NP-complétude

Alphabets, Langages

- Alphabet Σ: ensemble fini.
- Σ^n: les mots de longueur n.
- $w = a_1 a_2 \dots a_n \in \Sigma^n$, $|w|$ = n: longueur du mot w.
- ϵ: mot vide. $|\epsilon| = 0$.
- $\Sigma^* = \cup_{i \in \mathbb{N}} \Sigma^i$: les mots sur l'alphabet Σ.
- Concaténation de $w, w' \in \Sigma^*$ notée ww'.
  $|ww'| = |w| + |w'|$.

Changement d'alphabet

- On considére dorénavant des alphabets ayant au moins 2 lettres.

- Etant donnés un alphabet Σ et un alphabet Γ, il est possible de coder les mots de Σ^* en mots de Γ^*.
- Exemple: $\Sigma = \{a, b, c\}$, $\Gamma = \{0, 1\}$, on définit $codage_{\Sigma\to\Gamma} : \Sigma^* \to \Gamma^*$ par $codage_{\Sigma\to\Gamma}(a) = 00$, $codage_{\Sigma\to\Gamma}(b) = 01$, $codage_{\Sigma\to\Gamma}(c) = 10$, et $codage_{\Sigma\to\Gamma}(w.w') = codage_{\Sigma\to\Gamma}(w).codage_{\Sigma\to\Gamma}(w')$.

- Remarque: $codage_{\Sigma\to\Gamma}(w)$ se calcule en temps polynomial en $|w|$.

Codage de couples et n-uplets

Soit Σ l'alphabet. Soit $\Sigma' = \Sigma \cup \#$ avec $\# \notin \Sigma$.

- Un couple de mots $(w, w') \in \Sigma^* \times \Sigma^*$ peut se coder par le mot, dénoté par $< w, w' >$, définit par $< w, w' >= codage_{\Sigma' \to \Sigma}(w.\#.w')$.
- Un n-uple, pour $n > 2$ de mots $(w^1, \ldots, w^n) \in (\Sigma^*)^n$ peut se coder récursivement par le mot, noté $< w^1, \ldots, w^n >$ définit par $< w^1, \ldots, w^n >=< w^1, < w^2, \ldots, w^{n-1} >>$

- Remarques:
  - La fonction qui à $w^1, \ldots, w^n$ associe $< w^1, \ldots, w^n >$ se calcule bien en temps polynomial en $|w^1|, \cdots, |w^n|$.
  - Les fonctions qui à $w =< w^1, \ldots, w^n >$ associe w^i se calculent bien en temps polynomial en $|w|$.

Codages

- Un entier: se code en binaire sur $\Sigma = \{0, 1\}$.
- Un tableau T: se code par $< n, T[1], \ldots, T[n] >$, où n est la taille du tableau, et $T[i]$ désigne le codage du $i$ème élément.
- Une liste peut se coder de la même façon.
- Un graphe $G = (V, E)$ peut se coder par:
  - sa liste d'adjacence: $< n, L[1], \ldots, L[n] >$, où $L[i]$ est le codage de la liste des sommets adjacents au sommet numéro i (chaque sommet est codé par son numéro en binaire).
  - sa matrice d'adjacence: $< n, M[1,1], M[1,2], \ldots, M[n,n] >$.
- Un sommet d'un graphe peut se coder par son numéro en binaire.

Abstraction par rapport au codage

- On peut transformer en un temps polynomial le codage d'un graphe sous forme d'une liste d'adjacence en son codage sous forme de matrice d'adjacence et réciproquement.
- Conséquence: un algorithme polynomial pour l'un des codages est polynomial pour l'autre, quitte à traduire la représentation du graphe en ce codage.

- Le nombre de sommets d'un graphe est polynomial en la longueur de son codage (quel qu'il soit).
- Conséquence: un algorithme polynomial en le nombre de sommets d' un graphe est polynomial en son codage.

Problèmes de décisions = Langages

- Tout problème de décision Pb peut se formuler de la façon suivante: Problème Pb:
  - Instance: w un mot
  - Question: Est-ce que $w \in L$

  pour un certain langage L.
- Exemple: pour $REACH$, $L = \{< G, u, v > | G$ code un graphe et il y a un chemin entre u et v dans $G\}$.
- En général, on confondra L et Pb: on note $w \in Pb$ pour "Pb répond "vrai" sur l'instance w.

Abs. par rapp. au modèle de calcul

- Algorithme: Programme en $basic$, C, $pascal$, $logo$, $Java$, $machine\ de\ Turing$ ...
- Résultat fondamental: Un programme A_1 en langage 1 se traduit en un programme A_2 en langage 2 avec

$$TempsExecution(A_2) = p_{1\to 2}(TempsExecution(A_1))$$

pour un polynôme $p_{1\to 2}$.

- Conséquences: La notion d'algorithme "polynomial" est indépendant du langage/modèle de calcul utilisé.

Classes P et NP

- Classe P: classe des problèmes de décision pour lesquels il existe un algorithme polynomial en la taille des données.
- Classe NP: classe des problèmes de décision pour lesquels on peut vérifier un certificat donnant la solution en temps polynomial en la taille des données.

Formellement

- Un problème L est dans NP, si
  1. il existe un problème A dans P, tel que, pour tout w

     $$w \in L \text{ ssi } \exists u \in \Sigma^* \text{ tel que } < w, u > \in A$$

  2. il existe un polynome q tel que pour tout w, il y a un u avec $< w, u > \in A$ implique il y a un u de longueur $\leq q(|w|)$ avec $< w, u > \in A$.

Exemple

- Exemple: $HAMILTON \in NP$, car
  - $G \in HAMILTON$ ssi $\exists u \in \Sigma^*$ tel que $< G, u > \in A$, où A est le problème de décision de P défini par Problème A:
    - Instance: $< G, u >$
    - Question: Est-ce que u code un cycle hamiltonien de G?
- D'autre part, un cycle hamiltonien est de longueur au plus le nombre de sommets, donc la condition 2 est Ok.

$P = NP$?

- Proposition: $P \subset NP$.
  - (Preuve = par définition).
- Grand Problème: $P \neq NP$?
- Proposition:
  - $P \neq NP$
  - ssi il existe un problème de NP qui ne possède pas d'algorithme polynomiale.

Comparer les problèmes

- Soient $L_1, L_2 \subset \Sigma^*$. On dit que $L_1 \leq L_2$ s'il existe une fonction $f : \Sigma^* \to \Sigma^*$ calculable en temps polynomial, telle que, pour tout $w \in \Sigma^*$,

$$w \in L_1 \text{ ssi } f(w) \in L_2.$$

Préordre de réductibilité

- Théorème:
  1. $L \leq L$
  2. $L_1 \leq L_2$, $L_2 \leq L_3$ implique $L_1 \leq L_3$.

- Preuve de 1: prendre l'identité comme fonction f.
- Preuve de 2: si $L_1 \leq L_2$ via la fonction f, et $L_2 \leq L_3$ via la fonction g, on a $w \in L_1$ ssi $g(f(w)) \in L_2$, et $g \circ f$ est bien calculable en temps polynomial.

Réductibilité: application

- Théorème: $L_1 \leq L_2$, $L_2 \in P$ implique $L_1 \in P$.
- Preuve: L_1 est décidé par l'algorithme qui sur une entrée w calcule $f(w)$ et retourne la réponse de l'algorithme pour L_2 sur $f(w)$.
- Théorème: $L_1 \leq L_2$, $L_2 \in NP$ implique $L_1 \in NP$.
- Preuve: Même principe.

Conséquences

- $L_1 \leq L_2$, $L_1 \notin P$ implique $L_2 \notin P$.

Problème NP-complet

- Un problème $L \in C$ est NP-complet si
  1. $L \in NP$.
  2. $\forall L' \in NP$, $L' \leq L$.

- Un problème qui vérifie seulement 2. est dit NP-difficile.

Equivalence des problèmes complets

- Définition: Deux problèmes L_1 et L_2 sont équivalents, noté $L_1 \equiv L_2$, si $L_1 \leq L_2$ et $L_2 \leq L_1$.

- Théorème: Tous les problèmes complets sont équivalents.

  Preuve: si L_1 et L_2 sont NP-complets, alors $L_1 \leq L_2$ puisque L_2 est complet et $L_1 \in NP$. D'autre part, $L_2 \leq L_1$ par l'argument symétrique.

Equivalence des problèmes complets

- Théorème [Cook]: Il existe un problème NP-complet.

- Soit Pb n'importe quel problème complet.
- Théorème:
  - $P \neq NP$
  - ssi $Pb \notin P$.

- Preuve:
  1. Si $Pb \in P$, puisque tous les pbs de NP s'y réduisent, alors $NP = P$.
  2. Réciproquement, si $P \neq NP$, il y a un problème $L \in NP - P$. Puisque $L \leq Pb$, $Pb \notin P$.

Théorème de Cook

- Problème SAT:
  - Instance: Une formule booléenne F en forme normale conjonctive (définition transparent suivant)
  - Question: Existe-t'il une façon d'instancer les variables $x_1, \ldots, x_n$ pour que la formule s'évalue en vraie?
- Théorème [Cook] SAT est NP-complet

Preuve NP: facile. Complétude: nécessite de revenir aux machines de Turing (= modèle de calcul utilisé): non donnée ici.

Une formule F est en forme normale conjonctive

- F possède n variables $x_1, \ldots, x_n$.
- F s'écrit $F = C_1 \wedge C_2 \cdots \wedge C_p$ où les C_i sont appelés des "clauses".
- Chaque C_i s'écrit $x^*_{i_1}$,. . . ,$x^*_{i_{f(i)}}$, où les $x^*_{i_j}$ sont appelés des "littéraux".
- Chaque $x^*_{i_j}$ est soit un des x_{i_j} soit sa négation $\overline{x_{i_j}}$.

Variantes NP-complètes

- $SAT - k$: SAT avec au plus k occurences de chaque variable x_i
- $3 - SAT$: SAT où chaque clause a 3 littéraux. $C_i = x^*_{i_1} \vee x^*_{i_2} \vee x^*_{i_3}$.
- $3 - SAT\ Not - All - Equal$: $3 - SAT$ + on impose que les trois variables de chaque clause n'ont pas toutes la même valeur.
- $3 - SAT\ One - In - Three$: $3 - SAT$ + on impose qu'exactement une variable est vraie dans chaque clause.

Prouver la NP-complétude

Pour prouver un problème L est NP-complet.

1. On montre que L est dans NP.
2. On chosit un problème L' NP-complet, et on prouve que $L' \leq L$.

(Par $1.$, L est NP. Par $2.$, puisque L' est complet, $L'' \leq L' \leq L$ pour tout $L'' \in NP$: le problème L est donc NP-complet).

$3 - SAT$

- Théorème: $3 - SAT$ est NP-complet.

Preuve

- Preuve: $3 - SAT \in NP$
  car $F \in 3 - SAT$ ssi $\exists u \in \{0,1\}^n$ tel que $< F, u > \in A$
  avec
  Problème A:
  - Instance: <Formule F à n variables, $u \in \{0,1\}^n$>
  - Question: F est vrai avec les valeurs pour $x_1, \ldots, x_n$ données par les lettres de u

  Le problème A est bien dans P.
- On montre que $SAT \leq 3 - SAT$.

Soit F une instance de SAT. On transforme chaque clause C_i de F de la façon suivante:

- si C_i a 1 variable x^*, on ajoute deux variables a_i, b_i, et on remplace C_i par
  $(x^* \vee a_i \vee b_i) \wedge (x^* \vee a_i \vee \overline{b_i}) \wedge (x^* \vee \overline{a_i} \vee b_i) \wedge (x^* \vee \overline{a_i} \vee \overline{b_i})$,
- si C_i a 2 variables $x^* \vee y^*$, on ajoute une variable c_i, et on remplace C_i par $(x^* \vee y^* \vee c_i) \wedge (x^* \vee y^* \vee \overline{c_i})$
- si C_i a 3 variables, on la laisse telle qu'elle.
- si C_i a $k > 3$ variables, $C_i = x^*_{i_1} \vee \cdots x^*_{i_k}$, on ajoute $k-3$ nouvelles variables $z^i_1, z^i_2, \ldots, z^i_{k-3}$, et on remplace C_i par
  $(x_{i_1} \vee x_{i_2} \vee z^i_1) \wedge (x_{i_3} \vee \overline{z^i_1} \vee z^i_2) \wedge \cdots \wedge (x_{i_{k-1}} \vee \overline{z^i_{k-3}} \vee x_{i_k})$

- On note F' la formule obtenue, et $f : F \mapsto F'$ qui est bien calculable en temps polynomial.
- On a $F \in SAT$ ssi $f(F) \in 3-SAT$:
  - Sens $\Rightarrow$: une instantiation qui satisfait F se complète en une de F' en prenant $a_i = VRAI$, $b_i = FAUX$, $c_i = VRAI$, $z^i_i = \cdots = z^i_{n_i-2} = VRAI$, $z^i_{n_i-1} = \cdots, z^i_{k-3} = FAUX$, où $x^*_{n_i}$ est le premier littéral vrai de la clause C_i.
  - Sens $\Leftarrow$: une instantiation qui satisfait F' restreinte aux variables $x_1, \ldots, x_n$ donne une instantiation qui satisfait F.

Fin Preuve du Théorème.

CLIQUE

- Théorème: $CLIQUE$ est NP-complet.

- Problème $CLIQUE$:
  - Instance: Un graphe $G = (V, E)$, k un entier
  - Question: Existe-t'il un sous-graphe complet de k sommets?

Preuve

- Preuve: $CLIQUE \in NP$
  car $< G, k >$ ssi $\exists u \in \{0,1\}^*$ tel que $< G, k, u > \in A$
  avec
  Problème A:
  - Instance: <G,k,u>
  - Question: u code un sous-ensemble de k sommets de G qui forme un clique

  Le problème A est bien dans P.
- On montre que $3 - SAT \leq CLIQUE$.

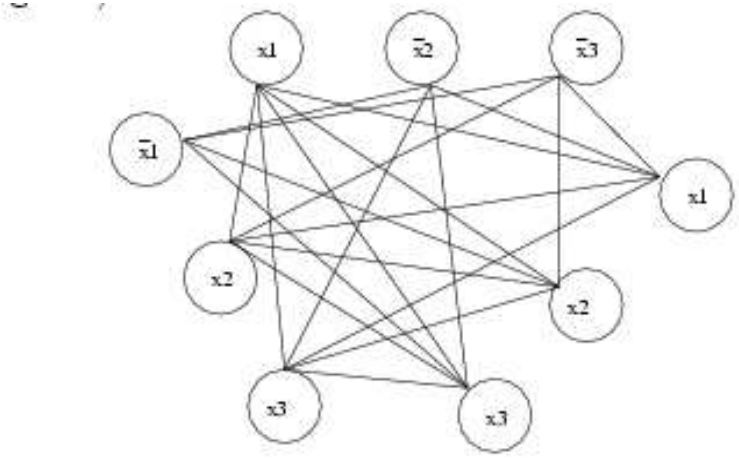
Soit F une instance de $3-SAT$ avec n variables et p clauses de taille 3.
On construit un graphe G qui contient 3 sommets pour chaque clause, et une arrête entre deux sommets ssi

- ils ne font pas partie de la même clause
- l'un des sommets n'est pas la négation de l'autre.

On note $f : F \mapsto (G, p)$ qui est bien calculable en temps polynomial.

Exemple

Pour $C = (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$.
On a G:



+-

- On a $F \in 3-SAT$ ssi $f(F) \in CLIQUE$:
  - Sens $\Rightarrow$: on sélectionne un sommet correspondant à une variable vraie dans chaque clause, et on a bien une clique de taille p, car deux sommets ne sont pas contradictoires.
  - Sens $\Leftarrow$: si on a une clique de taille p, alors il y a un sommet par clause. On le sélectionne pour instancier les variables, et c'est cohérent car on ne fait ainsi de choix contradictoires.

Fin Preuve du Théorème.

VERTEXCOVER

- Théorème: $VERTEXCOVER$ est NP-complet.

- Problème $CLIQUE$:
  - Instance: Un graphe $G = (V, E)$, k un entier
  - Question: Existe-t'il k sommets $v_1, \ldots, v_k$ tels que toute arête de G soit incidente à au moins l'un des v_i?

Preuve

- Preuve: $VERTEXCOVER \in NP$: exercice.
- $CLIQUE \leq VERTEXCOVER$: G a une clique de taille k ssi le complémentaire de G a une vertex cover de taille $|V \vdash k$.
  Considérer $f :< G, k > \mapsto < G^c, |V| - k >$.

HAMILTON

- Théorème: $HAMILTON$ est NP-complet.

- Problème $HAMILTON$:
  - Instance: Un graphe $G = (V, E)$
  - Question: Existe-t'il un cycle hamiltonien (qui viste tous les sommets une seule fois)?

(à partir de 3-SAT)

COLOR

- Théorème: $COLOR$ est NP-complet.

- Problème $COLOR$:
  - Instance: Un graphe $G = (V, E)$, k un entier
  - Question: Existe-t'il une façon de colorier les sommets avec k couleurs de telle sorte qu'aucune arête n'aie les extrémités de la même couleur?

Preuve

- $COLOR \in NP$: exercice (certificat= coloriage).
- On montre que $3 - SAT \leq COLOR$.

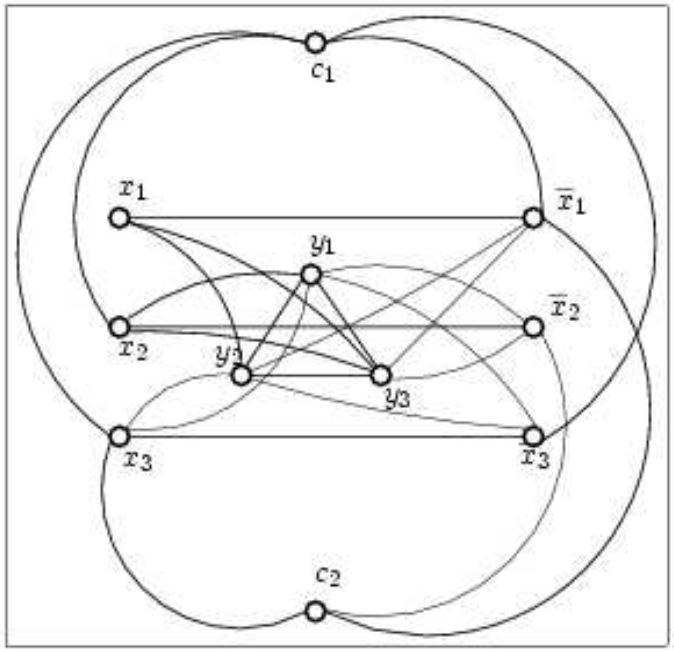
Soit F une instance de $3-SAT$ avec n variables et p clauses de taille 3.
On suppose $n \geq 5$ (sinon, il n'y a qu'un nombre fini de cas que l'on peut gérer).
On construit un graphe G qui contient $3n+p$ sommets $x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}, y_1, \ldots, y_n, c_1, \ldots, c_p$ avec les arêtes

- $(x_i, \overline{x_i})$
- (y_i, y_j), $i \neq j$
- (y_i, x_j), $i \neq j$
- $(y_i, \overline{x_j})$, $i \neq j$
- (x_i, c_k) si x_i n'est pas dans la clause C_k
- $(\overline{x_i}, c_k)$ si $\overline{x_i}$ n'est pas dans la clause C_k.

On note $f : F \mapsto G$ qui est bien calculable en temps polynomial.

Exemple

Pour $C = (x_1 \vee \overline{x_2}) \wedge (x_1 \vee x_2 \vee \overline{x_3})$ le graphe G est:



- On a $F \in SAT$ ssi $(f(F), n+1) \in COLOR$:
  - Sens $\Rightarrow$: A partir d'une instanciation des variables $x_1, \ldots, x_n$ qui rend F vrai, on pose $color(x_i) = i$ si x_i est vrai, $n+1$ sinon; $color(\overline{x_i}) = i$ si $\overline{x_i}$ est vraie, $n+1$ sinon; $color(y_i) = i$; $color(c_k) = color(x_i)$ pour un des x_i qui met la clause C_k à vraie ($color(\overline{x_i})$ pour un des $\overline{x_i}$ si C_k ne contient que des négations de variables).
    On vérifie arête par arête qu'elles ont toutes leurs extrémités de couleurs distinctes.

- Sens $\Leftarrow$. Supposons que l'on aie un coloriage du graphe avec $n+1$ couleurs.
  Fait 1: Les y_i forment une clique: ils ont donc tous une couleur distincte. Sans perte de généralité, on appelle $i = color(y_i)$.
  Fait 2: Il y a une arête entre x_i et $\overline{x_i}$ et une arête entre ces deux sommets et tous les y_j, pour $j \neq i$.
  Donc soit $color(x_i) = i, color(\overline{x_i}) = n+1$ ou $color(x_i) = n+1, color(\overline{x_i}) = i$
  Fait 3: c_k ne peut pas avoir la couleur $n+1$: car $n \geq 5$, et il y a un x_i avec $x_i \notin C_k$, $\overline{x_i} \notin C_k$.

- On considère l'instantiation définie par $x_i = 1, \overline{x_i} = 0$ si $color(x_i) = i, color(\overline{x_i}) = n+1$, et $x_i = 0, \overline{x_i} = 1$ si $color(x_i) = n+1, color(\overline{x_i}) = i$.
  Une variable (ou sa négation) qui est de la même couleur que la clause C_k est dans la clause C_k (sinon il y aurait une arête entre les deux). Elle met la clause C_k a vraie car sa couleur n'est pas $n+1$.
  Comme c'est vrai pour chaque C_k, l'instantiation construite satisfait bien la conjonction des clauses.

Fin Preuve du Théorème.

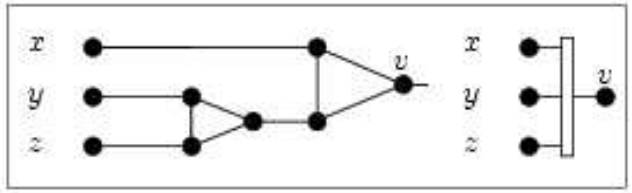
3-COLOR

- Théorème: $3 - COLOR$ est NP-complet.

- Problème $3 - COLOR$:
  - Instance: Un graphe $G = (V, E)$
  - Question: Existe-t'il une façon de colorier les sommets avec 3 couleurs de telle sorte qu'aucune arête n'aie les extrémités de la même couleur?

Un gadget pour 3-COLOR

A gauche le gadget, à droite son symbole.

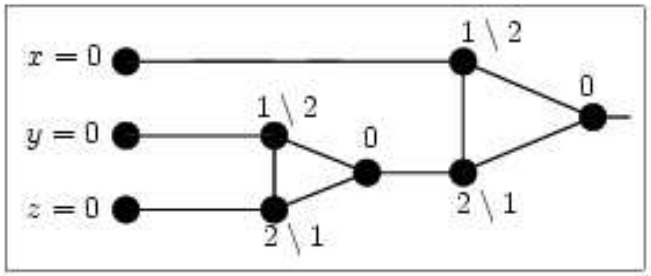


Proposition: si le gadget colorié avec 3-couleurs $0, 1, 2$.

1. Si $color(x) = color(y) = color(z) = 0$ alors $color(v) = 0$.
2. Toute autre possibilité pour x, y, z, permet de colorier v avec la couleur 1 ou 2.

Preuve

Dans les cas $color(x) = color(y) = color(z) = 0$, on a que les possibilités

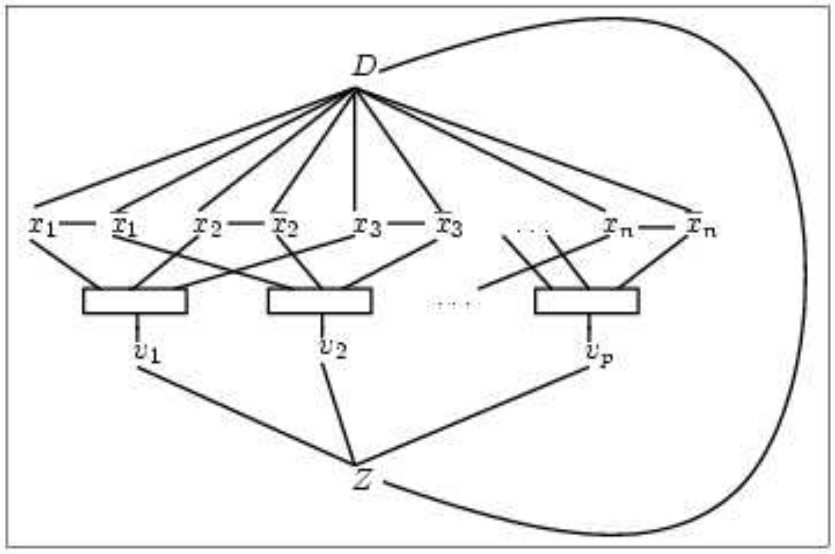


Assertion $2.$: exercice.

Preuve du théorème

- $3 - COLOR \leq COLOR$, et donc $3 - COLOR \in NP$.
- On montre que $3 - SAT \leq COLOR$.

Soit F une instance de $3-SAT$ avec n variables et p clauses de taille 3.
On construit le graphe G avec $1+2n+6p+1$ sommets suivant (qui se calcule bien en temps polynomial à partir de F).



On montre que F et satisfiable ssi G est 3-coloriable.

Sens $\Rightarrow$. A partir d'une instantiation qui satisfait F, on poste $color(x_i) = 1, color(\overline{x_i}) = 0$ si x_i est vrai, $color(x_i) = 0, color(\overline{x_i}) = 1$ si x_i est faux. Comme il y a une variable à vraie dans chaque clause, il y a pour chaque gadget une entrée non-nulle.
La propriété $2.$ du gadget permet d'avoir $color(v_i) \neq 0$ pour tout i. En posant $color(D) = 2, color(Z) = 0$ on obtient un 3-coloriage de G.

Sens $\Leftarrow$. Supposons que G est colorié avec 3-couleurs. Quitte à permuter les couleurs, on peut supposer $color(D) = 2, color(Z) = 0$ et $color(x_i) = 1, color(\overline{x_i}) = 0$ ou $color(x_i) = 0, color(\overline{x_i}) = 1$. Puisque $color(Z) = 0$, la couleur de v_i doit être 1 ou 2. La propriété $1.$ du gadget permet de dire que dans chaque clause il y a une variable de couleur 1.
Par conséquent, l'instantiation $x_i = 1, \overline{x_i} = 0$ si $color(x_i) = 1$, $x_i = 0, \overline{x_i} = 1$ sinon satisfait F.