



## RAPPORT PFA

# **Modélisation et Résolution avec CPLEX du problème du Sac à dos**

par

*BELAIZI Nassima et HAYAOUI Abdelhakim*

**Professeur encadrant :**

Mr. BENADADA Youssef



# Remerciements

Au terme de ce travail, nous profitons de l'occasion pour remercier du fond du Cœur tous ceux qui ont participé de près ou de loin à la réalisation de ce projet et qui ont contribué à en faire une expérience enrichissante, nous adressons donc, en particulier nos sincères remerciements à notre encadrant Mr Youssef BENADADA pour sa disponibilité et ses précieux conseils.

Nos remerciements vont aussi à Mme Fatima Zahra Mhada pour avoir jugé notre travail.

Enfin nous tenons à remercier l'ensemble du corps enseignant de l'Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes. En espérant que ce travail sera à la hauteur de vos attentes.

Rabat, 28 Mai 2020

---

BELAIZI Nassima et HAYAOUI Abdelhakim

## **Résumé**

Le présent document est le fruit de notre travail dans le cadre de projet de fin de 1ère année. Ce projet avait comme but une étude générale du problème du sac à dos, en plus d'une modélisation et résolution avec Cplex du "Problème du sac à dos" et l'implémentation d'une interface graphique pour faciliter la résolution du problème pour les utilisateurs. Dans le premier chapitre on va introduire le sujet, expliquer la problématique et le but générale du projet. Le deuxième chapitre sera dédié au problème de sac à dos et ses méthodes de résolution avec ses différentes variantes. Le troisième chapitre sera réservé à la résolution avec Cplex et la mise en œuvre d'une interface graphique conviviale.

## **Abstract**

This document is a result of our work as part of the end of 1st year project. The goal of this project is modelling and solving the famous Knapsack Problem and the implementation of a graphical interface to facilitate the solving for users. In the first chapter we will introduce the subject, explain the problem and the general goal of the project. The second chapter will be dedicated to the problem of Knapsack and its methods of resolution with its different variants. The third chapter will be reserved for solving with Cplex and implementing a user-friendly graphical interface

# Table des matières

<b>1</b>	<b>Introduction générale</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.2	Objectif Générale du projet . . . . .	2
<b>2</b>	<b>Le problème du sac à dos</b>	<b>3</b>
2.1	Définition . . . . .	3
2.2	Domaines d'utilisation . . . . .	4
2.3	Méthodes de résolution . . . . .	5
2.3.1	Méthode approchée (Algorithme glouton) . . . . .	5
2.3.2	Méthodes exactes . . . . .	6
2.3.3	Résolution à l'aide d'EXCEL . . . . .	8
2.4	Les variantes du problème du sac à dos . . . . .	10
2.4.1	Sac à dos multiple : . . . . .	10
2.4.2	Sac à dos quadratique . . . . .	11
2.5	Conclusion . . . . .	13
<b>3</b>	<b>Résolution de KP avec Cplex</b>	<b>14</b>
3.1	Introduction . . . . .	14
3.2	Résolution du KP . . . . .	15
3.2.1	1Programme de résolution avec Cplex . . . . .	15
3.2.2	exemples d'exécution . . . . .	16
3.3	Interface graphique convivial . . . . .	18
3.3.1	Introduction . . . . .	18
3.3.2	La mise en œuvre . . . . .	19
3.4	Conclusion . . . . .	25
<b>4</b>	<b>Conclusion générale</b>	<b>26</b>
<b>5</b>	<b>Références bibliographiques</b>	<b>27</b>

# Table des figures

2.1	Probleme de Sac a dos . . . . .	3
2.2	Forme mathématique . . . . .	4
2.3	Tableau des objets . . . . .	5
2.4	La procédure par séparation et évaluation . . . . .	6
2.5	arbre de recherche . . . . .	7
2.6	Tableau Excel . . . . .	8
2.7	Le complément solveur Excel . . . . .	9
2.8	Le résultat du complément solveur Excel . . . . .	9
2.9	Le résultat du complément solveur Excel . . . . .	10
2.10	Sac à dos quadratique . . . . .	11
3.1	Logo de IBM CPLEX . . . . .	14
3.2	Le fichier .mod (model) . . . . .	16
3.3	Le fichier .dat(data) . . . . .	16
3.4	Tableau de données . . . . .	17
3.5	Le résultat de 7 échantillons . . . . .	17
3.6	La solution est optimale . . . . .	18
3.7	La solution est optimale avec tolérance . . . . .	18
3.8	Logo de JAVA . . . . .	19
3.9	Logo de JAVA FX . . . . .	20
3.10	Tableau de données avec noms . . . . .	21
3.11	Etape 1 . . . . .	21
3.12	Le fichier .dat générée après l'étape 1 . . . . .	22
3.13	La méthode java DataFileModify . . . . .	22
3.14	Etape 2 avant le remplissage du tableau . . . . .	23
3.15	Etape 2 après le remplissage du tableau . . . . .	23
3.16	La méthode JAVA append dat file . . . . .	24
3.17	La méthode JAVA ExportCplexFiles . . . . .	24
3.18	Etape 3 . . . . .	25
3.19	La méthode JAVA importLPFile . . . . .	25

# **1 | Introduction générale**

## **1.1 Introduction**

Le problème du sac à dos a été largement étudié en raison de ses vastes applications dans de nombreux domaines scientifiques. C'est l'un des problèmes classiques d'optimisation combinatoire qui trouve de nombreuses applications dans le domaine informatique. À partir d'un ensemble donné d'articles (chacun avec un poids et une valeur), l'objectif est de trouver le nombre d'articles qui peuvent être inclus afin que le poids total soit inférieur ou égal à une limite donnée et que la valeur totale soit maximale.

## **1.2 Objectif Générale du projet**

Le projet consiste à résoudre le problème de sac à dos en utilisant " IBM ILOG CPLEX Optimization Studio " pour la version simple du problème et développer une version simplifiée qu'un utilisateur qui peut l'utiliser sans avoirs des prérequis en CPLEX .



## 2 | Le problème du sac à dos

### 2.1 Définition

Le problème classique du sac-à-dos (knapsack, ou KP) est un problème d'optimisation combinatoire qui ne contient qu'une seule contrainte sur les objets de la solution. Toutefois, il constitue un défi pour les chercheurs par sa complexité, puisqu'il appartient à la classe des problèmes NP-difficiles. 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 : il lui faut choisir les objets à emporter de façon à avoir un sac le plus utile possible, tout en respectant son volume, la question qui se pose est de savoir quels objets il doit mettre dans le sac. D'une façon générale, le problème du sac-à-dos consiste à remplir un sac dont la capacité est fixée, avec un sous ensemble d'objets de poids et de profit connus, de manière à satisfaire les deux conditions suivantes :

- le poids cumulé du sous-ensemble d'éléments choisi ne dépasse pas la capacité du sac
- le profit généré par le sous-ensemble d'éléments choisi est maximal. Dans ce cas, il faut savoir quels sont les objets qu'on mettra dans le sac pour maximiser ce profit.

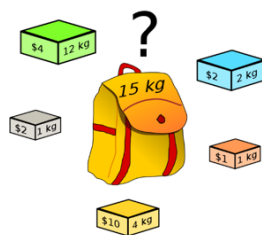


FIGURE 2.1 – Probleme de Sac a dos

L'énoncé de ce problème est simple : étant donné un ensemble de  $n$  objets, où chaque objet  $i$  est caractérisé par un poids  $w_i$  et un profit  $p_i$  on cherche le sous-ensemble d'objets à charger dans un sac de capacité  $c$  afin de maximiser la somme des profits. Ainsi, le problème du sac à dos se présente sous la forme mathématique suivante :

$$(KP) \left\{ \begin{array}{l} \max \sum_{i=1}^n p_i \cdot x_i, \\ \sum_{i=1}^n w_i \cdot x_i \leq c, \\ x_i \in \{0,1\}, i \in \{1, \dots, n\} \end{array} \right.$$

FIGURE 2.2 – Forme mathématique

Les poids  $w_i$  et les profits  $p_i$  ainsi que la capacité  $c$  sont des entiers positifs.

La variable  $x_i$  est la variable de décision; elle prend la valeur 1 si l'objet  $i$  est chargé dans le sac, sinon elle prend la valeur 0.

## 2.2 Domaines d'utilisation

L'intérêt porté au sac à dos est dû au fait qu'il permet de modéliser de très nombreux problèmes comme les problèmes de gestion de capital, de chargement de cargaison, de rotation d'équipage, de tournées et de livraison; par ailleurs, ce problème apparaît comme un sous problème de nombreux problèmes d'optimisation combinatoire.

## 2.3 Méthodes de résolution

Il existe deux grandes catégories de méthodes de résolution de problèmes d'optimisation combinatoire :

Les méthodes exactes et les méthodes approchées. Les méthodes exactes prennent beaucoup de temps si le problème est compliqué mais aboutissent toujours à la solution optimale.

Par contre, Les méthodes approchées, ou bien « heuristiques », ne garantissent pas l'optimalité de la solution, elle est souvent approchée mais elles sont rapides. Ce sont des ajustements entre la qualité de la solution et le temps de calcul.

### 2.3.1 Méthode approchée (Algorithme glouton)

1. Calculer le rapport valeur sur poids ( $v_i/p_i$ ) pour chaque objet  $i$  (efficacité) ;
2. Trier tous les objets par ordre décroissant de cette valeur ;
3. Sélectionner les objets un à un dans l'ordre du tri et ajouter l'objet sélectionné dans le sac si le poids maximal reste respecté.

Pour quatre objets ( $n=4$ ) et un sac à dos d'un poids maximal de 30kg ( $P=30$ ), nous avons par exemple les résultats suivants :

Objets	1	3	2	4
$v_i$	7	3	4	3
$p_i$	13	8	12	10
$v_i / p_i$	0,54	0,37	0,33	0,30

FIGURE 2.3 – Tableau des objets

La solution trouvée est donc de mettre les objets 1 et 3 dans le sac, ce qui donne une valeur de 10. Cette solution n'est pas optimale, puisqu'une solution avec une valeur totale de 11 existe. Néanmoins, cet algorithme reste rapide même si le nombre d'objets augmente considérablement.

Ce type d'algorithme ne discute jamais un choix fait auparavant. On n'essaie pas d'enlever un objet du sac pour le remplacer avec un autre.

### 2.3.2 Méthodes exactes

Pour être certain de l'optimalité de la solution il faut utiliser une des méthodes exactes qui peuvent demander un temps de calcul plus long, chaque problème a sa méthode correspondante à laquelle il est mieux adapté.

- La procédure par séparation et évaluation (PSE) :

La procédure par séparation et évaluation (PSE), ou en anglais branch and bound est un algorithme qui permet d'énumérer toutes les solutions possibles. Plus pratiquement, on procède souvent à éliminer les solutions de mauvaise qualité face aux autres qui sont plus probables à être optimales.

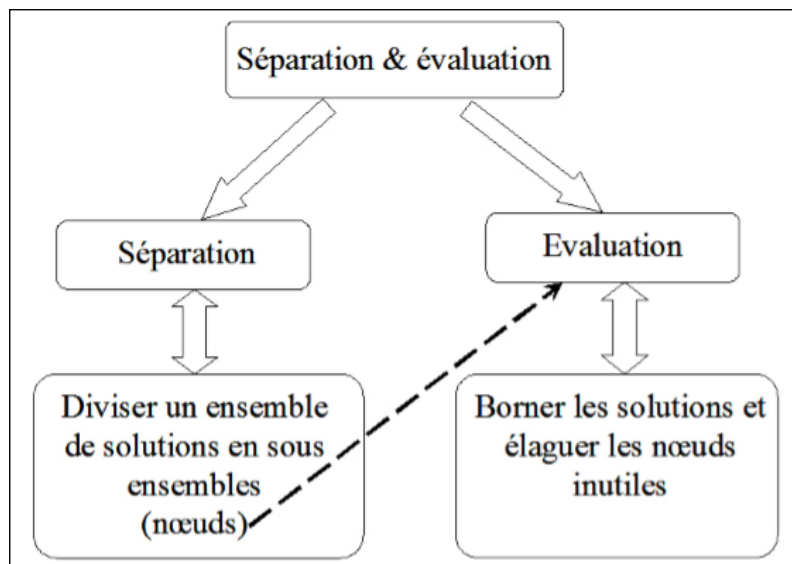


FIGURE 2.4 – La procédure par séparation et évaluation

Pour représenter une PSE, nous utilisons un «arbre de recherche» constitué :

- de nœuds ou sommets, où un nœud représente une étape de construction de la solution (des objets auront été mis dans le sac, d'autres auront été laissés dehors).
- d'arcs pour indiquer certains choix faits pour construire la solution (Chaque arc indique l'action de mettre un objet dans le sac ou, au contraire, de ne pas le mettre dans le sac).

La figure suivante représente un arbre de recherche :

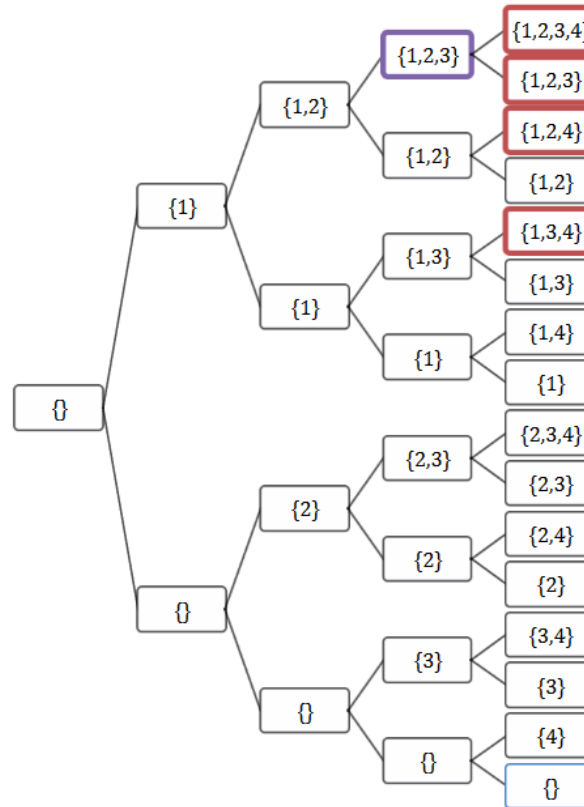


FIGURE 2.5 – arbre de recherche

On associe l'ensemble vide à la racine (dont la profondeur est 0) et pour un nœud de profondeur  $i-1$ , on construit deux fils : celui du haut où l'on ajoute l'objet  $i$  dans le sac et celui du bas où le sac reste tel quel.

Chaque feuille représente alors une solution potentielle mais pas forcément réalisable. Pour déterminer la solution, il suffit de calculer la valeur du sac pour chaque nœud feuille acceptable et de prendre la solution ayant la plus grande valeur.

- Eliminer les cas échéants évidents par les bornes supérieures et inférieures :

Calculer des bornes supérieures ou inférieures permet de stopper l'exploration d'un sous-ensemble de solutions qui ne sont pas candidates à l'optimalité, Ces

bornes sont ensuite, utilisées pour le développement de méthodes de résolution exacte s'appuyant sur des procédures d'énumération.(3)

### 2.3.3 Résolution à l'aide d'EXCEL

- On étudie le cas de 3 objets et d'un sac de capacité 10 kg;
- On commence par insérer les valeurs et les poids des objets ;
- Initialiser les valeurs des variables de décision par 1 ;
- Définir des fonctions qui calculent la somme des valeurs et la somme des poids;

	A	B	C	D	E
1	xi	vi	pi	xi*vi	xi*pi
2	1	2	5	2	5
3	1	3	6	3	6
4	1	5	2	5	2
5				10	13

FIGURE 2.6 – Tableau Excel

On utilise le complément solveur pour résoudre le problème :

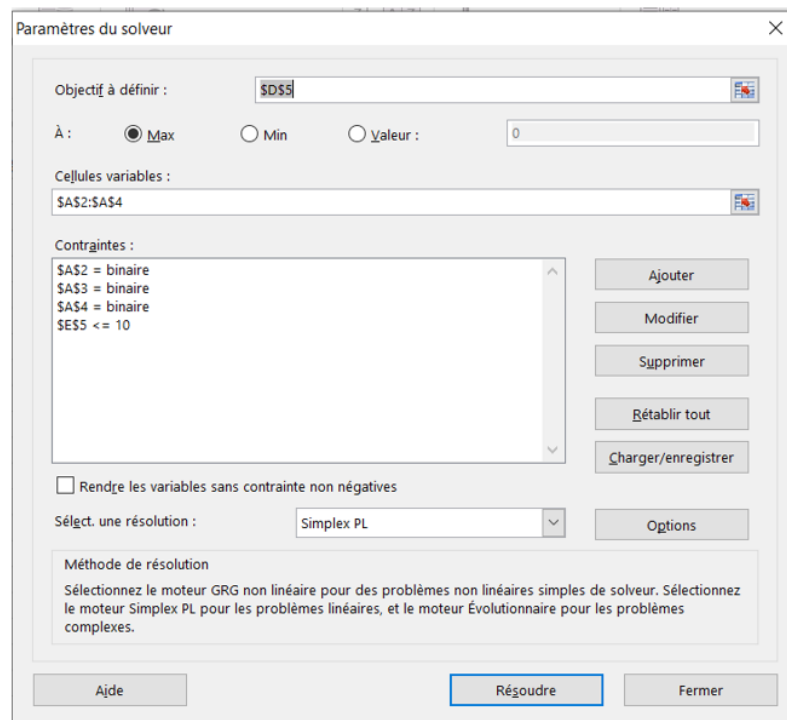


FIGURE 2.7 – Le complément solveur Excel

- Le résultat :

Le solveur exclu l'objet 1 car la somme des poids a dépassé la capacité du sac et garde les deux objets maximisant la fonction objectif tout en respectant le poids maximal.

	A	B	C	D	E
1	xi	vi	pi	xi*vi	xi*pi
2	0	2	5	0	0
3	1	3	6	3	6
4	1	5	2	5	2
5				8	8

FIGURE 2.8 – Le résultat du complément solveur Excel

## 2.4 Les variantes du problème du sac à dos

### 2.4.1 Sac à dos multiple :

#### Généralités

Le problème du sac à dos multiple (MKP) est une généralisation du problème standard du sac à dos en variable 0-1, où on essaie de remplir  $m$  sacs à dos de différentes capacités au lieu de considérer un seul sac à dos.

Soit  $N = \{1, \dots, n\}$  l'ensemble des indices d'articles à charger où chaque article d'indice  $j$  est caractérisé par un profit  $p_j$  et un poids  $w_j$ . Nous considérons  $m$  sacs à dos où chaque sac d'indice  $i$ ,  $i \in \{1, \dots, m\}$  est de capacité de chargement  $c_i$ . Alors le problème du sac à dos multiple consiste à remplir tous les sacs à dos de façon à maximiser le profit total et de sorte que la somme des poids dans chaque sac à dos d'indice  $i$  ne dépasse pas la capacité  $c_i$ .

Nous notons par  $x_{ij}$  la variable binaire de décision qui prend la valeur 1 si l'article d'indice  $j$  est affecté au sac à dos d'indice  $i$  et 0 dans le cas contraire. Le problème du sac à dos multiple (MKP) peut être formulé de la manière suivante :

$$(MKP) \left\{ \begin{array}{l} \max \sum_{i=1}^m \sum_{j=1}^n p_j x_{ij}, \\ s.c. \sum_{j=1}^n w_j x_{ij} \leq c_i, i \in \{1, \dots, m\}, \\ \sum_{i=1}^m x_{ij} \leq 1, j \in \{1, \dots, n\}, \\ x_{ij} \in \{0, 1\}, i \in \{1, \dots, m\}, j \in \{1, \dots, n\}; \end{array} \right.$$

FIGURE 2.9 – Le résultat du complément solveur Excel

où  $p_j$ ,  $c_i$  et  $w_j$  sont des entiers positifs et les contraintes, assurent respectivement, que le remplissage du sac à dos  $i$  ne dépasse pas sa capacité correspondante  $c_i$  et que chaque article sélectionné est attribué, au plus, à un seul sac à dos.(2)



**Heuristique de résolution (Martello et Toth heuristic method) :**

L'heuristique dite MTHM de Martello et pour la résolution du problème du sac à dos multiple se compose de trois phases.

La première phase de MTHM obtient une première solution réalisable en appliquant l'algorithme Glouton au premier sac à dos ; puis la même procédure est appliquée pour le deuxième sac à dos ; ceci est poursuivi jusqu'à m-ième sac à dos.

La deuxième phase tente d'améliorer la solution initiale en échangeant toutes les paires d'articles affectés à différents sacs à dos et en essayant d'insérer un nouvel article afin d'augmenter le bénéfice total.

La dernière phase tente d'exclure tour à tour chaque article sélectionné si il est possible de le remplacer par un ou plusieurs éléments restants afin que la somme totale des bénéfices soit augmentée.

L'avantage de l'heuristique MTHM est que certains éléments peuvent être échangés d'un sac à dos à un autre ou exclus de l'ensemble de solutions afin que le profit total augmente.

Le principal inconvénient de l'heuristique MTHM est qu'elle considère les échanges entre une paire d'éléments au lieu de combinaisons d'éléments.

## 2.4.2 Sac à dos quadratique

### Généralités

Le problème de sac à dos quadratique noté QKP est caractérisé par un gain  $g_{ij}$  supplémentaire lorsque deux objets ( $i$  et  $j$ ) sont pris simultanément.

$$\max \{z(X)\} = \sum_{i=1}^n x_i p_i + \sum_{i=1}^n \sum_{j=1}^n x_i x_j g_{ij}$$

FIGURE 2.10 – Sac à dos quadratique

où  $p_i$  modélise le profit.

### Heuristique de résolution

L'algorithme de force brute pour résoudre ce problème consiste à :

- Identifier tous les sous-ensembles possibles des articles sans dépasser la capacité .
- Sélectionner celui avec la valeur optimale.

Le pseudo-code est fourni comme suit :

```
// Profits (p)
// profits quadratique (Pq)
// poids (w)
// nombre d'éléments (n)
// capacité du sac (W)

max = 0
Pour toute instance S faire :
    valeur, poids = 0
    pour i de 0 à S.taille-1 faire :
        valeur = valeur + p[i]
        poids = poids + w[i]
        pour j de i+1 à S.taille -1 faire :
            valeur = valeur + Pq[i][j]
        si poids <= W alors :
            si valeur > max alors :
                max = valeur
```

L'algorithme Quadknap est plus efficace, c'est un algorithme de branchement et de liaison exact soulevé par Caprara, il a été rapporté pour générer des solutions exactes d'instances avec jusqu'à 400 variables binaires.

## 2.5 Conclusion

Dans ce chapitre, nous avons présenté des généralités à propos du sac à dos. Nous avons entamé notre présentation par le problème du knapsack classique (unitaire) puis nous avons discuté quelques méthodes de résolution de ce type de problèmes puis on a cité quelques variantes de celui-ci, ainsi que quelques domaines d'application.

## 3 | Résolution de KP avec Cplex

### 3.1 Introduction

IBM ILOG CPLEX Optimization Studio (souvent appelé simplement CPLEX de manière informelle) est un progiciel d'optimisation. L'optimiseur CPLEX a été nommé pour la méthode simplexe telle qu'implémentée dans le langage de programmation C, bien qu'aujourd'hui il supporte également d'autres types d'optimisation mathématique et offre des interfaces autres que C. Il a été initialement développé par Robert E. Bixby et vendu dans le commerce à partir de 1988 par CPLEX Optimization Inc. Il a été acquis par ILOG en 1997 et ILOG a ensuite été acquis par IBM en janvier 2009. CPLEX continue d'être activement développé par IBM. (4)



FIGURE 3.1 – Logo de IBM CPLEX

## 3.2 Résolution du KP

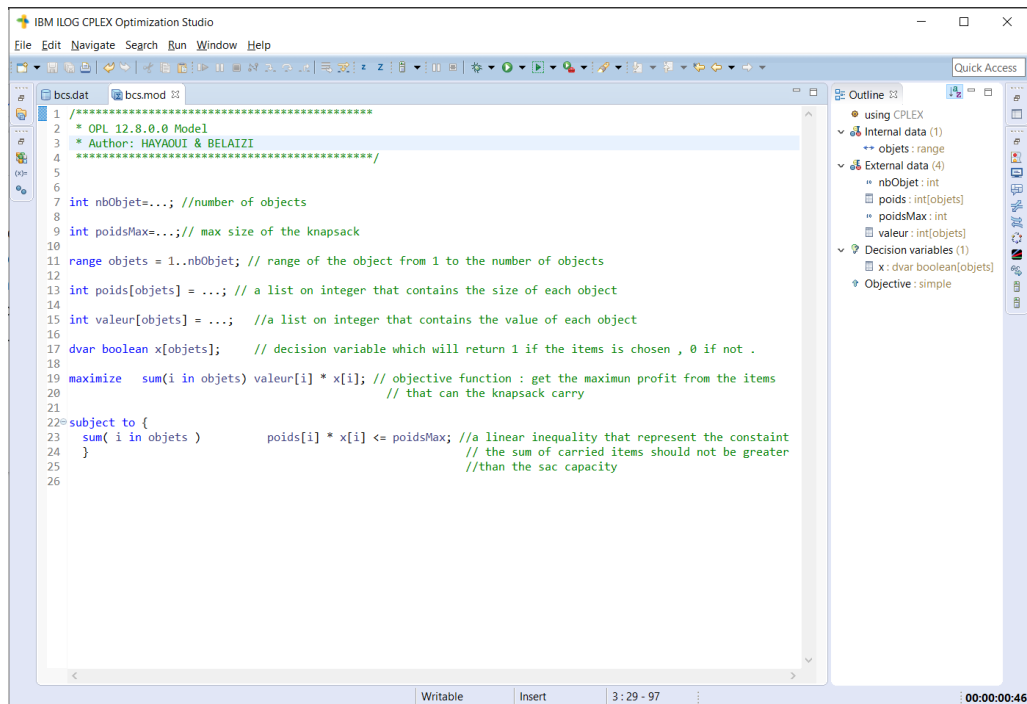
### 3.2.1 Programme de résolution avec Cplex

Pour la résolution avec CPLEX, on a tout d'abord écrit un programme qui contient tous les paramètres et les données. Pour notre version on a opté à faire deux fichiers séparés, le premier fichier qui porte l'extension .mod, et qui est le programme principal sans ajouter les données, Le deuxième fichier .dat qui contient les données du problème.

Les différents paramètres du programme sont :

- nbObjet : un nombre entier qui représente le nombre des objets.
- poidsMax : un nombre entier qui représente le poids maximal que le sac à dos peut porter.
- Objets : une suite arithmétique qui énumère les objets, par exemple si le nombre des objets est 4, Object sera donc : 1 2 3 4
- poids[objets] : une liste des entiers qui contient le poids de chaque élément. Exemple : poids[1] représente le poids du premier élément .
- valeur[objets] : une liste des entiers qui contient la valeur de chaque élément. Exemple : valeur[1] représente la valeur du premier élément .
- dvar boolean x[objets] : la variable de décision est une liste des booléennes , d'où pour chaque objet elle va être soit 1 ou 0 , par exemple si l'objet numéro 1 sera mis dans le sac x[1] sera égale à 1 «  $x[1] = 1$  » sinon elle va être 0 «  $x[1] = 0$  »
- maximize sum(i in objets) valeur[i] \* x[i] : l'objectif de ce problème qui est la maximisation de la somme des valeurs des objets mis dans le sac à dos .
- sum( i in objets ) poids[i] \* x[i] <= poidsMax : la contrainte du problème est que la somme des poids des objets mis dans le sac à dos qui ne doit pas dépasser le poids que ce dernier peut porter .

#### Le programme :



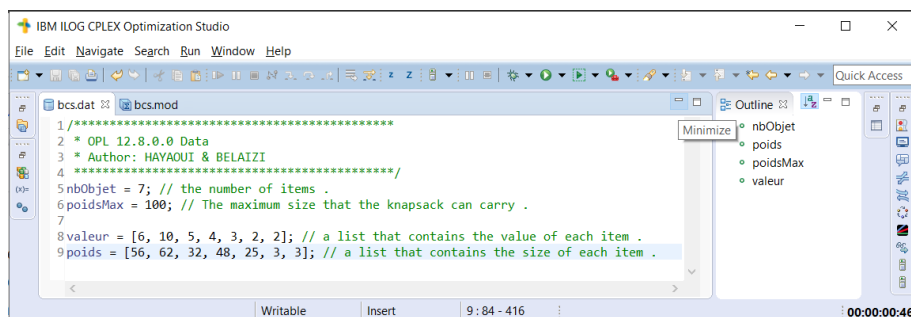
```
1 /*****
2  * OPL 12.8.0.0 Model
3  * Author: HAYAOUI & BELAIZI
4  *****/
5
6
7 int nbObjet=...; //number of objects
8
9 int poidsMax=...; // max size of the knapsack
10
11 range objets = 1..nbObjet; // range of the object from 1 to the number of objects
12
13 int poids[objets] = ...; // a list on integer that contains the size of each object
14
15 int valeur[objets] = ...; //a list on integer that contains the value of each object
16
17 dvar boolean x[objets]; // decision variable which will return 1 if the items is chosen , 0 if not .
18
19 maximize sum(i in objets) valeur[i] * x[i]; // objective function : get the maximum profit from the items
20 // that can the knapsack carry
21
22 subject to {
23     sum(i in objets) poids[i] * x[i] <= poidsMax; //a linear inequality that represent the constraint
24 } // the sum of carried items should not be greater
25 //than the sac capacity
26
```

FIGURE 3.2 – Le fichier .mod (model)

#### 3.2.2 exemples d'exécution

##### 1er exécution :

Pour cette exécution on a choisi un échantillon de 7 objets et un sac de capacité 100 qu'on a ajouté leurs données dans le fichier .dat ci-dessous




```
1 /*****
2  * OPL 12.8.0.0 Data
3  * Author: HAYAOUI & BELAIZI
4  *****/
5 nbObjet = 7; // the number of items .
6 poidsMax = 100; // The maximum size that the knapsack can carry .
7
8 valeur = [6, 10, 5, 4, 3, 2, 2]; // a list that contains the value of each item .
9 poids = [56, 62, 32, 48, 25, 3, 3]; // a list that contains the size of each item .

```

FIGURE 3.3 – Le fichier .dat(data)

Les objets sont représentés dans le tableau suivant avec leurs poids et leurs prix.



objet	poids	valeur
1	56	6
2	62	10
3	32	5
4	48	4
5	25	3
6	3	2
7	3	2




FIGURE 3.4 – Tableau de données

**Résultat :**

Après avoir lancer l'exécution le résultat est comme suite :

	Name	Value
▼	Data (5)	
10	nbObjet	7
10	objets	1..7
10	poids	[56 62 32 48 25 3 3]
10	poidsMax	100
10	valeur	[6 10 5 4 3 2 2]
▼	Decision variables (1)	
10	x	[0 1 1 0 0 1 1]

FIGURE 3.5 – Le résultat de 7 échantillons

**Interprétation du résultat :**

Le programme a trouvé la solution comme suite :

Choisir les objet : 2,3,6,7

La somme des poids des objets choisis est égale à 100 qui sont le poids maximal qui ne dépasse pas la capacité du sac à dos.

La somme des valeurs des objets est égale à 19

La solution est optimale.

```
// solution (optimal) with objective 19  
// Quality Incumbent solution:
```

FIGURE 3.6 – La solution est optimale

Le temps de résolution est 0.02 sec.

#### Deuxième exécution

Cette fois on a choisi un échantillon de taille 1,000 et on a considéré que la capacité du sac est 1,000,000 les valeurs des valeurs des objets sont aléatoires entre 0 et 1,000 et les valeurs des poids sont aussi aléatoires entre 0 et 10,000.

Pour des raisons d'organisation ces valeurs ne peuvent pas être mises dans ce document ainsi que le résultat.

#### Interprétation du résultat

Le programme a trouvé que le sac peut porter 117 objets avec une valeur de 999,551 sur 1,000,000

La solution est optimale avec tolérance.

```
// solution (integer optimal, tolerance) with objective 79972  
// Quality Incumbent solution:
```

FIGURE 3.7 – La solution est optimale avec tolérance

Le temps de résolution est 0.11 sec

## 3.3 Interface graphique convivial

### 3.3.1 Introduction

Comme on a vu dans la partie précédente le programme retourne une liste des booléennes qui montre si l'objet est choisi ou non avec des 0 et des 1 ce qui rend ce résultat difficile à lire pour un utilisateur normale, pour cela le rôle une interface graphique est important pour faciliter cette tâche



### 3.3.2 La mise en œuvre

La création d'une interface graphique qui est liée directement au programme dans cplex est impossible, pour cela on utilise l'API qu'IBM fournit aux développeurs pour utiliser cplex.

Pour notre cas on a utilisé l'API de cplex dans JAVA pour manipuler CPLEX, et JAVAFX pour créer et contrôler l'interface graphique

**JAVA :** Java est un langage de programmation évolué pour devenir un ensemble cohérent d'éléments techniques et non techniques. Java est utilisé dans une grande variété de plates-formes depuis les systèmes embarqués et les téléphones mobiles, les ordinateurs individuels, les serveurs, les applications d'entreprise, les superordinateurs, etc.



FIGURE 3.8 – Logo de JAVA

**JAVAFX :** JavaFX est un framework et une bibliothèque d'interface utilisateur issue du projet OpenJFX, qui permet aux développeurs Java de créer une interface graphique pour des applications de bureau, des applications internet riches et des applications smartphones et tablettes tactiles.



FIGURE 3.9 – Logo de JAVA FX

#### **Théorie**

notre idée a été de :

1. Crée un programme java qui écrit des fichiers .dat qui contient les valeurs des objets que l'utilisateur écrit dans l'interface pour les utiliser avec le fichier .mod qui contient le programme cplex.
2. Exporter le résultat des deux fichiers en un fichier .lp qui contient le programme linéaire avec le résultat.
3. Importer le fichier .lp dans java à l'aide de l'API de CPLEX et extraire le résultat
4. Traiter le résultat et l'afficher d'une manière que l'utilisateur peut lire facilement

#### **Pratique**

Dans cette partie on va considérer les valeurs de Table 1 avec le nom de chaque objet.

Nom	objet	poids	prix
Table 1	1	56	6
Table 2	2	62	10
Chaise 1	3	32	5
Chaise 2	4	48	4
Clavier	5	25	3
Verre	6	3	2
Lampe	7	3	2

FIGURE 3.10 – Tableau de données avec noms

La création du fichier .dat se fait en deux étapes

### Etape 1

L'utilisateur entre la valeur des nombre d'objets et la taille maximale du sac à dos dans les cases dédiés. Et clique sur valider, ceci va ouvrir une nouvelle interface et crée un fichier (s'il n'existe pas) qui s'appelle SacADos.dat dans le répertoire du programme et va écrire deux line qui contiennent le nombre des objets et la taille maximale du sac à dos, tout cela a l'aide de la méthode JAVA :

The screenshot shows a Java Swing window titled "Sac a dos". Inside, under the heading "Etape 1 :", there are two text input fields. The first is labeled "Le nombre des objets" and the second is labeled "Capacite Maximal du sac". Below these fields is a button labeled "Valider".

FIGURE 3.11 – Etape 1

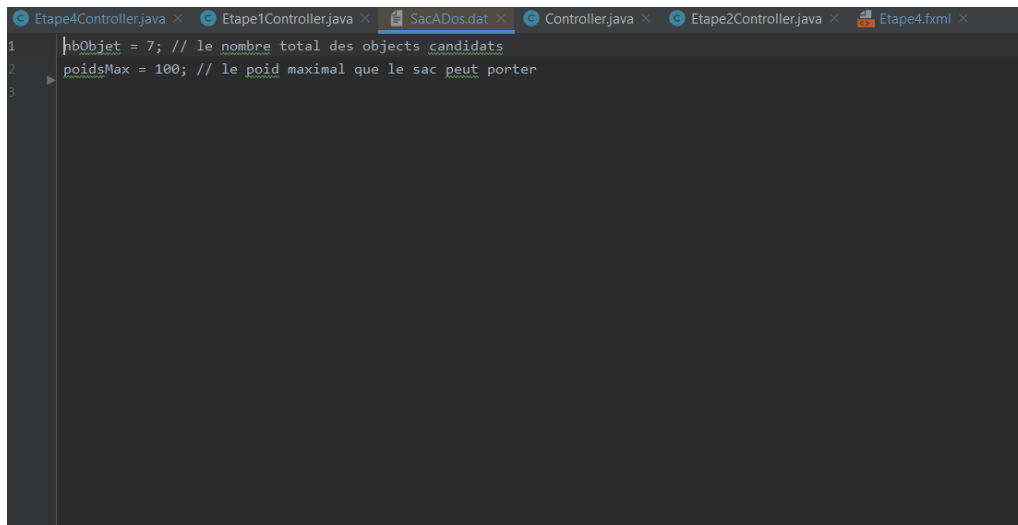


FIGURE 3.12 – Le fichier .dat générée après l'étape 1



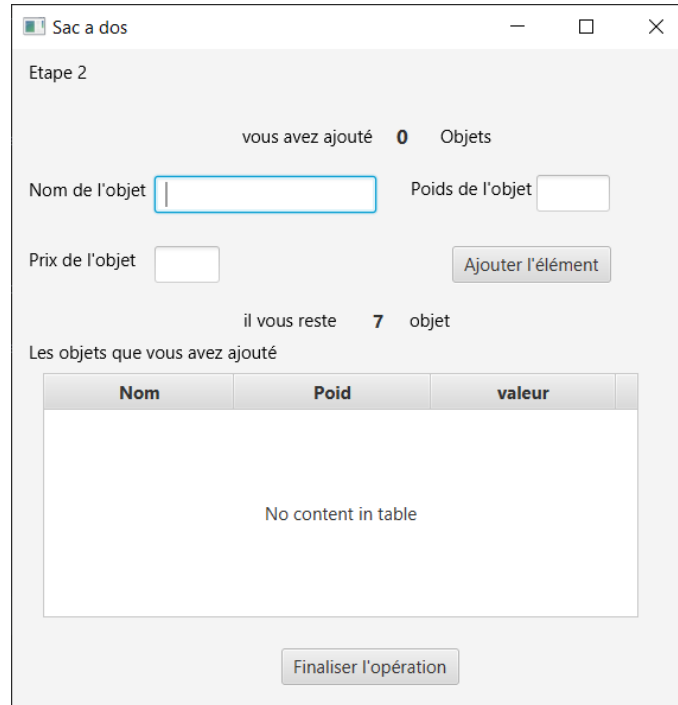
FIGURE 3.13 – La méthode java DataFileModify

## Etape 2

L'utilisateur dans cette étape doit écrire les détails de chaque objet et l'ajouter dans le tableau en cliquant sur le bouton « ajouter l'objet »

Le nom de chaque objet ainsi que son poids et sa valeur vont être ajoutée et stockée dans des Listes JAVA dynamiques .une fois le bouton « Finaliser L'opération » est déclenchée la méthode JAVA append dat file va ajouter les lignes qui contiennent la valeur et le poids des objets dans le fichier .dat déjà existant , puis

va exécuter une commande de OPL qui va exporter un fichier SacADos.lp a la base des deux fichier SacADos.mod et SacADos.dat .



Sac a dos

Etape 2

vous avez ajouté **0** Objets

Nom de l'objet  Poids de l'objet

Prix de l'objet  Ajouter l'élément

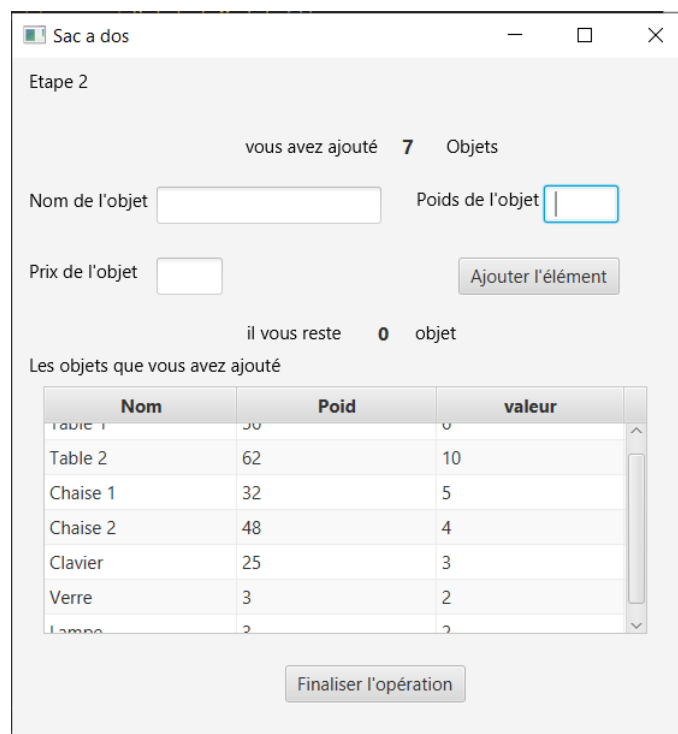
il vous reste **7** objet

Les objets que vous avez ajouté

Nom	Poid	valeur
No content in table		

Finaliser l'opération

FIGURE 3.14 – Etape 2 avant le remplissage du tableau



Sac a dos

Etape 2

vous avez ajouté **7** Objets

Nom de l'objet  Poids de l'objet

Prix de l'objet  Ajouter l'élément

il vous reste **0** objet

Les objets que vous avez ajouté

Nom	Poid	valeur
Table 1	50	8
Table 2	62	10
Chaise 1	32	5
Chaise 2	48	4
Clavier	25	3
Verre	3	2
Lampe	2	2

Finaliser l'opération

FIGURE 3.15 – Etape 2 après le remplissage du tableau

```
//Append parameters into the dat file
public void append_dat_file(){
    final File f = new File(Main.class.getProtectionDomain().getCodeSource().getLocation().getPath());
    String ProgPath = f.toString().replace( target: "%20", replacement: " ");
    try(FileWriter fw = new FileWriter( fileName: ProgPath + "\\SacADos.dat", append: true);
        BufferedWriter bw = new BufferedWriter(fw);
        PrintWriter out = new PrintWriter(bw))
    {
        out.println("valeur = " + Elements_PriceList + ";");
        out.println("poids = " + Elements_SizeList + ";");
    } catch (IOException e) {
    }
}
```

FIGURE 3.16 – La méthode JAVA append dat file

```
public void ExportCplexFiles() throws IOException {

    final File f = new File(Main.class.getProtectionDomain().getCodeSource().getLocation().getPath());

    String ProgPath = f.toString().replace( target: "%20", replacement: " ").replace( target: "\\ ", replacement: "\\\\ ");
    System.out.println("cd " + ProgPath);

    ProcessBuilder pb = new ProcessBuilder( _command: "oplrun",
        "-e",
        "SacADos.lp",
        "SacADos.mod",
        "SacADos.dat");
    pb.directory(new File(ProgPath));
    Process pr = pb.start();
    BufferedReader br = new BufferedReader(new InputStreamReader(pr.getInputStream()));
    String str = null;
    while ((str = br.readLine()) != null) {
        System.out.println(str);
    }
}
```

FIGURE 3.17 – La méthode JAVA ExportCplexFiles

### Etape 3

Cette étape affiche les objets choisis ainsi que leurs valeurs et leurs poids et calcule la valeur maximale et le poids occupé par ces objets sur la taille du sac à dos. cela arrive après avoir importer le fichier SacADos.lp , extraire le résultat et la traiter à l'aide de JAVA puis générer des listes qui contiennent les objets choisis , leurs valeurs et leur poids et afficher les résultats dans un tableau .

Sac a dos

Les objets choisis sont :

Nom	Poids	Valeur
Table 2	62	10
Chaise 1	32	5
Verre	3	2
Lampe	3	2

La valeur maximal est : 19

Le poids occupée par ces objets 100 / 100

Recommencer

FIGURE 3.18 – Etape 3

```

74
75 {
76     try {
77         final File f = new File(Main.class.getProtectionDomain().getCodeSource().getLocation().getPath());
78         String ProgPath = f.toString().replace( target: "%20", replacement: " ").replace( target: "\\", replacement: "\\\\");
79
80         String LPModel = ProgPath + "\\SacADos.lp";
81         model = new IloCplex();
82         model.importModel(LPModel);
83         model.solve();
84
85         IloLPMatrix lp = (IloLPMatrix) model.LPMatrixIterator().next();
86
87         double[] incx = model.getValues(lp);

```

FIGURE 3.19 – La méthode JAVA importLPFile

### 3.4 Conclusion

Le but de ce chapitre a été la résolution du sujet de ce document à l'aide de CPLEX, au cours du travail on a constaté que les ressources qu'IBM fournit au développeurs (les documentations) sont rares et difficile à comprendre pour ceux qui souhaitent utiliser CPLEX pour la première fois, pour cela on a créé un environnement autonome pour qu'un utilisateur ne se dérange pas de tout ce qui est code (back end) et se focalise sur le core du problème dans l'interface graphique (Front-end).

## 4 | Conclusion générale

Dans ce projet, nous avons étudié le problème du sac à dos, un des problèmes d'optimisation combinatoire qui occupe une place importante en mathématiques discrètes et en informatique. Son importance se justifie d'une part par sa grande difficulté et d'autre part par les nombreuses applications pratiques pouvant être formulées sous la forme d'une variante de ce problème. Dans la première partie, on a défini le problème en tant que contraintes et objectives ainsi que ses domaines d'application. Puis nous avons attaqué les méthodes adoptées pour le résoudre : les heuristiques et les méthodes exactes ; les méthodes exactes risquent de rencontrer des difficultés face aux applications de taille importante. L'appel aux méthodes approchées (heuristiques) devient une alternative lorsque certaines instances semblent complexes à résoudre à l'optimum. Nous avons aussi introduit une résolution simple à l'aide d'Excel avec 3 objets. Dans une deuxième partie nous avons présenté une vue générale sur quelques variantes de ce problème, le sac à dos multiple et quadratique, ainsi que quelques approches de résolution. Enfin, nous avons attaqué la modélisation à travers le solveur CPLEX ainsi que la résolution, nous avons aussi présenté une interface homme-machine qui rend l'utilisateur proche de la résolution du problème sans avoir accès à tout ce qui est code et langage d'optimisation.



## 5 | Références bibliographiques

- (1) <http://repository.usthb.dz/bitstream/handle/123456789/6421/TH8934.pdf>
- (2) <https://tel.archives-ouvertes.fr/tel-00439824/document>
- (3) Projet de mémoire Université Toulouse 3 Paul Sabatier :Titre : Contribution à la résolution de problèmes d'optimisation combinatoire : méthodes heuristiques et parallèles
- (4) <https://web.archive.org/web/20120927094255/https://www304.ibm.com/jct03002c/press/us/en/pressrelease/26403.wss>

