polynomial pour certaines structures de précédences comme les arbres. Le problème à nombre non borné de machines pour des tâches quelconques avec précédence est polynomial (algorithme de plus long chemin).

On rappelle le problème de l'ordonnancement à 2 machines :

O2M

INSTANCE : : $n$ tâches indépendantes de durées $p_j$ (entières) pour $1 \leq j \leq n$, $D$ un entier.

QUESTION : : Existe-t-il un ordonnancement réalisable de durée inférieure ou égale à $D$ ?

On réduit facilement ce problème à partir de partition.

On peut également introduire un élément de comparaison supplémentaire avec le problème du Sac-à-dos. Il est possible de montrer formellement l'équivalence de ces trois problèmes (Partition, Sac-à-dos et O2M). Ceci fait l'objet d'un exercice à la fin du chapitre. En particulier, comme on a montré que Partition était NP-complet, les deux autres se sont également...

## 4 Autres classes de Complexité

### 4.1 Complémentaire

Suivant le type de réponses que l'on cherche, on peut s'interroger sur la classe des langages $L$ (problèmes) dont le complément est dans $\mathcal{NP}$. Ceci définit la classe co-$\mathcal{NP} = \{L \mid \overline{L} \in \mathcal{NP}\}$.

Remarquons tout d'abord que si un problème est dans $\mathcal{NP}$, son complément ne l'est pas forcément. En effet, le caractère non déterministe de la TM ne permet pas simplement d'inverser les réponses :

elle n'accepte une exécution que s'il existe une exécution qui l'accepte, elle refuse donc si TOUTES les exécutions refusent, condition qui n'est pas forcément vérifiable par une TM non déterministe.

Par exemple, il a été facile d'établir HC $\in \mathcal{NP}$. Vérifier qu'il n'y a pas de cycle hamiltonnien dans un graphe (ce qui correspond à HC $\in$ co-$\mathcal{NP}$) est plus difficile sans les énumérer tous...

De façon symétrique, considérons le problème qui cherche à établir la primalité d'un entier. Rappelons ce problème qui a déjà été introduit dans la section 1 :

PRIME
INSTANCE : un entier $N$.
QUESTION : $N$ est-il premier ?

On montre facilement que PRIME est dans co-$\mathcal{NP}$ en exhibant un certificat concis pour le "non". En effet, si l'entier n'est pas premier, il s'écrit comme produit de ses diviseurs dont il est facile de vérifier que le produit (avec leurs ordres de multiplicité) est bien égal à l'entier. Par contre, comme nous l'avons déjà évoqué, il n'est pas évident de montrer que PRIME $\in \mathcal{NP}$.

On a facilement $\mathcal{P} = co - \mathcal{P}$, il est en effet équivalent de chercher les instances positives ou négatives si le temps est polynomial sur une machine de Türing déterministe. Flotmax et Coupemin sont deux problèmes duaux qui appartiennent tous deux à $\mathcal{NP} \bigcap$ co-$\mathcal{NP}$. En fait, ils appartiennent à $\mathcal{P}$.

On de manière évidente $\mathcal{P} \subset \mathcal{NP} \bigcap$ co-$\mathcal{NP}$. L'inclusion stricte reste une question ouverte.

## 4.2 Retour sur la notion de complexité

Nous avons introduit la notion de complexité pour distinguer (classifier) les problèmes par niveaux de difficulté. Nous avons beaucoup insisté sur la complexité en temps (nombres d'opérations élémentaires requises pour résoudre un problème). Cependant, la complexité en espace est également très importante et bien sûr, elle est liée la complexité en temps.

Plusieurs tentatives ont été lancées pour définir la complexité par le *contenu en calcul* (longueur du programme de résolution du problème). La complexité de Kolmogorov est définie par la longueur du plus petit programme existant pour résoudre un problème. Elle se mesure également en fonction de la taille de l'instance.

Considérons à titre d'exemple le problème de compter le nombre de "1" d'une suite binaire de longueur $n$. La complexité en temps de ce problème est trivialement dans $O(n)$, en effet, il existe un algorithme simple qui répond à la question en parcourant les termes de la suite où l'on incrémente un compteur à chaque fois que l'on rencontre un "1". C'est égelement une borne inférieure car il est nécessaire de parcourir tous les termes... La complexité en espace est dans $O(log_2(n))$, c'est l'espace nécessaire à stocker un compteur (entier) et le terme courant de la suite. La complexité en contenu en calcul est dans $O(1)$ car

l'algorithme ne dépend pas de la taille de la suite.

Nous présentons dans la suite de cette section les principaux résultats existants dans chacun de ces domaines. Pour la complexité en temps, nous nous interrogerons sur les classes au delà de $\mathcal{NP}$, puis sur la complexité parallèle, qui est une tentative pour réduire la complexité en temps des problèmes en considérant des modèles de calcul plus puissants. Nous détaillerons ensuite des classes de complexité en espace, au delà de $\mathcal{NP}$, mais aussi à l'intérieur de $\mathcal{P}$ (cette classification est liée à la complexité parallèle). Enfin, nous ouvrirons sur la complexité en contenu en calcul en liaison avec la théorie de l'information.

## 4.3 Complexité en temps

Nous avons défini les classes de complexité $\mathcal{P}$ et $\mathcal{NP}$ et nous avons étudié leurs relations. Comme nous l'avons évoqué, une des questions importantes non résolue est de déterminer si l'inclusion de $\mathcal{P}$ dans $\mathcal{NP}$ est stricte ou non. De manière analogue à ces deux classes, nous pouvons définir la classe $EXPTIME$ des algorithmes à complexité bornée par une exponentielle sur une machine de Türing déterministe :

$$EXPTIME = \bigcup_k TIME(2^{n^k})$$

On définit de manière analogue $NEXPTIME$ pour les machines non déterministes. On a de manière évidente l'inclusion de $EXPTIME$ dans $NEXPTIME$. De même que pour $\mathcal{P}$ et $\mathcal{NP}$, on se pose naturellement la question de l'inclusion stricte de $EXPTIME$ dans $NEXPTIME$, qui est également un problème ouvert. On peut par contre démontrer que l'inclusion $\mathcal{P}$ dans $EXPTIME$ est stricte, $\mathcal{P} \subset EXPTIME$.

Remarquons que $EXPTIME = co-EXPTIME$ car le modèle sous-jacent est la machine de Türing déterministe. De même, on montre facilement que co-$\mathcal{NP} \subset EXPTIME$.

## 4.4 Complexité en mémoire

La classe $PSPACE$ est définie par les langages acceptés par une machine de Türing déterministe dont le nombre de cases utilisées est bornée par un polynôme en la taille du mot d'entrée. Plus formellement :

$PSPACE = \bigcup_{k=1} SPACE(n^k)$ où $SPACE(n^k)$ est l'ensemble des langages acceptés par une machine de Türing déterministe dont la complexité en espace est bornée par $O(n^k)$. De même, $NSPACE$ est l'ensemble des langages reconnus en espace polynomial sur une TM non déterministe.

On a en particulier $PSPACE = NPSPACE$. On en déduit $\mathcal{NP} \subset PSPACE$.

Il est intéressant de pouvoir distinguer quels problèmes possèdent des solutions efficaces en mémoire de celles qui n'en ont pas. Ainsi, on définit de manière analogue à la complexité en temps, la classe $PSPACE$-complet. Nous considérons ci-dessous un problème de jeu à deux joueurs $I$ et $II$. Le premier joueur commence par donner un nom de ville (par exemple *La Bérarde*), le second doit en trouver un autre qui commence par la dernière lettre du nom de la ville précédente (par exemple *Entragues*), et ainsi de suite sans redite entre les villes. En utilisant un formalisme de graphe, on peut le reformuler de la façon suivante :

JEUGEO
INSTANCE : un tableau de $n$ mots (suites de k caractères au plus) et un mot initial.
QUESTION : est-ce que la configuration est gagnante pour le joueur $I$ ?

Dire que le problème JEUGEO $\in PSPACE$-complet assure qu'il n'existe pas d'algorithmes polynomiaux qui permettent de décider qu'une position arbitraire est gagnante...

Un autre problème $PSPACE$-complet est celui de la satisfaisabilité d'une formule booléenne dont toutes les variables sont associées à un quantificateur.

## 4.5 Structuration de $\mathcal{P}$

De même que l'on a défini la classe PSPACE, on peut définir une nouvelle classe des langages acceptés en temps polylogarithmique sur une machine de Türing déterministe (et sur une TM non déterministe) :

$L = \bigcup_{k=1} SPACE((log(n))^k)$.

$NL = \bigcup_{k=1} NSPACE((log(n))^k)$.

Avec la même idée, on définit $\alpha_L$ une réduction au sens de l'espace polylogarithmique.

GAP - GRAPH ACCESSIBILITY PROBLEM
INSTANCE : un graphe orienté $G = (V, E)$ d'ordre $n$ et deux sommets $s$ et $t$ appartenant à $V$
QUESTION : existe-t-il un chemin de $s$ à $t$ dans $G$ ?

Montrons que ce problème est $NL$-complet.
Tout d'abord, GAP est dans $NL$. En effet, on peut décrire un algorithme qui passe en revue les sommets successifs d'un chemin élémentaire (donc, de longueur inférieure à $n$). Il ne garde en mémoire à chaque étape que 3 entiers : les indices des deux sommets consécutifs sur le chemin et un compteur.

Il faut ensuite montrer que $\forall$ langage de $NL$, il existe une réduction vers GAP. L'idée est proche de celle qui a permis d'établir le théorème de Cook. On considère une machine de Türing quelconque qui accepte un programme de $NL$. Soit $s$ le mot d'entrée. Elle accepte ce mot si et seulement si la sortie de GAP appliqué au graphe des exécutions possibles de cette TM est positive. En effet, la séquence des descriptions instantanées représente un chemin dans ce graphe.

On peut également montrer que le problème 2SAT est $NL$-complet.

## 4.6 Bilan

On peut récapituler les résultats actuellement connus :

$\mathcal{P} \subseteq \mathcal{NP} \subseteq PSPACE \subseteq EXPTIME$
$\mathcal{P} \subset EXPTIME$
$\mathcal{P} \subset PSPACE$
$\mathcal{P} \subseteq \mathcal{NP}$ et $\mathcal{P} \subseteq$ co-$\mathcal{NP}$
$\mathcal{P} \subseteq (\mathcal{NP} \bigcap$ co-$\mathcal{NP})$ sans information sur l'inclusion stricte
$EXPTIME \subseteq NEXPTIME$
$EXPTIME = co - EXPTIME$
$\mathcal{P} = co - \mathcal{P}$
$PSPACE = NPSPACE$
$PSPACE \subseteq EXPTIME$
La figure suivante 11 représente graphiquement les résultats que nous venons de présenter.

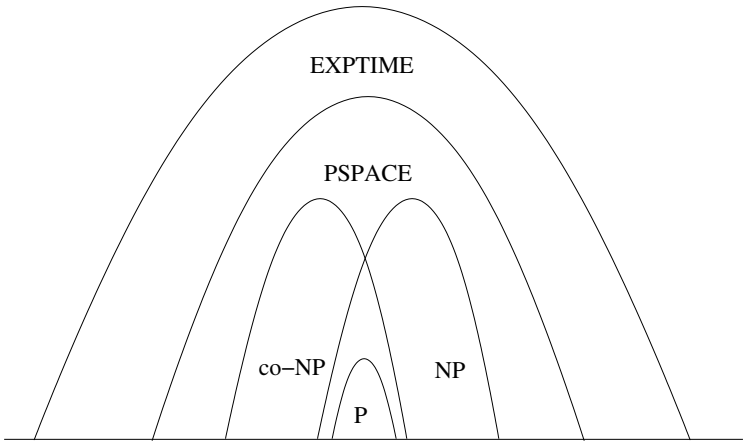



FIG. 11 – Structure complète (en supposant $\mathcal{NP} \neq$ co-$\mathcal{NP}$).

# 5 $\mathcal{NP}$-complétude au sens fort

Les langages $\mathcal{NP}$-complet sont les langages les plus "difficiles" de $\mathcal{NP}$ au sens de la réduction polynomiale. On peut partitionner cette classe en distinguant les langages $\mathcal{NP}$-complet au sens fort (strongly $\mathcal{NP}$-complet) et les langages $\mathcal{NP}$-complet au sens faible ($\mathcal{NP}$-complet in the ordinary sense). Cette distinction est liée aux problèmes numériques, dans lesquels des nombres interviennent. Intuitivement, le sens fort signifie que la complexité du problème provient de sa structure même, tandis que le sens faible est induit pas la nature des instances traitées.

PARTITION
INSTANCE : Une suite d'entiers $a_1, \ldots, a_n$.
QUESTION : Existe-t-il une partition $A \cup B$ de $[1..n]$ telle que

$$\sum_{i \in A} a_i \;=\; \sum_{i \in B} a_i$$

Le problème PARTITION $\in \mathcal{NP}$-complet. Cependant il peut être résolu par programmation dynamique en calculant le prédicat $P(i, a)$, *vrai* ssi il existe une partition $A \cup B$ de $[1..i]$ telle que $s(A) = a$. La relation liant les prédicats $P(i, a)$ est simplement :

$$P(i+1, a) \;= P(i, a) \vee (P(i, a - a_{i+1}) \wedge a \geq a_{i+1})$$

En posant $2C = \sum_i a_i$, une instance de PARTITION est positive ssi $P(n, C)$ est *vrai*. L'algorithme dynamique résout donc PARTITION en temps $\mathcal{O}(nC)$. Faut-il en déduire que PARTITION $\in \mathcal{P}$, et donc $\mathcal{P} = \mathcal{NP}$,... ?Non, car la taille d'une instance $x$ de PARTITION n'est pas $n$, mais $\sum_i \log a_i$, puisque chaque nombre doit être codés. En faisant intervenir $a_{max} = \max_i\{a_i\}$, on peut borner la taille $|x|$ par $\mathcal{O}(n \log a_{max})$, tandis que notre algorithme s'exécute en temps $\mathcal{O}(n^2 a_{max})$, qui n'est pas bien sûr polynomialement borné par $|x|$...L'algorithme de programmation dynamique pour PARTITION est dit *pseudo-polynomial* :

**Définition 11** *Pour une instance $x$ d'un problème de décision numérique, on note* MAX$(x)$ *la valeur du plus grand entier apparaissant dans l'instance. Par convention, pour les problèmes non numérique, on pose* MAX$(x) = 1$.

**Définition 12** *Un algorithme est dit pseudo-polynomial si son temps d'exécution sur une instance $x$ est polynomialement borné par $|x|$ et* MAX$(x)$.

Bien qu'un algorithme pseudo-polynomial comme l'algorithme de programmation dynamique pour PARTITION puisse avoir un temps d'exécution exponentiel par rapport à la taille de l'instance, il reste un algorithme efficace dans bien des cas, en particulier si MAX$(x)$ est faible. Peut-on obtenir un algorithme pseudo-polynomial pour tout problème de $\mathcal{NP}$-complet ? La réponse est liée à la définition de la $\mathcal{NP}$-complet au sens fort :

**Définition 13** *Pour un problème de décision* $\Pi$ *et un polynôme* $p$*, on note* $\Pi_p$ *le sous-problème de* $\Pi$ *restreint aux instances* $x$ *telles que* $\textsc{Max}(x) \leq p(|x|)$.

**Définition 14** *Un problème* $\Pi \in \mathcal{NP}$*-complet est* $\mathcal{NP}$*-complet au sens fort ssi il existe un polynôme* $p$ *tel que* $\Pi_p$ *est* $\mathcal{NP}$*-complet. Sinon il est* $\mathcal{NP}$*-complet au sens faible.*

Les problèmes $\mathcal{NP}$-complet au sens faible sont simplement définis ici en creux par rapport au sens fort, comme $\mathcal{NP}$-complet en creux par rapport au sens fort, comme $\mathcal{NP}$-complet / $\mathcal{NP}$-complet au sens fort.

**Propriété 3** *Si* $\mathcal{P} \neq \mathcal{NP}$*, un problème* $\mathcal{NP}$*-complet au sens fort n'admet pas d'algorithme pseudo-polynomial.*

Par contre rien n'assure qu'un problème au sens faible admette un tel algorithme pseudo-polynomial. En effet, si pour tout polynôme $p$, le sous-problème $\Pi_p$ est dans $\mathcal{P}$, un algorithme pseudo-polynomial existe effectivement. Mais la $\mathcal{NP}$ complétude au sens faible implique seulement que $\Pi_p \notin \mathcal{NP}$-complet,...

# Exercices du Chapitre II

## Retour sur HC

Nous avons déjà présenté le problème de la recherche d'un cycle hamiltonnien dans un graphe.

On demande de montrer la réduction HC $\alpha$ SAT de HC vers SAT (bien sûr, ceci n'amène rien de nouveau en vertus du théorème de Cook sinon une démonstration beaucoup plus courte !).

**Indication :**

On introduit des variables logiques pour faire le lien entre les deux problèmes : $x_{ij}$ = VRAI si le sommet $i$ est en position $j$ sur le cycle (pour $1 \leq i, j \leq n$).

Il faut traduire sous forme logique les contraintes suivantes :

- un sommet $i$ apparaît à la jième position : $\forall j,\ 1 \leq j \leq n$

$$\bigvee_i x_{ij}$$

- au plus un sommet doit apparaître à la jème place :

$$\bigwedge_{<\leq i,j,k \leq n} (\overline{x_{ij}} \vee \overline{x_{kj}})$$

  pour $i \neq k$
- le sommet $i$ n'apparaît qu'une seule fois dans le cycle : $\forall i,\ 1 \leq i \leq n$

$$\bigvee_j x_{ij}$$

  et $(\overline{x_{ij}} \vee \overline{x_{kj}})$ pour $j \neq k\ \forall i, j, k$
- respect de la structure du graphe :

  Deux sommets non liés par un arc ne sont pas consécutifs dans le cycle.

On vérifie facilement que cette réduction est bien polynomiale $O(n^2)$ variables et $O(n^3)$ clauses.

## 5.1 Equivalence Partition, Sac-à-dos et ordonnancement sur deux machines

Cet exercice a été suggéré par Evripidis Bampis. Il s'agit de montrer l'équivalence de trois problèmes *proches* avec le minimum d'efforts.

Ces trois problèmes Partition, Sac-à-dos et O2M ont été défini dans le cours. Evidemment, il est inutile de montrer les 6 étapes de réduction. Certaines sont triviales, d'autres moins...

## îndications

- Partition $\alpha$ O2M est très facile en posant $D = \frac{1}{2}\sum p_j$.
- La réciproque est plus dure à montrer, mais aussi plus intéressante. O2M $\alpha$ partition :
  On définit $I = 2D - \sum_{j=1,n} p_j$ (la justification de ce $I$ est le cumul des périodes d'inactivités. On introduit des xx dont les valeurs sont telles que d'une part, la somme est égale à $I$ et d'autre part, on peut construire n'importe quel xx entre 0 et $I$ en additionnant les valeurs d'un sous-ensemble de ces nouveaux xx.
  Attention ici à garder la réduction polynomiale. On introduit toutes les puissances de 2 dont la valeur est strictement inférieure à I sur 2 qui ramène leur somme à $I$ (exemple 56, on prend 1, 2, 4, 8, 16 et 25). La réduction est polynomiale car on rajoute $log_2 I$ nombres...
- Sac-à-dos $\alpha$ Partition.
  Indication : On introduit deux variables supplémentaires à celles du sac à dos $a_{n+1} = 2H + 2K$ et $a_{n+2} = 4H$ et l'instance de partition avec ces $n + 2$ données. De plus, on remarque que les variables $a_{n+1}$ et $a_{n+2}$ sont forcément dans deux ensembles différents.
- Il reste la dernière réduction Partition $\alpha$ Sac-à-dos...

## 5.2 2SAT

L'objectif de cet exercice est d'étudier la complexité du problème SAT pour des formules formées d'exactement deux littéraux.

Nous avons vu que 3SAT était un problème difficile, on demande de démontrer que 2SAT est polynomial.

## Indications

l'idée est de représenter la formule logique par un graphe, puis, de montrer que 2SAT correspond à un problème de cheminement dans ce graphe.

Pour chaque clause $(x \vee y)$ on associe quatre sommets : $x, \overline{x}, y$ et $\overline{y}$. Si $x$ est FAUX, alors $y$ doit être VRAI. On écrit alors $\overline{x} \Rightarrow y$ ce qui se traduit par un arc entre $\overline{x}$ et $y$. Symétriquement, il existe un arc entre $x$ et $\overline{y}$.

Il suffit de vérifier que dans le graphe ainsi construit, il n'y a pas de chemin entre un sommet et son complémentaire, pour tous les sommets.

## 5.3 NP et co-NP

Que pensez-vous des classes (co-NP)-complet et co-(NP-complet) ?