NP-complétude

François Schwarzentruber

ENS Rennes

Table des matières

1 Problèmes de décision dits "de recherche" [1]

1.1 Mon premier exemple

Exemple 1 ***3-COLORATION***

— *entrée* : *Un graphe $G = (S, A)$ non orienté ;*

— *sortie* : *oui s'il existe une fonction $c : S \to \{$  $\}$ qui est un 3-coloriage, c'est à dire telle que pour tout $(s,t) \in A$, $c(s) \neq c(t)$; non sinon.*

Vocabulaire
— Instance : graphe $G = (S, A)$ non orienté ;
— Certificat : une fonction $c : S \to \{$  $\}$.

1.2 Définition informelle

Ce sont des problèmes de décision d'existence d'une solution où :
— La recherche d'une solution semble difficile ;
— mais la vérification est facile (en temps polynomial).
Ils peuvent être mis sous la forme suivante :

A
— entrée : une **instance** i ;
— sortie : oui s'il existe un **certificat** c telle que $\mathcal{P}(i, c)$ est vraie ; non sinon.

où vérifier si $\mathcal{P}(i, c)$ s'effectue en temps polynomial en $|i|$.

1.3 Transformer un problème d'optimisation en problème de décision

On transforme un **problème d'optimisation** en **problème de décision** en ajoutant une valeur seuil dans l'instance.

Exemple 2

SAC-A-DOS-OPTIMISATION

— *entrée* :
  — *Des valeurs $(v_1, \ldots, v_n)$;*
  — *des poids $(p_1, \ldots, p_n)$;*
  — *un poids maximal P ;*
— *sortie : la valeur maximale de $\sum_{i=1}^{n} x_i v_i$ telle que $\sum_{i=1}^{n} x_i p_i \leq P$ avec $(x_1, \ldots, x_n) \in \{0,1\}^n$.*

$\wr$

SAC-A-DOS

— *entrée* :
  — *Des valeurs $(v_1, \ldots, v_n)$;*
  — *des poids $(p_1, \ldots, p_n)$;*
  — *un poids maximal P ;*
  — *une valeur seuil V ;*
— *sortie : oui s'il existe $(x_1, \ldots, x_n) \in \{0,1\}^n$ telle que $\sum_{i=1}^{n} x_i v_i \geq V$ et $\sum_{i=1}^{n} x_i p_i \leq P$.*

1. terminologie de [DPV06]

2 Classe P

Définition 1 (classe P)
La classe **P** est la classe des problèmes de décision **A** tel qu'il existe
un algorithme déterministe A
une machine de Turing déterministe A
tel(le) que
- i est une instance positive ssi l'exécution $A(i)$ est acceptante ;
- et il existe un polynôme f telle que l'exécution $A(i)$ est de longueur au plus $f(|i|)$.

Thèse Cobham–Edmonds[AB09] : problème facile = problème dans **P**

Exemple 3 *Le problème de décision suivant est dans* ***P*** *[Cor09] :*
PLUS-LONGUE-SOUS-SUITE-COMMUNE
- *entrée* : *Deux mots $w, u \in \Sigma^*$, un entier k ;*
- *sortie* : *oui s'il existe un mot z telle que z est sous-suite commune à w et u de longueur $\geq k$; non sinon.*

On arrive à montrer que certains problèmes de recherche sont dans **P** en rusant pour la recherche d'une solution.

Problèmes sur les graphes.
- Satisfiabilité d'un ensemble de 2-clauses en logique propositionnelle [DPV06] (calcul des fortement connexes) ;

Gloutons.
- Satisfiabilité d'un ensemble de clauses de Horn en logique propositionnelle [DPV06] ;
- Arbre couvrant minimum [DPV06]

Flots.
- Couplage maximal ;
- Elimination d'une équipe au baseball.

Programmation dynamique.
- Plus longue sous-suite commune [Cor09] ;
- Appartement d'un mot au langage engendré par une grammaire algébrique.

Programmation linéaire (réelle).
- Maximiser la vente de chocolats.

3 EXPTIME

Définition 2 (classe EXPTIME)
La classe **EXPTIME** est la classe des problèmes de décision **A** tel qu'il existe
un algorithme déterministe A
une machine de Turing déterministe A
tel(le) que
- i est une instance positive ssi l'exécution $A(i)$ est acceptante ;
- et il existe un polynôme f telle que l'exécution $A(i)$ est de longueur au plus $2^{f(|i|)}$.

Proposition 1 ***3-COLORIAGE*** *est dans* ***EXPTIME****.*

Démonstration.
Voici un algorithme qui décide **3-COLORIAGE** en temps exponentiel :

procedure 3col(G)
  for all $c : S \to \{$  , ,  $\}$ **do**
    if c est un 3-coloriage **then**
      accepter
  rejeter

■

Mais la classe **EXPTIME** semble trop grosse pour caractériser nos problèmes de recherche.

4 Classe NP

Définissons la classe de complexité qui correspond à nos problèmes de recherche. Modélisons la recherche de certificat avec un **jeu à un joueur**.

4.1 Jeu à un joueur = algorithme non-déterministe

Exemple 4 *Voici une instance de* ***3-COLORATION****. C'est une instance positive si et seulement si vous avez une stratégie gagnante au jeu suivant :*

procedure *3-coloration(G)*
  for $s \in S$ ***do***
    choisir $c[s]$ *dans* {  } ;
  if *c est un 3-coloriage* ***then***
    accepter *(gagné)*
  else
    rejeter *(perdu)*

4.2 Définition de la classe NP

Définition 3 (classe NP)
La classe **NP** est la classe des problèmes de décision **A** tel qu'il existe

une description d'un jeu A à un joueur
un algorithme non-déterministe A
une machine de Turing non-déterministe A

tel que

- i est une instance positive ssi vous avez une stratégie gagnante dans le jeu $A(i)$ / A accepte i ;
- et il existe un polynôme f telle que

  toutes les parties du jeu $A(i)$ sont de longueur au plus $f(|i|)$
  toutes les exécutions de $A(i)$ sont de longueur au plus $f(|i|)$

.

4.3 Définition alternative : vérifieur par certificat

Un vérifieur pour un problème de décision **A** est un algorithme déterministe $V(.,.)$ tel que i est une instance positive de **A** ssi il existe c tel que $V(i,c)$ est une exécution acceptante.

Définition 4 (classe NP, définition alternative)
La classe **NP** est la classe des problèmes **A** tel qu'il existe un vérifieur V pour un problème de décision **A** et un polynôme f tels que

- pour toute instance i, pour tout certificat c, la longueur de l'exécution $V(i,c)$ est majoré par $f(|i|)$.

5 Un problème ouvert à craquer : P = NP ?

NP
Est-ce que tout problème décidé par un algorithme **non-déterministe en temps polynomial**...

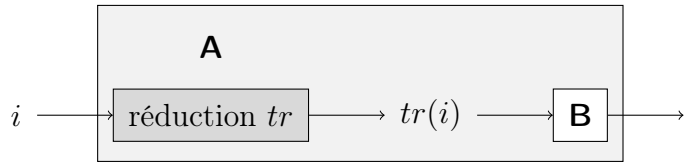
$\subseteq$?

P
l'est aussi par un algorithme **déterministe en temps polynomial** ?

Démarche pour le craquer : s'intéresser aux problèmes les plus durs de la classe **NP**.

5.1 Réduction polynomiale

Définition 5 (Réduction)
Une **réduction polynomiale** d'un problème **A** à un problème **B** est une fonction tr calculable en temps polynomial telle que pour toute instance i de **A**, i est instance positive **A** ssi $tr(i)$ est instance positive **B**.



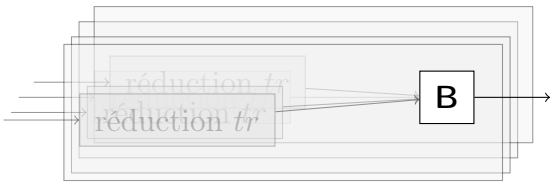
On dit que **A** se réduit en temps polynomial à **B** s'il existe une réduction polynomiale de **A** à **B**. Intuitivement, si **A** se réduit en temps polynomial **B**, alors **B** est plus dur que **A**.

Théorème 1 *Si **A** se réduit en temps polynomial **B**, alors :*
- *$\boldsymbol{B} \in P$ implique $\boldsymbol{A} \in P$;*
- *$\boldsymbol{B} \in NP$ implique $\boldsymbol{A} \in NP$.*

Démonstration.
Le schéma ci-dessus donne un algorithme en temps polynomial pour **A**. ■

5.2 NP-dureté

Définition 6 (NP-dur)
Un problème **B** est **NP-dur** si pour tout problème **A** $\in$ **NP**, **A** se réduit en temps polynomial à **B**.



Définition 7 (NP-complet)
Un problème est **NP-complet** s'il est dans **NP** et **NP**-dur.

5.3 Notre problème ouvert

Proposition 2 *S'il existe un problème **NP**-dur qui est décidé par un algorithme en temps polynomial alors **P** = **NP**.*

Démonstration.
Supposons qu'il existe un problème **NP**-dur **B** décidé en temps polynomial. Montrons que **NP** $\subseteq$ **P**. Soit **A** un problème dans **NP**. **A** se réduit en temps polynomial à **B**. Le schéma ci-dessus donne un algorithme en temps polynomial. ■

Par ailleurs, si on peut montrer qu'un problème n'admet pas d'algorithme (déterministe) polynomial, c'est certainement un problème **NP**-complet...

6 Mon premier problème NP-complet : SAT

SAT
- entrée : Une formule φ de la logique propositionnelle ;
- sortie : oui si φ est satisfiable, c'est à dire il existe une valuation ν telle que $\nu \models \varphi$; non sinon.

CNF-SAT
- entrée : Une formule φ de la logique propositionnelle en forme normale conjonctive ;
- sortie : oui si φ est satisfiable ; non sinon.

Théorème 2 (de Cook) *SAT et CNF-SAT sont NP-complets.*

Proposition 3 *Si* $\mathbf{SAT} \in \mathbf{P}$ *alors* $\mathbf{P} = \mathbf{NP}$.

6.1 Dans NP

Théorème 3 *SAT est dans NP.*

Démonstration.

Voici un algorithme non-déterministe qui décide **SAT** en temps polynomial.

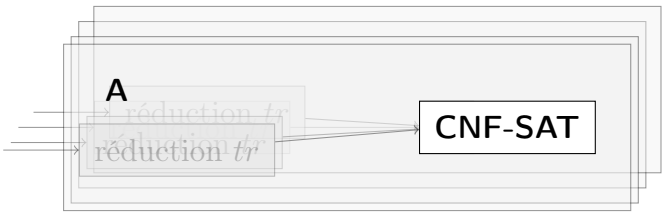
```
procedure sat(φ)
    for all proposition atomique p dans φ do
        choisir ν[p] dans {faux, vrai} ;
    if ν ⊨ φ then
        accepter (gagné)
    else
        rejeter (perdu)
```

■

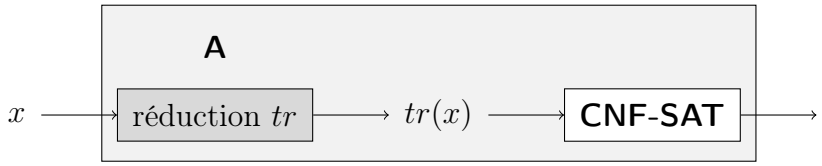
6.2 NP-dur

Théorème 4 *CNF-SAT est NP-dur.*

Démonstration.
On montre que tout problème dans **NP** s'y réduit en temps polynomial. Soit $\mathbf{A} \in \mathbf{NP}$.



On va donner une réduction polynomiale tr tel que x est une instance positive de **A** ssi $tr(x)$ est une formule satisfiable.



Soit $M = (\Sigma, \Gamma, Q, \delta, q_0, q_{acc})$ une machine de Turing qui décide **A**. Sans perte de généralité, on suppose que l'état final acceptant q_{acc} est un état qui boucle sur n'importe quelle configuration du ruban. Il existe un polynôme f tel que, sur une entrée x, toutes les exécutions de la machine de Turing sont de longueur au plus $f(|x|)$. De la même façon, seules les $f(|x|)$ cases du ruban sont pertinentes car la tête de lecture n'a pas eu le temps d'aller au delà de la $f(|x|)^{\text{ème}}$ case. On définit la réduction tr transforme une **A**-instance x en une formule $tr(x)$ qui exprime

> "il existe une exécution acceptante de M sur le mot x en temps $f(|x|)$ et et qui n'utilise que les $f(|x|)$ premières cases du ruban".

Soit $\mathcal{C} = \{0, \ldots, f(|x|)\}$. Pour écrire $tr(x)$, on introduit les propositions atomiques suivantes :

au temps t, l'état est q — au temps t, la position du curseur est i — au temps t, la case n° i contient a — au temps t, on tire la transition τ

où $t, i \in \mathcal{C}$, $q \in Q$, $a \in \Gamma$ et $\tau \in \delta$.

Définissons à présent $tr(x)$ comme la conjonction des formules suivantes.

Unicité et existence des valeurs.

1. $\bigwedge_{t\in\mathcal{C}} \bigvee_{q\in Q}$ au temps t, l'état est q — La machine est dans un état à tout instant t

2. $\bigwedge_{t\in\mathcal{C}} \bigwedge_{q,q'\in Q|q\neq q'} \left(\neg \text{au temps } t, \text{ l'état est } q \vee \neg \text{au temps } t, \text{ l'état est } q'\right)$ — La machine n'est jamais dans deux états à la fois

3. $\bigwedge_{t\in\mathcal{C}} \bigvee_{i\in\mathcal{C}}$ au temps t, la position du curseur est i — Le curseur est positionné quelque part à tout instant t

4. $\bigwedge_{t\in\mathcal{C}} \bigwedge_{i,i'\in\mathcal{C}|i\neq i'} \left(\neg \text{au temps } t, \text{ la position du curseur est } i \vee \neg \text{au temps } t, \text{ la position du curseur est } i'\right)$ — Le curseur n'a jamais deux positions différentes

5. $\bigwedge_{t\in\mathcal{C}} \bigwedge_{i\in\mathcal{C}} \bigvee_{a\in\Sigma}$ au temps t, la case n° i contient a — À tout instant, toute case du ruban contient une lettre

6. $\bigwedge_{t\in\mathcal{C}} \bigwedge_{a,b\in\Sigma|a\neq b} \left(\neg \text{au temps } t, \text{ la case n° } i \text{ contient } a \vee \neg \text{au temps } t, \text{ la case n° } i \text{ contient } b\right)$ — Une case ne contient au plus qu'une lettre

7. $\bigwedge_{t\in\mathcal{C}} \bigvee_{\tau\in\delta}$ au temps t, on tire la transition τ — 'A tout instant t, on tire une transition pour aller vers l'instant $t+1$

8. $\bigwedge_{\tau,\tau'\in\delta|\tau\neq\tau'} \left(\neg \text{au temps } t, \text{ on tire la transition } \tau \vee \neg \text{au temps } t, \text{ on tire la transition } \tau'\right)$ — On ne tire jamais plus d'une transition

---

Configuration initiale

9. au temps 0, la case n° 0 contient $\sqcup$ $\wedge$ au temps 0, la case n° 1 contient x_1 $\wedge \ldots$ au temps 0, la case n° n contient x_n $\wedge$ au temps 0, la case n° $n+1$ contient $\sqcup$ $\wedge \ldots$ au temps 0, la case n° $f(|x|)$ contient $\sqcup$ — À l'instant 0, le ruban contient $\sqcup x_1 \ldots x_n \sqcup \ldots \sqcup$

10. au temps 0, l'état est q_0 $\wedge$ au temps 0, la position du curseur est 1

À l'instant 0, la machine est dans l'état initial q_0 à l'instant 0 et le curseur est à la position 1.

Exécution acceptante

11. $\bigvee_{t\in\mathcal{C}}$ au temps t, l'état est q_{acc}

La machine atteint l'état d'acceptation q_{acc}

Exécution des transitions

12. $\bigwedge_{t\in\mathcal{C}}\bigwedge_{i\in\mathcal{C}}\bigwedge_{a\in\Sigma}\left(\left(\neg\left(\text{au temps } t, \text{ la position du curseur est } i\right)\wedge\left(\text{au temps } t, \text{ la case n° } i \text{ contient } a\right)\right)\rightarrow\left(\text{au temps } t+1, \text{ la case n° } i \text{ contient } a\right)\right)$

on ne change pas le contenu du ruban si le curseur n'y est pas

13. $\bigwedge_{t\in\mathcal{C}\setminus\{f(|x|)\}}\bigwedge_{(q,a,e,b,d)\in\delta}\left(\text{au temps } t, \text{ on tire la transition } q,a,e,b,d\rightarrow\text{au temps } t, \text{ l'état est } q\right)$

14. $\bigwedge_{t\in\mathcal{C}\setminus\{f(|x|)\}}\bigwedge_{(q,a,e,b,d)\in\delta}\bigwedge_{i\in\mathcal{C}}\left[\left(\text{au temps } t, \text{ on tire la transition } q,a,e,b,d\wedge\text{au temps } t, \text{ la position du curseur est } i\right)\rightarrow\text{au temps } t, \text{ la case n° } i \text{ contient } a\right]$

15. $\bigwedge_{t\in\mathcal{C}\setminus\{f(|x|)\}}\bigwedge_{(q,a,e,b,d)\in\delta}\left(\text{au temps } t, \text{ on tire la transition } (q,a,e,b,d)\rightarrow\text{au temps } t+1, \text{ l'état est } e\right)$

16. $\bigwedge_{t\in\mathcal{C}\setminus\{f(|x|)\}}\bigwedge_{(q,a,e,b,d)\in\delta}\bigwedge_{i\in\mathcal{C}}\left[\left(\text{au temps } t, \text{ on tire la transition } (q,a,e,b,d)\wedge\text{au temps } t, \text{ la position du curseur est } i\right)\rightarrow\text{au temps } t+1, \text{ la case n° } i \text{ contient } b\right]$

17. $\bigwedge_{t\in\mathcal{C}\setminus\{f(|x|)\}}\bigwedge_{(q,a,e,b,d)\in\delta}\bigwedge_{i\in\mathcal{C}|i+d\in\mathcal{C}}\left(\text{au temps } t, \text{ on tire la transition } (q,a,e,b,d)\wedge\text{au temps } t, \text{ la position du curseur est } i\right)\rightarrow\text{au temps } t+1, \text{ la position du curseur est } i+d$

La formule $tr(x)$ est bien en forme normale conjonctive. Aussi, on peut la calculer en temps polynomial en $|x|$ à partir de x . On a bien x est une instance positive de **A** ssi $tr(x)$ est une formule satisfiable.

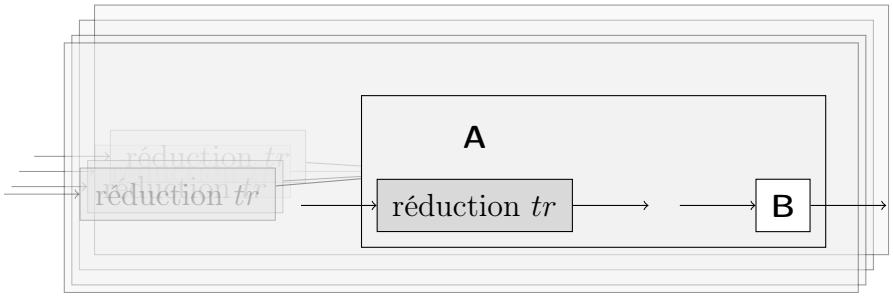
$\boxed{\Rightarrow}$ Soit x une instance positive de **A**. Alors il existe une exécution de M acceptante sur le mot x en temps $f(|x|)$ et sur une longueur de ruban de $f(|x|)$. On construit une valuation qui satisfait $tr(x)$.

$\boxed{\Leftarrow}$ Si $tr(x)$ est satisfiable. Soit ν une valuation qui satisfait $tr(x)$. On construit à partir de ν une exécution de M acceptante sur le mot x. Donc le mot x est une instance positive de **A**.
■

7 Réductions polynomiales pour montrer qu'un problème est NP-dur

Proposition 4 *Si* **A** *se réduit polynomialement à* **B** *et que* **A** *est NP-dur alors* **B** *est NP-dur.*

Démonstration.



■

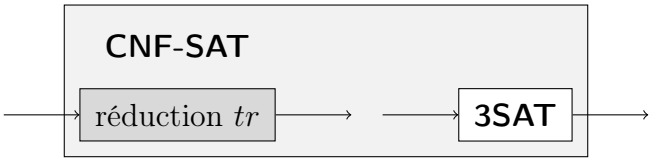
7.1 3-SAT

3-SAT
- entrée : Une formule φ de la logique propositionnelle en 3-forme normale conjonctive ;
- sortie : oui si φ est satisfiable ; non sinon.

Proposition 5 ***3-SAT*** *est NP-dur.*

Démonstration.

Nous allons réduire polynomialement **CNF-SAT** à **3SAT**.



Si φ est une forme normale conjonctive, $tr(\varphi)$ est obtenue à partir de φ en remplaçant chaque clause par un ensemble de 3-clauses en introduisant des variables supplémentaires.

Exemple 5 $(a \vee b \vee c \vee d \vee e) \quad \rightsquigarrow \quad (a \vee b \vee \alpha) \wedge (\neg\alpha \vee c \vee \beta) \wedge (\neg\beta \vee d \vee e)$.

Montrons que φ est satisfiable ssi $tr(\varphi)$ satisfiable.

$\boxed{\Leftarrow}$ Supposons que φ est vraie pour une certaine valuation. En particulier, chaque clause $(a \vee b \vee c \vee d \vee e)$ est vraie. Montrons que l'on peut donner des valeurs aux variables intermédiaires de sorte que $(a \vee b \vee \alpha) \wedge (\neg\alpha \vee c \vee \beta) \wedge (\neg\beta \vee d \vee e)$ soit vraies. L'une des variables a, b, c, d, e est à vraie. Par exemple, c est à vraie. Il suffit de mettre les premières variables intermédiaires à vraies jusqu'à être dans la 3-clause où apparaît c. Ici : α à vraie. Puis on met les variables intermédiaires suivantes à faux.

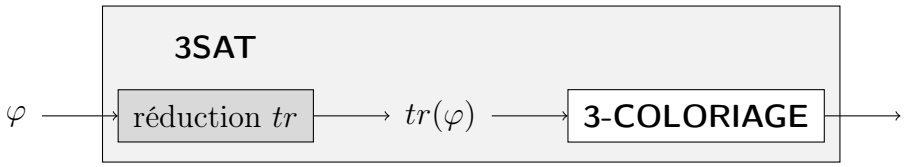
$\boxed{\Rightarrow}$ Supposons que $tr(\varphi)$ soit satisfiable pour une certaine valuation. En particulier, chaque sous-conjunction de clauses $(a \vee b \vee \alpha) \wedge (\neg\alpha \vee c \vee \beta) \wedge (\neg\beta \vee d \vee e)$, obtenu à partir d'une clause de l'instance de SAT est vraie. Montrons que l'une des variables a, b, c, d, e est mise à vraie. Par l'absurde, supposons que toutes les variables a, b, c, d, e sont à faux. On a alors $\alpha, \beta, \neg\beta$ à vraie. Contradiction.

■

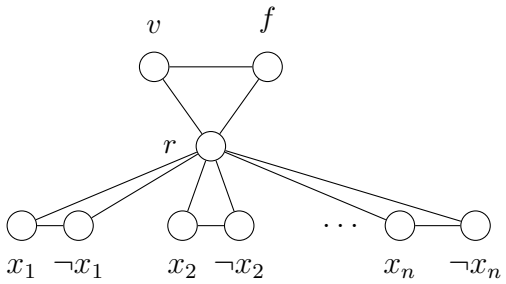
7.2 3-coloration

Théorème 5 ***3-COLORIAGE*** *est NP-dur.*

Démonstration.



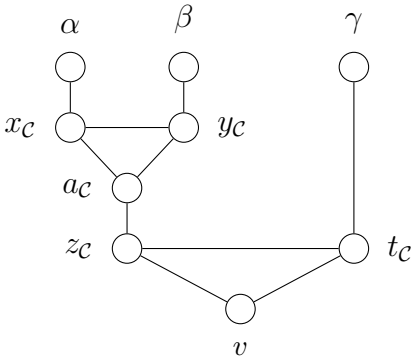
On définit une réduction tr qui à toute **3SAT**-instance φ associe une **3-COLORIAGE**-instance $tr(\varphi)$. $tr(\varphi)$ est un graphe basé sur le **gadget** suivant :



où $x_1, x_2, \ldots, x_n$ sont les propositions atomiques qui apparaissent dans φ. On code le fait qu'un littéral est vrai par le fait qu'il a la couleur du sommet v et faux s'il a la couleur du sommet f. La couleur de $\neg x$ est toujours différente de celle de x.

Ensuite, pour chaque 3-clause $(\alpha \vee \beta \vee \gamma)$ de φ, $tr(\varphi)$ contient aussi le gadget à ci-dessous. Pour coder une 2-clause $\alpha \vee \beta$, on prend le même gadget mais avec f à la place de γ.

Lemme 1 *Dans tout coloriage, l'un des sommets α, β, γ a avoir la couleur de v.*



Démonstration.
Par l'absurde, supposons qu'ils ont tous la couleur de f (on rappelle qu'ils ne peuvent pas avoir la couleur de r car la structure initiale ne le permet pas). Dans ce cas, comme γ est de la couleur de f, pour sûr t_C est de la couleur d'r donc z_C est de la couleur de f. D'autre part, comme α et β sont de la couleur de f, pour sûr, x_C et y_C ne sont pas de couleur de f donc a_C est de couleur de f. Contradiction. ■

Lemme 2 *On peut compléter tout coloriage où les sommets α, β, γ sont de la couleur de f ou v et l'un d'eux est de la couleur de v.*

Démonstration.
Faire tous les cas. ■

Lemme 3 *φ satisfiable ssi $tr(\varphi)$ est 3-coloriable.*

Démonstration.
$[\Rightarrow]$ Supposons que φ est satisfiable et soit ν une valuation telle que $\nu \models \varphi$. On colorie les noeuds de la façon suivante :

— $c[r] =$; $c[v\,] =$; $c[f\,] =$;

— $c[p] =$ si $\nu[p] = 1$; sinon.

— $c[\neg p] =$ si $\nu[p] = 1$; sinon.

Par le lemme 2, on compléte le coloriage pour tous les gadgets. Donc $tr(\varphi)$ est 3-coloriable.
$[\Leftarrow]$ Supposons que $tr(\varphi)$ est 3-coloriable. On construit la valuation ν :
— $\nu[p_i] = 1$ si p_i est colorié avec la couleur de v : 0 sinon.
Par le lemme 1, $\nu \models \varphi$. ■ ■

8 Exemples de problèmes NP-complets

Voyageur de commerce	Faire des trous dans une taule en déplaçant le moins possible le forêt
Arbres de Steiner	Connexion de circuits
k-coloration de graphes si $k \geq 3$	Fréquences radio
Sac à dos	Cryptosystème de Merkle–Hellman (vulnérable)

21 problèmes de Karp : [Kar72], [DPV06] Autre référence : [GJ79]

9 En pratique

Même s'il n'existe pas d'algorithmes polynomiaux connus à ce jour, il existe des techniques en temps exponentielle dans le pire cas mais efficaces en pratique.

9.1 Branch and bound

Exemple du voyageur de commerce : lire [DPV06].

9.2 Algorithmes d'approximation

Lire le chapitre correspondant dans [DPV06] ou la référence [Vaz13]. On a des fois des schémas d'approximation en temps polynomial. Par exemple, pour le voyageur de commerce, il existe un algorithme *vdcapprox* tel que $vdcapprox(G, \epsilon)$ retourne un voyage de longueur $\geq \ell_{opt} + \epsilon$ où ℓ_{opt} est la longueur optimale, en temps polynomial en G et ϵ.

9.3 Réduction à SAT

Exemple 6 *logimage, sudoku, etc.*

— Compétitions SAT : `http://www.satcompetition.org/`
— Outils maison : `http://people.irisa.fr/Francois.Schwarzentruber/dpll_demo/`

9.4 Réduction à la programmation linéaire entière

`https://www.gnu.org/software/glpk/`

10 Supposons que P $\neq$ NP...

Soit **NPI** les problèmes de **NP** de difficulté "intermédiaire" : ils sont dans **NP** mais ni **NP**-complet, ni dans **P** [GJ79][p. 154].

Théorème 6 *Si* $\boldsymbol{P \neq NP}$*, alors* ***NPI*** *est non vide.*

Ce théorème provient du lemme suivant.

Lemme 4 *[Lad75] Soit B récursif tel que $B \notin P$. Il existe $D \in P$ tel que :*
— $D \cap B \notin P$ *;*
— $D \cap B$ *se réduit à B (trivial) en temps polynomial ;*
— B *ne se réduit pas à $D \cap B$ en temps polynomial.*

Exemple 7 *Problèmes dont on pense qu'ils sont dans* ***NPI*** *:*

ISOMORPHISME DE GRAPHES

- *entrée :* Deux graphes G_1, G_2
- *sortie :* oui si G_1 et G_2 sont isomorphes ; non, sinon.

FACTORISATION D'ENTIERS

- *entrée :* un entier n, un entier $1 < m < n$
- *sortie :* oui s'il existe un facteur $d \in \{1, \ldots, m\}$ de n ; non, sinon.

Contrairement à ce qui est dit dans [GJ79][p. 154], la primalité [AKS04] et la programmation linéaire ([Kha80], [Wri05]) ont été montrés dans **P**.

Références

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity : a modern approach*. Cambridge University Press, 2009.

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of mathematics*, pages 781–793, 2004.

[Cor09] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[DPV06] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Virkumar Vazirani. Algorithms. 2006.

[GJ79] Michael R Garey and David S Johnson. Computers and intractability : a guide to np-completeness, 1979.

[Kar72] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[Kha80] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1) :53–72, 1980.

[Lad75] Richard E Ladner. On the structure of polynomial time reducibility. *Journal of the ACM (JACM)*, 22(1) :155–171, 1975.

[Vaz13] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[Wri05] Margaret Wright. The interior-point revolution in optimization : history, recent developments, and lasting consequences. *Bulletin of the American mathematical society*, 42(1) :39–56, 2005.