2.C.1 – Complexité

Emmanuel Beffara & Lionel Vaux

Université d'Aix-Marseille

Formation ISN
12 décembre 2012

http://isn.irem.univ-mrs.fr/

Additioner est-il plus facile que multiplier ?

Soient A et B deux nombres à n chiffres.
Combien d'étapes faut-il pour calculer $A + B$?

Additioner est-il plus facile que multiplier ?

Soient A et B deux nombres à n chiffres.
Combien d'étapes faut-il pour calculer $A + B$?

- en comptant sur ses doigts : $B \simeq 10^n$
  il faut beaucoup de doigts !

Additioner est-il plus facile que multiplier ?

Soient A et B deux nombres à n chiffres.
Combien d'étapes faut-il pour calculer $A + B$?

- en comptant sur ses doigts : $B \simeq 10^n$
  il faut beaucoup de doigts !
- en posant l'addition : n

Additioner est-il plus facile que multiplier ?

Soient A et B deux nombres à n chiffres.
Combien d'étapes faut-il pour calculer $A + B$?

- en comptant sur ses doigts : $B \simeq 10^n$
  il faut beaucoup de doigts !
- en posant l'addition : n

Combien d'étapes faut-il pour calculer $A \times B$?

Additioner est-il plus facile que multiplier ?

Soient A et B deux nombres à n chiffres.
Combien d'étapes faut-il pour calculer $A + B$?

- en comptant sur ses doigts : $B \simeq 10^n$
  il faut beaucoup de doigts !
- en posant l'addition : n

Combien d'étapes faut-il pour calculer $A \times B$?

- en comptant sur ses doigts : $A \times B$ ... peut-on faire mieux ?

Additioner est-il plus facile que multiplier ?

Soient A et B deux nombres à n chiffres.
Combien d'étapes faut-il pour calculer $A + B$?

- en comptant sur ses doigts : $B \simeq 10^n$
  il faut beaucoup de doigts !
- en posant l'addition : n

Combien d'étapes faut-il pour calculer $A \times B$?

- en comptant sur ses doigts : $A \times B$
- en posant la multiplication : n^2 ... peut-on faire mieux ?

Additioner est-il plus facile que multiplier ?

Soient A et B deux nombres à n chiffres.
Combien d'étapes faut-il pour calculer $A + B$?

- en comptant sur ses doigts : $B \simeq 10^n$
  il faut beaucoup de doigts !
- en posant l'addition : n

Combien d'étapes faut-il pour calculer $A \times B$?

- en comptant sur ses doigts : $A \times B$
- en posant la multiplication : n^2
- en utilisant des transformations de Fourier, on peut faire le calcul en $K \times n \times \log(n) \times \log(\log(n))$ étapes ... peut-on faire mieux ?

Additioner est-il plus facile que multiplier ?

Soient A et B deux nombres à n chiffres.
Combien d'étapes faut-il pour calculer $A + B$?

- en comptant sur ses doigts : $B \simeq 10^n$
  il faut beaucoup de doigts !
- en posant l'addition : n

Combien d'étapes faut-il pour calculer $A \times B$?

- en comptant sur ses doigts : $A \times B$
- en posant la multiplication : n^2
- en utilisant des transformations de Fourier, on peut faire le calcul en $K \times n \times \log(n) \times \log(\log(n))$ étapes

Comment savoir s'il y a mieux ?

Quelques rappels

Efficacité des algorithmes

- le cas des tris
- la manipulation de dictionnaires

Qu'est-ce qu'on compte ?

Les questions que l'on se pose

Intuitivement, certaines choses semblent vraies :

- Il est plus facile d'additionner que de multiplier.
- Il est plus facile de trouver un élément dans une liste si la liste est triée.
- Pour savoir s'il est possible de colorier une carte avec seulement trois couleurs, on ne peut pas faire beaucoup mieux en général que d'essayer toutes les façons possible.

Le but de la théorie de la complexité est d'apporter des justifications précises à ce genre d'affirmation (ou de les réfuter). On cherche donc à

quantifier les ressources (temps, mémoire) nécessaires à la résolution d'un problème

comparer les degrés de difficulté de différents problèmes

La notion générale de problème

Définition

Un *problème* est la spécification d'une fonction pour laquelle il peut exister des algorithmes de calcul. On parle de *problème de décision* si la sortie est une valeur de vérité.

Par exemple :

- **entrée :** un graphe G et deux sommets A et B, **sortie :** la longueur du plus court chemin de A à B
- **entrée :** un dictionnaire D un mot m, **sortie :** la définition du m dans D
- **entrée :** un message codé, **sortie :** la clé de codage utilisée

Un problème de décision est en général formulé comme une question :

- **entrée :** un nombre naturel n, **question :** n est-il premier ?
- **entrée :** le code d'un programme P, **question :** l'exécution de P termine-t-elle toujours ?

Comment définir la complexité d'un problème

Première tentative

On considère donc un problème P.

Définition

La complexité (en temps) d'un algorithme A est la fonction c_A qui, pour chaque instance X du problème, donne le nombre $c_A(X)$ d'étapes de calcul utilisées par A pour répondre.

Définition

La complexité intrisèque du problème P est la fonction c_P qui, pour chaque instance X du problème, donne la plus petite valeur de $c_A(X)$ pour tous les algorithmes qui résolvent P.

Comment définir la complexité d'un problème

Première tentative

On considère donc un problème P.

Définition

La complexité (en temps) d'un algorithme A est la fonction c_A qui, pour chaque instance X du problème, donne le nombre $c_A(X)$ d'étapes de calcul utilisées par A pour répondre.

Définition

La complexité intrisèque du problème P est la fonction c_P qui, pour chaque instance X du problème, donne la plus petite valeur de $c_A(X)$ pour tous les algorithmes qui résolvent P.

Ça ne peut pas marcher !

Comment définir la complexité d'un problème

Pourquoi la première tentative échoue

Même pour un problème P très difficile, on peut toujours écrire un programme extrêmement efficace pour une entrée Z donnée :

```
if X == Z :
    return valeur_pour_Z
else :
    ... algorithme compliqué ...
```

Il suffit de pré-calculer une valeur, mais on ne peut pas pré-calculer *toutes* les valeurs possibles !

Comment définir la complexité d'un problème

Deux remarques s'imposent donc :

1. Il faut définir précisément ce qu'est le nombre d'étapes de calcul.

   $\rightsquigarrow$ *on doit utiliser un* ***modèle de calcul***

2. La complexité n'a des sens que quand on considère le problème dans son ensemble, pas pour des instances particulières.

   $\rightsquigarrow$ *seul le comportement* ***asymptotique*** *a un sens dans l'absolu*

Rappel : les machines de Turing

On se donne un alphabet A plus une lettre spéciale $\varnothing \notin A$ (« case vide »).
Une machine de Turing :

- travaille sur un nombre fixé de rubans infinis dont chaque case contient une lettre de $A \cup \{\varnothing\}$,
- a une tête de lecture/écriture positionnée sur chaque ruban ;
- est contrôlée par un programme écrit avec les instructions suivantes :
  - si la tête numéro i lit la lettre x alors aller à l'instruction a
  - écrire la lettre x sur le ruban i
  - déplacer la tête numéro i d'une case vers la droite
  - déplacer la tête numéro i d'une case vers la gauche
  - terminer

Rappel : calculer avec une machine de Turing

On note A^* l'ensemble des mots *(= suites finies)* sur A.
On peut toujours écrire un mot de A^* sur un ruban, quitte à mettre $\varnothing$ partout ailleurs.

Définition

Une machine M calcule une fonction $f : A^* \to A^*$ si, quand on la fait fonctionner à partir d'un état où le premier ruban contient m et les autres sont vides, elle finit par s'arrêter dans un état où le dernier ruban contient le mot $f(m)$.

Rappel : calculer avec une machine de Turing

On note A^* l'ensemble des mots *(= suites finies)* sur A.
On peut toujours écrire un mot de A^* sur un ruban, quitte à mettre $\varnothing$ partout ailleurs.

Définition

Une machine M calcule une fonction $f : A^* \to A^*$ si, quand on la fait fonctionner à partir d'un état où le premier ruban contient m et les autres sont vides, elle finit par s'arrêter dans un état où le dernier ruban contient le mot $f(m)$.

Un petit exemple pour se remettre les idées en place ?

Complexité d'un programme

Définition

Soit M une machine de Turing. La *fonction de complexité* de M est la fonction $c_M : \mathbb{N} \to \mathbb{N}$ qui, pour chaque n, donne le nombre maximum d'étapes de calcul que peut prendre M avant de s'arrêter si on l'exécute sur un mot de longueur n.

Exercice

Évaluer la fonction de complexité pour des machines calculant les fonctions suivantes :

- $a_1 a_2 ... a_{k-1} a_k \mapsto a_k a_{k-1} ... a_2 a_1$ (retourner un mot)
- $\underline{n}_2 \mapsto \underline{n+1}_2$ (incrémenter un entier écrit en binaire)
- $\underline{a}_2 \sharp \underline{b}_2 \mapsto \underline{a+b}_2$ (additioner deux nombres écrits en binaire)

Influence de la représentation des données

Un arrière-goût de « A.1 : Codage numérique »

Les problèmes que l'on se pose font intervenir des notions abstraites (nombres, structures de données) mais les machines de Turing travaillent sur des suites de symboles (donc des *représentations*).

Exercice

Évaluer la complexité en temps de l'addition en fonction de la représentation utilisée pour les entiers :

- unaire
- base 2
- base 10
- chiffres romains

Exercice

Évaluer le coût de la conversion d'une représentation vers l'autre.

Ordres de grandeur de complexité

Le coût précis d'un calcul est donc dépendant de la façon dont les données sont représentées, mais entre deux représentations *raisonnables*, la différence n'est pas significative :

> *Quelle que soit la base employée, le nombre d'étapes de calcul nécessaire pour additionner deux entiers de taille n est majoré par un nombre de la forme $\alpha n + \beta$, pour deux constantes α et β.*

Changer de modèle de calcul *(autres formes de machines de Turing, machines RAM, processeurs réels)* est de la même nature : on ne fait que changer la valeur des constantes.

Définition

Soit $f : \mathbb{N} \to \mathbb{N}$ une fonction croissante, on dit qu'une machine fonctionne en $O(f(n))$ s'il existe une constante K telle que $c_M(n) \leqslant Kf(n)$ pour n assez grand.

Ordres de grandeur de complexité

Le coût précis d'un calcul est donc dépendant de la façon dont les données sont représentées, mais entre deux représentations *raisonnables*, la différence n'est pas significative :

> *Quelle que soit la base employée, le nombre d'étapes de calcul nécessaire pour additionner deux entiers de taille n est majoré par un nombre de la forme $\alpha n + \beta$, pour deux constantes α et β.*

Changer de modèle de calcul *(autres formes de machines de Turing, machines RAM, processeurs réels)* est de la même nature : on ne fait que changer la valeur des constantes.

Définition

Soit $f : \mathbb{N} \to \mathbb{N}$ une fonction croissante, on dit qu'une machine fonctionne en $O(f(n))$ s'il existe une constante K telle que $c_M(n) \leqslant Kf(n)$ pour n assez grand.

N.B. : Là encore, il faut s'en tenir aux modèles de calcul raisonnables, *sinon c'est différent : les machines à compteurs comptent avec les doigts !*

Classes de complexité

L'utilisation d'ordres de grandeur permet donc de définir une mesure de complexité *à une constante près* pour un problème indépendamment des détails de la représentation et du modèle de calcul.

Définition

Un problème est de classe Time($f(n)$) s'il existe une machine de Turing qui résoud ce problème en temps $O(f(n))$ pour une représentation particulière des données.

Un peu de vocabulaire :

- Time(n) : *temps linéaire* (addition, longueur d'un mot, concaténation...)
- Time(n^2) : *temps quadratique* (multiplication usuelle, recherche d'un sous-mot)

Classes de complexité classiques

Pour des problèmes qui font intervenir des données plus compliquées que des nombres, le choix de la représentation peut être plus coûteux que $O(n)$ ou $O(n^2)$, dans ce cas $\mathrm{Time}(n)$ n'a pas un sens aussi précis.
Pour cette raison on définit des classes plus larges et très significatives :

Définition

La classe du temps *polynomial* est $\mathbf{P} := \bigcup_{k \in \mathbb{N}} \mathrm{Time}(n^k)$.

Définition

La classe du temps *exponentiel* est $\mathbf{EXP} := \bigcup_{k \in \mathbb{N}} \mathrm{Time}(2^{n^k})$.

Le temps polynomial

On considère généralement que la classe **P** est celle des problèmes pour lesquels il existe un programme *raisonnablement efficace*, et qu'à l'inverse un problème qui n'est *pas* de classe **P** n'est pas accessible au calcul en pratique.

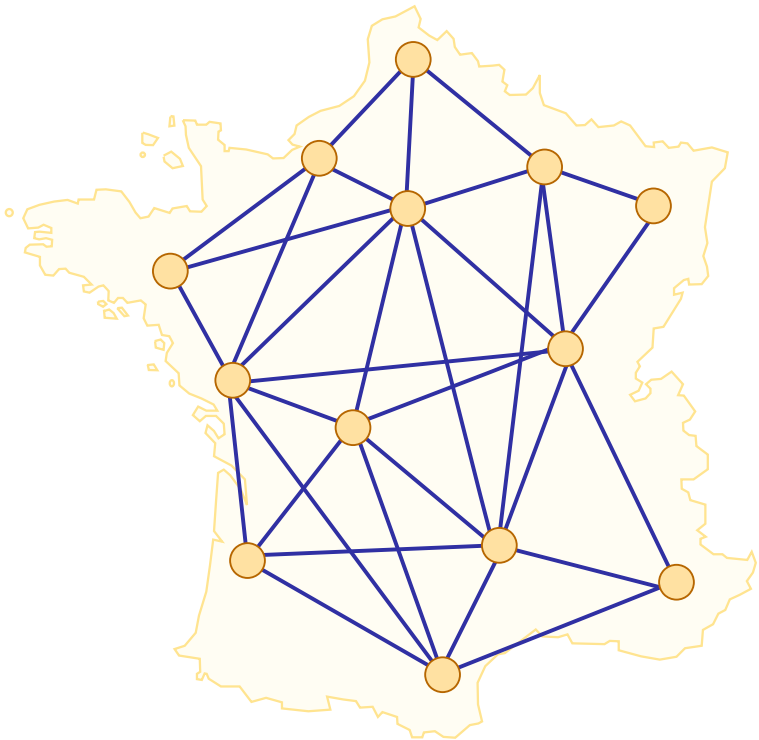
Quelques précisions s'imposent :

- Effectivement, presque tous les algorithmes utilisés en pratique ont une borne polynomiale.
- On peut se demander si un programme qui fonctionne en temps $O(n^{100})$ est effectivement utilisable, de plus la valeur de la constante multiplicative n'est pas anodine. *Mais pour les problèmes réels, on parvient souvent à réduire l'exposant et la constante pour obtenir des chiffres raisonnables.*
- Par définition, en mesurant la complexité d'un algorithme, on considère les pires cas possibles à chaque taille, mais dans la pratique on ne tombe pas forcément sur les pires cas.

Le problème du voyageur de commerce

Entrée : un graphe, une mesure sur les arêtes, un nombre L

Question : existe-t-il un chemin de longueur au plus L passant par chaque points du graphe exactement une fois et revenant à son point de départ ?



- On peut résoudre ce problème par *recherche exhaustive*, ou par des méthodes un peu plus raffinées, mais toutes les méthodes connues ont un coût **exponentiel** en la taille du problème.
- En revanche il est facile de vérifier une solution.

La satisfaisabilité

Entrée : une expression booléenne, dépendant uniquement de variables booléennes
Question : existe-t-il un choix de valeurs pour les variables booléennes qui rende vraie l'expression ?

Ici encore :

- On peut résoudre ce problème par *recherche exhaustive*, avec un coût **exponentiel** en la taille du problème, et on ne connaît pas de méthode *générale* qui ne soit pas exponentielle.
- Il est facile de vérifier une solution.

Deux citations

Si [la satisfaisabilité avait un algorithme quadratique] alors les conséquences seraient de la plus grande importance. C'est-à-dire que cela indiquerait clairement que, malgré l'impossibilité du problème de la décision (de Hilbert), l'effort mental du mathématicien dans le cas des questions attendant une réponse oui ou non pourrait être complètement remplacé par des machines [...] ceci me semble, néanmoins, dans le domaine du possible.

— Kurt Gödel dans une lettre à John von Neumann, 1956

Je conjecture qu'il n'y a pas de bon algorithme pour le problème du voyageur de commerce. Mes raisons sont les mêmes que pour n'importe quelle conjecture mathématique : (1) c'est une possibilité mathématique légitime, et (2) je ne sais pas.

— Jack Edmonds, 1966

La classe **NP**

Définition

Un problème de décision P est de classe **NP** s'il existe un problème de décision P' de complexité polynomiale et un polynôme p tels que pour toute entrée x de taille n il existe une donnée y de taille $p(n)$ telle que x est accepté par P si et seulement si (x, y) est accepté par P'.

y est appelé *certificat* pour x

Exemple (si P est le problème du voyageur de commerce)

- x désigne la donnée d'un un graphe G et d'un nombre L,
  y est la description d'un chemin dans G
- P' est le problème consistant à vérifier si y est un chemin de G de longueur au plus L passant par tous les sommets
- pour une entrée (G, L) on prend pour y le chemin recherché s'il existe, n'importe quel chemin sinon

N comme Non déterministe

Pour étudier **NP**, on a souvent recours aux machines de Turing non déterministes : on ajoute une instruction

- aller en a ou en b

et à l'exécution il y a deux choix possibles.

On définit que la machine répond positivement s'il est possible de faire des choix à chaque passage sur une telle instruction de sorte qu'en fin de compte la machine réponde par l'affirmative.

Cette instruction revient à deviner un bit du certificat.

Théorème

*Un problème est de classe **NP** si et seulement s'il est possible de le résoudre au moyen d'une machine de Turing non déterministe en temps polynomial.*

Histoire du test de primalité

Entrée : un nombre entier N (en binaire), **Question :** N est-il composé ?

- Méthodes naïves :
  - tester tous les entiers jusqu'à N ; $\rightsquigarrow$ *2^n étapes : c'est exponentiel*
  - un peu mieux : tester tous les entiers jusqu'à $\sqrt{N}$ pour savoir s'ils divisent N. $\rightsquigarrow$ *$2^{n/2}$ étapes, c'est toujours exponentiel*
- Méthodes non-déterministes :
  - Le certificat est un diviseur, l'algorithme fait la division.
    *Le problème de la non-primalité est **NP**, celui de la primalité est donc co-**NP***
  - En 1975, Pratt a mis au point une notion de certificat de primalité qui prouve que le problème est aussi dans **NP**.
- Méthodes déterministes plus efficaces :
  - Le *test cyclotomique* donne une complexité $O(n^{c \log(\log(n))})$.
  - En 2002, Agrawal, Saxena et Kayal ont publié un algorithme en $O(n^{12})$.
- Il y a aussi des méthodes probabilistes très efficaces.

Résoudre ou vérifier

Dans le cas du test de non-primalité, on a fini par montrer que le problème, naturellement de classe **NP**, est en fait résoluble en temps polynomial.
Est-ce le cas pour les autres problèmes de la même espèce ?

Problème à 1 000 000 $

Est-ce que $\mathbf{P} = \mathbf{NP}$?

Résoudre ou vérifier

Dans le cas du test de non-primalité, on a fini par montrer que le problème, naturellement de classe **NP**, est en fait résoluble en temps polynomial.
Est-ce le cas pour les autres problèmes de la même espèce ?

Problème à 1 000 000 $

Est-ce que $\mathbf{P} = \mathbf{NP}$?

On n'en a pas la moindre idée !

Résoudre ou vérifier

Dans le cas du test de non-primalité, on a fini par montrer que le problème, naturellement de classe **NP**, est en fait résoluble en temps polynomial.
Est-ce le cas pour les autres problèmes de la même espèce ?

Problème à 1 000 000 $

Est-ce que $\mathbf{P} = \mathbf{NP}$?

On n'en a pas la moindre idée !

L'opinion majoritaire est que $\mathbf{P} \neq \mathbf{NP}$, mais personne n'a réussi à le démontrer (il y a beaucoup de démonstrations fausses).

La **NP**-complétude

Définition

Un problème de décision est **NP**-complet s'il est de classe **NP** et si l'existence d'un algorithme polynomial pour le résoudre entraîne l'existence d'un algorithme polynomial pour chaque problème **NP**, autrement dit **P** = **NP**.

Théorème (Cook)

*Le problème de la satisfaisabilité est **NP**-complet.*

La **NP**-complétude

Définition

Un problème de décision est **NP**-complet s'il est de classe **NP** et si l'existence d'un algorithme polynomial pour le résoudre entraîne l'existence d'un algorithme polynomial pour chaque problème **NP**, autrement dit **P** = **NP**.

Théorème (Cook)

*Le problème de la satisfaisabilité est **NP**-complet.*

Idée de la démonstration : Considérer un problème **NP** quelconque et une machine non déterministe M qui le résoud en temps $p(n)$. Représenter l'assertion « sur l'entrée x, il existe une exécution de M qui répond oui » par un problème de satisfaisabilité. *C'est un peu technique à détailler.*

Réductions entre problèmes

Le théorème de Cook établit la **NP**-complétude d'un problème, on peut en déduire la **NP**-complétude d'autres problèmes sans repartir à zéro.

Définition

Une *réduction polynomiale* d'un problème A en un problème B est un algorithme qui, en temps polynomial, transforme une instance a du problème A en une instance b du problème B de sorte que a satisfait A si et seulement si b satisfait B.

Il s'agit de dire que le problème B est au moins aussi difficile que le problème A.

Théorème

*S'il existe une réduction polynomiale de A vers B et si A est **NP**-complet et B est **NP**, alors B est **NP**-complet.*

Exemple de réduction

Définition (Problème du chemin hamiltonien)

Entrée : un graphe G, **Question :** existe-t-il un chemin passant par chaque sommets de G exactement une fois ?

Réduction vers le problème du voyageur de commerce :

- on donne la longueur 1 aux arêtes de G,
- on ajoute des arêtes de longueur 2 entre les sommets qui ne sont pas connectés,
- on pose la question du voyageur de commerce avec $L = n + 1$ où n est le nombre d'arêtes.

Ainsi il suffit de prouver que le problème du chemin hamiltonien est **NP**-complet pour prouver que celui du voyageur de commerce l'est.
On peut le faire en y réduisant le problème de la satisfaisabilité !

Divers problèmes **NP**-complets

- **Clique** : étant donnés un graphe G et un nombre N, existe-t-il un sous-graphe complet (ou clique) de taille N dans G ?

Divers problèmes **NP**-complets

- **Clique** : étant donnés un graphe G et un nombre N, existe-t-il un sous-graphe complet (ou clique) de taille N dans G ?
- **Somme de sous-ensembles** : étant donnée une liste d'entiers L et un nombre N, peut-on choisir une partie de L dont la somme est N ?

Divers problèmes **NP**-complets

- **Clique** : étant donnés un graphe G et un nombre N, existe-t-il un sous-graphe complet (ou clique) de taille N dans G ?
- **Somme de sous-ensembles** : étant donnée une liste d'entiers L et un nombre N, peut-on choisir une partie de L dont la somme est N ?
- **Sac à dos** : étant donné un ensemble d'objets ayant chacun un poids et une valeur, peut-on choisir une partie des objets pour que le poids total soit inférieur à A et la valeur totale supérieure à B ?

Divers problèmes **NP**-complets

- **Clique** : étant donnés un graphe G et un nombre N, existe-t-il un sous-graphe complet (ou clique) de taille N dans G ?
- **Somme de sous-ensembles** : étant donnée une liste d'entiers L et un nombre N, peut-on choisir une partie de L dont la somme est N ?
- **Sac à dos** : étant donné un ensemble d'objets ayant chacun un poids et une valeur, peut-on choisir une partie des objets pour que le poids total soit inférieur à A et la valeur totale supérieure à B ?
- **Sudoku** *généralisé* : étant donnée une grille de Sudoku partiellement remplie, peut-on trouver une solution ?

Divers problèmes **NP**-complets

- **Clique** : étant donnés un graphe G et un nombre N, existe-t-il un sous-graphe complet (ou clique) de taille N dans G ?
- **Somme de sous-ensembles** : étant donnée une liste d'entiers L et un nombre N, peut-on choisir une partie de L dont la somme est N ?
- **Sac à dos** : étant donné un ensemble d'objets ayant chacun un poids et une valeur, peut-on choisir une partie des objets pour que le poids total soit inférieur à A et la valeur totale supérieure à B ?
- **Sudoku** *généralisé* : étant donnée une grille de Sudoku partiellement remplie, peut-on trouver une solution ?
- **Lemmings** : est-il possible de résoudre un tableau donné du jeu Lemmings ?