MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ MOULOUD MAMMERI DE TIZI OUZOU
FACULTÉ DES SCIENCES
DÉPARTEMENT : MATHÉMATIQUES

MÉMOIRE DE MASTER

Filière : **Mathématiques**

Spécialité : **Mathématiques Appliquées à la Gestion**

Par

Talem Sonia

Résolution du Problème du Sac à Dos avec Contraintes Disjonctives

soutenu le Septembre 2022 devant le jury :

M^{me} N.GRAICHE ;	MAA	U.M.M.T.O ;	Président.
M^{me} F.AKLOUCHE ;	MAA	U.M.M.T.O ;	Examinatrice.
M^{me} R.KHEFFACHE ;	MCB	U.M.M.T.O ;	Promotrice.

Soutenu le 05/10 /2022

Dédicaces

Je dédie ce modeste travail

Aux deux personnes les plus nobles et les plus chères au monde, **mes chers parents**

qui ont sacrifié les plus belles années de leurs vie pour me voir un jour réussir.

Qui ont toujours été là pour moi, et qui m'ont donné un honorable modèle de prévenance,

qu'ils trouvent ici l'expression de toute ma profonde reconnaissance et ma parfaite considération,

ce modeste travail soit le fruit de vos innombrables sacrifices.

Aux anges qui éclairent ma vie **ma soeur** et **mon frère**.

A ma promotrice M^{me} **KHEFFACHE REZIKA**.

Remerciements

Je remercie mes très chers parents, qui sont toujours là pour moi, je remercie ma soeur et mon frère pour leurs encouragements.

Je tiens à exprimer ma gratitude à M^{me} **KHEFFACHE Rezika** , pour l'aide et le temps qu'elle a bien voulu me consacrer et que je ne la remercier jamais assez pour son soutiens et sa patience.

Ensuite à M^{me} F. AKLOUCHE et M^{me} N.GRAICHE les membres du jury qui ont eu amabilité d'accepter d'évaluer ce travail, qu'elles trouvent ici l'expression de ma reconnaissance .

Je remercie tous les enseignants qui ont contribué à notre cursus particulièrement ceux du département "MATHEMATIQUES", et qu'ils trouvent ici le témoignage de ma profonde reconnaissance.

Finalement, je remercie toutes les personnes ayant contribué de prés ou de loin à l'élaboration de ce mémoire.

Table des matières

1 Optimisation combinatoire **6**

1.1 Introduction 6

1.2 Optimisation combinatoire 6

1.2.1 Une instance 6

1.3 Complexité 7

1.3.1 Concepts de base 7

1.3.2 Algorithme 7

1.3.3 Complexité d'un algorithme 7

1.3.4 Complexité dans le pire cas 7

1.3.5 Algorithme efficace 8

1.3.6 Complexité d'un problème 8

1.3.7 Les classes de problèmes 8

1.4 Méthodes de résolution 9

1.4.1 Méthodes exactes 9

1.4.2 Méthodes approchées 10

1.5 Conclusion 10

2 Programmation linéaire en nombres entiers **12**

2.1 Introduction 12

2.2 Programmation linéaire 12

2.3 Programmation linéaire en nombres entiers 13

2.4 Relaxation linéaire 14

2.4.1 Propriétés de la relaxation linéaire 14

2.5 Méthode de Branch and Bound 15
2.5.1 Arbre de recherche 15
2.5.2 Résolution d'un programme linéaire en nombres entiers par la méthode de Branch and Bound 16
2.5.3 Algorithme général 16
2.6 Méthodes de résolution de la relaxation linéaire 17
2.6.1 Méthode graphique 17
2.6.2 Méthode du simplexe 18

3 Problèmes de type sac à dos **23**
3.1 Introduction 23
3.2 Problème du sac à dos 23
3.2.1 Modélisation 24
3.3 Les variantes du problèmes du sac à dos 24
3.3.1 Sac à dos en variables continues 25
3.3.2 Sac à dos en variables entières 25
3.3.3 Sac à dos multiple 25
3.3.4 Sac à dos multidimensionnel 25
3.3.5 Sac à dos multi-objectif 25
3.3.6 Sac à dos quadratique 26
3.3.7 Sac à dos à contraintes disjonctives 26
3.4 Conclusion 26

4 Sac à dos à contraintes disjonctives et sa résolution **28**
4.1 Introduction 28
4.2 Définition du problème et Modélisation 28
4.3 Les applications du DCKP 29
4.4 Quelques méthodes de résolution de DCKP 30
4.4.1 Algorithme glouton 30
4.4.2 Amélioration de l'algorithme glouton 30
4.5 Résolution du DCKP par une méthode exacte 30

4.6 Présentation de la méthode 31
4.7 Exemple 33

Annexe **44**

Bibliographie **46**

Introduction générale

Tout programme linéaire en nombres entiers contenant une seule contrainte est appelée un problème du sac à dos ou(knapsack problem). Grâce à son utilité, ce problème particulier est de plus en plus utilisé dans le domaine décisionnel.
Le domaine d'application de ce problème ainsi que ses variantes inclut des cas du domaine de transport, de logistique ainsi que de la production.
Dans ce travail nous nous sommes intéressés au problème de sac à dos à contraintes disjonctives l'une des variantes de sac à dos classique (Knapsack).
Le présent mémoire est réparti en cinq chapitres :
Le premier chapitre introduit l'optimisation combinatoire et les méthodes de résolution utilisées, tout en présentant quelques définitions et concepts de base nécessaires pour la compréhension de notre travail ainsi que la notion de complexité et les classes de problèmes.
Le deuxième chapitre constitue une introduction sur la programmation linéaire en nombres entiers puis nous avons présenté l'une des méthodes de résolution d'un problème linéaire en nombres entiers qui est la méthode de Branch and Bound.
Le troisième chapitre, constitue une introduction au problème du sac à dos et ses variantes où nous avons présenté les notions essentielles,la modélisation du problème sous forme mathématique.
Le quatrième chapitre est consacré au problème du sac à dos à contraintes disjonctives,sa modélisation, ses divers applications,ainsi que les méthodes de sa résolution. A la fin, nous avons présenté
la méthode que nous avons utilisée, basée sur la méthode de branch and bound et le simplexe. Nous avons illustré cette méthode à l'aide d'un exemple.

Nous terminons ce mémoire par une conclusion générale, quelques définitions en annexe et une bibliographie.

Chapitre 1

Optimisation combinatoire

1.1 Introduction

Dans ce chapitre, nous introduisons les notions de base de l'optimisation combinatoire et les méthodes de résolution des problèmes d'optimisation tout en présentant quelques définitions et concepts de base nécessaires pour la comprehension de notre travail ainsi que la notion de complexité algorithmique.

1.2 Optimisation combinatoire

L'optimisation combinatoire est une branche des mathématiques appliquées et en informatique [9]. De nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire cela rend ce domaine très important. Un problème d'optimisation combinatoire est défini par la donnée d'un ensemble fini S et d'une application $f : S \rightarrow \mathbb{R}$, il s'agit de déterminer s^*(s^* solution optimale), tel que $f(s^*) \geq f(s)$, pour tout élément de S (problème de maximisation).

1.2.1 Une instance

Une instance d'un problème est obtenue en affectant une valeur à tous les paramètres de ce problème.

1.3 Complexité

La complexité a été développée pour classer les problèmes faciles et difficiles[8].

1.3.1 Concepts de base

A toute instance I d'un problème (P) nous pouvons associer un nombre $\mu(I)$ qui mesure la longueur des données de cette instance et que nous appelons la taille de l'instance I .

1.3.2 Algorithme

Un algorithme est une procedure de calcul bien définie qui prend en entrée une valeur ou un ensemble de valeurs, un algorithme est donc une séquence d'étapes de calcul qui transforment l'entrée en sortie.

1.3.3 Complexité d'un algorithme

La théorie de la complexité des algorithmes étudie formellement la difficulté intrinsèque des problèmes algorithmique, elle définit plusieurs classes de complexité $(P, NP, ...)$ permettant de classer les problèmes selon leurs caractéristiques.
Si A est un algorithme de résolution d'un problème (P) et (I) est une instance de se dernier, alors au couple (A,I) on associe un entier $\Gamma(A, I)$ representant le nombre d'operations élémentaires $(+, -, \times, ...)$ effectuées par l'algorithme A dans la résolution de l'instance I du problème (P). Le plus grand nombre ayant la même taille est appelé complexité de l'algorithme A.

1.3.4 Complexité dans le pire cas

La complexité dans le pire cas correspond au temps d'exécution le plus long que puisse avoir l'algorithme et permet de comparer l'efficacité de deux algorithmes .

1.3.5 Algorithme efficace

Un algorithme est efficace si sa complexité est majorée par un polynôme en taille des données, la notion d'efficacité d'un algorithme est liée à la manière dont le nombre d'operations augmente avec la taille de l'instance à laquelle il est appliqué.

1.3.6 Complexité d'un problème

La complexité d'un problème est une estimation du nombre d'instructions à exécuter pour résoudre les instances de ce problème, cette estimation étant un ordre de grandeur par rapport à la taille de l'instance, elle est définie en considérant son instance la plus difficile.

1.3.7 Les classes de problèmes

L'idée est de classer les problèmes selon leurs complexités : pour cela, il est d'abord nécessaire de distinguer les problèmes de décision des problèmes d'optimisation combinatoire.

Problème de décision

Un problème de décision est un problème dont la solution est formulée en terme OUI /NON.

Notons qu'il est possible d'associer à chaque problème d'optimisation un problème de décision en introduisant un seuil k correspondant à la fonction f. Le problème de décision devient existe-t-il une solution réalisable s telle que : $f(s) \leq k ou \geq k$?

La classe des problèmes P

Un problème appartinrent à la classe P s'il existe un algorithme polynomial pour le résoudre. les problèmes appartenant à celle classe sont dits faciles.

La classe des problèmes NP et les algorithmes non déterministes

La notion NP signifie "NON-Deterministe polynomial" cette classe renferme tous les problèmes de decision dont on peut associer à chaque un d'eux un ensemble de solu-

tions potentielles. Le terme non deterministe désigne un pouvoir qu'on incorpore à un algorithme pour qu'il puisse deviner la bonne solution.

La classe des problèmes NP-complets

La classe NP-COMPLET contient les problèmes les plus difficiles de NP, ces problèmes semblent vraiment ne pas faire partie de P . Si nous sommes capables de trouver un algorithme pour résoudre efficacement un problème de NP-complet nous pouvons alors utiliser cet algorithme pour résoudre tous les problèmes de NP .

1.4 Méthodes de résolution

Étant donné l'importance de ces problèmes, de nombreuses méthodes ont été développées en recherche opérationnelle et en intelligence artificielle. Ces méthodes peuvent être classées en deux grandes catégories : Méthodes exactes et approchées.

1.4.1 Méthodes exactes

Les méthode exactes permettent de trouver une solution optimale pour les problèmes de tailles raisonnables, toutefois, ces méthodes peuvent devenir rapidement coûteuses en temps de calcul nécessaire pour trouver une solution notamment pour les problèmes de grande taille. Parmi ces méthodes :

a- Méthode de résolution par séparation et évaluation

Cette méthode est basée sur trois principes :

a) le principe de séparation :consiste à établir l'ensemble des règles qui vont permettre la séparation d'un ensemble en sous ensembles de cardinal plus petites.

b) Le principe d'évaluation : permet de diminuer l'espace de recherche

c) Parcours de l'arborescence : cette phase se fait selon une stratégie de développement précise qui peut être en profondeur, d'abord ou en largeur d'abord. Cette stratégie permet de savoir quel sommet choisir pour la séparation.

b- Méthode de programmation dynamique

La programmation dynamique est une méthode exacte de résolution de problèmes d'op-

timisation séquentielle, due essentiellement à RICHARD BELLMAN. Elle est similaire à la méthode de Branch and Bound, elle divise le problème en sous problèmes et la solution d'un sous problème est utilisée pour résoudre un autre sous problème.

1.4.2 Méthodes approchées

Pour les problèmes de grande taille, les méthodes exactes ne sont pas envisageable de par leur temps de calcul qui est exponentiel et donc sans aucun intérêt en pratique. Dans ce cas, il est possible d'utiliser les méthodes approchées qui donnent une solution réalisable obtenue rapidement, parmi ces méthodes :

Les heuristiques

Les heuristiques sont utilisées pour un problème particulier et il existe deux classes d'heuristique :

a-Les algorithmes gloutons

Le principe de ces algorithmes et de construire pas à pas une solution réalisable et à chaque étape de l'algorithme, la solution en cours est utilisée pour construire une meilleure solution.

b- Les algorithmes par exploration locale

Les algorithmes par exploration locale définissent un voisinage à partir d'une solution réalisable dans lequel ils cherchent une meilleure solution que celle dont ils disposent.

Métaheuristiques :

Une métaheuristique est un ensemble de concepts utilisés pour définir des méthodes heuristiques pouvant être appliquées à une grande variété de problèmes. Les métaheuristiques sont plus puissantes et adaptables pour résoudre un grand nombre de problèmes. La plupart des métaheuristiques sont d'inspiration biologique. Parmi lesquelles, nous pouvons citer les algorithmes génétiques, le circuit simulé, les réseaux de neurones...

1.5 Conclusion

Nous avons présenté dans ce chapitre quelques notions de base sur l'optimisation combinatoire, la complexité et nous avons aussi présenté quelques méthodes de résolution

de problèmes.

Chapitre 2

Programmation linéaire en nombres entiers

2.1 Introduction

De nombreux problèmes peuvent être modélisés sous forme d'un problème d'optimisation dans lequel on cherche les valeurs des variables pour lesquelles la fonction objectif prend une valeur maximale (ou minimale), ces variables étant soumises à un ensemble de contraintes.

2.2 Programmation linéaire

En mathématiques, les problèmes de programmation linéaire (PL) sont des problèmes d'optimisation, où la fonction objectif et les contraintes sont toutes linéaires. La programmation linéaire désigne également la manière de résoudre les problèmes de (PL). Le terme programmation linéaire suppose que les solutions à trouver doivent être représentées en variables réelles, ces variables étant appelées variables de décision.

2.3 Programmation linéaire en nombres entiers

En pratique, on est souvent amené à traiter des problèmes linéaires se présentant sous forme de programmes linéaires dont les variables prennent des valeurs entières, par exemple, les problèmes d'affectation, les problèmes d'emploie du temps, de production comme (tables, chaises, nombre d'appartements...)

Ce type de problèmes s'appellent programmes linéaires en nombres entiers c'est à dire que les variables sont contraintes à ne prendre que les valeurs entières [5, 10].

Remarques :

- Un programme linéaire en nombres entiers dont les variables sont contraintes à prendre que les valeurs 0 ou 1 est un programme linéaire binaire.
- Un programme mixte en nombres entiers est un programme dont certaines variables sont contraintes à ne prendre que des valeurs entières.

Un programme linéaire en nombres entiers est représenté comme suit :

$$(P)\begin{cases} Z(x) = cx \rightarrow max & (1) \\ Ax \leq b & (2) \\ x \geq 0 & (3) \\ x\ entiers & (4) \end{cases}$$

Avec $c \in \mathbb{R}^n$ vecteur de coûts, $b \in \mathbb{R}^m$ vecteur des termes constants des contraintes et $A \in \mathbb{R}^{m \times n}$ matrice des coefficients des contraintes, où n est le nombre de variables et m est le nombre de contraintes.

- (1)représente la fonction économique ou fonction objectif.
- (2) représente les contraintes principales.
- (3) représente la contrainte de non négativité.

Remarques

a) Toute inégalité $\leq$(resp $\geq$) peut être transformée en égalité, en rajoutant (resp en soustrayant) des variables d'écart.

b) Toute inégalité $\geq$ est équivalente à une inégalité $\leq$ en multipliant ses termes par (-1).

c) $minZ = -max(-Z)$.

2.4 Relaxation linéaire

Avant de résoudre un problème de programmation linéaire en nombres entiers, on doit l'écrire sous forme d'un programme linéaire continu, en relaxant les contraintes d'intégrité [2].

La relaxation linéaire de (P) est donc :

$$\begin{cases} Z(x) = cx \rightarrow max \\ Ax \leq b \\ x \geq 0 \end{cases}$$

On note ce programme par (LP).

Si le programme linéaire est binaire c'est à dire x prend la valeur 0 ou 1 :

$$(P) \begin{cases} Z(x) = cx \rightarrow max \\ Ax \leq b \\ x \in \{0, 1\} \end{cases}$$

Alors sa relaxation linéaire sera :

$$(LP) \begin{cases} Z(x) = cx \rightarrow max \\ Ax \leq b \\ x \in [0, 1] \end{cases}$$

2.4.1 Propriétés de la relaxation linéaire

a) La valeur de la solution optimale de (LP) est une borne inférieur de la valeur de la solution optimale de (P) dans le cas d'une maximisation.

b) La valeur d'une solution admissible de P fournit une borne inférieure sur la valeur de la solution optimale de P dans le cas de maximisation et une borne supérieure sur la valeur optimale de P dans le cas d'une minimisation.

c) Si la solution optimale de (LP) est entière (admissible pour P), elle est également

la solution optimale de P.

2.5 Méthode de Branch and Bound

L'algorithme de Branch and Bound est une méthode qui permet d'énumérer implicitement l'ensemble des solutions réalisables d'un problème de programmation linéaire en nombres entiers. Il suit le principe de « diviser pour régner » en décomposant le problème initial en plusieurs sous-problèmes plus simples à résoudre. La solution optimale du problème est retrouvée en prenant la solution du meilleur sous-problème[3].
L'algorithme parcourt un arbre de recherche, où chaque noeud de l'arbre de recherche correspond à un sous-domaine du domaine réalisable du problème original. La racine de l'arbre de recherche correspond au domaine réalisable du problème, tandis que les noeuds représentent les domaines réalisables des sous-problèmes. À chaque itération, l'algorithme de Branch and Bound résout la relaxation du sous-problème courant. Cette dernière fournit une borne supérieure (cas de maximisation) de la valeur optimale du sous-problème courant.

2.5.1 Arbre de recherche

Dans la méthode de branch and Bound, on divise le problème en sous problèmes comme suit :

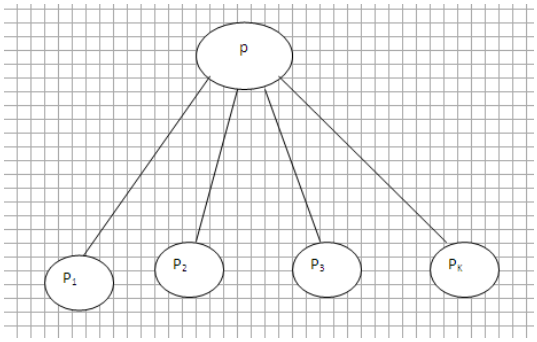

FIG. 2.1 – Sous problème de (P)

A priori, les sous problèmes peuvent être difficiles que le problème original, donc on applique le même système, c'est à dire, on divise les sous problèmes.

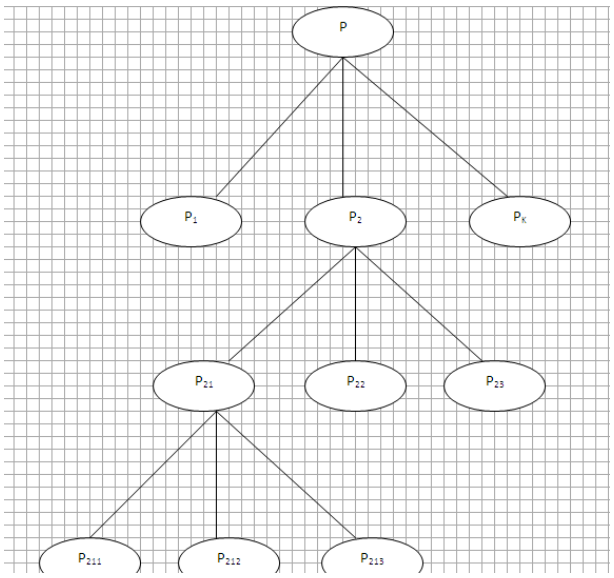

FIG. 2.2 – Arbre de recherche

2.5.2 Résolution d'un programme linéaire en nombres entiers par la méthode de Branch and Bound

Pour résoudre le programme linéaire en nombres entiers (P) par la méthode de Branch and Bound on résout son programme relaxé (LP).
Si la solution optimale de (LP) est entière, elle est aussi optimale pour le programme (P)
Si la solution de (LP) n'est pas entière, soit x_i une variable prenant une valeur fractionnaire dans la solution optimale de (LP). Le problème peut être diviser en deux sous problèmes en ajoutant les contraintes : $x_i \leq \lfloor x_i \rfloor$ et $x_i \geq \lceil x_i \rceil$.

2.5.3 Algorithme général

a) Cas d'une minimisation

A chaque instant, on maintient une liste de sous problèmes actifs.

Le coût U de la meilleure solution obtenue.

Soit $U = +\infty$ valeur initiale.

1) Sélectionner un sous problème actif (P_i).

2)Si(P_i) est non admissible, le supprimer, sinon calculer $b(P_i)$ une borne inférieure sur le coût optimal de (P) qui est valeur optimale de (LP).

3) Si $b(P_i) \geq U$, supprimer (P_i)

4) Si $b(P_i) < U$ Soit résoudre (P_i) directement, soit créer de nouveaux sous problèmes et les ajouter à la liste des sous problèmes actifs.

b) Cas d'une maximisation

Soit $U = -\infty$ valeur initiale.

1) Sélectionner un sous problème actif (P_i).
2) Si (P_i) est non admissible, le supprimer, sinon, calculer $b(P_i)$ une borne supérieure sur le coût optimal de la relaxation de (P).
3) Si $b(P_i) \leq U$, supprimer (P_i).
4) Si $b(P_i) > U$ Soit résoudre directement (P_i), soit créer de nouveaux sous problèmes et les ajouter à la liste des sous problèmes actifs.

Remarques

1- Il n'y a pas de règle générale pour le choix de la variable de branchement et de la branche à examiner en premier.

2- Ce choix peut avoir un impact important sur le nombre de nœuds à examiner dans l'arbre de Branch and Bound.

2.6 Méthodes de résolution de la relaxation linéaire

2.6.1 Méthode graphique

La méthode graphique permet de résoudre des problèmes linéaires à deux ou trois variables de décision puisque'il n'est pas possible d'illustrer graphiquement plus de trois dimensions.

2.6.2 Méthode du simplexe

Dans la plupart des problèmes réels, on a plus de deux variables à déterminer .
Une procédure algébrique pour résoudre les programmes linéaires avec plus de deux variables fera l'objet de cette section. C'est la méthode du simplexe .
La méthode du simplexe est une méthode itérative, elle démarre d'un point réalisable (sommet de départ) et passe à un autre point en augmentant la valeur de la fonction objectif dans le cas de maximisation.
On arrête le déroulement de l'algorithme lorsqu'il n'est plus possible d'augmenter la valeur de la fonction objectif [9].

Définitions

- Un programme linéaire est dit sous forme canonique si toutes les inégalités sont dans le sens ($\leq$), les variables de décision sont positives ou nulles.
- Tout programme linéaire sous forme canonique peut s'écrire sous forme standard, en ajoutant pour chaque contrainte i une variable s_i appelée variable d'écart positive tel que : $ax \leq b \Leftrightarrow \ ax + s = b,\ s \geq 0$.

Soit le problème linéaire suivant :

$$\begin{cases} Z(x) \rightarrow max, \\ Ax \leq b, \\ x \geq 0, \end{cases}$$

La méthode comprend les étapes suivantes :
1) Ajouter des variables d'écart.
2) Déterminer une solution de base réalisable
Est-elle optimale ? Si oui STOP
Sinon
3)Trouver une solution admissible
a/ On dessine le tableau
b/ On définie la variable entrante
c/ On définit la variable sortante
d/ Mise à jour du tableau

f/ On vérifie le critère de sortie

Exemple :

$$(P)\begin{cases} Z = 5x_1 + 4x_2 \rightarrow Max \\ x_1 + x_2 \leq 5 \\ 10x_1 + 6x_2 \leq 45 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_1 \text{ et } x_2 \text{ entiers.} \end{cases}$$

La relaxation linéaire :

$$(LP)\begin{cases} Z = 5x_1 + 4x_2 \rightarrow Max \\ x_1 + x_2 \leq 5 \\ 10x_1 + 6x_2 \leq 45 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_1 \text{ et } x_2 \text{ entiers.} \end{cases}$$

La forme standard :

$$\begin{cases} Z = 5x_1 + 4x_2 \rightarrow Max \\ x_1 + x_2 + x_3 = 5 \\ 10x_1 + 6x_2 + x_4 = 45 \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0 \end{cases}$$

Base/Variable	x_1	x_2	x_3	x_4	Second Membre
x_3	1	1	1	0	5
x_4	10	6	0	1	45
Coût réduit	-5	-4	0	0	0

La plus petite valeur négative entre $\{-5, -4\}$ est -5 correspond à la variable x_1, donc faire rentrer x_1

$min\left\{\frac{5}{1}, \frac{9}{2}\right\} = \frac{9}{2}$ correspond à x_4, donc faire sortir x_4, pivot autour de 10.

Base/Variable	x_1	x_2	x_3	x_4	Second Membre
x_3	0	$-\frac{2}{5}$	1	$-\frac{1}{10}$	$\frac{1}{2}$
x_1	1	$\frac{3}{5}$	0	$\frac{1}{10}$	$\frac{9}{2}$
Coût réduit	0	-1	0	$\frac{1}{2}$	$\frac{45}{2}$

-1 est la plus petite valeur négative correspond à la variable x_2, donc faire rentrer x_2

$min\left\{\frac{5}{4}, \frac{15}{2}\right\} = \frac{5}{4}$ correspond à x_3, donc faire sortir x_3, pivot autour de $\frac{2}{5}$.

Base/Variable	x_1	x_2	x_3	x_4	Second Membre
x_2	0	1	$\frac{5}{2}$	$-\frac{1}{4}$	$\frac{5}{4}$
x_1	1	0	$-\frac{3}{2}$	$\frac{1}{4}$	$\frac{15}{4}$
Coût réduit	0	0	$\frac{5}{2}$	$\frac{1}{4}$	$\frac{95}{4}$

$x_1 = 15/4 = 3.75$, $x_2 = 5/4 = 1.25$, $Z = 95/4 = 23.75$

La solution n'est pas entière, on crée deux sous problèmes (P_1) et (P_2), en ajoutant les contraintes :

$x_1 \leq \lfloor 3.75 \rfloor = 3$, $x_1 \geq \lceil 3.75 \rceil = 4$

$$(P_1) \begin{cases} Z = 5x_1 + 4x_2 \rightarrow Max \\ x_1 + x_2 \leq 5 \\ 10x_1 + 6x_2 \leq 45 \\ x_1 \leq 3 \\ x_1, x_2, x_3, x_4 entiers \end{cases}$$

$$(P_2)\begin{cases} Z = 5x_1 + 4x_2 \rightarrow Max \\ x_1 + x_2 \leq 5 \\ 10x_1 + 6x_2 \leq 45 \\ x_1 \geq 4 \\ x_1, x_2, x_3, x_4 entiers \end{cases}$$

On résout la relaxation linéaire de P_1 et P_2

La solution de P_1 est : $(x_1, x_2) = (3, 2)$ avec $Z = 23$

La solution de P_2 est : $(x_1, x_2) = (4, 0.83)$ avec $Z = 23,33$

U= 23

$b(P_2) = 23.33 > U$, on crée deux sous problèmes P_{21} et P_{22} :

$x_2 \geq \lceil 0.83 \rceil = 1$, $x_2 \leq \lfloor 0.83 \rfloor = 0$.

$$(P_{21})\begin{cases} Z = 5x_1 + 4x_2 \rightarrow Max \\ x_1 + x_2 \leq 5 \\ 10x_1 + 6x_2 \leq 45 \\ x_1 \geq 4 \\ x_2 \leq 0 \\ x_1, x_2, x_3, x_4 \ entiers \end{cases}$$

$$(P_{22})\begin{cases} Z = 5x_1 + 4x_2 \rightarrow Max \\ x_1 + x_2 \leq 5 \\ 10x_1 + 6x_2 \leq 45 \\ x_1 \geq 4 \\ x_2 \geq 1 \\ x_1, x_2, x_3, x_4 \ entiers. \end{cases}$$

Liste de sous problèmes actifs $\{P_1, P_2, P_{21}, P_{22}\}$. On résout la relaxation des sous problèmes P_{21} et P_{22} : La solution de P_{21} est $x = (4.5, 0)$ avec $Z = 22.5$

Le problème P_{22} n'a pas de solution. On le supprime.

Liste des sous problèmes actifs $\{P_1, P_2, P_{21}\}$.

la solution de P_{21} est $x = (4.5, 0)$, $Z = 22.5$

$b(P_{21}) = 22,5$

$b(P_{21}) \leq U = 23$, on supprime P_{21} car il ne contient pas la meilleure solution.

La solution de P est $(x_1, x_2) = (3, 2)$ avec $Z = 23$.

l'arborescence ci dessous nous résume la méthode de branch and bound.

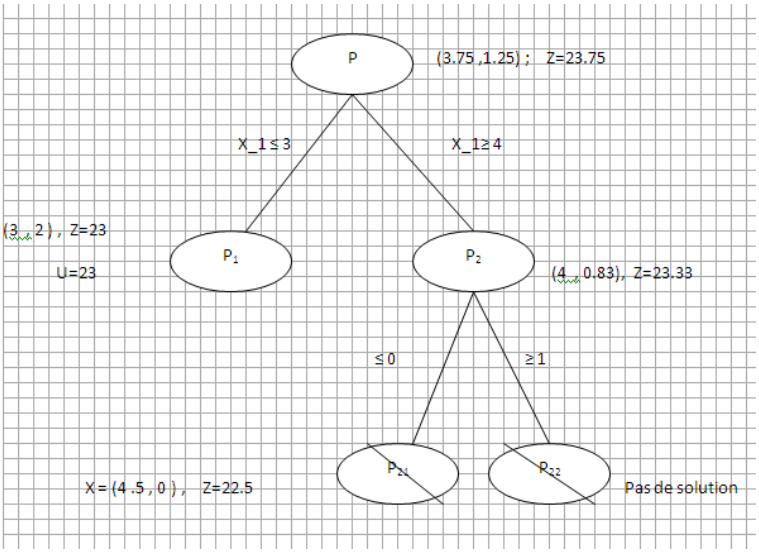

FIG. 2.3 – Arborescence de branch and bound

Chapitre 3

Problèmes de type sac à dos

3.1 Introduction

Dans ce chapitre, on s'interesse à une classe de problèmes d'optimisation connue sous le nom du problème de sac à dos qui signifie le remplissage d'un sac à dos dont il ne faut pas dépasser un poids maximal et l'objectif est de maximiser le profit des objets qui sont mis dans le sac.

3.2 Problème du sac à dos

Le problème du sac à dos noté également (KP), en anglais Knapsack problem, est un problème d'optimisation combinatoire NP-COMPLET de Richard Karp [1, 4]. Il modélise une situation analogue au remplissage d'un sac à dos, ne pouvant supporter plus d'un certain poids(w)avec un ensemble donné d'objets ayant chacun un poids (w) et valeur (v).

Les objets mis dans le sac à dos doivent maximiser la valeur totale sans dépasser le poind maximum du sac.

3.2.1 Modélisation

Soient n objets numérotés de 1 à n possédant chacun une valeur v_i et un poids w_i et un sac de capacité C, il faut indiquer pour chaque objet s'il est pris ou non.

$$\begin{cases} x_i = 1 \ si \ l'objet \ i \in au \ sac, \\ x_i = 0 \ sinon, \end{cases}$$

et la somme des poids associés aux objets choisis ne dépasse pas la capacité du sac C, donc on a :

$$\sum_{i=1}^{n} w_i x_i \leq C.$$

L'objectif du problème est la recherche d'un vecteur $X = (x_1, x_2, x_3, ...x_n)^t$ dont les composantes valent 0 ou 1 réalisant le maximum de la fonction de la valeur totale $Z(X) = \sum_{i=1}^{n} v_i x_i$ sous la contrainte : $\sum_{i=1}^{n} w_i x_i \leq C$. C'est à dire que la somme des poids choisis ne dépasse pas la capacité C du sac à dos d'où le problème se modélise de la manière suivante :

$$\begin{cases} Z(X) = \sum_{i=1}^{n} v_i x_i \longrightarrow \max, \\ \sum_{i=1}^{n} w_i x_i \leq C, \\ x_i \in \{0,1\}, i \in \{1,..,n\}, \end{cases}$$

En général, on ajoute les contraintes suivantes afin d'éviter les cas singuliers : $\sum_{i=1}^{n} w_i > C$: on ne peut pas mettre tous les objets dans le sac.

$w_i \leq C, \forall i = 1,..,n$: aucun objet n'est plus lourd que ce que le sac peut porter.

$w_i > 0, \forall i \in (0,..,n)$ tout objet a un poids.

3.3 Les variantes du problèmes du sac à dos

Le problème présenté dans la section précédente est, plus précisément, le problème de sac à dos en variables binaires $(0-1KP)$. Il s'agit en fait d'une variante parmi d'autres. Les particularités se font sur le domaine des variables, le nombre de valeurs des objets, le nombre de dimensions du sac, etc. Ces particularités peuvent aussi être combinées [6].

3.3.1 Sac à dos en variables continues

Le problème du sac à dos en variables continues (LKP) est obtenu en enlevant la contrainte d'intégrité sur les variables. C'est-à-dire qu'on peut prendre une fraction des objets dans le sac à dos ($x_i \in [0, 1]$). (LKP) appartient à la classe de complexité P.

3.3.2 Sac à dos en variables entières

Dans le problème de sac à dos en variables entières, on considère que l'on a plusieurs exemplaires de chaque objet. Le problème consiste à trouver le nombre d'exemplaires à prendre pour chacun.
Si le nombre d'exemplaires est limité, on parlera de sac à dos borné (BKP), sinon on parlera de sac à dos non borné (UKP)[6].

3.3.3 Sac à dos multiple

Le problème du sac à dos multiple (MKP) est une généralisation du problème (KP), il consiste à répartir un ensemble d'objets dans plusieurs sacs à dos de capacités différentes, en respectant la contrainte de capacité de chaque sac et ce en maximisant le profit total c'est à dire, le nombre de contraintes de capacité est strictement supérieur à 1. Il existe une variante de ce problème où tous les sacs ont la même capacité.

3.3.4 Sac à dos multidimensionnel

On considère ici que le sac à dos a d dimensions, avec $d > 0$ (d-KP). Par exemple, le problème du remplissage d'un container dont le volume et la charge maximale sont limitées.

3.3.5 Sac à dos multi-objectif

Le problème du sac à dos multi-objectif consiste, à partir d'objets ayant plusieurs valeurs, à maximiser plusieurs fonctions objectifs, c'est le problème du sac à dos multi-objectif (MOKP). On parle alors d'optimisation multi-objectif. La notion d'optimalité

laisse place à la notion d'efficacité puisque nous avons plusieurs fonctions objectifs à optimiser.

3.3.6 Sac à dos quadratique

Le problème du sac à dos quadratique noté (QKP) est un problème dans le quel un gain G_{ij} supplémentaire est définit lorsque deux objets (i et j) sont pris simultanément.

3.3.7 Sac à dos à contraintes disjonctives

Le problème du sac à dos à contraintes disjonctives noté (DCKP) est un problème dans lequel un objet peut ne pas être compatible avec d'autres objets. Ce problème sera étudié dans le chapitre suivant [7].

3.4 Conclusion

Nous avons présenté, dans ce chapitre le problème du sac à dos (kp) ainsi que ses variantes les plus étudiées et les plus connues. Ces problèmes sont obtenus soit en modifiant ou en ajoutant une ou plusieurs contraintes de capacité au problème (KP).

‘

Chapitre 4

Sac à dos à contraintes disjonctives et sa résolution

4.1 Introduction

Dans ce chapitre, nous allons étudier le problème du sac à dos à contraintes disjonctives, l'une des variantes du problème du sac à dos qui est aussi un problème NP difficile qui a été introduit par Yamada et Kataoka en 2002[12, 13].

4.2 Définition du problème et Modélisation

Le problème du sac à dos avec contraintes disjonctives (DCKP) est un problème NP-difficile[11], en effet en enlevant la contrainte de disjonctivité ce problème se réduit au problème du sac à dos unidimensionnel, qui est lui même NP-difficile.
Le DCKP contient deux types de contraintes :

- La contrainte unique de capacité du sac à dos.
- Les contraintes disjonctives, qui traduisent la non-compatibilité entre certains objets et deux objets incompatibles ne doivent pas participer à la même solution.

Une instance du problème du sac à dos à contraintes disjonctives est définie par un ensemble de n objets et par une capacité C du sac, on associe à chaque objet $i = 1, ..., n.$ un profit v_i et un poids w_j. Certains couples (i,j) sont incompatibles, c'est

à dire que les deux objets du couple ne peuvent pas appartenir à la même solution. Soit $E = \{(i,j), 1 \leq i \neq j \leq n\}$ l'ensemble des couples disjonctifs et soit x_i la variable de decision associée à l'objet i

$$x_i = \begin{cases} 1, & \text{si l'objet i est pris ;} \\ 0, & \text{sinon .} \end{cases}$$

La contrainte disjonctive peut être formuler de la manière suivante :

$$x_i + x_j \leq 1, \forall (i,j) \in E.$$

La contrainte de capacité du sac à dos disjonctif est définie par :

$$\sum_{i=1}^{n} w_i x_i \leq C.$$

Le programme linéaire décrivant le problème du sac à dos à contraintes disjonctives s'écrit de la manière suivante :

$$(DCKP) \begin{cases} MaxZ(x) = \sum_{i=1}^{n} v_i x_i \\ \sum_{i=1}^{n} w_i x_i \leq C \\ x_i + x_j \leq 1, \forall (i,j) \in E \\ x_i \in \{0,1\}, i = 1,2,...,n \end{cases}$$

4.3 Les applications du DCKP

Nous pouvons retrouver le problème du sac à dos à contrainte disjonctives dans de nombreux domaines :

Dans les systèmes financiers : Étant donné un certain montant d'investissement dans les projets, quel projet choisir pour que le tout rapporte le plus de gain.

La contrainte disjonctive se traduit par le fait qu'il existe des projets qui ne peuvent pas se réaliser en même temps.

Dans le chargement de cargaisons, avions, bateaux, camions...

Il modélise un problème de sélection de sites pour installer des centrales nucléaire, soit un certain capital d'investissement, noté C et n sites pour construire un ensemble de

centrales nucléaire. Chaque site nécessite un coût et permet de produire une quantité d'énergie. Deux sites choisis doivent être à une distance pour des raison de sécurité et d'environnement.
L'objectif du problème est de sélectionner un sous ensemble de site de manière à maximiser la quantité total d'énergie.
En 2002, Yamada et kataoka[12] ont proposé deux méthodes approchées qui permettent de trouver des solutions réalisables. La première méthode est une méthode gloutonne qui permet d'avoir une solution initiale, la deuxième méthode utilise une recherche locale pour améliorer la solution donnée par la première méthode.

4.4 Quelques méthodes de résolution de DCKP

4.4.1 Algorithme glouton

Le principe de la méthode consiste à construire pas à pas une solution réalisable du problème. Il s'agit donc de fixer la valeur de la variable i sachant que les variables $x_1, x_2, \cdots, x_{i-1}$ sont fixées.
Si le poids de l'objet i ne dépasse pas la valeur $(C - \sum_{i=1}^{i-1} w_i x_i)$ appelée capacité résiduelle et que tous les objets du sac sont compatibles avec l'objet i alors $x_i = 1$ sinon $x_i = 0$.

4.4.2 Amélioration de l'algorithme glouton

Cet algorithme vise à améliorer la solution donnée par l'algorithme glouton.son principe consiste à remplacer un objet par un autre dans un couple $(i, j) \in E$
si $x_i = 1$ avec l'algorithme glouton, $x_j = 0$.
Dans l'algorithme d'amélioration on échange les objets c'est à dire : $x_i = 0$ et $x_j = 1$ si une amélioration de la solution courante est enregistrée.

4.5 Résolution du DCKP par une méthode exacte

Dans cette section, nous allons présenter une méthode exacte basée sur la méthode du simplexe et la méthode de branch and bound pour la résolution du problème du sac à dos

à contraintes disjonctives, puis nous allons illustrer cette méthode à l'aide d'un exemple.

4.6 Présentation de la méthode

Soit le problème du sac à dos à contraintes disjonctives suivant :

$$(P)\begin{cases} MaxZ(x) = \sum_{i=1}^{n} v_i x_i \\ \sum_{i=1}^{n} w_i x_i \leq C \\ x_i + x_j \leq 1, \forall (i,j) \in E \\ x_i \in \{0,1\}, i = 1,2,...,n \end{cases}$$

La méthode consiste à :

1- Déterminer une borne inférieure LB pour le problème (P) en utilisant l'heuristique suivante :

a) On représente un graphe G où les sommets sont les objets et les arêtes représentent la relation d'incompatibilité c'est à dire : deux sommets sont liés s'ils sont incompatibles.

b) On ordonne les objets par ordre décroissant de la valeur $\delta_i = \frac{v_i}{w_i + |\Gamma_i|}$ où Γ_i est l'ensemble des voisins de i (les objets qui sont incompatibles avec l'objet i).

$$\delta_1 \geq \delta_2 \geq .... \geq \delta_n$$

c) L'objet ayant la plus grande valeur de δ sera mis dans le sac, soit x cet objet.

d) On supprime le sommet x, ses voisins et toutes les arêtes reliant ces sommets. On obtient ainsi un sous graphe G' de G.

e) On réordonne les sommets du sous graphe par ordre décroissant de $\delta_i = \frac{v_i}{w_i + |\Gamma_i|}$

f) L'objet ayant la plus grande valeur de δ sera mis dans le sac, soit y cet objet.

g) On supprime le sommet y, ses voisins et toutes les arêtes reliant ces sommets. On obtient un sous graphe G'' de G'.

On répète les étapes e), f) et g) jusqu'à épuiser tous les sommets. Ainsi, on obtient une solution réalisable dont la valeur de Z sera la borne inférieure LB.

2- Construire deux problèmes (P_s)et $(P_{s'})$ de la manière suivante :

$$(P_s)\begin{cases} Min\sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} w_i x_i \leq C \\ \sum_{i=1}^{n} v_i x_i \geq LB \\ x_i + x_j \leq 1, \forall (i,j) \in E \\ x_i \in \{0,1\}, i = 1,2,...,n \end{cases} \qquad (P_{s'})\begin{cases} Max\sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} w_i x_i \leq C \\ \sum_{i=1}^{n} v_i x_i \geq LB \\ x_i + x_j \leq 1, \forall (i,j) \in E \\ x_i \in \{0,1\}, i = 1,2,...,n \end{cases}$$

3- Résoudre leurs relaxations continues c'est à dire $x_i \in [0,1]; i = 1,..,n$ en appliquant la méthode du simplexe, pour obtenir les deux solutions s et s' des problèmes respectifs (P_s)et $(P_{s'})$.

4- Injecter les contraintes $\sum_{i=1}^{n} x_i \geq s$ et $\sum_{i=1}^{n} x_i \leq s'$ dans le problème (P) pour obtenir le problème (P')suivant :

$$(P')\begin{cases} MaxZ(x) = \sum_{i=1}^{n} v_i x_i \\ \sum_{i=1}^{n} w_i x_i \leq C \\ \sum_{i=1}^{n} x_i \geq s \\ \sum_{i=1}^{n} x_i \leq s' \\ x_i + x_j \leq 1, \forall (i,j) \in E \\ x_i \in \{0,1\}, i = 1,2,...,n \end{cases}$$

5- Résoudre(P') par la méthode de branch and bound.

Lemme 4.6.1. Les deux problèmes (P) et (P') ont la même valeur optimale.

Preuve 1. Soient z_1 et z_2 les deux valeurs optimales respectivement de (P) et de (P').
On montre que $z_1 = z_2$ c'est à dire que $z_1 \leq z_2$ et $z_1 \geq z_2$.
On suppose qu'il existe deux solutions $X = (x_1,..,x_n)$ et $Y = (y_1,..,y_n)$ pour(P) et (P')dont les valeurs de la fonction objectif correspondante sont respectivement z_1 et z_2.
On a $z_1 \geq LB$ car LB est une borne inférieure du problème (P), donc $\sum_{i=1}^{n} v_i x_i \geq LB$, d'où X est solution réalisable de (P_s) et $(P_{s'})$ par consequent X est une solution réalisable de (P').

Y est une solution optimale de (P') de valeur objectif z_2 et X est une solution réalisable de (P') de valeur objectif z_1. Comme (P') est un problème de maximisation (toute solution réalisable est plus petite ou égale à la solution optimale) alors $z_1 \leq z_2$ (1) De plus, comme Y est une solution réalisable de (P') (car toute solution optimale est réalisable), alors $z_1 \geq z_2$ (2).
De (1) et (2), on déduit que $z_1 = z_2$, donc les deux problèmes (P) et (P') ont la même valeur optimale.

4.7 Exemple

Nous allons appliquer cette méthode pour l'exemple suivant :

$$(P)\begin{cases} MaxZ(x) = 32x_1 + 52x_2 + 64x_3 + 35x_4 + 96x_5 + 18x_6 + 66x_7 + 15x_8 + 13x_9 + 4x_{10} \\ 3x_1 + 6x_2 + 9x_3 + 12x_4 + 38x_5 + 9x_6 + 77x_7 + 40x_8 + 96x_9 + 95x_{10} \leq 200 \\ x_1 + x_2 \leq 1 \\ x_1 + x_3 \leq 1 \\ x_1 + x_9 \leq 1 \\ x_3 + x_6 \leq 1 \\ x_3 + x_7 \leq 1 \\ x_4 + x_9 \leq 1 \\ x_5 + x_6 \leq 1 \\ x_7 + x_9 \leq 1 \\ x_8 + x_9 \leq 1 \\ x_8 + x_{10} \leq 1 \\ x_4 + x_5 \leq 1 \\ x_i \in \{0, 1\}, i = 1, 2, ..., 10 \end{cases}$$

Étape 1 :

On applique l'heuristique précédente pour avoir une solution réalisable qui sera la borne inférieure LB. Pour cela, on représente la relation d'incompatible entre les objets à l'aide du graphe suivant :

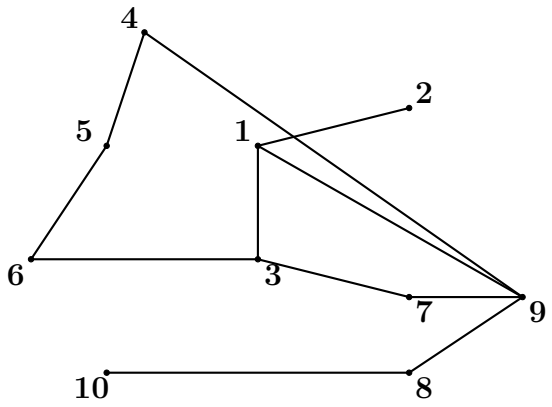

FIG. 4.1 – Graphe G

On calcule pour chaque sommet i la valeur de $\delta_i = \frac{v_i}{w_i+|\Gamma_i|}, i = 1, .., 10$ où Γ_i sont les voisins du sommet i.

$\delta_1 = \frac{v_1}{w_1+3} = \frac{32}{3+3} = 3,555$

$\delta_2 = \frac{v_2}{w_2+1} = \frac{52}{6+1} = 7.42$

$\delta_3 = \frac{v_3}{w_3+3} = \frac{64}{9+3} = 5,33$

$\delta_4 = \frac{v_4}{w_4+1} = \frac{35}{12+2} = 2,5$

$\delta_5 = \frac{v_5}{w_5+2} = \frac{96}{82+2} = 2,40$

$\delta_6 = \frac{v_6}{w_6+2} = \frac{18}{9+2} = 1,63$

$\delta_7 = \frac{v_7}{w_7+2} = \frac{66}{77+2} = 0,83$

$\delta_8 = \frac{v_8}{w_8+2} = \frac{15}{40+2} = 0,35$

$\delta_9 = \frac{v_9}{w_9+4} = \frac{13}{96+4} = 0,13$

$\delta_{10} = \frac{v_{10}}{w_{10}+1} = \frac{4}{95+1} = 0,04$

Puis, on ordonne les sommets par ordre décroissant de la valeur δ_i

i	2	3	1	4	5	6	7	8	9	10
δ_i	7.42	5.33	3.55	2.5	2.40	1.63	0.83	0.35	0.13	0.04

Soit $X = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10})$, la solution réalisable à trouver.

On remarque que le sommet de la plus grande valeur de δ est le sommet 2, donc l'objet 2 sera mis dans le sac ($x_2 = 1$), et les voisins de 2 ne seront pas pris. le sommet 1 est le voisin du sommet 2, donc ($x_1 = 0$).

On supprime les sommets 1 et 2 et toutes les arêtes reliant ses sommets, ainsi on obtient le sous graphe suivant :

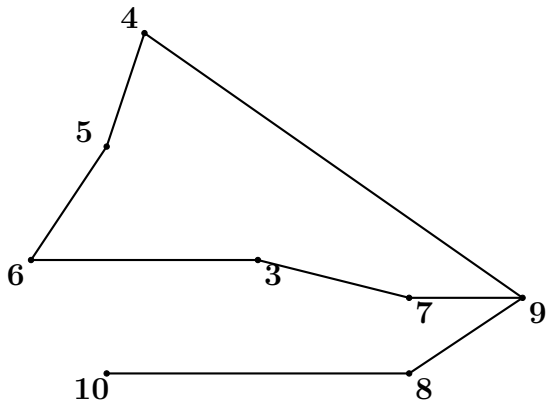

FIG. 4.2 – Graphe G'

Puis, on réordonne les sommets du sous graphe par ordre décroissant de la valeur δ_i.

i	3	4	5	6	7	8	9	10
δ_i	5.81	2.5	2.40	1.63	0.83	0.35	0.13	0.04

Le sommet de la plus grande valeur de δ est le sommet 3, donc l'objet 3 sera mis dans le sac ($x_3 = 1$), et les voisins de 3 ne seront pas pris. Les sommets 6 et 7 sont les voisins du sommet 3, donc ($x_6 = 0$) et ($x_7 = 0$).

On supprime les sommets 3,6 et 7 et toutes les arêtes reliant ses sommets, ainsi on obtient le sous graphe de G' suivant :

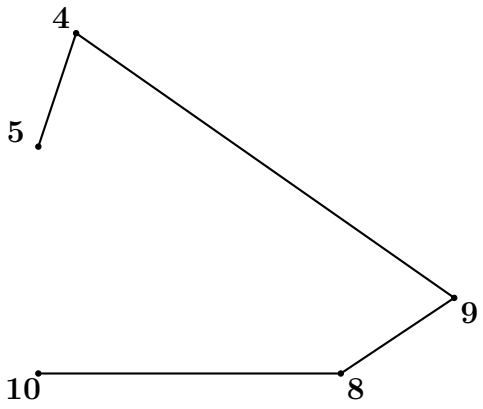

FIG. 4.3 – Graphe G"

Puis, on réordonne les sommets du sous graphe G'' par ordre décroissant de la valeur δ_i.

i	4	5	8	9	10
δ_i	2.5	2.46	0.35	0.13	0.04

Le sommet de la plus grande valeur de δ est le sommet 4, donc l'objet 4 sera mis dans le sac ($x_4 = 1$), et les voisins de 4 ne seront pas pris. Les sommets 5 et 9 sont les voisins du sommet 4, donc ($x_5 = 0$ *et* $x_9 = 0$).

On supprime les sommets 4, 5 et 9 et toutes les arêtes reliant ses sommets, ainsi on obtient le sous graphe de G''' suivant :

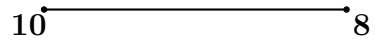

FIG. 4.4 – Graphe G"'

Puis, on réordonne les sommets du sous graphe G''' par ordre décroissant de la valeur δ_i.

i	8	10
δ_i	0.36	0.04

Le sommet de la plus grande valeur de δ est le sommet 8, donc l'objet 8 sera mis dans le sac ($x_8 = 1$), et les voisins de 8 ne seront pas pris. Le sommets 10 est le voisin du sommet 8, donc ($x_{10} = 0$).

Ainsi, on obtient la solution réalisable : $X = (0, 1, 1, 1, 0, 0, 0, 1, 0, 0)$, avec $Z = 52 + 64 + 35 + 15 = 166$ qui est la borne inférieure du problème (P). C'est à dire : $LB = 166$.

Etape2 :

On construit les deux problèmes (P_s) et $(P_{s'})$:

$$
(P_s)\begin{cases}
x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \rightarrow Min \\
3x_1 + 6x_2 + 9x_3 + 12x_4 + 38x_5 + 9x_6 + 77x_7 + 40x_8 + 96x_9 + 95x_{10} \leq 200 \\
32x_1 + 52x_2 + 64x_3 + 35x_4 + 96x_5 + 18x_6 + 66x_7 + 15x_8 + 13x_9 + 4x_{10} \geq 166 \\
x_1 + x_2 \leq 1 \\
x_1 + x_3 \leq 1 \\
x_1 + x_9 \leq 1 \\
x_3 + x_6 \leq 1 \\
x_3 + x_7 \leq 1 \\
x_4 + x_9 \leq 1 \\
x_5 + x_6 \leq 1 \\
x_7 + x_9 \leq 1 \\
x_8 + x_9 \leq 1 \\
x_8 + x_{10} \leq 1 \\
x_4 + x_5 \leq 1 \\
x_i \in \{0, 1\}, i = 1, 2, ..., 10
\end{cases}
$$

$$(P_{s'})\begin{cases} x_1+x_2+x_3+x_4+x_5+x_6+x_7+x_8+x_9+x_{10} \rightarrow Max \\ 3x_1+6x_2+9x_3+12x_4+38x_5+9x_6+77x_7+40x_8+96x_9+95x_{10} \leq 200 \\ 32x_1+52x_2+64x_3+35x_4+96x_5+18x_6+66x_7+15x_8+13x_9+4x_{10} \geq 166 \\ x_1+x_2 \leq 1 \\ x_1+x_3 \leq 1 \\ x_1+x_9 \leq 1 \\ x_3+x_6 \leq 1 \\ x_3+x_7 \leq 1 \\ x_4+x_9 \leq 1 \\ x_5+x_6 \leq 1 \\ x_7+x_9 \leq 1 \\ x_8+x_9 \leq 1 \\ x_8+x_{10} \leq 1 \\ x_4+x_5 \leq 1 \\ x_i \in \{0,1\}, i=1,2,...,10 \end{cases}$$

Etape3 :

On résout par la méthode du simplexe les problèmes relaxés des problèmes (P_s) et $(P_{s'})$ c'est à dire remplacer la contrainte $x_i \in \{0,1\}, i = 1,2,...,10$ par la contrainte $x_i \in [0,1], i = 1,2,...,10$, pour trouver les valeurs de s et s'.

FIG. 4.5 – Solution de P_s

FIG. 4.6 – Solution de $P_{s'}$

Les solutions des problèmes (P_s) et $(P_{s'})$ sont respectivement $X_s = (0, 0.077, 0, 0, 1, 0, 1, 0, 0, 0)$ avec$s = 0.77 + 1 + 1 = 2.077$ et $X_{s'} = (0.45, 0.55, 0, 1, 0, 1, 1, 0, 0, 1)$ et $s' = 0.45 + 0.55 + 1 + 1 + 1 + 1 = 5$

Etape4 :

On injecte les contraintes $\sum_{i=1}^{10} x_i \geq s$ et $\sum_{i=1}^{10} x_i \leq s'$ dans le problème (P) pour obtenir le problème (P') suivant :

$$(P') \begin{cases} MaxZ(x) = 32x_1 + 52x_2 + 64x_3 + 35x_4 + 96x_5 + 18x_6 + 66x_7 + 15x_8 + 13x_9 + 4x_{10} \\ 3x_1 + 6x_2 + 9x_3 + 12x_4 + 38x_5 + 9x_6 + 77x_7 + 40x_8 + 96x_9 + 95x_{10} \leq 200 \\ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \geq 2.077 \\ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \leq 5 \\ x_1 + x_2 \leq 1 \\ x_1 + x_3 \leq 1 \\ x_1 + x_9 \leq 1 \\ x_3 + x_6 \leq 1 \\ x_3 + x_7 \leq 1 \\ x_4 + x_9 \leq 1 \\ x_5 + x_6 \leq 1 \\ x_7 + x_9 \leq 1 \\ x_8 + x_9 \leq 1 \\ x_8 + x_{10} \leq 1 \\ x_4 + x_5 \leq 1 \\ x_i \in \{0, 1\}, i = 1, 2, ..., 10 \end{cases}$$

Etape5 :

On résout le problème (P') par la méthode de branch and bound, Pour cela, on résout tout d'abord le problème relaxé de (P') par la méthode du simplexe, si la solution obtenue est entière, alors elle sera la solution optimale de(P'), si elle n'est pas entière, on applique le branchement.

Le problème relaxé est :

$$
(LP') \begin{cases}
MaxZ(x) = 32x_1 + 52x_2 + 64x_3 + 35x_4 + 96x_5 + 18x_6 + 66x_7 + 15x_8 + 13x_9 + 4x_{10} \\
3x_1 + 6x_2 + 9x_3 + 12x_4 + 38x_5 + 9x_6 + 77x_7 + 40x_8 + 96x_9 + 95x_{10} \leq 200 \\
x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \geq 2.077 \\
x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \leq 5 \\
x_1 + x_2 \leq 1 \\
x_1 + x_3 \leq 1 \\
x_1 + x_9 \leq 1 \\
x_3 + x_6 \leq 1 \\
x_3 + x_7 \leq 1 \\
x_4 + x_9 \leq 1 \\
x_5 + x_6 \leq 1 \\
x_7 + x_9 \leq 1 \\
x_8 + x_9 \leq 1 \\
x_8 + x_{10} \leq 1 \\
x_4 + x_5 \leq 1 \\
x_i \in [0, 1]
\end{cases}
$$

FIG. 4.7 – Solution de LP'

La solution Optimale de (LP') est $X = (0, 1, 0, 0, 1, 0, 1, 1, 0, 0)$, avec $Z = 52 + 96 + 66 + 15 = 229$ qui est entière, donc elle est la solution optimale du problème (P') par consequent la solution optimale du problème (P).

Conclusion générale

Dans ce mémoire, nous avons étudié le problème du sac à dos dos à contraintes disjonctives pour lequel nous avons utilisé une méthode exacte pour le résoudre.
Le problème du sac à dos à contraintes disjonctives qui est un problème NP-difficile étudié en optimisation combinatoire.
Il est considéré comme une variante du problème du sac à dos classique (KP), pour lequel on ajoute la contrainte d'incompatibilité entre les objets, c'est à dire deux objets incompatibles ne doivent pas participer à la même solution.
Ce travail, nous a permis de comprendre certaines méthodes d'optimisation combinatoire basées sur Branch and Band et le Simplexe qui sont des méthodes exactes.
Nous avons ensuite présenté la résolution d'un exemple pratique par les deux dernières méthodes dans un problème du sac à dos à contraintes disjonctives en nombres entiers pour obtenir une solution optimal à la fin.
En perspectives, nous envisageons d'étudier d'autres variantes du problème du sac à dos.

Annexe

Graphe non orienté

Un graphe non-orienté G est une paire (V,E), où V est un ensemble fini de sommets, et E un ensemble fini d'arêtes, i.e. de paires de sommets (non-ordonnées). On note par V(G) et E(G) les ensembles de sommets et d'arêtes de G, respectivement, on note aussi par $n = |V|$ et $m = |E|$ le nombre de sommets et d'arêtes de G, respectivement. Le nombre de sommets est appelé **ordre** du graphe G.

Définition 4.7.1. Une boucle est une arête reliant un sommet à lui même.
Si e est une arête reliant deux sommets u et v, alors u et v seront appelés les extrémités de e, et nous écrirons e = uv ou $e = \{u, v\}$. Si u est une extrémité de e, alors u (resp.e) est dit incident à e (resp. u). De même, deux sommets u et v formant une arête sont dits adjacents.
Le degré d'un sommet x est égal au nombre d'arêtes dont ce sommet est une extrémité. On le note par $d(x)$

Définition 4.7.2. Un graphe simple est un graphe sans boucle tel que, entre deux sommets, il y ait au plus une arête.
Les graphes contenant des boucles, ou plusieurs arêtes reliant les deux mêmes sommets sont appelés multi graphes

Chaîne

Définition 4.7.3. Soient u et v deux sommets de V. Une chaîne P entre u et v est une séquence alternée de sommets et d'arêtes $(v_0, e_1, v_1, e_2, v_2, ..., v_{k-1}, e_k, v_k)$ où $v_0 = u$, $v_k = v$ et $e_i = v_{i-1}v_i$ pour $i = 1, ..., k$.
P est dite élémentaire si elle passe au plus une fois par le même sommet (à l'exception de v_0 et v_k si ces derniers représentent le même sommet de G).
Une chaîne élémentaire est donc totalement identifiée par son ensemble d'arêtes.
Une chaîne est simple si chaque arête apparaît au plus une fois.

Cycle

Définition 4.7.4. Un cycle, dans un graphe, est une chaîne dont les extrémités coïncident, composée d'arêtes toutes distinctes.
Un cycle élémentaire est une chaîne élémentaire dont les extrémités coïncident.
Un cycle simple est une chaîne simple dont les extrémités coïncident.

Graphe connexe

Un graphe est connexe lorsque, pour chaque paire de sommets, il existe au moins une chaîne reliant les deux sommets.

Arbre

Le graphe G est un arbre si et seulement si G est connexe et ne contient pas de cycle. De manière équivalente, G est un arbre si et seulement si G est connexe et comporte n-1 arêtes.

Définition 4.7.5. Une racine est un sommet relié a tous les sommets de graphe par une chaîne.

Définition 4.7.6. Une arborescence est un arbre contenant une racine.

Bibliographie

[1] Hans K, Ulright P et *al*. Knapsack Problems, Springer, 2004 (ISBN 3-540-40286-1).

[2] Jaques T. Recherche Opérationnelle Tome 1,Ellipses, Paris, 2012.

[3] Jens C. Branch and bound algorithms-principles and examples. Department of Computer Science, University of Copenhagen, pages 1–30, 1999.

[4] Kellerer H, Pferschy U et *al*. Knapsack Problems. Springer, 2003.

[5] Michel B. Optimisation en Nombres Entiers, Cours Recherche Opérationnelle, Ecole Polythechnique Fédérale de Lausanne, 2010.

[6] Michrafy M. Contribution à la résolution de quelques problèmes de type sac-à-dos : méthodes exactes et heuristiques. PhD thesis, CERMSEM, Université Paris 1, 2005.

[7] Mohamed Elhavedh O A M. Contribution à la résolution du sac-à-dos à contraintes disjonctives. PhD thesis, Université de Picardie Jules Verne, 2009.

[8] Papadimitriou Ch.Combinatorial Optimization :Algorithms and Complexity.Prentice-Hall, New York, 1982.

[9] Pierre F. Optimisation combinatoire :Programmation linéaire et algorithmes. Université Pierre et Marie Curie, 2015.

[10] Stanislaw W.Integer Programming. Mathematics and its Applications. Springer, Varsovie, 1990.

[11] Yamada T, Seiji K et *al*. Heuristic and Exact Algorithms for the Disjonctivety Constrained Knapsack Problem. IPSJ Journal,vol 43, NO 9. SEP 2002.

[12] Yamada T, Kataoka S et *al*. Heuristic and exact algorithms for the disjunctively constrained knapsack problem, Information Processing Society of Japan Journal, 43 : 2864-2870, 2002.

[13] Yamada T and Kataoka S. Heuristic and exact algorithms for the disjunctively constrained knapsack problem, Presented at EURO 2001, Rotterdam, The Netherlands, July 9-11, 2001.