Examen Partiel pour Advanced Complexity (M1)

Nov. 12th, 2026. Durée : 2h

Exercice : Du temps et de l'espace

En cours il a été démontré que $\mathsf{NTIME}(f) \subseteq \mathsf{SPACE}(f)$ pour toute fonction « propre » $f : \mathbb{N} \to \mathbb{N}$. La preuve donnée utilisait l'hypothèse qu'on peut calculer $f(n)$ en utilisant une mémoire de travail bornée par $f(n)$, ce qui était garanti par la condition « f est propre ». La preuve a 3 composants : (1) une *construction* permettant de simuler une machine non déterministe par une machine déterministe ; (2) un *argument de terminaison* prouvant que cette simulation termine ; (3) un *argument de correction* prouvant que la simulation est fidèle et n'utilise pas trop d'espace mémoire. Dans cet exercice on cherche à généraliser le résultat d'inclusion à une famille plus large de fonctions f. On demande une réponse en trois parties distinctes.

1. Donnez une construction (comme la partie (1) de la preuve évoquée plus haut) montrant que $\mathsf{NTIME}(f) \subseteq \mathsf{SPACE}(f)$ pour f quelconque. Comme il s'agit d'un algorithme soyez précis dans la description des différentes étapes et de leur enchaînement. Si vous avez besoin d'hypothèses sur f (p.ex. f monotone, ou f récursive, ou ..) mentionnez les explicitement.

   Solution:
   Si $L \in \mathsf{NTIME}(f)$ alors il existe une NTM $\mathcal{M}$ qui reconnaît L, qui répond en temps $f(n)$ et qui donc n'utilise pas plus de $f(n)$ mémoire de travail. Pour simplifier on va d'abord supposer que le non-déterminisme est *binaire* c.-à-d. que dans chaque configuration, $\mathcal{M}$ a le choix entre au plus deux règles de transition.

   Rappelons *la construction classique* : elle transforme $\mathcal{M}$ en une machine déterministe $\mathcal{M}'$ qui, avant de décider si $x \in L$, calcule $n = |x|$, calcule $f(n)$ et stocke le mot $0^{f(n)}$ sur un ruban de travail supplémentaire r qui contiendra uniquement des mots de $\{0,1\}^*$. Après ces préparatifs $\mathcal{M}'$ se comporte comme $\mathcal{M}$ mais, à chaque étape i où deux règles de transition sont possibles, $\mathcal{M}'$ utilise la i-ème lettre de r pour choisir d'appliquer la première règle (si $r[i] = 0$) ou la seconde (si $r[i] = 1$). Ainsi la simulation est déterministe (guidée par le mot dans r). Si cette simulation atteint un état où $\mathcal{M}$ accepte x alors $\mathcal{M}'$ accepte. Si elle atteint un état où $\mathcal{M}$ rejette alors $\mathcal{M}'$ incrémente le contenu de r vu comme un entier en binaire et redémarre alors une nouvelle simulation de $\mathcal{M}$ sur x en faisant cette fois-ci des choix (légèrement) différents puisque r a été modifié. Quand tous les mots de $\{0,1\}^{f(n)}$ auront été essayés sur r sans parvenir à accepter x, alors enfin $\mathcal{M}'$ termine et rejette.

   La nouvelle construction ne calcule pas $f(n)$ et démarre avec $r = \epsilon$ vide. La simulation démarre comme précédemment avec la modification suivante : à chaque fois que $\mathcal{M}'$ demande à lire la valeur de $r[i]$ et constate que $i > |r|$ (c.-à-d., le mot dans r

est trop court), on interrompt la simulation et on recommence tout depuis le début après avoir remplacé tous les $r[i]$ par des 0 et leur avoir adjoint un 0 supplémentaire (on a donc allongé le contenu de r).

NB : si le non-déterminisme n'est pas binaire il reste borné par une constante k (qui dépend de $\mathcal{M}$ mais pas de x) et on écrira sur r des mot de $\{0, 1, \ldots, k-1\}^*$.

2. Prouvez la terminaison de votre simulation. Sans redonner la définition complète des classes $\mathsf{NTIME}(f)$ et $\mathsf{SPACE}(f)$, mentionnez explicitement les points de cette définition qui sont nécessaires à votre preuve.

Solution:

Notre définition pour $L \in \mathsf{NTIME}(f)$ exige que toutes les exécutions de $\mathcal{M}$ sur un input x terminent et soient toutes de longueur au plus $f(n)$, donc une fois que r aura atteint une longueur suffisante on ne le rallongera plus, on testera toutes les exécutions possibles (il y en a un nombre fini) et on terminera forcément.

3. Prouvez la correction de votre simulation, là aussi en explicitant les hypothèses utilisées.

Solution:

Évidemment $\mathcal{M}'$ accepte x ssi $\mathcal{M}$ l'accepte. Le point plus délicat concerne l'espace mémoire utilisé, en particulier si f est petit (imaginons $f(n) = \log\log n$). Notre définition des TMs leur interdit de modifier le ruban d'entrée, et celle de $\mathsf{SPACE}(f)$ ne prend pas en compte le ruban d'entrée dans la mémoire utilisée (ni le nombre d'états de contrôle de la TM, ni la taille des positions des têtes de lecture). Donc $\mathcal{M}'$ peut toujours recommencer un nouveau calcul sur x *sans nécessiter aucune mémoire auxiliaire pour remettre à zéro les rubans de travail hérités de* $\mathcal{M}$. Le ruban supplémentaire n'utilise que $f(n)$ cases au plus, donc $\mathcal{M}'$ prouve $L \in \mathsf{SPACE}(2f)$ et on conclut avec le théorème de compression.

Problème : Complexité paramétrée

On s'intéresse ici à des *problèmes de décision paramétrés*. Un tel problème est un sous-ensemble $L \subseteq A^* \times \mathbb{N}$ où A est un alphabet fini quelconque. On peut voir L comme une famille indexée $(L_k)_{k\in\mathbb{N}}$ de langages. Une *instance* du problème est un couple (x, k) et on s'intéresse aux algorithmes qui décident si $(x, k) \in L$. L'entier k est le *paramètre*.

Considérons par exemple le problème bien connu SAT qui demande de décider si une formule booléenne φ en forme normale conjonctive (CNF), aussi appelée « forme clausale », est satisfaisable. Ce problème existe aussi dans une version paramétrée : $\mathsf{kVars{-}SAT}$ est l'ensemble des couples (φ, k) tels que φ utilise au plus k variables booléennes distinctes (et est satisfaisable).

Quand on considère les algorithmes résolvant un problème paramétré, on mesure le temps d'exécution comme dépendant à la fois de $|x|$ et de k. Formellement, on dit que $L \subseteq A^* \times \mathbb{N}$ est dans $\mathsf{FPTIME}(f)$ s'il existe une machine de Turing déterministe (une DTM) $\mathcal{M}$ qui reconnaît L et qui répond toujours en au plus $h(k)\cdot f(n)$ pas de calcul, avec n la longueur de x, et où $h : \mathbb{N} \to \mathbb{N}$ est une fonction calculable (qui dépend de $\mathcal{M}$). Comme d'habitude on utilise la notation grand-O pour des unions telles que $\mathsf{FPT} \stackrel{\text{def}}{=} \mathsf{FPTIME}(n^{O(1)}) \stackrel{\text{def}}{=} \bigcup_{c\in\mathbb{N}} \mathsf{FPTIME}(n^c)$.
Autrement dit, $L \in \mathsf{FPT}$ (on dit aussi que « L est *polynomial à paramètre fixé* ») si, *pour tout k fixé*, la DTM décide si $(x, k) \in L$ en temps $O(p(|x|))$ pour un polynôme p, et si le degré de p ne dépend pas de k. Ce qui dépend de k est le coefficient multiplicatif $h(k)$ dont on exige seulement qu'il soit calculable.

4. Montrez que $\mathsf{kVars{-}SAT}$ est dans FPT.

   Solution:
   Algo simple : (1) on vérifie que x est bien une CNF, on construit Var, l'ensemble des variables qui apparaissent dans x, et on rejette si $|Var| > k$; (2) sinon on énumère toutes les valuations sur Var pour voir si l'une rend x vrai. La première étape est en temps polynomial, la seconde en temps $O(2^k \cdot n)$.

Il existe d'autres façons de paramétrer SAT. P.ex. $\mathsf{kWide{-}SAT}$ est l'ensemble des (φ, k) tels que $\varphi \in \mathsf{SAT}$ et tels que les clauses de φ ont au plus k littéraux.

5. Montrez que $\mathsf{kWide{-}SAT} \in \mathsf{FPT}$ si et seulement si $\mathsf{P} = \mathsf{NP}$.

   Solution:
   ($\Rightarrow$) Si $\mathsf{kWide{-}SAT} \in \mathsf{FPT}$ alors, en fixant $k = 3$, on obtient un algorithme en temps polynomial pour $\mathsf{3SAT}$ qui est NP-complet. On déduit $\mathsf{NP} \subseteq \mathsf{P}$.
   ($\Leftarrow$) Si $\mathsf{P} = \mathsf{NP}$, on peut décider SAT par un algorithme déterministe en temps polynomial $p(n)$, et on peut donc décider si $(\varphi, k) \in \mathsf{kWide{-}SAT}$ en temps $p(n) + O(n + k)$ puisque il suffit d'ajouter un test comparant k au nombre de littéraux de φ. On déduit facilement $\mathsf{kWide{-}SAT} \in \mathsf{FPT}$ (cf. question suivante).

6. Montrez que $L \in \mathsf{FPT}$ ssi il existe un polynôme p, une fonction calculable h, et une DTM $\mathcal{M}$ qui décide si $(x, k) \in L$ en au plus $p(|x|) + h(k)$ pas de calcul.

   Solution:
   ($\Rightarrow$) : Soit $\mathcal{M}$ décidant L en temps borné par $p(n) \cdot h(k)$, alors c'est aussi borné par $p(n)^2 + h(k)^2$ (puisque $a \cdot b \leq a^2 + b^2$ pour tous $a, b \in \mathbb{N}$). On a bien p^2 polynomial

et h^2 calculable puisque p et h le sont.
($\Leftarrow$) : Si le temps est borné par $p(n)+h(k)$ alors il est borné par $(p(n)+1)\cdot(h(k)+1)$.

On dit qu'un problème paramétré L est « en PTIME pour x assez grand » s'il existe une fonction calculable s et une DTM $\mathcal{M}$ fonctionnant en temps polynomial qui décide correctement si $(x,k) \in L$ quand $|x| \geq s(k)$ (dans le cas où $|x| < s(k)$, $\mathcal{M}$ peut accepter ou rejeter (x,k) de façon arbitraire).

7. Montrez que $L \in$ FPT si, et seulement si, L est décidable et en PTIME pour x assez grand.

Solution:
($\Leftarrow$) Supposons L « en PTIME pour x assez grand ». Il existe alors une DTM $\mathcal{M}$ en temps polynomial $p_{\mathcal{M}}$ et un s calculable tels que $\mathcal{M}$ décide $(x,k) \in L$ quand $|x| \geq s(k)$. Si on suppose de plus que L est décidable, il existe une DTM $\mathcal{M}_0$ pour L mais on n'a pas de borne sur son temps de calcul.
On utilise alors l'algorithme suivant sur un input (x,k) :
(1) Calculer $s(k)$.
(2) Si $s(k) \leq |x|$:
(2.1) alors décider $(x,k) \in L$ en utilisant $\mathcal{M}$,
(2.2) sinon décider $(x,k) \in L$ en utilisant $\mathcal{M}_0$.

La correction de cet algorithme est évidente. Quid de sa complexité ? Il prend un temps $T_s(k)$ pour l'étape (1), $O(s(k)+|x|)$ pour la comparaison à l'étape (2), et enfin soit un temps polynomial $p_{\mathcal{M}}(k,|x|)$ pour (2.1) ou un $T_0(k)$ pour (2.2). De fait, l'étape (2.2) ne concerne qu'un nombre fini de valeurs possibles pour (x,k) une fois que k est fixé et qu'on satisfait $|x| < s(k)$, donc on peut définir (et calculer !) $T_0(k)$ comme le temps maximum utilisé par $\mathcal{M}_0$ sur ces couples (x,k). Au final notre algorithme pour L est en temps borné par $T_s(k) + O(s(k)) + O(|x|) + p_{\mathcal{M}}(k,|x|) + T_0(k)$. En invoquant la question 6, on voit $L \in$ FPT avec un h en $O(s + T_0)$.
($\Rightarrow$) Tout d'abord $L \in$ FPT nous donne la décidabilité de L. De plus, avec la question 6, on sait qu'il existe une DTM $\mathcal{M}$ décidant L en temps borné par $h(k)+p(|x|)$ pour un h calculable et un polynôme p. On ne peut pas immédiatement dire que $\mathcal{M}$ montre « L en PTIME pour x assez grand » puisque $\mathcal{M}$ n'est pas forcément en temps polynomial.
On considère alors l'algorithme suivant sur un input (x,k) :
(1) On simule le calcul de $h(k)$ pendant $|x|$ étapes au plus. Si ce calcul n'a pas terminé on rejette (x,k).
(2) Si on est parvenu à calculer $h(k)$ en au plus $|x|$ étapes, on compare $h(k)$ et $|x|$ et on rejette si $h(k) > |x|$.
(3) Sinon on utilise $\mathcal{M}$ pour décider si $(x,k) \in L$.
Cet algorithme donne une réponse correcte quand $|x| \geq \max(h(k), T_h(k))$ où $T_h(k)$ est le temps de calcul de $h(k)$. On écrira plus simplement « quand $|x| \geq h'(k)$ ». L'étape (1) prend un temps $|x|^{O(1)}$, suffisant pour simuler $|x|$ étapes d'une TM fixée. L'étape (2) prend un temps $O(|x|)$ pour comparer deux nombres même si k, écrit en binaire par exemple, est exponentiellement plus grand que $|x|$. L'étape (3) prend un temps $h(k)+p(|x|)$, qui est $O(|x|)+p(|x|)$ puisque on ne l'invoque que si $h(k) < |x|$. Au final notre algorithme répond en temps polynomial $|x|^{O(1)}+p(|x|)$ sur tout input (x,k), et sa réponse est correcte quand $|x| \geq h'(k)$, avec un h' calculable.

Algorithmes pour complexité paramétrée

Un exemple de problème paramétré est kNodes−VertexCover, l'ensemble des (G, k) tels que G est (le codage d') un graphe non orienté $G = (V, E)$ admettant une couverture des sommets $C \subseteq V$ de taille $|C| \leq k$. On rappelle que C est une couverture si $\forall e \in E$: $e \cap C \neq \emptyset$ (c.-à-d., chaque arête $\{x, y\}$ de G a au moins une extrémité dans C).

8. Pour kNodes−VertexCover, on considère l'algorithme « force brute » naïf suivant : « essayer toutes les parties $C \subseteq V$ de taille k et vérifier si l'une d'entre elles couvre bien toutes les arêtes ». Cet algorithme prouve-t-il kNodes−VertexCover $\in$ FPT ?

Solution:
Non. Il y a $\binom{|V|}{k}$ candidats pour C. Ce nombre est en $\Omega(|V|^k)$ pour k fixé, et n'est donc pas borné par un polynôme de degré indépendant de k.

On considère maintenant l'algorithme « plus élaboré » suivant :
Pour une instance (G, k) avec $G = (V, E)$:
1 : On calcule $U = \{v \in V \mid d(v) > k\}$, où $d(v)$ est le degré du sommet v, c.-à-d., le nombre d'arêtes de E qui touchent v. Si $|U| > k$ l'algorithme rejette.
2 : Sinon on pose $k' \stackrel{\text{def}}{=} k - |U|$ et on construit $G' = (V', E')$, défini comme « G *après avoir retiré tous les sommets de* U*, toutes les arêtes touchant* U*, et tous les sommets qui se retrouvent isolés après cette opération* ». Maintenant si $|E'| > kk'$ l'algorithme rejette.
3 : Sinon on vérifie si G' a une couverture C' de taille $\leq k'$ avec l'algorithme naïf de la question précédente et on accepte ou on rejette (G, k) selon que (G', k') est accepté ou rejeté.

9. Montrez que ce nouvel algorithme est correct (c.-à-d., il donne toujours la bonne réponse) pour kNodes−VertexCover. *Indication :* puisque l'algorithme a 3 parties, il est utile de découper l'argument de correction selon 3 cas.

Solution:

1. Supposons qu'il existe une couverture C de taille $\leq k$. Tout sommet $u \in U$ doit se trouver dans C sinon les extrémités des arêtes adjacentes à u seront trop nombreuses pour être toutes couvertes par C. Donc si $|U| > k$, toute couverture est de taille $> k$ et l'algorithme est correct puisqu'il rejette.
2. Supposons $|U| \leq k$ et supposons qu'il existe une couverture C de taille $\leq k$. Par construction, E' contient toutes les arêtes qui ne sont pas déjà couvertes par U, donc G admet une couverture de taille $\leq k$ ssi G' admet une couverture C' de taille $\leq k - |U| = k'$: on a $C = U \cup C'$. Puisque les sommets de G' sont de degré $\leq k$ ils couvrent chacun au plus k arêtes. Si $|E'| > kk'$ alors G' n'a pas de couverture de taille k' et l'algorithme est correct puisqu'alors il rejette.
3. Si $|U| \leq k$ et $|E'| \leq kk'$, il nous reste à vérifier que G' admet une couverture de taille $\leq k'$, ce que fait bien notre algorithme dans sa dernière partie.

10. Quelle est la complexité de l'algorithme plus élaboré. Est-ce qu'il permet de conclure que kNodes−VertexCover $\in$ FPT ?

Solution:
Notons n pour $|G|$. Les étapes 1 et 2 sont polynomiales en n. Lors de l'étape 3 il y a au plus kk' arêtes dans E', donc au plus $2kk'$ sommets dans V' (les sommets isolés ont été retirés). On teste donc au plus $\binom{2kk'}{k'}$ ensembles de taille k' dans G'. Chaque

ensemble est testé en temps $O(n)$. Puisque $\binom{2kk'}{k'} \leq (2k^2)!$ (en comptant large) est ici un facteur multiplicatif, la complexité totale de cet algorithme est polynomiale en n avec un polynôme de degré indépendant de k.

On revient aux versions paramétrées de SAT : kClauses−SAT contient tous les (φ, k) tels que $\varphi \in$ SAT a au plus k clauses.

11. Montrez que kClauses−SAT est dans FPT.

Solution:
Voici un algorithme possible sur un input (φ, k) : (1) on teste si φ a au plus k clauses (sinon on rejette), (2) on retire de φ toutes les clauses ayant plus de k littéraux, ce qui donne une sous-formule φ' ayant au plus k^2 variables, (3) on accepte ssi φ' est satisfaisable, ce qu'on décide en essayant toutes les valuations possibles (il y en a au plus 2^{k^2}).
La correction de cet algorithme repose sur le lemme suivant : « *Supposons φ de la forme $C \wedge \psi$ où C est une clause ayant strictement plus de littéraux que le nombre m de clauses de ψ. Alors φ est satisfaisable ssi ψ l'est* ». Effectivement, si ψ est satisfaisable par une valuation v il suffit de choisir un littéral satisfait par clause, soit au plus m variables *cruciales*, et toute valuation qui coïncide avec v sur ces $\leq m$ variables satisfera ψ elle aussi. On choisit alors dans C un littéral $\ell = \pm x$ indépendant des variables cruciales. On modifie v de sorte que $v(\ell) = \top$. On satisfait alors C tout en respectant les valeurs des variables cruciales, donc on satisfait $C \wedge \psi$. [NB : si la clause C contient un littéral et sa négation elle ne contient peut-être pas assez de variables distinctes mais C est trivialement valide.]