

Méthodes d'optimisation

BUT Info 2e année

Florent Foucaud
Dipayan Chakraborty, Malika More, Adrien Wohrer



IUT CLERMONT AUVERGNE

Aurillac - Clermont-Ferrand - Le Puy-en-Velay
Montluçon - Moulins - Vichy

2022-2023

PL en nombres entiers

Solution pas entière ?

- Solution d'un PL : pas forcément des valeurs entières.

Solution pas entière ?

- Solution d'un PL : pas forcément des valeurs entières.
- Dans de nombreux contextes, nos variables doivent prendre une solution entière
(exemple : nombre de machines, groupes pour l'emploi du temps, etc).

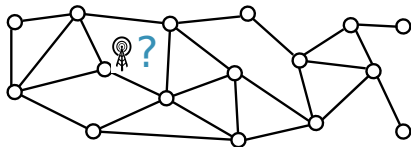
Solution pas entière ?

- Solution d'un PL : pas forcément des valeurs entières.
- Dans de nombreux contextes, nos variables doivent prendre une solution entière
(exemple : nombre de machines, groupes pour l'emploi du temps, etc).
- Pour cela on va modéliser des programmes linéaires en nombres entiers
(ou mixtes)

Couverture d'un réseau par des antennes

Problème : couvrir un **réseau** avec des **antennes** (*ensemble dominant*).

Objectif : **minimiser** le nombre d'antennes



Couverture d'un réseau par des antennes

Problème : couvrir un **réseau** avec des **antennes** (*ensemble dominant*).

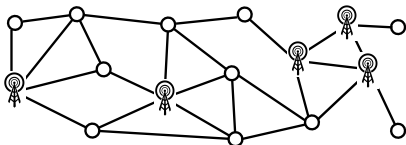
Objectif : **minimiser** le nombre d'antennes



Couverture d'un réseau par des antennes

Problème : couvrir un réseau avec des antennes (*ensemble dominant*).

Objectif : minimiser le nombre d'antennes



Le réseau est un graphe non-orienté $G = (V, E)$.

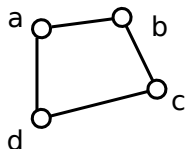
On écrit le PL en nombres entiers suivant :

Une variable x_v pour chaque sommet v : $x_v = 1$ si on a une antenne sur v , 0 sinon.

$$\text{minimiser : } \sum_{v \in V} x_v$$

$$\begin{aligned} \text{tel que : } \quad & \sum_{u \in E} x_u + x_v \geq 1 \quad \forall v \in V \\ & x_v \leq 1 \quad \forall v \in V \\ & x_v \geq 0 \quad \forall v \in V \\ & x_v \in \mathbb{N} \quad \forall v \in V \end{aligned}$$

Couverture d'un réseau par des antennes : exemple



Le réseau est un **graphe** non-orienté $G = (V, E)$, ici $V = \{a, b, c, d\}$ et $E = \{ab, bc, cd, de\}$.

On écrit le **PL en nombres entiers** suivant :

Une variable x_v pour chaque sommet v : $x_v = 1$ si on a une antenne sur v .

minimiser : $x_a + x_b + x_c + x_d$

tel que : $x_a + x_b + x_d \geq 1$ *Sommet a*

$x_b + x_c + x_a \geq 1$ *Sommet b*

$x_c + x_d + x_b \geq 1$ *Sommet c*

$x_d + x_c + x_a \geq 1$ *Sommet d*

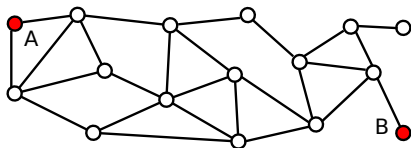
$x_a, x_b, x_c, x_d \leq 1$

$x_a, x_b, x_c, x_d \geq 0$

$x_a, x_b, x_c, x_d \in \mathbb{N}$

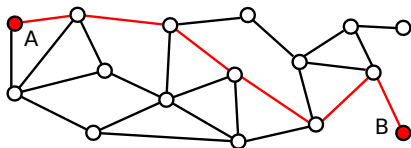
Plus court chemin

Problème : trouver un **plus court chemin** dans un réseau de A à B.



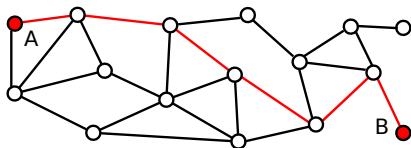
Plus court chemin

Problème : trouver un **plus court chemin** dans un réseau de A à B.



Plus court chemin

Problème : trouver un **plus court chemin** dans un réseau de A à B.



Le réseau est un **graphe** $G = (V, E)$.

On écrit le **PL en nombres entiers** suivant :

Une variable x_e pour chaque arête e : $x_e = 1$ si on sélectionne e , 0 sinon.

$$\text{minimiser : } \sum_{e \in E} x_e$$

$$\text{tel que : } \sum_{u: u \rightarrow v} x_{uv} = \sum_{w: v \rightarrow w} x_{vw} \quad \forall v \in V - \{A, B\}$$

$$\sum_{u: A \rightarrow u} x_{Au} = 1$$

$$\sum_{u: u \rightarrow B} x_{uB} = 1$$

$$x_e \leq 1 \quad \forall e \in E$$

$$x_e \geq 0 \quad \forall e \in E$$

$$x_e \in \mathbb{N} \quad \forall e \in E$$

La méthode du Branch and Bound

Exemple : presser ou tourner

Exemple 1

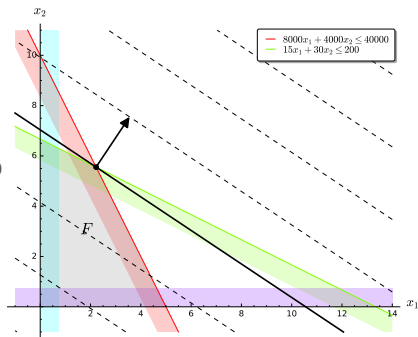
Un start-upper dispose d'un budget de 40k€ pour équiper son atelier de 200m² avec des presses et des tours.

- Une presse coûte 8k€, un tour 4k€.
- Une presse prend 15 m², un tour prend 30 m².
- Profit journalier d'une presse : 100 €, celui d'un tour : 150 €.

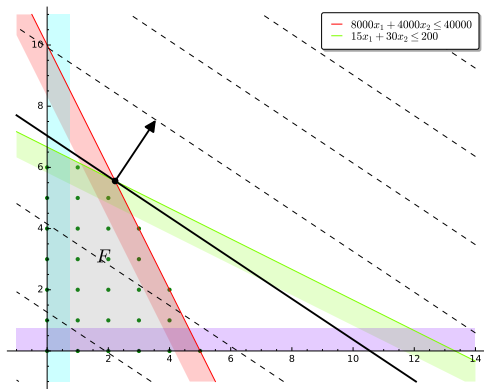
Presser ou Tourner

maximiser z :
tel que :

$$\begin{array}{rcll} 100x_1 & + & 150x_2 & \\ 8000x_1 & + & 4000x_2 & \leq 4000 \\ 15x_1 & + & 30x_2 & \leq 200 \\ x_1, x_2 & \geq & 0 & \end{array}$$



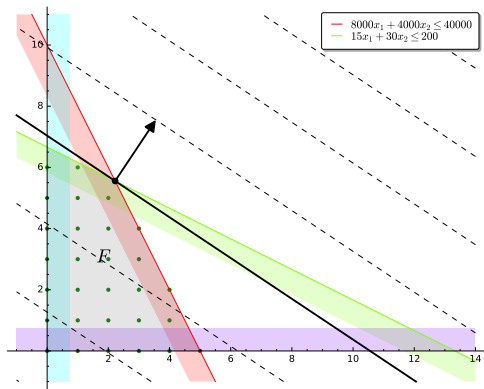
Solution non entière : que faire ?



Solution optimale : $z = \frac{9500}{9} \approx 1055.55$, avec $(x_1, x_2) = \left(\frac{20}{9}, \frac{50}{9}\right) \approx (2.22, 5.56)$

Mince : solution **non entière** ! → On ne peut pas acheter 2.22 presses et 5.56 tours.

Solution non entière : que faire ?

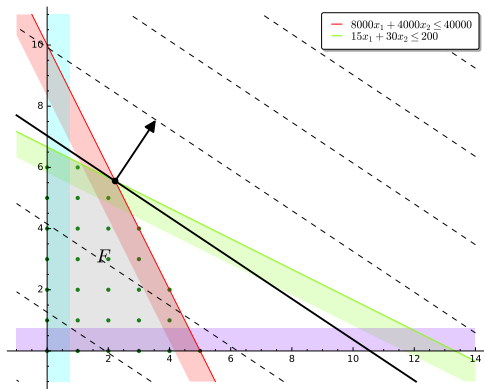


Solution optimale : $z = \frac{9500}{9} \approx 1055.55$, avec $(x_1, x_2) = \left(\frac{20}{9}, \frac{50}{9}\right) \approx (2.22, 5.56)$

Mince : solution **non entière** ! \rightarrow On ne peut pas acheter 2.22 presses et 5.56 tours.

Si on arrondit **vers le bas** la solution : on obtient $(2, 5)$ et $z = 950$.

Solution non entière : que faire ?



Solution optimale : $z = \frac{9500}{9} \approx 1055.55$, avec $(x_1, x_2) = \left(\frac{20}{9}, \frac{50}{9}\right) \approx (2.22, 5.56)$

Mince : solution **non entière** ! → On ne peut pas acheter 2.22 presses et 5.56 tours.

Si on arrondit **vers le bas** la solution : on obtient $(2, 5)$ et $z = 950$.

Autres solutions entières :

$(3, 4)$ avec $z = 900$

$(0, 6)$ avec $z = 900$

$(1, 6)$ avec $z = 1000$

Comment faire en général ?

- Dans notre petit exemple, il y a un petit nombre de combinaisons possibles, on pourrait juste les énumérer et calculer celle qui est la plus intéressante.
- Impossible en pratique : même si les variables sont à valeur dans $\{0, 1\}$ pour x , on aurait 2^x cas à regarder.
(Estimation du nombre de protons dans l'univers : $10^{80} < 2^{266}$ — nombre d'Eddington)

Comment faire en général ?

- Dans notre petit exemple, il y a un petit nombre de combinaisons possibles, on pourrait juste les énumérer et calculer celle qui est la plus intéressante.
- Impossible en pratique : même si les variables sont à valeur dans $\{0, 1\}$ pour x , on aurait 2^x cas à regarder.
(Estimation du nombre de protons dans l'univers : $10^{80} < 2^{266}$ — nombre d'Eddington)

Idée : On va couper l'espace de recherche en petit bouts

On choisit une **variable x non-entière** dans la solution. On regarde les 2 problèmes produits en forçant x à prendre soit une valeur inférieure, soit une valeur supérieure (**brancher**), et on réitère.

Brancher

problème initial

$$\begin{aligned} \text{maximiser :} \quad & 100x_1 + 150x_2 \\ \text{tel que :} \quad & 8000x_1 + 4000x_2 \leq 40000 \\ & 15x_1 + 30x_2 \leq 200 \\ & x_1, x_2 \geq 0 \end{aligned}$$

solution optimale pas entière $\bar{x} = (\frac{20}{9}, \frac{50}{9}) \approx (2.22, 5.56)$

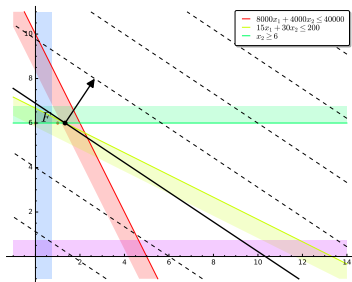
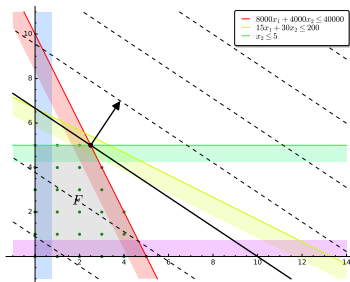
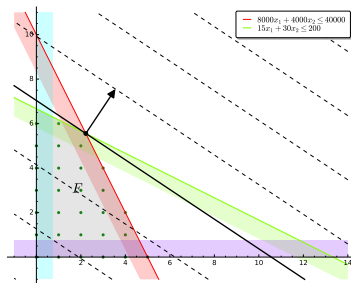
branche gauche

$$\begin{aligned} \text{maximiser :} \quad & 100x_1 + 150x_2 \\ \text{tel que :} \quad & 8000x_1 + 4000x_2 \leq 40000 \\ & 15x_1 + 30x_2 \leq 200 \\ & \quad \quad \quad x_2 \leq 5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

branche droite

$$\begin{aligned} \text{maximiser :} \quad & 100x_1 + 150x_2 \\ \text{tel que :} \quad & 8000x_1 + 4000x_2 \leq 40000 \\ & 15x_1 + 30x_2 \leq 200 \\ & \quad \quad \quad x_2 \geq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Brancher (graphiquement)



Borner

problème initial

$$\begin{aligned} \text{maximiser :} \quad & 100x_1 + 150x_2 \\ & 8000x_1 + 4000x_2 \leq 40000 \\ & 15x_1 + 30x_2 \leq 200 \\ x_1, x_2 \geq 0 \end{aligned}$$

branche gauche

$$\begin{aligned} \text{maximiser :} \quad & 100x_1 + 150x_2 \\ & 8000x_1 + 4000x_2 \leq 40000 \\ & 15x_1 + 30x_2 \leq 200 \\ & \quad \quad \quad x_2 \leq 5 \\ x_1, x_2 \geq 0 \\ z^* = 1000 \end{aligned}$$

branche droite

$$\begin{aligned} \text{maximiser :} \quad & 100x_1 + 150x_2 \\ & 8000x_1 + 4000x_2 \leq 40000 \\ & 15x_1 + 30x_2 \leq 200 \\ & \quad \quad \quad x_2 \geq 6 \\ x_1, x_2 \geq 0 \\ z^* = \frac{3100}{3} \approx 1033 \end{aligned}$$

Brancher et Borner (Branch and Bound)

Exploration d'un arbre

- On découvre des branches en ajoutant de **nouvelles contraintes** pour une variable (au dessus/ en dessous d'une valeur non entière dans la solution précédente).
- Borne inf au cours du temps : meilleur z d'une solution **entière** rencontrée.
- Borne sup (à un noeud) : z^* pour le problème de ce noeud (problème relâché puisque pas forcément une solution entière).
- On ignore définitivement une branche si elle n'a pas de solution, ou bien si la borne sup associée est plus basse que la borne inf.
- On peut s'arrêter si on trouve une solution **entière** optimale à un noeud qui a un z^* plus grand ou égal que toutes les bornes sups des autres noeuds.

Note historique

La méthode “branch and bound” est développée en 1960 par deux chercheuses à Londres, Alison Harcourt (née Doig) et Ailsa Land (née Dicken).



Alison G. Harcourt
(1929-)



Ailsa H. Land
(1927-2021)

Complexité algorithmique de la PL et PLNE

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** $f(n) \rightarrow 10n, 2n, 1000n, n \dots$

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

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

→ 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** $f(n) \rightarrow 10n, 2n, 1000n, n \dots$

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

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

→ 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** $f(n) \rightarrow 10n, 2n, 1000n, n \dots$

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

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

→ 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**
arrêt(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**
arrêt(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 arrêt(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**
`arrêt(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 `arrêt(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**
`arrêt(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 `arrêt(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, \dots, a_n et b_1, \dots, 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, \dots, a_n et b_1, \dots, 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)

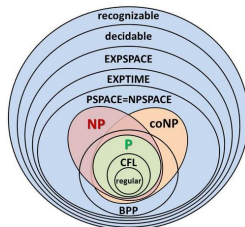
Liens avec la logique mathématique :

théorème d'incomplétude de Gödel (1931)



Kurt Gödel (1906-1978)

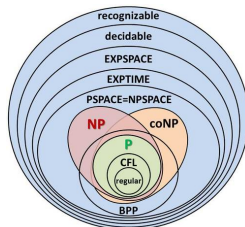
Quelques classes de complexité algorithmique



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

Au-dessus : problèmes (probablement) algorithmiquement difficiles

Quelques classes de complexité algorithmique



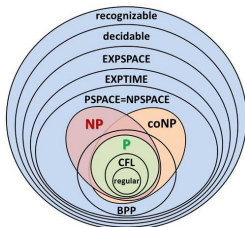
Classe P (pour “polynomiaux”) : problèmes “raisonnables”

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)

→ 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 souvent peu efficace !
(mais quand même mieux que tester toutes les possibilités)