Recherche Opérationnelle - APP 2
Allocating & packing

Sophie Demassey

15 novembre 2011

Résumé

Cette seconde série d'études s'attache aux problématiques classiques du *placement* (knapsack, bin-packing, cutting stock), et à leurs relations avec les problématiques de l'*affectation de ressources*, de la planification d'horaires et de l'ordonnancement. Du point de vue *méthodes et techniques de résolution*, ce projet sera axé sur les aspects décomposition de problèmes et renforcement de modèles. On considérera les classes de méthodes suivantes : *programmation dynamique, relaxations linéaire et lagrangienne, branch-and-bound, génération de coupes et branch-and-cut, génération de colonnes et branch-and-price.*

Organisation du projet

Le projet est composé de 6 séances de 2h30 chacune. Il est à mener par groupes de 3 ou 4 personnes chacun. Le rapport de groupe reprendra, de manière homogène et rigoureuse, les exercices proposés en séances. Il n'est pas demandé de respecter l'ordre des questions, ni de répondre à l'intégralité des questions. En particulier, les questions marquées d'une astérisque (*) peuvent être considérées comme optionnelles. En revanche, toute question abordée devra être traitée, avec preuves et références à l'appui. Ce projet donnera lieu à une évaluation individuelle (devoir sur table). Une évaluation collective, par groupe, portera sur l'étude d'implémentation de deux approches de résolution de problématiques du projet : la première approche consistera en un algorithme ad-hoc, type programmation dynamique ou heuristique, la seconde portera sur l'utilisation d'un solveur linéaire (Xpress-MP, GLPK ou Coin-OR, par exemple). L'implémentation sera accompagnée de tests numériques et d'une analyse des résultats à consigner dans un rapport.

Notations

- $\mathbb{Z}$, $\mathbb{Z}_+$, $\mathbb{Z}_+^*$ dénotent les ensembles, respectivement, des entiers, des entiers positifs, des entiers strictement positifs ;
- $\lceil x \rceil$ et $\lfloor x \rfloor$ dénotent les arrondis entiers de $x \in \mathbb{R}$, respectivement, à la valeur supérieure, à la valeur inférieure : $\lceil x \rceil \in \mathbb{Z}$, $\lfloor x \rfloor = \lceil x \rceil - 1$ et $\lfloor x \rfloor \leqslant x \leqslant \lceil x \rceil$

Références

[1] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8 :101–111, 1960.

[2] G. B. Dantzig. Discrete variable extremum problems. *Operations Research*, 5 :266–277, 1957.

[3] Michael R. Garey and David S. Johnson. *Computers and Intractability : A guide to the theory of NP-completeness*. W.H. Freeman & Company, 1979.

[4] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem, part ii. *Operations Research*, 11 :863–888, 1963.

[5] Silvano Martello and Paolo Toth. *Knapsack problems*. Wiley, 1990.

Séance 1 : knapsack 0-1 et programmation dynamique

Problème 1 *Une fabrique de pièces détachées livre chaque semaine, par avion cargo, un ensemble de pièces à une usine d'assemblage. Les pièces sont toutes distinctes, généralement lourdes et de petite taille. Par conséquent, la quantité de pièces transportées à chaque livraison est limitée, et cette limite est uniquement due à la capacité de poids total de l'avion. Les pièces disponibles qui ne sont pas livrées une semaine, sont simplement mises en stock. Avant chaque livraison, on estime pour chaque pièce disponible l'intérêt de la livrer au plus tôt : la valeur numérique de* profit *associée à une pièce est d'autant plus élevée que la livraison de la pièce est urgente. Le problème posé est alors :* parmi l'ensemble des pièces disponibles, quel sous-ensemble de pièces de meilleur profit doit-on livrer par le prochain cargo ?

Question 1 *Décrire le problème par un modèle de programmation mathématique.*

Le problème posé est une instance du problème de sac-à-dos binaire ou *0-1 knapsack problem*. Une instance générale de ce problème est définie par un ensemble $I_n = \{1, \dots, n\}$ d'*items* (ici, les pièces disponibles) à placer dans un *container* ou *knapsack* (ici, le cargo) de *capacité* $c \in \mathbb{Z}_+$. À chaque item $i \in I_n$, sont associés un *profit* $p_i \in \mathbb{Z}$ et un *poids* $w_i \in \mathbb{Z}_+$.

Question 2 *Montrer que résoudre une instance générale du knapsack 0-1 peut toujours se ramener à résoudre une instance de taille inférieure, vérifiant les conditions ci-dessous :*
- *le profit et le poids de chaque item, et la capacité du container sont strictement positifs ;*
- *le poids de chaque item est inférieur à la capacité ;*
- *la somme des poids des items est strictement supérieure à la capacité ;*

Remarque : la condition $w_i \geqslant 0$ n'est pas imposée dans la définition classique du problème, mais elle se déduit de la même manière.

Question 3 Complexité.
Donner la classe de complexité du problème de knapsack 0-1, sachant que :
- *un problème d'optimisation est* NP-difficile *si le problème dans sa variante décisionnelle (réponse par OUI ou NON) est* NP-complet *;*
- *un problème de décision est* NP-complet *s'il possède un problème* NP-complet *comme cas particulier ;* [1]
- *le problème suivant est* NP-complet *[3].* D(Subset Sum) : soient n+2 entiers positifs $\{a, b, a_1, \dots, a_n\}$, existe-t'il un sous-ensemble $S \subseteq \{1, \dots, n\}$ tel que $a \leqslant \sum_{i \in S} a_i \leqslant b$?

Le problème du knapsack 0-1 n'est pas *NP-difficile au sens fort*, car il existe pour le résoudre un algorithme de complexité temporelle *pseudo-polynomiale* (polynomiale en terme de valeurs). Les questions suivantes ont pour but d'exhiber un tel algorithme. Il s'agit d'un algorithme de *programmation dynamique*. Le principe de la programmation dynamique est proche du principe de récursivité. Il s'applique à tout problème d'optimisation combinatoire pouvant se décomposer en une séquence finie croissante de sous-problèmes imbriqués de même nature, telle que :
- le dernier sous-problème de la séquence inclut le problème global ;
- les solutions d'un sous-problème peuvent être obtenues à partir des solutions du sous-problème précédent au moyen d'une formule de transfert.

Ainsi, la résolution d'un problème par programmation dynamique consiste à résoudre le plus petit des sous-problèmes, puis chacun des sous-problèmes en séquence, de manière itérative, au moyen de la formule de transfert.

Dans la suite, on cherche à résoudre $P_n(c)$ une instance du knapsack 0-1 sur un ensemble d'items (I_n, w, p) et un container de capacité c. On suppose sans perte de généralité que $P_n(c)$ vérifie les conditions de la question 2. Le système dynamique associé possède n étapes : à chaque étape k, on considère des problèmes de knapsack 0-1 portant sur le sous-ensemble d'items $I_k = \{1, \dots, k\}$.

Question 4 *Soit $S_n \subseteq I_n$ une solution optimale de $P_n(c)$. Donner une valeur $c' \in \mathbb{Z}_+$ pour laquelle $S_n \setminus \{n\}$ est une solution optimale de $P_{n-1}(c')$. (On considèrera au moins deux cas, selon que S_n contient ou non l'item n ; on notera par ailleurs que l'ensemble vide est une solution réalisable du problème de knapsack).*

1. ces deux conditions sont suffisantes et non nécessaires : la réduction est l'outil usuel de calcul de complexité d'un problème, et la référence en complexité est le livre de Garey et Johnson [3].

Question 5 *On note $f_n(c)$ la valeur maximale du profit pour le problème $P_n(c)$. Exprimer $f_n(c)$ pour tout $c \in \mathbb{Z}_+$ par une formule de récurrence sur $n \in \mathbb{Z}_+$.*

Question 6 *On cherche à calculer $f_n(c)$, par cette formule. Pour quelles valeurs de c', est-il nécessaire de calculer $f_{n-1}(c')$? $f_{n-2}(c')$? $f_{n-3}(c')$?... $f_1(c')$? Quel est le nombre maximal de valeurs c' pour lesquelles il est nécessaire de calculer $f_1(c')$?*

Question 7 *Proposer un algorithme itératif de calcul de $f_n(c)$ tel qu'à chaque étape k, un nombre suffisant de valeurs $f_k(c)$ soit calculé. Donner sa complexité temporelle et spatiale.*

Question 8 *Modifier votre algorithme afin qu'il retourne également une solution optimale de $P_n(c)$. Donner sa complexité temporelle et spatiale.*

Question 9 ** Retrouvez l'algorithme de Bellman pour le calcul des plus courts chemins dans un graphe depuis un sommet vers tous les autres. Cet algorithme de programmation dynamique est basé sur les observations suivantes :*
- *un plus court chemin dans un graphe $G(V, E)$ sans cycle négatif, est simple et possède au plus $|V| - 1$ arcs ;*
- *soit γ un chemin de longueur minimale parmi l'ensemble des chemins d'au plus k arcs reliant un sommet s à un autre sommet v, et soit (v', v) le dernier arc de γ, alors $\gamma \setminus (v', v)$ est un chemin... [compléter !]*

Question 10 *Le système dynamique associé au kapsack 0-1 est identique au système associé à l'algorithme de Bellman mais dans un graphe particulier. Représenter ce graphe.*

La recherche de chemins optimaux dans un graphe est l'application la plus courante de la programmation dynamique. L'algorithme de Bellman peut en effet être adapté de manière à tenir compte des contraintes aditionnelles sur les chemins, notamment des contraintes de ressources, de capacité de véhicules ou de fenêtres de temps. Un autre domaine typique d'application de la programmation dynamique est l'optimisation stochastique, pour la gestion de stock par exemple :

Problème 2 *Un chef de magasin doit planifier le réapprovisionnement de son produit phare sur N semaines. La capacité de son stock étant limité, un réapprovisionnement est effectué chaque fin de semaine. Les demandes (ventes) hebdomadaires sont aléatoires et indépendantes d'une semaine à l'autre. On connaît uniquement leur loi de probabilités. On connaît enfin, pour chaque unité de produit, le coût de réapprovisionnement, le coût de stockage, le coût de pénurie.*

Question 11 ** Soit x_k la quantité de produit en stock au début de la semaine k, d_k la quantité de produit vendue pendant la semaine k, et u_k la quantité de produit commandée pour la fin de la semaine k. Exprimer x_k par une formule de récurrence sur $k \in \mathbb{Z}_+^*$. Exprimer $f_k(x_k, d_k, u_k)$ le coût de gestion de la semaine k. Le coût total optimal est alors égal à l'espérance de la somme des coûts à la semaine.*

Les algorithmes de programmation dynamique sont de complexités temporelles diverses – polynomiale (Bellman), pseudo-polynomiale (knapsack), ou exponentielle (TSP) – mais, de manière générale, ils sont gourmands en espace. L'optimisation de ce type d'algorithmes porte ainsi sur l'élimination progressive des calculs intermédiaires inutiles, par exemple par des arguments de redondance. On trouvera dans [5] des exemples d'algorithmes de programmation dynamique optimisés pour le knapsack.

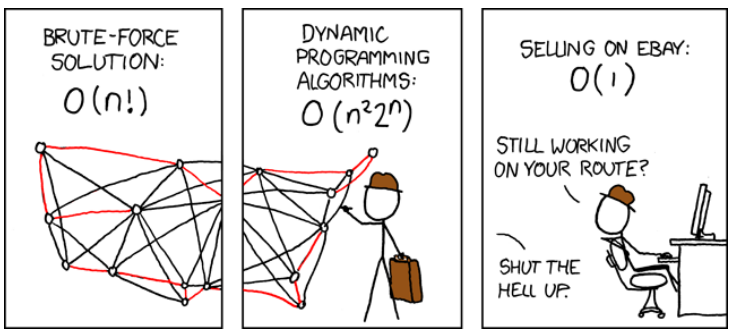

FIGURE 1 – TSP on XKCD

Séance 2 : variantes du knapsack

change-making problem

Problème 3 *Faire le change de 5 euros et 32 centimes au moyen d'un nombre minimal de pièces de 1, 2, 5, 10, 20 et 50 centimes, 1 et 2 euros.*

Question 12 *Proposer un algorithme intuitif de résolution pour le problème 3. De quel type d'algorithme s'agit-il ? Quelle est sa complexité ? Est-il assuré de terminer sur une solution réalisable ? une solution optimale ?*

Question 13 *Formaliser le* change-making problem *avec données entières. Proposer une condition – vérifiable en temps polynomial – suffisante d'existence d'une solution réalisable. La variante de réalisabilité du* change-making problem *vous semble-t'elle « simple » dans le cas général ?*

Le problème de réalisabilité du *change-making problem* est en fait *NP-complet* [5] et la variante d'optimisation est donc *NP-difficile*. On considère la classe des instances du problème qui vérifient la condition de réalisabilité ci-dessus. Même avec cette restriction, l'algorithme intuitif de résolution ne possède pas de garantie de performance :

Problème 4 *Soit un entier* $k > 1$, *faire le change de* $2k$ *au moyen de trois types de pièces de valeurs respectives* 1, k *et* $k+1$.

Question 14 *Quel est le nombre minimale de pièces à rendre dans le problème 4 ? Quelle est la valeur retournée par votre algorithme ? Conclure sur la performance de cet algorithme.*

Définition 1 *Soit* (P) *un programme linéaire en nombres entiers,* $(\bar{P})$ *sa relaxation continue et* C *une inégalité linéaire en les variables de* (P).
- C *est une* inégalité valide *de* (P) *si elle est satisfaite par toute solution optimale de* (P) ;
- C *est un* plan coupant *(ou* coupe*) de* (P) *si c'est une inégalité valide de* (P) *qui n'est pas satisfaite par au moins une solution optimale de* $(\bar{P})$;
- C *est* dominée *par une autre inégalité valide* C' *si toute solution de* C' *est solution de* C.

Question 15 *Soit une instance générale du* change-making problem *avec* $c \in \mathbb{Z}$ *la valeur du change et* $w_1, \ldots, w_n \in \mathbb{Z}$ *les valeurs des pièces. On note* (P) *le PLNE correspondant.*

1. *montrer que l'on peut supposer, sans perte de généralité, que les pièces sont classées par ordre strictement décroissant de leurs valeurs ;*
2. *en considérant uniquement des pièces de plus haute valeur* w_1, *exhiber une solution optimale de la relaxation continue de* (P). *En déduire que* $\lceil \frac{c}{w_1} \rceil$ *est une borne inférieure du problème ;*
3. *soit* $q_1 = \lfloor \frac{c}{w_1} \rfloor$, *montrer que* $x_1 \leqslant q_1$ *est une coupe de* (P) ;
4. *en considérant également des pièces de valeur* w_2, *exhiber une solution optimale de la relaxation continue du programme* (P) *augmenté de cette coupe. En déduire une borne inférieure du problème au moins aussi bonne que la précédente ;*
5. * *soient* $c_1 = c - q_1 w_1$, $q_2 = \lfloor \frac{c_1}{w_2} \rfloor$ *et* $c_2 = c_1 - q_2 w_2$, *noter que toute solution optimale vérifie soit* $x_2 > q_2$ *et* $x_1 \leqslant q_1 - 1$, *soit* $x_2 \leqslant q_2$. *En déduire que* $q_1 + q_2 + \min(\lceil \frac{c_2}{w_3} \rceil, \lceil \frac{c_2 + w_1}{w_2} \rceil - 1)$ *est une borne inférieure du problème [5].*

Question 16 * *On note* $f_n(c)$ *la valeur optimale du change-making problem avec, par convention,* $f_n(c) = +\infty$ *si le problème est irréalisable. Exprimer* $f_n(c)$ *pour tout* $c \in \mathbb{Z}_+$ *par une formule de récurrence sur* $n \in \mathbb{Z}_+$. *En déduire un algorithme de programmation dynamique et la classe de complexité du problème.*

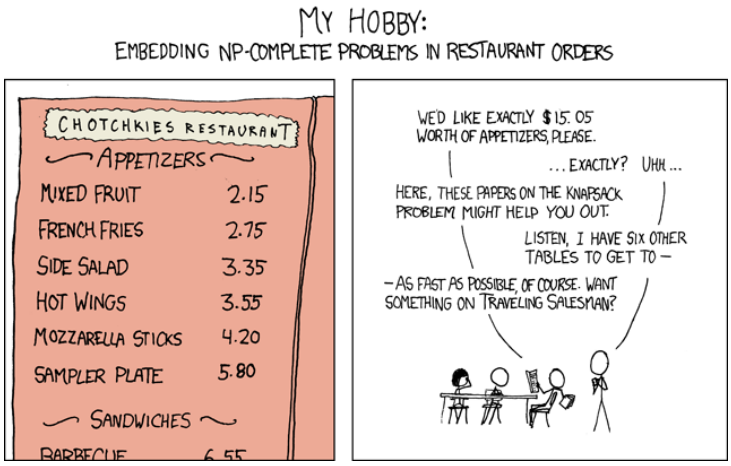

FIGURE 2 – KS on XKCD

continuous knapsack problem

Dans cette section, on considère une instance (c, I_n, w, p) du knapsack 0-1 vérifiant les conditions de la question 2 et telle que les items sont ordonnés comme suit $\frac{p_1}{w_1} \geqslant \frac{p_2}{w_2} \geqslant \ldots \geqslant \frac{p_n}{w_n}$. (on notera que le tri des items peut être effectué en temps polynomial). Comme pour le problème précédent, il est facile de trouver une solution fractionnaire.

Question 17 ** Le résultat suivant est dû à Dantzig [2] :*

1. *montrer que l'on peut supposer, sans perte de généralité, que les rapports $\frac{p_i}{w_i}$ sont tous différents ;*
2. *montrer que la solution optimale fractionnaire vérifie la contrainte de knapsack à l'égalité ;*
3. *montrer que l'ensemble des items peut être partitionnés suivant le seuil $s = \min\{j \in I_n \mid \sum_{i=1}^{j} w_j > c\}$;*
4. *soit $\bar{c} = \sum_{i=1}^{s-1} w_j$, montrer que la solution fractionnaire suivante est optimale :*
   $\bar{x}_1 = 1, \ldots, \bar{x}_{s-1} = 1, \bar{x}_s = \frac{\bar{c}}{w_s}, \bar{x}_{s+1} = 0, \ldots, \bar{x}_n = 0$.
   (indice : si x^* est une solution fractionnaire optimale telle que $x_i^* < 1$ pour $1 \leqslant i < s$ alors, d'après le second point, il existe $\epsilon > 0$ et $j \geqslant s$ tel que x_i^* peut être augmenté de ϵ et x_j^* peut être diminué de...)

bounded knapsack problem

Problème 5 *Chaque année, le conseil régional procède à l'allocation de son budget à des projets sélectionnés parmi une liste de projets soumis. Pour chaque type de projet (construction d'école maternelle, agrandissement de lycée, achat de véhicules de transports publics, réfection de portions de voirie, etc.), un coût et un bénéfice sont estimés. Plusieurs simulations sont effectuées afin d'attribuer au mieux le budget disponible : pour chaque simulation, on impose un nombre minimal et maximal de projets à réaliser de chaque type. On fait ensuite varier ces bornes pour simuler différents scenarii.*

Question 18 *Formaliser le* bounded knapsack problem *(BKS) illustré ici. Transformer une instance générale du problème en une autre instance, dans laquelle les bornes inférieures sur les quantités d'items sont toutes nulles. Transformer cette instance en une instance du* 0-1 knapsack problem. *Donner la taille de l'instance.*

Question 19 ** Montrer qu'il existe une transformation du bounded knapsack problem vers le 0-1 knapsack problem tel que le nombre de variables du second problème est égal à $\sum_{i=1}^{n}(\lceil \log_2(b_i + 1) \rceil)$ où b_i est la borne supérieure (entière) sur la quantité d'items $i \in I_n$ dans le premier problème. (indice : tout entier positif b peut être décomposé par $b = \sum_{k=0}^{q} 2^k + r$ avec q et r entiers tels $q = \lfloor \log_2(b+1) \rfloor$ et $0 \leqslant r < 2^{q+1}$.)*

Question 20 *Quand les quantités d'items ne sont pas spécifiquement contraintes (bornes supérieures égales à $+\infty$), on parle alors du* unbounded knapsack problem *(UKS). Montrer qu'il s'agit d'un cas particulier du BKS en exhibant des bornes supérieures induites.*

Séance 3 : multi-knapsack et relaxations linéaires

Problème 6 *Un système informatique de vidéo à la demande (VOD) doit pouvoir « jouer » simultanément différents flux de centaines de vidéos, basé sur la demande des utilisateurs. Un tel système consiste en un processeur central et une collection* J *de disques partagés de capacités finies* $c_j, \forall j \in J$. *Une vidéo* $i \in I$ *est rendue disponible comme un fichier MPEG de taille* w_i *enregistré sur l'un des disques et à chaque vidéo est associé un taux de probabilité de téléchargement* p_i. *Sachant que la capacité totale des disques est insuffisante pour y stocker toutes les vidéos, le problème consiste à affecter un sous-ensemble des vidéos aux disques de façon à maximiser le nombre total de téléchargement.*

Question 21 0-1 Multiple Knapsack Problem.
Exprimer cette problématique en terme de conteneurs et d'items et la formuler en un programme linéaire en variables binaires, noté (MKP). *Montrer qu'un tel programme est réalisable et borné quelque soit l'instance. On note* $z(MKP)$ *sa valeur optimale.*

Question 22 Surrogate relaxation.
Soit $(\pi_j)_{j\in J} \in \mathbb{R}_+^J$ *un vecteur de* multiplicateurs *et le programme linéaire* $S(MKP, \pi)$ *suivant :*

$$\max\ z(S(MKP,\pi)) = \sum_{i\in I}\sum_{j\in J} p_i x_{ij} \tag{1}$$

$$s.t.\ \sum_{j\in J}\pi_j \sum_{i\in I} w_i x_{ij} \leqslant \sum_{j\in J}\pi_j c_j \tag{2}$$

$$\sum_{j\in J} x_{ij} \leqslant 1 \qquad \forall i \in I, \tag{3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I, j \in J. \tag{4}$$

- *montrer que* $S(MKP,\pi)$ *est une relaxation de* (MKP) *– appelée relaxation* surrogate *– et comparer l'optimum de* $S(MKP,\pi)$ *à l'optimum de* (MKP) *;*
- *les questions suivantes ont pour but de chercher un* vecteur optimal des multiplicateurs, *qu'entend-on par là ?*
- *soit* $j' \in \arg\min\{\pi_j, j \in J\}$ *montrer qu'il existe une solution optimale* x^* *de* $S(MKP,\pi)$ *telle que* $x^*_{ij} = 0$ *pour tout* $j \neq j'$ *;*
- *calculer* $z(S(MKP,\pi))$ *si* $\pi_{j'} = 0$ *;*
- *on suppose* $\pi_{j'} > 0$, *montrer que* $S(MKP,\pi)$ *est équivalent au programme linéaire suivant (nommer ce problème) :*

$$\max \sum_{i\in I} p_i y_i \tag{5}$$

$$s.t.\ \sum_{i\in I} w_i y_i \leqslant \lfloor \sum_{j\in J}\pi_j c_j/\pi_{j'} \rfloor \tag{6}$$

$$y_i \in \{0,1\} \qquad \forall i \in I. \tag{7}$$

- *Pour quelles valeurs de* π, *la capacité dans ce programme (membre droit de l'inégalité (6)) est-elle minimale ? En déduire la famille des vecteurs de multiplicateurs optimaux et un algorithme d'évaluation par excès (i.e. un algorithme de calcul d'une borne supérieure) de la valeur optimale de* (MKP).

Question 23 * Relaxation surrogate et relaxation continue.
On note $S(MKP)$ *la relaxation surrogate optimale de* (MKP), *c'est à dire le programme linéaire en nombres entiers (5)-(7) correspondant à un vecteur optimal* π.

- *montrer que la relaxation continue de* $S(MKP)$ *coincide avec la relaxation surrogate optimale de la relaxation continue de* (MKP) *:* $C(S(MKP)) = S(C(MKP))$, *en déduire que* $z(C(S(MKP))) \geqslant z(C(MKP))$ *la borne de la relaxation continue du problème.*
- *en réordonnant les items, exhiber une solution optimale* $\bar{y}$ *de la relaxation continue de* $S(MKP)$ *(voir question 15)*

– *transformer cette solution en un solution $\bar{x}$ réalisable de* $C(MKP)$ *telle que* $\sum_{j\in J}\bar{x}_{ij} = \bar{y}_i$ *our tout* $i \in I$ *(on place les items dans l'ordre induit : on remplit le premier container, puis le second, etc.)*
– *en déduire que* $z(C(S(MKP))) = z(C(MKP))$ *et donc, que la borne de la relaxation surrogate* $z(S(MKP))$ *du multi-knapsack est au moins aussi bonne que la borne de la relaxation continue* $z(C(MKP))$.

Question 24 Lagrangian relaxation of the knapsack constraints.
Soit $(\mu_j)_{j\in J} \in \mathbb{R}^J_+$ *un vecteur de* multiplicateurs *et le programme linéaire* $L^K(MKP,\mu)$ *suivant :*

$$\max\ z(L^K(MKP,\mu)) = \sum_{i\in I}\sum_{j\in J} p_i x_{ij} - \sum_{j\in J}\mu_j\Big(\sum_{i\in I} w_i x_{ij} - c_j\Big) \tag{8}$$

$$s.t.\ \sum_{j\in J} x_{ij} \leqslant 1 \qquad \forall i \in I, \tag{9}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I, j \in J. \tag{10}$$

– *montrer que* $L^K(MKP,\mu)$ *est une relaxation de* (MKP) *– appelée relaxation* lagrangienne *de paramètre* μ *associée aux constraintes de sac-à-dos – et comparer l'optimum de* $L^K(MKP,\mu)$ *à l'optimum de* (MKP) *;*
– *comme pour la relaxation surrogate, on cherche le vecteur multiplicateur optimal en résolvant le programme non-linéaire suivant, appelé* dual lagrangien *:*

$$z(L^K(MKP)) = \min\{z(L^K(MKP,\mu)) \mid \mu \in \mathbb{R}^J_+\}$$

. Montrer que $z(L^K(MKP)) = \min\{z(L^K(MKP,\mu)) \mid \mu \in \mathbb{R}^J_*+\}$*;*
– * *soit* $j' \in \arg\min\{\mu_j, j \in J\}$*, montrer que la solution suivante est optimale pour* $L^K(MKP,\mu)$ *et aussi pour sa relaxation continue* $C(L^K(MKP,\mu))$ *:* $x_{ij} = \begin{cases} 1 & \text{si } j = j' \text{ et } p_j - \mu_j w_i > 0, \\ 0 & \text{sinon.} \end{cases}$
– *en déduire que la borne de cette relaxation lagrangienne* $z(L^K(MKP))$ *du multi-knapsack n'est pas meilleure que la borne de la relaxation continue* $z(C(MKP))$.

La relaxation lagrangienne précédente n'est pas intéressante car sa borne – calculable de manière analytique cependant – n'est pas meilleure que la borne de la relaxation continue. Dans tous les cas, la borne de relaxation lagrangienne est toujours au moins aussi bonne que la borne de relaxation continue et, sous certaines conditions, elle peut être égale à l'optimum entier.

La relaxation lagrangienne est souvent utilisée en pratique, quand un programme linéaire présente des contraintes *compliquantes* : une fois ces contraintes relâchées (i.e. supprimées), le programme linéaire résultant est « facile » à résoudre. Plutôt que de relâcher totalement ces contraintes, la relaxation lagrangienne les *dualise* en pénalisant leur violation dans la fonction objectif du programme. Les *multiplicateurs de Lagrange* μ sont une pondération de ces pénalités. Résoudre le problème du dual lagrangien consiste à trouver la meilleure pondération : celle qui induit la borne la plus serrée. Malheureusement, en général, il n'est pas possible de résoudre le dual lagrangien – minimiser (ou maximiser) une fonction convexe linéaire par morceaux sur $\mathbb{R}^m_+$ – en temps polynomial. En revanche, plusieurs algorithmes efficaces existent, tels que *l'algorithme du sous-gradient*, *la génération de coupes*, *l'algorithme des faisceaux*, etc. Ces algorithmes itératifs consistent à résoudre des *sous-problèmes lagrangiens* $L(P,\mu)$ en séquence pour différentes valeurs des multiplicateurs μ. Des propriétés grantissent alors la convergence vers le sous-problème lagrangien de coût minimal.

Question 25 *Soit* $(\lambda_i)_{i\in I} \in \mathbb{R}^I_+$ *un vecteur de* multiplicateurs*, formuler* $L^A(MKP,\lambda)$ *le sous-problème lagrangien obtenu par dualisation dans* (MKP) *des contraintes d'affectation* $\sum_{j\in J} x_{ij} \leqslant 1$, $\forall i \in I$*. Montrer que ce programme linéaire peut être décomposé en plusieurs problèmes indépendants de knapsack 0-1, qui diffèrent uniquement par la valeur de la capacité du conteneur.*

Question 26 *Estimer des bornes supérieures de l'optimum pour l'instance du multi-knapsack 0-1 suivant :*
$|I| = 6$, $|J| = 2$, $c = (65, 85)$
$p = (110, 150, 70, 80, 30, 5)$
$w = (40, 60, 30, 40, 20, 5)$.

Séance 4 : cutting-stock et bin-packing

Problème 7 *Une usine fabrique des feuilles metalliques sous la forme de rouleaux de 10m de large (*rouleaux initiaux*) puis les débite en rouleaux moins larges (*rouleaux finaux*) suivant les commandes des clients. Étant donné l'ensemble de commandes résumé dans la table ci-dessous, procéder à la découpe des rouleaux de façon à minimiser le nombre de rouleaux initiaux à découper.*

table des commandes								
taille des rouleaux (cm)	*14*	*31*	*36*	*45*	*98*	*125*	*164*	*217*
nombre de rouleaux	*211*	*395*	*610*	*97*	*123*	*390*	*43*	*22*

Question 27 bornes supérieures et heuristiques.
- *Exhiber une borne supérieure triviale pour ce problème ;*
- *proposer un algorithme glouton sur les rouleaux initiaux : à chaque itération, un nouveau rouleau initial est considéré et découpé ;*
- *proposer un algorithme glouton sur les rouleaux finaux : à chaque itération, un nouveau rouleau final est considéré et affecté à un rouleau initial ;*
- *en pratique, les commandes arrivent de manière dynamique : on ne connaît pas, a priori, les quantités et tailles de tous les rouleaux finaux. Laquelle de ces deux heuristiques s'applique alors ?*
- ** définir les heuristiques connues pour les problèmes de découpe (*cutting-stock*) et de* bin-packing *à 1 dimension suivantes :* Next Fit, First Fit, Best Fit, Worse Fit*; donner leur complexité ; existe-t'il une relation de dominance entre ces heuristiques ?*
- ** une stratégie consiste à trier au préalable les rouleaux finaux par ordre de largeurs décroissantes. Donner l'intuition que cette stratégie implémente. Est-elle toujours gagnante ?*

Question 28 formulation linéaire du bin-packing 1D.
Formuler le problème par un programme linéaire sur des variables booléennes y, x *définies par :*

$y_i = 1$ *si et seulement si le rouleau initial* i *est utilisé, et*

$x_{ij} = 1$ *si et seulement le rouleau final* j *est découpé dans le rouleau initial* i.

Question 29 formulation linéaire du cutting-stock 1D.
Proposer une seconde formulation linéaire par agrégation des rouleaux finaux de même largeur découpés sur un même rouleau initial.

On dit qu'une formulation souffre de *symétrie de variables* si deux solutions distinctes de la formulation encodent une même solution du problème modélisé.

Question 30 *Comparer les deux formulations en termes de taille et de symétrie de variables. Améliorer ces modèles pour réduire la symétrie des variables* y.
Montrer que les bornes inférieures de relaxation continue des deux modèles sont identiques.

Question 31 Décomposition. *À partir de la première formulation, montrer que le problème se décompose comme suit : il s'agit d'un problème de partitionnement (*exact cover *ou* set partition*) dont les données peuvent être générées par résolution d'un second problème combinatoire que l'on qualifiera.*
Proposer une troisième formulation linéaire du problème basée sur cette décomposition, puis une quatrième formulation par agrégation des rouleaux finaux de même largeur.

Ce programme linéaire est appelé la formulation de Gilmore-Gomory pour le cutting-stock problem [4]. Ici, nous l'avons obtenu par décomposition sémantique du problème. Il existe également une méthode de décomposition systématique des programmes linéaires : la *méthode de décomposition de Dantzig-Wolfe* [1]. Cette méthode appliquée au deuxième modèle (question 29)– dit, par opposition, *modèle compact* – résulte aussi en la formulation de Gilmore-Gomory.

Question 32 *Estimer la taille du programme linéaire obtenu par décomposition. Estimer le nombre de variables non nulles dans toute solution optimale. Le programme possède-t'il une symétrie de variables ?*
Montrer que la borne inférieure de relaxation continue du modèle décomposé est au moins aussi bonne que celle du modèle compact.

Construire une solution réalisable à partir d'une solution de la relaxation continue et proposer un test d'optimalité de cette solution.

La valeur optimale de relaxation continue du modèle décomposé fournit une excellente borne : il fut longtemps conjecturé que sa valeur entière supérieure ($\lceil LB \rceil$) était égale à l'optimum entier (OPT), jusqu'à ce qu'un contre-exemple soit trouvé. Sur la pire instance connue à ce jour, le gap fractionnaire ($OPT - \lceil LB \rceil$) est « seulement »[2] égal à $\frac{6}{5}$.

Ainsi, la solution réalisable obtenue par approximation de la solution fractionnaire du programme linéaire décomposé conduit le plus souvent en pratique à une solution quasi-optimale. Si celle-ci n'est pas optimale, on peut commencer une recherche exacte par branch-and-bound. Cependant le branchement classique qui consiste à forcer l'intégralité d'une variable fractionnaire (dans la solution courante) n'est pas efficace ici :

- une première raison est que cette formulation possède une sorte de symétrie, dans le sens où les variables correspondent à des motifs de découpe – toujours distincts mais, pour beaucoup, très proches. À cause de cette similarité entre les variables, prendre une décision (i.e. brancher) sur une variable a souvent peu d'impact sur la borne calculée aux noeuds fils de l'arbre de recherche (on imagine que les solutions optimales sont nombreuses et réparties diffusément dans presque tous les sous-arbres de l'arbre de recherche) ;
- une deuxième raison est que l'arbre de recherche résultant de ce branchement est non-balancé : affecter une variable à 0 (interdire un motif de découpe) est une décision moins forte (i.e. le sous-arbre est beaucoup plus large) que la décision contraire (forcer la découpe d'un rouleau suivant ce motif).

Une meilleure stratégie de branchement consiste à brancher sur des ensembles de variables, autrement dit, sur des contraintes. Cette stratégie est appelée *constraint branching*. Elle permet d'obtenir des arbres de recherche mieux balancer et moins symétriques. Pour être applicable, elle nécessite cependant de prouver qu'elle est complète : (1) chaque feuille de l'arbre de recherche correspond bien à une solution réalisable entière, et (2) toute solution correspond à au moins une feuille de l'arbre de recherche.

Question 33 * *Définir les stratégies de branchement suivantes :* constraint branching, GUB (Generalized Upper Bound) branching, SOS (Special Ordered Set) branching, strong branching.
Montrer comment le GUB branching s'applique au problème de set-partitioning.

Avec le défaut d'une stratégie de branchement simple et efficace, l'autre principal inconvénient de la formulation de Gilmore-Gomory est le nombre de variables potentiellement exponentiel. Pour résoudre la relaxation continue d'un tel modèle, on fait ainsi généralement appel à la méthode de *génération de colonnes* : cette méthode de résolution itérative consiste, à chaque itération, à résoudre la relaxation continue du programme sur seulement un sous-ensemble des variables, puis à générer et ajouter dynamiquement de nouvelles variables (i.e. des colonnes) « améliorantes ». Comme dans l'algorithme du simplexe, une colonne améliorante est une colonne hors-base dont le coût réduit est négatif. Quand il n'existe plus de variables améliorantes, la procédure s'arrête et la dernière solution fractionnaire calculée est nécessairement optimale pour le programme linéaire complet.

On verra cette méthode à la prochaine séance. D'ici là :

Question 34 *Définir les termes de programmation linéaire suivants :* variables basiques, variables hors-base, variables duales ou multiplicateurs du simplexe, coût réduit.
Rappeler l'algorithme du simplexe (primal).

Problème 8 *On reprend les données du problème 1 : le transport hebdomadaire par avion cargo de pièces détachées, toutes uniques, depuis l'usine de fabrication des pièces, vers l'usine d'assemblage. On suppose que la capacité de production hebdomadaire de n'importe quel sous-ensemble de pièces est supérieure à la capacité de transport d'un avion. On suppose également que la commande de l'ensemble des pièces à acheminer est connue au temps initial (une semaine avant le premier vol). Le problème général posé est de planifier la production et le transport des pièces dans une durée minimale.*

Question 35 *Qualifier et modéliser ce problème.*

2. Pour des problèmes où l'ouverture d'un seul *container* (ici un rouleau initial) est très coûteux, ce gap est non négligeable.

Séance 5 : génération de colonnes

Problème 9 *Un éditeur musical prépare une nouvelle collection « best-of » des airs d'opéra et de ballets en disques compacts. Les morceaux ont déjà été sélectionnés. Connaissant la durée de chaque morceaux, il s'agit de déterminer le nombre minimal de CD audio, de 80 minutes chacun, permettant l'enregistrement de l'ensemble des morceaux.*

Question 36 modèles mathématiques.

- *qualifier ce problème et donner sa complexité ;*
- *proposer une formulation de programmation linéaire compacte ;*
- *donner une seconde formulation comme un programme linéaire de set-partitioning sur un ensemble de variables que l'on définira ;*
- *en relâchant les contraintes d'égalité en inégalité dans ce modèle, montrer que le problème se modélise également en un problème de set-covering ; montrer aussi que l'on peut relâcher la condition de binarité des variables en condition d'intégralité ;*
- ** on note (MP) la relaxation continue de ce dernier modèle ; montrer que dans toute solution optimale de (MP), chaque composante est au plus égale à 1 ;*
- *construire une solution réalisable du problème à partir d'une solution (fractionnaire) optimale de (MP).*

De nombreuses applications peuvent être formulées autant par un problème de set-partitioning que par un problème de set-covering (ou set-packing). La formulation de set-covering (ou set-packing) est souvent préférée alors car : (i) il est trivial de construire une solution réalisable entière, et (ii) la résolution de la relaxation continue est numériquement plus stable (moins d'erreurs sur les valeurs flottantes) et donc plus facile à résoudre.

Question 37 Dualité.

- *on considère un sous-ensemble $J' \subseteq J$ des colonnes de (MP) et on note (MP') le programme linéaire obtenu par projection sur J'. Associer à toute solution réalisable de (MP') une solution réalisable de (MP) de même coût.*
- *écrire le programme linéaire dual (MD') de (MP') ; comparer avec le programme linéaire dual (MD) de (MP) ; sous quelles conditions une solution optimale de (MD') est elle une solution optimale de (MD) ?*
- *soit $x' \in \mathbb{R}^{J'}$ et $\lambda' \in \mathbb{R}^I$ des solutions optimales de (MP') et (MD') respectivement (on suppose qu'elles existent) ; exprimer une relation arithmétique entre ces solutions (dualité forte) ;*
- *en déduire une condition nécessaire et suffisante portant sur λ' sous laquelle la solution associée à x' est une solution optimale de (MP).*

La génération de colonnes, tout comme l'algorithme du simplexe, repose sur le principe précédent. Ce sont des algorithmes itératifs de résolution des programmes linéaires, écrits sous forme canonique :

$$\min\{\, cx \mid Ax = b,\ x \in \mathbb{R}^J_+ \,\},\ \text{avec } c \in \mathbb{R}^J, b \in \mathbb{R}^I, A \in \mathbb{R}^{I \times J}$$

Ils construisent, de base en base, une suite de solutions réalisables de coût décroissant (pour un problème de minimisation). À chaque itération, la base courante est modifiée par introduction d'une variable (colonne) de coût réduit[3] strictement négatif. L'algorithme s'arrête – avec une solution optimale (réalisable dans le primal et dans le dual) – dès que les coûts réduits sont tous positifs.

Dans le cadre de la génération de colonnes, seule une partie des colonnes du programme linéaire est considérée à chaque instant, l'ensemble étant géré de manière implicite : dès que les colonnes actuellement présentes dans le programme sont toutes de coût réduit positif, on cherche alors s'il existe une colonne non générée dont le coût réduit est strictement négatif. Si une telle colonne existe, on l'ajoute au programme puis on réitère, sinon la dernière solution calculée est optimale.

Le programme linéaire restreint est appelé le *problème maître* ou *master problem*. Le problème dont une solution est une colonne de coût réduit strictement négatif est appelé le *sous-problème* ou *pricing problem*.

3. le coût réduit de la j-ème colonne (c_j, A_j) est donné par $\tilde{c}_j = c_j - \lambda' A_j$ où $\lambda' \in \mathbb{R}^I_+$ est le vecteur des multiplicateurs du simplexe (solution duale), $c_j \in \mathbb{R}$ le j-ème coefficient de la fonction objective et $A_j \in \mathbb{R}^I$ la j-ème colonne du système de contraintes.

L'algorithme de génération de colonnes est le suivant :

input : un sous-ensemble initial de colonnes $J' \subseteq J$ tel que le problème maître restreint correspondant (MP') soit réalisable.

step 1. on résoud (MP'), soit x' la solution optimale et λ' le vecteur des multiplicateurs du simplexe associé.

step 2. on cherche une nouvelle colonne (c_j, A_j), $j \in J$ telle que $\tilde{c}_j = c_j - \lambda' A_j < 0$

step 3. s'il existe une telle colonne, on l'ajoute à (MP'), **GOTO step 1**
sinon **STOP RETURN** x'

L'algorithme est souvent plus efficace si plusieurs colonnes de coûts réduits négatifs sont générées dans l'étape 2, ou encore si on génère les colonnes de coûts réduits les plus négatifs. Enfin, on peut également chercher à supprimer les colonnes de coûts réduits les plus positifs à intervalles réguliers pour éviter que le programme linéaire ne devienne trop large.

Question 38 *Appliquer la génération de colonnes au problème 9 :*
- *déterminer un sous-ensemble initial de colonnes suffisant (exhiber éventuellement une base initiale)*
- *formuler le sous-problème de génération de colonnes en un problème d'optimisation que l'on nommera.*

La solution retournée par la génération de colonnes est fractionnaire. On peut alors construire une solution entière approchée (voir question 36) ou bien résoudre le problème entier de manière optimale par branch-and-bound de sorte que, à chaque noeud de l'arbre de recherche, le programme linéaire soit résolu par génération de colonnes. On parle alors de *branch-and-price*.

On sait que brancher sur les variables du modèle linéaire décomposé n'est pas efficace (l'arbre est totalement asymétrique). L'autre alternative est de brancher sur des ensembles de variables, autrement dit sur des contraintes : *constraint branching*. À chaque branchement, on rajoute une contrainte dans le programme linéaire du premier noeud fils, et une contrainte opposée dans le programme linéaire du second noeud fils.

Question 39 *branch-and-price. *On applique un branch-and-price au problème 9 sur le modèle de set-partitionning (la génération de colonnes est identique au modèle de set-covering). On considère la stratégie de branchement suivante : on choisit une paire (i, i') de morceaux de musique à graver et on pose l'une des deux décisions alternatives : (1) soit i et i' sont gravés sur le même CD, (2) soit i et i' sont gravés sur des CD différents.*
- *Comment se modélisent chacune de ces deux contraintes dans le formalisme (MP) ? Ces contraintes modifient-elles la structure du programme maître ? du sous-problème de génération de colonnes ?*
- *Montrer que la recherche est complète si la stratégie de branchement est appliquée aux couples (i, i') tels que $0 < \sum_{j \in J | i,i' \in S_j} x'_j < 1$ où x' est une solution fractionnaire optimale du noeud courant et S_j est l'ensemble des morceaux formant le j-ème pattern. (Il s'agit de montrer que si x' n'est pas entière alors il existe nécessairement un tel couple).*
- *Améliorer le branch-and-price en calculant une borne supérieure à chaque noeud.*

Séance 6 : packing pour l'ordonnancement, la planification, le transport

Problème 10 *Une flotte de véhicules de livraison, tous identiques, doivent acheminer des colis de différents volumes stockés dans un dépôt central, aux clients qui les ont commandés (un seul colis par client). Minimiser la distance totale parcourue.*

Question 40
- ** qualifier ce problème et le formuler en un programme linéaire en variables binaires compact ;*
- *exprimer ce problème en un problème de set-partitionning et modéliser ;*
- *pour une résolution par génération de colonnes, écrire le dual du programme maître ; en déduire une formulation du sous-problème en un problème d'optimisation ;*
- *Pour chacune des contraintes supplémentaires suivantes, lequel du programme maître ou du sous-problème est modifié :*
  1. *chaque client est visité dans une fenêtre de temps donnée ;*
  2. *les véhicules sont de capacités différentes ;*
  3. *les colis sont livrés par ordre croissant de poids ;*

Question 41 *Discuter de la modélisation des problèmes suivants :*

Problème 11 Multi-Item Lot-Sizing.
Une usine fabrique un ensemble de produits classés par types sur un horizon de temps donné, par exemple 1 mois. L'usine fabrique chaque jour un lot de produits tous nécessairement de même type. La capacité journalière de l'usine, c'est à dire la quantité maximale de produits que peut fabriquer l'usine dans une journée, est connue et cette quantité dépend du type de produits et du jour. En revanche, l'usine peut stocker une quantité infinie de produits fabriqués. On connaît également la demande journalière : la quantité de produits de chaque type qui doivent sortir de l'usine chaque jour. Étant donnés des coûts journaliers de fabrication et de stockage d'une unité de produit ainsi qu'un coût journalier de fonctionnement de l'usine (tous ces coûts dépendent du type de produit et du jour), déterminer le plan de production de coût minimal.

Problème 12 Timetabling.
Dans un centre d'appels, il s'agit de minimiser le coût de la planification du personnel (les horaires des opérateurs) connaissant une prévision des charges de travail par période de temps sur un horizon donné. Le coût d'une planification est égal à la somme des coûts de sur-charge (nombre de personnes en trop fois un coût unitaire) et des coûts de sous-charge (nombre de personnes en défaut fois un coût unitaire) sur l'ensemble des périodes, plus la somme des coûts des emplois du temps de l'ensemble des opérateurs. L'emploi du temps de chaque opérateur doit répondre à des contraintes de faisabilité variées et complexes dûes à la réglementation du travail (par exemple : durée de travail journalière et hebdomadaire, durée minimale d'une pause ou d'un repas, nombre de week-end consécutifs travaillés, etc.) ou à des préférences et indisponibilités de l'opérateur (temps partiel, jours de congé). Le coût d'un emploi du temps d'un opérateur dépend du nombre et du type de périodes travaillées.

Problème 13 Crew Rostering.
Le problème de planification du personnel navigant d'une compagnie aérienne consiste à affecter des équipages aux vols programmés par la compagnie. À la différence du problème précédent, les employés sont classés par fonction (commandant, hôtesse, etc.) et l'équipage de chaque vol doit être formé d'un nombre précis d'employés de chaque type. Le coût de la planification ne tient pas donc compte ici des coûts de sous- et sur-charge. L'enchaînement des vols dans l'emploi du temps d'un employé doit être valide (temporellement et géographiquement) et répondre à de nombreuses règles légales et de sécurité sur les repos.

Problème 14 Resource-Constrained Project Scheduling.
Le problème d'ordonnancement de projets à contraintes de ressources (RCPSP) consiste à exécuter un ensemble de tâches au moyen de plusieurs ressources. Durant l'exécution d'une tâche, une certaine quantité de chacune des ressources est mobilisée. Les ressources étant renouvelables*, cette quantité est de nouveau disponible à la fin de l'exécution. Les ressources sont également* cumulatives *: elles peuvent exécuter plusieurs tâches à la fois dans la limite de leurs capacités. On suppose ici que toutes les tâches sont de même durée, déterminer l'ordonnancement de durée minimale.*