Recherche Opérationnelle
Première Partie

Paul Feautrier

ENS Lyon

1er novembre 2005

Plan

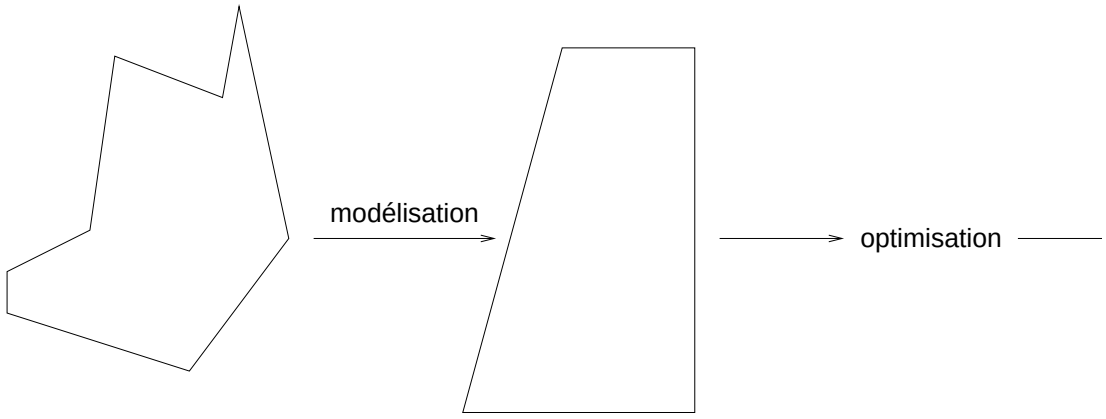
- Introduction et principaux concepts
- Optimisation continue sans contrainte
- Programmation linéaire
- Optimisation continue sous contrainte
- Optimisation combinatoire
  - Programmation linéaire en nombres entiers.
  - Exploration
  - Métaheuristiques
  - Programmation dynamique
- Éléments de Complexité

Plan

Qu'est ce que la recherche opérationnelle ?

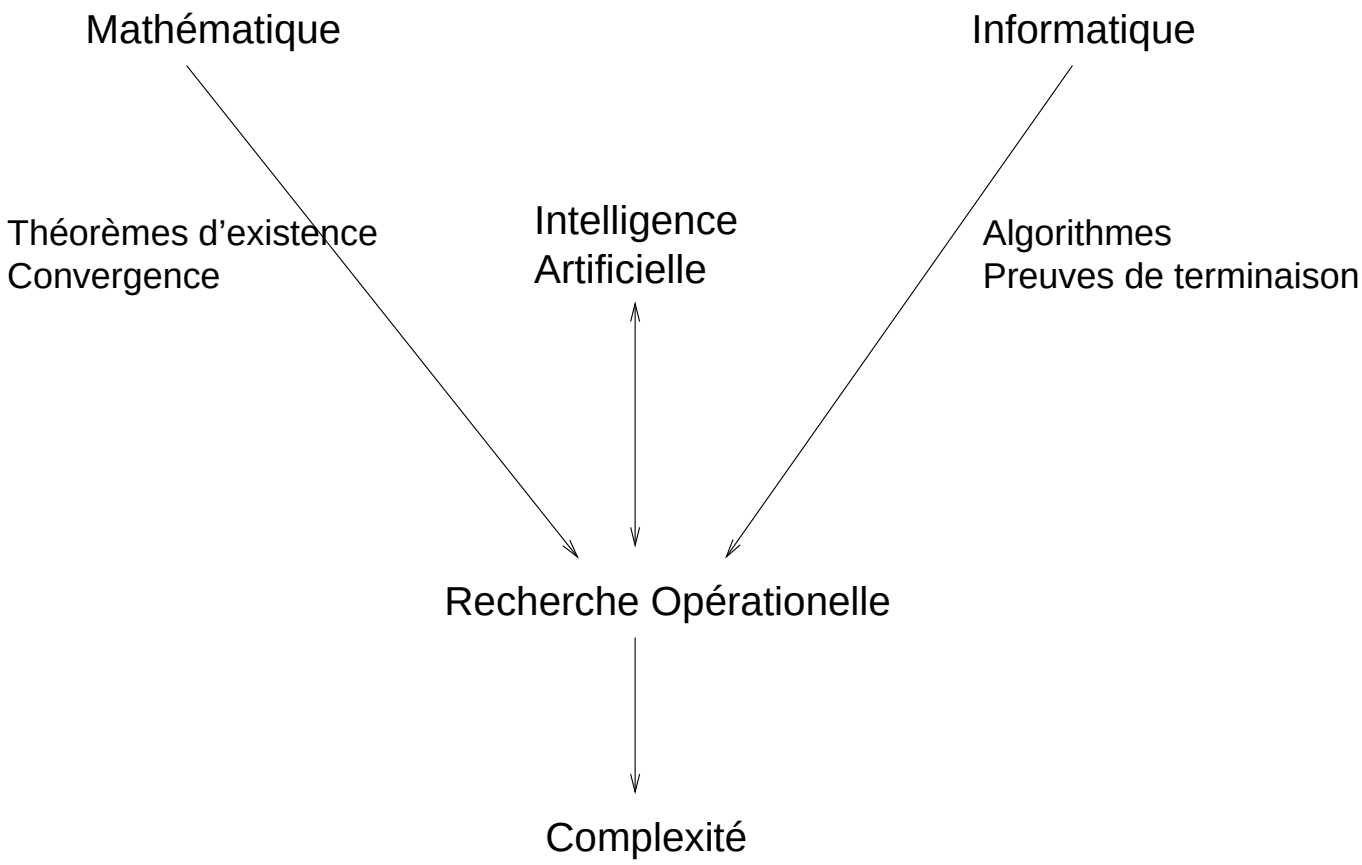
- Vocabulaire: Recherche opérationnelle = programmation mathématique = optimisation (mais pas optimisation de programme).
- Recherche opérationnelle = modélisation mathématique des processus de prise de décision.
  - Inconnues : les variables de décision.
  - Évaluation de la décision = fonction économique ou fonction «objectif».
  - Trouver les valeurs des variables de décision qui minimisent (ou maximisent) la fonction objectif.

Recherche opérationnelle



- La modélisation est un art, l'optimisation est une science.
- Applications : planification du débarquement de Normandie, optimisation d'un programme de calcul intensif, investissement en bourse.
  - Investissement en bourse = optimisation avec information incomplète ou aléatoire.
  - Planification d'une opération militaire = il y a un adversaire = théorie des jeux.
  - Optimisation d'un programme = en principe, on a une information complète.
  - Le cours est essentiellement consacré à l'optimisation avec

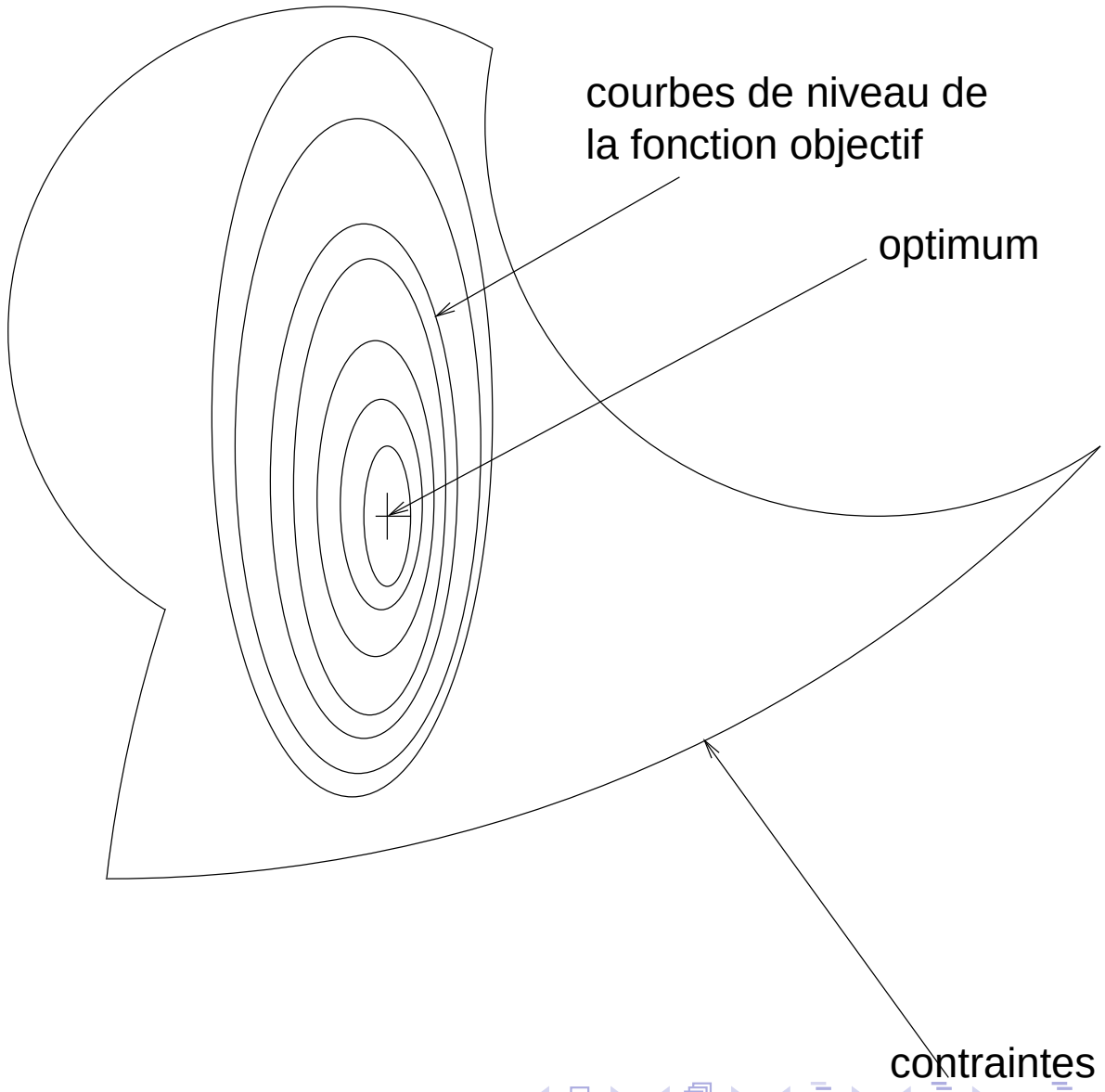
Informatique ou mathématique ?



Vocabulaire

Forme canonique :
trouver $x \in D$
qui minimise f.

$$\begin{array}{ll} \min & f(x) \\ x & \in \quad D \end{array}$$



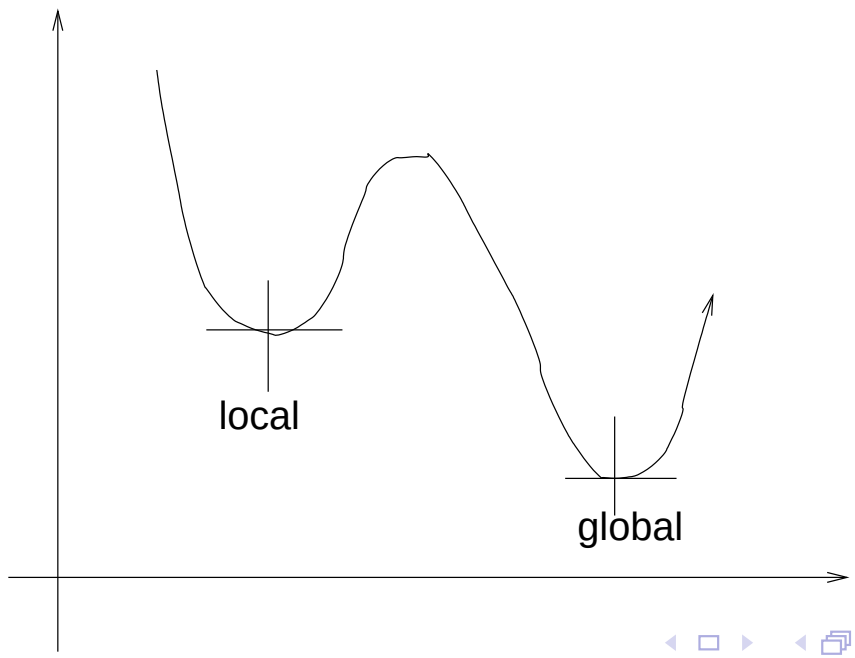
Optimum local, global

- Minimum local : a est un minimum local de f s'il existe un voisinage V de a tel que :

$$x \in V \Rightarrow f(x) \geqslant f(a).$$

- Minimum global : a est un minimum global de f dans D si et seulement si :

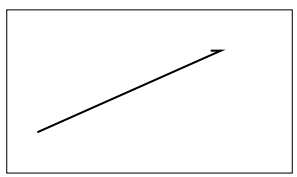
$$x \in D \Rightarrow f(x) \geqslant f(a).$$



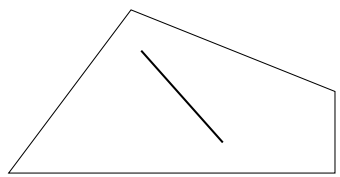
Convexité

Un ensemble S est convexe si, pour toute paire de points a, b de S, S contient aussi le segment ab.

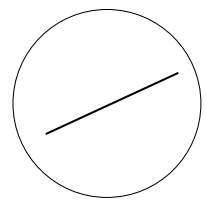
$$a, b \in S \Rightarrow (0 \leqslant \lambda \leqslant 1 \Rightarrow \lambda a + (1 - \lambda) b \in S.$$



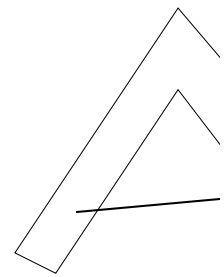
convexe



convexe



convexe

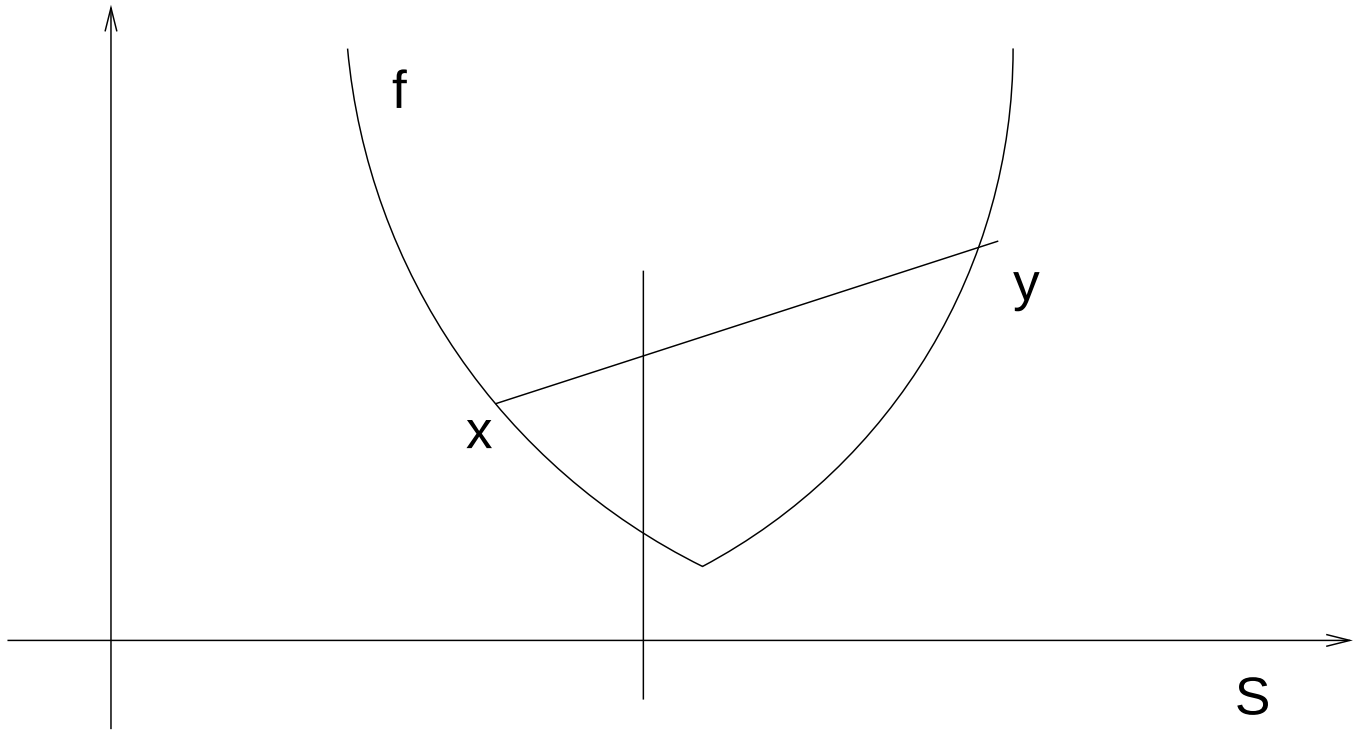


non

Fonction convexe

- f est convexe dans un ensemble convexe S si et seulement si :

$$x, y \in S, 0 \leqslant \lambda \leqslant 1 \Rightarrow f(\lambda x + (1-\lambda)y) \leqslant \lambda f(x) + (1-\lambda) f(y)$$



Intérêt de la convexité

Théorème

Si f est convexe dans un ensemble convexe S, alors tout minimum local de f est un minimum global.

Démonstration.

Soit a un minimum local, et V l'ouvert contenant a dans lequel :

$$x \in V \Rightarrow f(x) \geqslant f(a).$$

Si on suppose qu'il existe un point $b \in S$ tel que $f(b) < f(a)$ alors on a :

$$f(\lambda a + (1-\lambda)b) \leqslant \lambda f(a) + (1-\lambda) f(b).$$

Il est possible de trouver un λ suffisamment proche de 1 pour que $x = \lambda a + (1-\lambda)b$ soit dans V. Contradiction en ce point :

$$f(x) \leqslant \lambda f(a) + (1-\lambda) f(b) \leqslant f(a),$$

□

Classification

- Selon la nature des variables de décision :
  - Optimisation continue.
  - Optimisation discrète ou optimisation combinatoire.
- Selon la nature des contraintes :
  - Pas de contraintes ou contraintes faciles à satisfaire (un segment de la droite réelle) : optimisation sans contraintes.
  - Optimisation sous contraintes : il est difficile de trouver un point satisfaisant les contraintes.
- Propriétés spéciales des éléments du problème : linéarité, convexité.

Plan

Un exemple: l'allocation de registre

- Les processeurs modernes utilisent des registres pour éviter les accés à la mémoire.
- Il est intéressant de conserver une donnée en registre le plus longtemps possible ...
- Mais les registres sont en nombre fini.
- Le compilateur commence par écrire le code comme s'il y avait un nombre illimité de registres *virtuels*, puis alloue les registres virtuels aux registres *physiques*.
- Il ne faut pas utiliser plus de registres physiques qu'il n'y en a sur le processeur.

Plan

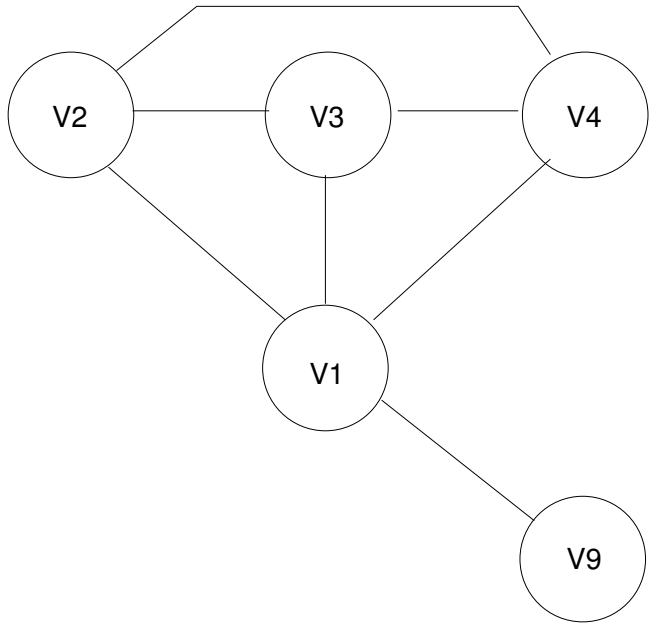
Intervalle de vivacité

- **Bloc de base** : suite d'instructions sans test et sans goto entrant ni sortant. En première approximation, on alloue les registres indépendemment pour chaque bloc de base.
- Un registre virtuel est vivant depuis sa première écriture jusqu'à sa dernière lecture.

```
1   v1 = stack ;        [1,11]
2   v2 = *v1 ;          [2,7]
3   v3 = 2 ;            [3,7]
4   v4 = *(v1+8) ;      [4,8]
5   v5 = *(v1+16) ;     [5,9]
6   v6 = *(v1+24) ;     [6,10]
7   v7 = v2*v3 ;        [7,8]
8   v8 = v7*v4 ;        [8,9]
9   v9 = v8*v5 ;        [9,10]
10  v10 = v9*v6 ;       [10,11]
11  *v1 = v10 ;
```

Graphe d'interférence

- Deux registres virtuels interfèrent s'ils peuvent être simultanément vivants, i.e. si leurs intevalles de vivacité ont une intersection nulle.
- Deux registres virtuels qui interfèrent ne peuvent pas être alloués au même registre physique.
- On peut construire le graphe d'interférence d'un bloc de base.



Colorier le graphe d'interférence

- **Contrainte** : Il faut affecter un registre physique à chaque registre virtuel de façon à ce que deux registres virtuels adjacents ne soient pas affectés au même registre physique.
- **Objectif** minimiser le nombre de registres physiques.
- C'est un problème classique de coloriage de graphe.
- On peut le prendre de deux façons:
  - Existe-t-il un coloriage avec k couleurs ?
  - Quel est le nombre minimum de couleurs nécessaires (nombre chromatique = *register pressure*).
- On va se concentrer sur la deuxième question.

Plan

NP-complétude

- On doit toujours commencer par une estimation de la difficulté du problème.
- Problème NP: problème dont on peut *vérifier* la solution en temps polynomial.
- Il existe une liste (provisoire) de problèmes NP pour lesquels on ne sait pas (aujourd'hui) trouver la solution en temps polynomial, et qui peuvent être réduits l'un à l'autre en temps polynomial : les problèmes NP-complets. Exemples canoniques : SAT (une formule booléenne est-elle satisfiable) et PLNE (un système d'inéquations en nombres entiers est-il satisfiable).
- Le problème du coloriage est NP-complet: on peut le réduire à un problème SAT, et on peut réduire un problème SAT à un problème de coloriage.
- Il ne faut donc pas espérer une solution facile.

Programmation linéaire en nombres entiers

- On numérote les couleurs par des entiers ; soit c_i la couleur du sommet V_i. Les contraintes sont:

$$\begin{aligned} c_i &\geqslant 0 \\ c_i &\neq c_j, V_i \text{ adjacent } V_j. \end{aligned}$$

- Objectif: minimiser $\max c_i$:

$$\begin{aligned} \min \quad & z, \\ z \quad \geqslant \quad & c_i. \end{aligned}$$

- Pour rendre le problème convexe, on pose: $x_{ij} = 1$ ssi $c_i > c_j$. On obtient un problème en variables entières dont les contraintes sont:

$$c_i - c_j + (1 - x_{ij})M > 0, \quad c_j - c_i + x_{ij}M > 0.$$

où M est un "grand nombre". On s'est ramené à un PLNE.

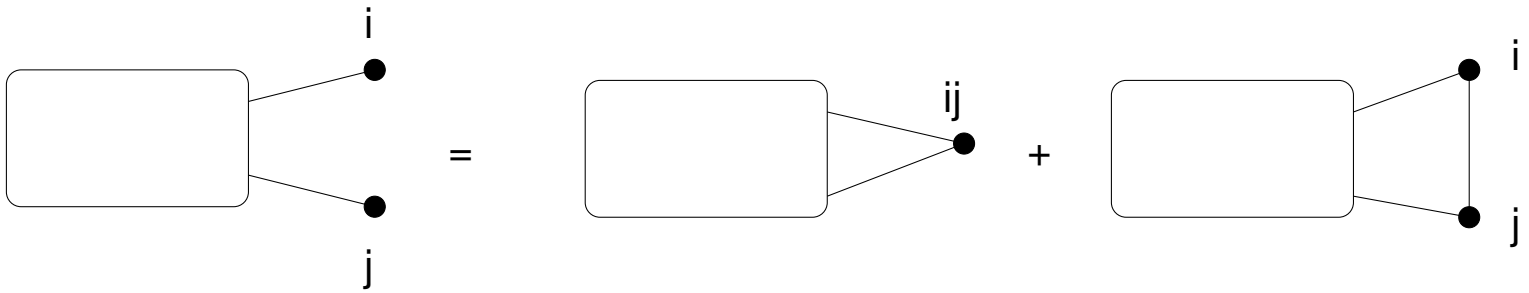
Branch-and-Bound, I/III

- **Méthode** : on construit un arbre de sous-problèmes en divisant l'espace des solutions récursivement en deux.
- La recherche s'arrête quand la solution est "évidente".
- On utilise une borne inférieure pour ignorer les sous-problèmes qui ne sont pas "prometteurs".

Branch-and-Bound, II/III

- On engendre les sous-problèmes en remplaçant $c_i \neq c_j$ par $c_i > c_j$ ou par $c_j > c_i$. Chaque sous-problème est un problème de plus court chemin, qui se résoud en temps polynomial.
- S'il n'y a pas de solution (cycle de poids positif) il est inutile de poursuivre l'exploration.
- Sinon, on compare la valeur du problème à la meilleure solution connue. On arrête l'exploration si elle est supérieure.
- On obtient une solution (candidate) quand on a exploité toutes les contraintes.

Branch-and-Bound, III/III



- Sélectionner deux sommets i et j non connectés.
- Il y a deux possibilités:
  - Ils ont la même couleur: on peut les fusionner.
  - Ils ont des couleurs différentes: on peut ajouter l'arc $i - j$.
- La division s'arrête quand on a formé un graphe complet, dont le coloriage est trivial.
- On peut avoir une borne inférieure en cherchant une *clique* dans le graphe.

Recherche aléatoire

- **Initialisation** : on part d'un coloriage légal arbitraire: par exemple, on donne à chaque sommet une couleur différente.
- **Exploration** : on choisit au hasard un sommet et on change sa couleur au hasard.
- **Progression** : on conserve le nouveau coloriage s'il est légal et s'il améliore la solution (moins de couleurs).
- Il existe de nombreuses variantes sur le même thème (recuit simulé, tabou, algorithmes génétiques): voir la suite du cours.

Plan

Conclusion

- De nombreux problèmes de la vie pratique (emploi du temps), de programmation (utilisation optimale d'un réseau), de compilation (allocation de registres, parallélisation) peuvent se mettre sous la forme "optimisation sous contraintes".
- Les inconnues peuvent être continues ou discrètes (couleurs).
- Avantage: Il existe toute une panoplie de méthodes de résolution efficace.
- Toute l'intelligence se concentre dans la phase de modélisation, pour laquelle il n'existe que des "recettes de cuisine".
- Le modèle peut être plus ou moins détaillé (exemple: allocation de registres en tenant compte des identités remarquables de l'algèbre).

Plan

Optimisation continue sans contrainte

On considère une seule variable

$$\begin{array}{ccc} \mathit{min} & f(x) \\ x & \in & \mathbb{R} \end{array}$$

- Si on connaît la dérivée f' de f, le problème se ramène à trouver les racines de f', puis à les tester une par une pour savoir si elles sont un minimum, un maximum ou un point d'inflexion.
- On peut utiliser pour cela des méthodes classiques : itération de Newton (si on peut calculer la dérivée seconde), dichotomie, méthode de la sécante.
- Si on ne connaît pas la dérivée, la première chose à faire est de trouver un encadrement du minimum. Il n'y a pas de méthode générale, on utilise les renseignements que l'on peut avoir sur f.

Fonctions unimodales

- Une fonction f est unimodale dans l'intervalle $[a, b]$ s'il existe un point $a \leqslant c \leqslant b$ tel que si $x \leqslant y \leqslant c$ alors $f(x) \geqslant f(y)$ et si $c \leqslant x \leqslant y$ alors $f(x) \leqslant f(y)$.
- Il est évident que si f est unimodale dans $[a, b]$, alors c est un minimum global.

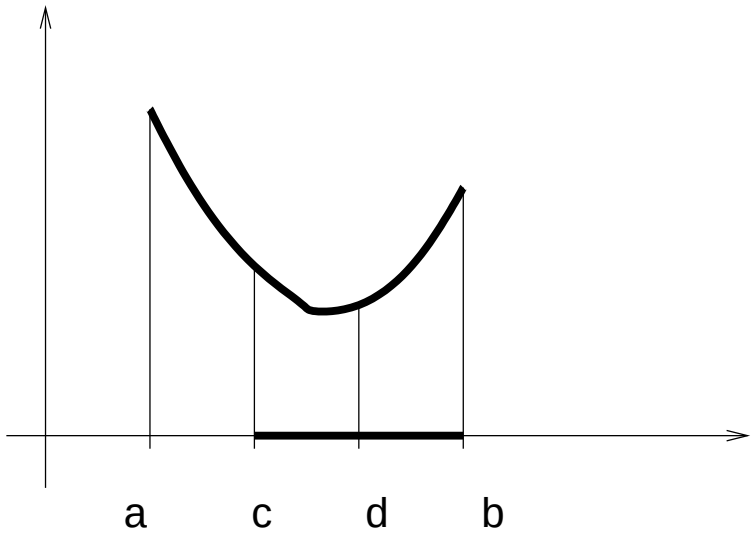
Théorème

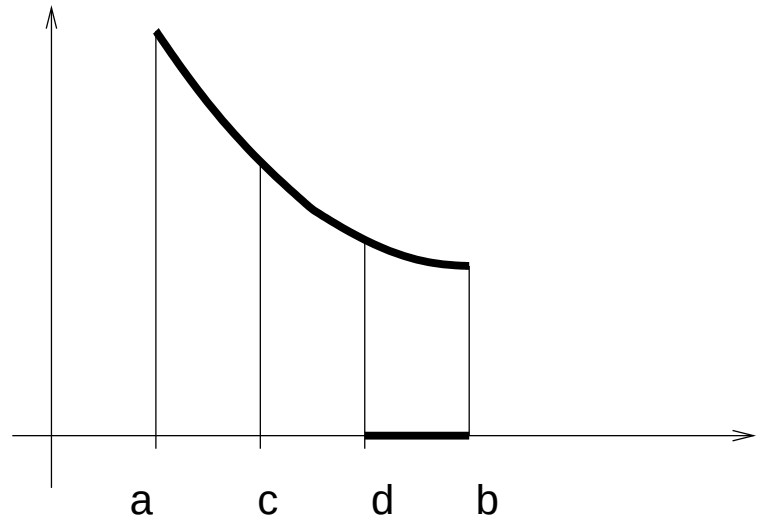
Si f est continue convexe dans $[a, b]$, alors f est unimodale.

Démonstration.

Soit c le minimum de f, et x et y qui violent la condition d'unimodalité, par exemple $c \leqslant x \leqslant y$ et $f(c) < f(y) < f(x)$. Soit $0 \leqslant \lambda \leqslant 1$ tel que $x = \lambda c + (1 - \lambda)y$. Par convexité on doit avoir $f(x) \leqslant \lambda f(c) + (1 - \lambda)f(y)$ mais aussi $\lambda f(c) + (1 - \lambda)f(y) \leqslant f(y) < f(x)$ une contradiction. □

Méthode par trichotomie





- On divise l'intervalle $[a, b]$ en trois parties égales à l'aide des points $a < c < d < b$. On calcule $f(c)$ et $f(d)$.
- On détermine le minimum de f parmi les 4 points a, c, b, d.
- Le minimum continu appartient à l'intervalle encadrant le minimum discret.
- L'intervalle est réduit au moins par un facteur 2/3. On poursuit jusqu'à la précision voulue.

Améliorations

- Vitesse de convergence : la taille de l'intervalle est multipliée par $\frac{2}{3}^{n/2}$ après n évaluations de la fonction.
- La division en segment égaux n'est pas optimale. On a intérêt à agrandir les segments extrêmes.
- On peut passer à un découpage en 4 parties égales. Il faut évaluer f trois fois à chaque étape, mais l'intervalle est au moins divisé par 2. La convergence est en $\frac{1}{2}^{n/3}$ donc plus rapide.

Méthodes par exploration

- Il est toujours indispensable de connaître une encadrement $[a, b]$ du minimum.
- On suppose toujours que f est unimodale. On choisit un pas d'exploration d. On calcule f aux points $x_k = a + k.d$ jusqu'à trouver soit une configuration $f(x_k) \leqslant f(x_{k+1})$, soit jusqu'à atteindre le point b.
- On fait $a := x_{k-1}, b := x_{k+1}, d := \xi d$ et on recommence. $\xi < 1$ est le facteur de convergence.
- On s'arrête quand d est devenu suffisamment petit.
- La méthode peut s'appliquer à une fonction non unimodale. On obtient alors un optimum local sans garantie qu'il soit global.

L'algorithme converge

- Le nombre de pas d'exploration est au plus $\frac{b-a}{b}$. L'exploration s'arrête en un temps fini.
- Soit $[a_n, b_n]$ le n^{e} intervalle d'exploration et d_n le n^{e} pas d'exploration. On a $d_n = d\xi^n$ et $b_n - a_n \leqslant 2d_n$.
- Les $[a_n, b_n]$ forment une suite d'intervalles emboîtés dont la longueur tend vers 0. Ils convergent donc vers une limite c.

Optimisation à plusieurs variables

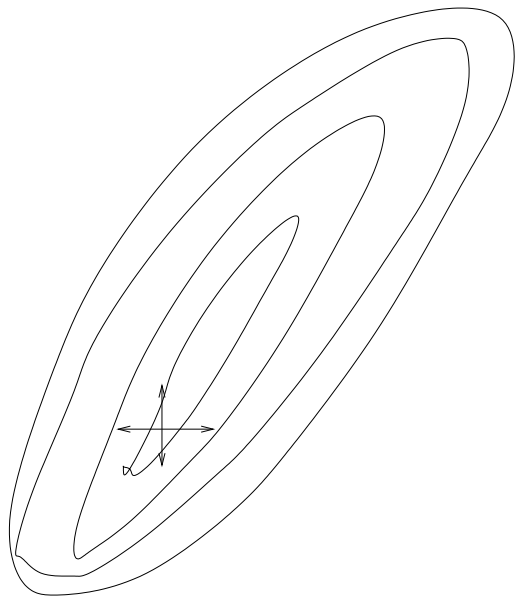
- Forme du problème : $\begin{array}{ccc} & \min & f(x) \\ x & \in & \mathbb{R}^n \end{array}$
- Les inconnues sont les n composantes du vecteur x.
- La notion de fonction unimodale ne se généralise pas.

Recherche directionnelle

- On se ramène au cas à une seule variable. Pour cela on choisit un point de départ a et une direction d.
- On minimise la fonction à une variable $f(a + t.d)$ à l'aide de l'une des méthodes vues plus haut.
- Si le déplacement $t.d$ est suffisamment petit, on arrête.
- Sinon, on change de direction et on recommence.
- Le point important est le choix des directions.

Recherche suivant les axes

- On prend comme directions les vecteurs canoniques de la base. Ceci revient à fixer $n-1$ variables de la fonction f, et à optimiser suivant la n^{e}. On passe ensuite à la variable suivante.
- La méthode est très lente et peut même ne pas converger si les courbes de niveau sont à peu près parallèles aux diagonales.
- On peut l'accélérer en effectuant N pas puis en utilisant la direction $a_N - a_0$.



Gradient

- On suppose que la fonction f a une dérivée.
- Le gradient de f au point a est le vecteur $(\frac{\partial f}{\partial x_1}(a), \ldots, \frac{\partial f}{\partial x_n}(a))^T$. On le note $\nabla f(a)$.
- On a le développement de Taylor :

$$f(a+h) = f(a) + h\nabla f(a) + \ldots.$$

  Ceci montre que $-\nabla f(a)$ est la direction dans laquelle f décroît le plus rapidement (*steepest descent*).
- D'où l'algorithme :
  1. Calculer le gradient $\nabla f(a)$.
  2. Minimiser la fonction à une variable $f(a - x\nabla f(a))$.
  3. Si le critère de convergence n'est pas vérifié, recommencer en 1.

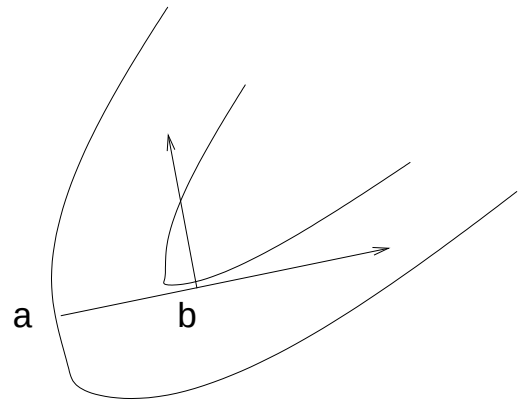
Propriété

Théorème

Dans l'algorithme ci-dessus, les directions de recherche successives sont orthogonales.

Démonstration.

Soit a le point de départ
d'une recherche unidimensionnelle suivant
la direction $\nabla f(a)$ et b son point d'arrivée.
La dérivée de la fonction à minimiser est :



$$\frac{df(a - x\nabla f(a))}{dx} = -\nabla f(a).\nabla f(x).$$

En b cette dérivée est nulle, d'où la propriété. □

Directions conjuguées

- Une matrice A de dimension $n \times n$ est définie positive si et seulement si :

$$\forall x : x^T Ax \geqslant 0.$$

  Une matrice définie positive définit un produit scalaire.

- Deux vecteurs u, v sont conjugués par rapport à A si et seulement si : $u^T Av = 0$. C'est une généralisation de la notion d'orthogonalité.
- Soit n vecteurs $d_1, \ldots, d_n$ mutuellement conjugués :

$$i \neq j \Rightarrow d_i^T . A . d_j = 0.$$

- Soit la fonction $f(x) = 1/2x^T Ax + b^T x + c$. Si on la minimise successivement suivant les directions $d_1, \ldots, d_n$, on atteint le minimum exact en n étapes.

Notations

- Soit $x^{(k)}, k = 0, \ldots$ les minima successifs.
- $x^{(k+1)} = x^{(k)} + \lambda_k d_k$.
- Le gradient de f en x est $Ax + b$.
- D'après la propriété ci-dessus et la conjugaison des d_k, on a :

$$\lambda_k = -\frac{d_k^T (Ax^{(0)} + b)}{d_k^T A d_k}.$$

- Les coordonnées de $x^{(k+1)}$ sont données par la formule :

$$x^{(k+1)} = x^{(0)} + \sum_{i=1}^{k} \lambda_i d_i.$$

Preuve

Lemme

En tout point $x^{(k)}$ le gradient de f est orthogonal au sous-espace engendré par $d_1, \ldots, d_k$.

Démonstration.

Le gradient en $x^{(k)}$ est $Ax^{(k)} + b = Ax^{(0)} + \sum_{i=1}^{k-1} \lambda_j A d_j + b$. Si on multiplie par d_i^T et qu'on remplace λ_j par sa valeur il vient :

$$d_i^T A x^{(0)} - \frac{d_i^T (Ax^{(0)} + b)}{d_j^T A d_j} . (d_i^T A d_j) + d_i^T . b = 0.$$

□

Théorème

Le point $x^{(n)}$ est le minimum de f.

Démonstration.

En effet, le gradient en $x^{(n)}$ doit être conjugué de n vecteurs linéairement indépendants, et donc doit être nul. □

Gradient conjugué

- C'est la transposition de la méthode ci-dessus au cas où la fonction f est quelconque, mais où on sait calculer son gradient.
- On part d'un point a_0 et on pose $d_0 = -\nabla f(a_0)$.
- Supposons que l'on soit parvenu en un point a_k avec la direction d_k. On minimise $f(a_k + \lambda.d_k)$. Soit λ_k la solution obtenue.
- On pose :

$$\begin{aligned} a_{k+1} &= a_k + \lambda_k.d_k, \\ \beta_k &= \frac{||\nabla f(a_{k+1})||^2}{||\nabla f(a_k)||^2}, \\ d_{k+1} &= -\nabla f(a_{k+1}) + \beta_k d_k. \end{aligned}$$

- On montre que si f est quadratique définie positive, la méthode est identique à celle des directions conjugués et converge en n étapes.

Recherche aléatoire

- Au lieu de calculer la direction de recherche optimale pour une approximation quadratique de f, on peut la choisir aléatoirement en tirant n nombres au hasard dans l'intervalle $[0, 1]$.
- La méthode ne nécessite pas le calcul du gradient. Elle fonctionne même si f n'est pas dérivable.
- Mais en général, sa convergence est beaucoup plus lente.

Test d'arrêt

- Le choix d'un test d'arrêt est difficile.
- Si f est dérivable, son gradient doit être nul à l'optimum. On peut donc utiliser $||\nabla f(a_k)|| < \varepsilon$ comme test d'arrêt.
- Sinon, on peut arrêter les itérations quand la solution ne change plus : $||a_{k+1} - a_k|| < \varepsilon$.
- ε doit refléter la précision requise. Il ne doit pas être plus petit que la précision des calculs sous peine de blocage.
- Il est prudent d'attendre que le test ait été satisfait plusieurs fois avant d'arrêter.
- En général, la valeur du minimum est mieux définie que sa position.

Plan

Programmation linéaire

$$\begin{aligned} min \quad & c.x \\ Ax + b \quad & \geqslant \quad 0 \end{aligned}$$

- x est le vecteur des inconnues, de dimension n.
- A est la matrice des contraintes, de dimension $m \times n$.
- b est le terme constant, de dimension m.
- c de dimension n est le gradient de la fonction objectif.

Autres formes d'un programme linéaire

- Un programme linéaire peut se mettre sous de multiples formes, toutes équivalentes.
- On peut changer le sens de l'inégalité, ou passer le terme constant de gauche à droite.
- On peut remplacer les inégalités par des égalités en introduisant des *variables d'écart* toutes positives:

$$Ax + b \geqslant 0 \equiv Ax + b - y = 0, y \geqslant 0$$

- On peut imposer que toutes les variables soient positives, en posant $x := x^+ - x^-, \quad x^+, x^- \geqslant 0$
- On peut enfin transposer le programme :
  $Ax + b \geqslant 0 \equiv x^T A^t + b^t \geqslant 0$

Polyèdre Convexe

- L'ensemble $\mathcal{P} = \{x \mid Ax + b \geqslant 0\}$ est convexe. On l'appelle un *polyèdre convexe* ou simplement un polyèdre.
- La fonction $c.x$ est trivialement convexe.
- Donc, si un programme linéaire a un minimum local, c'est un minimum global.

Test de faisabilité

- Pour trouver un minimum, il faut que le polyèdre :

$$\mathcal{P} = \{x \mid ax = b \geqslant 0\}$$

  soit non vide, ou encore qu'il existe au moins un point x_0 qui satisfasse toute les inégalités $Ax + b \geqslant 0$.
- On peut vérifier cette condition à l'aide de l'algorithme d'élimination de Fourier-Motzkin.
- Notations : $x^{(n)}$ le vecteur x amputé de ses n premières composantes. $x^{(0)} = x$.
- $A^{(n)}x^{(n)} + b^{(n)} \geqslant 0$ le système obtenu après l'élimination de n variables.

Test de Fourier-Motzkin I / II

- Soit à éliminer x_1. On réparti les contraintes en trois classes :
  - $k \in I_0$ ssi $a_{k1} = 0$.
  - $k \in I_+$ ssi $a_{k1} > 0$.
  - $k \in I_-$ ssi $a_{k1} < 0$.
- Dans une contrainte de I_0, l'inconnue x_1 est déjà éliminée.

Test de Fourier-Motzkin II / II

- Une contrainte $k \in I_+$ donne une borne inférieure de x_1 :

$$x_1 \geqslant -\frac{b_k + a_{k,2}x_2 + \dots}{a_{k1}};$$

- Une contrainte $k \in I_-$ donne une borne supérieure de x_1 :

$$x_1 \leqslant \frac{b_k + a_{k,2}x_2 + \dots}{-a_{k1}};$$

- Pour éliminer x_1, il suffit d'écrire que chaque borne inférieure est inférieure à chaque borne supérieure.
- On poursuit jusqu'à élimination de toutes les variables. Au bout de n étapes, le système est de la forme : $b^{(n)} \geqslant 0$, qu'il suffit d'inspecter.

Correction

On dit que le test réussit si $b^{(n)} \geqslant 0$, et qu'il échoue dans le cas contraire.

Théorème

Si le test échoue, alors le système initial est infaisable.

Démonstration.

Supposons *a contrario* que le système initial a une solution u. Les transformations effectuées sur les contraintes sont de simples manipulations algébriques valides ; on en conclu que les intervalles obtenus en comparant une borne inférieure et une borne supérieure sont non vides, et donc que le système $A^{(1)}x(1) + b(1) \geqslant 0$ est faisable.
En poursuivant l'élimination, on en arrive au système d'ordre $n-1$, qui n'a plus qu'une seule inconnue x_n et qui est également faisable. Mais le fait que l'un des $b^{(n)} < 0$ indique que l'un des intervalles de variation de x_n est vide, une contradiction. □

Complétude

Théorème

Si le test réussit, le système initial est faisable.

Démonstration.

On exhibe une solution du système initial en la construisant de proche en proche à partir de sa dernière composante. On part du système

$$A^{(n-1)}x^{(n-1)} + b^{(n-1)} \geqslant 0.$$

Le fait que les $b^{(n)} \geqslant 0$ garantit que l'intervalle des valeurs possibles de x_n est non vide. On en choisit une arbitrairement et on la reporte dans le système d'ordre $n-2$. Ce système n'a plus qu'une inconnue, x_{n-1}, dont l'intervalle des valeurs possibles est non vide. On poursuit ainsi jusqu'à avoir donné une valeur à toutes les composantes de x. □

Remarques

- Si on s'astreint à choisir à chaque pas la solution la plus petite, *i.e.* la borne inférieure de l'intervalle de variation, on obtient le minimum lexicographique de P, les inconnues étant prises dans l'ordre $x_n, \ldots, x_1$.
- Il n'est pas obligatoire de poursuive l'élimination jusqu'à la fin. Si on s'arrête à l'étape p, les variables de $x^{(p)}$ deviennent des paramètres. Les conditions $b^{(p)} \geqslant 0$ délimitent les valeurs des paramètres pour lesquelles le système est faisable. Enfin, le procédé de sélection ci-dessus donne la valeur paramétrique de la solution.
- L'algorithme peut s'exécuter sans division. La combinaison de la contrainte $j \in I_+$ et de la contrainte $k \in I_-$ se fait en multipliant la première par $-a_{k1} > 0$ et l'autre par $a_{j1} > 0$ et en additionnant.

Complexité

- On évalue d'abord une borne du nombre de contraintes à l'étape p, m_p, soit $m_p = x_0 + x_+ \times x_+$.
- Comme $x_0 + x_+ + x_- = m_{p-1}$, m_p prend sa valeur maximum pour $x_0 = 0$ et $x_+ = x_- = m_{p-1}/2$, à condition que $m_{p-1} > 4$.
- Pour le cas le pire, on a donc la récurrence $m_p = (\frac{m_{p-1}}{2})^2$ dont la solution est $m_n = (\frac{m}{2})^{2^n}$. C'est aussi une borne du travail à effectuer.
- La complexité est donc énorme sauf pour les petits systèmes. Mais la redondance est également énorme, surtout si le système est creux (a beaucoup de coefficients nuls).
- Enfin, il est possible que l'algorithme se termine prématurément.
- L'algorithme de Fourier-Motzkin est très simple à programmer, mais il doit être réservé à de petits problèmes.

Un exemple I / II

- Soit le code :

```
for(j=i+1; j<n; j++)
    for(k=i+1; k<n; k++)
        a[j][k] -= a[j][i]*a[i][k]/a[i][i];
```

- L'exécution de ces deux boucles modifie-t-elle le terme `a[i][i]` (le pivot) ?
- Réponse : le système :

$$\begin{aligned} i+1 \leqslant\ & j & < n, \\ i+1 \leqslant\ & k & < n, \\ & i & = j, \\ & i & = k, \end{aligned}$$

est il faisable ?

Un exemple II / II

$$
\begin{array}{cccc}
+ & j-i-1 & \geqslant & 0, \\
- & n-j-1 & \geqslant & 0, \\
0 & k-i-1 & \geqslant & 0, \\
0 & n-k-1 & \geqslant & 0, \\
- & i-j & \geqslant & 0, \\
0 & i-k & \geqslant & 0, \\
+ & j+k-2i & \geqslant & 0.
\end{array}
\qquad
\begin{array}{cccc}
+ & k-i-1 & \geqslant & 0, \\
- & n-k-1 & \geqslant & 0, \\
- & i-k & \geqslant & 0, \\
0 & n-i-2 & \geqslant & 0, \\
0 & -1 & \geqslant & 0.
\end{array}
$$

Un exemple II / II

$$
\begin{array}{cccc}
+ & j-i-1 & \geqslant & 0, \\
- & n-j-1 & \geqslant & 0, \\
0 & k-i-1 & \geqslant & 0, \\
0 & n-k-1 & \geqslant & 0, \\
- & i-j & \geqslant & 0, \\
0 & i-k & \geqslant & 0, \\
+ & j+k-2i & \geqslant & 0.
\end{array}
\qquad
\begin{array}{cccc}
+ & k-i-1 & \geqslant & 0, \\
- & n-k-1 & \geqslant & 0, \\
- & i-k & \geqslant & 0, \\
0 & n-i-2 & \geqslant & 0, \\
0 & -1 & \geqslant & 0.
\end{array}
$$

Bingo !

Algorithme de Fourier-Motzkin étendu

- On peut au cours de l'exécution du test, garder la trace des combinaisons effectuées. On voit alors que chaque contrainte du système d'ordre p est une combinaison linéaire à coefficients positifs d'au plus deux contraintes du système d'ordre $p-1$.
- En généralisant, toute contrainte figurant dans l'algorithme est combinaison linéaire positive de lignes de $Ax+b$. Soit $y \geqslant 0$ le vecteur des coefficients.
- Comme dans le système d'ordre n toutes les variables ont été éliminées, on en déduit $yA = 0$.

Lemme de Farkas

Théorème

$$(\exists x : Ax + b \geqslant 0) \Leftrightarrow (\forall y : y \geqslant 0, yA = 0 \Rightarrow yb \geqslant 0).$$

Démonstration.

De gauche à droite soit u tel que $Au + b \geqslant 0$. Soit un y quelconque tel que $y \geqslant 0$ et $yA = 0$. On a $y(Ax + b) \geqslant 0$. Mais

$$y(Ax + b) = yAx + yb = yb.$$

De droite à gauche, on exécute l'algorithme de Fourier-Motzkin étendu. On en tire un $y \geqslant 0$ tel que $yA = 0$. On en déduit que $yb \geqslant 0$, ce qui veut dire que le test a réussi et qu'il est possible de construire un u tel que $Au + b \geqslant 0$. □

Programmation linéaire

- On adjoint au système $Ax + b \geqslant 0$ la contrainte $z \geqslant c.x$, où z est une nouvelle variable.
- On exécute l'algorithme de Fourier-Motzkin en prenant soin d'éliminer z en dernier.
- Si l'algorithme échoue, le problème n'est pas faisable.
- Sinon, la valeur de z dans la solution donne la valeur minimum de $c.x$.
- Le reste de la solution caractérise un point ou ce minimum est atteint.

Lemme de Farkas affine

Théorème

Si le système $Ax + b \geqslant 0$ *est faisable, alors :*
$\forall x : Ax + b \geqslant 0 \Rightarrow cx + d \geqslant 0$ *est équivalent à:*

$$\exists \lambda_0, \lambda \geqslant 0 : \forall x : cx + d = \lambda_0 + \lambda(Ax + b).$$

Démonstration.

L'implication de droite à gauche est évidente. De gauche à droite, l'hypothèse revient à dire que le système $Ax + b \geqslant 0, cx + d < 0$ n'a pas de solution. D'après le lemme de Farkas, ceci implique l'existence de $y_0, \lambda \geqslant 0$ tel que $\lambda A - y_0 c = 0$ et $\lambda b - y_0 d < 0$. De plus, y_0 ne peut être nul car cela impliquerait que $Ax + b \geqslant 0$ n'a pas de solution. On peut donc prendre $y_0 = 1$. On pose $\lambda b - d = -\lambda_0, \lambda_0 > 0$ et il vient:

$$\lambda(Ax + b) - cx - d = \lambda b - d = -\lambda_0,$$

d'où la conclusion du théorème. □

Cônes

- Un cône $\mathcal{C}$ est un polyèdre convexe dont les contraintes sont de la forme $Ax \geqslant 0$.
- Propriété fondamentale : $u, v \in \mathcal{C}, \lambda, \mu \geqslant 0 \Rightarrow \lambda u + \mu v \in \mathcal{C}$.

Théorème

L'objet $\{\sum_i \lambda_i u_i \mid \lambda_i \geqslant 0\}$ *est un cône.*

Démonstration.

Il suffit de considérer le système :

$$\begin{aligned} x - \sum_i \lambda_i u_i &= 0, \\ \lambda_i &\geqslant 0. \end{aligned}$$

et d'utiliser la méthode de Fourier-Motzkin pour éliminer les λ_i. Ce qui reste est un système de contraintes linéaires en x, qui définissent bien un polyèdre. □

Réciproque

Théorème

Tout cône $\mathcal{C} = \{x \mid Ax \geqslant 0\}$ *est engendré par un système fini de rayons* $u_1, \ldots, u_p$.

Démonstration.

On considère l'objet $\mathcal{C}^* = \{yA \mid y \geqslant 0\}$. C^* est un cône dont on peut déterminer les contraintes comme ci-dessus :

$$\mathcal{C}^* = \{c \mid cB \geqslant 0\}.$$

Quelque soit $y \geqslant 0$, yA appartient à $\mathcal{C}^*$, donc $yAB \geqslant 0$. Comme on peut prendre pour y les m vecteurs unitaires, on en déduit $AB \geqslant 0$ ce qui signifie que les vecteurs colonnes de B appartiennent à $\mathcal{C}$.
Soit maintenant x un vecteur quelconque de $\mathcal{C}$. Pour tout $y \geqslant 0$ on a $yA.x = y.Ax \geqslant 0$. En d'autre termes, pour tout c tel que $cB \geqslant 0$, on a $cx \geqslant 0$. On peut donc appliquer le lemme de Farkas affine : il existe $\lambda \geqslant 0$ tel que $x = \lambda B$. $\mathcal{C}$ est donc engendré par les vecteurs colonnes de B. $\square$

Théorème de Minkovsky

Théorème

Tout polyèdre $\mathcal{P}$ peut être mis sous la forme : $\mathcal{P} = \mathcal{Q} \oplus \mathcal{C} \oplus \mathcal{H}$, où $\mathcal{Q}$ est un polytope (polyèdre borné), $\mathcal{C}$ est un cône et $\mathcal{H}$ un sous espace linéaire.

Démonstration.

Soit $\mathcal{P} = \{x \mid Ax + b \geqslant 0\}$. On considère le cône $\mathcal{D} = \{x, z \mid Ax + zb \geqslant 0\}$ où z est une nouvelle variable. Il est clair que $\mathcal{P}$ est l' intersection de $\mathcal{D}$ avec l'hyperplan $\{z = 1\}$. On construit les rayons de $\mathcal{D}$. Ceux dont la coordonnées z n'est pas nulle engendrent $\mathcal{Q}$. Les rayons dont l'opposé est également un rayon engendrent $\mathcal{H}$. Enfin, ceux qui restent engendrent $\mathcal{C}$. □

On peut écrire:

$$\mathcal{P} = \left\{ \sum \lambda.u + \sum \mu.v + \sum \nu.w \;\middle|\; \sum \lambda = 1, \lambda \geqslant 0, \mu \geqslant 0 \right\}$$

Les u sont les sommets, les v les rayons et les w les lignes.

Programmation linéaire, bis

- Le minimum de $c.x$ sous la contrainte $x \in \mathcal{P} = \mathcal{Q} + \mathcal{C} + \mathcal{H}$ est atteint en l'un des sommets de $\mathcal{Q}$ à condition que $\mathcal{H}$ soit vide et que $c \in \mathcal{C}$.
- En effet on peut écrire :

$$c.x = \sum \lambda c.u + \sum \mu c.v + \sum \nu c.w.$$

Puisque ν n'est pas contraint, on peut faire décroître $c.x$ a volonté s'il existe un w. Il en est de même s'il existe un v tel que $c.v < 0$, puisque $\mu \geqslant 0$. Si $\mathcal{H}$ est vide, le dernier terme n'existe pas, et si $c \in \mathcal{C}$, on minimise $c.x$ en prenant $\mu = 0$. Soit u_0 un sommet où $c.x$ est minimum, et u_1 un sommet où $c.u_1 > c.u_0$. Supposons que dans l'expression de x, u_1 ait un coefficient λ_1 non nul. Le point $x - \lambda_1(u_1 - u_0)$ est dans $\mathcal{P}$ et sa fonction objectif a diminué. On en déduit que la solution d'un programme linéaire est l'un quelconque des sommets de $\mathcal{Q}$ où $c.x$ atteint son minimum.

Critique

- La méthode ci-dessus est inefficace car il faut utiliser Fourier-Motzkin pour trouver la décomposition de $\mathcal{P}$, et aussi parce que le nombre de sommets peut être très grand (de l'ordre de C_m^n, le coefficient du binôme).
- Il existe un algorithme plus efficaces que Fourier-Motzkin pour décomposer un polyèdre, l'algorithme de Chernikova, mais le nombre de sommets ne change pas.

Dualité I / II

Théorème

Si les deux ensembles $\{x \mid Ax \leqslant b\}$ *et* $\{y \mid y \geqslant 0, yA = c\}$ *sont non vides, on a :*

$$\ell = \max\{cx \mid Ax \leqslant b\} = \min\{yb \mid y \geqslant 0, yA = c\} = r.$$

Soit par exemple x^* (resp. y^*) un point de l'ensemble de gauche (resp. de droite). On a :

$$c.x^* = y^*Ax^* \leqslant y^*.b.$$

Il en résulte que ℓ et r existent et que $\ell \leqslant r$.

Dualité II / II

On peut supposer que x^* (resp. y^*) est le point où le maximum (resp. le minimum) est atteint. En tout point x tel que $b - Ax \geqslant 0$ on sait que $c.x^* - c.x \geqslant 0$, on peut donc appliquer le lemme de Farkas affine :

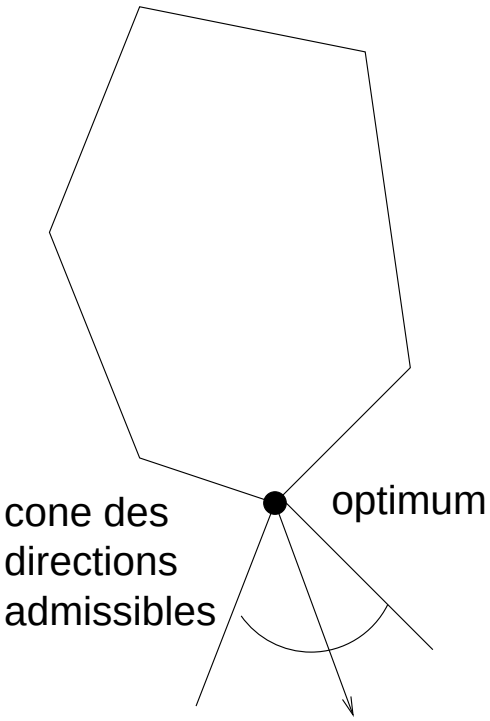
$$\exists \lambda_0, \lambda \geqslant 0 : \forall x : c.x^* - c.x = \lambda_0 + \lambda(b - Ax).$$

On en déduit $c.x^* = \lambda_0 + \lambda b$ et $c = \lambda A$. Il en résulte que λ fait partie de l'ensemble de droite:

$$r \leqslant \lambda b = c.x^* - \lambda_0 \leqslant \ell.$$

On en déduit $\ell = r$.

Analyse de sensibilité



- Variation de
  $\omega(b) = \max\{cx \mid Ax \leqslant b\}$ avec b ?
- Par dualité,
  $\omega(b) = \min\{yb \mid y \geqslant 0, yA = c\}$. Or ce polyèdre ne dépend pas de b.
  Interprétation géométrique : aussi longtemps que b reste dans le cône des directions admissibles, l'optimum y^* ne change pas. Donc:

$$\omega(b) = y^*.b.$$

- Pour en savoir plus, il faut faire de la programmation linéaire paramétrique.

Complémentarité

Théorème

$$\forall j : y_j^*.(b_j - A_{\bullet j}.x^*) = 0.$$

Démonstration.

$$y^*.(b - A.x^*) = y^*.b - y^*.A.x^* =$$

$$= c.x^* - y^*.A.x^* = (c - y^*A).x^* = 0.$$

Mais chaque terme du produit scalaire $y^*.(b - A.x^*)$ est positif, donc si la somme est nulle chaque terme est nul. □

Dualité généralisée

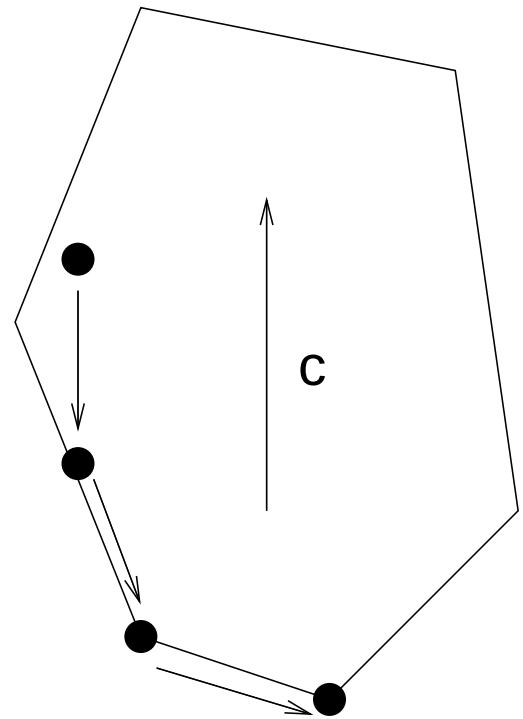
Il existe une très grande variété de théorèmes de dualité, suivant la nature des contraintes et le signe des variables. En première approximation, on peut utiliser le tableau suivant :

Primal	Dual
objectif (Min)	second membre
second membre	objectif (Max)
A	A^T
Contrainte i: $\geqslant$	variable $u_i \geqslant 0$
Contrainte i: $=$	variable non contrainte en signe
Variable $x_j \geqslant 0$	contrainte j : $\leqslant$
Variable x_j non contrainte en signe	contrainte j : $=$

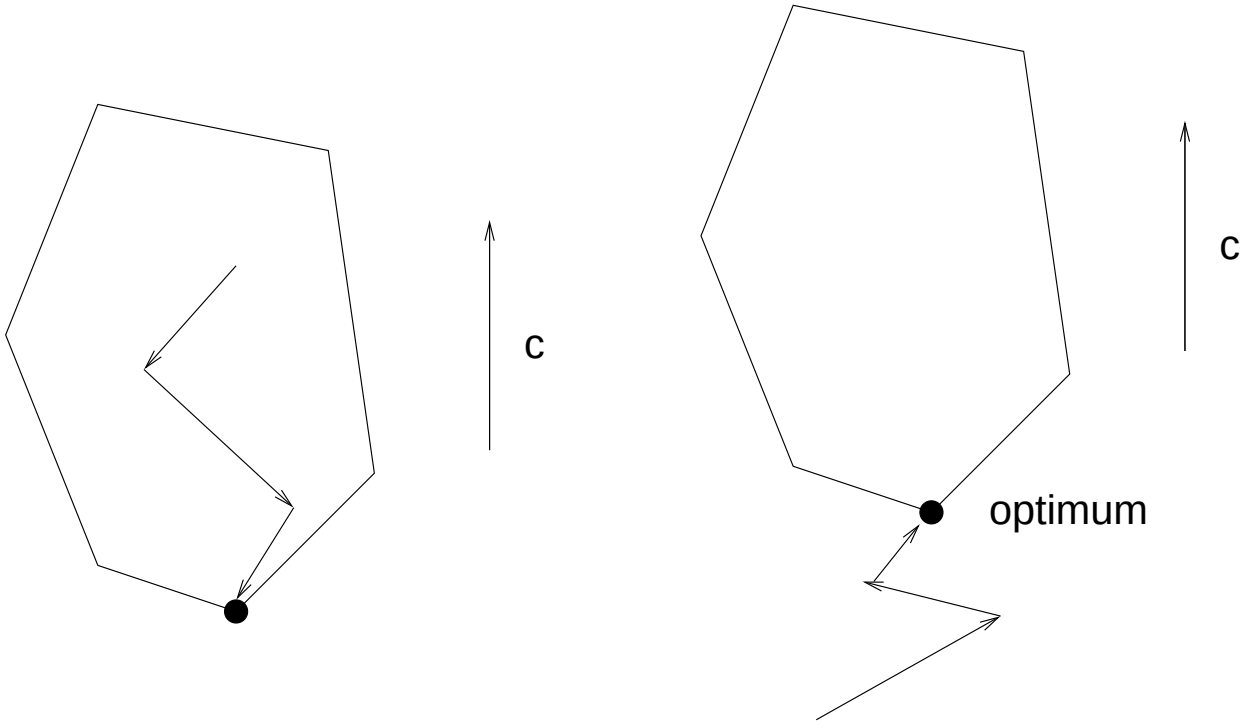
On trouvera dans Schrijver une formulation plus précise.

Algorithme du Simplexe

- «Trouver le point le plus bas d'un vase». Fourier, 1828.
- Formalisation par Danzig, 1950.



Méthodes externes, méthodes internes



- Méthodes internes : il faut connaître un point faisable. On peut arrêter la recherche avant l'optimum.
- Méthodes externes : il n'y a pas besoin de connaître un point faisable.

Ordre lexicographique

- Définition :

$$x \ll y = \exists k : x_{1,\cdots,k-1} = y_{1,\cdots,k-1}, x_k < y_k.$$

- $\ll$ est un ordre total.
- L'ordre par composantes n'est pas total.
- $c.x < c.y$ n'est pas un ordre.
- D'où l'intérêt de remplacer $\min c.x$ par $\min_{\ll}$.
- On peut toujours ajouter une nouvelle variable u et la contrainte $u \geqslant c.x$ à condition que u soit la première inconnue.
- Cette technique évite les problèmes bien connus de *dégénérescence*.

Algorithme du simplexe externe ou dual

Résoudre :

$$\begin{array}{rcl} \min\limits_{\ll} & & x \\ x & \geqslant & 0 \\ Ax + b & \geqslant & 0 \end{array}$$

Généralisation :

$$\begin{array}{rcl} \min\limits_{\ll} & & x \\ y = Sz + t & \geqslant & 0 \\ z & \geqslant & 0 \end{array}$$

Tailles : $x : n$, $A : m \times n$, $b : m$.
Au commencement,

$$z = x,\ S = \begin{bmatrix} I \\ A \end{bmatrix},\ t = \begin{pmatrix} 0 \\ b \end{pmatrix}.$$

Invariants

- Les vecteurs colonnes de S sont lexicopositifs au démarrage et le restent tout au long de l'algorithme.
- z est un sous-ensemble de x, y. La condition $z \geqslant 0$ est donc toujours vérifiée.
- D'une étape à l'autre, t croît dans l'ordre lexicographique.
- Ces invariants sont vérifiés au début de l'algorithme.

Cas de base

- Si $t \geqslant 0$, on a trouvé la solution. Il suffit de faire $z = 0$, ce qui satisfait les contraintes. On a $x = t_{1,\cdots,n}$.
- De plus, c'est le minimum lexicographique : toute autre valeur positive de z ajoute à x un vecteur lexicopositif.
- Soit $t_i < 0$. Si $\forall j : S_{ij} \leqslant 0$, le problème n'a pas de solution.

Changement de variable

- Soit $t_i < 0$ et $S_{ij} > 0$ (le pivot). On élimine z_j en faveur de y_i :

$$z_j = y_i/S_{ij} - \sum_{\ell \neq j} S_{i\ell}/S_{ij} z_\ell - t_i/S_{i\ell}.$$

$$y_k = \sum_{\ell \neq j} (S_{k\ell} - S_{kj} S_{i\ell}/S_{ij}) z_\ell + S_{kj}/S_{ij} z_j + t_k - S_{kj} t_i/S_{ij}.$$

- Remarquer que $-t_i/S_{ij}$ est positif, et que S_{kj} est lexicopositif. Donc, t croît selon l'ordre lexicographique.
- Comme $S_{ij} > 0$, le vecteur colonne j reste lexicopositif.
- Il reste à garantir que le vecteur colonne ℓ reste lexicopositif.

Choix du pivot

- Si $S_{i\ell}$ est négatif, il n'y a pas de problème.
- Sinon le nouveau vecteur colonne est égal, à un coefficient positif près, à :

$$S_{\bullet\ell}/S_{i\ell} - S_{\bullet j}/S_{ij}.$$

- Il faut donc choisir j pour que $S_{ij} > 0$ et que le «vecteur réduit» $S_{\bullet j}/S_{ij}$ soit le plus petit possible.
- Un tel choix est toujours possible sauf si on est dans le cas d'un système infaisable.

Convergence

- Observer que l'état de l'algorithme est entièrement déterminé quand on sait quelles sont les composantes de y qui sont dans z (les variables en base).
- Or y est de taille $m+n$ et z de taille n, il n'y a donc que C_{m+n}^{n} combinaisons possibles.
- L'algorithme ne peut pas boucler, car t croît dans l'ordre lexicographique.
- Comme C_{m+n}^{n} n'est pas un polynôme en n et m, l'algorithme n'est pas polynomial.
- On peut construire des cas pathologiques qui demandent un temps exponentiel.
- Mais en pratique (et en probabilité) le nombre d'opérations est en $O(n^2(n+m))$.

Questions numériques I / II

- Du point de vue numérique, l'algorithme du Simplexe est analogue à la méthode de Gauss, avec une règle particulière pour le choix du pivot.
- Si l'on connaît la matrice des inconnues de base, l'algorithme ne fait qu'inverser celle-ci, tout en appliquant les mêmes transformations aux inconnues hors base.
- Les résultats sont donc donnés par des formules de Cramer.
- On peut faire les calculs en virgule flottante. Il y a alors accumulation d'erreurs d'arrondi, qui peuvent faire que la solution finale n'est pas faisable (en particulier pour les contraintes saturées).
- Il faut alors développer des méthodes de correction. En général la solution est faisable, mais l'optimalité n'est pas garantie.

Questions numériques II / II

- On peut rendre la matrice des contraintes entières, et essayer de mener les calculs exactement (algorithmes «tout entiers»).
- Les nombres à manipuler sont des déterminants de Cramer. On peut donc les borner à l'aide de l'inégalité de Hadamard:

$$|det(A)| \leqslant |A_1| \dots |A_n|,$$

où les A_i sont les vecteurs colonnes (ou les vecteurs lignes) de A.

- Il en résulte que la taille des nombres à manipuler est bornée par n fois la taille du plus grand élément de A. Cette borne est rarement atteinte.
- Il faut utiliser des arithmétiques en précision arbitraire, telle la librairie GMP.

Algorithme primal

- On prend le problème sous la forme équivalente suivante :

$$\begin{aligned} \min f(x) &= c.x \\ Ax &= b \\ x &\geqslant 0 \end{aligned}$$

- On peut supposer que les lignes de la matrice des contraintes A sont linéairement indépendantes : on peut éliminer les lignes redondantes.
- A est de dimension $m \times n$ avec nécessairement $m < n$.

  Dans le cas contraire, le système $Ax = b$ aurait au plus une solution et le problème serait trivial.

Base

- Une «base» est une matrice carrée $n \times n$ extraite de A inversible. On partitionne A en deux blocs B, la base, et N le reste de la matrice. On partitionne de même x en x^B, x^N et c en c^B, c^N.
- La solution associée à une base B est le vecteur $(B^{-1}b, 0)^T$. Il satisfait évidemment à la contrainte $Ax = b$.
- La base B est réalisable si et seulement si la solution correspondante satisfait également à la contrainte $x \geqslant 0$, c'est-à-dire si $\overline{x} = B^{-1}b \geqslant 0$.
- A une base réalisable correspond une valeur de l'objectif, $c^B.B^{-1}b$.
- Est-il possible d'améliorer cet objectif en faisant varier x^N ?

Recherche de l'optimum I / III

- Si x^N n'est plus nul, on a :

$$\begin{aligned} x^B &= B^{-1}(b - Nx^N) \\ f(x) &= c^B.B^{-1}b + (c^N - c^B.B^{-1}N)x^N \end{aligned}$$

  Le vecteur $\overline{c} = c^N - c^B.B^{-1}N$ est le vecteur des coûts réduits.
- Si x_i fait partie de x^N, comme il est nul on ne peut que l'augmenter pour que la solution reste faisable. Ceci fait décroître $f(x)$ à condition que $\overline{c}_i < 0$.
- Si tous les coûts réduits sont positifs, on a trouvé l'optimum.
- Sinon, on choisit un x_i dont le coût réduit est négatif (par exemple celui dont le coût réduit est minimum).

Recherche de l'optimum II / III

- Puisque on fait croître x_i et que les autres composantes de x^N restent nulles, la seule contrainte sur la valeur de x_i est $B^{-1}(b - Nx^N) \geqslant 0$. Il y a deux cas possibles :
- Toutes les composantes de la colonne i de $\overline{B} = B^{-1}N$ sont négatives. Alors x_i n'est pas borné, et le minimum est $-\infty$.
- Sinon, à chaque composante $\overline{B}_{ik} > 0$ correspond la borne $x_i \leqslant \overline{x}_k/\overline{B}_{ik}$. Soit j l'indice de la plus petite de ces bornes.
- Le point correspondant à $x^k = 0$ sauf pour $k = j$: $x_j = \overline{x}_j/\overline{B}_{ij}$ est faisable et la valeur de $f(x)$ est inférieure à celle du point de départ.

Recherche de l'optimum III / III

- La base qui correspond au nouveau point courant s'obtient en remplaçant dans b le colonne i par la colonne j.
- On poursuit l'algorithme en inversant la nouvelle base et en calculant la nouvelle solution et les nouveaux coûts réduits.
- L'algorithme se termine si à chaque pas la valeur de l'objectif décroît strictement.

  En effet il n'y a que C_n^m bases possibles et la condition de décroissance stricte empêche tout bouclage.

- Cependant l'algorithme peut boucler en cas de dégénérescence (il semble que ce soit très rare).
- Comme pour l'algorithme dual, on peut mener les calculs de façon incrémentale (il suffit d'un seul pivot de Gauss pour inverser la nouvelle base).

Recherche du point faisable initial

- Il s'agit de trouver un point dans le polyèdre $P = \{x \mid x \geqslant 0, Ax \geqslant b\}$. Soit $\mathbf{1}$ le vecteur dont tous les éléments sont égaux à 1, et soit y un nouveau vecteur de même taille que x. On considère le problème :

$$\begin{array}{rcl} & \min & \mathbf{1}.y \\ x & \geqslant & 0 \\ y & \geqslant & 0 \\ Ax + y & \geqslant & b \end{array}$$

- Il est facile de voir que le point $x = 0, y = \max(b, 0)$ est faisable. On peut donc appliquer l'algorithme précédent.
- Si $y^* = 0$, il est facile de voir que le point $x^* \in P$.
- Réciproquement, si $x^* \in P$, alors $(0, x^*)$ est faisable pour le problème augmenté, donc le minimum est nul. Si inversement le minimum n'est pas nul, P est vide.