Programmation Linéaire en Nombres Entiers
I. Définition et principes généraux

Sandra U. Ngueveu

INP-ENSEEIHT / LAAS-CNRS
sandra.ngueveu@toulouse-inp.fr - ngueveu@laas.fr

2020/2021

Définition

Soient :

- S un ensemble fini
- f une fonction qui attribue un coût $f(s)$ à chaque élément $s \in S$

Un problème combinatoire consiste à trouver l'élément s_0 de S qui minimise f.

Définition

Soient :

- S un ensemble fini
- f une fonction qui attribue un coût $f(s)$ à chaque élément $s \in S$

Un problème combinatoire consiste à trouver l'élément s_0 de S qui minimise f.

Cela revient à résoudre le problème :

$$\min f(s) \tag{1}$$

sous les contraintes (s.c.)

$$s \in S \tag{2}$$

où S est un ensemble discret

Définition

Cette définition est plus large qu'il n'y paraît au premier abord car elle englobe également :

- les problèmes de maximisation, car $\min f <=> \max(-f)$
- les problèmes de calcul d'une valeur optimale
- les problèmes de décision ou d'existence dont on attend une réponse du type "oui ou "non"

Elle peut être adaptée au cas multi-objectif, stochastique ou dynamique.

Difficulté de l'Optimisation Combinatoire

- Différence avec l'optimisation continue

s.c.

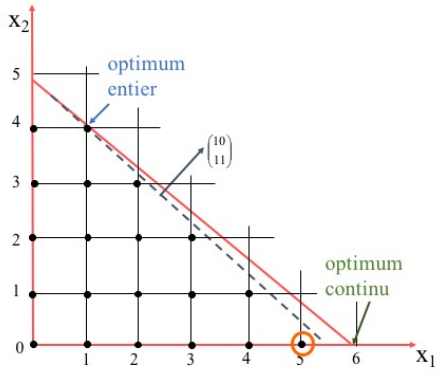
$$\max f(x_1, x_2) = 10x_1 + 11x_2 \quad (3)$$

$$10x_1 + 12x_2 \leq 59 \quad (4)$$

$$x_1 + x_2 \geq 6 \quad (5)$$

$$x_1 \geq 0, x_2 \geq 0 \quad (6)$$

cas continu : $x_1, x_2 \in \mathbb{R}^+$
cas discret : $x_1, x_2 \in \mathbb{N}$



- L'énumération complète est hors de question

$\Rightarrow$ explosion combinatoire (cf slide 11/23)

- Apporter la solution à une instance particulière a peu d'intérêt

Les outils de l'optimisation combinatoire sont empruntés de :

- Programmation linéaire en nombre entiers ou mixte
- Programmation par contraintes (I.A.)
- Théorie des graphes
- Simulation
- Algorithmique + complexité

Les outils de l'optimisation combinatoire sont empruntés de :

- Programmation linéaire en nombre entiers ou mixte
- Programmation par contraintes (I.A.)
- Théorie des graphes
- Simulation
- Algorithmique + complexité

Certains calculs peuvent être faits par des outils commerciaux d'optimisation, mais bien souvent il faut concevoir un algorithme car il n'existe pas de méthode générique performante quelque soit le problème ou l'instance en optimisation combinatoire.

Complexité d'un problème / Complexité d'un algorithme

Les problèmes d'Optimisation Combinatoire se répartissent en deux grandes classes :

- ceux qui sont résolus de manière optimale par des algorithmes polynomiaux (classe P)
- ceux dont la résolution optimale peut prendre un temps exponentiel dans le pire cas (classe NP)

La notion de problème *NP-complet/NP-difficile* permet de mieux comprendre pourquoi certains problèmes ne disposent toujours pas d'algorithmes efficaces à l'heure actuelle.

La complexité d'un problème corresponds à la complexité du meilleur algorithme capable de le résoudre.

Algorithme

Un algorithme est une suite d'opérations élémentaires qui, lorsqu'on lui fournit une instance d'un problème en entrée s'arrête après exécution de la dernière opération en nous renvoyant la solution.
(*Source : Optimisation Combinatoire - Vangelis Th Paschos - 2005*)

Algorithme

Un algorithme est une suite d'opérations élémentaires qui, lorsqu'on lui fournit une instance d'un problème en entrée s'arrête après exécution de la dernière opération en nous renvoyant la solution.
(*Source : Optimisation Combinatoire - Vangelis Th Paschos - 2005*)

Opération élémentaire = opération simple (affectation de vars, tests, ...)

- 1 opération algébrique : $+$, -, *, /
- 1 test logique ou connecteurs logiques : ou, et, non, xor
- 1 test booléen : $<$, $>$, $=$, $\neq$
- ...

Evaluation d'un algorithme

Comment évaluer la qualité d'un algorithme ou comparer 2 algorithmes ?

Les deux critères les plus importants sont :

- le temps de calculs
- l'espace mémoire utilisé

Définition, Difficultés, Outils et Complexité

Evaluation d'un algorithme

Comment évaluer la qualité d'un algorithme ou comparer 2 algorithmes ?

Les deux critères les plus importants sont :

- le temps de calculs
- l'espace mémoire utilisé

D'autres critères, comme la difficulté d'implémentation, la fiabilité, ... peuvent rentrer en compte.

Certains résultats peuvent fortement dépendre des instances résolues et des données utilisées. Mais en général, ce que l'on veut mesurer en priorité c'est si l'algorithme gardera, même dans les pires cas, des temps de calcul raisonnables.

Evaluation d'un algorithme

Complexité [1]

La complexité est une fonction de la taille des données pour prédire l'ordre de grandeur de la variation des temps de calculs quand on passe de petits jeux d'essai de grands problèmes.

Cela permet aussi d'estimer la taille maximale des problèmes qui pourront être résolus en pratique.

1. rien à voir avec la difficulté de programmation ! !

Evaluation d'un algorithme

Complexité [1]

La complexité est une fonction de la taille des données pour prédire l'ordre de grandeur de la variation des temps de calculs quand on passe de petits jeux d'essai de grands problèmes.

La complexité d'un algorithme A travaillant sur des données est une fonction qui exprime le nombre d'instructions exécutées :

- à l'ordre près
- dans le pire des cas,
- en fonction de la taille n des données.

1. rien à voir avec la difficulté de programmation ! !

Evaluation d'un algorithme

Il y a donc deux types d'algorithmes, en fonction de leur complexité :

- ceux dits polynomiaux
  - complexité de l'ordre d'un polynôme
  - ex : $logn$, $n^{0.5}$, $n \log n$, n^2, $O(n^{\log n})$, $O(n^{2.5})$, ...
- ceux dits exponentiels
  - vraie exponentielle au sens mathématique e, 2^n, ...
  - fonction comme factorielle n !, ou n^n, k^n, n^n, ...

Explosion Combinatoire

Taille-> Complexité	20	50	100	200	500	1000
10^3n	0.02 s	0.05 s	0.1 s	0.2 s	0.5 s	1 s
10^3n $\log_2$n	0.09 s	0.3 s	0.6 s	1.5 s	4.5 s	10 s
100 n^2	0.04 s	0.25 s	1 s	4 s	25 s	2 min
10 n^3	0.02 s	1 s	10 s	1 min	21 min	27 h
$n^{\log 2\, n}$	0.4 s	1.1 h	220 j	12500 ans	5.10^{10} ans	--
$2^{n/3}$	0.0001 s	0.1 s	2.7 h	3. 10^6 ans	--	--
2^n	1 s	36 ans	--	--	--	--
3^n	58 min	2. 10^{11} ans	--	--	--	--
n!	77100 ans	--	--	--	--	--

FIGURE – Croissance du temps de calcul pour différentes complexités

Les cases non remplies ont des durées supérieures à 1000 miliards d'années.

Comment résoudre un problème NP-difficile ?

- énumération complète (si très petits problèmes)
- énumération intelligente (problèmes de taille moyenne)
  - programmation dynamique (résolution de sous-problèmes emboités),
  - méthodes arborescentes (séparation en sous-cas dont beaucoup sont éliminés par des tests).
  - ...
- méthodes approchées (problèmes de grande taille) : bonnes solutions mais sans garantie d'optimalité.
  - (méta-)heuristiques
  - recherches locales
  - ...

Par ex, pour des problèmes de transport NP-difficiles (TSP, ...), seules des heuristiques peuvent traiter les très grandes instances (milliers de sommets).

Exemple classique

Il suffit parfois d'ajouter une contrainte/variable/donnée/variante pour qu'un problème passe de *polynomial* à *NP-difficile*.

Le pb d'affectation

Soient n produits et n machines. Fabriquer i coûte d_{ij} si i est affecté au à la machine j.
On veut affecter les produits aux machines pour tout traiter en minimisant le coût total.

... MODELE MATH ... ?

POLYNOMIAL ! ! !

Le pb d'affectation généralisée

Soient m produits, n machines. Fabriquer i dure a_{ij} et coûte d_{ij} si i est fabriqué par la machine j. La durée max de travail d'une machine j est b_j.
On veut affecter les produits aux machines pour tout traiter en minimisant le coût total.

... MODELE MATH ... ?

NP-DIFFICILE ! ! !

Résumé de ce qui a été vu

- Définition de l'optimisation combinatoire
- Relation et différence avec l'optimisation continue
- Problème vs instance : besoin d'algorithmes
- Comment évaluer un algorithme
- Explosion combinatoire : pourquoi un PC plus puissant ne résouds rien

Exercice

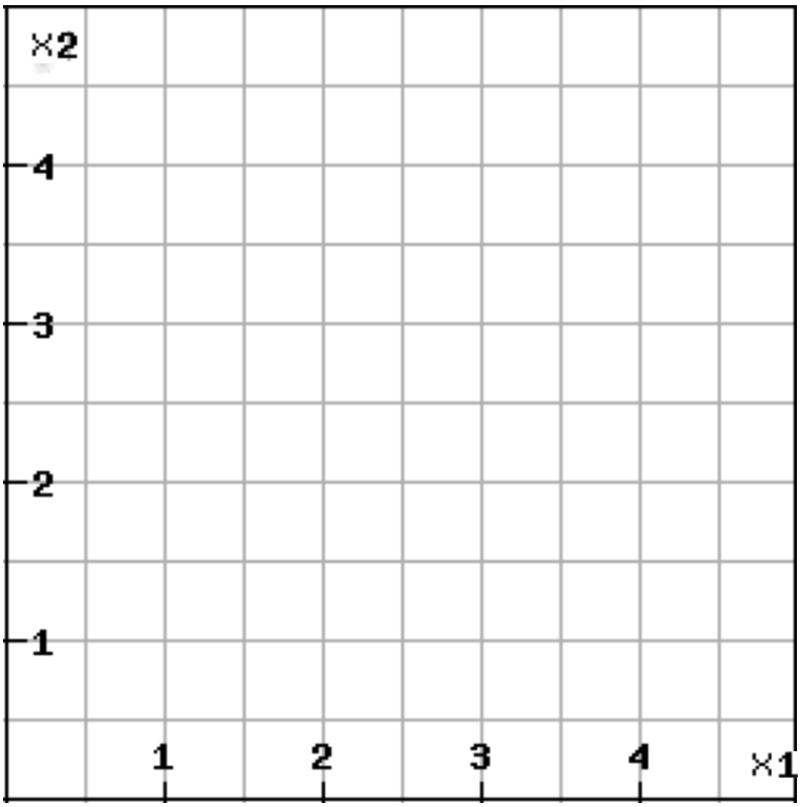
$$(\mathrm{P})\ \min x_1 + x_2 \tag{7}$$

s.c.

$$-2x_1 + 2x_2 \geq 1 \tag{8}$$
$$-8x_1 + 10x_2 \leq 13 \tag{9}$$
$$x_1, x_2 \geq 0 \tag{10}$$
$$x_1, x_2 \quad \text{entiers} \tag{11}$$

1. Appelons (PR) le problème dit relaxé de (P), correspondant à (7)-(10), c'est à dire en ignorant les contraintes d'intégralité (11)
2. Résoudre graphiquement (PR) sur la grille du slide suivant
3. Donner la solution entière obtenue par arrondi de la solution obtenue de (PR)
4. Donner la solution optimale du problème original (P)
5. Conclure.

Exercice



Quelles sont les solutions candidates de (P) et (PR) ?

Quelles sont les solutions réalisables/admissibles de (P) et (PR) ?

Quelles sont les solutions optimales de (P) et (PR) ?

Conclusion : cas Continu vs cas Discret

- Domaines de définition
- Polyèdres
- Solutions Optimales

A lire (scholarvox)

Christian Prins, Marc Sevaux
Programmation linéaire avec Excel : 55 problèmes d'optimisation modélisés pas à pas et résolus avec Excel
Eyrolles, 2011.

2.2.1 Définitions (page 30)

2.2.2 Difficulté de la PLNE (page 30 à 31)

Borne Inférieure et Borne Supérieure

Définitions (en minimisation)

- Borne supérieure = toute valeur supérieure ou égale à la valeur optimale
- Borne inférieure = toute valeur inférieure ou égale à la valeur optimale

Exemples de méthodes pour obtenir des bornes

- Borne supérieure : toute solution réalisable a un coût qui correspond à une borne supérieure
- Borne inférieure : toute solution optimale d'un problème relaxé a un coût qui correspond à une borne inférieure du problème original

Intérêt

- Evaluer la qualité d'une solution
- Estimer l'écart à l'optimum alors qu'on ne connait pas la solution
- Evaluer s'il est pertinent d'allouer plus de temps de calcul à la recherche d'une meilleure solution

PLNE

Une machine M1 peut produire 3 types de pièces A, B, C à partir d'une ressource H.

- Pour chaque A produit, M1 consomme 11 H
- Pour chaque B produit, M1 consomme 7 H
- Pour chaque C produit, M1 consomme 3 H

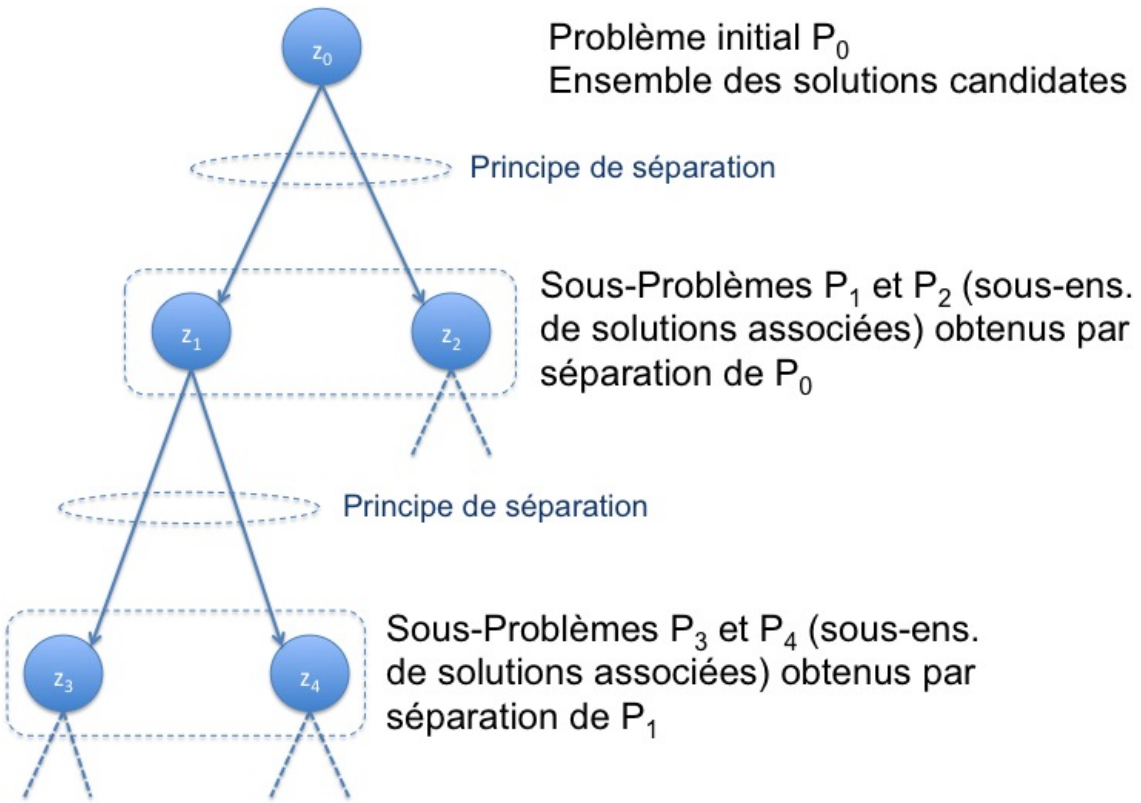
Les pièces produites sont destinées à la vente.

- Chaque A vendu rapporte 45 euros
- Chaque B vendu rapporte 34 euros
- Chaque C vendu rapporte 14 euros

Sachant que l'on dispose de 19H, et que l'on souhaite maximiser le chiffre d'affaire :

1. Modéliser ce problème sous forme d'équation et sous forme matricielle
2. Construire un arbre des décisions
3. En déduire un arbre des sous-problèmes

Procédures de séparation et évaluation pour PLNE



A lire (scholarvox)

Christian Prins, Marc Sevaux
Programmation linéaire avec Excel : 55 problèmes d'optimisation modélisés pas à pas et résolus avec Excel
Eyrolles, 2011.

2.2.1 Méthode arborescente de Dakin (pages 39 à 41)

Procédures de séparation et évaluation pour PLNE

Principe : (schéma slide précédent)

- Séparer progressivement le problème en sous-problèmes traitables (aussi appelés "sondables")

Objectif

- Enumérer intelligemment l'espace des solutions
- Hiérarchiser les sous-problèmes sous forme d'arbre
- "tuer" des branches/nœuds au plus tôt lors de l'exploration de l'arbre

3 composantes :

- Règle de Séparation
- Evaluation/Tests de Sondabilité
- Stratégie d'exploration