Partie 2

Techniques de modélisation

Problèmes de mélange

Les problèmes de mélange (*blending problems*) sont fréquents dans les secteurs suivants :

- raffinage de pétrole
- aliments pour bétail
- industrie pharmaceutique
- fabrication de peintures
- fabrication d'engrais (exercice vu en TD)
- fabrication d'aciers et d'alliages métalliques, etc.

On dispose de ni ingrédients (indice i) contenant ne éléments intéressants (indice e). On connaît la teneur T_{ie} de l'ingrédient i en élément e, en %. L'objectif est de faire un mélange en quantité Q avec des contraintes de qualité : la teneur pour chaque élément e doit être entre L_e et U_e (en %).

Problèmes de mélange

Par exemple, à partir de ni métaux recyclés contenant chacun un certain pourcentage de chrome, on veut faire un alliage contenant entre 2 et 3% de chrome.

Soit x_i les quantités d'ingrédient i (variables) dans le mélange. Les contraintes de qualité pour un élément e sont :

$$L_e \leq \frac{\sum_{i=1}^{ni} T_{ie} \cdot x_i}{Q} \leq U_e$$

Souvent, des ingrédients ont seulement une limite inférieure ou bien une limite supérieure. Le dernier cas correspond à des impuretés indésirables.

Contraintes de flot

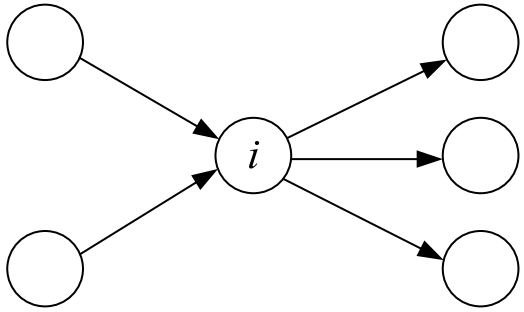
Elles expriment la conservation du flot qui traverse un nœud dans un réseau. Par exemple, le nombre de voitures arrivant à un carrefour est égal au nombre de voitures en repartent.

Le réseau peut être constitué de lignes électriques, de pipe-lines, de convoyeurs dans une usine, des routes, etc.

Les flots peuvent être des courants électriques, de l'eau, du gaz, du pétrole, des matières premières, des composants, des palettes, des véhicules etc.

Pour les réseaux électriques, c'est la fameuse "loi de Kirchhoff" ou "loi des nœuds".

Contraintes de flot



Soit x_{ij} le flot (ou "flux") d'un nœud i vers un nœud j. On a :

$$\sum_{j \text{ prédécesseur de } i} x_{ji} = \sum_{j \text{ successeur de } i} x_{ij}$$

Contraintes de flot

Un nœud i en début de réseau n'a pas de prédécesseurs mais il a souvent un débit limité a_i, par exemple une usine avec un stock de produit. La contrainte de flot se simplifie :

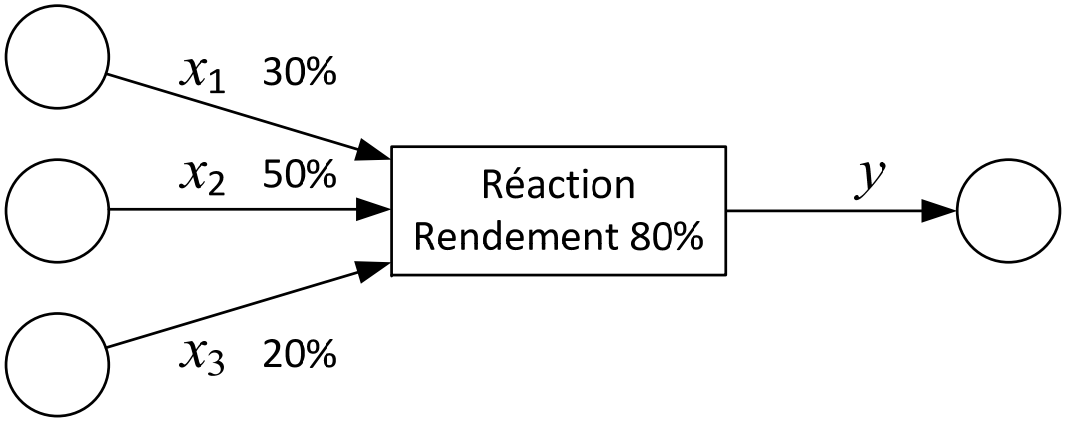
$$\sum_{j \text{ successeur de } i} x_{ij} \leq a_i$$

Un nœud i en fin de réseau, par exemple un client, n'a pas de successeurs mais il a souvent une demande ou besoin b_i qui doit être satisfait. Là aussi, la contrainte de flot se simplifie :

$$\sum_{j \text{ prédécesseur de } i} x_{ji} = b_i$$

Contraintes de flot

En chimie, on combine des produits dans des pourcentages donnés pour faire un produit fini, avec un certain rendement. Exemple, 3 produits en entrée (quantités x_1, x_2, x_3, 30, 50 et 20%) pour faire un produit fini (quantité y, rendement 80%).



$$y = 0.8\ (x_1 + x_2 + x_3)$$

$$x_1 / (x_1 + x_2 + x_3) = 0.3$$
$$x_2 / (x_1 + x_2 + x_3) = 0.5$$
$$x_3 / (x_1 + x_2 + x_3) = 0.2$$

Les solveurs refusant les quotients de variables, il faut écrire $x_1 = 0.3\ (x_1 + x_2 + x_3)$ au lieu de $x_1 / (x_1 + x_2 + x_3) = 0.3$

Conservation des stocks

C'est une sorte de contrainte de flot pour un nœud i muni d'un stock, sur un horizon de temps découpé en périodes. Soit S_t le stock en fin de période t, x_t la quantité qui entre en stock en période t et y_t la quantité qui sort. On a :

$$S_t = S_{t-1} + x_t - y_t$$

On l'a vue en TD pour les problèmes des verres et du barrage. Pour $t = 1$, la variable S_{t-1} n'existe pas. Il faut écrire un cas particulier, avec un stock initial SI inclus dans les données :

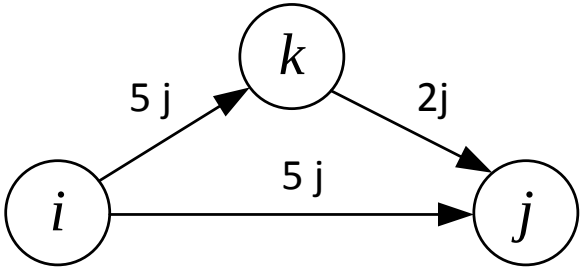
$$S_1 = SI + x_1 - y_1$$

Ordre de tâches

En gestion de projet ou en ordonnancement de production, on a n tâches de durée d_i et des contraintes d'ordre : un arc (i,j) signifie que la tâche i doit être finie avant de commencer la tâche j. Soit t_i la date de début de la tâche i, on a :

$$t_i + d_i \leq t_j$$

Cette contrainte est appelée "contrainte de potentiel". Il faut l'écrire pour chaque arc (i,j) du graphe de projet. Ce n'est pas une égalité car j peut être retardé par une autre tâche k :



Contraintes de sac à dos

Soit n objets de poids a_i et de valeur c_i et un sac de capacité b qui ne peut pas contenir tous les objets. Le problème du sac à dos en 0-1 consiste à choisir des objets, de poids total inférieur ou égal à B, et de valeur totale maximale.

On obtient un PL en 0-1 très simple à une seule contrainte, avec des variables binaires x_i valant 1 si on prend l'objet i.

$$(1)\ \text{Max} \sum_{i=1}^{n} c_i x_i$$

$$(2)\ \sum_{i=1}^{n} a_i x_i \leq b$$

$$(3)\ \forall\, i = 1...n,\ x_i \in \{0,1\}$$

Contraintes de sac à dos

Les contraintes (2) sont appelées "contraintes de sac à dos" (knapsack constraints). Elles sont très utilisées en PLNE pour exprimer des contraintes de capacité ou de chargement.

Les "objets" peuvent être des caisses à mettre dans un camion, des livres à ranger sur des étagères, des fichiers à stocker sur un disque, des tâches à placer sur une machine.

Le "poids" est respectivement le poids des caisses, l'épaisseur des livres, la taille des fichiers en octets, la durée des tâches.

La "capacité du sac" est la charge utile du camion, la largeur de l'étagère, la capacité du disque, le temps de travail maximum de la machine.

Contraintes de sac à dos

Le problème du sac à dos en 0-1 est difficile car on doit prendre un objet complètement ou bien on ne le prend pas : l'objet ne peut pas être coupé. Exemple :

- une bobine de 6970 m de câble,
- *n* commandes de clients : 2463, 2597, 315, 1089, 759, 599, 1283, 1625, 2292 et 211 m
- comment maximiser la longueur de câble livrée aux clients?

Solution : 6970 = 2597+315+759+1283+1625+211.

Ce cas particulier de problème de sac à dos avec des valeurs égales aux poids est appelé problème d'empilement.

Comment résoudre le problème si on peut couper les objets?

Contraintes d'affectation

Ou "assignment constraints". Soit *n* objets à affecter à *n* places. Chaque objet *i* doit être affecté à une place *j* et chaque place *j* doit recevoir un objet *i*. Par exemple, affecter des personnes à des postes de travail.

On définit des variables binaires x_{ij}, égales à 1 si l'objet *i* est affecté à la place *j*. On a alors 2 types de contraintes : chaque objet est affecté à une place et chaque place reçoit un objet.

$$\forall i : \sum_{j=1}^{n} x_{ij} = 1 \qquad \text{et} \qquad \forall j : \sum_{i=1}^{n} x_{ij} = 1$$

Souvent, on a besoin d'un seul des deux types, par exemple des étudiants sont placés dans des TD de 24 places.

Expressions logiques

Les variables binaires sont utiles pour traduire des expressions logiques. On associe à chaque proposition une variable binaire valant 1 si et seulement si la proposition est vraie. Exemples avec deux variables p et q pour deux propositions P et Q.

Expression logique vraie	**Contrainte équivalente**
non P	$p = 0$ (ou $1 - p = 1$)
P ou Q (ou inclusif)	$p + q \geq 1$
P ou bien Q (ou exclusif)	$p + q = 1$
P et Q	$p = 1$ et $q = 1$ (ou $p + q = 2$)
$P \Rightarrow Q$ (P implique Q)	$p \leq q$
$P \Leftrightarrow Q$ (P équivalent à Q)	$p = q$

Variables discrètes

Soit une variable x qui ne peut prendre qu'un ensemble fini V de k valeurs $\{v_1, v_2, \ldots, v_k\}$. On définit k variables binaires y_i, $i = 1 \ldots k$ et on effectue le changement de variable :

$$x = \sum_{i=1,k} v_i . y_i$$

Enfin, on ajoute la contrainte :

$$\sum_{i=1,k} y_i = 1$$

Ainsi, la première somme (x) prend sa valeur dans V.

Petites variables entières

Une technique similaire est utilisable pour des petits entiers, en exploitant leur codage en binaire.

En effet, un entier x inférieur à 2^p peut s'écrire avec p chiffres binaires (bits) significatifs y_i :

$$x = \sum_{i=0,p-1} 2^i . y_i$$

Par exemple, 13 s'écrit 1101 en binaire : 1x8 + 1x4 + 1x1.

On peut donc remplacer x par les p variables y_i.

Petites variables entières

Par exemple, si on a un PLNE avec des variables entières inférieures à $16 = 2^4$, on peut remplacer chaque variable entière par 4 variables binaires.

Malgré l'augmentation du nombre de variables, cette transformation est souvent profitable car les PL en 0-1 sont plus faciles que les PL à variables entières quelconques.

Comparaison avec les variables discrètes (slide 62) pour une variable entière comprise entre 0 et $v = 2^p - 1$:

- la technique de la slide 62 nécessite v variables binaires
- celle de la slide 63 en nécessite seulement $\lfloor \log_2 v \rfloor + 1$.

Les contraintes à grand M

Elles servent quand on veut mettre en relation une variable réelle x, pouvant dépasser 1, et une variable binaire y.

1er cas – Traduction de "si y (binaire) = 0 alors x (réel) = 0".
Noter que ceci équivaut à "si $x \neq 0$ alors $y \neq 0$".

La difficulté est que x doit être libre (non contrainte) si $y \neq 0$.

Considérons la contrainte "$x \leq y$". Si $y = 0$, on a bien $x = 0$.
Mais si $y \neq 0$, on a $y = 1$ et donc $x \leq 1$: x n'est pas libre.

Il faut écrire $x \leq M.y$ où M est une grande constante positive.

Les grands nombres comme M créent des difficultés numériques dans les solveurs (précision dégradée).

Les contraintes à grand M

La règle est d'utiliser toujours la plus petite valeur numérique qui convient, donc ici une borne supérieure pour x. Ainsi, si x est une quantité dans un stock de capacité Q, $M = Q$ suffit.

2ème cas – Traduction de "si x (réel) = 0 alors y (binaire) = 0". La difficulté est que y doit être libre si $x \neq 0$.

Ecrire "$y \leq x$" convient si x est entier. Si x est réel, et non nul, y n'est pas libre : elle est forcée à 0 si $0 < x < 1$.

On peut écrire "$y \leq M.x$". Pour limiter M, soit ε le plus petit réel considéré comme non nul. On a "si $x = 0$ alors $y = 0$" en prenant $M = 1/\varepsilon$. Si par exemple $\varepsilon = 10^{-6}$, alors $M = 10^{6}$.

Equivalence entre réelles et binaires

Autre exemple de contrainte à grand M. On peut ouvrir des magasins dans trois villes. Des variables binaires x_1, x_2, x_3 sont égales à 1 si un magasin est créé dans la ville correspondante.

On veut mettre une variable binaire y à 1 si et seulement si au moins une des variables x_j est non nulle. On peut écrire :

(1) $x_1 + x_2 + x_3 \leq M.y$
(2) $y \leq x_1 + x_2 + x_3$

Si des x_j sont non nuls, (1) force y à 1 et (2) est désactivée.
Si tous les x_j sont nuls, (2) force y à 0 et (1) est vérifiée.

Quelle valeur minimale de M convient ici?

Equivalence entre réelles et binaires

Encore un exemple de contrainte à grand M. Une usine peut faire ou pas un produit *j*. S'il est fait, il faut au moins une quantité Q_j pour être rentable. Donc, $x_j = 0$ ou bien $x_j \geq Q_j$.

On introduit une variable binaire y_j égale à 1 si et seulement si le produit est fabriqué, et on écrit :

$$(1)\quad x_j \geq Q_j . y_j$$

$$(2)\quad x_j \leq M . y_j$$

$$(3)\quad y_j \in \{0,1\}$$

Si $y_j = 0$, $x_j = 0$ d'après (2) et (1) est trivialement vérifiée.
Si $y_j = 1$, $x_j \geq Q_j$ d'après (1) et (2) est trivialement vérifiée.

Se faire aider par l'objectif

Dans une équivalence entre variables réelles et binaires, un des sens est souvent assuré par l'optimisation.

Soit n usines à créer et m clients.

L'usine i peut être créée avec une capacité Q_i et un coût fixe f_i.

Chaque client j a une demande d_j en tonnes qui peut venir des usines ouvertes, avec un coût c_{ij} par tonne.

Quelles usines ouvrir pour satisfaire les demandes en minimisant le coût total?

Se faire aider par l'objectif

On définit des variables binaires $y_i = 1$ si l'usine *i* est ouverte, et des variables réelles $x_{ij} \geq 0$ pour les quantités livrées par chaque usine *i* à chaque client *j*.

L'objectif (1) inclut les coûts d'ouverture et ceux de transport.

$$(1) \quad \min \sum_{i=1}^{m} f_i \cdot y_i + \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} \cdot x_{ij}$$

Les contraintes (2) concernent la satisfaction des demandes :

$$(2) \quad \forall j = 1 \ldots n : \sum_{i=1}^{m} x_{ij} \geq d_j$$

La difficulté est qu'une usine *i* peut livrer des clients ($x_{ij} \neq 0$) si et seulement si elle est ouverte ($y_i = 1$).

Se faire aider par l'objectif

On peut écrire :

$$(3)\ \forall i = 1 \ldots m: \sum_{j=1}^{n} x_{ij} \leq Q_i \cdot y_i$$

Ces contraintes servent d'abord de contraintes de capacité. Elles mettent aussi y_i à 1 si des produits partent de l'usine.

La réciproque n'est pas vraie : l'usine peut être ouverte et ne rien livrer. En fait, ce cas est impossible à l'optimum : le solveur va mettre $y_i = 0$ pour éviter le coût fixe f_i.

Cette technique (une contrainte pour un sens de l'équivalence, l'autre sens étant assuré par l'optimisation) est très utile.

Contraintes disjonctives

En ordonnancement, elles interdisent l'exécution de deux tâches en même temps. Soit 2 tâches i et j de durées p_i et p_j, à traiter sur une machine à des temps t_i et t_j à trouver. On évite le chevauchement si $t_i + p_i \leq t_j$ ou bien $t_j + p_j \leq t_i$.

Soit une variable binaire $y_{ij} = 1$ si et seulement si i est avant j, et les deux contraintes suivantes :

$$(1)\ \ t_i + p_i \leq t_j + M.(1 - y_{ij})$$

$$(2)\ t_j + p_j \leq t_i + M.y_{ij}$$

Si $y_{ij} = 1$, (1) signifie que i est avant j et (2) est désactivée. Sinon, (2) signifie que j est avant i et (1) est désactivée.

Contraintes disjonctives

(1) est aussi utilisée dans les problèmes de tournées. On donne n villes à visiter et des durées de déplacement d_{ij}.

On utilise des variables t_i pour les heures d'arrivée aux villes. Et des variables binaires $x_{ij} = 1$ si on visite la ville j après la ville i (sans ville intermédiaire).

$$\forall i = 1 \ldots n, \forall j = 1 \ldots n, i \neq j : t_i + d_{ij} \leq t_j + M.(1 - x_{ij})$$

Si $x_{ij} = 1$, on retrouve les contraintes de potentiel (slide 56).

Si on visite 1, 5, 3 etc. la contrainte ne concerne pas 1 et 3 : comme $x_{13} = 0$, la contrainte est désactivée pour $(i,j) = (1,3)$.

Respect partiel des contraintes

Parfois, on ne peut pas satisfaire toutes les contraintes d'un PLNE, par exemple des préférences pour un emploi du temps.

On peut alors chercher une solution qui respecte au moins *k* contraintes sur les *m* contraintes du PLNE :

$$\forall i = 1 \ldots m : \sum_{j=1,n} a_{ij} . x_j \leq b_i$$

$$\begin{cases} \forall i = 1 \ldots m : \sum_{j=1,n} a_{ij} . x_j \leq b_i + M . y_i \\ \sum_{i=1,m} y_i = m - k \\ y \in \{0,1\}^m \end{cases}$$

La variable binaire $y_i = 1$ signifie que la contrainte n° *i* peut être violée. On peut minimiser la somme des y_i.

Respect partiel des contraintes

Une autre technique est de minimiser la somme des violations de contraintes (solution "presque réalisable"). Dans ce cas, y_i devient une variable (pénalité) qui mesure la violation :

$$\forall i = 1 \ldots m : \sum_{j=1,n} a_{ij}.x_j \leq b_i + y_i$$

$$\forall i = 1 \ldots m : y_i \geq 0$$

Si certaines contraintes sont plus importantes que d'autres, on peut multiplier les pénalités par des poids différents.

On peut aussi minimiser la somme des violations relatives, c'est-à-dire la somme des $(b_i + y_i)/b_i$.

Récupération de positions

Supposons qu'on affecte *n* robots aux *n* postes de travail d'une ligne d'assemblage. Il faut des variables binaires x_{ij} (slide 60):

$$\forall i : \sum_{j=1}^{n} x_{ij} = 1 \qquad \text{et} \qquad \forall j : \sum_{i=1}^{n} x_{ij} = 1$$

Comment faire si le robot *i* doit être avant *j* sur la ligne?
On écrit que le n° de poste de *i* est inférieur à celui de *j* :

$$\sum_{k=1}^{n} k \cdot x_{ik} \leq \sum_{k=1}^{n} k \cdot x_{jk} + 1$$

Cette astuce fonctionne car une seule variable binaire vaut 1 dans chaque somme, grâce aux contraintes d'affectation.

Optimisation "bottleneck"

Soit *n* diamants à tailler sur *n* machines indépendantes. Le diamant *i* dégage une chaleur c_{ij} s'il est taillé sur la machine *j*.

Pour limiter le risque de casse, on peut minimiser la chaleur totale dégagée, en résolvant un problème d'affectation :

$$(1)\ \min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij}$$
$$(2)\ \forall\, i = 1 \ldots n : \sum_{j=1}^{n} x_{ij} = 1$$
$$(3)\ \forall\, j = 1 \ldots n : \sum_{i=1}^{n} x_{ij} = 1$$
$$(4)\ \forall\, i = 1 \ldots n, \forall\, j = 1 \ldots n : x_{ij} \in \{0,1\}$$

Mais cela n'empêche pas un diamant en particulier d'atteindre une trop haute température. Comment minimiser ce pire cas?

Optimisation "bottleneck"

Il faut trouver une affectation telle que le pire dégagement de chaleur soit minimal, c'est-à-dire :

$$\min \max \{c_{ij} | x_{ij} = 1\}$$

On conserve les contraintes d'affectation (2) et (3) et leurs variables binaires (4).

On ajoute une variable $y \geq 0$ pour majorer le dégagement de chaleur via les contraintes (5) :

$$(5) \quad \forall\, i = 1 \ldots n, \forall\, j = 1 \ldots n : c_{ij} \cdot x_{ij} \leq y$$

Il reste à minimiser y, avec la fonction-objectif "$\min y$".
A l'optimum, y sera exactement égal à $\min \max \{c_{ij} | x_{ij} = 1\}$.

Optimisation "bottleneck"

Dans beaucoup de programmes mathématiques, l'objectif est une somme de *n* fonctions $f_j(\mathbb{x})$. En minimisation, on a donc :

$\min \ \sum_{j=1}^{n} f_j(\mathbb{x})$

Dans le cas des diamants, il devient : $\min \ \max_{j=1,n} \{f_j(\mathbb{x})\}$.

Ce PM où on minimise un maximum est dit "min-max".
Parfois, on maximise un minimum ("problème max-min").
Les 2 cas sont appelés "problèmes d'optimisation bottleneck".

Pour les PM min-max, la technique est toujours la même :

- majorer les $f_j(\mathbb{x})$ à l'aide d'une variable $y \geq 0$
- minimiser y.

Optimisation multi-objectif

Supposons que pour un vecteur de variables $\mathbb{x}$, on ait deux critères $f(\mathbb{x})$ et $g(\mathbb{x})$ à minimiser, par exemple, un niveau de stock et le temps moyen pour traiter une commande-client.

On peut minimiser une somme pondérée : $\alpha \cdot f(\mathbb{x}) + \beta \cdot g(\mathbb{x})$.

Cette idée est OK pour minimiser une somme de coûts (cas des usines slide 70). Mais notre exemple avec le niveau de stock et le temps de service montre deux problèmes :

- on ajoute des quantités non comparables
- les critères sont antagonistes : si on réduit le niveau de stock on augmente le temps de service, et vice versa.

Optimisation multi-objectif

Un cas sans problème est l'optimisation hiérarchique :

- on souhaite minimiser $f(\mathbb{x})$ en priorité
- on départage les optima en minimisant $g(\mathbb{x})$.

Il suffit alors de minimiser $z = M \cdot f(\mathbb{x}) + g(\mathbb{x})$.

M est une constante positive telle que $M \cdot f(\mathbb{x}) \geq g(\mathbb{x})$ sur le domaine des solutions réalisables.

A cause du poids M, un solveur va d'abord minimiser $f(\mathbb{x})$. Ensuite, il va tenter de réduire encore z en agissant sur $g(\mathbb{x})$.

Optimisation multi-objectif

Dans le cas de critères antagonistes, comment comparer des solutions?

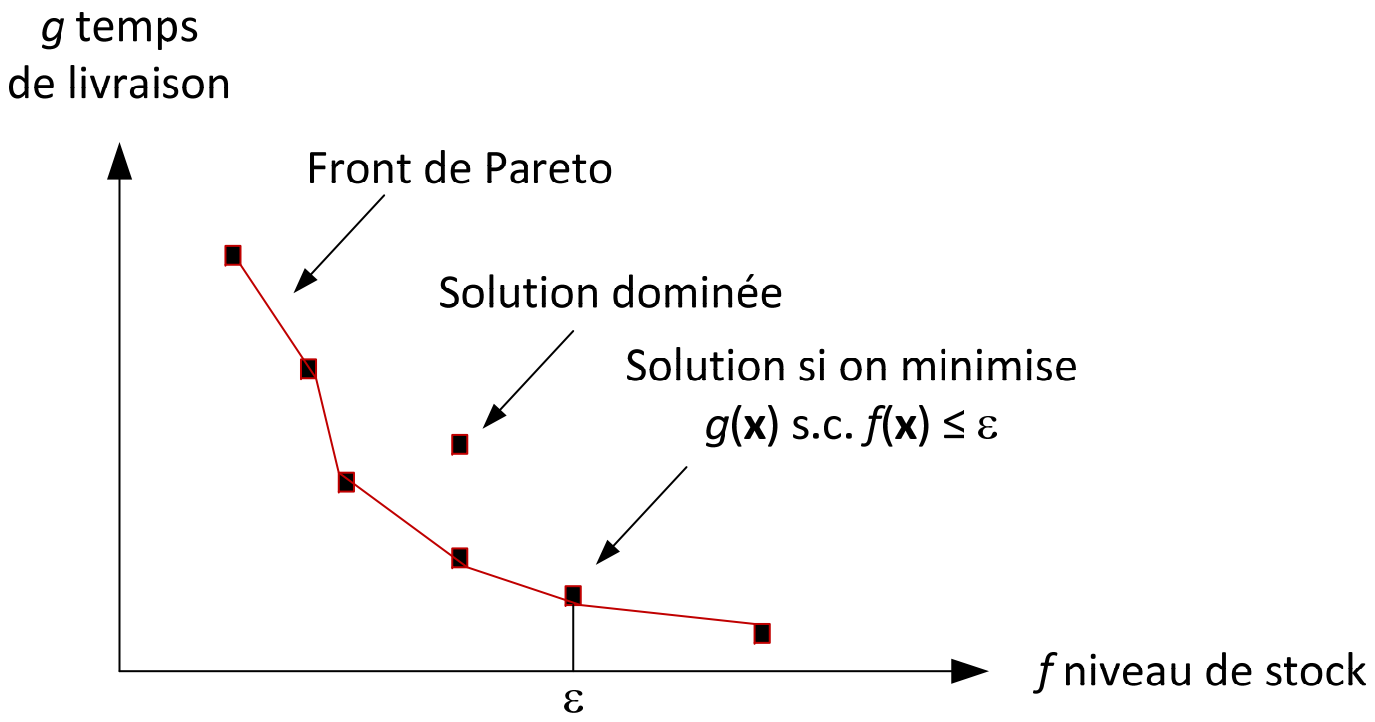
Une solution $\mathbb{x}$ domine au sens de Pareto une solution $\mathbb{y}$ si $f(\mathbb{x}) \leq f(\mathbb{y})$ et $g(\mathbb{x}) \leq g(\mathbb{y})$, avec au moins une inégalité stricte.

Dans notre exemple, une solution (200,17) domine (210,18) et (200,19) car elle améliore au moins un des critères.
Un décideur préfèrera donc cette solution.

Une solution est Pareto-optimale ou non dominée si elle n'est dominée par aucune autre. Il peut exister plusieurs solutions de ce type, par exemple (200,17) et (190,18).

Optimisation multi-objectif

Dans le cas général, résoudre un problème d'optimisation multi-objectif revient à trouver l'ensemble des solutions non dominées, appelé "front de Pareto" ou "ensemble efficace".



Optimisation multi-objectif

L'ensemble Pareto-optimal est énorme pour certains problèmes. Aussi, le décideur est-il satisfait si on lui donne quelques compromis, incluant les deux solutions extrêmes : celle qui minimise $f(\mathbb{x})$ et celle qui minimise $g(\mathbb{x})$.

On peut utiliser la méthode epsilon-contrainte, où on convertit un des objectifs en contrainte.

On résout d'abord les deux cas extrêmes.
Soit f_{min} et f_{max} les valeurs de f des deux solutions extrêmes.

Puis on choisit par exemple 10 valeurs ε, régulièrement espacées dans $[f_{min}, f_{max}]$, puis on résout $\min g(\mathbb{x})$ s.c. $f(\mathbb{x}) \leq \varepsilon$, d'où 12 problèmes d'optimisation à résoudre en tout.