Chapitre 6

(En partie, chapitre 7 de Sipser)

Brins de complexité du calcul

Le temps

Soit M une MT déterministe qui s'arrête sur toutes ses entrées.

Quel est le temps de calcul de M sur w ?

Facile et naturel : c'est le nombre de transitions avant l'arrêt de M sur entrée w.

Quel est le temps de calcul de M tout court ?

Définition (Sipser 7.1) Le temps de calcul de M est une fonction

$$f : \mathbb{N} \to \mathbb{N}$$
$$n \mapsto \max_{|w|=n} [\text{temps de calcul de } M \text{ sur } w].$$

On dira que M fonctionne en temps $f(n)$, ou que sa complexité de temps est $f(n)$.

Exemple 6.1. Le langage

$$\{ u\#u \mid u \in \{a, b\}^* \}$$

est décidé ...

- ... par une machine de Turing à une bande en un temps dans l'ordre de n^2, et

- ... par une machine de Turing à deux bandes en un temps dans l'ordre de n. ▲

Exemple 6.2. La fonction

$$\begin{aligned} \text{ADD} : \{0,1,\#\}^* &\to \{0,1\}^* \\ \texttt{bin}(a)\#\texttt{bin}(b) &\mapsto \texttt{bin}(a+b) \end{aligned}$$

où $\texttt{bin}(i)$ est le binaire renversé de l'entier i est calculée ...

- ... par une machine de Turing à une bande en un temps dans l'ordre de n^2, et
- ... par une machine de Turing à deux bandes en un temps dans l'ordre de n. ▲

Exemple 6.3. Le langage PATH, soit

$$\left\{ \langle G, s, t\rangle \mid \text{un chemin de } s \text{ à } t \text{ existe dans le graphe orienté } G \right\}$$

est décidé en temps $O(n^2)$.

Preuve.
Par la MT M qui fait le calcul suivant :

Sur entrée $\langle G, s, t\rangle$:

1. Marque le sommet s.
2. Pour chaque arc (a, b), si a est marqué alors marque b.
3. Si au moins un nouveau sommet a été marqué en 2, recommence l'étape 2.
4. Si t est marqué $accepte$ sinon $rejette$.

Bien sûr $L(M) =$ PATH.

Quel est le temps de calcul de M ?

Appelons m le nombre de sommets de G.

Choisissons d'encoder les graphes sous forme de matrices d'adjacences ; alors

$$m^2 < n = |\langle G, s, t\rangle| \leq m^2 + 2\log_2 m + 4.$$

Sur une machine M multi-rubans, la stratégie décrite ci-dessus utilise :

Étape 1 : temps dans $O(m)$,

Étapes 2 et 3 : pas plus de m fois $O(m^2)$, en prenant soin de marquer efficacement,

Étape 4 : temps dans $O(m)$.

M fonctionne donc en un temps dans

$$\begin{aligned} O(m) + m \times O(m^2) + O(m) = & \quad O(m^3) \\ \subset & \quad O(m^4) \\ = & \quad O(n^2). \end{aligned}$$

▲

Exemple 6.4. (Rappel) Le langage

$$\text{3-COL} = \{ \langle G \rangle : G \text{ est 3-coloriable } \}$$

est décidable. En quel temps ?

En un temps dans l'ordre de

$$\sim n^2 3^m \leq n^2 3^n$$

où m est le nombre de sommets, car notre stratégie nécessitait en pire cas
· l'examen de 3^m coloriages et
· des calculs intermédiaires pour chacun.

▲

Réductibilité

L'Espagne a-t-elle gagné son match de foot hier ?

——————> (réduction)

Hat Spanien seinen Fußball Match gestern gewonnen ?

↓

Oui ! < – – – – – – – – – – Ja !

Réductibilité

Définition 6.5. Le langage A se réduit au langage B, noté $A \leq B$, si il existe une fonction calculable

$$f : \Sigma^* \to \Sigma^*$$

telle que

$$\forall w \in \Sigma^* : w \in A \Leftrightarrow f(w) \in B.$$

▲

Exemple 6.6. PATH $\leq \{ u\#u \mid u \in \{a,b\}^* \}$.

Preuve. Nous connaissons un algorithme pour PATH, donc la fonction

$$f : (\Sigma \cup \{a,b,\#\})^* \to (\Sigma \cup \{a,b,\#\})^*$$
$$w \mapsto \begin{cases} \# & \text{si } w \in \text{PATH} \\ a\#b & \text{sinon} \end{cases}$$

est calculable. Elle vérifie $\forall w \in (\Sigma \cup \{a,b,\#\})^*$:

$$w \in \text{PATH} \Leftrightarrow f(w) \in \{ u\#u \mid w \in \{a,b\}^* \}.$$

▲

Exemple 6.7. Fixons $\Sigma = \{0,1,\#\}$.

PRODUIT est le langage des mots sur Σ

$$\texttt{bin}(a)\#\texttt{bin}(b)\#\texttt{bin}(i)$$

tels que le $i^{\text{ième}}$ bit de $\texttt{bin}(a \times b)$ est 1.

SOMMEITÉRÉE est le langage des mots

$$\texttt{bin}(z_1)\#\texttt{bin}(z_2)\#\ldots\texttt{bin}(z_m)\#\texttt{bin}(i)$$

tels que le $i^{\text{ième}}$ bit de $\texttt{bin}(\Sigma_{j=1}^m z_j)$ est 1.

Alors PRODUIT $\leq$ SOMMEITÉRÉE.

Preuve. Une MT T peut remplacer son entrée

$$a_0a_1\ldots a_m\#b_0b_1\ldots b_n$$

par

$$\texttt{bin}(z_0)\#\texttt{bin}(z_1)\#\ldots\#\texttt{bin}(z_m)$$

où

$$z_i = \begin{cases} 2^i \times b & \text{si } a_i = 1 \\ 0 & \text{si } a_i = 0. \end{cases}$$

et $b_0b_1\ldots b_n = \texttt{bin}(b)$.

Ainsi

$$f : \Sigma^* \to \Sigma^*$$
$$w \mapsto \begin{cases} y\#x & \text{si } w \text{ est de forme } u\#v\#x \\ 0 & \text{sinon} \end{cases}$$

où y est le résultat de T sur entrée $u\#v$, est une fonction calculable qui vérifie pour tout $w \in \Sigma^*$:

$$w \in \text{PRODUIT} \Leftrightarrow f(w) \in \text{SOMMEITÉRÉE}.$$

▲

Exemple 6.8. 3-COL $\leq$ 4-COL.

Preuve. La fonction ci-dessous est calculable :

$$f : \{0,1\}^* \to \{0,1\}^*$$
$$\langle G \rangle \mapsto \langle G' \rangle$$

où G' est le graphe G avec un sommet supplémentaire relié par une arête à chacun des anciens sommets.

Ainsi, pour tout G :

$$\langle G \rangle \in \text{3-COL} \Leftrightarrow f(\langle G \rangle) \in \text{4-COL}.$$

▲

Définition 6.9. SAT = $\{ \langle \phi \rangle : \phi$ est une expression booléenne satisfaisable $\}$. ▲

Par exemple,

- $x \wedge y \wedge \bar{x} \notin$ SAT
- $((x \vee y) \wedge (\bar{x} \vee y \vee \bar{z})) \to (\bar{y} \wedge z) \in$ SAT
- $(x \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee \bar{z} \vee x) \wedge (y \wedge \bar{z}) \in$ SAT.

Exemple 6.10. Soit A un langage décidé par

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r).$$

en temps $\tau(n)$ calculable. Alors A $\leq$ SAT.

Preuve. Ayoye. Comment faire ? Il nous faut une fonction calculable f telle que pour tout w :

$$w \in \text{A} \Leftrightarrow f(w) \in \text{SAT}$$

et on ne connaît pas A.

Mais on connaît M !

Une idée : produire, sur entrée w, une expression booléenne ϕ dont la satisfaisabilité (le cas échéant) exprime

> que le tableau des configurations de la machine M sur entrée w mène M à son état acceptant.

Tableau des configurations de M sur entrée w :

	1	2	3	4	$\cdots$		**j**		$\cdots$	$\cdots$	$\tau(n)+3$
1	#	q_0	w_1	w_2	$\cdots$	$\sqcup$	$\sqcup$	$\cdots$	$\cdots$	$\sqcup$	#
2	#	d	q_4	w_2	$\cdots$	$\sqcup$	$\sqcup$	$\cdots$	$\cdots$	$\sqcup$	#
3	#	d	c	q_{67}	$\cdots$	$\sqcup$	$\sqcup$	$\cdots$	$\cdots$	$\sqcup$	#
4	#	d	q_{31}	c	$\cdots$	$\sqcup$	$\sqcup$	$\cdots$	$\cdots$	$\sqcup$	#
:	#	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	#
$\vdots$	#	c	a	$\cdots$	b	b	q_{52}	c	$\cdots$	a	#
$\vdots$	#	c	a	$\cdots$	b	q_8	b	d	$\cdots$	a	#
t	#	c	a	$\cdots$	b	f	q_{12}	d	$\cdots$	a	#
	#	c	a	$\cdots$	b	f	g	q_7	$\cdots$	a	#
$\vdots$	#	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	#
$\vdots$	#	$\cdots$	$\cdots$	$\cdots$	c	q_9	d	$\cdots$	$\cdots$	$\cdots$	#
$\tau(n)$	#	$\cdots$	$\cdots$	$\cdots$	q_a	c	a	$\cdots$	$\cdots$	$\cdots$	#

- Utilisons dans ϕ les variables booléennes :

$$\{\, x_{t,j,\sigma} \mid 1 \le t \le \tau(n), 1 \le j \le \tau(n)+3, \\ \sigma \in Q \cup \Gamma \cup \{\#\} \,\}.$$

- Signification de $x_{t,j,\sigma}$:

une affectation satisfaisant ϕ où $x_{t,j,\sigma}$ = VRAI décrira un tableau dont la case (t,j) est σ ;

une affectation satisfaisant ϕ où $x_{t,j,\sigma}$ = FAUX décrira un tableau dont la case (t,j) n'est pas σ.

$$f : \Sigma^* \to \Delta^* \\ w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}$$

où la satisfaction de . . .

1. . . . ϕ_{case} assure qu'à chaque case du tableau est affecté un unique σ
2. . . . ϕ_{initiale} assure que la ligne 1 du tableau est la configuration initiale de M sur w
3. . . . ϕ_{accepte} assure que la dernière ligne est une configuration acceptante de M
4. . . . $\phi_{\text{légale}}$ assure que chaque ligne suit la précédente selon l'unique action légale de M.

ϕ_{case} : un unique symbole par case

Pour chaque $1 \le t \le \tau(n)$
et chaque $1 \le j \le \tau(n)+3$

inclure dans ϕ_{case} :

$$(x_{t,j,q_0} \vee x_{t,j,q_1} \vee \cdots \vee x_{t,j,a} \vee x_{t,j,b} \vee \cdots \vee x_{t,j,\#}) \\ \bigwedge$$

$$\neg[\ (x_{t,j,q_0} \wedge x_{t,j,q_1}) \vee$$
$$(x_{t,j,q_0} \wedge x_{t,j,q_2}) \vee$$
$$\vdots$$
$$(x_{t,j,q_0} \wedge x_{t,j,a}) \vee$$
$$\vdots$$
$$(x_{t,j,q_0} \wedge x_{t,j,\#}) \vee$$
$$(x_{t,j,q_1} \wedge x_{t,j,q_2}) \vee$$
$$(x_{t,j,q_1} \wedge x_{t,j,q_3}) \vee$$
$$(x_{t,j,q_1} \wedge x_{t,j,\#}) \vee$$
$$\vdots$$
$$(x_{t,j,a} \wedge x_{t,j,b}) \vee$$
$$\vdots \qquad]$$

ϕ_{initiale} : première ligne = config initiale sur w

$$x_{1,1,\#} \wedge$$
$$x_{1,2,q_0} \wedge$$
$$x_{1,3,w_1} \wedge$$
$$x_{1,4,w_2} \wedge$$
$$\vdots$$
$$x_{1,n+2,w_n} \wedge$$
$$x_{1,n+3,\sqcup} \wedge$$
$$\vdots$$
$$x_{1,\tau(n)+2,\sqcup} \wedge$$
$$x_{1,\tau(n)+3,\#}$$

ϕ_{accepte} : dernière ligne = config acceptante

$$x_{\tau(n),2,q_a}$$
$$\vee$$
$$x_{\tau(n),3,q_a}$$
$$\vee$$
$$x_{\tau(n),4,q_a}$$
$$\vee$$
$$\vdots$$
$$\vee$$
$$x_{\tau(n),\tau(n)+2,q_a}$$

$\phi_{\text{légale}}$: suite légale de configurations

Pour chaque $1 \leq t < \tau(n)$ et $1 < j < \tau(n) + 3$,
inclure dans $\phi_{\text{légale}}$ un $\bigvee$ de chaque $\begin{array}{|c|c|c|}\hline A & B & C \\ \hline \alpha & \beta & \gamma \\ \hline \end{array}$ possible, représenté par

$$(x_{t,j-1,A} \wedge x_{t,j,B} \wedge x_{t,j+1,C}$$
$$\wedge$$
$$x_{t+1,j-1,\alpha} \wedge x_{t+1,j,\beta} \wedge x_{t+1,j+1,\gamma})$$

• $f : w \mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}$ est calculable car τ calculable et M connue.

• $w \in$ A $\Rightarrow$ une suite de configs mène M sur entrée w à l'état $q_a \Rightarrow$ le tableau résultant prescrit une affectation satisfaisant $\phi \Rightarrow \phi \in$ SAT.

• $\phi \in$ SAT $\Rightarrow \phi$ décrit un tableau $\Rightarrow$ ce tableau décrit un calcul acceptant à partir de la config initiale de M sur $w \Rightarrow w \in$ A.

• • • A $\leq$ SAT. ▲

Supercherie !

Exemple 6.11. Tout langage décidable A se réduit à SAT par une réduction triviale.

Preuve. A $\leq$ SAT puisque la fonction

$$f : \Sigma^* \to \Sigma^*$$
$$w \mapsto \begin{cases} x & \text{si } w \in \text{A} \\ x \wedge \bar{x} & \text{sinon} \end{cases}$$

est calculable et vérifie $\forall w \in \Sigma^*$:

$$w \in \text{A} \Leftrightarrow f(w) \in \text{SAT}.$$

▲

À quoi donc servent les réductions ?

Les réductions A $\leq$ B prendront tout leur sens dans un contexte où leur complexité sera "faible" par rapport aux complexités des langages A et B eux-mêmes.

Exemple 6.12. Si A $\leq$ B et B est décidable alors A est décidable.
Preuve. Laissée en exercice. ▲

Enfonçons bien le clou !

Tout langage décidable A se réduit à tout langage B ($B \neq \emptyset, \Sigma^*$) par une réduction $\leq$ tout à fait triviale...

MAIS

...ceci changera lorsque nous imposerons une borne de temps polynômiale à la réduction...

ce sera là la base de la NP-complétude.

La classe P

Définition 6.13. (Sipser 7.7)
TIME$(t(n)) = \{ L \mid L$ est décidé par une MT déterministe en temps $O(t(n))\}$. ▲

Définition 6.14. (Sipser 7.12)
La classe de complexité P est

$$\mathsf{P} = \bigcup_{k \in \mathbb{N}} \mathsf{TIME}(n^k).$$

▲

Qu'ont de spécial les polynômes ?

- P est robuste : une bande, k bandes, k têtes, et plusieurs autres modèles engendrent la même classe P.

- Si $p(n)$ et $q(n)$ sont des polynômes alors $p(n) + q(n)$, $p(n)q(n)$ et $p(q(n))$ sont aussi des polynômes.

- P correspond aux langages décidables en pratique en un temps réalisable.

- Voici une idée de la signification concrète d'un temps de calcul polynômial, sous l'hypothèse d'un ordinateur exécutant un milliard d'opérations à la seconde :

Temps de calcul ininterrompu

$t(n)$	taille 25	taille 50	taille 100	taille 200
n	$0,025\ \mu$s	$0,05\ \mu$s	$0,1\ \mu$s	$0,2\ \mu$s
n^2	$0,625\ \mu$s	$2,5\ \mu$s	$10\ \mu$s	$40\ \mu$s
n^5	$9,5\ m$s	$0,3$ sec	10 sec	5 min
$2^{n/3}$	$0,3\ \mu$s	$0,1\ m$s	10 sec	37 siècles
2^n	$33\ \mu$s	13 jrs	10^{11} siècles	10^{41} siècles

La morale

Temps polynômial = le paradis !

Temps exponentiel = le fuir comme la peste !

L'ordinateur est peu utile face à un problème dont les seuls algorithmes connus nécessitent un temps exponentiel !

Quelques langages de la classe P

· $\{\, u\#u \mid u \in \{a,b\}^* \,\}$
· SOMMEITÉRÉE
· PRODUIT
· $\{\, \langle S \rangle : S = (w_1, w_2, \ldots, w_k)$ avec $w_i \in \Sigma^*$ et S est en ordre lexicographique$\}$
· PATH
· $\{\, \langle A, B, C \rangle : A, B, C$ sont des matrices entières telles que $AB = C\}$
· $\{\, \langle M, w, 1^n \rangle :$ la MT déterministe M accepte w en au plus n étapes $\}$.

NB : polynômial en fonction de quoi ?

Toujours en fonction de la longueur de l'entrée, que l'on appelle habituellement n.

Ex : $a, b \geq 0$ entiers :

longueur de $\langle a, b \rangle$ est $n \simeq \log_2 a + \log_2 b$.

Ainsi, avec des entiers a, b représentés en binaire (ou en décimal)

- on peut calculer $a + b$, $a - b$
  en temps $O(\log a + \log b) = O(n)$.

- on peut calculer ab, $\lfloor a/b \rfloor$, $a \bmod b$
  en temps $O\Big((\log a + \log b)^2\Big) = O(n^2)$.

Mais attention :

un temps de calcul dans $\Theta(a)$ sur une entrée entière a est exponentiel car $n = |\langle a \rangle| \simeq \log_2 a$ et ainsi $a \simeq 2^n$.

Dans P ou pas ?

Le langage COMPOSÉ, soit

$$\Big\{ \langle m \rangle : m \text{ est un entier composé } \Big\},$$

est décidable.

Est-il dans P ?

Stratégie naïve : essayer tous les diviseurs possibles entre 2 et $\sqrt{m}$.

Temps de calcul en pire cas : au moins $\sqrt{m}$.

En fonction de $n = |\langle m \rangle| \simeq \log_2 m$,

$$\sqrt{m} = \sqrt{2^{\log_2 m}} \simeq 2^{n/2},$$

donc cette stratégie n'est pas polynômiale.

Existe-t-il une stratégie différente qui résoud COMPOSÉ en temps polynômial ?

Note : la variante $\text{COMPOSÉ}_{\text{UNAIRE}}$ de ce problème où l'encodage unaire de m est utilisé est dans P puisque dans ce cas $n = |\langle m \rangle| = m$, mais cette variante est sans intérêt.

PRIMES is in P

Manindra Agrawal
Neeraj Kayal
Nitin Saxena
Dept of CSE, IIT Kanpur

Clay Research Award 2002

Conférence de mai 2003 au Fields Institute

Magnifique "état de l'art" de A. Granville du DMS

Informaticien qui s'intéresse à la complexité, à la théorie calculatoire des nombres, à la cryptographie. Résoud en 2002 le très vieux problème de la primalité avec deux de ses étudiants de 1er cycle.

Manindra Agrawal, 1966- ...

Dans P ou pas ?

Rappel : le langage

$$\text{3-COL} = \{\, \langle G \rangle : G \text{ est 3-coloriable} \,\}$$

est décidable en un temps dans $O(n^2 3^m)$ où m est le nombre de sommets.

Est-il dans P ?

Notre stratégie examine à coup sûr, en pire cas, les 3^m coloriages possibles.

Puisque $n = |\langle G \rangle| = m^2$, le temps de calcul selon cette stratégie est $> 3^{\sqrt{n}}$.

Quel que soit le polynôme $p(n)$, $3^{\sqrt{n}} \notin O(p(n))$, donc cette stratégie n'est pas polynômiale.

Existe-t-il une stratégie différente qui résoud 3-COL en temps polynômial ?

Réductions polynômiales

Définition 6.15. (Sipser 7.29) On dit que le langage A se réduit polynômialement au langage B, noté $A \leq_P B$ si

$$A \leq B$$

à l'aide d'une fonction

$$f : \Sigma^* \to \Sigma^*$$

calculable en temps polynômial. ▲

Exemples de réductions polynômiales

Exemples 6.16.

- PATH $\leq_{\mathrm{P}}$ $\{ u\#u \mid u \in \{a,b\}^* \}$
- PRODUIT $\leq_{\mathrm{P}}$ SOMMEITÉRÉE
- 3-COL $\leq_{\mathrm{P}}$ 4-COL
- Si A $\in$ P alors A $\leq_{\mathrm{P}}$ SAT.

Preuve. Laissée en exercice : sauf A $\leq$ SAT qui dépendait d'un $\tau(n)$, les réductions déjà vues fonctionnaient en temps polynômial. ▲

Une propriété importante

Théorème 6.17. (A $\leq_{\mathrm{P}}$ B et B $\in$ P) $\Rightarrow$ A $\in$ P. ■

Preuve. Nous sont donnés :
- polynômes $p(n)$ et $q(n)$,
- $f : \Sigma^* \to \Sigma^*$ calculable en temps $p(n)$ et vérifiant $\forall w \in \Sigma^* \ : \ w \in$ A $\Leftrightarrow f(w) \in$ B,
- M_2 une MT décidant B en temps $q(n)$.

Alors la MT M qui, sur entrée w,

calcule $w' = f(w)$;
simule M_2 sur w';

décide A et fonctionne en temps $O(q(p(n)))$, donc en temps polynômial.

Ainsi, la réductibilité polynômiale permet de lier les complexités de deux langages.

Exemple 6.18. Si 4-COL est dans P alors 3-COL est dans P.

Preuve. Appliquer le théorème 6.17 à la réduction 3-COL $\leq_{\mathrm{P}}$ 4-COL de 6.16. ▲

Incidemment, est-ce que 4-COL $\leq_{\mathrm{P}}$ 3-COL ?

Vérificateurs polynômiaux

Définition 6.19. (Sipser 7.18) Un vérificateur polynômial pour un langage A est une MT $\mathcal{V}$ telle que pour un polynôme p et pour tout $w \in \Sigma^*$:

1. si $w \in$ A alors
   il existe c tel que $\langle w, c\rangle \in L(\mathcal{V})$
2. si $w \notin$ A alors
   pour tout c $\langle w, c\rangle \notin L(\mathcal{V})$
3. le temps de $\mathcal{V}$ sur $\langle w, c\rangle$ est au plus $p(|\langle w\rangle|)$.

▲

Remarque 6.20. Un c tel que $\langle w, c\rangle \in L(\mathcal{V})$ est appelé certificat ou preuve ou témoin de l'appartenance de w au langage A. ▲

Remarque 6.21. Pas nécessaire de calculer le certificat, il suffit que celui-ci existe ! ▲

Remarque 6.22. Le temps polynômial du vérificateur est en fonction de $|\langle w\rangle|$ et non pas de $|\langle w, c\rangle|$. ▲

La classe NP

Définition 6.23. (Sipser 7.19)
La classe de complexité NP est l'ensemble des langages possèdant un vérificateur polynômial. ▲

Exemple 6.24. Le langage 4-COL est dans NP.

Preuve.
L'idée : le témoin c prescrira un 4-coloriage légal des sommets de G, i.e. qui donne à des sommets adjacents des couleurs différentes.

Comme il se doit, un tel témoin c existe ssi $\langle G \rangle \in$ 4-COL.

En détail maintenant, que fait le vérificateur $\mathcal{V}$?

Le vérificateur $\mathcal{V}$ est une banale MT qui décide

$$\{\langle G, c \rangle : c \text{ prescrit un 4-coloriage de } G\}.$$

En détail, $\mathcal{V}$ fait :

Sur entrée $\langle G, c \rangle$:

1. Si $c \notin \{N, R, V, B\}^*$ ou $|c| < m$ alors rejette.
2. Pour $1 \leq i < j \leq m$: si l'arête (i, j) est présente dans G et $c_i = c_j$ alors rejette.
3. accepte.

Examinons les 3 propriétés requises de $\mathcal{V}$:

I. Si $\langle G \rangle \in$ 4-COL...

...alors il existe un coloriage légal c de $G \Rightarrow$
$\exists c$ tel que $\mathcal{V}$ accepte $\langle G, c \rangle$.

II. Si $\langle G \rangle \notin$ 4-COL...

...alors pour toute suite $c \in \{N, R, V, B\}^*$, c n'est pas un coloriage légal de $G \Rightarrow$
$\forall c$, $\mathcal{V}$ rejette $\langle G, c \rangle$.

III. Temps de calcul

Le temps de $\mathcal{V}$ sur entrée $\langle G, c \rangle$ est mesuré en fonction non pas de la longueur de $\langle G, c \rangle$ mais uniquement de la portion excluant le témoin, ici $n = |\langle G \rangle| = m^2$.

Étape 1 : $O(m)$ opérations.
Étape 2 : $O(m^2)$ itérations, chacune faisable en $O(n)$ étapes, sur une multi-rubans.

En fonction de n, $\mathcal{V}$ utilise en temps $O(nm^2) = O(n^2)$, ie polynômial.

Ainsi $\mathcal{V}$ est bel et bien un vérificateur polynômial pour 4-COL, et ceci conclut la preuve que 4-COL est dans NP. ▲

Théorème 6.25. ($A \leq_P B$ et $B \in$ NP) $\Rightarrow A \in$ NP.

Preuve.
Soit $f : \Sigma^* \to \Sigma^*$ calculable en temps polynômial $p(n)$ telle que $\forall w \in \Sigma^* \ : \ w \in A \Leftrightarrow f(w) \in B$.

Soit $\mathcal{V}_2$ un vérificateur pour B de temps polynômial $q(n)$.

Appelons $\mathcal{V}$ la MT qui

Sur entrée $\langle w, c \rangle$:
calcule $w' = f(w)$;
simule $\mathcal{V}_2$ sur $\langle w', c \rangle$.

Alors le temps de $\mathcal{V}$ sur entrée $\langle w, c \rangle$ est :

$$\begin{array}{c} \text{temps de } f \text{ sur } w + \text{temps de } \mathcal{V}_2 \text{ sur } \langle w', c \rangle \\ \leq \\ p(|\langle w \rangle|) + q(|\langle w' \rangle|) \\ \leq \\ p(|\langle w \rangle|) + q((p(|\langle w \rangle|)) \end{array}$$

donc polynômial en fonction de $|\langle w \rangle|$.

De plus on a :

$$\begin{array}{rcl} w \in A & \Rightarrow & w' = f(w) \in B \\ & \Rightarrow & \exists c \ : \ \mathcal{V}_2 \text{ accepte } \langle w', c \rangle \\ & \Rightarrow & \exists c \ : \ \mathcal{V} \text{ accepte } \langle w, c \rangle. \end{array}$$

D'autre part :

$$\begin{array}{rcl} w \notin A & \Rightarrow & w' = f(w) \notin B \\ & \Rightarrow & \forall c \ : \ \mathcal{V}_2 \text{ rejette } \langle w', c \rangle \\ & \Rightarrow & \forall c \ : \ \mathcal{V} \text{ rejette } \langle w, c \rangle. \end{array}$$

▲

Corollaire 6.26.
Le langage 3-COL est dans NP.

Preuve. Nous avons vu que

$$\text{3-COL} \leq_{\text{P}} \text{4-COL}$$

et que

$$\text{4-COL} \in \text{NP}.$$

■

Exemple 6.27.
Le langage COMPOSÉ est dans NP.

Preuve 1.
P ⊆ NP (exercice !), et Agrawal *et al* ont montré que COMPOSÉ est dans P.

Preuve 2.
(directement, sans utiliser Agrawal *et al*).

Témoin : un diviseur non trivial d.

Vérificateur $\mathcal{V}$: MT qui fait, sur entrée $\langle m, d\rangle$:

1. si $d = 0$ ou $d = 1$ ou $d \geq m$ alors rejette
2. si d ne divise pas m alors rejette
3. accepte.

$$L(\mathcal{V}) = \Big\{\langle m, d\rangle : 1 < d < m, \text{ et } d \text{ divise } m\Big\}.$$

I. Si $\langle m\rangle \in$ COMPOSÉ...

...alors il existe d tq $\langle m, d\rangle \in L(\mathcal{V})$.

II. Si $\langle m\rangle \notin$ COMPOSÉ...

...alors aucun d ne vérifie $\langle m, d\rangle \in L(\mathcal{V})$.

III. Temps de calcul

Temps que prend $\mathcal{V}$: $O((\log m)^2) = O(|\langle m\rangle|^2)$.

Donc M est bel et bien un vérificateur polynomial pour COMPOSÉ. ▲

Exemple 6.28.
Le langage HAMPATH, soit

$\{ \langle G, s, t \rangle \mid$ un chemin existe de s à t dans le graphe orienté G rencontrant une et une seule fois chaque sommet $\}$

est dans NP.

Preuve.

L'idée : le certificat c prescrira un chemin hamiltonien entre s et t.

Formellement, le vérificateur $\mathcal{V}$ fait :

Sur entrée $\langle \langle G, s, t \rangle, \langle c_1, \ldots, c_k \rangle \rangle$:

1. si $k < m$ ou $c_1 \neq s$ ou $c_m \neq t$ alors rejette
2. pour $1 \leq i < m$: si (c_i, c_{i+1}) n'est pas un arc de G alors rejette
3. pour $1 \leq i < j \leq m$: si $c_i = c_j$ alors rejette
4. accepte

où m est le nombre de sommets de G.

$\mathcal{V}$ ne fait que vérifier qu'un préfixe de c est un chemin hamiltonien. Ainsi :

I. Si $\langle G, s, t \rangle \in$ HAMPATH...

...il existe un chemin hamiltonien c de s à $t \Rightarrow$
$\exists c$ tel que $\mathcal{V}$ accepte $\langle \langle G, s, t \rangle, c \rangle$.

II. Si $\langle G, s, t \rangle \notin$ HAMPATH...

...pour toute suite c de sommets, c n'est pas un chemin hamiltonien de s à $t \Rightarrow$
$\forall c$, $\mathcal{V}$ rejette $\langle \langle G, s, t \rangle, c \rangle$.

III. Temps de calcul

Le temps de $\mathcal{V}$ est mesuré en fonction
de $n = |\langle G, s, t \rangle| \simeq m^2$.

Étape 1 : $O(m^2)$ opérations (c_i en unaire).
Étape 2 : $O(m^3)$ opérations.
Étape 3 : $O(m^2 \log m)$ opérations.

Donc $\mathcal{V}$ fonctionne en temps $O(m^3) \subset O(n^2)$.

Ainsi $\mathcal{V}$ est un vérificateur polynômial pour HAMPATH, d'où HAMPATH est dans NP. ▲

Exemple 6.29. SAT $\in$ NP.

Aperçu de la preuve.
Le certificat c d'une expression booléenne satisfaisable ϕ prescrira une affectation aux variables de ϕ rendant ϕ vraie.

Le vérificateur $\mathcal{V}$ sur entrée $\langle \phi, c \rangle$ n'a qu'à évaluer ϕ sur l'affectation c, ie

$$L(\mathcal{V}) = \{\langle \phi, c \rangle : c \text{ satisfait } \phi\}.$$

Exercice : $\mathcal{V}$ fonctionne en temps poly($|\langle \phi \rangle|$). ▲

Exemple 6.30. (Sipser 7.24)
CLIQUE = $\{ \langle G, k \rangle \mid G$ contient un sous-graphe complet de taille $k \} \in$ NP.

Preuve. Laissée en exercice. ▲

Exemple 6.31. (Sipser 7.25)

Le langage SUBSET-SUM, soit

$\Big\{ \langle \{x_1, \cdots, x_m\}, t \rangle \mid$
les x_i et t sont des entiers, et
$\exists \{y_1, \cdots, y_l\} \subseteq \{x_1, \cdots, x_m\}$
tel que $\sum_{i=1}^{l} y_i = t \Big\}$

est dans NP.

Preuve. Laissée en exercice. ▲

P = NP ?

P : les langages décidables efficacement.

NP : les langages vérifiables efficacement.

N'oubliez pas de réclamer votre prix de 1 000 000 $US du *Clay Mathematical Institute* si vous découvrez la réponse à cette question !

Définition 6.32.

$$\mathsf{EXPTIME} = \bigcup_{k \geq 0} \mathsf{TIME}(2^{n^k}).$$

▲

Théorème 6.33.

$$\mathsf{NP} \subseteq \mathsf{EXPTIME}.$$

Aperçu de la preuve. Soit $A \in \mathsf{NP}$ et $\mathcal{V}$ le vérificateur pour A qui fonctionne en temps polynômial $p(n)$.

Un décideur M pour A, sur entrée w, avec $n = |w|$, teste tous les certificats de longueur $\leq p(n)$ en invoquant $\mathcal{V}$ à chaque fois.

La MT M accepte w si et seulement si un certificat a été trouvé.

Estimons le temps de calcul $t(n)$ de M, en posant d la taille de l'alphabet de $\mathcal{V}$:

$$\begin{aligned} t(n) &\approx d^{p(n)} \times p(n) \\ &\approx 2^{\log_2(d) p(n)} \times 2^{\log_2(p(n))} \\ &\in O(2^{n^k}) \end{aligned}$$

pour un certain k. ▲

Pourquoi NP s'appelle NP

On peut définir une notion (plutôt artificielle) de temps de calcul d'une MT non déterministe (comme dans Sipser 7.9).

La classe NP fut d'abord définie en ces termes et le nom est resté :

NP = Nondeterministic Polynomial Time.

La NP-complétude

Définition 6.34. Le langage B est NP-difficile ou NP-ardu si pour tout A $\in$ NP, A $\leq_P$ B. ▲

Définition 6.35. (Sipser 7.27) Le langage B est NP-complet si

1. B $\in$ NP et
2. B est NP-difficile.

▲

Intuition à retenir

Si B est NP-complet alors

1. B n'est pas trop difficile (il est dans NP, donc dans EXPTIME, alors que ça aurait pu être encore bien pire),
2. B est au moins aussi difficile que n'importe quel langage de NP (tout autre langage de NP se réduit à lui).

Résoudre un seul problème suffit !

Théorème 6.36. Si B est un langage NP-ardu et B $\in$ P, alors P = NP.

Preuve. On a vu à l'exemple 6.27 que P $\subseteq$ NP.

Pour montrer que NP $\subseteq$ P, prenons un A $\in$ NP.
Puisque B est NP-ardu, on a A $\leq_P$ B ; et A $\in$ P car B $\in$ P.

On a donc P = NP. ■

Corollaire 6.37. (Sipser 7.35) Si B est NP-complet et B $\in$ P, alors P = NP. ■

La "magie" de la NP-complétude est que si l'on peut résoudre un seul problème NP-complet efficacement (c'est à dire en temps polynômial), alors on aura résolu efficacement tous les problèmes de NP.

Théorème de Cook et Levin

Bien beau tout ça, mais est-ce qu'il existe des langages NP-complets ?

Théorème 6.38. (Sipser 7.37) (Cook-Levin)
Le langage SAT est NP-complet.

Esquisse de preuve de Cook-Levin

Nous savons que SAT $\in$ NP.
À montrer : SAT est NP-ardu.

Soit donc A un langage quelconque de NP.
À montrer : A $\leq_{\mathrm{P}}$ SAT.

Nous rentabiliserons ici la supercherie de l'exemple 6.10, qui réduisait péniblement à SAT le langage d'une MT M de temps $\tau(n)$.

(Rappel) : la réduction de l'exemple 6.10 produisait ϕ dont la satisfaisabilité exprimait que l'unique suite de configurations de M partant de

	1	2	3	4	$\cdots$			$\cdots$	$\tau(n)+3$
1	#	q_0	w_1	w_2	$\cdots$	w_n	$\sqcup$	$\cdots$	$\sqcup$	#

se terminait par une configuration acceptante.

Désormais :

M = le vérificateur $\mathcal{V}$ de A

$\tau(n)$ = le temps d'exécution (polynômial) de $\mathcal{V}$.

Désormais :

La satisfaction de ϕ doit impliquer l'initialisation de la première ligne du tableau à

	1	2	3	4	$\cdots$	$n+2$			$\cdots$	$\tau(n)+4$
1	#	q_0	w_1	w_2	$\cdots$	w_n	\$	c_1	$\cdots$	c_m	#

où $c_1 c_2 \ldots c_m$ est n'importe quelle suite de symboles susceptible de constituer un certificat
(note : $m = \tau(n) - n$ sans perte de généralité).

ϕ_{initiale} : ligne 1 = config initiale sur w

devient donc

ϕ_{initiale} : ligne 1 = config initiale sur $w\$c_1 \ldots c_m$

comme suit :

ϕ_{initiale} auparavant :

$$
\begin{aligned}
& x_{1,1,\#} \ \wedge \\
& x_{1,2,q_0} \ \wedge \\
& x_{1,3,w_1} \ \wedge \\
& x_{1,4,w_2} \ \wedge \\
& \quad \vdots \\
& x_{1,n+2,w_n} \ \wedge \\
& x_{1,n+3,\sqcup} \ \wedge \\
& \quad \vdots \\
& x_{1,\tau(n)+2,\sqcup} \ \wedge \\
& x_{1,\tau(n)+3,\#}
\end{aligned}
$$

ϕ_{initiale} maintenant :

$$
\begin{aligned}
& x_{1,1,\#} \ \wedge \\
& x_{1,2,q_0} \ \wedge \\
& x_{1,3,w_1} \ \wedge \\
& x_{1,4,w_2} \ \wedge \\
& \quad \vdots \\
& x_{1,n+2,w_n} \ \wedge \\
& \\
& \bigwedge
\end{aligned}
$$

$$
\begin{aligned}
& x_{1,n+3,\$} \wedge \\
& \Big(\bigvee_{\sigma\in\Sigma} x_{1,n+4,\sigma}\Big) \wedge \\
& \Big(\bigvee_{\sigma\in\Sigma} x_{1,n+5,\sigma}\Big) \wedge \\
& \quad \vdots \\
& \Big(\bigvee_{\sigma\in\Sigma} x_{1,\tau(n)+3,\sigma}\Big) \wedge \\
& x_{1,n+\tau(n)+4,\#}
\end{aligned}
$$

où Σ est l'alphabet de $\mathcal{V}$.

La nouvelle ϕ_{initiale} force donc le remplacement des $\sqcup$ de la ligne 1 par des symboles de Σ.

Le reste est inchangé :

$$
\begin{aligned}
f : \Sigma^* &\to \Delta^* \\
w &\mapsto \phi = \phi_{\text{case}} \wedge \phi_{\text{initiale}} \wedge \phi_{\text{accepte}} \wedge \phi_{\text{légale}}.
\end{aligned}
$$

De calculable auparavant, f devient calculable polynômialement car $\tau(n)$ est un polynôme.

• $w \in A \Rightarrow$ pour un certain $c_1c_2\ldots c_m$, $\mathcal{V}$ accepte $w\$c_1c_2\ldots c_m \Rightarrow$ un tableau ayant $\#w\$c_1c_2\ldots c_m\#$ à la ligne 1 dépeint le calcul de $\mathcal{V}$ sur $w\$c_1c_2\ldots c_m \Rightarrow$ ce tableau prescrit une affectation satisfaisant $\phi \Rightarrow \phi \in$ SAT.

• $\phi \in$ SAT $\Rightarrow \phi$ prescrit un tableau ayant un certain $\#w\$c_1c_2\ldots c_m\#$ à la ligne $1 \Rightarrow$ ce tableau dépeint un calcul acceptant de $\mathcal{V}$ sur $w\$c_1c_2\ldots c_m \Rightarrow c_1c_2\ldots c_m$ témoigne de l'appartenance de w à A $\Rightarrow w \in$ A.

Donc A $\leq$ SAT, et ceci conclut la preuve du théorème de Cook et Levin. ▲

Aussi difficile de trouver un algorithme polynômial pour SAT que de trouver du coup un algorithme polynômial pour tous les langages de NP !!!

Un algorithme polynômial pour SAT fournirait :
- un algorithme polynômial pour 3-COL
- un algorithme polynômial pour HAMPATH
- un algorithme polynômial pour CLIQUE
- etc.

SAT est-il le seul problème NP-complet ?

Non ! Des milliers ont été répertoriés à ce jour !

2) Doit-on refaire la preuve de Cook-Levin à chaque fois qu'on veut démontrer qu'un problème est NP-complet ?

Non ! Voyons pourquoi.

La NP-complétude est contagieuse

Théorème 6.39. (Sipser 7.36)
Si B est NP-complet et $B \leq_P C$ avec $C \in$ NP, alors C est NP-complet.

Preuve. Comme $C \in$ NP, montrons C NP-ardu.

Prenons $A \in$ NP. Alors $A \leq_P B$ car B est NP-ardu. Donc $A \leq_P B \leq_P C$.

On conclut (exercice !) $A \leq_P C$, donc que C est NP-ardu. ■

Variantes NP-complètes de SAT

Définition 6.40. Une expression booléenne est en forme normale conjonctive si elle ressemble à

$$(*\vee *)\wedge(*\vee *\vee *\vee *\vee *)\wedge(\cdots)\wedge\ldots$$

où $*$ est une variable x ou sa négation $\bar{x}$. ▲

SATFNC

Théorème 6.41. SATFNC = $\{ \langle \phi \rangle : \phi$ est une expression booléenne en FNC et ϕ est satisfaisable $\}$ est NP-complet.

Preuve.
Suffit d'appliquer De Morgan à ϕ_{case} et de distribuer les $\vee$ sur les $\wedge$ dans $\phi_{\text{légale}}$ de la preuve de Cook-Levin. ■

Définition 6.42. Une expression booléenne est en forme normale 3-conjonctive si elle ressemble à

$$(\ * \vee\ *\ \vee *\ ) \wedge (\ * \vee\ *\ \vee *\ ) \wedge (\ * \vee\ *\ \vee *\ ) \wedge \ \ldots$$

où chaque clause implique 3 variables distinctes. ▲

3SAT

Théorème 6.43. (Sipser 7.42) 3SAT = $\{ \langle \phi \rangle : \phi$ est une expression booléenne en FN3C et ϕ est satisfaisable $\}$ est NP-complet.

Aperçu de la preuve.
• 3SAT $\in$ NP : exercice.
• 3SAT est NP-ardu : il suffit d'une réduction polynômiale de SATFNC à 3SAT, pour ensuite appliquer le théorème 6.39.

Alors comment réduire SATFNC à 3SAT ?

En transformant chaque clause de l'exemplaire ϕ de SATFNC comme suit :

$$(x) \mapsto (x \vee y \vee z) \wedge (x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee \bar{y} \vee \bar{z})$$

$$(x_1 \vee x_2) \mapsto (x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \bar{y})$$

$$(x_1 \vee x_2 \vee x_3) \mapsto (x_1 \vee x_2 \vee x_3)$$

$$(x_1 \vee x_2 \vee \cdots \vee x_k) \mapsto (x_1 \vee x_2 \vee y_1) \wedge (\overline{y_1} \vee x_3 \vee y_2) \wedge (\overline{y_2} \vee x_4 \vee y_3) \wedge \cdots \wedge (\overline{y_{k-4}} \vee x_{k-2} \vee y_{k-3}) \wedge (\overline{y_{k-3}} \vee x_{k-1} \vee x_k)$$

▲

Faisons le point

1. SAT et 3SAT sont NP-complets.
2. Pour prouver que B est NP-complet, il suffit d'identifier un langage A tel que
   2.1 A est NP-ardu (ou NP-complet),
   2.2 $A \leq_P B$,
   2.3 $B \in NP$.

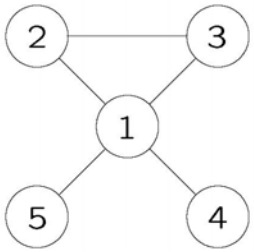
Souvent, l'étape 2.2 requiert une bonne dose d'ingéniosité.

COUVERTURE

Notre premier problème NP-complet qui s'éloigne du problème de la satisfaisabilité d'expressions booléennes :

Théorème 6.44. (Sipser 7.44)
COUVERTURE = $\{ \langle G, s \rangle \mid$ il existe un ensemble de s sommets qui "couvre" le graphe non orienté $G \}$ est NP-complet.

Par exemple, le graphe G ci-dessous est "couvert" par $\{1, 2\}$ puisque chaque arête de G possède une extrémité dans $\{1, 2\}$.



G est aussi couvert par $\{1, 3\}$, par $\{2, 3, 4, 5\}$, et par $\{1, 2, 3, 4, 5\}$: $\langle G, 2 \rangle \notin$ COUVERTURE,
$\langle G, 3 \rangle \in$ COUVERTURE,
$\langle G, 1 \rangle \notin$ COUVERTURE.

Preuve.
• COUVERTURE $\in$ NP : exercice.

• COUVERTURE est NP-ardu : suffit de réduire polynômialement 3SAT à COUVERTURE et d'appliquer le théorème 6.39.

Il nous faut donc f calculable en temps polynômial telle que

$$\langle \phi \rangle \in 3SAT \text{ ssi } f(\langle \phi \rangle) = \langle G, s \rangle \in \text{COUVERTURE}.$$

Comment diable faire ceci ?

Voici. L'action de f sur entrée $\langle\phi\rangle$ est de :

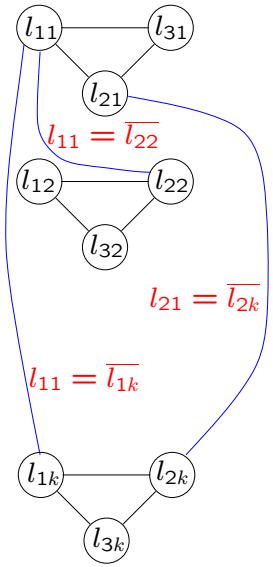
1) créer un triangle pour chacune des clauses de $\phi = (l_{11} \vee l_{21} \vee l_{31}) \wedge \ldots \wedge (l_{1k} \vee l_{2k} \vee l_{3k})$,

2) relier chaque paire de sommets complémentaires par une arête, et

3) poser $s = 2 \times k =$ deux fois le nombre de clauses de ϕ.

Une MT sur entrée $\langle\phi\rangle$ peut certes calculer $f(\langle\phi\rangle)$ en temps polynômial.

Mais pourquoi ça marche ?



Ça marche puisque d'une part,

$\langle\phi\rangle \in$ 3SAT $\Rightarrow$

il existe une affectation qui satisfait $\phi \Rightarrow$

cette affectation permet d'identifier k litteraux, un par clause, affectés à VRAI $\Rightarrow$

les $s = 2 \times k$ sommets restants couvrent $G \Rightarrow$

$f(\langle\phi\rangle) = \langle G, s\rangle \in$ COUVERTURE.

D'autre part,

$f(\langle\phi\rangle) = \langle G, s\rangle \in$ COUVERTURE $\Rightarrow$

un ensemble E de s sommets couvre $G \Rightarrow$

E possède 2 sommets par triangle et on peut sans contradiction affecter VRAI au littéral de chaque sommet omis $\Rightarrow$

cette affectation satisfait $\phi \Rightarrow$

$\langle\phi\rangle \in$ 3SAT.

Ainsi 3SAT $\leq_{\mathrm{P}}$ COUVERTURE, tel que désiré. ▲

Le jeu Minesweeper : préparatifs

Et maintenant un langage NP-complet très loin des expressions booléennes, mais d'abord, pause publicitaire...

• Richard Kaye, Minesweeper is NP-complete, *The Mathematical Intelligencer*, Springer Verlag, vol. 22, no. 2, 2000, 9–15.

"...Finally, it is nice to know that to current knowledge, there may still be an efficient algorithm for Minesweeper, and finding it could solve one of mathematics's most important open problems."

• Ian Stewart, Million-Dollar Minesweeper, *Scientific American*, vol. 283, oct. 2000, 94–95.

"The Clay Mathematics Institute, a nonprofit educational foundation in Cambridge, Mass., is offering million-dollar prizes for the solutions to seven infamous unsolved problems. One of them is the P versus NP question. [...] A clever amateur may be able to solve it with the help of Minesweeper."

• Lisa Lipman,
http ://news.excite.com/news/ap/001102/15/
minesweeper-mystery

BOSTON (AP) - "Minesweeper, a seemingly simple game included on most personal computers, could help mathematicians crack one of the field's most intriguing problems."

Retour aux préparatifs

Rappel. 3SAT est NP-complet.

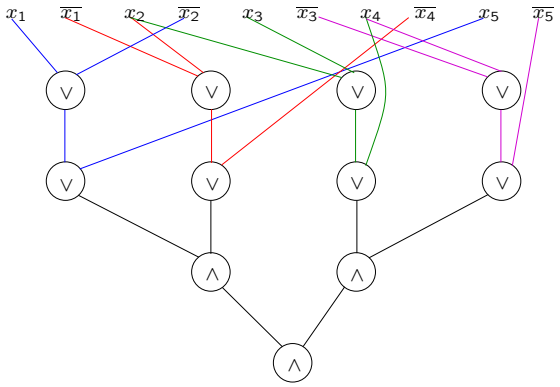
Remarque 6.45. On peut associer à un exemplaire de 3SAT un circuit booléen. ▲

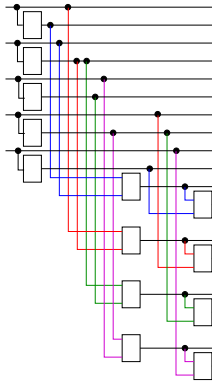
Remarque 6.46. On peut même supposer sans perte de généralité que ce circuit booléen ne contient pas de porte $\vee$. ▲

Par exemple :

$$(x_1 \vee \overline{x_2} \vee x_5) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_4}) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\overline{x_3} \vee x_4 \vee \overline{x_5})$$

donne lieu à





CIRCBOOL

Théorème 6.47. CIRCBOOL = $\{ \langle C \rangle : C$ est un $\{x_i, \neg, \wedge\}$-circuit et il existe une affectation aux portes x_i qui rend VRAIE l'unique sortie de $C\}$ est NP-complet.

Preuve.
• CIRCBOOL $\in$ NP : exercice.

• CIRCBOOL est NP-ardu : exercice
(facile si l'on choisit bien le langage NP-complet A dont on montrera ensuite que A $\leq_{\mathrm{P}}$ CIRCBOOL).

■

DÉMINEUR

Quel problème de décision pouvons-nous associer au jeu de démineur ? ?

Problème considéré par Richard Kaye :

Donnée : tableau $m \times m$ de symboles de
$\{ \bullet, \sqcup, 0, 1, 2, 3, 4, 5, 6, 7, 8 \}$

Question : est-il possible de remplir les cases
$\sqcup$ de manière à obtenir un tableau légal ?

Le problème que nous allons considérer :

DÉMINEUR :

Donnée : tableau $m \times m$ de symboles de
$\{ \bullet, \sqcup, 0, 1, 2, 3, 4, 5, 6, 7, 8 \}$, et
deux entiers $1 \leq i, j \leq m$

Question : est-il possible qu'une mine se
trouve en position (i, j) ?

Le langage DÉMINEUR est l'ensemble des $\langle$tableau, $i, j\rangle$ pour lesquels la réponse à la question est oui.

Théorème (Kaye 2000).
DÉMINEUR est NP-complet.

Preuve.
• DÉMINEUR $\in$ NP : exercice.

• CIRCBOOL $\leq_m^p$ DÉMINEUR :
L'idée : transformer le circuit C en un tableau de DÉMINEUR, et se servir de "fils" propageant VRAI ou FAUX d'une porte à l'autre :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1			1			1			1			1		
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

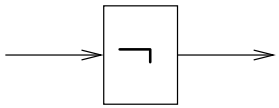
Un fil transportant la valeur VRAI :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
●	1		●	1		●	1		●	1		●	1		
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

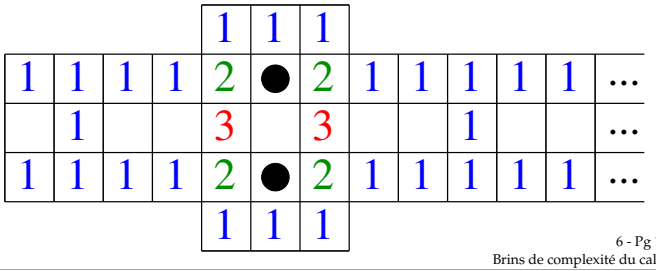
Un fil transportant la valeur FAUX :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	●		1	●		1	●		1	●		1	●	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

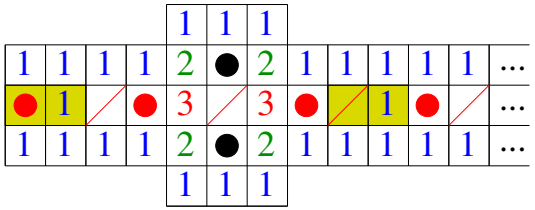
Transformer dans le circuit



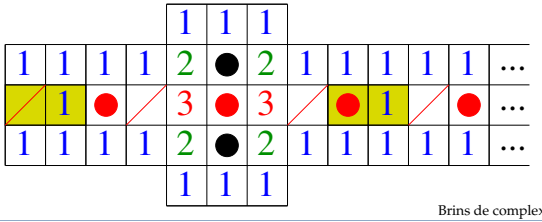
en un bloc de cases :



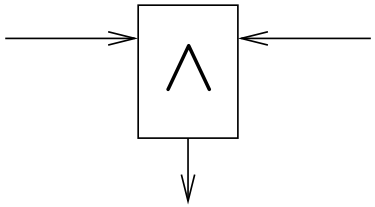
Un fil VRAI devient FAUX :



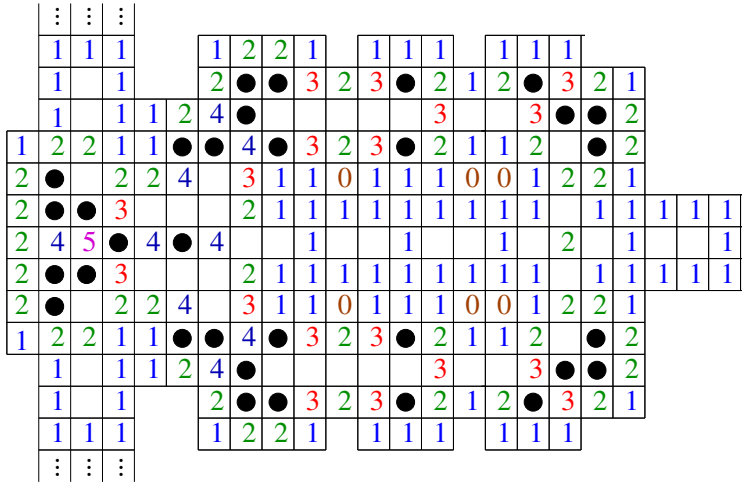
Un fil FAUX devient VRAI :



Transformer dans le circuit



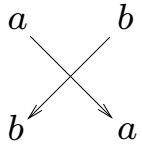
en un bloc de cases :



Que faire lorsque des fils se croisent dans le $\{x_i, \neg, \wedge\}$-circuit de départ ?

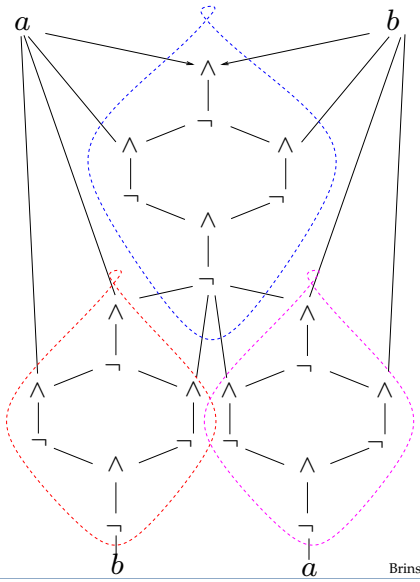
S'en débarrasser avant de commencer !

En effet, un croisement



équivaut à un $\{\neg, \wedge\}$-circuit planaire... (huh ?)

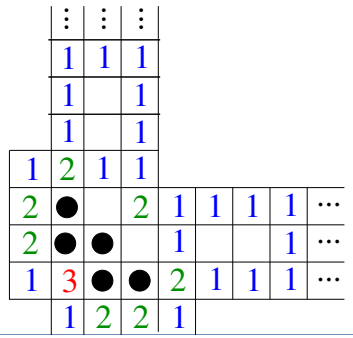
On transforme donc d'abord le "layout" du circuit de départ pour éliminer tout croisement.
Note. Ce raisonnement montre incidemment que le problème CIRCBOOL restreint aux circuits booléens planaires reste NP-ardu.



Que faire lorsqu'un fil tourne dans le "layout" rectangulaire du $\{x_i, \neg, \wedge\}$-circuit planaire ?

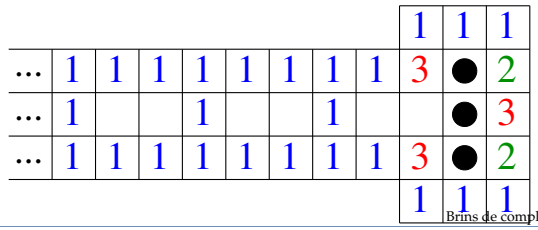
Remplacer

par l'amalgame de cases :

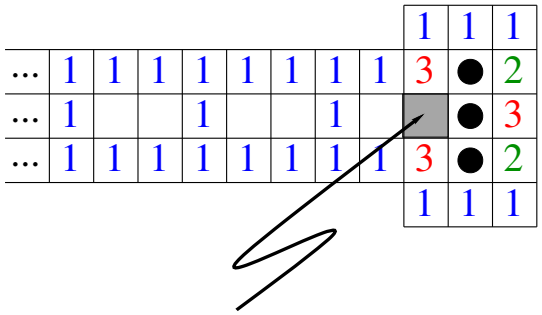


Comment se termine un fil ?

Par une fort jolie "mailloche"...

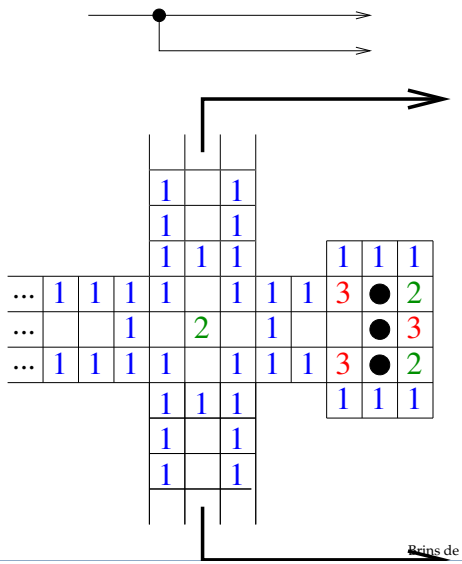


Note : le fil qui correspond à la porte de sortie du $\{x_i, \neg, \wedge\}$-circuit se termine de la même façon.



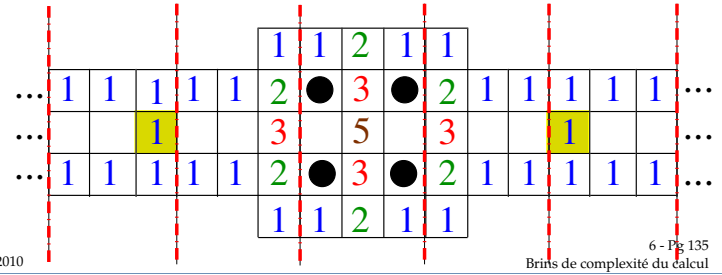
Note : position critique

Duplication d'un fil dans le circuit de départ :



Dernière technicalité : lorsque deux fils entrent dans un bloc simulant une porte $\wedge$, ils doivent être en phase, i.e. leurs 1s de référence doivent être à égale distance de la jonction du bloc.

Exemple de bidule modifiant la phase d'un fil :



Après tout ceci, il ne reste qu'à

- calculer $m \in O(|\langle C \rangle|^2)$ tel qu'un tableau T de dimension $m \times m$ englobe toute la construction,
- placer des 0 dans toutes les cases de ce tableau T qui sont à l'extérieur des fils,
- déterminer l'abscisse i et l'ordonnée j de la position critique (le long du fil de sortie).

Le résultat est $f(\langle C \rangle) = \langle T, i, j \rangle$.

On se convainc que f est calculable en temps polynômial.

Et pourquoi ça marche ?

Ça marche parce que, d'une part,

$\langle C \rangle \in$ CIRCBOOL $\Rightarrow$

$\exists$ une affectation σ qui "satisfait" C $\Rightarrow$

σ induit des mines aux positions réflétant le statut VRAI ou FAUX de chaque porte et de chaque fil $\Rightarrow$

σ induit une mine en position critique $\Rightarrow$

la position critique peut contenir une mine $\Rightarrow$

$f(|\langle C \rangle|) = \langle T, i, j \rangle \in$ DÉMINEUR,

D'autre part,

$\langle C \rangle \notin$ CIRCBOOL $\Rightarrow$

aucune affectation σ ne "satisfait" C $\Rightarrow$

toutes les affectations laissent la position critique libre $\Rightarrow$

$T(i, j)$ ne peut contenir de mine $\Rightarrow$

$f(|\langle C \rangle|) = \langle T, i, j \rangle \notin$ DÉMINEUR.

Donc CIRCBOOL $\leq_{\mathrm{P}}$ DÉMINEUR, et on applique le théorème 6.39.
Donc DÉMINEUR est NP-complet. ▲

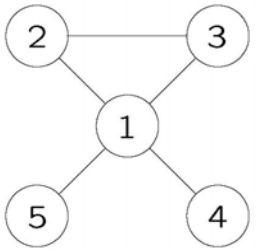
STABLE

Théorème 6.48.

STABLE = $\{ \langle G, k \rangle \mid$ le graphe non orienté G possède au moins k sommets qu'aucune arête ne relie entre eux $\}$ est NP-complet.

Exemple :

$\{3, 5\}$ est un ensemble stable du graphe G ci-dessous puisque l'arête $(3, 5)$ est absente.



D'autres stables de G sont $\{1\}$ et $\{2, 4, 5\}$ par exemple.
$\langle G, 1 \rangle \in$ STABLE.
$\langle G, 3 \rangle \in$ STABLE.
$\langle G, 4 \rangle \notin$ STABLE.

Preuve.
• STABLE $\in$ NP : exercice.
• STABLE est NP-ardu : montrons que

$$\text{COUVERTURE} \leq_{\text{P}} \text{STABLE}$$

pour appliquer le théorème 6.39. Par chance, ceci s'avère facile, la réduction étant simplement :

$$f(\langle G, s\rangle) = \langle G, m - s\rangle,$$

où m est le nombre de sommets de G.

Clairement, calculer ce $f(\langle G, s\rangle)$ est possible en temps polynômial.

Et pourquoi ça marche ?

Parce que :
$\langle G, s\rangle \in$ COUVERTURE $\Leftrightarrow$
un ensemble S de s sommets couvre toutes les arêtes de G $\Leftrightarrow$
les sommets de $G \setminus S$ ne sont reliés entre eux par aucune arête $\Leftrightarrow$
il existe $G \setminus S$ de taille $|G \setminus S| \geq m - s$ sans arêtes $\Leftrightarrow$
$f(\langle G, s\rangle) = \langle G, m - s\rangle \in$ STABLE.

Donc COUVERTURE $\leq_{\text{P}}$ STABLE.
Ainsi STABLE est NP-complet. ▲

3-COL

Théorème 6.49. 3-COL est NP-complet.

Aperçu de la preuve.

• 3-COL $\in$ NP : déjà vu.

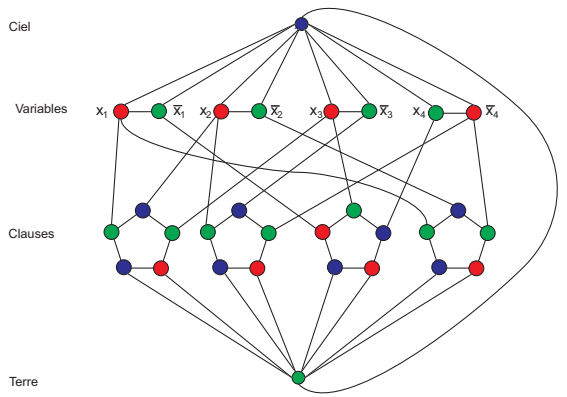
• 3SAT $\leq_{\text{P}}$ 3-COL.

Comment ?

Gadgets : gadget ciel/terre, gadget variable, gadget clause.

Coloriage : ciel est bleu, terre est verte...
comme suit :

$\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$
$\wedge (\overline{x_1} \vee x_3 \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_4})$



Lemme : le pentagone de la clause $l_1 \vee l_2 \vee l_3$ est coloriable ssi au moins un des l_i est rouge. ▲

COL

Théorème 6.50. Le langage

$$\text{COL} = \left\{ \langle G, k \rangle : G \text{ est k-coloriable} \right\}$$

est NP-complet.

Preuve.

• COL $\in$ NP : exercice.

• 3-COL $\leq_{\text{P}}$ COL : exercice.

■

Autres problèmes NP-complets

HORAIRE est un ensemble de mots $\langle M, k \rangle$ où
• M = matrice booléenne dont l'entrée (i, j) est 1 ssi le "participant" i est inscrit à "l'activité" j, et
• k = un nombre maximal de "périodes".

$\langle M, k \rangle \in$ HORAIRE ssi, en permettant des activités en parallèle, k périodes sont suffisantes pour que toutes les activités aient lieu sans qu'un participant n'ait de conflit d'horaire.

Les langages CLIQUE, HAMPATH, SUBSET-SUM, 3COL, 4COL et DOUBLE-SAT sont NP-complets, de même que, littéralement, des milliers de problèmes tirés de tous les domaines : combinatoire, VLSI, confection d'horaires, logique, etc.

Liste de milliers d'autres exemples sur le Web, dont quelques centaines dans le classique :

M. Garey et D. Johnson, *Computers and intractability–A guide to the theory of* NP*-completeness*, Freeman, 1979.

Pour chacun des langages NP-complets A de cette liste :

$$\boxed{\text{A} \in \text{P ssi P} = \text{NP}.}$$