Contribution à la résolution du sac-à-dos à contraintes disjonctives

Mohamed Elhavedh Ould Ahmed Mounir

THÈSE

présentée

devant l'Université de Picardie Jules Verne

pour obtenir

le grade de : DOCTEUR DE L'UNIVERSITÉ DE PICARDIE JULES VERNE
Mention INFORMATIQUE

par

Mohamed Elhavedh OULD AHMED MOUNIR

Équipe d'accueil : MIS

Titre de la thèse :
Contribution à la résolution du sac-à-dos à contraintes disjonctives

À soutenir le 15 octobre 2009 devant la commission d'examen

M. :	Ridha	MAHJOUB	Professeur	Président
MM. :	Marc	BUI	Professeur,	Rapporteurs
	Imed	KACEM	Professeur	
MM. :	Vassilis	GIAKOUMAKIS	Professeur,	Examinateurs
	Stéphane	NEGRE	Maitre de Conférences	
M. :	Mhand	HIFI	Professeur	Directeur

Remerciements

Je tiens tout d'abord à remercier Monsieur Mhand HIFI, Professeur à l'Université de Picardie Jules verne, d'avoir accepté d'être mon directeur de thèse et d'avoir encadré mes travaux de recherche. Nos nombreuses réunions ont permis de faire progresser ce travail grâce à un échange constructif et cette collaboration fut pour ma part véritablement enrichissante.

Je remercie également Messieurs Imed KACEM, Professeur à l'Université de Metz, et Marc BUI, Professeur au EPHE, qui m'ont fait l'honneur de rapporter cette thèse.
Je remercie aussi Messieurs MAHJOUB Ridha, Professeur à l'Université de Paris Dauphine, GIAKOUMAKIS Vassilis, Professeur à l'Université de Picardie jules verne, et NEGRE Stéphane, memebre de l'INSEET, d'avoir accepté de participer à mon jury.

Je profite de ces quelques lignes pour remercier l'ensemble des personnes que j'ai côtoyé durant cette période au sein du laboratoire MIS dirigé par Monsieur Gilles KASSEL. J'insisterai plus particulièrement sur les doctorants et ex-doctorants du troisième étage.

Je n'oublie pas de remercier ceux qui ont eu le courage de relire ce rapport et qui m'ont ainsi permis de l'améliorer : Naoil, Hakim, Pascal, Nawal, Khalil et Yamina.

Table des matières

Chapitre 1

Introduction

De nombreux problèmes, de la vie de tous les jours, peuvent être modélisés sous la forme de problèmes d'optimisation combinatoire. La résolution de ces derniers peut intervenir dans divers contextes, et les méthodes de résolution déstinées à les résoudre doivent fournir des solutions de bonne qualité en un temps raisonnable. L'importance de la contrainte liée au temps est perceptible dans de nombreuses situations, comme par exemple dans le cas où les solutions doivent être utilisées dans une interaction avec les utilisateurs, ou bien, dans le cas où le nombre de problèmes à résoudre dans une application est très important.

Certains problèmes réels peuvent être modélisés sous la forme de programmes linéaires, de manière à ce que cette modélisation soit la plus proche possible d'un modèle théorique de la littérature pour lequel différentes approches de résolution efficaces ont été proposées. L'étude de ce genre de modèles théoriques est d'une grande importance pour les preneurs de décisions, dans la mesure où elle met à leur dispostion un ensemble de méthodes efficaces, capables de résoudre leurs problèmes particuliers. Définir une méthode de résolution efficace, capable d'obtenir une solution optimale, avec une compléxité spatiale et temporelle raisonnable, est un objectif difficile à atteindre, en particulier si l'instance traitée est de grande taille.

Les méthodes de résolution capables de générer une solution optimale sont appelées méthodes exactes. Elles consistent en général à énumérer l'ensemble des solutions de l'espace de recherche de manière implicite. Cependant, elles ne permettent de résoudre que des problèmes de petites tailles, puisque leur temps d'exécution augmente

exponentiellement avec la taille du problème. En général, les difficultés liées à la complexité de ce genre de méthodes rendent leur application difficile, voire impossible. Et ce, malgré le développement fulgurant des logiciels de résolution de problèmes d'optimisation. Face à un tel constat, une autre cathégorie d'approches de résolution est apparue, à savoir les méthodes de résolution approchées.
Généralement, les méthodes de résolution approchées permettent de trouver rapidement des solutions réalisables du problème traité, mais sans garantie sur la qualité de celle-ci. Parmi ces méthodes on peut citer les heuristiques, les méthodes gloutonnes et les métaheuristiques représentées essentiellement par les méthodes de voisinage telles que la recherche tabou, le recuit simulé et les algorithmes évolutifs (les algorithmes génétiques et les recherches dispersées). Il existe aussi d'autres méthodes de résolution appelées méthodes hybrides, ces méthodes combinent des méthodes approchés avec des méthodes exactes pour tirer profit des points forts de chaque méthode et améliorer le comportement global de l'algorithme de résolution des problèmes d'optimisation combinatoire.

L'objectif du travail présenté dans ce mémoire est de proposer plusieurs méthodes de résolution exactes ou approchées pour obtenir des solutions de bonne qualité en un temps d'exécution raisonnable pour une classe de problèmes de l'optimisation combinatoire, à savoir le problème du sac-à-dos à contraintes disjonctives. Le choix de ce problème s'explique par le fait que c'est un cas particulier de la programmation linéaire en nombres entiers. Il appartient à la famille des problèmes du sac-à-dos, et reste un problème particulier difficile à résoudre. Les problèmes de la famille du sac-à-dos ont plusieurs caractéristiques. Ils permettent de formuler de nombreux problèmes réels et sont souvent utilisés comme sous-problèmes d'autres problèmes plus complexes. Ils appartiennent à la classe des problèmes NP-difficiles, et le fait de leur appliquer des méthodes exactes ne permet pas, en général, d'obtenir de solution optimale pour des instances de grande taille.
Il existe de nombreuses instances du problème du sac-à-dos à contraintes disjonctives sur internet, permettant ainsi une comparaison des résultats obtenus avec ceux des autres approches de la littérature. **Le chapitre** 2 de cette thèse décrit la famille des problèmes du type sac-à-dos. Il nous permet de situer le contexte de notre travail, en

apportant des précisions sur des éléments nécessaires à la compréhension de la suite de cette thèse. **Le chapitre** 3 aborde le problème du sac-à-dos à contraintes disjonctives, ainsi qu'un ensemble d'approches de résolution efficaces proposées pour le résoudre. Nous mettons en évidence les algorithmes les plus connus de la littérature pour ce problème. Nous décrivons dans **le chapitre** 4 deux algorithmes augmentés pour résoudre de manière approchée le problème du sac-à-dos à contraintes disjonctives. Le premier algorithme est basé principalement sur le concept d'arrondi des variables fractionnaires et sur les contraintes d'encadrement de la somme des variables du problème couplé avec une méthode appelée Hill Climbing. Le deuxième résout une série de relaxations linéaires en utilisant une méthode d'arrondi et des contraintes de cardinalité, cet algorithme réduit la taille du problème en fixant une partie des variables fractionnaires. Dans **le chapitre** 5 nous décrivons un algorithme basé sur le concept de branchement local. Après une présentation de la méthode pour les programmes linéaires en nombres entiers mixtes, nous abordons différents composants de l'algorithme que nous mettons en oeuvre. Nous présentons ensuite une étude expérimentale réalisée sur un ensemble d'instances de problèmes du sac-à-dos à contraintes disjonctives. Cette étude nous permet de mettre en évidence l'apport de certains composants sur les performances de notre algorithme.

Nous présentons finalement dans **le chapitre** 6 différents algorithmes exacts basés sur les contraintes d'encadrement de la somme des variables du problème. Nous terminons ce mémoire en présentant nos conclusions générales et les perspectives de recherche.

Chapitre 2

Les problèmes du type sac-à-dos

Dans le présent chapitre, nous présentons le contexte dans lequel vont s'inscrire nos travaux de recherche. Ces travaux s'articulent essentiellement autour de la résolution de problèmes d'optimisation à variables binaires. Nous présenterons quelques problèmes issus de la famille des problèmes du sac-à-dos. Cette famille de problèmes a suscité et suscite encore un interêt majeur pour la communauté des chercheurs, dans le domaine de la recherche opérationnelle. Cet interêt est d'autant plus comprehensible au vu de la simplicité de la formulation de ces problèmes. Tout d'abord nous présenterons les problèmes du sac-à-dos unidimensionnel (noté KP) et multidimensionnel (noté MKP). Ensuite nous évoquerons le problème qui fait l'objet de cette thèse à savoir le sac-à-dos à contraintes disjonctives (noté DCKP). Les méthodes de résolution du DCKP les plus performantes en terme de taille d'instance et de temps d'exécution seront abordées lors du second chapitre. Enfin nous verrons d'autres variantes du sac-à-dos qui sont obtenues en modifiant sa fonction objectif ou en lui ajoutant une ou plusieurs contraintes.

2.1 Le problème du sac-à-dos

Le problème classique du sac-à-dos (knapsack, ou KP) est un problème d'optimisation combinatoire qui ne contient qu'une seule contrainte sur les objets de la solution. Toutefois, il constitue un challenge pour les chercheurs de par sa complexité, puisqu'il appartient à la classe des problèmes NP-difficiles. Dans la vie de tous les jours, le problème du sac-à-dos peut se présenter sous de nombreuses formes. À titre d'exemple, nous donnons le problème de l'alpiniste qui dispose d'un sac et d'une liste d'objets qu'il est suceptible d'emporter dans le sac. Chaque objet apporte un certain confort à l'alpiniste et prend une certaine place dans le sac. La capacité du sac est limitée. Le problème consiste alors pour l'alpiniste à maximiser son confort tout en ne dépassant pas la capacité du sac. On peut aussi citer le problème d'investissement, où l'on dispose d'un budget fixé, et de n projets. Chaque projet est identifié par un indice j, $j = 1, \ldots, n$, un profit, et un coût d'investissement. L'investissement optimal peut être déterminé par la résolution d'un problème du type sac-à-dos. D'une façon générale, le problème du sac-à-dos consiste à remplir un sac dont la capacité est fixée, avec un sous ensemble d'objets de poids et de profit connus, de manière à satisfaire les deux conditions suivantes :

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

En pratique, le problème du sac-à-dos possède de nombreuses applications. Et il apparait dans plusieurs situations comme sous-problème aidant à la résolution d'autres problèmes (cf., Gilmore et Gomory [27], et Hifi et Roucairol [47]). Ce qui fait de lui un modèle théorique particulièrement intéressant. Il a été largement étudié (cf., Balas et Zemel [1], Fayard et Plateau [22], Horowitz et Sahni [49], Kellerer *et al.* [55], Martello et Toth [62]), et Elkihel et Plateau [16]. Sous l'appellation sac-à-dos sont regroupées plusieures formulations. Bien qu'elles semblent très proches, ces formulations ne font pas appel aux mêmes techniques de résolution.

Dans la littérature, il existe plusieurs problèmes qui se réduisent au problème du sac-

à-dos. L'objet de notre étude sera la formulation du KnapSack à variables binaires (en $0-1$).
On considère un ensemble d'objets étiquetés de 1 à n. Chaque objet $j \in \{1, \ldots, n\}$ dispose d'un poids w_j de valeur entière et d'un profit p_j. On dispose d'un sac dont le contenu ne doit pas excéder une capacité c de valeur entière fixée. On désire le remplir de façon à maximiser la somme des objets placés, et ce en respectant la contrainte de capacité. Plus précisement, on s'intéresse à la résolution du programme linéaire à variables binaires suivant :

$$x \in \arg\,(\max \sum_{1\leq j\leq n} p_j x_j, \text{ sous la contrainte } \sum_{1\leq j\leq n} w_j x_j \leq c).$$

Le vecteur $x := (x_j, 0 \leq j \leq n) \in \{0,1\}^n$ est une solution du problème où $x_j = 1$ si l'objet j est mis dans le sac et $x_j = 0$ si cet objet n'est pas mis dans le sac. Le problème consiste donc à choisir un sous-ensemble d'objets parmi la liste d'objets initiale afin de maximiser la fonction objectif suivante :

$$Z(x) = \sum_{j=1}^{n} p_j x_j$$

Afin d'éviter les cas triviaux, nous considérons pour toute la suite de ce mémoire l'hypothèse suivante :

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

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

$$\sum_{j=1}^{n} p_j \bar{x}_j \leq \sum_{j=1}^{n} p_j {x^\star}_j$$

La formulation du problème du sac-à-dos en variables binaires est donnée par :

$$(KP)\begin{cases} \text{Maximiser} & Z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \in \{0,1\}, \qquad \text{pour } j = 1, \ldots, n. \end{cases}$$

2.2 Le problème du sac-à-dos multidimensionnel en variables 0-1

Le problème du sac-à-dos multidimensionnel en variables 0-1 (noté MKP) est une généralisation du problème KP. Il correspond au cas où le nombre de contraintes de capacité est strictement superieur à 1. Il existe différentes nominations du MKP. Parmi lesquelles on peut citer : le problème du sac-à-dos multi-contraint ou sac-à-dos multiple. Mais la plus répandue reste celle du sac-à-dos multidimensionnel. Le MKP a deux spécifictés par rapport aux autres problèmes de la famille du sac-à-dos. D'une part, la matrice associée au MKP est en général dense. D'autre part, obtenir une solution réalisable du MKP est une opération facile. En effet, en fixant toutes les variables du MKP à 0, on obtient une solution de valeur égale à 0. Plusieurs auteurs ont effectué des travaux sur le MKP. On peut en citer Dommayer et Voss [12], Glover et Kochenberger [30], Hanafi et Fréville [35], Chu et Beasley [11]. Le MKP est un problème NP-Difficile. En effet, si $m = 1$ le MKP se réduit au problème du sac-à-dos. On peut modéliser le MKP de la manière suivante :

$$(MKP)\begin{cases} \text{Maximiser} & Z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} & \sum_{j=1}^{n} w_{ij} x_j \leq c_i \qquad \text{pour } i = 1, \ldots, m. \\ & x_j \in \{0,1\}, \qquad \text{pour } j = 1, \ldots, n. \end{cases}$$

avec p_j, w_{ij}, c_i des entiers positifs, (pour $j = 1, \ldots, n$ et pour $i = 1, \ldots, m$) Les instances du MKP qui existent dans la littératue ne contiennent que très peu de contraintes. Cependant, leur résolution reste assez difficile pour les logiciels d'optimisation.

À titre d'exemple, des instances qui comportent 10 contraintes et 500 variables ne sont pas résolues de manière optimale en un temps raisonnable, par les logiciels d'optimisation.

2.3 Le problème du sac-à-dos à contraintes disjonctives

Le problème du sac-à-dos à des contraintes disjonctives (sac-à-dos disjonctif ou DCKP) est considéré comme une variante du problème du sac-à-dos. C'est un problème dans lequel un objet peut ne pas être compatible avec d'autres d'objets. Comme dans un sac-à-dos normal, on dispose d'un sac de capacité fixée, et d'un ensemble d'objets. Chaque objet est associé à un profit et à un poids. Chercher une solution du sac-à-dos unidimensionnel, c'est choisir un ensemble d'objets dont le poids n'éxcède pas la capacité du sac. Alors que pour trouver une solution du sac-à-dos à contraintes disjonctives, il faut choisir un ensemble d'objets qui verifie :

1. Le poids total de l'ensemble des objets choisi ne doit pas dépasser la capacité du sac.
2. Il n'existe pas de contraintes disjonctives entre les objets de l'ensemble choisi.

Le problème du sac-à-dos à contraintes disjonctives a été l'objet de très peu d'études. Yamada et *al.* [89, 90] ont été les premiers à introduire ce problème. Ils modélisaient un problème de sélection de sites pour installer des centrales nucléaires. Supposons que nous disposons d,Äôun certain capital d,Äôinvestissement, noté c, et de n sites possibles pour construire un ensemble de centrales nucléaires. Chaque site j nécessite un coût w_j et permettra de produire une quantité d,Äôénergie p_j. Deux sites choisis doivent obligatoirement être à une distance minimale notée d l,Äôun de l,Äôautre, pour des raisons de sécurité et d,Äôenvironnement. L,Äôobjectif du problème consiste alos à sélectionner un sous- ensemble de sites de manière à maximiser la quantité totale d,Äôénergie produite sans dépasser le capital total à la disposition, et de telle sorte que deux sites soient au moins distants de d. Cette contrainte oblige les objets sélectionnés à être compatibles deux à deux. Les auteurs ont proposé plusieurs algorithmes −exactes

et approchés−. Au départ, les auteurs ont proposé une résolution qui s'appuie sur l'application d'une heuristique basée sur une stratégie gloutonne. Ensuite, ils ont proposé deux versions d'un algorithme exact (l'une est une amélioration de l'autre). Ces algorithmes s'appliquent principalement à résoudre des instances non-corrélées. La taille de ces instances varie entre 100 et 1000 objets. Alors que leur densité varie entre 0.001 et 0.02. Hifi et Michrafy [43] ont récemment proposé une heuristique basée sur une recherche locale réactive pour la résolution d'intances du DCKP fortement correlées. cette heuristique se base principalement sur l'autorisation de mouvements dégradant, pour diversifier la recherche. Elle utilise une liste pour ne pas visiter plusieurs fois la même solution. Pour les instances de petite taille, les auteurs ont proposé un algorithme exact basé sur une stratégie de réduction d'intervalles et sur une recherche dichotomique [46].

2.4 Description du sac-à-dos à contraintes disjonctives

Le problème du sac-à-dos à contraintes disjonctives (DCKP) est un probème NP-difficile. En effet, si on supprime les contraintes disjonctives, le DCKP devient le problème du sac-à-dos unidimensionnel. Ce dernier est lui même NP-difficile (cf., Garey et Johnson [25]). Le DCKP contient deux types de contraintes :

1. La contrainte classique et unique de capacité.
2. Les contraintes dijonctives traduisant la non-compatibilité entre certains objets. Deux objets sont incompatibles s'ils ne doivent pas participer à la même solution.

Une instance du sac-à-dos à contraintes disjonctives est définie par un ensemble de n objets et par une capacité c du sac-à-dos. On associe à chaque objet $j,\ j = 1, \ldots, n$, un profit p_j et un poids w_j. Certains couples d'objets (i, j) sont incompatibles. Autrement dit, les deux objets du couple ne peuvent appartenir à la même solution. Soit $E \subset \{(i, j)/1 \leq i \neq j \leq n\}$ l'ensemble des couples disjonctifs. Et soit x_j la variable de décision associée à l'objet d'indice j. La variable x_j prend la valeur 1 si l'objet j est sélectionné dans la solution et 0 sinon. Le formalisme de ce type de contraintes

disjonctives est le suivant :

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

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

$$\sum_{1}^{n} w_i x_i \leq c, \quad i = 1, \ldots, n$$

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

$$DCKP \begin{cases} \text{Maximiser} & Z(x) = \sum_{i=1}^{n} p_i x_i \\ \text{s.c.} & \sum_{i=1}^{n} w_i x_i \leq c \\ & x_j + x_i \leq 1, \forall (j,i) \in E \\ & x_i \in \{0,1\}, \quad \text{pour } i = 1, \ldots, n. \end{cases}$$

Sans perdre en généralités, on peut supposer que :

1. Les objets sont triés dans l'ordre décroissant du rapport profit sur poids. Plus formellemnt :
$$p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n.$$
2. w_i, p_i $(i = 1, \ldots, n)$, et c sont des entiers.
3. $\sum_{j=1}^{n} w_j > c, \exists w_j \leq c$, de manière à éviter les solutions triviales.

2.5 Le problème du sac-à-dos généralisé à choix multiple

Le problème de sac-à-dos généralisé à choix multiple (noté MMKP) consiste à maximiser une fonction objectif linéaire sous deux types de contraintes linéaires. La

première contrainte représente la contrainte de capacité, alors que la deuxième est une contrainte de choix. Le MMKP appartient à la classe des problèmes NP-difficile. Il peut être considéré comme une généralisation de certains problèmes de type sac-à-dos connus dans la littérature :

(i) Le problème du MMKP se réduit au problème du sac-à-dos multidimensionnel MDKP, si les contraintes de capacité sont négligées (cf., Chu et Beasly[11], Gavish et Pirkul [26], Lorie et Savage [57], Shih[73], et Vasquez[84]).

(ii) La suppression des contraintes de choix ainsi que la réduction des contraintes de capacité à une seule contrainte permettent de réduire le MMKP au sac-à-dos unidimensionnel (cf., Horowitz et Sahni [49], et Kellerer *et al.* [55]).

(iii) Le fait de retenir uniquement les contraintes de capacité réduit le MMKP au problème du sac-à-dos à choix multiple MCKP (Nauss [72] et Pisinger [74]).

Le MMKP est un programme linéaire à variables bivalentes. Chaque instance du MMKP possède n classes et m sacs. Le sac d'indice $k,\ k = 1, \ldots, m$, est caractérisé par sa contrainte k de capacité C^k. Chaque classe $i,\ i = 1, \ldots, n$, est caractérisée par un ensemble d'objets. À chaque objet j appartenant à la clasee i est associé un profit v_{ij} et un vecteur poids $w_{ij} = (w_{ij}^k)$, où w_{ij}^k dénote le k-ème poids de l'objet j dans la classe i.

À tout objet d'indice j de la classe i on associe une variable de décision x_{ij}. Cette variable est définie par :

$$x_{ij} = \begin{cases} 1 & \text{si l'objet d'indice j de la classe i est sélectionné} \\ 0 & \text{sinon} \end{cases}$$

Un seul objet est sélectionné par classe :

$$\sum_{j=1}^{r_i} x_{ij} = 1,\ i = 1, \ldots, n,$$

où r_i est le nombre d'objets dans la classe i. Chaque sac est contraint par sa capacité donnée par :

$$\sum_{i=1}^{n} \sum_{j=1}^{r_i} w_{ij}^k x_{ij} \leq C^k,\ k = 1, \ldots, m.$$

Le profit cumulé est représenté par la quantité suivante :

$$Z(x) = \sum_{i=1}^{n} \sum_{j=1}^{r_i} v_{ij} x_{ij}.$$

Le MMKP peut donc être formulé de la manière suivante :

$$(MMKP) \begin{cases} \text{Maximiser} & Z(x) = \sum_{i=1}^{n} \sum_{j=1}^{r_i} v_{ij} x_{ij} & \\ \text{s.c} & \sum_{i=1}^{n} \sum_{j=1}^{r_i} w_{ij}^k x_{ij} \leq C^k, & k = 1, \ldots, m \\ & \sum_{j=1}^{r_i} x_{ij} = 1, & i = 1, \ldots, n \\ & x_{ij} \in \{0, 1\}, & i = 1, \ldots, n,\ j = 1, \ldots, r_i. \end{cases}$$

2.6 Autres problèmes du type sac-à-dos

2.6.1 Le problème de la somme d'un sous-ensemble

On parle de problème de la somme d'un sous-ensemble uniquement si les coefficients associés à une variable dans l'objectif du problème et dans la contrainte de capacité sont similaires, (SSS) subset sum problem en anglais. La formulation du problème KP permet d'obtenir aisément la formulation du problème de la somme d'un sous-ensemble.

En effet, le problème SSS n'est qu'un sous cas du problème KP. Il a été utilisé notamment par Pisinger [76] comme sous-problème pour résoudre le problème du sac à dos multiple 2.6.8. Les solutions du problème sont également utilisées dans le cas de problèmes de planification. Elles ont notamment servi pour obtenir des bornes inférieures de bonne qualité (Guéret et Prins [34]).

Pour résoudre le problème SSS, les chercheurs ont appliqué et adapté différents algorithmes proposés pour le KP. Leurs efforts ont permis d'arriver à des résultats satisfaisants, notamment l'algorithme hybride combinant à la fois la programmation dynamique et la méthode de séparation et évaluations de Martello et Toth [66].

La formulation du problème SSS en variables binaires est donnée par :

$$(KP)\begin{cases} \text{Maximiser} & Z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} & \sum_{j=1}^{n} p_j x_j \leq c \\ & x_j \in \{0,1\}, \qquad \text{pour } j = 1, \ldots, n. \end{cases}$$

2.6.2 Le problème du sac-à-dos bi-dimensionnel

Le problème dit du sac à dos bi-dimensionnel est une extension du problème KP. Il correspond au cas $m = 2$ et ne comporte que deux contraintes. Cependant, la difficulté rencontrée dans sa résolution, due à l'absence de schéma d'approximation complétement polynomial, en fait un problème considéré comme à part entiére Gens et Levner [33].

Le problème du sac à dos bi-dimensionnel a donné lieu à de nombreuses recherches. Une grande partie de ces recheches a porté sur le calcul de bornes supérieures et ce en raison de la performance des méthodes de relaxation. Les problèmes réduits se raménent ainsi à des instances de problèmes KP ou des relaxations de problèmes KP. Certains chercheurs (cf, Plateau et *al.* [81]) ont ainsi proposé un algorithme basé sur le calcul d'un sous gradient afin de trouver la valeur du dual lagrangien du problème du sac à dos bi-dimensionnel.

Différentes techniques existent pour la résolution exacte du problème du sac à dos bi-dimensionnel. Quelques unes de ces techniques proposées par Plateau et Fréville [20, 21] sont basées en partie sur la résolution d'un problème dual agrégé (surrogate). La méthode se compose de deux phases. Tout d'abord, une phase dans laquelle l'algorithme va s'atteler à réduire le problème. Pour ce faire, l'algorithme fixe le plus de variables possible, en se servant de bornes supérieures dérivées à partir de solutions du dual agrégé. Puis, dans la seconde phase, un algorithme de séparation et évaluations est alors exécuté avec les mêmes bornes supérieures générées. Un algorithme exact a été plus récemment trouvé par les chercheurs Martello et Toth [60]. Cet algorithme permet de résoudre de façon optimale une grande partie d'un ensemble d'instances de grande taille générées aléatoirement.

Le problème du sac-à-dos bi-dimensionnel a ainsi fait l'objet de nombreux travaux. Nous citerons notamment ceux entrepris par Hill et Reilly [50]. Ces derniers ont ainsi étudié l'influence de la corrélation entre les coefficients de la fonction objectif et les deux valeurs des capacités du problème. Ces travaux ont abouti à des résultats dépendant en partie des algorithmes utilisés, et mettant en avant le fait qu'il est difficile de classer telle instance particulière d'un problème comme étant facile ou bien difficile.

2.6.3 Le problème du sac-à-dos quadratique

Dans le problème du sac-à-dos quadratique noté QKP, le profit obtenu en choisissant un objet ne dépend pas seulement de cet objet, mais aussi des autres objets choisis. Le QKP a été introduit pour la première fois par Gallo, Hammer et Simeone [24]. Il a permis de modéliser certaines problèmes en théorie des graphes. La formulation du problème QKP en variables binaires est donnée par :

$$(QKP)\begin{cases} \text{Maximiser} & Z(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}x_i x_j \\ \text{s.c} & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \in \{0,1\}, \qquad \text{pour } j = 1,\ldots,n. \end{cases}$$

Caprara, Pisinger et Toth [7] ont proposé un algorithme capable de résoudre certaines grandes instances de la littérature.

2.6.4 Le problème du sac-à-dos avec contraintes de précédence (PCKP)

Le PCKP peut être considéré comme une généralisation du sac-à-dos. Cette généralisation comporte un ordre de précédence entre une partie des objets. Le problème est de déterminer les objets qui seront mis dans le sac, sachant qu'un objet ne peut y être mis avant ceux qui le précèdent. Une sous-classe du PCKP est le tree-Knapsack problem, où G est un arbre orienté à la racine. Hirbayashi et al [51] ont formulé un problème sur les graphes biparties comme étant un problème du sac-à-dos

avec contraintes de précédences. Le programme linéaire suivant décrit le sac-à-dos avec contraintes de précédence :

$$PCKP \begin{cases} \text{Maximiser} & Z(x) = \sum_{i=1}^{n} p_i x_i \\ \text{s.c.} & \sum_{i=1}^{n} w_i x_i \leq c \\ & x_j \geq x_i, \forall (j,i) \in E \\ & x_i \in \{0,1\}, \quad \text{pour } i = 1, \ldots, n. \end{cases}$$

Yamada et You [51] ont proposé un algorithme basé sur la relaxation Lagrangienne. Dans cet algorithme ; les auteurs réduisent la taille des instances du PCKP.

2.6.5 Le problème du sac-à-dos avec contraintes de demande

Les contraintes de demande sont les contraintes d'inégalité supériorité ou égalité. Le problème du sac-à-dos avec contraintes de demande consiste à introduire une ou plusieurs contraintes de demande en plus des contraintes d'inégalité normales de type inférieure ou égale. Cappanera est le premier à introduire ce problème [6]. Cependant la littérature sur ce problème reste assez pauvre, ce qui est sans doute dû à sa difficulté. Cette difficulté augmente avec l'augmentation des contraintes de demande. On peut citer les travaux de Cappanera et Trubian [6], où ils déffinissent une heuristique de type recherche locale en deux étapes. La première étape consiste à essayer de trouver un ensemble de solutions rélisables, alors que la deuxième étape consiste à appliquer une recherche locale à partir de chacune des solutions. Le problème du sac-à-dos avec contraintes de demande s'écrit de la manière suivante :

$$\begin{cases} \text{Maximiser} & Z(x) = \sum_{i=1}^{n} p_i x_i \\ \text{s.c.} & \sum_{i=1}^{n} w_{ij} x_j \leq c_i \quad i \in M^1 = \{1, \ldots, m_1\}. \\ & \sum_{i=1}^{n} w_{ij} x_j \geq c_i \quad i \in M^2 = \{1, \ldots, m_2\} \\ & x_i \in \{0,1\}, \quad \text{pour } i = 1, \ldots, n. \end{cases}$$

2.6.6 Le problème de la distribution équitable

Dans cette partie, nous présentons le problème de la distribution équitable appelé le *Knapsack Sharing Problem (KSP)*. Dans le domaine d'application, le problème de la distribution équitable dispose de certaines variantes d'applications dans le commerce, le transport, les communications, l'allocation des ressources, etc. (voir Brown [4] et Tang [80]). Ce problème doit son nom " Knapsack Sharing Problem " à Brown [4, 5], qui a été le premier à utiliser ce nom. Prenons l'exemple d'un groupe qui cherche à partager équitablement un certain budget c fixé sur les m différents secteurs d'activité du groupe pour la réalisation de certaines extensions dans chacun de ces secteurs d'activité. Donc, partager équitablement ne revient pas à diviser le budget c en parts égales i.e (c/m) entre les différents secteurs d'activité. Mais comme chacune des extensions d'un secteur i, $i = 1, ..., m$, dispose d'un coût de réalisation et d'un profit estimé, alors le problème de partage équitable revient à maximiser la plus petite valeur d'un profit global par secteur d'activité sans dépasser le budget prévu pour la réalisation de ces extensions. D'autres exemples sont présentés dans Brown [4] et Tang [80]. Pour ce problème la répartition équitable ne consiste pas à partager en parts égales mais plutôt à maximiser la plus petite part.

Plus précisement, nous considérons dans cette partie le problème KSP à variables binaires. Dans le KSP à variables binaires, à chaque objet j est associé un profit p_j et un poids w_j. On dispose d'un ensemble $\mathcal{N}$ de n objets où $\mathcal{N}$ est composé de m classes d'objets disjointes. Si J_i désigne la i-ème classe d'objets,$i = 1, ..., m$, alors $\forall p = 1, ..., m, p \neq q, J_p \cap J_q = \emptyset$ et $\cup_{i=1}^{m} J_i = \mathcal{N}$.

L'objet de ce problème est de déterminer le sous-ensemble d'objets de capacité c que nous allons retenir dans le sac. Ce sous-ensemble est choisi de manière à maximiser la valeur de la fonction objectif qui r éalise le minimum par rapport à l'ensemble de toutes les classes.

Soit x_j la variable de décision à valeur binaire, alors $x_j = 1$ si l'objet j est pris dans le sac, et $x_j = 0$ sinon. Le KSP peut alors être présent'e de la manière suivante :

$$(KSP)\begin{cases} Maximizer \quad min_{1\leq i\leq m} \quad Z(x)=\sum_{j\in J_i} p_j x_j \\ s.c \qquad \sum_{j\in\mathcal{N}} w_j x_j \ \leq c \\ \qquad\qquad x_j \in \{0,1\} \qquad pour \quad j=1,...,n. \end{cases}$$

Les classes sont indexées de J_1 à J_m et les éléments d'une classe $J_i, i = 1, ..., m$ sont indexés de 1 à $\mid J_i \mid$. Le KSP est considéré comme un problème NP-complet et on le retrouve dans la littérature sous la notation $KSP(B_n/m/1)$ Yamada et Futakawa [91], Hifi et Sadfi [41] et Hifi et *al.*[42], ce qui signifie que nous disposons de n objets de type binaire (B), répartis en m classes et avec une seule contrainte. Jusqu'à présent, peu d'algorithmes exacts ou approchés ont été proposés pour le KSP à variables binaires. En effet, Yamada et Futakawa [92] ont étendu l'approche développée pour le $KSP(C_n/m/1)$ au problème $KSP(B_n/m/1)$. Les résultats expérimentaux montrent que l'approche donne de bons résultats. Hifi et *al.* [42] ont proposé une méthode heuristique basée sur la recherche tabou et ils ont montré que l'approche donnait de bon résultats pour les instances de problèmes corrélés et les instances des problèmes non-corrélés. Pour le même problème, Yamada et al. [91] ont proposé deux algorithmes exacts. Le premier utilise la méthode de branch and bound et le deuxième utilise une adaptation de la méthode de recherche binaire. Les auteurs ont montré que leur deuxième algorithme est basé sur la méthode de branch and bound binaire. Hifi et Sadfi [41] ont proposé une approche basée sur la méthode de programmation dynamique, dans laquelle le problème de base est décomposé en une série de problèmes de knapsack unidimensionnels. Les auteurs ont montré que leur approches donnait de bons résultats, dans le sens où elle permet de résoudre certaines instances de problèmes de grande taille, avec un temps d'exécution très raisonnable. Par la suite, Hifi et *al.* [42] ont proposé une version accélérée de l'algorithme exact developpé dans [41]. Dans la continuté de ce travail, la thèse de Mhalla [68] propose une nouvelle méthode exacte hybride, combinant la programmation dynamique et le Branch and Bound. Notons aussi que Boyer et *al.* [3] ont proposé un algorithme basé sur la programmation dynamique creuse pour la résolution de certaines instances de la littérature dans le but d'améliorer la performance de la méthode basée sur la programmation dynamique classique.

2.6.7 Le problème du sac-à-dos multi-objectif

On parle d'optimisation multiobjectif dans la cas où le problème consiste à optimiser plusieurs fonctions objectif. La notion d'optimalité laisse place à la notion d'efficacité, puisque nous avons plusieurs fonctions objectif à optimiser. L'objectif est donc de trouver l'ensemble des solutions du problème non dominées. On dit qu'une solution x domine fortement uns solution y, si elle est meilleure sur l'ensembles des objectifs. Une solution x domine une solution y, si il existe au moins un critère sur lequel la valeur de x est strictement supérieure à la valeur de y. Ehrgott et Gandibleux [15] ont introduit de nombreux concepts qui concernent la notion d'optimalité. Pour l'illustration et pour plus de précision, nous nous limiterons au cas où le problème dispose uniquement de deux fonctions objectif (bi-objectif). Le programme linéaire suivant décrit le sac-à-dos bi-objectif :

$$\begin{cases} \text{Maximiser} & Z(x) = \sum_{i=1}^{n} p_{1i} x_i \\ \text{Maximiser} & V(x) = \sum_{i=1}^{n} p_{2i} x_i \\ \text{s.c.} & \sum_{i=1}^{n} w_i x_i \leq c \\ & x_i \in \{0,1\}, \quad \text{pour } i = 1, \ldots, n. \end{cases}$$

2.6.8 Le problème du sac-à-dos multiple

Le problème du sac-à-dos multiple est un génŕalisation du problème du sac-à-dos. C'est un problème qui contient plusieurs sacs. L'objectif est d'assigner chaque objet à au plus un sac, tout en respectant la contrainte de capacité de chaque sac et ce en maximisant le profit total. Martello et Toth [65] ont proposé un algorithme exact basé sur une méthode de séparation et évaluation. Les bornes inférieures sont calculées grâce à la résolution de m problèmes du type KnapSack. Les bornes supérieures ont été calculées à l'aide d'une relaxation surrogate. Pisinger [75] a aussi proposé un algorithme exact basé sur la résolution de plusieurs problèmes du type SSS 2.6.1. Le programme

linéaire suivant décrit le sac-à-dos multiple :

$$\begin{cases} \text{Maximiser} & Z(x) = \sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij} & \\ \text{s.c} & \sum_{j=1}^{n} w_j x_{ij} \leq c_i & \text{pour } i = 1, \ldots, n. \\ & \sum_{i=1}^{m} x_{ij} \leq 1 & \text{pour } j = 1, \ldots, n. \\ & x_{ij} \in \{0, 1\}, & \text{pour } j = 1, \ldots, n, \text{ pour } i = 1, \ldots, n. \end{cases}$$

Dans [55], les auteurs ont proposé une série de tests numériques qui mettent en valeur l'efficacité de ces deux algorithmes exacts.

2.7 Conclusion

Nous avons présenté dans ce chapitre les problèmes les plus étudiés et les plus connus de la famille du sac-à-dos. Ces problèmes qui généralisent le problème du sac-à-dos unidimensionnel sont obtenus soit en modifiant ou en ajoutant une ou plusieurs contraintes de capacité à ce dernier, soit en modifiant sa fonction objectif. Nous avons essayer de citer les principaux auteurs qui ont entrepris des travaux de recherches sur ces problèmes. Conscients que nous n'avons pas evoquer tous les problèmes de la famille du sac-à-dos, nous invitons le lecteur à consulter le livre de Kellerer, Pferschy et Pisinger [55].

Chapitre 3

Méthodes de résolution classiques du sac-à-dos à contraintes disjonctives (DCKP) en variables 0-1

Dans ce chapitre nous présenterons le problème du sac-à-dos unidimensionnel (noté KP), ainsi que le problème du sac-à-dos disjonctif (noté $DCKP$). Pour le KP nous citerons les méthodes de résolution les plus utilisées et les plus connues de la littérature. Ensuite, nous présenterons la modélisation du problème $DCKP$, et nous décrirons ses méthodes de résolution les plus performantes en termes de taille d'instance et de temps d'exécution.

3.1 Introduction

Le problème classique du sac-à-dos (knapsack, noté KP) est un problème d'optimisation combinatoire qui ne contient qu'une seule contrainte sur les objets de la solution. Toutefois, il constitue un challenge pour les chercheurs de par sa complexité puisqu'il

appartient à la classe des problèmes NP-difficiles. Dans la vie de tous les jours, le problème du sac-à-dos peut se présenter sous de nombreuses formes. À titre d'exemple nous donnons le problème de l'alpiniste qui dispose d'un sac et d'une liste d'objets qu'il est suceptible d'emporter dans le sac. Chaque objet apporte un certain confort à l'alpiniste et chaque objet prend une certaine place dans le sac. La capacité du sac est limitée, le problème consiste alors pour l'alpiniste à maximiser son confort tout en ne dépassant pas la capacité du sac. On peut aussi citer le problème d'investissement où l'on dispose d'un budget fixé, de n projets, où chaque projet est identifié par un indice j, $j = 1, \ldots, n$, un profit, et un coût d'investissement. L'investissement optimal peut être déterminé par la résolution d'un problème de type sac-à-dos. D'une façon générale, le problème du sac-à-dos consiste à remplir un sac dont la capacité est fixée, avec un sous ensemble des objets de poids et de profit connus, de manière à satisfaire les deux conditions suivantes :

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

En pratique, le problème du sac-à-dos possède de nombreuses applications, et il apparait dans plusieurs situations comme un sous-problème qui aide à la résolution d'autres problèmes (cf., Gilmore and Gomory [27], et Hifi et Roucairol [47]). Ce qui fait de lui un modèle théorique particulièrement intéressant, qui a été largement étudié (cf., Balas et Zemel [1], Fayard et Plateau [22], Horowitz et Sahni [49], Kellerer *et al.* [55], et Martello et Toth [62]). Sous l'appellation sac-à-dos sont regroupées plusieurs formulations qui, bien qu'elles semblent très proches, ne font pas appel aux mêmes techniques de résolution.

Dans la littérature, il existe plusieurs problèmes qui se réduisent au problème du sac-à-dos, l'objet de notre étude sera la formulation du KnapSack à variables binaires (en $0-1$).

On considère un ensemble d'objets étiquetés de 1 à n. Chaque objet $j \in \{1, \ldots, n\}$ dispose d'un poids w_j de valeur entière et d'un profit p_j. On dispose d'un sac dont le

contenu ne doit excéder une capacité c de valeur entière fixée. On désire le remplir de façon à maximiser la somme des objets placés, en respectant la contrainte de capacité. Plus précisement, on s'intéresse à la résolution du programme linéaire à variables binaires suivant :

$$x \in \arg\,(\max \sum_{1\leq j\leq n} p_j x_j, \text{ sous la contrainte } \sum_{1\leq j\leq n} w_j x_j \leq c).$$

Le vecteur $x := (x_j, 0 \leq j \leq n) \in \{0,1\}^n$ est une solution du problème où $x_j = 1$, si l'objet j est mis dans le sac et $x_j = 0$ sinon. Le problème consiste donc à choisir un sous-ensemble d'objets parmi la liste d'objets initiale afin de maximiser la fonction objectif suivante :

$$Z(x) = \sum_{j=1}^{n} p_j x_j$$

Afin d'éviter les cas triviaux, nous considérons pour toute la suite de ce mémoire, l'hypothèse suivante :

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

3.2 Certaines méthodes de résolution du sac-à-dos

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

$$\sum_{j=1}^{n} p_j \bar{x}_j \leq \sum_{j=1}^{n} p_j x^\star{}_j$$

La formulation du problème KP en variables binaires est donnée par :

$$(KP)\begin{cases} \text{Maximiser} & Z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c} & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_j \in \{0,1\}, \qquad \text{pour } j = 1, \ldots, n. \end{cases}$$

3.2.1 Certaines méthodes approchées

Pour les problèmes du type sac-à-dos, il est peu évident d'avoir un algorithme permettant de les résoudre à l'optimum en temps polynomial, il est donc intéressant d'avoir des algorithmes polynomiaux ; faciles à mettre en oeuvre ; permettant de calculer une solution approchée. Ce genre d'algorithme est souvent adapté à la structure du problème, il arrive en général à trouver une solution plus ou moins de bonne qualité en peu de temps. Parmi les méthodes heuristiques les plus utilisées pour la recherche d'une solution pour le sac-à-dos, on trouve la méthode qui s'appuie sur le principe *glouton*. Le principe de la méthode gloutonne, consiste à former pas à pas une solution réalisable du problème. À chaque étape de l'algorithme, la solution en cours, ou solution partielle est utilisée pour construire une meilleure solution. Ainsi l'algorithme s'arrête, après un certain nombre de phases, avec une solution réalisable pour le KP. Un algorithme glouton fait toujours le choix qui semble le meilleur sur le moment. Autrement dit, il fait un choix localement optimal, dans l'espoir que ce choix ménera à une solution optimale globale. Les algorithmes gloutons n'aboutissent pas toujours à des solutions optimales, mais la méthode gloutonne est très puissante et fonctionne correctement pour des problèmes variés.

Comme il existe plusieurs variantes pour ces algorithmes gloutons, nous présentons un des algorithmes glouton existants. Tout d'abord, nous effectuons un tri des objets dans l'ordre décroissant du rapport profit sur le poids, c'est-à-dire :

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

L'algorithme glouton que nous présentons, figure 3.1, consiste à sélectionner à

chaque étape un élément selon l'ordre précédemment défini. Si l'élément est admissible, c'est-à-dire si son poids ne dépasse pas la capacité disponible après fixation des autres éléments, alors il est mis dans le sac. Sinon, on sélectionne l'élément qui se situe juste après lui, et qui peut être admissible, et ainsi de suite, jusqu'à épuisement de tous les objets pouvant être mis dans le sac.

`Entrée :` Une instance d'un problème de sac à dos ;
`Sortie :` Une solution réalisable $\bar{x}$;
1. $\bar{c} := c$;
2. `Pour` $j := 1$ `jusqu'à` n `faire`
`Si` $w_j \leq \bar{c}$ `alors` $\bar{x}_j := 1$; $\bar{c} := \bar{c} - w_j$;
`sinon` $\bar{x}_j := 0$.
`FinSi`
`FinPour`
3. `Sortir avec` $\bar{x}$

FIG. 3.1 – Algorithme glouton pour le KP.

3.2.2 Quelques recherches de bornes

De façon générale, le calcul des bornes, supérieures et inférieures, permet d'encadrer la solution optimale pour les problèmes que l'on tente de résoudre. Aussi, elles sont utilisées dans le développement de méthodes de résolution exacte, qui utilisent, en général, des procédures d'énumération implicite (ou méthodes de séparation et évaluation).

Dantzig [13] a proposé une première borne supérieure facile à déterminer. L'idée consiste à relâcher chaque variable x_j, pour $j = 1, \ldots, n$, dans l'intervalle $[0,\ 1]$, puis de le résoudre (après la définition d'un certain ordre sur les objets) par application d'une procédure gloutonne. Le problème relâché (ou la relaxation linéaire du KP) peut être représenté par le programme linéaire suivant :

$$LP(KP)\begin{cases} \text{Maximiser} & Z(x)=\sum_{j=1}^{n} v_j x_j \\ \text{s.c} & \sum_{j=1}^{n} w_j x_j \leq c \\ & 0 \leq x_j \leq 1, \qquad j=1,\ldots,n. \end{cases}$$

La résolution du problème *LP(KP)* consiste à remplir le sac, objet après objet, jusqu'à sa saturation. Ensuite, on repère le premier objet, d'indice ℓ ($\ell \leq n$), ne pouvant être mis en totalité dans le sac. On appelle cet objet l'*élément critique* de l'instance du KP ayant comme variable de décision x_ℓ. Il s'agit d'un élément vérifiant

$$\sum_{j=1}^{\ell-1} w_j \leq c < \sum_{j=1}^{\ell} w_j.$$

Une solution du $LP(KP)$ peut être représentée par :

$$\bar{x}_j := \begin{cases} 1 & \text{si } j=1,\ldots,\ell-1, \\ \frac{(c-\sum_{j=1}^{\ell-1} w_j)}{w_\ell} & \text{si } j=\ell, \\ 0 & \text{si } j=\ell+1,\ldots,n. \end{cases}$$

La borne supérieure pour le problème KP (notée UB_d) est une solution optimale du $LP(KP)$. Une solution réalisable pour le KP l'est aussi pour le $LP(KP)$. Ceci est vrai puisque la contrainte d'intégrité sur les variables x_j, pour $j=1,\ldots,n$, de $LP(KP)$ est plus relâchée que celle du KP. On a alors :

$$Z_{LP(KP)}(x) = UB_d \geq Z_{KP}(x).$$

Sachant que toute solution du KP est entière, la partie entière inférieure de la solution optimale du $LP(KP)$ demeure une borne supérieure pour le KP. On en déduit donc la borne suivante :

$$UB_d = \sum_{j=1}^{\ell-1} v_j + \Big\lfloor \frac{(c-\sum_{j=1}^{\ell-1} w_j)}{w_\ell} v_\ell \Big\rfloor. \tag{3.1}$$

où $\lfloor x \rfloor$ désigne la partie entière de x.

Cette borne est communément appelée borne de Dantzig [13]. Notons aussi qu'une amélioration de cette borne a été proposée par Martello et Toth [64].

3.2.3 Certaines méthodes exactes

Une méthode de résolution exacte doit permettre l'obtention d'au moins une solution optimale. Elle doit aussi être capable de prouver que cette solution est optimale. Ce qui est parfois tout aussi difficile que l'obtention d'une solution optimale. Ce genre de méthode demande en genéral des temps d'exécution très élevés ainsi que des ressources mémoire importantes sur des instances de grande taille. Plusieurs méthodes de résolution exacte pour le KP ont été proposées, la plupart de ces méthodes sont basées sur des techniques énumératives. Il s'agit d'énumérer d'une manière implicite les solutions et de choisir la meilleure parmi toutes. Dans la littérature, on utilise souvent la méthode de séparation et évaluation (cf, Horowitz et Sahni [49]).

3.2.4 Méthodes de séparation et évaluation

L'algorithme de Branch and Bound figure parmi les approches les plus utilisées pour la résolution du problème KP. En effet, plusieurs versions de cet algorithme ont été développées et restent très utilisées dans l'optimisation combinatoire. Toutes ces versions partent du même principe qui est l'énumération implicite des valeurs des variables dans la solution et la sélection à chaque étape de la meilleure solution courante. Le concept général des algorithmes de Branch and Bound s'appuie essentiellement sur une structure arborescente de recherche de solutions. Chaque nœud de l'arborescence sépare l'espace des solutions en deux sous-espaces disjoints, jusqu'à ce que l'espace des solutions soit totalement exploré.

Dans le premier espace, la variable prend la valeur 1 et dans le second espace elle prend la valeur 0. L'élimination d'un sous-espace de recherche (ou de la branche) se fait à l'aide des bornes supérieures calculées à ce niveau de l'arborescence. Cette élimination nous permet de réduire l'espace de recherche, ce qui nous permet aussi d'obtenir la solution optimale en un temps d'exécution meilleur. Développer tout l'espace de recherche consiste, pour une instance de n éléments, à développer 2^n vecteurs binaires de taille n. Ce qui est excessivement lourd et irréalisable au regard du temps d'exécution nécessaire à l'exploration de toute l'arborescence (pour n grand).

Les méthodes de séparation et évaluation se basent sur les phases de la stratégie de séparation, le développement de l'arborescence et l'utilisation de la fonction

d'évaluation :

1. **Stratégie de séparation :** ce principe s'effectue au niveau de l'instance du KnapSack, il permet de décomposer le problème du KnapSack en deux sous ensembles de cardinalités plus petites. L'élément de séparation peut être pris selon une règle prédéfinie, ou selon un élément défini pendant l'exploration. Par exemple on peut choisir l'élément critique (l'élément non mis dans le sac et disposant du meilleur rapport profit sur poids) comme élément de séparation. Parfois, le choix de la règle de séparation est important pour améliorer les performances de l'algorithme

2. **Fonction d'évaluation :** Le choix de la fonction d'évaluation est très important pour limiter l'espace de recherche. En effet, certaines fonctions sont plus performantes que d'autres, mais la complexité de leurs évaluations rend l'algorithme plus lent, puisque le temps gagné au niveau de la troncature est moins important que celui perdu pour l'évaluation. L'idéal est de trouver le meilleur compromis entre la qualité et la difficulté de calcul d'une fonction d'évaluation.

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

Tous les algorithmes de séparation et évaluation s'appuient sur le même principe, la différence réside dans le choix de la stratégie de recherche. Dans ce qui suit, nous présentons l'algorithme de séparation et évaluation développé par Horowitz et Sahni [49]. Cette méthode considère les éléments ordonnés selon l'ordre décroissant du rapport profit sur poids. Ensuite, elle fait la séparation sur l'élément suivant. Depuis chaque noeud de l'arbre, et pour un élément j pris dans l'ordre prédéfini, on développe deux branches : la première branche ($x_j = 1$) correspond à l'élément j mis dans le sac et la deuxième branche ($x_j = 0$) correspond à l'élément j qui n'est pas mis dans le sac. Le développement de l'arborescence se fait en profondeur, en privilégiant les branches

pour lesquelles les éléments sont mis un à un dans le sac. La fonction d'évaluation utilisée est la borne de Dantzig UB_d présentée dans la section 3.2.2. Un élément n'est sélectionné que si son poids ne dépasse pas la capacité disponible du sac. Ainsi, à chaque développement d'une branche complète de l'arborescence, la solution obtenue reste réalisable. Le calcul des bornes de Dantzig se fait uniquement au niveau des nœuds développés par une branche correspondant à un élément j non sélectionné, c'est-à-dire correspondant à $x_j = 0$.
Pour les nœuds développés par une branche correspondant à $x_j = 1$, la valeur de la borne de Dantzig n'est autre que celle du noeud père. Celle-ci est comparée à la valeur de la meilleure solution obtenue jusque là. Si la valeur de la borne est inférieure à la valeur de cette meilleure solution, alors il est évident que l'on ne pourra jamais atteindre une meilleure solution à partir de ce noeuud. Donc, la troncature au niveau de cette branche courante est possible. En somme cet algorithme comporte deux phases principales qui se répètent tout au long de son exécution . La première phase est celle du développement d'une branche en profondeur et la deuxième phase est celle du retour arrière. Cette dernière phase intervient dans le cas où à partir de la première phase on a soit une branche totalement développée soit une branche troncaturée. Elle consiste donc à remonter la branche courante afin de retrouver le dernier élément fixé à 1 et le remettre à 0. L'algorithme continue à partir de cet élément en relançant la première phase. La figure 3.2 présente les principales étapes de cet algorithme.

3.3 Le problème du sac-à-dos à contraintes disjonctives (DCKP)

Le problème du sac-à-dos avec contraintes disjonctives (DCKP) est considéré comme une variante du problème du sac-à-dos. C'est un problème dans lequel un objet peut ne pas être compatible avec d'autres d'objets. Comme dans un sac-à-dos normal, on dispose d'un sac de capacité fixée, et d'un ensemble d'objets, où chaque objet est associé à un profit et à un poids. Chercher une solution du sac-à-dos unidimensionnel c'est choisir un ensemble d'objets dont le poids n'éxcède pas la capacité du sac. Alors que pour trouver une solution du sac-à-dos à contraintes disjonctives, il faut choisir un

```
Entrée : Une instance d'un problème de sac à dos ;
Sortie : Une solution optimale x* ;
1. j := 1 ; BEST :=0 ;
2. Calculer la borne supérieure UB_d accessible à partir de j
3. Si UB_d ≥ BEST alors
        Développer une branche de l'arbre en profondeur d'abord
        pour avoir une solution réalisable x' ;
        j := j + 1 ;
        Si Z(x') ≥ BEST alors
          x := x'
          BEST := Z(x') ;
        FinSi
   FinSi
4. j := max{ℓ : ℓ ≤ j et x'_ℓ = 1}
   Tant que j existe faire.
        x'_j := 0 ;
   Fin Tant Que ;
5. x* := x.
```

FIG. 3.2 – Algorithme de *branch and bound* pour le Knapsack.

ensemble d'objets qui verifie :

1. Le poids total de l'ensemble des objets choisis ne doit pas dépasser la capacité du sac.
2. Il n'existe pas de disjonction entre les objets de l'ensemble choisi.

Le problème du sac-à-dos à contraintes disjonctives a été l'objet de très peu d'études. Yamada et *al.* [89, 90] ont été les premiers à introduire ce problème, ils ont aussi proposé plusieurs algorithmes −exacts et approchés−. Au départ, les auteurs ont proposé une résolution qui s'appuie sur l'application d'une heuristique basée sur une stratégie gloutonne, à partir de là, ils ont proposé deux versions d'un algorithme exact, dont l'une est une amélioration de l'autre. Ces algorithmes exacts s'appliquent principalement à résoudre des instances non corrélées, pour lesquelles la taille varie entre 100 et 1000

objets et la densité varie entre 0.001 et 0.02. Hifi et Michrafy [89, 90] ont récemment proposé une heuristique basée sur une recherche locale réactive pour la résolution d'intances du DCKP fortement correlées, cette recherche locale se base principalement sur l'autorisation des mouvements dégradants pour diversifier la recherche ainsi qu'une liste pour ne pas visiter plusieurs fois la même solution. Pour des instances de petites tailles, ils ont proposé un algorithme exact basé sur une stratégie de réduction d'intervalles par dichotomie.

3.4 Description du problème du sac-à-dos à contraintes disjonctives

Le problème du sac-à-dos à contraintes disjonctives (DCKP) est un probème NP-difficile, car s'il n'avait pas de contrainte disjonctive, ce problème se réduirait au problème du sac-à-dos unidimensionnel, qui est lui même NP-difficile (cf., Garey et Johnson [25]). Le DCKP contient deux types de contraintes :

1. La contrainte classique et unique de capacité du sac-à-dos.
2. Les contraintes disjonctives, qui traduisent la non-compatibilité entre certains objets. Deux objets sont incompatibles s'ils ne doivent pas participer à la même solution.

Une instance du sac-à-dos à contraintes disjonctives est définie par un ensemble de n objets et par une capacité c du sac-à-dos. On associe à chaque objet $j,\ j = 1, \ldots, n$, un profit p_j et un poids w_j. Certains couples d'objets (i, j) sont incompatibles, autrement dit, les deux objets du couple ne peuvent appartenir à la même solution. Soit $E \subset \{(i, j)/1 \leq i \neq j \leq n\}$, cet ensemble de couples, et x_j la variable de décision associée à l'objet d'indice j tel que x_j prend la valeur 1 si l'objet j est sélectionné dans la solution, 0 sinon. Dans ce cas, le formalisme de ce type de contraintes disjonctives est le suivant :

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

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

$$\sum w_i x_i \leq c, \quad i = 1, \ldots, n$$

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

$$DCKP \begin{cases} \text{Maximiser} & Z(x) = \sum_{i=1}^{n} p_i x_i \\ \text{s.c.} & \sum_{i=1}^{n} w_i x_i \leq c \\ & x_j + x_i \leq 1, \forall (j,i) \in E \\ & x_i \in \{0,1\}, \quad \text{pour } i = 1, \ldots, n. \end{cases}$$

Sans perdre en généralités, on peut supposer que :

1. Les objets sont triés dans l'ordre décroissant du rapport profit sur poids, plus formellemnt :
$$p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n.$$
2. w_i, p_i $(i = 1, \ldots, n)$, et c sont des entiers.
3. $\sum_{j=1}^{n} w_j > c, \exists w_j \leq c$, de manière à éviter les solutions triviales.

3.5 Méthodes approchées pour le sac-à-dos à contraintes disjonctives

Yamada et *al.* [89] ont proposé deux méthodes approchées qui permettent de trouver des solutions réalisables de bonne qualité. La première méthode est une méthode gloutonne qui permet d'avoir une solution initiale. Alors que la deuxième méthode utilise une recherche locale pour l'amélioration de la solution fournie par la première méthode. Hifi et Michrafy [43], ont proposé une heuristique basée sur une recherche locale réactive.

3.5.1 Algorithme glouton pour le sac-à-dos à contraintes disjonctives

L'algorithme glouton suit le principe qui consiste à construire étape par étape une solution optimale locale en n étapes, dans l'espoir d'obtenir une solution optimale globale. L'étape i ($i \leq n$) n'est autre que la fixation de la valeur de la variable x_i, sachant que les variables $x_1, \ldots, x_{i-1}$ ont été fixées auparavant. Si le poids de l'objet dont l'indice est j ne dépasse pas la capacité résiduelle ($c - \sum_{k=1}^{i-1} w_k x_k$) et tous les objets du sac[1] sont compatibles avec l'objet en cours d'insertion, alors on met l'objet dans le sac (i.e. on pose $x_i = 1$). Dans le cas contraire, on pose $x_j = 0$. La figure 3.3 présente les principales étapes de cet algorithme :

Entrée : c, n, w_i, p_j pour $i = 1, \ldots, n$.
Sortie : x une solution et $z(x)$ est sa valeur

1. Poser $x_i = 0$ et $i = 1$
2. `Si` $\sum_{j=1}^{i-1} w_j x_j + w_i \leq c$ et $(\nexists j : j < i, x_j = 1)$, $(i, j) \in E$ `alors` $x_i = 1$
   `Sinon` $x_i = 0$
3. $i = i + 1$
4. `Répéter` les étapes de 2 et 3 tant que $i \leq n$.
5. `Sortir` avec $(x, z(x))$

FIG. 3.3 – Algorithme glouton pour le DCKP.

3.5.2 Amélioration de l'algorithme glouton

Les auteurs ont proposé une heuristique dans laquelle ils visent à améliorer la solution donnée par l'algorithme glouton ci-dessus. Cette heuristique se base sur un principe d'échange. En effet cet échange qui consiste à remplacer un objet par un autre n'est effectué que s'il améliore la solution courante. Soient x une solution réalisable du sac-à-dos à contraintes disjonctives, $I(x)$ un ensemble d'indices d'éléments fixés à 1, i.e, $I(x) := \{i \mid x_i = 1, 1 \leq i \leq n\}$, et $N(x)$ le voisinage de x représenté par l'ensemble

[1] les objets dont la valeur $x_k = 1$ avec $k \leq i - 1$.

des solutions obtenues par une permutation de certains élements de $I(x)$ par d'autres éléments de l'instance. Ce processus d'amélioration da la solution initiale est réalisé de la manière suivante :

Pour chaque élément i de $I(x)$, faire

(**i**) poser $x_i = 0$, remplacer cet objet par d'autres objets qui ne sont pas dans $I(x)$,

(**ii**) sélectionner la meilleure solution.

L'objectif de cet algorithme est d'améliorer la solution obtenue par l'algorithme glouton. L'amélioration de la solution se fait dans le voisinage $N(x)$, en appliquant les étapes (**i**) et (**ii**) de la boucle principale. Si une amélioration de la solution courante est enregistrée, alors il fait la mise à jour de la solution, puis il applique de nouveau le processus d'amélioration. Sinon, il s'arrête puis sort avec la meilleure solution trouvée. Cet algorithme est décrit dans la figure 3.4.

Entrée : $\bar{x}$ la solution de l'algorithme glouton **Sortie :** x est la solution de l'algorithme glouton améliorée, avec $z(x)$ sa valeur
étape 1. Poser $x = \bar{x}$ étape 2. `Si` $\exists e \in N(x)$ tel que $z(e) > z(x)$ `alors` $x := e$, `aller` à l'étape 2. étape 3. `sortir avec` $(x, z(x))$

FIG. 3.4 – Heuristique pour l'amélioration de la solution de l'algorithme glouton.

3.5.3 Relaxation Lagrangienne pour le sac-à-dos à contraintes disjonctives

Yamada *et al.* [90] ont proposé un algorithme basé sur la relaxation Lagrangienne pour calculer une borne supérieure du DCKP (figure 3.5). Cette relaxation a pour objectif de calculer la solution optimale du problème DCKP relaxé, à variables continues dans $[0, 1]$. L'idée est d'associer à chaque contrainte disjonctive $(1 - x_i - x_j \geq 0, (i, j) \in E)$ un multiplicateur de Lagrange, noté λ (que l'on notera λ_{ij}). Cette idée permet aux

auteurs de transformer le problème en un problème de maximisation avec une seule contrainte de capacité. Il s'agit d'un problème de type sac-à-dos à variables continues. Au départ, tous les multiplicateurs de Lagrange sont fixés à 0.

Ensuite, pour calculer la borne supérieure, l'algorithme procède en deux étapes :

1. mettre à jour des multiplicateurs de Lagrange.
2. résoudre le nouveau problème du sac-à-dos dont le profit est fonction de λ.

Soient $\lambda_{(i,j)}, (i,j) \in E$, un réel positif $L(x,\lambda)$ le problème dual défini par :

$$L : \begin{cases} \text{maximiser} & L := \displaystyle\sum_{j=1}^{n} p_j x_j + \sum_{(i,j)\in E} \lambda_{(i,j)}(1 - x_j - x_i) \\ \text{s.c.} & \displaystyle\sum_{j=1}^{n} w_j x_j \leq c \\ & 0 \leq x_j \leq 1, \quad j = 1, \ldots, n. \end{cases}$$

Le problème L représente le Lagrangien du sac-à-dos disjonctif, qu'on peut écrire d'une autre manière :

$$L := \sum_{j=1}^{n} (p_j - \sum_{e\in\delta_j} \lambda_e) x_j + \sum_{e\in E} \lambda_e,$$

où

$$\delta_j = \Big\{ e \in E \mid j \text{ est un antécédent de } e \Big\}.$$

De plus, si l'on pose $\lambda = (\lambda_e)$, le Lagrangien se réduira au sac-à-dos à variables continues, qui peut contenir des profits négatifs. La résolution du problème $L(\lambda)$ est réalisée par application de l'algorithme de Dantzig [13].

En effet, soit $x^\star(\lambda)$ la solution optimale de $L(\lambda)$ dont la valeur de la fonction objectif est $L^\star(\lambda) = L(x^\star(\lambda))$, et $\bar{Z}_\lambda = \lfloor L^\star(\lambda) \rfloor$, où $\lfloor x \rfloor$ désigne la partie entière de x. La quantité suivante définit une borne supérieure pour le sac-à-dos à contraintes disjonctives :

$$\min \Big\{ L^\star(\lambda) \mid \lambda \geq 0 \Big\}.$$

On remarquera que $L^\star(\lambda)$ est une fonction convexe en λ, linéaire par morceaux pour $(i,j) \in E$. Si $L^\star(\lambda)$ est différentiable en λ, alors

$$\partial L^\star(\lambda)/\partial\lambda_e = 1 - x_i^\star(\lambda) - x_j^\star(\lambda).$$

Finalement le calcul de λ s'effectue par application des étapes suivantes :

(i) Poser $\lambda^0 = 0$.

(ii) A l'étape $k, k \geq 1$, la mise à jour de λ s'effectue de la manière suivante :

$$\lambda^k = \lambda^{k-1} + \theta.\partial L^\star(\lambda^{k-1})/\partial\lambda.$$

Dans l'implémentation de cet algorithme, Yamada et *al.* [90] ont proposé de fixer le pas θ à 1.0.

Ils ont aussi proposé comme test d'arrêt :

$$L^\star(\lambda^k) \geq L^\star(\lambda^{k-1}) + 0.2.$$

cette quantité est notée *Gap*.

1. Initialisation
   $\lambda := (\lambda_e)_{e\in E}$ tel que $\lambda_e = 0, e \in E$
   $\bar{Z} := A$ /* A est un entier assez grand, Z est la solution optimale du L */
   $\bar{x} := 0$ /* $\bar{x}$ représente la borne supérieure de DCKP */
   $UB := 0$ /* UB est la borne supérieure en cours*/
   $Gap := A$ /* *Gap* désigne l'écart entre UB et $\bar{Z}$ */
2. Calculer la solution optimale $\bar{x}$ du problème $L(\lambda)$ en appliquant l'algorithme de Dantzig
3. Calculer la borne supérieure en cours UB donnée par :
   $UB := \sum_{j=1}^{n}(p_j - \sum_{e\in\delta_j}\lambda_e)x_j + \sum_{e\in E}\lambda_e$
4. Effectuer la mise à jour de $\bar{Z}$:
   $\bar{Z} := min(\bar{Z}, UB)$
5. Effectuer la mise à jour de λ :
   $\lambda := \lambda + \theta.\partial L^\star(\lambda^{k-1})/\partial\lambda$.
6. Effectuer la mise à jour de l'écart *Gap*
7. Répéter les étapes $2-6$ jusqu'à $Gap \geq 0.2$

FIG. 3.5 – Algorithme pour le calcul d'une borne supérieure pour le DCKP.

3.5.4 Algorithmes exacts pour le sac-à-dos à contraintes disjonctives

Dans le cadre de leurs travaux, Yamada *et al.* [90] ont proposé un algorithme exact pour la résolution du DCKP basé sur une méthode de *branch and bound*. Cet algorithme utilise une stratégie de parcours en profondeur. Comme borne inférieure (solution de départ) les auteurs ont utilisé la borne calculée par l'algorithme glouton de la section 3.5. Comme borne supérieure ils ont utilisé celle calculée par la relaxation Lagrangienne décrite dans la section 3.5.3. Ensuite ils ont proposé un algorithme exact qui utilise la réduction d'intervalle, dans cet algorithme ils introduisent une recherche dichotomique basée initialement sur la borne inférieure donnée par l'heuristique (voir section 3.5) et la borne supérieure donnée par la relaxation Lagrangienne (voir section 3.5.3). Cet algorithme est considéré comme étant une amélioration du premier, puisque la recherche d'une valeur cible est effectuée par le premier algorithme. Dans le cadre de leurs travaux, Hifi et Michrafy [46] ont proposé plusieurs versions d'un algorithme exact que nous détaillerons dans la section suivante.

3.5.4.1 Algorithme basé sur une procédure de réduction

Cet algorithme démarre avec une solution obtenue grâce à la recherche locale réactive, ensuite il utilise une procédure de réduction dans la perspective de fixer une partie des variables à l'optimum. Enfin, il utilise un algorithme de séparation et évaluation pour résoudre l'instance réduite. Les principales étapes de l'algorithme sont :

Soit $\bar{x}$ la solution de démarrage et $\bar{z}_{lb} := z(\bar{x})$ son évaluation.

Appliquer la procédure de réduction pour obtenir $I' \subseteq I$ l'ensemble des éléments fixés à l'optimum, et f^* la solution partielle associée à I' de valeur $z(f^*)$.

Appliquer au problème réduit $I \setminus I'$ un algorithme de séparation et évaluation. Soit ξ^* la solution optimale correspondant à $I \setminus I'$ de valeur $z(\xi^*)$.

Retourner $x^* = (f^* \mid \xi^*)$ de valeur objectif $z(x^*) = z(f^*) + z(\xi^*)$.

Pour le calcul de la borne supérieure, les auteurs ont résolu le problème suivant :

$$\text{RDCKP} \begin{cases} \text{maximiser} & h(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c.} & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_i + x_j \leq 1, \qquad \forall (i,j) \in E \\ & 0 \leq x_j \leq 1, \qquad j = 1, \ldots, n \end{cases}$$

La fixation des variables a principalement pour but d'accélérer la résolution des instances difficiles et de réduire la taille des instances traitées. La stratégie de réduction peut être utilisée soit à la racine de l'arborescence, soit à un noeud de l'arborescence, dans ce cas, il faut prendre le soin de considérer le retour en arrière, puisque la réduction n'est pas définitive pour l'instance initiale. Soit $x^{(k)} = (x_1, x_2, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n)$ la structure d'une solution du problème réduit $I' := I \setminus \{k\}$. Les auteurs ont proposé deux stratégies de réduction. La première stratégie partitionne l'ensemble I en deux sous-ensembles I_0 et I_1, telsque $I = I_0 \cup I_1$ et $I_0 \cap I_1 = \emptyset$, puis elle fixe les éléments de I_0 à 0 et les éléments de I_1 à 1. Le principe de cette stratégie est de trouver une borne supérieure du problème réduit notée $h(x^{(k)})$ et de la comparer avec une borne inférieure trouvée, notée $\bar{z}_{lb}$. Si un élément est fixé à 1 dans le processus de fixation, cet élément est supprimé du problème et tous ses voisins sont fixés à 0 ce qui implique la mise à jour de l'ensemble des contraintes disjonctives E. Si lors du processus de fixation, un élément est fixé à 0, alors cet élément ainsi que toutes les contraintes disjonctives dans lesquelles apparaît cet élément sont supprimées. On note que le processus de fixation est réalisable seulement si $\bar{z}_{lb} \geq h(x^{(k)})$. La deuxième stratégie de fixation qui est semblable à la première est plus coûteuse en terme de CPU dans la mesure où elle essaie de fixer chaque élément à 0 ou à 1.

3.5.4.2 Algorithme basé sur la réduction d'intervalles

Cet algorithme qui est basé principalement sur la réduction d'intervalles, peut être vu comme une alternative au premier algorithme exact, puisqu'il est déstiné à résoudre les instances pour lesquelles le premier algorithme a échoué dans l'obtention de l'optimum. Il est composé de deux phases. Dans la première phase l'algorithme applique

une procédure de réduction dans le but de fixer une partie des éléments à l'optimum. Ensuite, il construit l'intervalle de recherche en caractérisant ses deux bornes. Dans la deuxième phase, l'algorithme estime les bornes initiales utilisées dans la recherche dichotomique, puis applique une procédure de réduction afin de réduire la taille du sous-problème courant. Enfin, il applique un algorithme de branch and bound avec l'intervalle de réduction courant, et dès qu'il trouve une solution dans l'intervalle de recherche en cours, le processus de recherche s'arrête et met à jour l'intervalle de recherche.

1. Soit $\bar{x}$ la solution initiale et $z(\bar{x})$ son évaluation.
2. Appliquer la procédure de réduction et soit $I' \subseteq I$ l'ensemble des objets fixés à l'optimum, f^* désignant la solution partielle associée à I' de valeur $z(f^*)$.
3. Soit $[\bar{z}_{lb}, \bar{z}_{ub}]$ l'intervalle de recherche, où $\bar{z}_{ub}$ désigne la borne supérieure du problème traité, sa valeur initiale correspond à la valeur optimale de la relaxation continue du problème initial et $\bar{z}_{lb} := z(\bar{x})$.
   Poser $J := I \setminus I'$ l'ensemble des variables libres du sous-problème en cours et appliquer l'algorithme de branch and bound, en développant l'arborescence jusqu'à une certaine profondeur, afin d'affiner les deux bornes. Sortir si la solution obtenue est optimale.
4. *La recherche dichotomique :*

   Répéter

   4.1 Poser $\hat{z}_{lb} := \left\lfloor \frac{\bar{z}_{lb}+\bar{z}_{ub}}{2} \right\rfloor$, $\hat{z}_{ub} := \bar{z}_{ub}$ et $K := J$.

   4.2) Appliquer la procédure de réduction. Soit $K' \subseteq K$ l'ensemble des éléments fixés, g la solution partielle associée á K', de valeur $z(g)$.

   4.3 Appliquer l'algorithme de branch and bound sur $K \setminus K'$ et soit ξ *la première solution réalisable obtenue* sur $K \setminus K'$ de valeur d'objectif $z(\xi)$.
   Poser $z(x^*) = z(f^*) + z(g) + z(\xi)$ et considérer les deux sous cas suivants :

   4.3.1 si $\bar{z}(x^*) \geq \hat{z}_{lb}$, alors poser $\bar{z}_{lb} = \hat{z}_{lb}$, $I' = I' \cup K'$ et $J = J \setminus K'$.

   4.3.2 si la valeur cible est non atteignable, alors faire la mise á jour de l'intervalle de recherche : $\bar{z}_{ub} := \hat{z}_{lb} - 1$.

   tant que $\bar{z}_{lb} \geq \bar{z}_{ub}$.
5. Retourner la solution optimale x^* et son évaluation $z(x^*)$.

Les auteurs ont aussi proposé un autre algorithme exact, qui peut être vu comme étant une version revisitée de l'algorithme précédent, les problèmes réduits sont transformés selon un un modèle équivalent au modèle normal du DCKP, une contrainte de couverture est ajoutée pour faciliter la résolution des instances réduites par un algorithme de séparation et évaluation. L'algorithme commence par calculer une borne inférieure du problème initial grâce à la recherche locale réactive. Puis il applique une procédure de réduction afin de fixer une partie des objets du problème à l'optimum. Ensuite il calcule une borne supérieure associée à l'instance réduite pour construire l'intervalle de réduction sur lequel il démarre une recherche dichotomique.

3.6 Heuristique basée sur une recherche locale réactive

Toutes les méthodes approchées vues jusque là ont montré leur efficacité dans la résolution d'instances non-corrélées. En revanche, on s'aperçoit que pour des instances fortement corrélées, ces algorithmes montrent très vite leurs limites. Hifi et Michrafy ont proposé une heuristique basée sur une recherche locale réactive [43] capable de donner de bonnes solutions pour des instances du DCKP fortement corrolées. Les principales étapes de cette heuristique sont :

1. Appliquer un algorithme glouton pour avoir une solution de départ.
2. Utiliser la notion d'échange pour améliorer la solution en cours.
3. Explorer efficacement l'espace de recherche en utilisant une liste mémoire et une stratégie réactive qui autorise la dégradation de la solution.

Au cours de la première étape, l'algorithme utilise une procédure gloutonne pour démarrer avec une solution partielle. Cette procédure construit la solution pas à pas, en fixant à chaque étape un élément, mais ceci sans remettre en cause le choix de l'étape précédente. La procédure s'arrête et sort avec la solution obtenue lorsqu'elle n'arrive plus à fixer d'éléments. Dans la deuxième étape, l'algorithme applique une série de mouvements sur les éléments de la solution en cours, pour construire un

ensemble de solutions réalisables. Cet ensemble de solutions correspond au voisinage de la solution en cours, qui permettra d'engendrer une solution finale, et qui aura amélioré la qualité de la solution en cours. Au cours de la troisième étape, l'algorithme utilise une stratégie de dégradation de la solution. Il essaye d'intensifier la recherche pour éviter les optima locaux et ainsi visiter les zones non explorées, voire les régions prometteuses. Plusieurs paramètres sont introduits pour guider la recherche et identifier les régions non encore explorées. On peut citer, une liste de valeurs, une liste de configurations, un seuil de dégradation, un paramètre de fréquence, un paramètre de dégradation, etc.

3.6.1 Solution de démarrage

L'algorithme utilise une procédure gloutonne pour démarrer avec une solution partielle. Cette procédure construit la solution pas à pas, en fixant à chaque étape un élément, mais ceci sans remettre en cause le choix de l'étape précédente. La procédure s'arrête et sort avec la solution obtenue lorsqu'elle n'arrive plus à fixer d'éléments. La solution est construite selon un certain nombre de critères : le critère du coût moyen, le critère du coût marginal, etc. Généralement, le but principal d'une procédure du type glouton est d'obtenir une solution réalisable de bonne qualité en un temps faible.
Tout d'abord, nous présentons la structure correspondant à une solution du DCKP. Choisir une structure standard pour la représentation d'une solution ou d'une configuration du DCKP est simple, car les variables de décision sont des variables binaires. La figure 3.6 montre la structure d'une solution. Une solution réalisable, notée $\underline{x}$, vérifie obligatoirement (i) la contrainte de capacité, et (ii) toutes les contraintes disjonctives : $\sum_{j=1}^{n} w_j \underline{x}_j \leq c$ et $\underline{x}_i + \underline{x}_j \leq 1$, $\forall (i,j) \in E$. Ils existent des représentations binaires qui peuvent ne pas correspondre à des solutions réalisables. Les auteurs ont choisi d'utiliser la technique qui permet de transformer une configuration non réalisable en une solution réalisable.

colonne/objet $\longrightarrow$	1	2	3	4	5	6	7	...	$n-1$	n
vecteur $\longrightarrow$	1	0	0	0	1	0	0	...	0	0

FIG. 3.6 – Représentation binaire d'une solution du DCKP.

3.6.2 Définition du voisinage

L'algorithme applique une série de mouvements sur les éléments de la solution en cours, pour construire un ensemble de solutions réalisables, notées $x_1, \ldots, x_t, \ldots, x_n$. Cet ensemble de solutions correspond au voisinnage de la solution en cours, qui permettra d'engendrer une solution finale, et qui aura amélioré la qualité de la solution en cours. Soit le problème de maximisation suivant :

$$max\{f(x),\ x \in X\},$$

où X représente le domaine de définition de la variable x. Soit $h(x_t)$, l'opérateur qui transforme une solution en cours x_t en une solution voisine x_{t+1}. x_t est obtenue en itérant t fois l'opérateur $h(.)$. Ce processus de transformation est répété autant de fois qu'il permet l'amélioration de la solution en cours x_t. Le processus s'arrête et sort avec la meilleure solution trouvée, dès que l'opérateur $h(.)$ n'arrive plus à améliorer la solution.

Soit x_t une solution réalisable appartenant à X, obtenue à l'itération t. On suppose qu'il existe p voisinages associés à la solution x_t, notés $\mathcal{N}^1(x_t), \ldots, \mathcal{N}^p(x_t)$. Le problème $\mathcal{P}^k$ qui correspond au voisinage $\mathcal{N}^k(x_t)$, $k = 1, \ldots, p$, est défini par

$$(\mathcal{P}^k): \quad max\{f(x) \mid x \in \mathcal{N}^k(x_t)\}$$

L'opérateur $h(.)$ est défini par :

$$h(x_t) = x_{t+1.}$$

Le processus de calcul de x_{t+1} à partir de x_t comporte deux étapes :

1. pour $k = 1, \ldots, p$, x_t^k est la meilleure solution localisée dans le voisinage $\mathcal{N}^k$ correspondant au problème $\mathcal{P}^k$ dont l'évaluation est $f(x_t^k)$.
2. $x_{t+1} \in$ Argmax $\{f(y), y \in \{x_t^1, \ldots, x_t^k, \ldots, x_t^p\}\}$

3.6.3 Stratégies de guidage et d'exploration

Cette procédure dispose de deux objectifs principaux, à savoir éviter les optima locaux, et identifier les régions non explorées, voire les régions prometteuses. Il s'agit de diversifier, voire d'intensifier la recherche. De nombreuses stratégies peuvent être appliquées dans le but de mieux explorer l'espace de recherche. Ces stratégies s'inspirent pratiquement toutes de la même idée, à savoir autoriser la dégradation de la solution. La première stratégie pourrait être, commencer la recherche à partir d'une nouvelle configuration de départ. Ce qui permet d'éviter, les optima locaux, en explorant d'autres régions.

La deuxième stratégie consisterait à dégrader la configuration en cours, en supposant que la configuration finale sera réalisable. On autorise la construction d'une solution de moindre qualité, afin d'espérer repérer d'autres régions prometteuses. La troisième stratégie consisterait à appliquer une procédure dans le but d'améliorer la meilleure solution. Ce genre de procédure intensifie la recherche dans une zone donnée. Bien qu'elle amèliore la solution, cette stratégie consomme beaucoup de CPU. Il faut donc l'utiliser avec modération.
Plusieurs paramètres comme la liste de valeurs, la liste de configurations, le seuil de dégradation, le paramètre de fréquence, le paramètre de dégradation, etc, sont introduits pour permettre d'éviter d'explorer une même région plusieurs fois, et de retrouver les mêmes configurations. Ces paramètres servent aussi à guider la recherche et identifient les régions non encore et/ou peu explorées. Pour identifier une bonne stratégie, il faut d'abord identifier les différents paramètres nécessaires pour le modèle à traiter, ensuite régler ces paramètres.

Le test d'arrêt le plus simple est de fixer un nombre d'itérations à l'avance.

3.6.4 Une première recherche locale pour le DCKP

Pour générer une solution initiale, cette recherche locale, notée GP, fait appel plusieurs fois à une procédure gloutonne. Elle est constituée de trois phases principales :

1. adapatation de la procédure gloutonne appliquée au KnapSack unidimensionnel par Martello et Toth [63], un test est ajouté pour voir si les contraintes disjonctives

sont bien vérifiées.

2. chaque variable de décision x_i est fixée à 1, puis la méthode décrite dans la première phase est appliquée au problème du DCKP réduit.
3. La solution finale produite par GP est choisie, cette solution est la meilleure parmi toutes les solutions générées pendant les deux premières phases.

Entrée : Une instance du DCKP.

Sortie : une solution initiale $\underline{S} = (\underline{s}_1, \ldots, \underline{s}_n)$, de valeur $O(\underline{S})$.

1. Poser $\forall i \in I = \{1, \ldots, n\},\ \underline{s}_i = 0,\ O(\underline{s}) = 0$ and $\ell = 1$ and, $\overline{c} = c$.
2. Soit $S' = (s'_1, \ldots, s'_n)$ est une nouvelle solution :
   (a) Poser $s'_\ell = 1$;
   (b) Pose $s'_i = 0\ \forall i \in I \setminus \{\ell\}$ et $O(S) = p_\ell$;
   (c) $\overline{c} = c - w_\ell$.
3. `Pour` $i := 1$ $(i \neq \ell)$ à n `faire`
   `Si` $w_i \leq \overline{c}$ et $\nexists j \neq i,\ j \in I$ tel que $(i, j) \in E$ et $s'_j = 0$, `alors`
   set $s'_i \longleftarrow 1$ et $\overline{c}\ = \overline{c} - w_i$.
4. `Si` $O(\underline{S}) < O(S')$, `alors` pose $\underline{S} = S'$ et $O(\underline{S}) = O(S')$.
5. Incrémenter ℓ par unité et répéter les étapes 2-4 jusqu'à ce que $\ell > n$.
6. `Sortir` la meilleure solution $\underline{S}$ de valeur d'objectif $O(\underline{S})$.

FIG. 3.7 – la procédure GP pour développer une solution initiale pour le DCKP.

3.6.5 Une deuxième recherche locale pour le DCKP

Cette procédure, notée CGP, prend en entrée une solution développée par GP, puis applique un processus itératif, dans le but d'améliorer cette solution. La procédure CGP consiste d'abord à améliorer la solution en cours, ensuite elle doit essayer de repérer les solutions stagnantes. Une solution est dite stagnante si la procédure CGP est incapable de l'améliorer.

La procédure CGP applique un mécanisme d'échange qui consiste à faire sortir de la solution en cours un objet d'indice i, en posant $x_i = 0$, ensuite essayer d'introduire un autre objet, ou une liste d'autres objets notée L $(x_i = 1, i \in L)$. Cet échange est

accepté uniquement si la nouvelle configuration obtenue est réalisable pour le problème DCKP. Les auteurs ont introduit deux listes mémoire dans la perspective d'améliorer le comportement de CGP. Ces listes mémoire permettent, entre autre, de limiter le phénomène de cyclage. La première liste mémoire consiste à utiliser une liste tabou qui contient un ensemble de couples d'objets représentant les mouvements non autorisés (ou tabous). Son contenu est conservé pour un nombre d'itérations fixé. Il est renouvelé de temps en temps. La seconde liste est composée d'un ensemble de valeurs, où chaque valeur peut être associée à un ensemble de solutions réalisables dites stagnantes. La construction de cette liste fait appel à des techniques de hâchage.

Soit S une solution obtenue par la procédure GP, dont la valeur est $O(S)$. Soient S_1 l'ensemble des éléments fixés à 1 dans la solution S, et S_0 l'ensemble des éléments fixés à 0 dans la solution S. Formellement, S_0, et S_1 sont définis de la manière suivante :

$$S_1 = \{j \in I, \quad \text{si } x_j = 1\} \text{ et } S_0 = \{j \in I, \quad \text{si } x_j = 0\}.$$

On pose : $n_0 = |S_0|$ et $n_1 = |S_1|$, $n_0 + n_1 = n$.
n_0 et n_1 sont les cardinalitées respectives de S_0 et S_1.

La procédure CGP tente d'améliorer la solution en cours, pour chaque élément i de S_1 :

(i) en supprimant l'indice i de la solution (i.e. de S_1) ;

(ii) en appliquant la procédure GP sur l'ensemble S_0, afin de construire une solution voisine de S.

CGP choisit la meilleure solution obtenue par (i) et (ii), puis applique le même processus plusieurs fois sur cette solution.

3.6.6 Procédure de dégradation

Il s'agit de diversifier la recherche en autorisant le changement de direction, dans l'optique de favoriser les régions non explorées et d'assurer une bonne exploration de différentes régions prometteuses de l'espace de recherche. En ce qui concerne la dégradation, la solution d'entrée est relâchée dans le but d'obtenir une solution de moins bonne qualité. Ensuite, pour intensifier la recherche, le voisinage de la solution

Entrée : Solution produite par GP, notée $\underline{S}$, de valeur d'objectif $O(\underline{S})$.

Sortie : Solution améliorée S, de valeur d'objectif $O(S)$.

1. $I = \{1, \ldots, n\}$ désigne l'ensemble d'objets.
   Pose $n_1 = |(\underline{S})_1|$ /* voir 2.2.1.b) */
   Pose S'', copie de $\underline{S}$ et $i = 1$.
2. Soit $S''_i = S'' \setminus \{s''_i\}$, tel que s''_i désigne le i-ème élément de S''.
3. Soit $S''_i =$ `Complete_Sol`$(I \setminus \{s''_i\}, E, p, w, c, n-1, S''_i)$.
4. Soit $\underline{S} = \text{argmax}\{O(\underline{S}), O(S''_i)\}$ et incrémente i.
5. Répéter les étapes 2-4 tant que $i > n_1$;

`Complete_Sol`(I, E, p, w, c, n, S) procedure

(a) Posons $I' = \{j \in I \setminus S$ tel que $\nexists e \in S : (e, j) \in E\}$
Posons $E' = \{(i, j) \in I' \times I'$ tel que $(i, j) \in E\}$
Posons $r = \sum_{i \in S} w_i$, $O(S) = \sum_{i \in S} p_i$ et $n' = n - |I'|$
α est un entier naturel, engendré aléatoirement dans l'intervalle discret $[1, n']$;
Posons $H = \{\alpha, \alpha+1, \ldots, n', 1, \ldots, \alpha-1\}$ et soit e_j désigne le j-ème élément dans H.

(b) On pose $j = 1$.

(c) `Si` $r + w_{e_j} < c$ et $\nexists e \in S : (e, e_j) \in E'$, `alors`

(c.1) Mettre $S = S \cup \{e_j\}$;

(c.2) Mettre à jour les ressources : $r = r + w_{e_j}$;

(c.3) Mettre à jour la valeur économique : $O(S)$ avec $O(S) + p_{e_j}$.

(d) Incrémenter j d'une unité et répéter l'étape (c), jusqu' à ce que $j > n'$.

(e) Renvoyer S.

FIG. 3.8 – Algorithme de la procédure GCP.

dégradée est exploré grâce à des techniques d'échange. L'objectif est de construire une nouvelle solution à partir de la solution d'entrée. Pour ce faire, la procédure cherche d'abord le premier élément de la solution d'entrée S, dont la valeur est 0, noté i_0. Puis elle construit une nouvelle solution S', en fixant i_0 à 1. Ensuite la procédure cherche à compléter la solution S', en y ajoutant les éléments un par un suivant l'ordre croissant.

Une fois tous les éléments traités, la procédure s'arrête et retourne la nouvelle solution S'.

1. Soit S la solution en cours et $\bar{c} = c$ la capacité totale.
2. Posons $i_0 = \underset{i \in I}{\operatorname{argmin}} \left\{ s_i = 0 \right\}$ et $\bar{S}$ la solution en cours telle que $s_j = 0$, $\forall j \in I$.
3. Poser : $s_{i_0} = 1$,
   .$\bar{c} = \bar{c} - w_{i_0}$ {*mettre à jour la capacité restante*},
   .$i = 1$ {*Introduction du premier élément* }.
4. Si $(w_i \leq \bar{c})$ et $(\nexists j \in I \ : \ (i,j) \in E)$ alors
   . Poser $\bar{s}_i = 1$
   . Faire la mise à jour de la capacité : $\bar{c} = \bar{c} - w_i$;
   Sinon $\bar{s}_i = 0$;
5. Incrémenter i d'une unité, et répéter l'étape 3.6.6, jusqu'à $i > n$.

FIG. 3.9 – la procédure `Degrade`.

3.6.7 La recherche locale réactive

L'algorithme de la recherche locale réactive (RLS), est composé de deux phases. Dans la première phase, l'algorithme fait appel à la procédure GP pour construire une solution initiale notée S puis la procédure CGP est appelée pour améliorer cette solution, on notera $Best$ la solution obtenue. Ensuite, une liste mémoire notée $Tabu\_list$ est initialisée, ainsi qu'un indicateur noté r, cet indicateur sert à contrôler le processus de recherche, en comptant le nombre de fois que l'algorithme a échoué dans l'amélioration de la solution locale. Finalement, l'algorithme introduit deux paramètres notés *Depth* et *Limit*, qui servent pour le premier à compter le nombre maximum de fois que l'algorithme autorise la dégradation d'une solution, et pour le deuxième à compter le nombre de fois que l'algorithme n'a pas amélioré une solution après un processus de dégradation. Dans la deuxième phase, l'algorithme tente d'améliorer la solution S construite précédemment en appliquant une procédure notée CGP'. Cette procédure CGP' est une version modifiée de la procédure CGP. Elle ne nécessite pas forcément une solution initiale produite par GP, soit S' cette nouvelle solution. Si S' améliore la meilleure

solution $Best$, l'algorithme effectue une mise à jour de la meilleure solution et initialise l'indicateur r à 0. Si S' n'améliore pas la solution, l'indicateur r est incrémenté de 1. Enfin, l'algorithme applique la stratégie de dégradation et de diversification, qui se décline en deux cas : si l'indicateur r atteint la valeur $Limit$ alors, l'algorithme s'arrête, et sort avec la meilleure solution $Best$. Si l'indicateur r n'atteint pas la valeur $Limit$, l'algorithme applique la stratégie de dégradation.

3.7 Conclusion

Ce chapitre nous a permis de passer en revue un ensemble de méthodes de résolution existantes pour les problèmes du sac-à-dos , et sac-à-dos à contraintes disjonctives. Il ressort de cet état de l'art que les méthodes exactes semblent réellement limitées pour résoudre le problème du sac-à-dos disjonctif. Plusieurs méthodes permettent de déterminer des bornes supérieures efficaces à l'aide de relaxations lagrangiennes par exemple, mais cela ne permet toutefois pas de définir un processus de résolution exacte aussi performant que certaines heuristiques. Devant ce constat, nous nous sommes focalisés sur des méthodes approchées permettant d'obtenir des solutions réalisables le plus proche possible de l'optimum.

Entrée : Une instance DCKP.
Sortie : Une solution $Best$, de valeur objective $O(Best)$.

1. Initialisation :
   (a) Appel à $GP(p, w, E, n, c)$, afin de construire une solution initiale S.
   (b) Améliorer S en appliquant $CGP(S, Best)$, tel que $Best$ désigne la solution améliorée.
   (c) Soit $Tabu\_List = \emptyset$ et soit r un indicateur d'amélioration
   Au démarrage , mettre $r = 0$ et $S = Best$.
   (d) Soit *depth* le nombre de fois pour lequel la solution a été dégradée ;
   Soit *limit* le nombre de fois maximal pour lequel une solution peut être dégradée successivement.
2. `Répéter` les étapes suivantes :
   - `// Amélioration de la solution en cours :`
   (a) Appel à $CGP'(S, S', Tabu\_List)$;
   (b) Si $(O(S') > O(S))$, alors faire
      (b.1) $Best = S'$;
      (b.2) $S = Best$;
      (b.3) $r = 0$ // *l'indicateur est initialisé à zéro lors de l'enregistrement d'une amélioration*
   (c) Sinon mettre à jour r : $r = r + 1$;
   - `// Utilisation de la liste mémoire :`
   (d) Insérer la solution S' dans $Tabu\_List$.
   - `// Construction d'une nouvelle solution :`
   (e) Dégrader une meilleure solution $Best$, selon la valeur de r :
      (e.1) Si$(r < limit)$ alors Degrade$(Best, Tabu_List, depth)$
      (b.2) Sinon *Sortir* avec la meilleure solution $Best$, de valeur d'objectif $O(Best)$.

`Répéter` les étapes (a, b, c, d, e) $MaxIter$ fois.

FIG. 3.10 – La recherche locale réactive pour DCKP : RLS.

Chapitre 4

Algorithmes augmentés pour le sac-à-dos à contraintes disjonctives

Dans ce chapitre, nous proposons deux algorithmes augmentés, pour la résolution du problème du sac-à-dos à contraintes disjonctives (DCKP). Le DCKP est une variante du problème du sac-à-dos unidimensionnel. Il consiste à maximiser une fonction objectif sous deux types de contraintes. La première contrainte est liée à la capacité du sac, alors que la deuxième introduit une incompatibilté entre une partie des objets du problème. Deux objets sont incompatibles s'ils ne peuvent pas partciper simultanément à la même solution. Dans un premier temps, nous proposons un algorithme basé principalement sur le concept d'arrondi des variables fractionnaires et sur les contraintes d'encadrement de la somme des variables du problème. Ensuite, nous proposons un algorithme qui résout une série de relaxations linéaires, en utilisant une méthode d'arrondi et des contraintes de cardinalité. Cet algorithme réduit la taille du problème en fixant une partie des variables fractionnaires. Les performances de ces algorithmes sont évaluées sur une série d'instances corrélées, et comparées avec les meilleurs résultats de la littérature.

Ce chapitre a fait l'objet d'une communication au Congrès National de la ROADEF (cf., Hifi et Ould Ahmed Mounir [37]) et d'une soumission à la revue internationale Computatinal Optimimization and Applications (cf., Hifi, M'Hallah and Ould Ahmed Mounir [39]).

4.1 Introduction

Le problème du sac-à-dos disjonctif est une variante du problème du sac-à-dos. Il maximise une fonction objectif sous deux types de contraintes : la contrainte de capacité du sac et les contraintes disjonctives. Soient un sac de capacité c et un ensemble de n objets où chaque objet $j \in I = \{1, 2, \ldots, n\}$ est caractérisé par sa taille ou son poids w_j et son profit p_j. Certains objets de I sont incompatibles, dans ce cas la contrainte qui représente l'incompatibilité entre ces objets est ajoutée à la contrainte de capacité. Soit $E \subseteq \Big\{(i,j) \in I \times I \text{ tel que } i < j\Big\}$ l'ensemble de couples des objets incompatibles.

Deux objets i et j ne peuvent être mis simultanément dans le sac, si $(i,j) \in E$. Formellement, $DCKP$ s'écrit de la façon suivante :

$$\text{maximize} \qquad z(x) = \sum_{j=1}^{n} p_j x_j \tag{4.1}$$

$$\text{subject to} \qquad \sum_{j=1}^{n} w_j x_j \leq c \tag{4.2}$$

$$x_i + x_j \leq 1 \qquad \forall (i,j) \in E \tag{4.3}$$

$$x_j \in \{0,1\} \qquad j = 1, \ldots, n, \tag{4.4}$$

où la variable binaire $x_j,\ j \in I$ prend la valeur 1 si l'objet j est inclus dans le sac et 0 sinon. $z(x)$ définit la valeur de la fonction objectif associée à la solution optimale x du DCKP. Les inégalités (4.1), (4.2), et (4.4) décrivent le problème du sac-à-dos classique, alors que l'inégalité (4.3) correspond à l'ensemble des contraintes disjonctives. Sans perte de généralité, on suppose que

1. tous les paramètres $c,\ p_j,\ w_j,\ j \in I$, sont des entiers strictement positifs ;
2. les objets du problème sont triés selon l'ordre décroissant du rapport profit sur poids.

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

3. pour éviter les solution triviales nous supposons que $\sum_{j=1}^{n} w_j > c$.

Le $DCKP$, est un programme linéaire en nombres entiers à variables binaires. Il fait partie des problèmes d'optimisation combinatoire qui appartiennent à la classe des problèmes NP-difficiles. C'est une extension du problème de sac-à-dos à choix multiples 2.5. Il apparait souvent comme sous-problème dans d'autres problèmes d'optimisation combinatoire plus

complexes. Sa structure induite dans des problèmes plus complexes permet le calcul de bornes supérieures et la conception de méthodes heuristiques et exactes pour ces problèmes. Par exemple, le $DCKP$ a été employé dans la formulation de la décomposition de Dantzig-Wolfe pour le problème du bin packing à deux dimensions [77]. En effet, il a servi de sous-problème au problème de pricing, qui consiste à trouver un rangement réalisable dans une boite simple, vérifiant le plus petit coût réduit.

À notre connaissance, très peu de travaux de recherche ont porté sur le DCKP. Yamada et al [89, 90] ont abordé le problème en proposant deux algorithmes, un approché et l'autre exact. L' algorithme approché commence par générer une première solution réalisable de départ. Ensuite, il améliore cette solution à l'aide d'un mécanisme de $2-opt$ dans son voisinage. L'algorithme exact commence par générer une solution de départ à l'aide de l'heuristique décrite précédemment, ensuite, il entreprend une énumération implicite couplée avec une technique de réduction d'intervalle. L'algorithme exact résout des instances non-corrélées dont le nombre d'objets varie entre 100 et 1000, avec une densité variant de 0.001 à 0.020 (ce qui équivaut à un maximum de 10000 contraintes). Hifi et Michrafy [43] ont proposé une méthode basée sur une recherche locale réactive constituée de trois étapes. Cette méthode résout des instances de grande taille. Nous avons largement évoqué cette méthode au chapitre précédent. Finalement, Hifi et Michrafy [46] ont proposé trois algorithmes exacts dans lesquels des stratégies de réduction, un modèle équivalent et une recherche dichotomique coopèrent pour résoudre les instances du DCKP. La première version de l'algorithme réduit la taille du problème original en commençant par une borne inférieure et en résolvant successivement une série de relaxations du DCKP. La seconde combine une stratégie de réduction avec une recherche dichotomique afin d'accélérer le processus de recherche. Elle applique la stratégie de réduction au premier niveau de l'arbre développé et/ou après avoir généré quelques noeuds. La stratégie fixe quelques variables de décision à l'optimum, permettant ainsi la réduction de la taille du problème original ; et ainsi, facilitant dans beaucoup de cas la résolution de l'instance du problème. Deux stratégies sont considérées. La première fixe la variable de décision courante à sa valeur optimale ; c'est-à-dire à 0 ou à 1. Elle résout alors les sous-problèmes résultants. La seconde fixe la variable de décision courante à 0 et à 1, et résout ensuite les deux sous-problèmes résultants. La troisième et dernière version de l'algorithme propose une version modifiée de l'algorithme dans laquelle deux modèles équivalents coopèrent pour résoudre des instances du problème avec un nombre important de contraintes disjonctives. Dans ce chapitre, nous proposons deux algorithmes approchés qui appliquent un procédé d' arrondi aux solutions optimales d'une série de relaxations de programmes linéaires augmentées par

des contraintes d'encadrement ; c-à-d., des inégalités qui délimitent le nombre d'objets apparaissant dans la solution optimale du $DCKP$. Les algorithmes diversifient la recherche en variant les bornes des inégalités de cardinalité. Ils résolvent la relaxation linéaire du $DCKP$ augmenté en utilisant l'algorithme du simplexe. Ensuite, ils fixent quelques variables à leurs valeurs binaires, et arrondissent une partie des variables fractionnaires à 0 ou à 1. Ils répètent alors les étapes ci-dessus sur les problèmes réduits jusqu'à ce qu'une solution réalisable soit obtenue. Ils intensifient alors la recherche autour de la solution réalisable en utilisant une procédure appelée Hill Climbing. Ce chapitre est organisé comme suit. La section 4.2 aborde les deux nouveaux algorithmes approchés proposés pour la résolution du DCKP. Ces algorithmes sont principalement basés sur une stratégie d'arrondi des variables fractionnaires et les contraintes d'encadrement de la somme des objets de la solution optimale du $DCKP$. La section 4.3 évalue les performances des algorithmes proposés sur plusieurs instances de la littérature, et analyse les résultats obtenus. En conclusion, la section 4.4 récapitule ce chapitre.

4.2 Algorithmes augmentés

La section 4.2.1 introduit un modèle équivalent pour le DCKP. La section 4.2.4 décrit la procédure d'arrondi. La section 4.2.5 et la section 4.2.6 présentent le premier et le deuxième algorithme augmenté. Finalement, la section 4.2.7 détaille l'approche utilisée en conjonction avec la procédure d'arrondi pour améliorer les performances des deux algorithmes.

4.2.1 Modèle équivalent pour le DCKP

Hifi et Michrafy [46] tirent profit des contraintes disjonctives du DCKP en transformant le modèle initial en un modèle équivalent $EDCKP$. Dans ce nouveau modèle, les contraintes disjonctives sont remplacées par n contraintes valides. Le but de cette transformation est de réduire le nombre de contraintes disjonctives du problème initial. Chaque contrainte i, $i = 1, \ldots, n$, est exprimée en fonction de la cardinalité de l'ensemble des contraintes disjonctives lié à la variable i Soit $E_i \subseteq V_i = \{j \in I, (i,j) \in E\}$ vérifiant

$$E_i = \{j \in I, (i,j) \in E \text{ and } i < j\}.$$

l'ensemble des objets qui sont incompatibles avec i.

Le modèle équivalent au mdèle initial du $DCKP$ est donné par :

$$(EDCKP)\begin{cases} \max & \sum_{j=1}^{n} p_j x_j \\ \text{subject to} & \sum_{j=1}^{n} w_j x_j \leq c \\ & |E_i|x_i + \sum_{j\in E_i} x_j \leq |E_i|,\ i = 1,\ldots,n \\ & x_i \in \{0,1\},\ i = 1,\ldots,n, \end{cases}$$

Le résultat suivant établit l'équivalence entre les deux modèles $DCKP$ et $EDCKP$.

Théorème 4.1 *Les modèles $DCKP$ et le modèle $EDCKP$ sont équivalents [46].*

Preuve : Pour montrer l'équivalence entre les modèles $DCKP$ et $EDCKP$, il suffit de montrer que toute solution réalisable du modèle $DCKP$ (respectivement du modèle $EDCKP$) est une solution réalisable du modèle $EDCKP$ (respectivement du modèle $DCKP$).

- Soit x une solution réalisable du modèle $DCKP$ et i un élément de I ayant un voisinage non vide E_i.
  Montrons que $|E_i|x_i + \sum_{j\in E_i} x_j \leq |E_i|$, sachant que $x_i \in \{0,\ 1\}$:
  Les variables x_i ne peuvent prendre que deux valeurs, la valeur 1 si la variable x_i est retenue dans la solution, la valeur 0 sinon. On distingue donc deux cas :

  cas 1 ($x_i = 1$) : Puisque E_i est un sous-ensemble de V_i, alors nécessairement $x_j = 0,\ \forall j \in E_i$. Ce qui implique que l'inégalité $|E_i|x_i + \sum_{j\in E_i} x_j \leq |E_i|$ est vérifiée car $\sum_{i\in E_i} x_j = 0$.

  cas 2 ($x_i = 0$) : Puisque pour chaque élément i de I, on a $x_i \in \{0,\ 1\}$, donc $\sum_{j\in E_i} x_j \leq |E_i|$.

- Soit x une solution réalisable du problème $EDCKP$ et (i,j) un couple appartenant à l'ensemble des contraintes disjonctives E.
  On rappelle que pour le modèle $DCKP$, le fait de dire que le couple (i,j) appartient à E est équivalent à dire que la contrainte $x_i + x_j \leq 1$ n'est pas violée. En effet, puisque $(i,j) \in E$, alors $|E_i| \geq 1$. Ainsi nous obtenons l'inégalité suivante :

$$|E_i|x_i + \sum_{j\in E_i} x_j \leq |E_i|. \tag{4.5}$$

Considérons le cas où $i < j$. On fait un raisonnement par l'absurde dans lequel on suppose que $x_i = x_j = 1$. Alors,

$$|E_i|x_i + \sum_{j \in E_i} x_j \quad = \quad |E_i|x_i + x_j + \sum_{k \in E_i \setminus \{j\}} x_k \quad \geq \quad |E_i| + 1 \tag{4.6}$$

Le couple (i, j) forme une contrainte disjonctive et donc $(i, j) \in E$ et $i < j$. Alors la contrainte (4.5) est vérifiée.

Par (4.5) et (4.6) on en déduit que

$$|E_i| + 1 < |E_i|x_i + \sum_{j \in E_i} x_j \leq |E_i|,$$

ce qui est absurde.

Le cas $i > j$ peut être démontré de la même manière, on conclut donc que le modèle $DCKP$ et le modèle $EDCKP$ sont deux modèles équivalents.

4.2.2 Contraintes dominantes

On peut remplacer certaines contraintes de cardinalité par des contraintes dominantes. En effet, il faut montrer que si pour $i \in I$ avec $|E_i| \geq 1$, il existe un entier strictement positif $\alpha_i < |E_i|$ tel que $\sum_{j \in E_i} x_j \leq \alpha_i$ ne viole pas EDCKP, ce qui implique que la contrainte $\alpha_i x_i + \sum_{j \in E_i} x_j \leq \alpha_i$ domine la contrainte $\|E_i|x_i + \sum_{j \in E_i} x_j \leq \|E_i|$. Par conséquent, il faut résoudre le programme linéaire à variables binaires suivant

$$(P) \begin{cases} \max & \sum_{j=1}^{n} p_j x_j \\ \text{subject to} & \sum_{j=1}^{n} w_j x_j \leq c \\ & \alpha_i x_i + \sum_{j \in E_i} x_j \leq \alpha_i, \ i = 1, \ldots, n \\ & x_i \in \{0, 1\}, \ i = 1, \ldots, n, \end{cases}$$

P peut être vu comme un problème avec des contraintes surrogates, où chaque $\alpha_i, \ i = 1, \ldots, n$ peut être obtenu approximativement ou de manière exacte en résolvant le problème du stable suivant :

$$(Q_i) \begin{cases} \max & \alpha_i = \sum_{j \in E_i} x_j \\ \text{s.c} & x_k + x_j \leq 1, \ \forall (k, j) \in E_i \times E_i \\ & x_j \in \{0, 1\}, \ j \in E_i. \end{cases}$$

Ce problème appartient à la classe des problèmes NP-difficile. Les contraintes de chaque Q_i sont renforcées par l'ajout de la contrainte de capacité du sac-à-dos :

$$\sum_{j \in E_i} w_j x_j \leq c.$$

Une valeur approximative α_i de la fonction objectif de Q_i peut être obtenue en utilisant la procédure de Johnson, qui peut être resumée de la manière suivante :

1. trier les éléments de Q_i selon l'ordre décroissant de leur degrés $\gamma_j,\ j \in E_i$, où γ_j est la cardinalité de l'ensemble des objets qui sont compatibles à la fois avec i et j; i.e., $\gamma_j = |E_i \cap E_j|$.
   Posons $E'_i = E_i$.
2. Fixer à 1 la variable de décision x_k dont le $\gamma_k \geq \gamma_j$ pour tout $j \in E'_i$.
3. Retirer de E'_i l'objet k et le sous-ensemble d'objets $E_i \cap E_k$ qui sont incompatible à la fois avec i et k.
4. Si $E'_i = \emptyset$, alors stop ; sinon répéter l'étape 2.

4.2.3 Propriétés mathématiques

Dans ce qui suit, nous proposons d'établir quelques proporiétés mathématiques, pour minimiser la valeur $|E_i|$.

Définition 4.2 *Soit (i_1, j_1) et (i_2, j_2) deux éléments de E. Les éléments (i_1, j_1) et (i_2, j_2) sont distincts si et seulement si le sous-ensemble $\{i_1, j_1, i_2, j_2\}$ ne contient aucun élément dupliqué.*

Soit $J \subseteq I$ et $e(J) \subseteq \Big\{(i,j) \mid i,j \in J\Big\}$ les sous-ensembles de E contenant les couples distincts selon la définition 4.2.

Proposition 4.3 *[46]*
Soit $i \in I$ un élément vérifiant $|E_i| \geq 1$ et $\beta_i = |e(E_i)|$. Alors, la contrainte

$$(|E_i| - \beta_i)x_i + \sum_{k \in E_i} x_k \leq |E_i| - \beta_i$$

est valide et domine la contrainte $(\star)$ du modèle $EDCKP$.

Preuve :

1) Par hypothèse, β_i est la cardinalité du sous-ensemble $e(E_i)$ pour $i = 1, \ldots, n$. Alors, $e(E_i)$ peut s'écrire :

$$e(E_i) = \Big\{(i_1, j_1), (i_2, j_2), \ldots, (i_{\beta_i}, j_{\beta_i})\Big\}.$$

On pose $H = \Big\{i_1, j_1, i_2, j_2, \ldots, i_{\beta_i}, j_{\beta_i}\Big\}$ et $H^{\beta_i} = E_i \setminus H$. On en déduit que

$$\sum_{k \in E_i} x_k = \sum_{k \in H^{\beta_i}} x_k + \Big[\sum_{k=1}^{\beta_i} (x_{ik} + x_{jk})\Big].$$

On remarque que $\sum_{k=1}^{\beta_i} (x_{ik} + x_{jk}) \leq \beta_i$, alors nécessairement :

$$\sum_{k \in E_i} x_k \leq \beta_i \tag{4.7}$$

L'équation (4.7) est équivalente à $(|E_i| - \beta_i)x_i + \sum_{k \in E_i} x_k \leq |E_i| - \beta_i$.

2) En utilisant 1), l'inégalité $|E_i|x_i + \sum_{k \in E_i} x_k \leq |E_i| - \beta_i$ est valide et puisque $\beta_i x_i \leq \beta_i$ (car $x_i \in \{0, 1\}$), on en déduit que $|E_i|x_i + \sum_{k \in E_i} x_k \leq |E_i|$.

La proposition 2 peut être appliquée pour déterminer les paramètres α_i pour $i = 1, \ldots, n$. Dans ce qui suit, on montre la validité de la contrainte associée à chaque α_i. Soient A le sous-ensemble de I et (P_A) le programme linéaire en $0-1$ suivant :

$$(P_A) \begin{cases} \text{maximiser} & m(A) = \sum_{j \in A} x_j \\ \text{s.c.} & \sum_{j \in I} w_j x_j \leq c \\ & x_i + x_j \leq 1, \forall (i,j) \in E \\ & x_j \in \{0, 1\},\ j \in v(i), \end{cases},$$

où $m(A)$ est l'évaluation optimale de P_A.

Proposition 4.4 *[46] Soit i un élément de E_i de voisinage non vide pour le modèle $DCKP$. La valeur minimale associée à α_i, pour laquelle l'équation* (4.8) *suivante est valide est égale à $m(E_i)$.*

$$\sum_{j \in E_i} x_j \leq \alpha_i,\ \forall i \in I. \tag{4.8}$$

Preuve : Soit $C(i)$ un sous-ensemble de E_i tel que :

$$C(i) = \{\alpha \in \mathbb{N} \mid \sum_{j \in E_i} x_j \leq \alpha_i \text{ est valide pour le modèle DCKP}\}.$$

Alors, il suffit de montrer que la valeur minimale de $C(i)$ coincide avec $m(E_i)$.

- D'abord, on montre que tout élément α_i de $C(i)$ vérifie $\alpha_i \geq m(E_i)$:
  Soit $\alpha_i \in C(i)$, alors la coupe $\sum_{j \in E_i} x_j \leq \alpha_i$ est valide pour le modèle $DCKP$. Pour toute solution réalisable x du modèle $DCKP$, on a $\sum_{j \in E_i} x_j \leq \alpha_i$. Puisque le problème P_{E_i} et le modèle DCKP ont les mêmes contraintes, alors la coupe $\sum_{j \in E_i} x_j \leq \alpha_i$ est valide pour P_{E_i}. Donc α_i est une borne supérieure pour le problème P_{E_i}.
- Ensuite, on montre que $m(E_i) \in C(i)$:
  Soit x une solution réalisable du modèle $DCKP$, alors, l'inégalité suivante est vérifiée :

$$\sum_{j \in E_i} x_j \leq m(E_i).$$

  Ainsi, $m(E_i)$ est un élément de l'ensemble $C(i)$.

On en déduit que la valeur minimale de $C(i)$ est égale à $m(E_i)$.

4.2.4 Procédure d'arrondi

Le concept d'arrondi a été largement utilisé, en optimisation combinatoire, pour générer rapidement des solutions réalisables ou non du problème. Dans certains cas, la valeur de la solution optimale de la relaxation continue du problème est arrondie. Cette solution, dans certains cas, peut être proche de la solution optimale du problème. Il existe deux types de procédé d'arrondi : Le premier type consiste à tronquer la valeur de la solution de la relaxation en continue du problème, de manière à générer une solution réalisable du problème. Le deuxième type de procédé consiste à fixer les variables fractionnaires de la solution aux valeurs entières les plus proches. Les solutions générées peuvent ne pas être réalisables. Dans ce cas, on peut appliquer un opérateur adapté au problème pour transformer la solution obtenue en solution réalisable. Salkin [79] utilise ce type de procédé pour générer une solution de départ d'un algorithme exact. L'avantage de ces heuristiques est qu'elles génèrent rapidement des solutions réalisables, mais pas souvent de bonne qualité, en particulier pour les problèmes de grande taille. Pour générer une solution de départ pour le $DCKP$, nous appliquons *la*

procédure d'arrondi (BRSP) à la relaxation en continue du problème P :

$$(LP)\begin{cases} \max & \sum_{j=1}^{n} p_j x_j \\ \text{s.t.} & \sum_{j=1}^{n} w_j x_j \leq c \\ & \alpha_i x_i + \sum_{j \in E_i} x_j \leq \alpha_i, \ i = 1, \ldots, n \\ & x_i \in [0,1], \ i = 1, \ldots, n. \end{cases}$$

0891392000 NNS1 Le but de la procédure d'arrondi BRSP est de fournir des solutions approchées du problème P, où toutes les variables sont binaires. Elle arrondit les variables fractionnaires de la solution optimale de la relaxation en continue LP du problème original. La procédure d'arrondi procéde de la façon suivante : elle résout la relaxation en continue LP du problème P, en utilisant l'algorithme du simplexe, et obtient la solution $\mathbf{x}$.

Elle fixe chaque variable x_i $i \in I$, dont la valeur dans $\mathbf{x}$ est entière. Ensuite, elle séléctionne parmi les variables non-fixées la variable x_j qui a la plus grande valeur fractionnaire. Ensuite, elle l'arrondit à sa valeur entière. Enfin, elle élimine la variable fixée du problème LP et réapplique ce processus sur le problème réduit LP. Elle répéte ces étapes jusqu'à ce que toutes les variables soient fixées à 1 ou à 0 ; jusqu'à obtenir une solution réalisable de P. La procédure d'arrondi peut être vu comme une approche à deux phases : 1. Optimisation de la relaxation en continue du problème, par aplication de la méthode du simplexe. Ensuite, fixer et arrondir une partie des variables bineaires à leurs valeurs entières. 2. Répéter l'étape précédente sur la série de problèmes réduits, jusqu'à ce qu'il ne reste plus de variables fractionnaires.

En raison de la nature gloutonne de la procédure d'arrondi et de la complexité des contraintes disjonctives du DCKP, BRSP fournit des solutions de qualité moyenne. En fait, fixer une variable de décision x_j à 1 engendre la fixation d'un sous-ensemble de variables à 0 ; ce sous-ensemble est constitué de variables qui apparaissent dans les contraintes disjonctives impliquant x_j. Ainsi, dans certains cas, BRSP peut éliminer d'une manière très agressive quelques bons objets, en fixant leurs variables de décision à 0. La procédure d'arrondi est rarement employée toute seule comme heuristique. Mais souvent en tant qu'élément d'une approche informatique plus raffinée.

4.2.5 Un premier algorithme augmenté pour le DCKP

Le premier algorithme $\mathscr{P}_1$ ajoute au problème P une borne inférieure à la valeur de sa fonction objectif, et deux contraintes de cardinalité valides. La borne requiert de la valeur de la fonction objectif d'être au moins plus grande que LB (la valeur de la meilleure solution réalisable trouvée jusque-là). Les deux contraintes de cardinalité imposent que le nombre d'objets inclus dans le sac soit encadré par $\underline{\delta}$ et $\overline{\delta}$;

$$\underline{\delta} \leq \sum_{i \in I} x_i \leq \overline{\delta}.$$

Les bornes $\underline{\delta}$ et $\overline{\delta}$ sont respectivement les valeurs des solutions optimales de

$$(F_{\min}) \begin{cases} \min & \underline{\delta} = \sum_{j=1}^{n} x_j \\ \text{s.t.} & \sum_{j=1}^{n} p_j x_j \geq LB \\ & \sum_{j=1}^{n} w_j x_j \leq c \\ & \alpha_i x_i + \sum_{j \in E_i} x_j \leq \alpha_i, \ i = 1, \ldots, n \\ & x_i \in \{0,1\}, \ i = 1, \ldots, n, \end{cases}$$

et de

$$(F_{\max}) \begin{cases} \max & \overline{\delta} = \sum_{j=1}^{n} x_j \\ \text{s.t.} & \sum_{j=1}^{n} p_j x_j \geq LB \\ & \sum_{j=1}^{n} w_j x_j \leq c \\ & \alpha_i x_i + \sum_{j \in E_i} x_j \leq \alpha_i, \ i = 1, \ldots, n \\ & x_i \in \{0,1\}, \ i = 1, \ldots, n. \end{cases}$$

La résolution de $(F_{\min})$ ou de $(F_{\max})$ en utilisant une méthode exacte peut converger très lentement . Donc, au lieu de résoudre ces problèmes de manière exacte, on résout respectivement leurs relaxations en continue $LF_{\min}$ et $LF_{\max}$ à l'aide de l'algorithme du simplexe. Les valeurs des solutions optimales obtenues $\underline{\Delta}$ et $\overline{\Delta}$ respectivement sont une borne inférieure et une borne supérieure pour $\underline{\delta}$ et $\overline{\delta}$, respectivement. Donc, les deux contraintes suivantes

doivent être ajoutées au problème P sans affecter l'optimum :

$$\sum_{j=1}^{n} x_j \geq \lceil \underline{\Delta} \rceil$$

et

$$\sum_{j=1}^{n} x_j \leq \lfloor \overline{\Delta} \rfloor.$$

Pour une borne supérieure donnée $\Delta \in [\underline{\Delta},\ \overline{\Delta}]$, le problème associé est :

$$(P^{\Delta}) \begin{cases} LB_{\Delta} = \max & \displaystyle\sum_{j=1}^{n} p_j x_j \\ \text{s.t.} & \displaystyle\sum_{j=1}^{n} p_j x_j \geq LB \\ & \displaystyle\sum_{j=1}^{n} w_j x_j \leq c \\ & \displaystyle\sum_{j=1}^{n} x_j \leq \lfloor \overline{\Delta} \rfloor \quad \text{et} \quad -\sum_{j=1}^{n} x_j \leq -\lceil \underline{\Delta} \rceil \\ & \alpha_i x_i + \displaystyle\sum_{j \in E_i} x_j \leq \alpha_i,\ i = 1, \ldots, n \\ & x_i \in \{0,1\},\ i = 1, \ldots, n, \end{cases}$$

où LB_{Δ} dénote la borne inférieure qui satisfait à la fois les deux contraintes de cardinalité. La valeur de sa fonction objectif est plus grande que la meilleure borne inférieure connue LB.

L'algorithme $\mathscr{P}_1$ détaillé dans la figure 4.1 est une procédure itérative. Il résout une serie de LP^{Δ} les relaxations linéaires de $P^{\Delta},\ \Delta \in [\underline{\Delta}, \overline{\Delta}.]$ Il applique la procédure d'arrondi BRSP à chaque solution fractionnaire pour obtenir une solution réalisable de P. Il retient parmi toutes les solutions réalisables de P celle qui a la plus grande valeur.

4.2.6 Un deuxième algorithme augmenté pour le DCKP

Comme l'algorithme $\mathscr{P}_1$, le deuxième algorithme $\mathscr{P}_2$ résout une serie de problèmes augmentés par une borne de la valeur de la fonction objectif et des contraintes de cardinalité. Sauf que ces problèmes subissent une réduction au préalable. En effet, $\mathscr{P}_2$ démarre en fixant une partie des variables fractionnaires à des valeurs entières ; ensuite, il résout les problèmes réduits résultants. $\mathscr{P}_2$ procède de la façon suivante :

1. Résoudre la relaxation linéaire de P.

FIG. 4.1 – l'algorithme de $\mathscr{P}_1$

`Entrée :` Une instance du DCKP.
`Sortie :` Une solution approchée $\hat{\mathbf{x}}$.
`Phase d'Initialisation`
- Soit LB une borne inférieure initiale de la solution optimale du DCKP. Si une telle borne n'est pas disponible, poser $LB = 0$.
- Résoudre $LF_{\min}$ et $LF_{\max}$, et poser $\underline{\Delta}$ et $\overline{\Delta}$(respectivement) égales aux valeurs des solutions optimales de $LF_{\min}$ et $LF_{\max}$ (respectivement).
- Poser $\underline{\Delta}' = \lceil \underline{\Delta} \rceil$ et $\overline{\Delta}' = \lfloor \overline{\Delta} \rfloor$.

`Phase Itérative`
`Répéter`
(a) Poser $\Delta := \overline{\Delta}'$.
(b) Résoudre LP^{Δ}, la relaxation linéaire de P^{Δ}, en utilisant l'algorithme du simplexe. Soit UB_{Δ} la valeur de sa solution optimale.
(c) Si $UB_{\Delta} > LB$, alors
appliquer BRSP à la solution optimale (fractionaire) de LP^{Δ} pour avoir une solution à variables entières réalisable $\mathbf{x}$. Soit LB_{Δ} la valeur de sa fonction objectif.
Si $LB < LB_{\Delta}$, alors poser $LB = LB_{\Delta}$. Mettre à jour la meilleure solution $\hat{\mathbf{x}}$ en posant $\hat{\mathbf{x}} = \mathbf{x}$.
(d) Réduire l'intervalle des contraintes de cardinalité en posant $\overline{\Delta}' = \overline{\Delta}' - 1$.
`Jusqu'à` $\overline{\Delta}' = \underline{\Delta}'$.

---

2. Fixer les valeurs de ℓ variables fractionnaires à 1 ;
   fixer à 0 les variables correspondantes aux objets incompatibles avec un ou plusieurs objets des ℓ objets fixés au préalable ; et
   générer le problème résultant de ces fixations.
3. Résoudre la relaxation linéaire du problème obtenu lors de l'étape 2 en utilisant l'algorithme du simplexe.
4. En se basant sur la solution optimale obtenue lors de l'étape 3, réduire le problème en fixant à leurs valeurs courantes toutes les variables x_j qui ont des valeurs entières et toute variable x_i incompatible avec un objet j tel que x_j a été fixé à 1.
5. Augmenter le problème résultant en utilisant les contraintes de cardinalité et la borne de la valeur de la fonction objectif. Résoudre le tout avec l'algorithme du simplexe.
6. Appliquer la procédure BRSP pour obtenir une solution réalisable.

Plusieurs stratégies peuvent être appliquées pour réduire la taille du problème original. l'objectif des stratégies de réduction est de fixer certaines variables à l'optimum ; donc, elles réduisent la taille du problème et facilitent la résolution de plusieurs instances. La stratégie suivie est de fixer les variables aux valeurs entières de la solution donnée par l'algorithme du simplexe.

FIG. 4.2 – L'algorithme $\mathscr{P}_2$ avec $\ell = 2$

`Entrée :` Une instance de P.
`Sortie :` Une solution approchée $\hat{\mathbf{x}}$ dont la valeur est LB.
`Phase d'Initialization`
– soit $I_f = \{x_1, \ldots, x_p\}$ l'ensemble initial des variables fractionnaires et $p = |I_f|$ sa cardinalité associée.
– Soit LB une borne inférieure initiale de P. (LB peut être égale à zero.)
`Phase Itérative`
`Pour` $f = 1, \ldots, p$

(a) soit $P_0^f,\ f = 1, \ldots, p-1$, le problème avec x_f et x_{f+1} fixées à 1.

(b) Résoudre la relaxation linéaire de P_0^f, et soit P_1^f le problème réduit obtenu, en enlevant de P_0^f toutes les variables $x_j, j \in I$, tels quet $x_j = 1$ et toutes les variables $x_i, i \in E_j$.

(c) Soit $S = \{j \in I \ tel \ que \ x_j = 1\}$, et poser $\phi = \sum_{j \in S} p_j$.

(d) Augmenter P_1^f avec les contraintes de cardinalité et la contrainte sur la valeur de la fonction objectif $\sum_{j \in I} p_j x_j \geq LB - \phi$, résoudre la relaxation linéaire de ce problème.

(e) Appliquer BRSP à la solution optimale (fractionnaire) de la relaxation en continue du problème augmenté P_1^f, et soit LB_f la valeur de la solution optimale obtenue.

(f) Si $LB < LB_f + \phi$, alors poser $LB = LB_f + \phi$ et mettre à jour la meilleure solution $\hat{\mathbf{x}}$.

(g) Réduire l'ensemble des variables fractionnaires : Poser $I_f = I_f \setminus \{f\}$.

4.2.7 Hill Climbing

Cette méthode que nous appelons (HC) démarre sa recherche locale avec la solution obtenue par la procédure d'arrondi BRSP. Elle applique une stratégie d'intensification qui provoque quelques petites perturbations sur la solution en cours. Ensuite elle analyse leurs

effets. Si les perturbations augmentent la valeur de la fonction objectif, alors (HC) retient ces perturbations (i.e, elle met à jour la valeur de la meilleure solution $\tilde{\mathbf{x}}$) ; autrement les perturbations sont rejetées.

FIG. 4.3 – L'algorithme HC

`Entrée :` Une solution réalisable $\hat{\mathbf{x}} = (\hat{x}_1, \ldots, \hat{x}_n)$.
swap_type
`Sortie :` Une solution améliorée $\tilde{\mathbf{x}} = (\tilde{x_1}, \ldots, \tilde{x_n})$.
`Initialisation`
Poser $\tilde{\mathbf{x}} = \hat{\mathbf{x}}$.
Poser $S = \left\{ j \mid \hat{x}_j = 1,\ \forall j \in I \right\}$, and $\underline{z} = \sum_{j \in S} p_j$.

`Phase Itérative`
Poser tabu_list $= \emptyset$.
Pour $i = 1, \ldots, n-1$,
Si $\hat{x}_i = 1$
– Utiliser l'approche de la figure 4.4 pour générer N_i, l'ensemble des solutions voisines obtenues en enlevant du sac l'objet i (c.à.d, en posant $\hat{x}_i = 0$), et en insérant itérativement dans le sac un sous-ensemble d'objets de I, tant que cela ne provoque pas d'incompatibilités avec les objets déjà mis dans le sac et ne viole pas la contrainte de capacité.
– Soit $\mathbf{x}^{(i)}$ la meilleure solution réalisable dans N_i, et $z(\mathbf{x}^{(i)})$ sa valeur ; i.e., $z(\mathbf{x}^{(i)}) = \min_{\mathbf{x} \in N_i} \{z(\mathbf{x})\}$.
– Si $z(\mathbf{x}^{(i)}) > \underline{z}$, alors
pour tout $k \in I$ tel que $\hat{x}_k \neq x_k^{(i)}$, entrer (i, k) dans la tabu_list ;
poser $\underline{z} = z(\mathbf{x}^{(i)})$, et $\tilde{\mathbf{x}} = \mathbf{x}^{(i)}$;
si swap_type = exhaustif, poser $\hat{\mathbf{x}} = \mathbf{x}^{(i)}$.

La procédure d'amélioration décrite dans la figure 4.3 perturbe légèrement la solution en cours $\hat{\mathbf{x}}$. En effet, elle procède en supprimant itérativement du sac quelques objets et en les remplaçant par d'autres. Elle commence par initialiser la solution courante $\hat{\mathbf{x}}$ et la meilleur solution $\tilde{\mathbf{x}}$ avec la solution réalisable donnée par la procédure BRSP. Ensuite elle supprime itérativement chaque objet i qui apparaît dans $\hat{\mathbf{x}}$ et le remplace par un sous-ensemble d'objets qui est susceptible d'être ajouté au sac sans violer les contraintes disjonctives. Ce cas est illustré dans la figure 4.4. Comme il existe plusieurs sous-ensembles d'objets potentiellement éligibles pour remplacer l'objet i sans violer les différentes contraintes, l'ajout de chaque sous-ensemble conduit à une solution voisine réalisable. L'ensemble des solutions réalisables voisines de $\hat{\mathbf{x}}$ forme le voisinage N_i. Si le meilleur voisin $\mathbf{x}^{(i)} \in N_i$ améliore $\tilde{\mathbf{x}}$, HC met à jour

FIG. 4.4 – La génération du voisinage N_i

`Input` : Une solution réalisable $\hat{\mathbf{x}}$;
i l'objet qui sera éliminé de $\hat{\mathbf{x}}$.

`Output` : L'ensemble N_i des solutions voisines de $\hat{\mathbf{x}}$ en faisant un interchange entre i et un autre objet.

1. Poser $N_i = \emptyset$.
2. Poser $S = \{k \in I, \hat{x}_k = 1\}$ où S est l'ensemble des objets inclus dans le sac.
3. Poser $\overline{S} = I \setminus S$.
4. enlever i de S, et poser $c' = \sum_{k \in S} w_k$.
5. Poser I" $= \{j \in \overline{S}$ tel que $\forall k \in S$ il n'existe pas un $(k, j) \in E$ ou $(j, k) \in E\}$.
6. Pour $j = i + 1, \ldots, n$
   (a) Poser $S' = S$, et $k = j$.
   (b) Si $k \in I'$ et $c' + w_k \leq c$,
   Poser $S' = S' \cup \{k\}$, $I' = I$" $\setminus \{k\}$, et $c' = c' + w_k$.
   Si $I' \neq \emptyset$ et $c' < c$, poser $k = k + 1$, et aller à l'étape 6b.
   Soit $\mathbf{x}$ la solution obtenu en posant $x_k = 1$ pour $k \in S'$, et 0 autrement.
   Insérer $\mathbf{x}$ dans N_i et calculer $z(\mathbf{x}) = \sum_{k \in S'} p_k$.

$\tilde{\mathbf{x}}$, et rend tabou l'interchange qui a transformé $\hat{\mathbf{x}}$ vers $\tilde{\mathbf{x}}$. En effet, HC évite le cyclage en interdisant tous les interchanges de la liste tabou. La liste est mise à jour dynamiquement. En plus, si la solution est améliorée avec un 2-opt exhaustif, HC met à jour la solution courante $\hat{\mathbf{x}}$ en la posant égale à $\tilde{\mathbf{x}}$.

Le temps d'exécution de la procédure d'amélioration est de $O(n^2)$. En effet, une instance de P peut être représentée par une matrice n fois n ou par n vecteurs, où chaque vecteur $N_i,\ i = 1, \ldots, n$, contient les voisins de l'objet i. Donc par conséquent, l'étape d'initialisation requière $O(n)$ opérations. La phase de suppression demande $O(n^2)$ opérations. alors que la phase d'ajout demande $O(n^2)$ opérations.

4.3 Partie expérimentale

Dans cette section nous étudions le comportement des deux algorithmes sur deux groupes d'instances (issus de Hifi et Michrafy [43] et générés selon le schéma de Yamada *et al.* [89, 90]).

TAB. 4.1 – Performance de $\mathscr{P}_1$

n	ρ	c	$\|E\|$	Instance	z_{Cplex}	$\underline{z}$	z_1	t_1	z_{1s}	t_{1s}	z_{1e}	t_{1e}
500	1800	0.1	12475	1IA1	2221	2550	2172	4	2542	95	2555	168
				1IA2	2210	2550	2273	4	2558	109	2564	199
				1IA3	2240	2280	2100	4	2270	124	2288	200
				1IA4	2220	2260	2192	4	2241	103	2261	169
				1IA5	2228	2250	2158	4	2241	111	2259	193
		0.2	24950	2IA1	1914	2067	1581	4	2025	184	2037	256
				2IA2	1575	2060	1642	4	1971	165	2018	216
				2IA3	1658	2080	1484	4	2078	185	2078	256
				2IA4	1713	2080	1506	4	2030	125	2053	253
				2IA5	1632	2068	1607	4	1945	174	1982	252
		0.3	37425	3IA1	1215	1587	1219	4	1623	211	1632	264
				3IA2	1170	1732	1125	4	1683	197	1683	243
				3IA3	1291	1619	1053	4	1573	246	1573	313
				3IA4	1234	1686	1074	4	1598	240	1598	293
				3IA5	1371	1620	1061	4	1534	217	1620	270
		0.4	49900	4IA1	900	1072	921	4	1126	272	1126	331
				4IA2	842	1022	896	4	1007	231	1013	301
				4IA3	815	1217	868	4	945	252	1111	327
				4IA4	986	1208	883	4	1277	298	1277	367
				4IA5	914	1283	947	4	1197	268	1311	305
1000		0.05	24975	5I1	2600	2610	2480	4	2565	418	2595	1211
				5I2	2599	2610	2460	4	2544	479	2653	1190
				5I3	2610	2610	2467	6	2556	369	2650	1061
				5I4	2590	2590	2533	6	2565	442	2640	1197
				5I5	2598	2610	2468	6	2560	442	2625	1149
	2000	0.06	29970	6I1	2740	2760	2619	6	2687	482	2729	1412
				6I2	2687	2740	2650	6	2681	441	2708	1266
				6I3	2732	2740	2686	6	2710	432	2754	1427
				6I4	2729	2760	2638	6	2692	579	2760	1425
				6I5	2738	2740	2631	6	2712	528	2760	1349
		0.07	44955	7I1	2639	2660	2624	6	2652	612	2713	1341
				7I2	2660	2700	2613	6	2640	538	2698	1302
				7I3	2655	2670	2641	6	2669	464	2669	1186
				7I4	2659	2660	2651	6	2669	589	2702	1397
				7I5	2612	2670	2589	6	2666	559	2710	1281
		0.08	39960	8I1	2599	2630	2587	6	2606	359	2668	745
				8I2	2599	2620	2466	6	2607	340	2609	714
				8I3	2566	2600	2596	6	2617	344	2662	725
				8I4	2555	2607	2585	6	2610	348	2627	749
				8I5	2546	2620	2523	6	2600	357	2600	709
		0.09	44955	9I1	2548	2599	2470	6	2576	769	2602	1392
				9I2	2548	2590	2532	6	2557	692	2630	1300
				9I3	2528	2590	2455	6	2569	683	2601	1396
				9I4	2517	2580	2468	6	2562	697	2609	1398
				9I5	2540	2570	2501	6	2566	656	2619	1312
		0.1	49950	10I1	2479	2540	2464	6	2530	719	2570	1400
				10I2	2537	2599	2352	6	2539	761	2609	1452
				10I3	2527	2540	2427	6	2517	670	2576	1303
				10I4	2498	2590	2392	6	2530	692	2592	1323
				10I5	2496	2500	2308	6	2519	609	2590	1171

Le premier groupe contient 20 instances de taille moyenne avec $n = 500$ objets, la capacité $c = 1800$, et différentes densités). Le second groupe contient 30 instances corrélées, où chaque instance contient 1000 objets, avec c variant entre 1800 et 2000. Les algorithmes sont codés en C et testés sur un `UltraSparc-II` (450Mhz et 2Gb of RAM) ; la même machine utilisée

par Hifi et Michrafy [43].

4.3.1 Performance du premier algorithme augmenté ($\mathscr{P}_1$)

L'étape c de la partie itérative de $\mathscr{P}_1$ utilise la procédure d'arrondi BRSP pour transformer la solution fractionnaire de l'étape b en une solution binaire réalisable $\hat{\mathbf{x}}$. L'application de la méthode Hill Climbing sur la solution $\hat{\mathbf{x}}$ conduit en général à une solution $\tilde{\mathbf{x}}$ de meilleure qualité. De là, nous évaluons et comparons les performances de $\mathscr{P}_1$, $\mathscr{P}_{1s}$ et $\mathscr{P}_{1e}$ où $\mathscr{P}_{1s}$, et $\mathscr{P}_{1e}$ appliquent la méthode Hill Climbing avec respectivement le $2-opt$ standard et le $2-opt$ exhaustif à chaque solution obtenue par la procédure d'arrondi BRSP.

La table 4.1 compare les résultats de $\mathscr{P}_1$, $\mathscr{P}_{1s}$ et $\mathscr{P}_{1e}$ à ceux obtenus par le Cplex et la recherche locale réactive de Hifi et Michrafy [43]. Dans les colonnes 1–5 on a la taille du problème n, la densité ρ des contraintes disjonctives (calculée à partir de la matrice d'adjacence), la capacité du sac-à-dos c, le nombre de contraintes disjonctives $|E|$, et le nom de l'instance. La colonne 6 montre z_{Cplex} la valeur de la meilleure solution réalisable obtenue par le Cplex v9.0, après 1 heure de temps d'exécution, alors que la colonne 7 montre la meilleure solution connue de la littérature $\underline{z}$ obtenue par *RLS*. Finalement, les colonnes 8, 10, et 12 montrent z_1, z_{1s}, et z_{1e} les valeurs des solutions obtenues respectivement par $\mathscr{P}_1$, $\mathscr{P}_{1s}$, et $\mathscr{P}_{1e}$; alors que les colonnes 9, 11 et 13 affichent t_1, t_{1s}, et t_{1e}, les temps d'exécution correspondant.

L'analyse des résultats de la table 4.1 indique les points suivants.

- Bien que l'algorithme $\mathscr{P}_1$ améliore les résultats obtenus par le Cplex pour 9 instances (sept moyennes et deux instances de grande taille) avec un temps d'exécution très court (4 à 6 secondes), il échoue à améliorer les solutions obtenues par RLS. Cet échec est dû au fait que la procédure BRSP fixe les variables fractionnaires de manière agressive.
- l'algorithme $\mathscr{P}_{1s}$ améliore toutes les solutions obtenues par le Cplex pour les instances de taille moyenne. Il améliore 18 parmi 30 solutions pour les instances de grande taille. En plus, il améliore 8 solutions obtenues par RLS (4 instances de taille moyenne et 4 instances de grande taille). De plus, il est plus performant que l'algorithme $\mathscr{P}_1$ en terme de qualité de solution, avec en moyenne une amélioration de 12.33%. La plus grande amélioration est de 49.60% pour 3IA2. La recherche locale effectuée par la méthode Hill Climbing a un rôle important dans les performances de $\mathscr{P}_{1s}$. Ces performances ont été obtenues au prix d'un temps d'exécution non-négligeable.
- $\mathscr{P}_{1e}$ améliore 31 (resp. 48) parmi 50 solutions obtenues par RLS (resp. Cplex), alors

qu'il atteint 2 (resp. 0) autres solutions. Il améliore toutes les solutions obtenues $\mathscr{P}_1$ avec une moyenne d'amélioration de 14.53% et atteint 52.69% pour 3IA5. D'une façon simulaire, $\mathscr{P}_{1e}$ augmente 42 et égale 8 des solutions obtenues par $\mathscr{P}_{1s}$, avec une moyenne d'amélioration de 1.99% et atteint 17.57% pour 4IA3. Ceci prouve que le $2-opt$ exhaustif est plus efficace que le $2-opt$ standard.

4.3.2 Performance du deuxième algorithme augmenté ($\mathscr{P}_2$)

L'étape e de la phase itérative de l'algorithme $\mathscr{P}_2$ fixe ℓ variables fractionnaires à 1. Le choix de ℓ influe sur les performances de l'algorithme. La table 4.2 montre la variation de n_{Best} le nombre de fois que $\mathscr{P}_2$ améliore la meilleure solution connue dans la littérature et $\underline{t}_2$ et $\bar{t}_2$ le plus petit et le plus grand temps d'exécution pour différentes valeurs de ℓ. La table 4.2 montre que $\ell = 2$ fournit les meilleurs résultats en un temps d'exécution raisonnable. Par conséquent, ce choix de ℓ est adopté pour le reste du chapitre.

TAB. 4.2 – Performance de $\mathscr{P}_2$ en fonction de ℓ

	$\ell = 1$	$\ell = 2$	$\ell = 3$
n_{Best}	0/50	3/50	0/50
$\underline{t}_2$	52	45	25
$\bar{t}_2$	305	216	156

L'étape e de la phase itérative de $\mathscr{P}_2$ appelle la procédure BRSP pour transformer la solution fractionnaire en une solution réalisable $\hat{\mathbf{x}}$ pour le DCKP. Puisque la procédure BRSP ne conduit pas toujours à des solutions de bonne qualité, nous améliorons les solutions obtenues par $\mathscr{P}_2$ en appliquant la méthode Hill Clibing sur chaque solution réalisable obtenue par BRSP. De là, nous évaluons et comparons les performances de $\mathscr{P}_2$, $\mathscr{P}_{2s}$, et $\mathscr{P}_{2e}$ où $\mathscr{P}_{2s}$ et $\mathscr{P}_{2e}$ applique aux solutions obtenues par la procédure BRSP la méthode Hill Climbing couplée respectivement au $2-opt$ standard et au $2-opt$ exhaustif.

La table 4.3 présente les résultats de $\mathscr{P}_2$, $\mathscr{P}_{2s}$ et $\mathscr{P}_{2e}$. Les colonnes 1–5 montrent n, ρ, c, $|E|$ et le nom de l'instance. Les colonnes 6 et 7 montrent z_{Cplex} et $\underline{z}$. La colonne 8 présente $\underline{z}_1 = \max\{z_1,\ z_{1s},\ z_{1e}\}$, les meilleures solutions obtenues par les trois versions du deuxième algorithme augmenté. Finalement, les colonnes 9, 11, et 13 montrent z_2, z_{2s}, et

z_{2e} les valeurs des solutions obtenues respectivement par $\mathscr{P}_2$, $\mathscr{P}_{2s}$ et $\mathscr{P}_{2e}$; alors que les colonnes 10, 12 et 14 montrent t_2, t_{2s} et t_{2e} les temps d'exécution correspondants.

- L'algorithme $\mathscr{P}_2$ améliore les solution obtenues par le Cplex pour 19 instances (10 instances de taille moyenne et 9 instances de grande taille). Il améliore les solutions obtenues par RLS pour 3 instances de grande taille.
- L'algorithme $\mathscr{P}_{2s}$ améliore toutes les solutions obtenues par le Cplex pour les instances de taille moyenne et 19 instances de grande taille. En plus, il améliore 12 solutions obtenues apr RLS (4 pour les instances moyennes et 8 pour les instances de grande taille). Il montre de meilleures performances que l'algorithme $\mathscr{P}_2$ en terme de qualité de solution, avec une amélioration de 6.96% en moyenne. La plus grande amélioration est obtenue à 38.94% pour l'instance 3IA2.
- L'algorithme $\mathscr{P}_{2e}$ améliore 37 (resp. 48) parmi 50 solutions obtenues par RLS (resp. Cplex), alors qu'il atteint 2 (resp. 0) autres solutions. En moyenne il améliore 11.45% (resp. 1.26) et atteint en moyenne 47.98% (resp. 18.49). Il améliore toutes les solutions obtenues par $\mathscr{P}_2$ avec en moyenne une amélioration de 11.95% qui atteint 41.59% pour 3IA1. De manière similaire, $\mathscr{P}_{2e}$ améliore 49 solutions obtenues par $\mathscr{P}_{2s}$ avec en moyenne une amélioration de 4.57% et atteint 28.70% pour 3IA4. Ceci prouve que le $2-opt$ exhaustif appliqué par la méthode Hill Climbing est plus efficace que le $2-opt$ standard. Cependant, cette amélioration augmente encore le temps d'exécution. Finalement, z_{2e} égale $\underline{z}_1$ dans 24 instances et le dépasse dans 24 autres cas.

4.4 Conclusion

Nous avons proposé deux algorithmes augmentés. Ces algorithmes sont basés sur une procédure d'arrondi et sur les contraintes d'encadrement. L'objectif étant de résoudre de manière approximative le problème du sac-à-dos à contraintes disjonctives. Le premier algorithme applique une stratégie d'arrondi à la relaxation en continue du modèle équivalent couplée avec des contraintes d'encadrement valides et une borne sur la valeur de la solution optimale. Le deuxième algorithme résout la relaxation en continue d'une série de problèmes réduits. Pour réduire les problème, une partie des variables fractionnaires est fixée au préalable. Les performances des deux algorithmes ont été étudiées sur une série d'instances fortement corrélées, et comparées avec les meilleurs résultats de la littérature. Les résultats montrent que les deux algorithmes couplés avec la méthode Hill Climbing sont capables de résoudre des instances fortement corrélées, en un temps relativement raisonnable.

TAB. 4.3 – Performance of $\mathscr{P}_2$

n	ρ	c	$\|E\|$	Instance	z_{Cplex}	$\underline{z}$	$\underline{z}_1$	z_2	t_2	z_{2s}	t_{2s}	z_{2e}	t_{2e}
500	0.1	1800	12475	1IA1	2221	2550	2555	2264	58	2542	155	2555	251
				1IA2	2210	2550	2564	2274	60	2560	140	2571	243
				1IA3	2240	2280	2288	2222	72	2266	133	2288	225
				1IA4	2220	2260	2261	2176	53	2253	119	2280	208
				1IA5	2228	2250	2259	2230	54	2259	118	2269	146
	0.2		24950	2IA1	1914	2067	2037	1542	50	2029	156	2037	222
				2IA2	1575	2060	2018	1639	47	1970	149	2018	220
				2IA3	1658	2080	2078	1575	51	1781	147	2078	216
				2IA4	1713	2080	2053	1657	50	2033	174	2053	238
				2IA5	1632	2068	1982	1749	48	1768	153	2002	214
	0.3		37425	3IA1	1215	1587	1632	1171	45	1345	177	1658	234
				3IA2	1170	1732	1683	1212	45	1684	170	1684	221
				3IA3	1291	1619	1573	1186	58	1327	197	1589	249
				3IA4	1234	1686	1598	1202	52	1310	197	1686	262
				3IA5	1371	1620	1620	1218	53	1610	205	1678	265
	0.4		49900	4IA1	900	1072	1126	998	64	1073	246	1126	285
				4IA2	842	1022	1013	1012	64	1123	227	1211	266
				4IA3	815	1217	1111	1087	58	1103	233	1206	243
				4IA4	986	1208	1277	932	65	1113	252	1278	292
				4IA5	914	1283	1311	955	61	1063	238	1311	282
1000	0.05		24975	5I1	2600	2610	2595	2551	163	2565	432	2595	1546
				5I2	2599	2610	2653	2545	220	2573	445	2653	1782
				5I3	2610	2610	2650	2532	204	2558	550	2650	1812
				5I4	2590	2590	2640	2572	213	2580	495	2640	1853
				5I5	2598	2610	2625	2479	223	2589	482	2649	1762
	0.06	2000	29970	6I1	2740	2760	2729	2616	183	2707	552	2780	1944
				6I2	2687	2740	2708	2700	166	2710	428	2750	1388
				6I3	2732	2740	2754	2579	178	2704	490	2760	1478
				6I4	2729	2760	2760	2700	280	2710	627	2760	1791
				6I5	2738	2740	2760	2714	254	2723	592	2760	1670
	0.07		44955	7I1	2639	2660	2713	2611	305	2679	536	2726	1748
				7I2	2660	2700	2698	2709	281	2712	606	2720	1717
				7I3	2655	2670	2669	2639	204	2661	468	2710	1572
				7I4	2659	2660	2702	2569	213	2681	537	2710	1529
				7I5	2612	2670	2710	2568	246	2704	591	2710	1663
	0.08		39960	8I1	2599	2630	2668	2600	224	2606	724	2668	1540
				8I2	2599	2620	2609	2522	202	2618	566	2655	1517
				8I3	2566	2600	2662	2619	226	2625	583	2680	1481
				8I4	2555	2607	2627	2564	239	2621	587	2660	1582
				8I5	2546	2620	2600	2552	284	2583	570	2620	1474
	0.09		44955	9I1	2548	2599	2602	2552	284	2583	570	2602	1474
				9I2	2548	2590	2630	2504	241	2569	578	2630	1508
				9I3	2528	2590	2601	2568	217	2569	507	2601	1390
				9I4	2517	2580	2609	2536	218	2566	642	2620	1730
				9I5	2540	2570	2619	2468	225	2580	566	2619	1405
	0.1		49950	10I1	2479	2540	2570	2418	223	2530	681	2570	1569
				10I2	2537	2599	2609	2411	273	2528	651	2538	1513
				10I3	2527	2540	2576	2383	245	2529	630	2541	1470
				10I4	2498	2590	2592	2486	266	2527	646	2592	1504
				10I5	2496	2500	2590	2527	201	2519	521	2590	1315

Chapitre 5

Branchement local et programmes linéaires en nombres entiers

Ce chapitre est constitué de deux parties. La première partie aborde le concept de branchement local pour les programmes linéaires en nombres entiers mixtes. La deuxième partie propose un algorithme pour la résolution approchée du problème du sac-à-dos à contraintes disjonctives. Cet algorithme, qui peut être vu comme une méthode à deux phases complémentaires, est une adaptation du concept de branchement local au DCKP. La première phase utilise une stratégie d'arrondi pour fixer une partie des variables fractionnaires. Alors que la deuxième phase résout, de manière exacte, le problème réduit. Les performances de cet algorithme sont évaluées sur un ensemble d'instances de la littérature. Et les résultats obtenus sont comparés aux meilleures solutions de la littérature. La méthode proposée est capable d'améliorer plusieurs solutions pour ces instances, dont une partie ne peut être résolue à l'optimum en un temps raisonnable.

Ce chapitre a fait l'objet d'une présentation à la Conférence Internationale CIE39 (cf., Hifi, Negre and Ould Ahmed Mounir [38]).

5.1 Introduction

La programmation linéaire en nombres entiers joue un rôle important dans la modélisation des problèmes d'optimisation combinatoire difficiles à résoudre (NP-Difficile). Cependant, trouver des méthodes exactes capables de résoudre ces problèmes se revèle comme étant une

tâche difficile, voire impossible surtout si les instances de ces problèmes sont de très grande taille. D'où l'interêt du developpement des méthodes de résolution approchées (heuristiques).

Bien que plusieurs heuristiques ont été proposées pour résoudre differents types de problèmes, très peu de travaux ont abordé la question des heuristiques génériques pour les Programmes linéaires mixtes en nombres entiers.

De nos jours, les solveurs de programmes linéaires mixtes en nombres entiers sont considérés comme des outils sophistiqués, capables de délivrer avec un temps relativement acceptable des solutions optimales ou approchées pour des instances de taille significative. En fait, dans de nombreux cas pratiques, ils permettent de trouver des solutions de bonne qualité, et ce tout au début de la phase de calcul. C'est en ce sens qu'apparait le rôle important du comportement approximatif du solveur : la stratégie agressive, qui donne une solution tout au début des étapes de calcul, est preférée à la stratégie qui trouve une bonne solution mais seulement à la fin des étapes de calcul. Plusieurs solveurs permettent à l'utilisateur d'avoir un certain contrôle sur le comportement de leurs heuristiques, à travers un certain nombre de paramètres. Malheureusement, dans certains cas difficiles, l'utilisation des solveurs apparait non appropriée au vu du temps d'exécution. L'existence de méthodes de résolution (exactes ou approchées) capables de résoudre efficacement des programmes linéaires en nombres entiers est d'une importance cruciale pour de nombreuses applications pratiques. Dans ce chapitre, nous examinons l'apport des solveurs génériques dans la résolution de programmes linéaires mixte en nombres entiers, en étudiant une approche de résolution appelée branchement local.

5.2 Branchement local

Cette approche est dans le même esprit que la recherche locale, sauf que le voisinage est obtenu grâce à l'introduction, dans le modèle, d'inégalités linéaires génériques appelées coupes de branchement. La stratégie de branchement local est à l'origine une méthode exacte bien qu'elle vise à améliorer le comportement approximatif des solveurs de programmes linéaires mixte en nombres entiers. Elle alterne deux niveaux de branchement :

- un branchement à haut niveau, qui sert à définir le voisinnage d'une solution,
- un branchement à bas niveau, qui sert à explorer ce voisinnage.

Le branchement local vise à améliorer la qualité de la solution au début de l'exploration de l'espace de recherche mais aussi au début de chaque étape de calcul.

5.2.1 Fixation rigide contre fixation souple

L'heuristique la plus souvent utilisée, dans le branchement local, est basée sur le concept de fixation rigide de variables. Ce concept se décline de la manière suivante :

On suppose qu'on a un solveur, ou une boite noire, capable de résoudre, aussi bien de manière exacte que de manière approchée, l'instance du problème traité. Au départ, le solveur est appliqué sur l'instance du problème, mais ses paramètres sont posés de manière à arrêter, rapidement, l'exécution et retourner une solution $\bar{x}$ qui peut être non réalisable. Cette solution peut être calculée de plusieurs manières :

−Arrondir une partie des variables fractionnaires de la solution de la relaxation en continue du problème à leurs valeurs supérieures.
−Générer une solution du problème étudié grâce à une heuristique. La solution $\bar{x}$ est ensuite analysée et une partie de ses variables est à arrondie aux entiers les plus proches. Le processus est ensuite réappliqué, itérativement, au problème réduit, issu de la fixation des variables : la boite noire (solveur) est appelée, une nouvelle solution est trouvée et une partie de ses variables est fixée, ainsi de suite. De cette manière, la taille du problème est réduite après chaque fixation. La boite noire peut alors calculer des sous-problèmes avec des noyaux de plus en plus petits et avec une probabilité croissante de les résoudre à l'optimum.

La difficulté réside dans le choix des variables à fixer à chaque étape. En effet, pour des problèmes réputés difficiles, les solutions de bonne qualité sont obtenues après plusieurs étapes de fixation. D'autre part, détecter rapidement les mauvais choix de variables fixées s'avère être une tâche très difficile, même si on dispose de bornes qui encadrent la solution optimale avant chaque fixation : la borne reste, des fois, presque inchangée pour plusieurs fixations. Mais il arrive un moment où elle descent subitement, après une dernière fixation. Pour éviter les mauvaises fixations, on peut intégrer le mécanisme de backtraking dans ce processus de fixation, ce qui n'est pas chose facile.

La question est alors, comment fixer une bonne partie des variables fractionnaires sans perdre la possibilité de tomber sur une solution réalisable de bonne qualité. Pour mieux illustrer ce point de vue, supposons qu'une heuristique, quelconque, nous donne une solution $\bar{x}$ de taille n et à variables binaires. Supposons aussi qu'on désire fixer à 1 au moins 90% des variables non nulles de $\bar{x}$. La question qui se pose est alors comment choisir les variables qui vont être fixées. La réponse pourrait être : ajouter au modèle traité la contrainte suivante :

$$\sum_{j=1}^{n} \bar{x}_j x_j \geq \lceil 0.9 \sum_{j=1}^{n} \bar{x}_j \rceil$$

Ensuite, appliquer le solveur au modèle qui en résulte. De cette façon, nous évitons la fixation rigide des variables, en faveur d'une condition plus fléxible, qui définit le voisinage de la solution en cours. Enfin, le solveur se charge d'explorer le voisinage de cette solution.

5.2.2 Schéma de branchement local

Le mécanisme de fixation souple, vu dans la section précedente, conduit, naturellement, au schéma général de branchement local décrit dans la partie suivante : Considérons le programme linéaire mixte en nombres entiers noté P :

$$\begin{array}{lll} \text{Minimiser} & c^T x & \\ & Ax \geq b & \\ & x_j \geq 0 & \forall j \in \mathcal{G},\ x_j \text{ entier} \\ & x_k \geq 0 & \forall k \in \mathcal{C} \\ & x_i \in \{0,1\} & \forall i \in \mathcal{B} \neq \emptyset, \end{array}$$

L'ensemble des indices des variables du problème $\mathcal{N}$ est partitionné en $(\mathcal{B}, \mathcal{G}, \mathcal{C})$. $\mathcal{B}$ est l'ensemble des variables binaires du problèm P et $\mathcal{G}$ et $\mathcal{C}$ sont respectivement l'ensemble des variables entières et l'ensemble des variables continues. On suppose que $\mathcal{B} \neq \emptyset$. Soit $\bar{x}$ une solution réalisable de (P), soit $\bar{\mathcal{S}} = \{j \in \mathcal{B} : \bar{x}_j = 1\}$ l'ensemble des variables de $\bar{x}$ dont la valeur est 1. Etant donné un entier positif k, on définit le $k-OPT$ voisinage $\mathcal{N}(\bar{x}, k)$ de $\bar{x}$ par l'ensemble des solutions réalisables de P, vérifiant la contrainte de branchement local suivante :

$$\Delta(x, \bar{x}) = \sum_{j \in \bar{\mathcal{S}}} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus \bar{\mathcal{S}}} x_j \leq k$$

où les deux termes de gauche, de cette inégalité, dénombrent, respectivement, les variables binaires de $\bar{x}$, dont la valeur passe de 1 à 0, ou de 0 à 1, respectivement. On peut remarquer que cette contrainte n'est autre que la distance de Hamming d'ordre k parmi les solutions voisines de $\bar{x}$. Dans le cas où toutes les solutions réalisables de P ont la même cardinalité, la contrainte s'écrit conventionnellement de la façon suivante :

$$\Delta(x, \bar{x}) = \sum_{j \in \bar{\mathcal{S}}} (1 - x_j) \leq k' (= k/2)$$

Comme son nom l'indique, la contrainte de branchement local peut être utilisée comme critère de branchement dans un algorithme d'énumération implicite pour le problème P. En effet, étant donnée une solution de référence $\bar{x}$ de P, l'éspace correspondant au noeud en cours de traitement peut être divisé suivant les contraintes disjointes suivantes :

$$\Delta(x, \bar{x}) \leq k \text{ (branche gauche) ou } \Delta(x, \bar{x}) \geq k + 1 \text{ (branche droite)}$$

Le paramètre k, qui n'est autre que la taille du voisinage, devrait être choisi de manière à ce que sa valeur soit la plus grande possible. Le choix du paramètre k doit produire un sous problème à la branche gauche, plus facile à résoudre que le sous problème de son noeud père. L'idée est que le voisinage $\mathcal{N}(\bar{x}, k)$, correspondant à la branche gauche, doit être suffisament petit pour être exploré en un temps de calcul relativement court. Mais aussi suffisament large pour contenir des solutions meilleures que $\bar{x}$. L'expérience a montré que les valeurs prises dans l'intervalle $[10, 20]$ donnaient des solutions de bonne qualité, poue des programmes linéaires en nombres mixtes. Une première implémentation de l'idée de branchement local est illustrée dans la figure 5.1 où les triangles marqués par la lettre "T" (pour tactique) correspondent aux sous-arbres de branchement. L'exploration de ces sous-arbres est effectuée avec un critère de branchement standard et tactique. Par exemple le branchement sur les variables fractionnaires. Les triangles "T" représentent l'application de la boite noire correspondant aux solveurs de programmes linéaires en nombres mixtes.

Pour le noeud racine (noeud 1), on suppose disposer d'une solution de reférence $\bar{x}^1$. La branche gauche (le noeud 2 de l'arbre) correspond à l'optimisation avec le $k-OPT$ voisinage $\mathcal{N}(\bar{x}^1, k)$. Ce calcul est effectué grâce à un schéma de branchement tactique convergeant vers uns solution optimale $\bar{x}^2$ (dans ce voisinage). La solution $\bar{x}^2$ devient alors la nouvelle solution de référence, le schéma est alors réappliqué à la branche droite (noeud 3), où l'exploration du sous-espace $\mathcal{N}(\bar{x}^2, k)$ produit une nouvelle solution de référence $\bar{x}^3$. Le noeud 5 est alors abordé. Celui-ci correspond au problème initial auquel sont ajoutées les deux contraintes suivantes : $\Delta(x, \bar{x}^1) \geq k + 1$ et $\Delta(x, \bar{x}^2) \geq k + 1$. Dans notre exemple, la branche gauche du noeud 6 produit un sous-problème qui ne contient pas de solutions meilleures que la solution de référence. Dans cette situation, l'ajout de la contrainte de branchement $\Delta(x, \bar{x}^3) \geq k + 1$ conduit à la branche droite du noeud 7. Ce noeud est exploré à l'aide d'un branchement tactique. Notons que la solution fractionnaire du noeud 1 n'est pas nécessairement coupée dans les noeuds fils 2 et 3, comme c'est souvent le cas quand on applique un branchement

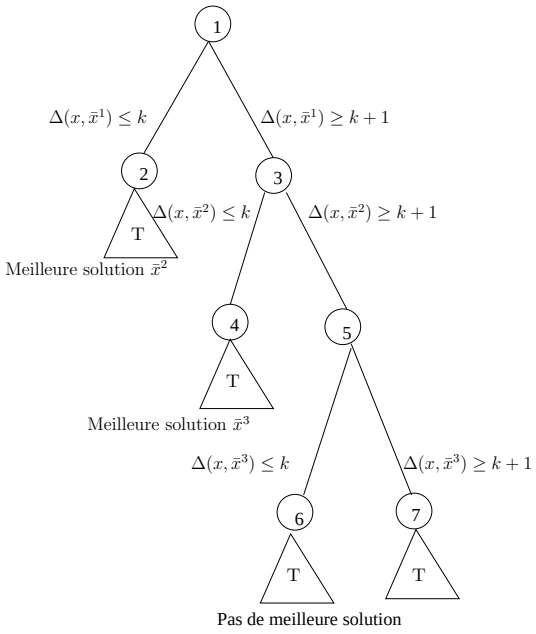

FIG. 5.1 – Le schéma de branchement local.

standard sur les variables. En effet, la philosophie du branchement local est différente de celle du branchement standard : l'idée est de ne pas forcer les variables fractionnaires à prendre des valeurs, mais d'essayer d'explorer quelques régions prometteuses de l'espace de recherche. Le but escompté du schéma de branchement local est une mise à jour précoce de la solution de référence. Autrement dit, il est souhaitable d'obtenir rapidement des solutions de qualité de plus en plus meilleure, jusqu'à ce que le branchement local ne peut plus être appliqué (noeud 7). Alors le branchement tactique est appliqué pour conclure l'énumération.

La première amélioration est relative au fait que, dans certains cas, la solution exacte de la branche gauche du noeud peut requérir un temps d'exécution important. Donc, il est raisonnable d'imposer une limite au temps d'exécution du calcul dans la branche gauche. Dans le cas où la limite est atteinte, deux situations sont à distinguer :

1. si la solution de référence a été améliorée, alors il faut effectuer un retour en arrière vers

le noeud père. Puis créer un noeud à la branche gauche associée à la nouvelle solution de référence, sans modifier la valeur de k. Cette situation est illustrée dans la figure 5.2, où le noeud 3 a trois fils : le noeud 4 (une meilleure solution $\bar{x}^3$ a été trouvée dans les temps), et les noeuds $4'$ et 5. Notons que dans l'exemple, le voisinage associé au noeud 4 n'a pas été complètement exploré, par conséquent, il est mathématiquement incorrect d'imposer aux noeuds $4'$ et 5 la condition $\Delta(x, \bar{x}^2) \geq k + 1$.

2. Si la limite au temps d'exécution est atteinte sans que la solution en cours ne soit améliorée, on réduit la taille du voisinage pour accélerer son exploration. Cette réduction pourrait être obtenue en réduisant par exemple la partie droite de $\lceil k/2 \rceil$. Cette situation est illustrée dans la figure 5.3 dans laquelle la limite au temps d'exécution est atteinte sans amélioration au noeud 4. Au noeud $4'$, la réduction du voisinage permet d'obtenir une solution optimale locale $\bar{x}^3$.

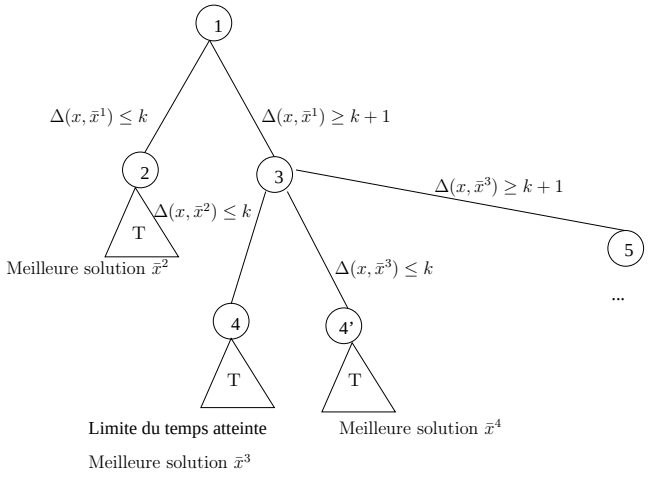

FIG. 5.2 – Le schéma de branchement local avec limite sur temps d'exécution.

5.2.3 Diversification

Une amélioration des performances de la méthode de branchement local peut être obtenue en utilisant le mécanisme de diversification issu du concept des métaheuristiques du type recherche locale. La diversification est appliquée dans le cas où il est prouvé que la branche gauche du noeud courant ne contient pas de meilleure solution que la solution en cours. Cette situation est illustrée au noeud 6 de la figure 5.1, où le schéma de branchement local change l'exploration, vers le noeud 7, à l'aide d'un branchement tactique.

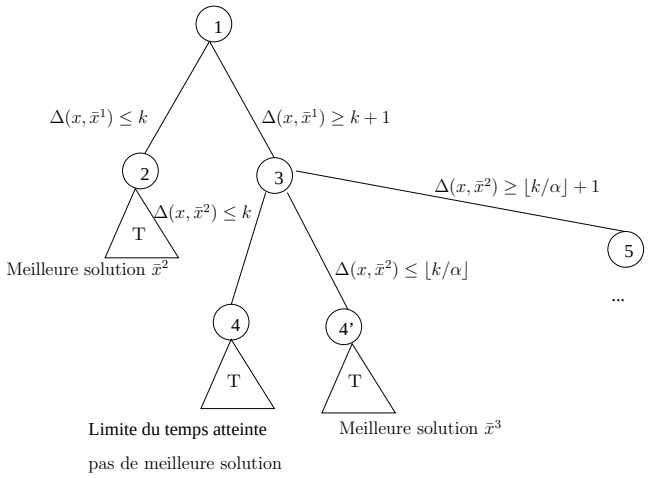

FIG. 5.3 – Le schéma de branchement local avec limite sur le temps d'exécution cas b.

Afin de garder un contrôle stratégique sur l'énumération, même dans une telle situation, on utilise deux mécanismes de diversification différents. Tout d'abord une diversification faible, qui consiste à élagir le voisinage courant en augmentant sa taille, par exemple de $\lceil k/2 \rceil$. La diversification va alors produire une branche gauche. Cette branche sera explorée à l'aide d'un branchement tactique (une limite au temps d'exécution est fixée). Dans le cas où on ne trouve pas de solution améliorée, même dans le voisinage élargi, on applique la phase de diversification forte. Dans cette phase, on cherche des solutions de plus mauvaise qualité que

la solution de référence, mais qui ne soient pas trop loin de la solution de référence actuelle $\bar{x}$. Ceci est réalisé en appliquant un branchement tactique au problème courant, auquel est ajoutée la contrainte $\Delta(x, \bar{x}^2) \leq k + 2\lceil k/2 \rceil$, mais sans imposer une borne supérieure à la valeur de la solution optimale. L'exploration est avortée dès qu'une solution réalisable est obtenue. Cette solution, qui est plus mauvaise que la solution en cours, est utilisée comme nouvelle solution de référence. Le processus est alors réappliqué dans le but d'améliorer la solution en cours ou la meilleure solution.

La figure 5.4 illustre le pseudo-code du schéma de branchement local.

La fonction **Locbra** prend en paramètre la taille du voisinage k, le limite du temps d'exécution total ($total\_time\_limit$), la limite du temps d'exploration d'un noeud ($node\_time\_limit$) et le nombre de diversifications autorisées (dv_max). En sortie, elle retourne la meilleure solution réalisable trouvée ($x^\star$), avec le statut final de l'optimisation (opt). Dans le cas où aucune solution réalisable n'est trouvée, mais sans garantie sur la non-réalisabilité, La fonction **Locbra** retourne $opt = faux$ et $x^\star = indefini$. La boucle principale répéter est itérée jusqu'à ce que la limite du temps total ou le nombre maximal de diversifications est atteint. À chaque itération, un problème est résolu par la boite noire **MIP_SOLVE** qui prend en paramètre : la limite du temps local TL, la borne supérieure UB utilisée pour interrompre le calcul dès que la borne inférieure devient plus grande ou égale à UB, et la variable $first$ qui prendra la valeur vrai dès que la première solution réalisable est trouvée (dans ce cas le calcul est arrêté). **MIP_SOLVE** retourne la meilleure solution $\bar{x}$. **LocBra** utilise une variable $diversify$ qui indique si la prochaine diversification sera faible ($diversify = faux$) ou forte ($diversify = vrai$). La diversification forte est réalisée uniquement si la diversification faible est réalisée à l'itération précédente, c'est-à-dire que toute itération qui ne requière pas de diversification met la variable $diversify$ à faux. Quatre cas différents peuvent apparaître après chaque appel de la procédure **MIP_SOLVE**.

1. **opt_sol_found** : l'optimalité du problème en cours de résolution a été prouvée. Par conséquent, la dernière contrainte de branchement local est reconstituée. La solution de référence est mise-à-jour, et le processus est itéré. Un retour immédiat est effectué dans le cas où cette situation se produit au noeud racine ($rhs \geq +\infty$), où aucune contrainte n'est imposée.

2. **proven_infeasible** : s'il a éte prouvé que le problème n'a pas de solution réalisable

```
fonction LocBra(k, total_time_limit, node_time_limit, dv_max, x*)
  rhs := bestUB := UB := TL : +∞ ; x* := undefined ;
  opt := first := vrai ;dv := 0 ; diversify := faux
  répéter
    si (rhs < +∞) alors
      ajouter la contrainte de branchement local Δ(xx̄) ≤ rhs finsi ;
    TL := min{TL, total_time_limitelapsed_time}
    stat := MIP_SOLVE(TL, UB, frist, x̂) ; TL := node_time_limit;
    si (stat = opt_sol_found) alors
      si (c^T x̂ < bestUB) alors bestUB := c^T x̂ ;x* := x̂ finsi ;
      si (rhs ≥ +∞) retourner(opt) ;
      changer la dernière contrainte de branchement local vers Δ(x, x̄) ≥ rhs + 1;
      diversify := frist := faux ; x = x̄ ; UB := c^T x̄ ; rhs := k
    finsi ;
    si (stat = feasible_sol_found) alors
      si (rhs < ∞) retourner(opt) ;
      changer la dernière contrainte de branchement local vers Δ(x, x̄) ≥ rhs + 1 :
      si (diversify) alors UB := TL := ∞ : dv := dv + 1; frist := vrai ;
      finsi ;
      rhs := rhs + ⌈k/2⌉ ; diversify := vrai
    finsi ;
    si (stat = feasible_sol_found) alors
      si (rhs < ∞) alors
        si(first) alors
          supprimer la dernière contrainte de branchement local Δ(x, x̄) ≤ rhs
        sinon
          remplacer la dernière contrainte de branchement local Δ(x, x̄) ≤ rhs par Δ(x, x̄) ≥ 1)
        finsi
      finsi
      REFINE(x̂) ;
      si(c^T x̂ < bestUB) alors bestUB := c^T x̂ ; x* := x̂ finsi
      first := diversify := faux ; x̄ = x̂; UB := c^T x̂ ; rhs := k
    finsi ;
    si( stat = no_feasible_sol_found) alors
      si(diversify) alors
        remplacer la dernière contrainte de branchement local Δ(x, x̄) ≤ rhs par Δ(x, x̄) ≥ 1 ;
        UB := TL := +∞; dv := dv + 1; rhs := rhs + ⌈K/2⌉; frist := vrai
      sinon
        supprimer la dernière contrainte de branchement local Δ(x, x̄) ≤ rhs;
        rhs := rhs − ⌈k/2⌉
      finsi
      diversify := vrai
    finsi
  jusqu'à (elapsed_time > total_time_limit) ou (dv > dv_max) ;
  TL := total_time_limitelapsed_time; first := faux ;
  stat := MIP_SOLVE(TL, bestUB, first, x*) ;
  opt := (stat = opt_sol_found ) ou stat = proven_infeasible) ;
  retourner ( opt)
fin.
```

FIG. 5.4 – La fonction *Locbra*

dont la valeur est strictement inférieure à la borne supérieure de départ UB, alors la dernière contrainte de branchement local est changée. Une diversification faible ou forte est effectuée en fonction de la valeur actuelle de la variable $diversify$, et le processus est réitéré. Un retour immédiat est effectué si cette situation se produit au noeud racine ($rhs \geq +\infty$), où aucune contrainte ni limite sur le temps d'exécution local n'est

imposée.

3. **feasible_sol_found** : une solution de valeur strictement inférieure à la borne supérieure UB (c'est-à-dire une solution qui améliore la solution de référence) a été trouvée. Mais le solveur n'a pas été capable de prouver son optimalité pour le problème actuel (en raison du délai imposé ou de l'obligation d'annuler l'exécution après que la première solution a été trouvée). Comme il a été déjà mentionné, dans ce cas, il n'est pas permis d'inverser la dernière contrainte de branchement local. Afin de couper la solution de référence $\bar{x}$, on remplace la dernière contrainte de branchement local $\Delta(x, \bar{x}) \leq rhs$ par la contrainte "tabu" $\Delta(x, \bar{x}) \geq 1$ (à moins que cette contrainte a déjà été introduite à l'étape 4, auquel cas la dernière contrainte de branchement locale est supprimée). De cette façon, on peut aussi couper la solution optimale du problème, car en principe nous n'avons aucune garantie que $\bar{x}$ est optimale sous la condition $\Delta(x, \bar{x}) < 0$, c'est-à-dire lors de la fixation $x_j = \bar{x_j}, \forall j \in \mathcal{B}$. Une telle garantie existe toujours, si par exemple toutes les variables du problème sont des variables binaires ($\mathcal{B} = \mathcal{C} = \emptyset$). Dans le cas général, la justesse de la méthode nécessite l'utilisation d'un "raffinage" procédure **REFINE**, qui remplace la solution de départ par la solution optimale (calculée par l'intermédiaire du solveur) dans le voisinage $\Delta(x, x) \leq 0$. Notez que la procédure **REFINE** est implicitement appliquée à toutes les solutions fournies par **MI_SOLVE**, même si elle est seulement invoquée à l'étape 3 (cela est manifestement inutile à l'étape 1). On notera aussi que le temps d'exécution de la procédure **REFINE** peut s'avèrer trop grand dans le cas où la fixation des variables binaires ne conduise pas à un sous-problème facile à résoudre. Et ceci en raison de la présence d'un grand nombre de variables générales. Dans cette situation, on est autorisé à désactiver la procédure **REFINE**. Mais la dernière contrainte de branchement local $\Delta(x, \bar{}) \leq rhs$ ne peut être remplacée par la contrainte $\Delta(x, \bar{x}) \geq 1$. Puisque ceci affecte l'optimalité de la méthode.

4. **no_feasible_sol_found** : Aucune solution de valeur strictement inférieure à UB n'a été trouvée, alors que la limite au temps d'exécution dans la branche a été atteinte. Mais nous n'avons pa de garantie que cette solution n'existe pas. Dans ce cas, la dernière contrainte de branchement est supprimée ou remplacée par la contrainte "tabu" $\Delta(x, \bar{x}) \geq 1$. Ceci dépend du type de diversifications à réaliser. En cas de faible diversification, un nouveau voisinage de branchement local est exploré (et la contrainte de branchement local précédente est supprimée). Alors que pour la diversification forte, la contrainte "tabu" $\Delta(x, \bar{x}) \geq 1$ est introduite afin d'échapper à la solution en cours,

et la borne supérieure UB est remise à $+\infty$. À la sortie de la boucle répéter, si la limite au temps d'exécution n'a pas été atteinte, on essaye de résoudre de manière exacte le problème en cours (et si possible prouver l'optimalité). Cette situation est illustrée au noeud 7 de la figure 1. Le branchement local se comporte comme un algorithme exact dans le cas où $total\_time\_limit = +\infty$ et $dv\_max \leq +\infty$. Alors qu'il se comporte comme un algorithme approché si $dv\_max = +\infty$.

5.3 Branchement local et le DCKP

La programmation linéaire en nombres entiers joue un rôle important dans la modélisation des problèmes d'optimisation combinatoire NP-difficiles. On peut modéliser des problèmes de grande taille de la vie de tous les jours. Cependant, ces modèles restent difficiles à résoudre à l'optimum. Par conséquent, le développement d'heuristiques efficaces capables de résoudre des problèmes de grande taille est d'une importance primordiale. Dans cette partie, nous traitons le problème du sac-à-dos à *contraintes disjonctives* (DCKP) où des objets sont incompatibles avec d'autres. Pour plus de clarté nous rappelons la déscription du DCKP. Nous disposons d'un sac-à-dos de capacité c fixée, et d'un ensemble de n objets. Chaque objet $j, j = 1, \ldots, n$ a une taille ou *poids* w_j et un *profit* p_j. L'objectif est de savoir si l'objet j peut être ou non mis dans le sac. Pour qu'un objet soit mis dans le sac, il faut qu'il respecte la contrainte de capacité, et les contraintes qui réprésentent l'incompatibilité entre lui et d'autres objets. Soit $E \subseteq \Big\{(i,j) \text{ tel que } i \neq j,\ i,j \in I = \{1,\ldots,n\}\Big\}$ l'ensemble des couples d'objets incompatibles. Si le couple $(i,j) \in E$ alors les objets i et j ne peuvent être mis simultanément dans le sac. Et une *contrainte disjonctive* est imposée pour ce couple. Cette contrainte s'écrit de la manière suivante :

$$x_i + x_j \leq 1, \tag{5.1}$$

où x_i et x_j dénotent respectivement les variables de décision associées aux objets i et j respectivement. Combinant la contrainte de capacité et des contraintes du même type que (5.1), le DCKP que nous noterons P_1 s'écrit de la manière suivante.

$$\max \qquad z(x) = \sum_{j=1}^{n} p_j x_j \tag{5.2}$$

$$\text{s.c} \qquad \sum_{j=1}^{n} w_j x_j \leq c \tag{5.3}$$

$$x_i + x_j \leq 1 \qquad \forall (i,j) \in E \tag{5.4}$$

$$x_j \in \{0,1\} \qquad j = 1,\ldots,n, \tag{5.5}$$

où $z(x)$ (equation 5.2) dénote la valeur de la fonction objectif associée à la solution x. Pour le reste du chapitre et sans perte de généralité nous supposons que :

1. Toutes les données $c,\ p_j,\ w_j$, pour $j = 1,\ldots,n$, sont des entiers positifs.
2. Les objets sont ordonnés selon l'ordre décroissant du rapport profit sur poids ; qui est

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

3. Dans le but d'éviter les solutions triviales, nous supposons que $\sum_{j=1}^{n} w_j > c$.

Le reste de ce chapitre est organisé comme suit. La section 5.3.1 expose brièvement les travaux qui existent sur le problème du DCKP. Dans la section 4.2.1, nous rappelons le concept du modèle équivalent du DCKP ; ce modèle est basé sur des contraintes de type surrogate. Dans la section 5.3.3, nous récapitulons le concept de branchement local. La section 5.3.4 expose l'algorithme hybride qui combine la procédure d'arrondi 5.3.4.1 et le branchement local. Dans la section 5.4, nous évaluons l'exécution de l'algorithme proposé sur plusieurs instances de la littérature et nous présentons nos observations sur les résultats obtenus.

5.3.1 Littérature du DCKP

Le DCKP -*un programme linéaire en nombres entiers*- est un problème d'optimisation combinatoire NP-difficile. Quand $E = \emptyset$ (la constrainte 5.4), le problème se réduit au problème du sac-à-dos unidimensionnel. Quand la contrainte de capacité (5.3) est supprimée, le problème devient le problème du stable maximal (Garey et Johnson [25]). Dans certains cas , le DCKP peut être un composant d'un problème d'optimisation combinatoire plus difficile. Sa structure induite dans des problèmes plus complexes permet le calcul de bornes supérieures, et la conception de méthodes exactes et approchées pour ces problèmes. Par exemple, dans les travaux de Pisinger et Sigurd [77], les auteurs ont proposé une formulation basée sur la décomposition de Dantzig-Wolfe pour le problème du bin paking bidimensionnel. Ils ont employé un problème du sac-à-dos disjonctif généralisé. Ce problème est spécifié comme un problème du sac-à-dos multi-contraint. À notre connaissance, très peu de travaux de recherche ont porté sur le DCKP. Yamada et al [89, 90] ont abordé le problème en proposant deux algorithmes, un approximatif et l'autre exact. L' algorithme approximatif commence par générer une première solution réalisable de départ. Ensuite, il améliore cette solution à l'aide d'un mécanisme de 2-opt dans son voisinage. L'algorithme exact commence par générer une solution de départ à l'aide de l'heuristique décrite précédemment. Ensuite, il entreprend une énumération implicite couplée avec une technique de réduction d'intervalle. L'algorithme exact résout des instances non-corrélées dont le nombre d'objets varie entre 100 et 1000, avec une densité variant de 0.001 à 0.020 (ce qui équivaut à un maximum de 10000 contraintes). Hifi et Michrafy [43] ont proposé une méthode basée sur une recherche locale réactive constituée de trois étapes. Cette méthodes résout des instances de grande taille. Nous avons largement évoqué cette méthode au chapitre précédent. Finalement, Hifi et Michrafy [46] ont

proposé trois algorithmes exacts. Dans ces algorithmes, des stratégies de réduction, un modèle équivalent et une recherche dichotomique coopèrent pour résoudre les instances du DCKP. La première version de l'algorithme réduit la taille du problème original en commençant par une borne inférieure et en résolvant successivement une série de relaxation du DCKP. La seconde combine une stratégie de réduction avec une recherche dichotomique afin d'accélérer le processus de recherche. Elle applique la stratégie de réduction au premier niveau de l'arbre développé et/ou après avoir généré quelques noeuds. La stratégie fixe quelques variables de décision à l'optimum, permettant ainsi la réduction de la taille du problème original ; et ainsi, facilitant dans beaucoup de cas la résolution du problème. Deux stratégies sont considérées. La première fixe la variable de décision courante à sa valeur optimale ; c'est-à-dire à 0 ou à 1. Elle résout alors les sous-problèmes résultants. La seconde stratégie fixe la variable de décision courante à zéro et à un, et résout les deux sous-problèmes résultants. La troisième et dernière version de l'algorithme propose une version modifiée de l'algorithme dans laquelle deux modèles équivalents coopèrent pour résoudre des instances du problème avec un nombre important de contraintes disjonctives.

5.3.2 Modèle équivalent pour le DCKP

Pour plus de précisions, nous rappelons la déscription du modèle équivalent du DCKP (4.2.1). Dans Hifi and Michrafy [46], un modèle équivalent qui tire partie de la particularité des contraintes disjonctives du DCKP a été proposé. L'objectif pour ce modèle est de réduire le nombre de contraintes de P_1, dans le but d'essayer de faciliter la résolution de certaines instances de taille modérée. En effet, afin de résoudre le DCKP, P_1 est transformé en un problème équivalent dénoté par P_2. Il se compose de contraintes exprimant le voisinage de chaque variable de décision de P_1. La description du problème équivalent P_2 est la suivante :

soit $E_i \subseteq V_i = \{j \in I, (i,j) \in E\}$ tel que $E_i = \{j \in I, (i,j) \in E,\ i < j\}$, et considérons le programme linéaire en nombre entier suivant :

$$(P_2) \begin{cases} \max & \displaystyle\sum_{j=1}^{n} p_j x_j \\ \text{s.c} & \displaystyle\sum_{j=1}^{n} w_j x_j \leq c \\ & |E_i| x_i + \displaystyle\sum_{j \in E_i} x_j \leq |E_i|,\ i = 1, \ldots, n \\ & x_i \in \{0,1\},\ i = 1, \ldots, n, \end{cases}$$

où $|E_i| \geq 1$, pour $i \in I$. Remarquons que P_2 peut êre vu comme un problème avec des contraintes de type surrogate. Chaque coefficient $|E_i|$ associé à une variable de décision peut être facilement optimisé par des procédures exactes ou approchées. En effet, optimiser le i-ème coefficient est équivalent à résoudre le programme linéaire suivant :

$$(Q_i) \begin{cases} \max & \alpha_i = \sum_{i \in E_i} x_i \\ \text{s.t.} & x_i + x_j \leq 1, \ \forall (i,j) \in E \\ & x_i \in \{0,1\}, \ i \in E_i. \end{cases}$$

Le problème Q_i est le problème du plus grand stable qui est aussi NP-difficile. Afin de renforcer les contraintes de chaque Q_i, on peut ajouter la contrainte suivante de sac-à-dos :

$$\sum_{j \in E_i} w_j x_j \leq c.$$

Afin de calculer une valeur approchée de α_i, on applique la procédure de Johnson(section 4.2.2) à Q_i, qui peut être résumée comme suit :

1. Trier dans l'ordre décroissant les indices des objets du problème initial Q_i selon leurs degrés $\delta_j, \ j \in E_i$. Le degré δ_j associé à $j \in E_i$ dénote la taille de son voisinage ou le nombre de ses voisins.
   Posons $E'_i = E_i$.
2. Fixer à 1 la variable de décision x_k tel que
$$k = \underset{j \in E'_i}{\operatorname{argmin}} \{\delta_k \mid \delta_k \geq \delta_j\}$$
3. Enlever k et ses voisins de E'_i.
4. S'arrêter si $E'_i = \emptyset$, répéter l'étape 2 sinon.

5.3.3 Branchement local standard

les principales étapes de l'algorithme de branchement local peuvent être décrites de la manière suivante :

1. Générer une solution initiale pour la première arborescence (noeud).
2. Initialiser la première arborescence en utilisant la solution générée précédemment.
3. Phase itérative

(a) Résoudre de manière complète l'arborescence locale.

(b) lorsque la recherche locale se termine nous distingons deux cas :
- Une meilleure solution réalisable a été obtenue dans cette arborescence locale. Alors créer une nouvelle arborescence en utilisant la solution améliorée comme solution initiale (solution de référence) ;
- La solution n'a pas été améliorée, alors interrompre le branchement local.

4. Résoudre le reste de l'arbre de recherche.
5. Retourner la meilleure solution trouvée.

Une version de la méthode de branchement local a été proposée par [18] comme méthode exacte et plus tard par [19] comme heuristique à deux niveaux. L'approche standard est basée sur l'utilisation de conditions de branchement exprimées à travers des inégalités linéaires. L'approche de branchement local est appliquée au programme linéaire mixte en nombre entiers (PP), dont la forme générale est la suivante :

$$\begin{array}{lll} \text{Minimiser} & c^T x & \\ & Ax \geq b & \\ & x_j \geq 0 & \forall j \in G,\ x_j \text{ entier} \\ & x_k \geq 0 & \forall k \in C \\ & x_i \in \{0,1\} & \forall i \in B \neq \emptyset, \end{array}$$

où $N := \{1, ..., n\}$ l'ensemble des indices des objets est divisé en trois sous-ensembles (β, G, C) avec $\beta \neq \emptyset$ représntant les variables binaires, et G et C, représentent les indices des variables entières et ceux des variables continues, respectivement.
Soient $\bar{x}$, une solution réalisable initiale que nous appelons *solution de référence* du problème PP, et k, un entier positif. Le k_{OPT} *voisinage* de $\bar{x}$ est l'ensemble des solutions réalisables de PP, satisfaisant la contrainte dite contrainte de branchement local suivante :

$$\Delta(x, \bar{x}) \quad := \quad \sum_{j \in \beta \ \mid \ \bar{x}=1} (1 - x_j) + \sum_{j \in \beta \ \mid \ \bar{x}=0} x_j \quad \leq \quad k,$$

où les deux termes gauches de l'inégalité représentent le nombre de variables binaires de la solution $\bar{x}$ dont la valeur passe de 1 à 0 ou de 0 à 1, respectivement. Notons que cette contrainte représente aussi la distance de Hamming maximale d'ordre k, entre x et $\bar{x}$. Si la cardinalité de β est fixée, l'inégalité représentant la distance de Hamming devient :

$$\Delta(x, \bar{x}) = \sum_{j \in \beta \ \mid \ \bar{x}=1} (1 - x_j) \leq k' \ \Big(= \frac{k}{2} \Big), \tag{5.6}$$

Comme on a pu le voir dans la première partie de ce chapitre, la contrainte de branchement local est utilisée par [18] comme critère de branchement dans un schéma énumératif pour les programmes linéaires mixtes. En effet, étant donnée une solution de référence $\bar{x}$, le sous-espace qui correspond au noeud en cours de traitement peut être divisé suivant les deux contraintes disjointes suivantes :

$$\Delta(x, \bar{x}) \leq k \qquad \text{ou} \qquad \Delta(x, \bar{x}) \geq k + 1,$$

où le paramètre k repréntant la taille du voisinage est choisi de manière adéquate.

L'approche continue en alternant entre les phases de haut-niveau (où les inégalités de branchement local sont utilisées pour définir des régions *prometteuses*), et les phases de bas-niveau (utilisée comme critère de branchement dans un schéma énumératif) l'approche immite le comportement des solveurs pour résoudre les voisinages induits par les branches.

Comme dans [18], le processus de recherche continue en alternant les branches *normales* et les branches *locales*. La branche locale est résolue complètement avant de résoudre la branche normale. Lorsqu'une nouvelle solution optimale $\bar{x}^2$ est trouvée dans la branche locale, le branchement local peut continuer avec une nouvelle solution, en ajoutant la nouvelle solution à la branche normale qui reste. Lorsque la branche gauche est complètement résolue, la solution obtenue $\bar{x^1}$ devient la nouvelle solution de référence. Par la suite, l'espace de recherche correspondant à la branche normale est séparé, et les deux nouvelles branches suivantes sont obtenues :

$$\Delta(x, \bar{x}) > k,\ \Delta(x, \bar{x}^2) \leq k \quad \text{(branche locale)}, \tag{5.7}$$

and

$$\Delta(x, \bar{x}) > k,\ \Delta(x, \bar{x}^2) \geq k + 1 \quad \text{(branche normale)}. \tag{5.8}$$

Le schéma de branchement local fonctionne tant que les branches locales conduisent à de nouvelles solutions optimales globales. La taille du sous-arbre correspondant à la branche locale dépend de la valeur du paramètre k. En effet, d'un côté, une petite valeur de k définit un voisinage de taille modérée, généralement facile à calculer. Mais, elle peut ne pas contenir de solution meilleure que la solution de référence. D'un autre côté, une grande valeur de k offre un grand degré de liberté durant la recherche. Mais, augmente considérablement la taille

du voisinage.

Comme on'a pu le voir dans la première partie de ce chapitre, d'autres extensions de l'algorithme de branchement local ont été proposées par [18]. Par exemple, limiter le temps de résolution des branches locales, ou utiliser des techniques de diversification lorsque celles-ci n'améliorent pas les meilleures solutions, etc.

5.3.4 Algorithme de branchement local pour le DCKP : un algorithme hybride

Nous proposons un algorithme hybride dans lequel le branchement local utilisant une procédure d'énumération complète est remplacé par :

(i) une procédure d'arrondi pour résoudre la relaxation en continue de P_2, et pour réduire l'espace de recherche (ii) un algorithme exact, pour résoudre le reste du problème. Les deux procédures sont combinées pour accélérer le processus de recherche et pour améliorer la qualité des solutions obtenues. Puisque le mécanisme de branchement local exploite les contraintes de branchement pour créer des stratégies de diversification et d'intensification, nous l'utilisons pour essayer d'améliorer les solutions fournies par la procédure d'arrondi.

5.3.4.1 Solution de départ pour l'algorithme de branchement local

Pour plus de précision, nous rappelons ici les principales étapes de la procédure d'arrondi que nous avons déjà évoquée au chapitre précédent. La *procédure d'arrondi* (BRSP) fonctionne généralement avec la relaxation en continue du problème linéaire en nombres entiers. En effet, une manière de chercher une approximation de la solution entière du problème P_2 pourrait être d'arrondir sa relaxation en continue notée LP_2, la relaxation LP_2 est définie par :

$$(LP_2)\begin{cases} \max & \displaystyle\sum_{j=1}^{n} p_j x_j \\ \text{s.t.} & \displaystyle\sum_{j=1}^{n} w_j x_j \leq c \\ & \alpha_i x_i + \displaystyle\sum_{j \in E_i} \leq \alpha_i,\ i = 1, \ldots, n \\ & x_i \in [0,1],\ i = 1, \ldots, n. \end{cases}$$

La procédure d'arrondi peut être considérée comme une approche à deux phases :

1. Nous optimisons la relaxation LP_2, en utilisant la méthode du simplexe. Ensuite, nous collectons la solution primale, et nous fixons à leurs valeurs les variables x_j qui ont la valeur 1. Toutes les variables x_i liées par une contrainte disjonctive à l'une des variables x_j sont fixées à 0. Enfin, nous sélectionnons la variable fractionnaire x_j pour l'aronndir à sa valeur supérieure (1). Cette variable est choisie de façon à ce qu'elle soit la variable fractionnaire qui a la plus grande valeur.
2. Le problème réduit resté après la fixation de quelques variables (à 1 et à 0) (étape 1) est soumis à un traitement équivalent. En effet, LP_2 est résolu par la méthode du simplexe, et la plus grande variable fractionnaire x_j dans la solution de LP_2 est arrondie. Ce processus est appliqué jusqu'à ce qu'il ne reste plus de variables fractionnaires.

On peut noter qu'une telle approche peut fournir des solutions modérées pour le DCKP, à cause de son caractère glouton. En fait, ce phénomène peut être expliqué par la particularité du DCKP. Puisque la fixation d'une variable de décision, à 1, implique la fixation d'un sous-ensemble de variables à zéro (des variables correspondant à un sous-ensemble de contraintes disjonctives ; c'est le problème du stable maximal). Ainsi, nous pensons que dans certains cas le procédure d'arrondi peut éliminer quelques objets (fixant leurs variables de décision à zéro) d'une manière très agressive.

La procédure à deux phases (décrite dans la figure 5.5) peut être vue comme une procédure qui combine la procédure d'arrondi avec une procédure qui résout le problème réduit de manière exacte. Dans la première phase, la procédure d'arrondi fixe un sous-ensemble d'objets à leurs valeurs. Alors que dans la seconde phase, une méthode exacte résout le problème réduit. L'objectif de la procédure SP est de construire une solution proche de l'optimum, en

appliquant une hybridation entre des variables "LP-favorisées" et d'autres variables. En effet, la première phase SP (voir la figure 5.5) sert à décomposer l'ensemble des variables en deux sous-ensembles disjoints S_1 et S_2. Chaque élément appartenant à S_1 est fixé à 1. S_2 contient les variables libres, parmi ces variables $\alpha\%$ seront fixées à 1, à l'aide de la procédure d'arrondi. La deuxième phase de SP va collecter le reste des variables caractérisant le problème réduit, qui sera résolu de manière exacte.

5.3.4.2 Algorithme de branchement local hybride

Dans cette section, nous décrivons le principe de l'algorithme de branchement local hybride utilisé pour résoudre le DCKP. Pour plus de clarté, nous décrivons son idée principale

FIG. 5.5 – La procédure à deux phases : SP.

La première Phase.

Soient S_1 et S_2 deux sous-ensembles tels que $|S_1 \cup S_2| = n$ et $S_1 \cap S_2 = \emptyset$, où S_1 dénote le sous-ensemble contenant les variables déjà fixées à 1 et S_2 est le sous-ensemble des variables non-fixées (variables libres). alors les étapes suivantes sont appliquées :

i. Appliquer la procédure d'arrondi sur les variables fractionnaires (de S_2) pour fixer $\alpha\%$ des variables fractionnaires de la solution de LP_2, où $\alpha \in]0,\ 100]$.

La deuxième Phase.

Appliquer un algorithme exact sur le problème réduit, et sortir avec une solution améliorée.

sur un exemple générique, comme dans [18]. En effet, la figure 5.3.3 récapitule le mécanisme appliqué par le branchement local. Supposons que nous disposons d'une solution réalisable de départ $\bar{x}$, qui est également associée au noeud de la racine (noeud 1 de la figure 5.6). C'est une solution réalisable obtenue en appliquant la procédure d'arrondi (décrite dans 4.2.4).

Chaque noeud crée deux successeurs : un noeud représentant la branche gauche et un deuxième noeud correspondant à la branche de droite. La procédure SP est appliquée au noeud de la branche gauche, auquel nous avons ajouté la contrainte suivante :

$$\Delta(x, \bar{x}) \leq k,$$

On distingue alors les deux cas suivants :

(1) Une solution améliorée $\overline{x}_1$ émerge de la branche gauche.

(2) SP n'est pas capable d'améliorer la solution $\overline{x}$ à ce niveau.

Supposons que nous sommes dans la première situation (cas (1)). Ce qui veut dire que nous avons obtenu une solution améliorée $\overline{x}_1$ dans le k_{opt} voisinage $N(\bar{x}, k)$ associé à $\bar{x}$. Par conséquent, la nouvelle solution $\overline{x}_1$ devient la nouvelle solution de référence, et ainsi le

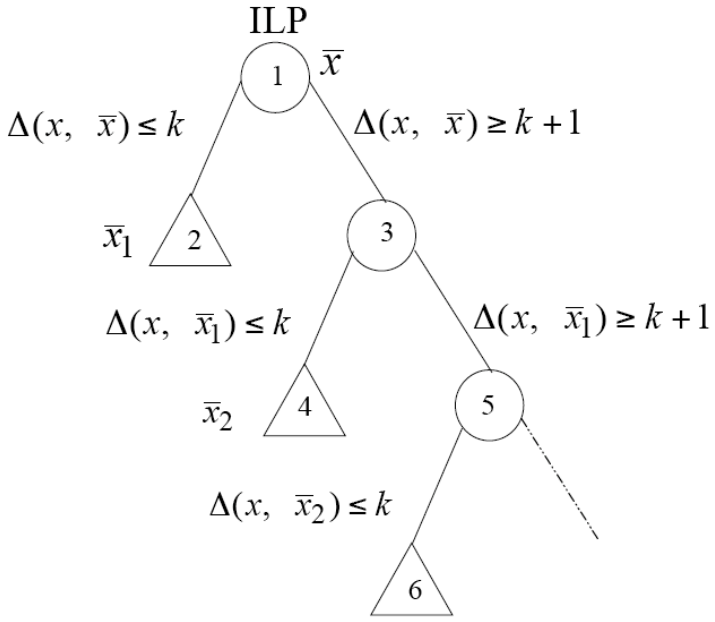

FIG. 5.6 – Heuristique basée sur le schéma de branchement local.

même processus est appliqué à la branche de droite (noeud 3, figure 5.6). Si nous supposons que SP atteint une nouvelle solution que nous appelons $\overline{x}_2$, lors de l'exploration du voisinage $N(\overline{x}_1, k) \setminus N(\overline{x}, k)$ (noeud 4, figure 5.6), alors la nouvelle branche de gauche associée au problème original LP est résolue en utilisant la procédure SP. Tandis que les contraintes de branchement local suivantes sont ajoutées :

$$\Delta(x, \overline{x}) \geq k+1 \quad \text{et} \quad \Delta(x, \overline{x}_1) \geq k+1.$$

Revenons maintenant au cas (2), c'est-à-dire SP n'est pas capable d'améliorer la solution de référence. En effet, supposons que le cas (2) se passe lors du dernier noeud, suivant la description ci-dessus (c'est-à-dire qu'au noeud 6 de la figure 5.6 la solution en cours $\overline{x}_2$ n'a pas été améliorée). On procède alors en fonction des résultats fournis par le processus de recherche. On distingue ce qui suit :

1. Aucune amélioration ne se produit : nous essayons alors d'enrichir l'espace de recherche en appliquant une stratégie d'intensification. Dans ce cas-ci, la stratégie appliquée consiste à réitérer la première phase de la procédure SP en fixant un sous-ensemble différent de variables à 1.

2. Le processus n'est pas capable de détecter une solution réalisable : nous essayons alors une stratégie de diversification afin d'éxplorer d'autres points de l'espace de recherche.

Avant de décrire les stratégies d'intensification et de diversification, nous récapitulons les étapes principales de l'algorithme de branchement local hybide appliqué 'a P_2 (noté HLB_{SP}) et décrit dans la figure 5.7.

FIG. 5.7 – Les principales étapes de HLB_{SP}.

Input. Une instance du DCKP.
Output. $\overline{x}$ une solution approchée du DCKP.

(I) **Phase d'initialisation.**
Générer une solution de départ pour le premier arbre (noeud) et initialiser la première arborescence en utilisant la solution générée précédemment.

(II) **Phase itérative.**
Répéter

(a) Ajouter la branche(es) au sous-arbre correspondant à P_2.

(b) Appeler SP pour résoudre le problème fourni.

(c) *Le branchement local termine :*

i. *Une meilleure solution a été trouvée* : créer une nouvelle arborescence en utilisant la meilleure solution trouvée dans l'arborescence en cours.

ii. *Aucune meilleure solution n'a été trouvée* : appliquer les phases d'intensification et de diversification.

Jusqu'à ce que la *condition d'arrêt* soit satisfaite.

(III) **Phase finale.**
Retourner la meilleure solution trouvée.

Dans ce qui suit, nous décrivons les mécanismes d'intensification (Section 5.3.4.3) et diversification (Section 5.3.4.4) utilisés par HLB_{SP}.

Soit $\bar{S}$ l'ensemble des variables fixées à 1 après l'application des deux phases de la procédure SP. Considérons les sous-ensembles $\overline{B}_1$ et $\overline{B}_2$, tel que $\overline{S} = \overline{B}_1 \cup \overline{B}_2$, $\overline{B}_1 \cap \overline{B}_1 = \emptyset$,

où $\bar{B}_1$ contient les variables fixées à 1 après la première phase de la procédure SP alors que $\overline{B}_2$ contient les variables fixées à 1 lors de la deuxième phase de la procédure SP.

5.3.4.3 Phase d'intensification

Soit $\bar{x}$ la solution réalisable en cours, on essaye d'explorer, efficacement, son voisinage obtenu en fixant quelques objets de $\overline{B}_1$ à 1. Ce qui veut dire que la majorité des objets fixés à 1 lors de la procédure d'arrondi reste fixer à 1. On introduit alors une stratégie d'intensification, qui peut être utilisée selon deux phases complémentaires.

1. La première phase injecte la contrainte suivante au problème :

$$\Delta_1(x, \bar{x}) = \sum_{j \in \overline{B}_1} (1 - x_j) \leq k_1,$$

   et résout le problème fourni en utilisant la procédure SP. De plus, la solution finale impose deux cas différents. D'une part, la nouvelle solution fournie représente une solution améliorée ; par conséquent, la meilleure solution $\bar{x}$ est mise à jour et la contrainte Δ_1 est enlevée du problème. D'autre part, le processus de recherche ne peut pas trouver une solution réalisable (ou le problème devient non réalisable en ajoutant la branche locale Δ_1) ; donc, la deuxième phase décrite ci-dessous est employée après élimination de la contrainte Δ_1 du problème dans le cas où ce dernier devient non réalisable.

2. Dans la deuxième phase, nous essayons de perturber les éléments déjà fixés à 1 par la procédure SP. D'abord, nous rappelons que la branche locale Δ_1 représente une contrainte du modèle courant. Ensuite, nous maintenons toutes les variables de décision appartenant à $\overline{B}_1$ à 1. Enfin, nous limitons le nombre de variables de décision déjà fixées à 1 par la seconde phase de la procédure SP (appartenant à $\overline{B}_2$).

   La fixation précédente applique le schéma suivant :

   (a) Soit k_2 (initialisé à 0) le nombre de variables forcées à la valeur un dans $\overline{B}_2$.
   Soit k_{step} le paramètre utilisé pour élargir l'espace de recherche et k_{max} la taille limite du voisinage (valeur maximale de k_2.)

   (b) Répéter les étapes :

      i. Injecter la contrainte locale suivante :

$$\Omega := \sum_{j \in \overline{B}_2} x_j = k_2$$

et résoudre le problème résultant, par la méthode SP.

ii. Soit $\hat{x}$ la solution fournie par SP, alors
- $\hat{x}$ est une amélioration de $\bar{x}$: Dans ce cas, il faut mettre à jour la solution de référence en remplaçant $\bar{x}$ par $\hat{x}$ et réinitialiser le paramètre k_2 à 0.
- $\overline{x} = \hat{x}$: nous supprimons alors la contrainte Ω et élargissons la taille du voisinage en posant $k_2 = k_2 + k_{step}$.

Jusqu'à $(k_2 = k_{max})$.

(c) Sortir avec la meilleure solution $\overline{x}$.

Les constraintes Δ_1 et Ω seront supprimées du modèle courant, après la fin de la phase d'intensification.

5.3.4.4 Phase de diversification

On applique la stratégie de diversifcation lorsque le processus de recherche (dans la branche gauche) n'est pas capable d'améliorer la solution de référence. En effet, la diversifcation dans ce cas devient nécessaire pour explorer d'autres espaces de recherche ou simplement compléter l'espace de recherche en cours. Ici, la stratégie utilisée consiste à élargir la taille du voisinage, en augmentant le nombre de variables de décision pouvant être fixées à 1, par au plus $\lceil k/2 \rceil$. Par conséquent, une nouvelle branche-gauche est créée et la contrainte suivante est injectée :

$$\Delta(x, \overline{x}) \leq k + \left\lceil \frac{k}{2} \right\rceil.$$

Après l'application de la procédure SP au problème, s'il n'y a pas d'amélioration nous imposons une contrainte tabou. Nous appliquons les étapes suivantes :

1. Rajouter la contrainte tabou suivante au problème pour modifier la structure de la solution courante :

$$\Delta(x, \bar{x}) \geq 1,$$

2. Résoudre le problème résultant en appliquant la procédure SP. Et considérer la solution obtenue comme la nouvelle solution de référence.

5.4 Partie expérimentale

Dans cette section, nous étudions les performances de l'algorithme proposé. Ces performances sont testées sur cinquante instances prises dans (Hifi et Michrafy [43] et générées selon le schéma de Yamada *et al.* [89, 90]). Les instances sont fortement corrélées, on peut les diviser en deux groupes. Le premier groupe est formé de 20 instances dont le nombre d'objets est 500 et la capacité est 1800. Le deuxième groupe est formé de 30 instances, chaque instance contient 1000 objets, et a une capacité qui varie de 1800 à 2000. Pour les deux groupes d'instances, le nombre de contraintes disjonctives varie entre 12000 et 50000. Toute l'approche a été codée en C et testée sur un `UltraSparc-II` (450Mhz avec 2Gb de RAM).

5.4.1 Performance de l'algorithme de branchement local standard

Lors de l'utilisation d'algorithmes approximatifs dans la résolution de problèmes d'optimisation combinatoire, le choix des valeurs des paramètres utilisés par l'algorithme de résolution a une grande influence sur la qualité des solutions obtenues. En particulier, l'algorithme de branchement local standard a besoin de deux paramètres de décision : (i) fixer la taille du voisinage qui sera exploré (ce qui correspond au paramètre k), et (ii) le critère d'arrêt. Pour la taille du voisinage, nos résultats expérimantaux ont été réalisés en faisant varier la valeur du paramètre k dans l'ensemble { 10, 20, 30, 40 }. Pour le critère d'arrêt, une limite au temps d'exécution de l'algorithme a été fixée parmi les éléments de l'ensemble { 400, 800, 1200 }. Cette limite est exprimée en secondes. Pour chaque $t \in$ { 400, 800, 1200 }, nous comparons la qualité des solutions fournies par l'algorithme de branchement local, en faisant varier la valeur du paramètre k dans l'intervalle discret suivant { 10, 20, 30, 40 }.

	k=10			k=20			k=30			k=40		
Nb Nodes CPU	t1	t2	t3	t1	t2	t3	t1	t2	t3	t1	t2	t3
1000	5	6	7	16	16	18	14	15	16	4	4	5
2500	4	5	6	13	15	20	16	18	20	3	4	6
5000	5	6	7	15	17	28	15	16	26	5	6	7

TAB. 5.1 – Comportement de l'algorithme de branchement local standard.

La table 5.1 résume les résultats obtenus par l'algorithme de branchement local standard,

en faisant varier la taille du voisinage. À partir de ces résultats, nous pouvons observer que les meilleures solutions sont obtenues avec les valeurs 20 et 30. Nous remarquons aussi que des valeurs de k trop petites ou trop grandes ne donnent pas necessairement de bonnes solutions. Il faut noter que tous les résultats obtenus par l'algorithme de branchement local standard restent modérés par rapport à ceux qui existent dans la littérature. Par la suite, nous maintenons, pour le paramètre k, uniquement les valeurs 20 et 30.

5.4.2 Performance de l'algorithme HLB$_{SP}$

Pour cette partie, nous allons continuer notre étude expérimentale sur les 50 instances de la littérature. En effet, nous étudions le compotement de l'algorithme hybride HLB$_{SP}$ pour $k = 20$ et $k = 30$. Tout d'abord, nous proposons d'étudier le comportement de HLB$_{SP}$ sans les mécanismes d'intensification et de diversification. Cette étude nous permettra de montrer l'apport de la procédure d'arrondi, en faisant varié α le pourcentage des variables fixées dans la première phase de la procédure SP. Pour cela, nous choisissons α dans {25%, 50%, 75% }. Les résultats sont rapportés dans les tables 5.2 et 5.3. La table 5.2 correspond au cas où le paramètre k est fixé à 20, alors que la table 5.3 correspnd à $k = 30$. Les deux tables affichent de la même manière :

1. - dans la première colonne le nom de l'instance à traiter,
2. - dans la deuxième colonne la meilleure solution connue dans la littérature,
3. - la colonne 3 (respectivment colonne 4 et colonne 5) montrent les solutions données par HLB$_{SP}$, pour $\alpha = 25\%$ (respectivement. $\alpha = 50\%$ et $\alpha = 75\%$).

Pour chaque valeur de α, les solutions données par HLB$_{SP}$ sont affichées pour les différents temps d'exécution : $t1 = 400$ sec, $t2 = 800$ sec et $t3 = 1200$ sec. À partir de la table 5.2 on peut observer que :

- Pour $\alpha = 25\%$:
  Premièrement, pour le temps d'exécution $t1$, l'algorithme HLB$_{SP}$ est capable d'améliorer 5 instances parmi 50. Deuxièment, quand on double le temps d'exécution ($t2 = 800$), l'algorithme HLB$_{SP}$ améliore 7 instances. Troisièment, avec un temps d'exécution égale à $t3$, l'algorithme HLB$_{SP}$ améliore 10 instances.
- Pour $\alpha = 50\%$: Si la limite au temps d'exécution est $t1$, alors l'algorithme améliore uniquement 6 solutions parmi 50. Alors qu'il améliore 9 solutions si la limite est fixée à $t2$. Enfin, si la limite est égale à $t3$, l'amélioration devient plus importante, et l'algorithme HLB$_{SP}$ améliore 41 instances parmi 50.

		HLB$_{SP}$.								
		$\alpha = 25\%$			$\alpha = 50\%$			$\alpha = 75\%$		
#Inst.	Best Sol.	t_1	t_2	t_3	t_1	t_2	t_3	t_1	t_2	t_3
1IA2	2571	2539	2543	2543	2584	2584	2584	2514	2514	2514
1IA3	2288	2270	2280	2280	2289	2290	2290	2250	2258	2258
1IA4	2280	2240	2266	2280	2250	2250	2280	2220	2220	2220
1IA5	2269	2276	2290	2290	2250	2250	2269	2250	2250	2250
2IA1	2067	1918	1918	1958	1954	2065	2067	1923	1923	1923
2IA2	2060	1837	1937	1937	2070	2070	2070	1743	1743	1743
2IA3	2080	1793	1793	1793	2060	2077	2080	1621	1621	1621
2IA4	2080	1988	1988	2038	1926	1926	2080	1662	1869	1869
2IA5	2068	1754	1963	1963	2068	2068	2070	1736	1736	1736
3IA1	1658	1301	1301	1301	1572	1596	1672	1244	1394	1394
3IA2	1732	1426	1437	1437	1447	1595	1732	1423	1567	1732
3IA3	1619	1425	1425	1481	1588	1588	1626	1343	1343	1343
3IA4	1686	1362	1362	1362	1548	1548	1686	1246	1246	1246
3IA5	1678	1398	1398	1671	1468	1558	1671	1330	1330	1330
4IA1	1126	1019	1019	1187	1099	1152	1187	996	996	996
4IA2	1222	1218	1218	1218	1218	1218	1218	1218	1218	1218
4IA3	1217	1195	1195	1195	1195	1195	1195	1195	1195	1195
4IA4	1278	1049	1049	1049	1270	1270	1278	938	938	938
4IA5	1311	1022	1022	1067	1095	1095	1312	1002	1002	1002
5I1	2610	2640	2648	2648	2640	2640	2648	2580	2580	2580
5I2	2653	2610	2648	2648	2600	2600	2653	2593	2593	2600
5I3	2650	2570	2600	2630	2570	2570	2650	2610	2610	2610
5I4	2640	2630	2660	2660	2610	2610	2648	2590	2620	2620
5I5	2649	2590	2630	2630	2599	2600	2649	2610	2610	2620
6I1	2780	2770	2770	2770	2710	2735	2780	2770	2770	2770
6I2	2750	2749	2760	2760	2750	2750	2750	2710	2749	2749
6I3	2760	2750	2780	2787	2709	2730	2760	2720	2720	2720
6I4	2760	2730	2749	2750	2740	2740	2751	2709	2709	2709
6I5	2760	2740	2750	2765	2740	2750	2760	2720	2720	2720
7I1	2726	2676	2680	2680	2659	2659	2728	2657	2657	2660
7I2	2720	2660	2700	2700	2650	2669	2702	2649	2649	2649
7I3	2710	2680	2680	2680	2678	2680	2710	2678	2678	2678
7I4	2710	2668	2668	2668	2705	2705	2710	2700	2700	2700
7I5	2710	2710	2719	2720	2700	2700	2710	2650	2650	2650
8I1	2668	2650	2650	2650	2610	2640	2668	2610	2610	2610
8I2	2655	2610	2630	2638	2600	2610	2657	2565	2565	2565
8I3	2680	2629	2629	2629	2630	2630	2675	2624	2624	2624
8I4	2660	2620	2630	2650	2610	2620	2660	2600	2600	2606
8I5	2657	2640	2640	2649	2580	2586	2659	2589	2589	2590
9I1	2602	2589	2589	2589	2580	2580	2610	2569	2569	2569
9I2	2630	2576	2590	2600	2600	2617	2630	2560	2560	2560
9I3	2601	2560	2560	2560	2568	2600	2612	2530	2550	2550
9I4	2620	2598	2598	2598	2569	2570	2610	2598	2598	2598
9I5	2619	2559	2559	2559	2560	2560	2620	2559	2559	2559
10I1	2570	2529	2529	2529	2549	2560	2564	2518	2518	2518
10I2	2609	2519	2530	2550	2536	2540	2627	2529	2529	2540
10I3	2576	2583	2583	2583	2540	2576	2580	2583	2583	2583
10I4	2592	2506	2542	2550	2544	2559	2601	2505	2530	2530
10I5	2590	2492	2550	2560	2546	2560	2583	2498	2498	2499

TAB. 5.2 – Performance de HLB$_{SP}$ comparée aux meilleures solutions de la littérature sur les 50 instances et avec $k = 20$

– Pour $\alpha = 75\%$:

HLB$_{SP}$ améliore une seule solution pour $t1$ et $t2$. Alors que pour $t3$ il en améliore deux, le tout parmi 50 instances.

		HLB$_{SP}$.								
		α = 25%			α = 50%			α = 75%		
#Inst.	Best Sol.	t_1	t_2	t_3	t_1	t_2	t_3	t_1	t_2	t_3
1IA1	2555	2537	2557	2557	2547	2547	2563	2473	2473	2475
1IA2	2571	2532	2573	2573	2524	2524	2573	2502	2514	2524
1IA3	2288	2290	2290	2290	2290	2290	2290	2220	2225	2225
1IA4	2280	2270	2280	2280	2248	2250	2260	2231	2237	2248
1IA5	2269	2250	2250	2250	2250	2250	2270	2250	2250	2250
2IA1	2067	1975	1975	1975	2007	2007	2067	1954	1954	1954
2IA2	2060	1994	1994	1994	1994	1994	2070	1994	1994	1994
2IA3	2080	2110	2110	2110	2110	2110	2110	2110	2110	2110
2IA4	2080	1758	1833	1833	1758	1772	1772	1833	1833	1848
2IA5	2068	1951	1958	1958	1941	1941	1941	1941	1941	1956
3IA1	1658	1372	1400	1411	1400	1400	1420	1400	1400	1411
3IA2	1732	1518	1518	1518	1518	1518	1521	1518	1518	1519
3IA3	1619	1446	1446	1446	1446	1446	1446	1446	1446	1446
3IA4	1686	1460	1460	1460	1460	1460	1460	1460	1460	1460
3IA5	1678	1398	1398	1398	1342	1342	1342	1330	1333	1380
4IA1	1126	1090	1090	1090	1090	1090	1090	1087	1087	1087
4IA2	1222	1211	1211	1211	1211	1211	1211	1198	1198	1198
4IA3	1217	1206	1206	1206	1206	1206	1206	1206	1206	1206
4IA4	1278	1136	1136	1136	1136	1136	1136	1136	1136	1136
4IA5	1311	1004	1004	1004	1004	1004	1004	1002	1002	1002
5I1	2610	2640	2640	2670	2579	2579	2630	2540	2546	2580
5I2	2653	2638	2640	2640	2620	2620	2640	2600	2610	2610
5I3	2650	2580	2610	2610	2600	2620	2620	2620	2620	2620
5I4	2640	2658	2660	2660	2600	2620	2620	2620	2620	2620
5I5	2649	2600	2610	2620	2609	2610	2610	2616	2620	2620
6I1	2780	2770	2770	2770	2770	2770	2770	2770	2770	2770
6I2	2750	2760	2770	2780	2750	2750	2750	2740	2743	2745
6I3	2760	2730	2730	2730	2730	2740	2760	2710	2710	2710
6I4	2760	2740	2750	2759	2720	2720	2720	2709	2709	2709
6I5	2760	2760	2760	2760	2748	2750	2750	2740	2740	2740
7I1	2726	2659	2690	2690	2660	2670	2680	2630	2650	2650
7I2	2720	2670	2670	2670	2668	2670	2670	2650	2650	2650
7I3	2710	2687	2700	2700	2680	2680	2680	2679	2679	2679
7I4	2710	2700	2710	2728	2697	2710	2710	2670	2670	2670
7I5	2710	2709	2710	2710	2680	2690	2690	2670	2670	2670
8I1	2668	2620	2620	2630	2625	2625	2625	2610	2610	2610
8I2	2655	2596	2600	2620	2620	2637	2650	2590	2590	2590
8I3	2680	2646	2648	2679	2639	2640	2650	2637	2637	2637
8I4	2660	2640	2660	2660	2641	2650	2660	2640	2650	2658
8I5	2657	2580	2617	2630	2590	2600	2600	2618	2618	2618
9I1	2602	2580	2580	2610	2580	2586	2590	2566	2579	2579
9I2	2630	2589	2600	2600	2590	2590	2590	2560	2561	2561
9I3	2601	2588	2590	2597	2570	2577	2577	2550	2550	2580
9I4	2620	2598	2598	2598	2598	2598	2598	2598	2598	2598
9I5	2619	2579	2579	2579	2560	2560	2560	2559	2559	2559
10I1	2570	2530	2539	2550	2523	2523	2523	2518	2518	2518
10I2	2609	2627	2627	2627	2627	2627	2627	2627	2627	2627
10I3	2576	2583	2583	2583	2583	2583	2583	2583	2583	2583
10I4	2592	2540	2560	2560	2520	2540	2540	2519	2540	2540
10I5	2590	2540	2568	2570	2500	2512	2540	2500	2522	2543

TAB. 5.3 – Performance de HLB$_{SP}$ comparée au meilleures solutions de la littérature, sur les 50 instances et avec $k = 30$.

L'étude de la table 5.3 montre les points suivants :

– Pour $\alpha = 25\%$:

L'algorithme HLB$_{SP}$ améliore 8 instances parmi 50 si la limite au temps d'exécution est fixée à $t1$. Pour $t2$, le nombre de solutions améliorées double pratiquement avec le temps d'execution (14 améliorations pour $t2$). Pour $t3$, on remarque que le nombre de solutions améliorées ne change pratiquement pas, comparé à $t2$ (15 solutions améliorées).

- Pour $\alpha = 50\%$: Pour $t1$ et $t2$, le nombre de solutions améliorées est sensiblement le même (5 et 6), alors que pour $t3$, il atteint 12 solutions améliorées.
- Pour $\alpha = 75\%$:
  En faisant varier le temps d'exécution entre $t1, t2$, et $t3$, nous remarquons que le nombre de solutions améliorées reste toujours égal à 3.

À la suite de ces résultats nous pouvons remarquer que les meilleures solutions sont globalement obtenues quand la limite du temps d'exécution est fixée à $t3$. Le couple $(k, \alpha) = (20, 50)$ fournit les meilleures solutions sur toutes les instances. Dans ce qui suit, nous maintenons ce couple de valeur dans le but de tester le comportement de l'algorithme combiné avec les stratégies de diversification et d'intensification.

5.4.3 Contribution de la diversification et de l'intensification

Lorsque l'algorithme HLB$_{SP}$ échoue à améliorer la solution en cours dans le noeud gauche du branchement local, les stratégies de diversification et d'intensification améliorent le comportement de l'algorithme, et de nouvelles solutions pour une partie des intances sont obtenues. Pour les paramètres de l'intensification nous considérons les valeurs suivantes : $k_1 = |\overline{B}_1| - 2,\ k_{max} = 10$ et $k_{step} = 2$, et aucune fixation n'est imposée à la phase de diversification. L'ajout des stratégies de diversification et d'intensification à l'algorithme HLB$_{SP}$ a permis d'améliorer 24 solutions parmi 50, voir tableau 5.4.

5.5 Conclusion

Dans ce chapitre, nous avons résolu le DCKP en utilisant trois algorithmes basés sur le concept de branchement local. Le premier algorithme est une adaptation du branchement local standard. Le deuxième algorithme est basé sur deux procédures complémentaires : une procédure d'arrondi combinée avec un branchement local. Le troisième algorithme enrichit le deuxième avec les stratégies de diversification et d'intensification. Les performances des trois algorithmes ont été téstées sur un ensemble d'instances de la littérature.

#Inst.	Meilleure Sol	HLB_{SP}
1IA1	2563	**2567**
1IA2	2584	**2594**
1IA3	2290	**2300**
1IA4	2280	2280
1IA5	2270	**2298**
2IA1	2067	**2069**
2IA2	2070	**2082**
2IA3	2110	2110
2IA4	2080	**2086**
2IA5	2070	**2073**
3IA1	1672	**1700**
3IA2	1732	**1743**
3IA3	1626	**1655**
3IA4	1686	**1703**
3IA5	1678	**1681**
4IA1	1187	**1266**
4IA2	1222	**1276**
4IA3	1217	**1316**
4IA4	1278	**1325**
4IA5	1312	1312
5I1	2648	2648
5I2	2653	2653
5I3	2650	2650
5I4	2660	2660
5I5	2649	2649
6I1	2780	**2794**
6I2	2780	2780
6I3	2787	2787
6I4	2760	2760
6I5	2765	2765
7I1	2728	**2729**
7I2	2720	2720
7I3	2710	2710
7I4	2710	2710
7I5	2720	2720
8I1	2668	2668
8I2	2657	**2672**
8I3	2680	2680
8I4	2660	2660
8I5	2659	2659
9I1	2610	**2621**
9I2	2630	2630
9I3	2612	**2623**
9I4	2620	2620
9I5	2620	2620
10I1	2570	2570
10I2	2627	**2630**
10I3	2583	2583
10I4	2601	**2609**
10I5	2590	2590

TAB. 5.4 – Comportement de HLB_{SP} avec la diversification et l'intensification.

Chapitre 6

Algorithmes exacts pour le DCKP

Dans ce chapitre, nous nous intéressons à la résolution exacte du sac-à-dos disjonctif. Nous proposons une méthode exacte qui s'appuie sur la génération de contraintes d'encardement de la somme des variables du problème à partir d'une solution réalisable. Les contraintes d'encadrement imposent des bornes sur la somme des variables du problème. L'intervalle construit à partir de ces bornes sera exploré de manière dichotomique, à la recherche de la solution optimale. La méthode est composée de trois phases de résolution. La première phase consiste à construire une borne inférieure, par application d'une heuristique gloutonne. Dans la deuxième phase, la méthode construit les inégaliés d'encadrement, afin d'encadrer la somme des variables et d'améliorer la qualité de la borne supérieure. Finalemnent, dans la troisième phase la méthode fait appel à une recherche dichotomique pour accélérer le processus de recherche de la solution optimale. La partie experimentale montre l'efficacité de la méthode proposée sur un ensemble d'instances de grande taille. La performance de la méthode est comparée à la performance de l'approche de Hifi et Michrafy [46].

6.1 Introduction

Généralement, un algorithme de résolution exact pour les problèmes de l'optimisation combinatoire s'appuie sur une procédure de séparation et évaluation. Cette approche se base sur le développement d'une arborescence ou d'un ensemble d'arborescences composé d'un ensemble de noeuds dispersés sur différents niveaux. À chaque niveau, on lui fait correspondre une évaluation à l'aide d'une fonction objectif représentant le problème à traiter. La solution obtenue sur un niveau est considérée comme une évaluation potentielle qui correspond à la meilleure solution en cours.

Dans cette partie, nous rappelons les travaux antérieurs de la littérature concernant la résolution exacte du problème du sac-à-dos à contraintes disjonctives. Nous rappeleons que Yamada *et al.* [89, 90] sont les premiers auteurs à avoir étudié le problème du DCKP. Ils ont proposé une première résolution. En effet, dans l'article précédemment cité les auteurs ont proposé deux algorithmes exacts. Le premier algorithme utilise la méthode de séparation et évaluation et le deuxième fait appel à des techniques de résolution par dichotomie. Par ailleurs, ils ont montré que le deuxième algorithme était plus performant dans le sens où celui-ci donnait les solution optimales sur un ensemble d'instances non-corrélées. Et ce en un temps de calcul moins important que le premier. On notera que ces deux algorithmes ne traitent que des instances non-corrélées.

Plus tard, Hifi et Michrafy [46] ont proposé différentes versions d'un algorithme exact. Cet algorithme s'appuie principalement sur des procédures de réduction, sur une recherche dichotomique, et sur un passage vers un modèle équivalent enrichi par des coupes de voisinage et de couverture. Les auteurs ont montré que ces différentes versions atteignaient leur limite lorsque la densité de l'instance traitée devenait plus importante (une densité forte). En effet, les auteurs ont montré que sur l'ensemble des instances fortement corrélées traitées, l'augmentation de la densisté d'une instance ne favorisait pas la fixation de variables à l'optimum. Par ailleurs, parfois l'application d'une procédure de réduction sur une telle instance peut engendrer une instance réduite dont la résolution devient plus complexe. Dans ce chapitre, nous proposons de résoudre les deux modèles équivalents du DCKP, en utilisant deux méthodes de résolution : La première injecte des contraintes d'encadrement avant d'appliquer une méthode de séparation et évaluation. La borne suéperieure est calculée par la méthode du simplexe et la borne inférieure est calculée par une heuristique gloutonne. La deuxième méthode combine une technique de recherche dichotomique avec les contraintes d'encadrement et un algorithme de séparation et évaluation. Pour plus de clarté, nous rap-

pelons que la formulation mathématique du problème du sac-à-dos disjonctif (DCKP) est donnée par :

$$(\text{DCKP}) \begin{cases} \text{maximiser} & Z(x) = \sum_{j=1}^{n} p_j x_j \\ \text{s.c.} & \sum_{j=1}^{n} w_j x_j \leq c \\ & x_i + x_j = 1, \qquad (i,j) \in E \\ & x_j \in \{0,1\}, \qquad j \in \{1,\ldots,n\}. \end{cases} \tag{1}$$

où x_j prend la valeur 1 si l'objet j est choisi, et 0 sinon.

Nous rappelons aussi que le modèle équivalent EDCKP peut s'écrire de la manière suivante :

$$(\text{EDCKP}) \begin{cases} \text{max} & \sum_{j=1}^{n} p_j x_j \\ \text{subject to} & \sum_{j=1}^{n} w_j x_j \leq c \\ & |E_i| x_i + \sum_{j \in E_i} x_j \leq |E_i|, \ i = 1,\ldots,n \\ & x_i \in \{0,1\}, \ i = 1,\ldots,n, \end{cases}$$

6.2 Contraintes d'encadrement

Le but de cette méthode est de déterminer un ensemble de variables ou "noyau", qui peut participer à la construction de la solution optimale. Cet ensemble est construit en général à partir de solutions réalisables du problème. En effet, les solutions réalisables données par les heuristiques permettent de caractériser cet ensemble de variables autour duquel s'articule l'instance du problème.

6.2.1 Problèmes associés au DCKP

Dans cette section, nous commencons par définir deux problèmes associés au DCKP, que nous noterons P_s et P_S. Ces deux problèmes sont construits à partir de solutions réalisables du DCKP, données par des heuristiques. Notons que la solution triviale de valeur zero peut construire ces deux problèmes. Les problèmes P_s et P_S sont des programmes linéaires en $0-1$

définis par :

$$(P_s)\begin{cases} \text{minimiser} & z(x) = \sum_{j=1}^{n} x_j \\ \text{sc.} & \sum_{j=1}^{n} x_j w_j \leq c \\ & \sum_{j=1}^{n} x_j p_j \geq LB \\ & x_i + x_j \leq 1, \qquad \forall (i,j) \in E \\ & x_j \in \{0,1\}, \qquad j = 1, \ldots, n, \end{cases}$$

et,

$$(P_S)\begin{cases} \text{maximiser} & z(x) = \sum_{j=1}^{n} x_j \\ \text{sc.} & \sum_{j=1}^{n} x_j w_j \leq c \\ & \sum_{j=1}^{n} x_j p_j \geq LB \\ & x_i + x_j \leq 1, \qquad \forall (i,j) \in E \\ & x_j \in \{0,1\}, \qquad j = 1, \ldots, n. \end{cases}$$

On peut associer à chaque solution réalisable du DCKP de valeur LB deux problèmes P_s et P_S. Soient s et S les valeurs des solutions optimales associées respectivement aux problèmes P_s P_S. Soit $d(LB)$ la distance associée à la borne inférieure LB, définie par :

$$d(LB) = S(LB) - s(LB).$$

Cette distance réprésente la cardinalité de l'ensemble des variables dans lequel on cherchera la solution optimale. La qualité de la recherche dépendera étroitement de la taille de la distance $d(LB)$. Soit le programme linéaire en $0-1$ suivant défini à partir des programmes P_s et P_S,

et associé au DCKP :

$$(P)\begin{cases} \text{maximiser} & z(x) = \sum_{j=1}^{n} x_j p_j \\ \text{sc.} & \sum_{j=1}^{n} x_j w_j \leq c \\ & \sum_{j=1}^{n} x_j \leq s \\ & \sum_{j=1}^{n} x_j \geq S \\ & x_i + x_j \leq 1 \qquad \forall (i,j) \in E \\ & x_j \in \{0,1\} \qquad j = 1, \ldots, n. \end{cases}$$

6.2.2 Quelques propriétés mathématiques

Dans ce qui suit, nous établissons quelques résultats théoriques qui justifient la validité des contraintes d'encadrement.

Corollaire 6.1 *Les deux inegaliés suivantes sont valides pour toute solution optimale du DCKP :*

$$(\mathbf{a})\ \sum_{j=1}^{n} x_j \geq S \quad \textit{et} \quad (\mathbf{b})\ \sum_{j=1}^{n} x_j \leq s$$

Preuve :
Pour montrer que les inégalités sont valides, il faut montrer que chaque solution optimale du problème DCKP vérifie les deux inégalités (**a**) et (**b**).
Supposons que y=$(y_1, \ldots, y_n)$ est une solution optimale du problème DCKP. Comme toute solution optimale est réalisable, alors y est une solution réalisable du DCKP. Dans tout ce qui suit nous supposons que LB est une borne inférieure du problème DCKP, alors

$$\sum_{j=1}^{n} p_j . y_j \geq LB.$$

Nous en déduisons que y est une solution réalisable de P_S et P_s. Ainsi, nous obtenons les inégalités (**a**) et (**b**) du corollaire .

Corollaire 6.2 *Les deux problèmes DCKP et P ont la même valeur optimale.*

Preuve :

Soient z_1 et z_2 les deux valeurs optimales respectivement de $DCKP$ et de P.

On va monter que $z_1 = z_2$: il suffit de montrer que $z_1 \leq z_2$ et $z_1 \geq z_2$.

On suppose qu'il existe deux solutions x=$(x_1, \ldots, x_n)$ et y=$(y_1, \ldots, y_n)$, dont les valeurs des fonctions objectif correspondantes sont respectivement z_1 et z_2.

- Nous savons que $z_1 \geq LB$ car LB est une borne inférieure de DCKP. En appliquant la Lemme 6.2.2, nous obtenons $\sum_{j=1}^{n} x_j \leq s$ et $\sum_{j=1}^{n} x_j \geq S$.
  D'où x est une solution réalisable de P.
- Nous savons aussi que y est une solution optimale de P de valeur objectif z_2 et x est une solution réalisable de P de valeur objectif z_1. Comme P est un problème de maximisation (toute solution réalisable est plus petite ou égale à la solution optimale) alors

  $$z_1 \geq z_2.$$

  De plus, comme y est une solution réalisable de P (car toute solution optimale est réalisable), alors $z_1 \geq z_2$.

Donc $z_1 = z_2$, donc les deux problèmes DCKP et P ont la même valeur optimale.

Corollaire 6.3 *Soient LB_1 et LB_2 deux bornes inférieures du problème DCKP. Alors on a :*

a) *si $LB_1 \geq LB_2$, alors $s(LB_2) \leq s(LB_1) \leq S(LB_1) \leq S(LB_2)$,*

b) *si l'ensemble des solutions du problème DCKP est non vide, alors*

$$\min\Big\{d(LB) \mid LB \text{ est une borne inférieure de } DCKP\Big\} = d(Z_0),$$

où Z_0 désigne la valeur optimale du problème DCKP.

Preuve :

a) Soit z une borne inférieure du problème DCKP à valeurs dans $[LB_1, LB_2]$. Et soient $P_s(z)$ et $P_S(z)$ les deux problèmes associés à la borne inférieure z. Soient $S(z)$ et $s(z)$ les valeurs optimales associées respectivement à $P_S(z)$ et $P_s(z)$, et y_s et y_S les solutions optimales associées respectivement à $P_S(z)$ et $P_s(z)$.

Comme $LB_1 > LB_2$, alors on a y_S est une solution réalisable de $P_S(LB_2)$, et y_s est une solution réalisable de $P_s(LB_2)$.

On en déduite que :

$$S(LB_1) \leq S(LB_2) \quad \text{et} \quad s(LB_2) \leq s(LB_1).$$

D'autres part, nous avons $s(LB_1) \geq S(LB_1)$, ce qui implique que

$$s(LB_2) \leq s(LB_1) \leq S(LB_1) \leq S(LB_2).$$

D'où la fonction qui associe LB à la distance d est décroissante.
b) Soient Z_0 la valeur optimale du problème DCKP et LB sa borne inférieure. Alors, nous avons $Z_0 \geq LB$. Nous en déduisons que

$$s(LB) \leq s(Z_0) \leq S(Z_0) \leq S(LB).$$

D'où $d(LB) \geq d(Z_0)$, et donc Z_0 est une borne inférieure du problème DCKP. □

Dans ce qui suit, nous notons le problème P par $\text{DCKP}_{(s,S)}$, sachant que LB est une borne inférieure du problème DCKP et que les valeurs s et S sont obtenues respectivement en résolvant les deux problèmes P_s et P_S.

Théorème 6.4 *Soit LB une borne inférieure du problème DCKP telle que $S(LB) \neq s(LB)$. Pour tout entier q tel que $q \in]s(LB), S(LB)[$ les deux ensembles réalisables des problèmes $DCKP_{(s,q)}$ et $DCKP_{(q+1,S)}$ forment une partition de l'ensemble des solutions réalisables du problème DCKP.*

Preuve :
Supposons que nous disposons de LB une borne inférieure du problème DCKP telle que $s(LB) < S(LB)$ et d'un entier q de l'intervalle ouvert $]s(LB), S(LB)[$. Si x est une solution réalisable du problème DCKP et z sa valeur. Alors d'après le Lemme 6.2.2, x vérifie les deux inégalités suivantes :

$$\sum_{j=1}^{n} x_j \leq S(LB) \quad \text{et} \quad \sum_{j=1}^{n} x_j \geq s(LB).$$

Si $\sum_{j=1}^{n} x_j \leq q$, alors x est une solution du problème $\text{DCKP}_{(s,q)}$. Sinon, x est une solution réalisable du $\text{DCKP}_{(q+1,S)}$.
L'ensemble des solutions réalisables du problème DCKP est alors formé des solutions réalisables des problèmes $\text{DCKP}_{(s,q)}$ et $\text{DCKP}_{(q+1,\ S)}$.

Pour montrer la partition, il suffit de montrer qu'il n'existe pas de solution réalisable appartenant à la fois à l'ensemble du $\text{DCKP}_{(s,q)}$ et à l'ensemble du $\text{DCKP}_{(q+1,S)}$. Supposons

que x est une solution réalisable du problème $\text{DCKP}_{(s,q)}$ et du problème $\text{DCKP}_{(q+1,S)}$. Alors, on a :

$$s \leq \sum_{j=1}^{n} x_j \leq q \quad \text{et} \quad q+1 \leq \sum_{j=1}^{n} x_j \leq S,$$

ce qui est absurde. □

6.3 Algorithmes exacts pour le DCKP

Dans cette section, nous présentons le principe de base commun aux différents algorithmes exacts proposés. Il s'agit d'introduire des inégalités valides pour le modèle traité. Ces inégalités vont permettre d'encadrer la valeur de la quantité de variables correspondant à la solution optimale. En fait, nous ajoutons au problème $DCKP$ une double inégalité de la forme : $\delta_{inf} \leq \sum_{j=1}^{n} x_j \leq \delta_{sup}$. Les valeurs δ_{inf} et δ_{sup} sont obtenues grâce à la résolution de deux problèmes de type sac-à-dos, qui sont construits à partir de la meilleure solution réalisable connue. L'objectif derrière l'encadrement de la somme des variables $\sum_{j=1}^{n} x_j$ est d'identifier un "noyau" autour du quel s'articule le problème traité, et la résolution du problème passe par la résolution de ce "noyau". La méthode exacte est composée des étapes suivantes :

1. Déterminer une borne inférieure initiale (correspond à une solution réalisable).
2. Construire les inégalités d'encadrement avec les évaluations δ_{inf} et δ_{sup}.
3. Ajouter les contraintes d'encadrement au problème original.
4. Résoudre le problème.

6.3.1 Construction des contraintes d'encadrement

Dans cette section, nous proposons différentes méthodes afin de générer les contraintes d'encadrement. Nous rappelons que le but est d'encadré la quantité Q suivante :

$$Q = \sum_{j=1}^{n} x_j.$$

Le calcul des contraintes d'encadrement fait appel à la résolution des deux problèmes vus précédemment, P_S et P_s. L'encadrmenet s'effectue pour toute solution réalisable dont la valeur est plus grande que LB, où LB dénote une borne inférieure du DCKP. Il peut aussi s'effectuer pour des solutions réalisables de valeurs inférieures à LB. Mais dans ce cas les

contraintes d'encadrement seront très larges, et l'éxploration de l'espace de recherche sera plus difficile.

On notera qu'une bonne estimation de Q peut rendre la résolution plus efficace par un algorithme de *séparation et évaluation*. En effet, ceci peut s'expliquer par le fait qu'un encadrement *très fin* de la somme des variables du problème permet de visiter moins de solutions ; vu que l'espace de recherche est réduit par les contraintes d'encadrement.

On peut remarquer que P_S et P_s sont deux problèmes difficiles à résoudre. Il est donc évident que parfois on peut être confronté au problème du temps d'exécution. Afin d'éviter ce problème, on fait appel à la résolution des relaxations continues des problèmes P_s et P_S. La résolution de ces deux problèmes peut être réalisée par application de l'algorithme du Simplexe. Plusieurs autres stratégies de résolutions peuvent être appliquées :

(**i**) La résolution par une méthode de *séparation et évaluation*.

(**ii**) Fixer des variables à l'optimum, puis tenter la résolution de l'instance réduite.

(**iii**) La résolution approchée des problèmes, ensuite utiliser d'autres méthodes pour affiner les bornes obtenues.

6.3.2 Algorithme basé sur les contraintes d'encadrement

Nous proposons deux versions d'un algorithme exact basé sur l'utilisation des contraintes d'encadrement de la solution opimale. La première version utilise le modèle initial du DCKP, alors que la deuxième utilise le modèle équivalent EDCKP.

Le premier algorithme noté $Algo1$ se déroule comme suit :

1. Considérer la borne inférieure fournie par l'application d'une heuristique au modèle $DCKP$.
2. Construire les deux problèmes P_S et P_s, et résooudre leurs relaxations continues par application de la méthode du simplexe.
3. Injecter les deux contraintes dans le problème DCKP et résoudre le nouveau problème P par application d'un algorithme de *branch and bound*.
4. Retourner la valeur optimale.

Nous rappelons que les inégalités d'encadrement sont construites à partir de la valeur de la meilleure solution réalisable. La qualité des inégalités dépend de la qualité de la valeur de la meilleure solution réalisable connue.

Le deuxième algorithme noté $Algo2$ se déroule comme suit :

1. Considérer la borne inférieure fournie par l'application d'une heuristque au modèle EDCKP.
2. Construire les deux problèmes P_S et P_s, et résooudre leur relaxation continue par application de la méthode du simplexe.
3. Injecter les deux contraintes dans le problème EDCKP et résoudre le nouveau problème P par application d'un algorithme de *branch and bound*.
4. Retourner la valeur optimale.

On notera que pour l'algorithme $Algo2$, nous avons proposé deux versions. La première version est celle dans la quelle les coefficients des contraintes de voisinage sont calculées par la méthode du simplexe. Dans la deuxième version notée $Algo3$, les coefficients des contraintes de voisinages sont calculés par la procédure de Jhonson (voir chapitre 4).

6.3.3 Algorithme basé sur une recherche dichotomique

La distance entre les bornes des inégalités d'encadrement δ_{inf} et δ_{sup} peut parfois être très large. Ce qui peut compliquer la résolution du problème P, dans la mesure où l'intervalle à explorer est important. Deux facteurs peuvent être à l'orgine de ce phénomène :

1. La borne inférieure utilisée pour la construction des contraintes d'encadrement est de mauvaise qualité.
2. La résolution des problèmes P_S et P_s demande un temps de calcul important.

Pour éviter le dit phénomène, nous proposons de résoudre le problème P en utilisant une recherche dichotomique dans l'intervalle $[\delta_{inf}, \delta_{sup}]$ combinée avec un algorithme de séparation et d'évaluation. Le principe général de l'algorithme dichotomique (noté A_{dico}) se résume par les étapes suivantes :

(**a**) Construire une borne inférieure LB en résolvant le problème DCKP (algorithme glouton).

(**b**) Construire les problèmes $P_{\delta_{inf}}$ et $P_{\delta_{sup}}$.

(**c**) Déterminer respectivement les évaluations δ_{inf} et δ_{sup} associées aux problèmes $P_{\delta_{inf}}$ et $P_{\delta_{sup}}$.

   (a) Poser $\Delta = (\delta_{inf} + \delta_{sup})/2$.

   (b) Construire le problème $P_{(\delta_i nf, \Delta)}$.

1. Initialiser la solution optimale en posant $Z = LB$.

2. Poser $P = P_{(\delta_i nf, \Delta)}$.

   Si $(\Delta \neq \delta_i nf)$ alors

   (a) résoudre P et soit LB sa solution optimale.

   (b) Si $(LB \geq Z)$ alors

   Poser $Z = LB$, et aller à l'étape (b)

   (c) Sinon résoudre $P_{(\Delta, \delta_m ax)}$, et retenir la valeur optimale LB

   (d) Si $(LB \geq Z)$ alors

   Poser $Z = LB$, et aller à l'étape (b)

   (e) Sinon arrêter

3. Renvoyer la valeur optimale Z.

On notera que deux autres versions de l'algorithme A_{dico} sont proposées dans ce chapitre. Les deux versions utilisent le modèle équivalent du DCKP noté EDCKP. La première version notée A_{dico1}, est celle dans laquelle les coefficients des contraintes de voisinage sont calculés par l'application de la procédure de Jhonson 4.2.2. Dans la deuxième version notée A_{dico2}, les coefficients des contraintes de voisinage sont calculés par la méthode du simplexe.

6.4 Partie expérimentale

Dans cette partie, nous nous intéressons à la performance de l'approche fondée sur les contraintes d'encadrement. Notre étude porte sur deux groupes d'instances, le premier groupe est composé de 10 instances extraites de Hifi et Michrafy [46]. Le deuxième groupe est composé de 10 instances fortement corrélées générées aléatoirement selon le schéma proposé par Yamada *et al.* [89, 90]. Cette section se divise en deux parties. Dans la première partie, nous étudions le comportement de l'approche sur le premier groupe d'instances. Dans la deuxième partie, nous évaluons les performances de l'approche sur le deuxième groupe d'instances. Nos algorithmes sont codés en C, ils utilisent les API du Cplex et ils sont exécutés sur une station SUN Ultra-Sparc10 (250 Mhz et avec 1Gb de mémoire RAM). Les performances de nos algorithmes sont comparées à celle du meilleure algorithme exact proposé par Hifi et Michrafy [46].

6.4.1 Le premier groupe d'instances

Le premier groupe d'instances est constitué de 10 instances fortement corrélées engendrées aléatoirement par Hifi et Michrafy [46], selon le schéma proposé par Yamada *et al.*([1]). Pour chaque instance, n (le nombre d'éléments) est égal à 1000. La capacité c du sac-à-dos varie dans l'ensemble {2000, 2500, 3000, 3500, 4500}. La densité associée aux contraintes disjonctives varie dans l'ensemble {0.007, 0.008, 0.009, 0.01, 0.11, 0.012, 0.013, 0.014, 0.015}. Le poids associé à chaque élément est choisi dans l'intervalle [1, 100]. Et le profit de chaque élément est exprimé en fonction de son poids $p_i = w_i + 10$, pour $i = 1, \ldots, n$.

Inst	BEST	cpu	*Algo*1	cpu	*Algo*2	cpu	*Algo*3	cpu	A_{dico}	cpu	A_{dico1}	cpu	A_{dico2}	cpu
p1	4810	178.1	∘	173	∘	139	∘	120	∘	57	∘	73	∘	79
p2	4880	122.9	∘	1	∘	113	∘	109	∘	19	∘	44	∘	81
p3	5490	131.6	∘	2	∘	131	∘	130	∘	25	∘	68	∘	97
p4	3560	155.1	∘	4	∘	111	∘	105	∘	23	∘	42	∘	56
p5	3500	235.5	∘	5	∘	109	∘	101	∘	19	∘	20	∘	86
p6	3490	76.5	∘	230	∘	145	∘	131	∘	124	∘	57	∘	60
p7	3450	214.4	∘	245	∘	138	∘	122	∘	25	∘	93	∘	78
p8	3420	460	∘	202	∘	131	∘	116	∘	25	∘	66	∘	67
p9	3400	255.6	∘	280	∘	147	∘	108	∘	22	∘	69	∘	51
p10	3410	134.9	∘	7	∘	121	∘	113	∘	28	∘	48	∘	67

TAB. 6.1 – Comportement de l'approche sur le premier groupe d'instances.

Le tableau 6.1 donne les résultats pour le premier groupe d'instances considérées. La première colonne affiche le nom de l'instance traitée. La colonne 2 affiche la valeur de la meilleure solution de la littérature, la colonne 3 affiche le temps CPU associé à cette solution. Les colonnes $4, 6, 8, 10, 12$ et 14 affichent respectivement les résultats des algorithmes *Algo*1, *Algo*2, *Algo*3, A_{dico}, A_{dico1}, et A_{dico2}. Les colonnes $5, 7, 9, 11, 13$ et 15 affichent les temps CPU associés aux algorithmes *Algo*1, *Algo*2, *Algo*3, A_{dico}, A_{dico1}, et A_{dico2}. Si l'un de ces algorithmes réussit à résoudre une instance à l'optimum, le symbole "∘" est affiché.

Nous remarquons que tous les algorithme réalisent un score de 10 sur 10 en résolvant à l'optimum toutes les instances. Ce qui différencie ces algorithmes, c'est le temps d'exécution. En effet, en consommant un temps d'exécution moyen égal à 196.7 secondes, l'algorithme de Hifi et Michrafy [46] (noté BEST) réalise le plus grand temps d'exécution. Tandis que les algorithmes *Algo*1, *Algo*2 et *Algo*3 utilisent respectvement des temps d'exécution moyens égaux à 106.9, 115.5 et 128.5 secondes. Parmi ces trois algorithmes, c'est l'algorithme *Algo*1 qui résout les instances à l'optimum avec le meilleure temps d'exécution moyen. Ce phénomène

[1]Ces instances sont disponibles sur le site `www.laria.u-picardie.fr/hifi/OR-Benchmark`.

est expliquable par le fait que les deux algorithmes $Algo2$ et $Algo3$ nécessitent un temps de calcul supplémentaire pour l'optimisation des contraintes de voisinage. Les algorithmes A_{dico}, A_{dico1}, et A_{dico2} réalisent respectivement les temps d'exécution moyens suivants : 36.7, 54.4 et 72.2. Ce qui peut être expliqué par le fait que l'algorithme A_{dico} est apliqué sur le modèle normal du DCKP. Alors que les deux autres algorithmes (A_{dico1} et A_{dico2}) sont appliqués sur le modèle équivalent (il y a un temps d'exécution pour l'optimisation des contraintes de voisinages). L'algorithme A_{dico1} résout les instances en un temps moyen plus petit que celui de l'algorithme A_{dico2}. Puisque l'optimisation des contraintes de voisinage par la procédure de Jhonson est plus rapide en terme de temps d'exécution que l'algorithme du Simplexe.

6.4.2 Le deuxième groupe d'instances

Le deuxième groupe d'instances est constitué de 10 instances fortement corrélées engendrées aléatoirement selon le schéma proposé par Yamada *et al* [89, 90]. Pour chaque instance, n (le nombre d'éléments) est choisi dans l'ensemble $\{500,\ 1000\}$. La capacité c du sac-à-dos varie dans l'ensemble $\{1800,\ 2000\}$. La densité associée aux contraintes disjonctives varie dans l'intervalle $[0.01,\ 0.04]$. Le poids associé à chaque élément est choisi dans l'intervalle $[1,\ 100]$, et le profit de chaque élément est exprimé en fonction de son poids $p_i = w_i + 10$, pour $i = 1, \ldots, n$.

Inst	BEST	$Algo1$	$Algo2$	$Algo3$	A_{dico}	cpu	A_{dico1}	cpu	A_{dico2}	cpu
I1	2219	2236	2237	2237	2247	516	2247	303	2247	108
I2	2243	2274	2274	2274	2284	473	2284	402	2284	378
I3	1920	1920	1940	1940	1980	248	1980	194	1980	126
I4	1949	1950	1950	1950	1970	223	1970	142	1970	96
I5	2470	2470	2480	2480	2520	243	2520	253	2520	178
I6	2010	2020	2020	2020	2020	246	2020	91	2020	88
I7	1970	1971	1971	1971	1974	330	1974	111	1974	103
I8	2001	2010	2010	2010	2020	439	2020	116	2020	106
I9	2011	2027	2027	2027	2030	218	2030	114	2030	75
I10	1970	1976	1976	1976	1980	101	1980	88	1980	67

TAB. 6.2 – Comportement des algorithmes sur le deuxième groupe d'instances.

Le tableau 6.2 donne les résultats pour le deuxième groupe d'instances considérées. La première colonne affiche le nom de l'instance traitée. La colonne 2 affiche la valeur de la meilleure solution de la littérature. Les colonnes $3, 4$ et 5 affichent respectivement les résultats des algorithmes $Alpha1$, $Alpha2$, $Alpha3$. Les colonnes $6, 8$ et 10 affichent respectivement les résultats des algorithmes A_{dico}, A_{dico1}, et A_{dico2}. Les colonnes $7, 9$ et 11 affichent le temps

CPU associé aux algorithmes A_{dico}, A_{dico1}, et A_{dico2}. Pour les algorithmes $Algo1$, $Algo2$, $Algo3$ et l'algorithme de Hifi et Michrafy [46] nous avons fixé le temps d'exécution à 1800 secondes. En observant le tableau 6.2 on remarque que :

- L'algorithme de Hifi et Michrafy [46] ne résout aucune instance à l'optimum.
- Les algorithmes $Algo1$, $Algo2$ et $Algo3$ réalisent chacun un score de 1 sur 10.
- Les algorithmes A_{dico}, A_{dico1}, et A_{dico2} réalisent un score de 10 sur 10 en résolvant à l'optimum toutes les instances du deuxième groupe.

L'algorithme A_{dico} réalise un temps d'exécution moyen égal à 303.7. Ce qui peut être expliqué par le fait que pour les instances du deuxième groupe, le modèle initial du DCKP est plus difficile à résoudre que le modèle équivalent. Les algorithmes A_{dico1} et A_{dico2} réalisent respectivement des temps d'exécution moyens égaux à 181.4 et 132.5 respectivement. L'algorithme A_{dico2} réalise le meilleur temps d'exécution à cause de la qualité des coefficients des contraintes d'encadrement donnés par la méthode du Simplexe.

Finalement on peut remarquer que la recherche dichotomique couplée avec les contraintes d'encadrement favorise l'accélération du processus de résolution.

6.5 Conclusion

Dans ce chapitre, nous avons proposé dans un premier temps trois algorithmes exacts basés sur des contraintes d'encadrement de la somme des variables du problème. En suite, nous avons couplé ces contraintes d'encadrement avec une recherche dichotomique. Le but étant de proposer trois autres algorithmes exacts. Ces algorithmes peuvent être considérés comme étant des améliorations des trois permiers. Nos algorithmes utilisent les deux modèles équivalent du sac-à-dos à contraintes disjonctives. Enfin, la partie expérimentale a montré l'efficacité des algorithmes. En particulier sur les instances denses et fortement corrélées.

Chapitre 7

Conclusion générale et perspectives

7.1 Conclusion

Dans cette thèse, nous avons traité le problème du sac-à-dos à contraintes disjonctives, pour lequel nous avons proposé plusieurs approches de résolution. Ces approches peuvent être divisées en deux grandes catégories : les méthodes approchées et les méthodes exactes. La première partie est consacrée à la résolution approchée du problème du sac-à-dos à contraintes disjonctives. Il s'agit de proposer des heuristiques, non seulement capables de résoudre efficacement des instances fortement corrélées du problème, mais aussi généralisables à d'autres problèmes de la famille du sac-à-dos. La deuxième partie s'intéresse à la résolution exacte du problème du sac-à-dos à contraintes disjonctives. Nous avons proposé plusieurs algorithmes exacts qui s'appuient aussi bien sur le modèle normal que sur le modèle équivalent du sac-à-dos à contraintes disjonctives.

Le chapitre 4 porte sur l'étude du problème de sac-à-dos à contraintes disjonctives. Nous avons proposé deux heuristiques augmentées qui s'appuient sur une procédure d'arrondi couplée avec des contraintes d'encadrement de la somme des variables. Les solutions générées par les deux algorithmes sont améliorées par une recherche locale qui utilise un mécanisme de Hill Climbing. Nos expérimentations montrent que pour un temps d'exécution équivalent, nos algorithmes donnaient des solutions de meilleure qualité, comparées aux solutions données par les heuristiques de la littérature.

Le chapitre 5 est dans la continuité de l'étude menée dans le chapitre 4, dans le sens

où l'objectif est de concevoir une heuristique capable d'améliorer les solutions obtenues par les algorithmes augmentés du chapitre 4. Cette heuristique est basée sur le mécanisme de branchment local, dans lequel une partie des variables est fixée grâce à un procédé d'arrondi, elle utilise un mécanisme de diversification très développé pour visiter les régions non encore explorées. Nous avons montré que notre heuristique était capable de produire une amélioration très significaive non seulement par rapport à l'olgorithme de branchement local standard, mais aussi par rapport aux meilleurs algorithmes de la littérature.

Le chapitre 6 propose une nouvelle approche pour le problème du sac-à-dos à contraintes disjonctives. L'idée est d'essayer d'encadrer le plus étroitement possible la solution optimale de l'instance traitée. Cet encadrement s'appuie sur la borne inférieure et la résolution approximative de deux problèmes de type sac-à-dos. Cette approche a donné lieu à plusieurs algorithmes exacts. Ces algorithmes se basent sur les deux modèles équivalents du sac-à-dos à contraintes disjonctives, dont ils résolvent les problèmes associés pour construire les contraintes d'encadrement. Ils appliquent une méthode de *branch and bound* afin de calculer la solution optimale. Une partie de ces algorithmes utilise une méthode de recherche dichotomique pour accélérer la résolution. Les performances de ces algorithmes sont comparées par rapport à celles des meilleurs algorithmes exacts de la littérature. De nouvelles instances de grande taille ont été générées pour tester l'efficacité de ces algorithmes, les résultats sont encourageants.

7.2 Perspectives

Plusieurs extensions de ce travail peuvent être possibles. Nous pensons que l'adaptation des deux algorithmes augmentés du chapitre 4 à d'autres problèmes de la famille du sac-à-dos, est une première direction de recherche. L'idée est de proposer une méthode générique qui combine les deux algorithmes, sauf qu'elle essaye au préalable de construire un noyau autour duquel va s'articuler l'instance. Dans ce noyau, on pourrait envisager une partie des variables fractionnaires qui serait fixée à un, et une autre partie des variables entières qui serait fixée à sa valeur. Ensuite, une résolution d'une série de sous-problèmes induite, par application d'une méthode par séparation et évaluation, permettrait la création de certaines contraintes d'encadrement. Ces dernières contraintes pourrait nous permettre de générer le prochain sous-problème à traiter.

Une deuxième direction de recherche consisterait à proposer une méthode approchée dérivant de la méthode exacte exposée dans le chapitre 6. En effet, l'idée consisterait à remplacer la

méthode de séparation et évaluation dichotomique par l'algorithme de branchement local du chapitre 5. Dans ce cas, nous avons constaté que la recherche dichotomique construisait des intervalles de plus en plus petits, alors que la méthode basée sur le branchement local pouvait explorer de manière rapide et efficace le voisinage de chaque noeud développé.
Une troisième direction de recherche consisterait à construire des contraintes valides ainsi que des coupes pertinentes permettant la résolution exacte de certaines instances du DCKP non résolues à l'optimum.

Bibliographie

[1] Balas E and Zemel E. An algorithm for large zero-one knapsack problem, *Operations Research,* **28** : 1130-1154, 1980.

[2] Beasley J.E and Chu P.C. A genetic algorithm for the set covering problem, *European Journal of Operational Research,* **94** : 392-404, 1996.

[3] Boyer V, El Baz D and Elkihel M. A dynamic programming method with dominance technique for the knapsack sharing problem, Proceedings of CIE39 -*International Conference on Computers & Industrial Engineering,* Troyes, France, Juillet, 2009.

[4] Brown JR. The Knapsack sharing, *Operation Reaserch,* **27** : 341-355, 1979.

[5] Brown JR. Solving Knapsack sharing with general tradeoff fonctions, *Mathematical Programming,* **5** : 55-73, 1991

[6] Cappanera P and Trubian M. A Local Search Based Heuristic for the Demand Constrained Multidimensional Knapsack Problem, *INFORMS JOC*, 17(1), 82-98, 2005.

[7] Caprara A, Pisinger D and Toth P. Exact solution of the quadratic knapsack problem, *INFORMS Journal on Computing,* **11** : 125-137, 1999

[8] Cherfi N and HIFI M. A Column Generation Method for the Multiple-Choice multi dimensional knapsack problem, *Combinatorial Optimization and Application*, DOI. 10.1007/s10589-008-9184-7, 2009.

[9] Cherfi N and HIFI M. Hybrid Algorithms for the Multiple-Choice Multi-Dimensional Knapsack Problem, *International Journal of Operational Research (IJOR)*, **5** : 89-109, 2009.

[10] Cherfi N. Méthodes de résolution hybrides pour les problèmes de type knapsack. PhD thesis, CERMSEM, Université de Paris 1, 2008.

[11] Chu P and Beasley J. A genetic algorithm for the multidimensional knapsack problem, *Journal of Heuristics,* **4** : 63-86, 1998.

[12] Dammeyer F and Voss S. Dynamic tabu list management using reverse elimination method, *Annals of Operations Research* 31-46, 1993.

[13] Dantzig G.B. Discrete variable extremum problems, *Operations Research*, **5** : 266-277, 1957.

[14] Dudinski K and Walukiewicz S. Exact methods for the knapsack problem and its generalizations, *Mathematical Programming*, **29** : 231-249, 1984.

[15] Ehrgott M and Gandibleux X (éditeurs). Multiple Criteria Optimization : State of the Art Annotated Bibliographical Surveys, Kluwer, 2002.

[16] Elkihel M, Authié G and Viader F. An efficient hybrid dynamic algorithm to solve 0-1 knapsack problems, International Symposium on Combinatorial Optimization (CO'2002), 2002, Paris (France).

[17] Elkihel M and Plateau G. A hybrid method for the 0-1 knapsack problem, 9ème Symposium de Recherche Operationnelle, 1984, Osnabruck (Allemagne).

[18] Fischetti M and Lodi A. Local branching, *Mathematical Programming*, **98** :23-47, 2003.

[19] Fischetti M, Polo C and Scantamburlo M. A lo- cal branching heuristic for mixed-integer programs with 2-Level Variables, *Networks*, **44** : 61-72, 2004.

[20] Fréville A and Plateau G. An exact search for the solution of the surrogate dual for the bidmensional knapsack problem, *European Journal of Operational Research,* **68** : 413-421, 1993.

[21] Fréville A and Plateau G. The 0-1 bidimensional knapsack problem : towards an efficient high-level primitive tool, *Journal of Heuristics,* **2** : 147-167, 1996.

[22] Fayard D and Plateau G. An algorithm for the solution of the 0-1 knapsack problem, *Computing* **28** : 269-287, 1982.

[23] Fréville A and Plateau G. An Efficient Preprocessing Procedure for the Multidimensional 0-1 Knapsack Problem, *Discrete Applied Mathematics,* **49** : 189-212, 1994.

[24] Gallo G, Hammer P. L and Simeone B. Quadratic knapsack problems, *Mathematical Programming Study*, **12** : 132-149, 1980.

[25] Garey M.R and Johnson D.S. Computers and intractability : a guide to the theory of NP-completeness, *Freeman and Company*, San Francisco, 1979.

[26] Gavish B and Pirkul H. Allocation of databases and processors in a distributing for data processing. J. Akola(Eds), *Management of Distributed Data Processing,* 215-231.

[27] Gilmore P.C and Gomory R.E. Multistage cutting problems of two and more dimensions, *Operations Research* **13** : 94-119, 1965.

[28] Gilmore P.C and Gomory R.E. The theory and computation of knapsack functions, *Operations Research* **14** : 1045-1074, 1966.

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

[30] Glover F and Kochenberger G. Critical Event tabu Search for Multidimensional Knapsack Problems. Dans I.H Osman et J.P Kelly (éditeurs), Meta Heuristics : Theory and Applications, Kluwer Academic Publishers, 407-427, 1996.

[31] Glover F. Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research*, **13**, 533-549, 1986.

[32] Glover F and Laguna M. Tabu search, *Kluwer Academic Publishers*, 1997.

[33] Gens and Levner. Computational complexity of approximation algorithms for combinatorial problems, *Mathematical Foundations of Computer Science*, 292-300, 1979.

[34] Guéret C and Prins C. A new lower bound for the open-shop problem. *Annals of Operations Research,* **92** : 165-183, 1999.

[35] Hanafi S and Fréville A. An efficient tabu search approach for the 0-1 multidimensional knapsack problem, *European Journal of Operational Research,* **106** : 659-675, 1998.

[36] Hansen P. The steepest ascent mildest descent heuristic for combinatorial programming, Presented at the *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy, 1986.

[37] Hifi M, Negre S and Ould Ahmed Mounir M. Local branching-based algorithm for the disjunctively constrained knapsack problem, Proceedings of CIE39 -*International Conference on Computers & Industrial Engineering,* Troyes, France, Juillet, 2009.

[38] Hifi M et Ould Ahmed Mounir M. Un algorithme augmenté pour le problème du knapsack disjonctif, *Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision,* Nancy, France, 10-12 Février, 2009.

[39] Hifi M, M'Hallah R and Ould Ahmed Mounir M. Augmented rounding algorithms for the disjunctively constrained knapsack problems, *Computational Optimization and Applications*, soumis, 2009.

[40] Hifi M, M'halla H and Sadfi S. An exact algorithm for the knapsack sharing problem, *Computers Operations Research*, **32** : 1311-1324, 2005

[41] Hifi M and Sadfi S. The knapsack sharing : an exact algorithm, *Journal of Combinatorial Optimization*, **6** : 35-55, 2002.

[42] Hifi M, Sadfi S and Sbihi A. An efficient algorithm for the knapsack sharing problem, *Computational Optimization and Applications*, **23** : 27-45, 2002.

[43] Hifi M and Michrafy M. A reactive search-based algorithm for the disjunctively knapsack problem, *Journal of the Operational Research Society*, **57** : 718-726, 2006.

[44] Hifi M, Michrafy M and Sbihi A. Heuristic algorithms for the multiple-choice multidimensional knapsack problem, *Journal of the Operational Research Society,* **55** : 1323-1332, 2004.

[45] Hifi M, Michrafy M and Sbihi A. A reactive local search-based algorithm for the multiple-choice multidimensional knapsack problem, *Computational Optimization and Applications*, **33** : 271-285, 2006.

[46] Hifi M and Michrafy M. Reduction strategies and exact Algorithms for the disjunctively constrained knapsack roblem, *Computers and Operations Research*, **34** : 2657-2673, 2007.

[47] Hifi M and Roucairol C. Approximate and exact algorithms for constrained (un)weighted two-dimensional two-staged cutting stock problems, *Journal of Combinatorial Optimization*, **5** : 465-494, 2001.

[48] Hifi M and Zissimopoulos V. A recursive exact algorithm for weighted two-dimensional cutting, *European Journal of Operational Research*, **91** : 553-564, 1996.

[49] Horowitz E and Sahni S. Computing partitions with applications to the knapsack problem, *Journal of ACM*, **21** : 277-292, 1974.

[50] Hill R and Reilly C. H. The effects of coefficient correlation structure in two- dimensional knapsack problems on solution performance, *Management Science,* **46 :** 302-317, 2000.

[51] Hirabayashi R, Suzuki H and Tuchiya N. Optimal tool module design problem for NC machine tools, *Journal of the Operations Research Society of Japan*, **27** : 205-229, 1983.

[52] Ibaraki T and Fukushima M, Fortran77 optimization programming. Iwanami, Tokyo (in Jananese), 1991.

[53] Khan S, Li K.F. Manning E-G and Akbar MD-M. Solving the knapsack problem for adaptive multimedia systems, *Studia Informatica, an International Journal*, Special Issue on Cutting, Packing and Knapsacking problems, **2** : 154-174, 2002.

[54] Khan S, Quality adaptation in a multi-session adaptive multimedia system : model and architecture, PhD Thesis, Department of Electronical and Computer Engineering, Uiversity of Victoria, May 1998.

[55] Kellerer H, Pferschy U and Pisinger D. *Knapsack Problems.* Springer, 2003.

[56] Kilby P, Prosser P and Shaw P. Guided local search for the vehicle routing problem, *Proceeding of the 2nd International Conference on Mataheuristics(MIC97)*, Sophia-Antipolis, France, 21-24, 1997.

[57] Lorie J. and Savage L.J. Three problems in capital rationing, *Journal of Business*, **28** : 229-229, 1955.

[58] Magazine M and Chern M. A note on approximation schemes for multidimensional knapsack, *Mathematics of Operations Research*, **9** : 244-247, 1984.

[59] Magazine M and Oguz O. A heuristic algorithm for the multidimensional zer-one knapsack problem, *European Journal of Operational Research*, **16** : 319-326, 1984.

[60] Martello S and Toth P. An exact algorithm for the two-constraint 0-1 knapsack problem, *Operations Research,* **51** : 826-835, 2003.

[61] Martello S Toth P. Algorithms for knapsack problems, *Annals of Discrete Mathematics*, **31** : 70-79, 1887 .

[62] Martello S and Toth P. A new algorithm for the 0-1 knapsack problem, *Management Science*, **34** : 633-644, 1980.

[63] Martello S and Toth P, Knapsack problems : algorithms and computer implementations, *Wiley* : Chichester, England, 1990.

[64] Martello S and Toth P. Upper bounds and algorithms for hard 0-1 knapsack problems, *Operations Research*, **45** : 768-778, 1997.

[65] Martello S and Toth P. A bound and bound algorithm for the zero-one multiple knapsack problem, *Discrete Applied Mathematics*, **3** : 275-288, 1981.

[66] Martello S and Toth P. A mixture of dynamic programming and branch-and- bound for the subset-sum problem, *Management Science,* **30** : 765-771, 1984.

[67] Martello M, Pisinger D and Toth T. Dynamic programming and strong bounds for the 0-1 knapsack problem, *Management Science*, **45** : 414-424, 1999.

[68] M'HALLA H. Sensibilité de l'optimum, méthodes adaptatives et algorithmes exacts pour des problèmes de type knapsack. PhD thesis, CERMSEM, Université de Paris 1, 2003.

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

[70] Minoux M, Programmation mathématique : théorie et algorithme . *Vol 1 et 2*, Dunod , Bordas , 1989.

[71] Moser M, Jokanovic DP and Shiratori N. An algorithm for the multidimesional multiple-choice knapsack problem, *IEECE Transactions on Fundamentals of Electronics*, **80** : 582-589, 1997.

[72] Nauss M.R. The 0-1 knapsack problem with multiple-choice constraints, *European Journal of Opeartional Research*, **2** : 125-131, 1978.

[73] Shih W. A branch and bound method for the multi-constraint zero-one kanp-sack problem, *Journal of the Operational Research Society* **30** : 369-378, 1979.

[74] Pisinger D. A minimal algorithm for the multiple-choice knapsack problem, *European Journal of Operational Research* **83** : 394-410, 1995.

[75] Pisinger D. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research,* 114 : 528-541, 1999.

[76] Pisinger D. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research,* **114** :528-541, 1999.

[77] Pisinger D and Sigurd M. M. Using decomposition techniques and constraint programming for solving the two-dimensional bin packing problem, *Technical Report* : **03/1**, Department of Computer Science, University of Copenhagen, Denmark, 2003.

[78] Richardson J, Palmer M, Liepins G and Hilliard M. Some guidelines for genetic algorithms with penalty functions. In : Schaffer J (eds). *Proceedings of the Third International Conference on Genetic Algorithms* : 191-197. Maurgan Kaufmann, 1989.

[79] Salkin M. On the merit of the generalized origin and restarts in implicit enumeration, *Operations Research,* **18** : 549-554, 1970.

[80] Tang CS. A max-min allocation problem ; its solutions and applications, *Operations Reaserch,* **36** : 359-367, 1988

[81] Thiongane B, Nagih A and Plateau G. Lagrangean heuristics combined with reoptimization for the 0-1 biknapsack problem, *Computer Science Laboratory, University of Paris-Nord*, 2003.

[82] Toyoda Y. A simplified algorithm for obtaining approximate solution to zero-one programming problems, *Management Science*, **21** : 1417-1427, 1975.

[83] Tsang E.P.K and Voudouris C. Fast local search and guided local search and their application to British Telecom's worforce scheduling problem, *Operations Research Letters*, **20** : 119-127, 1997.

[84] Vasquez M and Vimont Y. Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, **165** : 70-81, 2005.

[85] Voudouris C and Tsang E.P.K, Partial constraint satisfaction problems and guided local search for combinatorial, In *Proceedings of Practical Application of Constraint Technology (PACT'96)*, 337-356, 1996.

[86] Voudouris C and Tsang E.P.K , Guided local search and its application to the travelling salesman problem, *European Journal of Operational Research*, **113** : 469-499, 1999.

[87] WILBAUT C. Heuristiques Hybrides pour la Résolution de Problèmes en Variables 0-1 Mixtes. PhD thesis, université de Valenciennes et du Hainaut-Cambrésis, 2006.

[88] Wolsey L. A. Integer Programming, New York : Wiley, 1998 .

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

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

[91] Yamada T, Futakawa M and Kataoka S. Some exact algorithms for the knapsack sharing problem, *European Journal of Operational Research*, **106** : 177- 183, 1998.

[92] Yamada T and Futakawa M. Heuristic and exact reduction algorithms for the Knapsack sharing problem. *Computers and Operation Reaserch*, **21** : 961-967, 1997.