REPUBLIQUE TUNISIENNE
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

UNIVERSITE DE SFAX
Faculté des Sciences Economiques et de
Gestion de Sfax

THÈSE DE DOCTORAT

Pour L'obtention du Titre de Docteur en

Informatique

Méthodes d'optimisation exactes et heuristiques pour des variantes du problème du sac à dos

Présentée et soutenue publiquement par :

Mariem BEN SALEM

Membres du jury :

Mr. Mounir MARRAKCHI	**Maître de Conférences - FS – Sfax**	**Président**
Mme. Hanêne BEN-ABDALLAH	**Professeur - FSEG – Sfax**	**Directeur**
Mr. Faouzi BEN CHARADA	**Maître de Conférences – FS - Tunis al Manar**	**Rapporteur**
Mr. Imed KACEM	**Professeur - Université de Lorraine - France**	**Rapporteur**
Mr. Ridha MAHJOUB	**Professeur - Université Paris Dauphine - France**	**Membre**
Mr. Saïd HANAFI	**Professeur - Université de Valenciennes - France**	**Invité**

Année Universitaire : 2015-2016

Remerciements

Au terme de ce travail, je tiens à remercier tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce travail.

Je voudrais tout d'abord adresser mes chaleureux remerciements à ma directrice de thèse, Madame Hanêne BEN ABDELLAH, pour le courage et l'hospitalité dont elle a fait preuve envers moi en acceptant de diriger ma thèse. Je lui exprime toute ma reconnaissance de m'avoir encadré, orienté et m'avoir permis de mener ce travail jusqu'au bout.

Mes remerciements vont également à Monsieur Mounir MARRACHI qui m'a fait l'honneur de présider ce jury, aux deux rapporteurs de cette thèse, Monsieur Faouzi BEN CHARADA et Monsieur Imed KACEM pour avoir accepté d'évaluer ce travail. La version finale de ce manuscrit a bénéficié de la lecture attentive et des remarques précieuses de tous les membres du jury.

Je suis particulièrement très honorée par la participation de Monsieur Saïd HANAFI et Monsieur Ridha MAHJOUB à ce jury. Je tiens à leur témoigner ma profonde gratitude. Ils ont su m'accompagner tout au long de ces années avec constance, exigence et enthousiasme. Je les remercie sincèrement pour leur soutien sans faille, leur confiance et leur disponibilité. J'ai beaucoup appris à leurs côtés et je leur adresse toute ma reconnaissance. Je les remercie également de m'avoir accueilli dans leurs laboratoires respectifs à Valenciennes et à Paris.

Mes sincères remerciements s'adressent à Mademoiselle Raouia TAKTAK pour sa contribution à la valorisation scientifique de ce travail et pour tous les efforts qu'elle a fournis pour l'enrichir et l'améliorer.

Je voudrais associer à ces remerciements Madame Mariem GZARA : qu'elle trouve

ici l'expression de ma gratitude pour le soutien qu'elle m'a apporté durant les premières années de cette thèse.

Je tiens à remercier particulièrement Monsieur Mahdi KHEMAKHEM pour ses conseils, pour son aide et pour les discussions scientifiques que nous avons pu avoir au cours de ma thèse.

Je souhaite encore adresser mes remerciements à Monsieur Christophe WILBAUT pour son aide précieuse et pour son accueil lors de mon séjour à Valenciennes.

Pour terminer ces remerciements, j'exprime une énorme et affectueuse pensée à celui qui m'a inculqué l'envie d'apprendre, qui a nourri ma curiosité intellectuelle et m'a donné l'envie de faire de la recherche, mon cher père Pr. Moncef BEN SALEM qui nous a quitté en mars 2015. J'espère avoir été à la hauteur de ses espérances et avoir accompli avec humilité quelques peu de ses vœux les plus chers.

Ce préambule ne peut s'achever sans remercier particulièrement ma mère et mes frères qui m'ont constamment épaulé. J'exprime mes sincères remerciements à toute ma belle-famille pour leurs encouragements.

Enfin je remercie mes deux filles Emna et Hiba pour avoir supporté mes indisponibilités voire mes absences répétées de la maison et surtout mon époux pour sa grande patience, son soutien quotidien et la confiance qu'il m'a témoignée pendant toutes ces années.

Résumé

Plusieurs problèmes réels peuvent être ramenés à la maximisation d'un certain profit sous une certaine contrainte budgétaire ou de capacité à ne pas dépasser. Ces problèmes sont dits de type sac à dos et ne cessent de susciter l'intérêt des praticiens ainsi que les théoriciens. Les problèmes de type sac à dos sont des problèmes d'optimisation combinatoire qui peuvent être décrits comme suit. Etant donné une capacité fixe, et un ensemble d'objets caractérisés chacun par un profit et un poids, l'objectif est de sélectionner un sous ensemble de ces objets qui soit de profit maximum et dont la somme de poids ne dépasse pas la capacité donnée. Il existe plusieurs variantes du problème du sac à dos qui émanent d'applications réelles dans des domaines aussi différents que l'informatique (cloud computing, data mining, cryptographie), les télécommunications, la logistique et l'énergie.

Dans cette thèse nous nous intéressons à deux variantes intéressantes du problème du sac à dos, à savoir le problème du sac à dos avec contraintes disjonctives, et le problème du sac à dos multidimensionnel.

Pour le problème du sac à dos à contraintes disjonctives, nous exposons quatre formulations valides pour le problème, les trois premières en programmes linéaires en nombres entiers et la dernière est quadratique. Nous nous intéressons par la suite à la formulation dite standard. Nous étudions le polytope associé et décrivons de nouvelles classes d'inégalités valides. Nous donnons des conditions nécessaires et suffisantes pour que ces inégalités définissent des facettes. Nous discutons également d'algorithmes de séparation pour ces inégalités et en utilisant ces résultats, nous développons un algorithme de coupe et branchement pour le problème.
Parallèlement à cette méthode exacte, nous proposons de résoudre le problème en utilisant la métaheuristique de Recherche Tabou probabiliste, qui n'est autre qu'une Recherche Tabou dans laquelle on rajoute une pondération sur les mouvements favorisant

ceux avec les plus hautes évaluations.

Concernant le problème du sac à dos multiple, nous utilisons une approche hybride basée sur des relaxations linéaires pour laquelle nous utilisons une technique de décomposition afin de réduire l'espace des solutions.

Des résultats expérimentaux présentés suite à chaque méthode décrite ci-dessus montrent la performance des algorithmes proposés pour une résolution efficace d'instances difficiles du problème.

Mots clés : Problème du sac à dos à contraintes disjonctives, problème du sac à dos multidimensionnel, polytope, algorithme de coupe et branchement, recherche tabou probabiliste, approche hybride.

Abstract

Many real problems reduce to the maximization of a profit under a budget or a capacity constraint. This kind of problems are called knapsack problems and are more and more interesting praticians and researchers. Knapsack problems are combinatorial optimization problems that can be described as follows. Given a fixed capacity, and a set of items each caracterized by a profit and a weight, the objective is to select a maximum-profit subset of items whose total weight does not exceed the given capacity. There exist many variants of the knapsack problem that take origin in many real applications of a variety of domains such that computer science (cloud computing, data mining, cryptography), telecommunications, logistics, and energy.

In this thesis, we study two variants of the knapsack problem, *i.e.*, the Disjunctively Constrained Knapsak Problem (DCKP), and the Multidimensional Knapsack Problem (MKP).

For the DCKP, we first give four valid formulations for the problem. The three first ones are Integer Linear Programs, and the last one is a quadratic formulation. In this thesis, we are intrested in the so-called standard formulation. We study the polytope associated with this formulation and describe several classes of valid inequalities. We give necessary and sufficient conditions for these inequalities to be facet defining. We also discuss separation routines for these inequalities and then develop a branch-and-cut algorithm based on these results.
Along with this exact method of resolution, we develop a probabilistic tabu search heuristic with multiple neighborhood structures.
Both methods perform very well for an efficient resolution for the DCKP problem.

Concerning the MKP, we develop a hybrid method to solve the problem. This method is a linear-relaxation-based metaheuristic combined with a technique of solutions'

space reduction. We also present experimental results showing the performance of our approach.

Key words : Disjunctively Constrained Knapsack Problem, Multidimensional Knapsack Problem, polytope, branch-and-cut algorithm, probabilistic tabu search, hybrid approach.

Table des matières

Introduction

Contexte

Cette thèse s'inscrit dans le cadre de l'optimisation combinatoire, une des branches de la recherche opérationnelle et de l'informatique. Elle prote sur des variantes de l'un des problèmes classiques de l'optimisation combinatoire qui est le problème du sac à dos. Un problème de l'optimisation combinatoire consiste à trouver une meilleure solution dans un ensemble fini de solutions. Un tel problème doit se ramener toujours à maximiser ou minimiser une fonction linéaire sous des contraintes linéaires. L'objectif de l'optimisation combinatoire, n'est pas seulement de trouver une solution qui satisfait les contraintes, mais plus encore de trouver la meilleure solution possible. Toutefois, le temps d'exécution des méthodes de résolution des problèmes d'optimisation combinatoire, qui sont généralement difficiles, peuvent augmenter exponentiellement avec la taille du problème. Dans ce cas il est nécessaire de faire le choix entre trouver la meilleure solution et on parle ainsi de méthodes exactes, ou de se contenter d'une solution approchée et on parle ainsi de méthodes heuristiques.

De nos jours, l'optimisation combinatoire a connu un développement important aussi bien sur le plan théorique qu'au niveau des applications. En effet, l'étude des modèles théoriques reste d'actualité vu qu'elle permet aux décideurs d'avoir un ensemble de méthodes de résolution efficaces qu'ils peuvent adapter à leurs problèmes particuliers.

Le domaine de l'optimisation combinatoire comprend un nombre considérable de problèmes qui sont souvent très difficiles à résoudre. Ces problèmes sont très répandus dans les situations réelles et pratiques telles que le data mining, les télécommunications, la logistique, l'allocation des ressources, la conception de réseaux ...
L'un des problèmes classiques de l'optimisation combinatoire est celui du sac à dos

(knapsack problem).

Problématique

Le problème du sac à dos n'est rien d'autre que celui du remplissage d'un sac à dos d'un randonneur dont le poids ou le volume est limité. Cette limitation est représentée mathématiquement par des contraintes. L'intérêt du randonneur est de maximiser le profit des objets à emporter sans dépasser le volume du sac.
Chaque objet présente un profit pour le randonneur mais aussi une ou plusieurs valeurs liées aux contraintes. Dans le cas où les contraintes sont multiples, on parle du sac à dos multidimensionnel. Et dans le cas de l'existence d'objets incompatibles, on parle du sac à dos avec contraintes disjonctives.
Le problème du sac à dos est très étudié de nos jours. Il permet de modéliser des problèmes de chargement de cargos, de gestion de stocks, de sélection d'objets, de budgets, etc... Les premières études remontent à l'année 1896 dans l'article de George Ballard Mathews. [84].

La résolution de ce type de problème, tout en respectant les contraintes, avec des méthodes exactes s'avère difficile voire parfois impossible dans le cas des grandes dimensions. Cependant, plusieurs méthodes nouvelles ont vu le jour.

— Des méthodes de résolution exactes du problème du sac à dos telles que les méthodes de «Branch and Bound» et «Branch and Cut» ont été proposées.
— Des méthodes dites heuristiques ou approchées ont attiré l'attention comme des alternatives possibles ou des compléments aux approches plus classiques. Elles donnent des valeurs proches de l'optimum recherché.
— Plusieurs méthodes hybrides ont récemment été proposées pour résoudre les problèmes de type sac à dos. Certaines d'entre elles sont basées sur l'exploration complète de voisinages de petites tailles.

Le but de cette thèse est de concevoir des méthodes de résolution qui intègrent les meilleures caractéristiques des métaheuristiques et celles des techniques d'optimisation exacte pour résoudre efficacement des variantes du problème du sac à dos. Les approches proposées sont ensuite validées par une étude expérimentale approfondie.

Contributions

Dans ce travail de thèse, nous traitons deux variantes du problème du sac à dos, *i.e.*, le problème de sac à dos à contraintes disjonctives, et le problème de sac à dos multidimensionnel. Pour le premier, nous présentons une étude polyédrale poussée et développons une méthode de résolution exacte, à savoir un algorithme de coupe et branchement, pour résoudre le problème. Ceci a fait l'objet d'un article dans les proceedings de la conférence CODIT (2016) [12] et d'une publication dans *"Journal of Soft Computing"* [13]. Une communication sur cette approche est aussi présentée à la conférence CO (2016) (International Symposium on Combinatorial Optimization). Toujours concernant le même problème, nous avons proposé une méthode de résolution heuristique. En effet, nous avons utilisé une généralisation de la métaheuristique de recherche Tabou, à savoir la Recherche Tabou Probabiliste. Les résultats obtenus ont fait l'objet d'une publication [10] dans le Journal *"RAIRO-Operations Reserach"*. Nous nous sommes intéressés aussi au problème du sac à dos multidimensionnel. Pour ce problème, nous présentons une heuristique hybride basée sur la programmation linéaire et la réduction du domaine d'exploration de solutions par hyperplans. Les résultats obtenus ont fait l'objet d'une communication à la conférence *"META'12"* (2012) [11].

Organisation de la thèse

La thèse est structurée en six chapitres.

Dans le chapitre 1, nous présentons quelques notions de bases sur l'optimisation

combinatoire et la programmation mathématique ainsi que sur les méthodes approchées et métaheuristiques. Nous donnons des notions de base en optimisation combinatoire et faisons un bref rappel sur la théorie de complexité et la théorie des graphes.

Dans le chapitre 2, nous commençons par présenter le problème du sac à dos classique, connu aussi sous le nom du sac à dos unidimensionnel. Nous présentons ensuite certaines variantes intéressantes du problème du sac à dos, comme le sac à dos multidimensionnel, le sac à dos avec contraintes disjonctives, etc... Un état de l'art est donné afin d'exposer certains travaux récents et pertinents pour chaque variante du problème du sac à dos. A la fin du chapitre, nous présentons quelques domaines d'applications des variantes du sac à dos.

Dans les chapitres 3, 4 et 5, nous nous intéressons à la résolution du problème du sac à dos avec contraintes disjonctives, the Disjunctively Constrained Knapsack Problem (DCKP).

Dans le chapitre 3, nous nous intéressons à l'aspect théorique du problème DCKP. D'abord, nous présentons différentes formulations en termes de programmes linéaires en nombres entiers. Nous considérons la formulation classique, définissons le polytope associé et étudions l'aspect facial des contraintes de base. Ensuite, nous présentons de nouvelles familles de contraintes valides pour le problème. Nous discutons des conditions nécessaires et des conditions suffisantes sous lesquelles ces inégalités définissent des facettes pour le polytope associé.

En se basant sur ces résultats théoriques, nous développons, dans le chapitre 4, un algorithme de coupe et branchement pour une résolution exacte du problème DCKP. Nous commençons par exposer les algorithmes de séparation pour certaines familles de contraintes valides. Ensuite, nous exposons les résultats expérimentaux que nous avons obtenus par l'exécution de notre algorithme de coupe et branchement.

Dans le chapitre 5, nous nous intéressons à la résolution du problème DCKP par une

méthode approchée. En particulier, nous utilisons la métaheuristique de Recherche Tabou Probabiliste. Il s'agit d'une variante de la Recherche Tabou dans laquelle nous rajoutons une pondération sur les mouvements. Une étude expérimentale sur un ensemble d'instances de la littérature est par la suite présentée.

Dans le chapitre 6, nous traitons une autre variante du problème du sac à dos, à savoir le sac à dos multidimensionnel, Multiple Knapsack Problem (MKP). Afin de résoudre le problème MKP, nous proposons une heuristique basée sur la programmation linéaire. L'heuristique est inspirée de l'heuristique itérative basée sur la programmation linéaire proposée par Wilbaut et Hanafi [55] ainsi que la technique de réduction de l'espace d'exploration des solutions introduit par Vasquez et Hao [104]. Des résultats expérimentaux seront aussi présentés.
Nous finirons ce manuscrit par une conclusion dans laquelle nous présentons aussi les perspectives de ce travail.

Chapitre 1

Notions préliminaires

Dans ce chapitre, nous donnons quelques notions de base sur la programmation mathématique et l'optimisation combinatoire d'une part, et les méthodes approchées et heuristiques d'autre part. Nous commençons par quelques définitions en optimisation combinatoire, et rappelons des notions de base en théorie de complexité et théorie des graphes. Nous présentons brièvement les polyèdres combinatoires et la méthode de coupes et branchements. Nous présentons aussi des notions reliées aux méthodes approchées et métaheuristiques. Nous donnons enfin des définitions et des notations qui seront utilisées tout au long de ce mémoire.

1.1 Programmation mathématique

1.1.1 Optimisation combinatoire

L'*Optimisation Combinatoire* est une science émanant de l'informatique et des mathématiques appliquées. Elle consiste à chercher un élément de poids maximum ou minimum dans un ensemble fini. L'optimisation combinatoire traite les problèmes pouvant se formuler comme suit : soit $E = \{e_1, \ldots, e_n\}$ un ensemble fini appelé *ensemble de base*, où chaque élément e_i possède un *poids* $c(e_i)$. Soit $\mathcal{F}$ une famille de sous-ensembles de E. Pour F un élément de la famille $\mathcal{F}$, $c(F) = \sum_{e_i \in F} c(e_i)$ désigne le poids de F. L'objectif de l'optimisation combinatoire, n'est pas seulement de trouver une solution

F qui satisfait certaines contraintes, mais plus encore de trouver la meilleure solution possible, c'est à dire trouver un élément de $\mathcal{F}$, ayant le plus petit (ou le plus grand) poids.
Il s'agit alors d'un *problème d'optimisation combinatoire* dont $\mathcal{F}$ représente l'*ensemble des solutions* du problème. Cet ensemble peut avoir un nombre exponentiel d'éléments.
Le mot *optimisation* veut dire que l'on recherche le meilleur élément de l'ensemble de solutions.
Le domaine de l'optimisation combinatoire comprend un nombre considérable de problèmes qui sont souvent très difficiles à résoudre. Ces problèmes sont très répandus dans les situations réelles et pratiques telles que le data mining, le big data, les télécommunication, la logistique, l'allocation des ressources, la conception de réseaux etc.
L'optimisation combinatoire se trouve au carrefour de la théorie des graphes, de la programmation linéaire et de la programmation linéaire en nombres entiers. Elle est également très liée à la théorie de la complexité d'algorithmes.

1.1.2 Rappels de la théorie de la complexité

L'analyse des complexités temporelles des algorithmes permet d'obtenir des bornes supérieures sur le temps maximum nécessaire pour exécuter l'algorithme. Ces bornes sont, en général, en fonction de la taille de l'instance du problème à résoudre.
La théorie de la complexité a été initiée par les travaux d'Edmonds [26] et de Cook [20]. Elle permet de déduire si un problème donné est facile ou difficile.
Un *problème* est une question générale dont les paramètres ont une valeur inconnue. Chaque problème est défini par une description de tous les paramètres ainsi qu'une énumération des conditions que la solution doit satisfaire.
Une *instance* d'un problème est obtenue en accordant une valeur à chaque paramètre du problème.
Un *algorithme* permettant la résolution d'un problème donné est une procédure qui

peut être décomposée en opérations élémentaires offrant une solution pour chaque instance du problème.
La *taille* d'un problème représente le nombre de données nécessaires pour une instance. Un algorithme est dit *polynomial* s'il est capable de résoudre une instance par un nombre d'opérations borné par une fonction polynomiale en n (n étant la taille de l'instance).
La *classe de complexité P* décrit l'ensemble des problèmes pouvant être résolus par un algorithme polynomial. Les problèmes de la classe P sont dits *faciles*.
Un *problème de décision* est un problème admettant deux réponses possibles : oui ou non.
La *classe de complexité NP* (Nondeterministic Polynomial) est l'ensemble des problèmes de décision dont une réponse au problème peut être vérifiée en temps polynomial. Il est évident que la classe P est contenue dans la classe NP.
La *classe NP-complet* est basée sur la notion de réduction polynomiale. Un problème de décision P_1 se réduit polynomialement en un problème de décision P_2 s'il existe une fonction polynomiale f telle que, pour toute instance I de P_1, la réponse est oui si et seulement si la réponse de $f(I)$ pour P_2 est oui. Nous notons alors $P_1 \alpha P_2$. Un problème $\mathcal{P}$ est *NP-complet*, s'il appartient à la classe NP et s'il existe un problème Q connu comme étant NP-complet tel que $Q \alpha P$.
On peut associer un problème de décision à tout problème d'optimisation, celui-ci est dit *NP-difficile* si le problème de décision associé est NP-complet.

Pour de plus amples détails sur la NP-complétude, le lecteur peut se référer au livre de Garey et Johnson [85].

1.1.3 Notions de la théorie des graphes

Un *graphe non orienté* est noté $G = (V, E)$ où V est l'ensemble des *sommets* et E l'ensemble des *arêtes*. Si $e \in E$ est une arête qui relie les deux sommets u et v, alors

u et v sont dits les *extrémités* de e. Nous écrirons $e = uv$ ou $e = \{u, v\}$. Si u est une extrémité de e, alors u (resp. e) est dit *incident* à e (resp. u). De même, deux sommets u et v formant une arête sont dits *adjacents*.

Un *graphe orienté* est noté $D = (V, A)$ où V est l'ensemble des sommets et A l'ensemble des *arcs*. Si $a \in A$ est un arc partant du sommet u au sommet v, alors u et v seront appelés respectivement le *sommet origine* et le *sommet destination* de a. Nous noterons $a = (u, v)$.

Un graphe est dit *simple* s'il s'agit d'un graphe sans boucle de telle façon qu'entre deux sommets il y a au plus une arête. Un graphe *complet* est un graphe simple, dont tous les sommets sont adjacents les uns aux autres.

Le *degré* $d_G(v_i)$ d'un sommet $v_i \in V$ est égal au nombre d'arêtes dont ce sommet est une extrémité. Dans un graphe orienté, un sommet $v_i \in V$ est caractérisé par deux demi-degrés : le *demi-degré extérieur* $d_G^+(v_i)$ correspondants aux arcs sortant du sommet v_i et le *demi-degré intérieur* $d_G^-(v_i)$ qui désigne le nombre d'arcs y entrant. $d_G(v_i) = d_G^+(v_i) + d_G^-(v_i)$.

Considérons un graphe non orienté $G = (V, E)$ et soit $F \subseteq E$ un sous-ensemble d'arêtes. $V(F)$ représente l'ensemble des extrémités des arêtes de F. De même, si $W \subseteq V$ est un sous-ensemble de sommets, alors $E(W)$ dénote l'ensemble des arêtes ayant leurs deux extrémités dans W.

Soit $W \subseteq V$, $H = (W, E(W))$ est dit sous-graphe de G *induit* par W et sera noté par $G(W)$.

Soient u et v deux sommets de V. Une *chaîne* (resp. *un chemin*) P entre u et v (resp. allant de u à v) est une séquence alternant sommets et arêtes (resp. arcs), *i.e.*, $(v_0, e_1, v_1, e_2, v_2, \ldots, v_{k-1}, e_k, v_k)$ (resp. $(v_0, a_1, v_1, a_2, v_2, \ldots, v_{k-1}, a_k, v_k)$) où $v_0 = u$, $v_k = v$, $e_i = v_{i-1}v_i$ (resp. $a_i = (v_{i-1}, v_i)$) pour $i = 1, \ldots, k$ et $v_0, \ldots, v_k$ sont des sommets distincts de V. P peut également être désignée par sa séquence d'arêtes $(e_1, \ldots, e_p)$ (resp. sa séquence d'arcs $(a_1, \ldots, a_p)$) ou sa séquence de sommets $(v_1, \ldots, v_{p+1})$.

Un *cycle* est un chemin $v_0, v_1, ..., v_k$ tel que $v_0 = v_k$.

Si $F \subset E$, on notera par $G \backslash F$ le graphe obtenu à partir de G en supprimant les arêtes

de F. Si F est réduit à une seule arête e, nous écrirons $G\backslash e$ au lieu de $G\backslash\{e\}$.

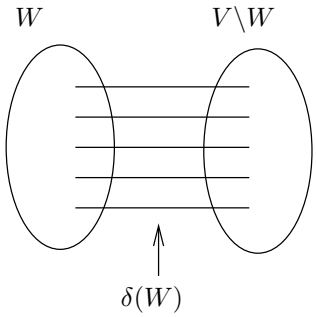

FIGURE 1.1 – Une coupe $\delta(W)$

Soit $W \subseteq V$, $W \neq \emptyset$, un sous-ensemble de sommets de V. L'ensemble des arêtes ayant une extrémité dans W et l'autre dans $V\backslash W$ est appelé *coupe* et noté $\delta(W)$ (voir Figure 1.1). En posant $\overline{W} = V\backslash W$, nous avons $\delta(W) = \delta(\overline{W})$. Si W est réduit à un seul sommet v, nous écrirons $\delta(v)$ au lieu de $\delta(\{v\})$.

Etant donnés W et W' deux sous-ensembles disjoints de V, alors $\delta_G(W, W')$ représente l'ensemble des arêtes de G qui ont une extrémité dans W et l'autre dans W'. Lorsque W et W' sont réduits à un seul sommet, on notera $[u, u']$ à la place de $\delta_G(\{u\}, \{u'\})$.

Si $V_1, \ldots, V_p$, $(p \geq 2)$ est une *partition* de V, alors $\delta(V_1, \ldots, V_p)$ représente l'ensemble des arêtes ayant leurs extrémités dans des éléments différents de la partition (*i.e.*, $e = uv$ telles que $u \in V_i$, $v \in V_j$ et $i \neq j$).

Un ensemble de sommets ou d'arêtes S dans un graphe est dit *maximal* respectant la propriété P, s'il vérifie la propriété P et s'il n'existe pas d'ensemble contenant S et vérifiant cette propriété. L'ensemble S est dit *maximum* respectant la propriété P si, parmi tous les sous-ensembles de sommets ou d'arêtes de G respectant la propriété P, S est celui qui a la cardinalité la plus grande. Les termes *minimal* et *minimum* sont définis de manières analogues.

Étant donné un graphe $G = (V, E)$, un *stable* dans G est un ensemble de sommets deux à deux non adjacents. Le *problème du stable de cardinalité maximum* consiste à trouver un stable dont le nombre de sommets est maximum [95]. La cardinalité du stable maximum est notée par $\alpha(G)$.

Un graphe est dit *biparti* s'il existe une partition de son ensemble de sommets en deux sous-ensembles U et V telle que chaque arête ait une extrémité dans U et l'autre dans V. En d'autres termes, un graphe est biparti si l'on peut regrouper les sommets en deux groupes distincts sans aucun lien entre les sommets de chaque groupe.
Une *clique* de G est un sous-ensemble de sommets $W \subset V$ dont le sous-graphe induit $G(W)$ est complet. Une clique est dite *maximale* si son cardinal est le plus grand.
Soit un ensemble limité $V = \{v_1, ..., v_n\}$, un *hypergraphe* H sur V est une famille $\mathscr{E} = \{E_1, E_2, ..., E_m\}$ de sous-ensembles de V tels que

$$E_i \neq \emptyset \ pourtout \ i = 1, ..., m$$
$$\bigcup_{i=1}^{m} E_i = V$$

On notera l'hypergraphe par $H = (V, \mathscr{E})$. Les éléments $v_1, ..., v_n$ de V sont appelés sommets, et les ensembles $E_1, ..., E_m$ sont appelés *hyperarêtes*. L'hypergraphe $H = (V, \mathscr{E})$ est dit *simple* si aucune hyperarête n'est strictement contenue dans une autre hyperarête, c'est à dire $E_i \not\subset E_j$ pour tout $E_i, E_j \in \mathscr{E}$. Un graphe sans boucles est un hypergraphe où chaque hyperarête contient exactement deux sommets. Soit un hypergraphe $H = (V, \mathscr{E})$, un *hypercycle* est une séquence alternée de sommets et d'hyperarêtes $(v_1, E_1, v_2, ..., v_k, E_k, v_1)$ avec $v_i, v_{i+1} \in E_i$, pour $i = 1, ..., k$, où les indices sont modulo k.

1.1.4 Approche polyédrale et méthode de coupes et branchements

Pour résoudre un problème d'optimisation combinatoire, on peut penser à une méthode énumérative qui consiste à énumérer toutes les solutions de ce problème, à calculer la valeur de chacune des solutions et à en choisir la meilleure. Cependant, bien

qu'il soit fini, le nombre de solutions peut être important et parfois exponentiel. Cette méthode énumérative peut en conséquence atteindre rapidement ses limites. Ainsi, il s'est avéré nécessaire de développer des techniques qui permettent de résoudre ce genre de problèmes plus efficacement. L'une des méthodes les plus puissantes est l'approche dite *polyédrale*, une approche introduite par Edmonds en 1965 [26] pour le problème du couplage.

Pour plus de détails sur cette approche, consulter par exemple [95, 76].

1.1.4.1 Définitions

Soit $n \in \mathbb{N}$. $\mathbb{R}^n$ désigne l'ensemble des vecteurs ayant n composantes réelles et $\mathbb{R}_+$ l'ensemble des nombres réels positifs.
Un point $x \in \mathbb{R}^n$ est dit *combinaison linéaire* de $x^1, \ldots, x^m$ de $\mathbb{R}^n$ s'il existe $\lambda_1, \ldots, \lambda_m \in \mathbb{R}$ tels que $x = \sum_{i=1}^{m} \lambda_i x^i$.
De plus :
Si $\sum_{i=1}^{m} \lambda_i = 1$, x est dit *combinaison affine* des points $x^1, \ldots, x^m$.
Si $\lambda_i \in \mathbb{R}_+$ pour i=1,...,m et $\sum_{i=1}^{m} \lambda_i = 1$, x est dit *combinaison convexe* des points $x^1, \ldots, x^m$.

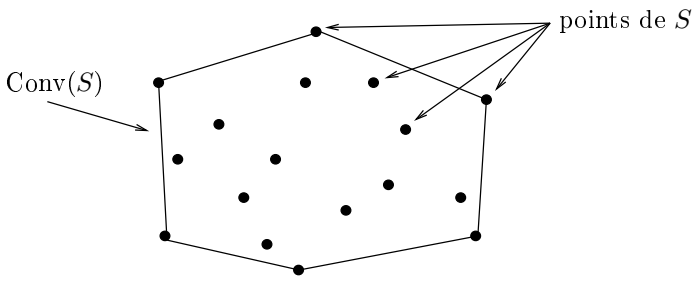

FIGURE 1.2 – Enveloppe convexe

Étant donné un ensemble S de points $x^1, \ldots, x^m \in \mathbb{R}^n$. On désigne par :
conv(S) = $\{x \in \mathbb{R}^n \mid x$ combinaison convexe de $x^1, \ldots, x^m\}$ l'*enveloppe convexe* (voir Figure 1.2).

aff(S) $=$ $\{x \in I\!R^n \mid x$ combinaison affine de $x^1, \ldots, x^m\}$ l'*enveloppe affine* de $x^1, \ldots, x^m$.

Des points $x^1, \ldots, x^m \in I\!R^n$ sont *linéairement indépendants* (resp. *affinement indépendants*) si le système

$$\sum_{i=1}^{m} \lambda_i x^i = 0$$

$$(\text{resp. } \sum_{i=1}^{m} \lambda_i x^i = 0 \text{ et } \sum_{i=1}^{m} \lambda_i = 0)$$

admet une solution unique $\lambda_i = 0$ pour $i = 1, \ldots, m$.

Si $0 \notin$ aff(S) alors les points $x^1, \ldots, x^m$ sont affinement indépendants si et seulement s'ils sont linéairement indépendants.

On appelle *polyèdre* P l'ensemble des solutions d'un système fini d'inégalités linéaires c'est à dire $P = \{x \in I\!R^n \mid Ax \leq b\}$, où A est une matrice de taille $m \times n$ et b un vecteur à m composantes.

Un *polytope* est un polyèdre borné. C'est à dire un polyèdre $P \subseteq I\!R^n$ est un polytope s'il existe $x^1, x^2 \in I\!R^n$ tel que $x^1 < x < x^2$ pour tout $x \in P$.

Si le nombre maximum de points de P de $I\!R^n$ affinement indépendants est $p+1$, on dit que le polyèdre P est de *dimension* p, et on note $dim(P) = p$. Un polyèdre P de $I\!R^n$ est *de pleine dimension* si $dim(P) = n$. Considérons un polyèdre $P \subseteq I\!R^n$ décrit par le système suivant :

$$P = \left\{ x \in I\!R^n : \begin{array}{ll} A_i x \leq b_i, & i = 1, \ldots, m_1 \\ B_j x = d_j, & j = 1, \ldots, m_2 \end{array} \right\}$$

Une inégalité $ax \leqslant \alpha$ est dite *valide* pour un polyèdre P de $I\!R^n$ si elle est vérifiée pour toute solution de P.

Soit $x^* \in I\!R^n$. Une inégalité $ax \leq \alpha$ est dite *violée* par x^* si $ax^* > \alpha$. (voir Figure 1.3).

Si $ax \leq \alpha$ est une inégalité valide de P, le sous-ensemble $F = \{x \in P \mid ax = \alpha\}$ est appelé *face* de P *définie* par $ax \leqslant \alpha$. Remarquons que $dim(F) \leqslant dim(P)$.

Une face F est dite *propre* si $F \neq P$ et $F \neq \emptyset$. Une face propre F est une *facette* de P

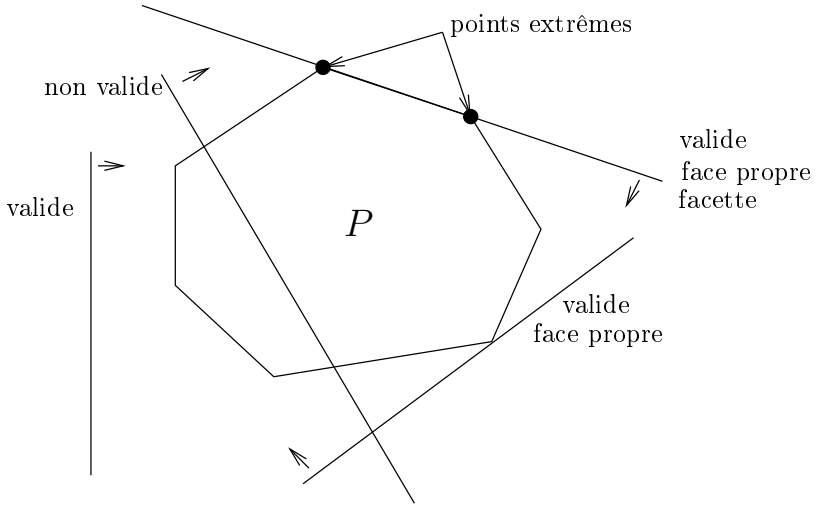

FIGURE 1.3 – Contraintes valides, faces et facettes

si $dim(F) = dim(P) - 1$.

Un *point extrême* d'un polyèdre P est une face de P de dimension 0. Il est facile de voir qu'un point $x \in \mathbb{R}^n$ est un point extrême d'un polyèdre P, s'il ne peut pas être écrit comme combinaison convexe d'autres points de P.

1.1.4.2 Approche polyédrale

L'approche polyédrale consiste à ramener la résolution d'un problème donné à la résolution d'un programme linéaire en décrivant l'enveloppe convexe de ses solutions par des inégalités linéaires. La méthode a été montrée puissante et très efficace pour la résolution à l'optimum de problèmes difficiles d'optimisation combinatoire.

Soient $\mathcal{P}$ un problème d'optimisation combinatoire, $\mathcal{S}$ l'ensemble des solutions de $\mathcal{P}$, E l'ensemble de base de $\mathcal{P}$ et c la fonction poids associée aux variables du problème. Le problème $\mathcal{P}$ s'écrit donc $max\{cx \mid x \in \mathcal{S}\}$.

Si T est un sous-ensemble de E, le vecteur $x^T \in \mathbb{R}^E$ (ayant $|E|$ composantes associées

aux éléments de E) tel que

$$x^T(e) = \begin{cases} 1 & \text{si } e \in T, \\ 0 & \text{sinon,} \end{cases}$$

est appelé *vecteur d'incidence* de T.

Le polyèdre

$$P(\mathcal{S}) = conv\{x^S \mid S \in \mathcal{S}\}$$

est appelé *polyèdre des solutions* de $\mathcal{P}$ (ou *polyèdre associé* à $\mathcal{P}$).

Le problème $\mathcal{P}$ est par conséquent équivalent au programme linéaire $max\{cx \mid x \in P(\mathcal{S})\}$. $P(\mathcal{S})$ peut ainsi être caractérisé par un ensemble de contraintes linéaires où chaque contrainte définit une facette. Si on peut décrire entièrement le polyèdre $P(\mathcal{S})$ par un système d'inégalités linéaires, le problème $\mathcal{P}$ se ramène donc à la résolution d'un programme linéaire.

Une étude approfondie du polyèdre associé au problème est nécessaire pour cette transformation. Notons qu'une caractérisation complète de ce polyèdre est en général difficile à établir. Cependant, l'utilisation de la *méthode de coupes et branchements* (Branch and Cut method), permet de garantir une description partielle du polyèdre pouvant être suffisante pour résoudre efficacement le problème à l'optimum. Cette méthode combine la méthode de génération de nouvelles contraintes (Cutting plane method) et la méthode de *séparations et évaluations* (Branch and Bound method).

1.1.4.3 Méthode de coupes et branchements

La méthode de coupes et branchements pour un problème d'optimisation combinatoire est basée sur le problème dit *de séparation*. Soit P un polyèdre dans $\mathbb{R}^n$. Le *problème de séparation* associé à P consiste à vérifier pour un point $x^* \in \mathbb{R}^n$ si x^* appartient à P, ou bien à trouver une contrainte $a^T x \leqslant b$ valide pour P et violée par x^* dans le cas contraire. Dans ce deuxième cas, l'hyperplan $a^T x = b$ sépare P et x^*

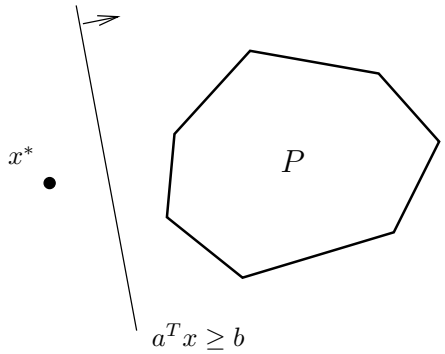

FIGURE 1.4 – Hyperplan séparant x^* et P

(voir Figure 1.4).

L'un des plus importants résultats de l'optimisation combinatoire est la relation étroite entre séparation et optimisation. Grötschel, Lovász et Schrijver [47] ont montré qu'un problème d'optimisation sur un polyèdre P est polynomial si et seulement si le problème de séparation associé à P peut être résolu en temps polynomial.

Considérons un problème d'optimisation combinatoire $\mathcal{P}$ de la forme $max\{cx \mid Ax \leqslant b, x \text{ entier}\}$ et soit P le polyèdre associé à $\mathcal{P}$. Supposons donné un système $\bar{A}x \leq \bar{b}$ de contraintes valides pour $\mathcal{P}$ contenant comme sous-système les contraintes de base du problème. La méthode de coupes et branchements peut être décrite comme suit. On commence par résoudre un programme linéaire $P_1 = max\{cx \mid \bar{A}_1 x \leq \bar{b}_1\}$ où $\bar{A}_1 x \leq \bar{b}_1$ est un sous-système de $\bar{A}x \leq \bar{b}$ ayant un nombre raisonnable de contraintes. Si la solution optimale obtenue pour P_1, disons x_1, est entière et satisfait le système $Ax \leq b$, alors elle est optimale pour $\mathcal{P}$. Sinon, on résout le problème de séparation associé à $\bar{A}x \leq \bar{b}$ et x_1. Si on trouve une contrainte $a_1 x \leq \alpha_1$ violée par x_1, celle-ci est rajoutée au système $\bar{A}_1 x \leq \bar{b}_1$ et nous obtenons un nouveau programme linéaire $P_2 = max\{cx \mid \bar{A}_1 x \leq \bar{b}_1, a_1 x \leq \alpha_1\}$. De nouveau, on résout P_2. Si la solution optimale x_2 de P_2 est entière et vérifie $Ax \leq b$ alors est optimale, sinon, comme pour x_1 on résout le problème de séparation associé à $\bar{A}x \leq \bar{b}$ et x_2. Si $a_2 x \leq \alpha_2$ est une contrainte violée par x_2, elle est rajoutée au système $\bar{A}_2 x \leq \bar{b}_2$ et ainsi de suite. En continuant ce processus appelé *phase de coupe*, on peut trouver, soit une solution optimale pour $\mathcal{P}$,

soit une solution x^*, fractionnaire et pour laquelle on ne peut plus générer de contrainte violée. A ce stade, on procède à une phase dite de *branchement* consistant à construire un arbre de Branch and Bound. On choisit une variable fractionnaire x_i^*. On résout ainsi deux nouveaux programmes linéaires (nouveaux nœuds de l'arbre de Branch and Bound) en ajoutant soit la contrainte $x_i = 0$, soit $x_i = 1$ au programme linéaire obtenu à la fin de la phase de coupes. Pour chacun de ces nouveaux sous-problèmes, on applique de nouveau une procédure de coupes. Si une solution optimale de $\mathcal{P}$ n'a pas été trouvée, on sélectionne une feuille de l'arbre et on recommence une phase de branchement.

L'approche de coupes et branchements est actuellement largement utilisée pour la résolution de problèmes d'optimisation combinatoire difficiles. Généralement, cette méthode de coupes et branchements est plus efficace pour les instances de petites à moyennes tailles. Lorsque les instances des problèmes deviennent assez large, les méthodes approchées (heuristiques ou métaheuristiques) permettent d'avoir des solutions de bonne qualité en un temps de résolution raisonnable. Ces méthodes sont décrites dans la section suivante.

1.2 Métaheuristiques

Les heuristiques et métaheuristiques jouent un rôle primordial dans l'optimisation combinatoire essentiellement pour de nombreux problèmes pratiques de grande taille. Leur objectif principal est de fournir en temps de calcul raisonnable des solutions de bonne qualité proches de l'optimum.
Les métaheuristiques sont des méthodes approximatives génériques qui peuvent s'appliquer à différents problèmes. Elles sont généralement définies par des algorithmes stochastiques itératifs, qui partent d'une solution initiale et progressent vers une solution optimale. Elles se comportent comme des algorithmes de recherche tentant d'apprendre les caractéristiques d'un problème afin d'en trouver une approximation de la meilleure

solution.

On peut distinguer les métaheuristiques mono-solution telles que la recherche tabou, le recuit simulé, la recherche à voisinages variables, et les métaheuristiques multi-solutions qui génèrent une population de solutions initiales telles que l'algorithme génétique et la recherche dispersée. Nous présentons par la suite certaines approches métaheuristiques assez connues. Pour plus de détails voir par exemple [38, 25, 44].

1.2.1 Métaheuristiques mono-solution

1.2.1.1 Recherche locale

La recherche locale est considérée comme une stratégie pour explorer l'espace de recherche qui se compose des solutions candidates.

Les algorithmes de recherche locale partent d'une solution initiale x_0 et passent d'une solution x à une autre x' tout en explorant un ensemble de solutions voisines $V(x)$ jusqu'à ce qu'une solution optimale (optimum local) soit trouvée ou que le critère d'arrêt défini soit atteint.

L'ensemble des solutions $V(x)$ est obtenu en appliquant les changements à l composantes de la solution x. La taille du voisinage varie selon le choix de la valeur de l. En effet, si l est petit, chaque solution ne peut avoir qu'un nombre réduit de solutions voisines. Par contre si l est plus grand, chaque solution aura plus de solutions voisines. En outre, la croissance du nombre de solutions voisines est exponentielle par rapport à l.

La taille du voisinage doit être choisie judicieusement. En effet, un algorithme ayant une taille de voisinage assez grande pourrait perdre en temps d'exécution ce qu'il pourrait gagner en qualité de la solution voisine. En d'autres termes, il prend du temps pour choisir la prochaine solution qui aura plus de chance d'être bonne.

En général, la recherche locale trouve une solution localement optimale, mais pas obligatoirement globalement optimale. Si une solution x est meilleure que toute solution

$x' \in V(x)$, elle peut être un optimum local si x est sur un plateau, ou si x est au sommet d'un pic qui n'est pas le plus haut.

1.2.1.2 Recherche tabou

La recherche avec tabou a été initalement introduite par Fred Glover en 1986 [41]. Elle est devenue par la suite très utilisée, grâce au succès qu'elle a connu pour résoudre de nombreux problèmes. La recherche tabou est une stratégie d'une recherche locale conçue pour échapper aux optima locaux. Son principe est de choisir à chaque itération la meilleure solution x' dans le voisinage $V(x)$, même si x' n'est pas meilleure que x. Avec ce choix on risque de revenir immédiatement à la solution x. Pour éviter ce risque, on crée une liste T, appelée *liste taboue*, qui mémorise les dernières solutions visitées afin d'empêcher le déplacement de nouveau vers ces solutions. La liste T est une mémoire à court terme (*i.e.*, les solutions ne demeurent dans T que pour un nombre limité d'itérations). Si une solution s est dans T on dit que s est une *solution taboue*. De même, tout mouvement qui amène de la solution courante à une solution de T est appelé *mouvement tabou*.
En pratique, la mémorisation de plusieurs solutions dans T demande habituellement beaucoup de place mémoire. De plus, il peut être difficile de vérifier si une solution donnée est dans T. Par conséquent, il est en général préférable de mémoriser seulement des attributs d'une solution ou des modifications apportées.
Deux mécanismes intéressants nommés *intensification* et *diversification* sont mis en oeuvre pour améliorer l'algorithme de la recherche tabou. Ces mécanismes utilisent une mémoire à long terme. L'intensification explore les zones prometteuses de l'espace de recherche tandis que la diversification consiste à diriger la recherche vers des solutions non visitées.

1.2.1.3 Recuit simulé

En métallurgie, on alterne, pour minimiser l'énergie, des cycles de refroidissement lent et de réchauffage. Par exemple, pour renforcer la résistance de la pièce à fabriquer et éviter la possibilité d'avoir des trous vides à l'intérieur du métal, le refroidissement de l'acier se fait lentement.
En 1983 et en s'inspirant de cette procédure, une métaheuristique dite *recuit simulé* a été mise au point par S. Kirkpatrick, et ses collègues [72].
Les algorithmes de la méthode recuit simulé font partie de la famille de recherche locale dont le principe est de diversifier la recherche en autorisant des mouvements moins bons en fonction d'une probabilité d'acceptation. Celle-ci décroît avec le temps pour éviter le blocage dans un optimum local. Elle est donnée par $p(\Delta f, T) = e^{-\frac{\Delta f}{T}}$, où Δf est la variation de la fonction objectif et T est un paramètre désignant la *température*.
Cette métaheuristique converge vers une solution optimale du problème sous certaines conditions. Toute modification permet de changer l'état du système et entraîne une variation de l'énergie qui est calculée à partir de la fonction à optimiser. Si la variation est négative, elle sera appliquée à la solution courante puisqu'elle baisse l'énergie du système. Si elle est positive, elle pourrait également être acceptée selon une certaine probabilité. L'idée est qu'on refroidit lentement et de temps en temps on réchauffe, c'est à dire on peut accepter une variation positive contraire à l'objectif. Ainsi, la température consiste un élément important de l'algorithme.
L'inconvénient de l'algorithme du recuit simulé réside dans le choix de manière expérimentale des différents paramètres. Ces paramètres peuvent concerner la température initiale comme ils peuvent concerner la loi de décroissance de la température ou les critères d'arrêt.

1.2.1.4 Recherche à voisinage variable

En 1997 Mladenovic et Hansen [86] proposent une nouvelle métaheuristique dite *recherche à voisinages variables* (Variable Neighborhood Search :VNS) utilisant plusieurs types de voisinages.
Cette méthode est basée sur le changement systématique de voisinage pour éviter le blocage dans un minimum local. Ce changement de voisinage s'opère au cas où l'on n'arrive pas à améliorer la solution courante x.
Soit un ensemble fini de voisinages $E = \{V(1)(x), ..., V(T)(x)\}$, où $V(t)(x)$ est l'ensemble des solutions voisines de x dans le t^{eme} voisinage.
Au cours d'une itération de l'algorithme de la recherche à voisinage variable, on génère une solution x' voisine de la solution courante x dans $V(t)(x)$. Si x' n'est pas meilleure, on change de voisinage pour générer une autre solution x' dans $V(t+1)(x)$. Sinon, x' devient la solution courante et on recommence le processus en partant de la première structure de voisinage $V(1)(x)$.

1.2.2 Métaheuristiques multi-solutions

Dans la section précédente, nous avons présenté quelques métaheuristiques partant d'une seule solution initiale et cherchant à l'améliorer. Dans cette section, nous décrivons quelques métaheuristiques qui se basent sur une population de solutions initiales.

1.2.2.1 Algorithme génétique

Le nom de cette méthode provient du processus de sélection naturelle fondé sur la lutte pour la vie et où les individus les plus adaptés tendent à survivre plus longtemps et à se reproduire plus facilement gagnant ainsi la compétition de la reproduction tandis que les moins adaptés meurent avant la reproduction.

En génétique humaine, la *combinaison* connue en Anglais par Crossing-Over consiste au croisement de 2 chromosomes et à l'échange de portions d'ADN donnant lieu à de nouveaux chromosomes. Dans le cas naturel, au sein d'un chromosome, un gène peut prendre la place d'un autre, il s'agit du phénomène de mutation.
En 1975, Holland [65] a proposé un algorithme qui se base sur la sélection naturelle et qui utilise des opérateurs inspirés de la génétique, *i.e.*, le *croisement* et la *mutation*, pour faire évoluer une population de chromosomes vers une nouvelle population.
Un chromosome ou individu qui se compose de gènes représente une solution possible du problème à résoudre. Une population est un ensemble d'individus. La technique de l'algorithme génétique part d'une population initiale dont le contenu peut être généré de manière aléatoire ou par un générateur adéquat.
Pour une itération t de l'algorithme, on commence par sélectionner certains individus de la population $P(t)$ (les parents). Ensuite, on procède à la transformation de quelques uns avec les opérateurs génétiques pour générer une nouvelle population $P(t+1)$ (les enfants). On s'attend à ce que les individus de cette nouvelle population soient meilleurs que ceux de la population de la génération précédente. Chaque individu est évalué par une fonction d'évaluation couramment connue sous l'appellation de *force* ou *fitness*. Cette fonction sert à guider la sélection des bons individus.

1.2.2.2 Recherche dispersée

Comme les algorithmes génétiques, la recherche dispersée est une métaheuristique évolutive travaillant sur une population de solutions initiales.
La recherche dispersée, ou « scatter search » en anglais, a été proposée par Glover en 1977 [40]. Elle est basée sur les cinq méthodes suivantes :

1) Une méthode de génération qui consiste à produire une population de solutions.
2) Une méthode d'amélioration ayant pour but de transformer une solution initiale en une ou plusieurs solutions améliorées.

3) Une méthode de mise à jour dont l'objectif est de construire et maintenir un sous-ensemble de la population appelé "référence" qui contient les meilleures solutions trouvées (le terme meilleur ne correspond pas à la seule valeur de l'objectif du problème mais intègre aussi la notion de diversité).
4) Une méthode de génération d'un sous-ensemble qui consiste à sélectionner un sous-ensemble de l'ensemble de référence afin de créer des solutions combinées.
5) Une méthode de combinaison de solutions dont l'objectif est de transformer le sous-ensemble de solutions produit par la méthode de génération précédente, en un ou plusieurs nouvelles solutions.

La méthode de recherche dispersée débute par produire une population initiale à partir de laquelle un ensemble de solutions "référence" (de petite taille) contenant les solutions de bonne qualité et diverses sera choisi.
De plus, la combinaison des solutions se fait sur un ensemble sélectionné de l'ensemble référence. Des nouvelles solutions se déduisent, seront améliorées et ajoutées à la population référence afin de la mettre à jour. Ce processus est itératif.
La recherche dispersée est alors basée sur l'obtention d'informations concernant la ou les solutions optimales à partir d'un ensemble de solutions élites diverses puis le choix d'une ou plusieurs heuristiques dans la combinaison des solutions pour générer des solutions dans un espace voisin. Le choix de deux solutions (ou plus) dans les combinaisons permet d'exploiter l'information contenue dans l'ensemble de ces solutions.

1.2.2.3 Algorithme de colonies de fourmis

Cette métaheuristique s'inspire du comportement des fourmis pour trouver de bonnes solutions à des problèmes d'optimisation combinatoire. Elle a été proposée par Dorigo, Maniezzo et Colorni [19, 24]. Son principe repose sur la façon avec laquelle les fourmis cherchent leur nourriture et retrouvent leur chemin pour retourner dans la fourmilière. Au début, la fourmi explore le pourtour du nid de façon aléatoire. Dés qu'une source

de nourriture est repérée, son intérêt est évalué selon la quantité et la qualité puis la fourmi ramène un peu de nourriture au nid. Durant le chemin de retour, elle dépose une quantité de phéromone, trace chimique qu'elle arrive à détecter, dépendant de l'intérêt de la source de la nourriture. Puisque toutes les fourmis feront la même chose, les quantités de phéromone laissées augmentent plus rapidement pour les sources de nourriture proches de la fourmilière et lorsque plusieurs traces mènent à la même source, les traces correspondant aux chemins les plus rapides menant aux sources de nourriture les plus importantes sont marquées par de fortes traces. De cette manière les fourmis arrivent à optimiser leurs déplacements.
Pour transposer ce comportement à un algorithme général d'optimisation combinatoire on fait une analogie entre l'aire dans laquelle les fourmis cherchent la nourriture et l'ensemble des solutions admissibles du problème, entre la quantité ou la qualité de la nourriture et la fonction objectif à optimiser et enfin entre les traces et une mémoire adaptative.

1.3 Approches hybrides

On parle d'hybridation quand la métaheuristique considérée est composée de plus d'une méthode se répartissant les tâches de recherche. Ces méthodes hybridées peuvent être déterministes ou approchées ou les deux selon les cas. Cependant il existe plusieurs types d'hybridations possibles dont ce qui suit :

— on peut utiliser une recherche locale dans les méthodes évolutives. Ce genre de recherche réduit le risque de passer à côté d'une solution optimale sans la concevoir et l'enregistrer. La méthode évolutive détecte les bonnes régions dans l'espace de recherche et la recherche locale explore efficacement les régions prometteuses.
— On peut exécuter simultanément diverses métaheuristiques, voire même plusieurs fois la même métaheuristique mais avec divers paramètres.
— On combine les métaheuristiques avec des méthodes exactes.

Toutes ces formes d'hybridation et d'autres sont étudiées sur des exemples concrets et chacune des méthodes présente des avantages et des inconvénients selon les circonstances de l'exemple traité ou selon les conditions de travail. La qualité de la solution recherchée (les dimensions de l'espace de recherche, les dimensions des contraintes à respecter) et le temps d'exécution sont des paramètres majeurs pour le choix des méthodes d'exécution.

Chapitre 2

Le problème du sac à dos : variantes, état de l'art et applications

---

Sous le terme du sac à dos sont regroupées différentes variantes de problèmes classiques d'optimisation appartenant à la classe de problèmes non déterministes polynomiaux. Nous présentons dans ce chapitre deux variantes du problème du sac à dos, le problème du sac à dos multidimensionnel (MKP : Multidimensional Knapsack Problem) et le problème du sac à dos avec contraintes disjonctives (DCKP : disjunctively constrained knapsack problem) qui seront l'objet de nos études dans cette thèse. Nous abordons une étude bibliographique pour chacune de ces variantes. Nous citons ensuite autres variantes du problème du sac à dos puis nous terminons par quelques domaines d'application de ce problème.

2.1 Sac à dos multidimensionnel

2.1.1 Formulation mathématique

Le problème du sac à dos est l'un des problèmes classiques les plus étudiés dans la littérature. En effet, les premières études de ce problème remontent à l'année 1896 dans l'article de George Ballard Mathews [84]. De plus sa formulation simple et sa complexité

de résolution (parmi les 21 problèmes démontrés NP-difficiles par Karp [70]) ont en fait un problème majeur traité par les méthodes d'optimisation combinatoire.

Les problèmes de type sac à dos sont des problèmes d'optimisation combinatoire semblables à celui du remplissage d'un sac à dos d'un randonneur dont le poids ou le volume est limité. Cette limitation est présentée mathématiquement par une ou plusieurs contraintes de capacité. L'intérêt du randonneur est de maximiser le profit des objets à emporter. Donc sous la contrainte de limitation de capacité, un choix d'objet s'impose.

Chaque instance du problème du sac à dos est caractérisée par n objets $j = 1, \ldots, n$, chaque objet j a un profit p_j, et un poids w_j et le sac a une capacité c. Tout vecteur binaire $x = (x_1, \ldots, x_n)$ est une solution réalisable du problème du sac à dos si elle vérifie la contrainte de capacité donnée par

$$\sum_{j=1}^{n} w_j x_j \leq c. \tag{2.1}$$

Une solution, notée x^*, est optimale si elle est réalisable et si elle maximise la somme des valeurs profits des objets mis dans le sac, on a alors pour toute solution réalisable x :

$$\sum_{j=1}^{n} c_j x_j^* \geq \sum_{j=1}^{n} c_j x_j. \tag{2.2}$$

Ainsi, la formulation du problème du sac à dos unidimensionnel en variable binaire est la suivante :

$$(KP) \begin{cases} \max & z = \sum_{j=1}^{n} p_j x_j \\ s.c. & \sum_{j=1}^{n} w_j x_j \leq c, \\ & x \in \{0,1\}^n, \end{cases} \tag{2.3}$$

où p_j, w_j $(j = 1 \ldots n)$ et c sont des entiers positifs et pour chaque objet j la variable

binaire $x_j = 1$ si j est pris dans le sac à dos, 0 sinon.

Le problème du sac à dos est parmi les premiers problèmes montrés NP-Complets. Parmi les méthodes utilisées pour la résolution du problème du sac à dos standard (unidimensionnel) on cite la méthode gloutonne qui construit une solution de manière incrémentale. L'algorithme consiste à dresser un ordre profit/poids de la manière suivante : $p_1/w_1 \geq p_2/w_2 \geq \ldots p_n/w_n$. Cet ordre sert à choisir des objets en respectant l'ordre et la capacité résiduelle du sac, la solution obtenue par cette méthode est généralement utilisable par les algorithmes approchés comme solution initiale. Dans [97], Shi propose un algorithme de colonie de fourmis pour résoudre le problème du sac à dos. Truong et al. [102], eux, proposent de résoudre le KP par une métaheuristique inspirée de la réaction chimique (CRO : Chemical Reaction Optimization). Les premiers algorithmes de séparation et évaluation pour le KP sont apparus au début des années 1970. Nous mentionnons à titre d'exemple, les algorithmes proposés par Horowitz et Sahni [66], Fayard et Plateau [29], Robert M. Nauss [87] et Martello et Toth [81]. D'autres algorithmes exacts qui reposent sur la programmation dynamique due à Bellman [9] peuvent aussi être cités ([80], [93]). Aussi, Marsten et al. [79] proposent un algorithme combinant la méthode de séparation et évaluation avec la programmation dynamique.
Pour plus de détails sur ce problème, consulter par exemple [71] et [82].

Le problème du sac à dos multidimensionnel (MKP) est un problème du sac à dos avec plusieurs contraintes de capacités (m contraintes). Il est donc une généralisation du problème KP qui correspond au cas ou $m = 1$. Le problème MKP peut être modélisé comme un programme linéaire de la manière suivante :

$$(MKP) \begin{cases} \max & z = \sum_{j=1}^{n} p_j x_j \\ s.c. & \sum_{j=1}^{n} w_{ij} x_j \leq c_i \quad \forall i \in \{1, \ldots, m\}, \\ & x \in \{0,1\}^n, \end{cases} \tag{2.4}$$

où p_j, w_j et c_i sont des entiers positifs, n nombre d'objets et m nombre de contraintes.

Il existe différentes appellations de ce problème autre que sac à dos multidimensionnel par exemple problème du sac à dos multiple ou du sac à dos multi-contraintes.

2.1.2 Méthodes de résolution du MKP

Le problème du sac à dos multidimensionnel a fait l'objet de plusieurs études dans la littérature. Il constitue aussi l'axe principal du sixième chapitre.

Plusieurs algorithmes basés sur la recherche tabou ont été proposés pour la résolution du problème du sac à dos multidimensionnel. Glover et Kochenberger [45] ont présenté un algorithme basé sur la recherche tabou qui utilise une mémoire flexible intégrant des informations de fréquence. Hanafi et Fréville [53] proposent une approche de recherche tabou basée sur la stratégie de l'oscillation et l'information obtenue de la contrainte surrogate. Cet algorithme alterne entre des phases constructives et destructives et permet la visite des solutions non réalisables au cours de la recherche. Dans [107] Wilbaut et al. présentent une méthode hybride appelée intensification globale obtenue par la création de la coopération entre les algorithmes de programmation dynamique et de la recherche tabou.
Dans [104] Vasquez et Hao proposent un algorithme tabou explorant des espaces de recherche réduits $\mathscr{X}_k$ d'une manière contrôlée autour d'un optimum continu $\bar{x}_{[k]}$ de la relaxation linéaire. L'ensemble $\mathscr{X}_k$ correspond à l'intersection de l'hyperplan $H_k = \{x \in \{0,1\}^n | ex = k\}$ avec $e = (1, \ldots, 1)$ et la sphère S de noyau $\bar{x}_{[k]}$ et de rayon bien choisi δ ; $\mathscr{X}_k = \{x \in \{0,1\}^n | Wx \leq c, \quad ex = k, \quad \delta(x_{[k]}, \bar{x}_{[k]}) \leq \delta_{max}\}$ où $\delta(x_{[k]}, \bar{x}_{[k]})$ correspond à la distance entre les solutions $x_{[k]}$ et $\bar{x}_{[k]}$, et δ_{max} est un rayon convenablement choisi et qui permet la disjonction des $\mathscr{X}_k$ de façon à éviter la redondance d'exploration des processus de la recherche tabou.

Dans [54], Wilbaut et Hanafi proposent une méthode basée sur la recherche dispersée

(Scatter Search) pour résoudre le MKP. Ils commencent par générer une population initiale diverse en utilisant un générateur basé sur la relaxation. Ensuite, ils utilisent la recherche locale pour améliorer chaque solution trouvée. L'algorithme proposé combine la recherche tabou et la méthode des chemins reliants avec la recherche dispersée.

Dans [106], Wilbaut et Hanafi proposent une heuristique basée sur la programmation linéaire. Le principe de l'algorithme est de résoudre de manière exacte une série de problèmes réduits générés à partir d'une série de relaxations linéaires.

Alaya, Solnon et Ghédira [2] décrivent un algorithme de colonie de fourmis (Ant colony Optimization). Les auteurs proposent de déposer des traces de la phéromone sur les couples d'objets sélectionnés dans une même solution. Fidanova [31] proposent de déposer de la phéromone sur le chemin formé par la suite d'objets sélectionnés.

Chu et Beasly [18] utilisent un algorithme génétique pour la résolution du problème MKP. Partant d'une population donnée d'individus, les auteurs mettent en évidence le processus de l'algorithme génétique en considérant qu'une solution possible du problème est semblable à un chromosome et que chaque individu de la population solution est un gène. Les auteurs proposent l'appel à une heuristique qui rend toute solution rencontrée non réalisable en une réalisable.

Oliva et al. [91] proposent une méthode hybridant la programmation linéaire (PL) et la programmation par contrainte (PPC). La procédure à suivre est celle de séparation et évaluation : à chaque nœud de l'arbre de séparation et d'évaluation, la relaxation linéaire courante du problème est résolue. Si la borne supérieure est plus petite que la valeur de la meilleure solution alors la contrainte de coût réduit est générée et ajoutée au sous problème courant dans le modèle (PPC). Il se peut que les réductions de domaine induites par la propagation coupent la solution optimale continue de la PL, dans ce cas la relaxation linéaire qui prend en compte les dernières réductions est résolue une fois de plus et le processus peut se répéter jusqu'à ce qu'une solution entière soit trouvée ou bien que la solution optimale de la relaxation linéaire soit encore validée après la

réduction des domaines.

Une autre approche hybride a été présentée par Gallardo et al. dans [35]. Cette approche consiste à mettre en évidence une coopération directe de l'algorithme évolutionnaire et la méthode de séparations et évaluation. Branch and Bound. Les deux méthodes s'exécutent en parallèle en échangeant des informations. Cette exécution parallèle a deux voies :

1) L'algorithme Branch and Bound peut utiliser la borne inférieure calculée par l'algorithme évolutionnaire (AE) pour purger le reste du problème en supprimant les solutions dont la borne supérieure est plus petite que celle obtenue par l'algorithme évolutionnaire.

2) L'algorithme Branch and Bound injecte des informations concernant plusieurs régions prometteuses de l'espace de recherche dans la population de l'algorithme évolutionnaire dans le but de guider sa recherche.

Outre les méthodes heuristiques et métaheuristiques, des algorithmes exacts pour le MKP ont été introduits dans Gavish et Pirkul [37], Martello et Toth, [83] et Shih [98]. Aussi dans [67], Kaparis et Letchford proposent de résoudre le MKP par la méthode de coupes et branchement (Branch and Cut method).

2.2 Sac à dos avec contraintes disjonctives

2.2.1 Description

C'est une variante du problème du sac à dos, où quelques objets sont en conflit avec d'autres. On considère un sac à dos de capacité c, un ensemble $V = \{1, 2, ..., n\}$ d'objets et un ensemble E de paires d'objets en conflit. C'est à dire $E \subseteq \{(i, j) \in V \times V | i < j\}$, et si une paire $(i, j) \in E$ cela signifie que les objets i et j sont incompatibles. En outre, à chaque objet i on associe un profit p_i et un poids w_i. Le problème du sac à dos avec

contraintes disjonctives consiste à déterminer un sous ensemble d'objets compatibles qui maximise la fonction objective et dont la somme des poids ne dépasse pas la capacité du sac. Une formulation compacte de programmation linéaire en nombres entiers utilise un ensemble de variables binaires x_i associée à l'objet $i \in V$ et qui prend la valeur 1 si i est emballé dans le sac à dos, et 0 sinon. La formulation standard du DCKP peut être énoncée de la manière suivante :

$$(DCKP) \begin{cases} \max & z = \sum_{j=1}^{n} p_j x_j \\ s.c. & \sum_{j=1}^{n} w_j x_j \leq c, \\ & x_i + x_j \leq 1 \qquad \forall (i,j) \in E, \\ & x \in \{0,1\}^n. \end{cases} \tag{2.5}$$

Les contraintes de type $x_i + x_j \leq 1$ pour $(i,j) \in E$ représentent les contraintes disjonctives. D'autres formulations du problème DCKP ont été proposées dans la littérature.

2.2.2 Méthodes de résolution du DCKP

Comme mentionné précédemment, le DCKP est un problème d'optimisation NP-difficile. Le premier document traitant le DCKP a été soumis par Yamada et al. en 2002 [111]. Les auteurs présentent une méthode heuristique ainsi qu'un algorithme d'énumération implicite et une méthode de réduction d'intervalle afin de résoudre le problème DCKP à l'optimalité. Une combinaison de toutes ces méthodes permet aux auteurs de résoudre des instances de taille allant jusqu'à $n = 1000$ où n est le nombre d'objets, avec une densité du graphe d'objets incompatibles $\mu \in \{0.001, 0.002, 0.005, 0.01, 0.02\}$.

Dans un autre travail, Senisuka et al. [96] proposent une méthode qui permet de

résoudre le DCKP en utilisant la relaxation Lagrangienne combinée avec un test de "pegging" du problème KP classique. Une borne supérieure est déterminée en utilisant la relaxation Lagrangienne, puis une borne inférieure est obtenue en appliquant une méthode de recherche locale utilisant le voisinage 2-opt. Une approche de "pegging" est ensuite utilisée afin de réduire la taille du problème. Des simulations ont été effectuées sur des instances non corrélées et faiblement corrélées avec $n \in [1000, 16000]$ et une densité $\mu \in \{0.1, 0.2, 0.4\}$.

Dans [58], Hifi et Michrafy proposent un algorithme basé sur une recherche locale réactive. Une solution initiale est calculée en utilisant deux procédures gloutonnes complémentaires. Une procédure de dégradation est ensuite appliquée pour éviter les optima locaux et pour introduire la diversification dans l'espace de recherche. Les auteurs ont opté également pour une liste tabou en vue d'interdire la répétition des configurations. Les auteurs ont utilisé deux types de listes mémoire. La première liste mémoire, appelée "move memory", consiste à stocker un ensemble de mouvements non autorisés. La deuxième liste mémoire, appelée "memory list", consiste à utiliser une liste de valeurs. Les résultats des calculs démontrent les performances des deux versions de l'algorithme par rapport aux résultats obtenus par Cplex. Les auteurs ont testé leur algorithme sur des instances de 500 objets avec une densité variant entre 0.1 et 0.3 et sur des instances de 1000 objets avec une densité $\mu \in \{0.05, 0.07, 0.09\}$.

Plus tard, Hifi et Michrafy [59] proposent plusieurs versions d'un algorithme exact pour le DCKP. Dans la première version, les auteurs appliquent une approche en trois phases : à partir d'une borne inférieure, ils utilisent une procédure de réduction combinée avec un algorithme exact de type séparation et évaluation. La deuxième version s'appuie sur une combinaison de la procédure de réduction et une recherche dichotomique qui vise à accélérer le processus de recherche. Dans la troisième version, les auteurs améliorent l'algorithme précédent en reformulant le problème par un modèle équivalent. Ces algorithmes ont été testés sur des instances avec $n = 1000$, une capacité $c \in [2000, 4000]$, et une densité de conflits comprise entre 0.007 et 0.0016.

Dans d'autres travaux, Hifi et al. [1, 61, 62, 63, 64] élaborent des méthodes heuristiques pour le DCKP.

Dans [1], les auteurs proposent trois versions d'algorithmes basées sur la méthode de branchement local "Local Branching". La première version est une adaptation simple et directe de la méthode de branchement local. La deuxième version est une combinaison de la méthode de branchement local et une procédure d'arrondissement de la solution. La dernière version est une amélioration de la seconde version par l'introduction d'une stratégie de diversification. Les trois algorithmes ont été testés sur un ensemble d'instances de problèmes issus de la littérature et ils se sont avérés efficaces.

Dans [61], Hifi et al. proposent un algorithme à deux phases qui combine une procédure d'arrondissement avec une procédure de réduction. Dans la première phase, la procédure d'arrondissement est utilisée pour fixer un sous ensemble d'objets de la programmation linéaire. Dans la deuxième phase, une méthode exacte de branchement local est mise en œuvre pour résoudre le problème réduit.

Dans [63], Hifi et Otmani proposent une version de la méthode de recherche dispersée (SS : Scatter Search) pour résoudre le DCKP. L'approche est basée sur le premier niveau de SS en utilisant à la fois la phase de démarrage et la phase évolutive. La méthode heuristique est appliquée sur un modèle équivalent au DCKP amélioré avec deux familles d'inégalités valides.
D'autres procédures heuristiques ont été développées ultérieurement.

Récemment, Hifi et al. [64] proposent de résoudre le DCKP par une recherche locale guidée. Les auteurs analysent la structure du problème qui est une association de deux problèmes d'optimisation combinatoire : le problème du stable maximum et le KP classique. Ils proposent une méthode hybride qui couple deux procédures de recherche locale, l'une déterministe et l'autre aléatoire. La recherche locale aléatoire est basée sur un système d'optimisation de colonie de fourmis modifiée. Des expérimentations numériques exhaustives montrent l'efficacité de la méthode avec des instances issues

de la littérature.

Parallèlement aux approches heuristiques, les méthodes exactes ont également été utilisées pour résoudre le DCKP. Dans [14], Bettinelli et al. développent la méthode séparation et évaluation pour résoudre le DCKP. Les auteurs proposent une formulation basée sur les cliques pour le problème avec une relaxation serrée utilisée pendant les phases de branchement et d'évaluation. Ils discutent de plusieurs procédures pour calculer les bornes supérieures ainsi que des stratégies de branchement efficace. L'ensemble est combiné dans quatre algorithmes de séparation et évaluation testés sur des instances inspirées des instances du problème de Bin Packing avec conflits.

2.3 Autres variantes du sac à dos

Outre le KP, le MKP et le DCKP il existe plusieurs autres variantes du problème du sac à dos parmi lesquelles on peut citer les variantes suivantes.

2.3.1 Sac à dos multidimensionnel à choix multiple

Si les objets à choisir sont partitionnés en différentes classes dans lesquelles un objet et un seul doit être sélectionné, on parle du sac à dos multidimensionnel à choix multiples (MMKP : Multidimensional Multiple choice Knapsack Problem). Soit d le nombre de classes, le problème peut être formulé mathématiquement comme suit :

$$(MMKP)\begin{cases} \max \quad z = \sum_{k=1}^{d} \sum_{j \in V_k} p_{kj} x_{kj} \\ s.c. \quad \sum_{k=1}^{d} \sum_{j \in V_k} w_{kj} x_j \leq c, \\ \qquad \sum_{j \in V_k} x_{kj} = 1 \qquad \forall k \in \{1, \ldots, d\}, \\ \qquad x \in \{0,1\}^n, \end{cases} \tag{2.6}$$

avec, V_k est l'ensemble d'objets de la classe k, $k \in \{1, \ldots, d\}$.
Supposons que $|V_k| = n_k$ alors $\sum_{k=1}^{d} n_k = n$, ainsi l'espace de recherche est partitionné en sous ensembles d'objets.

La contrainte $\sum_{j \in V_k} x_{kj} = 1$ rend le choix d'un objet et un seul de chaque sous ensemble obligatoire. Les classes V_k ne sont pas nécessairement de même cardinalité et le cas où $|V_k| = 1$ signifie qu'un objet de l'espace de recherche est trivialement choisi, donc il ne fait pas partie de la résolution du problème.

Diverses approches exactes et heuristiques ont été utilisées dans la résolution du problème du sac à dos à choix multiples.

Dans [94] Sbihi et al. proposent de résoudre le MMKP par la méthode de séparation et évaluation. Les auteurs génèrent une solution réalisable initiale pour obtenir une borne inférieure et résolvent un problème auxiliaire pour déterminer une borne supérieure. Ils utilisent aussi une stratégie de fixation d'objets au cours de l'exploration. La performance de l'algorithme exact est évaluée sur un ensemble d'instances de petite et moyenne taille, certaines d'entre elles sont extraites de la littérature et d'autres sont générées aléatoirement.

Dans [22] Crevits et al. proposent des nouvelles heuristiques itératives basées sur la relaxation. Le principe général de ces heuristiques est basé sur la résolution à chaque itération d'une relaxation du problème qui fournit une borne supérieure et génère un problème réduit. Après avoir résolu ce problème réduit pour améliorer la borne inférieure, des pseudo-coupes sont ajoutées dans le problème courant pour éliminer de l'espace de recherche les solutions déjà explorées ou dominées. Ce processus itératif est enrichi par quelques ingrédients spécifiques pour le MMKP : une recherche locale et une technique de fixation de variables. Ces heuristiques génèrent une séquence de bornes inférieures et supérieures.

En 2006 Hifi et al [60] proposent deux algorithmes approchés pour le MMKP. Le

premier s'appuie sur une recherche guidée qui se résume en 2 phases. Une première phase consiste à faire des inter-changements pour tenter d'améliorer la solution en cours. Une deuxième phase effectue une diversification de la solution en acceptant une éventuelle dégradation de la solution courante.
Le deuxième algorithme est composé de quatre phases. Une première phase consiste à partir d'une solution initiale pour effectuer une recherche locale utilisant la notion d'inter-changement. Une deuxième phase est ensuite appliquée sur chaque solution estimée non améliorante sur le voisinage courant. Une troisième phase consiste à effectuer une diversification. Enfin, une quatrième phase sert à introduire une mémoire afin d'éviter les recyclages sur certaines solutions explorées auparavant.

Parallèlement aux méthodes heuristiques, le MMKP a été résolu en utilisant des méthodes exactes.
Dans [17] Cherfi et al. proposent une technique de génération de colonnes pour le MMKP. Les auteurs présentent une méthode basée sur trois algorithmes. Le premier utilise une technique de génération de colonnes pour résoudre la relaxation linéaire en se basant sur la matrice des contraintes associées au MMKP. Le deuxième utilise une technique d'arrondissement pour rendre une solution réalisable. Et le troisième est un algorithme exact permettant de résoudre des sous problèmes.

2.3.2 Sac à dos quadratique

Le problème du sac à dos quadratique (QKP : Quadratic Knapsack Problem) consiste à considérer que le profit obtenu ne dépend pas seulement de l'objet choisi mais aussi des autres objets choisis. Ce problème, introduit par Gallo et al. [36], on le formule comme suit :

$$(QKP)\begin{cases} \max & z = \sum_{j=1}^{n}\sum_{i=1}^{n} p_{ij}x_i x_j \\ s.c. & \sum_{i=1}^{n} w_j x_j \leq c, \\ & x \in \{0,1\}^n, \end{cases} \tag{2.7}$$

où p_{jj} est le profit si l'objet j est sélectionné, $p_{ij} + p_{ji}$ est le profit si les deux objets i et j sont sélectionnés en même temps. QKP est une généralisation du problème KP, qui se pose lorsque $p_{ij} = 0 \; \forall i \neq j$.

Caprara et al. [16] proposent un algorithme exact en utilisant la méthode de séparation et évaluation pour QKP. Les bornes supérieures sont calculées en tenant compte d'une relaxation Lagrangienne qui est résoluble par un certain nombre de problèmes KP continus.

2.3.3 Sac à dos multiobjectif

Le problème du sac à dos multiobjectif (MOKP : MultiObjective Knapsack Problem) consiste à sélectionner un sous ensemble d'objets optimisant plusieurs objectifs à la fois tout en satisfaisant la contrainte de capacité :

$$(MOKP)\begin{cases} \max & z^k = \sum_{j=1}^{n} p_j^k x_j \qquad \forall k = 1, \ldots, o \\ s.c. & \sum_{j=1}^{n} w_j x_j \leq c, \\ & x \in \{0,1\}^n, \end{cases} \tag{2.8}$$

où o est le nombre d'objectifs, z^k la $k$ième composante de la fonction objectif.

En littérature plusieurs travaux couvrent ce genre de problème. Bazgan et al. [7] proposent une approche basée sur la programmation dynamique. Dans [75] Lukata et al. proposent une résolution du MOKP par la méthode de séparation et évaluation.

Barichard et Hao [6] présentent une heuristique hybride combinant un algorithme génétique et un opérateur de recherche tabou, ce dernier a pour but d'améliorer une configuration réalisable produite par l'opérateur de croisement. L'algorithme a été appliqué sur 9 instances dont $o \in \{2, 3, 4\}$) et $n \in \{250, 500, 750\}$. Les résultats obtenus sont encourageants.

2.4 Quelques applications

Depuis son apparition, le problème du sac à dos a toujours intéressé non seulement les théoriciens, mais aussi les praticiens de part le potentiel d'applications desquelles il prend origine. En effet, le problème du sac à dos, avec ses différentes variantes, émane d'applications intéressantes dans des domaines aussi diverses que l'informatique, la finance, la logistique, le transport, et l'énergie.
Dans cette section, nous détaillons certaines applications intéressantes du problème du sac à dos classique et certaines de ses variantes.

Un problème largement étudié en informatique et plus particulièrement dans le "cloud computing" est le problème de localisation des machines virtuelles dans un cluster de machines physiques. Le "cloud computing" est une technologie de pointe qui ne cesse d'influencer notre comportement informatique au travail et dans nos vies quotidiennes. Son principe est le suivant : des utilisateurs exécutent des tâches sur des machines virtuelles placées dans un cluster de machines physiques de façon à assurer un gain considérable dans les infrastructures initiales. A cause de l'hétérogénéité des différentes tâches, différentes machines virtuelles sur une même machine physique peuvent avoir des temps d'achèvement différents. Aussi, les machines physiques sont généralement hétérogènes. Ceci implique que différents placements de machines virtuelles peuvent à leur tour avoir des temps d'achèvement différents. L'objectif est alors de placer les différentes tâches sur les machines virtuelles, et les différentes machines virtuelles sur les machines physiques de façon à minimiser le temps d'achèvement global. Le problème

n'est autre qu'un MKP. Il peut aussi être assimilé à un Bin Packing selon l'objectif et les contraintes considérées.
Plusieurs travaux se sont intéressés à la résolution de ce problème en se référant à son aspect sac à dos. Nous citons quelques unes des plus récentes références [56, 15, 3].

Le problème du sac à dos a aussi des applications intéressantes en télécommunications. Dans [30], Ferdosian et al. étudient le problème d'allocation des ressources dans les réseaux mobiles LTE (Long Term Evolution) formulé comme un KP. Le problème consiste à déterminer une planification des liaisons descendantes multiservices (downlink multi-service scheduling) dans les systèmes LTE. Il s'agit, pour chaque intervalle de temps de transmission (Transmission Time Interval TTI), d'assurer une planification optimale tout en satisfaisant les contraintes de Qos (Quality Of Service) et les contraintes liées aux longueurs d'onde. Inspiré de la structure du sac à dos, les auteurs dans [30] utilisent une heuristique gloutonne pour résoudre le problème.

Le problème du sac à dos a également des applications en énergie comme dans les "smart grids". Avec l'apparition de la nouvelle tendance des "smart grids" et des maisons intelligentes la volonté d'atteindre une consommation énergétique efficace et réduite s'accroît de jour en jour. Ces maisons intelligentes, souvent connues sous le nom *end-users*, sont équipées de technologies poussées permettant de contrôler l'énergie utilisée. L'objectif est par conséquent d'accompagner les clients dans leur prises de décision afin d'assurer un comportement énergétique optimal. Dans [99], Sianaki et al. utilisent le modèle KP binaire pour modéliser le problème de la gestion optimale des appareils électroménagers pendant les heures de pointe. Les auteurs évoquent aussi la possibilité d'utiliser le problème de KP fractionnaire afin de déterminer pour chaque appareil sa durée maximale de fonctionnement de façon à ne pas excéder le coût énergétique objectif. Dans [74], kumaraguruparan et al. étudient le problème d'ordonnancement de l'utilisation des appareils dans les "smart houses" en tenant compte des prévisions de la consommation d'énergie en heures de pointe. L'objectif est d'assurer une utilisation optimale de l'énergie tout en minimisant la facture énergétique pour les clients. Le pro-

blème est assimilé à un MKP, où les m sac à dos correspondent aux m intervalles de temps pour lesquels le prix d'énergie est fixe et les n objets sont les appareils électroménagers. Le poids de chaque objet n'est autre que l'énergie que l'appareil consomme pour chaque pas de temps, et le profit pour une période de temps donnée est le coût de consommation de puissance par l'appareil. La capacité de chaque sac à dos correspond à la valeur de l'énergie maximale qu'on ne souhaite pas dépasser pour une période de temps.

Outre les smart grids, il existe des applications dans d'autres domaines de l'énergie. Dans [105], Wilbault rapporte en se référant de Yamada et Kataoka [110] une application du DCKP dans le domaine de l'énergie nucléaire. Il s'agit du problème de sélection de sites pour y construire des centrales nucléaires. Supposons donnés un capital noté c et n sites possibles pour installer un ensemble de centrales nucléaires. Chaque site j permet de produire une quantité d'énergie w_j et implique un coût d'installation p_j. Pour des raisons de sécurité et d'environnement, deux sites choisis doivent obligatoirement être à une distance minimale l'un de l'autre. Ceci implique un certain conflit entre les différents sites. L'objectif consiste donc à sélectionner un sous-ensemble de sites de façon à maximiser l'énergie totale produite, sans dépasser le capital disponible, et tel que deux sites en conflit ne sont pas choisis simultanément.

2.5 Conclusion

Nous avons discuté dans ce chapitre de quelques variantes du problème du sac à dos. Nous avons présenté une étude bibliographique approfondie sur les problèmes MKP et DCKP, problèmes auxquels nous nous sommes principalement intéressés dans le cadre de cette thèse. Nous avons aussi décrit un ensemble d'applications existantes pour ce problème.

Chapitre 3

Le problème DCKP : étude polyédrale

Dans ce chapitre, nous nous intéressons à l'aspect polyédral de la variante du problème du sac à dos avec contraintes disjonctives (Disjunctively Constrained Knapsack Problems, DCKP). Nous commençons par présenter différentes formulations du problème en programmes linéaires en nombres entiers. Nous nous intéressons ensuite à la formulation classique. Nous définissons le polytope associé à cette formulation et nous étudions l'aspect facial des contraintes de base. Ensuite, nous présentons quelques inégalités valides et nous donnons des conditions nécessaires et des conditions suffisantes sous lesquelles ces inégalités définissent des facettes.

3.1 Notations

Il convient tout d'abord d'introduire certaines notations qui seront nécessaires pour ce chapitre et le chapitre suivant.
Rappelons que le problème DCKP est une variante du problème du sac à dos, où quelques objets sont en conflit avec d'autres. On considère un sac à dos de capacité c, un ensemble $V = \{1, 2, ..., n\}$ d'objets et un ensemble E de paires d'objets en conflit. C'est à dire $E \subseteq \{(i, j) \in V \times V : i < j\}$, et si une paire $(i, j) \in E$ cela signifie que les objets i et j sont incompatibles. Il est naturel de représenter ces relations de conflit par un graphe de conflit non orienté $G = (V, E)$, où chaque sommet correspond à un objet et où une arête $(i, j) \in E$ indique que les objets i et j sont incompatibles.

Notons par $\mathcal{V}_i$ l'ensemble des objets qui sont en conflit avec l'objet $i \in V$, *i.e.* $\mathcal{V}_i = \{j \in V : (i,j) \in E\}$.
En outre, à chaque objet i, on associe un profit p_i et un poids w_i. Le problème du sac à dos avec contraintes disjonctives consiste à déterminer un sous ensemble d'objets compatibles qui maximise la fonction objective et dont la somme des poids ne dépasse pas la capacité du sac.

3.2 Différentes formulations

Le problème DCKP peut être modélisé en termes de programme linéaire en nombres entiers. Plusieurs formulations que nous présentons dans cette section peuvent être considérées.

3.2.1 Formulation standard

Une formulation naturelle du problème DCKP sous forme d'un programme linéaire en nombres entiers utilise une variable binaire x_i associée à chaque objet $i \in V$. x_i prend la valeur 1 si l'objet i est mis dans le sac, et 0 sinon. Le DCKP est alors équivalent au programme linéaire (PL) $0-1$ suivant :

$$\max \sum_{i \in V} p_i x_i \tag{3.1}$$

$$\sum_{i \in V} w_i x_i \leq c, \tag{3.2}$$

$$x_i + x_j \leq 1 \qquad \text{pour tout } e = (i,j) \in E, \tag{3.3}$$

$$0 \leq x_i \leq 1 \qquad \text{pour tout } i \in V, \tag{3.4}$$

$$x_i \in \{0,1\} \qquad \text{pour tout } i \in V. \tag{3.5}$$

(3.1) représente la fonction objectif. (3.2) est la contrainte de capacité du sac à dos. Les inégalités (3.3) sont appelées contraintes de disjonction (ou contraintes disjonctives). Par ces inégalités , on exprime le fait que deux objets $i \in V$ et $j \in V$ qui sont en conflit (*i.e.*, $(i,j) \in E$) ne peuvent pas être pris simultanément dans le sac. (3.4) et (3.5) sont respectivement les contraintes triviales et d'intégrité des variables.

3.2.2 Formulation Equivalente

Hifi et Michrafy [59] ont proposé une formulation qu'ils ont appelée *"modèle équivalent"* donnée par le PL en $0-1$ suivant :

$$\max \quad \sum_{i \in V} p_i x_i \tag{3.6}$$

$$s.t. \quad \sum_{i \in V} w_i x_i \leq c, \tag{3.7}$$

$$|\mathcal{V}_i| x_i + \sum_{j \in \mathcal{V}_i} x_j \leq |\mathcal{V}_i| \quad \text{pour tout } i \in V, \tag{3.8}$$

$$x_i \in \{0,1\} \quad \text{pour tout } i \in V. \tag{3.9}$$

Hifi et Michrafy [59] ont montré que le PL en $0-1$ (3.6)- (3.9) est équivalent au PL en $0-1$ (3.1)- (3.5). Notons que la différence entre les deux formulations réside dans la façon d'exprimer les contraintes de disjonction. En effet, les contraintes de disjonction (3.8) porposées par Hifi et Michrafy [59] ne sont autre qu'une agrégation de l'ensemble des contraintes de disjonction (3.3). Ceci implique par conséquent que la relaxation linéaire est plus faible que celle de (3.1)- (3.5).

3.2.3 Formulation en Cliques

Contrairement à la section précédente nous présentons dans cette section une formulation du DCKP en PL en $0-1$ ayant une relaxation linéaire plus forte que (3.1)- (3.5).

Pour ce faire, considérons le graphe de conflit G et notons $\mathcal{K}$ la famille de toutes les cliques du graphe de conflit G. Le DCKP est équivalent au PL en $0-1$ suivant, proposé par Bettinelli et al. [14].

$$\max \quad \sum_{i \in V} p_i x_i \tag{3.10}$$

$$s.t. \quad \sum_{i \in V} w_i x_i \leq c, \tag{3.11}$$

$$\sum_{j \in K} x_j \leq 1 \quad \text{pour tout } K \in \mathcal{K}, \tag{3.12}$$

$$x_i \in \{0,1\} \qquad \text{pour tout } i \in V. \tag{3.13}$$

La différence entre les PL en $0-1$ (3.1)- (3.5) et (3.10)- (3.13) est l'expression des contraintes de conflit entre les objets.
Notons que, pour tout $(i,j) \in E$, il existe une clique K telle que $\{i,j\} \subseteq K$ (une arête peut aussi être considérée comme une clique). Par la suite la contrainte exprimant la disjonction entre plusieurs objets mutuellement en conflit, *i.e.* (3.12), est plus forte que la contrainte de disjonction (3.3). Par conséquent le PL en $0-1$ (3.10)- (3.13) a une relaxation linéaire plus forte que la formulation standard (3.1)- (3.5) (voir Malaguti et al. [78]).

3.2.4 Formulation Quadratique

Dans cette section, nous présentons une formulation quadratique, équivalente à la formulation (3.1)- (3.5), donnée par le programme quadratique suivant :

$$\max \sum_{i \in V} p_i x_i - \overline{c} \sum_{(i,j) \in E} x_i x_j \tag{3.14}$$

$$s.t. \quad \sum_{i \in V} w_i x_i \leq c, \tag{3.15}$$

$$x_i \in \{0,1\} \quad \text{pour tout } i \in V, \tag{3.16}$$

où $\overline{c} = \sum_{i \in V} p_i$ représente la somme des profits de tous les objets.

Le programme quadratique (3.14)- (3.16) est basé sur la reformulation des contraintes de disjonction (3.3) sous forme de pénalités quadratiques dans la fonction objectif. En effet, pour avoir une solution optimale de la formulation (3.14)- (3.16), il est clair que le terme quadratique $\overline{c} \sum_{(i,j) \in E} x_i x_j$ doit avoir une valeur minimale. Or, par les contraintes d'intégrité (3.16) nous aurons à l'optimum $\sum_{(i,j) \in E} x_i x_j = 0$.

Remarquons que $\sum_{(i,j) \in E} x_i x_j = 0$ est équivalente à dire que pour tout $(i,j) \in E$, $x_i = 0$ ou $x_j = 0$ ce qui est équivalent à $x_i + x_j \leq 1$ pour tout $(i,j) \in E$ qui n'est autre que les contraintes de disjonction (3.3).

3.3 Etude polyédrale

Nous nous intéressons dans la suite à la formulation (3.1)- (3.5).

Sans perte de généralité, on suppose que l'ensemble d'arêtes E est composé de deux sous-ensembles E_d et E_c, c'est à dire, $E = E_d \cup E_c$, où E_d est l'ensemble d'arêtes représentant les conflits dus aux contraintes de disjonction (3.3), et E_c est l'ensemble d'arêtes représentant des conflits dus à la contrainte de capacité (3.2). En d'autres termes, pour tout $e = (i,j) \in E_c$, $w_i + w_j > c$, c'est à dire i et j forment une couverture (*cover*).

On va également supposer que $w_i < c$ pour tout $i \in V$, c'est à dire que chaque objet

unique est une solution pour le problème.

3.3.1 Polyèdre et dimension

Notons par DCKP(G) le polytope associé au DCKP, c'est à dire l'enveloppe convexe des vecteurs d'incidence de toutes ses solutions,

$$DCKP(G) = conv\Big\{x \in \{0,1\}^n \,|\, x \text{ vérifie } (3.2)\text{-}(3.5)\Big\}.$$

Une solution $S \subseteq V$ du DCKP sera représentée par l'ensemble d'objets retenus pour être rangés dans le sac et qui ne sont pas en conflit. C'est à dire, le vecteur d'incidence de S, x^S tel que $x_i^S = 1$ si $i \in S$ et $x_i^S = 0$ sinon, satisfait les contraintes de DCKP.

Nous donnons d'abord le résultat préliminaire suivant.

Théorème 3.1. *$DCKP(G)$ est de pleine dimension.*

Preuve. Considérons les solutions de DCKP, $S_0, S_1, \ldots, S_n$, définies comme suit, $S_0 = \emptyset$, $S_i = \{i\}$, pour tout $i \in V$. Il est clair que $x^{S_0}, x^{S_1}, \ldots, x^{S_n}$ forment $n+1$ points affinement indépendants de DCKP(G). Par conséquent, $\dim(DCKP(G)) = n$.

□

3.3.2 Etude faciale

Ayant établi la dimension de DCKP(G), nous allons maintenant déterminer quand est ce que les inégalités triviales et les inégalités de disjonction définissent des facettes.

Théorème 3.2. *$x_k \geq 0$ définit une facette de $DCKP(G)$.*

Preuve. Soient $S_0 = \emptyset$, et pour tout $i \in V \setminus \{k\}$, $S_i = \{i\}$. Ces ensembles constituent $n = |V|$ solutions de DCKP dont les vecteurs d'incidence satisfont $x_k = 0$, et sont affinement indépendants. □

Théorème 3.3. *$x_k \leq 1$ définit une facette de DCKP(G) si et seulement si pour tout $i \in V \setminus \{k\}$, $(i,k) \notin E$.*

Preuve. Supposons qu'il existe $i_0 \in V$ tel que $(i_0, k) \in E$, c'est à dire, $x_k + x_{i_0} \leq 1$. Comme cette inégalité domine $x_k \leq 1$, cette dernière ne peut pas définir une facette. Maintenant supposons que $(i,k) \notin E$ pour tout $i \in V \setminus \{k\}$. Considérons les solutions $S_1, \ldots, S_k, \ldots, S_n$ de DCKP définies par $S_k = \{k\}$, et pour tout $i \in V \setminus \{k\}$, $S_i = \{i,k\}$. Celles-ci forment n solutions de DCKP dont les vecteurs d'incidence satisfont $x_k = 1$, et sont affinement indépendants. □

Théorème 3.4. *Pour $(k,m) \in E$, $x_k + x_m \leq 1$ définit une facette de DCKP(G) si et seulement si pour tout $i \in V \setminus \{k,m\}$, i n'est pas en conflit avec au moins l'un des deux objets k et m.*

Preuve. Supposons qu'il existe un objet $i^* \in V$ en conflit avec k et m. Dans ce cas, on peut ranger dans le sac au plus un objet parmi k, m et i^*. Ceci implique que l'inégalité $x_k + x_m + x_{i^*} \leq 1$ est satisfaite pour toute solution du problème. Comme cette inégalité domine $x_k + x_m \leq 1$, cette dernière ne peut pas définir une facette.

Supposons maintenant que pour tout $i \in V \setminus \{k,m\}$, i n'est pas en conflit avec au moins l'un des objets k et m. Soient $U_{k,m} = \{i \in V \setminus \{k,m\} | (i,k) \notin E \text{ et } (i,m) \notin E\}$ l'ensemble des objets qui ne sont pas en conflit ni avec k ni avec m. Soient $U_k = \{i \in V \setminus \{k,m\} | (i,k) \notin E \text{ et } (i,m) \in E\}$ l'ensemble des objets qui ne sont pas en conflit avec k mais en conflit avec m, et $U_m = \{i \in V \setminus \{k,m\} | (i,m) \notin E \text{ et } (i,k) \in E\}$ l'ensemble des objets qui ne sont pas en conflit avec m mais en conflit avec k.

Considérons les solutions $S_k = \{k\}$, $S_m = \{m\}$, $S_i = \{k,i\}$ pour tout $i \in U_k \cup U_{k,m}$, $S_j = \{m,j\}$ pour tout $j \in U_m$. Il est clair que ces ensembles constituent une famille

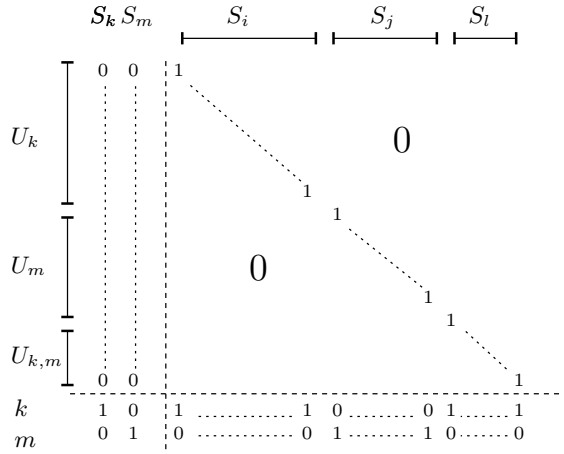

FIGURE 3.1 – Matrice d'incidence des solutions

de n solutions pour le problème, dont les vecteurs d'incidences satisfont l'équation $x_k + x_m = 1$. De plus, comme c'est illustré dans la Figure 3.1, ces vecteurs sont linéairement indépendants.

□

En plus des contraintes de base de la formulation, nous avons identifié de nouvelles familles d'inégalités valides pour DCKP(G). Elles seront présentées dans la section suivante.

3.4 Inégalités valides

Comme mentionné précédemment, le DCKP peut être considéré comme une combinaison de deux problèmes classiques, à savoir le problème du sac à dos et le problème du stable. Donc les inégalités valides pour ces problèmes peuvent être utilisées pour déterminer des familles d'inégalités valides pour le DCKP.

3.4.1 Inégalités de clique

Rappelons qu'étant donné un graphe $G = (V, E)$, une *clique* de G est un sous-ensemble de sommets $K \subset V$ tels que chaque paire de sommets différents forme une arête.
Une clique est dite *maximale* si elle n'est pas strictement contenue dans une autre clique.

Si K est une clique de G, alors tous les sommets de K sont en conflit deux à deux. Ceci implique qu'on peut ranger au plus un élément de K dans le sac. Donc, l'inégalité suivante est valide pour DCKP(G).

$$\sum_{i \in K} x_i \leq 1, \tag{3.17}$$

Théorème 3.5. *Considérons une clique K de G. L'inégalité (3.17) définit une facette de DCKP(G) si et seulement si K est maximale.*

Preuve. Si K n'est pas maximale, alors il y a un élément $j \in V \backslash K$ tel que $K' = K \cup \{j\}$ est une clique. Donc $\sum_{i \in K'} x_i = \sum_{i \in K} x_i + x_j \leq 1$ est valide pour DCKP(G). Comme cette inégalité domine (3.17), cette dernière ne peut pas définir une facette.

Supposons maintenant que K est maximale. Alors pour tout $j \in V \backslash K$, il y a un sommet $j' \in K$ tel que $(j, j') \notin E$. Considérons les ensembles $S_i = \{i\}$ pour tout $i \in K$, et $S_j = \{j, j'\}$ pour tout $j \in V \setminus K$, où $j' \in K$ est un sommet de K non adjacent à j.

Il est clair que ces ensembles sont des solutions de DCKP. De plus leurs vecteurs d'incidence satisfont (3.17) à égalité et sont affinement indépendants. □

3.4.2 Inégalités de couverture

Une *couverture* pour le DCKP est un ensemble C d'objets tel que $\sum_{i \in C} w_i > c$. Une couverture ne peut donc être rangée dans le sac. Par conséquent, les inégalités suivantes sont valides pour le DCKP(G)

$$\sum_{i \in C} x_i \leq |C| - 1 \quad \text{pour tout} \quad C \subset \mathscr{C} \tag{3.18}$$

où $\mathscr{C}$ est l'ensemble des couvertures de DCKP. Les inégalités (3.18) seront appelées *inégalités de couverture*.
Une couverture C est dite *minimale* si

$$\sum_{i \in C \setminus \{j\}} w_i \leq c \text{ pour tout } j \in C.$$

Le théorème suivant caractérise les couvertures qui peuvent induire des facettes pour le DCKP(G). Pour cela, notons que si une couverture n'est pas minimale, alors elle contient un sous-ensemble approprié qui est une couverture. De plus cet ensemble peut être constitué de deux éléments i, j tels que $(i, j) \in E$, c'est à dire i et j sont soit en conflit soit ils forment une couverture.

Théorème 3.6. *Une inégalité de couverture (3.18), induite par une couverture C, définit une facette pour le DCKP(G) si et seulement si*

1) C est minimale,
2) pour tout $j \in V \setminus C$, $w_j < w_{j^*}$, où $j^* \in C$ est tel que $w_{j^*} = max\{w_j | j \in C\}$,
3) chaque objet $j \in V \setminus C$ est en conflit avec au plus un objet de C, et si j est en conflit avec l'objet j' de C, alors $(C \setminus \{j'\}) \cup \{j\}$ n'est pas une couverture.

Preuve. *Nécessité*

1) Si C n'est pas minimale, alors un sous-ensemble approprié $\tilde{C}$ de C est une cou-

verture. Par conséquent, l'inégalité

$$\sum_{i \in \tilde{C}} x_i \leq |\tilde{C}| - 1 \tag{3.19}$$

est valide pour DCKP(G). En sommant l'inégalité (3.19) avec $x_i \leq 1$ pour tout $i \in C \backslash \tilde{C}$ on obtient (3.18). Comme $C \backslash \tilde{C} \neq \emptyset$, (3.18) peut donc être obtenue comme une combinaison linéaire d'inégalités valides et ainsi ne peut définir une facette.

2) Si pour un certain $j \in V \backslash C$, $w_j \geq w_{j^*}$, alors l'inégalité

$$\sum_{i \in C} x_i + x_j \leq |C| - 1 \tag{3.20}$$

est aussi valide pour DCKP(G). En effet, j ne peut pas être rangé avec $|C| - 1$ objets de C, sinon j^* l'aurait été et C ne serait donc pas une couverture. Cependant l'inégalité (3.18) est redondante par rapport à (3.20) et $x_j \geq 0$. Elle ne peut donc pas définir une facette.

3) Si un objet j de $V \backslash C$ est en conflit avec deux objets i_1, i_2 de C, alors toute solution de DCKP contenant j ne peut pas contenir plus de $|C| - 2$ éléments de C. Par conséquent, son vecteur d'incidence ne peut jamais satisfaire l'inégalité (3.18) à l'égalité. Ceci implique que l'inégalité (3.18) est équivalente à $x_j \geq 0$. Comme (3.18) n'est pas un multiple positif de $x_j \geq 0$, elle ne peut définir une facette. De plus, si un objet j de $V \backslash C$ est en conflit avec un objet j' et $(C \backslash \{j'\}) \cup \{j\}$ est une couverture, il en résulte, de la même manière, que l'inégalité (3.18) est équivalente à $x_j \geq 0$, et ne peut donc pas définir une facette.

Suffisance

Supposons que les Conditions 1)-3) sont vérifiées. Notons par $ax \leq \alpha$ l'inégalité (3.18), et supposons qu'il existe une inégalité $bx \leq \beta$ qui définit une facette de DCKP(G) telle

que $\{x \in \text{DCKP}(G) \mid ax = \alpha\} \subseteq \{x \in \text{DCKP}(G) \mid bx = \beta\}$. Nous allons montrer que $b = \rho a$.

Par 1) il s'en suit que tout ensemble $Q \subset C$ tel que $|Q| = |C| - 1$ est une solution de DCKP. Soit $i, j \in C$, et considérons les solutions

$$S_1 = C \backslash \{i\},\ S_2 = C \backslash \{j\}.$$

Comme $ax^{S_1} = ax^{S_2} = \alpha$, nous avons $bx^{S_1} = bx^{S_2}$. Cela donne $b_i = b_j$. Comme i et j sont choisis arbitrairement dans C, il en résulte que tous les b_i sont les mêmes dans C, et donc

$$b_i = \rho \text{ pour tout } i \in C \text{ pour un certain } \rho \in \mathbb{R}. \tag{3.21}$$

Considérons maintenant $j \in V \backslash C$. Par 2) nous avons $w_j < w_{j^*}$, et par 3) j est en conflit avec au plus un élément de C. Supposons, par exemple, que j est en conflit avec un élément, disons j', de C. Considérons l'ensemble $S = (C \backslash \{j'\}) \cup \{j\}$. Par 3), S est une solution de DCKP. De plus, nous avons $ax^S = \alpha$, et donc $bx^S = \beta$. Comme $S \backslash \{j\} = C \backslash \{j'\}$ est aussi une solution du problème et $ax^{S\backslash\{j\}} = \alpha$, nous avons $bx^{S\backslash\{j\}} = \beta$. Mais ceci implique que $b_j = bx^S - bx^{S\backslash\{j\}} = 0$. Si j n'est en conflit avec aucun des objets de C, comme par 2), $w_j < w_{j^*}$, $(C \backslash \{j^*\}) \cup \{j\}$ est une solution de DCKP. Et, d'une manière similaire, il s'en suit que $b_j = 0$. D'où $b_j = 0$ pour tout $j \in V \backslash C$. Ceci avec (3.21) impliquent que $b = \rho a$.

□

Balas [4], Hammer et al. [51] et Wolsey [108] ont remarqué que quand C est une couverture minimale, les inégalités de couverture (3.18) sont les plus efficaces. Dans [4], Balas propose une méthode pour renforcer les inégalités de couvertures. Soit $w^* = \max_{j \in C} w_j$, et considérons l'*extension* $E(C) = C \cup \{j \in V \setminus C | w_j \geq w^*\}$. Alors

l'inégalité

$$\sum_{j \in E(C)} x_j \leq |C| - 1 \tag{3.22}$$

est valide pour le DCKP(G). Les inégalités de type 3.22 sont appelées *inégalités de Couvertures Etendues*.

Balas [4] et Wolsey [108] ont aussi montré que, étant donnée une couverture minimale C, il existe au moins une inégalité de couverture liftée qui définit une facette, et qui est de la forme :

$$\sum_{j \in C} x_j + \sum_{j \in V \setminus C} \alpha_j x_j \leq |C| - 1, \tag{3.23}$$

où $\alpha_j \geq 0$ pour tout $j \in V \setminus C$.

Les inégalités (3.23) peuvent être obtenues par un lifting séquentiel des inégalités (3.22). Cela signifie que les coefficients du lifting $\alpha_j, j \in V \setminus C$ sont calculés un par un dans un ordre donné. Supposons que $V \setminus C = \{j_1, \ldots, j_t\}$, et que $\alpha_{j_1}, \ldots, \alpha_{j_{t-1}}$ sont calculés, autrement dit l'inégalité $\sum_{j \in C} x_j + \sum_{i=1}^{t-1} \alpha_{j_i} x_{j_i} \leq |C| - 1$, est valide pour KP. Afin de déterminer le coefficient α_{j_t}, on peut calculer

$$\xi_t = max \quad \{\sum_{j \in C} x_j + \sum_{i=1}^{t-1} \alpha_{j_i} x_{j_i} \quad | \quad x \quad solution \quad de \quad KP, \quad x_{j_t} = 1\}, \tag{3.24}$$

et soit $\alpha_{j_t} = |C| - 1 - \xi_t$. Les coefficients $\alpha_{j_i}, i = 1, \ldots, t$ dépendent de l'ordre dans lequel ils sont calculés. Comme cela apparaît, le calcul de chaque coefficient α_j nécessite la résolution d'un KP.

Dans [112], Zemel a montré que, étant donné une couverture C fixe et une séquence fixe de lifting, les coefficients de lifting peuvent être calculés en $O(n|C|)$.

Dans [49], Gu et al. se réfèrent aux inégalités (3.23) comme *inégalité de couverture simple liftée*. Ces inégalités ont été généralisées ultérieurement par Van Roy and Wol-

sey [103] qui ont introduit les *inégalités de couvertures liftées généralisées*. Celles-ci sont de la forme

$$\sum_{j \in C \setminus D} x_j + \sum_{j \in V \setminus C} \alpha_j x_j + \sum_{j \in D} \beta_j x_j \leq |C \setminus D| + \sum_{j \in D} \beta_j - 1, \tag{3.25}$$

où C est une couverture, $D \subset C$, $\alpha_j \geq 0$ pour tout $j \in V \setminus C$, et $\beta_j \geq 0$ pour tout $j \in D$.

Comme pour l'inégalité (3.23), l'inégalité (3.25) peut être obtenue à partir de (3.22) par un lifting séquentiel. Pour plus de détails sur la technique de lifting dans l'optimisation combinatoire, on peut se référer à [90].

3.4.3 Inégalités de cycles impairs et d'hypercycle

3.4.3.1 Inégalités de cycles impairs

Un *cycle* dans un graphe est une séquence $v_1, e_1, v_2, ..., v_k, e_k, v_1$ de sommets et d'arêtes telles que $e_i = (v_i, v_{i+1})$, $i = 1, ..., k-1$ et $e_k = (v_k, v_1)$. On notera un cycle C par sa séquence de sommets et on écrit $C = (v_1, ..., v_k)$. Un cycle de k sommets est dit de *longueur* k. Un cycle est dit *pair* (*impair*) si sa longueur est *paire* (*impaire*). Une *corde* d'un cycle est une arête liant deux sommets non consécutifs du cycle. Un cycle est dit *simple* si ses sommets $v_1, ..., v_k$ sont tous différents.
Les cycles impairs induisent des inégalités valides pour le problème du stable et donc pour le DCKP(G). Considérons un cycle C de G où les sommets sont $1, ..., k$ avec k impair. Alors les inégalités

$$x_i + x_{i+1} \leq 1, \text{ pour tout } i = 1, ..., k,$$

où les indices sont modulo k, sont valides pour le DCKP(G). En sommant ces inégalités, divisant par 2 et arrondissant vers le bas le second membre, on obtient l'inégalité :

$$\sum_{i \in C} x_i \leq \frac{k-1}{2}, \tag{3.26}$$

qui est valide pour DCKP(G). Les inégalités de type (3.26) sont appelées *inégalités de cycles impairs*. Les inégalités (3.26) peuvent définir des facettes pour DCKP(G). Une condition nécessaire pour que l'inégalité (3.26) définisse une facette est que le cycle C ne contienne pas de cordes. Si C contient une corde (i_0, j_0), $i_0 < j_0$, alors l'un des cycles $C_1 = (1, ...i_0, j_0, ..., k)$ et $C_2 = (i_0, ..., j_0)$, disons C_1, est impair. Il est clair que (3.26) peut être obtenue comme une combinaison linéaire de l'inégalité de cycle impair induite par C_1 et les inégalités de disjonction $x_{i_0+l} + x_{i_0+l+1} \leq 1$, $l = 1, ..., j_0 - 2$. Les inégalités (3.26) peuvent être renforcées par les inégalités dites de cycles impairs liftées pour qu'elles définissent des facettes pour le polytope du stable [89] et aussi bien DCKP(G).

3.4.3.2 Inégalités d'hypercycle

Dans ce qui suit nous allons introduire une classe plus générale d'inégalités valides pour le DCKP(G). Pour cela, nous allons d'abord donner un exemple.
Considérons le DCKP donné par le système

$$(P_1) \begin{cases} 5x_1 + 4x_2 + 3x_3 + 5x_4 + 2x_5 \leq 11, \\ x_1 + x_4 \leq 1, \\ x_4 + x_5 \leq 1. \end{cases}$$

Remarquons que $C_1 = \{1, 2, 3\}$ et $C_2 = \{2, 3, 4\}$ sont des couvertures. Donc les inégalités suivantes sont valides pour DCKP(G)

$$x_1 + x_2 + x_3 \leq 2,$$

$$x_2 + x_3 + x_4 \leq 2.$$

En sommant ces inégalités avec $x_1 + x_4 \leq 1$, on obtient l'inégalité $2(x_1 + x_2 + x_3 + x_4) \leq 5$. En divisant par 2 et en arrondissant vers le bas le membre de droite, on obtient l'inégalité :

$$x_1 + x_2 + x_3 + x_4 \leq 2, \tag{3.27}$$

qui est donc valide pour DCKP(G). De plus (3.27) définit une facette. En fait, il est facile de voir que les ensembles $S_1 = \{1, 2\}$, $S_2 = \{1, 3\}$, $S_3 = \{2, 3\}$, $S_4 = \{3, 4\}$, $S_5 = \{3, 4, 5\}$ sont des solutions du problème. De plus, leurs vecteurs d'incidence satisfont (3.27) à l'égalité et sont affinement indépendants.

Dans ce qui suit nous allons montrer que ceci est un cas particulier d'une famille plus générale de facettes. Cela sera présentée en utilisant le concept des hypergraphes. Soit un ensemble fini $V = \{v_1, ..., v_n\}$. Un *hypergraphe* H sur V est une famille $\mathscr{E} = \{E_1, E_2, ..., E_m\}$ de sous-ensembles de V tels que

$$E_i \neq \emptyset \text{ pour } i = 1, ..., m,$$

$$\bigcup_{i=1}^{m} E_i = V.$$

Les éléments $v_1, ..., v_n$ de V sont appelés *sommets*, et les ensembles $E_1, ..., E_m$ sont appelés *hyperarêtes*. Un hypergraphe $H = (V, \mathscr{E})$ est dit *simple* si aucune hyperarête n'est strictement contenue dans une autre hyperarête, c'est à dire $E_i \not\subset E_j$ pour tout $E_i, E_j \in \mathscr{E}$. Un graphe sans *boucles* est un hypergraphe où chaque hyperarête contient exactement deux sommets. Etant donné un hypergraphe $H = (V, \mathscr{E})$, un *hypercycle* est une séquence $(v_1, E_1, v_2, ..., v_k, E_k, v_1)$ avec $v_i, v_{i+1} \in E_i$, pour $i = 1, ..., k$, où les indices sont modulo k. Notons que les E_i peuvent ne pas être tous différents. Notons aussi que les E_i peuvent contenir des sommets différents de $v_1, ..., v_k$. Un cycle dans un graphe correspond au cas où $|E_i| = 2$ pour $i = 1, ..., k$.

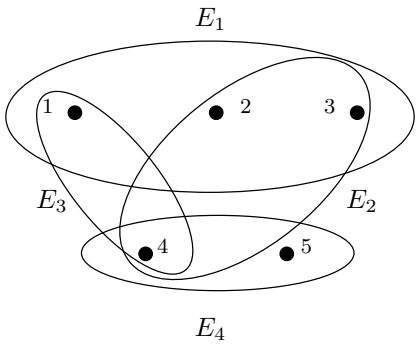

FIGURE 3.2 – L'hypergraphe associé à (P_1)

Considérons maintenant le DCKP avec le graphe de conflits correspondant $G = (V, E)$, et associons au problème l'hypergraphe $H = (V, \mathscr{E})$ où $\mathscr{E}$ est donné par l'ensemble des couvertures minimales du problème ainsi que les ensembles $\{i, j\}$ tels que $(i, j) \in E$, c'est à dire, i et j sont en conflit. Aussi, comme une couverture qui contient deux objets en conflit peut être considérée non minimale, nous supposons sans perte de généralité, qu'aucune hyperarête ne contient une arête de E. En d'autres termes, les hyperarêtes correspondent aux couvertures C telles que $C\backslash\{i\}$ est une solution de DCKP(G) pour tout $i \in C$. L'hypergraphe $H = (V, \mathscr{E})$, associé au problème (P_1) ci dessus, est représentée par la Figure 3.2. Ici $V = \{1, 2, 3, 4, 5\}$ et $\mathscr{E}$ contient les hyperarêtes $E_1 = \{1, 2, 3\}$, $E_2 = \{2, 3, 4\}$, $E_3 = \{1, 4\}$, $E_4 = \{4, 5\}$.

Considérons maintenant un hypercycle de $H = (V, \mathscr{E})$ dont les hyperarêtes sont $E_1, ..., E_k$. Soit $W = \bigcup_{i=1}^{k} E_i$. Soit $W' \subseteq W$ un sous ensemble de W tel que chaque sommet de W' apparaît dans exactement q hyperarêtes parmi $E_1, ..., E_k$. Pour tout $j \in W\backslash W'$, soit ρ_j le nombre d'hyperarêtes parmi $E_1, ..., E_k$ auxquelles j appartient. Supposons que $\rho_j < q$ pour tout $j \in W\backslash W'$ et $\sum_{i=1}^{k}(|E_i| - 1) + \sum_{j \in W\backslash W'}(q - \rho_j)$ n'est pas un multiple de q. Considérons maintenant les inégalités valides suivantes

$$\sum_{j \in E_i} x_j \leq |E_i| - 1 \textit{ pour tout } i = 1, ..., k,$$

$$(q - \rho_j)x_j \leq q - \rho_j \textit{ pour tout } j \in W\backslash W'.$$

En sommant ces inégalités on obtient l'inégalité

$$q(\sum_{i\in W} x_i) \leq \sum_{i=1}^{k}(|E_i|-1) + \sum_{j\in W\setminus W'}(q-\rho_j).$$

Comme le second membre de cette inégalité n'est pas un multiple de q, en divisant par q et en arrondissant vers le bas le membre de droite de l'inégalité résultante, on obtient l'inégalité valide suivante

$$\sum_{i\in W} x_i \leq \left\lfloor \frac{\sum_{i=1}^{k}|E_i| + \sum_{j\in W\setminus W'}(q-\rho_j) - k}{q} \right\rfloor . \tag{3.28}$$

Comme chaque sommet de W' appartient à q hyperarêtes de E_i $(i = 1,\ldots,k)$, et chaque sommet j de $W\setminus W'$ appartient à ρ_j hyperarêtes de E_i $(i = 1,\ldots,k)$ on a $\sum_{i=1}^{k}|E_i| + \sum_{j\in W\setminus W'}(q-\rho_j) = q|W|$.
D'où l'inégalité (3.28) peut être écrite comme suit

$$\sum_{i\in W} x_i \leq |W| - \left\lceil \frac{k}{q} \right\rceil . \tag{3.29}$$

Les inégalités de type (3.29) sont appelées *inégalités d'Hypercycle.*

Remarque 3.7. *Chaque solution T de DCKP dont le vecteur d'incidence satisfait (3.29) à l'égalité est telle que $|W\setminus T| = \left\lceil \frac{k}{q} \right\rceil$ et $W\setminus T$ couvre $\mathscr{E} = \{E_1,...,E_k\}$. Sinon, l'une des couvertures serait incluse dans $W\cap T$, ce qui n'est pas possible.*

Afin d'illustrer les inégalités d'hypercycle, considérons le problème (P_1) donné ci-dessus. Considérons l'hypercycle $(1, E_1, 3, E_2, 4, E_3, 1)$ dont les hyperarêtes sont $E_1 = \{1,2,3\}$, $E_2 = \{2,3,4\}$, $E_3 = \{1,4\}$. Remarquons que chaque sommet i de $E_1\cup E_2\cup E_3$ appartient à exactement deux ensembles parmi E_1, E_2, E_3. Donc ici $W = \{1,2,3,4\}$,

$k = 3$, $q = 2$ et $W' = W$. L'inégalité d'hypercycle correspondante (3.28) n'est rien d'autre que l'inégalité (3.27).
Supposons maintenant que dans (P_1), les objets 2 et 4 sont aussi en conflit. Donc $E_2 = \{2, 3, 4\}$ n'est plus une hyperarête de l'hypergraphe associé (puisqu'il n'est pas minimal), et doit donc être remplacée par $E'_2 = \{2, 4\}$. En considérant l'hypercycle dont les hyperarêtes sont E_1, E'_2 et E_3, tel que $W = \{1, 2, 3, 4\}$, $k = 3$, $q = 2$, $W' = \{1, 2, 4\}$ et $\rho = 1$, on obtient toujours l'inégalité (3.27).

Notons que l'inégalité (3.27) n'est autre qu'une inégalité de couverture étendue (ECI) obtenu à partir de la couverture $\{1, 2, 3\}$. Cependant, dans certains cas, les inégalités d'hypercycle peuvent être différentes des inégalités de couverture étendues et liftées comme indiqué dans l'exemple suivant. Considérons le problème suivant

$$(P_2) \begin{cases} x_1 + 4x_2 + 3x_3 + x_4 + x_5 + x_6 + x_7 \leq 7, \\ x_4 + x_5 \leq 1, \\ x_4 + x_6 \leq 1, \\ x_1 + x_5 \leq 1, \\ x_1 + x_6 \leq 1. \end{cases}$$

Il est clair que les ensembles $E_1 = \{1, 2, 3\}$, $E_2 = \{2, 3, 4\}$, $E_3 = \{2, 3, 5\}$ sont des couvertures minimales pour (P_2). En considérant l'hypercycle dans l'hypergraphe associé, induit par les hyperarêtes E_1, E_2 et E_3 ainsi que les hyperarêtes dues au conflit $E_4 = \{1, 6\}$, $E_5 = \{1, 5\}$, $E_6 = \{4, 5\}$ et $E_7 = \{4, 6\}$, tel que $W = \{1, 2, 3, 4, 5, 6\}$, $k = 7$, $q = 3$, $W' = \{1, 2, 3, 6\}$, $\rho_4 = 1$ et $\rho_6 = 2$, on obtient l'inégalité d'hypercycle

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3.$$

Celle-ci est différente d'une inégalité de couverture étendue ou liftée. De plus cette inégalité définit une facette pour le polytope DCKP(G) associé.

Pour illustrer encore ce concept, considérons le DCKP dont les contraintes sont

$$x_1 + 2x_2 + 3x_3 + x_4 \leq 5,$$

$$x_1 + x_4 \leq 1.$$

En considérant l'hypercycle induit par les hyperarêtes $\{1, 2, 3\}$, $\{2, 3, 4\}$ et $\{1, 4\}$ de l'hypergraphe associé on obtient l'inégalité

$$x_1 + x_2 + x_3 + x_4 \leq 2.$$

qui est valide et définit une facette pour le polytope associé.
Remarquons que les inégalités de cycles impairs (3.26), obtenues à partir du graphe de conflits G, ne sont rien d'autre que les inégalités d'hypercycle quand les hyperarêtes sont toutes des arêtes de G.

Nous avons le résultat suivant qui donne des conditions nécessaires et suffisantes pour obtenir une inégalité d'hypercycle définissant une facette.

Théorème 3.8. *Une inégalité d'hypercycle (3.29) définit une facette de DCKP(G) si et seulement si les conditions suivantes sont vérifiées.*

1) *Pour tout sommet $j \in V \backslash W$, il existe un ensemble $S \subset W$ de $\left\lceil \frac{k}{q} \right\rceil$ objets qui couvrent les hyperarêtes E_1, ..., E_k et tel que $(W \backslash S) \cup \{j\}$ n'est pas une couverture.*
2) *Il existe $|W|$ ensembles S_1, ..., $S_{|W|} \subset W$ qui couvrent E_1, ..., E_k tels que $|S_i| = \left\lceil \frac{k}{q} \right\rceil$, et $T_i = W \backslash S_i$ n'est pas une couverture pour $i = 1, ..., |W|$ et x^{S_1}, ..., $x^{S_{|W|}}$ sont affinement indépendants (Notons que deux objets i, j tels que $(i, j) \in E$ sont considérés comme une couverture et chaque ensemble contenant une couverture est une couverture).*
3) *k n'est pas un multiple de q.*

Preuve. *Nécessité*

1) Supposons que pour un certain $j \in V\backslash W$, chaque ensemble $S \subset W$ de $\left\lceil \frac{k}{q} \right\rceil$ éléments qui couvrent $E_1, ..., E_k$ est tel que $(W\backslash S) \cup \{j\}$ est une couverture. Par la Remarque 3.7, il s'en suit donc que j ne peut appartenir à aucune solution du problème dont le vecteur d'incidence vérifie (3.29) à l'égalité. Ceci implique que (3.29) est équivalente à l'inégalité $x_j \geq 0$. Comme (3.29) n'est pas un multiple positif de $x_j \geq 0$ et DCKP(G) est de pleine dimension, ceci implique que (3.29) ne définit pas une facette.
2) D'abord observons que les vecteurs d'incidence des ensembles S_1, ..., S_l de W avec $|S_i| = \left\lceil \frac{k}{q} \right\rceil$ pour $i = 1, ..., l$ sont affinement indépendants si et seulement si les vecteurs d'incidence des ensembles T_1, ..., T_l où $T_i = W\backslash S_i$, pour $i = 1, ..., l$, sont aussi affinement indépendants. Supposons que l'assertion n'est pas vraie. Par la Remarque 3.7 et l'observation ci dessus, il en résulte qu'il n'existe pas assez de solutions ($|V|$ solutions) du problème dont les vecteurs d'incidence vérifient (3.29) à l'égalité et sont affinement indépendants. D'où (3.29) ne peut pas définir une facette.
3) Si k est un multiple de q, alors (3.29) peut être obtenue comme une combinaison linéaire d'inégalités valides de DCKP(G), et ne peut donc pas définir une facette.

Suffisance :

Supposons que les Conditions 1)-3) sont vérifiées. Notons (3.29) par $ax \leq \alpha$ et soit $bx \leq \beta$ une inégalité qui définit une facette de DCKP(G) telle que $\{x \in \text{DCKP}(G) \mid ax = \alpha\} \subseteq \{x \in \text{DCKP}(G) \mid bx = \beta\}$. Nous allons montrer que $b = \rho a$ pour un certain $\rho \in \mathbb{R}$.

De 2), il s'en suit que les ensembles $T_i = W\backslash S_i$, $i = 1, ..., |W|$, sont des solutions du DCKP. De plus comme x^{S_1}, ..., $x^{S_{|W|}}$ sont affinement indépendants, il en résulte que x^{T_1}, ..., $x^{T_{|W|}}$ le sont aussi. Soit A la matrice carrée dont les colonnes sont x^{T_1}, ..., $x^{T_{|W|}}$. Comme ces vecteurs sont affinement indépendants et sont donc linéairement indépendants car ils ne contiennent pas le vecteur zéro, la matrice A est donc non singulière. De plus, comme $|T_i| = |W| - \lceil \frac{K}{q} \rceil$ pour tout $i = 1, ..., |W|$, il s'en suit que le

système

$$\lambda A = (\beta, ..., \beta),$$

a une unique solution donnée par

$$\lambda_i = \frac{\beta}{|W| - \left\lceil \frac{k}{q} \right\rceil}, \text{ pour } i = 1, ..., |W|.$$

Comme $T_1, ..., T_{|W|}$ sont tels que $ax^{T_i} = \alpha$ pour $i = 1, ..., |W|$, et donc $bx^{T_i} = \beta$ pour $i = 1, ..., |W|$, il s'en suit que $b_i = \frac{\beta}{|W| - \left\lceil \frac{k}{q} \right\rceil} = \rho$ pour $i = 1, ..., |W|$.

Considérons maintenant $j \in V \backslash W$. De 1) il y a $S \subset W$ avec $S = \left\lceil \frac{k}{q} \right\rceil$ qui couvre les hyperarêtes $E_1, ..., E_k$ et tel que $T = (W \backslash S) \cup \{j\}$ n'est pas une couverture. D'où $T \backslash \{j\}$ et T sont des solutions de DCKP. Comme $ax^{T \backslash \{j\}} = ax^T = \alpha$, et donc $bx^{T \backslash \{j\}} = bx^T = \beta$, il s'en suit que $b_j = 0$.

Par conséquent, on a :

$$b_i = \rho \text{ pour tout } i \in W,$$

$$b_i = 0 \text{ pour tout } i \in V \backslash W.$$

Donc $b = \rho a$, et la preuve est complète. □

Dans ce qui suit nous caractérisons un cas particulier dans lequel l'inégalité (3.29) peut définir une facette.

Théorème 3.9. *Considérons un hypercycle $v_1, E_1, v_2, ..., v_k, E_k, v_1$ dans H et supposons que $E_1, ..., E_k$ sont des ensembles distincts, et $v_1, ..., v_k$ sont des sommets distincts. Alors l'inégalité (3.29) définit une facette de DCKP(G) si les conditions suivantes sont vérifiées.*

1) Chaque sommet appartient à au moins deux ensembles parmi $E_1, ..., E_k$ (c'est à dire $q = 2$).

2) k est impair.

3) Pour chaque ensemble $S = \{v, v_{i+2}, ..., v_{i+2l}\}$ où $v \in E_i$ et l est tel que $k = 2l+1$, l'ensemble $W \backslash S$ ne contient pas une couverture. Ici les indices sont modulo k.

4) Pour chaque $j \in V \backslash W$, il y a un ensemble S parmi ceux introduits dans 3) tel que $(W \backslash S) \cup \{j\}$ n'est pas une couverture.

Preuve. Remarquons d'abord que d'après 1), il s'en suit que tout ensemble $S \subset W$ de la forme $\{v, v_{i+2}, ..., v_{i+2l}\}$, avec $v \in E_i$ dont $v \in E_i$,couvre toutes les hyperarêtes $E_1, ..., E_k$. De plus d'après 3), l'ensemble $W \backslash S$ induit une solution de DCKP.
Notons (3.29) par $ax \leq \alpha$ et soit $bx \leq \beta$ une inégalité qui définit une facette telle que $\{x \in \text{DCKP}(G) \mid ax = \alpha\} \subseteq \{x \in \text{DCKP}(G) \mid bx = \beta\}$. Nous allons montrer que $b = \rho a$ pour un certain $\rho \in \mathbb{R}$.
Considérons les ensembles

$$W_1 = W \backslash S_1 \text{ avec } S_1 = \{v_1, v_3, ..., v_k\},$$

$$W_2 = W_1 \backslash S_2 \text{ avec } S_2 = \{v_2, v_3, ..., v_k\}.$$

Ce n'est pas difficile de voir que W_1 et W_2 sont des solutions de DCKP tel que $ax^{W_1} = ax^{W_2} = \alpha$. Donc $bx^{W_1} = bx^{W_2} = \beta$, impliquant que $b_{v_1} = b_{v_2}$. Comme les sommets $v_1, ..., v_k$ jouent le même rôle, par symétrie, il en résulte que

$$b_{v_i} = b_{v_j} = \rho \text{ pour tout } i, j \in \{1, ..., k\}, \tag{3.30}$$

pour un certain $\rho \in \mathbb{R}$.
Considérons maintenant un sommet v de E_1 différent de v_1. L'ensemble $W_3 = (W_1 \backslash \{v\}) \cup \{v_1\}$ est aussi une solution de DCKP avec $ax^{W_3} = \alpha$. d'où $bx^{W_3} = \beta$. Comme $bx^{W_1} = \beta$, ceci implique que $b_v = b_{v_1}$. Comme v est arbitraire dans E_1, il en résulte que $b_v = \rho$ pour tout $v \in E_1$. Et d'après (3.30), on obtient $b_v = \rho$ pour tout $v \in W$.

Pour compléter la preuve, en utilisant 4), on peut montrer de la même manière que dans le Théorème 3.8 que $b_j = 0$ pour tout $j \in V \backslash W$ donc $b = \rho a$. □

Notons que les solutions de DCKP induisent un système indépendant dont les circuits sont précisément les couvertures minimales (voir [95] pour des notions de base sur les systèmes indépendants). Dans [27], Euler et al. introduisent une généralisation des inégalités de cycles impairs pour les systèmes indépendants. Ces inégalités ont une structure différente de celle de (3.29) et peuvent dans certains cas être considérées comme un cas particulier de (3.29).

Dans ce qui suit, on présente de nouvelles familles d'inégalités valides qui combinent les structures du problème du sac à dos et du problème de stable.

3.4.4 Inégalités de couverture-clique

Proposition 3.10. *Soient C une couverture de V et $K \subset V$ une clique. Soit $C^* \subset C$ telle que $|C^*| = |K|$, et supposons que pour tout $i \in C^*$, il existe un objet unique $j \in K$ qui est en conflit avec i.*

$$\sum_{i \in K} x_i + \sum_{j \in C^*} x_j \leq \left\lfloor \frac{|C| + |K|}{2} \right\rfloor, \tag{3.31}$$

est valide pour le DCKP(G).

Preuve. L'inégalité est obtenue par une procédure de Chvátal-Gomory. On a les in-

égalités valides suivantes

$$\begin{aligned}
\sum_{i\in K} x_i &\leq 1, \\
x_i + x_j &\leq 1 \qquad \text{pour tout } i \in C^*, j \in K, \\
\sum_{i\in C} x_i &\leq |C| - 1, \\
-x_l &\leq 0 \qquad \text{pour tout } l \in C \setminus C^*.
\end{aligned}$$

L'inégalité (3.31) est obtenue en additionnant les inégalités ci-dessus, en divisant par 2, et en arrondissant vers le bas le second membre. □

3.4.5 Inégalités de partition couverture-clique

Proposition 3.11. *Considérons une clique $K \subset V$ et soit $K_1, K_2, \ldots, K_r$ (r est pair) une partition de K, c'est à dire, $\cup_i K_i = K$, et $K_i \cap K_j = \emptyset$ pour tout $i \neq j$. Considérons un ensemble d'objets $T \subseteq V$ tel que pour tout $i = 1, 2, \ldots r,$, $T \cup K_i$ est une couverture. Alors l'inégalité*

$$x(K) + \frac{r}{2}x(T) \leq \left\lfloor \frac{r|T| + |K| - r + 1}{2} \right\rfloor, \tag{3.32}$$

est valide pour le DCKP(G).

Preuve. L'inégalité est obtenue par une procédure de Chvátal-Gomory. On a les inégalités valides suivantes

$$\begin{aligned}
\sum_{i\in K} x_i &\leq 1, \\
\sum_{j\in T} x_j + \sum_{j\in K_i} x_j &\leq |T| + |K_i| - 1 \text{pour tout } i = 1, 2, \ldots, r.
\end{aligned}$$

L'inégalité (3.32) est obtenue en additionnant les inégalités ci-dessus, en divisant par 2, et en arrondissant vers le bas le second membre. □

3.5 Conclusion

Dans ce chapitre nous avons décrit certaines familles d'inégalités valides pour le polytope associé au DCKP. Certaines de ces familles dépendent directement de la structure du sac à dos ou du stable. Mais d'autres combinent les deux structures. Comme ce sera présenté dans l'étude expérimentale dans le chapitre suivant, certaines classes parmi ces familles jouent un rôle central pour résoudre le DCKP à l'optimum par un algorithme de coupe et branchement.

Chapitre 4

Le problème DCKP : algorithme de coupe et branchement

Dans ce chapitre, nous présentons un algorithme de coupe et branchement pour résoudre le problème DCKP. Cet algorithme est basé sur les résultats théoriques développés dans le chapitre précédent. Dans un premier temps, nous décrivons une procédure pour générer une solution initiale. Ensuite, nous discutons d'algorithme de séparation. Et enfin nous présentons nos résultats expérimentaux.

4.1 Présentation de l'algorithme de coupe et branchement

Un algorithme de coupe et branchement alterne une phase de coupe et une phase de branchement. Durant la phase de coupe, on génère des inégalités violées en utilisant des algorithmes de séparation. Dans notre algorithme de coupe et branchement, on développe des algorithmes de séparation pour les inégalités valides (3.17), (3.22), (3.23), et (3.26). En fonction de la classe d'inégalités, on développe soit une heuristique soit une procédure exacte de séparation. De plus, en exploitant la relation étroite du problème DCKP avec le problème classique du sac à dos, on développe une heuristique gloutonne qui nous permet d'avoir une solution initiale réalisable pour le DCKP.

4.2 Solution initiale et prétraitement

Afin d'avoir une borne inférieure initiale, on génère une solution réalisable pour le DCKP en utilisant l'heuristique gloutonne donnée dans l'Algorithme 1. L'idée de cette heuristique est la suivante. Nous commençons d'abord par trier les objets $j \in \{1, \ldots, n\}$ dans un ordre décroissant de $\frac{p_j}{w_j+|\mathcal{V}_j|}$ (rappelons que $\mathcal{V}_j$ représente l'ensemble des objets qui sont en conflit avec l'objet j). Notons que le rapport utilisé pour trier les objets réfère à $\frac{p_j}{w_j}$ du problème du sac à dos, mais inclut aussi le degré de l'objet dans le graphe de conflits. En d'autres termes, il est plus intéressant de commencer par des objets ayant le moins de conflits possibles afin de ranger plus d'objets dans le sac à dos. Une fois les objets triés, on met le premier dans le sac et on supprime automatiquement tous ses voisins dans le graphe de conflits. Ce processus continue tant que la capacité du sac n'est pas violée.

Algorithm 1: Une solution initiale utilisant une heuristique gloutonne

Data: Une instance du DCKP
Result: Une solution initiale pour le DCKP

```
1  Soit x un vecteur;
2  for j ∈ {1, ..., n} do
3      x_j ← −1;
4  Δ ← c, F ← V;
   /* Δ : capacité actuelle du sac à dos                                  */
   /* F : ensemble d'objets pouvant être rajoutés à la solution
      actuelle                                                            */
5  while F ≠ ∅ do
6      i ← argmax{p_j/(w_j+|V_j|) | j ∈ F};
7      if w_i ≤ Δ then
8          x_i ← 1, Δ ← Δ − w_i, F ← F \ V_i;
9          for j ∈ V_i do
10             x_j ← 0;
11     else
12         x_i ← 0, F ← F \ {i};
13 return la solution x ;
```

4.3 Séparation des inégalités valides

Soit $\bar{x} \in \mathbb{R}^n$ la solution fractionnaire courante à couper, qui est la solution optimale de la relaxation linéaire du PL donnée par (3.1)- (3.5). Séparer une famille d'inégalités valides consiste à trouver une inégalité ou plus qui soit violée par $\bar{x}$, ou montrer qu'une telle inégalité n'existe pas.

Dans ce qui suit nous allons décrire la séparation des inégalités valides (3.17), (3.18), (3.23) et (3.26).

4.3.1 Séparation des inégalités de Clique

Les inégalités de Clique représentent une famille intéressante d'inégalités valides qui sont faciles à générer. Pour cette raison, on a choisi de générer un ensemble d'inégalités de clique dès la première relaxation linéaire à la racine de l'arbre de coupe et branchement. On résout donc la relaxation linéaire du programme linéaire en nombres entiers donné par (4.1)- (4.5).

$$\max \sum_{i \in V} p_i x_i \tag{4.1}$$

$$\text{s.t.} \quad \sum_{i \in V} w_i x_i \leq c \tag{4.2}$$

$$\sum_{j \in K} x_j \leq 1 \qquad \forall K \in \mathcal{K}, \tag{4.3}$$

$$x_i + x_j \leq 1 \quad \forall (i,j) \in E \,|\, \{i,j\} \nsubseteq K, \forall K \in \mathcal{K}, \tag{4.4}$$

$$x_i \in \{0,1\} \qquad \forall i \in V, \tag{4.5}$$

où $\mathcal{K}$ est une famille de cliques.

Comme, identifier l'ensemble entier des cliques est NP-difficile, on a choisi de générer un ensemble de cliques $\mathcal{K}$ en utilisant une heuristique gloutonne. La procédure sous-jacente est décrite dans l'Algorithme 2. On commence d'abord par trier les objets dans un ordre décroissant de leur degré dans le graphe de conflit, c'est à dire, $\mathcal{V}_j$, $j \in \{1, \ldots, n\}$. Soit $\mathcal{K} = \emptyset$ l'ensemble de cliques que nous cherchons et considérons $K = \emptyset$ qui représente une clique initialement vide. Dans K, on met le premier objet non encore couvert par une clique. Après, on met son premier voisin dans le graphe de conflit. Par la suite, on ajoute tous les objets possibles qui sont universels à des objets dans K, c'est à dire, des objets adjacents à tous les objets de K. A la fin, on obtient une clique K qu'on ajoute à la famille de cliques $\mathcal{K}$.

Algorithm 2: Heuristique de génération de cliques

Data: Une instance du DCKP
Result: Un vecteur de cliques

```
1  Soit 𝒦 ← ∅ l'ensemble des cliques;
2  for i = 1 to n do
3      K ← {i};
4      F ← V \ {i};
5      while F ≠ ∅ do
6          j ← argmax{|E_l| | l ∈ F};
7          if (j, l) ∈ E ∀ l ∈ K then
8              K ← K ⋃ {j};
9          F ← F \ {j};
10     if |K| > 2 et K ∉ 𝒦 then
11         𝒦 ← 𝒦 ⋃ {K};
12 return le vecteur des cliques 𝒦 ;
```

Concernant la phase de séparation, on a choisi d'appliquer, avec quelques minces modifications, une heuristique gloutonne simple présentée par Nemhauser et Sigismondi [88] pour le problème du stable. L'idée de l'heuristique est détaillée dans l'Algorithme 3 et peut être décrite comme suit. On choisit un sommet, disons j, ayant une valeur maximale dans $\bar{x}$, et on pose $K = \{j\}$. Après on répète le processus suivant.

On détermine, s'il existe, un sommet de poids maximum selon $\bar{x}$, disons t, parmi les sommets qui sont universels à K (c'est à dire, t est adjacent à tous les sommets dans K), on ajoute t à K et on répète la procédure jusqu'à ce que l'on ne peut plus trouver de sommet universel. Si $\bar{x}(K) > 1$ alors l'inégalité de clique induite par K est violée. Le processus entier est alors répété pour un autre sommet initial j jusqu'à ce qu'on trouve une inégalité violée ou jusqu'à ce qu'un nombre maximum d'itérations est atteint. Dans notre algorithme, on ne génère, si possible, qu'une seule inégalité de clique violée par itération.

Algorithm 3: Séparation des inégalités de Clique

Data: Solution fractionnaire $\bar{x}$
Result: Une inégalité de clique violée

```
Classer les objets j ∈ V tel que x̄_j ≥ x̄_{j+1};
stop ← faux;
i ← 1;
while i < n et stop=faux do
    K ← {i};
    F ← V \ {i};
    while F ≠ ∅ do
        j ← argmax{x̄_l | l ∈ F};
        if (j, l) ∈ E pour tout l ∈ K then
            K ← K ⋃ {j};
        F ← F \ {j};
    if |K| > 2 et Σ_{j∈K} x̄_j > 1 then
        stop ← vrai;
    i ← i + 1 ;
return l'inégalité de clique violée induite par K ;
```

4.3.2 Séparation des inégalités de cycles impairs

Le problème de séparation des inégalités de cycles impairs peut être résolu d'une manière exacte en un temps polynomial comme il a été montré par Grötschel et al. in [48]. La procédure de séparation est décrite dans [77]. On considère une solution

fractionnaire courante $\bar{x}$. Pour $e = (i, j) \in E$, soit $z_e = 1 - \bar{x}_i - \bar{x}_j$. Comme $\bar{x}$ satisfait (3.3) et (3.4), $z_e \geq 0$ pour tout $e \in E$. Par conséquent, les inégalités (3.26) peuvent être écrites sous la forme :

$$\sum_{e \in C} z_e \geq 1 \qquad \forall \quad C \text{ cycle impair de } G. \tag{4.6}$$

Il s'en suit alors que séparer les inégalités (3.26) par rapport à $\bar{x}$, se ramène à séparer les inégalités (4.6) par rapport à z. On considère un graphe G et soit z_e le poids de l'arête e pour tout $e \in E$. Afin de séparer les inégalités (4.6), il faut chercher un cycle de poids minimum dans un graphe muni de poids non-négatifs sur les arêtes. Ce problème peut être résolu en temps polynomial. Pour ce faire, on considère un graphe biparti $\widetilde{G} = (V' \cup V'', \widetilde{E})$ obtenu à partir de G de la manière suivante : pour chaque sommet $v \in V$, on considère deux sommets $v' \in V'$ et $v'' \in V''$, et pour chaque arête (u, v) on considère deux arêtes $(u', v'') \in \widetilde{E}$ et $(u'', v') \in \widetilde{E}$ avec le même poids $\tilde{z}_{u'v''} = \tilde{z}_{u''v'} = z_{uv}$. Maintenant, chercher un cycle impair de poids minimum dans G par rapport à z, passant par le sommet u, n'est rien d'autre que déterminer une chaîne de poids minimum dans $\widetilde{G}$ par rapport à $\tilde{z}$ entre u' et u''. Comme $\tilde{z} \geq 0$, ceci peut être fait en temps polynomial, en utilisant par exemple l'algorithme de Dijkstra. A la fin de cette étape, nous obtenons (s'il existe) un cycle de poids minimum dans G, appelé C, Si $\sum_{e \in C} z_e < 1$, alors une inégalité de cycle impair violée est détectée. Donc la séparation se ramène à calculer une plus courte chaîne entre chaque paire de sommets u' et u'' dans $\widetilde{G}$. Si le mimium parmi ces chaînes est de poids < 1, alors le cycle correspondant induit une contrainte de cycle impair violée. Sinon, aucune contrainte de cycle impair n'est violée par rapport à $\bar{x}$.

4.3.3 Séparation des inégalités d'hypercycle

Dans ce qui suit, on va décrire une heuristique pour séparer les inégalités d'hypercycle. Notons d'abord qu'une inégalité d'hypercycle (3.29) peut aussi être écrite sous la forme

$$\sum_{i \in W} (1 - x_i) \geq \left\lceil \frac{k}{q} \right\rceil .$$

L'heuristique fonctionne comme suit. Nous générons d'abord un ensemble de couvertures minimales $\{E_1, ..., E_r\}$ où $|E_i| \geq 3$, pour $i = 1, ..., r$. Soit $\mathcal{E}' = \{E_1, ..., E_r, E_{r+1}, ..., E_s\}$, où $E_{r+1}, ..., E_s$ sont toutes les arêtes du graphe de conflit G. Soit $U = \bigcup_{i=1}^{s} E_i$.
On construit ensuite un graphe biparti $\Gamma = (U \cup \mathcal{E}', F)$ dont les sommets à gauche sont les sommets de U, et les sommets à droite correspondent aux éléments de $\mathcal{E}'$. L'ensemble F des arêtes dans Γ est défini comme suit. On considère une arête entre un sommet u de U et un ensemble $E_i \in \mathcal{E}'$ si $u \in E_i$. On associe à chaque E_i le poids $\sum_{j \in E_i} (1 - \bar{x}_j)$. On associe aux sommets de U le poids zéro. Chaque chemin dans Γ entre un sommet i et un ensemble E_k, tel que $i \in E_k$, est un hypercycle dans l'hypergraphe $H = (V, \mathcal{E}')$. Remarquons que comme Γ est biparti chaque chemin alterne entre les sommets de U et les ensembles de $\mathcal{E}'$. L'heuristique calcule un chemin de poids minimum entre i et E_h pour tout E_h, $h = 1, ..., s$ et $u \in E_h$. Chaque chemin calculé induit une inégalité d'hypercycle. nous déterminons le q correspondant. Si le poids minimum de ces chemins est $< \left\lceil \frac{k}{q} \right\rceil$ alors l'inégalité d'hypercycle correspondante est violée par $\bar{x}$.

Dans notre algorithme de coupe et branchement, de telles inégalités n'étaient pas très efficaces dans la résolution. En fait, la séparation des inégalités d'hypercycle prend généralement beaucoup de temps, et dans la majorité des cas on ne réussit pas à trouver une inégalité violée. Pour cette raison, on a choisi de ne pas inclure cette famille d'inégalités valides dans le processus de séparation.

4.3.4 Séparation des inégalités de couverture

On s'intéresse maintenant aux inégalités de couverture. L'algorithme de séparation des inégalités de couverture (3.18) est un problème NP-difficile [73], et c'est probablement aussi le cas pour les inégalités de couverture étendues (3.22). Dans [23], Corwded et al. montrent que le problème de séparation associé aux inégalités de couverture est équivalent au pseudo-problème du sac à dos en 0-1 suivant :

$$\min \sum_{j \in V} (1 - \bar{x}_j) y_j \tag{4.7}$$

$$\sum_{j \in V} w_j y_j > c, \tag{4.8}$$

$$y_j \in \{0, 1\} \qquad \forall j \in V, \tag{4.9}$$

où y_j est une variable binaire égale à 1 si j est inséré dans la couverture C, et 0 sinon. Une inégalité de couverture est donc violée quand la solution optimale, disons y^*, de (4.7)- (4.9) a une valeur objective inférieure à 1. Dans ce cas, la couverture $C = \{j \in V | y_j^* = 1\}$ donne une inégalité de couverture violée. Afin d'accélérer la séparation, Crowded et al. [23] ont proposé une méthode heuristique qui s'exécute en $\mathcal{O}(n\,log(n))$. Celle-ci consiste à insérer des objets dans C dans un ordre non décroissant de $\frac{1-\bar{x}_j}{w_j}$ jusqu'à ce qu'une couverture soit obtenue. Quant aux inégalités de couverture étendues (3.22), Gabrel et Minoux [34] proposent une séparation exacte qui se ramène à la résolution d'une séquence de pseudo-problèmes du sac à dos en 0-1. Pour notre algorithme, on a choisi de séparer ces inégalités en utilisant une heuristique inspiré de celles proposées par Kaparis et al. dans [69]. Cette heuristique est décrite dans l'Algorithme 4. On trie d'abord les objets dans un ordre non décroissant de $\frac{1-\bar{x}_j}{w_j}$, et on les stocke dans une liste L. On initialise aussi la couverture C^* à l'ensemble vide et w^* à la capacité du sac à dos c. On retire ensuite un objet de la tête de la liste triée L. Si son poids est plus grand que w^* alors on l'ignore, sinon on l'insère dans C^*. Si C^* est une couverture alors on étudie l'inégalité de couverture étendue qui lui correspond. Si

l'inégalité de couverture étendue obtenue n'est pas violée, nous supprimons les objets les plus lourds de C^*, on forme une nouvelle inégalité de couverture C^*, et on vérifie à nouveau l'inégalité de couverture étendue correspondante si elle est est violée. Ce processus est alors répété jusqu'à ce qu'on trouve une inégalité ECI violée ou si la liste L est totalement explorée.

Algorithm 4: Séparation des couvertures étendues

Data: Une solution fractionnaire $\bar{x}$
Result: Une inégalité de couverture étendue violée ECI
1 Classer les objets $j \in V$ tel que $\frac{1-\bar{x}_j}{w_j} \leq \frac{1-\bar{x}_{j+1}}{w_{j+1}}$;
2 $L \leftarrow V$ en ordre;
3 $C^* \leftarrow \emptyset$; $w^* \leftarrow c$;
4 $stop \leftarrow faux$;
5 **while** $L \neq \emptyset$ *et* $stop$=$faux$ **do**
6 $i \leftarrow L[0]$;
7 $L \leftarrow L \setminus \{i\}$;
8 **if** $w_i < w^*$ **then**
9 $C^* \leftarrow C^* \bigcup \{i\}$;
10 **if** C^* *est une couverture* **then**
11 **if** *le ECI correspondant à* C^* *est violée* **then**
12 stop $\leftarrow$ vrai;
13 **else**
14 $w^* \leftarrow \max_{j \in C^*} w_j$;
15 $C^* \leftarrow C^* \setminus \{j \in C^* | w_j = w^*\}$;
16 **return** l'inégalité de couverture étendue violée ECI ;

En plus des inégalités de couverture étendues, on sépare également les inégalités de couverture simples liftées (3.23). Pour ce faire, on a développé une procédure décrite dans l'Algorithme 5. On commence d'abord par former une couverture C, on transforme ensuite C en une couverture minimale. On considère ensuite deux ensembles de C, c'est à dire, F et R, correspondant aux sous-ensembles d'éléments de C ayant $\bar{x}_j > 0$ et $\bar{x}_j = 0$, respectivement. Les étapes 10 et 13 de l'algorithme font référence à la phase de lifting pour les deux ensembles F et R. Ceci revient à calculer les coefficients α pour les inégalités (3.23) comme donné dans l'Algorithme 6. Notons que dans l'Algorithme 6,

on doit résoudre un problème du sac à dos. Dans notre cas, on utilise une heuristique gloutonne. On commence d'abord par trier les objets dans un ordre non croissant de $\frac{z_i}{w_i}$, où $z_i = 1$ si l'objet $i \in C$, et $z_i = \alpha_i$ sinon. On met ensuite les objets triés dans le sac tant que les contraintes de capacité correspondantes ne sont pas violées.
Pour une description plus profonde des séparations des inégalités de couverture le lecteur est référé à [67, 68, 69].

Algorithm 5: Séparation des inégalités de couvertures liftées

Data: Une solution fractionnaire $\bar{x}$
Result: Une inégalité de couverture liftée violée
Classer les objets $j \in V$ tel que $\frac{1-\bar{x}_j}{w_j} \leq \frac{1-\bar{x}_{j+1}}{w_{j+1}}$;
$C \leftarrow \emptyset$, $j \leftarrow 1$ et $\Delta \leftarrow c$;
while $j < n$ *et* $\Delta > 0$ **do**
    $C \leftarrow C \bigcup \{j\}$;
    $\Delta \leftarrow \Delta - w_j$;
    $j \leftarrow j + 1$;
supprimer des éléments de C pour qu'elle soit minimale;
Soit $F : \{j \in C | \bar{x}_j > 0\}$;
Soit $R : \{j \in C | \bar{x}_j = 0\}$;
Up-lifting dans F;
if *l'inégalité résultat n'est pas violée* **then**
    stop;
Up-lifting dans R;
return l'inégalité de couverture liftée violée;

En utilisant les algorithmes de séparation présentés précédemment, on développe un algorithme de coupe et branchement. Une étude expérimentale est effectuée sur un ensemble d'instances du problème. Celle-ci sera présentée dans la section suivante.

4.4 Etude expérimentale

L'algorithme de coupe et branchement est implémenté en C++ et testé sur un Bi-Xeon quad-core E5507 2.27GHz avec 8Go de RAM, tournant sous Linux. On utilise CPLEX 12.5 comme solveur linéaire.

Algorithm 6: La procédure du Up-lifting

Data: Une inégalité de couverture
Result: Les Coefficients de lifting $\alpha_j,\ j \in V \setminus C$

Soit $j_1, \ldots, j_r$ un ordre de $V \setminus C$;
$t \leftarrow 1$;
Considérons l'inégalité valide $\sum_{i=1}^{t-1} \alpha_{j_i} x_{j_i} + \sum_{j \in C} x_j \leq |C| - 1$;
`/* Résoudre le problème du sac à dos suivant */`

$$\zeta_t = \begin{cases} \max \sum_{i=1}^{t-1} \alpha_{j_i} x_{j_i} + \sum_{j \in C} x_j, \\ s.t. \sum_{i=1}^{t-1} w_{j_i} x_{j_i} + \sum_{j \in C} w_j x_j \leq c - w_{j_t}, \\ x \in \{0,1\}^{|C|+t-1}. \end{cases}$$

$\alpha_{j_t} \leftarrow |C| - 1 - \zeta_t$;
$t \leftarrow t + 1$;
Arrêter si $t = r$;
return Les coefficients du lifting $\alpha_j,\ j \in V \setminus C$;

4.4.1 Description des instances

Les instances du problème sont générées en utilisant le générateur d'instances de David Pisinger pour le problème du sac à dos classique (utilisé dans [92] et [80], voir `http://www.diku.dk/~pisinger/codes.html`). Certains paramètres sont fixés suivant les instances générées par Hifi et Michrafy dans [59] et Yamada et al. in [111]. Nous avons testé deux sortes d'instances : non corrélées et fortement corrélées. Pour les instances corrélées, les poids et les profits des objets sont aléatoirement générés de 1 à 100. En ce qui concerne les instances fortement corrélées, les poids des objets sont aléatoirement générés de 1 à 100, et chaque profit c_i est égal à $w_i + 10$ pour $i = 1, \ldots, n$ (rappelons que c_i correspond au profit et w_i au poids de l'objet i, respectivement). On a testé trois groupes d'instances de 100 à 1000 objets.

La capacité du sac à dos pour chaque groupe d'instances est calculée en utilisant la formule proposée par David Pisinger :

$$c = \frac{l \times \sum_{j \in V} w_j}{S + 1},$$

où l et S sont des paramètres. Nous fixons S à 1000, et l à 5 et 10, respectivement. Concernant l'aspect disjonctif, les conflits entre les objets sont générés aléatoirement et la densité η du graphe de conflit G prend, comme dans [59] et [111], les valeurs suivantes $\eta = 0.005,\ 0.007,\ 0.009,\ 0.010$, et 0.020. De plus, nous avons fixé un temps limite à 10800 secondes.

4.4.2 Résultats expérimentaux

Afin d'évaluer la performance de notre algorithme de coupe et branchement, on compare nos résultats aux résultats donnés par Cplex pour la formulation originale pour le DCKP, c'est à dire, ILP (3.1)-(3.5), sans prendre en compte les inégalités valides. Notons que nous désactivons également les inégalités valides générées par défaut par l'algorithme de Cplex, afin de pouvoir le comparer avec l'algorithme de coupe et branchement. Rappelons que dans notre algorithme de coupe et branchement nous avons généré un ensemble d'inégalités de clique dès la première relaxation linéaire à la racine de l'arbre et nous résolvons par la suite la relaxation linéaire du programme linéaire en nombres entiers donné par (4.1)- (4.5). Rappelons aussi que nous avons séparé les inégalités valides (3.17), (3.26), (3.22) et (3.23). Après avoir testé différents ordres de la séparation, nous avons choisi de séparer les inégalités valides dans l'ordre suivant. Nous avons d'abord séparé les inégalités de clique (3.17), puis les simples inégalités de couverture liftée (3.23), puis les inégalités de couverture étendue, (3.22) et enfin les cycles impairs (3.26).

Les résultats sont présentés dans les Tables 4.1, 4.2, 4.3 et 4.4 dans l'annexe. Les trois premières colonnes de chaque table représentent les caractéristiques principales de l'instance.

Ensuite les 8 colonnes suivantes représentent les résultats correspondants à l'algorithme de coupe et branchement. Finalement, les 4 colonnes restantes donnent les résultats obtenus par Cplex.

Les autres entrées des Tables sont les suivantes :

n	:	nombre d'objets
m	:	nombres d'arêtes dans G (contraintes de disjonction)
η	:	densité du graphe de conflit
nodes	:	nombre de sommets dans l'arbre de coupe et branchement
Gap-final	:	l'erreur relative entre la meilleure borne inférieure et la meilleure borne supérieure
Gap-root	:	l'erreur relative entre la meilleure borne inférieure et la borne supérieure à la racine
ECI	:	nombre d'inégalités de couverture étendues générées
LCI	:	nombre d'inégalités de couverture liftées générées
Cliques	:	nombre d'inégalités de cliques générées
OddCycles	:	nombre d'inégalités de chemins impairs générées
CPU	:	Temps total d'exécution (en secondes)

La valeur du Gap final est utilisée pour analyser l'efficacité de l'algorithme. Quand cette valeur est égale à 0, cela veut dire que l'instance est résolue à l'optimum. La valeur de Gap-root reflète la qualité de la relaxation linéaire au nœud racine de l'arbre de coupe et branchement avant le branchement comparé à la meilleure borne inférieure obtenue à la fin. Notons également que le nombre de contraintes disjonctives, c'est à dire m, est lié à la taille et à la densité du graphe de conflit à travers la formule suivante $m = \eta \frac{n(n-1)}{2}$.

Au total, on a testé notre algorithme sur 170 instances. Celles-ci peuvent être classées en instances faciles, moyennes et difficiles. La difficulté d'une instance dépend fortement du nombre d'objets n, du nombre de contraintes disjonctives m, et du paramètre l qui influe directement sur la valeur de la capacité du sac à dos (c.f. la formule de capacité est donnée plus haut). Plus les valeurs de n et m sont élevées, plus l'instance est difficile. On note également que les instances avec des capacités calculées avec $l = 10$ sont un

peu plus difficiles que celles avec $l = 5$.

Pour le groupe d'instances faciles, notre algorithme a pu résoudre à l'optimum les instances dans un laps de temps réduit. En fait, pour 61 instances on a atteint l'optimum en moins de 300 secondes. Les instances moyennes ont pris beaucoup plus de temps pour être résolues à l'optimum (8 instances), et les instances difficiles ont atteint la limite de temps sans arriver à une solution optimale. Dans le groupe d'instances difficiles, on a trouvé quelques instances "souples" pour lesquelles on a obtenu un gap final inférieur ou égal à 10% (23 instances). Les instances très difficiles ont atteint cependant d'importantes et même énormes valeurs de gaps finaux, montrant donc la difficulté des instances testées.

En se basant sur les résultats des quatre tables, on peut conclure que notre algorithme de coupe et branchement dépasse les résultats de Cplex pour quasiment toutes les instances. En fait, en appliquant l'algorithme de coupe et branchement, on a été capable de réduire les valeurs de gap pour toutes les instances. Nos gaps à la racine sont toujours meilleurs que ceux de Cplex, ce qui prouve l'efficacité des inégalités valides générées pour avoir serré la relaxation linéaire. Ces inégalités valides sont en fait extrêmement cruciales pour améliorer la résolution des instances difficiles. En fait, pour des instances avec plus de 600 objets, notre algorithme de coupe et branchement fournit des valeurs acceptables de gaps pour lesquelles Cplex a atteint d'énormes valeurs. Notre algorithme a aussi été capable de résoudre à l'optimum des instances que Cplex n'a pas pu résoudre dans la limite de temps. Considérons la Table 4.1 et prenons par exemple l'instance $n = 500$, $\eta = 0.1$ que Cplex n'a pas pu résoudre et qui a eu un gap final égal à 6.84%. La même instance a été résolue à l'optimum par l'algorithme de coupe et branchement après environ 4080 seconds. Même cas pour l'instance $n = 300$, $\eta = 0.2$ et l'instance $n = 400$, $\eta = 0.07$ de la Table 4.2. Nous remarquons aussi que, comparé à Cplex, notre algorithme de coupe et branchement permet une réduction du temps total d'exécution ainsi que la taille de l'arbre (donnée par le nombre de nœuds). A travers les résultats expérimentaux on note également que notre algorithme de coupe et branchement a été

performant, étant donné qu'il était capable de garantir l'optimalité pour 69 instances sur 170 instances qui sont généralement difficiles. Aussi 9 instances ont été résolues à l'optimum au nœud racine de l'arbre de coupe et branchement (5 pour Cplex), et pour 68 instances nous avons un gap à la racine très intéressant ne dépassant pas 10% (47 instances résolues par Cplex ont un gap à la racine inférieur à 10%). Ceci prouve qu'on a réussi à serrer la relaxation linéaire du DCKP en utilisant nos inégalités valides.

Comme c'est mentionné plus haut, on sépare 4 familles d'inégalités valides, c'est à dire, les cliques, les chemins impairs, les couvertures étendues et liftées. En se basant ces expérimentations, on peut voir que le nombre d'inégalités générées de chaque famille dépend du groupe d'instances. Pour presque toutes les instances, nous remarquons que le nombre d'inégalités de chemins impairs générées est le plus important, atteignant plus de 10000 pour certaines instances difficiles. Pour les autres familles d'inégalités, on génère un nombre raisonnable. De plus, comme signalé ci-dessus, le nombre d'inégalités générées de chaque famille d'inégalités valides dépend directement de l'instance. Par exemple, quand les instances sont de petite taille (c'est à dire, concernant n et m), on génère un nombre réduit de cliques et de cycles impairs. Ceci est normal parce que pour de telles instances le graphe de conflit est creux. Plus l'instance est grande, plus on génère de cliques et de cycles impairs étant donné que le graphe de conflit devient plus dense. Pour les deux familles d'inégalités de couverture, c'est à dire, les étendues et les liftées, on remarque que le nombre généré dépend non seulement de la taille de l'instance, mais aussi de son type et de sa capacité (et donc du paramètre l). En général, le nombre d'inégalités de couverture générées est plus important pour les instances fortement corrélées (Tables 4.3 et 4.4) comparées à celles non corrélées (Tables 4.1 et 4.2). Il est clairement plus grand pour des instances avec $l = 5$ que pour celles avec $l = 10$, puisque les capacités pour le second groupe sont plus grandes. On remarque aussi que pour les grandes instances, (plus que 500 objets, $l = 10$), on n'était pas capable de générer des inégalités de couverture étendues et liftées. Ceci est normal pour ces familles d'inégalités. Rappelons que pour séparer ces inégalités, on utilise une

méthode heuristique qui résout un problème du sac à dos ou un pseudo problème du sac à dos avec un algorithme glouton. Plus l'instance est grande, plus on perd de la qualité dans la solution du sous-problème du sac à dos. Par conséquent, ce serait intéressant si on pouvait améliorer la séparation de ces familles pour des instances de grande taille.

Tous les arguments ci-dessus prouvent l'efficacité de notre algorithme de coupe et branchement pour résoudre le problème de DCKP. Notons, cependant, que pour une instance (voir Table 4.3, l'instance $n = 400$, $\eta = 0.07$), Cplex a été capable d'atteindre l'optimalité mais pas l'algorithme de coupe et branchement. Notre algorithm a, par contre, un meilleur gap à la racine que celui de Cplex (5.32% pour l'algorithme de coupe et branchement, 8.06% pour Cplex), et il a été très proche de l'optimum (gap final égal à 0.19%). Ceci est principalement dû au nombre d'inégalités de couverture étendues et de couverture liftées générées qui ont atteint 10767 et 1839, respectivement.

Instance			Branch and Cut								Cplex			
n	m	η	nodes	Gap-final	Gap-root	ECI	LCI	Cliques	OddCycles	CPU	nodes	Gap-final	Gap-root	CPU
100	247	0,05	1	0,00	0,00	0	0	0	0	0,01	1	0,00	0,44	0,00
100	346	0,07	1	0,00	0,00	0	0	0	0	0,01	1	0,00	0,15	0,00
100	445	0,09	3	0,00	0,38	2	0	0	0	0,02	15	0,00	0,38	0,01
100	495	0,1	1	0,00	0,00	0	0	0	0	0,01	4	0,00	0,01	0,01
100	990	0,2	7	0,00	3,18	6	2	1	1	0,06	16	0,00	17,72	0,04
200	995	0,05	10	0,00	0,28	8	6	0	0	0,09	49	0,00	0,12	0,03
200	1393	0,07	17	0,00	0,38	3	1	0	0	0,14	58	0,00	0,43	0,11
200	1791	0,09	14	0,00	0,63	20	9	0	0	0,17	141	0,00	0,53	0,12
200	1990	0,1	36	0,00	2,13	39	24	0	0	0,45	283	0,00	2,59	0,94
200	3980	0,2	110	0,00	18,09	19	8	20	21	3,81	422	0,00	33,80	7,65
300	2242	0,05	8	0,00	0,43	4	1	0	0	0,32	115	0,00	0,30	0,40
300	3139	0,07	8	0,00	1,58	2	1	0	0	0,59	62	0,00	2,35	0,89
300	4036	0,09	26	0,00	3,02	11	1	2	2	1,23	198	0,00	8,77	3,16
300	4485	0,1	112	0,00	7,12	37	13	4	11	4,99	518	0,00	14,18	12,09
300	8970	0,2	1306	0,00	26,10	0	0	54	129	140,91	13241	0,00	48,19	558,21
400	3990	0,05	14	0,00	0,44	1	1	0	0	0,94	313	0,00	1,40	2,84
400	5586	0,07	100	0,00	4,52	33	16	4	14	6,65	919	0,00	11,20	28,65
400	7182	0,09	252	0,00	8,77	9	3	4	53	24,89	1516	0,00	19,73	67,11
400	7980	0,1	2261	0,00	17,03	23	12	36	432	255,46	21615	0,00	30,49	1050,50
400	15960	0,2	50751	5,75	41,33	0	0	358	3799	10800	170337	17,34	87,68	10800
500	6237	0,05	290	0,00	6,17	73	31	3	17	16,32	1020	0,00	9,48	21,58
500	8732	0,07	28346	0,00	17,62	100	43	44	2307	5799,25	168568	3,86	30,27	10800
500	11227	0,09	3100	0,00	15,48	40	15	32	596	613,15	76203	0,00	31,35	4590,42
500	12475	0,1	17556	0,00	19,57	18	6	86	2358	4079,55	184684	6,84	39,46	10800
500	24950	0,2	36773	29,85	58,99	0	0	240	708	10800	109854	64,38	134,75	10800
600	8985	0,05	831	0,00	8,15	90	30	5	72	78,0089	4437	0,00	13,47	287,31
600	12579	0,07	28346	0,00	17,62	100	43	44	2307	6541,56	168568	3,86	30,24	10800
600	16173	0,09	43234	10,95	27,15	18	4	107	4475	10800	148353	18,64	47,61	10800
600	17970	0,1	38137	10,43	26,74	58	15	131	3803	10800	136759	19,68	52,53	10800
700	12232	0,05	15951	0,00	13,20	103	50	16	2312	4566,04	162692	4,20	21,44	10800
700	17125	0,07	36071	10,06	21,87	50	22	64	2823	10800	142310	23,77	45,31	10800
700	22018	0,09	29930	25,87	39,93	0	0	111	1954	10800	107643	48,35	76,12	10800
700	24465	0,1	24776	27,34	42,61	0	0	131	1884	10800	104920	57,51	88,33	10800
800	15980	0,05	37207	3,21	14,08	22	7	20	2230	10800	128724	10,42	26,25	10800
800	22372	0,07	26240	10,54	22,20	1	1	60	1455	10800	119280	33,74	54,37	10800
800	28764	0,09	25021	36,23	48,65	0	0	116	1911	10800	93559	68,94	97,86	10800
800	31960	0,1	20355	35,76	49,45	1	1	125	1221	10800	88836	75,37	104,47	10800
900	20227	0,05	23988	9,81	19,18	24	10	26	7120	10800	112276	24,14	39,16	10800
900	28318	0,07	21564	26,19	36,37	2	2	97	3921	10800	86910	47,92	68,59	10800
900	36409	0,09	16757	45,39	56,89	0	0	198	3285	10800	79646	79,33	102,33	10800
900	40455	0,1	15846	61,45	74,10	0	0	205	2387	10800	64479	89,26	117,31	10800
1000	24975	0,05	25249	16,39	23,85	14	5	48	1730	10800	92720	34,58	48,10	10800
1000	34965	0,07	19158	36,82	46,04	11	8	88	1020	10800	76146	67,85	85,60	10800
1000	44955	0,09	15140	63,56	73,01	0	0	134	831	10800	66315	98,10	121,46	10800
1000	49950	0,1	12919	68,67	77,48	0	0	102	505	10800	65025	107,04	133,64	10800

TABLE 4.1 – Résultats obtenus sur les instances non corrélées avec $l = 5$

Instance			Branch and Cut								Cplex			
n	m	η	nodes	Gap-final	Gap-root	ECI	LCI	Cliques	OddCycles	CPU	nodes	Gap-final	Gap-root	CPU
100	247	0,05	2	0,00	0,25	0	0	0	0	0,01	162	0,00	0,38	0,04
100	346	0,07	6	0,00	0,23	2	0	0	0	0,03	84	0,00	0,35	0,06
100	445	0,09	1	0,00	0,21	0	0	0	0	0,01	1	0,00	0,21	0,00
100	495	0,1	28	0,00	2,67	15	9	0	1	0,09	132	0,00	2,52	0,06
100	990	0,2	21	0,00	7,54	1	0	6	5	0,26	89	0,00	15,55	0,63
200	995	0,05	26	0,00	0,21	23	12	0	0	0,19	194	0,00	0,28	0,17
200	1393	0,07	41	0,00	2,47	29	17	0	3	0,50	269	0,00	3,63	1,03
200	1791	0,09	39	0,00	3,92	16	6	4	9	0,79	283	0,00	9,95	1,98
200	1990	0,1	64	0,00	6,00	49	22	1	14	1,38	384	0,00	12,93	4,90
200	3980	0,2	146	0,00	16,24	0	0	21	16	8,66	3017	0,00	47,04	67,53
300	2242	0,05	67	0,00	3,27	50	17	0	6	2,48	505	0,00	6,40	4,39
300	3139	0,07	390	0,00	9,32	49	26	2	105	17,68	2586	0,00	18,65	69,10
300	4036	0,09	310	0,00	8,68	3	2	11	40	21,53	3774	0,00	20,82	138,75
300	4485	0,1	2062	0,00	16,38	7	4	21	383	156,28	32615	0,00	32,77	906,14
300	8970	0,2	39341	0,00	44,85	0	0	299	1044	6154,42	311652	40,67	116,01	10800
400	3990	0,05	3797	0,00	11,25	52	29	1	426	253,40	22268	0,00	18,07	510,46
400	5586	0,07	23940	0,00	16,46	4	3	31	4749	3765.15	284901	2,87	29,19	10800
400	7182	0,09	47050	6,52	24,05	2	0	88	7614	10800	325979	15,92	50,07	10800
400	7980	0,1	47956	13,32	30,72	0	0	146	8014	10800	316780	32,17	68,91	10800
400	15960	0,2	40904	57,82	92,60	0	0	168	271	10800	180179	102,62	187,73	10800
500	6237	0,05	37444	1,81	14,24	27	12	14	7752	10800	277193	5,73	23,10	10800
500	8732	0,07	35359	20,37	31,85	4	1	87	6615	10800	175741	42,21	64,95	10800
500	11227	0,09	38454	26,27	41,75	0	0	168	6988	10800	212334	54,22	84,63	10800
500	12475	0,1	40261	22,43	36,39	0	0	176	5058	10800	197321	49,97	80,33	10800
500	24950	0,2	22229	90,82	117,51	0	0	34	46	10800	104919	160,28	233,28	10800
600	8985	0,05	37375	11,41	21,55	18	6	21	8370	10800	208657	20,60	37,24	10800
600	12579	0,07	33158	12,41	22,55	4	1	83	6257	10800	158181	84,05	112,11	10800
600	16173	0,09	30552	48,96	61,92	0	0	195	3965	10800	158181	84,05	113,38	10800
600	17970	0,1	28568	46,42	59,91	0	0	173	2863	10800	149536	88,22	121,42	10800
700	12232	0,05	30753	24,64	33,38	0	0	46	7367	10800	164053	42,56	59,41	10800
700	17125	0,07	26705	51,52	62,70	0	0	134	4059	10800	145868	94,21	119,66	10800
700	22018	0,09	23008	65,76	78,09	0	0	136	1604	10800	115391	124,06	155,46	10800
700	24465	0,1	19870	67,19	80,44	0	0	143	931	10800	117491	125,22	160,55	10800
800	15980	0,05	26084	34,27	42,32	0	0	43	3961	10800	87782	61,82	77,17	10800
800	22372	0,07	21315	58,37	66,90	0	0	121	3379	10800	115179	96,29	118,57	10800
800	28764	0,09	16461	89,70	100,96	0	0	117	1107	10800	99850	128,93	158,44	10800
800	31960	0,1	14684	99,92	110,54	0	0	93	422	10800	46248	156,40	184,32	10800
900	20227	0,05	22181	45,59	52,07	0	0	65	6990	10800	109889	72,45	87,06	10800
900	28318	0,07	16465	68,35	77,12	0	0	115	3653	10800	106975	126,86	151,24	10800
900	36409	0,09	12159	94,10	102,07	0	0	93	709	10800	93020	165,11	196,53	10800
900	40455	0,1	10045	124,00	134,93	0	0	57	177	10800	83209	183,76	219,44	10800

TABLE 4.2 – Résultats obtenus sur les instances non corrélées avec $l = 10$

Instance			Branch and Cut								Cplex			
n	m	η	nodes	Gap-final	Gap-root	ECI	LCI	Cliques	OddCycles	CPU	nodes	Gap-final	Gap-root	CPU
100	247	0,05	16	0,00	0,00	25	0	0	0	0,05	4	0,00	0,00	0,01
100	346	0,07	92	0,00	0,00	120	7	0	0	0,25	5	0,00	0,00	0,01
100	445	0,09	197	0,00	0,00	254	4	0	0	1,16	5	0,00	0,00	0,01
100	495	0,1	540	0,00	0,00	377	4	0	1	2,14	2334	0,00	2,18	0,46
200	995	0,05	1	0,00	0,23	0	0	0	1	0,08	193	0,00	0,23	0,36
200	1393	0,07	1	0,00	0,95	1	1	0	0	0,07	1198	0,00	1,12	2,22
200	1791	0,09	293	0,00	1,82	474	47	2	2	6,79	1678	0,00	2,61	4,30
200	1990	0,1	1501	0,00	2,60	2412	238	5	12	88,38	3172	0,00	5,50	8,39
300	2242	0,05	39	0,00	0,84	17	0	1	4	1,98	179	0,00	2,19	2,19
300	3139	0,07	1449	0,00	2,13	1955	150	1	28	91,91	4145	0,00	3,64	23,67
300	4036	0,09	1483	0,00	4,41	850	112	16	262	112,55	4018	0,00	7,07	70,41
300	4485	0,1	2092	0,00	5,35	203	32	19	529	181,97	6752	0,00	8,83	149,99
400	3990	0,05	445	0,00	2,49	123	16	0	48	37,30	3081	0,00	4,17	70,31
400	5586	0,07	33137	0,19	5,32	10767	1839	17	4630	10800	153887	0,00	8,06	3883,62
400	7182	0,09	17804	0,00	6,00	680	171	45	3103	3321,01	136735	0,00	9,62	5594,97
400	7980	0,1	56196	1,63	7,02	136	24	116	5885	10800	293983	3,05	13,14	10800
500	6237	0,05	42994	1,29	5,41	946	152	6	5418	10800	330593	1,16	7,47	10800
500	8732	0,07	47034	2,65	6,79	269	59	51	6021	10800	271224	4,28	11,69	10800
500	11227	0,09	41337	5,76	9,82	97	33	121	7676	10800	223641	9,54	17,92	10800
500	12475	0,1	45661	6,51	11,35	70	11	204	6804	10800	182486	10,00	18,64	10800
600	8985	0,05	41001	2,11	5,55	59	2	14	6469	10800,1	218199	3,66	9,10	10800
600	12579	0,07	37767	6,22	9,40	49	19	65	7524	10800	177060	9,33	15,53	10800
600	16173	0,09	33690	8,91	11,81	135	50	171	4753	10800	149776	14,59	21,36	10800
600	17970	0,1	33574	11,90	15,17	99	53	212	4414	10800	132624	16,52	23,84	10800
700	12232	0,05	29680	5,58	8,48	138	48	29	10522	10800	187259	8,13	12,91	10800
700	17125	0,07	28335	8,53	11,27	15	11	113	9328	10800	152268	13,77	19,76	10800
700	22018	0,09	27815	13,39	16,18	53	20	286	6416	10800	115993	20,17	26,11	10800
700	24465	0,1	24943	13,81	16,65	105	38	245	4295	10800	108295	21,04	27,81	10800
800	15980	0,05	26222	6,88	9,15	139	36	41	8577	10800	138085	10,75	15,47	10800
800	22372	0,07	27344	12,06	14,43	59	18	144	5868	10800	113199	18,23	23,28	10800
800	28764	0,09	23309	14,48	16,77	21	6	203	3394	10800	97043	22,44	28,59	10800
800	31960	0,1	19067	15,67	17,84	26	11	235	2127	10800	83675	24,30	30,98	10800
900	20227	0,05	28333	9,60	11,57	128	31	42	3270	10800	119317	15,79	19,97	10800
900	28318	0,07	20497	13,56	15,54	14	6	116	1670	10800	72385	22,12	26,75	10800
900	36409	0,09	15899	16,70	18,60	36	5	156	1036	10800	81174	26,76	32,29	10800
900	40455	0,1	14142	18,34	20,33	2	2	132	593	10800	79987	30,07	36,30	10800
1000	24975	0,05	23104	10,40	12,12	149	56	70	3462	10800	103098	17,87	21,63	10800
1000	34965	0,07	15789	15,65	17,33	19	5	117	1369	10800	85208	26,02	30,59	10800
1000	44955	0,09	10999	19,13	20,81	0	0	89	289	10800	68456	28,87	34,12	10800
1000	49950	0,1	10606	20,49	22,24	0	0	78	169	10800	70896	35,42	41,44	10800

TABLE 4.3 – Résultats obtenus sur les instances fortement corrélées avec $l = 5$

Instance			Branch and Cut								Cplex			
n	m	η	nodes	Gap-final	Gap-root	ECI	LCI	Cliques	OddCycles	CPU	nodes	Gap-final	Gap-root	CPU
100	247	0,05	20	0,00	0,00	22	0	0	0	0,06	4	0,00	0,00	0,01
100	346	0,07	1	0,00	0,00	0	0	0	0	0,01	8	0,00	0,00	0,01
100	445	0,09	7	0,00	0,51	12	0	0	0	0,07	49	0,00	0,52	0,05
100	495	0,1	1	0,00	0,09	0	0	0	0	0,01	44	0,00	0,38	0,04
100	990	0,2	540	0,00	4,83	131	63	3	26	3.458	1084	0,00	9,76	2,67
200	995	0,05	602	0,00	0,00	851	45	0	0	11,7535	1864	0,00	0,94	3,37
200	1393	0,07	406	0,00	1,41	425	67	1	7	10,1888	745	0,00	2,63	5,45
200	1791	0,09	32	0,00	1,56	0	0	4	9	1,1102	462	0,00	2,78	6,74
200	1990	0,1	1212	0,00	3,76	49	19	11	544	56.71	13889	0,00	8,40	163,13
200	3980	0,2	77721	1,15	8,58	17	14	493	6890	10800	464451	0,00	18,97	6103,17
300	2242	0,05	368	0,00	1,75	160	27	0	92	24,5311	2636	0,00	3,07	56,25
300	3139	0,07	12515	0,00	3,53	89	30	17	2645	1301,46	249704	0,00	6,60	5536,75
300	4036	0,09	48336	2,06	5,51	24	8	69	10473	10800	516532	2,44	9,66	10800
300	4485	0,1	52848	3,73	7,58	40	27	110	10769	10800	651610	4,24	12,50	10800
300	8970	0,2	84625	7,35	11,84	0	0	423	2263	10800	462214	12,46	25,38	10800
400	3990	0,05	38856	0,54	3,31	200	74	5	7687	10800	408974	0,92	5,72	10800
400	5586	0,07	53857	3,05	5,49	19	15	45	7349	10800	355690	4,77	10,29	10800
400	7182	0,09	46274	5,96	8,57	51	29	175	9010	10800	293740	8,13	14,07	10800
400	7980	0,1	50873	7,11	9,94	36	26	215	7448	10800	436858	11,44	18,90	10800
400	15960	0,2	49995	17,92	21,77	0	0	261	497	10800	257266	39,20	51,82	10800
500	6237	0,05	33807	4,07	6,08	33	15	17	10591	10800	262019	6,12	9,82	10800
500	8732	0,07	35345	6,89	8,98	2	1	99	11190	10800	230617	12,00	16,60	10800
500	11227	0,09	40450	9,73	12,10	17	6	266	8008	10800	197699	16,04	21,42	10800
500	12475	0,1	41819	10,52	12,83	0	0	310	3924	10800	290677	15,33	21,26	10800
500	24950	0,2	26664	42,59	45,51	0	0	89	108	10800	177213	74,63	86,61	10800
600	8985	0,05	29370	7,07	8,72	3	2	29	10856	10800	238654	10,14	13,70	10800
600	12579	0,07	33348	9,02	10,67	0	0	157	7229	10800	168148	16,13	20,17	10800
600	16173	0,09	30007	11,82	13,63	7	7	250	3003	10800	144996	20,18	24,92	10800
600	17970	0,1	31490	13,85	15,65	0	0	285	2741	10800	144860	26,06	31,36	10800
700	12232	0,05	30179	8,77	10,24	9	5	61	10400	10800	116826	15,07	17,82	10800
700	17125	0,07	26342	11,82	13,29	0	0	137	4035	10800	137211	20,80	24,59	10800
700	22018	0,09	21134	14,43	15,91	0	0	192	2088	10800	126935	38,85	43,94	10800
700	24465	0,1	19815	18,24	19,76	0	0	166	900	10800	117362	44,34	50,05	10800
800	15980	0,05	25185	9,69	10,72	0	0	81	8884	10800	143579	14,24	16,87	10800
800	22372	0,07	21294	12,58	13,72	0	0	161	2194	10800	113037	35,22	38,77	10800
800	28764	0,009	16209	21,37	22,51	0	0	112	680	10800	97300	58,42	63,17	10800
800	31960	0,1	14099	31,41	32,73	0	0	76	433	10800	76526	63,01	67,62	10800
900	20227	0,05	22524	9,76	10,78	0	0	79	5713	10800	135516	20,35	23,01	10800
900	28318	0,07	15758	14,33	15,41	0	0	104	1057	10800	97675	56,58	60,41	10800
900	36409	0,09	11056	20,99	22,18	0	0	63	392	10800	81766	71,63	76,61	10800
900	40455	0,1	8778	47,66	48,94	0	0	43	248	10800	65160	97,34	103,02	10800
1000	24975	0,05	17449	13,29	14,24	0	0	60	2260	10800	45014	31,13	33,40	10800
1000	34965	0,07	10494	21,41	22,36	0	0	44	283	10800	86958	61,63	65,47	10800
1000	44955	0,09	7764	42,61	43,85	0	0	41	244	10800	27818	94,41	98,37	10800
1000	49950	0,1	6318	58,12	59,05	0	0	13	110	10800	67937	123,33	129,66	10800

TABLE 4.4 – Résultats obtenus sur les instances fortement corrélées avec $l = 10$

4.5 Conclusion

Nous avons développé dans cette section un algorithme de coupe et branchement pour le DCKP. Pour ce faire, des procédures de séparation exactes et heuristiques ont été d'abord décrites. Un grand nombre d'instances a été testé. Les résultats expérimentaux montrent clairement l'efficacité de notre algorithme pour résoudre ce problème difficile à l'optimum.

Chapitre 5

Le problème DCKP : résolution par Recherche Tabou

Comme nous l'avons remarqué dans le chapitre précédent, la méthode exacte de coupes et branchements est efficace pour la résolution des instances de petite et moyenne taille pour le problème DCKP. Pour les instances de grande taille, il convient d'utiliser des approches heuristiques et métaheuristiques permettant d'avoir des solutions proches de l'optimum dans un temps d'exécution raisonnable. Dans ce chapitre, nous nous proposons de résoudre le problème DCKP en utilisant la métaheuristique de Recherche Tabou Probabiliste. Il s'agit d'une variante de la métaheuristique Recherche Tabou dans laquelle on rajoute une pondération sur les mouvements de façon à favoriser ceux avec les plus hautes évaluations. Ce travail a fait l'objet de la publication [10].

5.1 Recherche Tabou

Comme il a été indiqué au Chapitre 1, la métaheuristique de Recherche Tabou, Tabu Search (TS), a été initialement introduite par Fred Glover (1986) [41]. Son principe se base sur des procédures permettant d'une part d'échapper aux optima locaux et d'autres part d'explorer largement l'espace de recherche en franchissant parfois les frontières de faisabilité. La Recherche Tabou est une méthode de recherche locale qui explore, au-delà de l'optimalité locale, l'espace de solution en utilisant une mémoire

adaptative pour plus de flexibilité dans la recherche. La métaheuristique commence par une solution initiale, réalisable ou non, et se déplace itérativement d'une solution à son voisin jusqu'à satisfaire les critères d'arrêt fixés à l'avance. La Recherche Tabou permet des déplacements qui détériorent la fonction objectif de la solution courante x à choisir. Elle peut être vue comme une méthode de recherche à voisinage dynamique puisque les mouvements sont sélectionnés d'un voisinage modifié $\mathcal{N}^*(x)$ de la solution courante x. En effet, les structures de court terme font une restriction du voisinage $\mathcal{N}(x)$, *i.e.*, $\mathcal{N}^*(x) = \mathcal{N}(x) - TL$, avec TL la liste tabou. Les structures de long terme, quant à elles, développent le voisinage $\mathcal{N}(x)$, *i.e.*, $\mathcal{N}^*(x) = \mathcal{N}(x) \cup ES$, avec ES est l'ensemble de solutions élites. La liste tabou TL permet de garder les traces des attributs des solutions qui ont changé sur un passé récent. Ces attributs correspondent à des informations sur des propriétés caractérisant les solutions en se déplaçant de l'une à l'autre. La mémoire adaptative de la Recherche Tabou est utilisée pour créer un équilibre entre les phases d'intensification et de diversification. Les stratégies d'intensification permettent d'intensifier la recherche autour des solutions considérées chronologiquement de bonnes solutions. Les stratégies de diversification conduisent la recherche à des régions différentes de celles déjà examinées.
Une description extensive de la métaheuristique de Recherche Tabou peut être trouvée dans [46, 52].

5.2 Recherche Tabou Probabiliste

La Recherche Tabou Probabiliste, Probabilistic Tabu Search (PTS), est une variante de la Recherche Tabou pour laquelle les mouvements sont choisis de façon probabiliste. En effet, une pondération sur les mouvements est considérée sur les solutions précédemment évaluées (ou un sous-ensemble des meilleures de ces solutions) afin de favoriser celles avec la meilleure évaluation [42, 43]. Plusieurs implémentations de la Recherche Tabou Probabiliste ont été développées dans la littérature, voir par exemple Faigle et

Kern [28], Crainic et al. [21], Glover et Lokketangen [39], Xu et al. [109] et Soriano et Gendreau [100].
Comme nous l'avons mentionné précédemment, nous nous proposons d'utiliser la métaheuristique de Recherche Tabou Probabiliste pour résoudre le DCKP. Pour ce faire, nous présentons dans les sections suivantes les étapes de base de notre approche.

5.2.1 Solution initiale

Exploitant la relation étroite du DCKP avec les problèmes classiques du sac à dos et du stable maximum, plusieurs procédures peuvent être envisagées pour générer une solution initiale pour le DCKP. Par exemple, Yamada et al. [111] proposent une adaptation de l'heuristique gloutonne appliquée au problème du sac à dos en rajoutant respectivement les objets en ordre décroissant de $\frac{p_i}{w_i}$, et tout en respectant les contraintes de disjonction. De façon similaire, des heuristiques constructives inspirées du problème de stable maximum peuvent être utilisées pour construire de façon gloutonne des solutions pour lesquelles on vérifie après la contrainte sac à dos.
Dans notre travail, nous proposons une heuristique gloutonne dans laquelle la sélection des objets prend en compte non seulement le profit et le poids des objets, mais aussi le degré des sommets dans le graphe de conflit (*i.e.*, le nombre de conflits pour chaque objet), et la densité du graphe de conflit. Le pseudo code de l'heuristique proposée est donné dans l'Algorithme 7.

L'heuristique gloutonne suppose que les objets sont ordonnés en ordre décroissant de $\frac{p_i}{w_i+\alpha\eta d_i}$, où $\delta(i) = \{j \in N | (i,j) \in E\}$ est l'ensemble des objets en conflit avec l'objet i, $d_i = |\delta(i)|$ est le degré de i, η est la densité du graphe de conflit $G = (V, E)$, et α est un paramètre. Dans nos expérimentations, le paramètre α est fixé à 0.1. L'algorithme commence par un sac à dos vide, *i.e.*, $x_i = 0$ pour tout $i \in V$. Ensuite, les objets non-sélectionnés sont explorés dans l'ordre et rajoutés tant que la contrainte de capacité n'est pas violée. Une fois l'objet i sélectionné, tous les objets $j \in \delta(i)$ en conflit avec i

Algorithm 7: Heuristique Gloutonne

Data: Une instance du DCKP
Result: Une solution réalisable x.

Ordonner les objets tels que $\frac{p_i}{w_i+\alpha\eta d_i} \geq \frac{p_{i+1}}{w_{i+1}+\alpha\eta d_{i+1}}$, $i = 1, 2, \ldots, n-1$;
for $i = 1$ *to* n **do**
    $x_i = 0$
;
$\Delta \leftarrow c$, $F \leftarrow V$;
while $F \neq \emptyset$ **do**
    Sélectionner le 1er objet i dans F;
    if $w_i \leq \Delta$ **then**
        $x_i \leftarrow 1$, $\Delta \leftarrow \Delta - w_i$, $F \leftarrow F \setminus \delta(i)$;
        for $j \in \delta(i)$ **do**
            $x_j \leftarrow 0$;
    else
        $x_i \leftarrow 0, F \leftarrow F \setminus \{i\}$;
return x ;

seront interdits dans les itérations futures.

5.2.2 Structure du voisinage et son exploration

La définition de la structure du voisinage est l'une des étapes cruciales d'une recherche locale. Chaque solution voisine x' est atteinte à partir de la solution courante x en appliquant un seul ou plusieurs déplacements élémentaires, *i.e.*, une série de modifications locales de x. Les structures de voisinage pour le problème du sac à dos et le problème du stable maximum impliquent souvent les mouvements élémentaires dits d'ajout "Add" et de suppression "Drop", qui permettent de fixer x_i à un ou zero (*i.e.*, en complémentant la variable x_i à $1 - x_i$).

Dans notre heuristique Recherche Locale Probabiliste, nous utilisons les voisinages de façon dynamique en incluant des mouvements multiples de "Add" et "Drop". Soit k un entier non-négatif, le voisinage d'une solution x est le sous ensemble de l'espace de recherche défini par :

$$\mathcal{N}^k(x) = \{x' \text{ obtenu à partir de } x \text{ en supprimant } k \text{ objets et en rajoutant d'autres tant que la faisabilité est maintenue }\}.$$

Dans nos expérimentations, nous choisissons de varier la valeur du paramètre k dans $\{1, 3, 5, 7\}$. Comme la taille du voisinage $\mathcal{N}^k(x)$ devient de plus en plus grande quand la valeur de k augmente, nous avons opté pour différentes stratégies de la liste candidate utilisées pour restreindre le nombre de solutions considérées dans une itération donnée. Afin d'équilibrer les phases d'intensification et de diversification, nous avons choisi quatre types de stratégies définissant la liste candidate en utilisant, semi-aléatoirement, un échantillon des solutions de $\mathcal{N}^k(x)$. Chaque liste candidate $CL^h(x)$, $h = 1, .., 4$ représente un sous-ensemble de $\mathcal{N}^k(x) \setminus TL$, avec TL est la liste tabou. La cardinalité des listes candidates est notée $l = |CL^h(x)|$. Dans nos expérimentations, nous fixons $l = 30$. Chaque élément de $CL^h(x)$ pour $h = 1, .., 4$ est obtenu en supprimant k objets de la solution courante x. La différence principale entre ces listes candidates survient dans la phase d'ajout de nouveaux éléments à la solution courante.
Plus précisément, après avoir supprimé les k objets, chaque solution des quatre listes candidates est construite comme suit :

— $CL^1(x)$: Les variables i fixées à 0 dans la solution courante x sont ordonnées selon un ordre décroissant du ratio $\frac{p_i}{w_i+\alpha\eta d_i}$, et on rajoute après les objets tant que la faisabilité est toujours garantie.

— $CL^2(x)$: Les variables i fixées à 0 dans la solution courante x sont ordonnées selon un ordre décroissant de $p_i - \eta d_i$, et on rajoute après les objets tant que la faisabilité est toujours garantie.

— $CL^3(x)$: Les variables i fixées à 0 dans la solution courante x sont ordonnées selon un ordre décroissant de $p_i - \eta d_i$, et on rajoute après aléatoirement les objets parmi les q meilleurs tant que la faisabilité est toujours garantie.

— $CL^4(x)$: On ajoute aléatoirement les objets tant que la faisabilité est toujours garantie.

Notons que les listes candidates $CL^1(x)$ et $CL^2(x)$ sont utilisées pour renforcer l'aspect "agressif" de la Recherche Tabou (*i.e.*, phase d'intensification). Les listes $CL^3(x)$ et $CL^4(x)$, quant à elles, intègrent l'aspect aléatoire afin d'explorer mieux l'espace de recherche (phase de diversification).

L'Algorithme 8, donné ci-dessous, décrit la stratégie utilisée pour explorer le voisinage $\mathcal{N}^k(x)$ d'une solution courante x. La première liste candidate CL^h est sélectionnée avec une certaine probabilité. Dans notre algorithme, les listes candidates $CL^1(x)$, $CL^2(x)$, $CL^3(x)$ et $CL^4(x)$ sont choisies avec les probabilités $\frac{1}{2}$, $\frac{1}{4}$, $\frac{3}{20}$ et $\frac{1}{10}$, respectivement. De plus, afin de soutenir le caractère agressif de la Recherche Tabou, nous avons choisi la meilleure solution voisine non-tabou dans chaque liste candidate $CL^h(x)$. C'est à dire, une fois la liste candidate $CL^h(x)$ est choisie, nous choisissons la solution voisine $x' \in CL^h(x)$ telle que

$$x' = argmax\{py | y \in CL^h(x) - TL\}.$$

Algorithm 8: Exploration du voisinage $\mathcal{N}^k(x)$

Data: Une solution x, $\mathcal{N}^k$ et la liste tabou TL.
Result: Une solution voisine $x' \in \mathcal{N}^k(x)$.

Fonction `Exploration`$(x, \mathcal{N}^k, TL)$

Générer aléatoirement une valeur réelle $r \in [0, 1]$;
if $r \in [0, 0.5]$ **then**
    fixer $h = 1$;
if $r \in [0.5, 0.75]$ **then**
    fixer $h = 2$;
if $r \in [0.75, 0.9]$ **then**
    fixer $h = 3$;
if $r \in [0.9, 1]$ **then**
    fixer $h = 4$;
Choisir la meilleure solution voisine $x' = argmax\{py | y \in CL^h(x) - TL\}$;
return x' ;

Dans notre implémentation, la liste tabou consiste en la paire (px, wx), représentant

le coût de la solution explorée et sa consommation de la ressource avec $l = 10$.

5.2.3 Algorithme de la Recherche Tabou Probabiliste

L'algorithme de Recherche Tabou Probabiliste commence par une solution initiale x^0 obtenue par une heuristique gloutonne constructive (voir Algorithme 7). La solution initiale devient ainsi la solution courante $x = x^0$ et elle est insérée dans la liste tabou $TL = \{x^0\}$. L'algorithme de Recherche Tabou Probabiliste explore, un à un, dans un ordre donné, et de façon cyclique, les structures de voisinage $\mathcal{N}^k$ pour $k \in \{1, 3, 5, 7\}$. Quand la structure de voisinage $\mathcal{N}^7$ est atteinte, l'algorithme reprend la recherche dans la première structure de voisinage $\mathcal{N}^1$. Le voisinage $\mathcal{N}^1$ est changé en fixant $k = (k + 2)$ *modulo* 8 après chaque q itérations (dans notre implantation, nous avons fixé $q = 200$ itérations).

Algorithm 9: L'algorithme de Recherche Tabou Probabiliste

Data: Une instance de DCKP
Result: La meilleure solution réalisable jusqu'à présent x^*.

```
1  Construire une solution initiale x^0 utilisant une heuristique gloutonne;
2  x* ← x^0 ; x ← x^0;
3  TL ← {x^0};
4  iter ← 0;
5  Sélectionner une valeur initiale k = 1;
6  while Le critère d'arrêt n'est pas atteint do
7      x' ← Exploration(x, N^k, TL);
8      TL ← TL + {x'};
9      if px' > px* then
10         x* ← x' ;
11     x ← x';
12     iter ← iter + 1;
13     if iter modulo q = 0 then
14         k ← (k + 2) modulo 8 ;
15 return x* ;
```

TABLE 5.1 – Charactéristiques des instances testées [64]

Inst.	n	m	η	c
1Iy	500	12475	0.10	1800
2Iy	500	24950	0.20	1800
3Iy	500	37425	0.30	1800
4Iy	500	49900	0.40	1800
5Iy	1000	24975	0.05	1800
6Iy	1000	29970	0.06	2000
7Iy	1000	44955	0.07	2000
8Iy	1000	39960	0.08	2000
9Iy	1000	44955	0.09	2000
10Iy	1000	49950	0.10	2000

5.3 Etude expérimentale

5.3.1 Description des instances

Nous avons évalué notre approche de Recherche Tabou Probabiliste décrite dans la section précédente sur un ensemble d'instances générées par Hifi et Michrafi [58]. Les caractéristiques principales de ces instances sont groupées dans le Tableau 5.1. La première colonne de ce tableau représente les noms des groupes d'instances testées. Ces groupes sont labellisés jIy, $j \in \{1, 2, \ldots, 10\}$ (avec $y \in \{1, \ldots, 5\}$). Pour chaque groupe d'instances, les autres colonnes du Tableau 5.1 donnent respectivement le nombre n d'objets, le nombre $m = |E|$ d'arêtes dans le graphe de conflit, la densité η du graphe, et la capacité c du sac à dos. Les instances labellisées de $1Iy$ à $4Iy$, avec $y \in \{1, \ldots, 5\}$, représentent des instances de taille moyenne avec $n = 500$, une capacité $c = 1800$, et une densité η allant de 0.1 à 0.4. Notons que le nombre m des contraintes de disjonction dépend directement de la densité du graphe. Le deuxième groupe d'instances labellisées de $5Iy$ à $10Iy$, avec $y \in \{1, \ldots, 5\}$, regroupe des instances de plus grande taille avec $n = 1000$, $c = 1800$, ou 2000 et une densité η qui varie de 0.05 à 0.10.

5.3.2 Contexte informatique

L'algorithme de Recherche Tabou Probabiliste est codé en Java et testé sur un Intel Pentium Core i5-6500, 3.2 GHz, 4Gb RAM. Vu la complexité du DCKP, nous avons choisi de fixer un temps d'exécution limite à 250, 500, et 1000 secondes, respectivement. Afin d'éviter l'aspect aléatoire de l'algorithme et avoir un aperçu moyen et général pour chaque groupe d'instance jIy, avec $j \in \{1, \ldots, 10\}$ et $y \in \{1, \ldots, 5\}$, les tests sont exécutés 10 fois.

5.3.3 Résultats expérimentaux

Les résultats de l'algorithme de Recherche Tabou Probabiliste (PTS) sont groupés dans les tableaux 5.2, 5.3, 5.4, et 5.5, et sont comparés à ceux obtenus par les deux méthodes les plus récentes en litérature [57, 64].

Dans le Tableau 5.2, nous rapportons les valeurs moyennes obtenues pour chaque groupe d'instances. Dans les deux premières colonnes, nous mettons les meilleures valeurs objectifs obtenues dans [57] et [64], respectivement. Les autres colonnes représentent les meilleures valeurs obtenues par notre algorithme pour un temps limite fixé à $T = 250$, $T = 500$, et $T = 1000$ secondes, respectivement. La dernière colonne donne la meilleure valeur parmi toutes celles obtenues par les différentes approches de résolution du DCKP. Sur cette colonne, les valeurs en gras sont les meilleures connues, pour lesquelles notre méthode (*i.e.*, La Recherche Tabou Probabiliste, PTS) est au moins aussi bonne que les meilleurs résultats connus selon les méthodes de résolution dans [57] et [64]. Les valeurs marquées en astérisque représentent les instances pour lesquelles notre méthode est meilleure que les approches précédemment mentionnées.

A partir du Tableau 5.2, on peut déduire que notre algorithme donne des résultats très satisfaisants en le comparant aux méthodes développées dans [64] et [57]. Pour la majorité des instances, nous avons atteint les meilleures valeurs actuellement connues

TABLE 5.2 – Etude comparative entre PTS vs HGNS [64] et IRS [57]

	résultats de Hifi et al. ([57, 64])		résultats PTS			
inst.	IRS [57]	HGNS [64]	T=250s	T=500s	T=1000s	Best
1I1	2567	2567	2567	2567	2567	**2567**
1I2	2594	2594	2594	2594	2594	**2594**
1I3	2320	2320	2320	2320	2320	**2320**
1I4	2303	2310	2310	2310	2310	**2310**
1I5	2310	2330	2330	2330	2330	**2330**
2I1	2100	2118	2115	2118	2118	**2118**
2I2	2110	2110	2110	2110	2110	**2110**
2I3	2128	**2132**	2115	2119	2119	2119
2I4	2107	2109	2109	2109	2109	**2109**
2I5	2103	2114	2110	2114	2114	**2114**
3I1	1840	1845	1845	1845	1845	**1845**
3I2	1785	1795	1795	1795	1795	**1795**
3I3	1742	1774	1774	1774	1774	**1774**
3I4	1792	1792	1792	1792	1792	**1792**
3I5	1772	1794	1794	1772	1794	**1794**
4I1	1321	1330	1330	1330	1330	**1330**
4I2	1378	1378	1378	1378	1378	**1378**
4I3	1374	1374	1374	1374	1374	**1374**
4I4	1353	1353	1324	1353	1353	**1353**
4I5	1354	1354	1332	1354	1354	**1354**
5I1	2690	2690	2700	2700	2700	**2700***
5I2	2690	2700	2700	2700	2700	**2700**
5I3	2689	2690	2690	2690	2690	**2690**
5I4	2690	2700	2700	2700	2700	**2700**
5I5	2680	2680	2680	2689	2689	**2689***
6I1	2840	2850	2850	2850	2850	**2850**
6I2	2820	2830	2830	2830	2830	**2830**
6I3	2820	2830	2830	2830	2830	**2830**
6I4	2800	2829	2830	2830	2830	**2830***
6I5	2810	2830	2830	2840	2840	**2840***
7I1	2750	2780	2780	2780	2780	**2780**
7I2	2750	2770	2780	2770	2780	**2780***
7I3	2747	2760	2760	2770	2770	**2770***
7I4	2773	2800	2790	2790	2800	**2800**
7I5	2757	2770	2760	2770	2770	**2770**
8I1	2720	2720	2720	2720	2720	**2720**
8I2	2709	2720	2720	2720	2720	**2720**
8I3	2730	2740	2726	2740	2740	**2740**
8I4	2710	2720	2710	2720	2720	**2720**
8I5	2710	2710	2700	2710	2710	**2710**
9I1	2650	2677	2680	2670	2670	**2680***
9I2	2640	2666	2666	2670	2670	**2670**
9I3	2635	2670	2670	2670	2670	**2670**
9I4	2630	**2668**	2660	2663	2663	2663
9I5	2630	2670	2661	2670	2670	**2670**
10I1	2610	2620	2624	2620	2620	**2624***
10I2	**2642**	2630	2628	2630	2630	2630
10I3	2618	2620	2620	2620	2620	**2620**
10I4	**2621**	2620	2620	2620	2620	2620
10I5	2606	2630	2619	2627	2627	**2627**
Avg.	2390.40	2401.66	2399.04	2401.34	2402.18	**2402.46**

pour les différents groupes d'instances (valeurs en gras). En effet, notre approche de Recherche Tabou Probabiliste est capable d'atteindre 46 meilleures solutions connues sur 50 benchmarks, c'est à dire plus que 90% des instances testées. De plus, pour 8 sur 50 des groupes d'instances, nous avons été capables d'avoir de *nouvelles meilleures solutions connues* (valeurs en gras avec astérisque). En effet, pour ces groupes d'instances, notre algorithme a été meilleur que la méthode de Recherche Arrondissante Itérative (Iterative Rounding Search, IRS) développée dans [57], et la méthode hybride de Recherche à Voisinage guidé (Hybrid Guided Neighborhood Search, HGNS) de [64]. Pour seulement 4 parmi les 50 instances, notre algorithme donne un résultat qui n'est pas meilleur que les autres trouvés dans [64] et [57]. Par contre, nous avons été très proches des meilleures solutions connues, *i.e.*, au plus 6% des meilleures valeurs connues.

Dans la dernière ligne du Tableau 5.2, nous rapportons la moyenne globale sur tous les groupes des instances testées. En se basant sur cette ligne, nous remarquons que notre algorithme avait la meilleure valeur des moyennes globales. Ceci montre qu'en moyenne notre méthode se comporte mieux que toutes les autres méthodes déjà développées dans la littérature.

Les autres tableaux de résultats, *i.e.*, tableaux 5.3, 5.4, and 5.5, donnent les détails des moyennes rapportées dans le Tableau 5.2. Comme il a été mentionné avant, pour chaque groupe d'instances jIy, $j \in \{1, \ldots, 10\}$ et $y \in \{1, \ldots, 5\}$, les tests sont exécutés 10 fois afin d'éviter le plus l'aspect aléatoire de l'algorithme. Les résultats groupés dans les tableaux 5.3, 5.4, et 5.5 correspondent respectivement à un temps limite fixé à $T = 250$, $T = 500$, et $T = 1000$ secondes. Pour ces tableaux, la première colonne correspond au nom de l'instance. Ensuite, les 10 colonnes suivantes correspondent aux valeurs obtenues par les 10 exécutions de la même instance (sur une ligne). Les trois dernières colonnes donnent la valeur maximale, minimale et moyenne obtenues des 10 valeurs précédemment rapportées. Une dernière ligne pour les trois tableaux rapporte

les valeurs moyennes globales pour les différentes exécutions et les valeurs maximales, minimales et moyennes.

A partir des tableaux 5.3, 5.4, et 5.5, nous pouvons remarquer que sur les 50 groupes d'instances, nous avons pu atteindre les meilleures valeurs connues avec moins de 250 secondes. Pour 17 sur 50, nous avons eu les meilleures valeurs connues avec un temps limite ne dépassant pas les 500 secondes. Aussi, pour seulement 1 instance, nous avons eu besoin d'atteindre un temps limite de 1000 secondes afin de trouver la meilleure valeur. Ceci montre que notre algorithme de Recherche Tabou Probabiliste donne de très bons résultats en un temps d'exécution très raisonnable.

TABLE 5.3 – Détails d'exécution des instances pour T=250 secondes

inst.	1	2	3	4	5	6	7	8	9	10	max	min	avg
1I1	2557	2567	2557	2567	2567	2567	2567	2567	2567	2567	2567	2557	2565
1I2	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594
1I3	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320
1I4	2307	2307	2310	2310	2310	2310	2310	2310	2310	2310	2310	2307	2309.4
1I5	2320	2320	2320	2330	2320	2320	2320	2320	2330	2320	2330	2320	2322
2I1	2115	2115	2115	2115	2115	2110	2115	2115	2115	2115	2115	2110	2114.5
2I2	2110	2110	2110	2110	2110	2109	2109	2110	2110	2110	2110	2109	2109.8
2I3	2109	2100	2100	2100	2099	2108	2110	2109	2110	2115	2115	2099	2106
2I4	2107	2109	2109	2100	2105	2100	2099	2100	2099	2109	2109	2099	2103.7
2I5	2110	2100	2100	2100	2110	2110	2110	2100	2100	2103	2110	2100	2104.3
3I1	1845	1743	1688	1714	1698	1845	1778	1845	1845	1845	1845	1688	1784.6
3I2	1751	1695	1795	1751	1795	1751	1795	1795	1667	1779	1795	1667	1757.4
3I3	1712	1774	1739	1755	1739	1774	1755	1714	1774	1714	1774	1712	1745
3I4	1663	1680	1792	1663	1734	1641	1792	1663	1709	1734	1792	1641	1707.1
3I5	1724	1705	1794	1695	1724	1724	1772	1764	1794	1699	1794	1695	1739.5
4I1	1330	1330	1316	1330	1316	1321	1321	1330	1330	1321	1330	1316	1324.5
4I2	1378	1295	1297	1378	1378	1313	1301	1378	1378	1378	1378	1295	1347.4
4I3	1327	1334	1334	1327	1327	1374	1374	1315	1327	1374	1374	1315	1341.3
4I4	1311	1278	1324	1259	1324	1324	1304	1324	1304	1324	1324	1259	1307.6
4I5	1330	1298	1321	1312	1321	1321	1321	1312	1332	1330	1332	1298	1319.8
5I1	2690	2690	2690	2690	2690	2700	2690	2700	2690	2700	2700	2690	2693
5I2	2700	2700	2700	2690	2700	2690	2690	2700	2689	2690	2700	2689	2694.9
5I3	2690	2690	2690	2680	2690	2690	2690	2690	2690	2690	2690	2680	2689
5I4	2680	2697	2690	2690	2700	2690	2690	2697	2697	2696	2700	2680	2692.7
5I5	2680	2670	2680	2680	2670	2679	2680	2680	2670	2670	2680	2670	2675.9
6I1	2830	2850	2850	2850	2850	2840	2830	2850	2820	2850	2850	2820	2842
6I2	2830	2820	2820	2830	2830	2820	2830	2820	2820	2820	2830	2820	2824
6I3	2810	2820	2830	2820	2830	2820	2820	2830	2820	2810	2830	2810	2821
6I4	2816	2820	2810	2820	2818	2810	2820	2830	2810	2810	2830	2810	2816.4
6I5	2820	2830	2820	2820	2820	2830	2820	2820	2820	2830	2830	2820	2823
7I1	2770	2760	2760	2760	2770	2750	2769	2780	2750	2770	2780	2750	2763.9
7I2	2750	2750	2750	2760	2770	2780	2770	2770	2770	2770	2780	2750	2764
7I3	2750	2756	2758	2760	2760	2750	2760	2750	2759	2750	2760	2750	2755.3
7I4	2780	2780	2780	2770	2790	2780	2790	2780	2790	2790	2790	2770	2783
7I5	2750	2740	2750	2759	2760	2750	2750	2750	2750	2760	2760	2740	2751.9
8I1	2710	2710	2720	2700	2710	2698	2710	2710	2705	2720	2720	2698	2709.3
8I2	2720	2700	2700	2700	2710	2710	2707	2700	2710	2700	2720	2700	2705.7
8I3	2710	2710	2708	2710	2720	2710	2726	2705	2710	2700	2726	2700	2710.9
8I4	2700	2710	2700	2700	2709	2700	2690	2710	2690	2690	2710	2690	2699.9
8I5	2700	2690	2690	2689	2690	2700	2680	2700	2700	2700	2700	2680	2693.9
9I1	2680	2660	2657	2670	2640	2670	2650	2680	2660	2650	2680	2640	2661.7
9I2	2650	2660	2660	2650	2650	2650	2640	2650	2640	2666	2666	2640	2651.6
9I3	2670	2650	2660	2660	2640	2659	2650	2640	2650	2660	2670	2640	2653.9
9I4	2650	2660	2650	2659	2650	2644	2650	2640	2650	2639	2660	2639	2649.2
9I5	2661	2650	2650	2660	2640	2650	2656	2660	2644	2660	2661	2640	2653.1
10I1	2619	2610	2610	2597	2600	2600	2609	2590	2624	2609	2624	2590	2606.8
10I2	2620	2610	2610	2619	2610	2617	2600	2610	2610	2628	2628	2600	2613.4
10I3	2620	2600	2610	2600	2620	2590	2600	2607	2600	2600	2620	2590	2604.7
10I4	2600	2620	2610	2600	2613	2600	2599	2595	2599	2598	2620	2595	2603.4
10I5	2600	2610	2610	2610	2600	2610	2610	2600	2619	2610	2619	2600	2607.9
Avg.	2385.52	2379.94	2385.16	2380.66	2385.12	2384.46	2386.86	2386.58	2385.42	2387.94	2399.04	2367.84	2384.766

TABLE 5.4 – Détails d'exécution des instances pour T=500 secondes

inst.	1	2	3	4	5	6	7	8	9	10	max	min	avg
1I1	2563	2567	2567	2567	2567	2567	2567	2567	2567	2567	2567	2563	2566.6
1I2	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594
1I3	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320
1I4	2310	2310	2310	2310	2310	2310	2310	2310	2310	2310	2310	2310	2310
1I5	2320	2320	2330	2330	2330	2330	2320	2320	2330	2320	2330	2320	2325
2I1	2115	2117	2115	2118	2117	2115	2115	2115	2115	2110	2118	2110	2115.2
2I2	2110	2110	2110	2110	2110	2110	2110	2110	2110	2110	2110	2110	2110
2I3	2110	2111	2109	2110	2119	2111	2119	2111	2111	2113	2119	2109	2112.4
2I4	2105	2107	2107	2107	2107	2100	2100	2107	2107	2109	2109	2100	2105.6
2I5	2110	2110	2110	2114	2110	2110	2110	2110	2110	2110	2114	2110	2110.4
3I1	1743	1772	1700	1742	1763	1845	1845	1743	1708	1742	1845	1700	1760.3
3I2	1758	1774	1759	1795	1672	1795	1795	1774	1758	1795	1795	1672	1767.5
3I3	1774	1755	1774	1755	1739	1755	1755	1774	1750	1739	1774	1739	1757
3I4	1792	1792	1792	1753	1753	1792	1792	1792	1753	1663	1792	1663	1767.4
3I5	1742	1772	1732	1769	1769	1738	1772	1719	1748	1748	1772	1719	1750.9
4I1	1330	1330	1321	1330	1330	1330	1330	1330	1330	1330	1330	1321	1329.1
4I2	1378	1378	1378	1378	1378	1378	1378	1378	1303	1378	1378	1303	1370.5
4I3	1374	1374	1334	1374	1374	1374	1374	1374	1374	1374	1374	1334	1370
4I4	1350	1298	1350	1324	1350	1350	1324	1353	1324	1353	1353	1298	1337.6
4I5	1330	1332	1332	1332	1330	1330	1332	1354	1330	1330	1354	1330	1333.2
5I1	2690	2690	2700	2700	2700	2700	2700	2700	2700	2699	2700	2690	2697.9
5I2	2700	2700	2700	2700	2700	2700	2700	2690	2700	2700	2700	2690	2699
5I3	2690	2690	2690	2690	2690	2690	2690	2690	2680	2690	2690	2680	2689
5I4	2700	2700	2700	2700	2700	2700	2700	2690	2700	2700	2700	2690	2699
5I5	2680	2689	2680	2680	2689	2689	2680	2680	2680	2680	2689	2680	2682.7
6I1	2850	2830	2840	2850	2850	2840	2830	2840	2850	2850	2850	2830	2843
6I2	2830	2820	2830	2830	2830	2830	2830	2830	2830	2830	2830	2820	2829
6I3	2830	2830	2830	2830	2830	2830	2830	2830	2830	2830	2830	2830	2830
6I4	2820	2830	2820	2830	2830	2829	2820	2820	2820	2828	2830	2820	2824.7
6I5	2820	2820	2830	2820	2820	2840	2820	2830	2830	2820	2840	2820	2825
7I1	2770	2770	2770	2770	2770	2770	2760	2770	2770	2780	2780	2760	2770
7I2	2760	2760	2770	2770	2760	2760	2760	2765	2760	2770	2770	2760	2763.5
7I3	2760	2750	2760	2760	2770	2770	2760	2760	2760	2770	2770	2750	2762
7I4	2780	2770	2790	2789	2790	2778	2780	2790	2789	2780	2790	2770	2783.6
7I5	2770	2760	2760	2760	2760	2760	2769	2760	2759	2770	2770	2759	2762.8
8I1	2720	2720	2710	2719	2720	2720	2720	2720	2720	2720	2720	2710	2718.9
8I2	2710	2720	2710	2710	2710	2700	2720	2719	2720	2717	2720	2700	2713.6
8I3	2730	2727	2730	2740	2720	2740	2728	2740	2720	2740	2740	2720	2731.5
8I4	2710	2710	2710	2710	2720	2720	2710	2710	2710	2710	2720	2710	2712
8I5	2700	2700	2710	2700	2710	2710	2700	2700	2710	2710	2710	2700	2705
9I1	2660	2669	2670	2660	2670	2670	2670	2670	2660	2670	2670	2660	2666.9
9I2	2660	2660	2660	2670	2668	2660	2660	2660	2660	2659	2670	2659	2661.7
9I3	2670	2660	2670	2660	2670	2670	2670	2669	2666	2660	2670	2660	2666.5
9I4	2660	2663	2660	2650	2656	2660	2654	2650	2660	2660	2663	2650	2657.3
9I5	2660	2660	2660	2670	2660	2670	2650	2670	2660	2660	2670	2650	2662
10I1	2611	2619	2620	2610	2609	2610	2610	2618	2610	2620	2620	2609	2613.7
10I2	2630	2620	2620	2620	2628	2620	2620	2610	2620	2620	2630	2610	2620.8
10I3	2620	2615	2620	2600	2610	2620	2610	2620	2610	2620	2620	2600	2614.5
10I4	2610	2600	2620	2610	2610	2610	2610	2610	2607	2610	2620	2600	2609.7
10I5	2610	2627	2610	2619	2610	2620	2620	2620	2620	2620	2627	2610	2617.6
Avg.	2392.78	2392.44	2391.88	2393.18	2392.04	2396.80	2394.86	2393.72	2389.26	2392.16	2401.34	2378.44	2392.912

TABLE 5.5 – Détails d'exécution des instances pour T=1000 secondes

inst.	1	2	3	4	5	6	7	8	9	10	max	min	avg
1I1	2567	2567	2567	2567	2567	2567	2567	2567	2567	2567	2567	2567	2567
1I2	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594	2594
1I3	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320	2320
1I4	2310	2310	2310	2310	2310	2310	2310	2310	2310	2310	2310	2310	2310
1I5	2320	2320	2320	2320	2320	2320	2320	2320	2320	2330	2330	2320	2321
2I1	2115	2117	2115	2118	2117	2115	2115	2115	2115	2110	2118	2110	2115.2
2I2	2110	2110	2110	2110	2110	2110	2110	2110	2110	2110	2110	2110	2110
2I3	2110	2111	2109	2110	2119	2111	2119	2111	2111	2113	2119	2109	2112.4
2I4	2105	2107	2107	2107	2107	2100	2100	2107	2107	2109	2109	2100	2105.6
2I5	2110	2110	2110	2114	2110	2110	2110	2110	2110	2110	2114	2110	2110.4
3I1	1743	1742	1700	1742	1763	1845	1845	1743	1708	1742	1845	1700	1757.3
3I2	1758	1774	1759	1795	1672	1795	1795	1774	1758	1795	1795	1672	1767.5
3I3	1774	1755	1774	1755	1739	1755	1755	1774	1750	1739	1774	1739	1757
3I4	1792	1792	1792	1753	1753	1792	1792	1792	1753	1663	1792	1663	1767.4
3I5	1742	1772	1732	1769	1769	1738	1772	1719	1794	1748	1794	1719	1755.5
4I1	1330	1330	1321	1330	1330	1330	1330	1330	1330	1330	1330	1321	1329.1
4I2	1378	1378	1378	1378	1378	1378	1378	1378	1303	1378	1378	1303	1370.5
4I3	1374	1374	1334	1374	1374	1374	1374	1374	1374	1374	1374	1334	1370
4I4	1350	1298	1350	1324	1350	1350	1324	1353	1324	1353	1353	1298	1337.6
4I5	1330	1332	1332	1332	1330	1330	1332	1354	1330	1330	1354	1330	1333.2
5I1	2690	2690	2700	2700	2700	2700	2700	2700	2700	2699	2700	2690	2697.9
5I2	2700	2700	2700	2700	2700	2700	2700	2690	2700	2700	2700	2690	2699
5I3	2690	2690	2690	2690	2690	2690	2690	2690	2680	2690	2690	2680	2689
5I4	2700	2700	2700	2700	2700	2700	2700	2690	2700	2700	2700	2690	2699
5I5	2680	2689	2680	2680	2689	2689	2680	2680	2680	2680	2689	2680	2682.7
6I1	2850	2830	2840	2850	2850	2840	2830	2840	2850	2850	2850	2830	2843
6I2	2830	2820	2830	2830	2830	2830	2830	2830	2830	2830	2830	2820	2829
6I3	2830	2830	2830	2830	2830	2830	2830	2830	2830	2830	2830	2830	2830
6I4	2820	2830	2820	2830	2830	2829	2820	2820	2820	2828	2830	2820	2824.7
6I5	2820	2820	2830	2820	2820	2840	2820	2830	2830	2820	2840	2820	2825
7I1	2760	2770	2770	2770	2780	2770	2780	2770	2770	2770	2780	2760	2771
7I2	2770	2750	2770	2768	2770	2770	2780	2770	2770	2780	2780	2750	2769.8
7I3	2760	2770	2770	2760	2760	2760	2760	2760	2760	2760	2770	2760	2762
7I4	2790	2790	2790	2790	2790	2800	2790	2789	2790	2800	2800	2789	2791.9
7I5	2760	2760	2760	2760	2766	2770	2750	2770	2770	2770	2770	2750	2763.6
8I1	2720	2720	2710	2719	2720	2720	2720	2720	2720	2720	2720	2710	2718.9
8I2	2710	2720	2710	2710	2710	2700	2720	2719	2720	2717	2720	2700	2713.6
8I3	2730	2727	2730	2740	2720	2740	2728	2740	2720	2740	2740	2720	2731.5
8I4	2710	2710	2710	2710	2720	2720	2710	2710	2710	2710	2720	2710	2712
8I5	2700	2700	2710	2700	2710	2710	2700	2700	2710	2710	2710	2700	2705
9I1	2660	2669	2670	2660	2670	2670	2670	2670	2660	2670	2670	2660	2666.9
9I2	2660	2660	2660	2670	2668	2660	2660	2660	2660	2659	2670	2659	2661.7
9I3	2670	2660	2670	2660	2670	2670	2670	2669	2666	2660	2670	2660	2666.5
9I4	2660	2663	2660	2650	2656	2660	2654	2650	2660	2660	2663	2650	2657.3
9I5	2660	2660	2660	2670	2660	2670	2650	2670	2660	2660	2670	2650	2662
10I1	2611	2619	2620	2610	2609	2610	2610	2618	2610	2620	2620	2609	2613.7
10I2	2630	2620	2620	2620	2628	2620	2620	2610	2620	2620	2630	2610	2620.8
10I3	2620	2615	2620	2600	2610	2620	2610	2620	2610	2620	2620	2600	2614.5
10I4	2610	2600	2620	2610	2610	2610	2610	2610	2607	2610	2620	2600	2609.7
10I5	2610	2627	2610	2619	2610	2620	2620	2620	2620	2620	2627	2610	2617.6
Avg.	2392.86	2392.44	2391.88	2392.96	2392.16	2397.24	2395.48	2394	2390.42	2392.56	2402.18	2378.72	2393.2

5.4 Conclusion

Dans ce chapitre, nous avons utilisé la métaheuristique de Recherche Tabou Probabiliste pour la résolution du problème DCKP. Notre algorithme commence par une solution initiale construite par une heuristique gloutonne. Nous utilisons par la suite différentes structures de voisinage basées sur des mouvements de suppression (Drop) et d'ajout (Add) explorés de façon cyclique. L'algorithme proposé est évalué sur un total de 50 instances de la littérature contenant jusqu'à 1000 objets. Les résultats expérimentaux montrent l'efficacité de notre algorithme pour la résolution de ces instances en un temps raisonnable. En particulier, notre approche nous a permis d'atteindre 46 meilleures solutions connues et découvrir 8 nouvelles meilleures solutions sur les 50 instances testées. Ceci montre aussi l'efficacité des méthodes heuristiques pour atteindre des solutions de bonne qualité en un temps raisonnable.
Dans le chapitre suivant, nous continuons l'intérêt aux méthodes heuristiques et métaheuristiques. Nous présentons une nouvelle heuristique et l'appliquons à une autre variante du sac à dos, à savoir le sac à dos multidimensionnel.

Chapitre 6

Le problème MKP : approches hybrides et relaxations

Dans ce chapitre, nous considérons le problème plus général du sac à dos multidimensionnel (MKP) introduit dans le Chapitre 2. Nous développons une heuristique basée sur la programmation linéaire pour ce problème. Notre heuristique est une amélioration de la méthode, dite Heuristique Itérative basée sur la Programmation Linéaire (HIPL), proposée par Wilbaut et Hanafi dans [55]. Cette dernière est inspirée d'une heuristique dite Heuristique basée sur la Programmation Linéaire (HPL), élaborée en 1978 par Soyerster et al. [101]. La méthode HPL consiste à résoudre exactement un sous problème généré à partir de la relaxation linéaire. La méthode HIPL est une implémentation itérative de la méthode HPL. Comme continuation à ces travaux nous proposons une Heuristique itérative basée sur la programmation linéaire et la décomposition de l'espace de recherche en hyperplans dite HIPL$_k$.

Elle repose sur la réduction de l'espace d'exploration par la fixation du nombre d'objets. HIPL$_k$ est une combinaison de la méthode HIPL et une technique de décomposition proposée par Vasquez et Hao [104].

Dans ce chapitre, nous présentons les heuristiques HPL, HIPL et HIPL$_k$, et nous discutons de résultats expérimentaux obtenus par la dernière pour le problème MKP.

6.1 Définitions

Avant de détailler les heuristiques susmentionnées, nous résumons, dans ce paragraphe, quelques notations qui seront utilisées par la suite. Rappelons tout d'abord que le problème MKP peut être représenté par le problème (P) suivant.

$$(P)\begin{cases} \max \quad z = \sum_{j=1}^{n} p_j x_j \\ s.c. \quad \sum_{j=1}^{n} w_{ij} x_j \leq c_i \quad \forall i \in \{1, \ldots, m\}, \\ \qquad x \in \{0,1\}^n, \end{cases} \tag{6.1}$$

où p_j et w_j représentent respectivement le profit et le poids d'un objet j, c_i la capacité de la contrainte i, n le nombre d'objets et m le nombre de contraintes.

Rappelons aussi la définition d'un problème réduit $P(x^0, J)$ obtenu à partir d'une solution réalisable x^0 du problème (P) et d'un sous ensemble de variables $J \subseteq V$.

$$(P(x^0, J))\begin{cases} \max \qquad z = \sum_{j=1}^{n} p_j x_j \\ s.c. \qquad Ax \leq c, \\ x_j = x_j^0 \qquad \forall j \in J, \\ x \in \{0,1\}^n, \end{cases} \tag{6.2}$$

où A est la matrice des contraintes.

Notons par :

$V = 1, ..., n$ l'ensemble des variables du problème.

S l'espace de toutes les solutions réalisables d'un problème donné P en variables 0-1.

$\nu(P)$ la valeur optimale du problème (P).

$z(x)$ la valeur de la solution x (*i.e.* $z(x) = \sum_{j=1}^{n} p_j x_j$).

$LP(P)$ la relaxation linéaire du problème (P).

x^* la meilleure solution obtenue.

Finalement, si Q un ensemble de contraintes, nous notons $(P|Q)$ le problème d'optimisation obtenu en ajoutant les contraintes Q au problème P.

6.2 Heuristiques et programmation linéaire

Des heuristiques basées sur la programmation linéaire ont été élaborées et utilisées par plusieurs auteurs pour résoudre des problèmes d'optimisation combinatoire. L'application de ces méthodes est souvent choisie convenablement pour aboutir à de bons résultats tant en qualité qu'en temps d'exécution. Ainsi, partant du fait qu'autour d'un optimum fractionnaire $\bar{x}$ - obtenu par la relaxation linéaire - les points binaires sont de bonne qualité, plusieurs auteurs ont proposé de réduire l'espace de recherche à une boule centrée en $\bar{x}$. Toutefois, il existe des cas où la solution optimale ne voisine pas la solution continue, et nécessite d'autres approches de réduction.

6.2.1 HPL : Heuristique basée sur la programmation linéaire

Cette méthode est mise en évidence depuis les années 70 par Soyster et al. [101]. L'heuristique se base sur la résolution exacte d'un sous problème réduit généré à partir de la résolution de la relaxation linéaire. Cette heuristique, décrite dans l'Algorithme 10, procède en deux phases.
Dans la première phase, on résout la relaxation linéaire (6.3) d'un problème P pour en obtenir une solution optimale $\bar{x}$.

$$PL(P)\begin{cases} \max \sum_{j=1}^{n} p_j x_j & \\ \sum_{j=1}^{n} w_{ij} x_j \leq c_i & \forall i \in \{1,\ldots,m\}, \\ x_j \in [0,1] & \forall j \in \{1,\ldots,n\}. \end{cases} \tag{6.3}$$

Dans la deuxième phase, on résout le problème réduit (6.4) généré en ne conservant

que les variables fractionnaires de la solution $\bar{x}$ comme variables libres. Il s'agit, en effet, de résoudre de manière exacte le problème réduit donné par (6.4) pour en obtenir une solution x^0 réalisable de (P).

$$P(\bar{x}, J(\bar{x})) \begin{cases} \max \sum_{j=1}^{n} p_j x_j & \\ \sum_{j=1}^{n} w_{ij} x_j \leq c_i & \forall i \in \{1, \ldots, m\}, \\ x_j = \bar{x_j} & \forall j \in J(\bar{x}), \\ x_j \in \{0, 1\} & \forall j \in \{1, \ldots, n\}, \end{cases} \tag{6.4}$$

où $J(\bar{x}) = \{j | \bar{x_j} \in \{0, 1\}\}$ correspond à l'ensemble d'objets à valeurs binaires dans $\bar{x_j}$, solution de la relaxation linéaire (6.3).

Algorithm 10: HPL

Data: Un problème P
Result: une solution réalisable x^0
1 Résoudre la relaxation linéaire de P et récupérer la solution fractionnaire $\bar{x}$;
2 Résoudre le problème réduit $P(\bar{x}, J(\bar{x}))$ avec $J(\bar{x}) = \{j | \bar{x_j} \in \{0, 1\}\}$ pour obtenir une solution réalisable x^0;
3 **return** x^0

Cette méthode est en général efficace lorsque le nombre m des contraintes du problème est petit par rapport au nombre n des variables du problème. Ceci peut s'expliquer par le théorème suivant.

Théorème 6.1. *[71, 105]*

Toute solution réalisable de la relaxation en continu d'un problème d'optimisation P comportant n variables et m contraintes $(m \leq n)$ a au plus m variables fractionnaires.

L'heuristique HPL est efficace pour une résolution rapide du problème du sac à dos et certaines de ses variantes telles que le MKP (voir [105]).
Dans [106] Wilbaut et Hanafi ont proposé une généralisation de la HPL en une méthode itérative présentée dans la section suivante.

6.2.2 HIPL : Heuristique itérative basée sur la programmation linéaire

Cette méthode est proposée par Wilbaut et Hanafi en 2009 [106], et est améliorée en 2011 par les mêmes auteurs [55]. Son principe consiste à appliquer l'heuristique HPL itérativement pour résoudre le problème du sac à dos multidimensionnel.

A chaque itération, le sous espace de recherche exploré est modifié en ajoutant une pseudo-coupe au problème pour diminuer la valeur de la borne supérieure à l'itération suivante tout en se dirigeant vers une région de recherche inexplorée. La contrainte ajoutée est inspirée de la coupe canonique proposée par Balas et Jeroslow sur l'hypercube unité [5]. Il s'agit d'ajouter à chaque itération l'inégalité :

$$f^t x \leq |J^1(\bar{x})| - 1, \tag{6.5}$$

avec $J^1(\bar{x}) = \{j \in V | \bar{x_j} = 1\}$ et f^t est la transposée du vecteur f de dimension $|V|$ défini par :

$$f_j = \begin{cases} 1 & \text{si } \bar{x_j} = 1, \\ -1 & \text{si } \bar{x_j} = 0, \\ 0 & \text{si } \bar{x_j} \in]0,1[. \end{cases}$$

Chaque variable égale à 1 (respectivement 0) dans la solution courante reçoit un coefficient égal à 1 (respectivement -1) dans la contrainte. Les autres variables (*i.e.*, les variables fractionnaires) ont un coefficient égal à 0. La contrainte sus-citée vise à éliminer de l'espace de recherche les solutions préalablement visitées lors de la résolution du problème réduit. Cette contrainte, est générée selon la solution optimale de la relaxation en continu du problème P à une itération donnée de l'algorithme.

L'Algorithme 11 (introduit dans [106]) décrit l'heuristique itérative HIPL. La première étape de l'algorithme consiste à calculer une solution initiale réalisable x^* par la méthode HPL. Dans la seconde étape on ajoute au problème courant Q une pseudo-

Algorithm 11: HIPL

Data: Un problème P et max_iter
Result: une solution réalisable x^*

1 Soit x^* une solution réalisable;
2 Soient $v^* \leftarrow z(x^*)$; $stop \leftarrow faux$; $Q \leftarrow P$; $iter \leftarrow 0$;
3 **while** $stop$=$faux$ **do**
4 soit $\bar{x}$ une solution optimale de $PL(Q)$;
5 soit x^0 une solution optimale de $Q(\bar{x}, J(\bar{x}))$;
6 **if** $z(x^0) > v^*$ **then**
7 $x^* \leftarrow x^0$; $v^* \leftarrow z(x^*)$;
8 $Q \leftarrow Q|\{f^t x \leq |J^1(\bar{x})|\}$;
9 $iter \leftarrow iter + 1$;
10 **if** $\lfloor z(\bar{x}) - z(x^*) \rfloor < 1$ *ou* $iter > max\_iter$ **then**
11 $stop \leftarrow vrai$
12 **return** x^*

coupe générée selon la contrainte (6.5) (ligne 8 de l'Algorithme 11) et on résout le nouveau problème par la méthode HPL afin d'obtenir une solution fractionnaire $\bar{x}$ et une solution réalisable x^0. Cette étape est itérée tant que les conditions d'arrêt ne sont pas vérifiées. En effet, l'algorithme s'arrête lorsqu'on atteint un nombre maximum d'itérations max_iter, ou lorsque la différence entre la borne supérieure donnée par $\bar{x}$ et la borne inférieure donnée par x^0 est inférieure à 1 (ligne 10 de l'Algorithme 11). Dans ce cas la borne inférieure est la solution optimale du problème P.

L'heuristique HIPL a connu beaucoup de succès pour la résolution de problèmes difficiles d'optimisation combinatoire. En effet, en plus du MKP, plusieurs auteurs ont opté pour le HIPL pour résoudre d'autres variantes du sac à dos. Par exemple, Haddar et al. [50] proposent une nouvelle présentation de HIPL en l'adaptant au problème de la distribution équitable appelé en anglais Knapsack Sharing Problem (KSP). Aussi, Crevits et al. [22] proposent une adaptation de HIPL au problème MMKP (introduit dans le chapitre 2).

Dans la Section 6.4, nous proposons une heuristique inspirée de l'heuristique HIPL et la Technique dite de décomposition de l'espace de recherche dans la section suivante.

6.3 Technique de décomposition de l'espace de recherche

Notons que toute solution d'un problème P en variables $0-1$ peut être caractérisée par le nombre k de variables fixées à 1. Plus formellement, chaque solution vérifie une égalité de type $\sum_{j=1}^{n} x_j = k$, $k \in \{1, \ldots, n\}$. Il est évident que k varie d'une solution à une autre.

Dans [104], Vasquez et Hao proposent de résoudre le problème MKP par la métaheuristique recherche tabou. Cette méthode introduit une technique de décomposition de l'espace de recherche S en Hyperplans appelés H_k, et par la suite, la décomposition du problème P en sous problèmes P_k (*i.e.*, P_k est le sous problème relatif à l'hyperplan H_k). Notons que la résolution du problème P_k consiste à fixer à k le nombre d'objets à mettre dans le sac.

Remarquons que l'exploration de tous les hyperplans H_k, de $k = 1$ à $k = n$, c'est à dire de l'espace tout entier, est inefficace et peut être à l'origine d'une grande perte de temps. L'idée est alors de se limiter à des hyperplans convenablement choisis et qui ont une importance capitale pour la recherche d'une solution optimale. Pour ce faire, les auteurs ont opté pour une réduction du nombre d'hyperplans à explorer, en se basant sur le travail de Fréville et Plateau [32] qui proposent un algorithme pour l'encadrement du nombre k. Cet encadrement fait appel à des outils de la programmation linéaire pour calculer la valeur de k_{min} (respectivement k_{max}) qui représente le nombre minimal d'objets (respectivement le nombre maximal d'objets) à mettre dans le sac.

Vasquez et Hao ont développé un algorithme simple dont le principe est d'utiliser la recherche tabou pour obtenir une borne inférieure z du problème, puis résoudre par le

simplexe le programme linéaire (6.6) (respectivement le programme linéaire (6.7)).

$$PL(\sigma_{min}(z)) \begin{cases} min \quad \sum_{j=1}^{n} x_j \\ \sum_{j=1}^{n} w_{ij} x_j \leq b_i \qquad \forall i \in \{1, ..., m\}, \\ \sum_{j=1}^{n} p_j x_j \geq z + 1, \\ x \in [0,1]^n, \end{cases} \tag{6.6}$$

$$PL(\sigma_{max}(z)) \begin{cases} max \quad \sum_{j=1}^{n} x_j \\ \sum_{j=1}^{n} w_{ij} x_j \leq b_i \qquad \forall i \in \{1, ..., m\}, \\ \sum_{j=1}^{n} p_j x_j \geq z + 1, \\ x \in [0,1]^n, \end{cases} \tag{6.7}$$

Soient $\sigma_{min}(z)$ la valeur optimale du problème (6.6) et $\sigma_{max}(z)$ la valeur optimale du problème (6.7). Notons que si on prend moins d'objets que $k_{min} = \lfloor \sigma_{min}(z) \rfloor$, la contrainte $\sum_{j=1}^{n} p_j x_j \geq z+1$ ne serait plus vérifiée. C'est à dire, dans ce cas, on ne peut pas avoir une solution meilleure que celle ayant la valeur z. De même, si on prend plus d'objets que $k_{max} = \lfloor \sigma_{max}(z) \rfloor$, l'une au moins des contraintes de capacité ne serait plus vérifiée. Ainsi, les seules valeurs intéressantes de k pour le processus de recherche locale sont celles comprises entre k_{min} et k_{max}.

Par la suite, en fixant des bornes inférieure k_{min} et supérieure k_{max} pour k, on se limite à l'exploration de $k_{max} - k_{min} + 1$ hyperplans H_k au lieu de n.

6.4 HIPL$_k$: Heuristique itérative basée sur la programmation linéaire et la décomposition de l'espace

L'heuristique HIPL$_k$ que nous proposons consiste à appliquer la méthode HIPL sur des hyperplans H_k. En effet, la première étape de notre méthode se base sur la décomposition de l'espace de recherche S en hyperplans H_k. La deuxième étape consiste à appliquer la méthode HIPL pour résoudre les sous problèmes P_k.
Plus précisément, notre contribution principale, présentée par l'Algorithme 12, se base sur les étapes suivantes :

1) Calcul d'une solution initiale. (ligne 1 de l'Algorithme 12)
2) Décomposition de l'espace en calculant les valeurs limites de k, *i.e.*, k_{min} et k_{max}. (lignes 2 et 3)
3) Tri des hyperplans H_k, selon un ordre décroissant des bornes supérieures u_k. (ligne 8)
4) Exploration des espaces de recherche relatifs aux hyperplans, et mise à jour de la décomposition de l'espace de recherche S. (lignes 10-20)

Ces étapes sont détaillées respectivement dans les sections suivantes.

6.4.1 Solution initiale

Pour obtenir rapidement une solution initiale, nous utilisons une heuristique gloutonne décrite dans l'Algorithme 13. Partant d'un sac vide, la méthode consiste à mettre dans le sac des objets ordonnés tant que toutes les contraintes de capacité sont vérifiées. Notons que les objets sont classés par ordre décroissant de profit par unité de poids p_j/w_j.
La solution initiale est utilisée pour le calcul des limites k_{min} et k_{max} de l'espace de

Algorithm 12: HIPL$_k$

Data: Une instance du problème P, max_iter et $hmax\_iter$
Result: une solution réalisable x^*

x^* une solution de P obtenue par une heuristique gloutonne(Algorithme 13);
$v^* \leftarrow z(x^*)$;
$k_{min} \leftarrow \lfloor \sigma_{min}(v^*) \rfloor$ (6.6);
$k_{max} \leftarrow \lfloor \sigma_{max}(v^*) \rfloor$ (6.7);
$H \leftarrow \{k | k \in [k_{min}, k_{max}]\}$;
$Q \leftarrow P$; $i \leftarrow 0$; $i_k \leftarrow 0$;
repeat
  $k \leftarrow argmax \quad \{\nu(PL(P_k)), k \in H\}$;
  $i_k \leftarrow 0$;
  while $i_k \leq hmax\_iter$ **do**
    Soit $\bar{x}(k)$ solution optimale de $PL(Q | \sum_{j=1}^{n} x_j = k)$;
    Soit x^0 solution optimale de $P(\bar{x}(k), J(\bar{x}(k))$;
    if $z(x^0) > v^*$ **then**
      $x^* \leftarrow x^0$;
      $v^* \leftarrow z(x^0)$;
      $k_{min} \leftarrow \lfloor \sigma_{min}(v^*) \rfloor$;
      $k_{max} \leftarrow \lfloor \sigma_{max}(v^*) \rfloor$;
    $Q \leftarrow Q | \{fx \leq |J^1(x(k))| - 1\}$;
    $i_k \leftarrow i_k + 1$;
    $i \leftarrow i + 1$;
  $H \leftarrow H \setminus k$;
  if $H = \emptyset$ **then**
    $H \leftarrow \{k, k \in [k_{min}, k_{max}]\}$;
until $i \geq max\_iter$;
return x^0

recherche décrit dans la section suivante. Ces limites seront au fur et à mesure modifiées par des solutions meilleures obtenues dans les itérations ultérieures de l'Algorirhme 12.

6.4.2 Décomposition de l'espace

Notons que l'ajout de la contrainte $\sum_{j=1}^{n} x_j = k$, pour un choix judicieux de k, accélère la recherche et évite le travail gaspillé sur des régions sans intérêt. Ainsi notre méthode est basée sur la résolution d'une série de problèmes P_k en ajoutant cette

Algorithm 13: Heuristique gloutonne

Data: Une instance de P
Result: Une solution réalisable x^0

$x^0 \leftarrow 0$; $r_i \leftarrow b_i \quad \forall i \in \{1, ..., m\}$;
Classer les objets $j \in V$ tel que $\frac{p_j}{w_j} \leq \frac{p_{j+1}}{w_{j+1}}$;
$z \leftarrow 0$, $j \leftarrow 1$;
while $j < n$ **do**
    if $r_i - w_{ij} > 0 \quad \forall i \in \{1, 2..., m\}$ **then**
        $z \leftarrow z + p_j$;
        $r_i \leftarrow r_i - w_{ij} \forall i \in \{1, 2..., m\}$;
        ${x_j}^0 \leftarrow 1$;
    $j \leftarrow j + 1$;
return x^0

contrainte valide au problème MKP. Le problème P_k, k allant de k_{min} à k_{max}, peut s'écrire sous la forme :

$$P_k \begin{cases} \max \sum_{j=1}^{n} p_j x_j & \\ \sum_{j=1}^{n} w_{ij} x_j \leq c_i & \forall i \in \{1, \ldots, m\}, \\ \sum_{j=1}^{n} x_j = k, & \\ x_j \in \{0, 1\} & \forall j \in \{1, \ldots, n\}. \end{cases} \tag{6.8}$$

Chaque sous problème P_k est traité sur une partie H_k de l'espace de recherche S. Pour limiter l'espace de recherche nous avons calculé les valeurs de k_{min} et de k_{max} en résolvant les programmes linéaires (6.6) et (6.7), respectivement. La résolution de ces programmes se base sur le choix de la valeur z qui représente la valeur de la fonction objective de la solution initiale x^0 présentée dans la section 6.4.1 ($z = \sum_{j=1}^{n} p_j x_j^0$). L'encadrement des nombres d'hyperplans est mis à jour chaque fois qu'une meilleure solution est obtenue. C'est à dire la valeur de z sera modifiée $z = \sum_{j=1}^{n} p_j x_j^*$, où x^* est la meilleure solution obtenue et par la suite les valeurs de k_{min} et k_{max} seront modifiées.

6.4.3 Ordre d'exploration des hyperplans

Pour chaque hyperplan H_k, k allant de k_{min} à k_{max}, on calcule une borne supérieure en résolvant la relaxation linéaire (6.9) du problème P_k relatif à l'hyperplan H_k .

$$PL(P_k) \begin{cases} \max \sum_{j=1}^{n} p_j x_j \\ \sum_{j=1}^{n} w_{ij} x_j \leq b_i \qquad \forall i \in \{1, ..., m\}, \\ \sum_{j=1}^{n} x_j = k, \\ x \in [0,1]^n. \end{cases} \tag{6.9}$$

Afin de maximiser notre chance d'obtenir la meilleure solution le plus rapidement possible, nous avons proposé de fixer un ordre d'exploration des différents hyperplans. En effet, nous avons choisi de trier les hyperplans H_k selon un ordre décroissant des bornes supérieures u_k des sous problèmes P_k correspondants (*i.e.*, $u_1 > u_2 > u_3 > ..... > u_i$, avec $i = k_{max} - k_{min} + 1$).

6.4.4 Exploration des hyperplans

Notons qu'à chaque itération de notre algorithme, une pseudo-coupe générée selon (6.5) est ajoutée dans le problème. Au cours de ces itérations, on explore les hyperplans H_k, sachant que, le passage d'un hyperplan à un autre se fait après $hmax\_iter$ itérations et le nombre maximal totale d'itérations est égale à max_iter.

Sur chaque hyperplan H_k l'exploration peut être décrite comme suit. Nous commençons par résoudre, par la méthode de simplexe, le problème relaxé (6.10) :

$$PL(P_k)\begin{cases} \max \sum_{j=1}^{n} p_j x_j \\ \sum_{j=1}^{n} w_{ij} x_j \leq c_i \qquad \forall i \in \{1,...,m\}, \\ \sum_{j=1}^{n} x_j = k, \\ x \in [0,1]^n. \end{cases} \tag{6.10}$$

Ceci donne comme résultat une solution en continue $\bar{x}$ à partir de laquelle nous cherchons une solution réalisable binaire en résolvant par la méthode de séparation et évaluation (Branch and Bound) le sous problème réduit suivant :

$$P_k(\bar{x}, J(\bar{x})) : \begin{cases} \max \sum_{j=1}^{n} p_j x_j \\ \sum_{j=1}^{n} w_{ij} x_j \leq c_i \qquad \forall i \in \{1,...,m\}, \\ x_j = \bar{x_j} \quad \forall j \in J(\bar{x}), \\ \sum_{j=1}^{n} x_j = k, \\ x \in \{0,1\}^n. \end{cases} \tag{6.11}$$

La résolution de (6.10) et (6.11) est répétée $hmax\_iter$ fois (en ajoutant à chaque fois une pseudo-coupe).

Le passage à l'hyperplan suivant (selon l'ordre présenté dans la Section 6.4.3) se fait en changeant seulement la contrainte $\sum_{j=1}^{n} x_j = k$ tout en gardant toutes les pseudo-coupes déjà ajoutées.

Une fois la meilleure solution réalisable trouvée est mise à jour, elle sera utilisée pour reformuler la contrainte $\sum_{j=1}^{n} p_j x_j \geq z + 1$ (voir Section 6.3). Une telle reformulation sert à accélérer l'exécution de notre algorithme puisqu'elle peut réduire l'intervalle $[k_{min}, k_{max}]$.

Une fois l'exploration de tous les hyperplans est faite, on continue la recherche en revenant au premier hyperplan (lignes 22 et 23 de l'Algorithme 12) tout en considérant le fait que les bornes sont mises à jour à chaque amélioration de la solution. Le pro-

cessus de notre algorithme s'arrête lorsqu'on atteint un nombre maximum d'itérations max_iter.

Plus la valeur du paramètre $hmax\_iter$ ($hmax\_iter$ étant le nombre maximum d'itérations pour l'exploration d'un hyperplan) est élevée plus l'exploration de la recherche dans l'hyperplan H_k est intensifiée. Toutefois une valeur de $hmax\_iter$ très élevée ne garantit pas l'exploration de tous les hyperplans vu la contrainte d'un nombre global max_iter d'itérations de l'algorithme et donc elle réduit la diversification dans l'algorithme. Le choix du nombre d'itérations $hmax\_iter$ doit en effet garantir l'exploration de tous les hyperplans au moins une fois. Par exemple nous pouvons le fixer à $max\_iter/2 * (k_{max} - k_{min} + 1)$.

6.5 Expériences Numériques

L'algorithme HIPL$_k$ a été codé en langage C++. Le solveur linéaire CPLEX est utilisé pour le calcul des bornes supérieures et pour la résolution du problème réduit. L'ensemble des tests a été réalisé par un processeur Intel core i5 à vitesse 2.40GHz avec Mémoire de 4Go.

6.5.1 Description des instances

L'algorithme que nous proposons a été testé sur un ensemble conséquent d'instances de MKP. Nous avons utilisé un ensemble d'instances corrélées et difficiles de la OR-Librairie [8]. Ces instances ont été générées selon la procédure proposée par Fréville et Plateau [33]. Pour toutes ces instances les coefficients w_{ij} sont des nombres entiers uniformément générés dans $U(0, 1000)$. Les membres droits des contraintes c_i sont calculés selon la formule $c_i = \alpha \sum_{j=1}^{n} w_{ij}$ avec $\alpha = 0.25, 0.5, 0.75$.

Les coefficients associés à l'objectif du problème p_j sont corrélés aux coefficients w_{ij} comme suit : $p_i = \alpha \sum_{i=1}^{m} w_i/m + 500\delta_j, \forall j \in N$ avec δ_j un réel uniformément généré dans $U(0,1)$. Les instances ont été générées en changeant le nombre de variables $(n = 250, 500)$, et la valeur de α $(\alpha = 0.25, 0.5, 0.75)$.

6.5.2 Choix des paramètres de l'heuristique

Pour choisir la valeur optimale du paramètre $hmax\_iter$, nous avons procédé à la variation de cette valeur et après plusieurs tests, notre choix s'est fixé sur la valeur 30 pour les instances de taille 250 et 50 pour les instances de taille 500, nous comparons notre algorithme avec celui de HIPL et celui de GA (algorithme génétique proposé par Chu et Beasley [18]) . Nous présentons les résultats obtenus après une durée totale égale à 300 secondes par le Tableau 6.1 (respectivement le Tableau 6.2) qui représente les résultats obtenus par HIPL$_k$ sur les instances de taille 250 (respectivement de taille 500). Pour les problèmes de taille 250 (Tableau 6.1), HIPL$_k$ trouve une meilleure solution que HIPL pour 7 instances (les valeurs en gras dans le tableau) alors que HIPL devance HIPL$_k$ seulement sur 3 instances. Pour les 20 instances restantes les deux algorithmes ont la même performance. De plus sur 13 parmi les 20 instances où HIPL$_k$ a réalisé une performance équivalente à celle de HIPL, la solution a été trouvée en moins de temps que HIPL.
Pour les problèmes de taille 500 (Tableau 6.2), HIPL$_k$ a trouvé une meilleure solution dans 12 instances, la même performance dans 9 instances et HIPL était meilleure que HIPL$_k$ dans 9 instances.

Cette heuristique peut être améliorée si on arrive à bien choisir la valeur du paramètre $hmax\_iter$. Ce choix peut ouvrir ultérieurement la porte pour de nouvelles recherches, étant donné que la valeur de $hmax\_iter$ doit varier d'une instance à une autre même si ces instances sont de même taille. Pour cela, on doit espérer l'intégrer ultérieurement dans notre algorithme pour qu'il devienne une variable calculée, par exemple en fonction

instance	GA	HIPL		HIPL_k	
		$z(x^*)$	cpu	$z(x^*)$	cpu
1	56693	**56824**	117.042	56765	257.044
2	58318	58351	23.311	**58416**	192.687
3	56553	56553	17.643	56553	1.939
4	56863	56930	2.763	56930	3.0820
5	56629	56629	126.048	56629	83.085
6	57119	**57189**	159.696	57146	15.487
7	56292	56222	281.486	**56229**	224.245
8	56403	56457	83.710	56457	104.923
9	57442	57323	271.017	**57349**	241.251
10	56447	56447	2.575	56447	2.309
11	107689	107746	3.903	107746	33.926
12	108338	108379	72.785	108379	71.534
13	106385	106409	207.011	**106415**	158.259
14	106796	106855	107.254	106855	72.912
15	107396	107382	20.837	107382	226.746
16	107246	107189	63.845	**107246**	216.583
17	106308	106305	22.408	106305	13.100
18	103993	**103998**	244.716	103991	32.460
19	106835	106800	211.469	106800	62.409
20	105751	105733	50.008	105733	21.187
21	150083	150138	8.686	150138	23.085
22	149907	149907	18.323	149907	4.267
23	152993	152993	58.7890	**153007**	197.526
24	153169	153177	146.650	**153190**	255.457
25	150287	150287	22.947	150287	0.865
26	148544	148559	53.831	148559	30.386
27	147471	147471	31.346	147471	97.287
28	152841	152877	30.50	152877	16.071
29	149568	149570	258.494	149570	115.570
30	149572	149601	41.108	149601	88.626

TABLE 6.1 – Résultats obtenus sur les instances $OR - 250 - 30$

du nombre d'hyperplans.

instance	GA	HIPL		HIPL_k	
		$z(x^*)$	cpu	$z(x^*)$	cpu
1	115868	115857	302.159	**115915**	105.099
2	114667	114722	305.195	114722	43.715
3	116661	116661	84.570	116661	1.939
4	115237	115223	200.444	**115228**	202.278
5	116353	116398	148.840	**116430**	190.509
6	115604	115741	188.996	115741	167.457
7	113952	**114010**	280.760	114003	278.917
8	114199	114160	174.170	**114249**	88.221
9	115247	115419	210.103	115419	54.400
10	116947	117002	137.559	**117023**	48.971
11	217995	218033	257.702	**218068**	196.721
12	214534	214532	72.785	**214645**	198.432
13	215854	**215903**	304.793	215856	66.828
14	217836	217827	14.814	217827	42.214
15	215566	215613	245.235	215591	71.902
16	215762	215837	225.428	**215847**	150.235
17	215772	**215800**	230.276	215798	98.166
18	216336	216444	303.681	**216450**	200.413
19	217290	217313	128.432	217313	208.439
20	214624	214621	50.008	**214639**	105.051
21	301627	301627	8.686	301627	24.943
22	299985	299999	182.600	**300014**	186.365
23	304995	**305080**	297.997	305028	19.123
24	301935	**301989**	260.683	301961	86.622
25	304404	304413	166.649	304413	81.793
26	296894	**296959**	259.143	296906	33.000
27	303233	303288	189.519	303288	77.744
28	306944	**306944**	107.671	306910	90.404
29	303057	303113	246.620	**303147**	98.858
30	300460	**300531**	41.108	300499	67.397

TABLE 6.2 – Résultats obtenus sur les instances $OR - 500 - 30$

6.6 Conclusion

Dans ce chapitre, nous avons proposé de résoudre le MKP en utilisant une heuristique basée sur la programmation linéaire que nous appelons HIPL_k. L'HIPL_k est inspirée

des travaux de Wilbaut et Hanafi [55, 106] combinés avec les techniques de décomposition de l'espace de recherche proposées par Vasquez et Hao [104]. Les résultats obtenus ont fait l'objet d'une communication à la conférence Méta12 [11]. L'étude de ces résultats montre l'efficacité de notre approche HIPL$_k$ pour la résolution du MKP. Cependant, plusieurs améliorations de notre heuristique peuvent être considérées. En effet, une étude approfondie des valeurs optimales des paramètres peut être dans ce cadre envisagée.

Conclusion

Le problème d'optimisation classique connu sous le nom du sac à dos a été vastement étudié et plusieurs méthodes ont été proposées pour le résoudre. Vu son importance cruciale dans plusieurs domaines d'applications et sa complexité difficile, le problème du sac à dos et ses diverses variantes ne cessent de nos jours d'intéresser les chercheurs qui proposent des méthodes de résolution de plus en plus efficaces. Dans cette thèse nous avons étudié deux variantes du problème du sac à dos : le problème du sac à dos à contraintes disjonctives (Disjunctively Constrained Knapsack Problem DCKP), et le problème du sac à dos multidimensionnel (Multidimensional Knapsack Problem MKP). Nous avons en particulier proposé des algorithmes exacts et/ou heuristiques pour la résolution de ces problèmes.

Au début, nous nous sommes intéressés à une résolution exacte du problème DCKP. Nous avons considéré une formulation standard du problème, lui avons défini le polytope associé et étudié l'aspect facial des contraintes de base. Ensuite, nous avons proposé de nouvelles familles d'inégalités valides pour le DCKP et étudié les problèmes de séparation qui leur sont associés. Basés sur ces résultats, nous avons développé un algorithme de coupe et branchement pour résoudre le problème. L'algorithme a été testé sur 170 instances plutôt difficiles. En comparant nos résultats aux résultats donnés par Cplex, nous avons conclu que notre algorithme de coupe et branchement est plus performant que Cplex pour quasiment toutes les instances. Nous avons aussi montré l'efficacité des contraintes valides identifiées dans l'obtention d'une meilleure relaxation linéaire.

Ensuite, nous avons proposé de résoudre le problème DCKP par une méthode basée sur la métaheuristique recherche tabou. La particularité de notre algorithme de recherche tabou est l'ajout d'un aspect probabiliste ainsi que l'utilisation de structures de voisinage multiples. La recherche tabou probabiliste est une variante de la

recherche tabou où les mouvements sont choisis de façon probabiliste favorisant ceux ayant les meilleures évaluations. Notre algorithme a été testé sur un total de 50 instances de la littérature allant jusqu'à 1000 objets. L'étude expérimentale a montré que notre algorithme de recherche tabou donne en majorité des résultats meilleurs que ceux trouvés par des algorithmes récents dans la littérature. En particulier, sur les 50 instances testées, notre approche a été capable de trouver 46 "best known solutions" dont 8 meilleures que les solutions déjà connues dans la littérature.

En fin, nous avons proposé une heuristique basée sur la programmation linéaire pour résoudre le problème MKP. Celle-ci s'inspire d'une heuristique de la littérature utilisant la programmation linéaire ainsi que des techniques connues de réduction et décomposition de l'espace de recherche. Cette heuristique, appelée heuristique basée sur la programmation linéaire et la décomposition de l'espace de recherche, nous a permis de résoudre efficacement des instances de la OR-Librairie pour le problème MKP.

En conclusion, les méthodes que nous avons proposées se sont avérées efficaces pour la résolution de certaines variantes du problème du sac à dos, à savoir le DCKP et le MKP. Cependant, plusieurs améliorations peuvent être envisagées pour ces méthodes. En effet, pour l'algorithme de coupe et branchement, nous pouvons penser d'abord à améliorer nos algorithmes de séparation et ensuite à identifier de nouvelles familles d'inégalités valides pour une meilleure relaxation linéaire. Concernant la métaheuristique de Recherche Tabou Probabiliste, on peut par exemple envisager de l'hybrider avec une autre métaheuristique. Enfin, pour la métaheuristique basée sur la programmation linéaire, des pistes d'amélioration intéressantes seraient de séparer des contraintes valides pour le MKP d'une part et d'améliorer la stratégie de réduction de l'espace de recherche d'autre part.
Il serait également intéressant d'appliquer les approches développées dans le cadre de la thèse à d'autres variantes du problème du sac à dos ou encore à des extensions des problèmes DCKP et MKP.

Bibliographie

[1] Hakim Akeb, Mhand Hifi, and Mohamed Elhafedh Ould Ahmed Mounir. Local branching-based algorithms for the disjunctively constrained knapsack problem. *Computers & Industrial Engineering*, 60(4) :811–820, 2011.

[2] Ines Alaya, Christine Solnon, and Khaled Ghedira. Ant colony optimization for multi-objective optimization problems. In *ICTAI (1)*, pages 450–457. Citeseer, 2007.

[3] Silvio Roberto Martins Amarante, Filipe Maciel Roberto, André Ribeiro Cardoso, and Joaquim Celestino. Using the multiple knapsack problem to model the problem of virtual machine allocation in cloud computing. In *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*, pages 476–483. IEEE, 2013.

[4] Egon Balas. Facets of the knapsack polytope. *Mathematical programming*, 8(1) :146–164, 1975.

[5] Egon Balas and Robert Jeroslow. Canonical cuts on the unit hypercube. *SIAM Journal on Applied Mathematics*, 23(1) :61–69, 1972.

[6] Vincent Barichard and Jin-Kao Hao. Genetic tabu search for the multi-objective knapsack problem. *Tsinghua Science and Technology*, 8(1) :8–13, 2003.

[7] Cristina Bazgan, Hadrien Hugot, and Daniel Vanderpooten. Solving efficiently the 0–1 multi-objective knapsack problem. *Computers & Operations Research*, 36(1) :260–279, 2009.

[8] John E. Beasley. Or-library : distributing test problems by electronic mail. *Journal of the operational research society*, 41(11) :1069–1072, 1990.

[9] RICHARD Bellman. Dynamic programming. *Princeton, USA : Princeton University Press*, 1(2) :3, 1957.

[10] Mariem Ben Salem, Saïd Hanafi, Raouia Taktak, and Hanêne Ben-Abdallah. Probabilistic tabu search with multiple neighborhoods for the disjunctively constrained knapsack problem. *RAIRO - Operations Research*, A apparaître.

[11] Mariem Ben Salem, Saïd Hanafi, Christophe Wilbaut, and Mariem Gzara. Improved hybrid heuristic for solving the 0-1 multidimensional knapsack problem. In *International Conference on Metaheuristics and Nature Inspired Computing, Port El-Kantaoui (Tunisie), octobre*, 2012.

[12] Mariem Ben Salem, Raouia Taktak, and Hanêne Ben-Abdallah. Polyhedral analysis for the disjunctively constrained knapsack problem. In *Control, Decision and Information Technologies (CoDIT)*, 2016.

[13] Mariem Ben Salem, Raouia Taktak, A Ridha Mahjoub, and Hanêne Ben-Abdallah. Optimization algorithms for the disjunctively constrained knapsack problem. *Soft Computing*, pages 1–19, 2016.

[14] Andrea Bettinelli, Valentina Cacchiani, and Enrico Malaguti. Bounds and algorithms for the knapsack problem with conflict graph. Technical report, Technical Report OR-14-16, DEIS–University of Bologna, Bologna, Italy, 2014.

[15] Ricardo Stegh Camati, Alcides Calsavara, and Luiz Lima Jr. Solving the virtual machine placement problem as a multiple multidimensional knapsack problem. *ICN 2014*, page 264, 2014.

[16] Alberto Caprara, David Pisinger, and Paolo Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11(2) :125–137, 1999.

[17] Nawal Cherfi and Mhand Hifi. A column generation method for the multiple-choice multi-dimensional knapsack problem. *Computational Optimization and Applications*, 46(1) :51–73, 2010.

[18] Paul C. Chu and John E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1) :63–86, 1998.

[19] Alberto Colorni, Marco Dorigo, and Vittorio Maniezzo. An investigation of some properties of an"ant algorithm". In *PPSN*, volume 92, pages 509–520, 1992.

[20] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[21] Teodor G. Crainic, Michel Gendreau, Patrick Soriano, and Michel Toulouse. A tabu search procedure for multicommodity location/allocation with balancing requirements. *Annals of Operations research*, 41(4) :359–383, 1993.

[22] Igor Crévits, Saïd Hanafi, Raïd Mansi, and Christophe Wilbaut. Iterative semi-continuous relaxation heuristics for the multiple-choice multidimensional knapsack problem. *Computers & Operations Research*, 39(1) :32–41, 2012.

[23] Harlan Crowder, Ellis L. Johnson, and Manfred Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5) :803–834, 1983.

[24] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system : optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1) :29–41, 1996.

[25] Johann Dréo, Alain Pétrowski, Patrick Siarry, and Eric Taillard. *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

[26] Jack Edmonds. Covers and packings in a family of sets. *Bulletin of the American Mathematical Society*, 68(5) :494–499, 1962.

[27] Reinhardt Euler, Michael Jünger, and Gerhard Reinelt. Generalizations of cliques, odd cycles and anticycles and their relation to independence system polyhedra. *Mathematics of Operations Research*, 12(3) :451–462, 1987.

[28] Ulrich Faigle and Walter Kern. Some convergence results for probabilistic tabu search. *ORSA Journal on Computing*, 4(1) :32–37, 1992.

[29] Didier Fayard and Gérard Plateau. Resolution of the 0–1 knapsack problem : comparison of methods. *Mathematical Programming*, 8(1) :272–307, 1975.

[30] Nasim Ferdosian, Mohamed Othman, Borhanuddin Mohd Ali, and Kweh Yeah Lun. Greedy–knapsack algorithm for optimal downlink resource allocation in lte networks. *Wireless Networks*, pages 1–14, 2015.

[31] Stefka Fidanova. Aco algorithm for mkp using various heuristic information. In *International Conference on Numerical Methods and Applications*, pages 438–444. Springer, 2002.

[32] Arnaud Fréville and G. Plateau. Sac à dos multidimensionnel en variables 0-1 : encadrement de la somme des variables à l'optimum. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 27(2) :169–187, 1993.

[33] Arnaud Freville and Gérard Plateau. An efficient preprocessing procedure for the multidimensional 0–1 knapsack problem. *Discrete Applied Mathematics*, 49(1) :189–212, 1994.

[34] Virginie Gabrel and Michel Minoux. A scheme for exact separation of extended cover inequalities and application to multidimensional knapsack problems. *Operations Research Letters*, 30(4) :252–264, 2002.

[35] José E. Gallardo, Carlos Cotta, and Antonio J. Fernández. Solving the multidimensional knapsack problem using an evolutionary algorithm hybridized with branch and bound. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 21–30. Springer, 2005.

[36] Giorgio Gallo, Peter L. Hammer, and Bruno Simeone. Quadratic knapsack problems. In *Combinatorial Optimization*, pages 132–149. Springer, 1980.

[37] Bezalel Gavish and Hasan Pirkul. Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality. *Mathematical programming*, 31(1) :78–105, 1985.

[38] Khaled Ghédira. Optimisation combinatoire par métaheuristiques. *Editions TECHNIP, France*, 2007.

[39] F. Glover and A. Lokketangen. Probabilistic tabu search for zero-one mixed integer programming problems. *University of Colorado, Boulder, USA*, 1994.

[40] Fred Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1) :156–166, 1977.

[41] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5) :533–549, 1986.

[42] Fred Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3) :190–206, 1989.

[43] Fred Glover. Tabu thresholding : Improved search by nonmonotonic trajectories. *ORSA Journal on Computing*, 7(4) :426–442, 1995.

[44] Fred Glover and Gary Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[45] Fred Glover and Gary A. Kochenberger. Critical event tabu search for multidimensional knapsack problems. In *Meta-Heuristics*, pages 407–427. Springer, 1996.

[46] Fred Glover and Manuel Laguna. *Tabu search, 1997*. 1997.

[47] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2) :169–197, 1981.

[48] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.

[49] Zonghao Gu, George L. Nemhauser, and Martin W.P. Savelsbergh. Lifted cover inequalities for 0-1 integer programs : Computation. *INFORMS Journal on Computing*, 10(4) :427–437, 1998.

[50] Boukthir Haddar, Mahdi Khemakhem, Saïd Hanafi, and Christophe Wilbaut. A hybrid heuristic for the 0–1 knapsack sharing problem. *Expert systems with Applications*, 42(10) :4653–4666, 2015.

[51] Peter L. Hammer, Ellis L. Johnson, and Uri N. Peled. Facet of regular 0–1 polytopes. *Mathematical Programming*, 8(1) :179–206, 1975.

[52] Saïd Hanafi. On the convergence of tabu search. *Journal of Heuristics*, 7(1) :47–58, 2001.

[53] Saïd Hanafi and Arnaud Freville. An efficient tabu search approach for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 106(2) :659–675, 1998.

[54] Said Hanafi and Christophe Wilbaut. Scatter search for the 0–1 multidimensional knapsack problem. *Journal of Mathematical Modelling and Algorithms*, 7(2) :143–159, 2008.

[55] Saïd Hanafi and Christophe Wilbaut. Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Annals of Operations Research*, 183(1) :125–142, 2011.

[56] Harisankar Haridas, Sriram Kailasam, and Janakiram Dharanipragada. Cloudy knapsack problems : An optimization model for distributed cloud-assisted systems. In *14-th IEEE International Conference on Peer-to-Peer Computing*, pages 1–5. IEEE, 2014.

[57] Mhand Hifi. An iterative rounding search-based algorithm for the disjunctively constrained knapsack problem. *Engineering Optimization*, 46(8) :1109–1122, 2014.

[58] Mhand Hifi and Mustapha Michrafy. A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *Journal of the Operational Research Society*, 57(6) :718–726, 2006.

[59] Mhand Hifi and Mustapha Michrafy. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & operations research*, 34(9) :2657–2673, 2007.

[60] Mhand Hifi, Mustapha Michrafy, and Abdelkader Sbihi. A reactive local search-based algorithm for the multiple-choice multi-dimensional knapsack problem. *Computational Optimization and Applications*, 33(2-3) :271–285, 2006.

[61] Mhand Hifi, Stephane Negre, and Mohamed Quid Ahmed Mounir. Local branching-based algorithm for the disjunctively constrained knapsack problem. In *Computers & Industrial Engineering, 2009. CIE 2009. International Conference on*, pages 279–284. IEEE, 2009.

[62] Mhand Hifi, Stephane Negre, Toufik Saadi, Saleh Saleh, and Lei Wu. A parallel large neighborhood search-based heuristic for the disjunctively constrained knapsack problem. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 1547–1551. IEEE, 2014.

[63] Mhand Hifi and Nabil Otmani. A first level scatter search for disjunctively constrained knapsack problems. In *Communications, Computing and Control Applications (CCCA), 2011 International Conference on*, pages 1–6. IEEE, 2011.

[64] Mhand Hifi, Sagvan Saleh, Lei Wu, and Jenhui Chen. A hybrid guided neighborhood search for the disjunctively constrained knapsack problem. *Cogent Engineering*, 2(1) :1068969, 2015.

[65] John H. Holland. *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence.* U Michigan Press, 1975.

[66] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM (JACM)*, 21(2) :277–292, 1974.

[67] Konstantinos Kaparis and Adam N. Letchford. Local and global lifted cover inequalities for the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, 186(1) :91–103, 2008.

[68] Konstantinos Kaparis and Adam N. Letchford. Cover inequalities. *Wiley Encyclopedia of Operations Research and Management Science*, 2010.

[69] Konstantinos Kaparis and Adam N. Letchford. Separation algorithms for 0-1 knapsack polytopes. *Mathematical programming*, 124(1-2) :69–91, 2010.

[70] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[71] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems. 2004*. Springer, Berlin.

[72] Scott Kirkpatrick and Mario P. Vecchi. Optimization by simmulated annealing. *science*, 220(4598) :671–680, 1983.

[73] Diego Klabjan, George L. Nemhauser, and Craig Tovey. The complexity of cover inequality separation. *Operations Research Letters*, 23(1) :35–40, 1998.

[74] N. Kumaraguruparan, H. Sivaramakrishnan, and Sachin S. Sapatnekar. Residential task scheduling under dynamic pricing using the multiple knapsack method. In *2012 IEEE PES Innovative Smart Grid Technologies (ISGT)*, pages 1–6. IEEE, 2012.

[75] Ulungu-Ekunda Lukata and Jacques Teghem. Solving multi-objective knapsack problem by a branch-and-bound procedure. In *Multicriteria analysis*, pages 269–278. Springer, 1997.

[76] Ali Ridha Mahjoub. Approches polyédrales. *V. Th. Paschos, éditeur, Optimisation combinatoire : concepts fondamentaux*, pages 263–329, 2004.

[77] Ali Ridha Mahjoub. Polyhedral approaches. *Concepts of Combinatorial Optimization, Volume 1*, pages 261–324, 2013.

[78] Enrico Malaguti, Michele Monaci, and Paolo Toth. An exact approach for the vertex coloring problem. *Discrete Optimization*, 8(2) :174–190, 2011.

[79] Roy E. Marsten and Thomas L. Morin. A hybrid approach to discrete mathematical programming. *Mathematical programming*, 14(1) :21–40, 1978.

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

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

[82] Silvano Martello and Paolo Toth. *Knapsack problems : algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[83] Silvano Martello and Paolo Toth. An exact algorithm for the two-constraint 0-1 knapsack problem. *Operations Research*, 51(5) :826–835, 2003.

[84] G.B. Mathews. On the partition of numbers. *Proceedings of the London Mathematical Society*, 1(1) :486–490, 1896.

[85] R. Garey Michael and S. Johnson David. Computers and intractability : a guide to the theory of np-completeness. *WH Free. Co., San Fr*, 1979.

[86] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11) :1097–1100, 1997.

[87] Robert M. Nauss. An efficient algorithm for the 0-1 knapsack problem. *Management Science*, 23(1) :27–31, 1976.

[88] George L. Nemhauser and Gabriele Sigismondi. A strong cutting plane/branch-and-bound algorithm for node packing. *Journal of the Operational Research Society*, pages 443–457, 1992.

[89] George L. Nemhauser and Leslie E. Trotter Jr. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6(1) :48–61, 1974.

[90] George L. Nemhauser and Laurence A. Wolsey. Integer and combinatorial optimization. interscience series in discrete mathematics and optimization. *ed : John Wiley & Sons*, 1988.

[91] Cristian Oliva, Philippe Michelon, and Christian Artigues. Constraint and linear programming : Using reduced costs for solving the zero/one multiple knapsack problem. In *Proc. of the Workshop on Cooperative Solvers in Constraint Programming (CoSolv 01), Paphos, Cyprus*, pages 87–98, 2001.

[92] David Pisinger. Core problems in knapsack algorithms. *Operations Research*, 47(4) :570–575, 1999.

[93] Mikhail Posypkin and Si Thu Thant Sin. Comparative analysis of the efficiency of various dynamic programming algorithms for the knapsack problem. In *2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW)*, pages 313–316. IEEE, 2016.

[94] Abdelkader Sbihi. A best first search exact algorithm for the multiple-choice multidimensional knapsack problem. *Journal of Combinatorial Optimization*, 13(4) :337–351, 2007.

[95] Alexander Schrijver. *Combinatorial optimization : polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2002.

[96] Aminto Senisuka, Byungjun You, and Takeo Yamada. Reduction and exact algorithms for the disjunctively constrained knapsack problem. In *International Symposium, Operational Research Bremen*, 2005.

[97] Hanxiao Shi. Solution to 0/1 knapsack problem based on improved ant colony algorithm. In *2006 IEEE International Conference on Information Acquisition*, pages 1062–1066. IEEE, 2006.

[98] Wei Shih. A branch and bound method for the multiconstraint zero-one knapsack problem. *Journal of the Operational Research Society*, 30(4) :369–378, 1979.

[99] Omid Ameri Sianaki, Omar Hussain, and Azadeh Rajabian Tabesh. A knapsack problem approach for achieving efficient energy consumption in smart grid for endusers' life style. In *Innovative Technologies for an Efficient and Reliable Electricity Supply (CITRES), 2010 IEEE Conference on*, pages 159–164. IEEE, 2010.

[100] Patrick Soriano and Michel Gendreau. Diversification strategies in tabu search algorithms for the maximum clique problem. *Annals of Operations Research*, 63(2) :189–207, 1996.

[101] AL. Soyster, Ben Lev, and W. Slivka. Zero-one programming with many variables and few constraints. *European Journal of Operational Research*, 2(3) :195–201, 1978.

[102] Tung Khac Truong, Kenli Li, and Yuming Xu. Chemical reaction optimization with greedy strategy for the 0–1 knapsack problem. *Applied Soft Computing*, 13(4) :1774–1780, 2013.

[103] Tony J. Van Roy and Laurence A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35(1) :45–57, 1987.

[104] Michel Vasquez and Jin-Kao Hao. A hybrid approach for the 0-1 multidimensional knapsack problem. In *IJCAI*, pages 328–333, 2001.

[105] Christophe Wilbaut. *Heuristiques hybrides pour la résolution de problèmes en variables 0-1 mixtes.* PhD thesis, Université de Valenciennes et du Hainaut-Cambresis, 2006.

[106] Christophe Wilbaut and Said Hanafi. New convergent heuristics for 0–1 mixed integer programming. *European Journal of Operational Research*, 195(1) :62–74, 2009.

[107] Christophe Wilbaut, Saïd Hanafi, Arnaud Fréville, and Stefan Balev. Tabu search : global intensification using dynamic programming. *Control and Cybernetics*, 35(3) :579, 2006.

[108] Laurence A. Wolsey. Faces for a linear inequality in 0–1 variables. *Mathematical Programming*, 8(1) :165–178, 1975.

[109] Jiefeng Xu, Steve Y. Chiu, and Fred Glover. Probabilistic tabu search for telecommunications network design. *Combinatorial Optimization : Theory and Practice*, 1(1) :69–94, 1996.

[110] Takeo Yamada and Seija Kataoka. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *EURO 2001*, Rotterdam, Pays-Bas, Juillet 2001.

[111] Takeo Yamada, Seija Kataoka, and Kohtaro Watanabe. Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Information Processing Society of Japan Journal*, 43(9), 2002.

[112] Eitan Zemel. Easily computable facets of the knapsack polytope. *Mathematics of Operations Research*, 14(4) :760–764, 1989.