Fiche 5: Machines de Turing non déterministes

Classe NP - Problèmes NP-complets

Motivation

Les algorithmes qu'on a vus dans la fiche 4 pour résoudre les problèmes HC et TS ne sont pas efficaces, ou plutôt en sont pas "traitables" en pratique [1] (c'est-à-dire, de complexité en ordre de grandeur supérieur à tout polynôme), car ils consistent à :

- énumérer *toutes* les solutions potentielles (par exemple, l'ensemble des permutations de n sommets ou villes), en nombre en général exponentiel (en c^n) ou pire (en $n!$), c'est-à-dire gigantesque même pour de relativement petites valeurs de n ;
- et ensuite, ou durant le traitement de chacune des possibilités, tester si l'une d'entre elles vérifie une certaine propriété facile à vérifier (c'est-à-dire dont la vérification peut se faire par un algorithme de complexité polynomiale, en $O(n^p)$ pour un p en général raisonnable, par exemple $p \leq 4$) [2].

Ces problèmes pourraient être résolus "efficacement" si on pouvait éviter de faire une énumération séquentielle de toutes les possibilités, en testant chacune des solutions potentielles "en parallèle" [3].

Pour mieux cerner la classe des problèmes polynomialement équivalents à HC et TS par un modèle de calcul, on fait appel au modèle des *Machines de Turing non déterministes (MTND)*, et on définit la *classe NP* des problèmes que l'on sait résoudre avec une MTND de complexité polynomiale, puis la classe des *problèmes NP-complets*.

Machines de Turing non déterministes (MTND) - classe NP

Définition 1. Une machine de Turing non déterministe (MTND) est une machine de Turing dont l'ensemble des quadruplets (qui définit son programme) n'est pas nécessairement consistant.

Une MTND peut donc contenir plusieurs quadruplets commençant par la même paire <état symbole> $<q_i\ S_j>$. L'arbre des calculs possibles (ou *arbre des exécutions possibles*) à partir de la configuration initiale est ce qu'on appelle classiquement en IA (intelligence artificielle) l'*espace de recherche* associé au problème D et à l'instance x.

Définition 2. Une MTND Z résout un problème de décision D si et seulement si, pour toute instance x de D, Z, lancée sur la configuration initiale $q_0\overline{x}$ (où $\overline{x}$ est le codage de x), est telle que :

- si $D(x)$ est VRAI, *il existe une exécution de Z qui s'arrête* avec le codage de OUI sur la bande,
- si $D(x)$ est FAUX, *toutes les exécutions de Z s'arrêtent* avec un résultat qui n'est pas le codage de OUI sur la bande.

Définition 3. La *complexité* d'une MTND Z résolvant un problème D est (l'ordre de grandeur de) la fonction qui détermine le nombre maximum d'instructions de **la plus courte exécution de** Z

1. *tractable*, en anglais.

2. Il y a des formalisms comme les TAG (grammaires d'arbres adjoints, ou *tree adjunct grammars*), pour lesquels les meilleurs algorithmes d'analyse grammaticale connus sont en $O(n^6)$, ce qui donne tout de même un coût de $64 \times |G| \times 10^6$ pour une phrase de seulement 20 mots, $|G|$ étant la taille de la grammaire ...

3. L'idée de base de la *formulation non déterministe* d'une procédure de décision est de considérer, comme on l'a déjà fait en L2 et en L3 pour les automates d'états finis et pour les automates à pile, qu'une configuration du modèle de calcul peut dériver vers plus d'une configuration, et donc qu'une configuraiton initiale donne lieu à un *arbre des calculs possibles*, tout en exprimant la procédure de façon presque classique, *comme si* il n'y avait qu'un seul calcul possible, c'est-à-dire en séparant l'expression qu'un choix est possible dans telle ou telle configuration du calcul en parallèle de ces choix, et de la synchronisation. En effet, on suppose que la procédure s'arrête dès qu'un des calculs répond oui au problème posé, et donc qu'il y a une communication instantanée entre ce calcul et tous les autres, pour arrêter leurs exécutions.

permettant de résoudre D.

On définit le pendant non déterministe des modèles de calcul RAM et pseudo-Pascal, en ajoutant la possibilité d'inclure des points de choix dans les instructions RAM et pseudo-Pascal. Ainsi, un || entre 2 instructions pseudo-Pascal indiquera un point de choix (voir un exemple dans la section suivante).
Comme dans le cas déterministe, on montre que ces différents modèles d'algorithmes non déterministes sont équivalents du point de vue de la calculabilité, et qu'un algorithme (non déterministe) de complexité polynomiale dans n'importe lequel de ces modèles de calcul peut être simulé par un algorithme (non déterministe) de complexité polynomiale dans n'importe quel autre de ces modèles de calcul.
Cela permet de définir la classe NP.

Définition 4. La classe NP est l'ensemble des problèmes de décision qui peuvent être résolus par un *algorithme non déterministe* de complexité polynomiale.

Cette classe soulève de nombreuses questions dont certaines sont encore ouvertes (c'est-à-dire non résolues). La question de la décidabilité des problèmes de NP est résolue par le théorème suivant qui montre que le non-déterminisme n'est pas plus puissant en termes de calculabilité.

Théorème 1. Soit D un problème de décision qui appartient à la classe NP. Il existe une MT déterministe Z et un polynôme π tel que Z résout D avec une complexité en $O(2^{\pi(n)})$, si n est la taille du codage de l'entrée.

Preuve. Par définition, si D est dans NP, il existe une MTND Z_{nd} dont une exécution d'au plus $q(n)$ pas de calculs résout D, où q est un polynome.
On peut construire une MT Z à 3 bandes qui résout D de la façon suivante.

1. Calcul de la taille n du mot d'entrée.
2. Calcul de $q(n)$ (en passant par exemple par un programme pseudo-Pascal pour calculer la valeur en n du polynôme q).
3. Simulation de chaque exécution de Z_{nd} jusqu'à $q(n)$ pas de calcul et arrêt dès qu'une exécution résout D.
4. Si aucune exécution tronquée à $q(n)$ pas de calcul ne retourne (le codage de) OUI, retourner (le codage de) NON comme résultat.

Pour la génération et la simulation des exécutions possibles de Z_{nd} (tronquées à $q(n)$ pas de calcul) :
— pour chaque paire $<q_i S_j>$ figurant au début d'un quadruplet de Z_{nd}, on numérote de 1 à au plus r les choix possibles, où r est le nombre maximum de choix de Z_{nd},
— une exécution de longueur l est identifiée par une séquence $[i_1, i_1, \ldots, i_l]$, où $i_j \leq r$, qui représente une exécution correspondant au i_j-ème choix pour la j-ème instruction, pour tout $j \in [1..l]$.
La bande 2 sert à générer, par ordre de longueur croissante, toutes les séquences possibles, de longueur inférieure ou égale à $q(n)$, formées de nombres inférieurs ou égaux à r.
La bande 3 est utilisée pour simuler une exécution.

Complexité de Z

1. Calcul de la taille n du mot d'entrée : $\Theta(n)$.
2. Calcul de $q(n)$: $O(q'(n))$ où q' est un polynôme (par exemple en passant par la simulation d'un programme pseudo-Pascal).
3. Génération et simulation d'au plus $r^{q(n)}$ exécutions de MT_{nd} : $r \times 1 + r^2 \times 2 + \ldots r^{q(n)} \times q(n)$ $\leq n \times q(n) \times r^{q(n)} \leq r^{q(n)+\pi'(n)} = 2^{log_2(r)\times(q(n)+\pi'(n))}$, où $\pi'(n)$ est le polynôme $n \times q(n)$, donc en $O(2^{p(n)})$ (où $p(n)$ est le polynome $log_2(r) \times (q(n) + \pi'(n))$.

Le gain potentiel du non-déterminisme en termes de complexité est une question ouverte qui s'énonce très simplement par : $\boxed{\mathbf{P} = \mathbf{NP}?}$ (on sait seulement que $\mathbf{P} \subseteq \mathbf{NP}$ puisque qu'une MT est un cas

particulier de MTND).

Tenter de répondre à cette question a suscité un très grand nombre de travaux qui ont mené à la définition et l'étude de différentes sous-classes de NP (dont celle des problèmes NP-complets). Malgré les avancées, il n'y a aujourd'hui aucune preuve ni que P = NP, ni que P $\neq$ NP.

Il y a un million de dollars à la clef (chercher sur le Web pourquoi).

Exemples

---

1. **Le problème SAT**
   - Étant donné un vocabulaire V constitué de v **variables propositionnelles**, $V = \{ p_1, \ldots, p_v \}$, une **interprétation** I de V est une application (fonction totale) de V dans les "valeurs de vérité", qui associe à chaque variable propositionnelle p_i la valeur $Vrai$ (aussi notée 1) ou $Faux$ (aussi notée 0).
   - Un **littéral** h est soit une variable propositionnelle p de V, soit la négation $\neg p$ d'une variable propositionnelle p de V.
     Notations : on note souvent un littéral négatif $\neg p$ par $-p$ ou $\bar{p}$. Deux littéraux $\neg p$ et p sont dits **conjugués**.
     Etant donnée une interprétation I :
     - Un littéral positif p_i est évalué à $Vrai$ dans I si $I(p_i) = Vrai$, à $Faux$ sinon.
     - Un littéral négatif $\neg p_i$ est évalué à $Vrai$ dans I si $I(p_i) = Faux$, à $Vrai$ sinon.
   - Une **clause** est une disjonction (finie) de littéraux : $c = h_1 \vee \ldots \vee h_k$. La *taille* de c est k.
     On considère les clauses sans répétition de littéraux.
     Notation : On utilise souvent les notations abrégées $h_1\ h_2 \ldots h_k$ ou $h_1, h_2, \ldots, h_k$ d'une clause $h_1 \vee h_2 \ldots \vee h_k$.

     Une clause peut être restreinte à un seul littéral (clause "unaire").
     On définit aussi la clause vide, notée $\Box$, insatisfaisable (toujours fausse).
   - Une **conjonction de clauses** est une formule (finie) $C = c_1 \wedge \ldots \wedge c_n$ où les c_i sont des clauses. Elle peut être restreinte à une seule clause.
     La taille d'une conjonction de clauses est la somme des tailles des clauses présentes dans la conjonction.
   - L'**évaluation** d'une clause dans une interprétation I renvoie $Vrai$ si *au moins* un de ses littéraux est évalué à $Vrai$ dans I et $Faux$ sinon.
   - L'évaluation d'une conjonction de clauses dans une interprétation I de ses variables propositionnelles renvoie $Vrai$ si *toutes* les clauses sont évaluées à $Vrai$ dans I, et $Faux$ sinon.
     Exemple : La conjonction de clauses $(\neg p_1 \vee p_3) \wedge (p_2 \vee \neg p_3) \wedge (\neg p_2)$ est de taille 5.
     Elle est évaluée à $Vrai$ dans l'interprétation I définie par : $I(p_1) = I(p_2) = I(p_3) = Faux$.
     Elle est évaluée à $Faux$ dans l'interprétation I' : $I'(p_1) = I'(p_2) = Faux, I'(p_3) = Vrai$.
     Elle est satisfaisable (I suffit pour le prouver).

   - **SAT, le problème de test de satisfaisabilité** s'énonce de la façon suivante : étant donnée une conjonction quelconque de clauses, existe-t-il une interprétation I qui satisfait cette conjonction, c'est-à-dire qui satisfait (rend vraies) toutes ses clauses.

     Théorème 2. SAT est dans NP.

     Preuve. On va écrire un algorithme pseudo-Pascal non déterministe qui résout SAT et on va montrer que sa complexité est polynomiale.
     Étant donné un ensemble de v variables propositionnelles, on code :
     - chaque variable propositionnelle par un numéro (de 1 à v) ;

— chaque littéral par le numéro de la variable propositionnelle correspondante, s'il est positif, par l'opposé du numéro de la variable propositionnelle correspondante, s'il est négatif ;
— chaque clause par la séquence ascendante des entiers relatifs codant ses littéraux ;
— une conjonction de clauses par un tableau F rempli séquentiellement en séparant par des cases contenant 0 les cases correspondant au codage de chaque clause, dans l'ordre d'apparition de chaque clause dans la conjonction, et en complétant les cases restantes par 0, avec au moins deux cases successives contenant 0.
Le codage de la conjonction de clauses de l'exemple précédent est donc :
F= | −1 | 3 | 0 | 2 | −3 | 0 | −2 | 0 | 0 | ... | 0 |
— une interprétation par un tableau I de taille v tel que $I[i]$ vaut 1 si la variable propositionnelle numéro i est interprétée à $Vrai$, 0 si elle est interprétée à $Faux$. Les codages de I et I' de l'exemple précédent sont donc : $I =$ | 0 | 0 | 0 | et $I' =$ | 0 | 0 | 1 |.

On considère le programme pseudo-Pascal suivant :

```
read(F);
/* On lit le tableau codant la conjonction de clauses fournie comme
 * entrée du programme.
 */
for i := 1 to v do I[i] := 1 || I[i] := 0  /* expression du choix possible */
i :=  1;
if F[i] = 0 then sat := 0 /* cas de la clause vide */
 else sat := 1 ;
while (F[i] <> 0) and (sat = 1) ) do
begin
c := 0  /* c vaut 1 dès que  la clause en cours de lecture est montrée satisfiable */
while F[i] <> 0 do
  begin
        if (c = 0) and  ((F[i] > 0) and (I[F[i]] = 1))
                                    or ((F[i] < 0) and (I[-F[i]] = 0))
        then c := 1;
        i := i+1;
  end
if c = 0 then sat := 0
           else  i := i+1            /* prochaine clause */
end
write(sat);
```

Ce programme (rendu non déterministe grâce à l'introduciton de l'opérateur ||)
— génère de façon non déterministe un tableau I (il y a 2^v choix pour I) représentant le codage d'une interprétation des v variables propositionnelles,
— puis renvoie 1 (codant Vrai) si la conjonction de clauses codée par le tableau F est évaluée à Vrai (et est donc satisfaisable), et 0 (codant Faux) sinon.

Il résout donc le problème SAT.

Question : Montrer qu'il est de complexité polynomiale.

2. **Théorème 3.** Le problème HC est dans NP.
Preuve. L'algorithme qui consiste à générer de façon non déterministe une permutation des sommets du graphe (il y en a $n!$ si n est le nombre de sommets) et à tester si elle satisfait la propriété d'être un circuit hamiltonien est un algorithme *non déterministe* qui résout HC. Sa complexité est polynomiale car pour chaque couple de sommets à tester (il y en a n) le test peut nécessiter de parcourir toutes les arêtes du graphe, dont le nombre est en $O(n^2)$. La complexité de l'algorithme est donc en $O(n^3)$, où n est le nombre de sommets. Comme n est en O(N) où N est la taille du graphe, la complexité est en $O(N^3)$.
3. **Théorème 4 :** Le problème TS est dans NP.
Question. Le montrer.

Le noyau dur de NP : les problèmes NP-complets

Définition 5. Un problème est NP-complet si :
- il est dans NP,
- il est NP-dur : tous les problèmes de NP lui sont polynomialement réductibles.

Théorème 5. S'il existe un problème D NP-complet et un algorithme polynomial pour le résoudre, alors P = NP.
Preuve. Soit un problème B de NP. La définition 5 dit que : $B \leq_p D$. D'après le théorème 1 de la fiche 4, si D est résoluble en temps polynomial, alors B aussi.

Corollaire. Pour démontrer qu'un problème D est NP-complet, il suffit de démontrer qu'il est dans NP, et qu'*il existe un problème D' NP-complet* tel que $D' \leq_p D$.

Question. Le démontrer.

La classe des problèmes NP-complets contient de très nombreux problèmes dont **aucun** n'a d'algorithme polynomial connu. De ce fait, il est généralement supposé que $P \neq NP$.
La conséquence du théorème 5 est que : *sous l'hypothèse que $P \neq NP$*, démontrer qu'un problème est NP-complet permet de prouver qu'il n'est pas dans P.
D'après le corollaire du théorème 5, l'outil principal pour démontrer qu'un problème est NP-complet est la réduction polynomiale entre problèmes. Cependant, il faut trouver au moins un problème NP-complet pour lequel on a une preuve directe que tout problème de NP se réduit polynomialement à lui. Le théorème de Cook nous fournit cette preuve.

Théorème de Cook. *Le problème SAT est NP-complet.*

Idée de la preuve. On a déjà démontré (Théorème 2) que SAT est dans NP.
Pour démontrer qu'il est NP-dur, on considère un problème $\mathcal{B}$ NP quelconque, et on veut montrer qu'il existe une fonction f de $\mathcal{P}$ (calculable par une MT déterministe en temps polynomial) telle que $\mathcal{B} \leq_p SAT$ via f.
Pour toute instance I de $\mathcal{B}$, $f(I) = F$ doit être une formule satisfaisable **ssi** $\mathcal{B}(I)$ est vraie.
En fait, on va construire F non seulement à partir de I, mais aussi de la (d'une !) MT non-déterministe Z calculant $\mathcal{B}(I)$ en temps $T(N) = O(\pi(N))$, où π est un polynôme et N la taille de l'instance I de $\mathcal{B}$.
Supposons que $Z = (E, \Sigma, T)$, où les états sont un sous-ensemble fini E de l'ensemble universel d'états $Q = \{q_0, \ldots q_n, \ldots\}$, les symboles sont un sous-ensemble fini Σ de l'ensemble universel de symboles $S = \{S_0, \ldots S_m, \ldots\}$, et T est la table de transition (l'ensemble des quadruplets). Supposons que l'instance I est codée par $S_{j_1} \ldots S_{j_N}$.
La configuration initiale est alors $c_0 = q_0\ S_{j_1} \ldots S_{j_N}$.

On va écrire une formule F exprimant l'existence d'un calcul terminal de Z s'arrêtant en q_n après au maximum $\pi(N)$ pas de calcul.
On sait que, dans ce temps maximum, la tête de lecture-écriture de Z ne peut parcourir plus de $\pi(N)$ cases à gauche ou à droite.
On conviendra que la case pointée par Z au début du calcul (en c_0) est la case $\pi(N)$ et on posera $M = \pi(N)$.

Atomes propositionnels de F

Nous aurons besoin de 4 indices :

i	$(0 \leq i \leq n)$	sera le numéro d'un état.
j	$(0 \leq j \leq m)$	sera le numéro d'un symbole.
c	$(0 \leq c \leq 2\ M)$	sera le numéro d'une case accessible durant le calcul.
t	$(0 \leq t \leq M)$	sera le numéro d'un pas de calcul (et donc d'une configuration c_t).

Les atomes propositionnels de F seront :

A tomes		Signification	Nombre
B_{jct}	$\equiv$	le symbole S_j est dans la case c à l'instant t.	$(m+1) \times (2M+1) \times (M+1)$
L_{ct}	$\equiv$	la tête pointe sur la case c à l'instant t.	$(2M+1) \times (M+1)$
E_{it}	$\equiv$	Z est dans l'état q_i à l'instant t.	$(n+1) \times (M+1)$

Formule F

La formule F cherchée doit exprimer l'existence d'un calcul de longueur inférieure ou égale à M commençant par la configuration initiale et arrivant dans une configuration finale d'état q_n. Si l'on observe Z au temps $t = M$, elle est nécessairement dans une configuration finale, puisque, par hypothèse, elle doit s'arrêter en au plus M pas de calcul. On construit donc F pour "conserver" toute configuration finale jusqu'au temps M.

Pour être sûr qu'il n'existera de valuation satisfaisant F que si l'unique calcul à partir de c_0 conduit à q_n, il faut aussi exprimer dans F les "conditions universelles" disant que, par exemple, une case ne peut pas recevoir deux symboles distincts.

On aura $F = F_{univ} \wedge F_{init} \wedge F_{transit} \wedge F_{final}$, avec :

Formule		Signification	*Longeur*	
F_{univ}	$\equiv$	Chaque case porte un symbole et un seul. Exactement une case est pointée par la tête. Z est dans un état et un seul.	$\leq M + 2(M+1)^2(m^3+4) + 2n^2$	
F_{init}	$\equiv$	$c_0 = q_0\ S_{j_1} \ldots S_{j_N}$	$2M+5$	
F_{final}	$\equiv$	Z est en q_n à l'instant M si elle accepte.	1	
$F_{transit}$	$\equiv$	La suite des configurations respecte T : Z s'arrête à un instant $f \leq M$. Si Z est arrêtée en $t < M$, $c_{t+1} = c_t$. Réécriture d'un symbole Mouvement vers la droite Mouvement vers la gauche	$2M-1$	

$$
\begin{array}{lcl}
F_{univ} & = & \bigwedge_{t=0}^{M} \quad (\bigwedge_{c=0}^{2M+1}(\bigvee_{j=0}^{m}(B_{jct} \bigwedge_{j' \neq j} \neg B_{j'ct})) \\
 & & \qquad \wedge \bigvee_{c=0}^{2M+1}(L_{ct} \bigwedge_{c' \neq c} \neg L_{c't}) \\
 & & \qquad \wedge \bigvee_{i=0}^{N}(E_{it} \bigwedge_{i' \neq i} \neg E_{i't})) \\
F_{init} & = & E_{00} \wedge L_{M0} \ \bigwedge_{c=0}^{M-1} B_{0c0} \ \bigwedge_{c=M}^{M+N-1} B_{j_{c-M+1}c0} \ \bigwedge_{c=M+N}^{2M+1} B_{0c0} \\
F_{final} & = & E_{Mn} \\
F_{transit} & = & \bigvee_{f=0}^{M} \bigvee_{\{i,j|(q_i\ S_j\ X\ Y \notin T)\}} E_{nf} \\
\wedge \bigwedge_{t=0}^{M-1} & & (\bigwedge_{\{i,j\ |\ (q_i\ S_j\ X\ Y) \in T\}} \bigwedge_{c=0}^{2M+1} \bigvee_{j=0}^{m}(\neg B_{jct} \vee B_{jc(t+1)})) \\
 & & \bigwedge_{\{i,j,k,l\ |\ (q_i\ S_j\ S_k\ q_l) \in T\}} \\
 & & (\neg(E_{it} \wedge L_{ct} \wedge B_{jct}) \\
 & & \qquad \vee (E_{l(t+1)} \wedge L_{c(t+1)} \wedge B_{kc(t+1)}) \bigwedge_{d \neq c} \bigwedge_{b=0}^{M}(\neg B_{bdt}) \vee B_{bd(t+1)})) \\
 & & \bigwedge_{\{i,j,l\ |\ (q_i\ S_j\ R\ q_l) \in T\}} \bigwedge_{k=0}^{m} \\
 & & (\neg(E_{it} \wedge L_{ct} \wedge B_{jct} \wedge B_{k(c+1)t}) \\
 & & \qquad \vee (E_{l(t+1)} \wedge L_{(c+1)(t+1)} \bigwedge_{d=0}^{2M+1} \bigwedge_{b=0}^{M}(\neg B_{bdt}) \vee B_{bd(t+1)})) \\
 & & \bigwedge_{\{i,j,l\ |\ (q_i\ S_j\ L\ q_l) \in T\}} \bigwedge_{k=0}^{m} \\
 & & (\neg(E_{it} \wedge L_{ct} \wedge B_{jct} \wedge B_{k(c-1)t}) \\
 & & \qquad \vee (E_{l(t+1)} \wedge L_{(c-1)(t+1)} \bigwedge_{d=0}^{2M+1} \bigwedge_{b=0}^{M}(\neg B_{bdt}) \vee B_{bd(t+1)}))
\end{array}
$$

F est calculable en temps polynomial à partir de I

Le programme consiste à écrire F à partir du schéma ci-dessus. Chaque quantificateur borné donne lieu à une boucle `pour` .

Si l'on compte en fonction des atomes, le temps de calcul est polynomial en fonction de la longueur de la description de l'instance I de $\mathcal{B}$, ajoutée à la longueur (fixe) de la description de la machine Z résolvant le problème $\mathcal{B}$.