UNIVERSITÉ YAHIA FARÈS DE MÉDÉA

Faculté des Sciences

Département de Mathématiques et de l'Informatique



MÉMOIRE DE MASTER

en mathématiques

Spécialité : Recherche Opérationnelle

THÈME

PROBLÈME DU SAC-À-DOS À CONTRAINTES DISJONCTIVES : MODÉLISATION ET RÉSOLUTION

présenté

par

BENYAHIA Aicha et FETTAK Nourelhoda

évalué devant le Jury composé de :

S. KOUIDER AIAD	M.A.A	U.YF.M	Examinatrice
R. LEMEDANI	M.A.A	U.YF.M	Examinateur
A. BENKACI	M.A.A	U.YF.M	Encadreur

Médéa, 30 – septembre – 2020

Résumé

Dans ce mémoire, nous nous intéressons au problème du sac-à-dos à contraintes disjonctives, qui est une extension directe du problème de sac-à-dos classique. Ce problème est classé NP-difficile, et il est formulé comme un programme linéaire en nombres binaires (0/1). Pour le résoudre une méthode approchée de type Recherche Locale (RL) est appliquée en élaborant un code de calcul en langage Matlab qui permet de résoudre un exemple du problème en temps réel. Nous reportons aussi des résultats numériques qui sont obtenus en testant notre code de calcul sur un exemple type. Ces résultats confirment l'efficacité de notre approche.

Mots–clés : modélisation mathématique, problème du sac-à-dos, programmation en nombre entiers, contraintes disjonctives.

Remerciements

Tout d'abord, nous tenons à remercier, ALLAH le Tout Puissant, pour nous avoir données le courage et la force pour accomplir ce modeste travail.

Nous tenons à exprimer toute notre gratitude à notre enseignant Monsieur Benkaci Azzedine, enseignant à la faculté des sciences de l'Université de Médéa, pour nous avoir encadrées et guidées tout au long de ce mémoire. Il nous a permis de réaliser ce travail en nous manifestant sa confiance et en nous encourageant dans notre travail. Il a été toujours à notre disposition.

Nous souhaitons également exprimer toute notre reconnaissance aux enseignants du département MI, pour leur immense effort consacré à la formation du Master mathématique au sein de la faculté des sciences.

Nous remercions aussi tous ceux qui ont contribué à la rédaction de ce manuscrit. Nous ne pourrons terminer sans mentionner tout ce que nous devons à nos parents et à nos frères et sœurs. Ils nous ont toujours soutenus et supportés. Sans eux ce travail n'aurait pu voir le jour.

Enfin, nous remercions toute personne qui nous a aidé par sont soutient et sa confiance afin d'accomplir ce modeste travail.

Table des matières

Résumé i

Remerciements ii

Table des matières iii

Liste des figures vi

Liste des tableaux vii

Liste des algorithmes viii

Introduction 9

1 Outils mathématiques 11

1.1 Introduction 11

1.2 Notions préliminaires de Graphes 11

1.2.1 Graphe partiel et sous-graphe 12

1.2.2 Matrice d'adjacence associée à un graphe 14

1.2.3 Différents types de graphe 14

1.3 Programmation Linéaire en Nombres Entiers 16

1.3.1 Qualité de formulation algébrique (Large ou Serrée) 17

1.3.2 Inégalité valide et coupe valide 17

1.3.3 Exemple (Stable de taille maximum) 18

1.4 Conclusion 19

2 Modélisation mathématique 20

2.1 Introduction 20

2.2 Problème du sac-à-dos unidimensionnel 20

2.2.1 Position du problème 20
2.2.2 Modèle mathématique 21
2.3 Problème du sac-à-dos à contraintes disjonctives 21
2.3.1 Position du problème 21
2.3.2 Description des contraintes 22
2.3.3 Modèle mathématique 22
2.3.4 Complexité du problème 23
2.4 Conclusion 23

3 Méthodes de résolution 24
3.1 Introduction 24
3.2 Problème d'optimisation combinatoire 24
3.3 Processus de résolution d'un problème réel 24
3.4 Méthodes de résolution 25
3.4.1 Méthodes exactes 26
3.4.2 Méthodes approchées 26
3.5 Principe de la méthode de Recherche Locale 27
3.6 Algorithme de Recherche Locale (RL) 27
3.7 Conclusion 28

4 Implémentation et tests numériques 29
4.1 Introduction 29
4.2 Adaptation et mise en œuvre 29
4.2.1 Reformulation du problème 29
4.2.2 Fonction de pénalité 30
4.2.3 Réduction de l'espace de recherche 30
4.3 Exemple Test 31
4.4 Résultats obtenus 32
4.5 Conclusion 33

Conclusion 34

A. Codes MATLAB 36

A.1 Fichier des données (paramètres) 36
A.2 Fichier de génération de la solution de départ 36
A.3 Script de résolution et affichage des résultats 37
A.4 Fichier de calcul du paramètre k 38

Notations et abréviations **39**

Références **40**

Liste des figures

Figure 1.1 : Un graphe $G = (V, E)$ d'ordre 6 comportant 7 arêtes. 13

Figure 1.2 : Graphe partiel et Sous-graphe de *G*. 13

Figure 1.3 : Différents types de graphe. 15

Figure 3.1 : Schéma conceptuel de résolution d'un problème réel. 25

Figure 3.2 : Principe général de la méthode de Recherche Locale (RL). 27

Figure 4.1 : Comportement de l'algorithme sur l'exemple test. 32

Liste des tableaux

Tableau 4.1 : Profit et Poids de chaque objet. 31

Tableau 4.2 : Relation d'incompatibilité entre les objets. 31

Tableau 4.3 : Meilleure sélection des objets compatibles. 32

Liste des algorithmes

Algorithme 3.1 : Algorithme de recherche locale typique. 28

Algorithme A.1 : Données pour la création de l'exemple. 36

Algorithme A.2 : Génération de la solution de départ. 36

Algorithme A.3 : Résolution de l'exemple. 37

Algorithme A.4 : Calcul du paramètre k. 38

Introduction

La Recherche Opérationnelle (RO) a pour but d'apporter une démarche scientifique à la résolution de problèmes de décision complexes issus du monde réel. Sa tendance scientifique est de construire des modèles mathématiques d'aide à la décision, en particulier les modèles liés à des problèmes d'optimisation et de proposer des méthodes efficace pour les résoudre. La RO intervient de manière cruciale dans des secteurs nombreux et diversifiés comme les transports, la production industrielle, la planification et la logistique.

Dans ce mémoire, nous nous intéressons au problème du sac-à-dos à contraintes disjonctives (ou, sac-à-dos disjonctif). Il est considéré comme une variante du problème du sac-a-dos classique, sauf qu'un objet peut ne pas être compatible avec d'autres d'objets du sac. Résoudre un problème du sac à dos disjonctif revient à trouver un sous ensemble d'objets maximisant le profit total tout en respectant la capacité du sac et que tous les objets soient compatibles.

Notre problème peut être formulé comme un programme linéaire en nombres binaires. Il est considéré comme une extension directe du problème du sac-à-dos classique, et il est classé NP-difficile. Pour le résoudre, nous appliquons une méthode approchée de type Recherche Locale (RL).

La Recherche Locale est une méthode générale utilisée pour résoudre des problèmes d'optimisation combinatoire, elle consiste à passer d'une solution à une autre solution proche dans l'espace des solutions candidates (l'espace de recherche) jusqu'à ce qu'une solution considérée comme optimale soit trouvée, ou que le temps imparti soit dépassé.

Dans ce mémoire, notre objectif est double :

- Appliquer une méthode approchée de type Recherche Locale pour une résolution efficace de notre problème.
- Implémenter un programme informatique permettant la résolution effective du problème.

Ce mémoire est composé de quatre chapitres, d'une conclusion générale et d'une annexe.

Le premier chapitre est consacré à la présentation de quelques définitions sur les graphes simples non orientés. Les principaux concepts de la Programmation Linéaire en Nombres Entiers (PLNE) sont rappelés, ainsi que leur développement mathématique qui va nous être utile tout au long de ce mémoire.

Dans le deuxième chapitre, nous exposons d'abord le modèle mathématique du problème du sac-à-dos classique, ensuite nous présentons le programme linéaire en nombres bivalents (0/1) modélisant le problème du sac-à-dos à contraintes disjonctives. La complexité de ce dernier est abordée de manière concise.

Dans le chapitre trois, nous rappelons brièvement la définition d'un problème d'optimisation combinatoire, Les définitions et les principes généraux des méthodes de résolution de ce dernier, sont cités. Nous présentons aussi, un pseudo algorithme de Recherche Locale typique (RL).

Le chapitre quatre est consacré à l'implémentation de notre approche de résolution. Nous reportons aussi des résultats numériques qui ont été obtenus en testant notre code de calcul sur un exemple réel du problème.

Dans l'annexe-A, nous présentons les scripts de nos programmes informatiques codés en langage MATLAB, ainsi que les résultats numériques obtenus sur un exemple type.

Enfin, nous terminons notre travail par une conclusion générale.

1

Outils mathématiques

1.1 Introduction

Dans ce chapitre nous présentons quelques définitions sur les graphes simples non orientés. Les principaux concepts de la Programmation Linéaire en Nombres Entiers (PLNE) sont rappelés, ainsi que leur développement mathématique qui va nous être utile tout au long de ce mémoire.

1.2 Notions préliminaires de Graphes

- Un graphe est défini par deux ensembles : un ensemble $V = \{v_1, v_2, \ldots, v_n\}$ dont les éléments sont appelés sommets, et un ensemble $E = \{e_1, e_2, \ldots, e_m\}$ dont les éléments sont appelés arêtes. On note $G = (V, E)$.
- Une arête e est une paire de sommets $(u, v) \in E \times E$. Les sommets u et v sont les extrémités de l'arête $e = (u, v)$.
- Deux sommets d'un graphe sont voisins ou adjacents s'ils ont une arête incidente en commun.
- Deux arêtes d'un graphe sont adjacentes si elles ont un sommet en commun.
- On appelle ordre d'un graphe le nombre de sommets de ce graphe.
- Le nombre d'arêtes incidentes à un sommet est le nombre d'arêtes sortant ou rentrant du sommet.
- Le degré d'un sommet v de G est le nombre d'arêtes incidentes à v. Il est noté $d(v)$. On note aussi :
- Le degré maximum $\Delta(G) = \max\{d(v) : v \in V\}$.

- Le degré minimum $\delta(G) = \min\{d(v) : v \in V\}$.
- Lorsque $d(v) = 0$, on dit que le sommet v est isolé. Lorsque $d(v) = 1$, on dit que le sommet v est pendant.
- L'ensemble de sommets voisins de *v*, est noté $\Gamma(v) = \{u \in V : (v,u) \in E\}$.

Exemple : Soit, le graphe $G = (V, E)$ représenté dans la Figure 1.1.

Dans cet exemple :

- $V = \{1,2,3,4,5,6\}$ et $E = \{(1,2),(1,6),(2,3),(2,6),(3,4),(3,5),(5,6)\}$.
- Trois arêtes sont incidentes au sommet 2. Le sous-ensemble $\Gamma(2) = \{1,3,6\}$ représente les voisins du sommet 2, et $d(2) = |\Gamma(2)| = 3$.
- Le sommet 4 est pendant : $d(4) = 1$
- Le degré maximal et minimal sont, respectivement : $\Delta(G) = 3$ et $\delta(G) = 1$.

Propriétés : On donne ici, deux propriétés très utiles en théorie des graphes.

- La somme de degrés des sommets d'un graphe est égale à deux fois son nombre d'arêtes. $\sum_{v \in V} d(v) = 2|E|$.
- Le nombre de sommets de degré impair d'un graphe est pair.

1.2.1 Graphe partiel et sous-graphe

Beaucoup de problèmes d'optimisation consistent à chercher un graphe partiel ou un sous-graphe vérifiant certaine propriété, dans un graphe G.

- Soit $G = (V, E)$ un graphe. Le graphe $G' = (V, E')$ est un graphe partiel de G, si E' est inclus dans E.
- Un sous-graphe de G est un graphe $H = (V', E(V'))$ tel que V' est un sous ensemble de V, et $E(V')$ sont les arêtes induites par E sur V'.

La Figure 1.2, présente un graphe partiel et un sous-graphe de G.

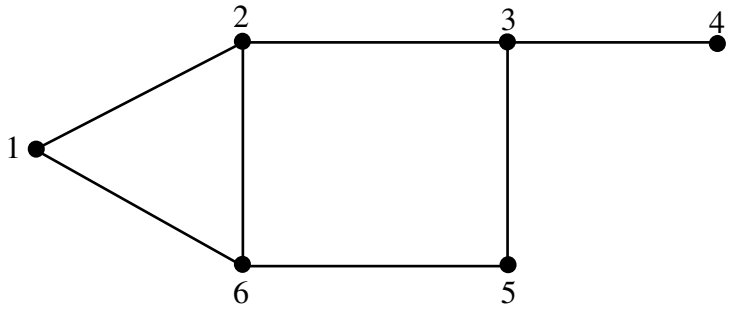
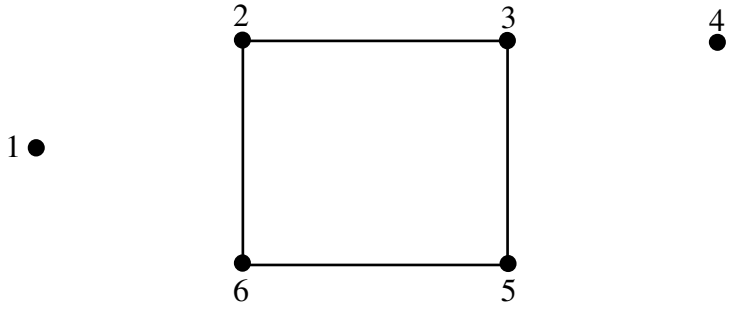
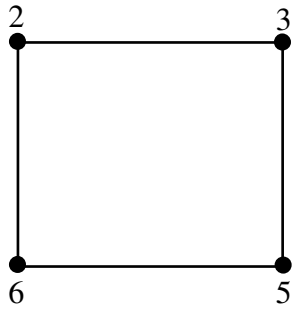


Figure 1.1 : Un graphe $G = (V, E)$ d'ordre 6 comportant 7 arêtes.



a) Graphe partiel $G' = (V, E')$ de G



b) Sous-graphe $H = (V', E(V'))$ de G

Figure 1.2 : Graphe partiel et Sous-graphe de G.

1.2.2 Matrice d'adjacence associée à un graphe

Parmi les matrices associées à un graphe donné, on cite la matrice d'adjacence. Elle joue un rôle déterminant pour la modélisation de problèmes d'optimisation.

Définition 1.1 : Soit $G = (V, E)$ un graphe simple, où $V = \{v_1, v_2, \ldots, v_n\}$. La matrice d'adjacence de G est la matrice carrée A dont les coefficients sont définis par :

$$A_{ij} := \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & \text{sinon} \end{cases}$$

Propriétés : La matrice A est symétrique à diagonale nulle, en plus les égalités suivantes sont vérifiées :

$$\sum_{j=1}^{n} A_{ij} = d(v_i), \qquad i \in \{1, 2, \ldots, n\} \tag{1.1}$$

$$\sum_{i=1}^{n} A_{ij} = d(v_j), \qquad j \in \{1, 2, \ldots, n\} \tag{1.2}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} = 2|E| \tag{1.3}$$

La matrice d'adjacence associée à G (voir Figure 1.1) est la suivante :

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

1.2.3 Différents types de graphe

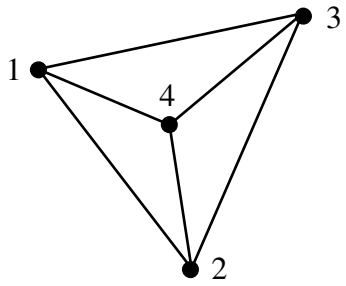
Dans cette section, nous présentons quelques types de graphes, à savoir le stable, la clique, le biparti et le biparti complet.

Définition 1.2 (Graphe stable) Soit $G = (V, E)$ un graphe. Un sous-ensemble S de V est dit stable s'il ne comprend que des sommets non adjacents deux à deux.

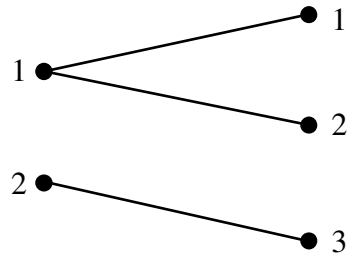
Définition 1.3 (Graphe complet) Un graphe complet est un graphe où chaque sommet est relié à tous les autres. Le graphe complet d'ordre n est noté K_n. Dans ce graphe chaque sommet est de degré $n-1$.

Définition 1.4 (Graphe biparti) Un graphe $G = (V, E)$ est biparti ou bipartite si l'ensemble de ses sommets V peut être divisé en deux ensembles disjoints V_1 et V_2, de sorte que les arêtes relient uniquement des éléments de V_1 à des éléments de V_2.

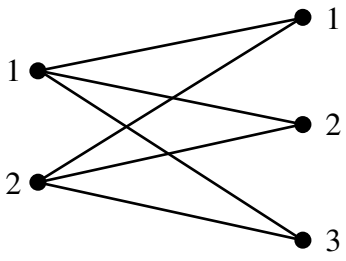
Définition 1.5 (Graphe biparti complet**)** Un graphe bipartite complet $G = (V_1, V_2, E)$ noté $K_{p,q}$ est un graphe bipartite où chaque sommet de V_1 est relié à tous les sommets de V_2 par une arête. Ici, $|V_1| = p$ et $|V_2| = q$.



a) Graphe complet K_4



b) Graphe biparti



c) Graphe biparti complet $K_{2,3}$

Figure 1.3 : Différents types de graphe.

1.3 Programmation Linéaire en Nombres Entiers

Dans cette section, nous rappelons, brièvement, les outils de base de la programmation en nombres entiers. On se focalise sur la notion de la coupe valide.

Tout programme linéaire en nombres entiers (PLNE) peut s'écrire ainsi :

$$(\text{PLNE}) \equiv \begin{cases} \max cx \\ x \in X = P \cap \mathbb{Z}^n \end{cases} \tag{1.4}$$

où, $P = \left\{ x \in \mathbb{R}^n : Ax \leq b \text{ et } x \geq 0 \right\}$ est un polyèdre (polytope borné et fermé).

Définition 1.6 (Enveloppe convexe) L'enveloppe convexe de X, notée $\text{conv}(X)$, est le plus petit ensemble convexe contenant toutes solutions de X, et on note :

$$\text{conv}(X) = \left\{ x \in \mathbb{Z}^n : A'x \leq b' \text{ et } x \geq 0 \right\}$$

On considère les problèmes d'optimisation suivants :

Problème originaire : $$Z_X = \max\{cx : x \in X\} \tag{1.5}$$

Problème équivalent : $$Z_{\text{conv}(X)} = \max\{cx : x \in \text{conv}(X)\} \tag{1.6}$$

Problème relaxé : $$Z_P = \max\{cx : x \in P\} \tag{1.7}$$

Propriétés :

i. La valeur Z_P est une borne supérieure de Z_X : $Z_X \leq Z_P$.

ii. Les points extrêmes de $\text{conv}(X)$ sont des solutions de X : $Z_X = Z_{\text{conv}(X)}$.

iii. Optimiser une fonction linéaire sur le domaine $\text{conv}(X)$ est la formulation idéale pour le problème d'optimisation (PLNE).

Remarque : Les propriétés (ii) et (iii) sont très importantes. Malheureusement, la détermination de l'enveloppe convexe pour l'ensemble X est un problème aussi difficile que le problème d'optimisation (PLNE) originaire.

1.3.1 Qualité de formulation algébrique (Large ou Serrée)

Un problème d'optimisation peut avoir plusieurs formulations différentes mais équivalentes. On pratique, il est judicieux d'adopter la bonne formulation possible servant à faciliter la résolution du problème. On parle sur la formulation serrée.

- Une «bonne» formulation est une formulation pour laquelle le polyèdre associé à la relaxation linéaire est proche de l'enveloppe convexe des solutions du problème, en particulier autour de la solution optimale.
- La relaxation linéaire d'une formulation serrée procure de meilleures bornes sur la solution optimale du problème.
- Une formulation serrée peut accélérer la convergence d'un algorithme de Séparation et Évaluation Progressive basé sur un programme linéaire relaxé.

1.3.2 Inégalité valide et coupe valide

Ici, on rappelle les définitions de base d'inégalité valide et de coupe valide. Le but est de déterminer une coupe valide associée un polyèdre P donné.

- Une inégalité : $\alpha x \leq \beta$ est dite valide pour le polytope P, si et seulement si, $P \cap \{x : \alpha x \leq \beta\} = P$.
- Une inégalité : $\alpha x \leq \beta$ est dite coupe pour le polytope P, si et seulement si, $P \cap \{x : \alpha x \leq \beta\} \subset P$ (inclusion stricte).
- Une inégalité : $\alpha x \leq \beta$ est dite coupe valide pour le polytope P si et seulement si, $\text{conv}(X) \subseteq P \cap \{x : \alpha x \leq \beta\} \subset P$.

Remarque : Dans la littérature, il existe plusieurs types de coupe valide, on cite, entre autre, les inégalités valides de Chvatal-Gomory, la coupe valide de recouvrement, … etc. Pour le plus ample d'informations sur ce sujet, nous invitons le lecteur à consulter les ouvrages spécialisés dans la matière ([1], [5],[9], [14]).

1.3.3 Exemple (Stable de taille maximum)

Le problème du stable de taille maximum ([1], [9], [14][1]), défini sur le graphe G peut se modéliser par le programme linéaire en nombres binaires suivant.

$$(S) \equiv \begin{cases} (\max) z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \\ \begin{pmatrix} 2 & 1 & & & & 1 \\ 1 & 3 & 1 & & & 1 \\ & 1 & 3 & 1 & 1 & \\ & & 1 & 1 & & \\ & & 1 & & 2 & 1 \\ 1 & 1 & & & 1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} \leq \begin{pmatrix} 2 \\ 3 \\ 3 \\ 1 \\ 2 \\ 3 \end{pmatrix} \\ x \in \{0;1\}^6 \end{cases}$$

La solution optimale $\hat{x}$ de (S) est le vecteur représentatif d'un stable de taille maximum. La valeur optimale $z(\hat{x})$ est la taille du stable obtenu. Ce paramètre est bien connu en théorie des graphes sous l'appellation « nombre de stabilité $\alpha(G)$ ».

En résolvant le problème relaxé de (S), on trouve :

$$\hat{x}_{RL} = \left(\tfrac{3}{5} \quad \tfrac{4}{5} \quad 0 \quad 1 \quad 1 \quad 0\right) \text{ et } z(\hat{x}_{RL}) = 3 + \tfrac{2}{5}$$

Inégalités valides : Le graphe G comporte, (voir Figure 1.1) .

- Une clique maximale d'ordre 3, alors : $x_1 + x_2 + x_6 \leq 1$ est une inégalité valide. On peut écrire : $z(\hat{x}) \leq 4$.
- Un cycle impair d'ordre 5, alors : $x_1 + x_2 + x_3 + x_5 + x_6 \leq 2$ est une inégalité valide. On peut écrire : $z(\hat{x}) \leq 3$.

Ces deux inégalités, que nous venons de présenter, sont des coupes valides pour le domaine relaxé du problème (S).

Si, on ajoute la contrainte : $x_1 + x_2 + x_6 \leq 1$ à (S), la solution du problème relaxé devient : $\hat{x}_{RL'} = \left(0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0\right)$ et $z(\hat{x}_{RL'}) = 3$, qui est optimale.

1.4 Conclusion

Dans ce chapitre, nous avons présenté quelques définitions sur les graphes simples non orientés en particulier les graphes bipartis. Les principaux concepts de la Programmation Linéaire en Nombres Entiers (PLNE) ont été rappelés, ainsi que leur développement mathématique qui va nous être utile tout au long de ce mémoire.

2
Modélisation mathématique

2.1 Introduction

Le problème du sac-à-dos à contraintes disjonctives ou sac-à-dos disjonctif (DCKP) est considéré comme une extension directe du problème du sac-a-dos classique (KP), sauf qu'un objet peut ne pas être compatible avec d'autres d'objets du sac. Résoudre un problème du sac à dos disjonctif revient à trouver un sous ensemble d'objets maximisant le profit total tout en respectant la capacité du sac et que tous les objets soient compatibles.

Dans ce chapitre, nous exposons d'abord la formulation mathématique du problème du sac-à-dos classique, ensuite nous présentons le programme linéaire en nombres bivalents (0/1) modélisant le problème du sac-à-dos à contraintes disjonctives ainsi que sa complexité.

2.2 Problème du sac-à-dos unidimensionnel

2.2.1 Position du problème

Le « problème du sac-à-dos » (KP) est un problème de sélection qui consiste à maximiser un critère de qualité sous une contrainte linéaire de capacité de ressource. Il doit son nom à l'analogie qui peut être faite avec le problème qui se pose au randonneur au moment de remplir son sac-à-dos : il lui faut choisir les objets à emporter de façon à avoir un sac le plus « utile » possible, tout en respectant son volume.

2.2.2 Modèle mathématique

Plus formellement, on peut le décrire de la façon suivante. Soit un ensemble de n éléments et une ressource disponible en quantité limitée b. Pour $j = 1$ à n, on note p_j le profit associé à la sélection de l'élément j et on note a_j la quantité de ressource que nécessite l'élément j, s'il est sélectionné. Les coefficients p_j et a_j prennent des valeurs positives pour tout $j = 1$ à n. Le problème du sac-à-dos consiste à choisir un sous-ensemble des n éléments qui maximise le profit total obtenu, en respectant la quantité de ressource disponible. On associe à chaque élément j une variable de sélection, x_j binaire, égale à 1 si j est sélectionné, égale à 0 sinon. Le profit total obtenu peut alors s'écrire comme la somme : $\sum_{j=1}^{n} p_j x_j$et la quantité totale de ressource utilisée comme la somme : $\sum_{j=1}^{n} a_j x_j$. Le problème du sac-à-dos se modélise donc sous la forme :

$$
\text{(KP)} \quad \left| \begin{array}{ll}
\sum_{j=1}^{n} p_j x_j = z(\max) & (2.1) \\
\sum_{j=1}^{n} a_j x_j \leq b & (2.2) \\
x_j \in \{0\,,1\} \qquad (j = 1, \ldots, n) & (2.3)
\end{array} \right.
$$

Il s'agit d'un programme linéaire en nombres binaires.

2.3 Problème du sac-à-dos à contraintes disjonctives

2.3.1 Position du problème

Le problème du sac-à-dos à contraintes disjonctives (DCKP) est une variante ou une extension directe du problème du sac-a-dos classique, sauf qu'un objet peut ne pas être compatible avec d'autres d'objets du sac. Le DCKP contient deux types de contraintes :

i. La contrainte classique de capacité.

ii. Les contraintes disjonctives traduisant la non-compatibilité entre certains objets. Deux objets sont incompatibles s'ils ne doivent pas participer à la même solution ou sélection.

2.3.2 Description des contraintes

Une instance du sac-à-dos à contraintes disjonctives est définie par un ensemble de n objets et par une capacité b du sac-à-dos. On associe à chaque objet j, $(j = 1, \ldots, n)$ un profit p_j et un poids a_j : Certains couples d'objets (i, j) sont incompatibles. Soit x_j la variable de décision associée à l'objet d'indice j: La variable x_j prend la valeur 1 si l'objet j est sélectionné dans la solution et 0 sinon. Le formalisme de ce type de contraintes :

$$x_i + x_j \leq 1 \qquad (i, j) \in E \tag{2.4}$$

ou, l'ensemble E est définit par $E \coloneqq \{(i, j) \text{ sont incompatibles}\}$.

Le deuxième type de contrainte dans le DCKP peut être simplement formulé par la seule contrainte de capacité donnée par :

$$\sum_{j=1}^{n} a_j x_j \leq b \tag{2.5}$$

2.3.3 Modèle mathématique

Une solution du sac-a-dos à contraintes disjonctives est un sous-ensemble ou une sélection d'objets qui vérifie la contrainte de capacité et toutes les contraintes disjonctives. Cette solution est optimale si le profit cumulé de la sélection est de valeur maximale. Le programme linéaire suivant, décrit le sac-à-dos à contraintes disjonctives :

$$\text{(DCKP)} \quad \left| \begin{array}{lll} \sum_{j=1}^{n} p_j x_j = z(\max) & & (2.6) \\ \sum_{j=1}^{n} a_j x_j \leq b & & (2.7) \\ x_i + x_j \leq 1 & (i, j) \in E & (2.8) \\ x_j \in \{0\,, 1\} & (j = 1, \ldots, n) & (2.9) \end{array} \right.$$

Il s'agit d'un programme linéaire en nombres binaires. Dans le chapitre 4, nous allons voir comment nous pouvons le résoudre.

2.3.4 Complexité du problème

Le problème du sac-à-dos à contraintes disjonctives est un problème NP-difficile. En effet, si on supprime les contraintes disjonctives, le DCKP devient le problème du sac-à-dos unidimensionnel. Ce dernier est lui même NP-difficile [14].

2.4 Conclusion

Dans ce chapitre, nous avons exposé d'abord la formulation mathématique du problème du sac-à-dos classique (KP), ensuite nous avons présenté le programme linéaire en nombres bivalents (0/1) modélisant le problème du sac-à-dos à contraintes disjonctives (DCKP). La complexité de ce dernier est abordée de manière concise, et l'algorithme proposé pour le résoudre fait appel à une méthode approchée que nous allons voir dans le chapitre 3.

3
Méthodes de résolution

3.1 Introduction

Dans ce chapitre, nous rappelons brièvement la définition d'un problème d'optimisation combinatoire, Les définitions et les principes généraux des méthodes de résolution de ce dernier, sont cités. Nous présentons aussi, un pseudo algorithme de Recherche Locale typique (RL).

3.2 Problème d'optimisation combinatoire

De nombreux problèmes peuvent être modélises sous la forme d'un problème d'optimisation combinatoire :

$$\begin{aligned} &\min f(s) \\ &s \in S \end{aligned} \tag{3.1}$$

où, S est l'espace de solutions supposé fini et de grande taille, et f est une fonction continue dans S.

La résolution de ce problème peut intervenir dans divers contextes, et les méthodes de résolution destinées à les résoudre doivent fournir des solutions de bonne qualité en un temps raisonnable.

3.3 Processus de résolution d'un problème réel

La résolution d'un problème concret passe par plusieurs étapes. Ces dernières peuvent être schématisées par le diagramme représenté dans la figure 3.1.

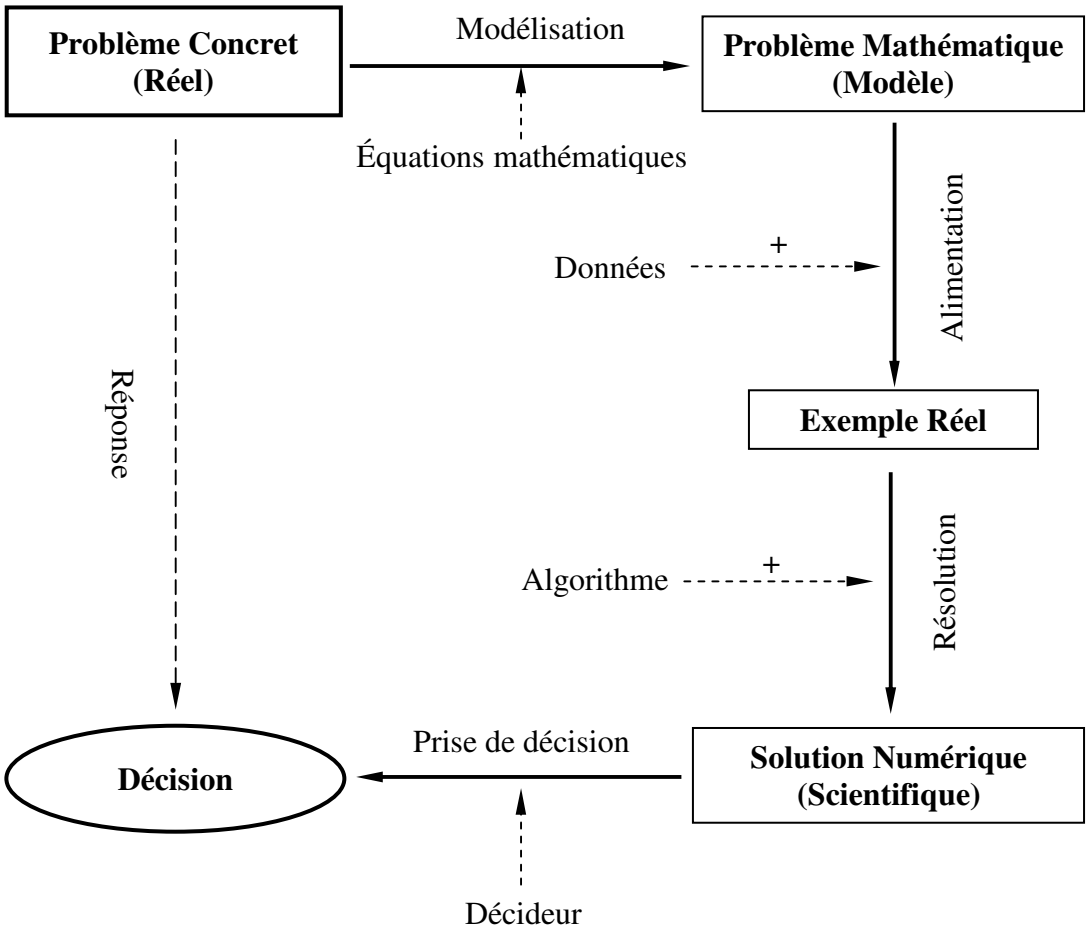


Figure 3.1 : Schéma conceptuel de résolution d'un problème réel.

3.4 Méthodes de résolution

Certains problèmes réels peuvent être modélisés sous la forme de programmes linéaires en nombres entiers (PLNE), de manière à ce que cette modélisation soit la plus proche possible d'un modèle théorique de la littérature pour lequel différentes approches de résolution efficaces ont été proposées. L'étude de ce genre de modèles théoriques est d'une grande importance pour les décideurs dans la mesure où elle met à leur disposition un ensemble de méthodes efficaces, capables de résoudre leurs problèmes particuliers.

3.4.1 Méthodes exactes

Les méthodes de résolution capables de générer une solution optimale sont appelées méthodes exactes. Elles consistent en général à énumérer l'ensemble des solutions de l'espace de recherche de manière implicite. Cependant, elles ne permettent de résoudre que des problèmes de petites tailles, puisque leur temps d'exécution augmente exponentiellement avec la taille du problème. En général, les difficultés liées à la complexité de ce genre de méthodes rendent leur application difficile, voire impossible.

3.4.2 Méthodes approchées

Les méthodes de résolution approchées permettent de trouver rapidement des solutions réalisables du problème traite, mais sans aucune garantie sur la qualité de celle-ci. Parmi ces méthodes on peut citer les heuristiques, les méthodes gloutonnes et les méta-heuristiques représentées essentiellement par les méthodes de voisinage telles que la recherche tabou, le recuit simulé et les algorithmes évolutifs (les algorithmes génétiques et les recherches dispersées). Il existe aussi d'autres méthodes de résolution appelées méthodes hybrides, ces méthodes combinent des méthodes approches avec des méthodes exactes pour tirer profit des avantages de chaque méthode et améliorer le comportement global de l'algorithme de résolution des problèmes d'optimisation combinatoire.

Dans le jargon de la Recherche Opérationnelle (RO) et de l'informatique, les définitions suivantes sont adoptées :

Définition 3.1 Une heuristique est une méthode de calcul qui fournit rapidement une solution réalisable, pas nécessairement optimale ou exacte, pour un problème d'optimisation difficile.

Définition 3.2 (Algorithme Glouton) Un algorithme glouton est un algorithme qui suit le principe de faire, étape par étape, un choix optimum local. Dans certains cas cette approche permet d'arriver à un optimum global, mais dans le cas général ce n'est qu'une heuristique.

Définition 3.3 Une Méta-heuristique est une méthode d'optimisation universelle visant à résoudre des problèmes NP-difficiles pour lesquels on ne connaît pas de méthode classique plus efficace.

Dans la littérature, il existe un grand nombre de Méta-heuristiques différentes, on cite :

Méthode à solution unique	Méthode à population de solutions
Recherche de décente	Algorithme génétique
Recherche tabou	Algorithme de fourmis
Recuit simulé	Algorithme d'optimisation par Essaim Particulaire

3.5 Principe de la méthode de Recherche Locale

La recherche locale consiste à passer d'une solution à une autre solution proche dans l'espace des solutions candidates (l'espace de recherche) jusqu'à ce qu'une solution considérée comme optimale soit à trouvée, ou que le temps imparti soit dépassé, comme l'indique la figure 3.2 .

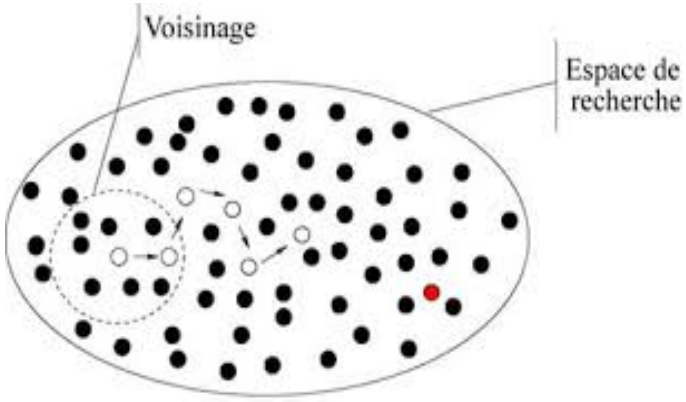


Figure 3.2 : Principe général de la méthode de Recherche Locale (RL).

3.6 Algorithme de Recherche Locale (RL)

Dans cette section, nous présentons un pseudo algorithme de la méthode de Recherche Locale pour la résolution du problème d'optimisation (3.1).

Algorithme 3.1 : Algorithme de recherche locale typique.

```
1   s* = créer_une_solution()
2   for all t = 1 à max_essais do
3   s = nouvelle_solution()
4           for all m = 1 à max_mouvements do
5           s'= choisir_voisin(s)
6           d = f(s') - f(s)
7           if d < 0 then s = s' end if
8           end for
9   if f(s*) > f(s) then s* = s end if
10  end for
11  retourner s*, f(s*)
```

3.7 Conclusion

Dans ce chapitre, nous avons rappelé brièvement la définition d'un problème d'optimisation combinatoire, Les définitions et les principes généraux des méthodes de résolution de ce dernier, ont été cités. Nous avons présenté aussi, un pseudo algorithme de Recherche Locale typique (RL).

4

Implémentation et tests numériques

4.1 Introduction

Dans ce chapitre, nous présentons l'implémentation de notre approche de résolution. Nous reportons aussi des résultats numériques obtenus en testant notre programme Matlab sur un exemple réel du problème, en utilisant l'algorithme vu dans le chapitre précédent.

4.2 Adaptation et mise en œuvre

Dans cette section, nous présentons quelques prétraitements nécessaires avant d'appliquer l'algorithme de Recherche Locale (RL), pour résoudre un exemple du problème.

4.2.1 Reformulation du problème

Le problème du sac-à-dos à contraintes disjonctives (DCKP) peut être reformulé par le modèle suivant :

$$
\text{(DCKP1)} \quad
\begin{cases}
\sum_{j=1}^{n} p_j x_j = z(\max) & & (4.1)\\
\sum_{j=1}^{n} a_j x_j \leq b & & (4.2)\\
x_i x_j = 0 & (i,j) \in E & (4.3)\\
x_j \in \{0\,,1\} & (j = 1,\ldots,n) & (4.4)
\end{cases}
$$

Il s'agit d'un programme quadratique en nombres binaires. Cependant le groupe de contraintes (4.3) peuvent être agrégées et remplacées par une seule contrainte (4.7), ce qui nous donne au final le modèle mathématique suivant :

$$
\text{(DCKP2)} \quad
\begin{cases}
\sum_{j=1}^{n} p_j x_j = z(\max) & & (4.5)\\
\sum_{j=1}^{n} a_j x_j \leq b & & (4.6)\\
\sum_{(i,j)\in E} x_i\, x_j = 0 & \text{Une seule contrainte} & (4.7)\\
x_j \in \{0\,,1\} & (j = 1, \ldots, n) & (4.8)
\end{cases}
$$

où, la contrainte (4.7) désigne le nombre d'objets incompatibles conflits.

4.2.2 Fonction de pénalité

Nous relaxons les deux contraintes (4.6) et (4.7) et nous les pénalisons dans la fonction objective, alors la fonction de pénalité s'écrit :

$$f(x) = \sum_{j=1}^{n} p_j x_j - r \max\left(\sum_{j=1}^{n} a_j x_j - b\,, 0\right) - r \sum_{(i,j)\in E} x_i\, x_j$$

où, r désigne le nombre positif assez grand.

4.2.3 Réduction de l'espace de recherche

Le choix de l'espace de recherche S est crucial, il doit être facile à explorer et avoir la plus petite taille possible s'approchant ainsi de l'ensemble de faisabilité. Dans notre cas, l'espace de recherche est défini par :

$$S = \{x \in \mathbb{B}^n \,:\; x_1 + x_2 + \cdots + x_n = k\}$$

où, k désigne le nombre maximal d'objets à emporter dans le sac. Il est obtenu en appliquant la technique des coupes planes, vue dans le chapitre 1. La taille de S est $|S| = C_n^k$.

Le problème consiste à chercher une solution x de S, de telle sorte que la valeur de la fonction de pénalité $f(x)$ soit maximale.

$$(\max) f(x)\,, x \in S \qquad (4.9)$$

4.3 Exemple Test

Dans cette section, nous considérons un exemple à $n = 15$ objets avec un graphe de conflits $G = (\{1, \ldots, n\}; E)$. Une instance du problème de sac-à-dos à contraintes disjonctives se caractérise par le triplé (p, a, b, E). Dans le tableau (4.1), on donne le profit et le poids de chaque objet. La capacité du sac est $b = 48$. Dans le tableau (4.2), nous indiquons les objets incompatibles qui ne partagent pas le même sac.

Tableau 4.1 : Profit et Poids de chaque objet.

Objets	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Profit	17	18	23	15	11	13	19	21	14	8	16	15	9	12	11
Poids	8	6	5	6	7	5	9	8	7	4	5	9	4	6	7

Tableau 4.2 : Relation d'incompatibilité entre les objets.

Arête : e	**Extrémité initiale : $x = I(e)$**	**Extrémité terminale : $y = T(e)$**
1	1	3
2	1	11
3	3	11
4	4	12
5	4	15
6	5	6
7	5	11
8	5	12
9	6	9
10	7	13
11	10	12
12	10	13
13	10	15
14	13	15
15	14	15

4.4 Résultats obtenus

Dans cette section, nous présentons les résultats numériques que nous avons obtenus sur notre exemple. À l'issue de l'algorithme (3.1) de Recherche Locale, les résultats obtenus sont reportés dans le tableau suivant :

Tableau 4.3 : Meilleure sélection des objets compatibles.

Objets j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\hat{x}_j$	0	1	0	1	0	1	1	1	0	0	1	0	0	1	0

Le profit total optimal est $f(\hat{x}) = p'\hat{x} = 114$. Le nombre d'objets emportés dans le sac est $k = 7$. Ces résultats sont obtenus après 4 cycles de recherche (4 voisinages) seulement. La figure 4.1 illustre l'évolution de $f(x)$ en fonction de la solution intermédière x.

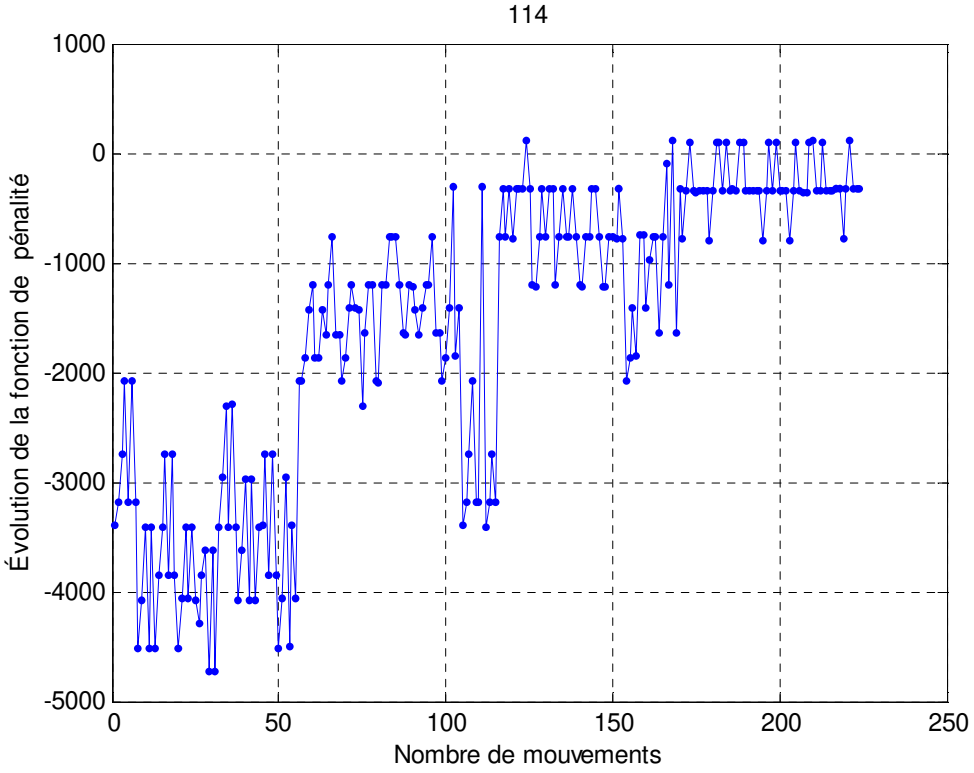


Figure 4.1 : Comportement de l'algorithme sur l'exemple test.

4.5 Conclusion

Dans ce chapitre, nous avons présenté l'implémentation de notre approche de résolution. Nous avons reporté aussi des résultats numériques obtenus en testant notre code de calcul Matlab sur un exemple réel du problème.

Durant notre étude, nous avons constaté que de nombreux tests numériques effectués, ont permis d'avérer les avantages de l'utilisation de l'algorithme de Recherche Locale (RL) pour la résolution de notre exemple. Les résultats numériques obtenus affirment l'efficacité de cette approche.

Conclusion

L'optimisation combinatoire occupe une place très importante dans la vie quotidienne. Son importance se justifie d'une part par la grande difficulté des problèmes d'optimisation et d'autre part par de nombreuses applications pratiques pouvant être formulées sous la forme d'un problème d'optimisation combinatoire. Les problèmes d'optimisation sont utilisés pour modéliser de nombreux problèmes dans différents secteurs de l'industrie telle que la télécommunication, la gestion,…., etc.

Dans ce mémoire, nous nous sommes intéressés au problème du sac-à-dos à contraintes disjonctives (ou, sac-à-dos disjonctif). Il est considéré comme une variante du problème du sac-a-dos classique, sauf qu'un objet peut ne pas être compatible avec d'autres d'objets du sac. Résoudre un problème du sac à dos disjonctif revient à trouver un sous ensemble d'objets maximisant le profit total tout en respectant la capacité du sac et que tous les objets soient compatibles

Ce problème d'optimisation est formulé comme un programme linéaire en nombres binaires. Pour le résoudre, nous avons appliqué une méthode approchée de type Recherche Locale (RL).

La démarche scientifique que nous avons adoptée, se résume comme suit : En premier lieu, nous avons posé le problème du sac-à-dos à contraintes disjonctives, ainsi qu'une formulation mathématique modélisant ce problème a été présentée. Ensuite, nous avons proposé une méthode approchée de type Recherche Locale pour la résolution du problème. Au final, nous avons élaboré et implémenté un programme informatique codé en langage de MATLAB pour la résolution d'une instance du problème. Les résultats numériques obtenus ont été apportés et commentés.

Durant notre étude, nous avons constaté que de nombreux tests numériques effectués pour la résolution des exemples du problème, ont permis d'avérer les avantages de l'utilisation de la méthode de recherche locale (RL). Les résultats obtenus affirment l'efficacité de cette approche.

Nous espérons que notre but est atteint, et le travail que nous avons réalisé dans ce mémoire, nous a permis d'approfondir nos connaissances théoriques et pratiques dans le domaine de l'optimisation numérique et d'acquérir des connaissances dans le domaine de la gestion et de l'informatique.

A. Codes MATLAB

A.1 Fichier des données (paramètres)

Ce fichier est conçu pour regrouper les données caractérisant un exemple du problème.

Algorithme A.1 : Données pour la création de l'exemple.

```
1   function [c, a, b, A] = exemple1
2   c = [17 18  23 15 11 13 19 21 14  8 16 15  9 12 11];
3   a = [ 8  6  5   6  7  5  9  8  7  4  5  9  4  6  7];
4   b = fix(0.5*sum(a));
5   %-----------------------------------
6   rand('state',130);
7   n = length(a);
8   p = 0.15;
9   A = tril(rand(n)< p,-1);
10  %[I,J] = find(A == 1); E = [I, J];
11  A = A + A.';
```

A.2 Fichier de génération de la solution de départ

Ce fichier sert à générer la solution de départ de l'algorithme (RL).

Algorithme A.2 : Génération de la solution de départ.

```
1   function x = genererx0(k,c,a)
2    n = length(c);
3    x = zeros(1,n);
4    [c,p] = sort(a./c);
5    x(p(1:k)) = 1; function x = genererx0(k,c,a)
6    n = length(c);
7    x = zeros(1,n);
8    [c,p] = sort(a./c);
9    x(p(1:k)) = 1;
```

A.3 Script de résolution et affichage des résultats

Ce fichier est conçu pour l'exécution de l'algorithme de recherche locale (RL), en affichant des résultats numériques relatifs à notre exemple.

Algorithme A.3 : Résolution de l'exemple.

```
1  function [xbest, fbest] = resoudre()
2  itermax = 20;
3  %---------------------------------------
4  [c, a, b ,A] = exemple1;
5  %---------------------------------------
6  [k, n, p, q] = nombrek(a,b,A);
7  %---------------------------------------
8  y = genererx0(k,c,a)
9  %---------------------------------------
10 r = sum(c);
11 fy = fun(y,c,a,b,A,r);
12 fbest = fy; fx = fy; xbest = y; x = y;  L = [];
13 for iter = 1:itermax
14 R = [fy];
15 I = find(y > 0); J = find(y < 1);
16 for i = I
17     x(i) = 0;
18     for j = J
19     x(j) = 1;
20     fx = fun(x,c,a,b,A,r);
21     if (fx > fy), fy = fx; y = x; end
22     R = [R, fx];
23     x(j) = 0;
24     end
25     x(i) = 1;
26 end
27 L = [L, R];
28 if  (fy > fbest), fbest = fy; xbest = y; else break;end
29
30 end
31
32
33 w = length(L);
34 plot(1:w,L,'.-'); title(num2str(fbest));
35 xlabel('Nombre de mouvements');
36 ylabel('Évolution de la fonction de  pénalité');
37 grid
38
39 %-----------------------------------------------------
40 function z = fun(x,c,a,b,A,r)
41 z = dot(c,x) - r*max(dot(a,x) - b,0) - r*dot(x,x*A);
42 %-----------------------------------------------------
```

A.4 Fichier de calcul du paramètre *k*

Ce fichiclcer sert à calculer le paramètre *k*, qui représente le nombre maximal d'objets compatibles, à emporter dans le sac-à-dos.

Algorithme A.4 : Calcul du paramètre *k*.

```
1  function [k,n,p,q] = nombrek(a,b,A)
2  n = length(a);
3  [a,p] = sort(a); s = cumsum(a);
4  k1 = n - sum(s > b);
5  %-----------------------------------
6  d = sum(A); cardE = sum(d)/2;
7  [d,q] = sort(d); s = cumsum(d);
8  k2 = n - sum(s > cardE);
9  k = min(k1,k2);
```

Notations et abréviations

$\mathbb{Z}$	L'ensemble des nombres entiers.
$\{0,1\}^n$	L'ensemble produit cartésien : $\underbrace{\{0,1\} \times \{0,1\} \times \cdots \times \{0,1\}}_{n\ fois}$.
$\lvert A \rvert$	Le nombre d'éléments de l'ensemble A.
$V = \{v_1, v_2, \ldots, v_n\}$	Un ensemble de sommets.
$E = \{\mathrm{e}_1, \mathrm{e}_2, \ldots, \mathrm{e}_m\}$	Un ensemble d'arêtes.
$\Gamma(v)$	L'ensemble de voisins du sommet v.
$d(v) = \lvert\Gamma(v)\rvert$	Le degré du sommet v.
$E(V)$	L'ensemble d'arêtes induites par E sur l'ensemble V.
$\mathrm{e} = (u, v)$	Une arête. Les sommets u et v sont les extrémités de l'arête e.
$G = (V, E)$	Un graphe simple non orienté.
$x = (x_1, x_2, \ldots, x_n)$	le vecteur représentatif d'un ensemble de taille n.
$\sum_{i=1}^{n} x_i$	La somme : $x_1 + x_2 + \cdots + x_n$.
x^t	Le vecteur transposé de x.
$x^t y = \sum_{i=1}^{n} x_i y_i$	Le produit scalaire du vecteur x et y.
$Z(\max)\,;\ Z(\min)$	Maximiser la valeur Z ; Minimiser la valeur Z.

Références

[1] V. Chvatal, Linear Programming. Freeman and Co, 1978.

[2] G. Cohen, Convexité et Optimisation. École Nationale des Ponts et Chaussées et INRIA, 2000.

[3] G. Cohen, Optimisation de grands systèmes. Cours de DEA. Cermics-École nationale des Ponts et Chaussées et INRIA, 2003.

[4] J.C. Culioli, Introduction à l'optimisation, École Nationale Supérieure des Mines de paris, Centre Automatique et Système, Fontainebleau, Ellipse, 1994.

[5] G.B. Dantzing, Linear Programming and Extensions. Princeton. University Press, 1964.

[6] Y. Dodge, S. Gonano-Weber et J.P. Renfer, Optimisation Appliquée, Springer-Verlag France, 2005.

[7] D. Dubois, Modèles mathématiques de l'imprécis et de l'incertain en vue d'application aux techniques d'aide à la décision. Thèse soutenue à l'Université de Grenoble I, 1983.

[8] R. Fourer, D. M. Gay and B. W. Kernigham : A Modeling language for Mathematical Programming. Second Edition, Thomson, USA, 2003.

[9] M. Gondran et M. Minoux, Graphes at algorithms. Eyrolles, 1979.

[10] G. Laporte, Y. Nobert Y and S. Taillefer. Solving a family of multi-depot vehicle routing and location-routing problems. Transportation Science, 22 : 172. 1988.

[11] M. Minoux, Programmation mathématique, Théorie et algorithmes, Collection Technique et Scientifique des Télécommunications, Dunod, Paris, 1983.

[12] J. Nocedal and S. J. Wright, Numerical Optimization, Springer-Verlag, New York, 1999.

[13] M. Sakarovitch, Graphes et Programmation linéaire : Méthodes Mathématiques et Algorithmiques, Programmation Discrète, Hermann, Paris, 1984.

[14] M. Sakarovitch, Optimisation Combinatoire : Méthodes Mathématiques et Algorithmiques, Programmation Discrète, Hermann, Paris, 1984.

[15] P. Toth, D. Vigo, The vehicle routing problem. SIAM, Philadelphia Monograph on Discrete Mathematics and Applications, 2000.