Recherche opérationnelle

DUT Info 2e année, parcours A

Complexité algorithmique de la PL et PLNE

Florent Foucaud

IUT CLERMONT AUVERGNE
Aurillac - Clermont-Ferrand - Le Puy-en-Velay
Montluçon - Moulins - Vichy

Complexité d'un problème algorithmique

Problème algorithmique : une entrée, une sortie (≈ programme informatique)

- Trier une liste de n entiers
- Trouver un plus court chemin de A à B dans un graphe à n sommets
- Résoudre un programme linéaire à n variables et m contraintes
- Couvrir un réseau à n sommets avec k antennes

Complexité d’un problème algorithmique

Problème algorithmique : une entrée, une sortie (≈ programme informatique)

- Trier une liste de n entiers
- Trouver un plus court chemin de A à B dans un graphe à n sommets
- Résoudre un programme linéaire à n variables et m contraintes
- Couvrir un réseau à n sommets avec k antennes

Pour un problème algorithmique P, quel est le plus petit nombre d’étapes de calcul nécessaire et suffisant pour résoudre P ?

Complexité d'un problème algorithmique

Problème algorithmique : une entrée, une sortie (≈ programme informatique)

- Trier une liste de n entiers
- Trouver un plus court chemin de A à B dans un graphe à n sommets
- Résoudre un programme linéaire à n variables et m contraintes
- Couvrir un réseau à n sommets avec k antennes

Pour un problème algorithmique P, quel est le plus petit nombre d'étapes de calcul nécessaire et suffisant pour résoudre P ?

C'est la complexité algorithmique du problème P.

Complexité d'un problème algorithmique

Problème algorithmique : une entrée, une sortie (≈ programme informatique)

- Trier une liste de n entiers
- Trouver un plus court chemin de A à B dans un graphe à n sommets
- Résoudre un programme linéaire à n variables et m contraintes
- Couvrir un réseau à n sommets avec k antennes

Pour un problème algorithmique P, quel est le plus petit nombre d'étapes de calcul nécessaire et suffisant pour résoudre P ?

C'est la complexité algorithmique du problème P.

On mesure cela par une fonction $f(n)$ de la taille n de l'entrée

(n : nombre de bits pour coder l'entrée)

Explosion combinatoire

Complexité algorithmique pour un problème donné :

$f(n)$ opérations pour une entrée de taille n

Meilleurs problèmes : complexité linéaire $f(n) \rightarrow 10n, 2n, 1000n, n$ ...

Problèmes "raisonnables" : complexité polynomiale $f(n) \rightarrow 4n^2, 10n^3, n^{1000}$ ...

Problèmes difficiles : complexité exponentielle $f(n) \rightarrow 2^n, n!, n^n$ ...

→ *Cela correspond à tester toutes les solutions possibles*

Explosion combinatoire

Complexité algorithmique pour un problème donné :

$f(n)$ opérations pour une entrée de taille n

Meilleurs problèmes : complexité linéaire $f(n) \to 10n, 2n, 1000n, n$ ...

Problèmes “raisonnables” : complexité polynomiale $f(n) \to 4n^2, 10n^3, n^{1000}$ ...

Problèmes difficiles : complexité exponentielle $f(n) \to 2^n, n!, n^n$ ...

→ *Cela correspond à tester toutes les solutions possibles*

$f(n)$	$n = 10$	$n = 50$	$n = 100$	$n = 200$	$n = 300$
n	10	50	100	200	300
$100n$	1000	5000	10000	20000	30000
n^2	100	2500	10000	40000	90000
2^n	1024	(16 chiffres)	(31 chiffres)	(60 chiffres)	(91 chiffres)
$n!$	3628800	(64 chiffres)	(157 chiffres)	(374 chiffres)	(614 chiffres)

Explosion combinatoire

Complexité algorithmique pour un problème donné :

$f(n)$ opérations pour une entrée de taille n

Meilleurs problèmes : complexité linéaire $f(n) \rightarrow 10n, 2n, 1000n, n$ ...

Problèmes “raisonnables” : complexité polynomiale $f(n) \rightarrow 4n^2, 10n^3, n^{1000}$ ...

Problèmes difficiles : complexité exponentielle $f(n) \rightarrow 2^n, n!, n^n$ ...

→ *Cela correspond à tester toutes les solutions possibles*

$f(n)$	$n=10$	$n=50$	$n=100$	$n=200$	$n=300$
n	10	50	100	200	300
$100n$	1000	5000	10000	20000	30000
n^2	100	2500	10000	40000	90000
2^n	1024	(16 chiffres)	(31 chiffres)	(60 chiffres)	(91 chiffres)
$n!$	3628800	(64 chiffres)	(157 chiffres)	(374 chiffres)	(614 chiffres)

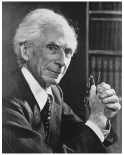
Question

Quels problèmes sont “raisonnables” ? Lesquels sont difficiles ?

Paradoxe du barbier

Dans un village, le barbier rase exactement tous les hommes qui ne se rasent pas eux-mêmes.

Question : Qui rase le barbier ?



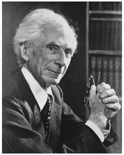
Bertrand Russell (1872-1970)

Paradoxe du barbier

Dans un village, le barbier rase exactement tous les hommes qui ne se rasent pas eux-mêmes.

Question : Qui rase le barbier ?

PARADOXE !



Bertrand Russell (1872-1970)

S’arrêter ou boucler ? Telle est la question

Problème de l’arrêt

Étant donné n’importe quel code de programme informatique, peut-on décider en temps fini :

1. s’il va s’arrêter un jour *ou bien* 2. s’il va tourner à l’infini ?

S’arrêter ou boucler ? Telle est la question

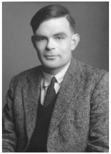
Problème de l’arrêt

Étant donné n’importe quel code de programme informatique, peut-on décider en temps fini :

1. s’il va s’arrêter un jour *ou bien* 2. s’il va tourner à l’infini ?

Théorème (Alan Turing, 1936)

Il n’existe aucun algorithme pour résoudre le problème de l’arrêt.



Alan Turing (1912-1954)

S’arrêter ou boucler ? Telle est la question

Problème de l’arrêt

Étant donné n’importe quel code de programme informatique, peut-on décider en temps fini :

1. s’il va s’arrêter un jour *ou bien* 2. s’il va tourner à l’infini ?

Théorème (Alan Turing, 1936)

Il n’existe aucun algorithme pour résoudre le problème de l’arrêt.

Démonstration : Supposons par l’absurde qu’il existe un programme en temps fini

```
arret(code, parametre)
```

qui renvoie
- VRAI si le code donné avec le paramètre s’arrêtera un jour, et
- FAUX si au contraire le code tourne à l’infini.

S'arrêter ou boucler ? Telle est la question

Problème de l'arrêt

Étant donné n'importe quel code de programme informatique, peut-on décider en temps fini :

1. s'il va s'arrêter un jour *ou bien* 2. s'il va tourner à l'infini ?

Théorème (Alan Turing, 1936)

Il n'existe aucun algorithme pour résoudre le problème de l'arrêt.

Démonstration : Supposons par l'absurde qu'il existe un programme en temps fini

`arret(code, parametre)`

qui renvoie
- VRAI si le code donné avec le paramètre s'arrêtera un jour, et
- FAUX si au contraire le code tourne à l'infini.

Soit le programme suivant :

```
def diag(x):
  • si arret(x,x) est VRAI alors:
      ▶ boucle infinie
  • sinon:
      ▶ retourner VRAI
```

S’arrêter ou boucler ? Telle est la question

Problème de l’arrêt

Étant donné n’importe quel code de programme informatique, peut-on décider en temps fini :

1. s’il va s’arrêter un jour *ou bien* 2. s’il va tourner à l’infini ?

Théorème (Alan Turing, 1936)

Il n’existe aucun algorithme pour résoudre le problème de l’arrêt.

Démonstration : Supposons par l’absurde qu’il existe un programme en temps fini

```
arret(code, parametre)
```

qui renvoie
- VRAI si le code donné avec le paramètre s’arrêtera un jour, et
- FAUX si au contraire le code tourne à l’infini.

Soit le programme suivant :

```
def diag(x):
  • si arret(x,x) est VRAI alors:
      ▶ boucle infinie
  • sinon:
      ▶ retourner VRAI
```

Que renvoie l’appel diag(diag) ?

S’arrêter ou boucler ? Telle est la question

Problème de l’arrêt

Étant donné n’importe quel code de programme informatique, peut-on décider en temps fini :

1. s’il va s’arrêter un jour *ou bien* 2. s’il va tourner à l’infini ?

Théorème (Alan Turing, 1936)

Il n’existe aucun algorithme pour résoudre le problème de l’arrêt.

Démonstration : Supposons par l’absurde qu’il existe un programme en temps fini

```
arret(code, parametre)
```

qui renvoie
- VRAI si le code donné avec le paramètre s’arrêtera un jour, et
- FAUX si au contraire le code tourne à l’infini.

Soit le programme suivant :

```
def diag(x):
  • si arret(x,x) est VRAI alors:
      ▶ boucle infinie
  • sinon:
      ▶ retourner VRAI
```

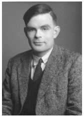
Que renvoie l’appel `diag(diag)` ?

PARADOXE !

Problèmes indécidables

Problèmes indécidables :

- Problème de l'arrêt (Alan Turing, 1936)
- Problème de correspondance de mots : 2 paquets de mots $a_1, \ldots, a_n$ et $b_1, \ldots, b_n$
  → Peut-on les arranger pour créer deux mots identiques ? (Emil Post, 1946)
- Trouver des solutions entières d'équations diophantiennes
  de type $2x^2 + 3y^3 - 2z = 0$ (Youri Matyasevitch, 1970 - 10e problème de Hilbert, 1900)



Alan Turing (1912-1954)

Emil L. Post (1897-1954)

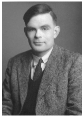
Youri Matyasevitch (1947-)

David Hilbert (1862-1943)

Problèmes indécidables

Problèmes indécidables :

- Problème de l'arrêt (Alan Turing, 1936)
- Problème de correspondance de mots : 2 paquets de mots $a_1, \ldots, a_n$ et $b_1, \ldots, b_n$
  $\rightarrow$ Peut-on les arranger pour créer deux mots identiques ? (Emil Post, 1946)
- Trouver des solutions entières d'équations diophantiennes
  de type $2x^2 + 3y^3 - 2z = 0$ (Youri Matyasevitch, 1970 - 10e problème de Hilbert, 1900)



Alan Turing (1912-1954)

Emil L. Post (1897-1954)

Youri Matyasevitch (1947-)

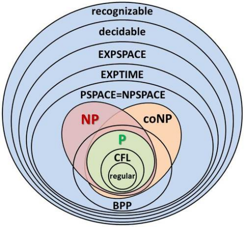
David Hilbert (1862-1943)

Liens avec la logique mathématique :

théorème d'incomplétude de Gödel (1931)

Kurt Gödel (1906-1978)

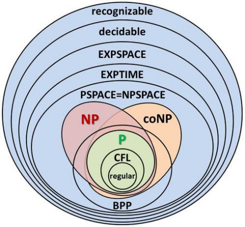
Quelques classes de complexité algorithmique



Classe P (pour “polynomiaux”) : problèmes “raisonnables”

Au-dessus : problèmes (probablement) algorithmiquement difficiles

Quelques classes de complexité algorithmique

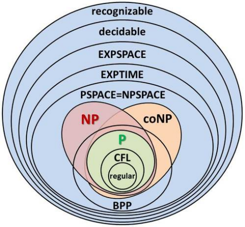


Classe P (pour “polynomiaux”) : problèmes “raisonnables”

Au-dessus : problèmes (probablement) algorithmiquement difficiles

Jack Edmonds (1934-)

Quelques classes de complexité algorithmique



Classe P (pour “polynomiaux”) : problèmes “raisonnables”

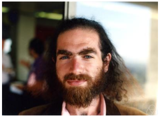
Au-dessus : problèmes (probablement) algorithmiquement difficiles

Question (P versus NP - une question à 1 million de $)

Est-ce que $P = NP$? *(On pense que non.)*

CLAY MATHEMATICS INSTITUTE

7 problèmes du millénaire à 1 million de $



Grigori Perelman (1966-)

PL vs PLNE

L'algorithme du simplexe n'est (en général) pas polynomial... mais il l'est souvent !

PL vs PLNE

L'algorithme du simplexe n'est (en général) pas polynomial… mais il l'est souvent !

Théorème (Leonid Khachiyan, 1979)

Le problème de trouver une solution à un PL est polynomial ("raisonnable").

Leonid Khachiyan (1952-2005)

→ Méthode de l'ellipsoïde ou des points intérieurs.

PL vs PLNE

L'algorithme du simplexe n'est (en général) pas polynomial... mais il l'est souvent !

Théorème (Leonid Khachiyan, 1979)

Le problème de trouver une solution à un PL est polynomial ("raisonnable").

Leonid Khachiyan (1952-2005)

$\longrightarrow$ Méthode de l'ellipsoïde ou des points intérieurs.

Théorème

Le problème de trouver une solution à un PLNE est NP-difficile (probablement pas "raisonnable").

En particulier : ensemble dominant (= couverture par des antennes), voyageur de commerce...

PL vs PLNE

L'algorithme du simplexe n'est (en général) pas polynomial... mais il l'est souvent !

Théorème (Leonid Khachiyan, 1979)

Le problème de trouver une solution à un PL est polynomial ("raisonnable").

Leonid Khachiyan (1952-2005)

→ Méthode de l'ellipsoïde ou des points intérieurs.

Théorème

Le problème de trouver une solution à un PLNE est NP-difficile (probablement pas "raisonnable").

En particulier : ensemble dominant (= couverture par des antennes), voyageur de commerce...

→ La méthode "brancher et borner" est généralement peu efficace !
(mais quand même mieux que tester toutes les possibilités)