Calculabilité et Complexité (11X008) - Printemps 2026

11. The Cook-Levin Theorem

Enseignant: Arnaud Casteigts

Assistants: M. De Francesco, M. Marseloo
Moniteurs: E. Bussod, N. Beghdadi

La stratégie habituelle pour montrer qu'un problème est NP-difficile est de choisir un autre problème NP-difficile et le réduire (en temps polynomial) au problème considéré. Mais comment a-t-on pu montrer cela la première fois, lorsqu'aucun autre problème n'existait pour effectuer une réduction ? L'objectif de ce dernier cours est de présenter le théorème de Cook-Levin, qui établit que le problème SAT est NP-difficile (bien sûr, il est aussi dans NP, donc NP-complet). Pour la définition de SAT, se référer aux cours précédents.

Ce résultat a été obtenu à peu près en même temps (et indépendemment) par Stephen Cook (canadien) et Leonid Levin (alors soviétique) dans les années 70s. La preuve que l'on connaît le mieux est celle de Richard Karp, qui a adapté leurs idées.

11.1 Principe général

La preuve utilise la définition d'origine de la classe NP, à savoir les langages décidables en temps polynomial par une machine de Turing *non-déterministe*. Pour tout langage L dans NP, il existe une telle machine M qui décide L. On montre alors que pour toute entrée w pour cette machine (une instance pour L), il existe une formule SAT ϕ de taille polynomiale en $|w|$ telle que ϕ est satisfaisable si et seulement si M accepte w.

11.2 Principe détaillé

Soit L un langage de NP. Par définition, il existe une machine de Turing non-déterministe M et une borne $B = n^{O(1)}$ sur son temps d'exécution tels que pour tout mot d'entrée w, $M(w)$ accepte ssi $w \in L$. Nous allons utiliser la description $\langle M \rangle$ et la valeur B pour construire une formule SAT satisfaisable si et seulement si $M(w)$ accepte. Par conséquent, si SAT est résoluble en temps polynomial, alors n'importe quel problème de NP l'est aussi.

11.2.1 Préambule

Sans perte de généralité, on peut se restreindre aux machines à une seule bande qui contient initialement le mot d'entrée à son début, et supposer que M possède un état initial

q_0 et un état acceptant q_{accept}. Pour un mot d'entrée w de longueur n, on note w_i le i$^{\text{ème}}$ symbole de w. Autrement dit, $w = w_1 \cdot w_2 \cdots w_n$.

Pour rappel, M accepte w si et seulement si au moins une branche d'exécution accepte. On ne connaît pas cette branche, mais on peut se la représenter mentalement comme une succession de configurations [1]. La première configuration est la configuration de départ de la machine, la dernière est une configuration dont l'état courant est q_{accept}, et chaque changement de configuration doit respecter la fonction de transition de la machine. Enfin, le nombre de configurations le long de cette branche est au plus B.

Pour rappel également, on peut décrire une configuration en donnant le contenu de la bande et en insérant l'état courant juste avant le symbole pointé par la tête de lecture. Par exemple, la configuration de départ s'écrit "$q_0 w_1 w_2 \cdots w_n \square \cdots$". La seconde configuration pourrait s'écrire (par exemple) à "$w_1 q_1 w_2 \cdots w_n \square \cdots$", etc.

Imaginons toujours cette branche acceptante (si elle existe), on peut rétrospectivement se la représenter par un tableau dont chaque ligne correspond à une configuration (la première ligne représente la configuration initiale, la seconde ligne représente la deuxième configuration, et plus généralement la ligne t correspond à la configuration au temps t). Par exemple :

Conf 1	q_0	w_1	w_2	$\cdots$	w_n	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
Conf 2	w_1	q_1	w_2	$\cdots$	w_n	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$	$\square$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
Conf accept.	w_1	w_2	$\cdots$	w_n	1	0	0	1	q_{accept}	1	0	$\square$	$\square$	$\square$

Bien sûr, on ne connaît pas à l'avance ce tableau, mais il existe si et seulement si $w \in L$.

11.2.2 La formule !

L'objectif est de créer une formule SAT ϕ qui est satisfaisable si et seulement si un tel tableau existe. La formule est grande, mais reste polynomiale en B, et donc aussi en n. Notre formule doit encoder les contraintes suivantes :

1. La première ligne du tableau correspond à la configuration initiale
2. Il existe une ligne du tableau dont la configuration est acceptante
3. Chaque transition entre deux lignes respecte la fonction de transition de M
4. Chaque case du tableau doit avoir exactement une valeur

Ces contraintes vont être encodées dans des parties différentes de la formule, qui aura alors la forme générale :

1. Pour rappel, une configuration = état courant, contenu de la bande et position de la tête de lecture.

$$\Phi = \text{Contrainte 1} \wedge \text{Contrainte 2} \wedge \text{Contrainte 3} \wedge \text{Contrainte 4}$$

Notons i les indices du tableau en abscisse et t les indices en ordonnée. Clairement, $t \leq B$. On a aussi $i \leq B$ (car la machine ne peut pas aller plus loin sur la bande en B étapes de calcul). Chaque case (i, t) du tableau doit avoir une valeur $v \in V$, où $V = \Gamma \cup Q$ est l'ensemble des valeurs possibles. Pour chaque case (i, t) et chaque valeur $v \in V$, nous définissons donc une variable $x_{i,t,v}$, qui vaudra vrai si et seulement si la case (i, t) a la valeur v. Par exemple, dans le tableau ci-dessus, $x_{0,0,q_0}$ vaut `vrai` et $x_{0,0,v}$ vaut `faux` pour tout $v \neq q_0$.

Nous pouvons maintenant traiter chaque contrainte séparément.

Contrainte 1. (La première ligne du tableau correspond à la configuration initiale.)

C'est facile, il suffit de forcer les variables de la première ligne comme suit :

$$x_{0,0,q_0} \wedge x_{1,0,w_1} \wedge \cdots \wedge x_{n,0,w_n} \wedge x_{n+1,0,\square} \wedge \cdots \wedge x_{B,0,\square}.$$

Contrainte 2. (Il existe une ligne du tableau dont la configuration est acceptante.)

Ici, on veut forcer au moins une case à contenir la valeur q_{accept}, quel que soit l'endroit (ce n'est pas forcément sur la ligne B, car B n'est qu'une *borne* sur le temps d'exécution). On peut faire cela avec un OU sur toutes les cases, en demandant que la valeur de cette case soit q_{accept}. On peut noter cela de manière plus compacte comme suit :

$$\bigvee_{i \leq B, t \leq B} x_{i,t,q_{accept}}$$

Contrainte 3. (Chaque transition entre lignes respecte la fonction de transition de M.)

Tout d'abord, observez que chaque transition de la machine ne concerne que deux lignes consécutives et que trois valeurs sur ces lignes (chaque transition est locale et effectue au plus un déplacement). Par exemple, si la configuration au temps t est $\ldots, a, b, \boxed{c, q_k, a}, b, c, a, \ldots,$ la transition suivante ne pourra affecter que les trois caractères encadrées, laissant le reste inchangé. On ne sait pas quels seront ces trois symboles pour une ligne donnée, ni quelle transition utilisée, on énumère donc toutes les possibilités.

Cette partie n'est pas détaillée, on se contentera de comprendre que c'est faisable. Essentiellement, pour chaque ligne i, pour chaque possibilité de trois cases consécutives dans cette ligne, et pour chaque valeur possible de ces trois symboles comprenant un état q_k de la machine, la ligne $i + 1$ doit avoir les mêmes valeurs en dehors de ces trois cases et doit avoir des valeurs sur ces trois cases qui correspondent à une transition valide depuis l'état q_k. Tout cela peut s'exprimer avec des $\vee$ et des $\wedge$ sur les valeurs des cases $x_{i,t,v}$. La taille de cette partie de la formule est grande, mais reste polynomiale en B (et donc en n).

Contrainte 4. (Chaque case du tableau doit avoir exactement une valeur.)

Pour avoir exactement une valeur, il faut avoir (1) au moins une valeur, et (2) au plus une valeur. Ces deux contraintes correspondant chacune à une sous-formule. Les valeurs elles mêmes correspondent soit à un symbole de l'alphabet $s \in \Gamma$, soit à un état $q \in Q$ de la machine. Cela donne :

$$\bigwedge_{i \leq B, t \leq B} \bigvee_{v \in V} x_{i,t,v}$$

Pour empêcher une case du tableau d'avoir deux valeurs différentes, on veut interdire que $x_{i,t,v}$ et $x_{i,t,v'}$ puissent être toutes les deux à `vrai` si $v \neq v'$. Comment encoder cela ? Et bien en disant que pour toute case (i, t) et pour toute paire de valeurs v et v', au moins l'une des deux variables correspondantes est fausse.

Concrètement, cela donne :

$$\bigwedge_{i \leq B, t \leq B, v \in V, v' \in V, v' \neq v} (\overline{x_{i,t,v}} \vee \overline{x_{i,t,v'}})$$

En mettant bout à bout les quatre morceaux de la formule, on obtient bien que cette formule est satisfaisable si et seulement si M accepte w.

11.2.3 Discussions

La preuve donnée ci-dessus n'est pas complète (condition 3), mais devrait suffire à se persuader qu'étant donné une machine M, on peut bien construire pour chaque entrée w une formule SAT ϕ telle que ϕ est satisfaisable si et seulement si M accepte w.

Le fait que cette formule, bien que grande, soit polynomiale, résulte du fait que $B = n^{O(1)}$, et donc B^2 ou même $B^3 = n^{O(1)}$ également. Tous les autres facteurs qui viennent se rajouter à la complexité de la formule viennent du nombre d'état, du nombre de symboles dans l'alphabet, ou du nombre de transitions de la machine, qui sont tous *constants* (ne dépendent pas de n). La formule est donc bien polynomiale en la taille de l'entrée n, ce qui implique que SAT est NP-difficile.

Fort heureusement, nous n'aurons jamais besoin de refaire une telle preuve. Une fois qu'un problème est NP-difficile, on peut l'utiliser pour montrer que d'autres problèmes le sont par des réductions depuis SAT (ou autre), comme nous l'avons fait précédemment !