Calculabilité et Complexité (11X008) - Printemps 2024

9. Classe NP

Enseignant: Arnaud Casteigts *Exercices: A. Berger, M. De Francesco, L. Heiniger*

9.1 Définitions

Nous avons vu les classes génériques TIME et SPACE qui caractérisent les langages que l'on peut décider de manière déterministe en utilisant une certaine quantité de temps ou d'espace, ainsi que leurs analogues non-déterministes NTIME et NSPACE.

Intéressons-nous au cas particulier où les ressources sont *polynomiales*. Pour l'espace, le théorème de Savitch nous dit que pour tout $f(n) \geq n$, on a $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$. Cela implique que $\mathsf{PSPACE} = \mathsf{NPSPACE}$. Qu'en est-il du temps ? Rappelons les définitions.

- P : Langages qui peuvent être décidés en temps polynomial par une machine de Turing *déterministe* (autrement dit $\mathsf{TIME}(n^{O(1)})$).
- NP : Langages qui peuvent être décidés en temps polynomial par une machine de Turing *non-déterministe* (autrement dit $\mathsf{NTIME}(n^{O(1)})$).

Bien que les machines non-déterministes n'existent pas, la classe NP est très importante. La raison est que cette classe admet une autre définition *déterministe*, qui a de nombreuses applications.

- NP : Langages qui admettent des *preuves d'appartenance* vérifiables en temps polynomial.

Plus précisément, un langage L est dans NP si pour tout mot $w \in L$, il existe une preuve vérifiable en temps polynomial par une machine *déterministe* que w est bel et bien dans L (une définition plus formelle sera donnée plus bas). Une telle preuve est appelée un **certificat positif**. Il n'est pas évident à première vue que les deux définitions sont équivalentes, mais c'est bien le cas. Pour commencer, voyons quelques exemples.

9.2 Exemples de langages dans NP

Plutôt que de définir les langages eux-mêmes, nous les formulons ici sous forme de question OUI/NON (problème de décision). C'est équivalent, car les **instances positives** du problème (ensemble de tous les mots d'entrée pour lesquelles la réponse est OUI) forment un langage.

- Connexité : Étant donné un graphe $G = (V, E)$, ce graphe est-il connexe ?

  Clairement, Connexité $\subseteq$ NP, car si la réponse est oui pour un graphe G, alors une preuve peut consister en un ensemble de chemins (un chemin pour chaque paire de sommets), qui permettent facilement à quelqu'un de vérifier que chaque paire de sommet est bien reliée, et donc le graphe est connexe. La taille de cette preuve, ainsi que son temps de vérification, sont bien polynomiaux en la taille de l'entrée $|G|$.

- 3-coloration : Étant donné un graphe $G = (V, E)$, ce graphe est-il 3-colorable ? Autrement dit, peut-on affecter à chaque sommet une couleur (parmi 3) telle que toute paire de sommets voisins a des couleurs différentes.

  Là aussi, 3-coloration $\subseteq$ NP, car si une telle coloration existe, elle constitue elle-même une preuve rapide à vérifier : il suffit de parcourir toutes les arêtes et de vérifier que les deux sommets correspondants ont à chaque fois une couleur différente.

- SAT : Étant donné une formule du type $(x_1 \vee \neg x_2 \vee x_4) \wedge (x_2 \vee \neg x_4) \wedge \cdots$, existe-t-il des valeurs `vrai`/`faux` pour chaque variable x_i telles que la formule est satisfaite ?
  Là encore, SAT $\subseteq$ NP, car si de telles valeurs existent, alors elles constituent elles-mêmes une preuve rapide à vérifier.

- Primalité : Étant donné un nombre N, ce nombre est-il premier ?
  Il se trouve que Primalité $\subseteq$ NP. Cette fois, l'argument est non-trivial, nous ne décrirons pas. Mais réfléchissez-y un peu. Il est plus facile de voir que Non-Primalité $\in$ NP : les facteurs eux-mêmes servant de preuve.

Attention : l'existence d'une preuve n'implique pas que cette preuve est facile à *trouver*. Pour plusieurs des exemples ci-dessus, nous ne connaissons pas d'algorithmes capables de trouver une telle preuve rapidement. La classe NP impose seulement que si une instance est positive, alors une telle preuve *existe*.

9.3 Équivalence entre les deux définitions de NP

Pourquoi les deux définitions de la classe NP sont équivalentes ?

Prenons l'exemple de la 3-coloration. On peut aisément construire une machine non-déterministe qui choisit (de manière non-déterministe) parmi les 3 couleurs possibles pour le premier sommet, ce qui donne naissance à 3 branches d'exécution. Dans chaque branche, la machine choisit alors de manière non-déterministe la couleur du second sommet, etc.. Au final, cela donne un arbre d'exécution semblable à celui de la figure 1. Puis, chaque branche dans laquelle la coloration est désormais complète peut vérifier si cette coloration est correcte ou non, sans nécessiter de non-déterminisme supplémentaire.

Plus généralement, notons NP_1 la définition de NP basée sur le non-déterminisme et NP_2 la définition de NP basée sur l'existence d'une preuve vérifiable rapidement (le nom de ces

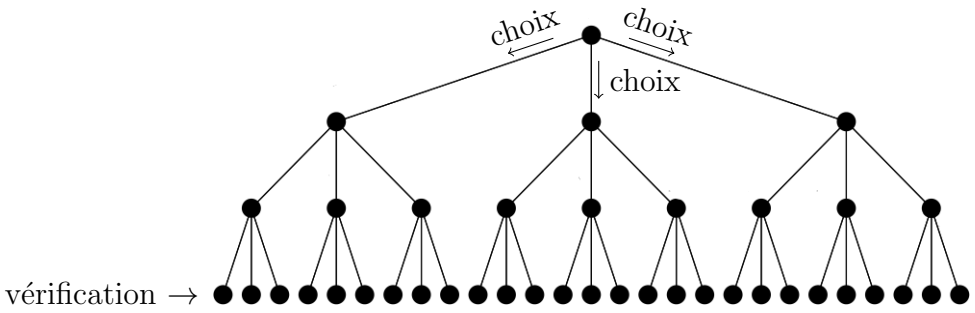

FIGURE 1 – Exploration non-déterministe d'une 3-coloration.

deux classes a été inventé pour les besoins du cours, ne les cherchez pas ailleurs). On peut démontrer l'équivalence comme suit.

Theorème 9.1. $NP_1 = NP_2$

Preuve. Il suffit de montrer que $\mathsf{NP}_2 \subseteq \mathsf{NP}_1$ et $\mathsf{NP}_1 \subseteq \mathsf{NP}_2$.

- $\mathsf{NP}_2 \subseteq \mathsf{NP}_1$: Soit L un langage dans NP_2. Par définition, pour tout $w \in L$, il existe une preuve de taille polynomiale et vérifiable en temps polynomial que w appartient bien à L. Une machine non-déterministe peut *deviner* cette preuve en explorant toutes les possibilités, puis la vérifier, comme pour le cas de la 3-coloration. Plus généralement, on peut imaginer que toute preuve admet une représentation binaire, dont la machine devine les bits les uns après les autres. Bref, $L \in \mathsf{NP}_1$.
- $\mathsf{NP}_1 \subseteq \mathsf{NP}_2$: Soit L un langage dans NP_1. Par définition, il existe une machine non-déterministe M qui décide L en temps polynomial. Autrement dit, pour tout $w \in L$, il existe un chemin d'exécution de longueur polynomiale qui mène la machine M à accepter. Dans ce cas, la description de M et la suite de choix correspondant à ce chemin constituent une preuve rapide à vérifier de manière déterministe : il suffit de simuler M en effectuant seulement les choix indiqués, et vérifiant que la machine accepte bien le mot considéré. [1] On a donc $L \in \mathsf{NP}_2$.

□

Attention : lorsque vous effectuez ce raisonnement, gardez en tête qu'une machine non-déterministe ne peut effectuer qu'un nombre *constant* de choix à chaque étape. Par exemple, vous ne pouvez pas deviner la coloration complète d'un coup ! De même pour le cas général, la machine ne devine qu'un bit (ou un nombre constant de bits) de la preuve à la fois.

Étant donné que les deux versions de NP sont équivalentes, à partir de la prochaine séance et sauf exception, nous utiliserons toujours la seconde définition, qui est plus intuitive et ne repose pas sur le non-déterminisme.

1. Pour être complet, il me semble qu'il faudrait aussi pouvoir vérifier que M est correct, je n'ai rien trouvé là-dessus et ne suis pas certain, mais globalement, vous avez l'idée.

9.4 Discussions

Donnons une définition plus formelle de la seconde définition, en termes de machines de Turing *déterministes* appelées **vérifieurs** :

- NP : Un langage L est dans NP si et seulement si il existe une machine de Turing *déterministe* M appelée un **vérifieur** tel que pour tout $w \in L$, il existe un certificat (preuve) p tel que $M(w, p)$ accepte si $w \in L$, et aucun certificat ne peut amener M à accepter si $w \notin L$. De plus, la taille du certificat et le temps d'exécution de M sont tous deux polynomiaux en $|w|$.

Cette définition est un peu lourde, mais elle ne fait que reformuler ce que l'on sait déjà. Cela vaut le coup de la comprendre au moins une fois complètement.

Revenons maintenant à la classe P des langages que l'on peut décider en temps polynomial déterministe. Notez que $\mathsf{P} \subseteq \mathsf{NP}$ par un argument simple : une machine déterministe est un cas particulier de machine non-déterministe (pour la deuxième définition, on peut aussi observer qu'un algorithme qui résoud un problème dans P n'est rien d'autre qu'un vérifieur utilisant un certificat vide).

Est que ces deux classes sont identiques, comme pour l'espace ? C'est moins clair. Tous les problèmes que nous avons listés plus haut sont dans NP, mais ils ne sont pas forcément dans P. Par exemple, 3-COLORATION et SAT ne sont pas dans P *à notre connaissance*. Pour les deux autres, on connait des algorithmes en temps polynomial, ce qui d'ailleurs est surprenant pour PRIMALITÉ (car ce n'est pas le cas pour FACTORISATION, nous y reviendrons).

La relation entre P et NP est parmi les questions les plus fondamentales en informatique et en mathématiques. Pourquoi en mathématiques ? En simplifiant un peu, on peut définir le langage L_{MATH} de tous les énoncés mathématiques qui sont vrais, ou disons plutôt qui sont vrais et admettent des preuves vérifiables en temps raisonnable (sinon, l'énoncé lui-même a moins d'intérêt). On a donc $L_{\text{MATH}} \in \mathsf{NP}$. Que se passerait-il si quelqu'un montrait que $\mathsf{P} = \mathsf{NP}$? Intuitivement, cela impliquerait qu'il existe un algorithme rapide pour décider si tout énoncé mathématique est vrai. La majorité des spécialistes pensent que ce n'est pas le cas et que $\mathsf{P} \neq \mathsf{NP}$, mais ce n'est pas démontré. En fait, cette question est elle-même un énoncé mathématique que l'on espère être dans L_{MATH}, lui même dans NP... (vertige)

Plus philosophiquement, on peut se demander si tout ce que l'on est capable de comprendre peut être découvert en temps raisonnable. Ou encore si le fait qu'on sache apprécier une symphonie de Mozart implique qu'on aurait pu la composer nous-même, etc., etc.