Semaine 8 : Bases de la théorie de la complexité

Novembre 2022

1 Premières définitions (rappels)

Soit un problème de décision $\mathcal{P}$ à résoudre. L'entier n désigne la taille de codage d'une instance du problème.

Définition 1 *la classe* DTIME($f(n)$) *est l'ensemble des problèmes de décision qui sont résolvables par un algorithme en temps* $\mathcal{O}(f(n))$ *(par une machine de turing déterministe).*

Définition 2 *La classe* PTIME *est l'ensemble des problèmes de décision qui peuvent être résolus par un algorithme de complexité polynomiale (par une machine de turing déterministe).*
PTIME= $\bigcup_{k>0}$DTIME(n^k).

Définition 3 *La classe* EXPTIME *est l'ensemble des problèmes de décision qui peuvent être résolus par un algorithme de complexité poly-exponentiel (par une machine de turing déterministe).*
EXPTIME= $\bigcup_{k>0}$DTIME($exp(n^k)$).

2 Machine de Turing non déterministe et problèmes NP

Une machine de Turing non déterministe (NDTM en abrégé) est une machine de Turing déterminisite (DTM en abrégé) qui aurait la possibilité d'examiner en parallèle plusieurs choix possible. Par exemple, supposons que l'on souhaite résoudre le problème 3-SAT en utilisant une NDTM.

3-SAT
Input : U un ensemble de variables booléennes, C un ensemble de clauses de 3 termes chacunes.
Question : Peut on fixer les valeurs des variables de sorte que toutes les clauses soient vérifiées ?

Soient $U = \{x_1, \ldots, x_n\}$. On pourrait commencer par consider les deux possibilités de la première variable x_1. Puis, pour chacune de ces possibilités, les deux possibilités de la variable x_2.etc.
Une autre façon de voir les choses est d'imaginer que la machine possède un oracle qui lui indique succéssivement quelles sont les bonnes valeurs à considérer pour construire une solution au problème, si elle existe.

Théorème 1 *Tout problème* $\mathcal{P}$ *resolvable par une NDTM l'est par une DTM, et inversement.*

Par contre, la complexité obtenu n'est pas la même dans les deux cas.

Définition 4 *La classe* NP « *non deterministic polynomial* »*est l'ensemble des problèmes de décision qui peuvent être résolus par un algorithme de complexité polynomiale en utilisant une machine de turing non déterministe.*

Définition 5 *Un vérificateur pour un problème* $\mathcal{P}$ *est une procèdure boolénne* V *à deux arguments* $\mathcal{I}$ *et* x *tels que la réponse au problème* $\mathcal{P}$ *pour l'instance* $\mathcal{I}$ *est «* oui »*ssi il existe* x *tel que* $V(\mathcal{I}, x)$ *est vraie.*
L'argument x *est appelé un certificat pour* $\mathcal{P}$ *associé à* $\mathcal{I}$*.*

Théorème 2 *La classe* NP *est formé des problèmes de décision qui possèdent un vérificateur de complexité en temps polynomial.*

On observe que PTIME $\subseteq$ NP. PTIME $\neq$ NP est une hypothèse communément admise.

Théorème 3 ([1]) *Tout problème* $\mathcal{P} \in$ NP *est résolvable par une DTM en temps* $O(2^{p(n)})$ *(*NP $\subseteq$ EXPTIME*).*

3 Réduction polynomiale, problème NP-complets et NP-difficiles

Définition 6 *Soient $\mathcal{P}$ et $\mathcal{Q}$ deux problèmes de décision. Une tranformation (ou réduction) est une fonction f qui associe à toute instance $\mathcal{I}_\mathcal{P}$ de $\mathcal{P}$, une instance $f(\mathcal{I}_\mathcal{P}) = \mathcal{I}_\mathcal{Q}$ de $\mathcal{P}$ telle que la réponse de $\mathcal{P}$ à $\mathcal{I}_\mathcal{P}$ est « oui »ssi la réponse de $\mathcal{Q}$ à $f(\mathcal{I}_\mathcal{P})$ est « oui ».*

On note alors $\mathcal{P} \propto \mathcal{Q}$, ou aussi $\mathcal{P} \leq_P \mathcal{Q}$.

Théorème 4 ([1]) *Soient $\mathcal{P}$ et $\mathcal{Q}$ deux problèmes de décision tels que $\mathcal{P} \propto \mathcal{Q}$.*

1. *Si $\mathcal{Q} \in$ PTIME, alors $\mathcal{P} \in$ PTIME ;*
2. *Si $\mathcal{Q} \in NP$, alors $\mathcal{P} \in NP$;*

Que pensez-vous de la réciproque ?

Définition 7 *Le problème de décision $\mathcal{P}$ est* NP*-difficile si tout problème $\mathcal{Q} \in NP$ vérifie $\mathcal{Q} \propto \mathcal{P}$. Si de plus $\mathcal{P} \in NP$, $\mathcal{P}$ est* NP*-complet.*

On observe que PTIME $\subseteq$ NP.

Théorème 5 ([1]) *Soit $\mathcal{P}$ un problème* NP*-complet. $\mathcal{P} \in$ PTIME ssi PTIME $= NP$.*

PTIME $\neq$ NP est une hypothèse communément admise.

Théorème 6 (Thèorème de cook) *Le problème* SATISFIABILITY *est NP-complet.*

4 Méthodologie

Si l'on souhaite résoudre concrètement le problème $\mathcal{P}$, on s'interroge sur la complexité du problème.

1. Si on peut trouver un algorithme de complexité en $\mathcal{O}(Poly(n))$, où $Poly(n)$ est une fonction polynomiale de n, alors on peut résoudre le problème pour des instances de taille (relativement) importante de n si la puissance de n n'est pas trop importante.
   Par exemple, soit $G = (V \cup \{s\}, A, v)$ un graphe non orienté connexe de n sommets et a arêtes, avec des valuation positives sur les arêtes. Alors, calculer la valeur du plus court chemin de $i \in V$ à tous les autres sommets peut-être réalisé par l'algorithme de Dijkstra avec une complexité en temps de l'ordre de $\mathcal{O}(a + n \log n)$. Donc, ce problème est polynomial.
   Autre exemple, l'algorithme de Prim permet de calculer un arbre couvrant dont la somme des valuations est minimale en un temps $\mathcal{O}(n^2)$ si le graphe est représenté par une matrice sommet-sommet.
2. Si on peut démontrer que le problème est NP-complet ou NP-difficile, alors si $P \neq NP$, il n'y a pas d'algorithme polynomial pour résoudre $\mathcal{P}$. Donc, pour résoudre $\mathcal{P}$ dans tous les cas, on doit développer un algorithme de complexité non polynomiale (et donc inutilisable pour des instances de grande taille). Si le problème est dans NP, on sait qu'un tel algorithme est possible.
   Par exemple, pour résoudre concrètement le problème SAT à n variables et m clauses, il suffit de tester toutes les valeurs possibles des variables. La complexité d'un tel algorithme est $\mathcal{O}(2^n \times m)$.

Références

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