$N°$ D'ORDRE :02/2018 - D/MT

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté de Mathématiques

THÈSE

Présentée pour l'obtention du diplôme de DOCTORAT EN SCIENCES

En : MATHÉMATIQUES

Spécialité : Recherche Opérationnelle

Par : DAHMANI Isma

Sujet

Contribution à la résolution de problèmes de type sac à dos

Soutenue publiquement, le 28/02/2018, devant le jury composé de :

M. SEMRI Ahmed	Professeur	à l'USTHB	Président.
M. OUAFI Rachid	Professeur	à l'USTHB	Directeur de thèse.
M. HIFI Mhand	Professeur	à l'UPJV, Amiens	Co-Directeur de thèse.
M. AIDER Meziane	Professeur	à l'USTHB	Examinateur.
M. AIDENE Mohammed	Professeur	à l'UMMTO	Examinateur.
M. SADI Bachir	Maitre de conférences A	à l'UMMTO	Examinateur.

L'optimisme est la foi qui mène à la réalisation. Rien ne peut se faire sans l'espoir et la confiance.

Helen Keller

Remerciements

Je voudrais en premier lieu remercier Messieurs Ouafi Rachid et Hifi Mhand de m'avoir permis d'effectuer une thèse sous leur direction.

Je remercie le professeur Ouafi Rachid, de m'avoir apporté son soutien scientifique tout au long de ce travail. Il a su me transmettre les connaissances, la rigueur, la motivation et la passion pour la recherche.

Je remercie très sincèrement Monsieur Hifi Mhand, professeur de l'Université de Picardie Jules Verne, pour toute l'aide qu'il m'a apportée durant cette thèse. Il n'a jamais douté de moi et de mes capacités , il m'a également appris, grâce a sa très haute exigence scientifique et surtout sa très grande gentillesse et humilité à ne jamais renoncer et a donner le meilleur de moi-même.

Mes remerciements vont à Monsieur Semri Ahmed, Professeur à l'USTHB, pour avoir accepté de présider le jury. Je remercie également Messieurs Aider Meziane, Professeur à l'USTHB, Aidene Mohammed, Professeur à l'UMMTO et Saadi Bachir, Maitre de Conférences à l'UMMTO qui m'ont fait l'honneur d'accepter d'examiner ce travail.

Je remercie particulièrement mes chers parents et mon mari, leur présence et leur amour m'ont permis de surmonter mes peurs et mes angoisses.

Je remercie mes amies envers lesquelles je leurs serai éternellement reconnaissante de m'avoir soutenu et encouragé, surtout de ne pas m'avoir laissé tomber et de m'avoir offert des moments de détente, de soutien, d'écoute et de réconfort. Je ne les nommerai pas car elles se reconnaitront dans cette description.

Un grand merci à mon frère et mes soeurs pour leur compréhension et leur attention.

Un grand merci pour tous ceux qui aiment Isma et que Isma aime.

Résumé

Dans cette thèse, nous nous intéressons à la résolution approchée et exacte de deux problématiques appartenant à une classe de problèmes de l'optimisation combinatoire, à savoir : la famille des problèmes de type-sac à dos (KP).

Le premier problème étudié de cette famille est le problème de la distribution équitable(noté KSP). Notre contribution consiste à développer une approche basée sur la méthodologie de la recherche dispersée. Nous développons un nouveau générateur capable de produire la population de départ. Ensuite une population élite est choisie d'une manière efficace pour reproduire de nouvelles solutions en introduisant deux stratégies de combinaison. De plus, une stratégie d'amélioration est incorporée pour avoir des solutions de bonne qualité. Les résultats des expériences numériques sont comparés aux solutions optimales existantes dans la littérature mentionnant que l'approche produit des solutions très proches ou égales aux solutions optimales.

La deuxième variante de cette famille est la généralisation du problème de la distribution équitable (noté GKSP). Dans cette partie, nous proposons une méthode de résolution exacte qui se base principalement sur une décomposition du problème original en deux sous-problèmes(les deux variantes de sac à dos : KP et KSP). Ensuite, nous développons une borne supérieure et une borne inférieure pour le problème originale. Nous incorporons trois stratégies de réduction de l'espace de recherche afin d'accélérer le processus. Les résultats obtenus sont comparés avec ceux obtenus en appliquant les meilleurs algorithmes existants (Fujimoto et Yamada) pour ce problème et confirment l'efficacité de l'algorithme.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

L'optimisation combinatoire occupe une place importante en mathématiques discrètes et en informatique. 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.

Un problème d'optimisation combinatoire consiste à trouver une meilleure solution dans un ensemble discret, appelé espace de recherche ou ensemble des solutions réalisables du problème. De plus, cet ensemble, contenant les solutions réalisables, est défini de manière implicite et il est aussi parfois très difficile de trouver ne serait ce qu'une solution réalisable pour le problème. Par ailleurs, une solution optimale dans un ensemble discret et fini est un problème facile en théorie, puisqu'il suffit d'énumérer toutes les solutions possibles, de comparer leurs qualités, puis de déterminer la meilleure d'entre elles. Toutefois, en pratique, l'énumération de toutes ces solutions peut prendre un temps exorbitant. En effet, souvent le temps de calcul nécessaire pour trouver une meilleure solution sur un espace de recherche risque d'augmenter exponentiellement avec la taille de l'instance représentant la problématique étudiée. Dans ce cas, souvent les méthodes exactes rencontrent des difficultés face aux applications de taille importante.

Lorsque la taille des instances devient importante ou lorsque certaines instances semblent complexes à résoudre à l'optimum, l'appel aux méthodes approchés (heuristiques) devient une alternative pour la résolution. En effet, les méthodes approchées constituent une alternative intéressante pour traiter les problèmes d'optimisation de grande taille, en particulier lorsque l'optimalité n'est pas primordiale. Par ailleurs, déterminer une solution approchée peut ensuite servir comme un point de départ pour une méthode exacte : elle permet souvent d'effectuer un encadrement de l'optimum en la combinant avec d'autres bornes supérieures (resp. inférieures) dans le cas d'une maximisation (resp. minimisation). Ces dernière années, l'apparition des méthodes hybrides a aussi permis d'élargir la résolution exacte vers des problèmes combinatoire de plus en plus complexes. En effet, la combinaison entre des méthodes approchées et exactes a mené vers la résolution de certaines instances de grande taille (ou des instances ardues) pour des problèmes de l'optimisation combinatoire bien connus, comme par exemple des problèmes appartenant au domaine du transport : tournées de véhicules, chargement et déchargement de véhicules, tournées avec chargement, etc.

Nous nous sommes intéressés dans cette thèse à la résolution heuristique et exacte de deux problématiques appartenant à une classe de problèmes de

l'optimisation combinatoire : la famille des problèmes de type-sac à dos. Les problèmes de type knapsack interviennent dans de nombreux domaines industriels tels que le transport, la logistique et la production. Ils apparaissent soit en tant que problème principal, soit en tant que sous-problèmes de problèmes plus complexes. Cette thèse est organisée en trois chapitres. Nous présenterons quelques méthodes de résolution existantes que nous avons étudiées au cours de cette thèse. Nous terminerons par une conclusion qui résume le contenu du chapitre.

Dans une première partie, nous présenterons un cadre général des problématiques de type sac à dos. Nous présenterons le problème de sac à dos classique qui fait partie des problèmes d'optimisation combinatoire/continue les plus connus de la littérature. Nous donnerons également une classification des problèmes de type sac à dos très utilisée dans la littérature et la position du problème classique et les variantes étudiées dans cette classification. Par la suite, nous ferons un zoom sur deux variantes du problème de sac à dos, à savoir, la distribution équitable et la distribution équitable généralisée.

La deuxième partie contient nos contributions. Elle est divisée en deux chapitres.

Nous expliquerons dans le chapitre 2 une méthode diversifiante pour la résolution approchée du problème de la distribution équitable (le *knapsack sharing*). La méthode s'appuie sur un principe de diversification ("*Scatter Search*"), qui consiste en premier lieu à construire une solution réalisable de départ complète, puis de construire une population de solutions induite à partir d'une série d'éléments critiques associés à une série de sous-problème de sac à dos classiques. Ensuite, une profondeur autour de divers éléments critiques est introduite afin d'explorer différentes parties de l'espace de recherche. Pour terminer, certaines stratégies d'intensification et de diversifications autour de certaines parties de la population sont introduites afin de d'assurer une convergence lente pour la population finale.

Le Chapitre 3 est consacré à la résolution exacte du problème de la distribution équitable généralisé ("*Generalized Knapsack Sharing Problem*" $-GKSP$). La méthode s'appuie sur un principe de décomposition du problème original $GKSP$ en une série de sous-problèmes, où chacun de ces sous-problèmes est résolu par application de méthode spécifiques efficaces. Afin d'accélérer le processus de recherche et d'augmenter la taille des instances à résoudre, une borne supérieure ainsi que différentes stratégies de réduction de l'espace de recherche sont proposées.

Nous terminerons par une conclusion sur les travaux de recherche menés dans cette thèse ainsi que des perspectives de recherche et des directions de nos travaux futurs.

Chapitre 1

Problèmes de type sac à dos

1.1 Introduction

Les problèmes de type sac à dos (ou “Knapsack" : noté KP) ont été intensément étudiés depuis les travaux pionniers de Dantzig [5] dans les années 50. Ces problèmes possèdent de nombreuses applications dans l'industrie, la gestion de portefeuille, le transport, la gestion financière, les problèmes de découpe et de placement ainsi que d'autres problématiques liées au quotidien.

Ils surviennent souvent par la relaxation de divers problèmes de la programmation en nombres entiers. Dans de telles applications, nous faisons souvent recours à la résolution d'un problème de sac à dos à chaque fois qu'une sélection d'un sous-ensemble d'objets, parmi un ensemble d'objets de départ, est exigée afin de mettre en oeuvre des techniques de résolution performantes. Ce type de problème peut, en effet, apparaitre dans plusieurs situations comme sous-problème facilitant la résolution d'autres problèmes plus complexes (cf., Gilmore et Gomory [7], et Hifi et Roucairol [10]). Ce qui fait de lui un modèle théorique particulièrement intéressant.

Dans ce chapitre, nous présenterons le problème du sac à dos à variables binaires ainsi que quelques méthodes de résolution : exactes et approchées. Ensuite, nous aborderons le problème de la distribution équitable (noté KSP : “Knapsack Sharing Problem") et le problème de la distribution équitable généralisé (noté $GKSP$: “Generalized Knapsack Sharing Problem"), qui font l'objet de cette thèse. Pour ces deux dernières problématiques, nous présenterons quelques méthodes de résolution de la littérature. Enfin, nous discuterons certaines variantes du sac à dos qui sont souvent considérées comme des extensions du sac à dos, obtenues en ajoutant une ou plusieurs contraintes ou bien en modifiant sa fonction objectif ou le domaine de définition des variables de décision (pour plus de détails, le lecteur peut se référer à Martello *et al.* [18], Dudzinski *et al.* [14]).

1.2 Problème du sac à dos à variables binaires

Le problème du sac à dos à variables binaires (noté 0-1 KP ou parfois simplement KP) est l'un des problèmes les plus connus dans le domaine de

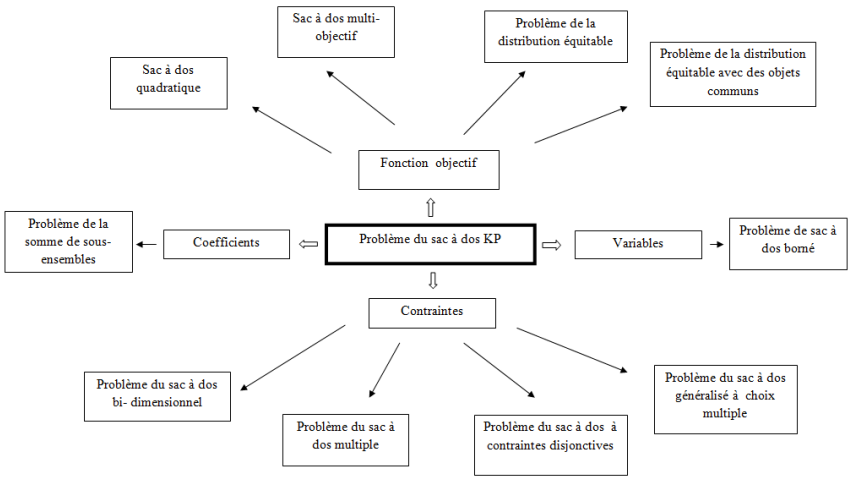

FIGURE 1.1 – Quelques variantes du problème du sac à dos

l'optimisation combinatoire. Il est représenté par un programme linéaire en nombres entiers constitué d'une seule contrainte charactérisant les objets de la solution. Bien qu'il soit simple à modéliser, le problème du sac à dos fait partie des problèmes NP-difficiles (cf. Garey and Johnson [19]). Nous rencontrons souvent ce problème dans la vie courante. A titre d'exemple, nous citons le problème du randonneur qui possède un sac à dos de poids limité à R. Chaque objet i qu'il peut emporter pèse un poids w_i et possède une utilité c_i pour la randonnée. Le randonneur doit répondre à la question suivante :

Quels objets doit-il emporter pour maximiser l'utilité totale, sans dépasser le poids total du sac ?

Ce problème peut être modélisé comme un problème de sac à dos (à variables binaires / bivalentes / 0-1). D'une manière générale, ce problème s'énonce comme suit : Étant donnés plusieurs objets possédant chacun un poids et une valeur (un profit), le but est d'effectuer une sélection sur les objets (déterminer un sous-ensemble de ces objets) dont le poids total soit inférieur ou égal à une valeur donnée (la capacité du sac) et dont la valeur (profit) totale (somme des valeurs (profits) des objets sélectionnés) soit maximum.

1.2.1 Description du problème

Le problème du sac à dos à variables binaires consiste à choisir un sous-ensemble d'éléments parmi n éléments telles que (i) la somme associée aux profits des éléments choisis parmi les n éléments soit maximale et (ii) la somme

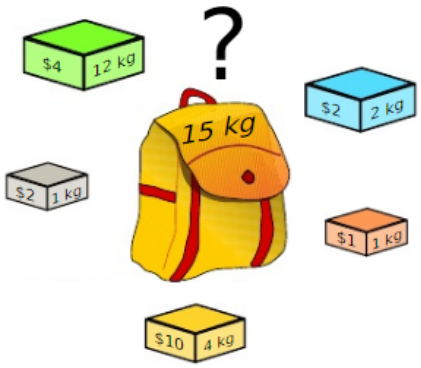

FIGURE 1.2 – Problème du sac à dos en 0-1

associée aux poids des éléments choisis parmi les n éléments ne dépasse pas la capacité associée à la contrainte du problème. D'une façon formelle, le problème peut s'écrire de la manière suivante :

$$KP: \quad \max \quad f(x) = \sum_{j=1}^{n} c_j x_j \tag{1.1}$$

$$s.c. \quad \sum_{j=1}^{n} w_j x_j \leq R \tag{1.2}$$

$$x_j \in \{0,1\} \qquad \forall j \in \{1,...,n\}, \tag{1.3}$$

où x_j est une variable (de décision) binaire qui vaut 1 si l'élément j est inclus dans le sac, 0 sinon. La contrainte (1.2) est appelée ***contrainte de capacité*** ou bien **contrainte du problème du sac à dos**.

Dans ce qui suit, sans perte de généralité et pour éviter les cas triviaux, nous considérons que les coefficients c_j, w_j et R sont des nombres entiers positifs et $\sum_{j=1}^{n} w_j > R$.

1.2.2 Méthodes de résolution du problème KP

Les problèmes de sac à dos appartiennent à la famille des problèmes NP-difficiles (cf. Garey and Johnson [19]). Parmi les techniques de résolution permettant de produire une solution optimale pour ce problème, on cite l'approche par l'énumération de l'espace des solutions admissibles / réalisables. Cependant, plusieurs méthodes de résolution exacte ont été développées pour

réduire l'effort de calcul que peut induire une énumération de l'espace de recherche. Parfois, le décideur peut aussi se limiter à des solutions approchées, ce qui revient à concevoir des méthodes heuristiques. Ce type d'approches prend place lorsqu'une résolution exacte demande un temps de calcul exorbitant. Dans ce qui suit, nous présentons quelques méthodes approchées et exactes pour le problème de sac à dos KP en $0-1$.

1.2.2.1 Heuristiques

Une heuristique est un algorithme approché qui permet d'identifier en un temps (souvent) polynomial au moins une solution réalisable. Cette solution peut ne pas être optimale, mais sa qualité peut être mesurée par rapport à une borne supérieure (resp. inférieure) pour un problème de maximisation (resp. minimisation). Cette analyse peut être aussi l'objet d'une partie expérimentale où la qualité des solutions heuristiques est mesurée expérimentalement. L'usage d'une heuristique reste efficace pour calculer une solution approchée d'un problème et parfois il s'avère intéressant pour l'accélération de la recherche pour une méthode exacte. Parmi les méthodes heuristiques, on peut citer l'algorithme *glouton* (connu sous le nom de "Greedy Algorithm").

Cet algorithme est exprimé de tel sorte qu'à chaque itération, la valeur d'une (*ou plusieurs*) variable (*variables*) de décision charactérisant le problème est (sont) fixée(s) sans remettre en cause les choix antérieurs. Le principe d'une telle procédure revient donc de démarrer avec une solution incomplète que l'on complète de proche en proche en effectuant des choix locaux définitifs : à chaque étape, on traite une partie des variables sur lesquelles on ne revient plus.

Dans la littérature, on trouve différentes variantes qui s'appuient sur le principe "glouton". En particulier, pour le KP, nous nous intéresserons à une une version simplifiée que nous présenterons dans ce qui suit.
En premier lieu, on calcule le rapport profit par poids (c_j/w_j) de chacun des éléments j, $j \in \{1, \ldots, n\}$, et on trie les objets selon l'ordre décroissant de ces valeurs, i.e.,

$$\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq ..... \geq \frac{c_n}{w_n}. \tag{1.4}$$

Ensuite, on sélectionne les éléments un par un dans l'ordre du tri et on procède à l'ajout d'objets successifs dans le sac jusqu'à saturation de la capacité du sac. Chacune des sélections s'effectue sur l'élément qui se situe le plus à gauche selon l'orde de trie. Finalement, les éléments non sélectionnés sont automatiquement réduit à zéro. L'algorithme 1 décrit les étapes principales de l'heuristique qui s'appuie sur le principe glouton.

Algorithm 1 Heuristique gloutonne pour le problème de sac à dos

Entrées: Une instance du problème KP en 0-1.
Sorties: Solution réalisable $\overline{x}$.

1: Soit $\overline{R} = R$ la capacité résiduelle initiale ;
2: **pour** $i = 1$ à n **faire**
3: **si** $w_i \leq \overline{R}$ **alors**
4: $\overline{x}_j = 1$;
5: $\overline{R} = \overline{R} - w_i$;
6: **sinon**
7: $\overline{x}_j = 0$;
8: **finsi**
9: **fin pour**
10: **Retourner** $\overline{x}$;

1.2.2.2 Calcul de bornes

Pour un problème d'optimisation combinatoire, le calcul de bornes supérieures ou inférieures permet souvent de réduire l'espace de recherche des solutions et d'accélérer le processus de recherche. Ces bornes sont utilisées pour le développement de méthodes exactes basées sur des procédures d'énumération implicite (ou méthodes par séparation et évaluation).

Dantzig [5] a proposé une borne supérieure efficace pour le problème de sac à dos en résolvant le problème de sac à dos relaxé. Cela signifie que les variables x_j sont prises dans l'intervalle continu $[0, 1]$. La méthode qu'il a proposé pour la résolution de cette relaxation linéaire est équivalente à l'application de la méthode du simplexe sur cette même relaxation, mais en plus simple. La relaxation linéaire du problème KP en 0-1 (noté LP_{KP}) est donnée par le programme linéaire suivant :

$$\begin{aligned} LP_{KP}: \quad Max: \quad & Z(x) = \sum_{j=1}^{n} c_j x_j \\ s.c. \quad & \sum_{j=1}^{n} w_j x_j \leq R \\ & 0 \leq x_j \leq 1, \quad j = 1, \ldots., n \end{aligned}$$

où seules les variables de décision sont relaxées.

La résolution du problème (LP_{KP}) consiste à : (i) ordonner les éléments selon l'ordre de l'équation (1.4) et (ii) de remplir le sac, élément après élément, jusqu'à sa saturation. Ensuite, le premier élément x_e $(1 \leq e \leq n)$ ne

pouvant pas être mis en totalité dans le sac est repéré. On appelle l'indice e caractérisant la variable x_e l'élément critique de KP ; il s'agit de l'élément qui réalise l'inégalité suivante :

$$\sum_{j=1}^{e-1} w_j \leq R < \sum_{j=1}^{e} w_j$$

La solution du (PL_{KP}) peut être exprimée de la manière suivante :

$$\overline{x} := \begin{cases} 1 & \text{si} \quad j = 1, \ldots, e-1 \\ \dfrac{\left(R - \sum_{j=1}^{e-1} w_j\right)}{w_e} & \text{si} \quad j = e \\ 0 & \text{si} \quad j = e+1, \ldots, n, \end{cases}$$

et la valeur de la solution optimale de (PL_{KP}) est donnée par :

$$Z_{(PL_{KP})} = \sum_{j=1}^{e-1} c_j + \Big(\frac{R - \sum_{j=1}^{e-1} w_j}{w_e}\Big) c_e.$$

On en déduit que la borne supérieure du problème KP, appelée aussi borne de Dantzig [5] (notée U_{BS}), est égale à

$$U_{BS} = \sum_{j=1}^{e-1} c_j + \Big\lfloor \Big(\frac{R - \sum_{j=1}^{e-1} w_j}{w_e}\Big) c_e \Big\rfloor, \tag{1.5}$$

où $\lfloor x \rfloor$ est la partie entière inférieure de x.

Si nous posons $r^* = R - \sum_{j=1}^{e-1} w_j$, la borne de Dantzig devient :

$$U_{BS} = \sum_{j=1}^{e-1} c_j + w \Big\lfloor \Big(\frac{r^*}{w_e}\Big) c_e \Big\rfloor. \tag{1.6}$$

Une amélioration de cette borne a été proposée par Martello et Toth [17]. Cette borne domine celle de Dantzig en imposant la contrainte d'intégralité de l'élément x_e de la manière suivante :

$$U_0 = \sum_{j=1}^{e-1} c_j + \left\lfloor \left(\frac{r^*}{w_{e+1}} \right) c_{e+1} \right\rfloor \tag{1.7}$$

$$U_1 = \sum_{j=1}^{e-1} c_j + \left\lfloor c_e - \frac{(w_e - r^*)}{w_{e-1}} c_{e-1} \right\rfloor. \tag{1.8}$$

Ainsi, la borne de Martello et Toth est donnée par : $U_{MT} = Max(U_0, U_1)$. Il existe dans la littérature plusieurs méthodes qui améliorent les bornes mais elles restent plus coûteuses en terme de complexité temporelle (temps de calcul). Les deux précédentes bornes ont une complexité chacune de l'ordre de $O(nlogn)$ (pour plus de détails, le lecteur peut se référer à Martello et Toth [18]).

1.2.2.3 Méthodes exactes

Dans cette section nous présentons deux méthodes exactes pour la résolution du KP : une méthode par séparation et évaluation et une autre méthode qui s'appuie sur la programmation dynamique. Ces méthodes garantissent l'optimalité de la solution finale, en particulier pour des instances de grande taille.

A) Méthodes par séparation et évaluation

La méthode qui s'appuie sur une procédure par séparation et évaluation est une méthode permettant la recherche d'une solution optimale dans un espace de recherche de solutions réalisables. Elle peut aussi être appliquée pour la recherche de plusieurs solutions optimales. Cette dernière se repose sur les stratégies suivantes :

- **Séparation**
  La séparation consiste à diviser le problème en sous-problèmes. Ainsi, en résolvant tous les sous-problèmes et en gardant la meilleure solution trouvée, on est assuré d'avoir résolu le problème initial. Cela revient à construire un arbre permettant d'énumérer toutes les solutions. L'ensemble des nœud de l'arbre non encore visités (parcourus) susceptibles de contenir une solution optimale, c'est-à-dire encore à diviser, représente l'ensemble des nœud actifs.

- **Évaluation**
  L'évaluation permet de réduire l'espace de recherche en éliminant des sous-ensembles qui ne contiennent pas la solution optimale. L'objectif revient à évaluer la fonction objectif à différents niveaux, puis de décider de l'intérêt de l'exploration dans cette direction. Cette technique permet donc l'élimination de branches dans l'arborescence de recherche : la recherche d'une solution de profit maximum par exemple, consiste à mémoriser la solution de plus grand profit rencontrée lors de l'exploration, et à comparer le profit associé à chaque nœud parcouru à celui de la meilleure solution. Dans le cas où le profit du nœud considéré est inférieur au meilleur profit obtenu, l'exploration de la branche s'arrête et toutes les solutions induites de cette branche seront nécessairement de profit moins important que la meilleure solution déjà trouvée.

- **Stratégie de parcours**
  Les méthodes par séparation et évaluation utilisent souvent de trois types de parcours.
  - **Parcours en largeur** : Cette stratégie favorise les sommets les plus proches de la racine en faisant moins de séparations du problème initial. Elle est moins efficace que les deux autres stratégies présentées ci-dessous et elle est moins utilisée lors de la recherche d'une solution optimale pour un problème d'optimisation combinatoire.

  - **Parcours en profondeur** : Cette stratégie favorise les sommets les plus éloignés de la racine (de profondeurs plus élevées : plus vers les feuilles de l'arborescence de recherche) en appliquant plus de séparations au problème initial. Cette stratégie mène souvent vers une solution optimale en favorisant l'espace mémoire à utiliser.

  - **Parcours par le meilleur d'abord** : Cette stratégie consiste à explorer des sous-problèmes possédant la meilleure estimation de la borne sur l'ensemble des nœuds de l'arborescence. Elle permet aussi d'éviter l'exploration de tous les sous-problèmes qui possèdent une mauvaise évaluation par rapport à la valeur optimale.

La plupart des méthodes par séparation et évaluation s'appuient sur le même principe à la différence qui se focalise souvent sur le choix de la stratégie de séparation. Dans cette partie, nous décrivons la méthode par séparation et évaluation proposée par Horowitz et Sahni [13] (l'une des première méthodes proposée pour la résolution exacte du KP en 0-1 et reste l'une des plus performantes). Cette dernière consiste à ordonner les éléments selon

Algorithm 2 Séparation et évaluation en profondeur pour le KP

Entrées: Une instance du KP en 0-1.
Sorties: Solution optimale $x*$.

1: $j = 1$; $Meilleure\_Valeur = 0$;
2: Calcul de la borne supérieure U_{BS} que l'on peut atteindre depuis j ;
3: **si** $U_{BS} \geq Meilleure\_Valeur$ **alors**
4: Développer une branche de l'arborescence en profondeur d'abord pour obtenir une solution réalisable $\overline{x'}$;
5: $j = j + 1$;
6: **si** $Z(\overline{x'}) \geq Meilleure\_Valeur$ **alors**
7: $\overline{x} = \overline{x'}$;
8: $Meilleure\_Valeur = Z(\overline{x'})$;
9: **finsi**
10: **finsi**
11: $j = max\{k : k \leq j \quad et \quad \overline{x'} = 1\}$
12: **Tant que** j existe **faire**
13: $\overline{x'}_j = 0$;
14: Aller à 2 ;
15: **fin du « Tant que »**
16: **Retourner** $x^* = \overline{x}$;

l'ordre décroissant du rapport profit sur poids. Ensuite, la séparation se fait sur l'élément suivant. A partir de chaque nœud de l'arborescence, et pour un élément j pris dans l'ordre prédéfini, deux branches sont développées : une première branche pour $x_j = 1$, qui correspond à l'élément j mis dans le sac et une deuxième branche pour $x_j = 0$, qui correspond à l'élément j qui est exclu du sac. Le développement de l'arborescence se fait en profondeur, en privilégiant les branches pour lesquelles les éléments sont mis dans le sac. La fonction d'évaluation utilisée est la borne de Dantzig U_{BS} (présentée dans la section 1.2.2.2).

Un élément est sélectionné si son poids ne dépasse pas la capacité résiduelle du sac. Ainsi, à chaque développement d'une branche complète de l'arborescence, la solution obtenue reste réalisable. Le calcul des bornes de Dantzig se fait uniquement au niveau des nœuds développés par une branche correspondant à un élément j non sélectionné $x_j = 0$. Pour les nœuds développés par une branche correspondant à $x_j = 1$, la valeur de la borne de Dantzig n'est autre que celle obtenue au niveau du nœud père. Cette dernière est comparée à la valeur de la meilleure solution obtenue jusqu'à présent. Dans le cas où la borne

est inférieure à la valeur de cette meilleure solution (un problème de maximisation), alors il est évident que l'on ne pourra jamais atteindre une meilleure solution à partir de ce nœud (les solutions qui en découlent seront toutes avec une valeur moins importante que celle du nœud père). L'algorithme 2 résume les principales étapes de l'algorithme de Horowitz et Sahni [13].

B) Application de la programmation dynamique

La programmation dynamique est une méthode exacte de résolution de problèmes d'optimisation séquentielle, due essentiellement à Bellman [1]. Cette méthode est adressée aux problèmes qui vérifient le principe d'optimalité. Ce principe repose sur le fait que toute solution optimale s'appuie elle-même sur des sous-problèmes résolus de façon optimale. En d'autres termes, nous pouvons déduire la solution optimale du problème en combinant des solutions optimales d'une série de sous-problèmes. Les solutions des problèmes sont étudiées *de bas en haut*, c'est-à-dire qu'on calcule les solutions des sous-problèmes les plus petits pour ensuite en déduire petit à petit les solutions de l'ensemble.

Quelques notations :

Nous introduisons les notations suivantes pour une étape t, $t = 1, \ldots, n$, où n est le nombre d'étapes.

- X_t : est l'ensemble de *décisions* x_{t_i} que l'on prend à l'étape t,
- E_t : est l'ensemble des états e_{t_i} dans lesquels le système se trouve à l'étape t,
- $P(e_{t-1}, x_t)$: est le profit associé à la décision x_t, qui fait passer le système de l'état e_{t-1} vers l'état e_t,
- $f(t, e_t) = \sum_{j=1}^{t} P(e_{j-1}, x_j)$: est le profit total de la séquence de décisions $x_1, x_2, ..., x_t$ qui fait passer le système de l'état e_0 vers l'état e_t,
- $F(e_{t-1}, X_t) = e_t$: est la transition d'état.

Trouver une solution pour le problème revient à trouver une séquence optimale $x^* = (x_1, x_2, ..., x_n)$ qui à partir de l'état initial e_0 nous amène à l'état e_n tout en maximisant la fonction profit :

$$f(n, e_n) = \max \Big\{ \sum_{j=1}^{n} P(e_{j-1}, x_j) \mid x_j \in X_j, e_j = F(e_{j-1}, X_j), j \in \{1, ..., n\} \Big\}.$$

Nous pouvons calculer de proche en proche $f(n, e_n)$, en appliquant le principe d'optimalité de *Bellman*. En effet, la formule de récurrence est la suivante :

$$f(t, e_t) = \max_{x_t \in X_t, e_t = F(e_{t-1}, X_t)} \Big\{ P(e_{t-1}, x_t) + f(t-1, e_{t-1}) \Big\}.$$

Dans ce qui suit, nous décrivons les étapes de la méthode adaptées au problème KP. Nous reprenons la formulation du KP donné par le modèle linéaire à variables binaires et nous définissons pour chacune des valeurs de $t,\ t = 1, \ldots, n$, et $g,\ g = 0, \ldots, R$, le sous problème $KP(t, g)$ de la manière suivante :

$$KP(t,g) \begin{cases} \max \quad Z(x) = \sum_{j=1}^{t} c_j x_j \\ sc \quad \sum_{j=1}^{t} w_j x_j \leq g \\ \quad x_j \in \{0,1\}, \quad j = 1, ..., t. \end{cases} \tag{1.9}$$

Le profit $f(t, g)$ du sous-problème $KP(t, g)$ est obtenu en remplissant le sac à dos de capacité g avec les t premiers éléments. Par conséquent, la solution optimale du KP est équivalente à la solution optimale de $KP(n, R)$:

- L'ensemble des décisions prises à l'étape t : $X_t = \{0, 1\}$
- L'ensemble des états dans lesquels le système se trouve à l'étape t :

$$E_t = \Big\{ e_t = g \mid g \in \Big\{ 0, 1, \ldots, \min\{R, \sum_{j=1}^{t} w_j\} \Big\} \Big\}.$$

- Le profit de la décision x_t :

$$P(e_{t-1}, x_t) = c_t w_t$$

.

- La transition d'état :

$$F(e_{t-1}, x_t) = e_t = e_{t-1} + w_t x_t$$

.

- Le profit total de la séquence des décisions $x_1, x_2, ..., x_t$ qui fait passer le système de l'état e_0 à l'état e_t :

$$\sum_{j=1}^{t} P(e_{j-1}, x_j) = \sum_{j=1}^{t} c_j x_j$$

.

Notons que le KP est défini pour chacune des valeurs de t et g, donc la séquence des décisions optimales $x_g = (x_{g1}, ..., x_{gt})$ qui maximise le profit pour un sac à dos de capacité g.

Pour chaque valeur de g, $g = 1, \ldots, R$, :

$$f(t, g) = \sum_{j=1}^{t} P(e_{j-1}, x_{gj})$$

.

La relation récursive permettant de construire la liste contenant l'ensemble des solutions des problèmes $f(t, g)$ pour une étape t est donnée par :

$$f(t,g) = \begin{cases} f(t-1,g), \ g \in \{0, ..., w_t - 1\} \\ \max\Big\{f(t-1,g), c_t + f(t-1, g-w_t)\Big\}, \ g \in \Big\{w_t, \ldots, \min\big\{R, \sum_{j=1}^{t} w_j\big\}\Big\}. \end{cases}$$

L'algorithme 3 décrit les différente étapes utilisées par la procédure récursive qui renvoie la valeur optimale du problème KP.

1.3 Le problème de la distribution équitable

Dans cette partie, nous présenterons le problème de la distribution équitable (Knapsack Sharing Problem : KSP), qui fait partie des problèmes d'optimisation de type max-min. Nous le trouvons dans des domaines multiples, comme le commerce, le transport, les communications, l'allocation des ressources, etc (cf., Brown [3]). Pour mieux comprendre l'aspect de "*distribution équitable*", prenons l'exemple d'une société qui dispose de M projets, où chaque projet nécessite un coût de réalisation et dispose d'une estimation du profit. La société cherche à partager équitablement son budget R pour la réalisation de ses M projets. Partager équitablement ne revient pas à diviser le budget en parts égales (i.e., $\frac{R}{M}$) entre les différents projets mais plutôt à maximiser la plus petite valeur du profit global par projet ; sans dépasser le budget total alloué pour la réalisation de ces projets. La distribution équitable ne consiste donc pas à partager en parts égales les ressources mais plutôt de maximiser la plus petite part. Dans ce qui suit, nous considérons le problème du KSP à variables binaires.

1.3.1 Description du problème KSP

Une instance du KSP binaire est définie par un sac à dos de capacité R et d'un ensemble N d'éléments. Cet ensemble est réparti sur m classes disjointes.

Algorithm 3 Algorithme récursif pour le KP

Entrées: C, W sont les vecteurs profit et poids et M, V représentent les matrices de taille $(n \times R)$.

Sorties: La valeur optimale du problème $M[n, R]$.

```
1: Remplir la matrice M par la valeur −1 ;
2: Remplir la matrice Booléenne V par faux ;
3: ProgDynamique (W, C, n, R)
4: si M[n, R] ≥ 0 alors
5:    Retourner M[n, R] ;
6: sinon
7:    si n = 0 ou R = 0 alors
8:       M[n, R] = 0 ;
9:    sinon
10:      q1 = ProgDynamique (W, C, n − 1, R) ;
11:      si R − W[n] < 0 alors
12:         q2 = −∞ ;
13:      sinon
14:         q2 = ProgDynamique (W, C, n − 1, R − W[n]) + C[n]
15:      finsi
16:      si q1 < q2 alors
17:         M[n, R] = q2 ; V[n, R] = vrai ;
18:      sinon
19:         M[n, R] = q1 ;V[n, R] = faux ;
20:      finsi
21:   finsi
22: finsi
```

Si N_i représente l'ensemble des éléments de la $i^{\text{ème}}$ classe $i \in \{1, ..., m\}$, alors $\forall p = 1, ..., m$ et $\forall q = 1, ..., m$, $p \neq q$ $N_p \bigcap N_q = \emptyset$ et $\bigcup_{i=1}^{m} N_i = N$. De plus, à chaque élément j de la classe i est associé un poids w_{ij} et un profit c_{ij}.

L'objectif du problème est de déterminer le sous-ensemble d'objets à retenir dans le sac tout en satisfaisant la contrainte de capacité. Ce sous-ensemble d'éléments est choisi de façon à maximiser la valeur de la fonction objectif qui réalise le minimum par rapport à l'ensemble de toutes les classes. Finalement, ce problème peut être formulé comme suit :

$$KSP: \quad \max \quad \min_{1\leq i\leq m}\Big\{\sum_{j\in N_i} c_{ij}x_{ij}\Big\} \tag{1.10}$$

$$s.c. \quad \sum_{i=1}^{m}\sum_{j\in N_i} w_{ij}x_{ij} \leq R \tag{1.11}$$

$$x_{ij}\in\{0,1\}, \quad j\in N_i, \quad i=1,\ldots,m. \tag{1.12}$$

La variable de décision x_{ij} vaut un si l'élément j de la classe i est retenu dans le sac et zéro sinon. L'équation (1.11) désigne la contrainte de capacité du sac. Le programme linéaire en nombres entiers associé au KSP peut être écrit sous la forme suivante :

$$PLNE_{KSP}: \quad \max \quad \gamma \tag{1.13}$$

$$s.c. \quad \sum_{i=1}^{m}\sum_{j\in N_i} w_{ij}x_{ij} \leq R, \tag{1.14}$$

$$\sum_{j\in N_i} c_{ij}x_{ij} \geq \gamma, \quad i=1,...,m \tag{1.15}$$

$$x_{ij}\in\{0,1\}, \quad \gamma\in N,\ j\in N_i,\ i=1,\ldots,m. \tag{1.16}$$

L'inégalité (1.14) est la contrainte de capacité du sac et les équations (1.15) sont les contraintes représentant la fonction objectif dur problème KSP.

Le KSP est un problème d'optimisation combinatoire appartenant à la classe NP-complet, noté dans la littérature par KSP $(Bn/m/1)$ (voir Yamada et Futakaw[20] ; Hifi et Sadfi[11]). Cette notation signifie que nous disposons de n éléments de type binaire B, répartis sur m classes et avec une seule contrainte.

Un grand nombre de méthodes exactes et approchées sont développées pour la résolution du problème. Yamada et al[21] ont proposé deux méthodes exactes, l'une basée sur la méthode par séparation et évaluation et l'autre basée sur la méthode de recherche dichotomique. Les résultats numériques montrent que l'algorithme basé sur la méthode de recherche binaire est plus performant que l'algorithme basé sur la méthode par séparation et évaluation. Hifi et Sadfi[11] puis Hifi *et al.* [9] ont proposé des algorithmes basés sur la technique de la programmation dynamique permettant une résolution exacte sur un bon nombre d'instances de la littérature. Par la suite, Boyer *et al.* [2] ont proposé une méthode qui a amélioré la performance de la méthode basée sur la programmation dynamique, en introduisant plusieurs notions de dominances.

Finalement, Hifi et Wu [12] ont proposé une nouvelle borne supérieure ainsi qu'une méthode exacte pour la résolution du KSP. La méthode proposée applique une approche de décomposition, où le problème original a été décomposé en une série de problèmes de minimisation et maximisation de sac à dos. Cette décomposition a été auparavant proposé dans Hifi et Sadfi[11]. Dans une partie expérimentale, une étude comparative entre différentes méthodes a été menée où les auteurs ont montré la compétitivité de la nouvelle méthode.

Dans ce qui suit, nous décrivons quelques méthodes exactes et approchées de la littérature pour la résolution du KSP.

1.3.2 Méthodes de résolution pour le problème KSP

1.3.2.1 Heuristique

Yamada et Futakawa [20] ont proposé un algorithme basé sur une méthode gloutonne. Le principe de l'approche est de partir d'une solution initiale partielle réalisable et d'ajouter, d'une manière itérative, des composantes de la solution jusqu'à obtenir une solution complète. L'algorithme utilise les étapes suivantes :

1. Afin de construire une solution de départ pour l'algorithme, les auteurs utilisent le problème relaxé R_{KPS} suivant :

$$R_{KSP} \begin{cases} \max & \min\limits_{1\leq i\leq m} \{\ Z(r_i)\} \\ \text{s.c.} & \sum\limits_{i=1}^{m} r_i \leq R \\ & r_i \geq 0, \quad i = 1, ..., m, \end{cases}$$

   où r_i et $Z(r_i)$ dénotent respectivement la capacité et la valeur de l'objectif de la i-ème classe.

   Notons par $\underline{Z}$ la solution optimale de R_{KSP} et u_i l'indice j réalisant $u_i = \min\left\{j \mid \sum_{k=1}^{j} c_{ik} > \underline{Z}\right\}$. Une solution réalisable initiale est obtenue en fixant les variables de décision de chaque classe i, $i = 1, \ldots, m$, de la manière suivante :

$$x_{ij} = \begin{cases} 1 & j < u_i - 1 \\ 0 & \text{sinon.} \end{cases}$$

2. Ensuite, la solution courante est complétée en prenant les éléments un par un tout en tentant de mettre le maximum d'éléments jusqu'à la saturation du sac. A chaque étape, l'élément considéré est le premier élément, en respectant l'ordre décroissant des rapports du profit par poids des éléments. De plus, cet élément doit appartenir à la classe pour

laquelle la valeur de la somme des profits des éléments déjà mis dans le sac est la plus petite par rapport à la valeur de la somme des profits des éléments des autres classes déjà mis dans le sac.

1.3.2.2 Méthode par séparation et évaluation

Yamada et al [21] ont proposé la première méthode exacte pour le KSP. Cette dernière utilise une méthode par séparation et évaluation, où la borne supérieure utilisée à chaque niveau de l'arborescence est obtenue par la résolution du problème relaxé Rel_{KSP} suivant :

$$Rel_{KSP} \begin{cases} \max \quad \min\limits_{1\leq i\leq m} \{\sum\limits_{j\in N_i} c_{ij}x_{ij}\} \\ sc \quad \sum\limits_{i=1}^{m}\sum\limits_{j\in N_i} w_{ij}x_{ij} \leq R, \\ x_{ij} \in [0,1], \quad j \in N_i, \quad i = 1,\ldots,m. \end{cases}$$

Afin d'obtenir la borne inférieure, l'algorithme incorpore une heuristique auparavant proposée par Yamada *et al.* [20] (précédemment décrite dans la Section 1.3.2.1). Comme stratégie de séparation, les auteurs font la comparaison entre deux stratégies : (i) un *branchement séquentiel* qui consiste à choisir le premier élément non encore mis dans le sac et (ii) un *branchement par le premier minimum* qui consiste à prendre le premier élément de la classe dont la somme de ces objets mis dans le sac est la plus petite. Dans une partie expérimentale, les auteurs ont montré que les résultats obtenus par application de la stratégie du branchement séquentiel étaient meilleurs.

1.3.2.3 Méthode exacte basée sur la programmation dynamique

Plusieurs méthodes exactes pour le KSP basées sur la programmation dynamique ont été proposées dans la littérature. L'intérêt porté par les auteurs (cf., Hifi et Sadfi [11], Yamada *et al.* [21] et Hifi *et al.* [9]) à cette méthode s'appuie principalement sur l'efficacité de cette technique lorsqu'elle est appliquée à des instances de taille plus ou moins modérée.

Nous décriront dans ce qui suit l'un des algorithmes utilisant ce concept proposé par Hifi et Sadfi [11], qui représente la première version utilisant la technique de la programmation dynamique. En premier lieu, les auteurs proposent de décomposer le problème original KSP en une série de problèmes auxiliaires de sac à dos KP, noté $KP_{J_i}(r_i)$, $i = 1, ..., m$. Ces problèmes sont

formulés comme suit :

$$KP_{J_i}(r_i): \quad \max \quad Z(x) = \sum_{j \in J_i} c_{ij} x_{ij}$$
$$s.c. \quad \sum_{j \in J_i} w_{ij} x_{ij} \leq r_i$$
$$x_{ij} \in \{0,1\} \qquad \forall j \in J_i,$$

où $KP_{J_i}(r_i)$ est le problème du sac à dos associé à la classe J_i de capacité r_i vérifiant $r_i \leq R \quad \forall i, i = 1, ..., m$.

En s'appuyant sur cette décomposition, les auteurs ont montré que toute solution optimale peut être représentée par un vecteur d'entiers positifs $r = (r_i, ..., r_m)$, avec $\sum_{i=1}^{m} r_i = R$. De ce fait, l'objectif consiste à trouver ce vecteur en utilisant une procédure de la programmation dynamique. L'algorithme commence par l'initialisation de toutes les valeurs r_i à zéro. Ensuite, il incrémente à chaque itération la capacité du problème auxiliaire $KP_{J_i}(r_i)$, $i = 1, ..., m$ dont la valeur de la fonction objectif est la plus petite. L'algorithme s'arrête lorsque la somme des différentes capacités r_i coïncide avec la capacité initiale R. L'algorithme 4 résume les étapes principales de l'algorithme de Hifi et Sadfi [11].

Algorithm 4 Un algorithme exact pour le KSP (Hifi et Sadfi [11])

Entrées: Une instance du KSP.
Sorties: La valeur de la solution optimale $VSO(KSP)$.

1: Étape 1 : Initialisation.
2: L'ensemble des classes est indexé de 1 à m c'est à dire $J_1, J_2, ..., J_m$
3: Les éléments de chaque classe J_i sont indexés 1 à $|J_i|$
4: Soit $\overline{r_i} := 0$ et $VSO(SK_{Ji}^{\overline{r_i}}) = 0$ pour $i = 1, ..., m$
5: Étape 2 : Principale
6: **Tant que** $\sum_{i=1}^{m} \overline{r_i} < R$ **faire**
7: Sélection de l'indice de classe pour laquelle la valeur minimale courante est réalisée
8: soit $ell := argmin\{VSO(SK_{J_i}^{\overline{r_i}}), i = 1, ..., m\}$
9: La capacité de la classe sélectionné est incrémentée $\overline{\ell} := \overline{\ell} + 1$
10: La procédure de la programmation dynamique est appliquée sur la classe ℓ
11: Résoudre Knapsack(ell, J_{ell}, $\overline{r_{ell}}$)
12: **fin du « Tant que »**
13: **Retourner** $VSO(KSP) = min\{VSO(SK_{J_i}^{\overline{r_i}}), i = 1, ..., m\}$;

Les auteurs ont montré que cet algorithme avait de meilleurs performance que celui proposé par Yamada *et al.* [21] tout en permettant de résoudre efficacement des instances de grande taille en temps un temps d'exécution moyen assez faible.

Une version améliorée de l'algorithme de Hifi et Sadfi [11] a été proposée par Hifi *et al.* [9]. Cette dernière est principalement basée sur une approche à deux étapes : (i) la première étape consiste à construire un ensemble d'éléments critiques et (ii) la deuxième étape permet de stabiliser cet ensemble d'éléments critiques. La partie principale de l'algorithme est aussi basée sur la technique de la programmation dynamique. Les calculs numériques de cette approche ont montré que cette version était plus performante que la version standard proposée dans Hifi et Sadfi [11].

1.3.2.4 Une méthode exacte basée sur une recherche dichotomique

Dans cette section, nous présentons une approche qui utilise un intervalle de recherche afin de résoudre le KSP à l'optimum. Cette méthode a été proposée par Hifi et Wu [12] qui s'appuie sur le concept de la recherche dichotomique. En premier lieu, la méthode commence par décomposer le problème initial en une série (cf. Hifi et Sadfi [11]) de problèmes de minimisation, notés $PA^i_{ksp},\ i = 1, ..., m-1$, et un problème de maximisation PA^m_{ksp} (cf. Brown [**?**]) de type sac à dos. Chacun des problèmes de minimisation $PA^i_{ksp},\ i = 1, ..., m$, peut être représenté de la manière suivante :

$$\begin{aligned} PA^i_{ksp}: \qquad & R_i = \min \sum_{j \in N_i} w_{ij} x_{ij} \\ s.c. \quad & \sum_{j \in N_i} c_{ij} x_{ij} \geqslant V \\ & x_{ij} \in \{0,1\} \qquad \forall j \in N_i, \end{aligned}$$

où V est une valeur prédéfinie. De la même manière, le problème de maximisation PA^m_{ksp} peut être écrit sous la forme suivante :

$$\begin{aligned} PA^m_{ksp}: \qquad & Z = \max \sum_{j \in N_m} c_{ij} x_{ij} \\ s.c. \quad & \sum_{j \in N_m} w_{ij} x_{ij} \leqslant R' \\ & x_{ij} \in \{0,1\} \qquad \forall j \in N_m, \end{aligned}$$

où $R' = R - \sum_{i=1}^{m-1} R_i$.

Le principe de cet algorithme consiste à s'approcher pas à pas de la meilleure solution en utilisant une recherche dichotomique. Soit $\underline{Z}$ et $\overline{Z}$, respectivement des bornes inférieure et supérieure de KSP, et V est une valeur

cible fixée à $[\frac{\underline{Z}+\overline{Z}}{2}]$. A chaque itération, l'algorithme tente de résoudre les $m-1$ premiers problèmes de minimisation, puis compléter la résolution du dernier problème (problème de maximisation). Si la valeur de la fonction objectif de PA^m_{ksp} (notée Z^*) est inférieure à V, alors $V-1$ est la nouvelle borne supérieure, sinon V est la nouvelle borne inférieure. La valeur de la solution optimale est atteinte lorsque $\underline{Z} = \overline{Z}$. Les principales phases de cet algorithme sont décrites par l'algorithme 5.

Algorithm 5 Un algorithme exact pour le KSP (Hifi et Wu [12])

Entrées: Une instance du KSP.
Sorties: La valeur de la solution optimale Z_{opt}.

1: Étape 1 : Initialisation.
2: Fixer $V = [\underline{Z} + \overline{Z}/2]$ ou $\underline{Z}$ est la borne inférieure et $\overline{Z}$ est la borne supérieure
3: Les éléments de chaque classe J_i sont indexés 1 à $|J_i|$
4: Soit $\overline{r_i} := 0$ et $VSO(SK^{\overline{r_i}}_{J_i}) = 0$ pour $i = 1, ..., m$
5: Étape 2 : Principale
6: **Tant que** $\overline{Z} > \underline{Z}$ **faire**
7: Résoudre les $m-1$ problèmes de minimisation (PA^i_{KSP}) et le problème de maximisation (PA^m_{KSP}) de sac à dos
8: Soit Z^* la solution optimale de PA^m_{KSP}
9: **si** $Z^* < V$ **alors**
10: $\overline{Z} = V - 1$
11: **si** $Z_* > \underline{Z}$ **alors**
12: $\underline{Z} = Z^*$
13: **finsi**
14: **sinon**
15: $\underline{Z} = V$
16: **si** $Z^* < \overline{Z}$ **alors**
17: $\overline{Z} = Z^*$
18: **finsi**
19: **finsi**
20: $V = [\underline{Z} + \overline{Z}/2]$
21: **fin du « Tant que »**
22: **Retourner** La valeur de la solution optimale Z_{opt} égale à $\underline{Z}$ ou $\overline{Z}$

De plus, une nouvelle borne supérieure a été introduite afin d'accélérer le processus de recherche. Cette borne est calculée d'une manière analogique à la borne supérieure de Dantzig lorsqu'elle est appliqué pour le KP, en résolvant

le problème $RPLNE_{KSP}$ obtenu par la relaxation du $PLNE_{KSP}$ (précédemment décrit dans la Section 1.3.1 et l'agrégation des contraintes 1.15). Le programme linéaire $RPLNE_{KSP}$ peut s'écrire de la manière suivante :

$$RPLNE_{KSP} \begin{cases} max \quad \gamma \\ sc \quad \sum_{i=1}^{m} \sum_{j \in N_i} w_{ij} x_{ij} \leq R, \\ \quad \sum_{i=1}^{m} \sum_{j \in N_i} c_{ij} x_{ij} > m \times \gamma, \\ x_{ij} \in [0,1], \quad \gamma \in N \quad j \in N_i, \quad i = 1, ..., m. \end{cases}$$

D'après l'étude expérimentale menée sur l'ensemble des instances de la littérature (Boyer *et al.* [2] et Hifi *et al.* [9]), cet algorithme était capable de résoudre l'ensemble des instances en un temps d'exécution moyen moins important, comparé aux temps moyens des algorithmes de la littérature.

1.4 Problème de la distribution équitable généralisé

Supposons que nous disposons d'une instance du KSP définie par un sac à doc de capacité R et un ensemble N d'éléments de cardinalité n qui sont répartis sur $m+1$ sous-ensembles disjoints. Si nous supposons qu'un sous-ensemble doit appartenir obligatoirement (sous certains contraintes) à tous les autres sous-ensembles (les m sous-ensembles), alors nous parlons du problème du KSP généralisé connu dans la littérature sous le nom du "*Generalized Knapsack Sharing Problem*" $-GKSP$. Ce problème possède de nombreuses applications dans le domaine du transport, les communications, l'allocation des ressources...etc. En effet, prenons l'exemple d'un distributeur de produits pharmaceutiques qui veut satisfaire la demande de ses clients. Le distributeur dispose d'une liste de demandes contenant des produits propres à chaque client et d'autres produits communs à tous les clients. Si le distributeur a un stock limité, alors il ne peut pas satisfaire toutes les commandes. Dans ce cas, le distributeur cherche à distribuer équitablement les produits sur ses clients pour les préserver.

Dans la section suivante, nous décrirons formellement le problème $GKSP$.

1.4.1 Description du problème $GKSP$

Une instance du problème $GKSP$ est définie par la donnée d'un sac de capacité limitée R et d'un ensemble d'éléments N. Cet ensemble est composé

de $m+1$ sous ensembles disjoints, où les m sous-ensembles, notés N_i, $i \in \{1,\ldots,m\}$, contiennent des éléments non-communs et le $(m+1)^{\text{ème}}$ sous-ensemble, noté N_0, contient des éléments communs à ces m sous-ensembles. Dans ce cas, une classe i est représentée par un sous-ensemble d'éléments non-communs N_i, $i \in \{1,\ldots,m\}$ et le sous-ensemble des éléments communs N_0. De plus chaque élément j d'une classe i est caractérisé par un poids w_{ij} et un profit c_{ij}.

Notons par x_{ij} la variable de décision à valeur binaire, où

$$x_{ij} = \begin{cases} 1 & \text{si l'élément } j \text{ de la classe } i \text{ est affecté au sac} \\ 0 & \text{sinon.} \end{cases}$$

L'objectif du problème est de déterminer le sous-ensemble d'objets de N à retenir dans le sac, qui satisfait la contrainte de capacité. Ce sous-ensemble est choisi de façon à maximiser la valeur de la fonction objectif réalisant le minimum par rapport à l'ensemble de toutes les classes. Le problème $GKSP$ peut être formulé comme suit :

$$P_{GKSP} \begin{cases} \max \quad \min\limits_{1\le i\le m} \{C^i(x)\} + C^0(x) \\ sc \quad \sum\limits_{i=0}^{m} \sum\limits_{j\in N_i} w_{ij}x_{ij} \le R, \\ \quad x_{ij} \in \{0,1\}, \quad j \in N_i, \quad i = 0,\ldots,m. \end{cases}$$

où $C^i(x) = \sum\limits_{j\in N_i} c_{ij}x_{ij}$ dénote le profit des éléments non-communs de la $i^{\text{ème}}$ classe et $C^0(x) = \sum\limits_{j\in N_0} c_{0j}x_{0j}$ est le profit des éléments communs.

Le programme linéaire en nombres entiers associé au P_{GKSP} peut s'écrire comme suit :

$$PLNE_{GKSP}: \quad \max \quad \gamma \tag{1.17}$$

$$s.c. \quad \sum_{j\in N_0} w_{0j}x_{0j} \le r_0 \tag{1.18}$$

$$\sum_{i=1}^{m} \sum_{j\in N_i} w_{ij}x_{ij} \le R - r_0 \tag{1.19}$$

$$\sum_{j\in N_i} c_{ij}x_{ij} > \gamma, \quad i = 1,...,m \tag{1.20}$$

$$x_{ij} \in \{0,1\},\ \gamma,\ r_0 \in N,\ j \in N_i, i = 1,...,m \tag{1.21}$$

où l'équation (1.18) (resp. (1.19)) représente la contrainte associée aux éléments communs (resp. non-communs) et les équations (1.20) caractérisent les contraintes liées aux profits.

A notre connaissance, très peu de travaux dans la littérature traitent de la résolution du $GKSP$. Parmi ces travaux de recherche, nous citons les travaux de Fujimoto et Yamada [6], où une méthode d'énumération exacte a été proposée. Cette méthode est basée sur l'énumération des valeurs discrètes représentant la capacité des deux problèmes $KP(r_{KP})$ et $KSP(r_{KSP})$. Pour chaque valeur discrète, la méthode utilise pour la résolution du KP l'algorithme de Horowitz et Sahni [13] et pour la résolution du KSP la méthode de Yamada *et al.* [21]. D'après les résultats numériques proposés par les auteurs, il a été souligné que cette méthode était très l'efficacité, en particulier pour les cas d'instances non-corrélées et faiblement corrélées.

Dans ce qui suit, nous décrirons la méthode de Fujimoto et Yamada [6] qui permet de donner une solution optimale pour le problème $GKSP$.

1.4.2 Une méthode exacte pour le $GKSP$

Yamada *et al.* [6] ont proposé une méthode exacte qui s'appuie sur une le principe de l'énumération exacte des solutions dans un espace de recherche bien encadré. Dans un premier temps, les auteurs ont montré que le $GKSP$ pouvait être décomposé en deux sous-problèmes : $KP(r_{kp})$ et $KSP(r_{ksp})$.

Supposons que la valeur optimale de la fonction objectif du $KP(r_{kp})$ (resp. $KSP(r_{ksp})$) est notée par $Z^*_{KP}(r_{kp})$ (resp. $Z^*_{KSP}(r_{ksp})$). Alors, une solution optimale pour le P_{GKSP} est équivalente à la valeur r_{kp} qui maximise le problème suivant :

$$\begin{aligned} P_{GKSP}: \quad & Z^*(r_{kp}) = Z^*_{kp}(r_{kp}) + Z^*_{ksp}(R - r_{kp}) \\ & s.c. \quad 0 \leq r_{kp} \leq R, \end{aligned}$$

où $R - r_{kp}$ est la capacité résiduelle.

L'idée principale de l'algorithme s'appuie sur l'énumération des valeurs r_{kp}, appelées des points de discontinuité, pour la résolution du P_{GKSP}. A chaque point de discontinuité, la méthode fait appel à l'algorithme de Horowitz et Sahni [13] pour la résolution du KP ainsi que l'algorithme de Yamada *et al.* [21] pour la résolution du KSP.

Afin de tenter la réduction de l'espace de recherche, la méthode ils ont développé une borne supérieure basée sur la relaxation de $KP(r_{kp})$ et $KSP(r_{ksp})$. En effet, par la résolution du $KP(r_{kp})$ et $KSP(r_{ksp})$ relaxés sur l'intervalle $[0\ R]$, ils ont obtenu une fonction $\overline{Z}(r)$ égale à la somme des valeurs des fonctions objectifs de ces deux problèmes. Cette fonction est concave et linéaire par morceaux, ce qui signifie que la borne supérieure représente exactement la valeur maximale $\overline{Z}(\overline{r})$ obtenue par la fonction $\overline{Z}(r)$. Ensuite, les auteurs ont pris comme borne inférieure la valeur $\underline{Z}$ obtenue par la résolution exacte du

$KP(\overline{r})$ et $KSP(\overline{r})$.

De plus, une réduction de l'intervalle de recherche a été effectuée pour accélérer la recherche en calculant les points d'intersection entre la fonction $\overline{Z}(r)$ et la valeur de la borne inférieure. L'intervalle résultant est noté par $[r_L \quad r_U]$. L'algorithme 6 montre les principales étapes de méthode.

Algorithm 6 Algorithme exact pour pour le $GKSP$ (Yamada *et al.* [6])

Entrées: $\underline{z}$, $[r_L, r_U]$.
Sorties: La valeur de la solution optimale de $GKSP$ z^*.

A chaque point de discontinuité $r \in [r_L, r_U]$ de $z^*_{kp}(.)$
Étape 1 : Résoudre KP(r) à l'optimum et calculer $z^*_{kp}(r)$
Les éléments de chaque classe J_i sont indexés 1 à $|J_i|$
Étape 2 :
si $z^*_{ksp}(r) + z^*_{kp}(r) < \underline{z}$ **alors**
    Aller à l'étape 4
finsi
Étape 3 : Mettre à jour la borne inférieure par $\underline{z} = z^*_{ksp}(r) + z^*_{kp}(r)$
Étape 4 : Aller au point de discontinuité r suivant
Retourner $z^* = \underline{z}$

1.5 Variantes du problème de sac à dos

La famille de type sac à dos est une classe de problèmes contenant un nombre important de variantes. Dans cette section, nous nous intéressons particulièrement à trois variantes qui peuvent être considérées comme des extensions directes du sac à dos classique KP. Nous allons donc formellement les décrire dans ce qui suit.

1.5.1 Problème du sac à dos avec contraintes multiples

Le problème du sac à dos avec des contraintes multiples ("*Multiple Knapsack Problem*" $-MKP$) est une variante du problème du sac à dos classique KP. Dans ce problème, nous disposons de m sacs à dos de différentes (ou de même) capacité(s) $R_i,\ i = 1, \ldots, m$. Le but du problème consiste à remplir tous les sacs de façon à maximiser le profit total et de sorte que la somme des poids dans chaque sac d'indice i ne dépasse pas sa capacité $R_i,\ i = 1, \ldots, m$.

Formellement, le MKP peut s'écrire comme suit :

$$MKP: \quad \max \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_j x_{ij} \tag{1.22}$$

$$\text{s.c.} \quad \sum_{j=1}^{n} w_j x_{ij} \leq R_i, \quad i = 1, \ldots, m \tag{1.23}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \quad j = 1, ..., n, \tag{1.24}$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n, \tag{1.25}$$

où x_{ij} est une variable décision qui vaut 1 si l'élément j est inclus dans le i-ème sac, 0 sinon. Les contraintes (1.23) assurent que le remplissage de chaque contrainte caractérisant un sac ne dépasse pas sa capacité R_i, et les contraintes (1.24) assurent que chaque élément est choisi au plus une fois. Nous supposons que les coefficients c_j, w_j et R_i sont des valeurs entières positives.

1.5.2 Problème du sac à dos généralisé à choix multiple

Le problème de sac à dos à choix multiple et multidimensionnel ("*Multidimensional Multiple-choice Knapsack Problem*" $-MMKP$) est aussi considéré comme une variante du problème de sac à dos classique avec des contraintes spécifiques. Une instance du problème $MMKP$ est représentée par un vecteur de m capacité (ressources disponibles) $R = (R^1, R^2, ..., R^m)$ et un ensemble N contenant n d'éléments répartis sur k classes disjointes : $N_i,\ i \in \{1, ..., k\}$, où $|N_i| = n_i$ (n_i est le nombre d'éléments de la classe N_i). A chaque élément j de la classe i est associé un profit c_{ij} et un vecteur poids $W_{ij} = (w_{ij}^1, w_{ij}^2, ..., w_{ij}^m)$. Le but du problème est d'attribuer au sac exactement un et un seul objet par classe tout en maximisant la valeur totale du choix, en tenant en compte des contraintes de capacité.

D'une façon formelle, le problème MMKP peut s'écrire de la façon suivante :

$$MMKP: \quad \max \quad \sum_{i=1}^{k}\sum_{j=1}^{n_i} c_{ij} x_{ij} \tag{1.26}$$

$$\text{s.c.} \quad \sum_{i=1}^{k}\sum_{j=1}^{n_i} w_{ij}^{\ell} x_{ij} \leq R^{\ell},\ \ell = 1, \ldots, m \tag{1.27}$$

$$\sum_{j=1}^{n_i} x_{ij} = 1, \quad i = 1, \ldots, k \tag{1.28}$$

$$x_{ij} \in \{0, 1\}, \quad j = 1, ..., n_i, \quad i = 1, \ldots, k, \tag{1.29}$$

où x_{ij} représente la variable de décision qui vaut 1 si l'élément j de la $i^{\text{ème}}$ classe est dans le sac à dos, 0 dans le cas contraire. Les inégalités (1.27) représentent les contraintes de capacité. Les équations (1.28), appelées *contraintes de choix*, assurent qu'un seul élément est sélectionné de chaque classe.

Sans perte de généralité et pour éviter les solutions triviales, nous formulons l'hypothèse suivante :

$$\sum_{i=1}^{k} \min_{1\leq j\leq n_i} \{w_{ij}^{\ell}\} \leq R^{\ell} \leq \sum_{i=1}^{k} \max_{1\leq j\leq n_i} \{w_{ij}^{\ell}\},\ \ell = 1,\ldots,m.$$

1.5.3 Problème de sac à dos avec contraintes disjonctives

Le problème du sac à dos avec des contraintes disjonctives (*Disjunctively Constrained Knapsack Problem*" $-DCKP$) est une autre variante du problème de sac à dos classique KP. Le DCKP contient deux types de contraintes : le premier type représente la contrainte classique de capacité, et le deuxième représente les contraintes disjonctives. Deux éléments sont dans le sac à dos s'ils sont deux à deux compatibles. Autrement dit, deux éléments ne sont pas compatibles s'ils ne peuvent pas participer à la même solution c'est à dire le choix d'un élément exclut l'autre l'élément.

Le DCKP est caractérisé par la donnée d'un ensemble de n éléments et par une capacité R du sac. A chaque objet j, $j \in \{1,...,n\}$, est associé un profit c_j et un poids w_j . De plus, certains couples d'éléments (i,j) peuvent être incompatibles, dans le sens où l'appartenance de l'un des éléments à la solution, exclut l'autre élément. Notons par B l'ensemble des couples d'éléments non compatibles, donc $B = \{(i,j) \mid i$ n'est pas compatible avec $j\}$.

Le but du problème consiste à choisir le sous-ensemble d'objets à mettre dans le sac à dos tel que la contrainte de capacité et les contraintes disjonctives soient satisfaites, et que le profit cumulé soit de valeur maximale. Le poblème peut être formulé comme suit :

$$P_{DCKP}: \quad \max \quad \sum_{j=1}^{n} c_j x_j \tag{1.30}$$

$$s.c. \quad \sum_{j=1}^{n} w_j x_j \leq R \tag{1.31}$$

$$x_i + x_j \leq 1, \quad \forall (i,j) \in B \tag{1.32}$$

$$x_j \in \{0,1\}, \quad j = 1,\ldots,n, \tag{1.33}$$

où x_j est la variable de décision associée à l'objet d'indice j telle que x_j vaut 1 si l'objet j est sélectionné, 0 sinon. L'équation (1.31) représente la contrainte

de sac à dos et les équations (1.32) sont les contraintes disjonctives, où un seul objet peut être sélectionné parmi un ensemble de paires d'objets.

Il existe d'autres problèmes appartenant à la famille de sac à dos comme, le problème du sac à dos quadratique, le problème du sac à dos multi-objectif ainsi que d'autres. Chaque variante est différente des autres par la caractérisation des éléments et / ou la distribution des objets dans une ou plusieurs sacs (pour plus de détails, le lecteur peut se reporter à Martello *et al.* [18]).

1.6 Conclusion

Dans ce chapitre, nous avons présenté certains problèmes de type sac à dos, parmi les plus étudiés de cette famille. Ces problèmes représentent des variantes du problème du sac à dos dit classique. Ils sont souvent représentés par des modifications sur la fonction objectif, des contraintes liées à la capacité d'un ou plusieurs sac ou encore l'ajout d'autres contraintes qui parfois compliquent la problématique. En parallèle, nous avons décrit quelques méthodes exactes et approchées de la littérature conçues pour la résolution du problème de sac à dos classique. Ensuite, nous avons présenté deux variantes du sac à dos, à savoir, le problème de la distribution équitable et le problème de la distribution équitable généralisé. Pour ces deux problématiques, nous avons décrit les principales méthodes de résolutions de la littérature. Finalement, afin de montrer la richesse de la famille des problèmes de type sac à dos, nous avons aussi évoqué trois autres variantes du sac à dos.

Chapitre 2

Une méthode diversifiante pour le problème de la distribution équitable

Dans ce chapitre nous proposons un algorithme approché pour la résolution du problème de la distribution équitable (connu sous le nom du "*Knapsack Sharing Problem*" $-KSP$). La méthode s'appuie sur une recherche diversifiante / dispersée ("*Scatter Search*"). Le principe de la méthode consiste en premier lieu à construire une solution réalisable de départ. Ensuite, en se basant sur cette solution, l'idée est de construire une population de solutions obtenues autour de l'élément critique associé à un problème de sac à dos classique. Afin d'explorer différentes parties de l'espace de recherche, une profondeur autour de divers éléments critiques est appliquée. Finalement, une série de combinaisons entres les éléments de la population est introduite afin de rafraîchir la population courante tout en préservant certaines solutions élites.

Les travaux de recherche de ce chapitre ont fait l'objet d'une présentation et d'une publication sous le titre : *A Scatter Search-Based Algorithm for the Knapsack Sharing Problem*, dans une conférence internationale ≪ the Second International Symposium on Operational Research 2011≫.

2.1 Résolution du KSP par application d'une recherche dispersée

2.1.1 Décomposition du problème KSP

Dans cette section, nous utilisons une décomposition similaire à celle utilisée dans Hifi et Sadfi [11]. Cette décomposition permet de représenter le KSP sous forme d'une série de sous problèmes de type sac à dos classique KP. Les auteurs ont prouvé que chacun des sous-problèmes KPs, noté $KP_{J_i}(r_i),\ i = 1, \ldots, m$, dénote un problème auxiliaire du problème original KSP. La série

des knapsacks peut être représentée comme suit :

$$KP_{J_i}(r_i: \quad \max \quad Z(x) = \sum_{j \in J_i} c_{ij} x_{ij}$$
$$s.c. \quad \sum_{j \in J_i} w_{ij} x_{ij} \leq r_i$$
$$x_{ij} \in \{0,1\} \qquad \forall j \in J_i,$$

où r_i est une valeur entière positive vérifiant $r_i \leq R,\ \forall i \in \{1,\ldots,m\}$ et $KP_{J_i}(r_i)$ représente le problème du sac à dos classique associé à la classe J_i de capacité r_i.

2.1.2 Les éléments critiques du KSP

Nous rappelons qu'un problème de sac à dos en 0-1 est caractérisé par son élément critique (voir Section 2 du Chapitre 1). Dans notre étude, nous utilisons cette notion pour déterminer le vecteur des éléments critiques du KSP. En effet, nous définissons d'une manière analogique les éléments critiques et cela en considérant la relaxation continue de chacune des variables de décision, i.e., $0 \leq x_j \leq 1,\ j \in J_i,\ i = 1,\ldots,m$).

Ainsi chaque problème auxiliaire $KP_{J_i}(r_i),\ i = 1,\ldots,m$ dispose d'un élément critique défini de la manière suivante :

$$\sum_{k_{J_i}=1}^{e_{J_i}-1} w_{k_{J_i}} \leq r_i \tag{2.1}$$

$$\sum_{k_{J_i}=1}^{e_{J_i}} w_{k_{J_i}} > r_i \tag{2.2}$$

$$\sum_{i=1}^{m} r_i \leq R. \tag{2.3}$$

L'équation (2.1) signifie que tous les éléments d'indice inférieur à l'indice de l'élément critique e_{J_i} sont fixés à un et dans ce cas la solution obtenue est réalisable. Dans l'équation (2.2), la solution devient non réalisable une fois l'élément critique e_{J_i} est fixé à un. La dernière équation (2.3) signifie que la solution reste réalisable pour le KSP tant que la contrainte de capacité est vérifiée.

La figure 2.1 représente une configuration d'une solution réalisable du KSP. Par exemple : l'indice 3 dans la première classe représente l'indice de l'élément critique et il est égale à 4 pour le $2^{\text{ème}}$ classe et ainsi de suite. Nous

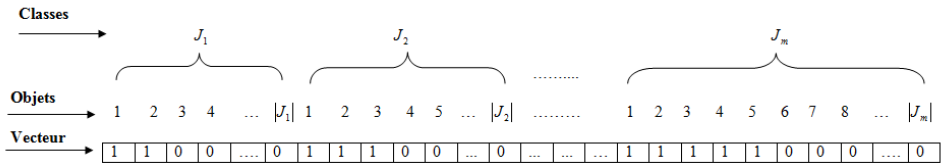

FIGURE 2.1 – Représentation d'une solution réalisable pour le KSP

remarquons que les éléments qui se trouvent à gauche de l'élément critique de chaque classe sont fixés à un et les autres à zéro. Partant de ce principe, nous pouvons représenter une solution réalisable du KSP par un vecteur des éléments critiques de taille m.

2.1.3 Construction d'une solution réalisable de départ

La solution initiale du KSP peut être obtenue par la recherche des éléments critiques de toutes les classes en fixant les éléments qui se trouvent à gauche de l'élément critique à la valeur 1 et les éléments qui se trouvent à droite de cet élément à la valeur 0. Cette idée peut être développée par la construction d'une heuristique gloutonne (noté Heur) de la manière suivante :

1. Ordonner les éléments de chaque classe selon l'ordre décroissant du rapport profit par poids.
2. Sélectionner la classe de profit cumulé minimum et fixer son élément à un si son ajout ne dépasse pas la capacité du sac.
3. Réitérer le processus jusqu'à ce que l'ajout d'un élément excède la capacité totale du sac.

L'initialisation de l'algorithme est effectuée à l'étape 1 telle que : la capacité cumulée est fixé à zero, l'indice de la classe qui réalise le minimum est égale à un, le profit et le poids cumulés de chaque classe sont fixés à 0.

Les étapes de l'algorithme glouton sont données par l'algorithme 7. A chaque itération (la boucle principale : étape 2), un élément de la classe de profit minimal est ajoutée au sac si sa capacité est inférieure ou égale à la capacité résiduelle. Le processus est réitéré jusqu'à épuisement de tous les éléments pouvant être mis dans le sac.

Exemple : Soit une instance du KSP composée de 16 éléments répartis sur 2 classes et un sac de capacité 221. Le modèle mathématiques de cette

Algorithm 7 Algorithme glouton pour le *KSP Heur*

Entrées: Une instance du KSP.
Sorties: Solution réalisable.

1: **Étape 1**
2: Mettre la capacité initiale à zéro c-à-d capacité cumulative totale notée $Somme\_Cap = 0$;
3: Soit min l'indice de la classe réalisant le profit minimum au niveau actuel de la solution, $min = 1$
4: Pour chaque classe $i \in 1, ..., m$, l'élément de la classe i : $j_i = 1$, le profit (resp. poids) cumulé des éléments sélectionnés dans la classe i : $C_i = 0$ (resp. $W_i = 0$),
5: **Étape 2**
6: **répéter**
7: **si** $Somme\_Cap + w_{j_{min}} \leq R$ **alors**
8: $Somme\_Cap = Somme\_Cap + w_{j_{min}}$;
9: $j_{min} = j_{min} + 1$;
10: **finsi**
11: Soit min l'indice réalisant $min_{1 \leq i \leq m}\{C_i\}$;
12: **jusqu'à** $jmin >\mid J_{min} \mid$.
13: **Retourner** Solution réalisable ;

instance est donné comme suivant :

$$KSP \begin{cases} \max \quad \mathbf{z(x)} := \min\Big\{14x_1 + 90x_2 + 78x_3 + 94x_4 + 4x_5 + 20x_6 + 29x_7 + 32x_8 + 95x_9 + 47x_{10}, \\ \qquad\qquad 40x_{11} + 69x_{12} + 84x_{13} + 88x_{14} + 63x_{15} + 89x_{16}\Big\} \\ \mathbf{s.t.} \quad 21x_1 + 28x_2 + 7x_3 + 17x_4 + 10x_5 + 37x_6 + 31x_7 + 22x_8 + 37x_9 + 44x_{10} + 7x_{11} \\ \qquad\qquad +31x_{12} + 47x_{13} + 28x_{14} + 41x_{15} + 35x_{16} \leq 221 \\ x_j \in \{0, 1\},\ j \in \{1, 2, .., 16\} \end{cases},$$

La table 2.1 (resp. table 2.2) représente les profits et les poids des éléments de la première (resp. deuxième) classe. Ces éléments sont triés dans l'ordre décroissant du rapport profit par poids.

La solution réalisable de départ est représentée par le vecteur $Y = (y_j)$, $j = 1, ..., 16$ comme le montre les deux tables 2.3 et 2.4 :

Le vecteur des éléments critiques est $(y_5,\ y_{15})$ et la valeur de la fonction objectif est $z = \min\{z_1\ ;\ z_2\} = \min\{357,\ 286\} = 286$.

TABLE 2.1 – Trie des variables de la première classe selon l'ordre décroissant de c/w

Classe 1	x_3	x_4	x_2	x_9	x_8	x_{10}	x_7	x_1	x_6	x_5
Profit	78	94	90	95	32	47	29	14	20	4
Poids	7	17	28	37	22	44	31	21	37	10

TABLE 2.2 – Trie des variables de la deuxième classe selon l'ordre décroissant de c/w

Classe 2	x_{11}	x_{14}	x_{16}	x_{12}	x_{13}	x_{15}
Profit	40	88	89	69	84	63
Poids	7	28	35	31	47	41

TABLE 2.3 – Solution réalisable de la première classe

Classe 1	x_3	x_4	x_2	x_9	x_8	x_{10}	x_7	x_1	x_6	x_5
Classe 1	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}
Solution	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**

TABLE 2.4 – Solution réalisable de la deuxième classe

Classe 2	x_{11}	x_{14}	x_{16}	x_{12}	x_{13}	x_{15}
Classe 2	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}	y_{16}
Solution	**1**	**1**	**1**	**1**	**0**	**0**

2.1.4 Les principale étapes de l'algorithme

Dans cette partie, nous allons donner une description de la recherche dispersée utilisée pour la résolution du KSP. Cette recherche est basée principalement sur différentes stratégies, dites méthodes. Dans un premier temps, nous décrirons ces différentes stratégies et nous présentation l'aspect général de l'algorithme dans un deuxième temps.

2.1.5 Une méthode pour la génération d'une population diversifiante

La méthode de génération de diversification permet de générer une population P de diverses solutions. Cette dernière est construite à partir d'une

solution réalisable de départ, où dans notre cas l'algorithme glouton Heur est appliqué pour réaliser cette solution. Cette population est réalisée par application de la notion de profondeurs sur l'élément critique.

Sachant qu'une solution du KSP peut être représentée par un vecteur d'éléments critiques, pour générer la population, nous procédons par déplacer l'élément critique de chaque classe d'une unité à gauche ou d'une unité à droite. En premier lieu, nous déplaçons l'élément critique à gauche pour construire une profondeur à gauche afin de garantir un certain nombre de voisins. Ensuite, nous déplaçons cet élément à droite pour construire une profondeur à droite permettant ainsi de diversifier les solutions. Ce processeur peut être décrit de la manière suivante :

- Profondeur à gauche : Pour chaque classe $J_i,\ i = 1, \ldots, m$, nous déplaçons l'élément critique d'une unité à gauche. Dans ce cas, le nouveau élément critique $e_{J_i} - 1$ est fixé à zéro.
- Profondeur à droite : Pour chaque déplacement à gauche nous introduisons plusieurs déplacements à droite. Ainsi, nous obtenons une population avec des solutions diversifiées.

Afin de limiter le voisinage des solutions, nous avons fixé le déplacement à gauche et à droite à un certain nombre (ce nombre est discuté dans la partie expérimentale).

Exemple : Pour illustrer ce principe considérons la solution réalisable de l'exemple décrit précédemment. D'après la table 2.5) le générateur construit 8 solutions. Chaque ligne dans cette table correspond à une variable et chaque colonne à une solution. La dernière ligne représente la valeur de la fonction objectif.

2.1.6 Une méthode d'amélioration

Cette méthode tente d'améliorer les solutions générées par la méthode de diversification (précédemment décrite et utilisée pour la génération de la population de départ). Dans notre étude, nous utilisons l'algorithme glouton Heur (décrit dans la section 2.1.3) pour compléter et tenter d'améliorer la solution.

Exemple : D'après la table 2.6 ci-dessous, nous remarquons que les solutions $S1$ et $S3$ sont améliorées. La valeur de la fonction objectif de $S1$ passe de 286 à 301 et la valeur de la fonction objectif de $S3$ s'améliore de 294 à 301.

TABLE 2.5 – Population diversifiante

Variables	$S1$	$S2$	$S3$	$S4$	$S5$	$S6$	$S7$	$S8$
y_1	1	1	1	1	1	1	1	1
y_2	1	1	1	1	1	1	0	0
y_3	1	1	1	0	0	0	1	1
y_4	0	1	0	1	1	1	1	1
y_5	1	0	1	1	1	1	1	1
y_6	0	0	0	0	0	0	0	0
y_7	0	0	0	0	0	0	0	0
y_8	0	0	0	0	0	0	0	0
y_9	0	0	0	0	0	0	0	0
y_{10}	0	0	0	0	0	0	0	0
y_{11}	1	1	1	1	1	1	1	1
y_{12}	1	1	1	1	1	0	1	0
y_{13}	1	1	1	1	0	1	1	1
y_{14}	1	0	0	1	1	1	1	1
y_{15}	0	1	1	0	1	1	0	1
y_{16}	0	0	0	0	0	0	0	0
Z	286	301	294	286	281	282	286	282

2.1.7 Une méthode de mise à jour de l'ensemble référent

L'ensemble référent est un ensemble (noté ***RefSet***) constitué de b solutions. Ces solutions sont choisies parmi la population. Une solution appartient à l'ensemble référent ***RefSet*** si elle fait partie de l'un des deux sous-ensembles suivants :

- Le premier sous-ensemble des solutions élites qui comporte b_1 solutions. Elles constituent les meilleures solutions de la population selon leurs évaluations (valeurs de la fonction objectif associée à chaque solution).
- Le deuxième sous-ensemble est composé de b_2 solutions prises des solutions de la population restante $P - b_1$. Elles sont les plus éloignées des meilleures solutions de b qui représentent des solutions diversifiées.

La diversité d'une solution est mesurée par la distance de Hamming de la manière suivante :

$$d_{min}(X) = \min_{Y \in RefSet} \Big\{ \sum_i |x_i - y_i| \Big\} / X \in P - RefSet \tag{2.4}$$

où $X = (x_i)_i$ et $Y = (y_i)_i$.

TABLE 2.6 – Une population après une amélioration

Variables	$S1$	$S2$	$S3$	$S4$	$S5$	$S6$	$S7$	$S8$
y_1	1	1	1	1	1	1	1	1
y_2	1	1	1	1	1	1	0	0
y_3	1	1	1	0	0	0	1	1
y_4	0	1	0	1	1	1	1	1
y_5	1	0	1	1	1	1	1	1
y_6	0	0	0	0	0	0	0	0
y_7	0	0	0	0	0	0	0	0
y_8	0	0	**1**	0	0	0	0	0
y_9	0	0	0	0	0	0	0	0
y_{10}	0	0	0	0	0	0	0	0
y_{11}	1	1	1	1	1	1	1	1
y_{12}	1	1	1	1	1	0	1	0
y_{13}	1	1	1	1	0	1	1	1
y_{14}	1	0	0	1	1	1	1	1
y_{15}	0	1	1	0	1	1	0	1
y_{16}	**1**	0	0	0	0	0	0	0
Z	**294**	301	**301**	286	281	282	286	282

Nous supposons dans ce qui suit que l'ensemble référent est organisé de manière à ce que les solutions soient triées selon l'ordre décroissant de leur valeurs.

Exemple : l'ensemble référent est constitué à partir de la population diversifiante donnée précédemment. La table 2.7 correspond au solutions élites et les solutions diversifies choisies parmi les solutions de la population.

L'ensemble référent est constitué de 6 solutions ordonnées selon l'ordre décroissant de leur valeurs de la fonction objectif : $ReSet = \{S2, S3, S1, S4, S8, S5\}$.
.

2.1.8 Une méthode de génération des sous-ensembles

Une méthode de génération des sous-ensembles consiste à construire des sous-ensembles non répétés mais ils peuvent partager des solutions à partir de l'ensemble de référence $RefSet$. Ces sous-ensembles seront fournis en entrée pour la méthode de combinaison.

Une méthode de génération a été présentée par Glover [8] pour construire les sous-ensembles avec des propriétés utiles et qui évitera la duplication des

TABLE 2.7 – Ensemble référent

	Meilleures solutions			Solutions diversifiées		
Variables	$S2$	$S3$	$S1$	$S5$	$S8$	$S4$
y_1	1	1	1	1	1	1
y_2	1	1	1	1	0	1
y_3	1	1	1	0	1	0
y_4	1	0	0	1	1	1
y_5	0	1	1	1	1	1
y_6	0	0	0	0	0	0
y_7	0	0	0	0	0	0
y_8	0	1	0	0	0	0
y_9	0	0	0	0	0	0
y_{10}	0	0	0	0	0	0
y_{11}	1	1	1	1	1	1
y_{12}	1	1	1	1	0	1
y_{13}	1	1	1	0	1	1
y_{14}	0	0	1	1	1	1
y_{15}	1	1	0	1	1	0
y_{16}	0	0	1	0	0	0
Z	301	301	294	281	282	286

sous-ensembles précédemment générés. Cette méthode permet de construire quatre types de sous-ensembles $E_i,\ i = 1, \ldots, 4$, où

- E_1 : sous-ensembles de type 1 contenant tous les sous-ensembles distincts ayant deux éléments, si le cardinal de l'ensemble de référence est b, alors cela produira $b * (b-1)/2$ sous-ensembles différents.
- E_2 : sous-ensembles de type 2 contenant les sous-ensembles à trois éléments générés à partir des sous-ensembles à deux éléments auxquels on rajoute la meilleure solution qui n'appartient pas à ces derniers.
- E_3 : sous-ensembles de type 3 contenant les sous-ensembles à quatre éléments issus des sous-ensembles à trois éléments augmentés d'une meilleure solution de l'ensemble référent qui n'est pas inclue dans ces derniers.
- E_4 : sous-ensembles de type 4 contenant les sous-ensembles qui sont constitués des i meilleures solutions, pour $i = 5$ jusqu'à b (b étant la taille de l'ensemble de référent).

Exemple : les quatre type des sous ensembles sont générés a partir de l'ensemble référent et il sont donnés comment suit :
Sous-ensembles de Type1

$E_1 = \{\{s_2, s_3\}, \{s_2, s_1\}, \{s_2, s_4\}, \{s_2, s_8\}, \{s_2, s_5\}, \{s_3, s_1\}, \{s_3, s_4\}, \{s_3, s_8\}, \{s_3, s_5\}, \{s_1, s_4\}, \{s_1, s_8\}, \{s_1, s_5\}, \{s_4, s_8\}, \{s_4, s_5\}, \{s_8, s_5\}\}$

Sous-ensembles de Type2
$E_2 = \{\{s_2, s_3, s_1\}, \{s_2, s_1, s_4\}, \{s_2, s_4, s_8\}, \{s_2, s_8, s_5\}, \{s_3, s_1, s_4\}, \{s_3, s_4, s_8\}, \{s_3, s_8, s_5\}, \{s_1, s_4, s_8\}, \{s_1, s_8, s_5\}\{s_4, s_8, s_5\}\}$

Sous-ensembles de Type3
$E_3 = \{\{s_2, s_3, s_1, s_4\}, \{s_2, s_1, s_4, s_8\}, \{s_2, s_4, s_8, s_5\}, \{s_3, s_1, s_4, s_8\}, \{s_3, s_4, s_8, s_5\}, \{s_1, s_4, s_8, s_5\}\}$

Sous-ensembles de Type4
$E_4 = \{\{s_2, s_3, s_1, s_4, s_8\}, \{s_2, s3, s_1, s_4, s_8, s_5\}\}$

Afin de déterminer les sous-ensembles, deux stratégies peuvent être utilisées : une gestion statique ou une gestion dynamique de l'ensemble référent (Glover [8]). Une gestion statique des sous-ensembles consiste à générer à chaque itération tous les sous-ensembles associés à $RefSet$.

L'algorithme 8 décrit cette approche pour les quatre types des sous-ensembles. Nous notons par $RefSet[i]$ la solution placée à la i^{ime} position dans $RefSet$, $sols$ est un sous-ensemble de solutions et Se_k, k=1,2,3,4, représente l'ensemble des solutions de type $E1$, $E2$, $E3$ ou $E4$.

L'algorithme est constitué en premier lieu de deux boucles imbriquées permettant de générer tous les sous-ensembles possibles. Les étapes de 3 à 5 permettent de construire l'ensemble Se_1 des sous-ensembles de type E_1. Ensuite, afin de construite les ensembles Se_2 de type E_2, l'ensemble Se_1 est augmenté d'une solution (étapes de 6 à 9). De la même façon, l'ensemble Se_2 est augmenté d'une solution pour construire l'ensemble Se_3 de type E_3 (étapes de 10 à 13). Finalement, la dernière boucle (étapes de 19 à 22) permet de construire l'ensemble Se_4 de type E_4 contenant de cinq jusqu'à b solutions.

Avec cette approche le nombre de sous-ensembles est relativement important, et cela est dû au fait que les sous ensembles peuvent être régénérés à chaque itération. En effet, lorsqu'un sous-ensemble est obtenu au cours de la recherche, il n'y a pas d'intérêt à le créer de nouveau par la suite puisque la solution obtenue par la combinaison sera toujours la même.

Afin de réduire le nombre des sous-ensemble générés et ainsi accélérer la recherche, une gestion dynamique de la population des solutions références est utilisée dans l'algorithme 9. Cette stratégie repose sur un matrice (notée "Comb") qui permet de gérer les couples de solutions (i, j) de $RefSet$. Autrement dit, si le couple (i, j) a été utilisé pour la création d'une (ou plu-

Algorithm 8 Génération des sous ensembles dans une version statique

Entrées: la population $RefSet$.
Sorties: Sous-ensembles Se.

pour i allant de 1 à $b-1$ **faire**
  pour j allant de $i+1$ à b **faire**
    Sous ensembles de Type 1
    $sols[1] = RefSet[i]$; $sols[2] = RefSet[j]$;
    $Se_1 = \{sols[1], sols[2]\}$;
    si $j+1 \leq b$ **alors**
      Sous ensembles de Type 2
      $sols[3] = RefSet[j+1]$;
      $Se_2 = \{sols[1], sols[2], sols[3]\}$;
      si $j+2 \leq b$ **alors**
        Sous ensembles de Type 3
        $sols[4] = RefSet[j+2]$;
        $Se_3 = \{sols[1], sols[2], sols[3], sols[4]\}$;
      finsi
    finsi
  fin pour
fin pour
Sous ensembles de Type 4
pour i allant de 1 a b **faire**
  $sols[i] = RefSet[i]$;
  si $i \geq 5$ **alors**
    $Se_4 = \{sols[1], sols[2], sols[3], ..., sols[i]\}$;
  finsi
fin pour
Retourner Sous-ensembles Se.

sieurs) solution avec la méthode de combinaison des sous ensembles, alors "Comb[i][j]" reçoit la valeur zéro pour éviter la reconstruction d'une même solution. Ajoutons que d'un point de vue algorithmique, la matrice "Comb" est initialement remplie par des "un" lors du premier appel de l'algorithme.

2.1.9 Une méthode combinant des solutions

Cette méthode consiste à définir les procédures permettant de créer les nouvelles solutions à partir d'un sous-ensemble généré par les quatre types précédemment décrits. Nous utilisons deux types de combinaisons dans notre

Algorithm 9 Génération des sous ensembles en version dynamique

Entrées: la population $RefSet$.
Sorties: Sous-ensembles Se.

pour i allant de 1 à b-1 **faire**
  pour j allant de i+1 à b **faire**
    si $comb[i][j] = 1$ **alors**
      $comb[i][j] = 0$;
      Sous ensembles de Type 1
      $sols[1] = RefSet[i]$; $sols[2] = RefSet[j]$;
      $Se_1 = \{sols[1], sols[2]\}$;
      si $j + 1 \leq b$ **alors**
        Sous ensembles de Type 2
        $sols[3] = RefSet[j + 1]$;
        $Se_2 = \{sols[1], sols[2], sols[3]\}$;
        si $j + 2 \leq b$ **alors**
          Sous ensembles de Type 3
          $sols[4] = RefSet[j + 2]$;
          $Se_3 = \{sols[1], sols[2], sols[3], sols[4]\}$;
        finsi
      finsi
    finsi
  fin pour
fin pour
Sous ensembles de Type 4
pour i allant de 1 a b **faire**
  $sols[i] = RefSet[i]$;
  si ($i \geq 5$ et $comb4[i - 4] = 1$) **alors**
    comb4[i-4]=0 ;
    $Se_4 = \{sols[1], sols[2], sols[3], ..., sols[i]\}$;
  finsi
fin pour
Retourner Sous-ensembles Se.

étude. La première combinaison s'appuie sur une combinaison linéaire entre les solutions du sous-ensemble. Elle repose sur la notion de score utilisée par Laguna *et al.* [15]. L'idée principale est de calculer le score de chaque élément de la manière suivante :

$$score(j) = \frac{\sum_{x \in S} Z(x) * x_j}{\sum_{x \in S} Z(x)},$$

où $Z(x)$ est la valeur de la fonction objectif de la solution x.

Ensuite, l'élément j est fixé a "un" si son score est supérieur à un certain seuil (nous utiliserons la valeur 0.5 dans notre étude) et à zéro sinon, c'est-à-dire :

$$x(j) = \begin{cases} 1 & si \quad score(j) > 0.5 \\ 0 & si \quad sinon \end{cases}$$

Nous notons que ce genre de combinaison n'est pas efficace si la taille du sous-ensemble est petite. Dans notre étude, nous utilisons cette combinaison pour les types E_2, E_3 et E_4.

Exemple : illustration de la combinaison score. Nous choisissons le sous-ensemble de type 2 composé des trois solutions $\{s_2, s_3, s_1\}$ pour produire une nouvelle solution. Les scores (colonne 5) obtenus pour les 16 variables du problème sont donnés dans la table 2.8 suivante :

TABLE 2.8 – Combinaison score entre trois solutions

Variables	$S2$	$S3$	$S1$	Score	Nsol	**Amélioration**
y_1	1	1	1	1	1	**1**
y_2	1	1	1	1,00	1	**1**
y_3	1	1	1	1,00	1	**1**
y_4	1	0	0	0,34	0	**0**
y_5	0	1	1	0,66	1	**1**
y_6	0	0	0	0	0	**0**
y_7	0	0	0	0	0	**0**
y_8	0	1	0	0,34	0	**1**
y_9	0	0	0	0	0	**0**
y_{10}	0	0	0	0	0	**0**
y_{11}	1	1	1	1	1	**1**
y_{12}	1	1	1	1,00	1	**1**
y_{13}	1	1	1	1,00	1	**1**
y_{14}	0	0	1	0,33	0	**0**
y_{15}	1	1	0	0,67	1	**1**
y_{16}	0	0	1	0,33	0	**0**
Z	301	301	294		294	**301**

La colonne 6 correspond la nouvelle solution $Nsol$ après la combinaison et

ligne 7 représente cette solution après l'amélioration.

Algorithm 10 Méthode de combinaison entre deux solutions

Entrées: Deux solutions x_1 et x_2 et leurs évaluations $z(x_1)$ et $z(x_2)$.
Sorties: La meilleure solution x^* distincte de x_1 et x_2.

1: Poser $Z(x*) = -\infty$
2: **si** $Z(x_1) > Z(x_2)$ **alors**
3: $y = x_1$
4: **sinon**
5: $y = x_2$
6: Soit Res la capacité résiduelle du sac à dos associe à y
7: **finsi**
8: Soit $V_0 = \{j/x_1(j) \neq x_2(j) \text{et} x_1(j) = 0\}$ et $V_1 = \{j/x_1(j) \neq x_2(j) \text{et} x_1(j) = 1\}$
9: **pour** $j \in V_1$ **faire**
10: poser $x_1(j) = 0$, $Res = Res + w_j$ et $V_1 = V_1 - \{j\}$.
11: Soit $\ell^{'} = argmax\{\frac{c_\ell}{w_\ell}, \ell \in V0\}$
12: **si** $Res \leq w_{\ell'}$ **alors**
13: $x_1(\ell^{'}) = 1$, $Res = Res - w_{\ell'}$ et $V_0 = V_0 - \{\ell^{'}\}$.
14: **si** $Z(y) > Z(x*)$ **alors**
15: $x* = y$
16: **finsi**
17: **finsi**
18: **fin pour**
19: **Tant que** $V_0 \neq \emptyset$ **faire**
20: Soit $\ell^{'} = argmax\{\frac{c_\ell}{w_\ell}, \ell \in V0\}$
21: **si** $Res \leq w_{\ell'}$ **alors**
22: $x_1(\ell^{'}) = 1$, $Res = Res - w_{\ell'}$ et $V_0 = V_0 - \{\ell^{'}\}$.
23: **si** $Z(y) > Z(x*)$ **alors**
24: $x* = y$
25: **finsi**
26: **sinon**
27: $V_0 = V_0 - \{\ell^{'}\}$.
28: **finsi**
29: **fin du « Tant que »**
30: **Retourner** La meilleure solution x^* trouvée distincte de x_1 et x_2.

Nous utilisons pour combiner les sous-ensembles de type E_1 un algorithme

basé sur un principe glouton. L'idée est de fixer un élément x_j à un (resp. à zéro) si la valeur de cet élément égale à un (resp. zéro) dans les deux solutions. Dans le cas où l'élément n'a pas la même valeur dans les deux solutions, l'algorithme tente de fixer l'élément non encore fixer à un si la contrainte de capacité n'est pas violée où à zéro dans le cas contraire.

L'algorithme 10 décrit les différentes étapes de la méthode approchée. A partir des deux solutions x_1 et x_2, les étapes de 1 à 5 de l'algorithme permettent l'initialisation de la valeur de la fonction objectif de x^* et le choix de la solution de départ y. Ensuite, construire l'ensemble V_0 (resp. V_1) des indices des éléments nuls (égaux à un) de y tel que $x_1 \neq x_2$ (étape 8). A chaque itération de la boucle principale (étapes de 9 à 18), l'algorithme fixe un élément de V_1 à 0 et essaye de mettre l'élément de plus grande valeur du rapport profit$/$poids de V_0 à un si son poids ne dépasse pas la valeur résiduelle Res. Finalement, l'algorithme tente d'améliorer la solution en fixant les éléments qui reste dans V_0 si la contrainte de capacité est respectée (étapes de 19 à 29).

Nous rappelons que le problème consiste à maximiser la classe de profits minimale, d'où cet algorithme est appliqué à cette classe.

Exemple : Comme application de cette stratégie, nous utilisons le sous ensemble de type 1 composé des solutions $S3, S4$. La table 2.9 représente la nouvelle solution produite par cette combinaison.

La colonne 6 représente la nouvelle solution $Nsol$ après la combinaison et la ligne 7 correspond à la solution après l'amélioration.

2.2 Les différentes étapes de l'algorithme

Dans cette section, nous décrivons les principales étapes de l'algorithme (noté SSKSP : "Scatter Search for the KSP"). Ces différentes étapes son données par l'algorithme 11. A la première étape (étape1 : initialisation), l'algorithme procède à construire la population P (de taille $|P|$) à partir de la solution construite par l'algorithme glouton Heur et à explorer le voisinage de cette solution par les deux profondeurs à gauche et à droite. Ensuite (lignes de 3 à 5), les b_1 meilleures solutions et b_2 solutions diversifiées de P sont sélectionnées pour construire l'ensemble référent de taille $b = b_1 + b_2$. L'étape 2 représente la partie principale de l'algorithme SSKSP. Elle est constituée de deux boucles imbriquées. La première (boucle externe) est utilisée pour reproduire une nouvelle population (ligne 16) et évite la stagnation de la recherche dans une solution locale, le nombre d'itération est fixé a une valeur finie (voir la partie expérimentale).

Algorithm 11 Algorithme approché SSKSP pour le KSP

Entrées: Une instance du KSP.
Sorties: Une meilleure solution approchée.

1: **Etape 1** ;
2: Soit x' la solution trouvée avec l'algorithme *Heur* ;
3: Appliquer les profondeurs à gauche (de taille t_1) et à droite (de taille t_1) sur les éléments critiques de la solution x' pour obtenir une population initiale de taille $\mid P \mid = t_1 * t_2$,
4: Sélectionner b_1 meilleures solutions de P et b_2 solutions diversifiées ;
5: $\mid RefSet \mid = b = b_1 + b_2$;
6: **Etape 2**
7: **Tant que** le nombre maximum de générations n'est pas atteint **faire**
8: **Tant que** une nouvelle solution est introduite dans $RefSet$ **faire**
9: Pour chaque deux solutions de $RefSet$, utiliser l'opérateur de combinaison pour produire une nouvelle solution $NSol$,
10: Améliorer la solution $NSol$ avec l'opérateur d'amélioration
11: **si** $NSol$ améliore la qualité de $RefSet$ **alors**
12: Ajouter $NSol$ aux $RefSet$
13: Enlever de $RefSet$ la mauvaise solution
14: **finsi**
15: **fin du « Tant que »**
16: Régénérer une nouvelle population P,
17: **fin du « Tant que »**
18: **Etape 3**
19: **Retourner** La meilleure solution trouvée ;

TABLE 2.9 – Combinaison entre deux solutions

Variables	$S3$	$S4$	Nsol	Amélioration
y_1	1	1	1	1
y_2	1	1	1	1
y_3	1	0	0	0
y_4	0	1	1	1
y_5	1	1	1	1
y_6	0	0	0	0
y_7	0	0	0	0
y_8	1	0	0	0
y_9	0	0	0	0
y_{10}	0	0	0	1
y_{11}	1	1	1	1
y_{12}	1	1	1	1
y_{13}	1	1	1	1
y_{14}	0	1	0	0
y_{15}	1	0	1	1
y_{16}	0	0	0	0
Z	301	286	299	301

A chaque itération de la deuxième boucle (boucle interne), nous procédons à la combinaison répétée des solutions (lignes de 8 à 13). L'algorithme commence par la combinaison des deux solutions pour construire une nouvelle, ensuite à tenter de l'améliorer (ligne 10) et enfin, la mise à jour de ***RefSet*** (lignes de 11 à 14). Les facteurs d'intensification se retrouvent dans l'application systématique d'une méthode de recherche locale et les facteurs de diversification se retrouvent dans le choix des sous-ensembles diverses. Finalement, l'algorithme s'arrête si le nombre maximum d'itérations est atteint et renvoie la meilleure solution trouvée(voir la figure 2.2).

2.3 Partie expérimentale

Dans cette partie nous évaluons la performance de la méthode proposée sur un ensemble composé de 240 instances de la littérature. Ces instances sont réparties en deux groupes d'instances : le premier groupe représente des instances non-corrélées et le deuxième contient des instances corrélées.

– Premier groupe : il est composé de 168 instances, où le nombre d'éléments n appartient à l'intervalle discret $\{1000,\ 20000\}$, le nombre de

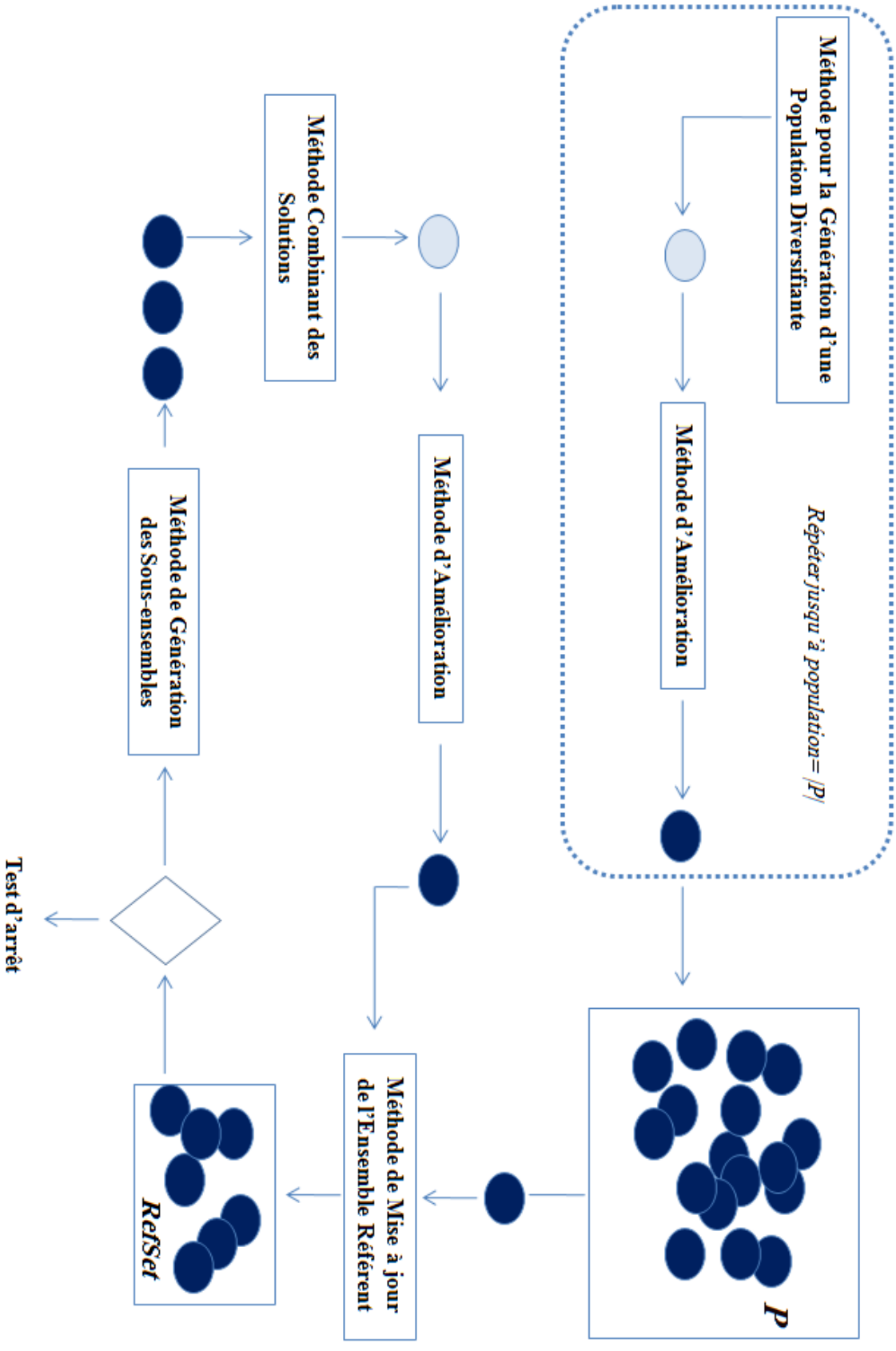

FIGURE 2.2 – Schéma général de la méthodologie de la recherche dispersée

Groupe Inst	n	m	Groupe Inst	n	m
A02.x	1000	2	D02.x	7500	2
A05.x	1000	5	D05.x	7500	5
A10.x	1000	10	D10.x	7500	10
A20.x	1000	20	D20.x	7500	20
A30.x	1000	30	D30.x	7500	30
A40.x	1000	40	D40.x	7500	40
A50.x	1000	50	D50.x	7500	50
B02.x	2500	2	E02.x	10000	2
B05.x	2500	5	E05.x	10000	5
B10.x	2500	10	E10.x	10000	10
B20.x	2500	20	E20.x	10000	20
B30.x	2500	30	E30.x	10000	30
B40.x	2500	40	E40.x	10000	40
B50.x	2500	50	E50.x	10000	50
C02.x	5000	2	F02.x	20000	2
C05.x	5000	5	F05.x	20000	5
C10.x	5000	10	F10.x	20000	10
C20.x	5000	20	F20.x	20000	20
C30.x	5000	30	F30.x	20000	30
C40.x	5000	40	F40.x	20000	40
C50.x	5000	50	F50.x	20000	50
A02C.x	1000	2	D02C.x	7500	2
A05C.x	1000	5	D05C.x	7500	5
A10C.x	1000	10	D10C.x	7500	10
B02C.x	2500	2	E02C.x	1000	2
B05C.x	2500	5	E05C.x	1000	5
B10C.x	2500	10	E10C.x	1000	10
C02C.x	5000	2	F02C.x	20000	2
C05C.x	5000	5	F05C.x	20000	5
C10C.x	5000	10	F10C.x	20000	10

TABLE 2.10 – Information sur les instances testées.

classes m est généré dans l'intervalle $\{2,\ 50\}$ et la cardinalité de chaque classe $J_i,\ i = 1, \ldots, m$ varie dans l'intervalle discret $\{1,\ n-m+1\}$. Le poid w_j et le profit p_j du j-ème élément sont mutuellement indépendants et uniformément générés dans l'intervalle discret $\{1,\ 100\}$ et $\{1,\ 50\}$, respectivement. La capacité R du sac est fixée à la valeur suivante : $\left(\sum_{j=1}^{n} w_j\right)/2$ (détaillées dans le Tableau 2.10).

– Deuxième groupe : ce groupe est composé de 72 instances générées de la même façon que les instances du premier groupe à l'exception des valeurs associées aux profits p_j, caractérisant les éléments, sont égaux à $w_j + 100$.

La Table 2.10 représene les informations caractérisant les instances non-corrélées et corrélées. Comme il est indiqué dans la table, les instances du premier groupe sont celles représentées par Ayy.x, Byy.x, Cyy.x, Dyy.x, Eyy.x et Fyy.x, où yy=02, 05, 10, 20, 30, 40 et 50 et, x varie de 1 à 4 (quatre instances par petit groupe). De même, le deuxième groupe contient les instances corrélées représentées par AyyC.x, ByyC.x, CyyC.x, DyyC.x, EyyC.x et FyyC.x, où yy=02, 05, 10, 20, 30, 40 et 50 et, x varie de 1 à 4. Notons que la colonne 2

(resp. colonne 5) dénote le nombre n d'objets variant entre 5000 et 20000.

Notons aussi que la méthode proposée a été codée en langage C et exécutée sur un PC (1.5 Ghz et 256 MB de RAM). Comme pour la plupart des méthodes à base de population, l'implémentation de l'algorithme SSKSP nécessite des réglages sur les paramètres utilisés par cet algorithme :

- La taille de la population P a tété fixée dans l'intervalle discret $\{400, 5000\}$.
- La taille de l'ensemble référent a est fixée à 40 (il s'agit d'une valeur auparavant utilisée comme valeur standard dans Laguna *et al.* [15] et qui a été confirmée pour le KSP).
- Le nombre d'itération de la méthode a été fixé à 100. Cette valeur a été fixée vu que la méthode ne réalisait pas de meilleurs résultats pour des valeurs supérieures.
- La valeur du *paramètre profondeur* pour se déplacer vers la gauche de l'élément critique a été fixée à 10 et à 20 lorsque le déplacement s'effectue vers la droite.

En premier, les résultats obtenus par l'algorithme proposé SSKSP pour les instance A, B et C non-corrélées sont reportés dans la Table 2.11 et ceux réalisés pour les groupe D, E et F sont reporté dans la Table 2.12. Ces deux tables affichent :

- Une colonne (colonne 1) indiquant le nom de l'instance.
- La valeur de la solution optimale (notée Opt) extraite de Hifi et Sadfi [11].
- La solution heuristique (notée S_{SS}) obtenue par l'algorithme SSKSP.
- Le pourcentage de la déviation (noté D) entre les résultats obtenus par SSKSP et les solutions optimales. Cette déviation est calculée comme suit : $D = \frac{Opt - S_{SS}}{Opt} * 100$.
- La moyenne sur les pourcentages de déviation (notée Av. D.) sur l'ensemble des instances du groupe.

A partir des résultats reportés sur les Tables 2.11 et 2.12, on peut observez que :

- Le pourcentage de déviation moyen pour les instances non-corrélées (Tables 2.11 et 2.12) est égale à 0.077% de l'optimum (qui représente la moyenne des valeurs des deux dernières lignes des deux tables). Par ailleurs, l'algorithme est capable de résoudre 58 instances à l'optimum, ce qui représente un pourcentage de 34.52%.
- Le meilleur pourcentage de déviation moyen est atteint pour le groupe d'instances B ; il est égale à 0.054%, alors que le moins important est atteint pour le groupe d'instances E qui est égal à 0.104%.

Pour le deuxième type d'instances corrélées, les résultats sont donnés par la table 2.13. Cette table affiche : **Instance**, **Opt**, **SSKSP** et **PD** définies précédemment.

A partir de la table 2.13, on peut remarquer que le pourcentage de déviation

Instance	*Opt*	S_{SS}	*D*	Instance	*Opt*	S_{SS}	*D*	Instance	*Opt*	S_{SS}	*D*
A02.1	20490	20490	0,000	B02.1	50803	50803	0,000	C02.1	102284	102284	0,000
A02.2	20419	20419	0,000	B02.2	50136	50134	0,004	C02.2	101565	101565	0,000
A02.3	20889	20888	0,005	B02.3	50873	50873	0,000	C02.3	101551	101551	0,000
A02.4	20564	20564	0,000	B02.4	52143	52143	0,000	C02.4	104568	104568	0,000
A05.1	8071	8058	0,161	B05.1	20371	20363	0,039	C05.1	40564	40556	0,020
A05.2	7995	7984	0,138	B05.2	20349	20346	0,015	C05.2	40798	40794	0,010
A05.3	7960	7947	0,163	B05.3	20424	20412	0,059	C05.3	40438	40432	0,015
A05.4	8115	8112	0,037	B05.4	20376	20370	0,029	C05.4	40449	40444	0,012
A10.1	4054	4039	0,370	B10.1	10047	10027	0,199	C10.1	20391	20381	0,049
A10.2	3525	3525	0,000	B10.2	9905	9894	0,111	C10.2	20252	20236	0,079
A10.3	4087	4078	0,220	B10.3	10129	10118	0,109	C10.3	20213	20197	0,079
A10.4	4037	4032	0,124	B10.4	10360	10348	0,116	C10.4	20858	20842	0,077
A20.1	1989	1981	0,402	B20.1	4997	4982	0,300	C20.1	10113	10092	0,208
A20.2	1465	1465	0,000	B20.2	4164	4164	0,000	C20.2	10066	10046	0,199
A20.3	2001	1999	0,100	B20.3	4996	4979	0,340	C20.3	10079	10065	0,139
A20.4	1972	1968	0,203	B20.4	5115	5105	0,196	C20.4	10377	10363	0,135
A30.1	1088	1088	0,000	B30.1	2525	2525	0,000	C30.1	6719	6698	0,313
A30.2	747	747	0,000	B30.2	3123	3123	0,000	C30.2	6698	6681	0,254
A30.3	1277	1277	0,000	B30.3	3218	3218	0,000	C30.3	6594	6585	0,136
A30.4	1198	1198	0,000	B30.4	2716	2716	0,000	C30.4	6206	6206	0,000
A40.1	712	712	0,000	B40.1	2173	2173	0,000	C40.1	4644	4644	0,000
A40.2	595	595	0,000	B40.2	2177	2177	0,000	C40.2	4846	4846	0,000
A40.3	716	716	0,000	B40.3	2181	2181	0,000	C40.3	4588	4588	0,000
A40.4	668	668	0,000	B40.4	2057	2057	0,000	C40.4	5122	5102	0,390
A50.1	550	550	0,000	B50.1	1811	1811	0,000	C50.1	3935	3935	0,000
A50.2	459	459	0,000	B50.2	1615	1615	0,000	C50.2	3992	3980	0,301
A50.3	536	536	0,000	B50.3	1511	1511	0,000	C50.3	3633	3633	0,000
A50.4	489	489	0,000	B50.4	1780	1780	0,000	C50.4	3994	3994	0,000
Av. D			0.069				0.054				0.086

TABLE 2.11 – Performance de la méthode diversifiante sur les instances non-corrélées : A, B et C.

en moyenne pour ce type d'instance est plus faible par rapport aux instances non corrélées, car ce pourcentage est de 0.014% de l'optimum. Ainsi, le nombre de solutions optimales est 11 avec un pourcentage de 15.28%.

2.4 Conclusion

Dans ce chapitre, nous nous sommes intéresses à la résolution approchée du problème de la distribution équitable. Cette approche est basée sur le principe de la recherche dispersée. Nous avons proposé en premier, un nouveau générateur de solutions permettant d'obtenir une population initiale. Ce générateur utilisé deux paramètres profondeurs qui limitent la recherche des solutions voisines autour des éléments critiques. Ensuite, nous avons utilisé un algorithme glouton pour compléter et améliorer la population. Nous avons utilisé aussi une décomposition de la population en sous-populations de différentes tailles ainsi que nous avons incorporé deux méthodes de combinaison. Notre approche permet d'obtenir des résultats proches en moyenne à l'optimum.

Instance	Opt	SSKSP	PD	Instance	Opt	SSKSP	PD	Instance	Opt	SSKSP	PD
D02.1	153401	153401	0,000	E02.1	203577	203577	0,000	F02.1	409305	409305	0,000
D02.2	152374	152374	0,000	E02.2	204750	204750	0,000	F02.2	409818	409760	0,014
D02.3	153455	153455	0,000	E02.3	204462	204462	0,000	F02.3	409741	409728	0,003
D02.4	155556	155556	0,000	E02.4	207203	207203	0,000	F02.4	406213	406053	0,039
D05.1	61486	61478	0,013	E05.1	81275	81273	0,002	F05.1	163514	163506	0,005
D05.2	61000	60995	0,008	E05.2	81240	81238	0,002	F05.2	162566	162562	0,002
D05.3	60690	60680	0,016	E05.3	81956	81954	0,002	F05.3	163850	163846	0,002
D05.4	60750	60743	0,012	E05.4	81196	80922	0,337	F05.4	163202	163199	0,002
D10.1	30574	30561	0,043	E10.1	40681	40668	0,032	F10.1	81739	81726	0,016
D10.2	30460	30451	0,030	E10.2	40828	40814	0,034	F10.2	81957	81950	0,009
D10.3	30619	30614	0,016	E10.3	40839	40817	0,054	F10.3	81917	81908	0,011
D10.4	31029	31010	0,061	E10.4	41406	41386	0,048	F10.4	81168	81162	0,007
D20.1	15276	15259	0,111	E20.1	20274	20256	0,089	F20.1	40884	40864	0,049
D20.2	15151	15132	0,125	E20.2	20382	20361	0,103	F20.2	40926	40912	0,034
D20.3	15256	15244	0,079	E20.3	20368	20356	0,059	F20.3	40936	40910	0,064
D20.4	15468	15447	0,136	E20.4	20634	20616	0,087	F20.4	40528	40510	0,044
D30.1	10129	10103	0,257	E30.1	13438	13422	0,119	F30.1	27217	27199	0,066
D30.2	10103	10080	0,228	E30.2	13556	13533	0,170	F30.2	27250	27237	0,048
D30.3	10153	10136	0,167	E30.3	13590	13566	0,177	F30.3	27223	27203	0,073
D30.4	9591	9591	0,000	E30.4	13200	13200	0,000	F30.4	26938	26923	0,056
D40.1	7606	7582	0,316	E40.1	10098	10078	0,198	F40.1	20393	20372	0,103
D40.2	7505	7488	0,227	E40.2	9838	9838	0,000	F40.2	20425	20399	0,127
D40.3	7074	7074	0,000	E40.3	10150	10126	0,236	F40.3	20428	20404	0,117
D40.4	7663	7646	0,222	E40.4	10260	10244	0,156	F40.4	20218	20191	0,134
D50.1	6057	6040	0,281	E50.1	8081	8061	0,247	F50.1	16262	16240	0,135
D50.2	5797	5797	0,000	E50.2	8081	8061	0,247	F50.2	16291	16267	0,147
D50.3	5407	5407	0,000	E50.3	8111	8090	0,259	F50.3	16326	16303	0,141
D50.4	6131	6116	0,245	E50.4	8195	8174	0,256	F50.4	16123	16104	0,118
V.PD			0.093				0.104				0.056

TABLE 2.12 – Performance de la méthode diversifiante sur les instances non-corrélées : D, E et F.

Instance	Opt	SSKSP	PD	Instance	Opt	SSKSP	PD
A02C.1	41492	41486	0,0145	D02C.1	312223	312217	0,0019
A02C.2	41397	41389	0,0193	D02C.2	311947	311947	0,0000
A02C.3	41565	41565	0,0000	D02C.3	311690	311675	0,0048
A02C.4	41556	41556	0,0000	D02C.4	311419	311400	0,0061
A05C.1	16554	16554	0,0000	D05C.1	124794	124779	0,0120
A05C.2	16561	16554	0,0423	D05C.2	124758	124752	0,0048
A05C.3	16590	16587	0,0181	D05C.3	124669	124662	0,0056
A05C.4	16584	16584	0,0000	D05C.4	124529	124520	0,0072
A10C.1	8245	8245	0,0000	D10C.1	62334	62317	0,0273
A10C.2	8203	8198	0,0610	D10C.2	62311	62292	0,0305
A10C.3	8250	8242	0,0970	D10C.3	62289	62276	0,0209
A10C.4	8255	8255	0,0000	D10C.4	62216	62203	0,0209
B02C.1	103672	103664	0,0077	E02C.1	416396	416374	0,0053
B02C.2	103572	103549	0,0222	E02C.2	415426	415409	0,0041
B02C.3	104034	104023	0,0106	E02C.3	415470	415463	0,0017
B02C.4	104031	104019	0,0115	E02C.4	415341	415324	0,0041
B05C.1	41393	41391	0,0048	E05C.1	166484	166463	0,0126
B05C.2	41437	41426	0,0265	E05C.2	166126	166120	0,0036
B05C.3	41580	41580	0,0000	E05C.3	166116	166094	0,0132
B05C.4	41561	41553	0,0192	E05C.4	166062	166052	0,0060
B10C.1	20657	20653	0,0194	E10C.1	83216	83199	0,0204
B10C.2	20648	20640	0,0387	E10C.2	82995	82974	0,0253
B10C.3	20740	20730	0,0482	E10C.3	83029	83009	0,0241
B10C.4	20721	20710	0,0531	E10C.4	82981	82974	0,0084
C02C.1	207675	207672	0,0014	F02C.1	830622	830593	0,0035
C02C.2	207916	207901	0,0072	F02C.2	831513	831513	0,0000
C02C.3	208043	208043	0,0000	F02C.3	831145	831125	0,0024
C02C.4	207973	207955	0,0087	F02C.4	831445	831423	0,0026
C05C.1	83013	82994	0,0229	F05C.1	332143	332122	0,0063
C05C.2	83129	83115	0,0168	F05C.2	332527	332520	0,0021
C05C.3	83185	83172	0,0156	F05C.3	332402	332388	0,0042
C05C.4	83188	83188	0,0000	F05C.4	332498	332467	0,0093
C10C.1	41466	41449	0,0410	F10C.1	166028	166012	0,0096
C10C.2	41521	41515	0,0145	F10C.2	166248	166238	0,0060
C10C.3	41554	41547	0,0168	F10C.3	166154	166152	0,0012
C10C.4	41554	41542	0,0289	F10C.4	166192	166187	0,0030

TABLE 2.13 – Performance de l'approche recherche dispersée sur les instances corrélées

Chapitre 3

Un algorithme exact pour le problème de la distribution équitable généralisée

---

Ce chapitre est consacré à la résolution exacte du problème de la distribution équitable avec des objets communs, appelé aussi problème de la distribution équitable généralisée (*Generalised Knapsack Sharing Problem* $-GKSP$). Nous proposons une méthode exacte basée sur une décomposition du problème original en une série de sous-problèmes plus ou moins faciles à résoudre. Ces sous-problèmes sont résolus par application d'algorithmes dédiés et souvent très performants. Par ailleurs, afin d'accélérer le processus de recherche, nous proposons un encadrement de l'optimum en introduisant un calcul de bornes et des stratégies de réduction sur l'instance traitée.

Les travaux de recherche de ce chapitre ont fait l'objet d'une publication dans une revue internationale (*European Journal of Operational Research 2016* [4]).

3.1 Introduction

Le problème de la distribution équitable avec des éléments communs est une variante du problème du sac à dos classique KP. Il peut être aussi considéré comme une extension du problème KSP étudié dans le chapitre précédent (Chapitre 2). Une instance du problème est caractérisée par la donnée d'un sac de capacité fixée C et d'un ensemble N de $n+1$ sous-ensembles disjoints : $N_0,\ N_1, \ldots, N_n$, où $\cup_{i=1}^{m},\ N_i \cap N_j = \emptyset$, avec $i \neq j$. Notons par N_0 l'ensemble des éléments communs à toutes les classes et $N_i,\ i = 1, \ldots, n$, l'ensemble des éléments non-communs. Une classe i est constituée des éléments du sous-ensemble $N_i,\ i = 1, \ldots., n$, et des éléments du sous-ensemble N_0, où une classe i contient $N_i \cup N_0$ éléments. Finalement, à chaque élément $j \in N$ est associé

un poids w_i et un profit p_i. L'objectif du problème est de trouver un sous-ensemble contenant des éléments à retenir dans le sac maximiser la classe de profit minimal.

Notons par P_{kp} le problème du sac à en dos 0-1 défini par l'ensemble des éléments communs de N_0 et P_{ksp} le problème de la distribution équitable KSP défini par l'ensemble des éléments non-communs telle que $\cup_{i=1}^{m} N_i$. Supposons que N_0 contient m_0 éléments ($|N_0| = m_0$) et N_i contient m_i éléments ($|N_i| = m_i$), $i = 1, \ldots, n$. Les deux problèmes P_{kp} et P_{ksp} peuvent s'écrire comme suit :

$$\begin{aligned} \mathrm{P}_{kp}: \quad \max \quad & Z_{kp} = p_0 \cdot x_0 \\ \text{s.c.} \quad & w_0 \cdot x_0 \leq C_{kp} \\ & x_0 \in \{0,1\}^{m_0}, \end{aligned}$$

et

$$\begin{aligned} \mathrm{P}_{ksp}: \quad \max \quad & Z_{ksp} = \min_{1 \leq i \leq n} \{p_i \cdot x_i\} \\ \text{s.c.} \quad & \textstyle\sum_{i=1}^{n} w_i \cdot x_i \leq C_{ksp} \\ & x_i \in \{0,1\}^{m_i}, \ \forall\ i = 1, \ldots, n, \end{aligned}$$

où C_{kp} (resp. C_{ksp}) représente la capacité du sac de chacun des problèmes, $p_0 = (p_1, ..., p_{m_0})$ (resp. $p_i = (p_1, ..., p_{m_i})$, $i = 1, ..., n$) est le vecteur profit des éléments communs (resp. non-communs), $w_0 = (w_1, ..., w_{m_0})$ est le vecteur poids (resp. $w_i = (w_1, ..., w_{m_i})$, $i = 1, ..., n$) , et x_0 est le vecteur des variables de décision (resp. x_i , $i = 1, ..., n$) de P_{kp} (resp. P_{ksp}).
Notons qu'à partir des deux problèmes P_{kp} et P_{ksp}, le programme associé au P_{gksp} peut être écrit de la façon suivante :

$$\begin{aligned} \mathrm{P}_{gksp}: \quad \max \quad & Z_{gksp} = \min_{1 \leq i \leq n} \{p_i \cdot x_i\} + p_0 \cdot x_0 \\ \text{s.c.} \quad & \sum_{i=1}^{n} w_i \cdot x_i + w_0 \cdot x_0 \leq C_{gksp} \\ & x_0 \in \{0,1\}^{m_0}, x_i \in \{0,1\}^{m_i}, i = 1, \ldots, n, \end{aligned}$$

où $C_{gksp} = C_{kp} + C_{ksp}$.

D'après cette formulation, nous pouvons considérer que le P_{gksp} est une combinaison des deux problèmes P_{kp} et P_{ksp}. En effet, nous pouvons écrire le problème P_{gksp} sous forme d'un programme linéaire PL_{gksp} de la manière

suivante :

$$
\begin{aligned}
\text{LP}_{gksp}: \quad \max \quad & Z_{gksp} = \gamma + p_0 \cdot x_0 \\
\text{s.c.} \quad & w_0 \cdot x_0 \leq C_{kp} && (3.1)\\
& \sum_{i=1}^{n} w_i \cdot x_i \leq C_{gksp} - C_{kp} && (3.2)\\
& p_i \cdot x_i \geq \gamma, \qquad \forall\ i = 1, \ldots, n && (3.3)\\
& \gamma \in \mathbb{Z},\ x_0 \in \{0,1\}_0^m,\ x_i \in \{0,1\}^{m_i},\ i = 1, \ldots, n. && (3.4)
\end{aligned}
$$

3.2 Méthode de décomposition pour le $GKSP$

Dans cette section, nous présentons l'algorithme exact pour la résolution du $GKSP$. Le principe de l'algorithme est basé sur une énumération des combinaisons des capacités C_{kp} et C_{ksp} relatives aux problèmes P_{kp} et P_{ksp} respectivement.

Soit $Z_{kp}(C)$ (resp. $Z_{ksp}(C)$) la valeur optimale de la fonction objectif du problème KP (resp. KSP) pour une capacité fixée à C. Selon la formulation mathématique des problèmes P_{gksp}, P_{ksp} et P_{kp}, la valeur de la fonction objectif du $GKSP$, notée Z_{gksp}, peut être exprimée comme suit :

$$Z_{gksp} = Z_{kp}(C_{kp}) + Z_{ksp}(C_{gksp} - C_{kp}), \tag{3.5}$$

où C_{kp} (entier positif) est la capacité fournie au problème du sac à dos classique KP.

Fujimoto et Yamada [6] ont défini les valeurs discrètes C_{kp} de l'intervalle $\{0,\ C_{gksp}\}$ par les points de discontinuité. Dans ce qui suit, notre étude est basée sur une recherche ciblée afin de trouver la valeur C_{kp}^* (cf., équation (3.5)) qui réalise l'évaluation optimale Z_{gksp^*} pour le $GKSP$.

3.2.1 Borne supérieure

La borne supérieure est un facteur important dans une méthode exacte, car elle participe d'une façon très efficace dans sa performance : accélération du processus de recherche. Plusieurs techniques de calcul de bornes pour le $GKSP$ peuvent être utilisées pour réduire l'espace de recherche. En effet, nous pouvons obtenir une telle borne en utilisant la méthode du simplexe

pour résoudre le problème $GKSP$ relaxé PR_{gksp} suivant :

$$\begin{aligned} \text{LR}_{gksp}: \quad \max \quad & Z_{gksp} = \gamma + p_0 \cdot x_0 \\ \text{s.c.} \quad & (3.1), (3.2), (3.3) \\ & \gamma \in \mathbb{R},\ x_0 \in [0,1]^{m_0},\ x_i \in [0,1]^{m_i},\ i = 1, \ldots, n. \end{aligned}$$

Dans notre étude nous utilisons une technique de décomposition du $GKSP$ permettant d'accélérer le processus de recherche.

Notons par $U_{kp}(C)$ (resp. $U_{ksp}(C)$) la borne supérieure du KP (resp. KSP) telle que la capacité correspondante est fixée à C. D'après l'équation (3.5) la borne supérieure du $GKSP$ est calculée de la manière suivante :

$$U_{gksp} = U_{kp}(C_{kp}) + U_{ksp}(C_{gksp} - C_{kp}). \tag{3.6}$$

A partir de l'équation (3.6), nous pouvons en déduire que la borne supérieure est une fonction de C_{kp}, c-à-d, $U_{gksp} = f(C_{kp})$. Dans ce cas, trouver une borne supérieure revient à trouver la valeur de C_{kp} qui maximise U_{gksp}, qu'on note par U^*_{gksp}. D'où la borne supérieure du $GKSP$ est celle réalisant :

$$U^\star_{gksp} = f(C^\star_{kp}) \text{ and } f(C^\star_{kp}) \geq f(C_{kp}),\ \forall\ C_{kp} \in \mathbb{N}.$$

De plus, la fonction f est une fonction concave et linéaire par morceaux (cf. Fujimoto et Yamada [6]). Selon la première propriété, nous développons l'algorithme de recherche dichotomique (Algorithme 12) pour la détermination d'une borne supérieure U_{gksp} pour P_{gksp}.

L'algorithme 12 consiste à explorer l'intervalle de recherche $[C^L,\ C^U]$ afin de trouver la valeur C^T qui maximise la fonction f définie ci-dessus. Dans ce cas, la borne supérieure du problème P_{gksp} est donnée par $U^*_{gksp} = f(C^T)$. Les deux étapes (1 et 2) permettent d'initialiser l'intervalle de recherche $[C^L,\ C^U]$ et de le partager en deux parties $[C^L,\ C^M]$ et $[C^M,\ C^U]$. La boucle principale (étapes de 3 à 19) tente de réduire les valeurs C^L et C^U en vérifiant si la valeur C^T est dans l'intervalle $[C^L,\ C^M]$ ou $[C^M,\ C^U]$. A chaque itération, le test permet d'éliminer une partie qui ne contient pas la solution optimale de LR_{gksp}. Ce test exclusif est dû au fait que la fonction f est concave. Finalement, l'algorithme s'arrête lorsque la différence entre C^L et C^U est inférieure à "un'.

3.2.2 Calcul d'une borne inférieure

On peut remarquer que l'équation 3.5 dépend essentiellement des valeurs discrètes C_{kp} ($C_{kp} \in \{0, 1, \ldots, C_{gksp}\}$). Chaque valeur de $Z_{gksp}(C_{kp})$ est une borne inférieure valide pour le P_{gksp}. Dans ce qui suit, les valeurs discrètes C_{kp}

Algorithm 12 Une borne supérieure valide pour le $GKSP$

Entrées: Une instance du problème $GKSP$ (P_{gksp}).
Sorties: UB est la borne supérieure de P_{gksp}.

Initialisation : Poser $C^L = 0$, $C^U = C_{gksp}$, $C^M = \frac{(C^L + C^U)}{2}$ et $UB = -\infty$;
Posre $V_{C^L} = f(C^L)$, $V_{C^U} = f(C^U)$ et $V_{C^M} = f(C^M)$;
Tant que $C^U - C^L \geq 1$ **faire**
  si $C^M - C^L > C^U - C^M$ **alors**
    Poser $C^T = \frac{(C^L + C^M)}{2}$ et $V_{C^T} = f(C^T)$;
    si $V_{C^T} > V_{C^M}$ **alors**
      Poser $C^U = C^M$, $V_{C^U} = V_{C^M}$, $C^M = C^T$ et $V_{C^M} = V_{C^T}$;
    sinon
      Poser $C^L = C^T$ et $V_{C^L} = V_{C^T}$;
    finsi
  sinon
    Poser $C^T = \frac{(C^U + C^M)}{2}$ et $V_{C^T} = f(C^T)$;
    si $V_{C^T} > V_{C^M}$ **alors**
      Posre $C^L = C^M$, $V_{C^L} = V_{C^M}$, $C^M = C^T$ et $V_{C^M} = V_{C^T}$;
    sinon
      Poser $C^U = C^T$ et $V_{C^U} = V_{C^T}$;
    finsi
  finsi
fin du « Tant que »
Poser $UB = \max\{V_{CL}, V_{CM}, V_{CU}\}$;
Retourner UB est la borne supérieure de P_{gksp}.

sont appelées : les points de discontinuité. Ainsi, chaque point de discontinuité est associé à une borne inférieure pour le problème P_{gksp}.

Finalement, déterminer une solution optimale pour P_{gksp} revient à chercher la meilleure valeur de C_{kp} telle que :

$$\exists\ C^{\star}_{kp} \in [0, C_{gksp}],\ \forall C \in [0, C_{gksp}],\ Z_{gksp}(C^{\star}_{kp}) \geq Z_{gksp}(C).$$

La borne inférieure de départ pour le $GKSP$ est la valeur de la fonction objectif du P_{gksp} relative au point de discontinuité C^0_{kp} réalisant la capacité maximisant LR_{gksp} (borne supérieure de P_{gksp}). Cette valeur C^0_{kp} est obtenue par l'application de l'algorithme 12.

Une fois la valeur C^0_{kp} fixée, la borne inférieure du problème $GKSP$ est obtenue par l'utilisation des deux étapes suivantes :

1. Dans la première étape, on calcule $Z_{kp}(C^0_{kp})$ par la résolution du problème P_{kp} au point de discontinuité C^0_{kp}, puis on cherche $C_{kp'}$ la valeur minimale de C_{kp} telle que sa valeur objectif coïncide avec $Z_{kp}(C^0_{kp})$.
2. Ensuite à l'étape deux, on calcule $Zksp(C_{gksp} - C^{'}_{kp})$ par la résolution du problème P_{ksp} en fixant sa capacité à $C_{gksp} - C^{'}_{kp}$.

Finalement, la borne inférieure du P_{gksp} est obtenue comme suit :

$$L_{gksp} = Z_{kp}(C'_{kp}) + Z_{ksp}(C_{gksp} - C'_{kp}).$$

Afin de trouver une solution optimale pour le problème P_{gksp}, on peut utiliser une méthode d'énumération proposée par Fujimoto et Yamada [6], où les deux étapes précédentes sont appliquées aux points de discontinuité.

Afin de restreindre le processus de recherche, nous présentons dans la section suivante quelques stratégies de réduction. Ces stratégies permettent de réduire l'intervalle de recherche (c- à- d les points de discontinuité), ce qui permet une accélération du processus de recherche.

3.2.3 Réduction de l'intervalle globale de recherche

Rappelons que la fonction $f : C_{kp} \rightarrow U_{gksp}$ (définie dans la Section 3.2.1) est une fonction concave et linéaire par morceaux (cf. Fujimoto et Yamada [6]) dans l'intervalle $[0,\ C_{gksp}]$. Ce dernier peut être considéré comme le plus large intervalle contenant la solution optimale de P_{gksp}. Dans ce qui suit, nous utiliserons la borne inférieure et la borne supérieure pour tenter d'affiner l'intervalle initial de recherche $[0,\ C_{gksp}]$. Notons par L_{gksp} la borne inférieure du P_{gksp} obtenue par l'utilisation de la méthode décrite dans la Section 3.2.2.

L'algorithme 13 permet de construire l'intervalle réduit $[LC_{kp},\ UC_{kp}]$ qui contient moins de points de discontinuité que l'intervalle initial $[0,\ C_{gksp}]$.

Algorithm 13 : Réduction de l'intervalle de recherche global de P_{gksp}.

Entrées: Une borne inférieure L_{gksp} de P_{gksp}.
Sorties: Ensemble réduit de points de discontinuité $[LC_{kp},\ UC_{kp}]$.

1: Initialisation : poser $LC_{kp} = 0$ et $UC_{kp} = C_{gksp}$.
2: **Tant que** $\left(f(LC_{kp}) < L_{gksp}\right)$ **faire**
3: $\quad LC_{kp} = LC_{kp} + 1$;
4: **fin du « Tant que »**
5: **Tant que** $\left(f(UC_{kp}) < L_{gksp}\right)$ **faire**
6: $\quad UC_{kp} = UC_{kp} - 1$;
7: **fin du « Tant que »**
8: **Retourner** $[LC_{kp}, UC_{kp}]$.

Cette réduction est due au fait que la fonction f est concave. L'étape 1 de l'algorithme permet d'initialiser l'intervalle de recherche. La première boucle (étapes de 2 à 4) élimine les valeurs LC_{kp} de l'intervalle de recherche d'une manière itérative. Si la borne supérieure au point de discontinuité LC_{kp} est plus petite que la borne inférieure du P_{gksp} $(f(LC_{kp}) < L_{gksp})$, alors ce point est supprimé de l'intervalle. De façon identique, la deuxième boucle (étapes de 5 à 7) permet d'éliminer les valeurs UC_{kp} de l'intervalle de recherche d'une façon itérative en décrémentant la valeur de la limite de l'intervalle. Plus précisément, si la borne supérieure au point UC_{kp} est plus petite que la borne inférieure du P_{gksp} $(f(UC_{kp}) < L_{gksp}\ )$, alors cette valeur est écarté de l'intervalle. Finalement, l'algorithme renvoie l'intervalle de recherche réduit $[LC_{kp},\ UC_{kp}]$.

Dans ce qui suit, nous discutons deux stratégies utilisées pour fournir des intervalles de recherche plus fins que celui trouvé par l'algorithme 13.

3.2.4 Réduction et exploration des sous-intervalles de recherche

Explorer un point de discontinuité revient à résoudre une série de problèmes P_{kp} et P_{ksp} respectivement. Ces problèmes sont NP-difficiles (cf., Garey et Johnson [19]) et la résolution du problème P_{ksp} demande plus d'effort de calculs que la résolution du problème P_{kp}.

Dans cette section, nous décrivons une procédure booléenne (notée "Évaluer") afin d'éviter la résolution du problème P_{ksp}. Cette procédure vérifie si la résolution du P_{ksp} permet d'améliorer ou pas la borne inférieure en cours. Ensuite, nous développerons deux stratégies pour réduire les intervalles de recherche à droite et à gauche.

Afin de développer la procédure "Évaluer", nous définissons les problèmes primaux et duaux du KP et KSP de la manière suivante.

Pour le KP, son primal :

$$\begin{array}{lll} \mathrm{P}_{kp}(\mathrm{C}): & \max & p_0 \cdot x_0 \\ & \text{s.c.} & w_0 \cdot x_0 \leq C \\ & & x_0 \in \{0,1\}^m. \end{array}$$

et son dual

$$\begin{array}{lll} \mathrm{D}_{kp}(\mathrm{V}): & \min & w_0 \cdot x_0 \\ & \text{s.c.} & p_0 \cdot x_0 \geq V \\ & & x_0 \in \{0,1\}^m, \end{array}$$

Pour le KSP, ses primal et dual :

$$\begin{array}{llll} \mathrm{P}_{ksp}(\mathrm{C}): & \max \min\limits_{1\leq i\leq n} \{p_i \cdot x_i\} & \mathrm{D}_{ksp}(\mathrm{V}): & \min \sum_{i=1}^n w_i \cdot x_i \\ \text{s.c.} & \sum_{i=1}^n w_i \cdot x_i \leq C & \text{s.c.} & p_i \cdot x_i \geq V,\ \forall\ i = 1,\ldots,n, \\ & x_i \in \{0,1\}^{m_i},\ \forall\ i = 1,\ldots,n, & & x_i \in \{0,1\}^{m_i},\ \forall\ i = 1,\ldots,n. \end{array}$$

L'objectif du problème primal $P_{kp}(C)$ (resp. $P_{ksp}(C)$) est de trouver le profit maximal sans violer la contrainte de capacité . Le problème dual $D_{kp}(V)$ (resp. $D_{ksp}(V)$) vise à déterminer les ressources minimales qui peuvent être atteintes pour un profit fixé V.

Notons par CT la capacité de P_{kp}, V_{CT} l'évaluation optimale associée à CT, $P_{ksp}(C_{res})$ est l'instance du KSP avec une capacité $C_{res} = C_{gksp} - CT$ et $LB_{gksp}^{meilleur}$ la meilleure borne inférieure en-cours. La procédure "Évaluer" est une procédure booléenne qui décide s'il est nécessaire de résoudre le problème $P_{ksp}(C_{res})$. Cette fonction renvoie la valeur "faux" si l'une des conditions suivantes est vérifiée :

C1 : $UB_{ksp}+V_{CT} \leq LB_{gksp}^{meilleur}$, où UB_{ksp} est la borne supérieure de $P_{ksp}(C_{res})$.

C2 : La condition **C1** n'est pas vérifiée et $LC_{ksp} \geq C_{res}$, où LC_{ksp} est la borne inférieure de $\mathrm{D}_{ksp}(LB_{gksp}^{meilleur} - V_{CT} - 1)$.

C3 : La condition **C2** n'est pas vérifiée et $C_{ksp} \geq C_{res}$, où C_{ksp} représente la valeur de la solution optimale de $\mathrm{D}_{ksp}(LB_{gksp}^{meilleur} - V_{CT} - 1)$.

Premièrement, la condition $C1$ est utilisée pour décider si la résolution du $P_{ksp}(C_{res})$ est rentable ou non. Dans ce cas, la borne supérieure de P_{ksp} est calculée en fixant la capacité à $Cres$. Ensuite, le processus vérifie la validité de l'inégalité $UB_{ksp} + V_{CT} \leq LB_{gksp}^{meilleur}$. Si cette inégalité est vérifiée, alors $LB_{gksp}^{meilleur}$ ne peut pas être améliorée par la résolution $P_{ksp}(C_{res})$. Sinon, la procédure d'évaluation continue à vérifier le reste des conditions.

Deuxièmement, la condition $C2$ vérifie si les ressources disponibles sont suffisantes pour atteindre le profit cible $V' = LB_{gksp}^{meilleur} - V_{CT} - 1$ et par conséquent la meilleure borne inférieure en cours peut être améliorée. Donc, par la considération de la capacité $Cres$, le processus vérifie si la borne inférieure LC_{ksp} de $D_{ksp}(V')$ est plus grande que $Cres$.

Troisièmement, la condition $C3$ est appliquée quand $C2$ n'est pas vérifiée. Dans ce cas, si la valeur optimale de $D_{ksp}(V')$ est plus grande que C_{res} (c-à-d $C_{ksp} \geq C_{res}$) alors l'amélioration de la borne inférieure en cours induit à un dépassement des ressources disponibles.

Finalement, lorsque toutes les conditions ne sont pas réalisées la procédure "Évaluer" résout $P_{ksp}(C_{res})$ à l'optimum. Tandis que si une des conditions décrite précédemment est vérifiée alors la procédure de recherche évite la résolution du $P_{ksp}(C_{res})$ et se déplace vers le prochain point de discontinuité.

Nous pouvons donc conclure que la procédure d'évaluation qu'on notera par "Évaluer($P_{ksp}(C_{res})$, $LB_{gksp}^{meilleur}$)" retourne la valeur "Faux" si l'une des conditions $C1$, $C2$, $C3$ est vérifiée et " vraie" dans le cas contraire.

Dans ce qui suit, nous développons deux procédures de réduction basées sur la procédure d'évaluation précédemment décrite. Les deux stratégies de réduction commencent par la subdivision de l'intervalle réduit $[LC_kp,\ UC_kp]$ trouvé dans la Section 3.2.3 en deux intervalles complémentaire :

(i) Le premier intervalle est noté par $[LC_{kp},\ ML_{kp}[$ correspond à la partie gauche de $[LC_{kp},\ UC_{kp}]$ appelé intervalle gauche,

(ii) Le second intervalle est noté par $]MU_kp,\ UC_kp]$ correspond à la partie droite de $[LC_{kp},\ UC_{kp}]$ appelé intervalle droite.

L'algorithme 14 consiste à réduire l'intervalle gauche $[LC_{kp},\ ML_{kp}[$ et retourne une borne inférieure valide après la résolution exacte du P_{ksp}. A l'étape 1, l'algorithme résout le P_{kp} à l'optimum en fixant sa capacité à $CT = ML_{kp} - 1$ et renvoie la valeur de la solution VCT. L'étape 2 consiste à résoudre le problème dual $D_{kp}(VCT)$ afin de déterminer la capacité minimale CT reliée au profit cible VCT. Ensuite, à l'étape 3 la procédure "Évaluer()" est utilisée pour vérifier si la résolution de P_{ksp} de capacité $(C_{gksp} - CT)$ est rentable. Si "Évaluer()" renvoie la valeur "vraie" alors (étapes de 4 à 6) le $P_ksp(C_{gksp} - CT)$ est résolu à l'optimum et l'algorithme renvoie le nouvel intervalle $[LC_{kp},\ CT[$ et une borne inférieure valide pour le problème P_{gksp}.

Par analogie, l'algorithme 15 permet de réduire l'intervalle droit en appliquant le même principe que l'algorithme précédent. La procédure commence à fixer $VMU = Zkp(MU_{kp})$ à la valeur de la fonction objectif optimale de $P_{kp}(MU_{kp})$. Ensuite, déterminer la capacité minimale CT qui réalise le profit total $V_{MU} + 1$ en résolvant le problème dual $Dkp(V_{MU} + 1)$ à l'optimalité

Algorithm 14 : Réduction de l'intervalle gauche : $[LC_{kp}, ML_{kp}[$.

Entrées: L'ensemble des points de discontinuités $[LC_{kp}, ML_{kp}[$ et la borne inférieure L^{best}_{gksp} de $GKSP$.

Sorties: L'intervalle gauche réduit et une borne inférieure du P_{gksp}.

1: Poser $CT = ML_{kp} - 1$ et soit V_{CT} l'évaluation optimale de $\mathrm{P}_{kp}(CT)$;
2: Réinitialiser CT à la valeur objective optimale de $\mathrm{D}_{kp}(V_{CT})$;
3: **si** `Evaluer`$\Big(P_{ksp}(C_{gksp} - CT),\ L_{gksp}best\Big)$ **alors**
4: Résoudre P_{ksp} $(C_{gksp} - CT)$ à l'optimum ;
5: Poser $V_{ksp} = Z_{ksp}(C_{gksp} - CT)$;
6: **Retourner** $[LC_{kp}, CT[$ et V_{CT}+V_{ksp} (c.à.d une borne inférieure de P_{gksp}).
7: **finsi**
8: **Retourner** $[LC_{kp}, CT[$.

(étape 1). Puis, à l'étape 2 V_{CT} reçoit la valeur de la solution optimale du P_{kp} de capacité CT trouvée à l'étape 1. Ensuite par rappel de la procédure "Évaluer" (étape 3-6) le $P_{ksp}(C_{gksp} - CT)$ est résolu à l'optimum si cette procédure renvoie la valeur "vraie". Dans ce cas l'algorithme renvoie le nouvel intervalle $]CT,\ UC_{kp}]$ et une borne inférieure valide pour le P_{gksp}. Dans le cas contraire, l'algorithme retourne seulement l'intervalle de droite réduit sans résoudre le problème P_{gksp}.

Algorithm 15 : Réduire l'intervalle de droite $]MU_{kp}, UC_{kp}]$.

Entrées: L'ensemble des points de discontinuité $[LC_{kp}, ML_{kp}[$ et la borne inférieure L^{best}_{gksp} de $GKSP$.

Sorties: L'intervalle de droite réduit et une borne inférieure du P_{gksp}.

1: Poser $V_{MU} = Z_{kp}(MU_{kp})$ et soit CT l'évaluation optimale de $\mathrm{D}_{kp}(V_{MU} + 1)$;
2: Poser V_{CT} la valeur objective optimale de $\mathrm{P}_{kp}(CT)$;
3: **si** `Évaluer`$\Big(P_{ksp}(C_{gksp} - CT),\ L^{best}_{gksp}\Big)$ **alors**
4: Résoudre P_{ksp} (avec capacité $(C_{gksp} - CT)$) à l'optimum ;
5: Poser $V_{ksp} = Z_{ksp}(C_{gksp} - CT)$;
6: **Retourner** $]CT, UC_{kp}]$ et V_{CT}+V_{ksp} (c.à.d une borne inferieure de P_{gksp}).
7: **finsi**
8: **Retourner** $]CT, UC_{kp}]$.

3.3 Les étapes principales de la méthode exacte

Dans cette partie, nous décrivons les étapes principales de la méthode exacte pour la résolution du P_{gksp}. Le principe de base de cette méthode (Algorithme 16) est fondé sur la subdivision de l'intervalle initial (les points

de discontinuité) en deux sous-intervalles : un intervalle de gauche et un autre intervalle de droite. En premier lieu, deux bornes supérieure et inférieure sont calculées en résolvant une série de problèmes de P_{kp} et P_{ksp}. Ensuite, des stratégies de réductions sont appliquées sur les deux intervalles pour accélérer l'exploration de l'espace de recherche. L'algorithme s'arrête avec une solution optimale si l'une des conditions d'arrêt suivantes est vérifiée :

- La meilleure borne inférieure en cours coïncide avec la borne supérieure ;
- L'intervalle de gauche et l'intervalle de droit sont réduits à des ensembles vides.

Algorithm 16 Un algorithme exact pour la résolution du $GKSP$

Entrées: Une instance de $GKSP$ (notée P_{gksp}).
Sorties: Une solution optimale de P_{gksp}.

Poser $LC_{kp} = 0$ et $UC_{kp} = C_{gksp}$;
Appel de l'Algorithme 12 et soit UB_{gksp} la borne supérieure de P_{gksp} ;
Soit CT la capacité qui représente le point de discontinuité associé à UB_{gksp} ;
Calculer la borne inférieure LB_{gksp} de P_{gksp} associé à la capacité CT ;
Appel de l'Algorithme 13 et poser $[LC_{kp}, UC_{kp}]$, le nouvel intervalle de recherche réduit ;
Poser $ML_{kp} = MU_{kp} = CT$ et $UB^L_{gksp} = UB^U_{gksp} = UB_{gksp}$;
Tant que $\Big((LC_{kp} < ML_{kp}$ ou $MU_{kp} < UC_{kp})$ et $LB_{gksp} < UB_{gksp}\Big)$ **faire**
  si $UB^L_{gksp} \geq UB^U_{gksp}$ **alors**
    Appel de l'algorithme 14 pour la réduction de l'intervalle gauche $[LC_{kp}, ML_{kp}[$;
    si LB_{gksp} est améliorée, alors appeler L'algorithme 13 pour la réduction de l'intervalle gauche $[LC_{kp}, ML_{kp}[$;
    Appel de l'algorithme 12 sur $[LC_{kp}, ML_{kp}[$ et soit UB^L_{gksp} la borne supérieure sur l'intervalle gauche ;
    si $UB^L_{gksp} \leq LB_{gksp}$, alors poser $ML_{kp} = 0$;
  sinon
    Appel de l'algorithme 15 pour la réduction de l'intervalle de droite $]MU_{kp}, UC_{kp}]$;
    si LB_{gksp} est améliorée, Alors Appeler L'algorithme 13 afin de réduire l'intervalle de droite $]MU_{kp}, UC_{kp}]$;
    Appel de l'algorithme 12 sur $]MU_{kp}, UC_{kp}]$ et soit UB^U_{gksp} la borne supérieure sur l'intervalle de droite ;
    si $UB^U_{gksp} \leq LB_{gksp}$ alors poser $MU_{kp} = C_{gksp}$;
  finsi
  Poser UB_{gksp} égale au max $\{UB^L_{gksp},\ UB^U_{gksp}\}$;
fin du « Tant que »
Retourner Une solution optimale de P_{gksp}.

L'algorithme 16 fixe l'intervalle initial de recherche à $[0,\ C_{gksp}]$ (étape 1). Ensuite, il détermine la borne supérieure de départ du P_{gksp} notée UB_{gksp} suivant la Section 3.2.1. Par la suite, la capacité CT relative à cette borne est considérée comme le premier point de discontinuité (étapes de 2 à 3). Suivant cette capacité, l'étape 4 permet de calculer la borne inférieure LB_{gksp} pour le problème P_{gksp} par la résolution optimale successive des deux problèmes $P_{kp}(CT)$ et $P_{ksp}(C_{gksp}-CT)$ respectivement. Par conséquent, la borne inférieure LB_{gksp} trouvée est utilisée pour réduire l'intervalle globale de recherche $[LC_{kp}, UC_{kp}]$ (étape 5). A l'étape 6 de l'algorithme, l'intervalle en cours $[LC_{kp}, UC_{kp}]$ est subdivisé en deux parties : l'intervalle de gauche $[LC_{kp}, ML_{kp}[$ et l'intervalle de droite $]MU_{kp}, UC_{kp}]$. La boucle principale (étapes de 7 à 20) s'arrête quand l'optimalité du P_{gksp} est prouvée. Par ailleurs, selon la qualité de la borne supérieure calculée sur l'intervalle de gauche UB^{L}_{gksp} et la borne supérieure calculée sur l'intervalle de droite UB^{U}_{gksp}, l'algorithme explore alternativement l'intervalle de gauche $[LC_{kp}, ML_{kp}[$, puis l'intervalle de droite $]MU_{kp}, UC_{kp}]$ (étape 8). A l'étape 11 (resp. étape 16), l'algorithme 12 est utilisé pour le calcul de la nouvelle borne supérieure sur l'intervalle de gauche (resp. l'intervalle de droite) réduit. De plus, les nouvelles bornes supérieures obtenues sont utilisées pour vérifier si l'un des deux intervalles de gauche ou de droite peut être fermé ou réduit à zéro (étapes de 12 à 17).
A l'étape 19, la mise à jour de la borne supérieure globale UB_{gksp} est effectuée selon les bornes supérieures obtenues sur les sous-intervalles induits. Enfin, l'algorithme 16 sort avec une solution optimale pour le problème P_{gksp}.

3.4 Partie expérimentale

Dans cette section, nous évaluons la performance de la méthode exacte proposée (noté EM : "Exact Method") sur un ensemble d'instances de la littérature. Les résultats obtenue par cette méthode sont comparés à ceux réalisés par la méthode exacte de Fujiimoto et Yamada [6] (noté FY) et ceux obtenus par le solveur Cplex (noté Cplex). Cette étude expérimentale est réalisée sur des instances de différentes tailles (voir la Section 3.1). Tous les sous-problèmes, représentant des séries de problèmes KP et KSP, ont été résolus respectivement par la méthode exacte de Martello *et al.* [16] et la méthode exacte proposée par Hifi et Wu [12]. La méthode proposée a été codée en C++ et exécuté sur un processeur Intel Xeon avec 3.06 GZ.

3.4.1 Description des instances

Les instances testées sont obtenues par application du générateur de Fujimoto et Yamada [6]. Pour ces instances, le nombre d'objets n_{total} ($n_{total} = \sum_{i=1}^{n} m_i$) varie dans l'intervalle discret $2^9, 2^{10}, ..., 2^{15}$, le nombre de classes est fixé à $n = 2,\ 4$ et 8, et la capacité du sac C_{gksp} est égale à $200 \times n_{total}$. Le rapport du nombre d'objets communs relativement a n_{total} est noté par $\lambda = m_0/n_{total}$, où $\lambda = 1/2,\ 1/4$ et $1/8$ respectivement et, le nombre d'objets non-communs est fixé à $m_i = (n_{total} - m_0/n$, pour $i = 1, \ldots, n$. Selon le degré de corrélation entre les profits et les poids des éléments, les instances sont réparties en trois groupes :

- Le premier groupe est composé d'instances "non-corrélées" (noté par U), où w_j et p_j sont mutuellement indépendants et uniformément générés dans l'intervalle $[1,\ 1000]$.
- Le deuxième groupe représente les instances "corrélées" (noté par W). Ces instances sont générées de la même manière que les instances "non-corrélées" à la différence que les valeurs des profits p_j sont uniformément générées dans l'intervalle $[w_j, w_j + 200]$ pour $j = 1, \ldots, n_{total}$.
- Le troisième groupe contient des instances "fortement corrélées" (noté par S). Ces instances sont générées de la même façon que les autres groupes sauf que les valeurs des profits p_j associés aux éléments sont égales à $p_j = w_j + 100$, pour $j = 1, \ldots, n_{total}$.

Afin d'identifier les instances générées à partir des différents paramètres, chaque groupe d'instances est abrégé comme suit : $G{-}k{-}\lambda{-}n$, où G indique le degré de corrélation (c-à-d U, W et S), k indique le nombre total des éléments d'une instance (c-à-d $n_{total} = 2^k$), λ est le rapport de la distribution entre les éléments communs et non-communs (par exemple, si $n_{total} = 100$ et $\lambda = 1/2$, alors l'instance contient 50 éléments communs) et n correspond au nombre de classes des objets non-communs. De plus, pour chaque groupe et en fonction du nombre d'éléments, dix instances aléatoires sont considérées.

Pour le reste de la partie expérimentale, les notations suivantes seront utilisées pour l'ensemble des tables :

- La colonne "Groupe" affiche l'indice du groupe et chaque ligne correspond à la valeur moyenne sur dix instances.
- La colonne "Nopt" affiche, pour chaque groupe, le nombre de solutions optimales atteint par l'algorithme correspondant.
- Les notations N_{ksp} (resp. N_{kp}) affiche le nombre de problème de type KSP internes (resp. KP) résolus par l'algorithme correspondant.
- La colonne "cpu" affiche le temps d'exécution que nécessite l'algorithme pour résoudre l'instance, mesuré en secondes.
- Les colonnes associées à "UB", "Opt" et "Best" affiche la borne supé-

rieure, la valeur de la solution optimale et la meilleure borne inférieure réalisées par les méthodes considérées.

3.4.2 Performance de la méthode exacte sur les instances fortement corrélés

Dans cette partie, nous étudions la performance de la méthode EM par rapport à la méthode FY de Fujimoto et Yamada [6] pour des instances fortement corrélées.

La table 3.1 affiche les résultats obtenus par les deux méthodes EM et FY pour des instances de petite taille (instances de taille 2^9 et 2^{10}). Rappelons que dans la méthode FY, les auteurs ont utilisé une procédure de programmation dynamique, ce qui rend la méthode FY plus avantageuse puisqu'elle est simultanément en mesure d'atteindre le profit total maximum et d'utiliser la ressource minimum. Cependant, la méthode EM utilise l'algorithme hybride de Martello *et al.* [16] pour résoudre les problèmes KP, où seul le profit maximum est assuré. Par conséquent, afin de passer d'un point de discontinuité vers un autre point, EM doit résoudre deux problèmes de type KP : le premier pour déterminer le profit maximum et le second pour calculer la ressource minimale. Par conséquent, la méthode EM nécessite la résolution de plus de problèmes de type KP que la méthode FY.

Maintenant, en introduisant toutes les stratégies décrites à la Section 3.2, la méthode EM réussit de manière significative à réduire le nombre de problèmes de type KP qui doivent être résolus. Comme le montre la Table 3.1 (cf. colonne N_{kp}), on peut observer que EM est capable de résoudre moins de problèmes de type KP que FY pour plusieurs instances. En outre, FY ne parvient pas à prouver l'optimalité de la solution pour les deux instances appartenant aux deux groupes S-9-2-8 et S-10-4-8.

En particulier, l'algorithme FY n'a pas réussi à résoudre à l'optimalité la plupart des instances fortement corrélées, en particulier à chaque fois que la taille de l'instance devient plus importante. Pour cette raison, nous avons comparé la performance de la méthode EM par rapport à celle du solveur Cplex, où le temps d'exécution du Cplex a été fixé à 3600 secondes. A chaque fois que le Cplex dépasse la durée fixée, la solution produite par le solveur est retenue comme la meilleure valeur d'une solution (notée Best). Rappelons que, le contenu de toutes les table représente les valeurs moyennes des solutions de toutes les instances d'un même groupe. Les résultats de toutes les instances testées sont détaillés et disponibles dans l'annexe.

La Table 3.2 représente les résultats (la qualité de la solution et le temps moyen nécessaire) du Cplex et de EM sur les trois groupes d'instances.

	FY				EM			
Group	N_{opt}	N_{ksp}	N_{kp}	cpu	N_{opt}	N_{ksp}	N_{kp}	cpu
S-9-2-4	10	1	24.2	1.255	10	1	25.2	0.01
S-9-2-8	9	1	22.2	354.566	10	1	20.2	0.01
S-9-4-4	10	1	39.8	0.095	10	1	37.8	0.01
S-9-4-8	10	1	35.1	2.612	10	1	37	0.02
S-9-8-4	10	1	67.7	0.014	10	1	72	0.01
S-9-8-8	10	1	67.7	0.029	10	1	71	0.02
S-10-4-4	10	1	45.7	331.766	10	1	58	0.03
S-10-4-8	8	1.1	40.1	3373.485	10	1	44.4	0.04
S-10-8-4	10	1	72.4	0.73	10	1	78.2	0.03
S-10-8-8	10	1	61.4	4.29	10	1	74.6	0.03

TABLE 3.1 – Performance de la méthode EM comparée à celle de la méthode FY sur les instances du groupe S.

D'une part, on peut observer que EM est capable de résoudre des instances à l'optimum en un temps d'exécution moyen assez raisonnable, même pour les instances de grande taille (à savoir les instances contenant entre 2^{14} et 2^{15} éléments). D'autre part, le solveur Cplex n'arrive pas à fournir des solutions optimales pour certaines instances de petite taille, même avec une durée d'exécution limitée à une heure. Finalement, pour les instances fortement corrélées, la méthode EM nécessite une moyenne de 14.15 secondes pour résoudre toutes les instances à l'optimalité, alors que le solveur Cplex nécessite un temps d'exécution plus important (1916.59 secondes en moyenne) lorsqu'il arrive à les résoudre à l'optimum.

		Cplex		EM	
Group	UB	Opt/Best	cpu	Opt	cpu
S9	95191.11	95141.04	1248.88	95141.28	0.18
S10	189948.74	189897.44	1692.76	189897.78	0.32
S11	379759.48	379711.77	1698.49	379712.1	0.69
S12	760158.88	760112.11	1877.48	760112.48	2.99
S13	1520902.48	1520852.47	2393.5	1520852.83	9.13
S14	3042221.33	3042176.29	2254.82	3042176.37	30.96
S15	6080132.9	6080087.99	2250.17	6080088.09	54.8
Av.	1724044.99	1723997.02	1916.59	1723997.27	14.15

TABLE 3.2 – Performance de la méthode EM comparée au solveur Cplex sur les instances du groupe S .

3.4.3 Comportement de la méthode exacte sur les instances non-corrélées et faiblement corrélées

Dans ce qui suit, nous examinons la performance des trois méthodes EM, FY et Cplex sur les instances non-corrélées et faiblement corrélées. A partir des Tables 3.3 et 3.4, nous pouvons observer ce qui suit :

1. Les trois algorithmes sont capables de résoudre à l'optimum les instances du groupe U9 (à savoir les instances contenant 2^9 objets).
2. Les deux algorithmes FY et EM dominent le solveur Cplex puisqu'ils atteignent les solutions optimales pour toutes les instances des deux groupes U et W, contrairement au solveur Cplex.
3. Le solveur Cplex peut résoudre à l'optimum des instances avec un grand valeur pour λ, cependant, souvent il ne parvient pas à prouver l'optimalité des solutions même avec l'utilisation d'un temps d'exécution fixé à 3600 secondes.

		Cplex		FY		EM	
Group	UB	Opt/Best	cpu	Opt	cpu	Opt	cpu
U9	102397.58	102343.92	5.46	102343.92	0.01	102343.92	0.02
U10	205241.54	205205.17	218.65	205205.26	0.08	205205.26	0.06
U11	411636.98	411614.28	708.14	411614.81	0.5	411614.81	0.15
U12	821736.93	821722.34	1165.36	821723.37	3.27	821723.37	0.39
U13	1637092.06	1637082.52	1541.19	1637084.14	20.24	1637084.14	1.09
U14	3263163.19	3263156.36	1887.34	3263158.66	111.52	3263158.66	3.08
U15	6519329.42	6519324.49	2138.54	6519326.93	666.33	6519326.93	11.37
Av.	1851513.96	1851492.73	1094.95	1851493.87	114.56	1851493.87	2.31

TABLE 3.3 – Comportement de la méthode EM versus le Cplex et le FY sur les instances du groupe U.

Nous avons déjà mentionné que les deux algorithmes FY et EM résolvent des problèmes de type KP et KSP d'une façon différente. Dans la Table 3.5, nous affichons le temps d'exécution moyen que nécessitent les deux algorithmes et le nombre de fois que chaque algorithme résout les problèmes de type KP et KSP. A partir de lecture des résultats de la Table 3.5, on peut observer que :

1. FY et EM résolvent le même nombre de problèmes de type KP et KSP, en moyenne.
2. EM résout moins de problèmes de type KP pour les instances de petite taille (les instances contenant 2^9 jusqu'à 2^{12} éléments).

		Cplex		FY		EM	
Group	UB	Opt/Best	cpu	Opt	cpu	Opt	cpu
W9	96379.29	96360.47	552.03	96360.52	0.01	96360.52	0.01
W10	192289.02	192275.8	825.74	192276.1	0.08	192276.1	0.04
W11	384646.63	384636.24	1064.05	384636.98	0.32	384636.98	0.08
W12	768827.81	768819.3	1253.86	768820.33	1.23	768820.33	0.16
W13	1537176.98	1537170.86	1439.8	1537172.32	5.36	1537172.32	0.32
W14	3076173.3	3076169.08	1501.37	3076170.33	25.34	3076170.33	0.65
W15	6153569.07	6153565.03	1897.93	6153567.17	128.83	6153567.17	2.46
Av.	1744151.73	1744142.4	1219.25	1744143.39	23.02	1744143.39	0.53

TABLE 3.4 – Comportement de la méthode EM versus le Cplex et FY sur les instances du groupe W.

3. Le comportement de EM reste très intéressant sur les instances de grande taille. D'une part, le temps d'exécution moyen qu'il nécessite reste plus intéressant que celui que nécessite FY. D'autre part, la dominance devient plus important lorsque le nombre d'éléments non-communs est moins important (par exemple, $n = 2$: les instances représentant les groupes : U-15-2-2, 15-4-2-U, U-15-8 -2, W-15-2-2, 15-4-2-W et W-15-8-2, respectivement).

	FY			EM				FY			EM		
Group	N_{ksp}	N_{kp}	cpu	N_{ksp}	N_{kp}	cpu	Group	N_{ksp}	N_{kp}	cpu	N_{ksp}	N_{kp}	cpu
U9	2.83	579.28	0.01	3.00	644.64	0.02	W9	1.5	146.18	0.01	1.71	153.49	0.01
U10	2.84	1055.37	0.08	2.91	1188.09	0.06	W10	1.96	479.02	0.08	2.39	361.47	0.04
U11	3.28	1987.02	0.5	3.48	2232.77	0.15	W11	2.14	638.42	0.32	2.79	641.5	0.08
U12	3.34	3300.28	3.27	3.4	3659.62	0.39	W12	1.94	751.53	1.23	2.26	881.47	0.16
U13	3.19	4955.61	20.24	3.2	5362.39	1.09	W13	1.89	844.91	5.36	2.13	1038.02	0.32
U14	2.4	6484.74	111.52	2.44	6969.06	3.08	W14	1.9	918.63	25.34	1.96	1141.4	0.65
U15	2.06	7569.29	666.33	2.03	7514.84	11.37	W15	1.69	870.91	128.83	1.77	1143.96	2.46
Av.	2.85	3704.51	114.56	2.92	3938.77	2.31	Av.	1.86	664.23	23.02	2.14	765.9	0.53

TABLE 3.5 – Performance des méthodes EM et FY sur les deux groupes d'instances U et W.

3.5 Conclusion

Dans ce chapitre, nous nous sommes intéressé à la résolution optimale du problème de la distribution équitable généralisée. Pour ce faire, nous avons

proposé une nouvelle méthode exacte qui combine différents ingrédients afin d'accélérer le processus de recherche. Dans un premier temps, une méthode de décomposition a été proposée afin de reformuler le problème original en deux sous-problèmes, à savoir, le problèmes du sac à dos classique et le problème de la distribution équitable. Ensuite, un algorithme exact, basé sur l'énumération de certaines combinaisons entre les deux sous-problèmes, a été proposé. Afin d'améliorer les performances de cet algorithme, plusieurs stratégies d'évaluation et de réduction ont été étudiées. La performance de l'algorithme a été testé sur trois groupes d'instances de référence tirées de la littérature. Les résultats réalisés par la méthode proposée ont été comparés à ceux obtenus par le solveur Cplex et ceux fournis par un des meilleurs algorithmes de la littérature. Finalement, dans une partie expérimentale, il a été montré que l'algorithme proposé était capable de produire des résultats qui dominaient ceux réalisés par les deux méthodes précédemment citées.

Conclusion générale

Les travaux présentés dans cette thèse appartiennent au domaine d'optimisation combinatoire. Nous nous sommes intéressés plus particulièrement à des problèmes de type sac à dos.

Nous avons proposé l'heuristique SSKSP pour le problème de la distribution équitable qui repose essentiellement sur la recherche dispersée. Nous avons utilisé des stratégies basées sur les caractéristiques du problème du sac à dos pour générer une population élite ainsi qu'une décomposition de la population en sous-populations de différentes tailles. Pour améliorer la qualité des solutions, nous avons incorporé deux méthodes de combinaison. Notre approche permet d'obtenir des résultats proches en moyenne à l'optimum.

Par la suite, nous avons proposé un algorithme exact pour résoudre le problème de la distribution équitable généralisée. Nous avons appliqué une méthode de décomposition du problème original en deux problèmes : de sac à dos et de distribution équitable. Nous avons proposé un algorithme exact basé sur une énumération des combinaisons cadrées par plusieurs stratégies entre les deux problèmes. Les résultats donnés par la méthode proposée ont été comparés à ceux atteints par le solveur Cplex et ceux fournis par le meilleur algorithme publié dans la littérature.

Pour ce qui est des perspectives de ce travail relatives au chapitre 2, il s'agit de mettre en oeuvre des heuristiques efficaces de recherche de la population initiale telle que l'encadrement des variables. L'idée de cette approche consiste à résoudre le problème relaxé et extraire une borne qui nous permet de faire un encadrement a chaque fois en fixant une variable. On résout le problème relaxé jusqu'à l'obtention d'une population de départ. Ainsi, des techniques de reproduction de nouvelles solutions pour la méthode de combinaison sont envisagées telle que le chemin reliant (Path Relinking en anglais).

Pour ce qui est de la méthode exacte du chapitre 3, nous comptons construire une heuristique dans le même sens. L'idée est de remplacer les méthodes exactes pour la résolution du KSP et du KP par exemple avec des heuristiques efficaces afin de diminuer l'effort de calcul et pouvoir résoudre des instances de grande taille avec un temps moyen d'exécution faible.

Annexes

		Cplex		FY		EM	
Group	UB	Opt/Best	Cpu	Opt	Cpu	Opt	Cpu
U-9-2-2	146743.1	146714.5	0.12	146714.5	0.04	146714.5	0.03
U-9-2-4	120759.7	120737.0	0.13	120737.0	0.01	120737.0	0.01
U-9-2-8	107521.1	107501.9	0.11	107501.9	0.01	107501.9	0.01
U-9-4-2	132227.6	132158.6	0.79	132158.6	0.03	132158.6	0.04
U-9-4-4	90548.1	90499.1	4.19	90499.1	0.01	90499.1	0.02
U-9-4-8	69094.1	69054.8	1.44	69054.8	0.01	69054.8	0.01
U-9-8-2	127440.6	127324.0	1.08	127324.0	0.02	127324.0	0.05
U-9-8-4	76690.2	76618.1	32.64	76618.1	0.01	76618.1	0.02
U-9-8-8	50553.7	50487.3	8.60	50487.3	0.01	50487.3	0.01
U-10-2-2	293409.6	293390.3	0.35	293390.3	0.21	293390.3	0.09
U-10-2-4	242938.6	242922.8	0.60	242922.8	0.04	242922.8	0.04
U-10-2-8	215374.6	215359.2	0.45	215359.2	0.01	215359.2	0.02
U-10-4-2	263859.0	263822.5	2.40	263822.5	0.20	263822.5	0.10
U-10-4-4	183062.8	183036.8	10.65	183036.8	0.03	183036.8	0.04
U-10-4-8	138201.8	138172.9	14.24	138172.9	0.01	138172.9	0.02
U-10-8-2	254193.5	254098.6	16.89	254098.6	0.16	254098.6	0.12
U-10-8-4	155469.9	155423.0	806.76	155423.0	0.04	155423.0	0.05
U-10-8-8	100664.1	100620.4	1115.49	100621.2	0.01	100621.2	0.02
U-11-2-2	588381.2	588371.3	1.17	588371.3	1.06	588371.3	0.23
U-11-2-4	486051.6	486041.3	1.60	486041.3	0.21	486041.3	0.08
U-11-2-8	432305.2	432295.6	1.76	432295.6	0.04	432295.6	0.05
U-11-4-2	529653.4	529628.4	8.85	529628.4	1.41	529628.4	0.29
U-11-4-4	366049.2	366030.1	73.08	366030.1	0.23	366030.1	0.10
U-11-4-8	278313.1	278297.7	146.14	278297.7	0.05	278297.7	0.05
U-11-8-2	510048.1	509993.2	200.54	509993.2	1.24	509993.2	0.33
U-11-8-4	310570.5	310537.3	2686.08	310539.8	0.19	310539.8	0.13
U-11-8-8	203360.5	203333.6	3254.03	203335.9	0.03	203335.9	0.05
U-12-2-2	1173884.8	1173879.1	2.94	1173879.1	6.83	1173879.1	0.70
U-12-2-4	969655.4	969650.1	3.29	969650.1	0.90	969650.1	0.16
U-12-2-8	860277.8	860272.5	3.51	860272.5	0.18	860272.5	0.10
U-12-4-2	1058334.0	1058319.2	21.98	1058319.2	8.62	1058319.2	0.73
U-12-4-4	732138.7	732128.8	267.39	732128.8	1.39	732128.8	0.34
U-12-4-8	554371.9	554362.4	946.47	554362.6	0.18	554362.6	0.09
U-12-8-2	1019664.7	1019626.7	2040.95	1019627.4	10.13	1019627.4	1.00
U-12-8-4	622064.9	622041.7	3600.88	622046.1	1.04	622046.1	0.28
U-12-8-8	405240.2	405220.6	3600.87	405224.5	0.16	405224.5	0.10
U-13-2-2	2334470.7	2334467.3	5.74	2334467.3	41.85	2334467.3	1.58
U-13-2-4	1934441.2	1934438.4	7.73	1934438.4	6.31	1934438.4	0.49
U-13-2-8	1717486.3	1717483.5	6.85	1717483.5	1.10	1717483.5	0.20
U-13-4-2	2101187.4	2101178.8	357.47	2101178.8	49.31	2101178.8	2.52
U-13-4-4	1461334.5	1461328.8	1044.76	1461328.9	8.26	1461328.9	0.75
U-13-4-8	1108326.9	1108321.4	1792.36	1108321.9	0.86	1108321.9	0.20
U-13-8-2	2023252.3	2023225.8	3454.01	2023229.5	66.36	2023229.5	2.61
U-13-8-4	1242253.8	1242237.8	3600.94	1242243.2	7.30	1242243.2	1.23
U-13-8-8	811075.4	811060.9	3600.82	811065.8	0.80	811065.8	0.25
U-14-2-2	4657532.7	4657530.9	11.22	4657530.9	184.92	4657530.9	2.88
U-14-2-4	3854013.0	3854011.8	11.61	3854011.8	28.22	3854011.8	0.78
U-14-2-8	3430109.6	3430108.0	11.35	3430108.0	3.85	3430108.0	0.36
U-14-4-2	4190336.2	4190331.6	779.10	4190331.6	309.85	4190331.6	6.06
U-14-4-4	2903914.8	2903911.1	3014.60	2903911.7	48.77	2903911.7	1.86
U-14-4-8	2213427.6	2213424.6	2354.79	2213424.8	5.27	2213424.8	0.48
U-14-8-2	4034962.8	4034941.8	3601.28	4034948.8	366.03	4034948.8	11.82
U-14-8-4	2464096.2	2464082.7	3601.05	2464089.6	51.62	2464089.6	2.80
U-14-8-8	1620075.8	1620064.7	3601.02	1620070.7	5.11	1620070.7	0.71
U-15-2-2	9317274.7	9317273.9	14.14	9317273.9	1127.39	9317273.9	12.98
U-15-2-4	7703675.8	7703675.5	13.46	7703675.5	177.12	7703675.5	1.12
U-15-2-8	6851985.7	6851985.2	15.60	6851985.2	21.35	6851985.2	0.61
U-15-4-2	8380862.5	8380859.8	2390.23	8380860.0	1683.36	8380860.0	29.59
U-15-4-4	5799708.6	5799705.6	3066.46	5799706.8	237.39	5799706.8	3.07
U-15-4-8	4411426.9	4411424.4	2941.44	4411425.5	25.65	4411425.5	0.83
U-15-8-2	8070065.2	8070049.5	3602.71	8070056.8	2395.84	8070056.8	47.37
U-15-8-4	4917489.6	4917479.8	3601.56	4917485.8	290.68	4917485.8	5.02
U-15-8-8	3221475.8	3221466.7	3601.24	3221472.9	38.22	3221472.9	1.73

TABLE 3.6 – Performance de EM contre Cplex et FY sur groupe U.

		Cplex		FY		EM	
Group	UB	Opt/Best	Cpu	Opt	Cpu	Opt	Cpu
W-9-2-2	128132	128118	0.25	128118	0.08	128118	0.04
W-9-2-4	105924.2	105913.6	0.71	105913.6	0.01	105913.6	0.01
W-9-2-8	88045.1	88036.4	1.02	88036.4	0.01	88036.4	0.01
W-9-4-2	127531	127498.3	0.27	127498.3	0.03	127498.3	0.02
W-9-4-4	91543	91525.2	12.03	91525.2	0.01	91525.2	0.01
W-9-4-8	63378.8	63363.7	152.03	63363.7	0.01	63363.7	0.01
W-9-8-2	127460.1	127451.1	0.13	127451.1	0.01	127451.1	0.01
W-9-8-4	84349.9	84316.7	1614.29	84317	0.01	84317	0.01
W-9-8-8	51049.5	51021.2	3187.53	51021.4	0.01	51021.4	0.01
W-10-2-2	255486.3	255476.7	0.52	255476.7	0.32	255476.7	0.11
W-10-2-4	211682.7	211677.3	1.09	211677.3	0.01	211677.3	0.01
W-10-2-8	176212	176207.6	0.87	176207.6	0.01	176207.6	0.02
W-10-4-2	254153.6	254126.8	0.45	254126.8	0.18	254126.8	0.08
W-10-4-4	182551.1	182539.9	228.03	182539.9	0.01	182539.9	0.01
W-10-4-8	126686.4	126677.1	470.86	126677.1	0.01	126677.1	0.02
W-10-8-2	253920.2	253905.4	0.29	253905.4	0.19	253905.4	0.04
W-10-8-4	167989	167968.5	3128.67	167969.2	0.01	167969.2	0.01
W-10-8-8	101919.9	101902.9	3600.87	101904.9	0.01	101904.9	0.02
W-11-2-2	510867.9	510862.1	1.04	510862.1	1.47	510862.1	0.26
W-11-2-4	423612.7	423609.4	2.72	423609.4	0.02	423609.4	0.03
W-11-2-8	352011.6	352008.8	7.04	352008.8	0.01	352008.8	0.03
W-11-4-2	508400.6	508381.4	1.34	508381.4	0.82	508381.4	0.18
W-11-4-4	365983.4	365976.5	942.67	365976.7	0.01	365976.7	0.03
W-11-4-8	252725.8	252720.5	1457.94	252720.7	0.01	252720.7	0.03
W-11-8-2	507972.6	507949.3	0.71	507949.3	0.53	507949.3	0.10
W-11-8-4	337170.2	337157.1	3562.32	337158.8	0.01	337158.8	0.03
W-11-8-8	203074.9	203061.1	3600.71	203065.6	0.01	203065.6	0.03
W-12-2-2	1022023.4	1022020.3	1.29	1022020.3	5.31	1022020.3	0.46
W-12-2-4	845371.1	845369	5.14	845369	0.06	845369	0.06
W-12-2-8	703864.7	703863.4	9.21	703863.4	0.02	703863.4	0.06
W-12-4-2	1017210.9	1017199.1	3.12	1017199.1	3.41	1017199.1	0.35
W-12-4-4	729541.1	729537.3	1407.73	729537.8	0.03	729537.8	0.06
W-12-4-8	506110.1	506106.7	2653.59	506107.4	0.02	506107.4	0.06
W-12-8-2	1016476.9	1016445.5	3.21	1016445.5	2.16	1016445.5	0.27
W-12-8-4	671626.5	671615.2	3600.77	671619.9	0.03	671619.9	0.05
W-12-8-8	407225.6	407217.2	3600.67	407220.6	0.02	407220.6	0.06
W-13-2-2	2043745.3	2043743.5	3.15	2043743.5	23.14	2043743.5	0.82
W-13-2-4	1691967.6	1691966.5	10.50	1691966.5	0.11	1691966.5	0.11
W-13-2-8	1407123.5	1407122.9	16.19	1407122.9	0.07	1407122.9	0.12
W-13-4-2	2033539.5	2033532.2	6.98	2033532.2	14.93	2033532.2	0.81
W-13-4-4	1459937.6	1459934.2	3049.10	1459935.5	0.08	1459935.5	0.11
W-13-4-8	1010418.7	1010416.5	2641.20	1010417.4	0.05	1010417.4	0.11
W-13-8-2	2031873.1	2031851.6	29.34	2031851.6	9.73	2031851.6	0.55
W-13-8-4	1343919.8	1343910.5	3600.84	1343916.2	0.07	1343916.2	0.10
W-13-8-8	812067.7	812059.8	3600.86	812065.1	0.04	812065.1	0.11
W-14-2-2	4089951	4089950.1	5.55	4089950.1	104.01	4089950.1	1.31
W-14-2-4	3385547.6	3385547	11.46	3385547	0.32	3385547	0.23
W-14-2-8	2816095.4	2816095.1	63.40	2816095.1	0.12	2816095.1	0.25
W-14-4-2	4069868.2	4069864	18.11	4069864	76.98	4069864	1.66
W-14-4-4	2921317.9	2921316	3134.82	2921316.7	0.29	2921316.7	0.21
W-14-4-8	2022000	2021998.4	2656.52	2021999.2	0.11	2021999.2	0.23
W-14-8-2	4066621.6	4066606.8	420.62	4066606.8	45.87	4066606.8	1.53
W-14-8-4	2689205.5	2689198.8	3600.99	2689203.5	0.26	2689203.5	0.21
W-14-8-8	1624952.5	1624945.5	3600.90	1624950.6	0.11	1624950.6	0.22
W-15-2-2	8183629.1	8183628.6	11.97	8183628.6	563.09	8183628.6	4.00
W-15-2-4	6771352	6771351.5	378.94	6771351.5	0.99	6771351.5	0.47
W-15-2-8	5632025.4	5632025.3	399.52	5632025.3	0.37	5632025.3	0.50
W-15-4-2	8143781.8	8143779	84.18	8143779	346.61	8143779	8.13
W-15-4-4	5841755.1	5841753.7	3580.81	5841754.4	0.86	5841754.4	0.43
W-15-4-8	4044565.2	4044563.2	3023.05	4044564.8	0.33	4044564.8	0.46
W-15-8-2	8137227.3	8137216.8	2399.57	8137217.6	245.88	8137217.6	7.30
W-15-8-4	5376955.1	5376946.8	3601.95	5376953.6	0.94	5376953.6	0.41
W-15-8-8	3250830.6	3250820.4	3601.40	3250829.7	0.38	3250829.7	0.44

TABLE 3.7 – Performance de EM contre Cplex et FY sur groupe W.

	FY			EM				FY			EM		
Group	N_{ksp}	N_{kp}	Cpu	N_{ksp}	N_{kp}	Cpu	Group	N_{ksp}	N_{kp}	Cpu	N_{ksp}	N_{kp}	Cpu
U-9-2-2	3.9	467	0.04	3.8	574.3	0.03	W-9-2-2	3.3	593.6	0.08	5.2	674.9	0.04
U-9-2-4	2.6	188.2	0.01	2.1	245.4	0.01	W-9-2-4	1	18.5	0.01	1	19	0.01
U-9-2-8	1.8	68.9	0.01	1.6	89.4	0.01	W-9-2-8	1	18.7	0.01	1	17.4	0.01
U-9-4-2	4.8	1189.7	0.03	5.7	1367.4	0.04	W-9-4-2	3.2	464.4	0.03	4.2	449.6	0.02
U-9-4-4	2.9	525	0.01	2.7	599	0.02	W-9-4-4	1	35	0.01	1	34.4	0.01
U-9-4-8	1.6	130.1	0.01	1.4	144.4	0.01	W-9-4-8	1	33.4	0.01	1	29.6	0.01
U-9-8-2	4.4	1843.3	0.02	6.1	1955.5	0.05	W-9-8-2	1	17.5	0.01	0	14.1	0.01
U-9-8-4	2.3	627.2	0.01	2.4	645	0.02	W-9-8-4	1	70.7	0.01	1	71.8	0.01
U-9-8-8	1.2	174.1	0.01	1.2	181.4	0.01	W-9-8-8	1	63.8	0.01	1	70.6	0.01
U-10-2-2	4.2	1103.9	0.21	4.4	1413.9	0.09	W-10-2-2	3.3	1058.6	0.32	4.2	1242.7	0.11
U-10-2-4	2	386.2	0.04	2.2	529	0.04	W-10-2-4	1	18.8	0.01	1	19	0.01
U-10-2-8	2.1	116.9	0.01	2	165.4	0.02	W-10-2-8	1	16.9	0.01	1	18.6	0.02
U-10-4-2	3.7	1755.8	0.20	3.4	1977.8	0.10	W-10-4-2	4.2	1131	0.18	7	1147.6	0.08
U-10-4-4	3	576.2	0.03	2.4	642.8	0.04	W-10-4-4	1	34.9	0.01	1	36	0.01
U-10-4-8	1.6	226.9	0.01	1.5	281.8	0.02	W-10-4-8	1	30.9	0.01	1	36.4	0.02
U-10-8-2	5.1	3662.5	0.16	6.6	3900.2	0.12	W-10-8-2	4.1	1885.9	0.19	4.3	610.9	0.04
U-10-8-4	2.6	1326	0.04	2.4	1400.1	0.05	W-10-8-4	1	71	0.01	1	74.8	0.01
U-10-8-8	1.3	343.9	0.01	1.3	381.8	0.02	W-10-8-8	1	63.2	0.01	1	67.2	0.02
U-11-2-2	4.3	1693.7	1.06	3.9	2204	0.23	W-11-2-2	3.6	1662.3	1.47	4.3	1991.6	0.26
U-11-2-4	2.6	561.2	0.21	2.5	813.7	0.08	W-11-2-4	1	17.3	0.02	1	19.8	0.03
U-11-2-8	1.9	236.9	0.04	1.9	366.9	0.05	W-11-2-8	1	17.8	0.01	1	19.4	0.03
U-11-4-2	5.3	3816.3	1.41	5.7	4333.3	0.29	W-11-4-2	4.8	1810.1	0.82	6.1	2063	0.18
U-11-4-4	3.6	1220.4	0.23	3.5	1390.6	0.10	W-11-4-4	1	31.1	0.01	1	33.4	0.03
U-11-4-8	1.9	378	0.05	1.8	475.7	0.05	W-11-4-8	1	31.4	0.01	1	35.2	0.03
U-11-8-2	4.6	7029.3	1.24	6.3	7408.3	0.33	W-11-8-2	4.9	2051.4	0.53	8.7	1464.7	0.10
U-11-8-4	3.7	2381	0.19	3.9	2508.2	0.13	W-11-8-4	1	63.9	0.01	1	74.4	0.03
U-11-8-8	1.6	566.4	0.03	1.8	594.2	0.05	W-11-8-8	1	60.5	0.01	1	72	0.03
U-12-2-2	3.7	2713.3	6.83	3.4	3424.9	0.70	W-12-2-2	3.6	1522.6	5.31	4	2049.6	0.46
U-12-2-4	2.5	674.7	0.90	2.3	986.3	0.16	W-12-2-4	1.1	18.3	0.06	1.1	18.2	0.06
U-12-2-8	1.6	332.7	0.18	1.5	519	0.10	W-12-2-8	1	15.9	0.02	1	20.2	0.06
U-12-4-2	5.4	5723.3	8.62	5.6	6342.7	0.73	W-12-4-2	3.2	2150.3	3.41	3.8	2611.1	0.35
U-12-4-4	2.7	2056.1	1.39	2.7	2266.6	0.34	W-12-4-4	1	29.6	0.03	1	38.6	0.06
U-12-4-8	1.3	441.1	0.18	1.3	625	0.09	W-12-4-8	1	28.2	0.02	1	38.6	0.06
U-12-8-2	7.4	13466.4	10.13	8	14134.8	1.00	W-12-8-2	4.6	2883.1	2.16	6.4	3013.7	0.27
U-12-8-4	3.7	3443.2	1.04	3.9	3710.1	0.28	W-12-8-4	1	59.1	0.03	1	70.6	0.05
U-12-8-8	1.8	851.7	0.16	1.9	927.2	0.10	W-12-8-8	1	56.7	0.02	1	72.6	0.06
U-13-2-2	3.5	4041.2	41.85	3.4	4401.9	1.58	W-13-2-2	2.4	1557.7	23.14	2.8	2193.9	0.82
U-13-2-4	2.1	1437.8	6.31	2.3	1901	0.49	W-13-2-4	1	14.6	0.11	1	19	0.11
U-13-2-8	2.6	624.2	1.10	2.3	778.3	0.20	W-13-2-8	1	11.2	0.07	1	18	0.12
U-13-4-2	3.8	8096.1	49.31	3.8	8985.5	2.52	W-13-4-2	4	2460.4	14.93	4.5	3041	0.81
U-13-4-4	2.7	2825.8	8.26	2.7	3093.1	0.75	W-13-4-4	1	27.4	0.08	1	35.6	0.11
U-13-4-8	1.8	572	0.86	1.7	785.5	0.20	W-13-4-8	1	26.6	0.05	1	36.8	0.11
U-13-8-2	7.6	20408.4	66.36	8.1	21273.2	2.61	W-13-8-2	4.6	3403.9	9.73	5.9	3851.9	0.55
U-13-8-4	3.1	5453	7.30	3	5810.6	1.23	W-13-8-4	1	50.6	0.07	1	73	0.10
U-13-8-8	1.5	1142	0.80	1.5	1232.4	0.25	W-13-8-8	1	51.8	0.04	1	73	0.11
U-14-2-2	2.5	3955.5	184.92	2.6	4207.8	2.88	W-14-2-2	2.1	1460.9	104.01	2	1727.2	1.31
U-14-2-4	1.5	1712.9	28.22	1.5	1593.1	0.78	W-14-2-4	1	10.6	0.32	1	17.2	0.23
U-14-2-8	1	742.4	3.85	1	752.7	0.36	W-14-2-8	1	7.7	0.12	1	16.6	0.25
U-14-4-2	4.1	11746.6	309.85	4	12661.9	6.06	W-14-4-2	4	2757.4	76.98	4.1	3593.2	1.66
U-14-4-4	1.9	3654.6	48.77	1.9	4386.3	1.86	W-14-4-4	1	24.4	0.29	1	38.4	0.21
U-14-4-8	1.3	951.9	5.27	1.3	1285.9	0.48	W-14-4-8	1	17.9	0.11	1	37.2	0.23
U-14-8-2	5.9	25111.2	366.03	6.3	26582.4	11.82	W-14-8-2	5	3893	45.87	5.5	4698.8	1.53
U-14-8-4	2.2	8590.2	51.62	2.2	8964.8	2.80	W-14-8-4	1	47.6	0.26	1	71.2	0.21
U-14-8-8	1.2	1897.4	5.11	1.2	2286.6	0.71	W-14-8-8	1	48.2	0.11	1	72.8	0.22
U-15-2-2	2.1	5016.7	1127.39	2	3274.7	12.98	W-15-2-2	1.5	1570	563.09	1.8	918.6	4.00
U-15-2-4	1.4	2949	177.12	1.6	973.3	1.12	W-15-2-4	1	8.7	0.99	1	17.8	0.47
U-15-2-8	1	1254.4	21.35	1	487.2	0.61	W-15-2-8	1	6.5	0.37	1	14.8	0.50
U-15-4-2	3.1	11554.8	1683.36	3.1	13335.2	29.59	W-15-4-2	3.3	2038.9	346.61	3.5	3629.6	8.13
U-15-4-4	1.4	3489	237.39	1.4	3947.5	3.07	W-15-4-4	1	14.9	0.86	1	32.8	0.43
U-15-4-8	1.3	1203.9	25.65	1.3	1128.9	0.83	W-15-4-8	1	12.7	0.33	1	34.4	0.46
U-15-8-2	5.2	29784.8	2395.84	5	31326.1	47.37	W-15-8-2	4.4	4106.7	245.88	4.6	5508	7.30
U-15-8-4	1.7	9681.8	290.68	1.6	9871.6	5.02	W-15-8-4	1	43.5	0.94	1	68	0.41
U-15-8-8	1.3	3189.2	38.22	1.3	3289.1	1.73	W-15-8-8	1	36.3	0.38	1	71.6	0.44

TABLE 3.8 – Performance de EM contre FY sur les groupes U et W.

		Cplex		EM				Cplex		EM	
Group	UB	Opt/Best	Cpu	Opt	Cpu	Group	UB	Opt/Best	Cpu	Opt	Cpu
S-9-2-2	125768.9	125728.1	723.54	125728.1	0.99	S-12-4-8	498940.3	498889.6	1452.82	498889.6	0.10
S-9-2-4	104882.7	104850	0.10	104850	0.01	S-12-8-2	1001649	1001611.8	374.42	1001611.8	2.16
S-9-2-8	86385	86340.6	360.29	86340.6	0.01	S-12-8-4	673770.2	673714.8	3600.56	673717	0.12
S-9-4-2	125314.5	125248.8	360.28	125248.8	0.37	S-12-8-8	403849.7	403802.3	2899.86	403803.4	0.16
S-9-4-4	91196.4	91144.5	2164.28	91144.7	0.01	S-13-2-2	2011571.2	2011539.8	3245.77	2011540.4	66.50
S-9-4-8	62706.9	62651.3	1821.90	62651.3	0.02	S-13-2-4	1676445.7	1676389.2	1080.64	1676389.2	0.24
S-9-8-2	125243.9	125190	0.05	125190	0.18	S-13-2-8	1380275.5	1380218.4	360.78	1380218.4	0.22
S-9-8-4	84351.8	84300.9	2520.97	84301.3	0.01	S-13-4-2	2004641.6	2004595.4	3243.24	2004596.6	12.12
S-9-8-8	50869.9	50815.2	3288.51	50816.7	0.02	S-13-4-4	1456364.6	1456312.8	2161.75	1456312.8	0.22
S-10-2-2	251529.5	251504.1	827.85	251504.1	1.68	S-13-4-8	999663.8	999611.9	1441.05	999611.9	0.18
S-10-2-4	209165.1	209111.7	720.38	209111.7	0.02	S-13-8-2	2003479.1	2003430.5	2989.30	2003430.5	2.30
S-10-2-8	172458.8	172399.7	360.42	172399.7	0.02	S-13-8-4	1346322.3	1346269.5	3564.84	1346270.2	0.22
S-10-4-2	250558.7	250506.6	361.56	250506.6	0.70	S-13-8-8	809358.5	809304.7	3454.10	809305.5	0.21
S-10-4-4	181507	181448.8	3240.40	181448.9	0.03	S-14-2-2	4025311.5	4025286.2	3350.33	4025286.2	181.47
S-10-4-8	124983.6	124925.2	2162.40	124925.3	0.04	S-14-2-4	3353167.2	3353121.1	360.55	3353121.1	0.41
S-10-8-2	250411	250356.2	360.22	250356.2	0.37	S-14-2-8	2760665.9	2760619	720.73	2760619	0.39
S-10-8-4	167678.7	167625.7	3600.82	167626.7	0.03	S-14-4-2	4011431.8	4011388.5	3601.08	4011388.5	71.26
S-10-8-8	101246.3	101199	3600.81	101200.8	0.03	S-14-4-4	2912424	2912377.4	728.67	2912377.4	0.41
S-11-2-2	502628.3	502591.3	2224.07	502591.3	4.20	S-14-4-8	1998476.1	1998423.1	2161.80	1998423.1	0.69
S-11-2-4	418662.2	418605.9	360.47	418605.9	0.04	S-14-8-2	4009083.9	4009037.5	3279.00	4009037.5	23.23
S-11-2-8	344370.8	344325	360.51	344325	0.04	S-14-8-4	2692050.4	2692001.5	2884.76	2692002	0.41
S-11-4-2	500960.3	500917.8	1102.35	500917.8	1.18	S-14-8-8	1617381.2	1617332.3	3206.45	1617332.5	0.40
S-11-4-4	363709.6	363666.8	1801.09	363666.8	0.05	S-15-2-2	8051720	8051697.9	2541.84	8051697.9	235.48
S-11-4-8	249114.4	249062.8	1804.53	249062.9	0.05	S-15-2-4	6696722.4	6696672.4	1081.59	6696672.4	0.94
S-11-8-2	500677.8	500629.1	1440.66	500629.1	0.56	S-15-2-8	5519878.3	5519832.5	720.82	5519832.5	0.83
S-11-8-4	336228.7	336174.4	3344.15	336175.7	0.06	S-15-4-2	8024667.9	8024634.9	3338.54	8024634.9	179.20
S-11-8-8	201483.2	201432.8	2848.63	201434.4	0.05	S-15-4-4	5810762.6	5810711	2161.13	5810711	0.83
S-12-2-2	1005480.2	1005453.3	2169.63	1005453.3	18.55	S-15-4-8	3995779.1	3995724	1443.51	3995724	1.06
S-12-2-4	837917	837872.6	1080.85	837872.6	0.09	S-15-8-2	8020156.9	8020107.6	3601.24	8020107.6	73.16
S-12-2-8	689120.7	689055.5	1080.56	689055.5	0.10	S-15-8-4	5367779.8	5367731.3	2480.46	5367731.5	0.70
S-12-4-2	1002217.2	1002167.7	2430.30	1002167.7	5.56	S-15-8-8	3233729.1	3233680.3	2882.43	3233681	1.03
S-12-4-4	728485.6	728441.4	1808.33	728441.4	0.09						

TABLE 3.9 – Performance de EM contre Cplex sur le groupe S.

Bibliographie

[1] R Bellman. Dynamic programming. *Princeton University Press*, 28(1) :3–21, 1957. (Cité en page 12.)

[2] Vincent Boyer, Didier El Baz, and Moussa Elkihel. A dynamic programming method with lists for the knapsack sharing problem. *Computers & Industrial Engineering*, 61(2) :274–278, 2011. (Cité en pages 16 et 22.)

[3] J Randall Brown. The knapsack sharing problem. *Operations Research*, 27(2) :341–355, 1979. (Cité en page 14.)

[4] Isma Dahmani, Mhand Hifi, and Lei Wu. An exact decomposition algorithm for the generalized knapsack sharing problem. *European Journal of Operational Research*, 252(3) :761–774, 2016. (Cité en page 52.)

[5] George B Dantzig. Discrete-variable extremum problems. *Operations research*, 5(2) :266–288, 1957. (Cité en pages 3, 7 et 8.)

[6] Masako Fujimoto and Takeo Yamada. An exact algorithm for the knapsack sharing problem with common items. *European Journal of Operational Research*, 171(2) :693–707, 2006. (Cité en pages 24, 25, 54, 55, 57, 63, 64 et 65.)

[7] PC Gilmore and Ralph E Gomory. Multistage cutting stock problems of two and more dimensions. *Operations research*, 13(1) :94–120, 1965. (Cité en page 3.)

[8] Fred Glover. A template for scatter search and path relinking. In *European Conference on Artificial Evolution*, pages 1–51. Springer, 1997. (Cité en pages 36 et 38.)

[9] Mhand Hifi, Hedi M'Halla, and Slim Sadfi. An exact algorithm for the knapsack sharing problem. *Computers & Operations Research*, 32(5) :1311–1324, 2005. (Cité en pages 16, 18, 20 et 22.)

[10] Mhand Hifi and Catherine Roucairol. Approximate and exact algorithms for constrained (un) weighted two-dimensional two-staged cutting stock problems. *Journal of combinatorial optimization*, 5(4) :465–494, 2001. (Cité en page 3.)

[11] Mhand Hifi and Slim Sadfi. The knapsack sharing problem : An exact algorithm. *Journal of Combinatorial Optimization*, 6(1) :35–54, 2002. (Cité en pages 16, 17, 18, 19, 20, 29 et 48.)

[12] Mhand Hifi and Lei Wu. New upper bounds and exact methods for the knapsack sharing problem. *Applied Mathematics and Computation*, 227 :518–530, 2014. (Cité en pages 17, 20, 21 et 63.)

[13] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2) :277–292, 1974. (Cité en pages 10, 12 et 24.)

[14] D. Krzysztof and W. Stanislaw. Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research*, 28(1) :3–21, 1987. (Cité en page 3.)

[15] Manuel Laguna and Rafael Marti. *Scatter search : methodology and implementations in C*, volume 24. Springer Science & Business Media, 2012. (Cité en pages 40 et 48.)

[16] Silvano Martello, David Pisinger, and Paolo Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3) :414–424, 1999. (Cité en pages 63 et 65.)

[17] Silvano Martello and Paolo Toth. An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, 1(3) :169–175, 1977. (Cité en page 8.)

[18] Silvano Martello and Paolo Toth. *Knapsack problems : algorithms and computer implementations*. John Wiley & Sons, Inc., 1990. (Cité en pages 3, 9 et 28.)

[19] R Garey Michael and S Johnson David. Computers and intractability : a guide to the theory of np-completeness. *WH Freeman & Co., San Francisco*, 1979. (Cité en pages 4, 5 et 58.)

[20] Takeo Yamada and Mayumi Futakawa. Heuristic and reduction algorithms for the knapsack sharing problem. *Computers & operations research*, 24(10) :961–967, 1997. (Cité en pages 16, 17 et 18.)

[21] Takeo Yamada, Mayumi Futakawa, and Seiji Kataoka. Some exact algorithms for the knapsack sharing problem. *European Journal of Operational Research*, 106(1) :177–183, 1998. (Cité en pages 16, 18, 20 et 24.)