N° d'ordre : 29 /2010-M/MT

UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE

FACULTE DE MATHEMATIQUES

Mémoire présenté pour l'obtention du diplôme de Magister
en Mathématiques

Spécialité : Recherche Opérationnelle

par

ADLI Feriel

THEME

ETUDE DE LA SENSIBILITE DE LA SOLUTION OPTIMALE DU KNAPSACK PROBLEM

Soutenu publiquement, le 21/06/2010, devant le jury :

A. BERRACHEDI	Professeur à l'USTHB	Président
R. OUAFI	Maître de conférences (A) à l'USTHB	Directeur de mémoire
M. BOUDHAR	Professeur à l'USTHB	Examinateur
N.KAHOUL	Maître de conférences (B) à l'USTHB	Invitée

RÉSUMÉ. Les problèmes du type knapsack (sac à dos) sont parmi les problèmes NP-difficiles les plus traités de l'optimisation combinatoire. Dans un problème de type knapsack, on imagine un randonneur disposant d'un ensemble d'objets qu'il désir emporter dans sa randonnée. Chaque objet étant associé à un poids et une utilité (coût, profit). Le randonneur cherche à déterminer les objets à emporter de façon à maximiser l'utilité totale de son chargement tout en limitant le poids. Le but de ce mémoire est d'étudier deux problèmes de type knapsack avec des données incertaines (non fiables) ou imprécises (qui peuvent évoluer dans le temps). Plus précisément, nous analysons l'effet des perturbations que subissent certains paramètres de ces deux problèmes de manière à ce que la solution optimale reste inchangée. Dans un premier temps, nous faisons une analyse de sensibilité de la solution optimale du knapsack 0-1 (KP), cette analyse consiste à étudier la stabilité de la solution optimale face à une perturbation du profit du problème. Nous introduisons ensuite des bornes supérieures de la littérature dans le but d'améliorer les limites de l'intervalle de sensibilité du coefficient perturbé. Enfin, nous proposons un algorithme permettant de mesurer les bornes de l'intervalle de sensibilité du profit perturbé dans le problème BKP en le transformant en un problème KP binaire et en adaptant les résultats obtenus pour de ce dernier, en matière d'étude de la sensibilité, à celle du BKP. Des instances générées aléatoirement sont utilisées afin de vérifier les performances de l'algorithme proposé.

Mots clés : optimisation combinatoire, knapsack, bounded knapsack, étude de la sensibilité.

Remerciements

« Celui qui ne remercie les personnes ne remercie pas ALLAH ».

La première personne que je tiens remercier profondément est le Docteur Rachid OUAFI, Docteur à l'USTHB et directeur de cette thèse, pour avoir accepté de m'encadrer. Je le remercie pour sa rigueur scientifique et pour m'avoir toujours pousser à travailler et à aller de l'avant.

Je tiens également à exprimer ma profonde gratitude au Professeur Mhand HIFI, Professeur à l'université de Picardie Jules Verne, pour ses conseils et son aide précieuse.

Mes remerciements les plus sincères s'adressent au Professeur Abdelhafid BERRACHEDI, Professeur à l'USTHB, pour l'honneur qu'il me fait en président ce jury.

Le Professeur Mourad BOUDHAR, Professeur à l'USTHB, m'a fait l'honneur d'accepter le rôle d'examinateur de cette thèse, qu'il trouve ici la marque de mes sincères remerciements.

Pour m'avoir fait l'honneur d'accepter de participer à ce jury en tant qu'invitée, un grand merci s'adresse au Docteur Nawel KAHOUL, Docteur à l'USTHB, je la remercie également pour son aide et son encouragement.

Parallèlement à mon travail de thèse, j'ai découvert le plaisir d'enseigner. Je remercie les personnes avec qui j'ai eu le plaisir de travailler, et tout particulièrement Mr MASSIED, Mr AIT AKKACH et Mr AIT MESSAOUDENE.

Je n'oublie pas d'adresser mes vifs remerciements à mes collègues et à mes amies : A DAHMANI, M HAMRAT, M AIT ABDESSALAM, A BEN ANTAR, R KHEFFACHE et R MESSAOUDI.

Un merci très particulier à ma grande amie Leila OUZERI pour son aide et sa présence tout au long de mon travail.

Je ne peux qu'être infiniment reconnaissante envers mes parents pour leur soutien indescriptible, leur patience, leur confiance et leurs nombreux sacrifices.

Je suis reconnaissante envers mon mari Mohamed pour sa patience, son soutien et ses encouragements.

Un grand merci également à mes sœurs Fatima Zohra et Idia et à leurs maris respectifs Abdennacer et Sofiane qui m'ont épaulée, soutenue et aidée, durant mon travail, sans lassitude et avec beaucoup de patience. Je n'oublie pas, bien sûr, d'exprimer ma gratitude envers Mourad, Hania, Kamel, la petite Mimiche et aussi mon coussin Sid Ahmed.

Ces remerciements ne seraient pas complets si je n'y associe pas toutes les personnes ayant contribuer de près ou de loin à la réalisation de ce mémoire.

Enfin, s'il persiste dans ce mémoire des fautes, que j'ose espérer vénielles, il ne me reste plus qu'à solliciter l'indulgence du lecteur.

Sommaire

Chapitre 4

Analyse de la sensibilité pour le knapsack borné (BKP)

Dans le monde industriel, on fait de plus en plus appel à l'optimisation combinatoire (OC), discipline carrefour entre la programmation mathématique, les mathématiques discrètes et l'informatique. Ceci pour deux raisons majeures, la première étant le nombre important de problèmes pouvant se formuler sous forme d'un problème d'optimisation combinatoire. La seconde étant l'intérêt économique que représente cette discipline dans la résolution de ces problèmes.

Lorsqu'on utilise un modèle mathématique pour décrire la réalité, on est obligé de faire des approximations. Les différents problèmes d'optimisation combinatoire résolus dans la littérature sont plutôt bien structurés par rapport à ce qui se rencontre en pratique. La linéarité qu'on suppose lors de la modélisation de problèmes réels en programmes linéaires est, en fait, une approximation très significative. Aussi, notre connaissance du domaine dont fait partie le problème à modéliser peut nous amener à faire des approximations sur les valeurs des paramètres (temps, coûts, distances, poids, etc.). Une autre approximation à lieu lorsqu'on n'est pas sûr des données à mettre dans le modèle. D'autant plus que l'information est elle-même apte à changer si on est face à un environnement dynamique.

Tous ces facteurs sont une raison valable pour faire une analyse de la sensibilité du modèle formulé. Ainsi, ayant obtenu une solution optimale x^* du modèle simplifié, l'analyse de sensibilité de ce dernier est nécessaire pour déterminer la crédibilité (validité) de la solution optimale et pour tirer des conclusions sur la robustesse de cette solution. Cette analyse nous permet, entre autres, d'apporter une réponse à la question suivante : dans quel domaine (*intervalle de sensibilité*) peut varier chacun des coefficients de la fonction économique ou des contraintes pour que x^* reste une solution optimale.

C'est grâce à l'analyse de sensibilité que le décideur peut connaître le comportement de sa solution optimale face aux variations des paramètres du problème.

Dans un cadre plus théorique, l'analyse de sensibilité peut servir à *accélérer un schéma de résolution* dans lequel la connaissance de l'intervalle de sensibilité d'une solution évite des résolutions inutiles, ou dans lequel la résolution d'un problème peut être accélérée par *réoptimisation*, comme cela est le cas dans la résolution d'un problème de *paramétrisation*.

Dans notre travail, on se propose d'étudier deux problèmes de type knapsack avec des données approximatives (à cause de l'ambiguïté des objectifs). Plus précisément, nous analysons l'effet des perturbations que subissent les coefficients de la fonction objectif de manière à ce que la solution optimale reste inchangée.

En plus de ce chapitre introductif, ce mémoire comporte quatre autres chapitres. Ainsi, le chapitre 1 porte sur la problématique de l'optimisation combinatoire, la complexité algorithmique des problèmes d'optimisation combinatoire et la méthodologie de résolution des problèmes linéaires PL et linéaires en nombres entiers PLNE. Dans la dernière partie de ce chapitre, on décrit l'analyse de sensibilité des problèmes PL et PLNE.

Le chapitre 2 est consacré à la présentation des problèmes de type knapsack. Nous y décrivons le problème générique et en présentons les différentes variantes, cas particuliers, domaines applicatifs et propriétés fondamentales.

Le chapitre 3 décrit d'une façon détaillée le problème du knapsack 0-1 (KP) qui est la variante la plus traitée parmi les problèmes de type knapsack. Nous évoquons dans ce chapitre, en plus de la formulation mathématique et des méthodes de résolution, l'analyse de sensibilité du problème KP lors de la variation d'un profit arbitraire, en reprenant les principaux résultats de Hifi et al [26]. Ensuite, nous présentons différentes bornes supérieures de la littérature, pour le problème KP, dans le but d'apporter des améliorations sur les bornes de l'intervalle de sensibilité du profit perturbé. Des résultats expérimentaux sont présentés à partir de ces différentes bornes supérieures.

Dans le chapitre 4, nous traitons en détails le problème du knapsack borné (Bounded Knapsack Problem BKP) qui est une généralisation du problème KP, nous présentons une approche de résolution de BKP qui passe par sa transformation en un KP équivalent. Ensuite, nous analysons la sensibilité de BKP lorsqu'un coefficient profit est perturbé, à travers l'analyse de sensibilité du KP correspondant et nous développons une approche, pour le calcul d'un intervalle de sensibilité, inspirée de l'approche de Belgacem et Hifi [7], [8] pour un problème KP lorsque plusieurs coefficients profits sont perturbés. Des résultats expérimentaux sont également présentés afin d'évaluer l'efficacité de l'approche proposée.

1.1 Introduction

L'optimisation combinatoire (OC) est une discipline qui combine les techniques des mathématiques discrètes et de l'informatique dans le but de résoudre des problèmes d'optimisation dont la structure sous-jacente est de forme discrète.

Bien que les problèmes d'optimisation combinatoire soient généralement faciles à définir, ils sont, pour la majorité, difficiles à résoudre. En effet, la plupart de ces problèmes appartiennent à la classe des problèmes NP-difficiles et ne possèdent donc pas à ce jour de solution algorithmique efficace valable pour toutes les données.

1.2 Problématique

De manière générale, **un problème d'optimisation** [57] peut être défini par un espace *E* du problème (espace de recherche), qui contient un sous-*espace de solutions X* (aussi appelé région de faisabilité), $X \subseteq E$, ainsi qu'une *fonction objectif f*(·), $f : X \rightarrow IR$. L'espace du problème décrit en quelque sorte le monde, tandis que l'espace des solutions (souvent de taille exponentielle) contient l'ensemble des solutions *x*, $x \in X$, admissibles à la résolution du problème. Le rôle de la fonction objectif est de produire un critère d'évaluation permettant de comparer les solutions admissibles entre elles, et donc de déterminer, sur une échelle globale, monotone et unique, la qualité d'une solution.

Dans un problème d'optimisation ainsi défini, le but est de trouver une solution x^* de *X*, appelée *solution optimale*, qui *maximise*, ou *minimise*, la fonction objectif sur l'espace des solutions. Le problème peut donc s'écrire de la manière suivante [57]:

$$(P) \begin{cases} Max & f(x) \\ \text{s.c} & x \in X \end{cases} \qquad (1.1)$$

Sans perte de généralité, il est possible de se limiter au problème de maximisation. En effet, tout problème de minimisation peut être formulé comme un problème de maximisation avec :

$$Min\ f(x) = -\ Max\ \{-f(x)\} \qquad (1.2)$$

Notons qu'en général, il n'existe pas de critère explicite et efficace permettant d'affirmer qu'une solution x quelconque soit la solution optimale. Néanmoins, en comparant les solutions entre elles, toute solution optimale x^* doit satisfaire la condition suivante:

$$\forall x \in X \qquad f(x^*) \geq f(x) \qquad (1.3)$$

La condition (1.3) autorise l'existence de plusieurs solutions optimales distinctes. Elles sont de qualité égale, puisque leurs évaluations par $f(\cdot)$ sont identiques par définition. En général, un problème d'optimisation est considéré comme étant résolu lorsque le caractère optimal d'une solution a été démontré. Celui-ci est démontré lorsque l'algorithme de résolution permet d'établir la preuve que (1.3) est satisfaite. Deux conditions nécessaires doivent être satisfaites pour que l'existence d'une solution optimale soit établie :

a) X doit être non-vide, et

b) $f(\cdot)$ doit avoir une borne supérieure sur X.

L'optimisation combinatoire trouve des applications dans des domaines très variés. En effet, que l'on s'intéresse à l'optimisation d'un système de production, au design de réseaux de télécommunication, à la bioinformatique ou encore à l'extraction de connaissances, nous pouvons être confrontés à des problèmes d'optimisation combinatoire.

Sous la formulation générique de (1.1) sont englobés divers problèmes d'optimisation ayant des caractéristiques très variées. Selon les caractéristiques de la fonction objectif et de l'espace de problème, on discerne différentes classes de problèmes d'optimisation, chacune d'elles ayant donné lieu à différentes approches spécifiques utiles à leurs résolutions. Typiquement, les problèmes se distinguent les uns des autres en raison du caractère continu ou discret, linéaire ou non-linéaire de la fonction objectif. De même,

l'espace du problème, comme l'espace des solutions, peut être continu, convexe et/ou compact [57].

Traiter de l'ensemble de ces différentes classes de problèmes est hors du cadre de ce mémoire. Aussi, nous présenterons deux classes fondamentales de problèmes d'optimisation combinatoire, à savoir, *la programmation linéaire* et *la programmation linéaire en nombres entiers* à laquelle appartient le problème du sac à dos (objet de notre étude). Nous rappellerons également quelques notions de la théorie de la complexité, liée étroitement à la résolution de problèmes d'optimisation combinatoire.

1.3 Complexité

Très souvent, la taille et la structure de l'espace de solution *X*, ainsi que la nature de la fonction objectif rendent difficile, voire impossible, la recherche d'une solution optimale x^* en un temps de calcul raisonnable. Ce constat, concernant malheureusement la grande majorité des problèmes d'optimisation combinatoire définis sur la base d'applications réelles, a été synthétisé et formalisé au travers de la théorie de la complexité, développée au milieu des années 60, par Cobham, Edmonds, Cook et Karp en particulier. La plupart des problèmes « intéressants » en optimisation combinatoire étant NP-Complets (voir définition ci-après), il est illusoire d'espérer les résoudre rapidement tant que la conjecture « P ≠ NP » n'a pas été invalidée, si tenté qu'elle puisse l'être. Et même alors, il faudrait pouvoir en déduire un algorithme, ce qui n'est pas évident.

La théorie de la complexité [45] est une discipline qui a pour but de fournir un cadre formel à la difficulté des problèmes en informatique, notamment aux problèmes d'optimisation combinatoire. Elle concerne en particulier les problèmes pouvant être résolus afin de savoir s'ils peuvent l'être efficacement ou pas, en se basant sur une estimation des temps de calcul et des besoins en espace mémoire. Des développements récents au niveau de la théorie de la complexité ont permis de classifier les problèmes d'optimisation combinatoire selon leur difficulté face à un algorithme de résolution, en fonction du temps de calcul et du codage des données. Dans ce qui suit, nous allons voir les principales classes autours desquelles est axée la théorie de la complexité.

1.3.1 La complexité des problèmes d'optimisation

Lorsque l'on évoque le sujet des problèmes d'optimisation surgit inévitablement la question de leur complexité. Rappelons brièvement qu'en théorie de la complexité, on distingue entre les problèmes de décision et les problèmes d'optimisation. ***Un problème de décision*** *peut se formuler par une question du type « Existe-t-il une solution ayant telle caractéristique ? »,* et les réponses sont du type « Oui/Non ». ***Un problème d'optimisation****, quant à lui, donne lieu à une question telle que « Quelle est la solution maximale (minimale)? »*. Une réponse du type « Oui/Non » s'avère alors insuffisante, puisque, a priori, l'existence d'une solution maximale (minimale) n'est pas mise en cause. On cherche plutôt à connaître la solution optimale elle-même.

Il est néanmoins possible de formuler un problème d'optimisation par une suite de problèmes de décision. Ainsi, on peut se demander « Existe-t-il une solution de taille de au plus (au moins) k_0 ? ». En diminuant (augmentant) k_0 après chaque réponse positive, on se rapproche pas à pas de la solution optimale. Toutefois, la résolution de cette suite de problèmes de décision est au moins aussi complexe que la résolution du problème d'optimisation initial.

Notons que la théorie de la complexité traite des problèmes de décision et non des problèmes d'optimisation. Ainsi, les classes de complexité P, NP et NP-complet se rapportent aux problèmes de décision uniquement.

1.3.2 Classes de complexité

1.3.2.1 La classe P

Classe des problèmes de décision que l'on peut résoudre à l'aide d'un algorithme déterministe polynomial. Les problèmes de la classe P [18] sont dits « *faciles* ».

1.3.2.2 La classe NP

NP est une abréviation pour "*non-deterministic polynomial time*". La classe NP [18] renferme tous les problèmes de décision dont on peut associer à chacun d'eux un ensemble de solutions potentielles (de cardinal au pire exponentiel) tel qu'on puisse vérifier en un

temps polynomial si une solution potentielle satisfait la question posée. Le terme **non-déterministe** désigne un pouvoir qu'on incorpore à un algorithme pour qu'il puisse deviner la bonne solution.

Exemple de problème dans NP : le problème SAT (de satisfiabilité).

Soit $F(x_1, x_2, ..., x_n)$ une expression logique de *n* variables. Le problème SAT consiste à trouver des valeurs vrai ou faux pour chacune des variables x_k de telle manière à rendre vraie l'expression $F(x_1, x_2, ..., x_n)$.

Un algorithme non-déterministe résolvant SAT est comme suit :

Pour k = 1 **jusqu'à** n **faire**
$x_k = choix$ (vrai, faux) ;
Si $F(x_1, x_2, ..., x_n)$ = vrai **alors** F est satisfiable
sinon F n'est pas satisfiable.

Une autre manière, plus informelle pour définir la classe NP : c'est la classe des problèmes pour lesquels les seuls algorithmes connus de résolution sont ceux ayant une complexité exponentielle.

1.3.2.3 La classe NP-complet

La définition de la classe des problèmes NP-complets [18] constitue un des principaux résultats de la théorie de la complexité. De manière intuitive, un problème NP-complet est un problème aussi difficile qu'un ensemble de problèmes connus pour lesquels aucun algorithme de résolution efficace n'a été découvert. Un ***algorithme efficace*** est un algorithme de complexité polynomiale.

Un problème est NP-complet si, et seulement si, il appartient à NP et si tout problème de NP se réduit polynomialement à lui :

Un problème NP-complet n'est pas, à proprement parler, un problème d'optimisation combinatoire tel que nous l'avons défini dans la section précédente. Il s'agit plutôt de la variante simplifiée d'un tel problème, appelée problème de décision, à laquelle, rappelons-le, nous devons répondre par oui ou par non à une question du type : "Existe-t-il une

solution x admissible de valeur $f(x)$ supérieure ou égale à une constante k_0 donnée?". Un problème d'optimisation combinatoire est plus difficile à résoudre que le problème de décision auquel il est associé. Par la suite, nous utiliserons le terme NP-difficile pour désigner un problème d'optimisation combinatoire dont le problème de décision associé est NP-complet.

Nous ne savons pas si les problèmes NP-complets sont faciles ou compliqués mais nous savons qu'ils sont de difficulté égale. Les problèmes NP-complets sont souvent considérés comme les plus difficiles parmi les problèmes pour lesquels l'existence ou la non-existence d'un algorithme de résolution efficace n'a pu être établie. De nombreux chercheurs ont conjecturé qu'il n'existe aucun algorithme efficace pour résoudre un problème NP-complet. Toutefois, aucune preuve de cette conjecture n'a pu être établie à ce jour.

L'existence d'un premier problème NP-complet a été démontrée par Cook en 1971 [13]. Cook a montré en 1971 que tous les problèmes de la classe NP sont réductibles au problème de la satisfiabilité d'une expression logique quelconque. Autrement dit, si jamais quelqu'un venait à trouver un algorithme polynomial pour le problème de SAT, alors la question de savoir si P = NP est résolu de fait !

Une façon de montrer la complétude dans NP d'un problème est d'utiliser la notion de *réduction polynomiale* : étant donné un problème $\Pi \in$ NP et un problème Π^* NP-difficile, si Π^* se réduit polynomialement à Π (c'est-à-dire, en substance, s'il existe un algorithme polynomial qui transforme toute instance de Π^* en une instance de Π), alors Π est NP-complet (ou NP-difficile s'il n'est pas dans NP). En règle générale, on montre qu'un problème d'optimisation est NP-difficile en montrant que le problème de décision associé est NP-complet.

Dès lors, plusieurs chercheurs se sont intéressés au sujet de la NP-complétude de problèmes, et la liste des problèmes NP-complets et NP-difficiles n'a cessé de croître. L'importance de la classe des problèmes NP-complets (ou NP-difficiles) est liée au fait qu'il suffit désormais de montrer qu'un problème donné est aussi difficile qu'un problème NP-complet pour se convaincre que personne, à l'heure actuelle, n'est en mesure de proposer un algorithme efficace pour sa résolution.

1.4 Programmation linéaire

L'objectif de cette section est de présenter les notions et les outils de la programmation linéaire qui seront utilisées dans la suite de ce mémoire.

Historiquement, la programmation mathématique est née avec l'algorithme du Simplexe, mis au point dans les années 50 par Dantzig. Le principe de l'algorithme du Simplexe [58] est de déterminer une solution optimale en allant de sommet (du polyèdre représentant la région faisable) en sommet adjacent. Cet algorithme était le premier à résoudre les problèmes appartenant à la programmation linéaire. Un programme linéaire (PL) s'écrit de la façon suivante [57]:

$$(\text{PL}) \begin{cases} Maximiser\ z = Cx \\ Ax \le b \quad x \ge 0 \end{cases} \tag{1.4}$$

L'espace de problème E d'un programme linéaire est représenté par un vecteur de n variables réelles x_i, $x = (x_1, x_2, ..., x_n)$. Chaque x_i a comme domaine de définition IR^+, $x_i \in IR^+$. En conséquence, on a $E = IR_+^n$. La région de faisabilité X est alors un sous-espace de IR_+^n, $X \subseteq IR_+^n$. Elle est décrite par des contraintes qui sont des combinaisons linéaires entre variables. Ces contraintes sont représentées par la matrice A et le vecteur membre droit b. Avec p contraintes décrivant le problème, la matrice A est de dimension $p \times n$ et le vecteur membre droit b est de dimension p. La fonction objectif est, elle aussi, une combinaison linéaire de variables. Plus précisément, elle correspond au produit scalaire du vecteur de coûts C avec le vecteur des variables x. Une solution d'un programme linéaire est une instanciation des variables x_i de sorte qu'elle satisfait aux contraintes. La solution optimale est alors une instanciation des variables x_i qui satisfait les contraintes et maximise la fonction objectif.

En général, on distingue entre **un problème** et **une instance d'un problème**. La description d'un problème formalise ses structures génériques. Elle contient des paramètres. La description d'une instance de problème affecte des valeurs, souvent numériques, à ces paramètres. Ici la description d'un problème linéaire définit

essentiellement la structure de la matrice *A*. La description d'une instance de problème, elle, précise les valeurs numériques de *A*, *b* et *C*.

Un programme linéaire a une **interprétation géométrique** simple. L'ensemble *X* des solutions réalisables est par définition un polyèdre (un polytope si *X* est fini) de l'espace euclidien IR^n. Si *X* est non vide, c'est-à-dire si le problème (PL) possède une solution, (PL) est dit réalisable. Dans ce cas, toute solution optimale de (PL) se trouve nécessairement à un sommet (ou point extrême), ou éventuellement sur une face (intersection de *X* et d'un plan $\{ x \in IR^n / A_j x = b_j \}$), du polyèdre *X*. Pour le cas de maximisation, il s'agit des éléments de *X*, les plus loin de l'origine de IR^n, dans la direction induite par le vecteur *C*.

La programmation linéaire connaît d'importantes applications industrielles car l'optimisation de programmes linéaires est considérée comme *facile* (appartenant à la classe *P*), aujourd'hui, pour deux raisons. D'une part, il existe des variantes de l'algorithme du Simplexe qui sont remarquablement efficaces. Etant de complexité exponentielle, l'algorithme du Simplexe présente en pratique un comportement semblable à un algorithme de complexité polynomiale. D'autre part, les méthodes du type points intérieurs, pour la résolution de programmes linéaires, sont effectivement de complexité polynomiale.

1.5 Programmation linéaire en nombres entiers

De nombreux problèmes d'optimisation combinatoire, et en particulier les problèmes de type knapsack (sac à dos) qui nous intéressent ici, peuvent s'exprimer sous la forme de programmes linéaires, mais où les variables sont astreintes à prendre leurs valeurs dans un domaine discret.

Généralement, on restreint le programme linéaire aux seules solutions à coordonnées, toutes ou en partie, entières. On parle alors de programme linéaire en nombres entiers (PLNE) ou si les variables sont à valeurs booléennes 0 ou 1 on parle alors de programme linéaire en variables binaires (PLVB).

Un programme linéaire en nombres entiers s'écrit de la façon suivante [57]:

$$(\text{PLNE}) \begin{cases} Maximiser\ z = Cx \\ Ax \leq b \\ x \in IN^n \end{cases} \qquad (1.5)$$

Pour que les calculs restent dans l'ensemble des rationnels, on suppose généralement les données *A*, *b* et *C* à valeurs dans *IN*.

La condition d'intégralité ajoute un degré de complexité au problème et *la version décisionnelle d'un PLNE est un problème NP-complet*. Géométriquement, l'ensemble des solutions, dans ce cas, n'est plus un espace plein, mais un ensemble discret des points de IN^n contenus dans le polyèdre $\{ x \in IR^n / A\ x \leq b \}$.

1.6 Méthodes de résolution

Nous traitons ici des problèmes d'optimisation combinatoire NP-difficiles. Etant donné l'importance de ces problèmes, de nombreuses méthodes de résolution ont été développées en recherche opérationnelle et en intelligence artificielle. Ces méthodes peuvent être classées sommairement en deux grandes catégories : **les méthodes exactes** (complètes) qui garantissent la complétude de la résolution et **les méthodes approchées** (incomplètes) qui perdent en complétude pour gagner en efficacité. Sans avoir la prétention de présenter ici toutes les méthodes dédiées à l'optimisation combinatoire, nous nous contentons, dans la suite de ce chapitre, de survoler les principales approches en mettant l'accent sur les approches exactes.

1.6.1 Méthodes de résolution approchées

Les méthodes approchées constituent une alternative très intéressante aux méthodes exactes pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale. En effet, ces méthodes sont utilisées depuis longtemps par de nombreux praticiens. On peut citer, à titre d'exemples, les **méthodes gloutonnes** et l'**amélioration itérative.**

Depuis une dizaine d'années, des progrès importants ont été réalisés avec l'apparition d'une nouvelle génération de méthodes approchées puissantes et générales, souvent appelées métaheuristiques. Une **métaheuristique** est constituée d'un ensemble de concepts fondamentaux (par exemple, la liste tabou et les mécanismes d'intensification et de diversification pour la métaheuristique tabou), qui permettent d'aider à la conception de méthodes heuristiques pour un problème d'optimisation. Ainsi les métaheuristiques sont adaptables et applicables à une large classe de problèmes.

Les métaheuristiques sont représentées essentiellement par les *méthodes de voisinage* comme le *recuit simulé* et la *recherche tabou*, et les *algorithmes évolutifs* comme les *algorithmes génétiques* et les *stratégies d'évolution*. Grâce à ces métaheuristiques, on peut proposer aujourd'hui des solutions approchées pour des problèmes d'optimisation classiques de plus grande taille et pour de très nombreuses applications qu'il était impossible de traiter auparavant. On constate, depuis ces dernières années, que l'intérêt porté aux métaheuristiques augmente continuellement en recherche opérationnelle et en intelligence artificielle avec le développement d'ordinateurs de plus en plus performants.

1.6.2 Méthodes de résolution exactes

L'idée la plus simple pour résoudre un problème d'optimisation combinatoire consisterait à énumérer toutes les solutions de l'ensemble *X*, explicitement, et retenir parmi les solutions qui satisfont les contraintes, celle conduisant à la plus grande valeur de la fonction objectif. Mais pour des problèmes de grandes tailles, il faudrait des siècles à l'ordinateur le plus puissant pour effectuer une telle énumération. Les méthodes exactes sont un moyen de détourner cette énumération impossible.

Le principe essentiel d'une méthode exacte consiste généralement à *énumérer*, souvent de manière *implicite*, l'ensemble des solutions de l'espace de recherche. Pour améliorer l'énumération des solutions, une telle méthode dispose de *techniques* pour détecter le plus tôt possible les échecs (calculs de bornes) et *d'heuristiques spécifiques* pour orienter les différents choix. Parmi les méthodes exactes, on trouve la plupart des méthodes traditionnelles (développées depuis une trentaine d'années) telles les techniques de *séparation et évaluation progressive* (*Branch and Bound*) ou *les algorithmes avec retour arrière*. Les méthodes exactes permettent de trouver des solutions optimales pour des problèmes de taille raisonnable. Malgré les progrès réalisés (notamment en matière de

la programmation linéaire en nombres entiers), comme le temps de calcul nécessaire pour trouver une solution risque d'augmenter exponentiellement avec la taille du problème, les méthodes exactes rencontrent généralement des difficultés face aux applications de taille importante.

1.6.2.1 Procédures par séparation et évaluation

Les procédures par séparation et évaluation (*Branch and Bound*) sont le moyen le plus utilisé pour la résolution exacte des problèmes d'optimisation combinatoire, et en particulier pour la résolution des PLNE pour lesquels elles ont été initialement conçues.

On fait appel à ce type d'approches lorsque l'énumération de l'ensemble des éléments de X est impossible en raison de l'importance de son cardinal.

Comme dans toute recherche arborescente, il s'agit d'énumérer implicitement l'ensemble X en explorant progressivement les différentes parties de l'espace de recherche, et en tentant d'identifier et de supprimer les sous-espaces ne contenant pas de solution optimale. Cette méthode est donc représentée par un arbre, où chaque nœud correspond à une partie de l'espace de recherche, et la racine correspond à une partie incluant l'ensemble des solutions.

A chaque itération, à un nœud N donné, on tente de déterminer si l'espace E correspondant contient une solution optimale. Si on ne prouve pas le contraire et si E n'est pas réduit à un point, alors E est partitionné en sous-espaces, chacun étant représenté par un nœud fils de N ajouté à l'arbre. Dans tous les cas, le nœud N est marqué comme visité et la recherche se poursuit sur un autre nœud de l'arbre non déjà visité, tant qu'il en existe. A terme, soit il existe une feuille de l'arbre dont l'espace associé est réduit à une solution optimale du problème, soit le problème est irréalisable. Il y a donc trois composantes principales sur lesquelles repose toute procédure de séparation et évaluation, à savoir : la manière de séparer l'espace de recherche ou **schéma de branchement**, la **méthode d'évaluation** des nœuds pour tenter d'identifier ceux ne contenant pas de solution optimale et **l'ordre de sélection** des nœuds.

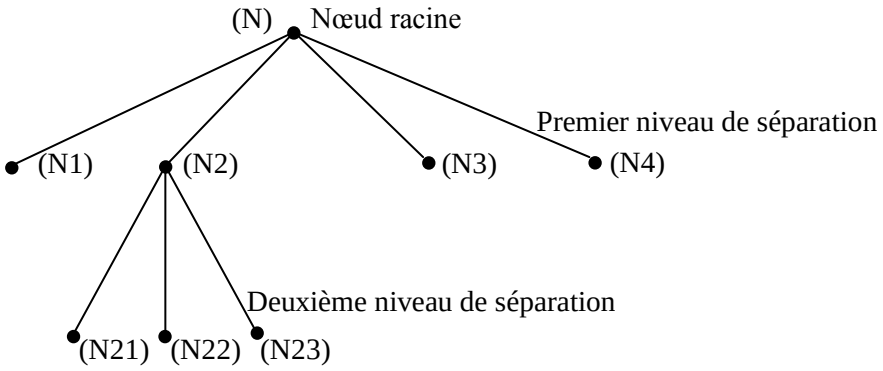

Fig. 1.1 - Arbre généré par décomposition du sous-problème initial.

- **Ordre de sélection**

L'ordre de sélection constitue la règle suivant laquelle est choisi le nœud devant être séparé parmi tous les nœuds pendants de l'arborescence. Il existe trois stratégies usuellement mises en œuvre :

Profondeur d'abord : quand un branchement est effectué, la recherche se poursuit sur l'un des nœuds créé. Si le nœud est une feuille (le domaine associé est réduit à un point ou bien ne contient pas de solution optimale), on effectue un backtracking en considérant un nœud frère, et ainsi de suite. Cette stratégie est la plus souvent utilisée car elle présente, entre autres, l'avantage majeur d'économiser l'espace mémoire et le temps de calcul en passant d'une itération à l'autre. En effet, le travail d'évaluation d'un domaine s'applique à tout sous domaine complété par la nouvelle information issue du branchement. Autrement dit, le travail effectué à un nœud peut être facilement réutilisé à l'itération suivante, pour l'évaluation d'un nœud fils (principe d'incrémentalité).

Largeur d'abord : l'arbre est parcouru de sorte qu'un nœud n'est évalué qu'après tous les nœuds de profondeur inférieure. A un branchement, tous les nœuds fils sont examinés l'un après l'autre, puis tous les nœuds du niveau suivant, etc.

Meilleur d'abord : le parcours de l'arbre est dépendant du résultat d'une estimation des nœuds restant à visiter. Selon cette estimation, on choisira alors d'examiner en premier, soit un nœud susceptible de contenir une solution optimale, soit un nœud menant rapidement à une meilleure solution réalisable. Les techniques d'estimation

sont similaires, voire identiques, aux techniques d'évaluation. Elles sont aussi utilisées dans les stratégies de profondeur ou largeur d'abord, au moment de choisir l'ordre de sélection des fils d'un nœud.

Pour cette dernière stratégie, l'ordre de sélection peut être étroitement lié à la façon dont les nœuds sont créés, c'est-à-dire selon le choix de séparation de l'espace.

- **Schémas de branchement**

En théorie, il existe une infinité de schémas de branchement possibles : l'espace de recherche peut être séparé en deux parties ou plus.

En pratique, pour la résolution des PLNE, on crée fréquemment un arbre binaire en séparant en deux l'intervalle des valeurs possibles d'une variable, selon un plan défini par $x_i = \alpha$.

Dans le cas d'un programme linéaire à variables binaires (cas du problème du sac à dos), cela revient à fixer la coordonnée x_i à 0 dans un nœud fils et à 1, dans l'autre. Un autre schéma généralise ce cas dans un PLNE où l'intervalle des valeurs de x_i est borné ($0 \le x_i \le \beta, \beta \in N$). On ajoute alors au nœud courant autant de branches qu'il y a de valeurs entières possibles pour x_i : $x_i = 0$, $x_i = 1$, ..., $x_i = \beta$.

Afin d'assurer l'optimalité de la solution fournie par le branch and bound, certaines règles doivent être respectées.

- *Règle* 1 : Aucune solution optimale ne doit être écartée lors d'une séparation. On utilise la règle suivante qui garantit la règle 1 :
- *Règle* 1' : La réunion des sous-ensembles obtenus lors d'une séparation doit être égale à l'ensemble séparé,
- *Règle* 2 : Le cardinal d'un sous-ensemble doit être inférieur à celui de son père.
- *Règle* 3 : Un sous-ensemble qui ne peut être séparé doit être sondable.

  Par ***un ensemble sondable*** en entends un ensemble qu'on ne peut plus séparer parce que :

  – on connaît la meilleure solution de l'ensemble,

  – on connaît une solution meilleure que toutes celles de l'ensemble,

  – on sait que l'ensemble ne contient aucune solution admissible.

- **Evaluations**

Le principe d'évaluation permet de diminuer l'espace de recherche. L'objectif étant d'essayer d'évaluer l'intérêt de l'exploration d'un sous-ensemble de l'arborescence.

Ainsi, afin de limiter la recherche, un nœud de l'arbre n'est exploré que si on le présume permettant de conduire à une solution optimale ou si une solution réalisable $\hat{x}$ est connue à cet instant, à une solution strictement meilleure que $\hat{x}$. A chaque nœud, on évalue ainsi la valeur de la meilleure solution contenue dans le sous-espace E associé au nœud (donc délimité par des plans) à savoir, ici pour notre problème de maximisation ; $\max\{Cx | Ax \leq b,\ x \in IN^n \cap \mathsf{E}\}$

Puisque ce PLNE est généralement aussi dur que le programme original, on se contente d'encadrer sa valeur optimale par une borne inférieure LB_{E} (évaluation par défaut) et par une borne supérieure UB_{E} (évaluation par excès).

La connaissance d'une borne inférieure du problème et d'une borne supérieure de la fonction objectif de chaque sous-problème permet de stopper l'exploration d'un sous-ensemble de solutions ne pouvant pas contenir de solutions candidates à l'optimalité : si pour un sous-problème la borne supérieure est plus petite que la borne inférieure du problème, l'exploration du sous-ensemble correspondant est inutile. D'autre part, lorsque le sous-ensemble est suffisamment "petit", on procède à une évaluation dite exacte : on résout alors le sous-problème correspondant.

Bien que la recherche arborescente s'applique à toute forme de résolution de problèmes d'optimisation combinatoire, les procédures par séparation évaluation sont particulièrement adaptées à la programmation linéaire en nombres entiers car on possède, dans ce cadre précis, des méthodes efficaces d'évaluation par excès en considérant, par exemple, une relaxation du PLNE.

Remarque :

- La qualité de la solution, dans les méthodes *Branch and Bound*, repose grandement sur le choix du schéma de branchement ainsi que la fonction d'évaluation.
- L'intérêt majeur de ces méthodes est que la convergence est garantie en temps fini, et qu'à toute solution réalisable intermédiaire est associée une borne inférieure et supérieure pour le coût optimal.

1.6.2.2 Décompositions et Relaxations

La décomposition présente une méthode de résolution des problèmes pour lesquels on peut isoler des variables qui, une fois fixées, simplifient le problème. Après reformulation, on se ramène à un programme contenant de nombreuses contraintes, qui ne sont pas identifiées a priori, mais générées progressivement par un algorithme de coupes. Par exemple, dans un programme en variables mixtes (entières et réelles), on considère généralement les variables entières comme difficiles.

La plupart des problèmes d'optimisation combinatoire présentent des structures particulières de sorte qu'ils se modélisent intuitivement en PLNE de grande taille, dans lesquels, on peut *reconnaître et isoler*, soit des contraintes, soit des variables plus « difficiles » que les autres. Les méthodes de décomposition s'appliquent plus particulièrement à ces programmes en les scindant de façon à résoudre des sous-programmes plus nombreux mais plus rapides à traiter.

La relaxation, quant à elle, est une technique de construction d'algorithmes classique en recherche opérationnelle. Lorsque l'on a affaire à un problème P qui est difficile à résoudre, on tente de relâcher quelques contraintes de P afin d'obtenir un problème $\overline{P}$ qui est plus simple, et pour lequel on dispose d'une technique de résolution efficace. $\overline{P}$ est alors la relaxation de P. On résout $\overline{P}$. Evidemment, la solution de $\overline{P}$ n'est en général pas une solution de P (ou du moins pas la solution optimale). Il existe alors deux possibilités : soit on essaye d'adapter la solution de $\overline{P}$ de sorte qu'elle devienne admissible pour P, soit on reformule P en réduisant sa région de faisabilité de manière à s'approcher de la solution désirée. Dans le premier cas on obtient une heuristique, dans le second cas la résolution peut être exacte au bout de plusieurs itérations.

Les techniques de relaxation appliquées à un problème de maximisation fournissent une évaluation *par excès* de l'optimum, en relâchant les contraintes du problème les plus difficiles à satisfaire, c'est-à-dire, en les supprimant ou bien en les prenant partiellement en compte. Le problème relaxé est plus facile à résoudre et sa valeur optimale constitue donc une borne pour le problème original.

On étudie, dans cette section, deux principales techniques de relaxation pour un programme linéaire en nombres entiers écrit sous la forme :

$$\text{(PLNE)} \begin{cases} \text{Maximiser } z = Cx \\ Ax \leq b \\ x \in IN^n \end{cases} \qquad (1.6)$$

avec :

- $A \in Z^p \times Z^n$, $b \in Z^p$, $C \in Z^n$,
- X l'ensemble des solutions réalisables,
- *conv*(X), son enveloppe convexe,
- P le polyèdre $\{ x \in IR^n / A\,x \leq b \}$
- x^* une solution optimale de (PLNE).

Pour simplifier, on suppose ici le problème réalisable (car si la relaxation est irréalisable alors il en est de même du problème initial), et parfois borné.

▪ Relaxation continue

Une technique simple et classique de relaxation consiste à ignorer toutes les contraintes d'intégralité. On obtient alors un programme linéaire en variables continues :

$$\text{C(PLNE)} \begin{cases} \text{Maximiser } z = Cx \\ x \in P \end{cases} \qquad (1.7)$$

appelé la relaxation continue, qui peut être résolu par les méthodes de séparation évaluation, simplexe, etc. Pour certains problèmes cependant, la borne supérieure ainsi obtenue (la valeur optimale de la relaxation continue, arrondie à la valeur entière inférieure), est parfois bien supérieure à la valeur optimale de (PLNE), et surtout, la solution optimale fractionnaire peut être très éloignée de la solution optimale entière. Ce comportement se rencontre plus souvent pour les programmes en variables binaires où la solution fractionnaire contient généralement de nombreuses coordonnées égales à 0,5.

La *faiblesse de la relaxation continue* est due à la faiblesse de la formulation du problème (c'est-à-dire la définition des variables), à savoir que P est large par rapport à *conv*(X) (sachons qu'une formulation est meilleure si P n'est pas trop grand par apport à *conv*(X)). Pour améliorer la relaxation continue, on peut ajouter au programme linéaire, comme nous allons le voir pour le calcul des bornes supérieures améliorées du problème

du knapsack, des inégalités valides qui coupent P. Cette technique permet ainsi de resserrer la formulation du problème, on résout alors :

$$\text{C(PLNE)} \begin{cases} Maximiser\ z_{C(PLNE)} = Cx \\ x \in P' \end{cases} \qquad (1.8)$$

avec $X \subseteq conv(X) \subseteq P' \subseteq P$

en prenant implicitement en compte les contraintes d'intégralité. Idéalement, on cherche à déterminer des inégalités définissant des facettes de *conv*(*X*), c'est-à-dire des plans supports de *conv*(*X*).

▪ Relaxation lagrangienne

La relaxation lagrangienne est une manipulation classique en optimisation sous contraintes. Bien qu'intimement liée à la convexité, elle est couramment utilisée dans le cas non convexe. Les méthodes de relaxation lagrangienne ont pour objectif de *donner une évaluation très serrée* du sous-ensemble, E, de l'ensemble des solutions réalisables, *diminuant de façon considérable le nombre de nœuds de l'arborescence* qu'il est nécessaire de visiter. En particulier, elle permet d'obtenir des bornes de la valeur optimale de certains problèmes d'optimisation combinatoire durs.

On suppose que le PLNE est décomposable, c'est-à-dire qu'on peut isoler les contraintes difficiles ($Ax = a$) des autres plus faciles ($Dx \le d$). Il peut donc s'écrire [57] :

$$\text{(PLNE)} \begin{cases} Maximiser\ z = Cx \\ Ax = a \\ x \in X \end{cases} \qquad (1.9)$$

avec $X = \left\{ x \in IN^n \middle| Dx \le d \right\}$.

L'idée ici est de relâcher les contraintes difficiles, non pas en les supprimant totalement, mais en les *dualisant*, autrement dit, en les prenant en compte dans la fonction objectif de sorte qu'elles pénalisent la valeur des solutions qui les violent. On définit ainsi, pour tout $\lambda \in IR^p$ ($\lambda > 0$), la relaxation lagrangienne de (PLNE) de paramètre λ :

$$L(\text{PLNE}, \lambda) \begin{cases} Maximiser\ z_\lambda = Cx + \lambda(a - Ax) \\ x \in X \end{cases} \qquad (1.10)$$

où λ est appelé le vecteur des multiplicateurs de Lagrange dans le cas de contraintes égalités, ou de Kuhn-Tuker dans le cas de contraintes inégalités, ou encore le vecteur des variables duales [57]. Il s'agit maintenant de chercher la meilleure borne supérieure de (PLNE), soit le λ qui fournit la meilleure relaxation lagrangienne, en résolvant le problème dual lagrangien suivant :

$$(DL)\begin{cases} Min\ z_{DL} = z_{\lambda} \\ \lambda \in IN^{p} \end{cases} \qquad (1.11)$$

On a ainsi la relation $z \leq z_{DL} \leq z_{\lambda}$, pour tout $\lambda \in IR^{p}$

1.6.3 La programmation dynamique

Il s'agit d'une méthode exacte pour résoudre des problèmes d'optimisation combinatoire. L'idée de la méthode consiste à plonger le problème proposé dans un problème plus général, dépendant de paramètres entiers, le problème initial correspondant à une valeur donnée des paramètres. On résout ensuite le problème général par récurrence sur les paramètres entiers. Son efficacité repose sur le principe d'optimalité de **Bellman** qui statue que « *toute sous-politique d'une politique optimale est optimale* » ou encore « *dans une séquence optimale de décisions, quel que soit la première décision prise, les décisions suivantes forment une sous-suite optimale, compte-tenu des résultats de la première décision* ». C'est-à-dire qu'on commence avec une petite portion du problème original, on trouve la solution optimale pour cette portion du problème. Ensuite on élargit progressivement le problème, en déterminant la nouvelle solution optimale à partir de la précédente.

Les problèmes traités par la programmation dynamique [47] ont une structure séquentielle. Un problème s'inscrit dans une structure séquentielle [47] lorsqu'il est possible de :

1. plonger ce dernier dans une famille de problèmes (Π_i) (i = 1, ..., n) de même nature ;

2. relier par une relation de récurrence les valeurs (x^*_i) (i = 1, ..., n) des solutions optimales de ces problèmes : $x^*_i = r(x^*_1, \ldots, x^*_{i-1})$.

L'algorithme de programmation dynamique (Algorithme 1.1) procède par décomposition et permet de trouver une solution optimale.

Algorithme 1.1 : Algorithme de programmation dynamique.

Entrée : Une famille de problèmes (Π_i) (i = 1, ..., n) ;
Une relation de récurrence $x_i^* = r\left(x_1^*,...,x_{i-1}^*\right)$, i = 1,..., n ;
Sortie : Une solution optimale x_n^* ;

1. Poser $x_1^* = 0$;
2. À l'étape i, la valeur x_i^* est déterminée grâce à la relation de récurrence ;
3. S'arrêter lorsqu'on obtient x_n^*. Sinon retourner au pas 2.

La programmation dynamique permet de résoudre un grand nombre de problèmes d'optimisation combinatoire en un temps pseudo polynomial (un **algorithme pseudo-polynomial** étant un algorithme dont la complexité est bornée par une fonction polynomiale de la *représentation unaire* des données du problème considéré). Notamment les problèmes de type knapsack, où les relations de dominance associées sont généralement efficaces et, l'ajout de tests supplémentaires permet de développer des algorithmes encore plus efficaces.

1.7 Analyse de sensibilité pour des problèmes linéaires

Lorsqu'on utilise un modèle mathématique pour décrire la réalité, on est obligé de faire des approximations. Les différents problèmes d'optimisation résolus dans la littérature sont plutôt bien structurés par rapport à ce qui se rencontre en pratique. La linéarité qu'on suppose lors de la modélisation de problèmes réels en programmes linaires est, en fait, une approximation très significative. Notre connaissance du domaine dont fait partie le problème à modéliser peut aussi nous amener à faire des approximations sur les valeurs des paramètres (temps, coûts, distances, poids, etc.). Une autre approximation à lieu lorsqu'on n'est pas sûr des données à mettre dans le modèle. D'autant plus que l'information est elle-même apte à changer si on est face à un environnement dynamique.

L'analyse de sensibilité [60] (ou l'analyse de stabilité) d'un problème d'optimisation combinatoire donné est *l'étude de la stabilité de la solution optimale de ce problème en fonction de l'évolution des données* (fonction objectif, contraintes), ou encore elle est

l'étude des possibilités de résolution efficace d'un problème obtenu à partir d'un autre en modifiant une partie des données (réoptimisation).

Ayant obtenu une solution optimale x^* d'un problème combinatoire donné, l'analyse de sensibilité de ce dernier nous permet d'apporter une réponse à la question suivante : dans quel domaine (*intervalle de sensibilité*) peut varier chacun des coefficients de la fonction objectif ou des contraintes pour que x^* reste une solution optimale.

Dans un cadre plus théorique, l'analyse de sensibilité peut servir à *accélérer un schéma de résolution* dans lequel la connaissance de l'intervalle de sensibilité d'une solution évite des résolutions inutiles, ou dans lequel la résolution d'un problème peut être accélérée par *réoptimisation*, comme cela est le cas dans la résolution d'un problème de *paramétrisation*. La **réoptimisation** consiste en l'étude théorique et algorithmique des possibilités de résoudre efficacement le problème perturbé en exploitant la résolution du problème initial.

L'analyse de sensibilité pour les problèmes d'optimisation combinatoire est apparue peu de temps après les méthodes de résolution exacte de ces problèmes. Pour les problèmes linéaires dont fait partie le knapsack, le premier algorithme a été proposé au début des années 70.

1.7.1 Calcul de l'intervalle de sensibilité

On considère le problème linéaire (PL) défini précédemment (1.4):

$$(\text{PL}) \begin{cases} \textit{Maximiser } z = Cx \\ Ax \le b \quad x \ge 0 \end{cases}$$

Soit x^* la solution optimale de (PL). Soit (PL′) le problème perturbé suivant :

$$(\text{PL}') \begin{cases} \textit{Maximiser } z = C'x \\ A'x \le b' \quad x \ge 0 \end{cases} \qquad (1.12)$$

On définit alors la région de stabilité $\mathcal{R}$ de la solution x^* par :

$$\mathcal{R} = \left\{ (C', A', b') \,\middle|\, C' \in IR^n, A' \in IR^p \times IR^n, b' \in IR^p, x^* \text{solution optimale de } (PL') \right\}$$

$\mathcal{R}$ représente l'ensemble de toutes les variations des données qui laissent la solution x^* optimale.

Définissons, sans perte de généralité, l'intervalle de sensibilité dans le cas de la perturbation du vecteur coût (perturbation du paramètre *C*), sachant que cette définition reste valable dans les deux autres cas (perturbation des paramètres *A* et/ou *b*). Il est également possible de chercher à déterminer un intervalle de sensibilité pour plusieurs coefficients à la fois.

Soit $k \in \{1,...,n\}$ l'indice de l'élément perturbé du vecteur $C = (c_j)_{j\in\{1,...,n\}}$,

$I_k = [\ I_k^-, I_k^+\ ]$ un intervalle de *IR* contenant l'élément perturbé c_k, et soit $C' \in IR^n$ tel que :

$$C' = (c'_j)_{j\in\{1,...,n\}} \text{ avec } \begin{cases} c'_j = c_j & \text{pour tout } j \in \{1,...,n\} \setminus \{k\} \\ c'_j \neq c_j & \text{pour } j = k \end{cases}$$

et $c'_k \in [\ I_k^-, I_k^+\ ]$

On dit que $[\ I_k^-, I_k^+\ ]$ est un intervalle de sensibilité, ou intervalle de tolérance, pour le coefficient c_k, si pour tout c'_k dans $[\ I_k^-, I_k^+\ ]$, x^* est une solution optimale de (PL′).

Etant donné une solution optimale, l'**intervalle de sensibilité** d'un coefficient est donc un intervalle de valeurs dans lequel cette solution reste optimale.

Remarque :

Si pour toute valeur $c'_k \notin [\ I_k^-, I_k^+\ ]$, x^* n'est plus optimale, alors $[\ I_k^-, I_k^+\ ]$ est un intervalle de sensibilité exact pour le coefficient c_k.

La plupart des algorithmes et approches existants pour l'analyse de sensibilité reposent sur des **conditions d'optimalité**.

En programmation linéaire, l'analyse de sensibilité découle de l'intervalle obtenu à partir de la condition nécessaire et suffisante d'optimalité basée sur le signe des coûts réduits des variables hors-base.

Or les conditions nécessaires et suffisantes d'optimalité n'existent pas pour bon nombre de problèmes d'optimisation combinatoire. Souvent c'est en exprimant finement, pour des problèmes spécifiques, des conditions suffisantes d'optimalité qu'on arrive à construire un algorithme pour l'analyse de sensibilité. Cinq types de conditions sont fréquemment rencontrés dans la littérature :

- Conditions triviales,
- conditions basées sur les K meilleures solutions,
- conditions relevant de méthodes spécifiques,
- conditions basées sur la programmation linéaire et
- conditions basées sur la dualité.

Nous allons voir par la suite comment il est possible, en s'appuyant sur les **conditions relevant de méthodes spécifiques**, de construire des algorithmes pour la détermination d'un intervalle de sensibilité.

Dans les méthodes d'optimisation, qu'elles soient classiques ou spécifiques, il existe des conditions particulières (suffisantes) qui garantissent l'optimalité de la solution obtenue. La méthode de branch and bound par exemple consiste à explorer l'espace des solutions du problème et à générer des solutions partielles en dirigeant l'exploration vers des sous-espaces prometteurs. Des tests d'élagage (optimalité, réalisabilité) permettent de détecter les solutions partielles non prometteuses. Ces tests constituent des conditions de non optimalité. Nous allons voir ici comment faire de l'analyse de sensibilité en exploitant les conditions de non optimalité avec une méthode de branch and bound.

Les premiers travaux sur l'analyse de sensibilité ont été basés sur les conditions induites par les méthodes d'exploration arborescente. A l'issue d'une méthode d'énumération implicite ou de branch and bound, on obtient un ensemble de solutions complètes (réalisables) qui contient la solution optimale, et un ensemble de solutions partielles (par arbitrage d'un sous-ensemble de variables) qui ont été élaguées par l'optimalité lors de l'énumération.

Supposons le problème (PLVB) suivant dans lequel les données $c_j, (j = 1,...,n)$, sont positives.

$$\text{(PLVB)} \begin{cases} \max Cx \\ s.c \quad x \in X \subset \{0,1\}^n \end{cases} \tag{1.13}$$

Après résolution de (PLVB), on obtient donc une arborescence associée à deux ensembles : E_c qui contient les solutions complètes et E_p contenant les solutions partielles.

Le principe de l'analyse de sensibilité consiste ici à exploiter les informations liées à cette arborescence pour déterminer l'intervalle de sensibilité d'une donnée de (PLVB).

Pour un coefficient $c_k, (k = 1,...,n)$ donné, on cherche un intervalle de valeurs $[\ I_k^-, I_k^+\ ]$ tel que la solution optimale x^* de (PLVB) est aussi solution optimale du problème (PLVB′) obtenu à partir de (PLVB) en ne modifiant que la valeur de c_k dans $[\ I_k^-, I_k^+\ ]$.

$$\text{(PLVB')} \begin{cases} \max\ C'x \\ s.c \quad x \in X \subset \{0,1\}^n \end{cases} \qquad \text{où } c'_j = c_j,\ j \in \{1,...,n\} \setminus \{k\} \qquad (1.14)$$

Pour déterminer cet intervalle de sensibilité, on distingue deux cas :

(i) $x_k^* = 0$ et,

(ii) $x_k^* = 1$.

Dans chacun de ces deux cas on obtient une borne évidente de l'intervalle de sensibilité. En effet les données étant supposées positives, dans le premier cas $I_k^- = 1$, dans le second $I_k^+ = +\infty$.

Le calcul de la deuxième borne de l'intervalle de sensibilité est présenté plus en détails, pour un problème de type knapsack, au Chapitre 3 (voir Section 3.6).

1.7.2 Complexité

La question de la complexité du problème de la détermination de l'intervalle de sensibilité pour les problèmes de programmation linéaire en 0-1 et pour les problèmes d'optimisation combinatoire a été traitée par Van Hoesel et Wagelmanns [62] et Ramaswamy et Chakravarti [55], respectivement. Dans [62], les auteurs ont montré, en considérant un problème linéaire de maximisation à variables en 0-1, que l'existence d'un algorithme polynomial pour la détermination des intervalles de sensibilité des coefficients c_k implique l'existence d'un algorithme polynomial pour résoudre le problème lui-même.

Par conséquent, à moins que P=NP, l'analyse de la sensibilité d'un problème NP-difficile est NP-difficile.

2.1 Introduction

Les problèmes du type knapsack (sac à dos) sont parmi les problèmes NP-difficiles les plus traités de l'optimisation combinatoire. Ils sont intensivement étudiés depuis les travaux pionniers de Dantzig dans les années 50 à cause de leur application immédiate en industrie et en finances (transport, logistique, télécommunication, gestion des spots publicitaires, fiabilité,...), mais aussi de façon plus prononcée pour leur intérêt théorique puisqu'ils surviennent dans différents problèmes de la programmation entière.

2.2 Description et formulation

Dans le problème knapsack [57], on imagine un randonneur disposant de n objets qu'il désir emporter dans sa randonnée. Chaque objet ayant un poids w_j et une utilité p_j ($j = 1,...,n$), le randonneur cherche à maximiser l'utilité totale de son chargement tout en limitant le poids.

Ce problème, ainsi posé, peut être formulé mathématiquement en associant à chaque objet une variable binaire x_j ($j = 1,...,n$) telle que :

$$x_j = \begin{cases} 1 & \text{si l'objet } j \text{ est séléctionné;} \\ 0 & \text{sinon.} \end{cases}$$

En supposant que c représente la capacité du sac, il s'agit alors de déterminer parmi les vecteurs binaires $x = (x_1, x_2, ..., x_n)$ satisfaisant à la contrainte $\sum_{j=1}^{n} w_j x_j \le c$, celui maximisant la fonction objectif $\sum_{j=1}^{n} p_j x_j$.

2.3 Applications

Le problème knapsack a de nombreuses applications aussi bien en théorie qu'en pratique.

D'un point de vue théorique, sa structure particulièrement simple favorise l'exploitation de certaines de ses propriétés dans le but de le rendre encore plus facile à résoudre. Le knapsack peut être considéré comme un problème à part ou comme un sous-problème ou une transformation d'autres problèmes plus complexes d'optimisation combinatoire et donc tout algorithme de résolution du knapsack peut être exploité pour la résolution du problème transformé.

En pratique, et contrairement à son nom, le knapsack a des applications assez éloignées du domaine de remplissage : exemple dans des systèmes d'aide à la spéculation boursière, pour maximiser les retours d'un capital à placer sur des actions.

Une autre application, dans la découpe de matériaux (**Gilmore et Gomory** 1966 [23]) pour minimiser les chutes lors de la découpe de sections de longueurs diverses dans des barres en fer par exemple.

Dans le domaine de la restauration, choisir par exemple un repas dans un menu de façon à ne pas dépasser la valeur calorique prescrite (**Balintify et *al.*** 1978).

Hormis ces applications qui peuvent paraître simples, le problème du knapsack peut intervenir dans des problèmes de plus grande envergure comme le chargement de conteneurs (**Bellmann et Dreyfus** 1962 [6]), le contrôle budgétaire (**Cord** 1964, **Kaplan** 1966 **[34]**, **Hansmann** 1961), la cryptographie (**Diff et Hellmann** 1976) etc.

Quant aux applications théoriques, il existe des problèmes qui peuvent être transformés en knapsack et qui sont résolus en résolvant le knapsack, comme par exemple les problèmes de la programmation entière transformés en problème de type knapsack 0-1 par la transformation de l'ensemble des contraintes en une contrainte globale (**Salkin** 1975, **Syslo**, **Deo et Kowalik** 1983).

Il arrive également que les problèmes knapsack soient des sous-problèmes dans des problèmes plus complexes ; on peut citer, dans cette catégorie, le problème de tournées de véhicules ou encore le problème d'affectation généralisé qui, dans leur processus de résolution, font intervenir le knapsack 0-1.

2.4 Différents types de problèmes

Sous le nom générique de « knapsack », on peut trouver une famille de problèmes qui diffèrent en fonction de la distribution des objets et/ou du (des) sac(s). Dans ce qui suit,

nous allons détailler quelques-uns des problèmes knapsack les plus traités dans la littérature.

2.4.1 Le 0-1 knapsack

Dans le 0-1 knapsack (*0-1 Knapsack problem*) ou sac à dos binaire (*Binary Knapsack Problem*), noté KP, on suppose donnés un ensemble de *n* objets et un sac, tel que :

p_j est le profit de l'objet *j*,

w_j est le poids de l'objet *j*, et

c est la capacité du sac.

Le but est de sélectionner un sous ensemble d'objets de manière à maximiser $\sum_{j=1}^{n} p_j x_j$ sous la contrainte $\sum_{j=1}^{n} w_j x_j \le c$ où :

$$x_j = \begin{cases} 1 & \text{si l'objet } j \text{ est séléctionné ;} \\ 0 & \text{sinon.} \end{cases}$$

En somme, il s'agit de résoudre le problème linéaire en nombres entiers suivant :

$$(\text{KP}) \begin{cases} \text{maximiser } z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} \quad \sum_{j=1}^{n} w_j x_j \le c \\ \qquad x_j \in \{0,1\} \quad \text{pour } j = 1,...,n \end{cases} \tag{2.1}$$

où p_j , w_j $(j = 1, ..., n)$ et c sont des entiers positifs, $\sum_{j=1}^{n} w_j > c$, et $w_j \le c$.

KP est considéré comme étant l'un des problèmes les plus étudiés en programmation discrète en raison de son importance théorique caractérisée par le fait qu'il soit :

(a) le problème le plus simple en programmation linéaire entière (une seule contrainte),

(b) un sous-problème ou une relaxation de problèmes plus complexes,

(c) une formulation pour de très nombreuses applications industrielles.

2.4.2 Le knapsack borné

Le knapsack borné (*Bounded Knapsack Problem*), noté BKP, peut être vu comme une généralisation du knapsack 0-1, dans lequel b_j ($j = 1,...,n$) objets de poids w_j et de profits p_j sont disponibles, avec $b_j \leq c / w_j$. BKP est formulé par :

$$(\text{BKP}) \begin{cases} \text{maximiser} & z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} & \sum_{j=1}^{n} w_j x_j \leq c \\ & 0 \leq x_j \leq b_j \qquad \text{pour } j = 1,...,n \\ & x_j \textit{ entier et } x_j \geq 0 \qquad \text{pour } j = 1,...,n \end{cases} \tag{2.2}$$

2.4.3 Le knapsack non borné

Le knapsack non borné (*Unbounded Knapsack Problem*) est une généralisation du knapsack borné tel que les objets sont disponibles en nombres infinis ($b_j = +\infty$). Le problème ainsi obtenu est noté UKP et est formulé comme suit :

$$(\text{UKP}) \begin{cases} \text{maximiser} & z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \textit{ entier}, x_j \geq 0 \qquad \text{pour } j = 1,...,n \end{cases} \tag{2.3}$$

Dans ce problème, chaque variable x_j *est bornée par la capacité c*, puisque le poids de chaque objet est égal à au moins 1.

UKP peut être considéré comme un BKP avec $b_j = \lfloor c / w_j \rfloor$.

Il intervient dans les problèmes de chargement de conteneurs (**Bellmann et Dreyfus** 1962 [6]).

2.4.4 Problème de la somme d'un sous-ensemble

Supposons que dans le problème KP le profit p_j est égal au poids w_j, pour chaque objet j ($j = 1,...,n$), on obtient alors le problème de la **somme d'un sous-ensemble** (*Subset Sum Problem SSP*), formulé comme suit :

$$(\text{SSP}) \begin{cases} \text{maximiser} \quad z(x) = \sum_{j=1}^{n} w_j x_j \\ \text{s.c} \qquad \sum_{j=1}^{n} w_j x_j \le c \\ \qquad\quad x_j \in \{0,1\} \qquad \text{pour} \quad j = 1,...,n \end{cases} \tag{2.4}$$

Comme son nom l'indique, SSP est un problème où il s'agit de choisir un sous ensemble parmi les poids $w_1,...,w_n$ de telle sorte que leur somme soit aussi grande que possible sans dépasser la capacité c.

2.4.5 Problème du Change-Making

Imaginons un caissier devant rendre une monnaie d'un montant *c* en utilisant des pièces de montants $w_1,...,w_n$. Quel est le nombre maximum de pièces qu'il va utiliser sachant que $b_j, j = 1,...,n,$ pièces de montant w_j sont disponibles. Le problème du change-making (CMP) est alors définit comme suit :

$$(\text{CMP}) \begin{cases} \text{maximiser} \quad z(x) = \sum_{j=1}^{n} x_j \\ \text{s.c} \qquad \sum_{j=1}^{n} w_j x_j = c \\ \qquad\quad 0 \le x_j \le b_j \quad \text{pour} \quad j = 1,...,n \\ \qquad\quad x_j \text{ entier} \quad \text{pour} \quad j = 1,...,n \end{cases} \tag{2.5}$$

où w_j est la valeur de la pièce *j*.

CMP est considéré comme un cas particulier du problème de sac à dos borné où $p_j = 1$ et où l'égalité est imposée dans la contrainte de capacité.

2.4.6 Le 0-1 knapsack multiple

Le 0-1 knapsack multiple (*0-1 Multiple Knapsack Problem* MKP) est une autre généralisation de KP, avec *m* sacs disponibles ayant chacun une capacité $c_i, i = 1,...,m$. Il s'agit, dans ce cas, de *choisir m sous-ensembles disjoints d'objets* tels que le profit total des objets sélectionnés soit maximum. De plus, chaque sous-ensemble est affecté à un sac différent ayant une capacité $c_i, i = 1,...,m$, supérieure au poids total des objets de ce sous-ensemble.

On introduit la variable binaire x_{ij} telle que :

$$x_{ij} = \begin{cases} 1 & \text{si l'objet } j \text{ du sac } i \text{ est séléctionné} \\ 0 & \text{sinon} \end{cases}$$

Le problème obtenu est noté MKP et se formule alors :

$$(\text{MKP}) \begin{cases} \text{maximiser } z(x) = \sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij} \\ \text{s.c} \quad \sum_{j=1}^{n} w_j x_{ij} \le c_i \quad \text{pour } i = 1,...,m \\ \quad\quad \sum_{j=1}^{m} x_{ij} \le 1 \quad \text{pour } j = 1,...,n \\ \quad\quad x_{ij} \in \{0,1\} \quad \text{pour } i = 1,...,m,\ j = 1,...,n \end{cases} \tag{2.6}$$

2.4.7 Problème de chargement de conteneur

Le problème, très connu, de chargement de conteneur (Bin-Packing Problem BPP) ne fait habituellement pas partie des problèmes de type knapsack mais il peut être considéré comme un problème SSP multiple où tous les conteneurs ont la même capacité *c*, tous les objets doivent être sélectionnés et, le but est de minimiser le nombre de conteneurs utilisés.

Supposant *m* une borne supérieure pour le nombre de conteneurs, on introduit *m* variables binaires $y_i, i = 1,...,m$ telles que :

$$y_i = \begin{cases} 0 & \text{si le conteneur i est utilisé} \\ 1 & \text{sinon} \end{cases}$$

On peut formuler le problème comme suit :

$$
(BPP)\ \begin{cases}
\text{maximiser } z(x) = \sum_{i=1}^{m} y_i & \\
\text{s.c} \quad \sum_{j=1}^{n} w_j x_{ij} \le c(1-y_i) & i = 1,...,m \\
\sum_{j=1}^{m} x_{ij} = 1 & j = 1,...,n \\
y_i \in \{0,1\} & i = 1,...,m \\
x_{ij} \in \{0,1\} & i = 1,...,m,\ j = 1,...,n
\end{cases} \qquad (2.7)
$$

2.5 Complexité

Nous allons, dans cette section, voir que les problèmes de type knapsack cités ci-dessus sont NP-difficiles.

Pour ce faire, nous allons montrer pour chaque problème P que le problème de décision associé D(P) est NP-complet ou que P lui-même est la généralisation d'un problème NP-difficile.

Soit le problème de décision suivant :

PARTITION : Etant donnés n entiers positifs $w_1,...,w_n$. Existe-t-il un sous-ensemble $S \subseteq N = \{1,...,n\}$ tel que $\sum_{j \in S} w_j = \sum_{j \in N \setminus S} w_j$?

PARTITION est connu pour être un problème NP-complet.

- ***Le problème de la somme d'un sous-ensemble (SSP) est un problème NP-difficile.***

  Preuve [42]

  Pour montrer que SSP est NP-difficile, il suffit de montrer que le problème de décision associé est NP-complet.

  Considérons le problème D(SSP), problème de décision associé au problème SSP, défini comme suit :

D(SSP) : Etant donnés (n + 2) entiers positifs $w_1,...,w_n,c$ et a. Existe-t-il un sous-ensemble $S \subseteq N = \{1,...,n\}$ tel que $\sum_{j \in S} w_j \leq c$ et $\sum_{j \in S} w_j \geq a$?

Toute instance I de PARTITION peut être polynomialement transformée en une instance I′ de D(SSP), en posant $c = a = \sum_{j \in N} w_j / 2$ (la réponse à I est « oui » si et seulement si la réponse à I′ est « oui »). ■

- ***Le 0-1 knapsack (KP) est NP-difficile.***

  Preuve [42]

  Comme le problème KP est une généralisation du problème SSP (dans le cas où $p_j = w_j, \forall j \in \{1,2,...,n\}$), et que SSP est NP-difficile alors KP l'est aussi. ■

- ***Le knapsack borné (BKP) est NP-difficile.***

  Preuve [42]

  BKP est une généralisation du problème KP (où $b_j = 1, \forall j \in \{1,2,...,n\}$). Comme KP est NP-difficile alors il en est de même pour BKP. ■

- ***Le knapsack non borné (UKP) est NP-difficile.***

  Preuve [42]

  Comme UKP est une généralisation du problème BKP alors il NP-difficile. ■

- ***Le problème du change-making CMP est NP-difficile.***

  Preuve [42]

  On montre la NP-complétude dans le cas particulier où $b_j = 1$ pour tout *j*.

  Considérons le problème de décision associé à CMP, défini par :

D(CMP) : Etant donné (n + 2) entiers positifs $w_1,...,w_n,c$ et a, existe-t-il un sous-ensemble $S \subseteq N = \{1,...,n\}$ tel que $\sum_{j \in S} w_j \le c$ et $|S| \ge a$?

Toute instance I de PARTITION peut être polynomialement transformée en une instance I′ de D(CMP), en posant $c = \sum_{j \in N} w_j / 2$ et a = 1. ■

Remarque :

- Les problèmes KP, BKP, UKP, SSP et CMP sont formulés en problèmes linéaires en nombres entiers à une seule contrainte fonctionnelle appelée contrainte de capacité.
- Ces problèmes sont NP-complets, et donc ne peuvent être résolus par un algorithme polynomial en *n*, à moins que *P = NP*. Toutefois, grâce à la programmation dynamique, ils admettent des algorithmes pseudo-polynomiaux (i.e. polynomiaux en *n* et *c*) de complexité $O(nc)$.
- Alors que pour les deux problèmes MKP et BPP, il n'existe pas d'algorithme pseud-polynomial, à moins que *P = NP*, car ils sont **NP-complets au sens fort** (un problème $\Pi \in$ NP-complet est NP-complets au sens fort ssi il existe un polynôme p tel que Π_p est NP-complet). Nous allons en voir la preuve ci-après.

Soit le problème de décision suivant :

3-PARTITION : Etant donné $n = 3m$ entiers positifs $w_1,...,w_n$ tels que $\sum_{j=1}^{n} w_j / m = B$ et $B/4 < w_j < B/2$ pour $j = 1,...,n$. Existe-t-il une partition de $N = \{1,...,n\}$ en *m* sous-ensembles $S_1,...,S_m$ tels que $\sum_{j \in S_i} w_j = B$ pour $i = 1,...,m$?

(Notons que chaque S_i contient exactement 3 éléments de *N*.)

3-PARTITION est le premier problème découvert comme étant NP-difficile au sens fort .

- ***Le* 0-1 *knapsack multiple (MKP) est NP-difficile au sens fort.***

  Preuve [42]

  Considérons le problème de décision associé à MKP, donné par :

D(MKP) : Etant donné $(2n+p+1)$ entiers positifs $p_1,...,p_n$, $w_1,...,w_n$, $c_1,...,c_p$ et a. Existe-t-il p sous-ensembles disjoints $S_1,...,S_p$ de $N=\{1,...,n\}$ tels que $\sum_{j\in S_i} w_j \le c_i$ pour $i=1,...,p$ et $\sum_{1\le j\le p}\sum_{j\in S_i} p_j \ge a$?

Toute instance I de 3-PARTITION peut être pseudo-polynomialement transformée en une instance équivalente I′ de D(MKP), en posant $c_i = B$ pour $i=1,...,p$, $p_j = 1$ pour $j=1,...,n$ et $a = n$ (ce qui implique que $\bigcup_{i=1}^{p} S_i = N$ dans toute instance « oui »). ■

- ***Le problème bin-paking (BPP) est NP-difficile au sens fort.***

  Preuve [42]

  Soit

  D(BPP) : Etant donné $(n+2)$ entiers positifs $w_1,...,w_n$, c et a. Existe-t-il une partition de $N=\{1,...,n\}$ en a sous-ensembles disjoints $S_1,...,S_a$ tels que $\sum_{j\in S_i} w_j \le c$ pour $i=1,...,n$?

  Toute instance I de 3-PARTITION peut être pseudo-polynomialement transformée en une instance équivalente I′ de D(BPP), en mettant $c = B$ et $a = m$. ■

2.6 Approches de résolution

La communauté scientifique s'est intéressée au problème du knapsack pour la première fois dans les années 50. Depuis, un grand nombre d'algorithmes et de techniques a été présenté pour trouver des solutions exactes et approchées : les méthodes de relaxations, le branch and bound, la programmation dynamique, les méthodes heuristiques, les algorithmes parallèles, etc.

Le premier algorithme de Programmation Dynamique est proposé par **Bellmann** [5] en 1957. Ensuite une borne supérieure de la valeur optimale du problème par **Dantzig** [14] (1957).

Les années 60 ont connu l'amélioration de la Programmation Dynamique par **Gilmore et Gomory** [23], la première approche par Branch and Bound grâce à **Kolesar** [36] en 1967.

Dans les années 70, l'algorithme de Branch and Bound est revu par **Horowitz et Sahni** [28] (1974). **Ingargiola et Korsh** [30] présentent en 1973 la première procédure de réduction du nombre de variables. **Sahni** [56] proposent le premier algorithme d'approximation en temps polynomial en 1975, une autre approximation en temps polynomial est présentée par **Ibarra et Kim** la même année. Grâce à **Martello et Toth** [39], on a en 1977 la première borne supérieure dominant celle obtenue par la relaxation continue.

Les années 80 ont connu des résultats sur la solution des problèmes de très grande taille où le classement de variables prend la plupart du temps, ces années ont connu aussi des travaux portant sur le noyau du problème, travaux réalisés par **Balas et Zemel** [2] en 1980 et **Plateau et Elkihel** [50] en 1985.

Les années 90 ont vu apparaître les algorithmes parallèles mis au point en 1992 par **Loots et Smith**. Ensuite, l'approche par réseau de neurones grâce à **Ohlsson, Peterson et Soderberg** en 1993 et l'approche par des techniques d'Intelligence Artificielle par **Ko** (1993). En 1994, **Penn et Hasson** ont fait une résolution du problème par échantillonnage.

2.7 Propriétés fondamentales

Le problème knapsack est un problème simple et structuré qui possède des propriétés essentielles à l'élaboration d'algorithmes de résolution. C'est d'ailleurs cette structuration qui fait que certaines instances complexes sont résolues en fractions de seconde.

▪ Résolution du problème continu

Une des propriétés importantes sinon la plus importante pour le knapsack, est la facilité de résolution dans le cas continu, où les contraintes telles que les variables $x_j \in \{0,...,b_j\}$ relâchées dans l'intervalle $[0, b_j]$ ($0 \le x_j \le b_j$), sont plus faciles à résoudre.

En 1957, Dantzig a montré une manière élégante qui permet de trouver une solution du knapsack 0-1, dans laquelle les objets sont classés dans un ordre décroissant par rapport au rapport profit/poids,

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq .... \geq \frac{p_n}{w_n},$$

En utilisant un algorithme glouton pour déterminer l'élément dit critique, noté *s* (premier élément ne pouvant aller dans le sac sachant que les éléments sont sélectionnés pour aller dans le sac selon l'ordre décroissant du rapport profit/poids), la solution optimale est alors obtenue en sélectionnant tous les objets de 1 jusqu'à *s* – 1, plus une fraction de l'élément *s*, correspondant à la capacité résiduelle du sac.

La résolution du problème continu permet de calculer une borne du problème d'origine et donne une idée sur sa solution optimale.

▪ Solution optimale entière immédiate

Une fois le problème continu résolu, seulement quelques variables auront besoin d'être changées pour aboutir à la solution optimale entière.

▪ Problème séparable

Une autre propriété importante est la possibilité de séparation du problème knapsack. En d'autres termes, un knapsack 0-1 par exemple peut être résolu en un temps $O(\sqrt{2^n})$ (**Horowitz et Sahni** 1974 [28]) au pire cas, en scindant l'ensemble d'objets en deux et en énumérant toutes les solutions possibles dans chacun des deux ensembles.

▪ Algorithmes de réduction efficaces

Des algorithmes de réduction efficaces ont été développés pour les problèmes de type knapsack, ce qui permet de fixer plusieurs variables de décision avant même de commencer la résolution, et donc de diminuer la tailles des instances.

2.8 Instances de problèmes

Lorsqu'on génère aléatoirement les instances d'un problème, la question qui se pose est la suivante : « ces instances sont-elles plus ou moins difficiles à résoudre ? ». Bien que la difficulté de résolution d'un problème soit liée à la taille de ce dernier, il arrive en considérant des tests sur des problèmes aléatoires, de se rendre compte que certains problèmes sont beaucoup plus difficiles à résoudre que d'autres.

Les instances des problèmes de type knapsack sont classées en fonction de la corrélation entre les poids et les profits des objets. On considère dans ce mémoire des instances générées aléatoirement et construites de manière à refléter certaines propriétés qui influencent le processus de résolution. Dans ce qui suit, nous allons présenter trois types d'instances générées aléatoirement :

2.8.1 Instances non corrélées

Dans ce type d'instances, le poids et le profit d'un objet ne sont pas corrélés.

Les coefficients p_j et w_j, $(j=1,..,n)$ sont tirés de manière uniforme dans, respectivement, $[1,U_p]$ et $[1,U_w]$ où U_p et U_w sont des paramètres fixés par l'utilisateur (U_p et U_w sont généralement pris égaux à 1000).

Ces instances illustrent bien les situations où le poids et le profit sont supposés indépendants, comme par exemple dans les problèmes de chargement de conteneurs : les objets de petite taille peuvent avoir des valeurs significatives ou vice versa.

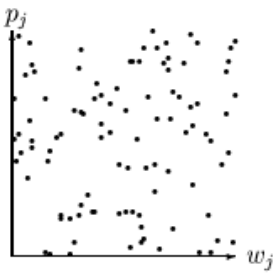

Fig 2.2 - Instances non corrélées.

Les instances non corrélées sont généralement faciles à résoudre. Comme il y à une grande différence entre les poids alors on peut facilement remplir le sac. Plus encore, il est facile d'éliminer plusieurs variables en utilisant les bornes supérieures ou les relations de dominance.

2.8.2 Instances faiblement corrélées

Contrairement au cas précédent, la génération des profits $p_j \ (j = 1,..,n)$ est liée aux poids $w_j \ (j = 1,..,n)$. Dans ces instances le profit et le poids d'un objet sont corrélés et diffèrent par une valeur proportionnelle. Les coefficients w_j sont tirés de manière uniforme dans $[1,U_w]$ et $p_j = \max\{1, \hat{p}_j\}$ avec $\hat{p}_j$ tiré de manière uniforme dans l'intervalle $[w_j - {}^{U_w}\!/_{10},\ w_j + {}^{U_w}\!/_{10}]$

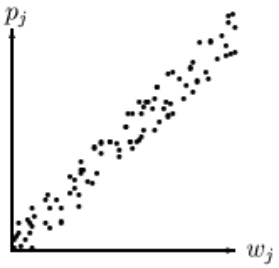

Fig 2.3 - Instances faiblement corrélées.

Ces instances sont très représentatives des situations financières où le gain est généralement proportionnel à la somme investie avec quelques petites variations.

La corrélation entre le profit et le poids est telle qu'il est généralement difficile d'éliminer des variables à travers des tests de bornes supérieures. Toutefois, les instances faiblement corrélées sont généralement faciles à résoudre, tout comme les instances non-corrélées.

2.8.3 Instances fortement corrélées

Les poids w_j sont tirés de manière uniforme dans $[1,U_w]$ et $p_j = w_j + 10$.

Ce type d'instances est le plus représentatif des situations rencontrées dans la réalité où le gain est une fonction linéaire de l'investissement plus (ou moins) quelques charges fixes.

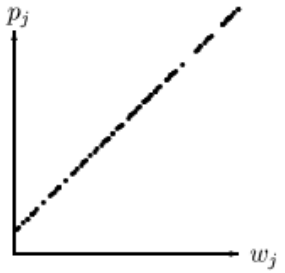

Fig 2.4 - Instances fortement corrélées.

Les instances fortement corrélées sont difficiles à résoudre pour deux raisons :

- Les objets autour de l'élément critique ont des poids très proches, ce qui les rend difficiles à manipuler.
- Des objets de petits poids peuvent être pris en premier, ce qui implique une perte en optimalité.

Les instances fortement corrélées sont généralement utilisées afin de mesurer la capacité des algorithmes à résoudre des problèmes difficiles.

3.1 Introduction

Le problème du knapsack 0-1 (noté KP pour *kanapsack problem*) est un problème d'optimisation combinatoire NP-difficile ayant un vaste champ d'application (**Gilmore et Gomory** [23], **Kellerer et al.** [35]. Il peut être considéré comme un problème à part ou comme un sous-problème ou une relaxation de problème d'optimisation combinatoire plus complexes.

KP a fait l'objet de nombreuses études dans la littérature ces quelques dernières décennies. Diverses approches aussi bien exactes qu'approchées ont été développées pour sa résolution. Elles incluent la programmation dynamique, le branch and bound et les approches hybrides.

En 1957, **Dantzig** [14] propose une méthode élégante et efficace pour résoudre la relaxation continue de KP et calculer une borne supérieure utilisée dans presque tous les travaux sur KP pendant près de 20 ans.

Dans les années 60, l'approche de la programmation dynamique pour la résolution du problème KP et des autres problèmes de type knapsack a été largement traitée par **Gilmore et Gomory** [23]. En 1967, **Kolesar** [36] expérimente le premier algorithme branch and bound pour les problèmes de type knapsack.

Dans les années 70, l'approche branch and bound et encore plus développée, puisqu'elle est, à ce jour, la seule méthode capable de résoudre des problèmes de grande taille, l'algorithme le plus connu à cette époque est celui de **Horowitz et Sahni** [28]. On doit un meilleur algorithme de programmation dynamique à **Garfinkel et Nemhauser** [21] en 1972. En 1973, **Ingargiola et Korsh** [30] présentent la première procédure de réduction, algorithme qui permet de réduire considérablement la taille des variables. En 1974, **Johnson** [32] donne le premier schéma d'approximation polynomiale pour le *subset sum problem*, résultat qui a été étendu par **Sahni** [56] au knapsack 0-1. En 1977, **Martello et Toth** [39] proposent la première borne supérieure dominant celle de Dantzig.

Les principaux travaux dans les années 80 concernent la résolution de problèmes de grande taille. En 1980, **Balas et Zemel** [2] proposent d'estimer une partie des variables de la solution optimale en fixant la stratégie de branchement sur les variables les plus intéressantes. Le sous-ensemble des variables sélectionnées, dit *noyau du problème*, est ensuite résolu par un algorithme branch and bound en profondeur d'abord. En 1985, **Plateau et El Kihel** [50] utilisent les approches hybrides.

En 1990, **Martello et Toth** [42] proposent une implémentation en branch and bound devenue une référence. Elle se distingue par une évaluation améliorée des nœuds et, une recherche locale lorsque le dernier objet ajouté au sac a mené à un échec.

Dans la suite de ce chapitre, nous allons détailler l'algorithme de **Dantzig**, l'algorithme branch-and-bound développé par **Horowitz et Sahni** ainsi que l'approche par la programmation dynamique de **Garfinkel et Nemhauser**.

3.2 Description et formulation du problème

Dans le problème KP, on considère un ensemble d'objets étiquetés de 1 à *n*. Chaque objet $j \in \{1,...,n\}$ étant associé à un poids w_j et un profit p_j de valeurs entières. On dispose d'un sac dont le contenu ne peut excéder une capacité *c* entière. On désire remplir le sac de façon à maximiser la somme des utilités des objets emportés, en respectant la contrainte de capacité. Autrement dit, Le problème consiste à résoudre un problème de programmation linéaire entière en vue de maximiser la fonction objective $z(x) = \sum_{j=1}^{n} p_j x_j$, les variables étant le vecteur

$x = (x_j)_{1 \le j \le n} \in \{0,1\}^n$, avec $x_j = \begin{cases} 1 & \text{si l'objet } j \text{ est emporté dans le sac} \\ 0 & \text{sinon.} \end{cases}$

sous *la contrainte* :

$$\sum_{j=1}^{n} w_j x_j \le c$$

Nous formulons, sans perte de généralité, l'hypothèse suivante :

$\sum_{j=1}^{n} w_j > c$, et $w_j \leq c$, pour $j = 1, ..., n$.

On appelle une instance d'un problème KP, la donnée des n poids $w_j \ (j = 1, ..., n)$, des n profits $p_j \ (j = 1, ..., n)$ et de la capacité c. Tout vecteur binaire $(x_1, x_2, ..., x_n)$ est appelé solution de KP. On dit qu'une solution est réalisable, notée $\hat{x}$, si elle vérifie la contrainte de capacité, c'est-à-dire $\sum_{1 \leq j \leq n} w_j \hat{x}_j \leq c$. La solution est dite optimale, notée x^*, si elle est à la fois réalisable et maximise la somme des profits des objets mis dans le sac. En d'autres termes, pour toute solution réalisable $\hat{x}$, on a :

$$\sum_{j=1}^{n} p_j \hat{x}_j \leq \sum_{j=1}^{n} p_j x^*_j$$

La formulation de KP est la suivante :

$$(\text{KP}) \begin{cases} \text{maximiser} \ \ z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} \qquad \sum_{j=1}^{n} w_j x_j \leq c \\ \qquad\quad x_j \in \{0,1\} \qquad \text{pour} \ \ j = 1,...,n \end{cases} \tag{3.1}$$

où $x_j = \begin{cases} 1 & \text{si l'objet} \ \ j \ \ \text{est séléctionné ;} \\ 0 & \text{sinon.} \end{cases}$

3.3 Heuristique pour le 0-1 knapsack

Rappelons que le terme « **heuristique** » est utilisé pour décrire un algorithme ou une procédure qui se base sur des expériences pratiques. Elle vise entre autres à résoudre des problèmes d'optimisation *NP-difficiles.*

Parmi les méthodes heuristiques, on peut citer la méthode dite gloutonne. Un **algorithme glouton** est un algorithme qui parcourt l'espace de recherche en optant à chaque

étape pour le chemin qui lui parait le plus perspicace, *ne mettant jamais en cause les choix antérieurs* (sans retour arrière).

Comme il existe plusieurs variantes pour ces méthodes gloutonnes, nous présentons ici une version simple et simplifiée, celle de Dantzig :

- On commence, tout d'abord, par ordonner les objets en fonction du rapport profit/poids selon un ordre décroissant, c'est-à-dire : $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq ... \geq \frac{p_n}{w_n}$.

  Cette indexation des objets peut se faire en un temps O($n \log n$) [42].

- On choisit à chaque étape l'élément ayant le plus grand rapport profit/poids, si cet élément est admissible, c'est-à-dire si son poids ne dépasse pas la capacité restante (résiduelle) du sac, après fixation des autres éléments, alors il est mis dans le sac, sinon on sélectionne l'élément qui se situe juste après et qui peut être admissible et ainsi de suite de proche en proche jusqu'à épuisement de tous les objets pouvant être mis dans le sac.

Ci-après, l'heuristique gloutonne qui permet de calculer la solution dite « critique » de KP. Notons qu'il existe d'autres heuristiques pour résoudre KP mais qui peuvent avoir des temps d'exécution très conséquents (heuristique AGNES de **Fréville et Plateau** , l'heuristique ADP-BH, ADP pour Approximate Dynamic Programming et BH pour Base Heuristics, de **Bertsimas et Demir**).

Algorithme 3.1 : Algorithme glouton pour KP.

Entrée : Une instance d'un problème de sac à dos ;
Sortie : Une solution réalisable $\bar{x}$;

1. $\bar{c} := c$;
2. **Pour** $j := 1$ **jusqu'à** n **faire**
   Si $w_j \leq \bar{c}$ **alors** $\bar{x}_j := 1$;
   $\bar{c} := \bar{c} - w_j$
   sinon $\bar{x}_j := 0$
   FinSi
   FinPour
3. **Sortir avec une solution réalisable** $\bar{x}$.

3.4 Relaxations et bornes supérieures de KP

Calculer des bornes supérieures ou inférieures (en fonction du contexte d'optimisation) permet d'encadrer la valeur de la solution optimale pour les problèmes que l'on tente de résoudre. Ces bornes sont ensuite utilisées pour le développement de méthodes de résolution exacte s'appuyant sur des procédures d'énumération implicite.

Les méthodes permettant d'obtenir les bornes supérieures sont les méthodes de relaxation du problème. En élargissant l'espace des solutions réalisables, nous acceptons de nouvelles solutions hors de l'espace d'origine. La plus grande valeur de la fonction objective associée à ces solutions nous conduit à une borne supérieure.

Pour ce faire, Les méthodes abordées sont la relaxation linéaire continue et la relaxation lagrangienne.

3.4.1 Relaxation continue et borne de Dantzig

La relaxation la plus classique et la plus naturelle de KP est la relaxation continue, notée C(KP), programme linéaire continu associé, obtenu en relâchant les variables x_j , $j = 1, ..., n$, dans l'intervalle $[0,1]$.

La formulation de C(KP) est donnée par :

$$\text{C(KP)} \begin{cases} \text{maximiser } z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} \quad \sum_{j=1}^{n} w_j x_j \le c \\ \quad 0 \le x_j \le 1 \qquad \text{pour } j = 1,...,n \end{cases} \tag{3.2}$$

En 1957, Dantzig propose une manière simple pour résoudre la relaxation continue du problème KP.

L'idée consiste à remplir le sac objet après objet et de proche en proche jusqu'à saturation selon l'algorithme glouton énoncé ci-dessus. On repère ensuite le premier objet ne pouvant être mis en totalité dans le sac. On appelle cet objet « *élément critique* ». Il s'agit d'un élément d'indice $s \in \{1,...,n\}$ tel que $s = \min\{\ j : \sum_{i=1}^{j} w_i > c\ \}$.

Il est clair que ce dernier joue un rôle important dans la résolution de KP : En connaissant l'élément critique, on peut déterminer la borne de Dantzig et construire la solution critique x' obtenue en mettant $x'_j = 1$ pour $j = 1,...,s-1$ et $x'_j = 0$ pour $j = s,...,n$, qui constitue une bonne borne inférieure. La connaissance de ces deux bornes permet de réduire la taille du problème et d'éliminer quelques variables. L'élément critique constitue aussi un point de départ pour l'arbre de séparation dans les méthodes branch-and-bound.

Remarque :

Le problème de recherche de l'élément critique et de complexité O(n) [2].

La solution optimale pour le problème C(KP), serait alors de prendre tous les éléments d'indice $j < s$ plus une fraction de l'élément s correspondant à la capacité résiduelle du sac.

Théorème 3.1 [14]

$$x^*_j = 1 \qquad \text{pour } j = 1,...,s-1,$$
$$x^*_j = 0 \qquad \text{pour } j = s+1,...,n,$$
$$x^*_s = \frac{\bar{c}}{w_s} \qquad \text{où } \bar{c} = c - \sum_{j=1}^{s-1} w_j$$

Preuve [42]

On peut remarquer que toute solution optimale x^* de C(KP) est maximale, c'est-à-dire : $\sum_{j=1}^{n} w_j x^*_j = c$.

On suppose, sans perte de généralité, que $\frac{p_j}{w_j} \leq \frac{p_{j+1}}{w_{j+1}} \quad \forall j = 1,...,n$,

et soit x^* la solution optimale de C(KP).

- Supposons par l'absurde que $\exists k < s$ tel que $x^*_k < 1$

  Dans ce cas, $x^*_q < \bar{x}_q$ pour au moins un objet d'indice $q \geq s$.

  Etant donné un $\varepsilon > 0$ (suffisamment petit), x^*_k augmente alors d'une valeur égale à ε,

  et x^*_q diminue de $\frac{\varepsilon \times w_k}{w_q}$,

  ce qui implique l'augmentation de la valeur optimale de C(KP) d'une valeur positive (puisque $\frac{p_k}{w_k} > \frac{p_q}{w_q}$) égale à $\varepsilon\left(p_k - p_q \frac{w_k}{w_q}\right)$ d'où *la contradiction*.

- De la même manière, on peut montrer qu'il est impossible d'avoir $x^*_k > 0$ pour $k > s$. ■

La valeur optimale du problème relaxé C(KP) est $z(C(KP)) = \sum_{j=1}^{s-1} p_j + \bar{c}\frac{p_s}{w_s}$ où $\bar{c} = c - \sum_{j=1}^{s-1} w_j$.

Ci-après, les étapes principales de l'algorithme de Dantzig.

Algorithme 3.2 : Algorithme de résolution de C(KP).

Entrée : Une instance d'un problème de sac à dos ;
Sortie : Une solution optimale x^* ;
Une valeur optimale z^* ;

0. **Ordonner les objets selon l'ordre suivant** :

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq ... \geq \frac{p_n}{w_n}$$

1. **Poser**
$\bar{c} := c$;
$z^* := 0$;

2. **Pour** $j := 1$ **jusqu'à** n **faire**
Si $w_j < \bar{c}$ **alors** $x_j^* := \min\left\{1, \frac{\bar{c}}{w_j}\right\}$;
$z^* := z^* + p_j \times \min\left\{1, \frac{\bar{c}}{w_j}\right\}$;
$\bar{c} := \bar{c} - w_j \times \min\left\{1, \frac{\bar{c}}{w_j}\right\}$;
sinon $x_j^* := 0$
FinSi
FinPour
3. **Sortir avec une solution optimale** x^*
et une valeur optimale z^*.

Une solution réalisable pour KP l'est aussi pour C(KP). Ceci est vrai puisque la contrainte d'intégralité sur les variables x_j ($j = 1,...,n$) dans C(KP) est plus relâchée que celle dans KP. On a alors, $z(C(KP)) \geq z(KP)$.

La valeur optimale du problème relaxé C(KP) constitue donc une borne supérieure associée au problème KP, et z(C(KP)) = UB(KP).

Mieux que ça, sachant que toute solution de KP est entière, alors la valeur entière inférieure de la valeur optimale de C(KP) demeure une borne supérieure pour KP, d'où l'idée de la **borne de Dantzig**, notée UB_D :

$$UB_D = \lfloor z(C(KP)) \rfloor = \sum_{j=1}^{s-1} p_j + \left\lfloor \left(c - \sum_{j=1}^{s-1} w_j \right) \frac{p_s}{w_s} \right\rfloor$$

Remarque :

La complexité de UB_D est en $O(n)$ [43].

Borne de Martello et Toth

En 1977, soit vingt ans après Dantzig, **Martello et Toth** proposent la première borne dominant celle de Dantzig. Ils suivent un raisonnement similaire à celui de Dantzig en imposant une contrainte d'intégralité sur la variable critique x_s (i.e. contraintes additionnelles sur x_s).

Théorème 3.4 [42]

Soient

$$U^0 = \sum_{j=1}^{s-1} p_j + \left\lfloor \bar{c} \frac{p_{s+1}}{w_{s+1}} \right\rfloor \quad \text{et} \quad U^1 = \sum_{j=1}^{s-1} p_j + \left\lfloor p_s - (w_s - \bar{c}) \frac{p_{s-1}}{w_{s-1}} \right\rfloor$$

Alors

(i) *U_{MT} est une borne supérieure pour KP telle que :*

$$UB_{MT} = \max (U^0, U^1)$$

(ii) *Pour toute instance de KP, nous avons $UB_{MT} \leq UB_D$.*

Preuve

(i) Puisque x_s est entière alors la valeur optimale de KP peut être calculée à partir de la solution continue de C(KP), soit en omettant l'objet d'indice *s* (i.e. en imposant $x^*_s = 0$), soit en l'insérant (i.e. en imposant $x^*_s = 1$) et donc en enlevant au moins un des $(s-1)$ premiers objets.

- U^0 est une borne associée à $x^*_s = 0$. La capacité résiduelle est donc comblée par une fraction de l'objet d'indice (s+1), car ayant la plus grande valeur $\frac{p_j}{w_j}$, $j = s+1, ..., n$.
- U^1 est associée à $x^*_s = 1$. Dans ce cas, l'objet enlevé (le $(s-1)^{\text{ème}}$, car ayant la plus petite valeur de $\frac{p_j}{w_j}$) doit avoir exactement le poids minimum nécessaire (i.e. $w_s - \bar{c}$).

(ii) - $U^0 \overset{?}{\leq} UB_D$:

comme $\bar{c}\frac{p_{s+1}}{w_{s+1}} \leq \bar{c}\frac{p_s}{w_s}$, alors :

$$\left\lfloor \bar{c}\frac{p_{s+1}}{w_{s+1}} \right\rfloor \leq \left\lfloor \bar{c}\frac{p_s}{w_s} \right\rfloor$$

d'où $$U^0 = \sum_{j=1}^{s-1} p_j + \left\lfloor \bar{c}\frac{p_{s+1}}{w_{s+1}} \right\rfloor \leq \sum_{j=1}^{s-1} p_j + \left\lfloor \bar{c}\frac{p_s}{w_s} \right\rfloor = UB_D$$

- $U^1 \overset{?}{\leq} UB_D$:

comme $\frac{p_s}{w_s} \leq \frac{p_{s-1}}{w_{s-1}}$ et $\bar{c} < w_s$, alors :

$$(\bar{c} - w_s)\left(\frac{p_s}{w_s} - \frac{p_{s-1}}{w_{s-1}}\right) \geq 0 \quad \Leftrightarrow \quad \bar{c}\frac{p_s}{w_s} \geq p_s - (w_s - \bar{c})\frac{p_{s-1}}{w_{s-1}}$$

d'où $$UB_D = \sum_{j=1}^{s-1} p_j + \left\lfloor \bar{c}\frac{p_s}{w_s} \right\rfloor \geq \sum_{j=1}^{s-1} p_j + \left\lfloor p_s - (w_s - \bar{c})\frac{p_{s-1}}{w_{s-1}} \right\rfloor = U^1$$ ■

Remarque :

La complexité temporelle pour le calcul de UB_{MT} est en $O(n)$ [42].

Bornes de Hudson et Fayard & Plateau

En agissant toujours sur l'élément critique, d'autres bornes ont été élaborées. En 1977, **Hudson** propose de calculer la borne $\overline{U}^1$ qui représente la solution de la relaxation continue de KP plus une contrainte additionnelle $x_s = 1$. **Fayard et Plateau** [50] en 1982, et indépendamment **Villela et Bornstein** [62] en 1983, proposent de calculer la borne $\overline{U}^0$ comme solution de C(KP) avec la contrainte additionnelle $x_s = 0$.

On définit $\sigma^0(j)$ et $\sigma^1(j)$ indice de l'élément critique quand on impose, respectivement, $x_j = 1$ ($j \geq s$) et $x_j = 0$ ($j \leq s$) :

$$\sigma^0(j) = \min\left\{ k : \sum_{\substack{i=1 \\ i \neq j}}^{k} w_i > c \right\} \quad j \in \{1,...n\} \quad \text{et} \quad \sigma^1(j) = \min\left\{ k : \sum_{i=1}^{k} w_i > c - w_j \right\} \quad j \in \{1,...s-1\}$$

On obtient les bornes :

$$\overline{U}^0 = \sum_{\substack{j=1 \\ j \neq s}}^{\sigma^0_{(s)}-1} p_j + \left\lfloor \left(c - \sum_{\substack{j=1 \\ j \neq s}}^{\sigma^0_{(s)}-1} p_j \right) \frac{p_{\sigma^0_{(s)}}}{w_{\sigma^0_{(s)}}} \right\rfloor \quad \text{et} \quad \overline{U}^1 = p_s + \sum_{j=1}^{\sigma^1_{(s)}-1} p_j + \left\lfloor \left(c - w_s - \sum_{j=1}^{\sigma^1_{(s)}-1} p_j \right) \frac{p_{\sigma^1_{(s)}}}{w_{\sigma^1_{(s)}}} \right\rfloor$$

et la nouvelle borne supérieure est :

$$UB_{H,FP} = \max\left(\overline{U}^0, \overline{U}^1\right)$$

Remarque :

Il est évident que

(a) $\overline{U}^0 \leq U^0$ et $\overline{U}^1 \leq U^1$ d'où $UB_{H,FP} \leq UB_{MT}$;

(b) la complexité temporelle pour le calcul de $UB_{H,FP}$ est la même que celle pour le calcul de UB_D et UB_{MT}. C'est-à-dire en $O(n)$ [42].

Exemple 3.1 :

Soit une instance de KP définie par :

$n = 8$,

$(p_j) = (15, 100, 90, 60, 40, 15, 10, 1)$,

$(w_j) = (2, 20, 20, 30, 40, 30, 60, 10)$,

$c = 102$.

La solution optimale est $x = (1, 1, 1, 1, 0, 1, 0, 0)$, de valeur $z = 280$. On a $s = 5$,

$$UB_D = 265 + \left\lfloor 30\frac{40}{40} \right\rfloor = 295$$
$$U^0 = 265 + \left\lfloor 30\frac{15}{30} \right\rfloor = 280$$
$$U^1 = 265 + \left\lfloor 40 - 10\frac{60}{30} \right\rfloor = 285$$
$$UB_{MT} = 285.$$

$\sigma^0(5) = 7$. $\overline{U}^0 = 280 + \left\lfloor 0\frac{10}{60} \right\rfloor = 280$;

$\sigma^1(5) = 4$. $\overline{U}^1 = 40 + 205 + \left\lfloor 20\frac{60}{30} \right\rfloor = 285$;

$UB_{H,FP} = 285$. □

3.4.2 Relaxation lagrangienne

L'approche lagrangienne peut être utilisée dans le but de produire un problème relaxé plus facile à résoudre à partir de n'importe quel problème difficile d'optimisation

combinatoire. De plus, la solution optimale du problème relaxé est une borne supérieure (cas de maximisation) de la solution optimale du problème original.

Dans cette section, nous présentons l'approche de relaxation lagrangienne basée sur la dualisation de la contrainte de capacité. Nous décrivons ainsi le problème relaxé obtenu et les bornes de **Müller-Merbach** [47] et de **Dudzinski & Walukiewicz** [15], calculées grâce à cette approche.

L'idée générale derrière une approche de relaxation lagrangienne est de remplacer un problème linéaire contraint par un problème non contraint généralement plus facile à résoudre.

Soit le problème L(KP, λ) la relaxation Lagrangienne de KP, où λ est un multiplicateur de Lagrange ($\lambda \geq 0$). L(KP, λ) est donné comme suit :

$$L(KP,\lambda) = \begin{cases} \max \sum_{j=1}^{n} p_j x_j + \lambda\left(c - \sum_{j=1}^{n} w_j x_j\right) \\ x_j \in \{0,1\} \qquad j = 1,...n \end{cases} \tag{3.3}$$

La fonction objectif peut être réécrite comme suit :

$$z\big(L(KP,\lambda)\big) = \sum_{j=1}^{n} \tilde{p}_j x_j + \lambda c$$

où $\tilde{p}_j = p_j - \lambda w_j$ pour $j = 1,..n$.

La solution optimale de L(KP, λ), dans ce cas, est déterminée en un temps polynomial [44] comme suit :

$$\tilde{x}_j = \begin{cases} 1 & si\ \tilde{p}_j > 0 \\ 0 & si\ \tilde{p}_j < 0 \end{cases}$$

(si $\tilde{p}_j = 0$, la valeur de $\tilde{x}_j$ devient immatérielle).

on définit $J(\lambda) = \left\{ j : \frac{p_j}{w_j} > \lambda \right\}$, la valeur optimale de L(KP, λ) est donc :

$$z\big(L(KP,\lambda)\big) = \sum_{j \in J(\lambda)} \tilde{p}_j + \lambda c \qquad (\lambda \geq 0)$$

cette valeur constitue une borne supérieure pour *z*(KP) et ne peut, cependant, être meilleure (plus serrée) que celle de Dantzig. En effet, (.) donne aussi la solution de la relaxation continue de L(KP, λ) (i.e. C(L(KP, λ)),

d'où

$$z\big(L(KP,\lambda)\big) = z\big(C\big(L(KP,\lambda)\big)\big) \geq z\big(KP\big) = UB_D$$

La valeur de λ qui minimise L(KP, λ) est :

$$\lambda^* = \frac{p_s}{w_s}$$

Dans ce cas, on a :

$$\begin{cases} \tilde{p}_j \geq 0 & j = 1,...s-1 \\ \tilde{p}_j \leq 0 & j = s,...n \end{cases}$$

Comme

$$\tilde{x}_j = \bar{x}_j, \quad j \in N \setminus \{s\}$$

Où $\bar{x}_j = 1 \qquad j = 1,...s-1$

$\bar{x}_j = 0 \qquad j = s+1,...n$

$\bar{x}_s = \frac{\bar{c}}{w_s} \qquad où \quad \bar{c} = \sum_{j=1}^{s-1} w_j$

on a $$z\left(L(KP,\lambda^*)\right)=\sum_{j=1}^{s-1}\left(p_j-\lambda^* w_j\right)+\lambda^* c=z\left(C\left(KP\right)\right)$$

et pour $\lambda=\lambda^*$, $\tilde{p}_j$ devient :

$$p^*{}_j = p_j - w_j \frac{p_s}{w_s}$$

En posant $\tilde{x}_j = 1-\tilde{x}_j$, z(L(KP, λ^*)) décroît de $\left|p^*{}_j\right|$, $\left|p^*{}_j\right|$ représente donc la borne inférieure pour la diminution de la solution optimale continue z(C(KP)). C'est-à-dire qu'on peut obtenir une borne meilleure que UB_D grâce à la relaxation Lagrangienne.

Borne de Müller-Merbach

L'idée sur laquelle repose le calcul de la borne de **Müller-Merbach** [47] (1978) consiste à déterminer une solution entière à partir de celle donnée par la relaxation continue, c'est-à-dire :

(a) en mettant la valeur de x_s^* égale à 0, sans changer les valeurs des autres variables.

Dans le cas (a), z(C(KP)) diminue de $\bar{c}\,\frac{p_s}{w_s}$,

et devient $\text{z(C(KP))}=\sum_{j=1}^{s-1} p_j$

Sinon ;

(b) au moins une des variables $x_j\ (j\neq s)$ change sa valeur de 0 à 1 ou bien de 1 à 0 ($x_j = 1-x_j$).

Dans ce cas, z(C(KP)) diminue d'au moins $\left|p^*{}_j\right|$.

Ainsi, la borne devient :

$$UB_{MM}=\max\left(\sum_{j=1}^{s-1} p_j, \max\left\{\left\lfloor z\left(C\left(KP\right)\right)-\left|p^*{}_j\right|\right\rfloor : j\in N\setminus\{s\}\right\}\right)$$

Remarque :

Il est évident que $UB_{MM} \le UB_D$. Par ailleurs, il n'existe pas de dominance entre UB_{MM} et les autres bornes déjà vues, mais il existe des exemples pour lesquels $UB_{MM} < UB_{H,FP} \le UB_{MT}$.

La borne UB_{MM} est de complexité $O(n)$.

Borne de Dudzinski & Walukiewicz

Le principe du calcul de $UB_{MT}, UB_{H,FP}$ et UB_{MM} a été exploité par **Dudzinski & Walukiewicz** [15] (1984) qui ont calculé une borne supérieure *qui domine toutes les précédentes* et qui est elle aussi de complexité $O(n)$.

Considérons une solution réalisable $\hat{x}$, de KP, obtenue comme suit :

1. **pour chaque** $k \in N \setminus \{s\}$ **faire** $\hat{x}_k = \bar{x}_k$;
2. $\hat{x}_s := 0$;
3. **pour chaque** *k* tel que $\hat{x}_k := 0$ **faire**

$$\textbf{si } w_k \le c - \sum_{j=1}^{n} w_j \hat{x}_j \textbf{ alors } \hat{x}_k := 1,$$

et on définit $\hat{N} = \{j \in N \setminus \{s\} : \hat{x}_j = 0\}$.

Une solution entière peut alors être obtenue :

(i) en posant $x_s = 1$, ou

(ii) en posant $x_s = 0$ et $x_j = 1$ pour au moins un $j \in \hat{N}$,

La borne de Dudzinski & Walukiewicz devient immédiate et est égale à :

$$UB_{DW} = \max\Big(\min\big(\overline{U}^1, \max\{\lfloor z(C(KP)) - p_j^* \rfloor : j = 1,...s-1\}\big),$$
$$\min\big(\overline{U}^0, \max\{\lfloor z(C(KP)) + p_j^* \rfloor : j \in \hat{N}\}\big),$$
$$\textstyle\sum_{j=1}^{n} p_j \hat{x}_j\Big)$$

Exemple 3. 1 (suite) :

On a $(p^*{}_j) = (13, 80, 70, 30, 0, -15, -50, -9)$

$UB_{DW} = \max(265, \max\{282, 215, 225, 265, 280, 245, 286\}) = 286$.

$(\hat{x}_j) = (1, 1, 1, 1, 0, 1, 0, 0)$;

$$UB_{DW} = \max\big(\min(285, \max\{282, 215, 225, 265\}),$$
$$\min(280, \max\{245, 286\}),\ 280) = 282.$$

□

3.5 Les méthodes de résolution exacte

Plusieurs approches de résolution exacte ont été élaborées pour KP. La plupart de ces méthodes sont basées sur les techniques énumératives. En général, on s'appuie sur une méthode de branch and bound où il s'agit d'énumérer d'une manière implicite les solutions réalisables et de choisir la meilleure parmi elles.

3.5.1 Algorithme branch and bound

Développer tout l'espace de recherche consiste, pour une instance du problème KP (instance *n* éléments), à développer 2*n* vecteurs binaires de taille *n*. Ce qui est excessivement lourd et irréalisable au regard du temps d'exécution nécessaire à l'exploration de toute l'arborescence.

La méthode branch and bound consiste ici à séparer le problème initial en plusieurs sous-problèmes et d'éliminer certains sous-problèmes à l'aide d'un système de majorants et minorants.

Dans la littérature, plusieurs auteurs ont présenté des algorithmes branch and bound pour la résolution de KP, notamment **Horowitz** [28], **Martello et Toth** [40].

La première approche B&B a été présentée par **Kolesar** [36] en 1967. Son algorithme consiste en un schéma de branchement en largeur-d'abord : la sélection se fait sur le nœud ayant le maximum du rapport profit sur unité de poids, à partir duquel deux nœuds sont générés ($x_j = 1$et $x_j = 0$). La fonction d'évaluation étant la borne de Dantzig.

Le temps de calcul et l'espace mémoire nécessaire à la mise en œuvre de cet algorithme sont considérablement réduits par l'approche en profondeur d'abord de **Greenberg et Hegerich** en 1970.

Horowitz et Sahni [28] (1974) et indépendamment **Ahrens et Finke** (1975) présentent un algorithme basé sur une stratégie gloutonne qui s'inspire du schéma de branchement en profondeur-d'abord de Greenberg et Hegerich et de la sélection de Kolesar.

D'autres algorithmes ont été développés à partir de l'approche de Greenberg et Hegerich (**Barr et Ross**, **Laurière**) et autres techniques (**Lageweg et Lenstra**, **Guignard et Spielberg** , **Fayard et plateau**, **Veliev et Mamedov**). L'approche de Horowitz et Sahni reste, cependant, la plus structurée et la plus facile à implémenter.

Dans ce qui suit, nous allons présenter l'algorithme de branch and bound développé par **Horowitz et Sahni** [28]. Cette méthode considère les éléments ordonnés selon l'ordre décroissant du rapport profit par poids. Ensuite, la séparation se fait sur l'élément suivant. Depuis chaque nœud de l'arborescence, et pour un élément *j* pris dans l'ordre prédéfini, on développe deux branches : la première branche, $x_j = 1$ correspondant à l'élément *j* mis dans le sac et la deuxième branche, $x_j = 0$ correspond à l'élément *j* qui n'est pas mis dans le sac.

Le développement de l'arborescence se fait en profondeur, en privilégiant les branches pour lesquelles les éléments sont mis dans le sac. La fonction d'évaluation utilisée est la borne de Dantzig. Un élément n'est sélectionné que si son poids ne dépasse pas la capacité restante ou encore disponible du sac. Ainsi, à chaque développement d'une branche complète de l'arborescence, la solution obtenue reste réalisable. Le calcul des bornes de Dantzig se fait

uniquement au niveau des nœuds développés par une branche correspondant à un élément *j* non sélectionné, c'est-à-dire correspondant à $x_j = 0$.

Pour les nœuds développés par une branche correspondant à $x_j = 1$, la valeur de la borne de Dantzig n'est autre que celle du nœud père. Celle-ci est comparée à la valeur de la meilleure solution obtenue jusque-là. Si la valeur de la borne est inférieure à la valeur de cette meilleure solution, alors il est évident qu'on ne pourra jamais atteindre une meilleure solution à partir de ce nœud. Donc, la troncature au niveau de cette branche courante est possible.

Algorithme 3.3 : Algorithme branch and bound pour KP.

Entrée : Une instance d'un problème knapsack ;
Sortie : Une solution optimale x^* ;

1. j := 1 ; *MeilleureValeur* :=0 ;
2. Calcul de la borne supérieure UB_d qu'on peut atteindre depuis *j*
3. **Si** $UB_d \geq MeilleureValeur$ **alors**
   Développer une branche de l'arborescence en profondeur d'abord pour obtenir une solution réalisable $\bar{x}'$;
   $j := j + 1$;
   Si $Z(\bar{x}') \geq MeilleureValeur$ **alors**
   $\bar{x} := \bar{x}'$
   MeilleureValeur := $Z(\bar{x}')$;
   FinSi
   FinSi
4. $j = \max\{s : s \leq j \text{ et } \bar{x}'_j = 1\}$
   Tant que *j* existe **faire**
   $\bar{x}'_j = 0$;
   Fin Tant Que ;
5. $x^* := x$.

3.5.2 Programmation dynamique

Le problème KP possède la propriété de sous-structure optimale, c'est-à-dire que l'on peut construire la solution optimale du problème à *i* variables à partir du problème à (*i* -1)

variables. Cette propriété a permis d'utiliser la méthode de résolution par programmation dynamique, l'approche de **Garfinkel et Nemhauser** est détaillée ci-après.

Soit $KP(i,m)$ le problème constitué des *i* premiers objets, $1 \le i \le n$, avec les poids $w_1,...,w_i$, les profits $p_1,...,p_i$ et la capacité m, $1 \le m \le c$. L'idée est la suivante :

Soit $Z[i,m]$ la valeur de la solution optimale du problème $KP(i,m)$. $Z[i,m]$ est donc la valeur du sous ensemble de valeur maximale des *i* premiers objets dont le poids total n'excède pas *m.* Il existe deux types de sous-ensembles des *i* premiers objets dont le poids total n'excède pas *m* :

- ceux qui incluent l'objet *i*,
- ceux qui n'incluent pas l'objet *i*.

Parmi les sous-ensembles (des *i* premiers objets) qui n'incluent pas l'objet *i*, la valeur du sous ensemble optimal est $Z[i-1,m]$ et parmi les sous-ensembles qui incluent l'objet *i* ($m-w_i \ge 0$), la valeur du sous ensemble optimal est $p_i + Z[i-1,m-w_i]$. La valeur maximale du sous ensemble optimal des *i* premiers objets est donc le maximum de ces deux valeurs. Nous avons donc la récurrence:

$$Z[i,m] = \begin{cases} \max(Z[i-1,m], p_i + Z[i-1,m-w_i]) & m-w_i \ge 0 \\ Z[i-1,m] & m-w_i < 0 \end{cases}$$

Nous avons également les conditions initiales suivantes:

- $Z[i,0] = 0 \quad i \ge 0,$ le knapsack à zéro capacité a une solution optimale de valeur nulle.
- $Z[0,m] = 0 \quad m \ge 0,$ le knapsack à zéro variable a aussi une solution optimale de valeur nulle.

Pour trouver la valeur de la solution optimale du problème initial, il suffit de construire le tableau $Z[i,m]$ du bas vers le haut à l'aide de la récurrence précédente.

Algorithme 3.4 : Algorithme de programmation dynamique pour résoudre KP.

Entrée : Une instance d'un problème knapsack ;
Sortie : une valeur optimale z^* ;
une solution optimale x^* ;

0. Pour m := 0 jusqu'à c faire
$Z[0,m] := 0$
Fin Pour
Pour i := 0 jusqu'à n faire
$Z[i,0] := 0$
Fin Pour

1. Pour $i := 1$ **jusqu'à n faire**
Pour $m := 1$ **jusqu'à c faire**
Si $m - w_i \geq 0$ **alors**
$Z[i,m] = \max(Z[i-1,m], p_i + Z[i-1,m-w_i])$
Sinon
$Z[i,m] = Z[i-1,m]$
Fin Si
Fin Pour
Fin Pour

2. Poser $m := c$ **;**
Pour $m := n$ **jusqu'à 1 faire**
Si $Z[i,m] > Z[i-1,m]$ **alors**
début
$x_i^* = 1$
$m = m - w_i$
fin
Sinon
$x_i^* = 0$
Fin Si
Fin Pour

3. **Sortir avec une solution optimale** x^*
et une valeur optimale z^*.

Cet algorithme a une complexité temporelle et spatiale en $O(nc)$. Cependant, elle peut être ramenée à $O(c)$ si seule la valeur de la solution optimale est importante.

Cet algorithme est pseudo-polynomial et on n'a pas besoin de trier les objets à priori mais il est gourment en espace mémoire.

3.6 Etude de la sensibilité de la solution optimale face à la perturbation du profit

Peu de travaux ont été réalisés dans l'analyse de la sensibilité des problèmes de type knapsack. Les premières études ont été proposées par **Ghosh et al.** [22]. **Blair** [2] et **Woeginger** [64] ont donné quelques résultats négatifs en analyse de la sensibilité des problèmes knapsack. **Hifi et al.** [26] ont fait une analyse de la sensibilité de l'optimum avec perturbation soit du profit ou du poids d'un objet du knapsack 0-1. Dans **Hifi et al.** [25] un algorithme adaptatif est proposé pour le calcul de l'intervalle de sensibilité lorsque le profit et/ou le poids d'un objet est perturbé. Dans **Belgacem et Hifi 2005** [7], les auteurs ont étudié le cas où les profits d'un ensemble d'objets sont perturbés indépendamment.

Dans cette section, nous allons détaillé l'algorithme de **Hifi et al.**[26] pour une analyse de la sensibilité de la solution optimale du problème KP face à la perturbation du coefficient profit d'un objet arbitraire. Les auteurs se basent pour la détermination d'un intervalle de sensibilité du profit perturbé sur les conditions relevant de méthodes spécifiques (pour ce cas, il s'agit de la méthode branch and bound) et utilisent la borne de Dantzig comme borne supérieure de KP.

Revenons au problème KP défini précédemment :

$$(\text{KP}) \begin{cases} \text{maximiser} \quad z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} \qquad \sum_{j=1}^{n} w_j x_j \le c \\ \qquad\quad x_j \in \{0,1\} \qquad \text{pour} \quad j = 1,...,n \end{cases}$$

Soit $x^* = (x_1^*,...,x_s^*,...,x_n^*)$ une solution optimale de KP,

et soit le problème perturbé, KP′, défini comme suit:

$$(\mathrm{KP}')\begin{cases} \text{maximiser } z(x)=\sum_{\substack{j=1\\ j\neq s}}^{n} p_j x_j + (p_k+\Delta p_k)x_k \\ \text{s.c} \quad \sum_{\substack{j=1\\ j\neq k}}^{n} w_j x_j + w_k x_k \le c \\ x_j \in \{0,1\} \qquad \text{pour } j=1,...,n \end{cases}$$

où KP′ est un 0-1 knapsack obtenu en faisant varier le $k^{ème}$ profit p_k ($k=1,...,n$) dans KP, c'est-à-dire en posant :

$$p'_k = p_k + \Delta p_k$$

où p'_k est le profit du $k^{ème}$ objet dans le problème (KP′).

avec Δp_k un entier équivalent à la variation du profit du $k^{ème}$ objet.

Résultat 3.1 [26]

(i) *Chacun des deux problèmes KP et KP′ possède le même ensemble de solutions réalisables.*

(ii) *Si* $\hat{x}=(\hat{x}_1,...,\hat{x}_k,...,\hat{x}_n)$ *est une solution réalisable des deux problèmes KP et KP′, et* $\hat{Z}(\hat{x})$ *(resp.,* $\hat{Z}'(\hat{x})$*) sa valeur objective dans KP (resp., KP′), alors* $\hat{Z}'(\hat{x})=\hat{Z}(\hat{x})+\Delta p_k \hat{x}_k$.

Preuve

(i) Le poids de chaque objet est le même dans KP et KP′, et la capacité c du sac est également la même dans les deux problèmes. Par conséquent, il est évident que KP et KP′ possèdent le même ensemble de solutions réalisables.

(ii) Soit $\hat{x}$ une solution réalisable de KP et KP′, soit $\hat{Z}(\hat{x})$ (resp., $\hat{Z}'(\hat{x})$) sa valeur objective dans KP (resp., KP′) :

$\hat{Z}(\hat{x}) = \sum_{j=1}^{n} p_j \hat{x}_j$ et $\hat{Z}'(\hat{x}) = \sum_{j=1}^{n} p_j \hat{x}_j + \Delta p_k \hat{x}_k$,

d'où $\hat{Z}'(\hat{x}) = \hat{Z}(\hat{x}) + \Delta p_k \hat{x}_k$. ■

Il s'agit, pour le problème de l'analyse de la sensibilité de la solution optimale x^*, de déterminer l'intervalle de sensibilité du profit perturbé p_k. Soit I_k cet intervalle.

$I_k = [\ I_k^-, I_k^+\ ]$, $k = 1,...,n$, est un intervalle de *IR* contenant p_k. On dit que I_k est un intervalle de sensibilité pour le profit p_k, si pour tout p_k dans I_k, la solution x^* demeure une solution optimale de KP′.

Les limites de l'intervalle I_k dépendent de la validité de la fonction objectif de KP′ lorsque x^* en est une solution optimale puisque les deux problèmes ont les mêmes solutions réalisables. Ceci étant, les limites de l'intervalle de sensibilité doivent garantir que $\hat{Z}'(\hat{x}) \le \hat{Z}'(x^*)$. Quatre cas sont alors possibles :

(i) $x_k^* = 1$ et $\Delta p_k \ge 0$,

(ii) $x_k^* = 0$ et $\Delta p_k \le 0$,

(iii) $x_k^* = 1$ et $\Delta p_k \le 0$, et

(iv) $x_k^* = 0$ et $\Delta p_k \ge 0$.

Dans le résultat suivant (Résultat 3.2) on donne les limites de l'intervalle de sensibilité I_k en considérant ces quatre cas.

Résultat 3.2 [25]

Soit x^* une solution optimale de KP. x^* est aussi une solution optimale de KP′ si

(i) $x_k^* = 1$ *et* $\Delta p_k \geq 0$ *ou bien* $x_k^* = 0$ *et* $\Delta p_k \leq 0$.

(ii) $x_k^* = 1,\ \Delta p_k \leq 0$ *et* $\Delta p_k \geq \hat{Z}(\hat{\omega}_0) - Z^*$.

(iii) $x_k^* = 0,\ \Delta p_k \geq 0$ *et* $\Delta p_k \leq Z^* - \hat{Z}(\hat{\omega}_1)$.

où :

$\hat{\omega}_0$ *est la meilleure parmi les solutions réalisables, de KP, dont la* $k^{ème}$ *variable est nulle,*

$\hat{\omega}_1$ *la meilleure parmi les solutions réalisables, de KP, dont la* $k^{ème}$ *variable est égale à 1.*

Preuve

Les auteurs dans [25], montrent que si (i) est vérifiée alors x^* demeure une solution optimale de KP′.

- En supposant, en premier lieu, que $x_k^* = 1$ et $\Delta p_k \geq 0$, il s'agit de montrer que x^* reste une solution optimale.

Soit $\hat{x} = (\hat{x}_1, ..., \hat{x}_k, ..., \hat{x}_n)$ une solution réalisable de KP.

Il est facile de voir que pour $x_k^* = 1$, on a $\hat{x}_k \leq x_k^*$.

Puisque $\Delta p_k \geq 0$, alors $0 \leq \Delta p_k \hat{x}_k \leq \Delta p_k x_k^*$.

Comme $\hat{x}_k$ est une solution réalisable et x_k^* une solution optimale de valeur Z^*, on a $\hat{Z}(\hat{x}) \leq Z^*$.

Par conséquent, $\hat{Z}(\hat{x}) + \Delta p_k \hat{x}_k \leq Z^* + \Delta p_k x_k^*$. Ce qui est équivalent à $\hat{Z}'(\hat{x}) \leq \hat{Z}'(x^*)$.

D'où x^* est une solution optimale de KP′.

- En second, en supposant que $x_k^* = 0$ et $\Delta p_k \leq 0$, il suffit de montrer que x^* demeure une solution optimale de KP′.

$\hat{x} = (\hat{x}_1, ..., \hat{x}_k, ..., \hat{x}_n)$ étant une solution réalisable de KP.

Puisque $\Delta p_k \leq 0$, alors $\Delta p_k \hat{x}_k \leq 0$. D'où $\hat{Z}(\hat{x}) + \Delta p_k \hat{x}_k \leq Z^*$.

Posons $x_k^* = 0$ et comme $\hat{Z}(\hat{x}) \le Z^*$, on a alors $Z^* + \Delta p_k x_k^* = Z^*$.

L'inéquation $\hat{Z}(\hat{x}) + \Delta p_k \hat{x}_k \le Z^*$ est alors équivalente à $\hat{Z}'(\hat{x}) \le \hat{Z}'(x^*)$.

D'où x^* est une solution optimale de KP′.

Il est montré aussi que si (ii) est vérifiée alors x^* est une solution optimale de KP′ ;

Soit $\hat{x}$ une solution réalisable de KP, on a $\hat{Z}(\hat{x}) \le Z^*$.

En ajoutant $\Delta p_k \hat{x}_k$ aux deux côtés de l'inégalité, on obtient $\hat{Z}(\hat{x}) + \Delta p_k \hat{x}_k \le Z^* + \Delta p_k \hat{x}_k$.

Comme $\hat{Z}(\hat{x}) + \Delta p_k \hat{x}_k = \hat{Z}'(\hat{x})$ (Résultat 3.1) alors $\hat{Z}'(\hat{x}) \le Z^*(x^*) + \Delta p_k \hat{x}_k$.

Sachant que la variable $\hat{x}_k$ est binaire, alors :

- Pour $\hat{x}_k = 1$, on a $\Delta p_k \hat{x}_k = \Delta p_k x_k^* \Rightarrow \hat{Z}'(\hat{x}) \le Z^* + \Delta p_k x_k^*$.

on sait que $Z^* + \Delta p_k x_k^* = \hat{Z}'(x^*)$, d'où $\hat{Z}'(\hat{x}) \le \hat{Z}'(x^*)$.

- pour $\hat{x}_k = 0$, on a $\hat{Z}(\hat{x}) \le \hat{Z}(\hat{\omega}_0)$

l'équation $\hat{Z}(\hat{x}) + \Delta p_k \hat{x}_k = \hat{Z}'(\hat{x})$ devient $\hat{Z}(\hat{x}) = \hat{Z}'(\hat{x})$, d'où $\hat{Z}'(\hat{x}) \le \hat{Z}(\hat{\omega}_0)$,

en supposant $\Delta p_k \ge \hat{Z}(\hat{\omega}_0) - Z^*$ avec $\Delta p_k \le \Delta p_k x_k$,

on déduit alors que $\hat{Z}'(\hat{x}) \le \hat{Z}'(x^*)$.

Quant au cas (iii) la démonstration se fait de la même manière que (ii) (voir [25]). ■

Le fait de perturber le profit p_k implique la modification de la valeur optimale de KP. Ainsi, dans le résultat 3.2 les auteurs permettent d'assurer la stabilité de la solution x^* de manière à ce que si $x_k^* = 0$ (resp, $x_k^* = 1$), la valeur objective $\hat{Z}(\hat{\omega}_1)$ (resp, $\hat{Z}(\hat{\omega}_0)$) ne dépasse pas la valeur Z^*.

Ce dernier a été résumé dans le résultat suivant :

Résultat 3.3 [25]

Soit x^* une solution optimale de KP. x^* demeure une solution optimale de KP′ telle que :

(i) *si* $x_k^* = 1$, *alors* $\Delta p_k \in \left[\hat{Z}(\hat{\omega}_0) - Z^*, +\infty\right[$

(ii) *si* $x_k^* = 0$, *alors* $\Delta p_k \in \left[1 - p_k, Z^* - \hat{Z}(\hat{\omega}_1)\right]$.

où $\hat{\omega}_0$ (resp. $\hat{\omega}_1$) est la meilleure parmi les solutions réalisables de KP dont la $k^{ème}$ variable est nulle (resp. égale à 1*).*

Notons que $\hat{Z}(\hat{\omega}_0)$ et $\hat{Z}(\hat{\omega}_1)$ sont les valeurs optimales des problèmes $KP \setminus \{k\}$ et $KP[c - w_k] \setminus \{k\}$ respectivement, où :

- $KP \setminus \{k\}$ est une instance de KP obtenue en enlevant le $k^{ème}$ objet, et
- $KP[c - w_k] \setminus \{k\}$ est une instance de KP, de capacité $c - w_k$, obtenue en enlevant le $k^{ème}$ objet.

Il est clair que la détermination des bornes de l'intervalle de sensibilité I_k passe par le calcul des deux valeurs $\hat{Z}(\hat{\omega}_0)$ et $\hat{Z}(\hat{\omega}_1)$. Or, ceci est aussi difficile que de résoudre le problème perturbé KP′. Par conséquent, on utilise à la place de $\hat{Z}(\hat{\omega}_0)$ et $\hat{Z}(\hat{\omega}_1)$ les bornes supérieures $UB(KP \setminus \{k\})$ et $UB(KP[c - w_k] \setminus \{k\})$ des problèmes $KP \setminus \{k\}$ et $KP[c - w_k] \setminus \{k\}$ respectivement. Ces bornes peuvent être calculées grâce à n'importe quelle approche de la littérature. Dans [25], les auteurs ont utilisé la borne supérieure de Dantzig et proposent un algorithme pour le calcul de l'intervalle de sensibilité en se basant sur la méthode résolution de KP, soit la méthode de résolution par branch and bound.

Algorithme 3.5 : Algorithme pour la détermination de l'intervalle I_k .

Notations :

- $KP \setminus \{k\}$ est le problème KP de capacité *c* avec $x_k = 0$.
- $KP[c = c - w_k] \setminus \{k\}$ est le problème KP de capacité $c - w_k$ avec $x_k = 1$.

Entrée : une solution optimale x^* de KP, sa valeur optimale $Z(x^*)$ et l'élément perturbé *k*.

Sortie : l'intervalle $I_k = [I_k^-, I_k^+]$

1. **Poser** $x'^* = x^*$.
2. **Si** $x_k^* = 1$ **alors**
   $I_k^- := p_k + calcul\_borne(KP \setminus \{k\}) - Z(x^*)$
   $I_k^+ := +\infty$
   FinSi
3. **Si** $x_k^* = 0$ **alors**
   $I_k^- := 1$;
   $I_k^+ := Z(x^*) - calcul\_borne(KP[c = c - w_k] \setminus \{k\})$
   FinSi

Remarque :

Dans cet algorithme, les auteurs font appel à la procédure *calcul_borne*( ) pour générer la borne supérieure d'un problème KP donné. La complexité de cet algorithme est la même que celle de la procédure *calcul_borne*( ). Comme les bornes que nous avons utilisé sont toutes de complexité $O(n)$, alors il en est de même pour la complexité de l'algorithme.

que celle de la procédure *calcul_borne*( ). Comme les bornes que nous avons utilisé sont toutes de complexité $O(n)$ alors il en est de même pour la complexité de l'algorithme.

3.7 Résultats expérimentaux

Dans cette section, on commence par donner une illustration de l'algorithme vu précédemment (Algorithme 3.5) en utilisant une instance de KP de petite taille [8]. Ensuite, on donne une analyse expérimentale de cet algorithme à travers des instances de grande taille dont les données sont tirées aléatoirement.

La table 3.1 fournit les intervalles de sensibilité pour une instance [8] de 15 objets, dont la capacité est égale à 145. Nous reprenons dans les colonnes 2 et 3 les données du problème (les profits et les poids). Dans la colonne 4, on retrouve la solution optimale de ce dernier. Et dans les autres colonnes (5, 6,…,14), nous dressons une comparaison entre les intervalles de sensibilité des profits de notre problème, calculés en appliquant l'algorithme (Algorithme 3.5) en introduisant différentes bornes supérieures de la solution optimale du problème.

Les objets sont classés dans un ordre décroissant en fonction du rapport profit sur unité de poids $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq ... \geq \frac{p_{15}}{w_{15}}$.

Concernant les bornes des intervalles de sensibilité de profits, nous avons la borne triviale (i.e. $I_k^+ = +\infty$ lorsque $x_k^* = 1$ et $I_k^+ = 0$ lorsque $x_k^* = 0$). La seconde borne est calculée par l'algorithme (Algorithme 3.5).

On constate sur cet exemple que les cinq borne supérieur de notre problème KP (i.e. borne de Dantzig, de Martello & Toth, de Hudson et Fayard & Plateau, de Müller-Merbach et celle de Dudzinski & Walukiewcz) permettent d'obtenir les mêmes intervalles de sensibilité pour les profits des objets {1, 2, 3, 4, 5, 8, 9, 11, 12 et 13} alors que pour certains d'entre eux {6, 7, 10 et 14} la borne de Dudzinski & Walukiewcz permet d'obtenir des intervalles de sensibilité relativement meilleurs.

On peut aussi constater que pour l'objet 1, par exemple, le profit $p_1 = 37$ peut varier entre 3 et $+\infty$ sans que la solution optimale en main ne soit affectée. Alors que pour les objets 8 et 9, on a $I_8^+ = p_8 = 14$ et $I_9^+ = p_9 = 25$ ce qui signifie que les profits p_8 et p_9 ne peuvent varier sans que la solution optimale du problème ne change.

On peut aussi conclure que les petites variations tolérables observées sur pratiquement l'ensemble des objets permettent de démontrer la robustesse de cette solution optimale.

Table 3.1 : Intervalles de sensibilité pour une instance de petite taille (quinze objets).

				$I_k(D)$		$I_k(MT)$		$I_k(HFP)$		$I_k(MM)$		$I_k(DW)$	
objet	poids	profit	x_k	I_k^-	I_k^+	I_k^-	I_k^+	I_k^-	I_k^+	I_k^-	I_k^+	I_k^-	I_k^+
1	1	37	1	3	infini	3	infini	3	infini	3	infini	3	infini
2	4	17	1	5	infini	5	infini	5	infini	5	infini	5	infini
3	14	30	1	11	infini	11	infini	11	infini	11	infini	11	infini
4	10	21	1	9	infini	9	infini	9	infini	9	infini	9	infini
5	12	23	1	10	infini	10	infini	10	infini	10	infini	10	infini
6	27	24	1	18	infini	18	infini	18	infini	18	infini	15	infini
7	32	24	1	21	infini	21	infini	21	infini	21	infini	18	infini
8	21	14	0	1	14	1	14	1	14	1	14	1	14
9	40	25	0	1	25	1	25	1	25	1	25	1	25
10	37	19	0	1	21	1	21	1	21	1	22	1	24
11	37	16	0	1	21	1	21	1	21	1	21	1	21
12	25	10	0	1	13	1	13	1	13	1	13	1	13
13	21	8	0	1	11	1	11	1	11	1	11	1	11
14	5	4	1	4	infini	4	infini	4	infini	3	infini	3	infini
15	21	2	0	1	11	1	11	1	11	1	11	1	11

Dans les tables 3.2, 3.3 et 3.4, trois instances sont considérées :

- Faiblement corrélées : w_j tirés uniformément dans $[1, 100]$, p_j tirés uniformément dans $[w_j - 10, w_j + 10]$;
- Fortement corrélées : w_j tirés uniformément dans $[1, 100]$, $p_j = w_j + 10$
- Instances non corrélées : w_j et p_j tirés uniformément dans $[1, 100]$.

Pour toutes les données, la valeur de la capacité *c* est prise égale à $0.5\sum_{j=1}^{n} w_j$ (ainsi près de la moitié des objets sont inclus dans le sac).

Toutes les exécutions sont réalisées sur un Pentium (R) Dual-Core CPU 2GHz. Les tailles des instances générées varient entre 50 et 400 (au-delà la machine se met à planter). Pour chacune des instances générées, on calcule les entités suivantes :

- $\overline{\Delta^-}$: la déviation négative moyenne du profit de l'objet inclus dans le sac ;

$\overline{\Delta^-} = 1/n_1 \sum_{k=1}^{n} (p_k - I_k^-) x_k^*$; où n_1 représente le nombre d'objets inclus dans le sac.

- $\overline{\Delta}^{+}$: la déviation positive moyenne du profit de l'objet non inclus dans le sac ;

$$\overline{\Delta}^{+} = 1/(n-n_1)\sum_{k=1}^{n}-(p_k - I_k^{+})x_k^{*} \ .$$

Les résultats montrent que les déviations moyennes positive et négative dépendent de la taille des instances du problème.

Table 3.2 : Instances faiblement corrélées : w_j tirés uniformément dans $[1, 100]$, p_j tirés uniformément dans $[w_j - 10, w_j + 10]$ et $c = 0.5\sum_{j=1}^{n} w_j$. Pentium (R) Dual-Core CPU 2GHz.

Moyennes prises sur 10 instances du problème KP

	$\overline{\Delta}\ (D)$		$\overline{\Delta}\ (MT)$		$\overline{\Delta}\ (HFP)$		$\overline{\Delta}\ (MM)$		$\overline{\Delta}\ (DW)$	
n	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$
50	1220	1211	1220	1212	1220	1212	1220	1211	1220	1211
100	2473	2469	2474	2469	2474	2470	2473	2469	2473	2469
200	4789	4785	4789	4785	4789	4785	4789	4785	4789	4785
300	7382	7383	7382	7383	7382	7383	7382	7383	7382	7383
400	9915	9911	9915	9911	9915	9911	9915	9911	9915	9911

Table 3.3 : Instances fortement corrélées : w_j tirés uniformément dans $[1, 100]$, $p_j = w_j + 10$, $c = 0.5\sum_{j=1}^{n} w_j$. Pentium (R) Dual-Core CPU 2GHz.

Moyennes prises sur 10 instances du problème KP

	$\overline{\Delta}\ (D)$		$\overline{\Delta}\ (MT)$		$\overline{\Delta}\ (HFP)$		$\overline{\Delta}\ (MM)$		$\overline{\Delta}\ (DW)$	
n	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$
50	273	269	273	269	273	269	273	269	273	269
100	1180	1176	1180	1176	1180	1176	1180	1176	1180	1176
200	2454	2450	2454	2450	2454	2450	2454	2450	2454	2450
300	2697	2692	2697	2692	2697	2692	2697	2692	2697	2692
400	2976	2970	2976	2970	2976	2970	2976	2970	2976	2970

Table 3.2 : Instances non corrélées : w_j et p_j tirés uniformément dans $[1, 100]$, $c = 0.5\sum_{j=1}^{n} w_j$. Pentium (R) Dual-Core CPU 2GHz.

Moyennes prises sur 10 instances du problème KP

	$\overline{\Delta}$ (D)		$\overline{\Delta}$ (MT)		$\overline{\Delta}$ (HFP)		$\overline{\Delta}$ (MM)		$\overline{\Delta}$ (DW)	
n	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$	$\overline{\Delta}^{-}$	$\overline{\Delta}^{+}$
50	310	280	311	301	312	302	310	300	310	301
100	629	600	629	611	629	611	629	611	629	611
200	1149	1100	1149	1125	1150	1125	1149	1125	1149	1125
300	1706	1657	1706	1683	1707	1683	1706	1683	1706	1683
400	2186	2145	2186	2160	2186	2160	2186	2160	2186	2160

4.1 Introduction

Le problème du knapsack borné (BKP *bounded knapsack problem*) est une généralisation du problème KP. Il peut être lui aussi considéré comme un problème à part ou comme un sous-problème ou une relaxation de problèmes d'optimisation combinatoire plus complexes.

Contrairement à KP, peu de travaux ont été consacrés au problème BKP. Nous citons ci-après quelques-uns des algorithmes présents dans la littérature.

En 1966, **Gilmore et Gomory** [23] utilisent la programmation dynamique pour résoudre les problèmes de type knapsack dont BKP. **Nemhauser et Ullmann** [49] améliorent en 1969 la relation récursive employée par **Gilmore et Gomory** mais rapportent dans leur article que la programmation dynamique est inefficace dans le cas de problèmes de grande taille.

En 1977, face à ce type de problèmes (de grande taille), **Martello et Toth** [40] proposent une approche branch and bound pour BKP en adaptant celle proposée pour KP [39]. Dans la même année, **Ingargiola et Korsh** [31] présentent un algorithme apparenté à celui proposé pour KP dans leur article de 1973 [30].

Bulfin, Parker et Shetty [10] proposent en 1979 une stratégie différente de branch and bound en introduisant des pénalités dans le but d'améliorer les bornes.

Aittoniemi [1] (1982) montre à travers une comparaison expérimentale que l'algorithme de **Martello et Toth** [43] est jusqu'à ce jour le meilleur algorithme de résolution pour BKP.

En 1990, **Martello et Toth** [43] mettent au point un algorithme qui transforme BKP en KP équivalent et démontrent que ce dernier est meilleur que tous les autres algorithmes proposés auparavant.

Cependant, cette transformation de BKP en KP correspondant engendre beaucoup d'objets ayant des poids proportionnels, ce qui rend le problème ainsi obtenu plus difficile à résoudre dans le cas d'instances de très grande taille. **Pisinger** [53] propose alors un algorithme pour résoudre BKP à travers la programmation dynamique, en réduisant la taille des données.

Dans la suite de ce chapitre, nous allons utiliser la transformation de Martello et Toth pour la résolution de BKP mais aussi pour l'analyse de la sensibilité de la solution optimale de ce dernier et nous proposons un algorithme pour le calcul d'un intervalle de sensibilité du profit d'un objet arbitraire. On se rapporte pour ce faire, aux travaux de **Belgacem et Hifi** [8] concernant l'analyse de la sensibilité du problème KP.

4.2 Présentation et formulation du problème

Nous rappelons que BKP est défini par la donnée de *n* objets à placer dans un sac de capacité *c*. chaque objet de type *j* ayant un profit p_j, un poids w_j et une borne b_j sur la disponibilité (avec p_j, w_j et b_j des entiers positifs). L'objectif étant de sélectionner un nombre d'exemplaires x_j ($0 \le x_j \le b_j$) de chaque type d'objets tel que le total des profits des objets sélectionnés soit maximisé et, ceci sans violer la contrainte capacité. Le problème peut être formulé comme suit :

$$(\text{BKP}) \begin{cases} \text{maximiser} & z(x) = \sum_{j=1}^{n} p_j x_j & \\ \text{s.c} & \sum_{j=1}^{n} w_j x_j \le c & \\ & 0 \le x_j \le b_j & \text{pour } j = 1,...,n \\ & x_j \textit{ entier et } x_j \ge 0 & \text{pour } j = 1,...,n \end{cases}$$

On suppose, sans perte de généralité, que :

$c > 0$, $p_j > 0$, $w_j > 0$ et $b_j > 0$, $j = 1, ..., n$

$\sum_{j=1}^{n} b_j w_j > c$, et $b_j w_j \le c$, $j = 1, ..., n$

Remarque :

BKP est un problème NP-difficile (section 2.3.2) puisque KP en est un cas particulier pour $b_j = 1,\ j = 1,2,...,n$.

4.3 Résolution du problème BKP

4.3.1 Transformation de BKP en KP

En 1990, **Martello et Toth** [43] mettent au point une approche assez élégante pour résoudre BKP en se basant sur le codage binaire de la borne b_j ($j = 1,...,n$) afin d'obtenir un knapsack à variables binaires.

L'idée de l'algorithme est la suivante, en supposant au préalable que les objets sont classés selon l'ordre décroissant suivant :

$$\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq ... \geq \frac{p_n}{w_n}$$

l'algorithme en question (Algorithme 4.1) transforme BKP en un KP équivalent avec :

$\hat{n}$: nombre de variables,

$\hat{p} = \left(\hat{p}_j\right)_{1 \leq j \leq \hat{n}}$: vecteur profit,

$\hat{w} = \left(\hat{w}_j\right)_{1 \leq j \leq \hat{n}}$: vecteur poids,

$\hat{c} = c$: capacité du sac.

Pour chaque type j d'objets de BKP, une série de $\lfloor \log_2 b_j + 1 \rfloor$ objets est introduite, leurs profits et poids sont respectivement $(p_j, w_j), (2p_j, 2w_j), (4p_j, 4w_j), ..., (2^{a-1}p_j, 2^{a-1}w_j), (dp_j, dw_j)$ avec $a = \lfloor \log_2 b_j \rfloor$, $d = b_j - \sum_{i=0}^{a-1} 2^i$.

Dans la figure ci-dessous (fig. 4.1), nous décrivons l'algorithme de transformation de Martello et Toth.

Algorithme 4.1 : Algorithme de transformation de BKP en KP.

Entrée : n, p_j, w_j, b_j

Sortie : $\hat{n}, \hat{p}_j, \hat{w}_j$

1. $\hat{n} = 0$;
2. **Pour** $j := 1$ **jusqu'à** n **faire**
   $\beta = 0$;
   $k = 1$;
   Tant que $(\beta < b_j)$ **faire**
   Si $(\beta + k > b_j)$ **alors** $k = b_j - \beta$;
   FinSi
   $\hat{p}[\hat{n}] = kp[j]$;
   $\hat{w}[\hat{n}] = kw[j]$;
   $\beta = \beta + k$;
   $k = 2 * k$;
   $\hat{n} = \hat{n} + 1$;
   FinTant que
   FinPour
3. **Sortir avec un problème KP avec** $\hat{n}$ **objets de profit et poids** $\hat{p}_j, \hat{w}_j$, $j = 1,...\hat{n}$

Remarque :

Le problème KP obtenu admet $\hat{n} = \sum_{j=1}^{n} \lceil \log_2(b_j + 1) \rceil$ variables binaires et l'algorithme de transformation de Martello et Toth est de complexité $O(\hat{n})$ [42].

Afin de voir que les deux problèmes sont équivalents [42], soient $x_{j_1},...,x_{j_q}$ (avec $q = \lceil \log_2(b_j + 1) \rceil$) les q variables binaires introduites à la place de x_j telles que j_h correspond aux n_h objets de type *j*, tel que $n_h = \begin{cases} 2^{h-1} & h < q \\ b_j - \sum_{i=1}^{q-1} 2^{i-1} & h = q \end{cases}$

Ainsi $x_j = \sum_{h=1}^{q} n_h \hat{x}_{j_h}$ peut prendre n'importe quelle valeur entière entre 0 et b_j.

Cette transformation introduit 2^q combinaisons binaires, soit $2^q - (b_j + 1)$ représentations redondantes pour les valeurs possibles de x_j (les valeurs de n_q à $2^{q-1} - 1$ ont une représentation double). Puisque q est le nombre minimum de variables binaires nécessaires à la représentation des entiers de 0 à b_j, alors toute transformation alternative doit introduire le même nombre de redondances.

Vu l'étroite relation qui existe entre KP et BKP, toutes les techniques mathématiques et algorithmiques pour KP peuvent être étendues à BKP [42].

4.3.2 Résolution exacte

Les algorithmes pour la résolution exacte de BKP sont peu nombreux dans la littérature à comparer avec ceux pour KP. La raison étant que la meilleure façon de résoudre les problèmes knapsacks bornés (hormis les problèmes de très grandes tailles) est de les transformer en knapsacks de forme 0-1 en appliquant l'algorithme de transformation de Martello et Toth énoncé ci-dessus.

Il est néanmoins possible de développer des approches propres à BKP en adaptant celles existant pour KP.

En 1977, **Martello et Toth** [40] adaptent leur approche branch and bound développée pour résoudre KP à la résolution de BKP. Dans la même année, **Ingargiola et Korsh** [31] présentent un algorithme apparenté à celui proposé pour KP dans leur article de 1973 [30].

Bulfin, Parker et Shetty [10] présentent en 1979 une nouvelle stratégie branch and bound différente en introduisant des pénalités afin d'améliorer les bornes.

Aittoniemi [1] (1982) montre à travers une comparaison expérimentale que l'algorithme de **Martello et Toth** [34] est jusqu'à ce jour le meilleur algorithme de résolution pour BKP.

4.3.3 Bornes supérieures

Martello et Toth **[43]** donnent la solution optimale x^* de la relaxation continue de BKP, où $0 \le x_j \le b_j$, et $x_j \in IR$, directement en utilisant le théorème 3.1.

En supposant que les objets sont classés selon l'ordre décroissant du rapport profit/poids ;

$$\frac{p_1}{w_1} \ge \frac{p_2}{w_2} \ge ... \ge \frac{p_n}{w_n}$$

Soit $s = \min\left\{ j : \sum_{i=1}^{j} b_i w_i < c \right\}$ l'élément critique,

La solution est donnée par :

$$x^*_j = 1 \qquad \text{pour } j = 1,...,s-1,$$
$$x^*_j = 0 \qquad \text{pour } j = s+1,...,n,$$
$$x^*_s = \frac{\bar{c}}{w_s} \qquad \text{où } \bar{c} = c - \sum_{j=1}^{s-1} b_j w_j$$

et la valeur optimale $z^* = \sum_{j=1}^{s-1} b_j p_j + \bar{c}\frac{p_s}{w_s}$

La borne supérieure est dans ce cas $U_1 = \sum_{j=1}^{s-1} b_j p_j + \left\lfloor \bar{c}\frac{p_s}{w_s} \right\rfloor$

Une borne plus serrée est trouvée par Martello et Toth (en 1977).

Soit $z' = \sum_{j=1}^{s-1} b_j p_j + \left\lfloor \frac{\bar{c}}{w_s} \right\rfloor p_s$ le profit total obtenu en sélectionnant *bj* objets *j*=1,…,*s*-1 et $\lfloor \bar{x}_s \rfloor$ du s^{ieme} objet.

La capacité résiduelle devient $c' = \bar{c} - \left\lfloor \frac{\bar{c}}{w_s} \right\rfloor p_s$

$U^0 = z' + \left\lfloor c'\frac{p_{s+1}}{w_{s+1}} \right\rfloor$ est une borne supérieure si on ne sélectionne pas d'objets supplémentaires du type s.

U^1 serait une borne supérieure si on en sélectionne exactement 1 de plus, avec

$$U^1 = z' + \left\lfloor p_s - (w_s - c')\frac{p_{s-1}}{w_{s-1}} \right\rfloor$$

et $U_2 = \max(U^0, U^1)$ est une autre borne supérieure pour BKP

Remarque :

- La complexité de U_1 et U_2 est en *O(n)*, une fois les éléments classés.
- Le calcul de U_1 pour BKP sous la forme en 0-1 donne les mêmes résultats que pour la relaxation continue de BKP.
- Il n'en est pas de même pour U_2 puisque U^0 et U^1 sont plus serrées que pour la forme en 0-1.

4.4 Etude de la sensibilité de la solution optimale face à la perturbation du profit

Dans cette section, nous nous intéressons à l'analyse de la sensibilité du problème BKP et nous présentons un algorithme pour le calcul d'un intervalle de sensibilité dans le cas de la perturbation d'un profit arbitraire.

Soit le problème (BKP) définit précédemment :

$$(\text{BKP}) \begin{cases} \text{maximiser } z(x) = \sum_{j=1}^{n} \text{p}_j x_j \\ \text{s.c} \quad \sum_{j=1}^{n} w_j x_j \le c \\ \qquad 0 \le x_j \le b_j \qquad \text{pour } j = 1,...,n \\ \qquad x_j \text{ entier} \qquad \text{pour } j = 1,...,n \end{cases}$$

et soit (BKP´) le problème perturbé obtenu à partir de (BKP) tel que le profit du k^{eme} objet p_k est remplacé par $\text{p}'_k = \text{p}_k + \Delta\text{p}_k$, $k = 1,...,n$, avec Δp_k un entier équivalent à la variation du profit de l'objet de type k. On note par $\theta_k = [\ \theta_k^-, \theta_k^+\ ]$ l'intervalle de sensibilité de p_k.

(BKP´) est formulé comme suit :

$$(\text{BKP}')\begin{cases} \text{maximiser} & z(x)=\sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{p}_j x_j + (\mathrm{p}_k + \Delta \mathrm{p}_k)x_k \\ \text{s.c} & \sum_{\substack{j=1\\ j\neq k}}^{n} w_j x_j + w_k x_k \le c \\ & 0 \le x_j \le b_j \qquad \text{pour } j=1,...,n \\ & x_j \textit{ entier} \qquad \text{pour } j=1,...,n \end{cases}$$

Lors de la transformation de BKP en KP équivalent, les objets de type k, $k=1,...,n$, de profit p_k sont remplacés par $\lfloor \log_2 b_k + 1 \rfloor$ objets de profits $\hat{\mathrm{p}}_{k1}, \hat{\mathrm{p}}_{k2}, ..., \hat{\mathrm{p}}_{kq}$, (avec $q = \lfloor \log_2 b_k + 1 \rfloor$), et la perturbation du profit de ces objets dans le problème BKP revient à la perturbation des profits des $\lfloor \log_2 b_j + 1 \rfloor$ objets correspondants dans KP. Ainsi, l'analyse de sensibilité de BKP avec la perturbation d'un profit p_k peut être ramenée à l'analyse de sensibilité du KP correspondant avec la perturbation des profits $\hat{\mathrm{p}}_{k1}, \hat{\mathrm{p}}_{k2}, ..., \hat{\mathrm{p}}_{kq}$ du sous-ensemble des *q* objets introduits.

Dans ce qui suit, nous proposons un algorithme qui permet de calculer l'intervalle $\theta_k = [\ \theta_k^-, \theta_k^+\ ]$ en perturbant un poids arbitraire d'un type d'objets dans BKP.

L'idée de cet algorithme consiste, une fois BKP transformé en KP équivalent, à :

1. Calculer les intervalles de sensibilité des profits $\hat{\mathrm{p}}_{k1}, ..., \hat{\mathrm{p}}_{kq}$ (avec $q = \lceil \log_2 (b_j + 1) \rceil$) obtenus dans KP,
2. Déterminer un seul intervalle de sensibilité pour le sous-ensemble des $\lfloor \log_2 b_j + 1 \rfloor$ profits. Cet intervalle étant limité par la plus petite perturbation des profits des objets du sous-ensemble considéré.
3. l'intervalle de sensibilité calculé en 2. est lui-même l'intervalle de sensibilité θ_k du profit p_k.

Belgacem et Hifi [7] sont les premiers à avoir fait une analyse de la sensibilité d'un problème KP dans le cas de la perturbation simultanée des profits d'un sous-ensemble d'objets sachant que ces perturbations sont indépendantes les unes des autres. Dans [7], les auteurs déterminent les limites des intervalles de sensibilité des profits perturbés à travers deux approches complémentaires. L'une permet de traiter de manière séparée chacun des objets, c'est-à-dire indépendamment des autres objets perturbés. Dans la seconde approche, les auteurs établissent une condition non pas globale mais spécifique sur les perturbations, et qui permet d'assurer la stabilité de la solution optimale du problème perturbé.

Reprenons le problème KP défini précédemment,

$$(\text{KP}) \begin{cases} \text{maximiser} \quad z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} \qquad \sum_{j=1}^{n} w_j x_j \le c \\ \qquad\quad x_j \in \{0,1\} \qquad \text{pour} \quad j = 1,...,n \end{cases}$$

Soit $\Gamma \subseteq N$ un sous-ensemble de i objets. $\forall l \in \Gamma, \Delta p_l$ représente la perturbation que subit le profit du $l^{ème}$ objet.

Soit KP′ le problème perturbé obtenu en faisant varier les profits des i objets de l'ensemble Γ. C'est-à-dire, en posant $p'_l = p_l + \Delta p_l$, $l \in \Gamma$.

KP′ est dans ce cas définit comme suit :

$$(\text{KP}') \begin{cases} \text{maximiser} \quad z(x) = \sum_{j \in N \setminus \Gamma} p_j x_j + \sum_{j \in \Gamma} (p_j + \Delta p_j) x_j \\ \text{s.c} \qquad \sum_{j=1}^{n} w_j x_j \le c \\ \qquad\quad x_j \in \{0,1\} \qquad \text{pour} \quad j \in N = \{1,...,n\} \end{cases}$$

Pour tout $l \in \Gamma$, $\Delta_l = \left[\Delta_l^-, \Delta_l^+\right]$ est l'intervalle de sensibilité correspondant au profit p_l.

Les résultats suivants sont une généralisation de ce qu'on a vu au chapitre précédant concernant l'analyse de sensibilité de KP.

Reprenant ce premier résultat sur les régions faisables et les valeurs optimales des problèmes KP et KP′.

Proposition 4.1 [7]

(i) *KP et KP′ ont le même ensemble de solutions réalisables.*

(ii) *Soit* x^* (*resp.* x'^*) *une solution optimale de KP* (*resp. KP′*), *alors* $Z'(x'^*) = Z(x^*) + \sum_{j\in\Gamma} \Delta p_j x_j^*$ *lorsque* x^* *et* x'^* *coïncident.*

Preuve

(i) Chaque objet $l \in N$ a le même poids dans KP et KP′, et la capacité *c* du sac est la même dans les deux problèmes. Par conséquent, il est évident que KP et KP′ ont le même ensemble de solutions réalisables.

(ii) Sachant que $Z(x^*) = \sum_{j=1}^{n} p_j x_j^*$ et $Z'(x'^*) = \sum_{j=1}^{n} p'_j x'^*_j$, et puisque $x^* = x'^*$, on a alors :

$$\begin{aligned} Z'(x'^*) &= \sum_{j=1}^{n} p'_j x_j^* \\ &= \sum_{j\in N\setminus\Gamma} p_j x_j^* + \sum_{j\in\Gamma} (p_j + \Delta p_j) x_j^* \\ &= \sum_{j=1}^{n} p_j x_j^* + \sum_{j\in\Gamma} \Delta p_j x_j^* \\ &= Z(x^*) + \sum_{j\in\Gamma} \Delta p_j x_j^* \end{aligned}$$

■

Dans [7], les auteurs partitionnent l'ensemble Γ en 4 sous-ensembles où chaque sous-ensemble dépend de la perturbation imposée et de la valeur de la variable d'état correspondant à cette perturbation dans la solution optimale.

$\Gamma = \Gamma_0^- \cup \Gamma_0^+ \cup \Gamma_1^- \cup \Gamma_1^+$, *où*

- $\Gamma_0^- = \left\{ l \in \Gamma \middle| x_l^* = 0 \; et \; \Delta p_l \le 0 \right\}$
- $\Gamma_0^+ = \left\{ l \in \Gamma \middle| x_l^* = 0 \; et \; \Delta p_l \ge 0 \right\}$
- $\Gamma_1^- = \left\{ l \in \Gamma \middle| x_l^* = 1 \; et \; \Delta p_l \le 0 \right\}$
- $\Gamma_1^+ = \left\{ l \in \Gamma \middle| x_l^* = 1 \; et \; \Delta p_l \ge 0 \right\}$

et les intersections deux à deux de ces quatre ensembles sont vides.

Les auteurs dans [7], commencent tout d'abord par établir que si le calcul des limites de l'intervalle de sensibilité du profit de chacun des objets de Γ ne dépend pas des autres objets perturbés, alors ces limites constituent des limites de sensibilité valides dans le cas où les profits des objets de l'ensemble Γ sont perturbés de manière simultanée. Ensuite, ils étudient l'influence de la perturbation de chaque objet de Γ sur les autres en considérant les quatre sous-ensembles $\Gamma_0^-, \Gamma_0^+, \Gamma_1^-$ et Γ_1^+ de Γ.

Théorème 4.1 [7]

Soit x^* une solution optimale de KP. x^* est aussi une solution optimale de KP´dans les cas suivants :

(1) $\Gamma = \Gamma_0^- \cup \Gamma_1^+$.

(2) $\Gamma = \Gamma_1^-$ et $\sum_{j \in \Gamma} \Delta p_j \ge VO(KP \setminus \{a\}) - VO(KP)$.

où $a = \arg\max_{j \in \Gamma} \{VO(KP \setminus \{j\})\}$.

(3) $\Gamma = \Gamma_0^+$ et $\sum_{j \in \Gamma} \Delta p_j \ge VO(KP) - VO(KP \setminus \{b\} \; [c - w_b] \;) - p_b$.

où $b = \arg\max_{j \in \Gamma} \{ \; VO(KP \setminus \{b\} \; [c - w_b] \;) + p_j \}$.

L'ensemble Γ peut être obtenu par la combinaison de ces trois cas. Si par exemple $\Gamma = \Gamma_1^- \cup \Gamma_0^+$ la perturbation des profits des objets de Γ, dans ce cas, est limitée par la plus petite perturbation individuelle obtenue en perturbant chacun de ces objets.

Preuve

Pour la preuve de ce théorème le lecteur peut se référé à l'article **[7]**.

Dans le théorème suivant, les auteurs proposent une amélioration des limites de l'intervalle de sensibilité de p_l dans les cas (2) et (3). Ils procèdent d'une manière différente. Ils donnent une condition spécifique sur la perturbation du profit de chaque objet de l'ensemble Γ, en utilisant la borne de Dantzig.

Théorème 4.2 [8]

Soit x^* *une solution optimale de KP et* KP'. *Soit* $\Gamma = \Gamma_1^- \cup \Gamma_0^+$, *alors :*

(i) $\Delta_l^- = \max\left\{I_l^-, w_l \times \min_{l\in\Gamma}\{\rho_l\} - p_l\right\}$ *lorsque* $l \in \Gamma_1^-$.

(ii) $\Delta_l^+ = \min\left\{I_l^+, w_l \times \max_{l\in\Gamma}\{\rho_l\} - p_l\right\}$ *lorsque* $l \in \Gamma_0^+$.

Où

$Cr(KP)$ *représente l'objet critique correspondant à la borne de Dantzig du problème KP,*

r_l *(resp,* r_l' *) est le ratio* $\frac{p_l}{w_l}$ *(resp.* $\frac{p_l'}{w_l}$*) et,*

ρ_l *est le ratio critique de l'objet* $l \in \Gamma$, calculé comme suit :

$$\rho_l = \begin{cases} r_{Cr(KP\setminus\{l\})} & x_l^* = 1 \\ r_{Cr(KP\setminus\{l\}[\hat{c}-\hat{w}_l])} & x_l^* = 0 \end{cases}$$

Preuve

Pour la preuve de ce résultat le lecteur peut se référé à l'article [8].

Ci-après l'algorithme détaillé pour le calcul d'un intervalle de sensibilité de KP avec la perturbation d'un sous-ensemble d'objets.

Algorithme 4.2 : Algorithme pour le calcul de Δ_l.

Entrée : une instance de KP, x^*, Γ et le signe de Δp_l, $\forall l \in \Gamma$.
Sortie : l'intervalle de sensibilité $\Delta_l = \left[\Delta_l^-, \Delta_l^+\right]$ $\forall l \in \Gamma$

0. **Poser** $\Delta\hat{p}_l = 0, \forall l \in \Gamma$.
1. **Décomposer** Γ **en** $\Gamma_1^+, \Gamma_0^+, \Gamma_1^-$ **et** Γ_0^-.
2. **Calculer** Δ_l^+, Δ_l^- **et** ρ_l, $\forall l \in \Gamma$.
3. **Si** $l \in \Gamma_1^+$ **alors** $\Delta p_l \in [0, +\infty[$.
   Si $l \in \Gamma_0^-$ **alors** $\Delta p_l \in [1 - p_l, 0]$.
   Si $l \in \Gamma_1^-$ **alors** $\Delta p_l \in [\max\{I_l^-, w_l \times \min_{l \in \Gamma^-}\{\rho_l\} - p_l\}, 0]$.
   Si $l \in \Gamma_0^+$ **alors** $\Delta\hat{p}_l \in [0, \min\{I_l^+, w_l \times \max_{l \in \Gamma}\{\rho_l\} - p_l\}]$.

Un algorithme pour le calcul d'un intervalle de sensibilité pour p_k

L'algorithme suivant décrit les principales étapes pour le calcul des limites de l'intervalle de sensibilité θ_k du profit d'un type d'objets quelconque dans le problème BKP.

On applique l'algorithme de Martello et Toth [43] afin de transformer BKP en un KP équivalent. Cette transformation remplace chaque type d'objets $k, k = 1,...,n$ par un ensemble de q objets où $q = \lfloor \log_2 b_k + 1 \rfloor$. Pour le calcul des limites de l'intervalle de sensibilité d'un type d'objets $k, k = 1,...,n$ dans BKP, nous utilisons l'algorithme de Belgacem et Hifi [8] et nous calculons les limites de l'intervalle de sensibilité des profits des q objets obtenus grâce à la transformation de BKP en KP.

Les q objets obtenus dans KP constituent l'ensemble perturbé Γ. Chacun de ces objets peut être dans l'un des sous-ensembles $\Gamma_0^-, \Gamma_0^+, \Gamma_1^-$ ou Γ_1^+ de Γ. Comme ils ont tous le même profit et que les quatre cas possibles peuvent avoir lieu en même temps. C'est-à-dire que pour un même type d'objets il peut y avoir des profits qui diminuent avec la variable d'état qui est nulle, des profits qui diminuent avec la variable d'état qui est égale à 1, des profits qui

augmentent avec la variable d'état qui est nulle et des profits qui augmentent avec la variable d'état égale à 1 dans la solution optimale. Dans ce cas, l'intervalle de sensibilité θ_k commun à l'ensemble perturbé Γ est l'intersection des intervalles de sensibilité de chacun des objets qui le constitue.

Algorithme 4.3 : Algorithme pour le calcul de θ_k.

Entrée : une instance de BKP, une solution optimale x^* et l'objet perturbé *k*.
Sortie : l'intervalle $\theta_k = [\theta_k^-, \theta_k^+]$

1. **Appliquer l'algorithme de Martello et Toth à BKP.**
2. **Calculer les intervalles de sensibilité Δ_{ki}, $i = 1,...,\lfloor \log_2 b_k + 1 \rfloor$ dans le KP obtenu en 1**
3. **Sortir avec les intervalles de sensibilité θ_k, $k = 1,...,n$ du problème BKP.**

Exemple 4.2 :

Dans la table 4.1, nous calculons les intervalles de sensibilité pour une instance de BKP composée de trois objets, transformée en une instance KP 0-1 à huit objets. Dans les colonnes 2, 3 et 4, nous avons les données et la solution optimale du problème considéré. Dans les colonnes 5 et 6, sont présentées les bornes des intervalles de sensibilité pour chacun des objets du knapsack KP. Et dans les deux dernières colonnes, nous retrouvons les bornes des intervalles de sensibilité pour les profits de chacun des objets de BKP.

On peut observer sur cet exemple, que l'intervalle de sensibilité du premier profit $\theta_1^- = p_8 = 10$ ce qui implique que le profit p_1 ne peut pas diminué sans influencer la solution optimale de notre problème, alors qu'il peut augmenter indéfiniment sans que cela n'ait de conséquences sur cette solution.

Pour l'objet 2, par contre, toute variation sur le profit p_2 va faire changer la solution optimale puisque $\theta_2^- = \theta_2^+ = p_2 = 15$.

Concernant l'objet 3, son profit peut varier de 1 jusqu'à 25 sans faire changer la solution optimale en main car on $\theta_3^+ = 25$.

Nous pouvons conclure, grâce à ces observations, que la solution optimale de ce problème est assez robuste puisque les variations tolérées des profits sont très petites.

Tableau 4.1 : Intervalles de sensibilité pour une instance de BKP.

objet	poids	profit	x_k^*	I_k		θ_k	
				I_k^-	I_k^-	θ_k^-	θ_k^+
1	1	10	1	10	Infini	10	Infini
	2	20	1	15	Infini		
	3	30	1	20	Infini		
2	3	15	1	15	Infini	15	15
	6	30	0	30	35		
	3	15	0	15	15		
3	5	11	0	1	25	1	25
	5	11	0	1	25		

Nous avons étudié dans ce mémoire, deux problèmes de type knapsack avec des données approximatives. Dans un premier temps, nous avons traité l'analyse de sensibilité de la solution optimale du 0-1 knapsack dans le cas de la perturbation du coefficient profit de la fonction objective, avant de nous pencher sur l'analyse de la sensibilité de la solution optimale du Bounded Knapsack Problem.

Nous avons présenté dans le chapitre 1 la problématique de l'optimisation combinatoire ainsi que l'analyse de sensibilité des problèmes linéaires et linéaires en 0-1.

Dans le chapitre 2 nous avons présenté les différents problèmes de type knapsack.

Dans le chapitre 3 nous avons décrit le problème du knapsack 0-1, sa formulation mathématique et les différentes méthodes de résolution. A la fin de chapitre, nous avons attaqué l'analyse de sensibilité du problème KP lors de la variation d'un profit arbitraire, où nous avons utilisé différentes bornes supérieures de la littérature, afin d'apporter des améliorations sur les bornes de l'intervalle de sensibilité du profit perturbé.

Nous avons traité, dans le chapitre 4, le problème du knapsack borné (Bounded Knapsack Problem BKP) qui est une généralisation du problème KP, où nous avons détaillé une approche de résolution de BKP qui le transforme en un KP équivalent grâce au codage binaire de la borne sur la disponibilité. Ensuite, nous avons analysé la sensibilité de BKP lorsqu'un profit est perturbé, où nous avons proposé une approche pour le calcul de l'intervalle de sensibilité du profit perturbé.

Notre réflexion de base s'appuie sur la construction d'un intervalle de sensibilité pour BKP lors de la perturbation d'un profit à partir de l'intervalle de sensibilité de KP lors de la perturbation des profits d'un sous-ensemble d'objets.

Les résultats sont assez satisfaisants.

Comme perspectives à ce travail, on peut citer :

- L'exploitation des conditions d'optimalité dans le cadre des méthodes autres que le branch and bound.
- Le développement de techniques spécifiques au BKP pour le calcul d'un intervalle de sensibilité (i.e. sans passer par sa transformation en KP).
- L'analyse de la sensibilité de la solution de BKP en cas de la variation du poids.

Références

[1] L. Aittoniemi (1982), Computational comparison of knapsack problems, presented at XIth international Symposium of Mathematical programming. Bonn, August 23-27.

[2] E. Balas, E. Zemel (1980), An algorithm for large zero-one knapsack problems. Operations Research, 28, 1130-1154.

[3] D.E Bell and J.F Shapiro (1977). A convergent duality theory for integer programming. Operations Research, 1, 467–477.

[4] R. Bellman (1954). Some applications of the theory of dynamic programming a review. Operations Research 2, 275-288.

[5] R.E. Bellman (1957). Dynamic Programming. Princeton University Press.

[6] R. Bellman and S.E. Dreyfus (1962). Applied Dynamic Programing. Princeton University Press, Princeton, NJ.

[7] T. Belgacem, M. Hifi (2005), Sensitivity Analysis of the Optimum to Perturbation of the Profit of a Subset of Items in the Knapsack Problem

[8] T. Belgacem, M. Hifi (2007), Sensitivity analysis of the knapsack problem: Tighter lower and upper bound limits.

[9] C. Blair (1998). Sensitivity analysis for knapsack problems : a negative result. Discrete Applied Mathematics, 81, 133-139.

[10] R.L. Bulfin, R.G. Parker, C.M. Shetty (1979), Computational results with a branch and bound algorithme for general knapsack problem. Naval Research Logistics Quarterly, 26, 41-46.

[11] P.J. Carstensen (1983). Complexity of some parametric integer and network programming problems. Mathematical Programming, 26, 64-75.

[12] N. Chakravarti and A.P.M. Wagelmans (1999). Calculation of stability radius for combinatorial optimization problems. Operations Research Letters, 23, 1-7.

[13] S.A. Cook, (1970). The Complexity of Theorem Proving Procedures. Conf. Record of Third ACM Symposium on Theory of Computing, 151-158.

[14] G.B. Dantzig (1957), Discrete variable extremum problems, Operations Research, 5, 266-277.

[15] K. Dudzinski, S. Walukiewicz (1984), Exact methods for the knapsack problem and its generalizations, Mathematical Programming, 29, 231-249.

[16] P. Fautrier (2004), Recherche opérationnelle. Distribué sous licence. Open-Content : http://opencontent.org//opl.shtml

[17] D. Fayard, G. Plateau (1982), An algorithm for the solution of the 0-1 knapsack problem, Computing, 28, 269-287.

[18] L. Ferro, S. Khin and N. Salman (2005). Résolution pratique de problèmes NP-complets.

[19] M.L. Fisher, W.D. Northup, and J.F. Shapiro. Using duality to solve discrete optimization problems: Theory and computational experience. Mathematical programming study, 3, 56-94.

[20] M.L. Fisher and J.F. Shapiro (1974). Constructive duality in integer programming. SIAM Journal on Applied Mathematics, 27, 31-52.

[21] R.S. Garfinkel, G.L. Nemhauser (1972), Integer programming. Wiley, New York.

[22] D. Ghosh, D. Charkravarti and G. Sierksma (2006). Sensitivity analysis of a greedy heuristic for knapsack problems. European Journal of Operational Research, 169, 340-350.

[23] P.C. Gilmore, R.E. Gomory (1966), The theory and computation of knapsack functions. Operations Research, 13, 879-919.

[24] E.N. Gordeev (1989). Solution stability in a shortest path problem. Discrete Mathematics 1, 3, 45-56.

[25] M. Hifi, H. Mhalla and S. Sadfi (2004). Adaptive algorithms for the knapsack problem: perturbation of an arbitrary item. Working Paper, CERMSEM-CNRS UMR 8095, MSE, Université Paris 1.

[26] M. Hifi, H. Mhalla and S. Sadfi (2005), Sensitivity of the optimum to perturbations of the profit or weight of an item in the binary knapsack problem, Journal of Combinatorial Optimization, 10, 239-260.

[27] S.V. Hoesel and A. Wagelmans(1999). On the complexity of postoptimality analysis of 0/1 programs. Discrete Applied Mathematics, 91, 251-263.

[28] E. Horowitz, S. Sahni (1974), Computing Partitions to the Knapsack Problem. Journal of the Association for Computing Machinery, 21 (2), 277-292.

[29] P.D. Hudson (1977), Improving the branch and bound algorithms for the knapsack problem. Queen's University Research Report, Belfast.

[30] G.P. Ingargiola, J.F. Korsh (1973), A reduction algorithm for zero-one single knapsack problems. Management Science, 20, 460-463.

[31] G.P. Ingargiola, J.F. Korsh (1977), A general algorithm for one-dimentional knapsack problems. Operations Research, 25, 752-759.

[32] D.S. Johnson (1974), Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences, 9, 256-278.

[33] D. Klein and S. Holm (1979). Integer programming postoptimal analysis with cutting planes. Management Science, 25, 64-72.

[34] S. Kaplan (1966). Solution of the Lorie-Savage and similar integer programming problems by the generalized Lagrange multiplier method. Operations Research, 10, 1130-1136.

[35] H. Kellerer, U. Pferschy and D. Pisinger (2003). Knapsack Problems. Springer

[36] P.J. Kolesar (1967), Abranch and bound algorithm for the knapsack problem. Management of Science, 13, 723-735.

[37] M. Libura (1977). Sensitivity analysis for integer knapsack problem. Archiwum Automatyki i Telemechaniki, 22, 313-322.

[38] M. Libura (1996). Optimality conditions and sensitivity analysis for combinatorial optimization problems. Control and Cybernetics, 25(6), 1165-1180.

[39] S. Martello, P.Toth (1977), An upper bound for the zero-one knapsack problem and a branch and bound algorithm. European Journal of Operational Research, 1, 169-175.

[40] S. Martello, P.Toth (1977), Computational experiences with large-size unidimentional knapsack problems. Presented at the TIMS/ORSA Joint National Meeting, San Francisco.

[41] S. Martello, P.Toth (1977), A new algorithm for the 0-1 knapsack problem, Management Science, 34, 633-644.

[42] S. Martello, P.Toth (1990), Knapsack problems: Algorithms and computer implementations, Wiley, Chichester, England.

[43] S. Martello and P. Toth (1990). An exact algorithm for large unbounded Knapsack problems. Operations Research Letters, 9, 15-20.

[44] J.F. Mauras (2002). Programmation Linéaire, Séparation et Optimisation.

[45] R.C. Merkle and M.E. Hellman (1977). Hiding Information and Receipts in Trap Door Knapsacks. Internal Symposium on Information Theory, Cornell University, Ithaca,NY.

[46] M. Minoux (1989). Programmation mathématique : théorie et algorithmes. Dunod.

[47] H. Müller-Merbach (1978), An improved upper bound for the zero-one knapsack problem: a note on the paper by Martello and Toth. European Journal of Operational Research, 2, 212-213.

[48] K. Nauss (1975). Parametric integer programming. Ph.D. thesis, Western Management Science Institute, UCLA.

[49] G.L. Nemhauser, Z. Ullmann (1969), Discrete dynamic programming and capital allocation. Management Sience, 15, 494-505.

[50] G. Plateau, M. Elkihel (1985), A hybrid method for the 0-1 knapsack problem. Methods of Operations Research, 49, 277-293.

[51] C.J. Piper and A. Zoltners (1975). Implicit enumeration based algorithms for post-optimising zero-one programs. Naval Research Logistics Quarterly, 22, 791-809.

[52] C.J. Piper and A. Zoltners (1976). Some easy postoptimality analysis for zero-one programming. Management Science, 22, 759-765.

[53] D. Pisinger (1995), Algorithms for knapsack problems. Ph.D. thesis. University of Copenhagen, Denmark.

[54] G.M Roodman (1972). Postoptimality analysis in zero-one programming by implicit enumeration. Naval Research Logistics Quarterly, 19, 435-447.

[55] G.M Roodman (1974). Postoptimality analysis in zero-one programming by implicit enumeration. the mixed integer case. Naval Research Logistics Quarterly, 21, 595-607.

[56] S. Sahni (1975), Approximate algorithms for the 0-1 knapsack problem. Journal of ACM, 22, 115-124.

[57] M. Sakarovitch (1984), Optimisation combinatoire : Méthodes mathématiques et algorithmiques, programmation discrète. Hermann.

[58] L. Schrage and L. Wolsey (1985). Sensitivity analysis for branch and bound integer programming. Operations Research, 33, 1008-1023.

[59] Y.N. Sotskov, V.K. Leontev, and E.N. Gordeev (1995). Some concepts of stability analysis in combinatorial optimization. Discrete Applied Mathematics, 58, 169-190.

[60] B. Thiongane, A. Nagih, G. Plateau (2003), Analyse de sensibilité pour les problèmes linéaires en variables 0-1. RAIRO Operations Research, 37, 291-309.

[61] C.P.M. Van Hoesel, A.P.M. Wagelmans (1999), On the complexity of postoptimality analysis of 0/1 programs. Discrete Applied Mathematics, 91, 251-263.

[62] P.R.C. Villela, C.T. Bornstein (1983), An improved bound for the 0-1 knapsack problem. Report ES31-83, COOPE-Federal University of Rio de Janeiro.

[63] A.P.M Wagelmans (1990). Sensitivity analysis in combinatorial optimization. Ph.D, Erasmus University, Rotterdam.

[64] G.J. Woeginger (1999). Sensitivity analysis for knapsack problems: another negative result. Discrete Applied Mathematics, 92, 247-251.

[65] E. Zemel (1981). Measuring the quality of approximate solutions to zero-one programming problems, Mathematics of Operations Research, 6, 319-332.