

Chapitre 11

Classes de complexité

I Classes de complexité pour un problème de décision

I.1 Complexité

Définition 11.1

Soit \mathcal{P} un problème abstrait défini par une fonction $f : I \rightarrow S$. On considère un algorithme \mathcal{A} "abstrait" qui le résout.

On y associe un problème *concret* en imposant des choix sur *l'encodage* des entrées et des sorties.

La *complexité du problème concret* correspond à la complexité de l'implémentation de l'algorithme \mathcal{A} (sa réalisation concrète) pour le choix d'encodage.

C'est-à-dire un ordre de grandeur du nombre d'opérations élémentaires réalisées par l'implémentation de l'algorithme pour une entrée en fonction de la *taille de l'entrée* (espace occupé en mémoire pour son encodage).

Remarque 11.2 - cas du graphe

La taille de l'encodage pour un graphe :

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

Exemple 11.3 - taille de l'encodage des entiers

Un entier n s'encode en $\mathcal{O}(\log_2(n))$, taille de l'écriture binaire.

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

Remarque 11.4 - relative aux problèmes

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

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

Exemple 11.5

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

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

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

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

Remarque 11.6

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

I.2 Classe P

Définition 11.7 - classe P

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

Remarque 11.8

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

Remarque 11.9

Les problèmes de la classe P sont raisonnablement traitables. (Attention, on parle toujours uniquement de problème de décision).

Pour considérer des problèmes d'optimisation, on se ramène à des problèmes de décision en introduisant un seuil à dépasser pour la fonction de coût.

Remarque 11.10 - classe EXP

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

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

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

Remarque 11.11 - comparaison entre EXP et P

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

Remarque 11.12

La complexité, soit le nombre d'opérations élémentaires, dépend du modèle de calcul choisi. Il en existe plusieurs.

En MPI, on se place dans le cadre d'un modèle de calcul intuitif, qui correspond au fonctionnement d'un ordinateur exécutant une programmation en C ou en OCaml. Les opérations élémentaires sont alors les suivantes :

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

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

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

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

Exemple 11.13 - problème du dernier exemple

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

Exemple 11.14 - problème 2-SAT

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

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

Construction du graphe :

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

se fait en $\mathcal{O}(k)$

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

I.3 Classe NP

Définition 11.15

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

Exemple 11.16

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

Exemple 11.17 - lié au problème *K-COLOR*

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

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

On peut choisir la donnée d'une fonction $c : S \rightarrow \llbracket 1, k \rrbracket$ comme certificat.

Définition 11.18 - classe *NP*

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

Exemple 11.19 - problème *SAT*

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

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

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

Remarque 11.20

L'algorithme vérificateur prend en entrée :

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

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

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

et pour i^- une instance négative :

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

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

Remarque 11.21 - sur la définition

Pour tout instance positive, l'algorithme vérificateur doit renvoyer Vrai pour l'instance accompagnée d'un certificat valide en temps polynomial en la taille de l'entrée.

Pour toute instance négative, l'algorithme vérificateur doit renvoyer Faux pour cette instance, quel que soit le certificat qui l'accompagne.

Définition 11.22 - *formelle de classe NP*

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

1. Σ un alphabet permettant d'encoder un ensemble C de certificats
2. \mathcal{P}' un problème de prédicat :

$$f' : I \times \Sigma^* \rightarrow S = \{Vrai, Faux\}$$

3. $g : I_1 \rightarrow I_2$ une fonction polynomiale telle que pour tout $e \in I$, $f(e)$ est Vrai ssi :

$$\exists c \in C, |c| \leq g(e) \quad \text{et} \quad f'(e, |c|) = Vrai$$

($|\cdot|$ désigne la taille de représentation)

4. le problème \mathcal{P}' est dans \mathcal{P} .

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

Remarque 11.23 - *sur la définition*

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

Remarque 11.24

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

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

Exemple 11.25

On considère le problème CYCLE HAMILTONIEN :

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

```
1 | VERIFICATEUR( $G, \sigma$ ):  
2 |    $n = |S|$   
3 |   vérifier que  $\sigma$  est dans  $\mathcal{S}_n$ .  
4 |   pour  $i=1$  jusqu'à  $n-1$ :  
5 |       Si  $(\sigma(i), \sigma(i+1)) \notin A$ :  
6 |           renvoyer false  
7 |   renvoyer  $((\sigma(n), \sigma(1)) \in A)$ 
```

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

Proposition 11.26 - classe P implique classe NP

$P \subset NP$.

Démonstration

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

Remarque 11.27 - sens réciproque

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

I.4 Le problème d'optimisation

Pour étudier la classe de complexité d'un problème d'optimisation, on le transforme en problème de décision en ajoutant un *seuil* à l'entrée. On répond alors à la question :

”Existe-t-il une solution réalisable dont la valeur atteinte par la fonction de coût est supérieur (maximisation) ou inférieur (minimisation) à un seuil s ?”

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

Remarque 11.28

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

On introduit la notation hors programme suivante.

La classe des problèmes d’optimisation dont le problème de décision associé est de classe P s’appelle la classe PO. De même, pour la classe NP, la classe NPO.

II Réduction polynomiale

Définition 11.29 - réduction polynomiale

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

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

Remarque 11.30

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

Proposition 11.31 - \leq_m^P est presque une relation d’ordre

La relation \leq_m^P est réflexive, transitive, mais non antisymétrique.

Voir démo 1

Proposition 11.32

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

Démonstration

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

Voir suite par Gwénolé.

III NP-complétude, NP-difficile

III.1 Définitions et problème SAT

Définition 11.33

Un problème de décision est *NP-difficile* ou *NP-dur* lorsqu'il est possible de se ramener à tout problème de classe NP par une réduction polynomiale depuis ce problème.

Moralement, il est plus difficile que tous les problèmes NP, mais on peut s'y ramener.

Définition 11.34

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

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

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

Remarque 11.35

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

III.2 Réduction polynomiale et NP-difficulté

Proposition 11.36

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

Démonstration

\leq_m^P est une relation transitive. On va l'utiliser. On a :

$$\begin{cases} \forall \mathcal{P} \in NP, \mathcal{P} \leq_m^P \mathcal{P}_1 \\ \mathcal{P}_1 \leq_m^P \mathcal{P}_2 \end{cases}$$

Par transitivité,

$$\forall \mathcal{P} \in NP, \mathcal{P} \leq_m^P \mathcal{P}_2$$

\mathcal{P}_2 est bien NP-difficile.

Remarque 11.37

Pour montrer qu'un problème \mathcal{P} est NP-complet :

- on montre qu'il est de classe NP l'aide d'un certificat et d'un algorithme vérificateur.
- On choisit un problème \mathcal{P}_{ref} de référence (**SAT**, **FNC-SAT** / **CNF-SAT** ou **3-SAT** sont permis), et on montre que $\mathcal{P}_{ref} \leq_m^P \mathcal{P}$.

III.3 Exemple du problème FNC-SAT ou CNF-SAT

Définition 11.38 - problème **CNF-SAT**

- **Instance** : φ une formule de la logique propositionnelle sous forme normale conjonctive (FNC)
- **Question** : Est-ce que φ est satisfiable ?

Proposition 11.39

CNF est NP-complet.

Remarque 11.40

Utilisable sans rappel au concours. (on l'a dit avant, c'est le cas pour trois problèmes)

Démonstration

Par étapes habituelles. La méthode suivie est décrite en *italique*. Montrons que **CNF-SAT** est de classe NP.

On choisit comme ensemble de certificats l'ensemble des valuations. *Justifions que pour une instance positive au problème, il existe un certificat.* Pour une formule φ satisfiable, il existe v_φ un modèle de φ sur \mathcal{V}_φ l'ensemble des variables que φ comporte. v_φ est de taille en $\mathcal{O}(|\varphi|)$.

Un algorithme vérificateur : on parcourt l'arbre de la formule en utilisant la valuation pour obtenir les valeurs de vérité pour les variables et on procède récursivement. Moralement, on regarde si la valuation est un modèle.

```
1 |   type formule = X of int | Not of formule | Or of formule*formule
2 |   | And of formule*formule | Top | Bot
3 |
4 |   let rec verifie f (v:bool array) =
5 |       match f with
6 |       | X (i) -> v.(i)
7 |       | Top -> true
8 |       | Bot -> false
9 |       | Or (f1, f2) -> (verifie f1 v) || (verifie f2 v)
10 |      | And (f1, f2) -> (verifie f1 v) && (verifie f2 v)
11 |      | Not f1 -> not (verifie f1 v)
```

Démonstration

Montrons que $\mathbf{SAT} \leq_m^P \mathbf{CNF-SAT}$

On doit définir une fonction g qui transforme une instance φ de \mathbf{SAT} en une instance de $\mathbf{CNF-SAT}$ $g(\varphi)$ telle que :

$$\varphi \text{ satisfiable} \Leftrightarrow g(\varphi) \text{ satisfiable}$$

On décrit un algorithme permettant de passer de φ à $g(\varphi)$:

1. On élimine tous les connecteurs \rightarrow et \leftrightarrow en appliquant les équivalences :

$$\begin{cases} \varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2 \\ \varphi_1 \leftrightarrow \varphi_2 \equiv (\neg\varphi_1 \vee \varphi_2) \wedge (\neg\varphi_2 \vee \varphi_1) \end{cases}$$

Cela s'effectue en $\mathcal{O}(|\varphi|)$, si on considère que la suppression des \leftrightarrow ne duplique pas réellement les variables φ_1 et φ_2 (avec des pointeurs par exemple).

2. On applique ensuite les lois de De Morgan :

```
1 | let rec morgan phi =
2 |   match phi with
3 |   | Not (And (phi1, phi2)) -> Or (morgan (Not phi1), morgan (Not phi2))
4 |   | Not (Or (phi1, phi2)) -> And (morgan (Not phi1), morgan (Not phi2))
5 |   | Not (Top) -> Bot
6 |   | Not (Bot) -> Top
7 |   | Not (Not phi1) -> morgan phi1
8 |   | And (phi1, phi2) -> And (morgan phi1, morgan phi2)
9 |   | Or (phi1, phi2) -> Or (morgan phi1, morgan phi2)
10 |   | _ -> phi
```

Cela s'effectue en $\mathcal{O}(|\varphi|)$. On obtient à l'issue une forme normale négative (\neg uniquement permis devant des variables seules, et uniquement \wedge et \vee).

3. À partir de $\hat{\varphi}$ la formule obtenue à l'étape 2, on construit inductivement $g(\varphi)$. On transforme donc une forme normale négative $f = \hat{\varphi}$ en une forme normale conjonctive F .

- Cas de base : f est un littéral \top ou \perp , on pose $F = f$
- Cas inductifs :
 - $f = f_1 \wedge f_2$. On construit F_1 et F_2 à partir de f_1 et f_2 . On pose $F = F_1 \wedge F_2$.
 - $f = f_1 \vee f_2$. Ça se complique. On construit inductivement F_1 et F_2 , à partir de f_1 et f_2 . On pose $F' = (F_1 \vee y) \wedge (F_2 \vee \neg y)$ avec g qui n'apparaît pas dans F_1 et F_2 . F' n'est pas une FNC. On la transforme en F en distribuant les littéraux y et $\neg y$ dans toutes les clauses de F_1 et F_2 (resp.).

Démonstration

On a alors $F = g(\varphi)$. Montrons que f est satisfiable si et seulement si F est satisfiable. Évidemment, on n'a pas directement $F = f$. On pose \mathcal{V}_F les variables de F , pareil pour f . On a $\mathcal{V}_f \subset \mathcal{V}_F$. On va montrer que

- $\text{Mod}(F) \subset \text{Mod}(f)$ si on restreint aux variables de f les modèles.
- $\text{Mod}(f) \subset \text{Mod}(F)$ si on prolonge aux variables de F les modèles.

On procède par induction : LE DEBUT EST MANQUANT

- Cas de base : $F = f$, donc on a $v \dots$
- – $f = f_1 \vee f_2$. On construit encore F via $F' = (F_1 \vee y) \wedge (F_2 \vee \neg y)$ en distribuant. Soit v un modèle de F , c'est un modèle de F' .
 - * Si $v(y) = F$, nécessairement, v est un modèle de F_1 pour satisfaire $(F_1 \vee y)$. Par hypothèse d'induction, $v|_{\mathcal{V}_f}$ modèle de f_1 . Donc modèle de f .
 - * Si $v(y) = V$, le cas est symétrique.

Soit v un modèle de f .

- * si v est un modèle de f_1 , alors par hypothèse d'induction, on peut prolonger v en v_1 modèle de F_1 . On pose alors $\tilde{v} : y \mapsto F; x \mapsto v_1(x)$ pour $x \neq y$.
- * si v est un modèle de f_2 , le cas est symétrique.

Il reste à justifier que

Photos du vendredi 07/03 à recopier ici.
Récapulatif

Proposition 11.41

3-SAT est un problème NP-complet

Démonstration

Etape 1 : **3-SAT** est de classe NP (*algo vérificateur* avec une valuation comme *certificat* en temps *polynomial*) (les mots-clés sont en italique)

Etape 2 : montrer que **CNF-SAT** \leq_m^P **3-SAT**.

L'idée est la suivante. Convertir $(l_1 \vee \dots \vee l_k)$ en $(l_1 \vee l_2 \vee x_{2,3}) \wedge (\neg x_{2,3} \vee l_3 \vee x_{3,4})$.

On définit une fonction g qui prend une formule φ sous FNC (instance de **CNF-SAT**) et construit une instance de **3-SAT** $g(\varphi)$ qui vérifie :

$$\varphi \text{ satisfiable} \iff g(\varphi) \text{ satisfiable}$$

Soit φ une telle FNC, qu'on écrit :

$$\varphi = \bigwedge_{i=1}^n C_i$$

Pour $i \in \llbracket 1, n \rrbracket$, on construit la formule :

$$F_i = \begin{cases} C_i & \text{si } C_i \text{ est de taille au plus 3.} \\ h(C_i) & \text{sinon} \end{cases}$$

où $h(C_i)$, pour $C_i = (l_{1,i} \vee \dots \vee l_{k,i})$ ($k \geq 4$), on introduit $k - 3$ variables propositionnelles spécifiques à la clause (non déjà présente dans \mathcal{V}_φ), notées $(x_{j,j+1,i})_{j \in \llbracket 2, k-2 \rrbracket}$. Ainsi :

$$F_i = (l_{1,i} \vee l_{2,i} \vee x_{2,3,i}) \wedge (\neg x_{2,3,i} \vee l_{3,i} \vee x_{3,4,i}) \wedge \dots \wedge (\neg x_{k-2,k-1,i} \vee l_{k-1,i} \vee l_{k,i})$$

Ainsi, $|F_i| = \mathcal{O}(|C_i|)$ (on triple la taille).

On pose alors :

$$g(\varphi) = \bigwedge_{i=1}^n F_i$$

Les F_i sont toutes des 3-FNC donc $g(\varphi)$ est une 3-FNC. De plus, $|g(\varphi)| = \mathcal{O}(|\varphi|)$ et est constructible en temps linéaire.

Démonstration

Il reste à montrer que :

$$\varphi \text{ satisfiable} \iff g(\varphi) \text{ satisfiable}$$

- $\boxed{\implies}$: Si φ est satisfiable, soit v un modèle de φ . On souhaite le prolonger sur les nouvelles variables en \tilde{v} de sorte que \tilde{v} satisfasse $g(\varphi)$.

On considère une clause C_i :

- Si $F_i = C_i$, v convient déjà
- Si $F_i \neq C_i$, écrivons $C_i = (l_{1,i} \vee \dots \vee l_{k,i})$.

v satisfait C_i donc v satisfait un littéral $l_{j,i}$ parmi ceux-là. Dans F_i , voir Figure 100.

On peut prolonger v en posant :

$$\tilde{v}(x_{p,p+1,i}) = \begin{cases} V & \text{si } p \leq j-1 \\ F & \text{sinon} \end{cases}$$

\tilde{v} satisfait bien F_i . On applique cela pour tous les F_i et comme les variables sont spécifiques à une clause, il n'y a pas de contradiction (au sens doublon de variable). \tilde{v} satisfait $g(\varphi)$. Donc $g(\varphi)$ est satisfiable.

- $\boxed{\impliedby}$: Supposons que $g(\varphi)$ satisfiable.
Soit v un modèle de $g(\varphi)$. Montrons que v satisfait φ (même s'il est défini sur plus que \mathcal{V}_φ).
On considère une clause C_i :

- Si $F_i = C_i$, v satisfait F_i donc C_i
- Si $F_i \neq C_i$, notons :

$$\begin{aligned} F_i = & (l_{1,i} \vee l_{2,i} \vee x_{2,3,i}) \wedge \\ & \dots \wedge \\ & (\neg x_{j,j+1,i} \vee l_{j+1,i} \vee x_{j+1,j+2,i}) \wedge \\ & \dots \wedge \\ & (\neg x_{k-2,k-1,i} \vee l_{k-1,i} \vee l_{k,i}) \end{aligned}$$

Supposons par l'absurde que v ne satisfait aucun $l(j,i)$.

La première clause est satisfaite donc $v(x_{2,3,i}) = V$ nécessairement. Par propagation de clause en clause, on a $v(x_{j,j+1,i}) = V$ pour tout j , et la dernière clause ne peut être satisfaite dans ces conditions. absurde.

v satisfait donc au moins un littéral l_i donc la clause C_i .

Au totale, v satisfait toutes les clauses de φ , qui est donc satisfiable.

Donc g est bien une réduction polynomiale :

$$\mathbf{CNF-SAT} \leq_m^P \mathbf{3-SAT}$$

Puis **CNF-SAT** est NP-complet. Ainsi **3-SAT**¹⁷ est NP-difficile.

Au total avec ces deux tirets principaux, on a bien que **3-SAT** est NP-complet.

III.4 Chemin hamiltonien

Définition 11.42 - *chemin hamiltonien*

Dans un graphe $G = (S, A)$, un *chemin hamiltonien* de $s \in S$ à $t \in S$ est un chemin de départ s , d'arrivée t passant exactement une fois par chacun des sommets de S .

Définition 11.43 - *cycle hamiltonien*

Dans un graphe $G = (S, A)$, un *cycle hamiltonien* est un chemin hamiltonien d'origine et d'extrémité égales.

On définit deux problèmes

Définition 11.44 - *Problème CHEMIN-HAM*

Instance : Un graphe $G = (S, A)$ orienté, s et t deux sommets de S . **Question** : Existe-t-il un chemin hamiltonien de s à t ?

Définition 11.45 - *Problème CYCLE-HAM*

Instance : Un graphe $G = (S, A)$ orienté **Question** : Existe-t-il un cycle hamiltonien dans G ?

Proposition 11.46 - *concernant ces deux problèmes*

CHEMIN-HAM et **CYCLE-HAM** sont NP-Complets.

Démonstration

Les mots-clés sont en italique.

- Montrons qu'ils sont de classe NP. On choisit comme ensemble des *certificats* les suites des sommets. On peut construire un *algorithme vérificateur* de complexité en la taille de la liste des sommets qui vérifie que chaque sommet du graphe apparaît une et une seule fois (avec un tableau de booléens), qu'il y a bien des arêtes entre deux sommets consécutifs (cela se fait en $\mathcal{O}(1)$ avec des matrices d'adjacence) et éventuellement que l'on a le bon départ, la bonne arrivée.
Moralement, on vérifie que c'est bien un cycle hamiltonien ou un chemin hamiltonien.
Pour une instance positive, il existe un bon certificat de taille $|S|$ qui permet de renvoyer Vrai avec l'algorithme vérificateur en *temps polynomial*.
Pour une instance négative, l'algorithme renvoie bien sûr toujours faux.
- Montrons maintenant que $\mathbf{3-SAT} \leq_m^P \mathbf{CYCLE-HAM}$ et $\mathbf{3-SAT} \leq_m^P \mathbf{CHEMIN-HAM}$.
Pour φ une 3-FNC, on cherche à construire le graphe G_φ tel que :

$$\varphi \in \mathbf{3-SAT}^+ \iff (G_\varphi, s, t) \in \mathbf{CHEMIN-HAM}^+$$

$\mathbf{3-SAT}^+$ est l'ensemble des instances positives du problème $\mathbf{3-SAT}$.

De même on cherche un graphe $G_{\varphi'}$ tel que :

$$\varphi \in \mathbf{3-SAT}^+ \iff G_{\varphi'} \in \mathbf{CYCLE-HAM}^+$$

G_φ et $G_{\varphi'}$ doivent être de taille polynomiale en $|\varphi|$ et constructible en temps polynomial.

On pose :

$$\varphi = \bigwedge_{i=1}^k C_i$$

Avec pour $i \in \llbracket 1, k \rrbracket$, $C_i = (l_{1,i} \vee l_{2,i} \vee l_{3,i})$ On pose le graphe gadget suivant : (Figure 101).

Ce graphe gadget a la particularité de présenter exactement deux chemins hamiltoniens de s à t :

- $s \rightarrow g \rightarrow \dots \rightarrow d \rightarrow t$
- $s \rightarrow d \rightarrow \dots \rightarrow g \rightarrow t$

Démonstration

On pose alors $\mathcal{V}_\varphi = \{x_1, \dots, x_n\}$ l'ensemble des variables qui vont apparaître dans φ .

On construit le graphe $G_{\mathcal{V}_\varphi}$ en utilisant n fois ce graphe gadget : une fois pour chaque variable ! (Voir figure 102).

$G_{\mathcal{V}_\varphi}$ a exactement 2^n chemins hamiltoniens (deux possibilités à chaque fois) de s à t , auxquels on peut faire correspondre une valuation du \mathcal{V}_φ .

À un chemin hamiltonien de s à t , on fait correspondre une valuation v telle que pour $i \llbracket 1, n \rrbracket$:

$$v(x_i) = \begin{cases} V & \text{si } g_i \text{ apparaît avant } d_i \text{ dans le chemin hamiltonien} \\ F & \text{sinon} \end{cases}$$

Pour chaque classe C_j , on rajoute un sommet en fonction des cas, comme dans la figure 103.

On obtient un graphe $G_\varphi = (S_\varphi, A_\varphi)$, avec :

$$\begin{cases} |S_\varphi| = \underbrace{k}_{\text{clauses}} + \underbrace{2}_{s,t} + \underbrace{n(2k+4)}_{\text{gadget}} & \text{bien polynomiale en } |\varphi| \\ |A_\varphi| \leq \underbrace{(2k+3)}_{\text{gadget}} 2n + \underbrace{n+1}_{\text{lien entre les gadget}} + \underbrace{6k}_{\text{clauses}} & \text{polynomiale en } |\varphi|. \end{cases}$$

G_φ est bien constructible en temps polynomial.

On construit $G_{\varphi'}$ à partir de G_φ en rajoutant une arête de t à s . **C'est la seule chose qui les sépare, d'où l'intérêt de factoriser les démos** Montrons que :

$$\begin{aligned} \varphi &\in \mathbf{3-SAT}^+ \\ \iff (G_\varphi, s, t) &\in \mathbf{CHEMIN-HAM}^+ \\ \iff G_{\varphi'} &\in \mathbf{CYCLE-HAM}^+ \end{aligned}$$

On doit la dernière équivalence au fait que (t, s) est nécessaire dans le cycle. On montre la première.

- $\boxed{\implies}$: Si φ est satisfiable, soit v un modèle de φ . On s'intéresse au chemin hamiltonien dans $G_{\mathcal{V}_\varphi}$. Pour chaque clause C_j , on a au moins un littéral satisfait, on transforme le chemin hamiltonien dans $G_{\mathcal{V}_\varphi}$ en chemin hamiltonien dans G_φ en prenant le détour vers C_j correspondant au littéral satisfait.
- Si $(G_\varphi, s, t) \in \mathbf{CHEMIN-HAM}^+$, on considère la valuation correspondant au chemin hamiltonien existant. on a au moins un détour pris par clause donc chacune est satisfiable.