Leçon 928 : Problèmes NP-complets : exemples et réductions.

Julie Parreaux

2018 - 2019

Références pour la leçon

[1] Carton, *Langages formels, calculabilité et complexité.*
[2] Cormen, *Algorithmique.*
[3] Floyd et Biegel, *Le langage des machines.*
[4] Kleinberg et Tardos, *Algorithms design.*
[5] Legendre et Schwarzentruber, *Compilation : Analyse lexicale et syntaxique du texte à sa structure en informatique.*
[6] Garey et Johnson, *Computers and Intractability : A Guide to the Theory of the NP-Completness.*
[7] Sipser, *Introduction to the Theory of Computation.*

Développements de la leçon

Le problème PSA est NP-complet

Approximation ou non du problème TSP

Plan de la leçon

Motivation

Défense

Thèse de Cobham–Edmonds

Ce qu'en dit le jury

L'objectif ne doit pas être de dresser un catalogue le plus exhaustif possible ; en revanche, pour chaque exemple, il est attendu que le candidat puisse au moins expliquer clairement le problème considéré, et indiquer de quel autre problème une réduction permet de prouver sa NP-complétude.

Les exemples de réduction polynomiale seront autant que possible choisis dans des domaines variés : graphes, arithmétique, logique, etc. Si les dessins sont les bienvenus lors du développement, le jury attend une définition claire et concise de la fonction associant, à toute instance du premier problème, une instance du second ainsi que la preuve rigoureuse que cette fonction permet la réduction choisie et que les candidats sachent préciser comment sont représentées les données.

Un exemple de problème NP-complet dans sa généralité qui devient P si on contraint davantage les hypothèses pourra être présenté, ou encore un algorithme P approximant un problème NP-complet.

1 Des problèmes NP-complets

1.1 Les classes P et NP

— *Définition* : Classe de problème P + exemple
— *Proposition* : Propriété de la classe P (stabilité)
— *Définition* : Classe de problème NP + exemple
— *Définition* : Vérifieur + caractérisation

1.2 La NP-complétude

— *Définition* : Réduction en temps polynomial + notation
— *Proposition* : Conséquences de la réduction
— *Définition* : Problème NP-complet
— *Remarque* : Caractérisation de P = NP
— *Théorème* : Cook
— *Exemple* : Vertex cover éclairage
— *Exemple* : Clique serveur web, communauté
— *Exemple* : independant set activité compatible

1.3 Technique de preuve de la NP-complétude

— Restriction d'un problème : 3SAT à SAT
— Remplacement locale : SAT à 3SAT
— Gadget : 3SAT à 3-Col
— Autre : Séparabilité d'un automate DEV
— Utilisation des classes supérieures : Regexp non universelle

2 la NP-complétude en pratique

Quand on tombe sur un problème NP-complet, on n'a pas tout perdu...

2.1 Restreindre des entrées

Idée : Perte d'expressivité du problème

— Réduction de la taille de l'entrée : 2SAT / 2Col

— Passage aux réels : problème du sac à dos réel

— Logique de Hoare + application à l'analyse syntaxique et prolog

2.2 Approximer les solutions

Idée : Perte de précision

— *Définition* : Problème d'optimisation

— *Remarque* : Lien avec les problèmes de décision

— *Définition* : Approximation + Schéma d'approximation

— Problème TSP DEV (glouton)

— Autre ?

2.3 Explorer l'ensemble des solutions

Idée : Perte de temps ;)

— Backtracking : DPLL (autre ?)

— Branch and Bound

— Programmation linéaire

Quelques notions importantes

Technique de preuve : la réduction

Pour montrer qu'un problème apparaît dans une certaine classe de complexité (et même sa dureté) ou qu'il est indécidable, nous utilisons une technique de preuve : la réduction. Pour appliquer le principe de réduction il nous faut connaître un premier problème possédant les propriétés que l'on souhaite montrer sur le deuxième.

Nous présentons ici le principe de la réduction dans sa généralité puis nous verrons comment le spécialiser pour en faire ce que nous souhaitons.

Définition. Une réduction d'un problème A à un problème B (Figure 1) est une fonction tr calculable telle que pour tout w instance de A, w est une instance positive de A si et seulement si $tr(w)$ est une instance positive de B. On note $A \leq B$.

On dit que A se réduit à B s'il existe une réduction de A à B (intuitivement, A est plus facile que B).

Remarque. En fonction des propriétés sur la fonction tr, on obtient différentes réductions qui vont nous permettre de spécialiser la réduction au résultat que nous souhaitons montrer.

Théorème (Principe de la réduction)**.** *Si A se réduit à B, alors si P est une propriété sur B alors P est une propriété sur A (dans notre cas, P peut être l'appartenance à une classe de complexité ou être le caractère indécidable d'un problème, ...).*

Démonstration. On raisonne par l'absurde et grâce à la fonction de traduction, on obtient une contradiction. □

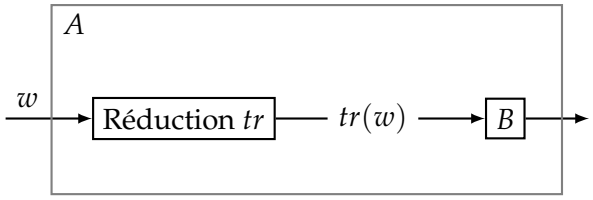

FIGURE 1 – Schéma du principe de réduction du problème A au problème B.

Nous allons donner quelques réductions de A à B et leurs propriétés. On note C une classe de complexité telle que $P \subseteq C$.

Réduction	Propriétés	Conséquences
Calculable	tr est calculable par une machine de Turing (variante de MT : équivalence)	A indécidable $\Rightarrow$ B indécidable B décidable $\Rightarrow$ A décidable
Temps polynomial	tr est calculable par une machine de Turing déterministe en temps polynomial	A est C-dur $\Rightarrow$ B est C-dur $(C \neq P)$ $B \in C \Rightarrow A \in C$
Espace logarithmique	tr est calculable par une machine de Turing déterministe en espace logarithmique (la MT a trois rubans)	A est D-dur $\Rightarrow$ B est D-dur $B \in D \Rightarrow A \in D$ $(D \neq P)$ avec $D \in \{L, NL, co-NL, P\}$

Remarque. La réduction en espace logarithmique est une réduction très spéciale car une machine qui travail en espace logarithmique a au moins deux rubans (il ne faut pas que l'entrée rentre dans la calcul). Mais pour la réduction, la sortie n'est pas non plus dans la borne de la mémoire utilisé (notons qu'elle est polynomiale (car une réduction en espace logarithmique s'exécute en temps polynomial)), il nous faut donc un troisième ruban. La machine de Turing effectuant la réduction a donc trois rubans : un ruban contenant l'entrée en lecture seule et en une seule passe ; un ruban de travail logarithmique et un ruban de sortie polynomial en écriture seule et en une seule passe.

Proposition. *Une réduction en espace logarithmique est une réduction en temps polynomial.*

Le théorème de Cook

Le théorème de Cook est un théorème historique car c'est le premier résultat de NP-complétude. Ce résultat n'applique donc pas une réduction polynomial à partir d'un problème NP-dur mais il explique quel est la réduction polynomiale à partir d'un problème NP quelconque.

Quelques rappels sur la NP-complétude [Papadimitriou ?]

Définition. La classe P est l'ensemble des problèmes de décision (vu comme un langage) décidé par une machine de Turing (ou un algorithme) déterministe en temps polynomial en la taille de l'entrée.

Remarque. Une définition alternative existe : elle fait apparaître la robustesse de cette classe. En effet, P est la réunion (sur $\mathbb{N}$) de tous les problèmes (langages) qui sont décidés par une machine de Turing déterministe en temps $O(n^k)$.

Définition. La classe NP est l'ensemble des problèmes de décision (vu comme un langage) décidé par une machine de Turing (ou un algorithme) non-déterministe en temps polynomial en la taille de l'entrée.

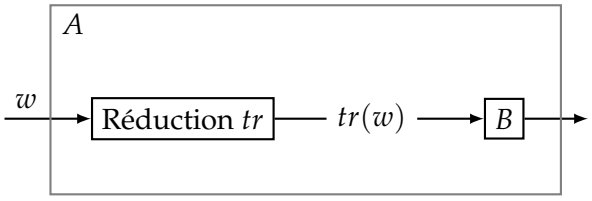

FIGURE 2 – Schéma du principe de réduction polynomiale du problème A au problème B.

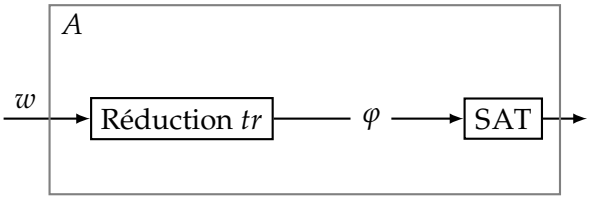

FIGURE 3 – Schéma du principe de réduction polynomiale dans le théorème de Cook d'un problème A dans NP au problème SAT.

Définition. Un vérifieur pour un problème A est une machine de Turing $\mathcal{M}$ déterministe telle que w est une instance positive de A si et seulement s'il existe c tel que $\mathcal{M}$ accepte (w, c). Un tel c est appelé certificat.

Proposition (Définition alternative de NP [7, p.294])**.** *$A \in NP$ si et seulement si il existe un vérifieur en temps polynomial de A.*

Idée de la démonstration. L'idée est de montrer comment convertir un vérifieur en une machine de Turing non-déterministe et réciproquement.

Algorithm 1 Algorithme convertissant un vérifieur en une machine de Turing non-déterministe.

```
1: procedure M(w)
2:     Choisir c de longueur f(|w|)
3:     if V(w, c) accepte then
4:         Accepter
5:     else
6:         Rejeter
7:     end if
8: end procedure
```

Algorithm 2 Algorithme convertissant un vérifieur en une machine de Turing non-déterministe.

```
1: procedure V(w, c)
2:     Simuler l'exécution M(w) en prenant les choix conseillés par c.
3:     if l'exécution accepte then
4:         Accepter
5:     else
6:         Rejeter
7:     end if
8: end procedure
```

□

Remarque. On a $P \subseteq NP$ et si $NP - C \cap P \neq \emptyset$ alors $P = NP$ où $NP - C$ est l'ensemble des problèmes NP-complets.

Définition. Une réduction polynomiale (Figure 2) de A vers B est une fonction tr telle que

— tr est calculable en temps polynomial ;

— w est une instance positive de A si et seulement si $tr(w)$ est une instance positive de B.

On note alors $A \preccurlyeq B$.

Théorème. *Si le problème A se réduit au problème B, alors :*

— *si $B \in P$ alors $A \in P$;*

— *si $A \in NP$ alors $B \in NP$.*

Définition. Un problème A est dit NP-dur si pour tout problème $B \in NP$, il existe une réduction polynomiale de B à A. Si, de plus, $A \in NP$, alors A est dit NP-complet

Le théorème en lui-même Nous allons maintenant énoncé et donner l'idée de la preuve du théorème de Cook (qui fut le premier à exhiber un problème NP-complet).

Définition. On définit le problème SAT pour les formules de la logique propositionnelle.

Problème : SAT
entrée une formule ϕ de la logique propositionnelle
sortie Oui si ϕ est satisfiable ; non sinon

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

Idée de la démonstration. Montrons que e problème SAT est NP-complet.

SAT $\in NP$ Choisir une valuation pour chacune des variables de la formule (choix non-déterministe) et vérifier si c'est une valuation. On accepte lorsqu'on a trouvé une valuation et sinon on rejette.

SAT est NP-dur Comme c'est la première démonstration qu'un problème est NP-complet, il nous faut utiliser la définition. On prend donc un problème NP B et on va le réduire au problème SAT (Figure 3). Par définition, il existe une machine de Turing non-déterministe $\mathcal{M}$ qui décide B en temps polynomial. On va alors coder l'exécution de cette machine dans des formules de la logique propositionnelle. Nous énonçons les propriétés que nous souhaitons coder sans en donner la traduction en formule de la logique propositionnelle.

1. La machine de Turing $\mathcal{M}$ et dans un état à tout instant t.
2. La machine de Turing $\mathcal{M}$ n'est jamais dans deux états à la fois.
3. Le curseur est positionné quelque part à tout instant t.
4. Le curseur n'est jamais à deux positions différentes.
5. À tout instant, toute case du ruban contient une lettre.
6. Une case du ruban ne contient pas plus d'une lettre.
7. À tout instant, on tire une transition pour aller vers l'instant $t + 1$.
8. On ne tire jamais plus d'une transition.
9. À l'instant 0, le ruban contient l'instance donnée à la machine.
10. À l'instant 0, la machine est dans l'état initial q_0 et son curseur est à la position 1.
11. La machine atteint son état d'acceptation.
12. On ne change pas le contenu du ruban, si le curseur n'y ait pas.
13. On ne tire qu'une transition de son état courant lisant la lettre sur le ruban.
14. On change s'état lorsqu'on applique une transition (le bon état).
15. On écrit la nouvelle valeur du ruban sur la case i.
16. Le curseur change de position lors d'une transition.

La formule que l'on cherche est la conjonction de toutes ses formules, ce qui prouve le théorème de Cook. □

Application La première application de ce théorème est une technique de preuve plus agréable pour montrer la NP-dureté. En effet, maintenant que nous avons un premier problème NP-complet, nous allons pourvoir réduire les autres à celui et par transitivité de la relation est réductible à, on prouvera la NP-dureté de ce problème. Depuis on en a trouvé des nombreux.

Références

[1] O. Carton. *Langages formels, calculabilité et complexité*. Vuibert, 2008.

[2] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Algorithmique, 3ème édition*. Dunod, 2010.

[3] R. Floyd and R. Biegel. *Le langage des machines*. International Thomson Publishing, 1994.

[4] J. Kleinberg and E. Tardos. *Algorithm Design*. Pearson international Edition, 2006.

[5] R. Legendre and F. Schwarzentruber. *Compilation : Analyse lexicale et syntaxique du texte à sa structure en informatique*. Reference Sciences. Ellipses, 2015.

[6] D.S. Johnson M.R. Garey. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[7] M. Sipser. *Introduction to the Theory of Computation*. Cengage learning, 1133187811.